Table of Contents

- Getting started
  - Try it!
    - Overview
    - Create a product
    - Create a data class
    - Create a worker site
    - Create a page
    - Create a process
    - Create a grid widget
    - Create add-object workflow
    - Create a widget button
    - Create edit-object workflow
    - Create delete-object workflow
    - Enable multi languages
    - Enable grid widget features
    - Deploy a Casewhere product
    - Try it! Wrap up
  - Deploy Casewhere
    - Deploy on Azure
    - Deploy on-premise
    - Deployment checklist
    - Configuration settings
    - Service certificates
    - Deploy MongoDB replica set
    - Authentication for Casewhere Admin
    - Generate self-signed certificates
- Fundamentals
  - Overview
  - Product
    - Overview
    - Product management
    - Text resource namespace
  - Worker site

- Overview
- Worker site management
- Worker site authentication
- Worker site localization
- Worker site api

Page and widget
- Overview
- Page management
- Page event
- Widget management
- Widget action
- Grid widget
- List widget
- Custom widget
- Tab container
- Workflow container
- Widget data query
- Widget API

Data class
- Overview
- Data class management
- Data object
- Configure audit logging
- Data validation
- Computed attribute
- Attribute default value
- Free-text search
- Data diagram
- Indexes

Data source
- Overview
- Data source management
- Scripting data source
- Restful data source

Process
- Overview
- Process management

- Case creation logic
- Rule
  - Overview
  - Rule management
  - Access rule
  - Scalar-function rule
  - Workflow navigation rule
  - Data protection rule
  - Data trigger rule
  - Web trigger rule
  - Security policy rule
- Workflow
  - Overview
  - Workflow definition
  - Steps and activities
  - Workflow navigation
  - Workflow navigation examples
  - Workflow execution
  - Workflow concurrency-control
  - Workflow transaction
  - Workflow triggers
    - UI trigger
    - Time trigger - Scheduler
    - Data trigger
    - Event trigger
    - Event trigger - Examples
    - Web trigger
  - Workflow activities
    - Form Activity
    - Form validation
    - Scripted Activity
    - Workflow Activity
    - Dynamic Workflow Activity
    - Loop Activity
    - Stop Activity
- Plugin
  - Overview

- Plugin management
- Plugin development
- Plugin API
- CMS
  - Overview
  - Widget template
  - Content editing
  - Content publishing
  - Content searching
- Theme
  - Overview
  - Theme management
- Automation test
  - Overview
- DevOps
  - Overview
  - DevOps packages
  - Working with DevOps

Advanced

- Overview
- Deployment
  - Hot and Cold - Event transferring
  - Enable Azure Application Insight codeless integration
- Operation and troubleshooting
  - Troubleshoot Casewhere on Azure App Service
  - Troubleshoot Casewhere on premise
  - Troubleshoot with Casewhere logs
  - Daily operation
- Security
  - GDPR - Right to be forgotten
  - OWASP Top Ten - 2021
- Integration use cases
  - Integrate with Power BI
  - Integrate with AAD
- Best practices
  - Form activity
  - Grid widget

- Widget data query
- Workflow delete data
- Workflow multiple steps

Products

- Overview
- Applications
  - Mailbox
  - User management
  - Text resources management
  - Jira to Targetprocess
  - Form management
    - Overview
    - Data management
    - Form action
    - Forms
    - Receipt forms
    - General settings of form component
    - Form components and specific settings
    - Custom form component
- Components
  - Application Changelog
  - Audit Log Export
  - Automatic logout
  - Comment Box
  - Data migration
    - Overview
    - Configuration guide
    - Preparation guide
    - Custom batch processing
    - File migration
  - Data export
  - IdP sync
  - Location data
  - OData URI Parser
  - REST data source
  - SFTP hub
  - Performance report

- Document management
- API key management
- Global settings
- Themes
  - Self-service theme
- Integrations
  - Service Platform CPR Update
  - Service Platform Print Service
  - Navision
- Plugins
  - Aspose Word
  - Aspose PDF
  - CVR
  - File compression
  - JWT
  - Query Filter String Parser
  - SQL client
  - SFTP
  - SSH
  - CSV
  - Service Platform CPR
  - Service Platform CVR
  - SMTP
- Template
  - Mobile app
    - Getting started
      - Overview
      - Environment setup
      - Project structure
      - Before you start
      - Start a project
      - Generate new feature
      - Custom URL Scheme
    - Authentication
      - Overview
      - Configuration
      - Set up identity provider

- Demo
- Biometric authentication
  - Overview
  - Set up biometric
  - Set up identify me
  - User guideline
- Multiple language support
  - Overview
  - Set up text resources key
  - Apply text resources
  - Configuration
  - Demo
- REST API
  - Overview
  - Configuration
  - Demo
- Theme and layout
  - List layout
  - Grid layout
- Notification
  - Overview
  - Configuration
  - Demo
- Roles and rights
  - Overview
  - Configuration
  - Demo
- form.io integration
  - Overview
  - Configuration
  - Demo
- Tutorials
  - Extracting an APK file

Releases

- Overview
- Release notes
  - Casewhere 2.9

- Casewhere 2.8
- Casewhere 2.7
- Casewhere 2.6
- Upgrade notes
  - Upgrade to 2.9
  - Upgrade to 2.8
  - Upgrade to 2.7
- FAQ
  - Overview
  - Casewhere platform
    - Architecture
      - How can Casewhere be customized and extended?
      - How does Casewhere protect data?
      - How is Tracing Secured via Event Sourcing and Logging?
    - Deployment
    - Troubleshooting
    - Security
  - Working with Casewhere
    - Working with data class
    - Working with process

Overview

Prerequisites

1. A Casewhere instance version 2.6 or later.

Practice instance: https://academy-admin.casewhere.com/

Introduction

Welcome to the first article of the **Casewhere TRY IT!** tutorial series.

The series is aimed at providing a quick introduction to Casewhere as well as an overview of the steps required to create a real-world product with Casewhere.

You will learn how to make a basic web-based application that **manages a bookstore** with the following features:

- Add a new book
- Edit a book
- Remove a book
- Search the database for books
- Sort or filter the books in the database
- Offer multiple languages (English/Vietnamese/Danish)
- *... and more*

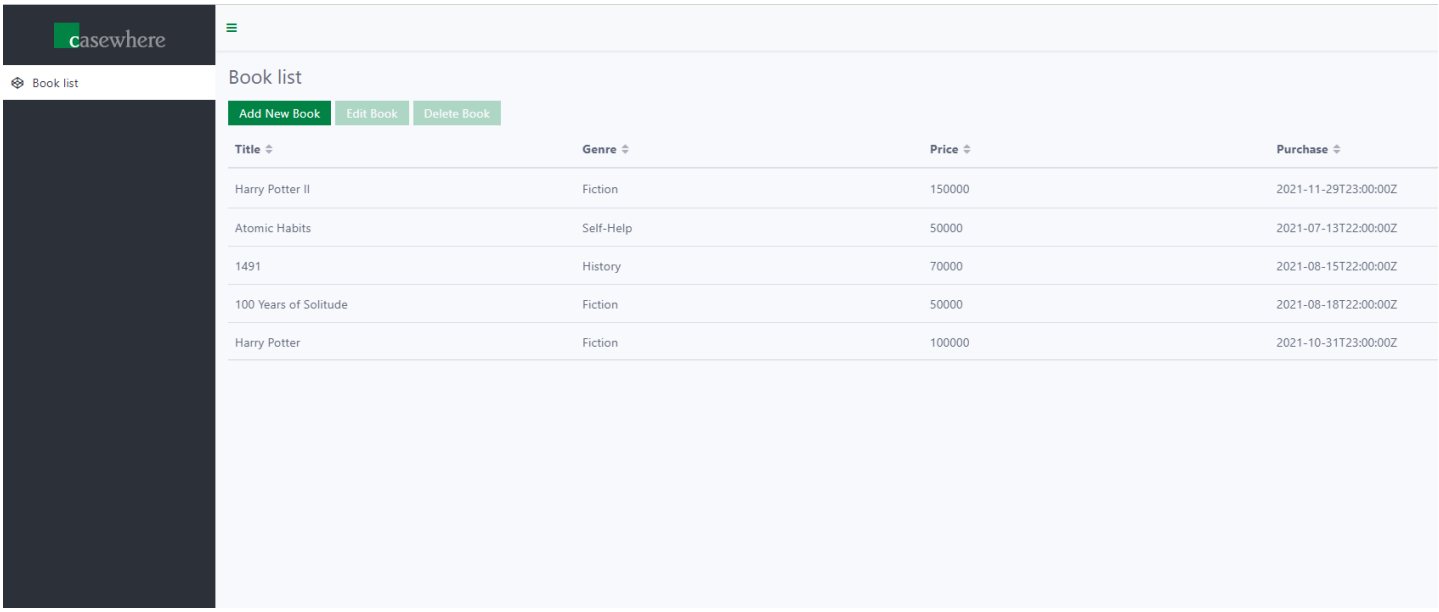
As you learn and move ahead with Casewhere, you will get ideas on how you can easily add more features to this initial project.

Goals

Toward the end of the **Casewhere Try it!** tutorial, you will have a cool Casewhere application that can be shared with others. To save time and move quickly, we're using Casewhere's default grid widget. You will learn how to customize your own widgets later.

The below image is an example of how your final application will look! Each of the available pages will be listed in the menu on the left panel.

The green buttons allow users to interact with your application by adding, editing, and removing books. Several buttons on the right side allow your user to sort, filter, and switch languages.



Looks neat, huh! And the good thing is that it's not hard nor takes a long time to build at all.

So, click on to the next step and let's get started!

Create a product

Introduction

Product in Casewhere refers to your application (read Product overview for more).

A Product's name is crucial since it must explain all core aspects of the application, including its interface and back end functionality.

Other elements of applications (such as data classes, pages, worker sites, and so on) are also encompassed within the product.

This name will also come up frequently in discussions about an application's technical aspects, as well as in meetings with coworkers, cross-functional teams, and clients.

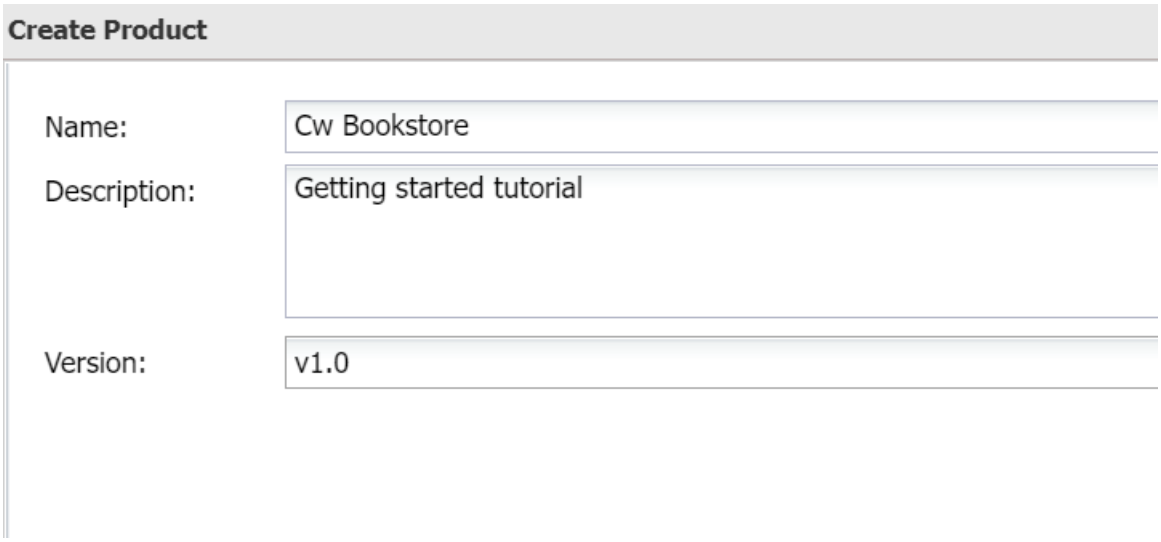
As a result, be descriptive when naming the product.

In this tutorial, the primary goal is to build a product to manage a database for a bookstore. We will name the product "Cw Bookstore".

Guide

- Step 1. Start your Casewhere instance
- Step 2. Navigate to the main menu
- Step 3. Select **Product** and click **Create** button 

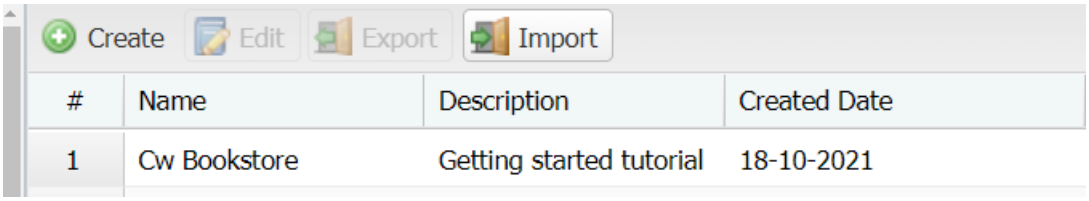
- Step 4. **Create Product** dialog will appear; fill in all the field as follows:



FIELD	ACTIONS
Name	Fill in "Cw Bookstore"
Description	Fill in "Getting started tutorial"
Version	Fill in "v1.0"

- Step 5. Click **Save**

That is it. If all went well, you should now see your product's name on the Product list, as shown in the image below.



You are ready to move to the next step.

Create a data class

Introduction

Designing data models is one of the most critical tasks in building software applications. In this step, you need to instruct Casewhere on how to store data by defining data classes.

Let's spend some time putting together a sample dataset.

TITLE	GENRE	PRICE	PURCHASE DATE
War and Peace	Fiction	60.000	20/5/2021
Atomic Habits	Self-Help	50.000	14/7/2021
1491	History	70.000	16/8/2021
100 Years of Solitude	Fiction	50.000	19/8/2021
Lessons of History	History	50.000	19/9/2021

Look at the above dataset. Do you notice how each data type is presented differently?

A title, for example, could be represented by characters or numbers. So could genre, but genre will only contain limited constants. Prices will simply be a number. Meanwhile, special date/month/year formatting will be required for the purchase date.

Each data type has its own list of criteria!

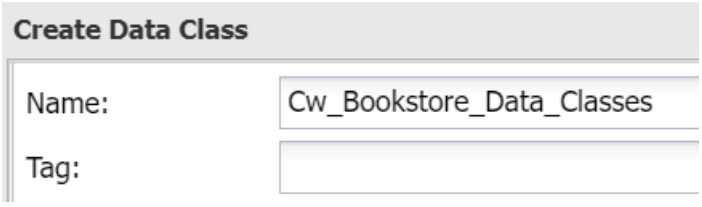
Casewhere supports many types of data. In this tutorial, we will although only deal with array, enum, integer, and datetime.

Guide

- Step 1. In Casewhere Admin site, navigate to **Data Classes**
- Step 2. Click **Create** button 

The **Create Data Class** dialog will appear.

Fill in your data class **Name** as "Cw_Bookstore_Data_Classes".



> Tip: Most dialogs in Casewhere Admin will have a similar steps flow: (1) Fill in name, (2) Config, and (3) Save it.

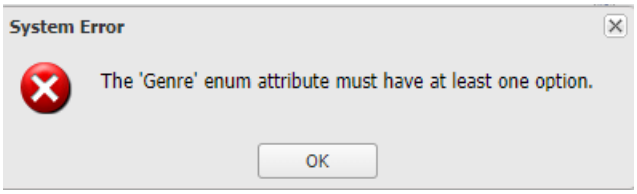
Fill in the following attributes to complete the data model:

NAME	TITLE	TYPE	REQUIRED
Title	title	Array	☑

NAME	TITLE	TYPE	REQUIRED
Genre	genre	Enum	☑
Price	price	Integer	☑
PurchaseDate	purchase_date	Datetime	☑

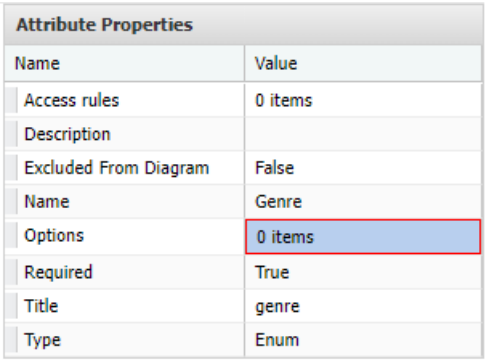
10	Title	title	Array (String)	☑
11	Genre	genre	Enum	☑
12	Price	price	Integer	☑
13	PurchaseDate	purchase_date	DateTime	☑

Looks good! But if you click **Save** now a system error will appear:



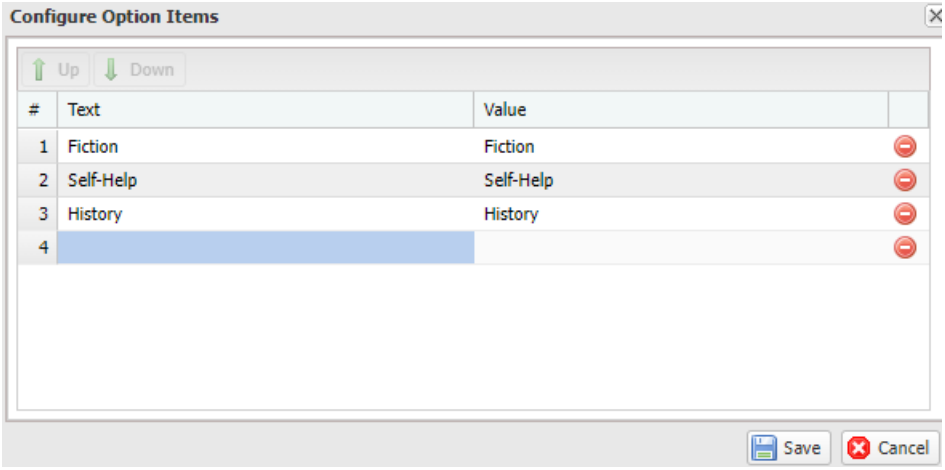
That's because the "Genre" attribute is set as Enum, which will require us to define a set of possible options.

- Step 3. To fix the error, simply choose "Genre", then in **Attribute Properties**, double click to **Options**



- Step 4. Fill in options for Genre. And click **Save**

TEXT	VALUE
Fiction	Fiction
Self-Help	Self-Help
History	History



Click **Save and close** of the **Configure Option Items** dialog.

Click **Save and close** of the **Create Data Class** dialog.

All good, now you can move to the next step.

Create a worker site

Introduction

In this tutorial, you will use the Casewhere Admin site to create a worker site for your application. (Read Worker site overview for more)

This tutorial will be divided into 2 parts:

- In Part 1, you will learn how to add a default theme and configure the language.
    - If your Casewhere instance is brand new and this is your first deployment, the steps in this section are critical.
    - If you're familiar with themes and languages or your Casewhere instance has themes and languages configured, you can skip this step.
- In Part 2, you will learn how to create a worker site with a URL accessible from the internet.

  You will set this link to be available for anonymous users, meaning anyone with the URL can see and interact with your application without any authentication requirement.

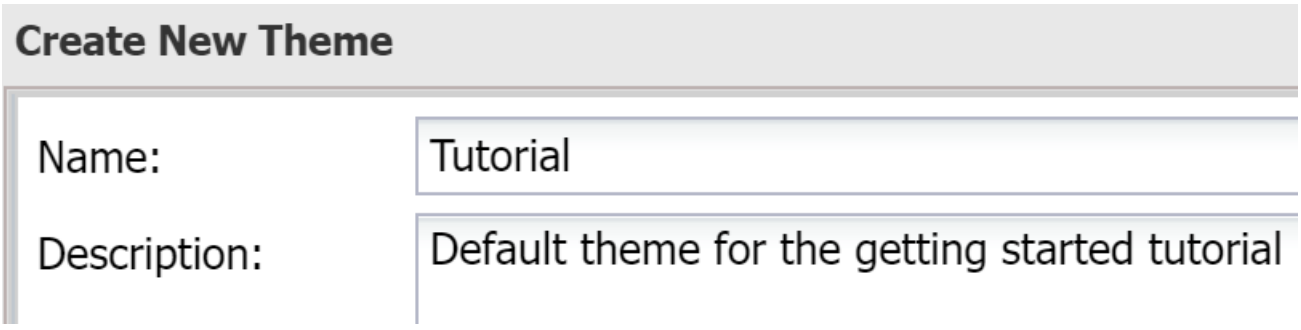
  We will walk through other authentication options of Casewhere, such as SAML 2.0 (i.e., integrating KK NemLog-in), username & password, and WS-Federation in the more advanced tutorials.

Guide

Part 1. Add default theme and set language

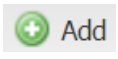
- Step 1. In Casewhere Admin site, navigate to **Themes**, click **Create** button 

FIELD NAME	ACTIONS
Name	Fill in "Tutorial"
Description	Fill in "Default theme for the getting started tutorial"



Click **Save All & Close**

- Step 2. Navigate to **Text Resources**, click **Set Languages** button 

Click **Add** button 

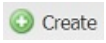
The **Create Language** dialog will open.

FIELD NAME	ACTIONS
Culture	Select "English"
Title	Fill in "English"
Default	☑

Done! English is successfully added as the default language for all of your products.



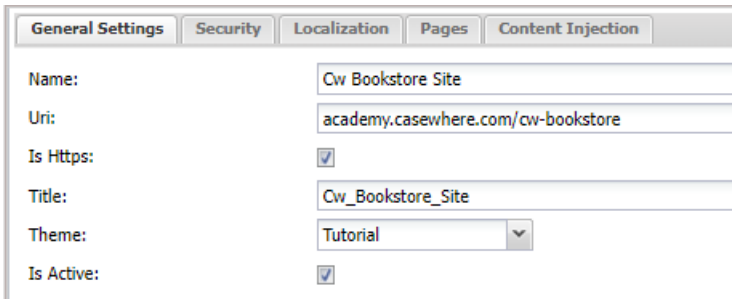
Part 2. Create a worker site

- Step 1. In Casewhere Admin site, navigate to **Worker Sites**.
- Step 2. Click **Create** button 

The **Create Worker Sites** dialog will appear.

- Step 3. Config **General setting** tab

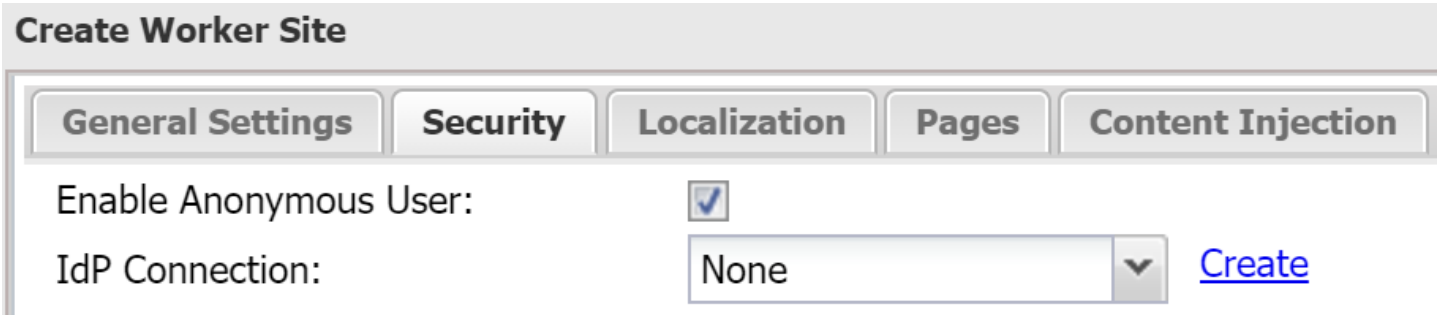
FIELD NAME	ACTIONS
Name	Fill in "Cw Bookstore Site"
Domain name	Fill in "academy.casewhere.com/cw-bookstore"
Is Https?	☑
Title	Fill in "Cw_Bookstore_Site"
Theme	Choose "Tutorials" theme
Is active?	☑



> **Tip**: The Domain name will also be the URL to your application.

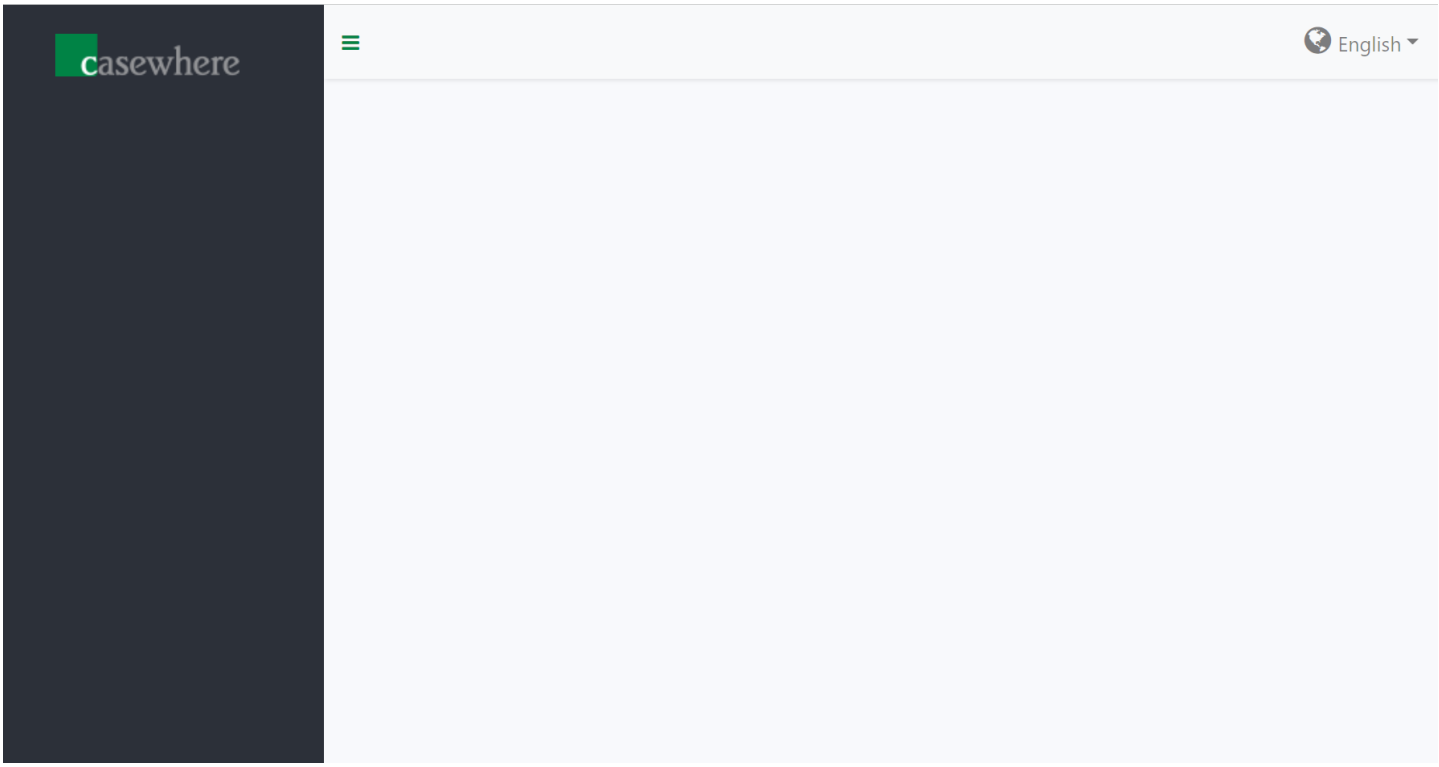
- Step 4. Config **Security** tab

FIELD NAME	ACTIONS
Enable Anonymous User	☑
IdP Connection	Choose "None"



Click **Save**

Way to go! Your website is now activated at https://academy.casewhere.com/cw-bookstore



You now have a worker site with a basic layout that contains a logo, a menu icon, and a language menu. In the next step, you will learn to make a page and link it with this worker site!

Create a page

Introduction

This tutorial will show you how to use Casewhere to make a page and add that page to a worker site.

You will not be required to write any coding scripts for this tutorial. With just a few clicks and drag-and-drop actions, you'll be able to create a basic page.

Meanwhile, Casewhere includes a scripting tool for users who want to create custom pages with unique looks or override a default behavior; these instructions will be covered in more advanced tutorials.

Guide

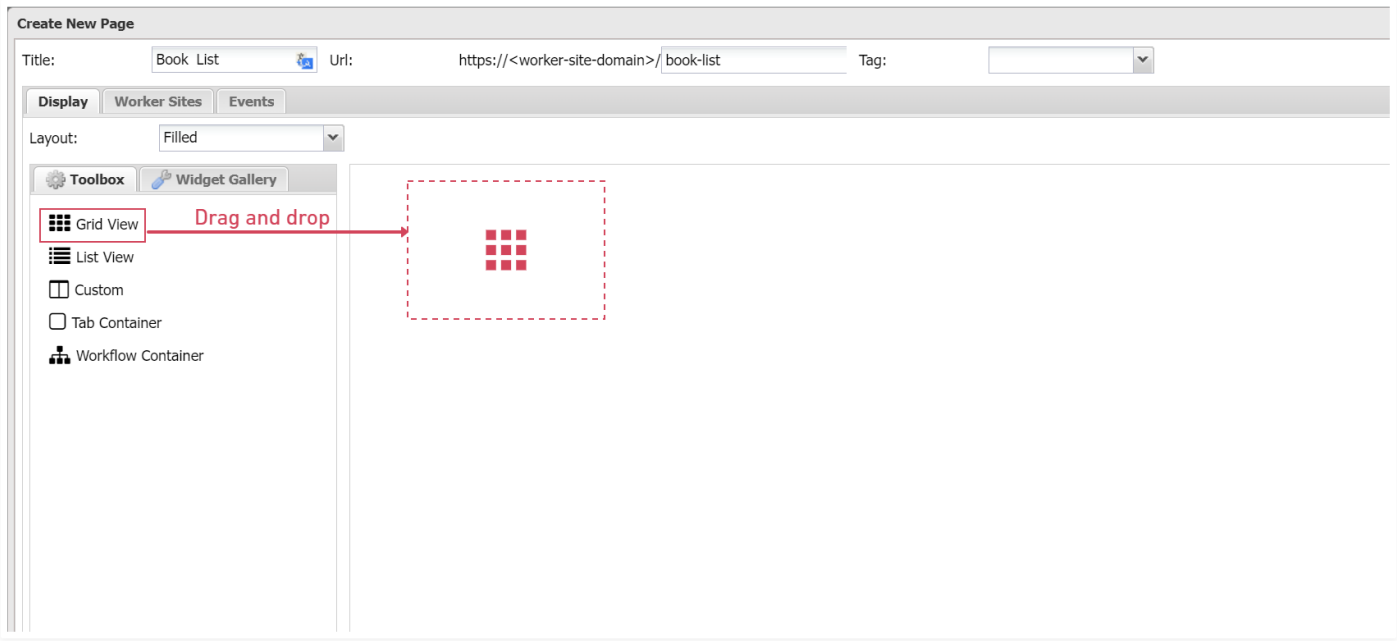
- Step 1. In Casewhere Admin site, navigate to **Pages**.
- Step 2. Click **Create** button 

The **Create New Page** dialog will appear.

Fill in page title and sub-domain that will link to it.

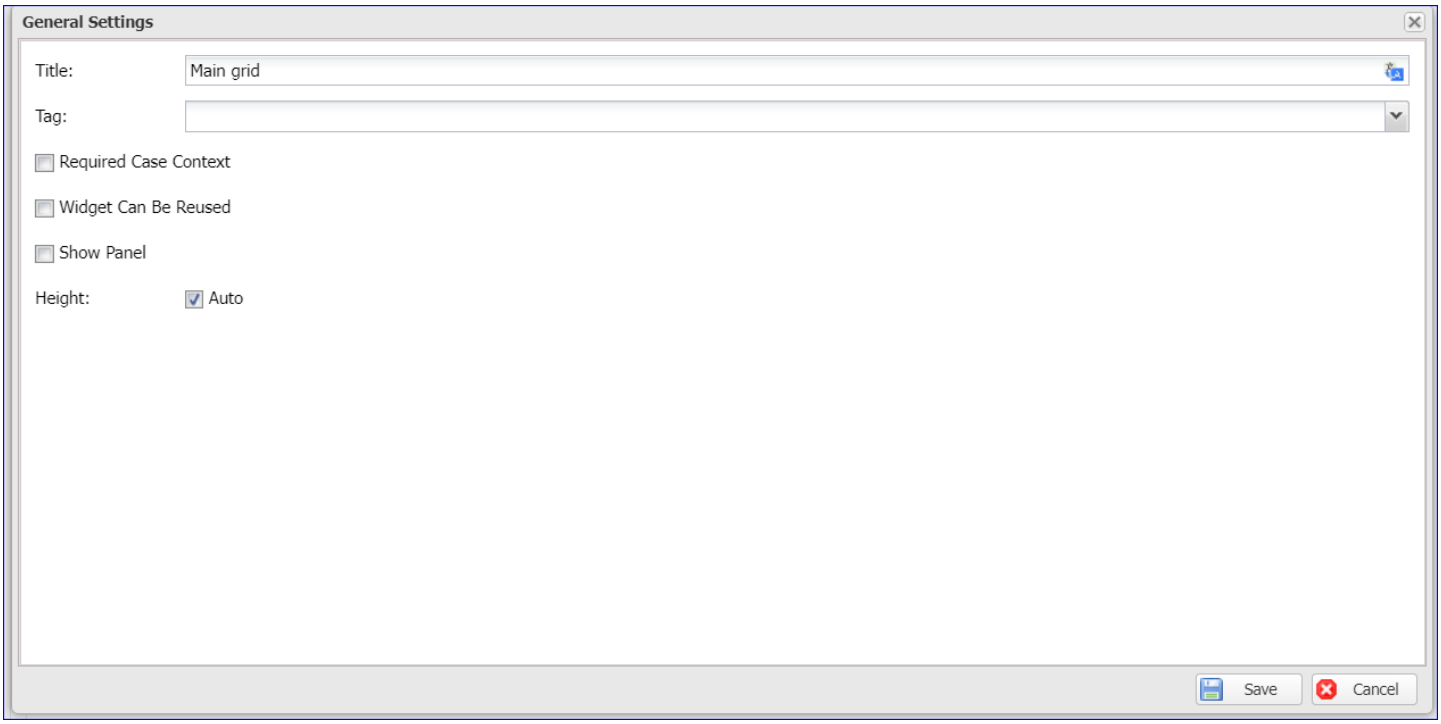
FIELD NAME	ACTIONS
Title	Fill in "Book List"
Url	Fill in "book-list"

Next, drag and drop the **Grid view** from **Toolbox** window to the empty window area.

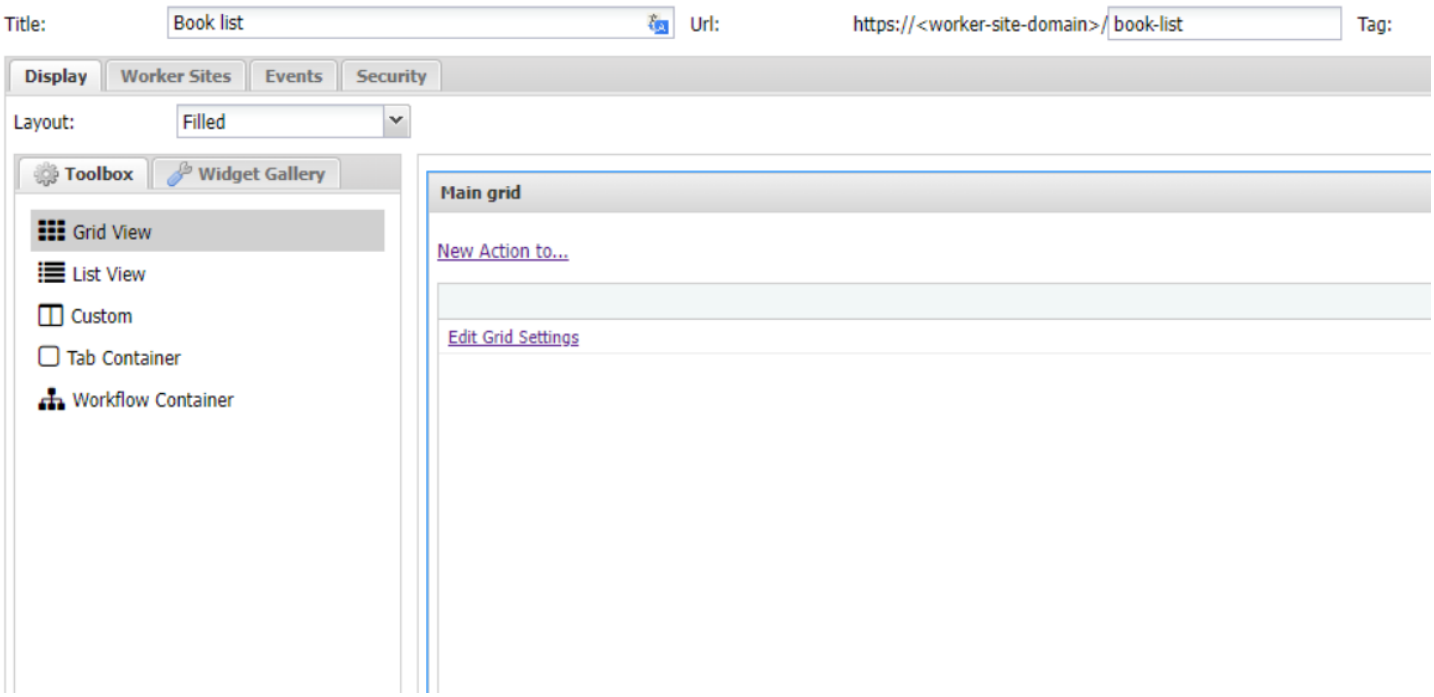


- Step 3. The **General Settings** dialog will show up. Name your grid as "Main grid" and click **Save**. You do not need to config any field in this dialog.

FIELD NAME	ACTIONS
Title	Fill in "Main grid"

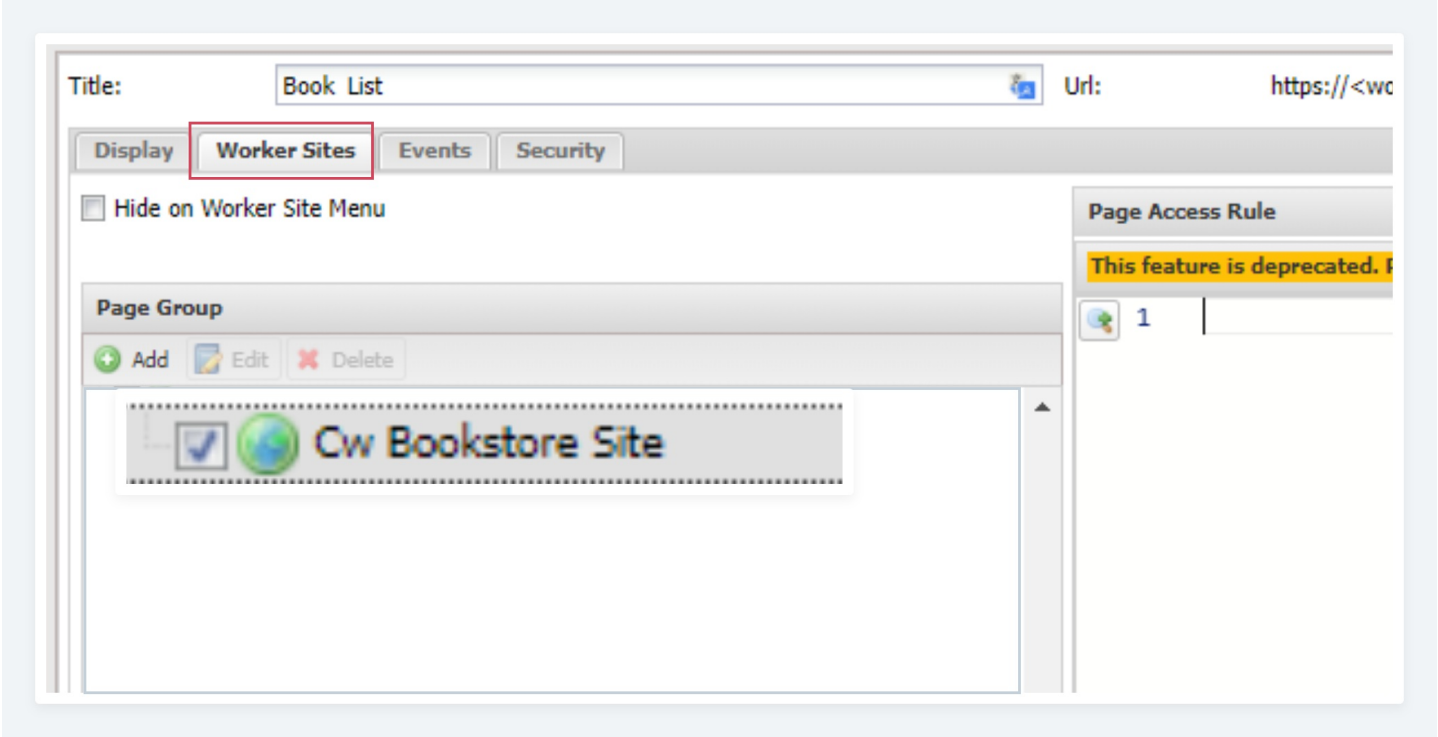


Your grid will be activated as below. Click **Save**.



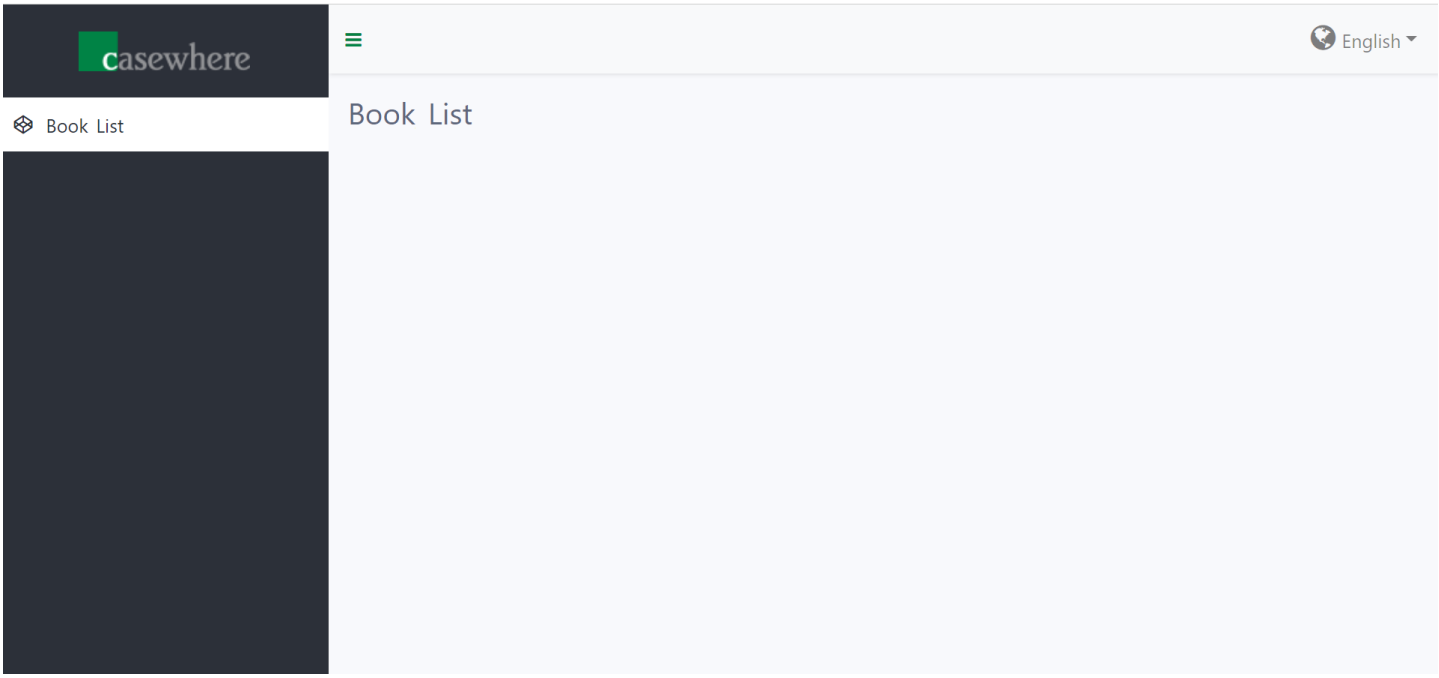
- Step 4. Now you will add you page into the worker site. Let navigate to **Worker Sites** tab.

In the **Page group** window, find and check your worker site "Cw Bookstore Site".



Click **Save**

Good work! Your page is now being added to the worker site, you could check it out at https://academy.casewhere.com/cw-bookstore#!/book-list



You are ready to move into next step!

Create a process

Introduction

You will learn how to create a process for your bookstore application in this tutorial.

This process will be named as "Cw Book Management".

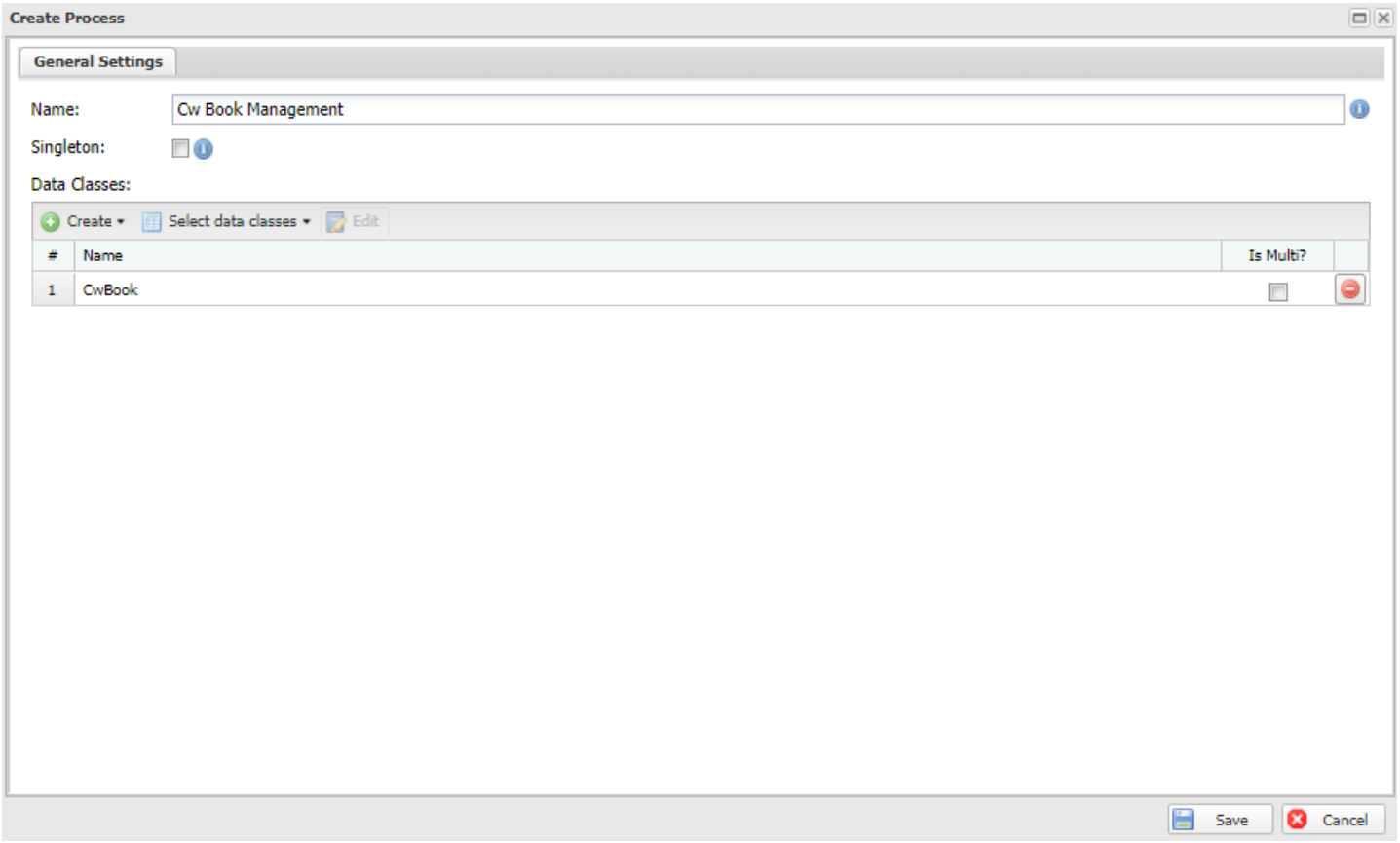
Casewhere will create a new "Cw Book Management" case to contain the book object. Every data object in Casewhere must belong to a case (read Process overview for more).

Guide

- Step 1. In Casewhere Admin site, navigate to **Processes**
- Step 2. Click **Create** button 

The **Create Process** dialog will show up

FIELD NAME	ACTIONS
Name	Fill in "Cw Book Management"
Select data classes	Select "CwBook"



Click **Save**

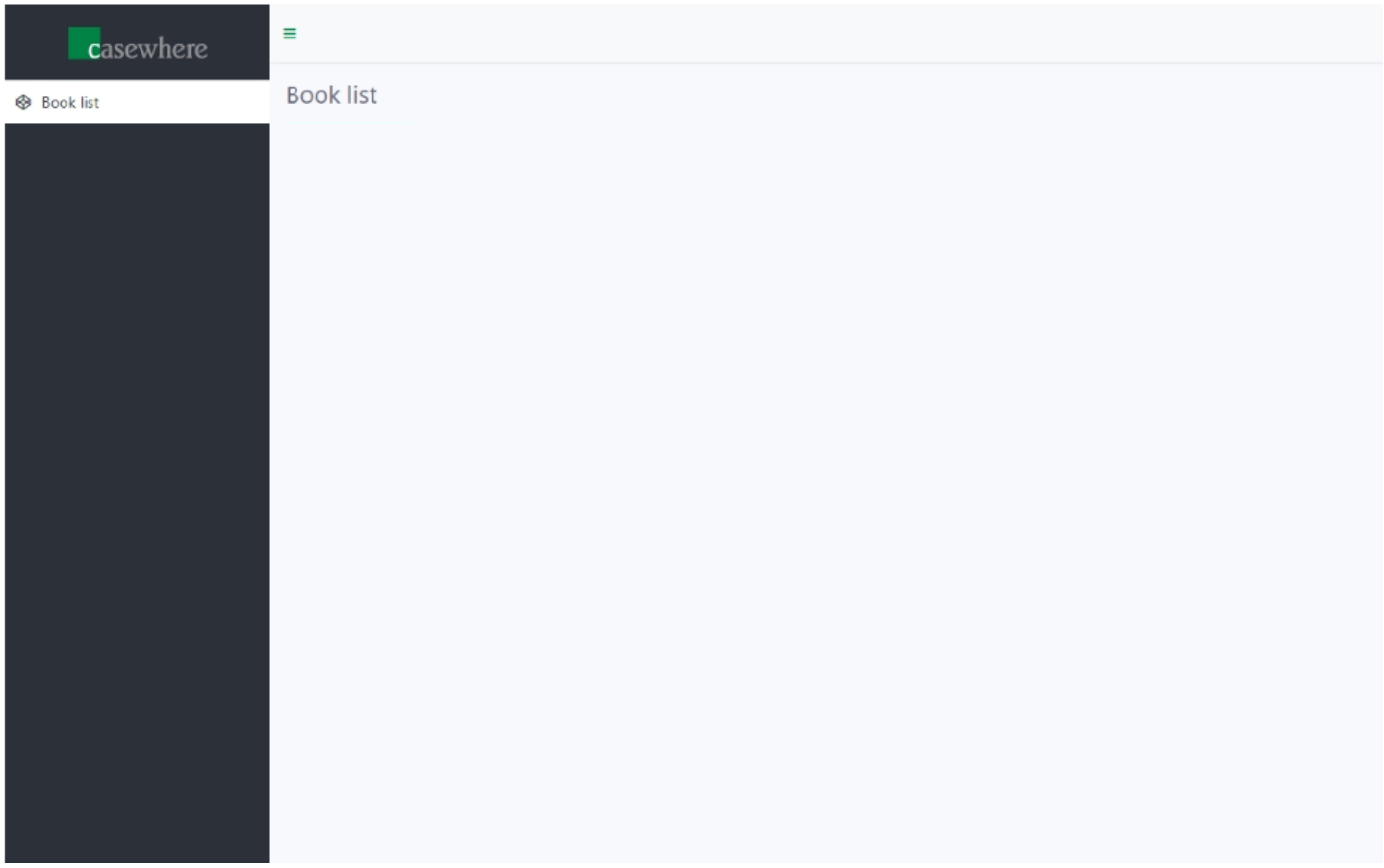
Way to go, now you are ready to move to the next step!

Create a grid widget

Introduction

In this tutorial, you will learn how to create a grid widget.

If you have already follow this tutorial series step by step. You should have the "Book list" page ready and look like the below image



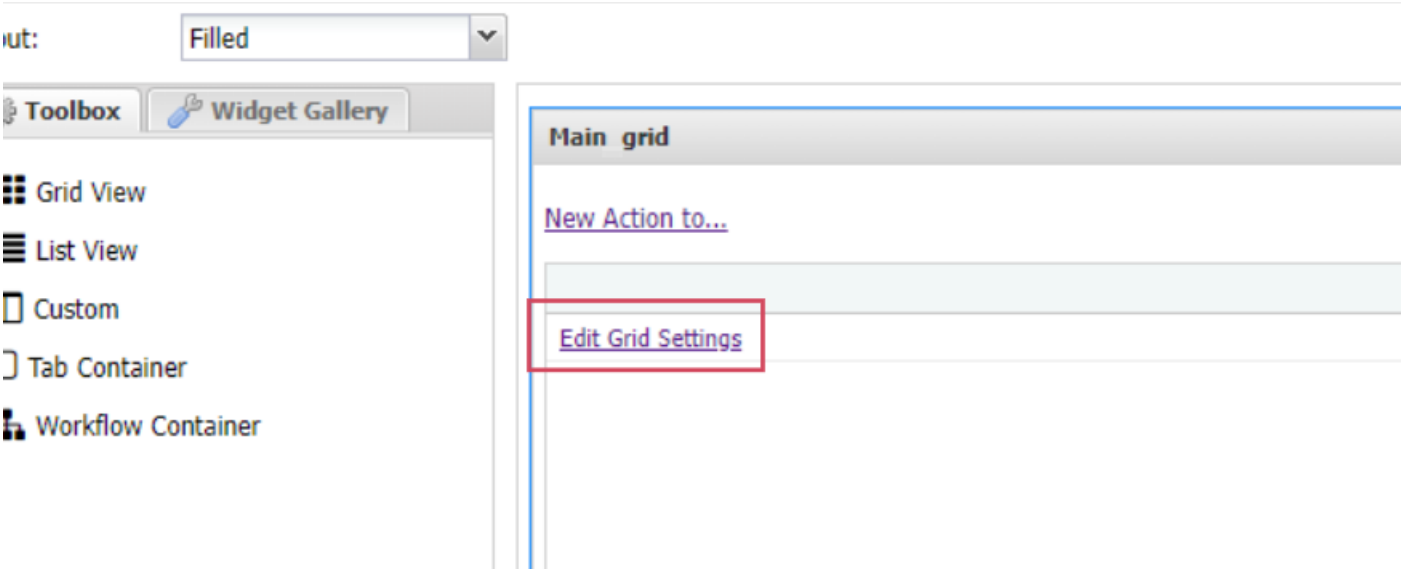
> Tip: The previous steps is required before you create your first workflow definition
>
> 1. Create a product
> 2. Create a data class
> 3. Create a worker site
> 4. Create a page
> 5. Create a process

Guide

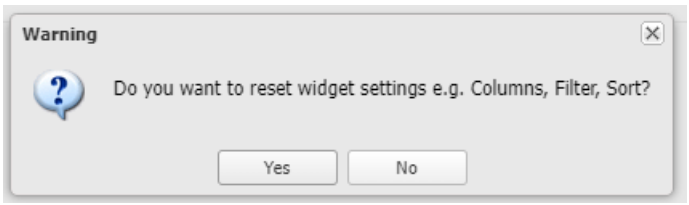
- Step 1. In Casewhere Admin site, navigate to **Pages**

Choose the pages name "book-list" that you already made in create a page step

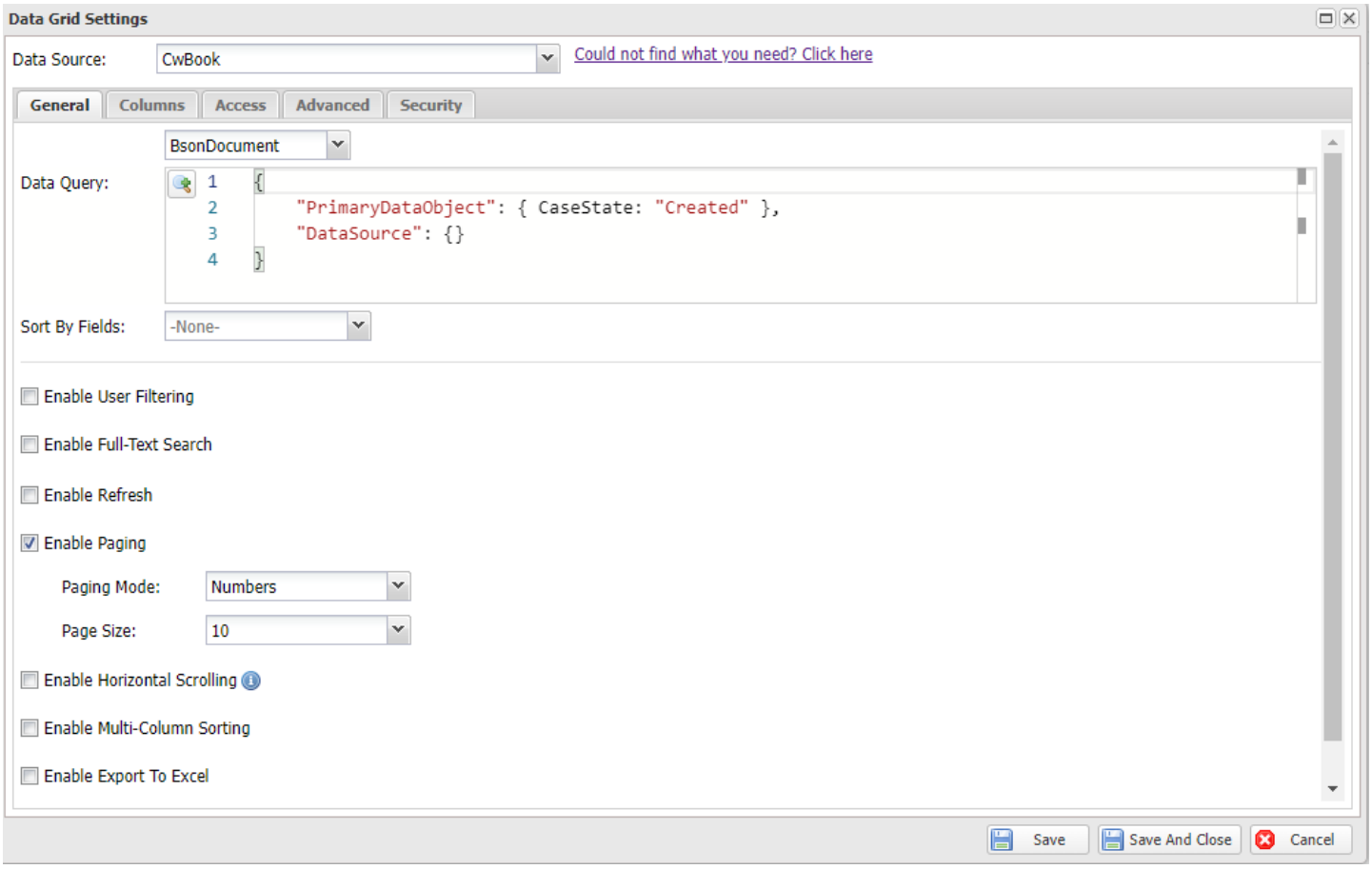
- Step 2. In the **Main grid** widget, click **Edit Grid Setting**



Select **Data Source** "CwBook", choose **Yes** if a warning dialog pop up

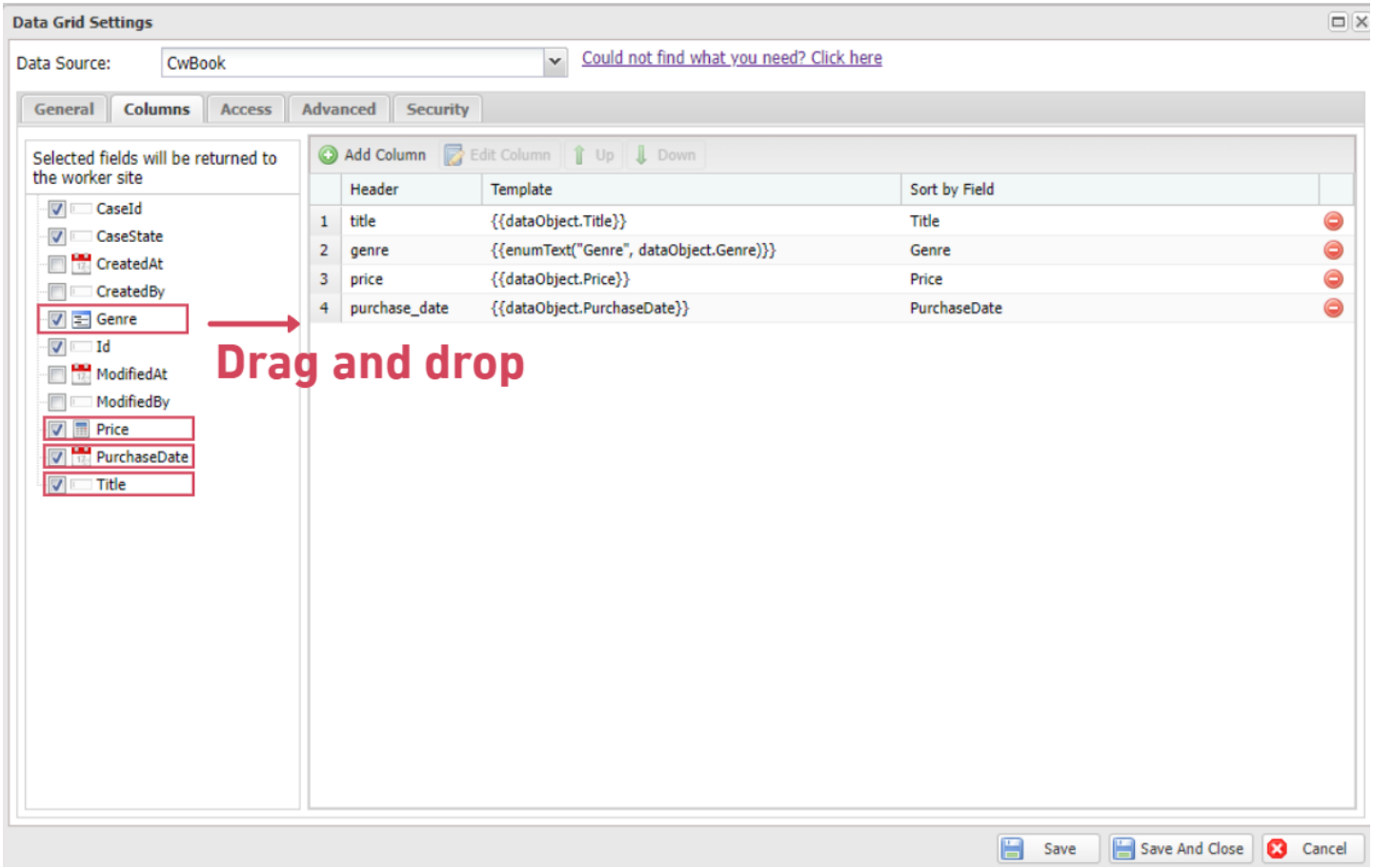


The **Data Query** will load and should look as below



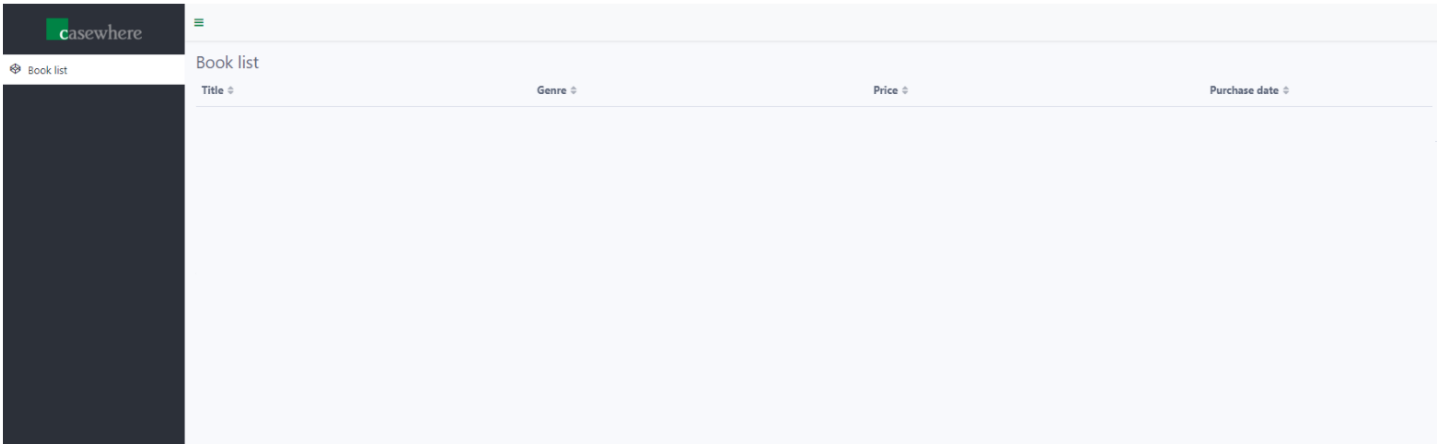
- Step 3. Navigate to **Columns** tab

  Drag and drop these fields into form canvas: Title, Genre, Price, PurchaseDate



Click **Save And Close**

You just added the very first grid to your page



You are ready to go to next step - Create add-object workflow

Create add-object workflow

Introduction

In this tutorial, you will learn how to create a workflow to add a new book to your bookstore.

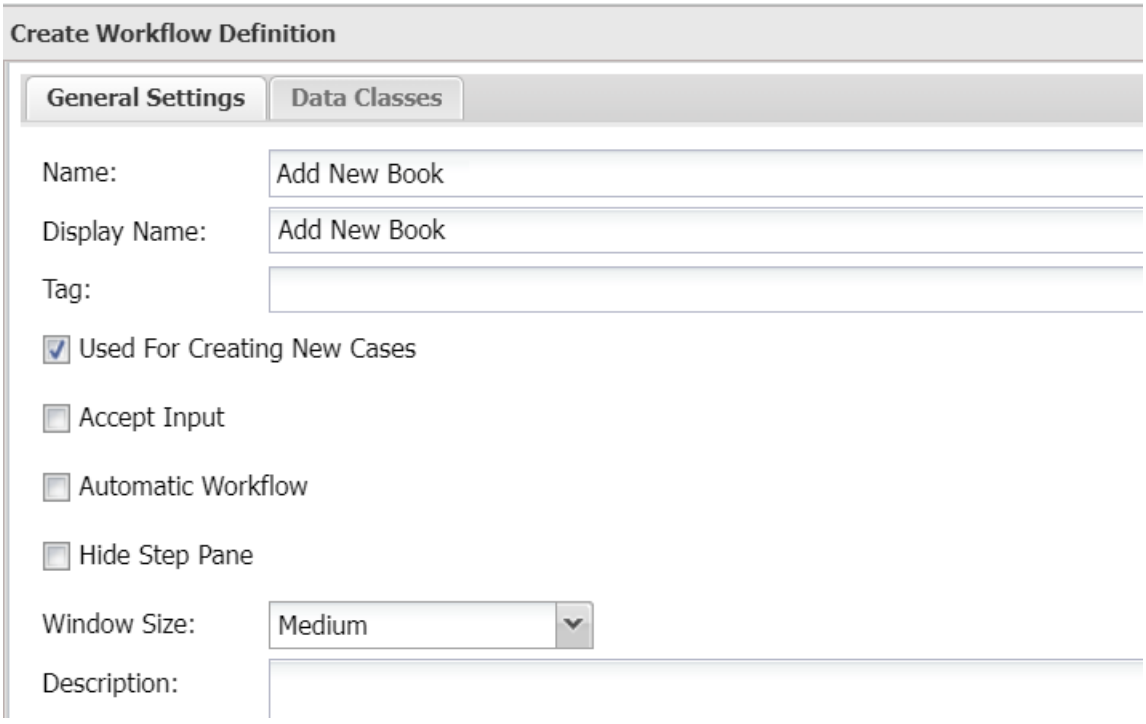
Guide

- Step 1. In Casewhere Admin site, navigate to **Workflow Definition**
- Step 2. Click **Create** button 

The **Create New Workflow definition** dialog will appear

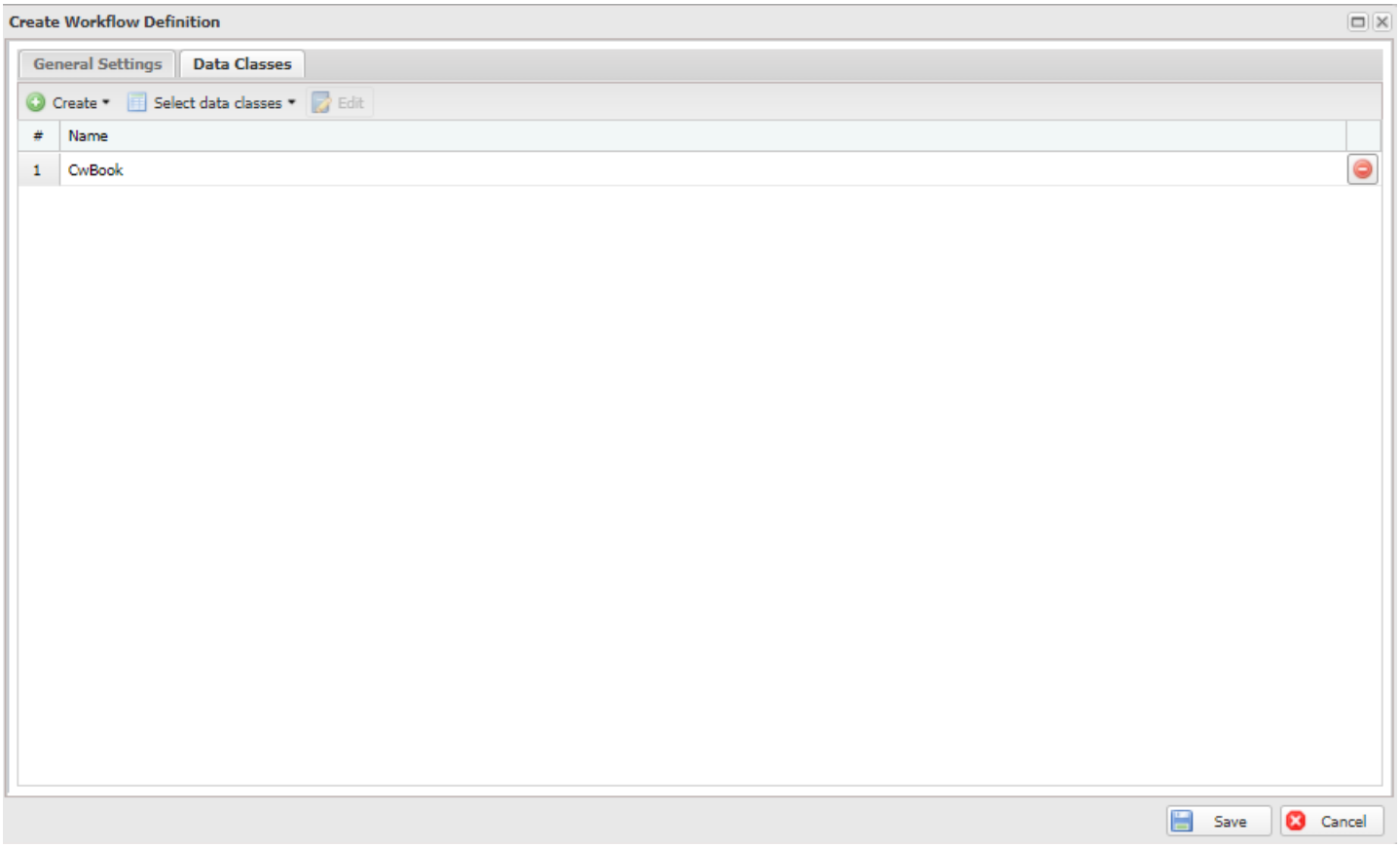
- Step 3. Config **General setting** tab as below

FIELD NAME	ACTIONS
Name	Fill in "Add new book"
Display Name	Fill in "Add new book"
Used For Creating New Cases	☑
Window Size	Select **Medium**



- Step 4. Switch to **Data classes** tab

Click **Select data classes** drop-down, and select "CwBook"

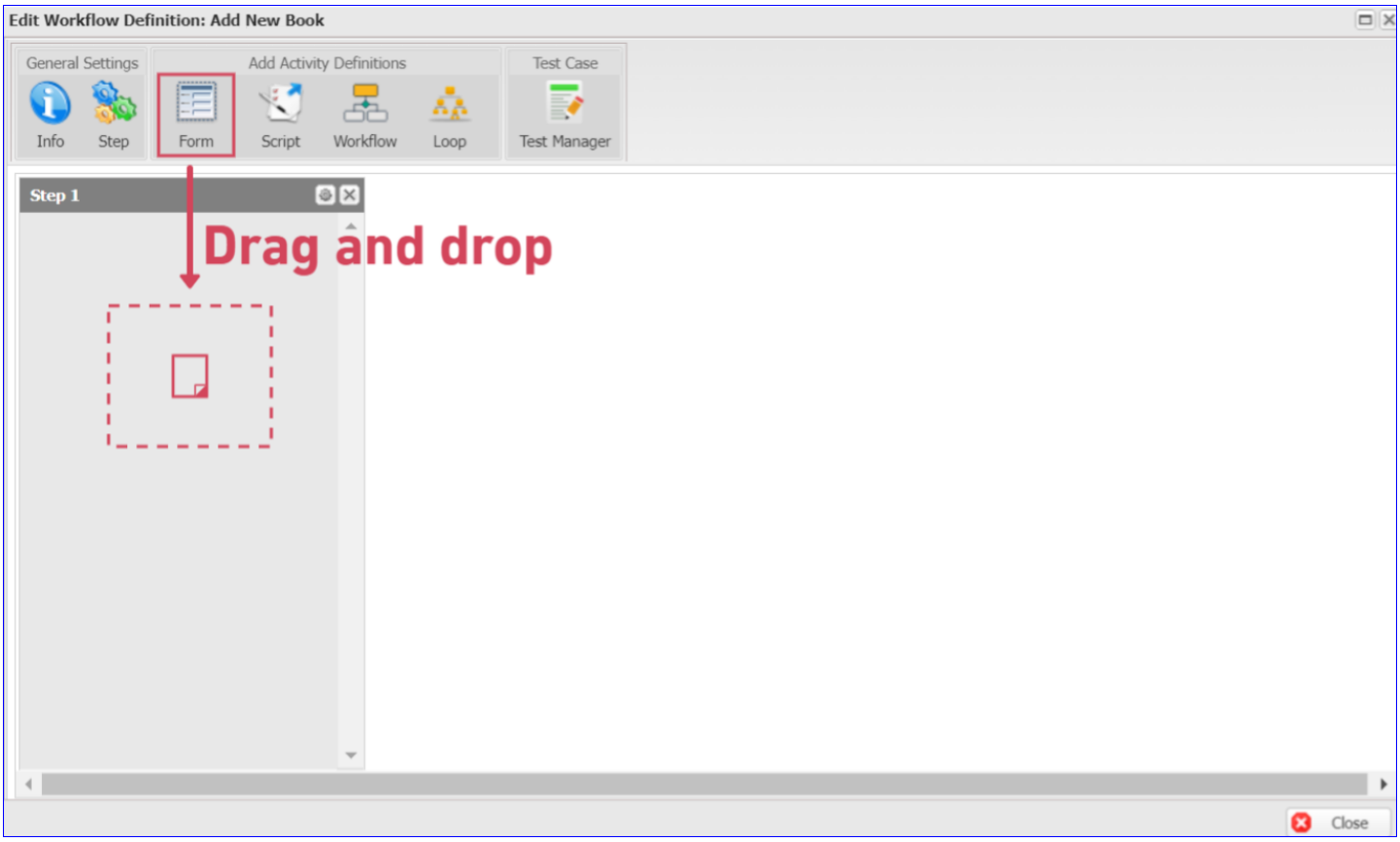


Tip: The data classes you just select is the one you made in Create a data class step

Click **Save**

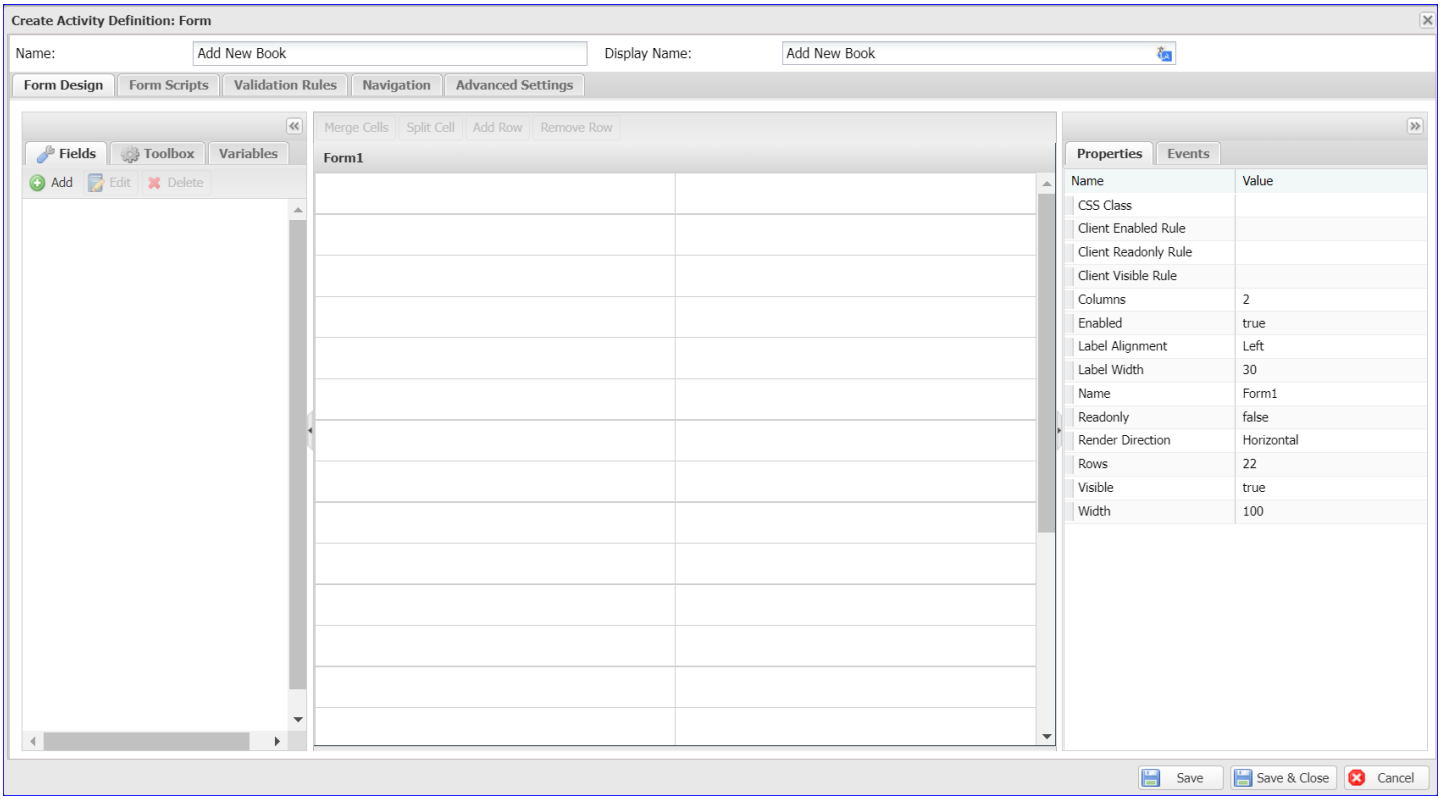
- Step 5. **Edit Workflow Definition: Add new book** dialog will display

Drag and drop **Form** activity definitions from **Add Activity Definitions** panel to the **Step 1** container



- Step 6: **Create Activity Definition: Form** will show up.

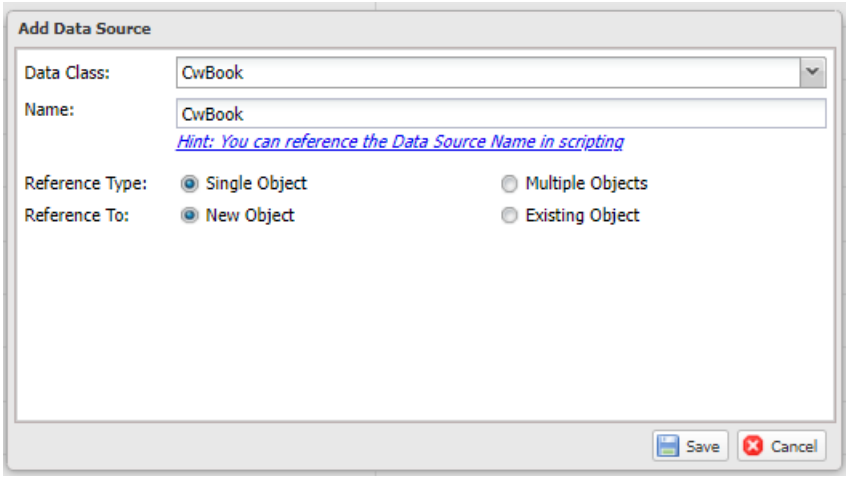
FIELD NAME	ACTIONS
Name	Fill in "Add new book"
Display Name	Fill in "Add new book"



- Step 7. Click **Add** button 

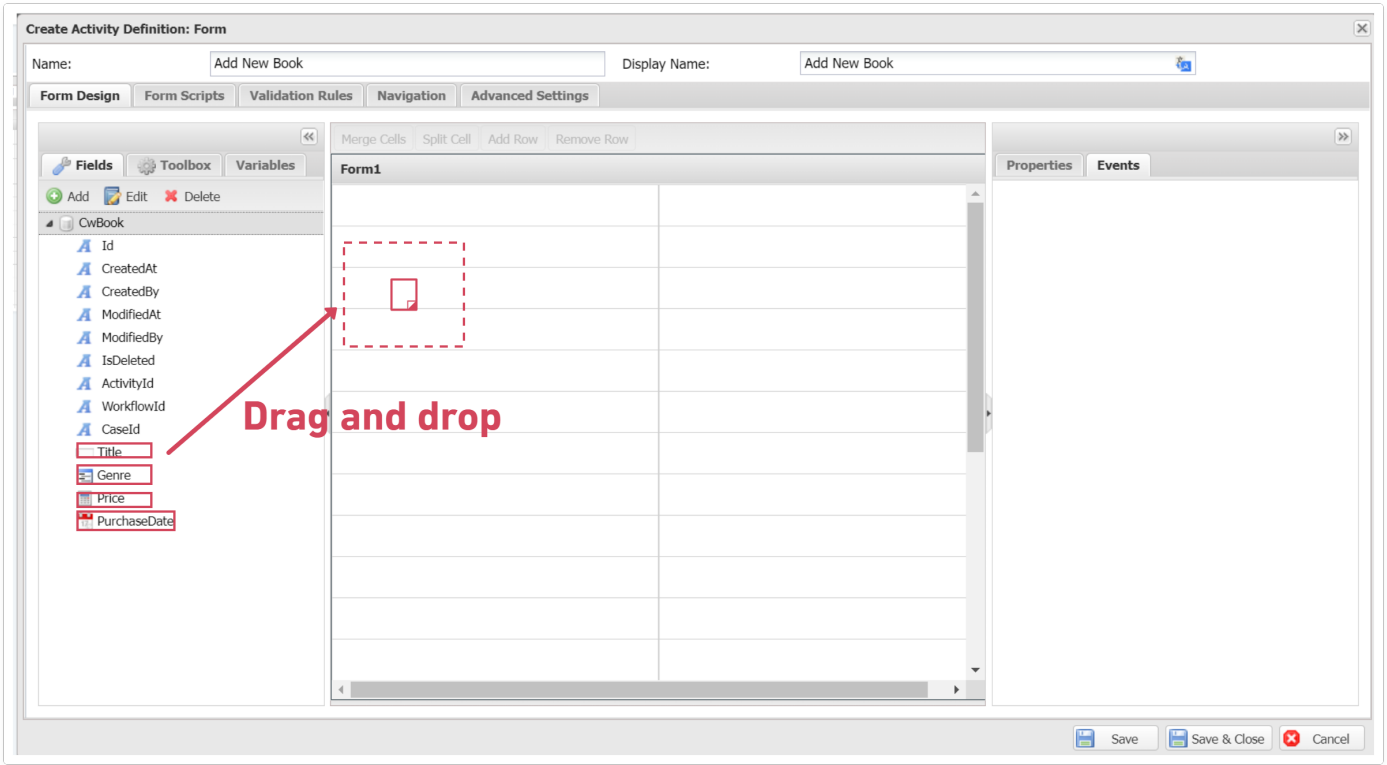
to open the **Add Data source** dialog.

FIELD NAME	ACTIONS
Data Class	Select "CwBook"
Name	CwBook
Reference Type field	Select "Single Object"
Reference To	Select "New Object"

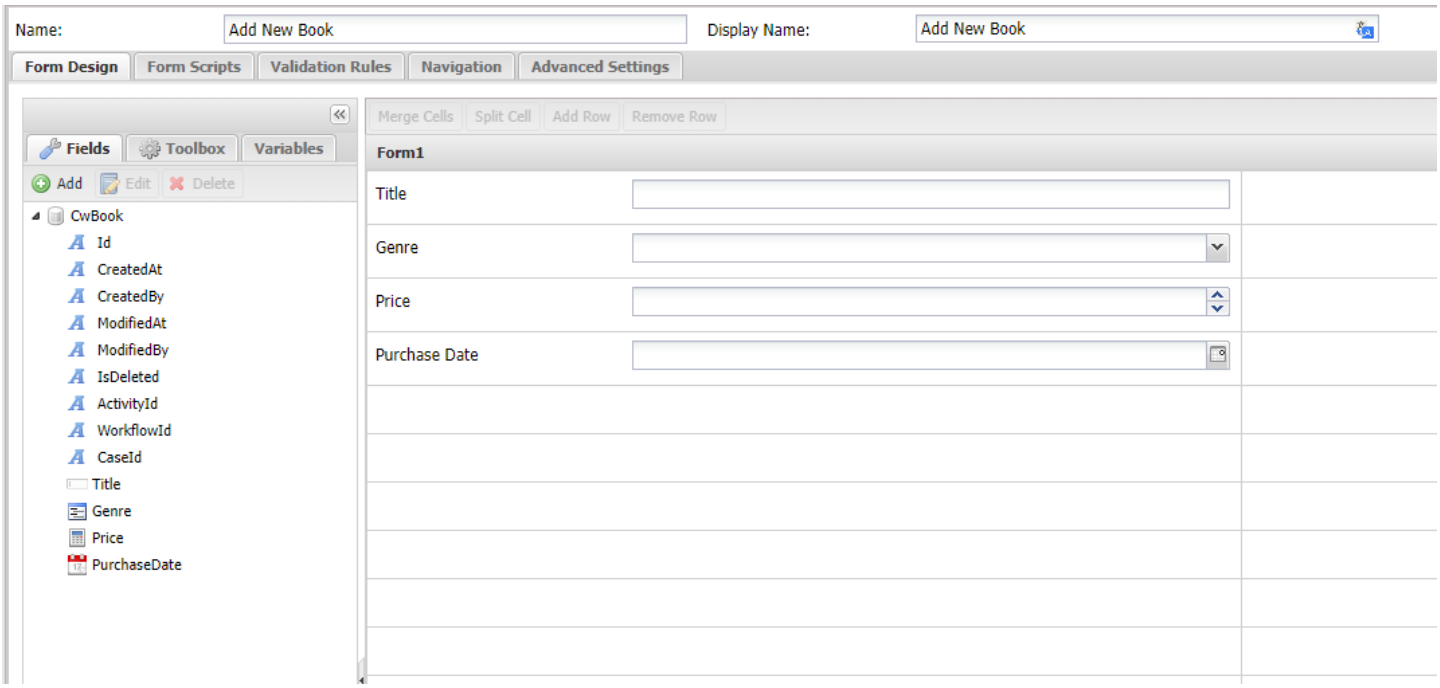


Click **Save**

- Step 8. Drag and drop these fields into form canvas:
  - Title
  - Genre
  - Price
  - PurchaseDate



After drad and drop all 4 fields as show below, click on **Save & Close**.



Good job! You are ready to use this workflow definition in your next step - Create a widget button

Create a widget button

Introduction

Welcome to the tutorial where you will create your very first interactive buttons with Casewhere.

In this tutorial, you will learn how to make three button named "Add New Book", "Edit Book", and "Delete Book" on the top of your page interface.

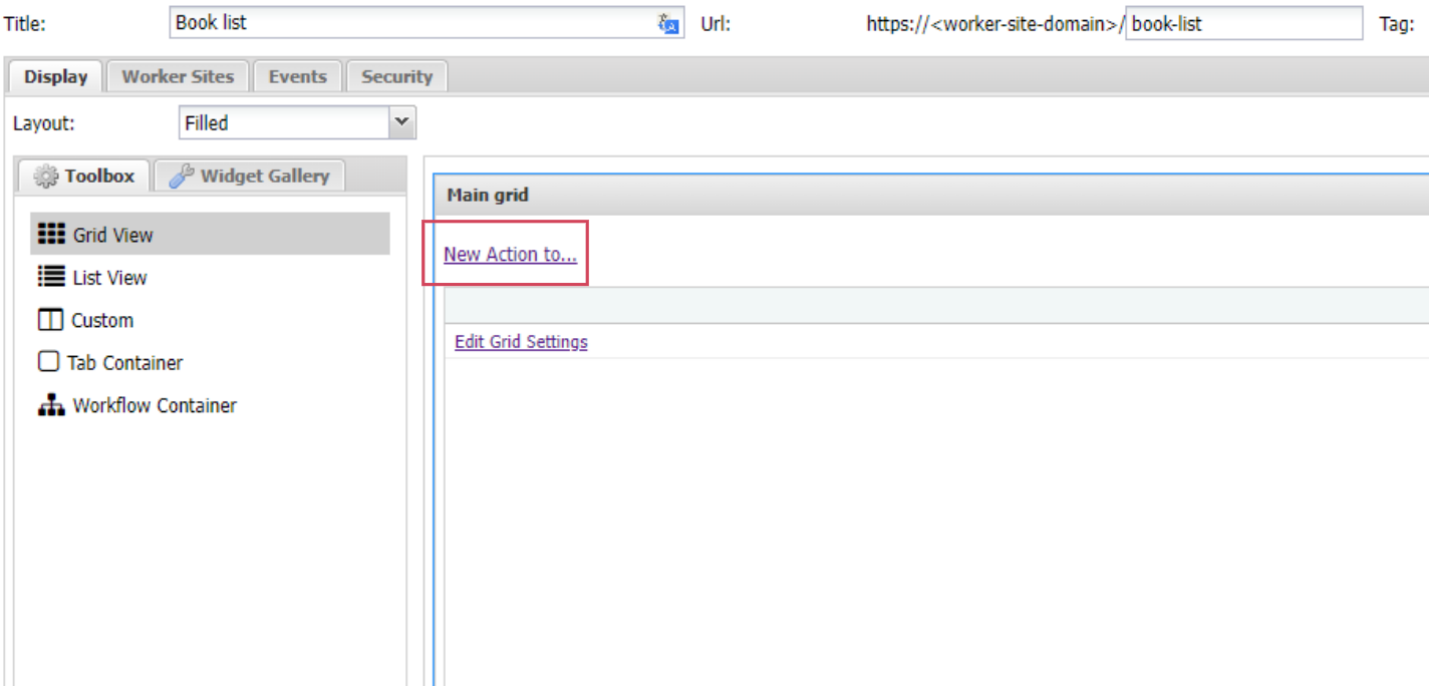
Every time a user clicks those button, a pop-up will appear and allow the user to fill in a new record of book's info with title, genre, price, and purchase date.

Guide

- Step 1. In Casewhere Admin site, navigate to **Pages**

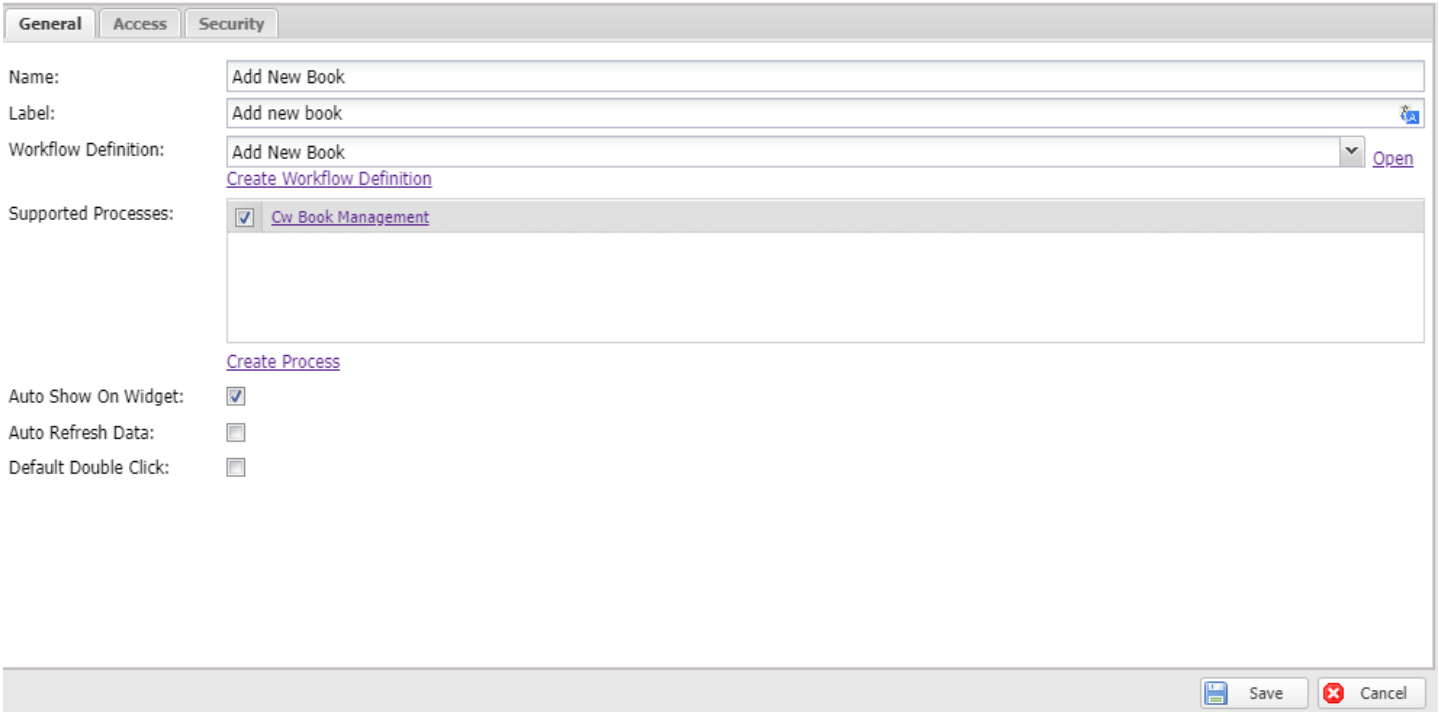
Choose the pages name "book-list" that you already made in create a page step.

- Step 2. In the **Main grid** widget, click **New Action to...**

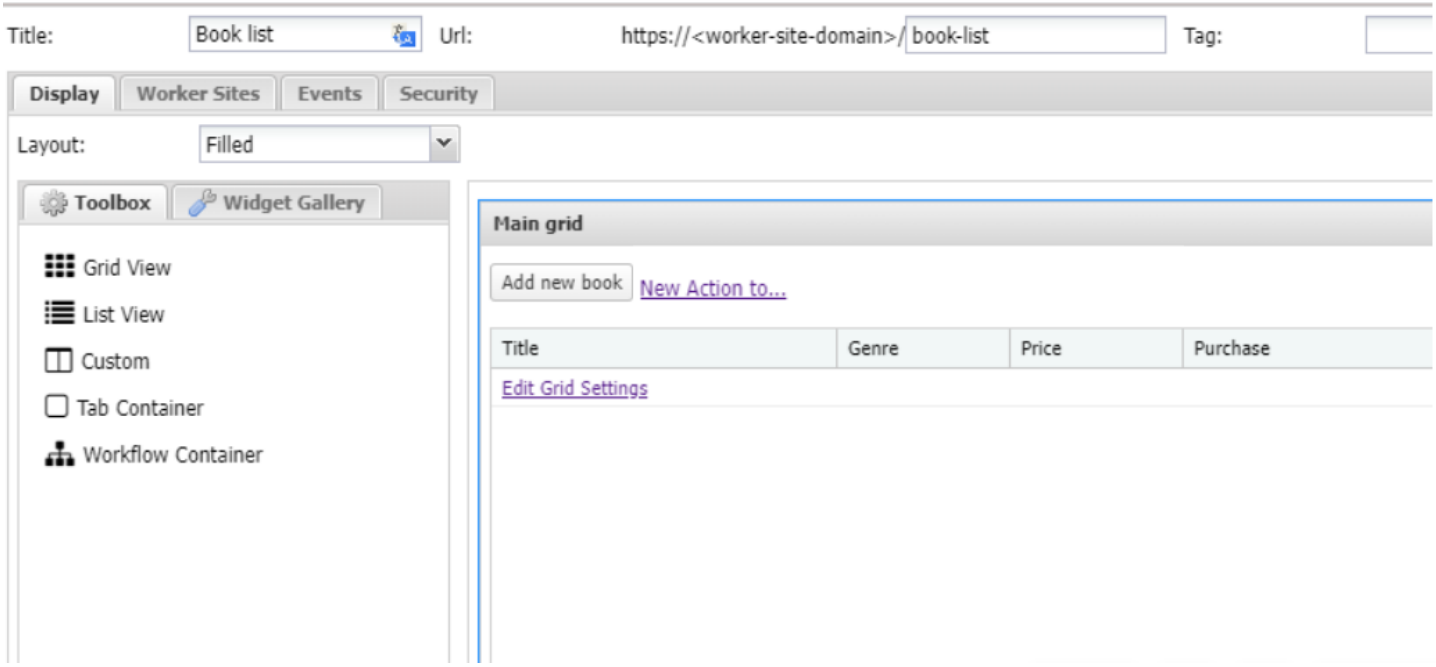


Select **Trigger a workflow**

FIELD NAME	ACTIONS
Name	Fill in "Add New Book"
Label	Fill in "Add new book"
Workflow Definition	Select "Add New Book"
Supported Processes	Select "Cw Book Management"

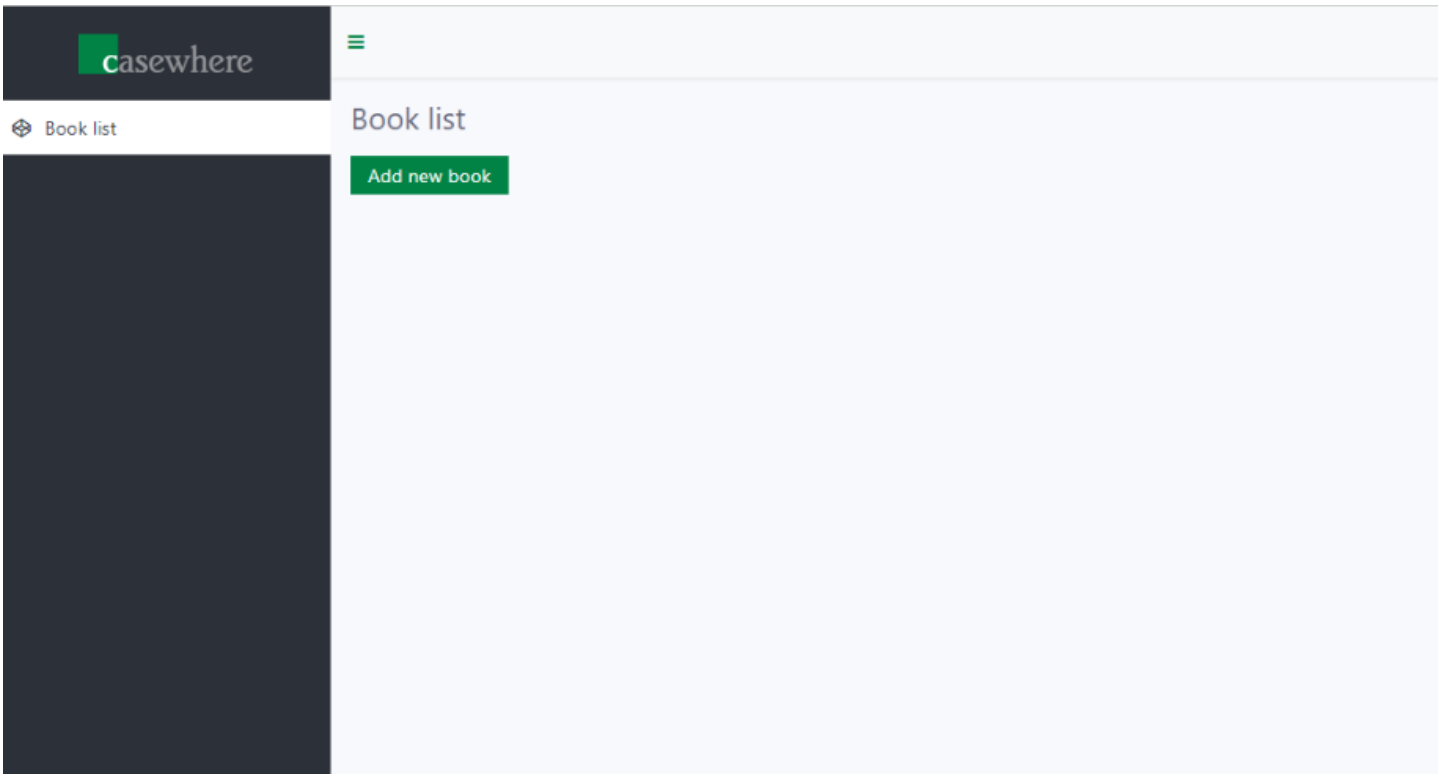


Click **Save**



Voilà! You just added the very first button to your page.

Reload your page and you should see the button now.



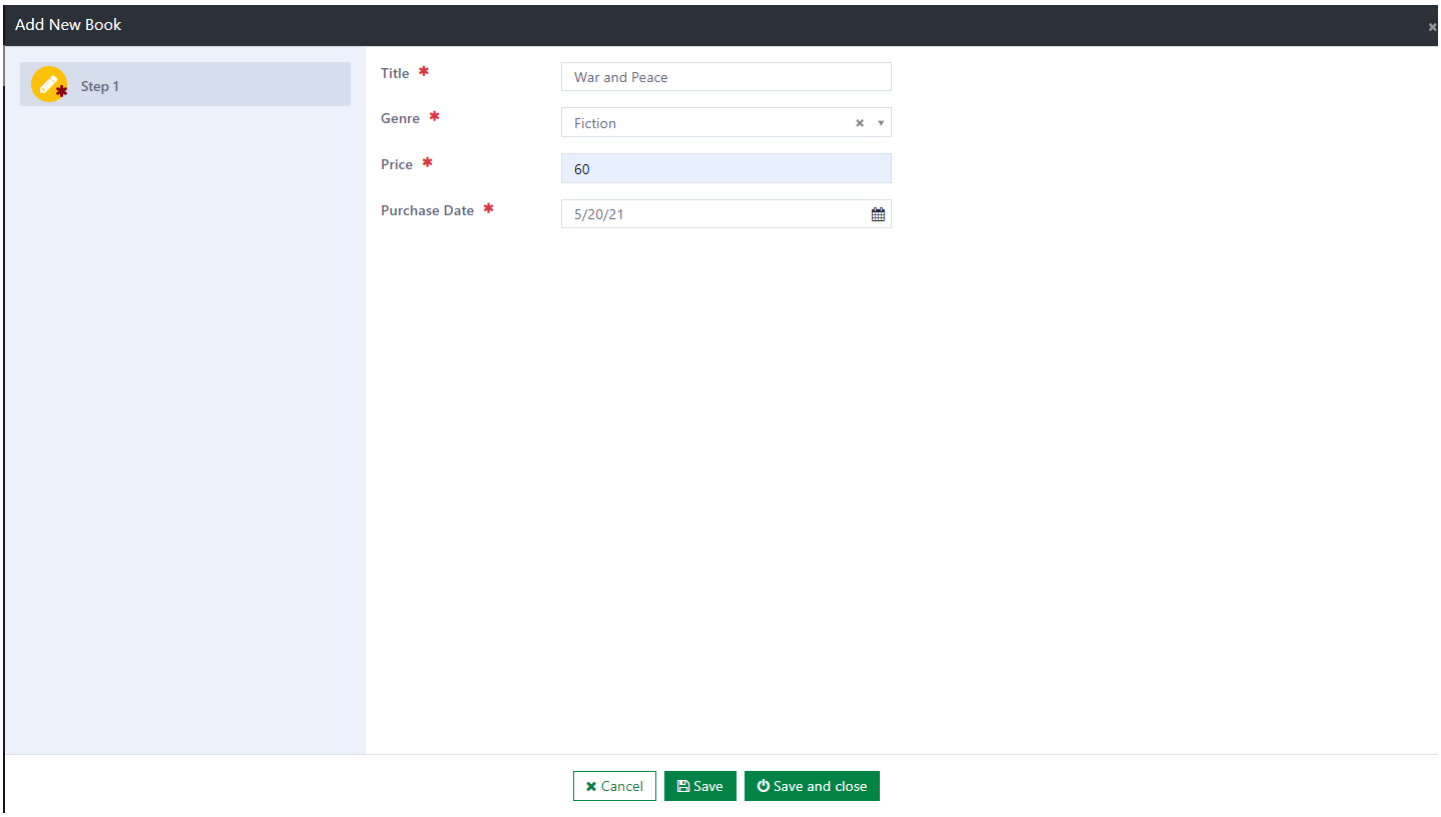
Test

- Step 1: Reload your page

  Click **Add New Book** button

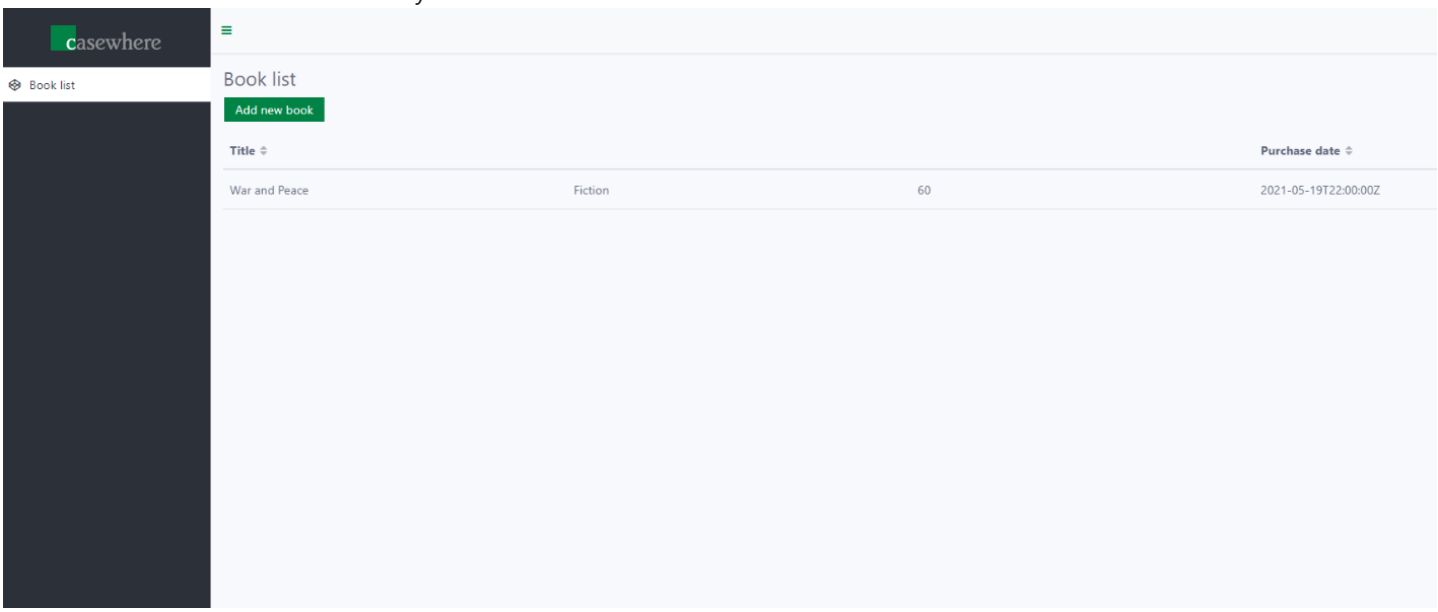
- Step 2: Fill in a data set

TITLE	GENRE	PRICE	PURCHASE DATE
War and Peace	Fiction	60.000	20/05/2021

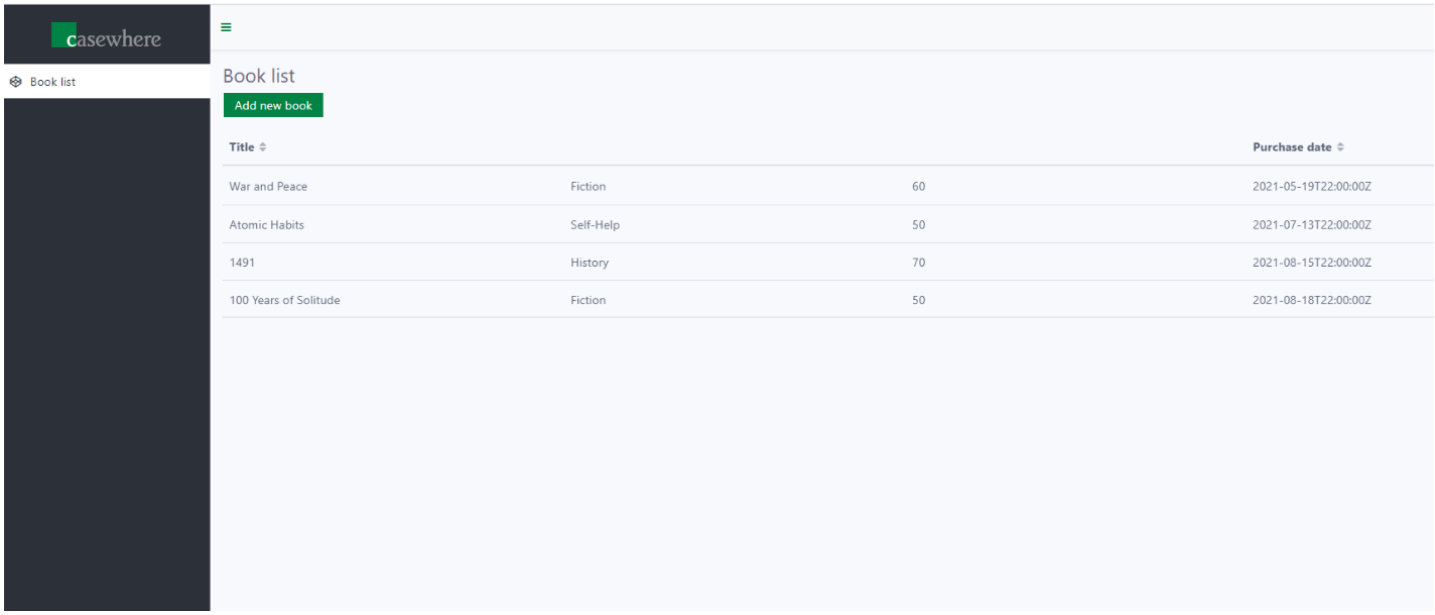


Click **Save and close**

Voilà! "War and Peace" was added in your book list.



Try it! Add a few more books to your list.



Ready to continue? You are ready to move to the next step - Create edit-object workflow

Create edit-object workflow

Introduction

In previous tutorials, you already learn how to create add-object workflow and also how to create a button to trigger the workflow.

In this tutorial, you will continue to define a edit-object workflow called "Edit book."

This edit action, as well as add and delete, are the fundamental functions that any application should be able to do.

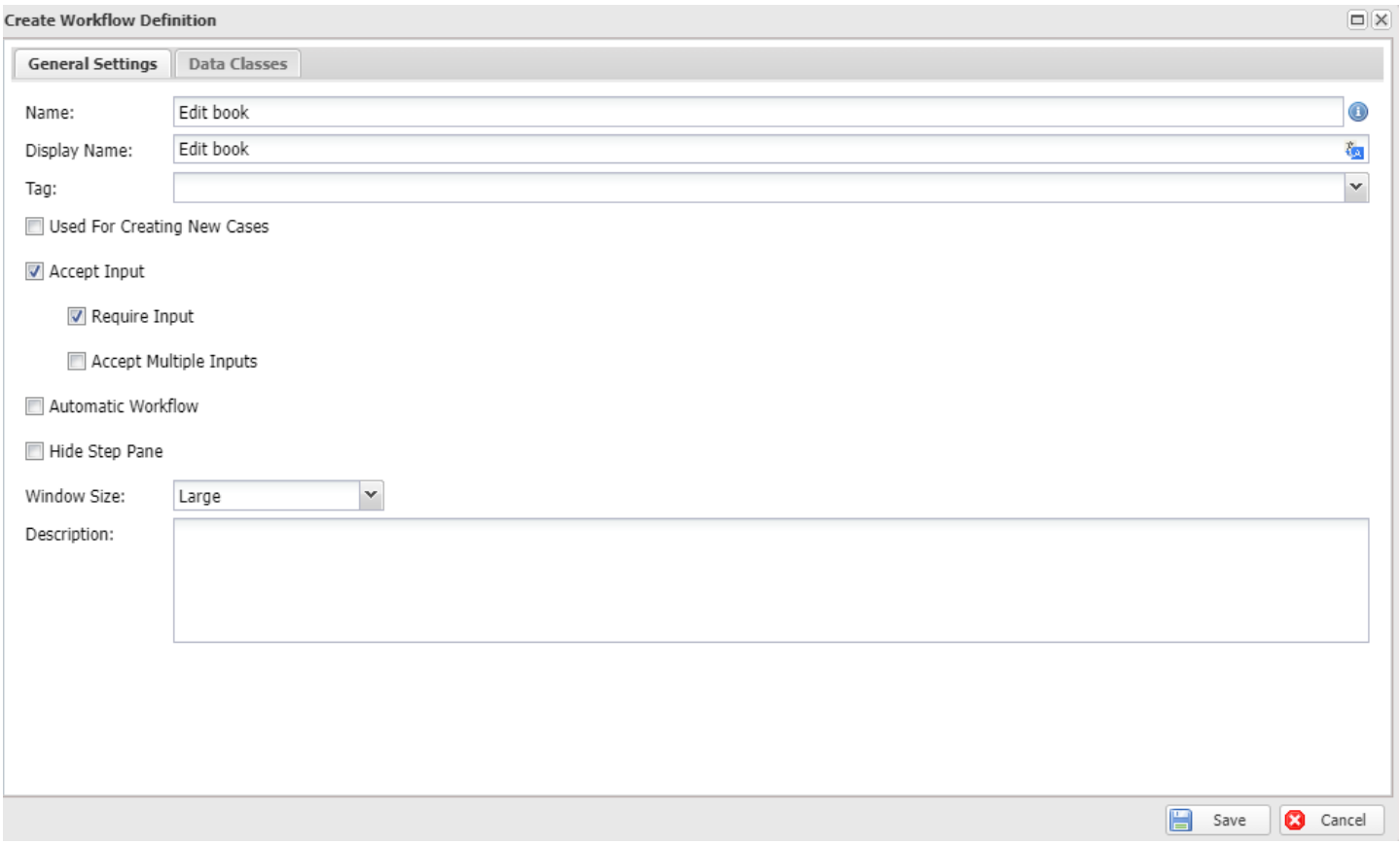
Guide

- Step 1. In Casewhere Admin site, navigate to **Workflow Definition**
- Step 2. Click **Create** button 

The **Create New Workflow definition** dialog will appear

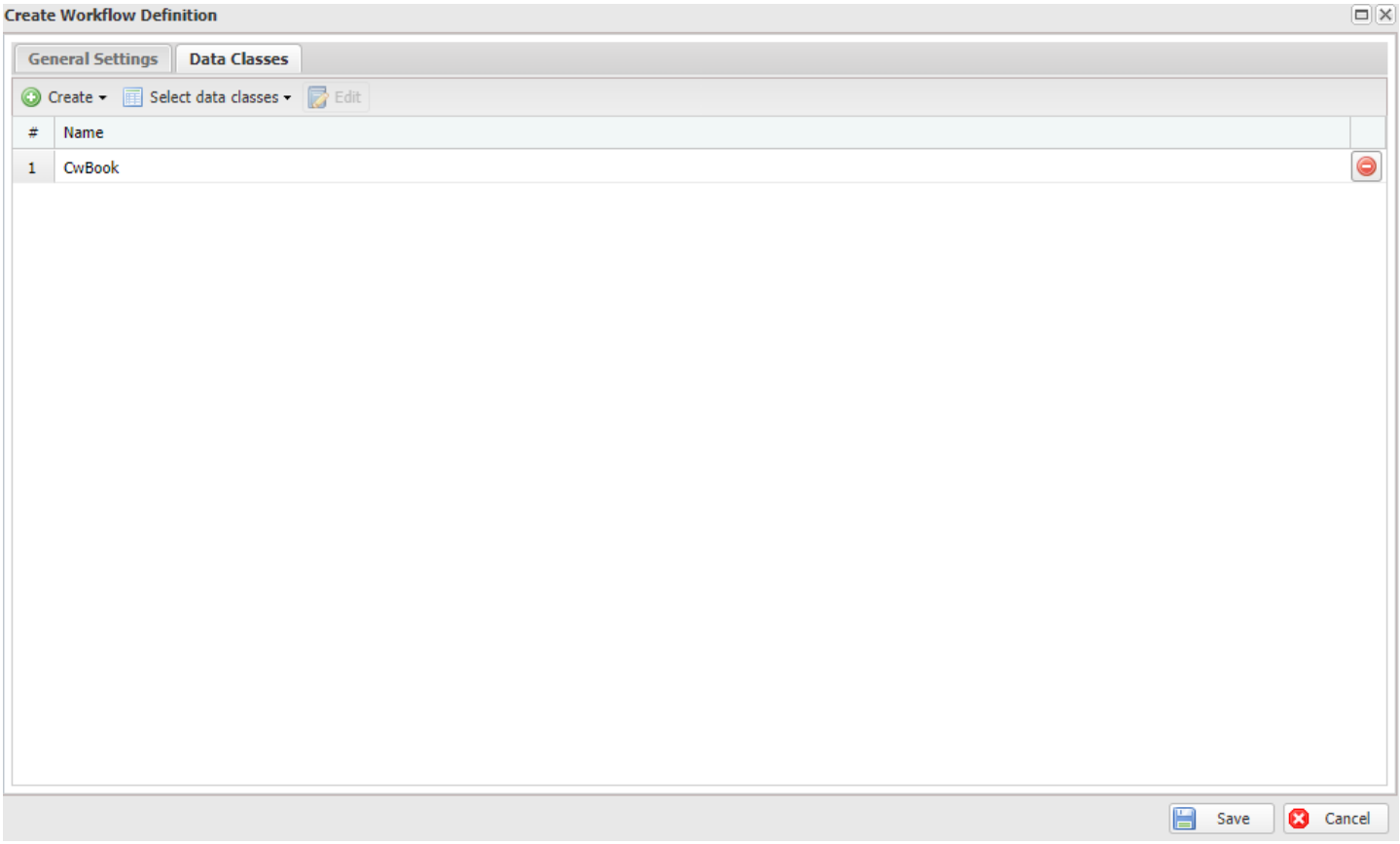
- Step 3. Config **General setting** tab as below

FIELD NAME	ACTIONS
Name	Fill in "Edit book"
Display Name	Fill in "Edit book"
Used For Creating New Cases	Make sure this field is uncheck ☐
Accept Input	☑
Require Input	☑
Window Size	Select **Medium**



- Step 4. Switch to **Data classes** tab

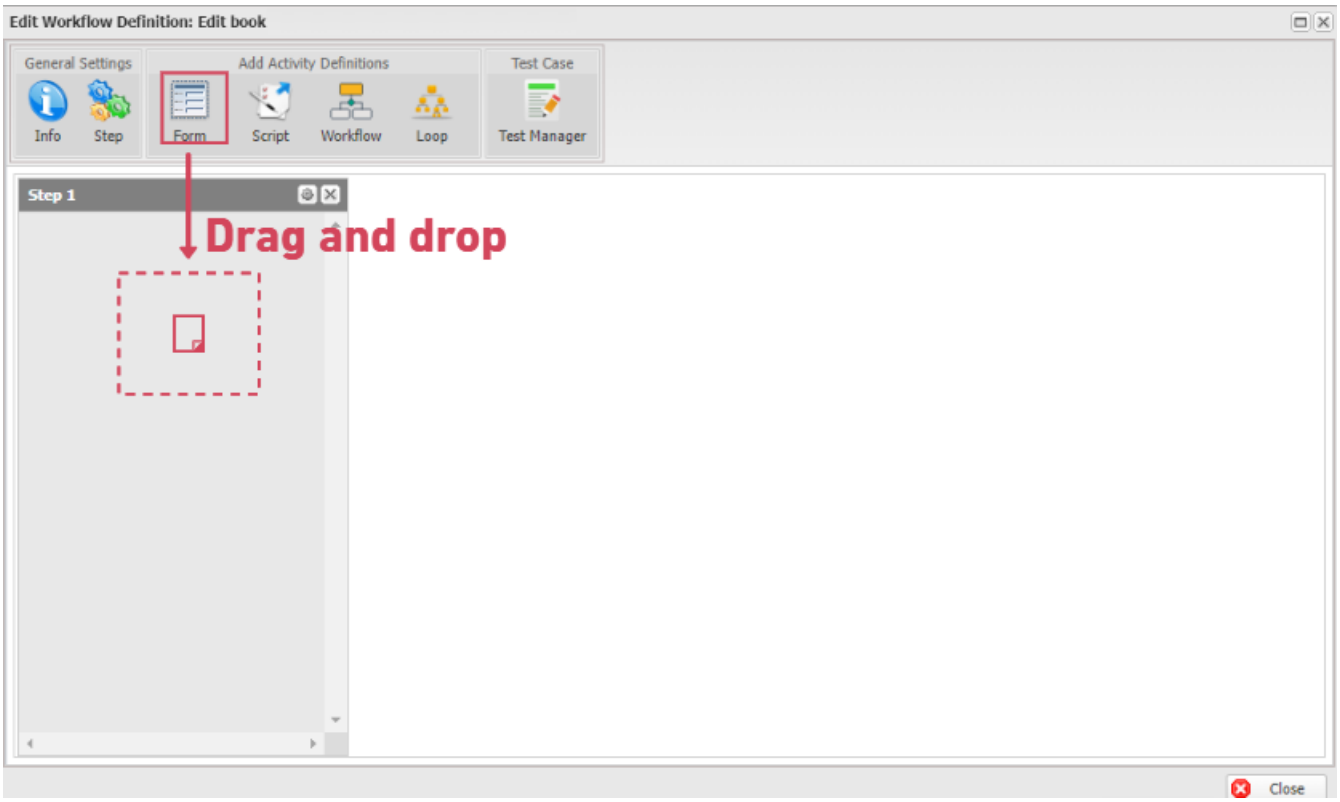
Click **Select data classes** drop-down, and select "CwBook"



Click **Save**

- Step 5. **Edit Workflow Definition: Edit book** dialog will display

Drag and drop **Form** activity definitions from **Add Activity Definitions** panel to the **Step 1** container



- Step 6: **Create Activity Definition: Form** will show up.

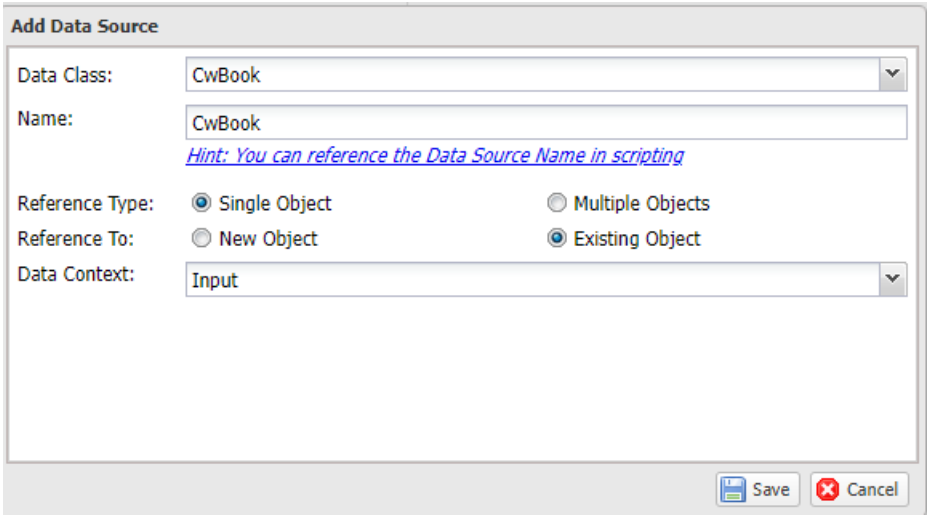
FIELD NAME	ACTIONS
Name	Fill in "Edit book"
Display Name	Fill in "Edit book"



- Step 7. Click **Add** button 

to open the **Add Data source** dialog.

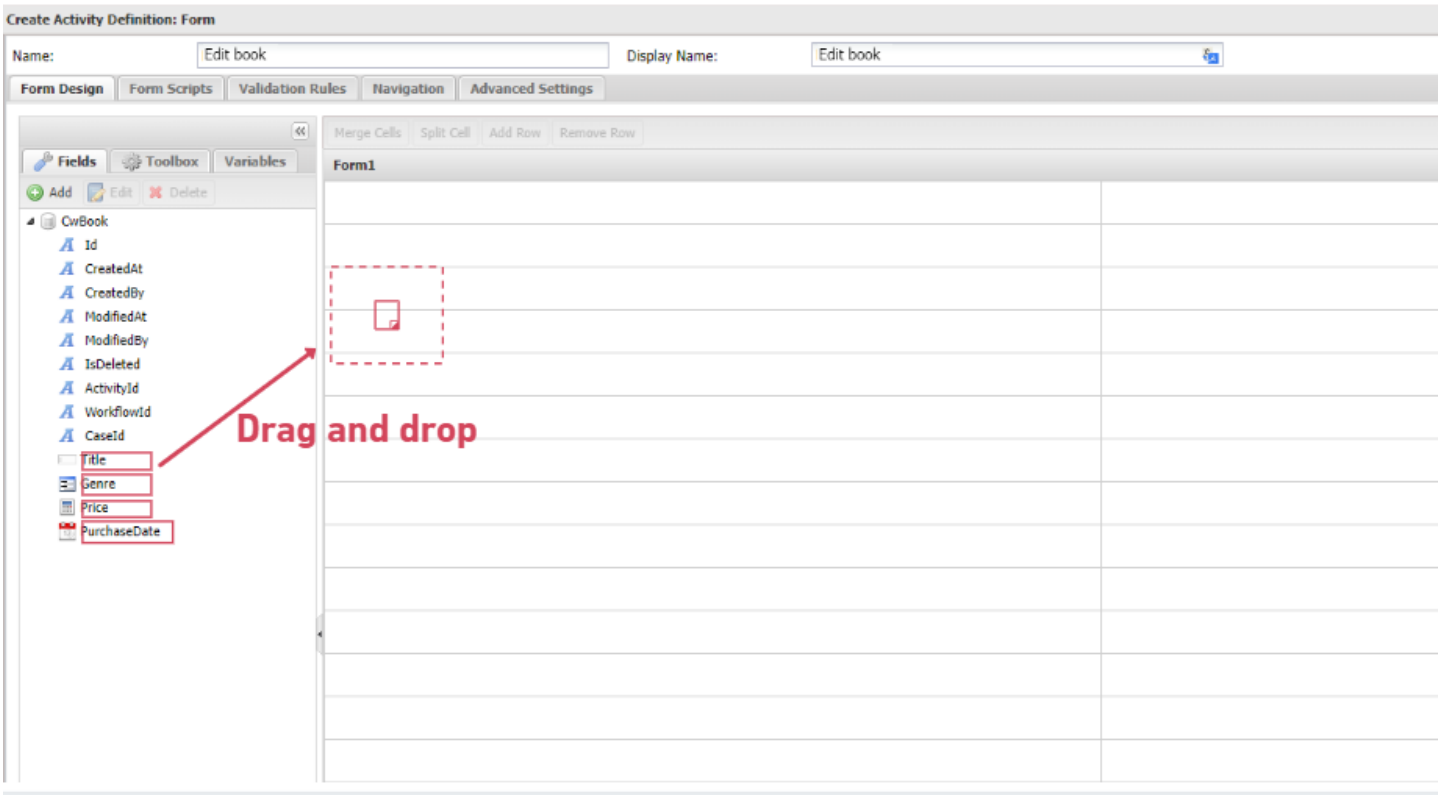
FIELD NAME	ACTIONS
Data Class	Select "CwBook"
Name	CwBook
Reference Type field	Select "Single Object"
Reference To	Select "Existing Object"
Data context	Input



Click **Save**

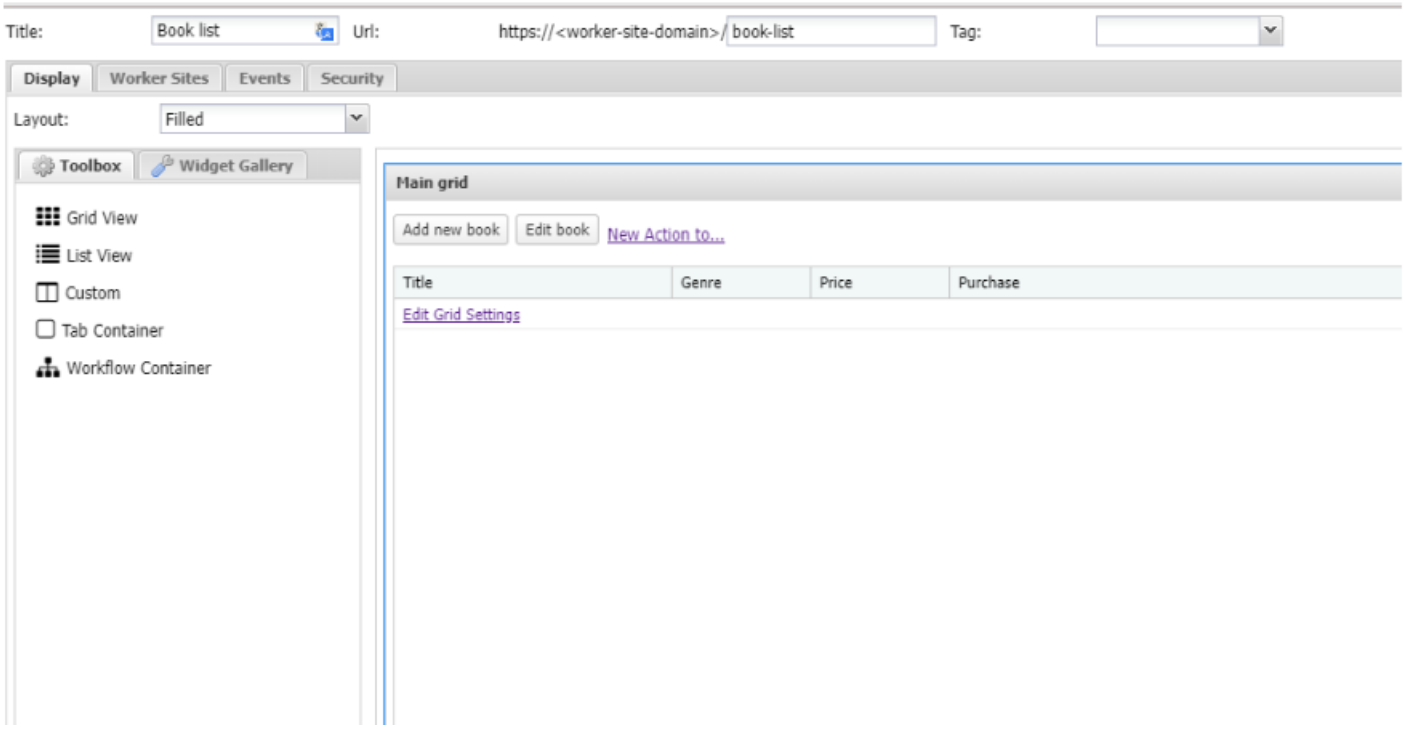
- Step 8. Drag and drop these fields into form canvas:
  - Title
  - Genre
  - Price
  - PurchaseDate

After drag and drop all 4 fields as show below, click on **Save & Close**.

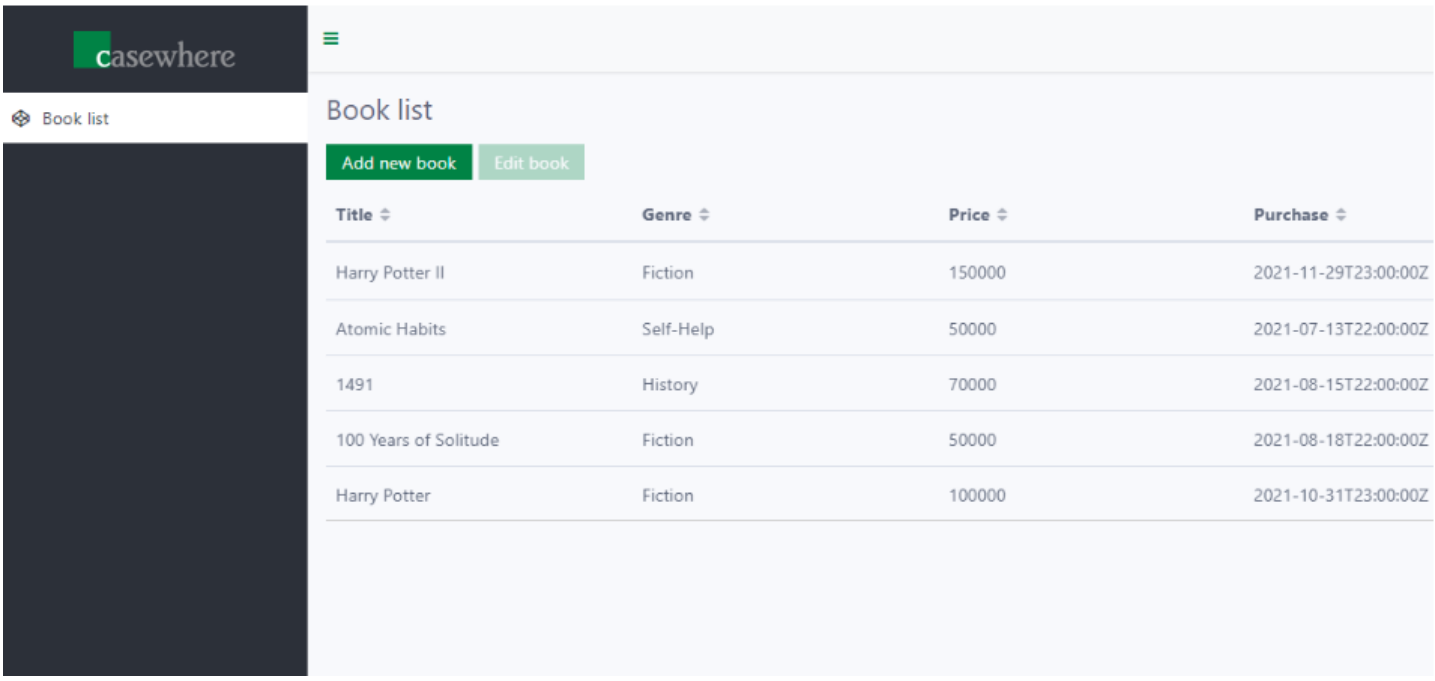


- Step 9. Create a button to trigger the workflow

Follow this guide to create and test a button to make "Edit book" button



- Step 10. Reload your page to see the button on-live



Good job! You are ready to move to the next step - Create delete-object workflow

Create delete-object workflow

Introduction

In this tutorial, you will learn how to create a workflow to delete books from your bookstore database.

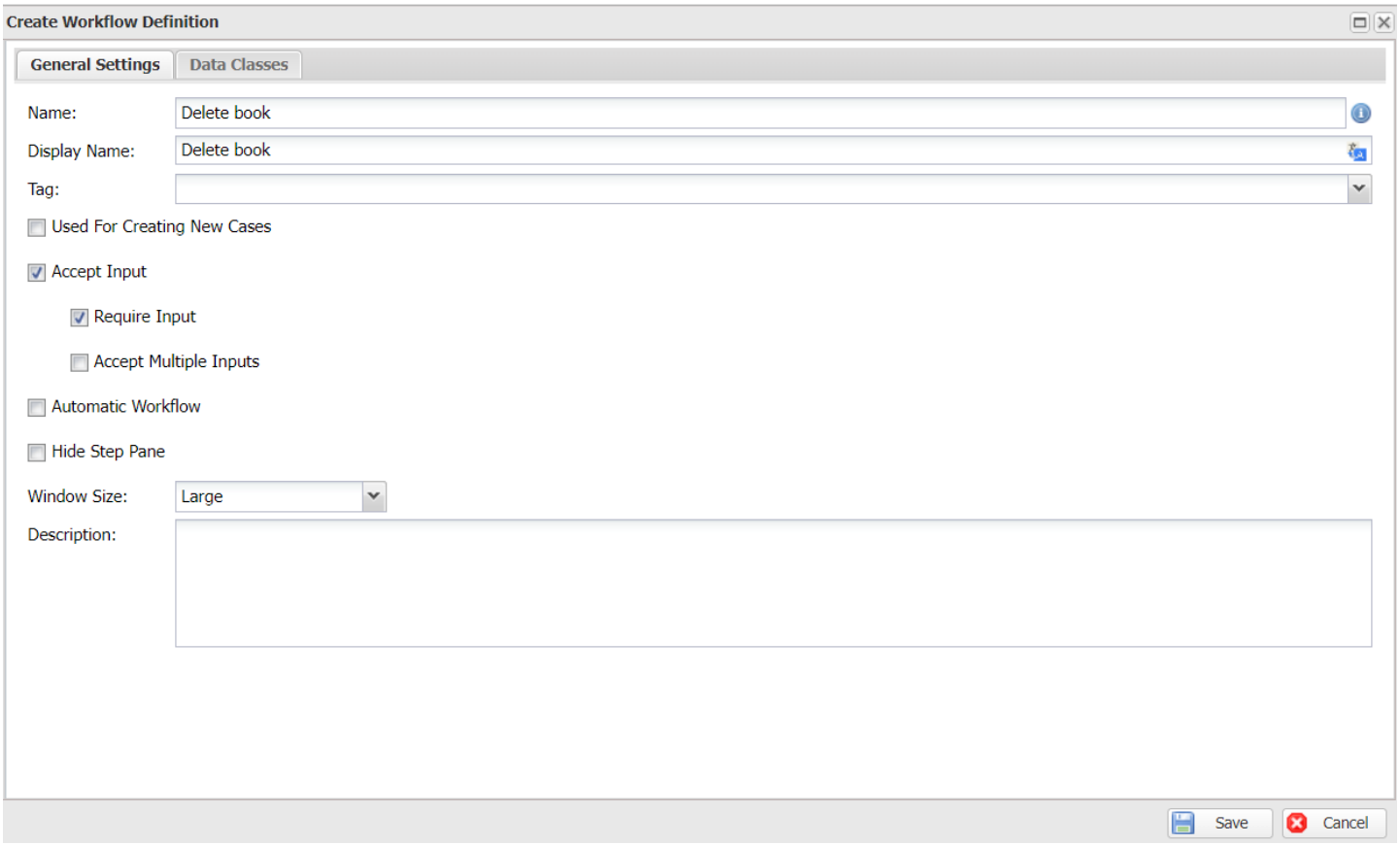
Guide

- Step 1. In Casewhere Admin site, navigate to **Workflow Definition**
- Step 2. Click **Create** button 

The **Create New Workflow definition** dialog will appear

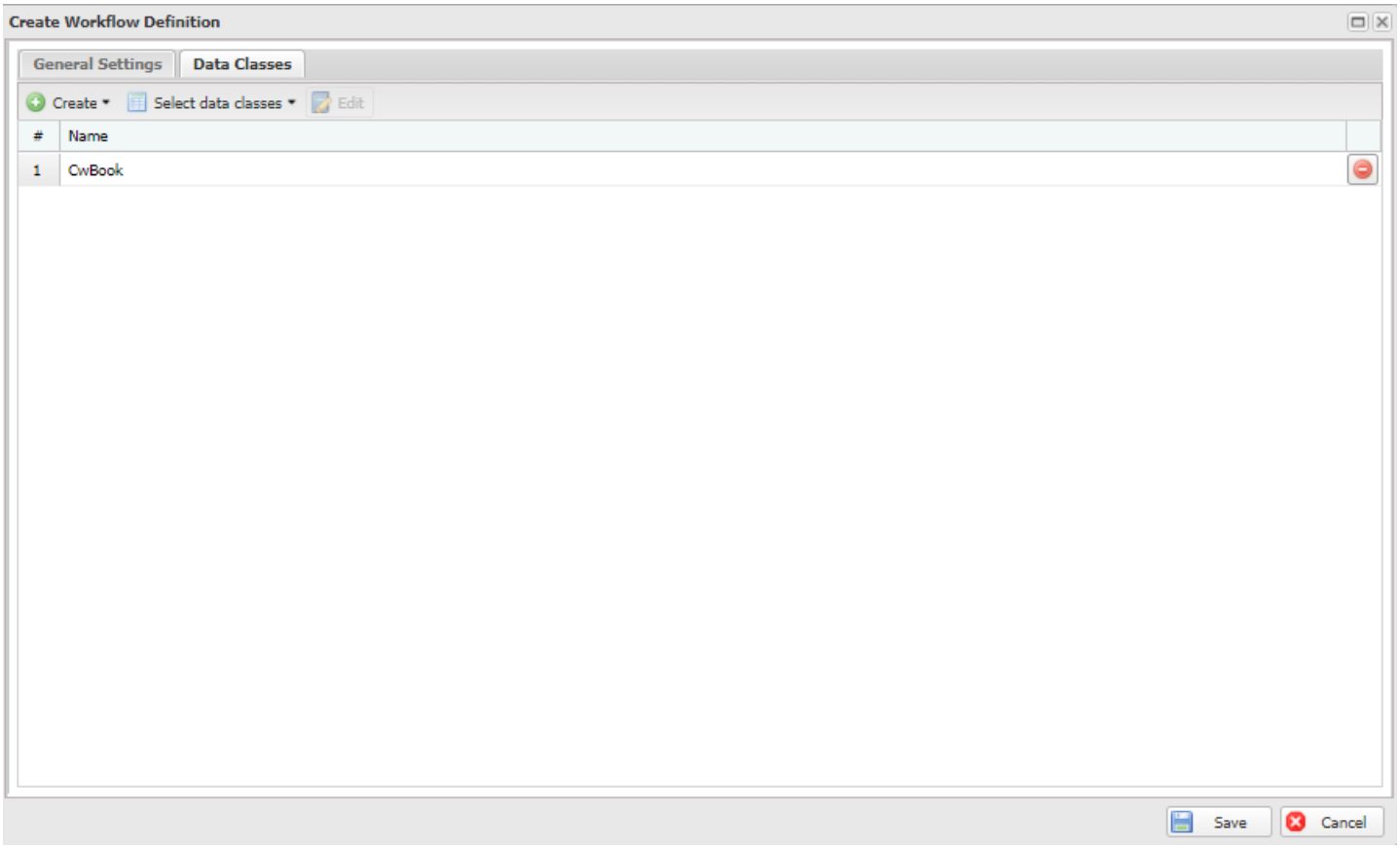
- Step 3. Config **General Setting** tab as below

FIELD NAME	ACTIONS
Name	Fill in "Delete Book"
Display Name	Fill in "Delete book"
Used For Creating New Cases	Make sure this field is uncheck ☐
Accept Input	☑
Require Input	☑
Window Size	Select **Small**



- Step 4. Switch to **Data classes** tab

Click **Select data classes** drop-down, and select "CwBook"

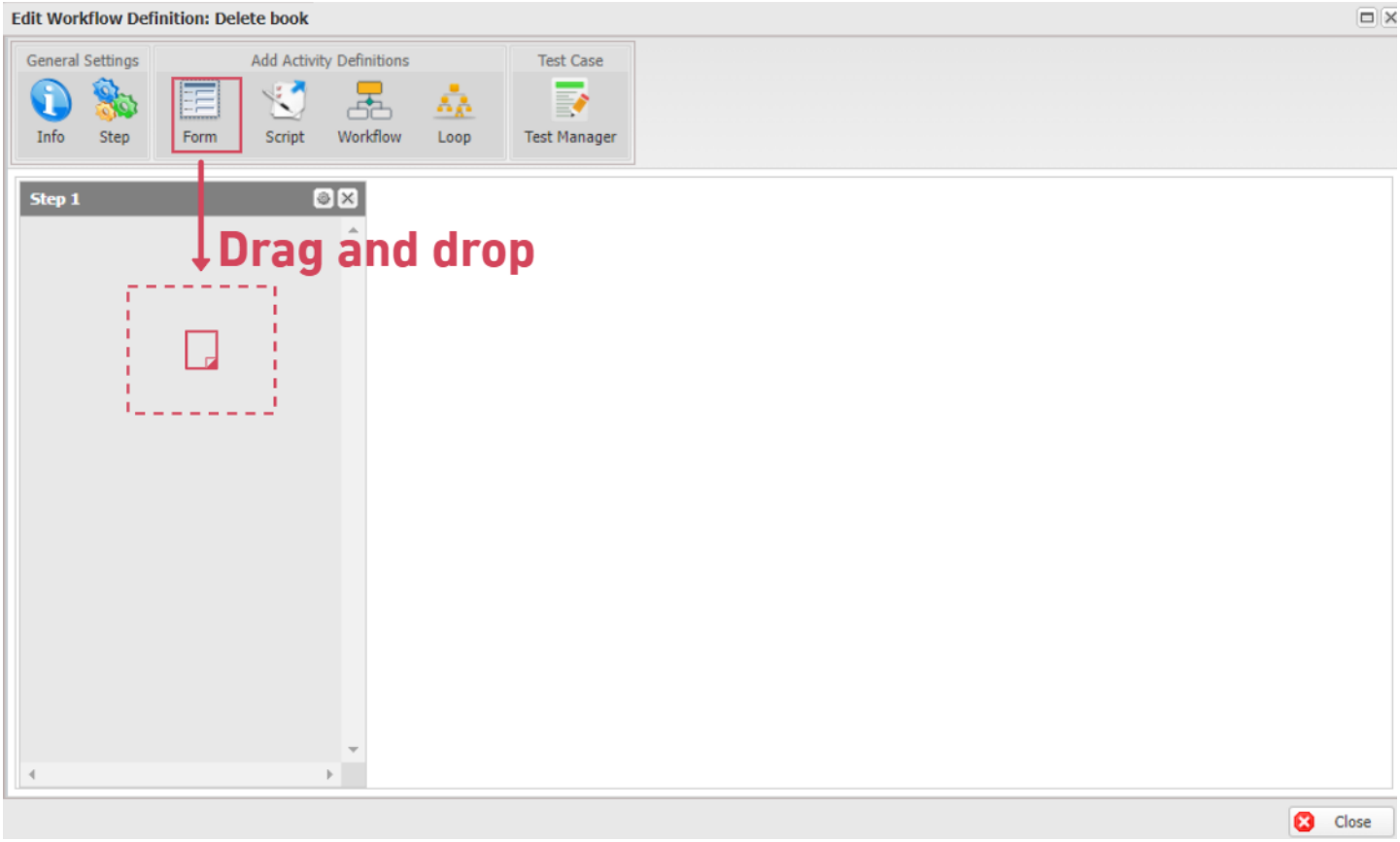


> Tip: The data classes you just select is the one you made in Create a data class step

Click **Save**

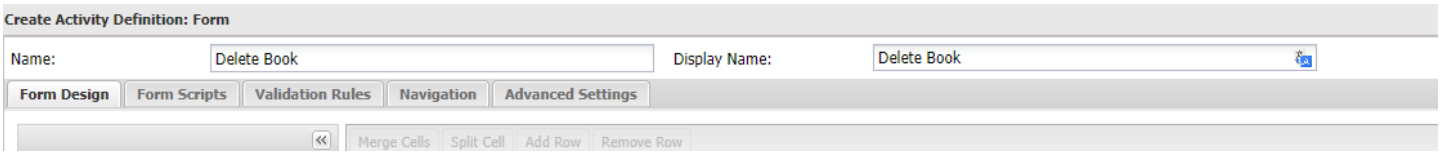
- Step 5. **Edit Workflow Definition: Delete book** dialog will display

Drag and drop **Form** activity definitions from **Add Activity Definitions** panel to **Step 1** container



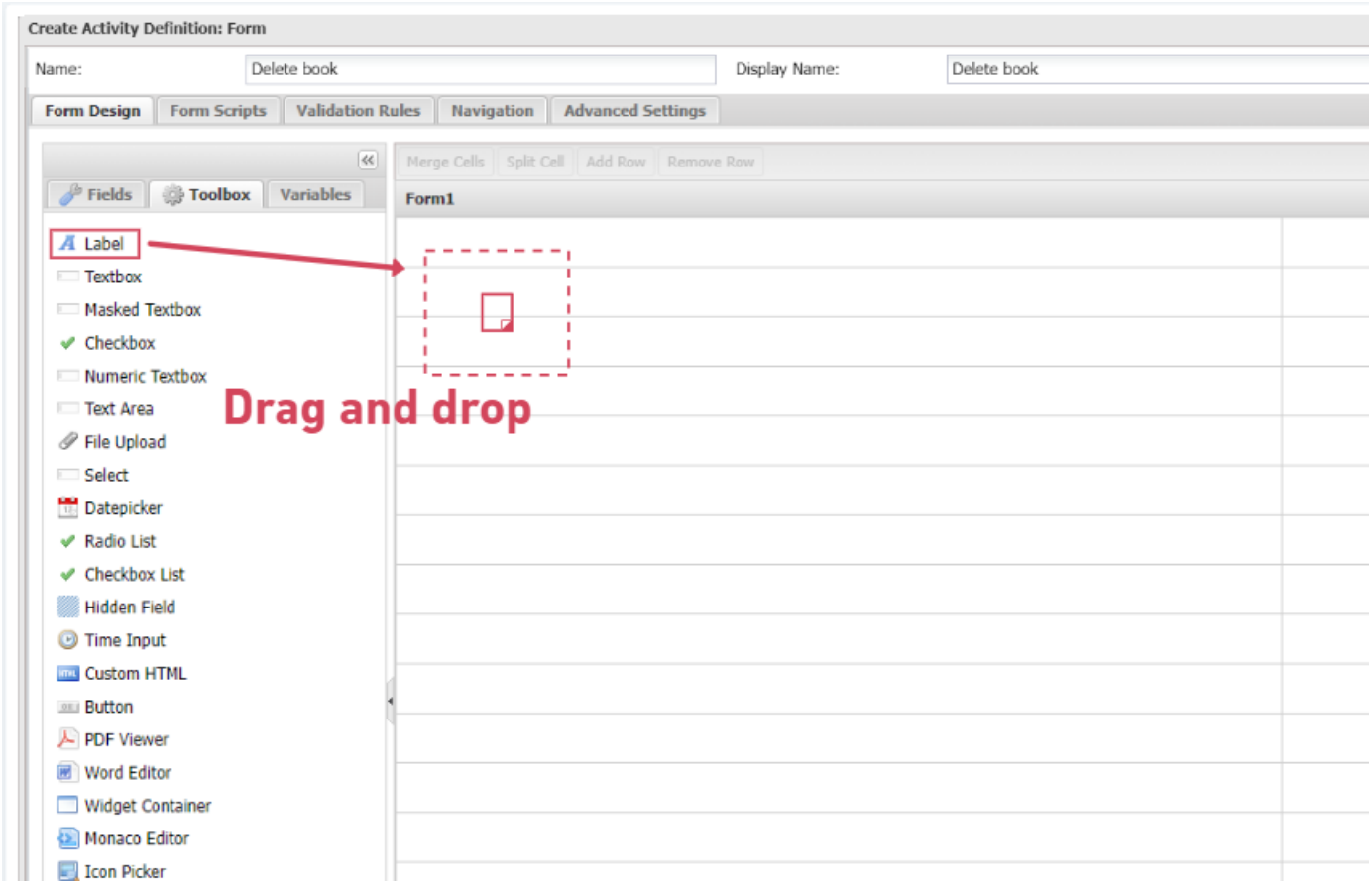
- Step 6: **Create Activity Definition: Form** will show up.

FIELD NAME	ACTIONS
Name	Fill in "Delete book"
Display Name	Fill in "Delete book"



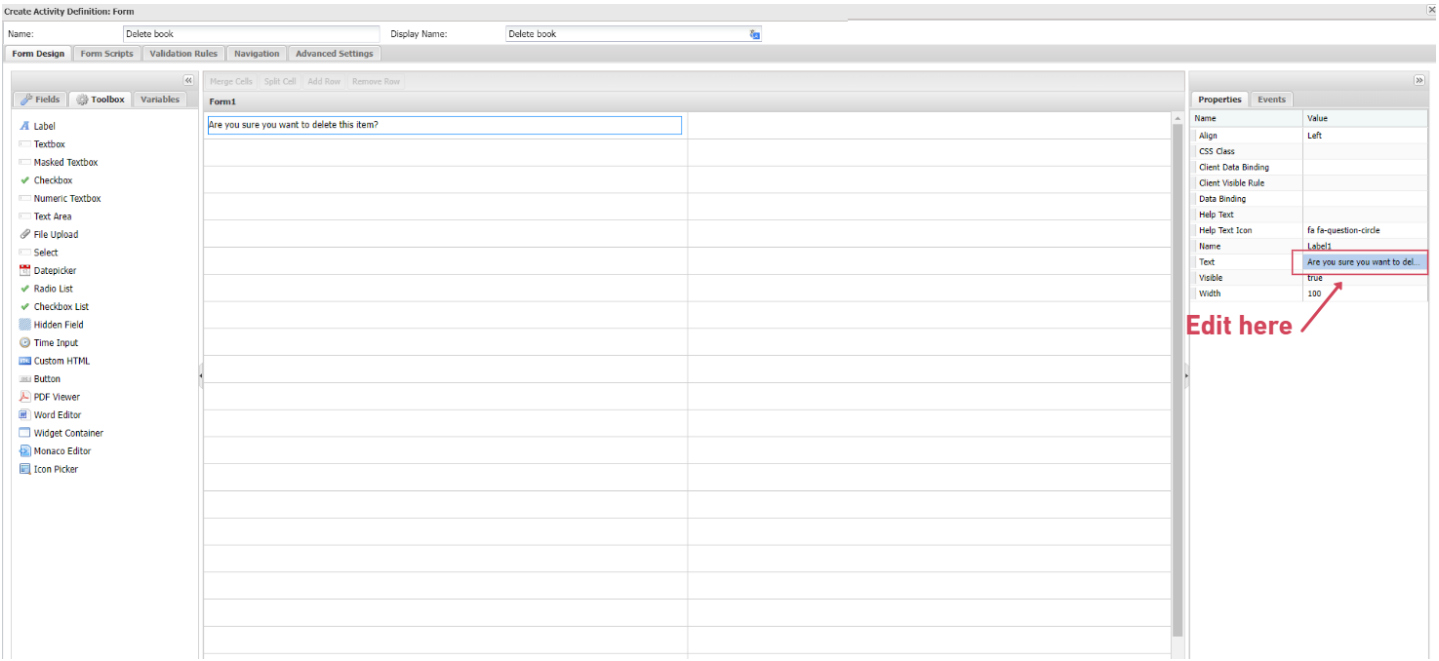
- Step 7. Navigate to the **Toolbox** tab

Drag and drop **Label** activity definitions from **Toolbox** panel to **Form1** canvas



- Step 8. In the **Properties** window, looking for **Text** property

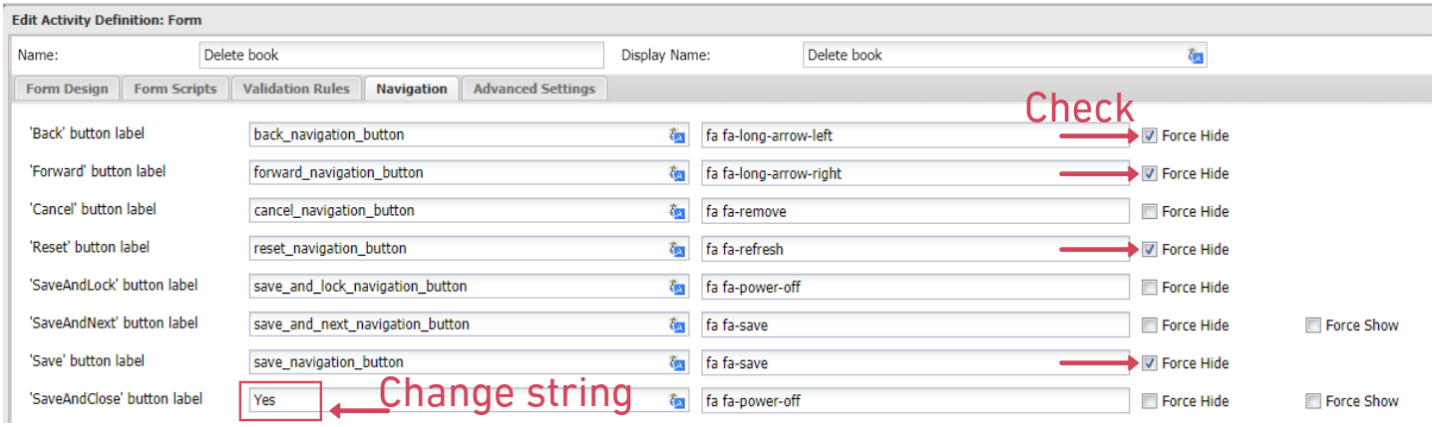
FIELD NAME	ACTIONS
Text	Fill in "Are you sure you want to delete this item?"



Click **Save & Close**

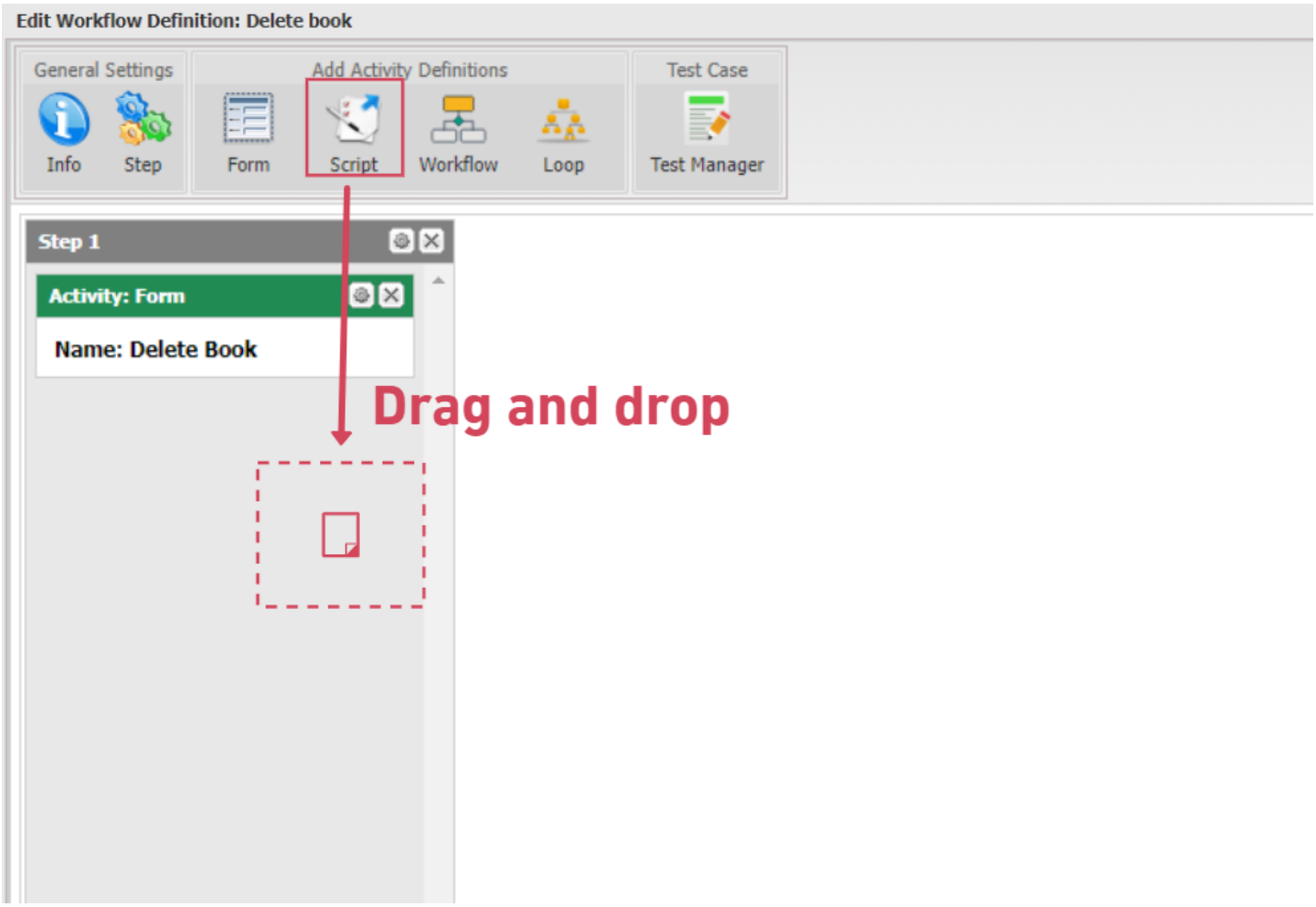
- Step 9. Switch to **Navigation** tab

FIELD NAME	ACTIONS
'Back' button label	☑
'Forward' button label	☑
'Reset' button label	☑
'Save' button label	☑
'SaveAndClose' button label	Change the string to "Yes"



Click **Save & Close**

- Step 10. Drag and drop **Scripted** activity definitions from **Add Activity Definitions** panel to the step container



- Step 11. The **Create Activity Defintion: Scripted** will open

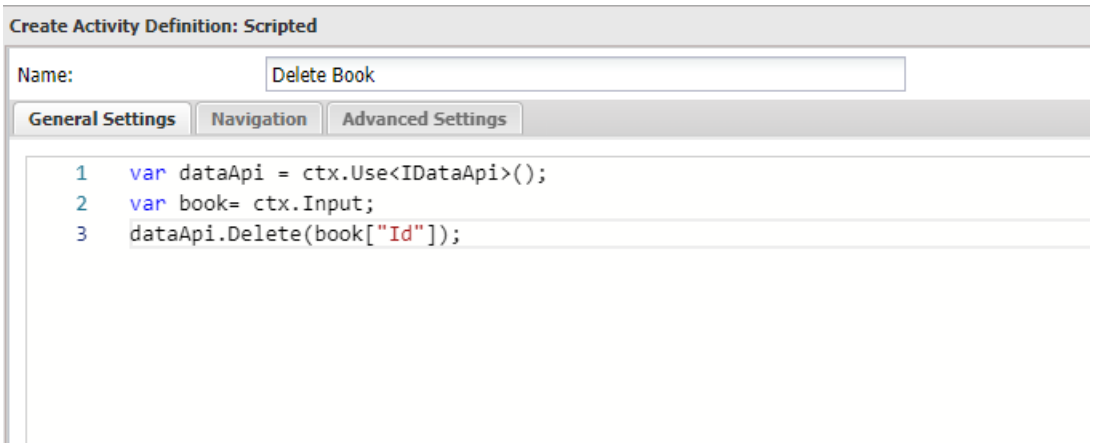
FIELD NAME	ACTIONS
Name	Fill in "Delete Book Script"

In **General Settings**, copy and paste the following code

```
var dataApi = ctx.Use<IDataApi>();

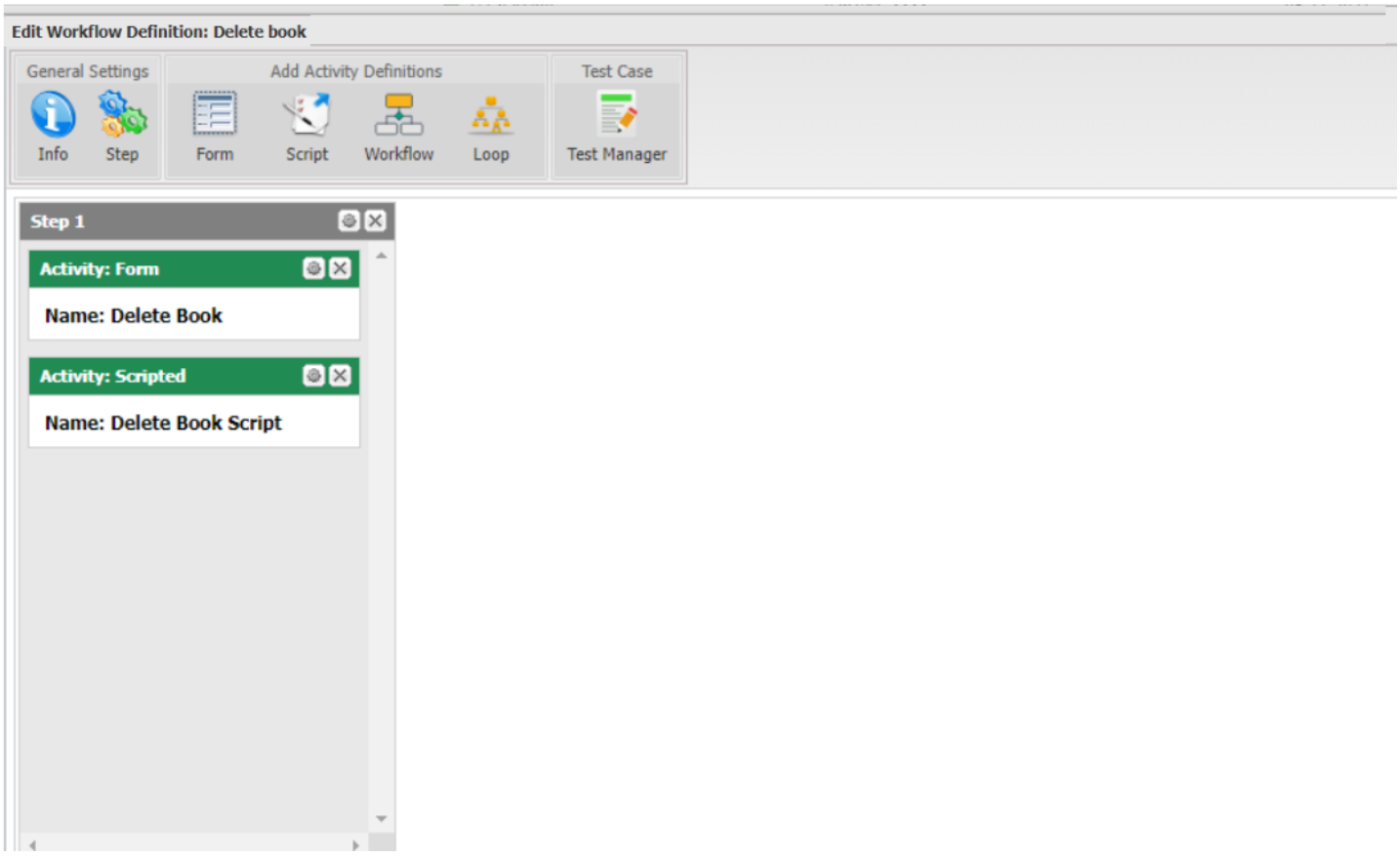
var book= ctx.Input;

dataApi.Delete(book["Id"]);
```



Click on **Save & Close**

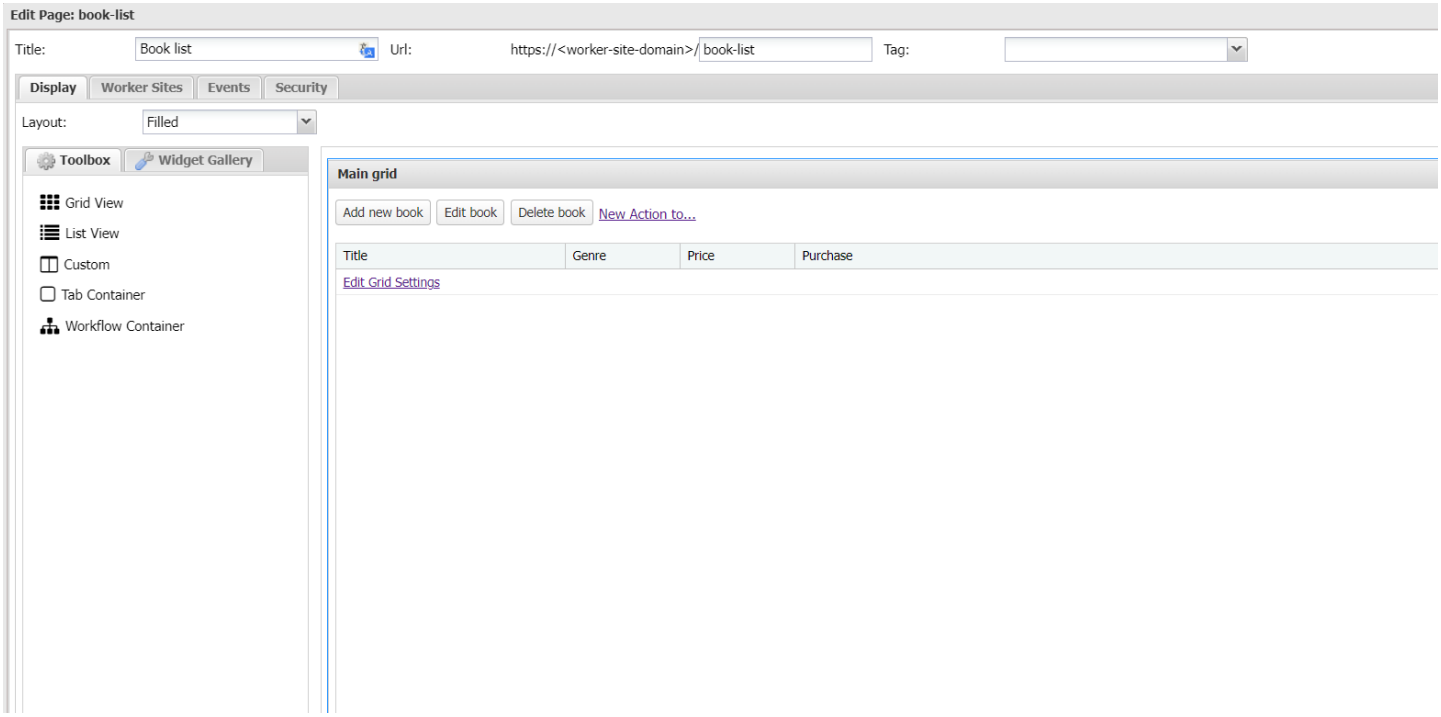
Now your **Edit Workflow Definition: Delete book** window should contains two activities in **Step 1** panel as show below



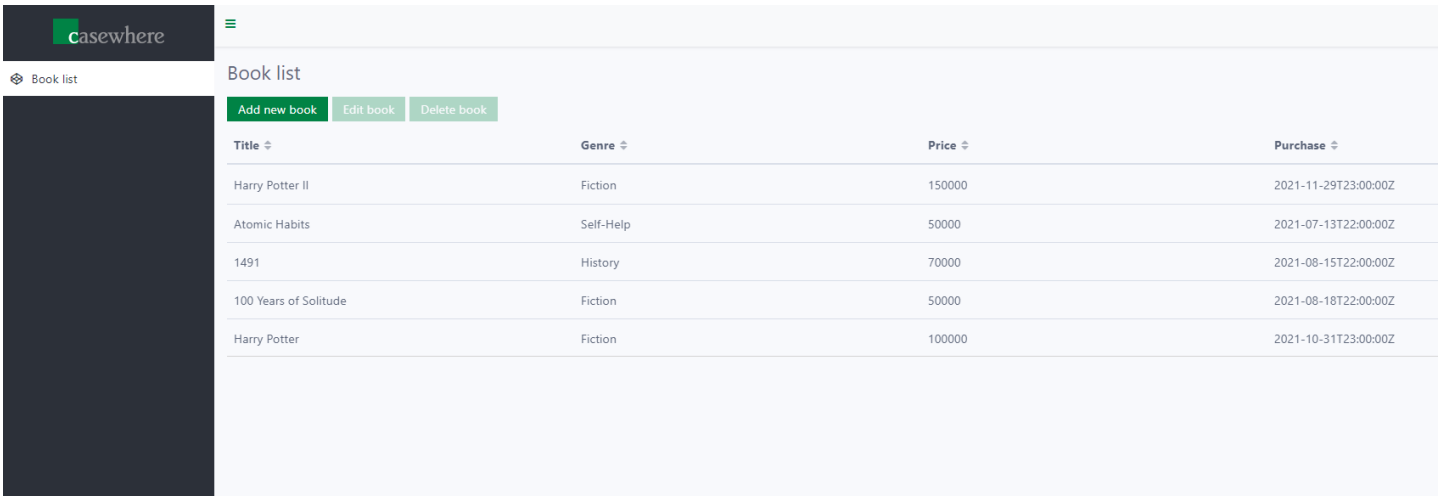
Good job! You are ready to use this workflow definition in your next step - Create "Delete Book" button.

- Step 12. Create a button to trigger the workflow

Follow this guide to create and test a button to make "Delete book" button



- Step 13. Reload your page to see the button on-live



Good job!

You are ready to move to the next step

Enable multi languages

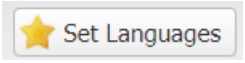
Introduction

Many Casewhere applications are created for users who come from different countries and/or speak different languages.

This tutorial will demonstrate how to enable multi languages for the three action buttons: add new book, edit book, and delete book that you already created in the previous steps.

Guide

Part 1. Create and manage multi language text

- Step 1. In Casewhere Admin site, navigate to **Text resources**
- Step 2. Click **Set Languages** button 

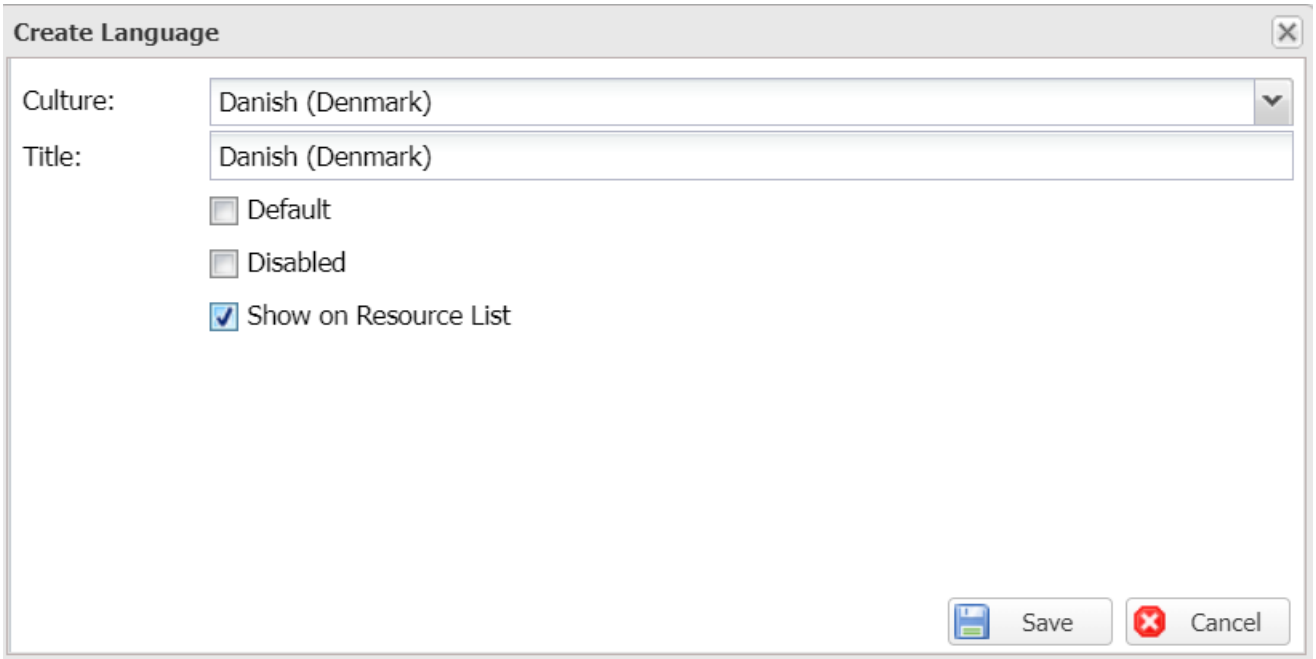
Languages dialog will show up

- Step 3. Click **Add** button 

Create languages dialog will show up

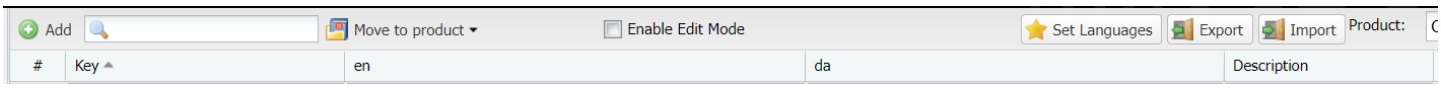
- Step 4. Choose the language that you want to create text with, here in this tutorial we will demonstrate with Danish

FIELD NAME	ACTIONS
Culture	Select "Danish (Denmark)"
Show on Resource List	☑



Click **Save**

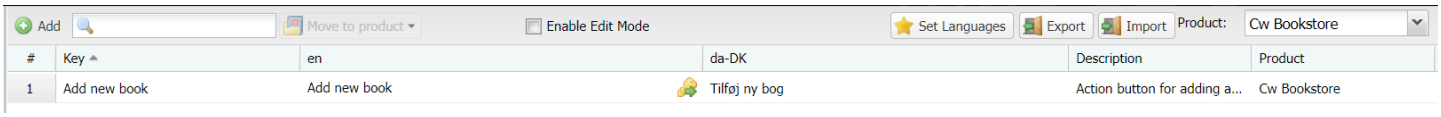
You should see the column's name "da" is now added to **Text resources** screen next to the column "en"



Those columns are where you will fill in the English and Danish version of your strings.

- Step 5. Click **Add** button  . Fill in the table as follows:

FIELD NAME	ACTIONS
Key	Fill in "Add new book"
en	Fill in "Add new book"
da	Fill in "Tilføj ny bog"
Description	Fill in "Action button for adding a new book"



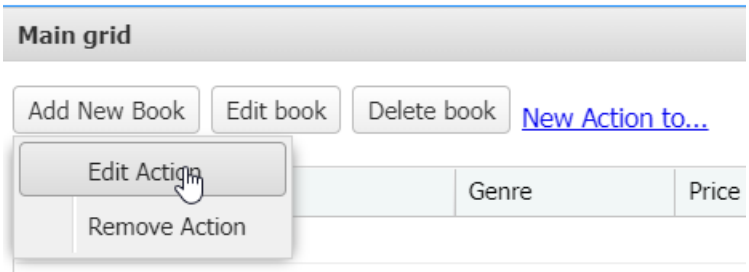
Click **Update**

Part 2. Enable multi languages

- Step 1. In Casewhere Admin site, navigate to **Pages**

Choose the pages name "book-list" that you already made in create a page step

- Step 2. Choose **Add New Book** action button and select **Edit Action**

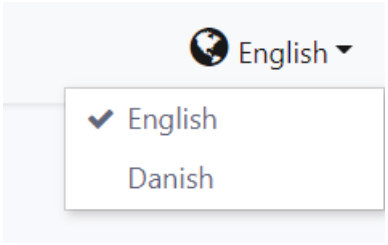




On the drop down menu, choose the key value that you added in Step 6.

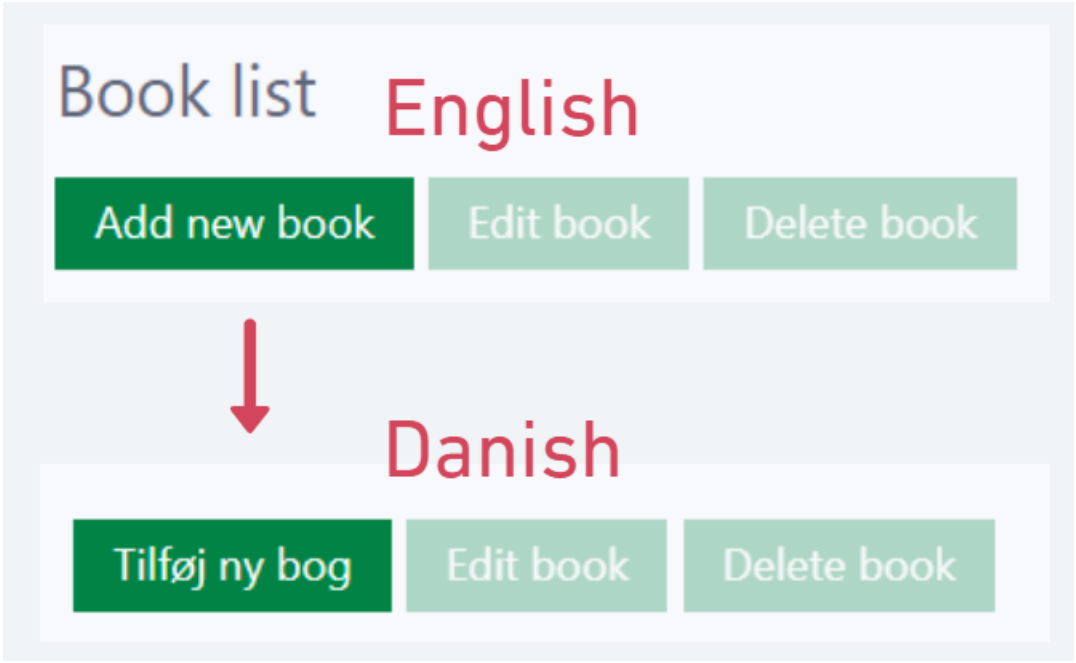


- Step 3. Reload your page



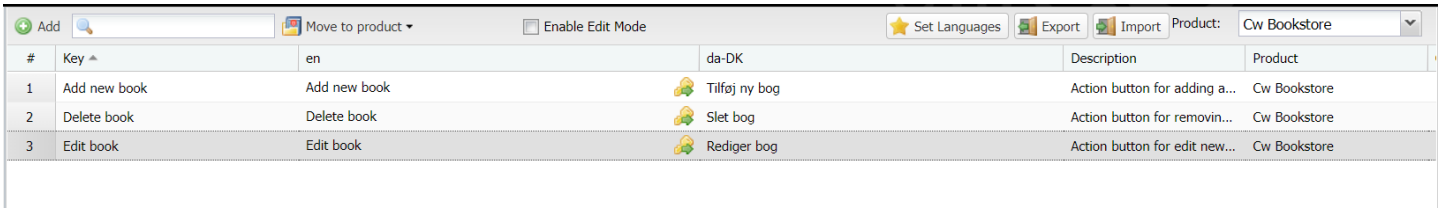
Now in the top right side, switch languages from English to Danish.

The string "Add New Book" will be changing as you switch between two languages.



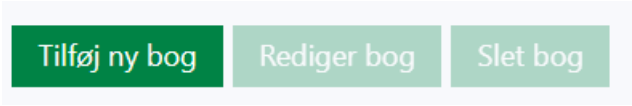
- Step 4. Way to go! Now let go back to **Text Resources** screen and add other translated strings follow part 1 of this tutorial

KEY	EN	DA-DK	DESCRIPTION
Add new book	Add new book	Tilføj ny bog	Action button for adding a new book
Edit book	Edit book	Rediger bog	Action button for editing book
Delete book	Delete book	Slet bog	Action button for removing book



- Step 5. Repeat step 1,2, and 3 of this part for the "Edit book" and "Delete book" buttons

Reload your site to see the result.



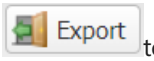
Tip: Text Resource icon  is the indicator that the field is translatable.

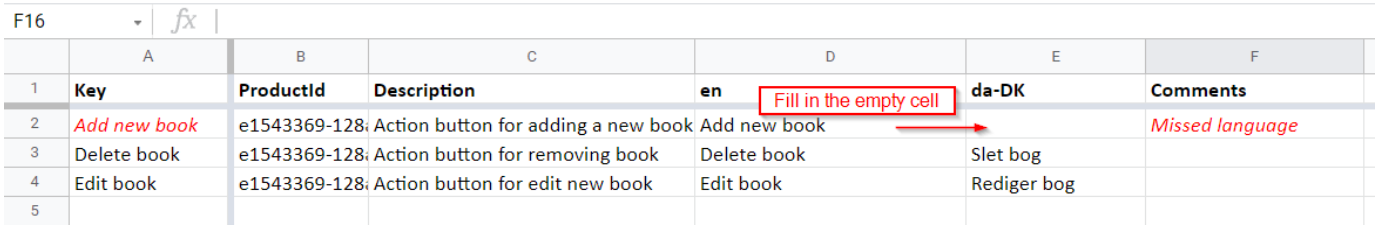
With Casewhere, all the labels that will appear on the user interface (UI) are able to be translated, it includes:

- Buttons
- Menu labels
- Column headers
- Error messages
- Form texts

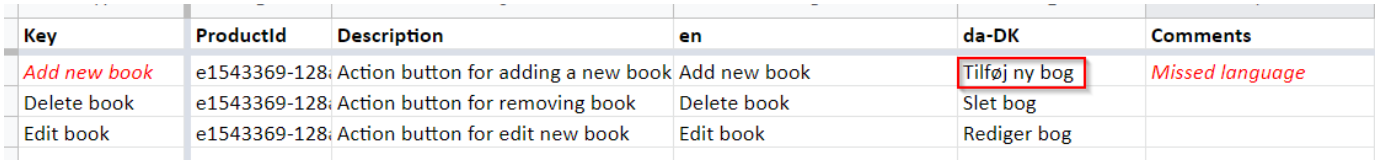
Part 3. Export and import text resources

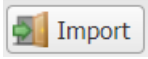
You can export your whole text resources list into an .xlsx file, you can edit that .xlsx file and then import it back into Casewhere instance's database.

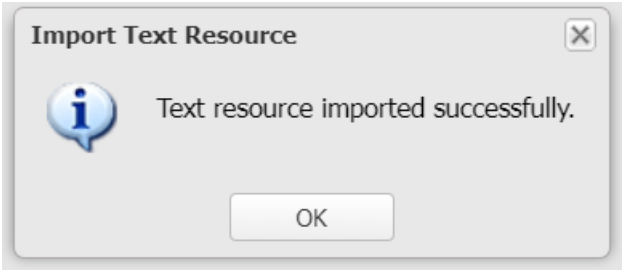
- Step 1. In Casewhere Admin site, navigate to **Text resources**
- Step 2. Click **Export** button  to download the .xlsx file
- Step 3. Open the .xlsx file. Any label that is missing some translations will be marked in red



Fill in all the missing place in your file and save.



- Step 4. Import your edited .xlsx file back into Casewhere by click **Import** button 

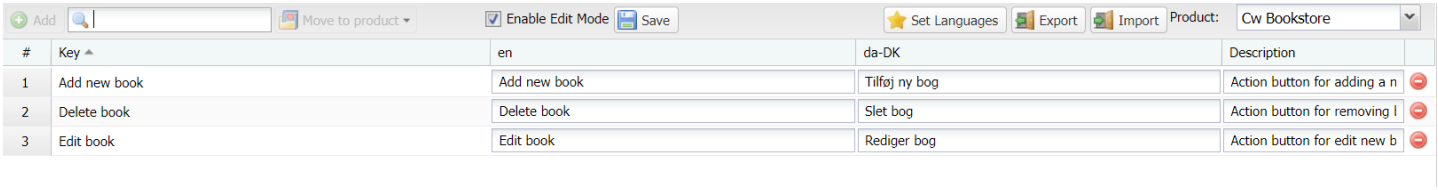


Part 4. Text resources - Edit mode

You can enable edit mode to edit faster right in your Casewhere instance.

- Step 1. In Casewhere Admin site, navigate to **Text resources**
- Step 2. Select **Enable Edit Mode** check box 

Editing view will be enable, allow you to edit all the translation strings in bunch.



Click **Save**  once you are done editing.

You are ready to move to the next step.

Enable grid widget features

Searching and filtering are basic functionalities that most applications provide.

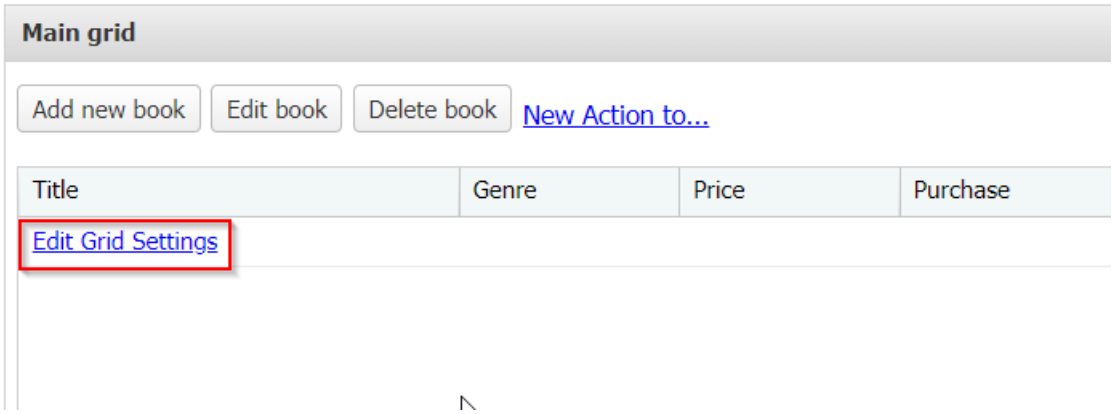
In this tutorial you will learn how to quickly config those functionalities using Casewhere.

Guide

- Step 1. In Casewhere Admin site, navigate to **Pages**

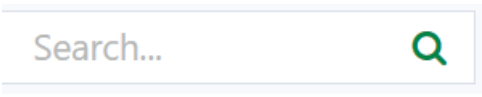
Choose the pages name "book-list" that you already made in create a page step

- Step 2. In the **Main grid** widget, click **Edit Grid Setting**

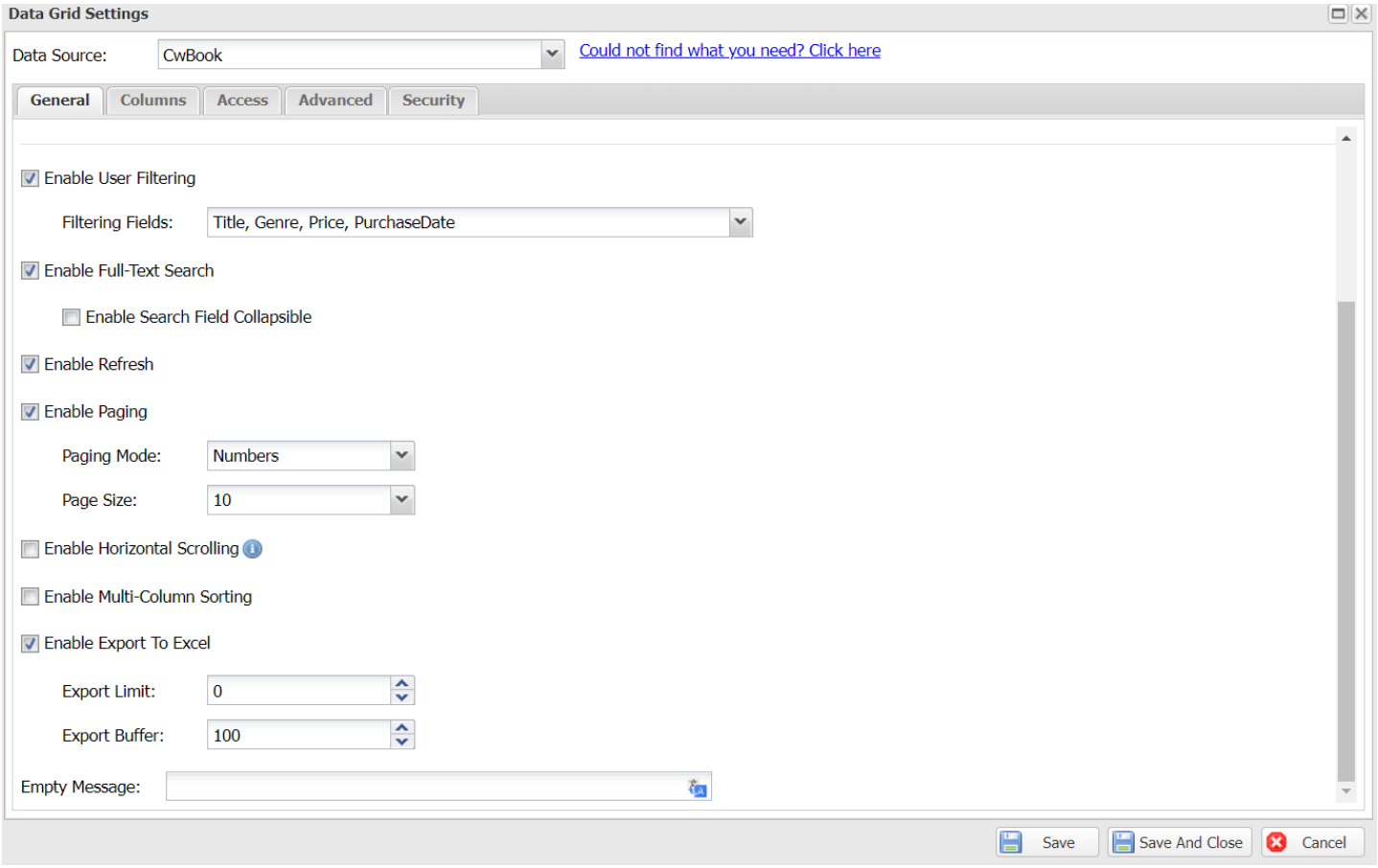


- Step 3. **Data Grid Settings** dialog will show up allowing you to config the **Main grid**. Simply check box ☑ to enable the field.
  - **Enable User Filtering**: allow end-users to create custom filters in the worker site
  - **Enable Full-Text Search**: allow end-users to search
  - **Enable Refresh**: provide end-users a shortcut to quickly refresh data
  - **Enable Paging**: allow end-users to enable loading and displaying data page by page. This should be enabled when you have a large data collection
  - **Enable Horizontal Scrolling**: allow end-users to scroll left and right to reveal content from the sides of the window
  - **Enable Multi-Column Sorting**: allow end-users to sort on multiple columns
  - **Enable Export To Excel**: allow end-users to export the displayed data to an .xlsx file
  - **Empty message**: allow end-users to define the content of empty message when downloading the file

For your application, let enable the functions below:

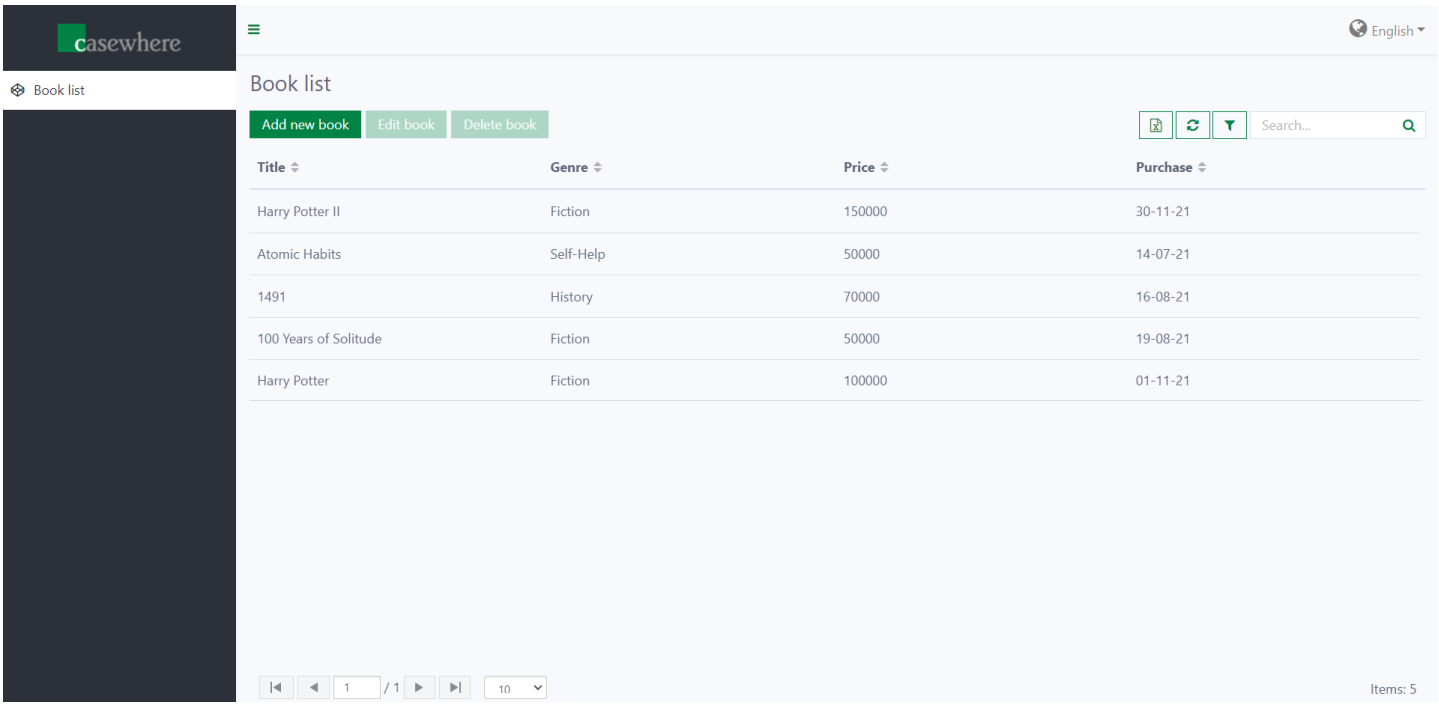
FIELD NAME	ACTIONS	HOW IT WILL LOOK IN UI
Enable User Filtering	☑	
Enable Full-Text Search	☑	
Enable Refresh	☑	

FIELD NAME	ACTIONS	HOW IT WILL LOOK IN UI
Enable Paging	☑	
Enable Export To Excel	☑	



Test

Reload your site, it should look like a fully-functional web page as below



Try it! Let's interact with all the cool functions that you created.

You can follow the following checklist:

- Add more 10 more books
- Edit the name/genre/price/purchase date of books
- Delete 1 book
- Refresh your site
- Switch between pages
- Filter for all "History" books
- Search for "Harry Potter" book
- Export the list to .xlxs

Way to go, now you are ready to move to the next step!

Deploy a Casewhere product

Introduction

In this tutorial you will learn how to deploy a Casewhere product to different environments.

> Tip: A typical enterprise solution usually has at least three different environments:
>
> - Development
> - Testing
> - Production

This tutorial has two parts

1. Link all resources to a product
2. Deploy the product to a new environment

Guide

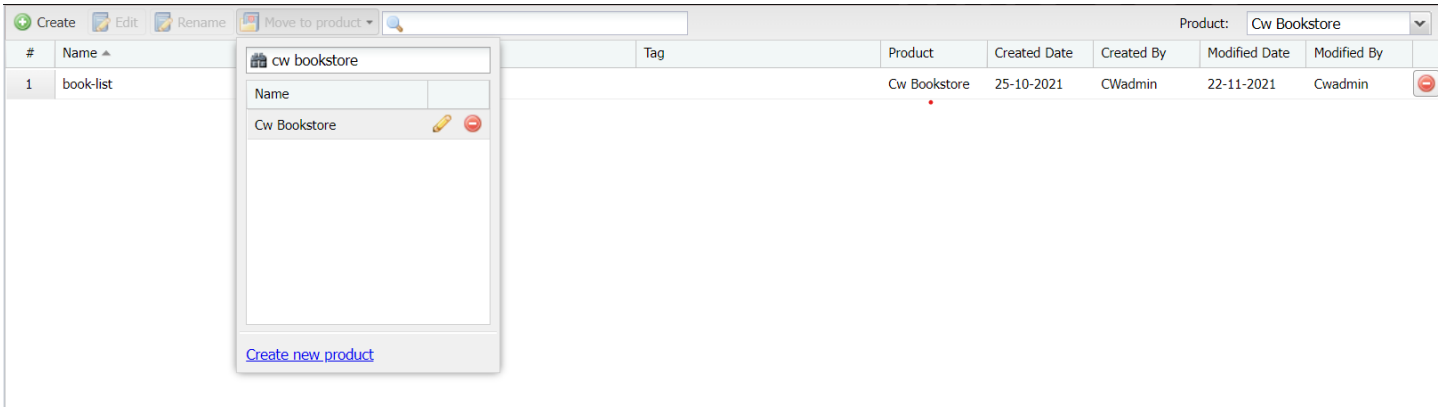
Part 1. Link all resources to a product

- Step 1. In Casewhere Admin site, navigate to **Pages**

Select the page's name "book-list"

- Step 2. Click **Move to product**

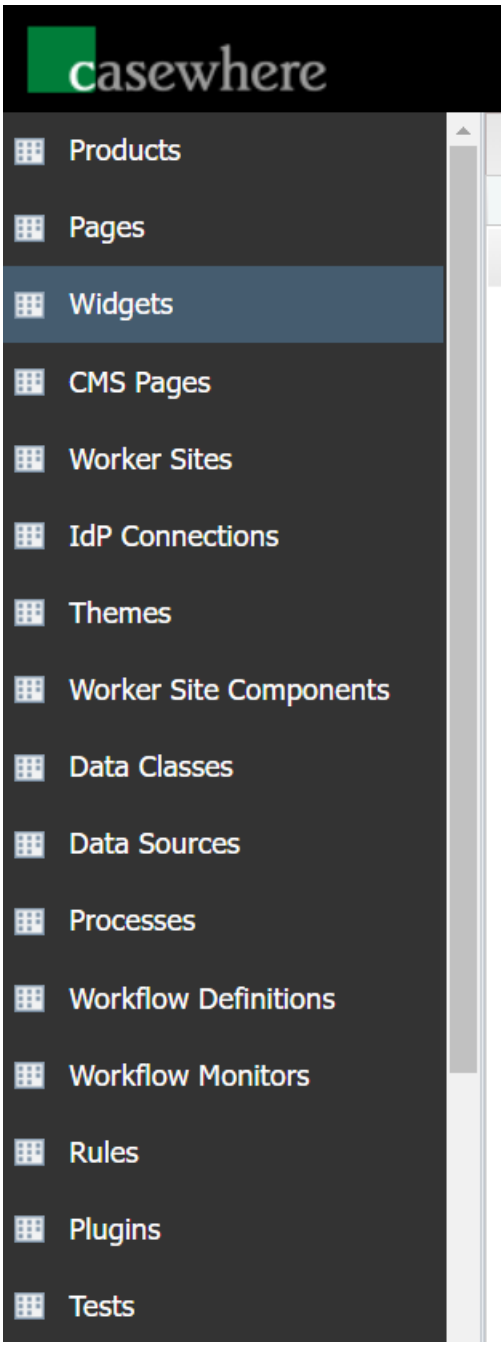
Select the product's name "Cw bookstore" that you already made in create a product step



- Step 3. Repeat step 1 and step 2 for **Widgets, Workflow definitions, Data sources, Data classes**
- Step 4. Choose the product name in the top-right drop down



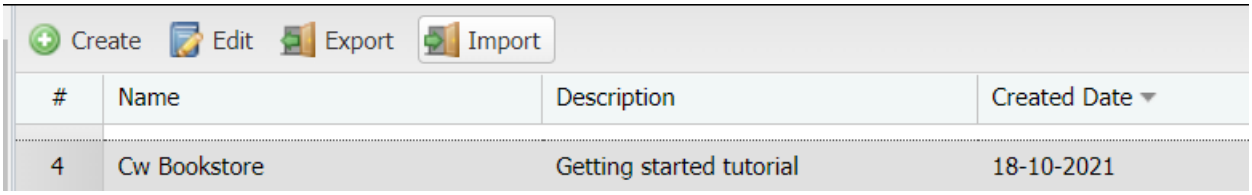
Navigate between **Pages, Widgets, Workflow definitions, Data sources, Data classes** and double-check if all resources is linked to the correct product



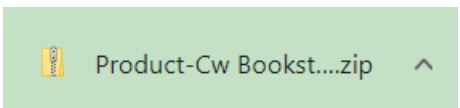
Part 2. Deploy the product to a new environment

- Step 1. In Casewhere Admin site, navigate to **Products**

Select the product's name "Cw bookstore" and click **Export** button

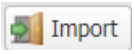


The package file .zip will automatically download

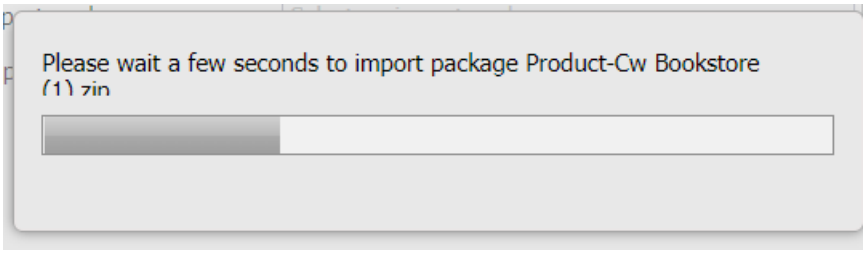


- Step 2. Open your new Casewhere environment. This tutorial will use the https://academy.casewhere.com to demonstrate

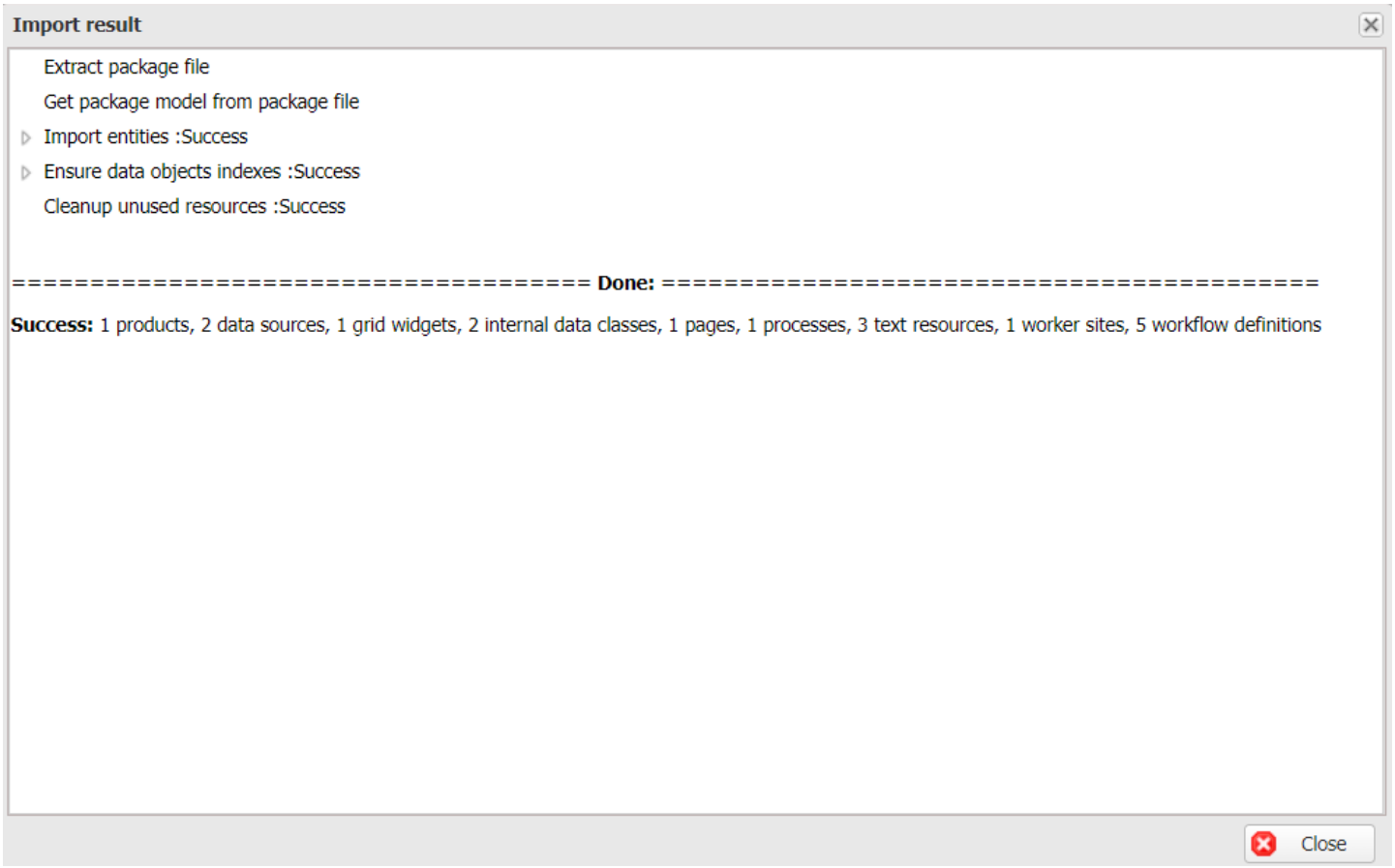
- Step 3. In the new environment's Admin site, navigate to **Products**

Click Import button  and browse to the file you downloaded in step 1

Wait for the package to be fully imported.



Import result dialog appear signifies that the import is complete.



Click **Close**.

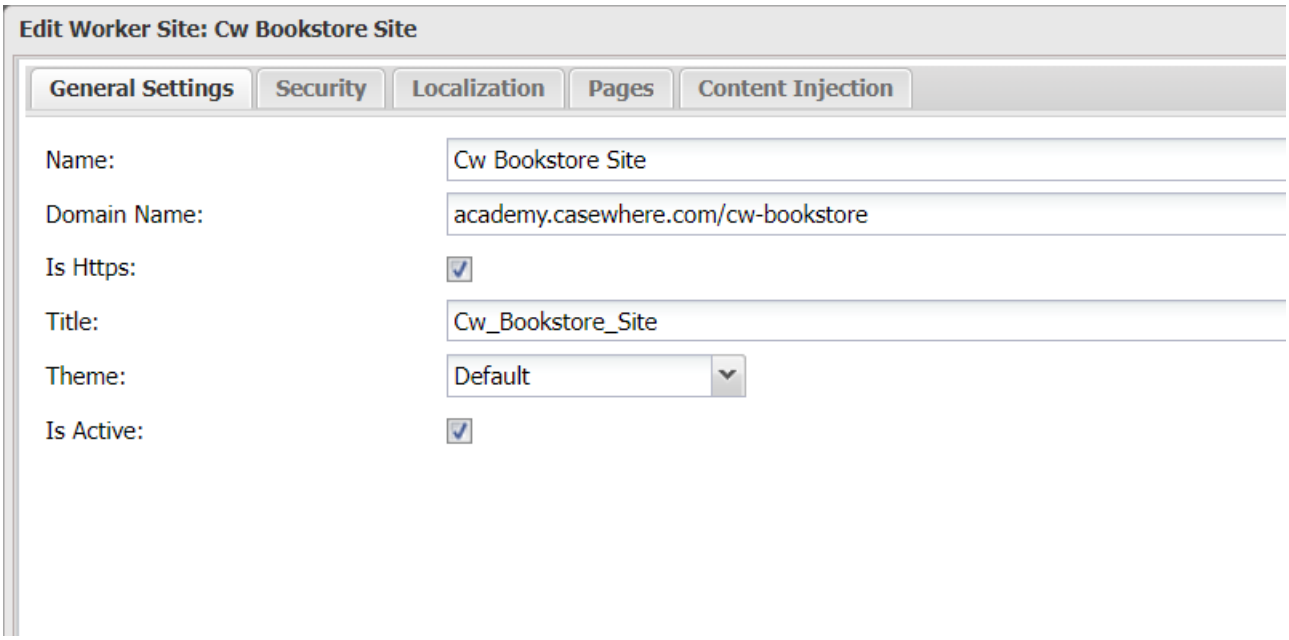
- Step 4. The package is almost ready to run. You will only have to config the domain name, which is environment-specific.

Note: Make sure to use the correct domain name. In this tutorial we use the environment https://academy.casewhere.com and the domain name is "worker-dev.casewhere.com".

Note: If you are using Casewhere version 2.7 or later **Domain name** will be called as **Uri**

Navigate to **Worker Sites**, select the site name and click **Edit** button

FIELD NAME	ACTIONS
Domain name	Change to "academy.casewhere.com/cw-bookstore"



Click **Save**.

Deploy the product package to new environment success!

Test

Go to your new site at https://academy.casewhere.com/cw-bookstore#!/book-list.

It will have all the functions that you set for your product.

Using several environments guarantees that your application could be thoroughly tested before released to the users.

You now know how to export/import a package in a variety of Casewhere environments. You could collaborate with other team members and operate across environments.

How cool is that!

You are ready to move to the next step.

Try it! Wrap up

Congratulations

Congratulations! You have finished the tutorial series Casewhere Try it!

Overview

Create a product

Create a data class

Create a worker site

Create a page

Create a process

Create a grid widget

Create add-object workflow

Create a widget button

Create edit-object workflow

Create delete-object workflow

Enable multi languages

Enable grid widget features

Deploy a Casewhere product

We expect you've grasped the following knowledge:

- Get to know and become familiar with Casewhere Administration as a application builder tool
- Way to design workflows for doing CRUD operations on an object
- Brief understanding of the Casewhere system architect

That's just the beginning. You're now ready to learn more advanced Casewhere features.

You will eventually have the confidence and knowledge to design or modify your own Casewhere applications.

So, keep up the good work, and happy learning!

Deploy Casewhere on Azure

Prerequisites

Minimum requirements for Dev and Test

- App Service: S2 - 2 cores, Ram 3.5 GB
- Azure VM for databases: 2 cores, Ram 4 GB, Standard SSD
- Azure Storage: Standard
- Application Insights: Standard

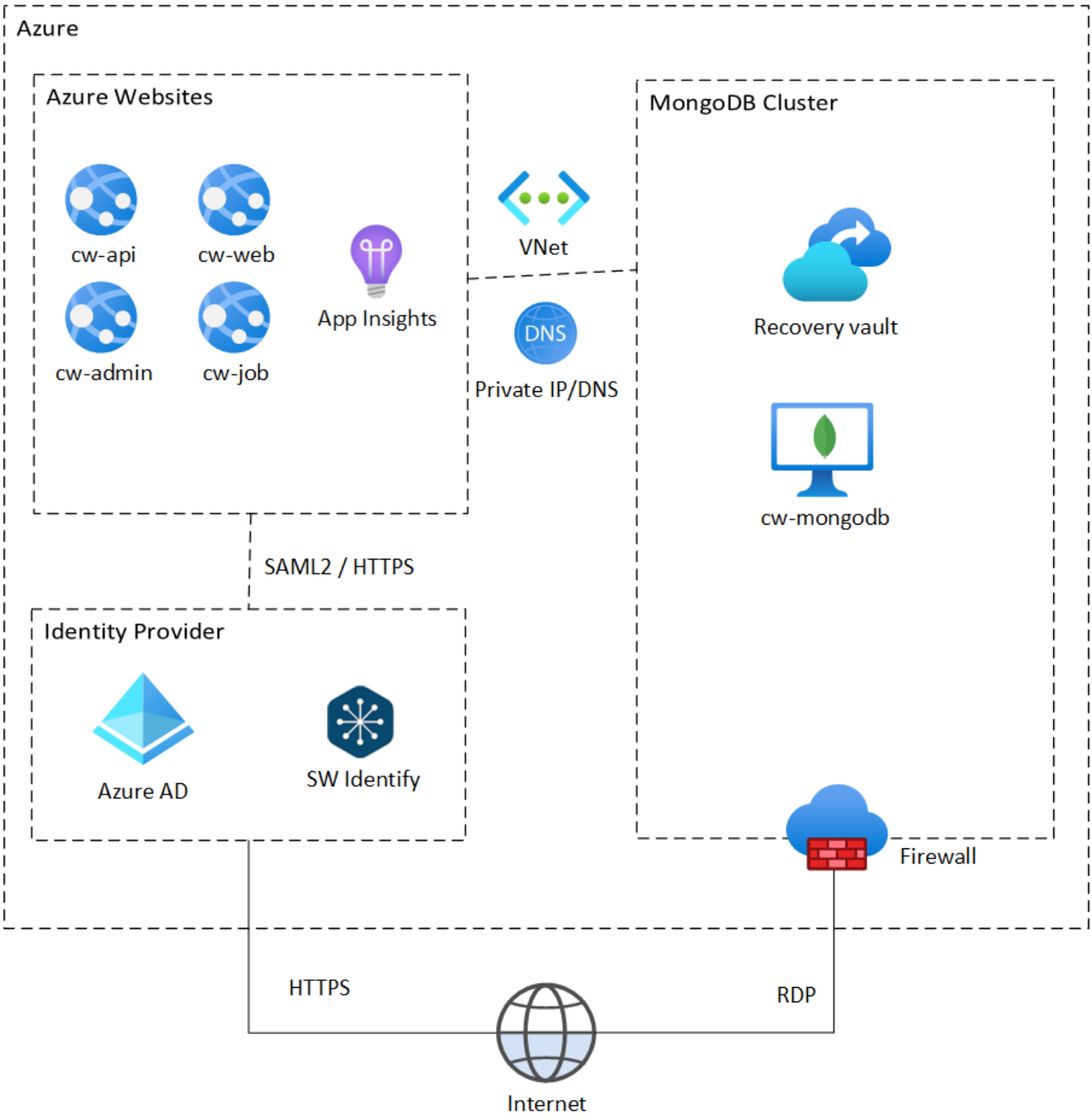
Minimum requirements for Production

- App Service: P1v3 - 2 cores, Ram 8 GB
- Azure VM for databases: 2 cores, Ram 8 GB, Premium SSD
- Recovery Services Vault
- Azure Storage: Standard
- Application Insights: Standard

Casewhere

- Version 2.5 or later
- Prepare the service certificates as described here
- Prepare the SSL certificates for your websites

Deployment diagram



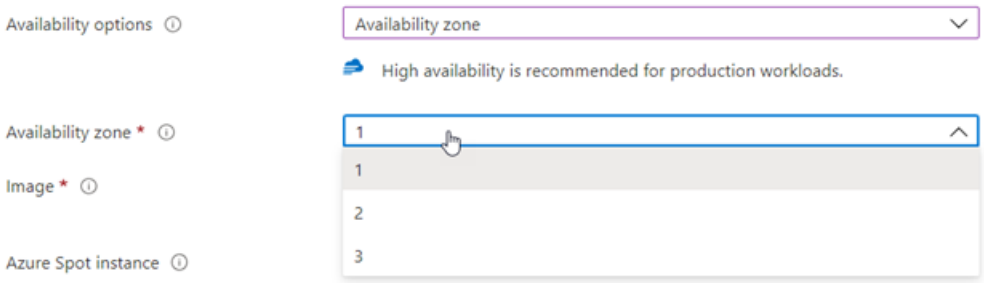
Deploy MongoDB

Deploy Azure VM

You can quickly learn to create a VM from Azure Portal here. Below are the key settings for deploying MongoDB for Casewhere. Make sure the VM fulfills the requirements of your project as well as the minimum specification required by Casewhere.

Availability

If you need to deploy a cluster of VMs, it is recommended to use the Availability Zone for production workload to achieve the highest availability, 99.99%. To ensure high availability, we must spread the VMs across the zones equally.



Networking

- Virtual Network: Specify an appropriate name.
- Public inbound ports: **3389**
- Make sure you close port **3389** in the **Network Security Group** when you finish the deployment.

Monitoring

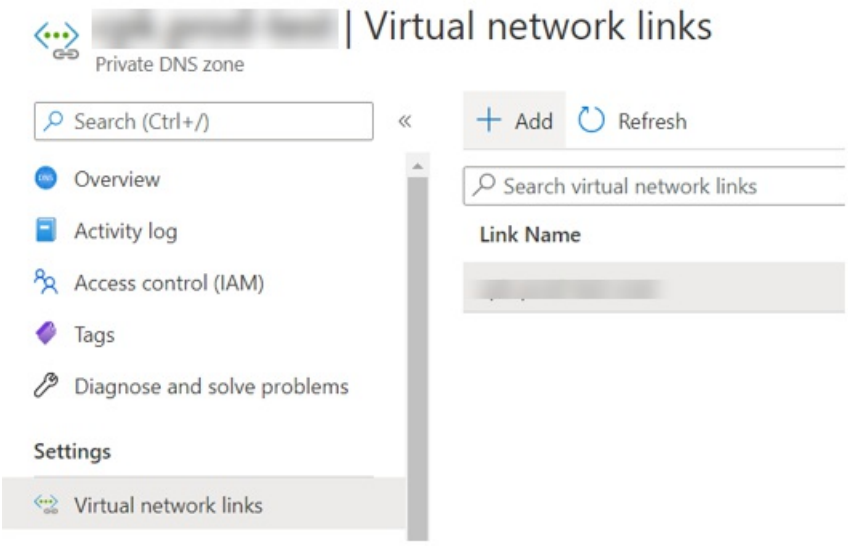
- Boot diagnostics: **Enabled**
- Enable OS guest diagnostics: **Enabled**

Set up Private DNS Zone

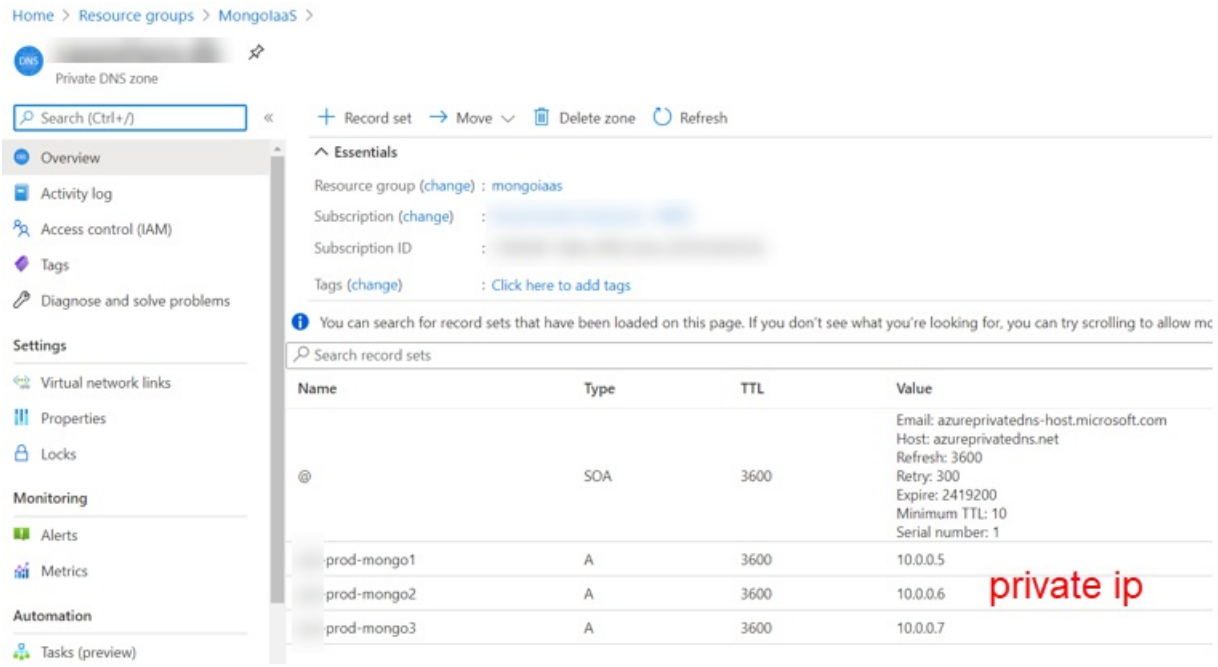
Create a Private DNS Zone.

Navigate to the **Private DNS Zone** resource and select the setting **Virtual network links**. On the upper menu, select **Add**.

Select the Virtual Network used by the MongoDB cluster created in the previous step:



Create A records for all VMs in your MongoDB cluster.



Deploy MongoDB replica set

MongoDB recommends using a replica set for production workloads. You can follow the guidelines here to install MongoDB and configure the replica set for Casewhere.

For development and testing environments, a single MongoDB instance can be used to host all Casewhere databases, providing a straightforward and easy setup. However, for high-performance systems where strict Service Level Agreements (SLAs) are required, it is recommended that the database workloads be separated into multiple replica sets. This approach helps optimize performance, scalability, and reliability.

- Read workload
  - CasewhereData
  - CasewhereFileStorage
  - CasewhereHangfire
  - CasewhereKeys
- Write workload
  - CasewhereEvents
  - CasewhereLog

We also recommend using private domain names to reference the VMs instead of IP addresses. For example, use "cw-prod-mongo1.casewhere.db".

Deploy Azure App Services

Create App Service Plan

Follow this guideline to create a **Windows App Service** Plan. Choose the **Pricing Tier** relevant for your deployment purpose.

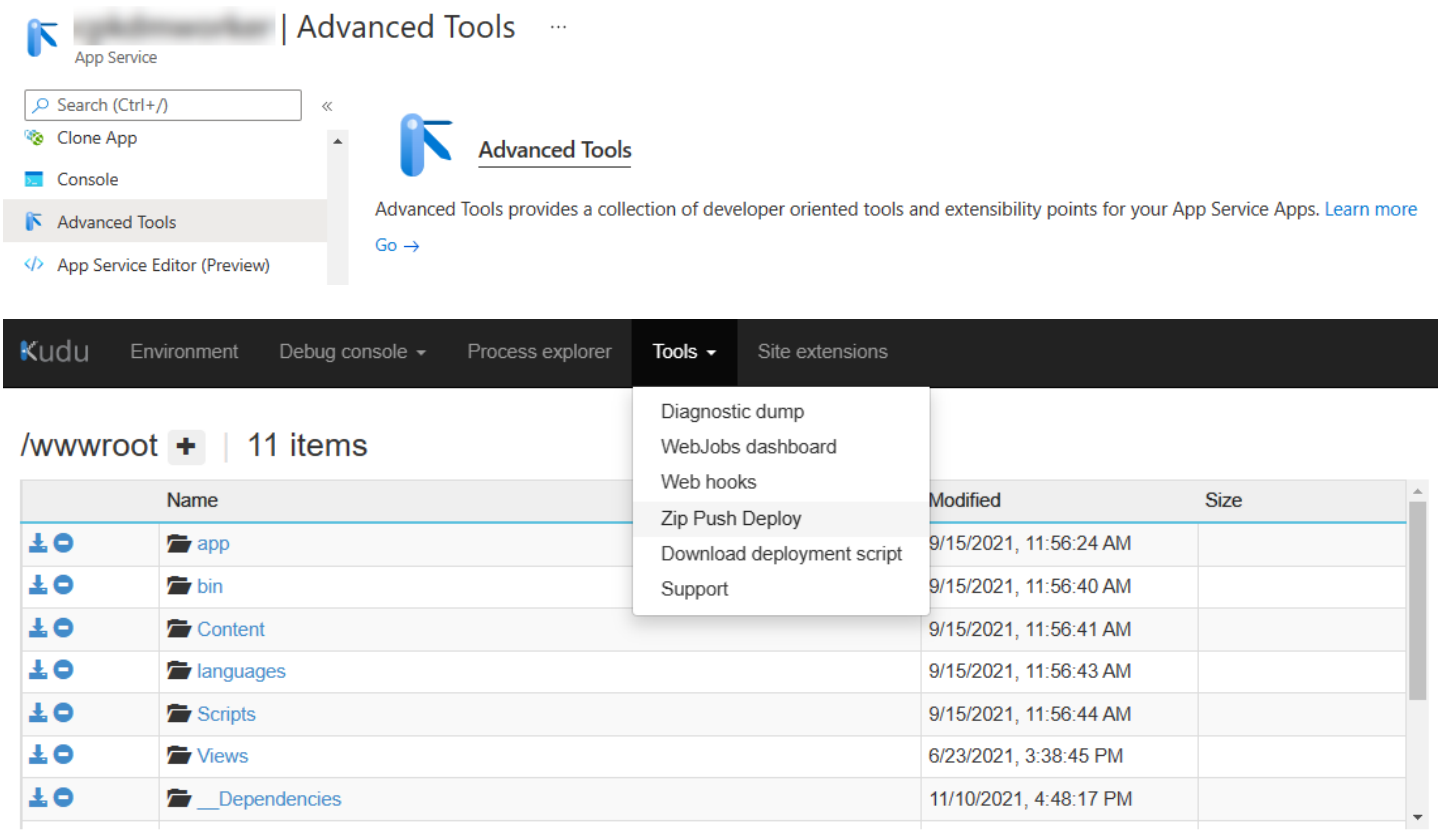
A Casewhere deployment will need the following sites created. In this document, we will use "cw-prod" as a resource-name prefix but make sure you use an approriate one for your project.

AZURE APP SERVICE	CASEWHERE SOURCE
cw-prod-web	Casewhere.Worker.Web
cw-prod-api	Casewhere.Worker.Api

AZURE APP SERVICE	CASEWHERE SOURCE
cw-prod-admin	Casewhere.Configuration.Web
cw-prod-job	Casewhere.Hangfire

Deploy files

There are many ways to deploy files to Azure App Service. In this document, to keep things simple, we recommend using Kudu Zip Push Deploy. Make sure you use the right package for the right app.



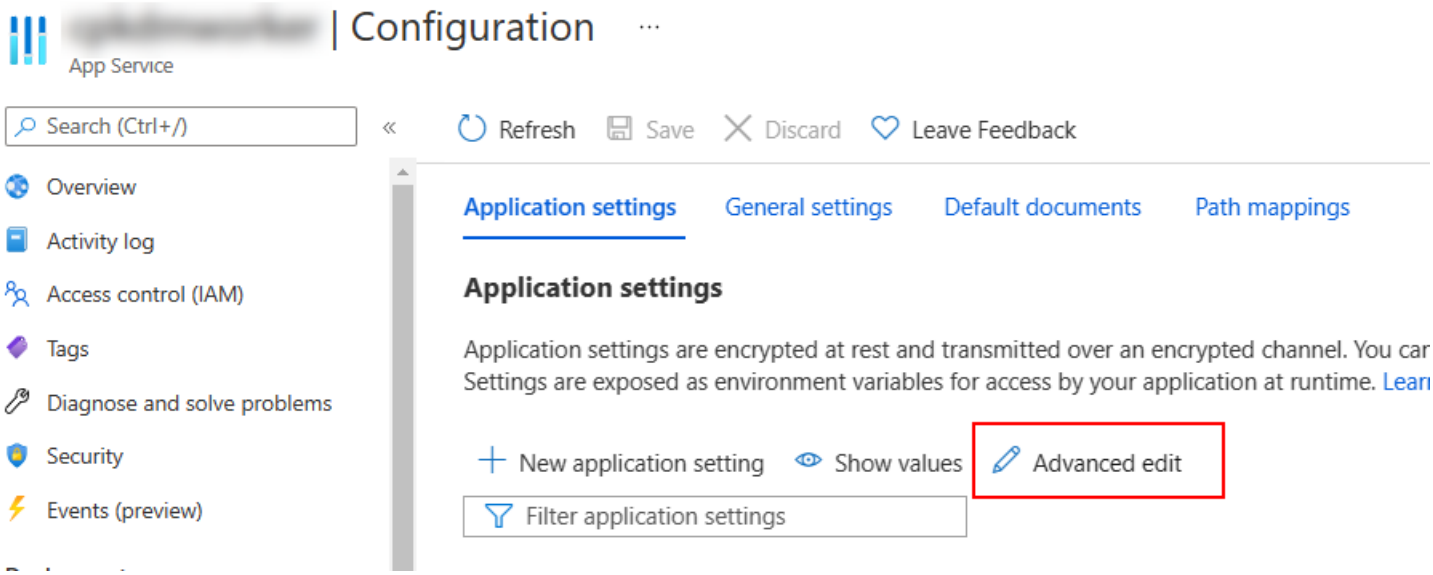
Common settings

The following settings should be done for all four Casewhere App Services.

Configuration

Application settings

This is where you configure the **application settings** and the **connection strings** for your app service. We can use the **Advanced edit** feature to quickly input the settings.



Append the keys in the below JSON files to the corresponding configuration sections. Make sure you review and edit the values before applying them, especially the ones related to database URIs, storage accounts, certificates, etc.

CASEWHERE SOURCE	WEBSITE	APPLICATION SETTINGS	CONNECTION STRINGS
Worker.Web	cw-prod-web	cw-prod-web-settings.json	*None*
Worker.Api	cw-prod-api	cw-prod-api-settings.json	cw-prod-api-connection-strings.json
Configuration.Web	cw-prod-admin	cw-prod-admin-settings.json	cw-prod-admin-connection-strings.json
Hangfire	cw-prod-job	cw-prod-job-settings.json	cw-prod-job-connection-strings.json

If you don't know what a key is about, you can find it in Configuration settings - a complete list of all Casewhere settings.

You can also learn more about the service certificates for Casewhere here.

General settings

Update the followings:

- Always on: **On**
- ARR affinity: **On**

For **Worker.Api** only, change the **Platform** to **64 bit**. The other sites should be **32 bit** as-is.

Custom domains

Use the following links to add custom domains for your apps. We recommend using the free-managed certificates offered by Azure:

- Map an existing custom DNS name to Azure App Service
- Secure a custom DNS with a TLS/SSL binding
- Add and manage TLS/SSL certificates

TLS/SSL settings

Configure the binding setings as below:

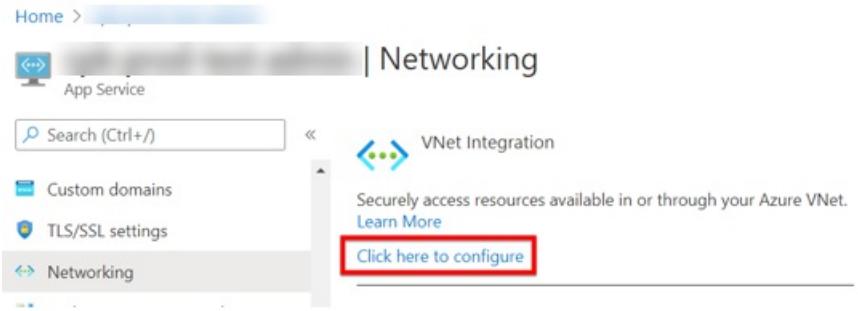
- HTTPS Only: **On**
- Minimum TLS Version: **1.2**

Install the service certificates as described here.

Networking

The replica set is not accessible from the internet. For the Azure App Service to connect to the replica set, you must link it to the VNet shared with the replica set.

From the **Networking** setting, you can configure VNet Integration.

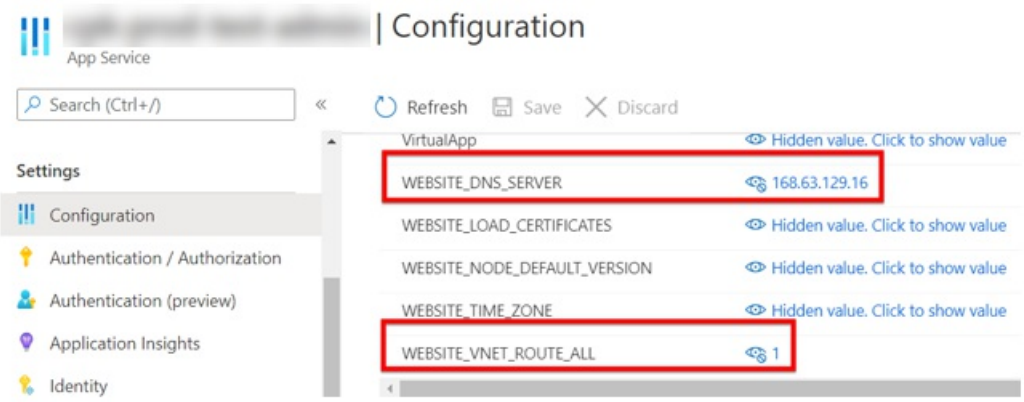


In the next screen, choose the VNet and create a new **Subnet** if not existing with the following settings:

- Subnet address range: **10.1.0.0/24**
- Network security group: select the existing
- Delegate subnet to a service: **Microsoft.Web/serverFarms**

Finally, add the following settings in the **Application settings** of your Azure Websites

- **WEBSITE_DNS_SERVER** with value **168.63.129.16**
- **WEBSITE_VNET_ROUTE_ALL** with value **1**



App Service Logs

Configure the following setings:

- Detailed error messages: **On**
- Failed request tracing: **On**

Configure API key for the background job

Update the following keys in both Casewhere Api (Worker.Api) and Casewhere Job (Hangfire). These keys are to protect the communication between the two sites.

KEY	SAMPLE VALUE	DESCRIPTION
cwjobs:RuntimeApiKey	8sK/3irhASZR/1orkNa50i3vvpSjKn/l7epMRIKSF4g=	A secret string

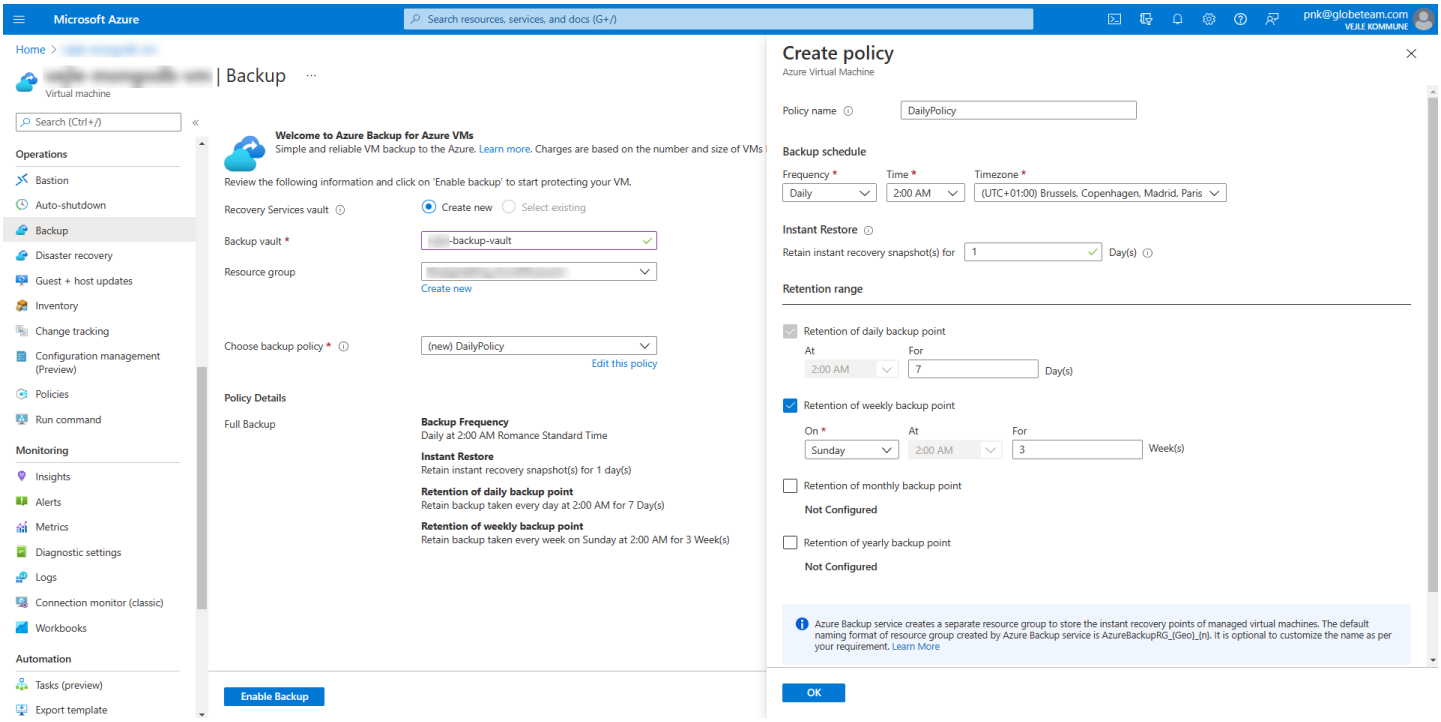
KEY	SAMPLE VALUE	DESCRIPTION
cwjobs:RuntimeApiEndpoint	https://cw-prod-api.casewhere.com/external/v0.1/	Link to your Casewhere Api

Authentication for Casewhere Admin and Casewhere Job

Follow this guideline to configure authentication for Casewhere Admin (Configuration.Web) and Casewhere Job (Hangfire). When this step is done, you can login to Casewhere Admin and start configuring your Casewhere solution.

Configure backup

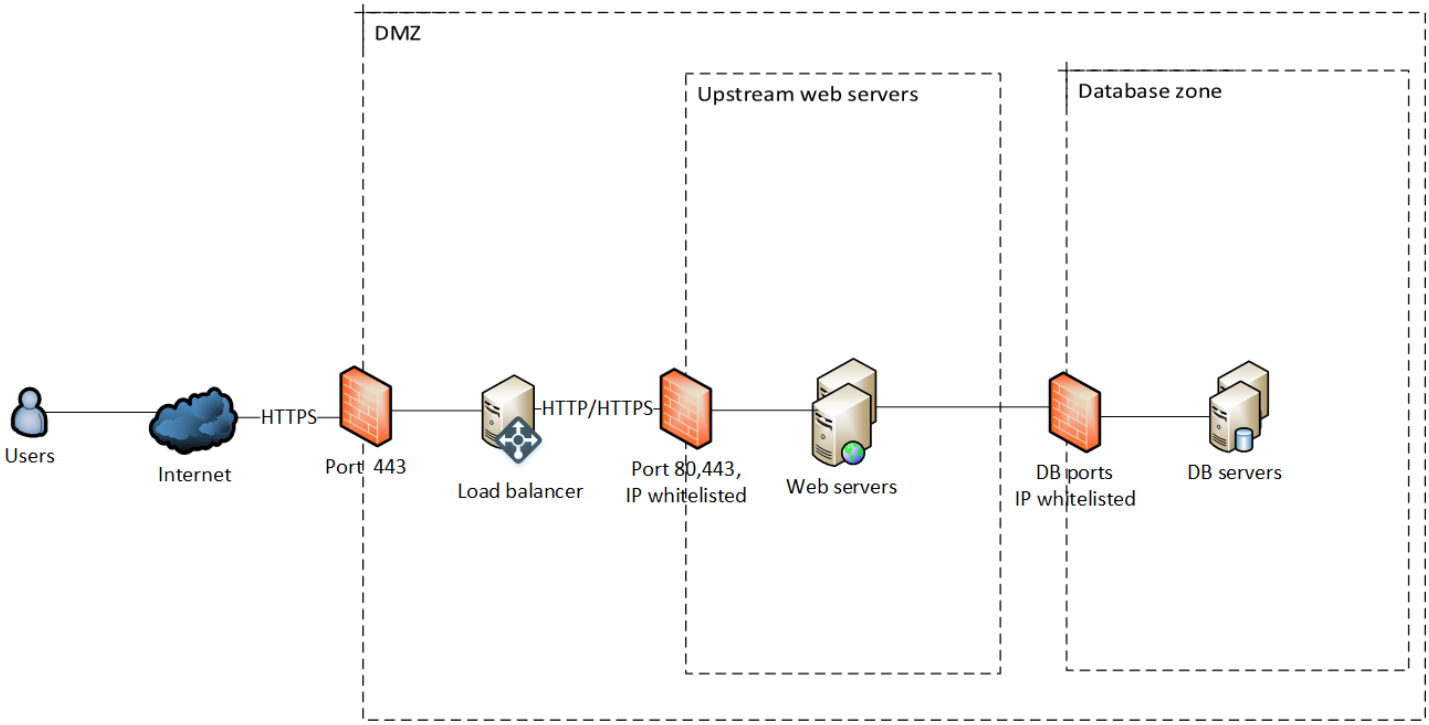
For production workload, it is recommened to enable backup for your MongoDB database. It's always a good idea to maintain a backup policy with multiple restore points.



Deploy Casewhere on-premise

1. Diagram

A typical enterprise deployment can look as follows.



Please note that setting up the network infrastructure is beyond the scope of this document.

2. Prerequisites

Minimum requirements for Dev and Test

- Web server: 2 cores, RAM, 4 GB
- Database server: 2 cores, RAM 8 GB
- Windows Server 2016 or later

Minimum requirements for Production

- Web server: 2 cores, RAM 8 GB
- Database server: 2 cores, RAM 8 GB, SSD
- Windows Server 2022 or later ### Software

The following must be installed before setting up Casewhere:

- ASP.NET 4.5 and Microsoft .NET Framework 4.8 runtime.
- Install Microsoft Visual C++ Redistributable latest version. Restarting the server is required.
- Casewhere 2.8 or later.
- Prepare the service certificates described here.

3. Deploy MongoDB Replica set

MongoDB recommends using a replica set for production workloads. You can follow the guidelines here to install MongoDB and configure the replica set for Casewhere.

For development and testing environments, a single MongoDB instance can be used to host all Casewhere databases, providing a straightforward and easy setup. However, for high-performance systems where strict Service Level Agreements (SLAs) are

required, it is recommended that the database workloads be separated into multiple replica sets. This approach helps optimize performance, scalability, and reliability.

- Read workload
  - CasewhereData
  - CasewhereFileStorage
  - CasewhereHangfire
  - CasewhereKeys

- Write workload
  - CasewhereEvents
  - CasewhereLog

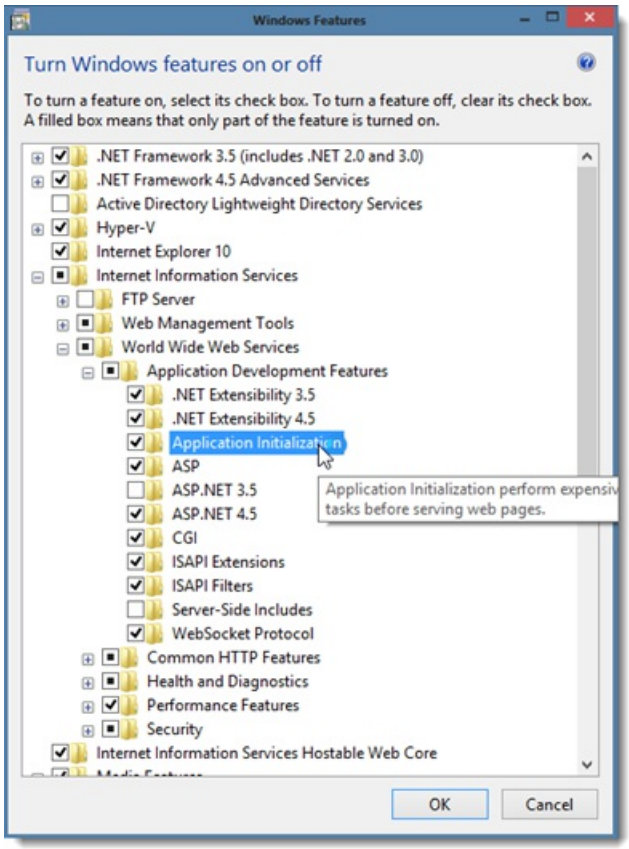
We also recommend using private domain names to reference the VMs instead of IP addresses. For example, use "cw-prod-mongo1.casewhere.db".

4. Deploy Web server

4.1 Add IIS Window Feature

To install Window Features: Internet Information Service (IIS) 8.0+ with Application Initialization, we will use:

- The **Turn Windows features on or off** tool.



- Or use the Powershell script below:

```
Install-WindowsFeature -Name Web-Server -IncludeAllSubFeature
```

4.2 Install Casewhere service certificates

Install the following certificates at the corresponding locations. Each certificate is described in detail here.

CERTIFICATE	SAMPLE FILE	INSTALL LOCATION
SSL Certificates	wildcard.casewhere.com.pfx	Local Machine/Personal
Data protection	encrypt.casewhere.com.pfx	Local Machine/Personal
Service Provider Signing	serviceprovider.signing.casewhere.com.pfx	Local Machine/Personal
IdP Login Public Key	Login Signing Cert.cer	Local Machine/Trusted People

Prepare the SSL certificates for your websites. See the guidelines here for installing an SSL certificate in IIS.

Grant access for **IIS_USRS** user to access the private keys:

- Open **Windows Certificate Manager** (run `certmgr.msc` from the command prompt).
- Navigate to **Personal > Certificates**.
- Right-click on installed private certificates one at a time, and select **All Tasks > Manage Private Keys**.
- Add local **IIS_IUSRS** user with full-control permission.

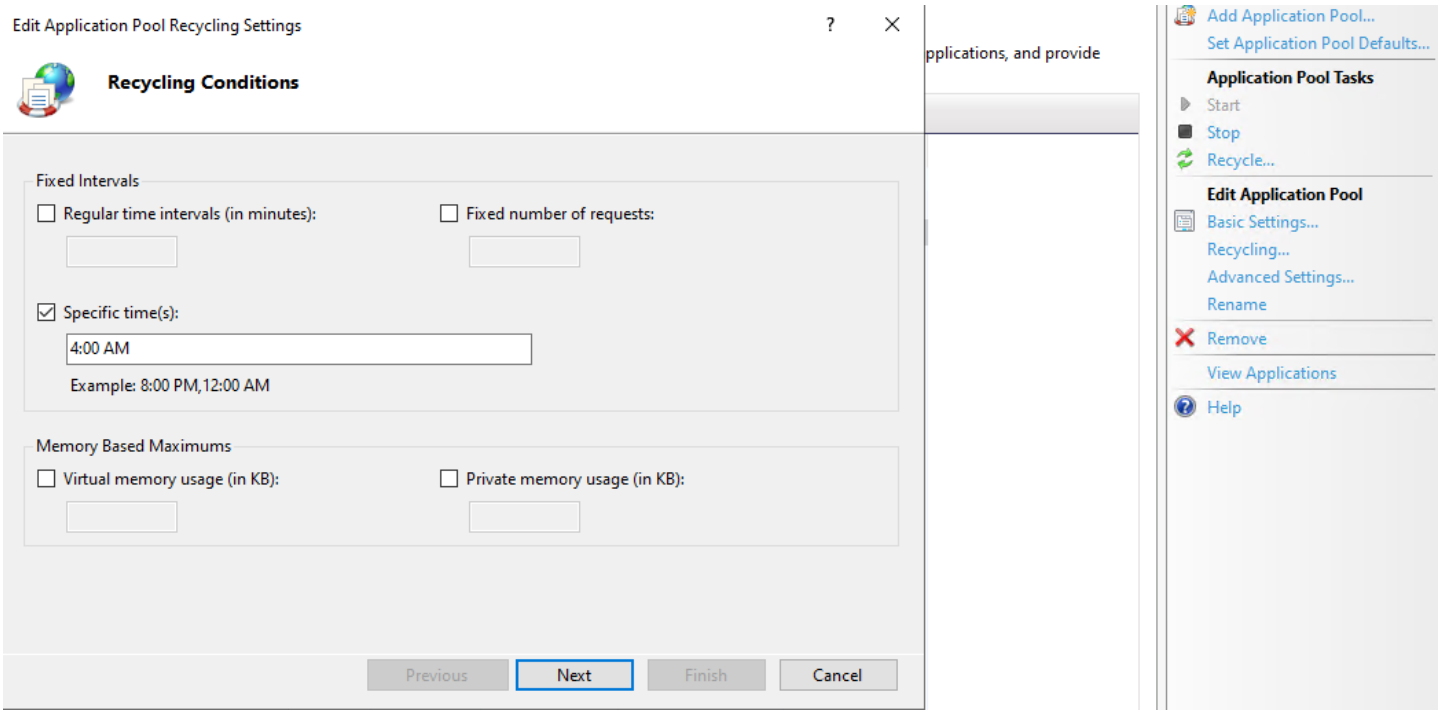
4.3 Create application pools

Each Casewhere site must have a dedicated application pool. It means you must create four application pools for Casewhere Admin, Casewhere Job, Casewhere Api, and Casewhere Site.

In the **Advanced Settings** for each application pool, make sure the following are correctly set up:

- Start Mode: **Always Running**
- Identity: **Network Service**
- Idle Time-out: **0**

To optimize performance, ensure that each application pool is recycled during your system's least busy times. Select the pool, and click **Recycling** in the Actions pane:

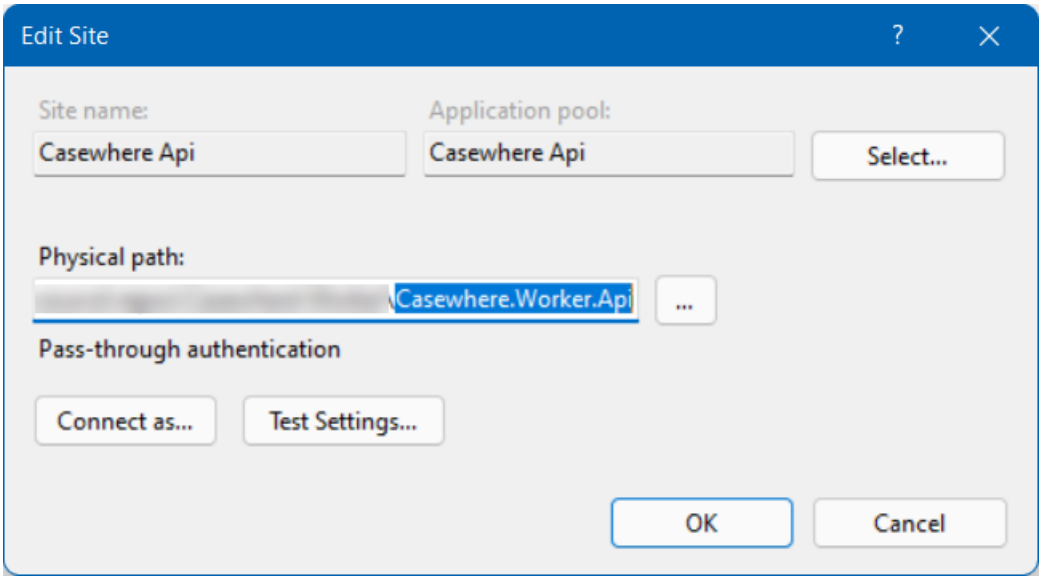


4.4 Create websites

Extract the Casewhere deployment package to the physical path.

For each Casewhere site, please create a new website and associate it with the corresponding application pool.

SITE NAME	CASEWHERE SOURCE (PHYSICAL PATH)
Casewhere Web	Casewhere.Worker.Web
Casewhere Api	Casewhere.Worker.Api
Casewhere Admin	Casewhere.Configuration.Web
Casewhere Job	Casewhere.Hangfire

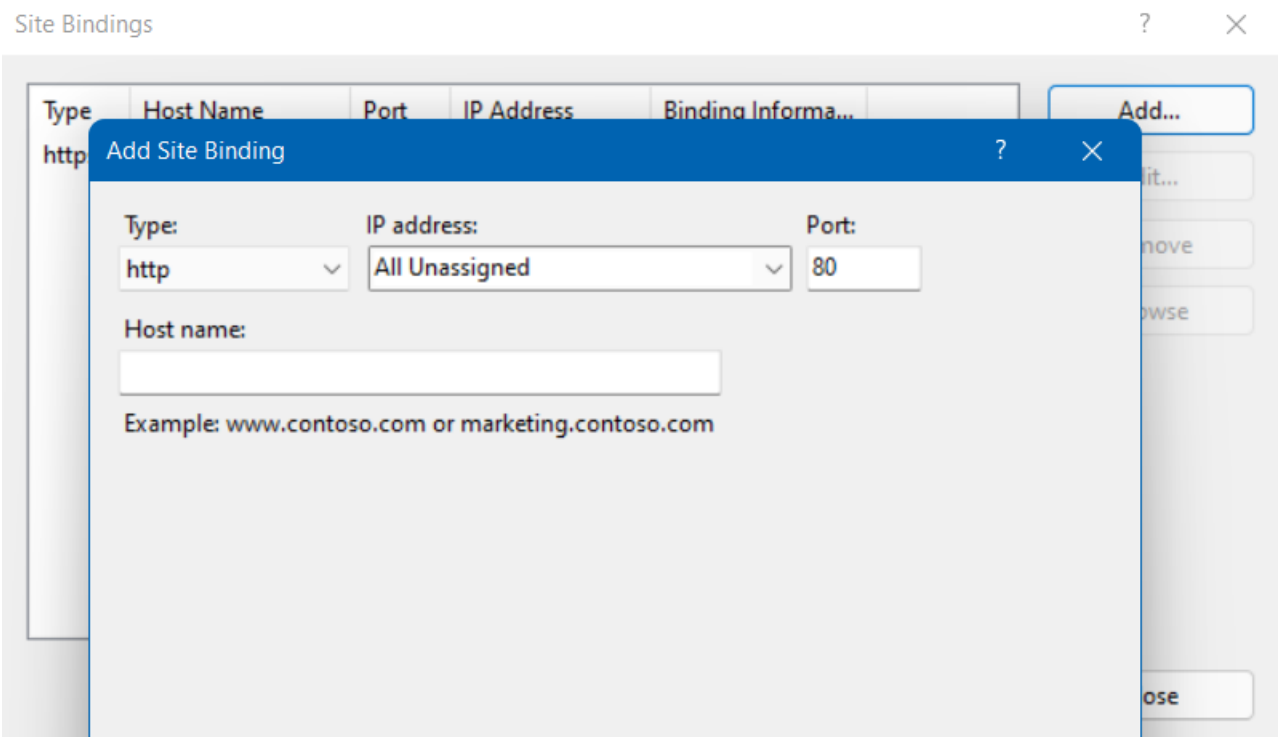


The following must be configured correctly:

- Binding: **Https**
- Hostname: the website domain name
- SSL certificate: the SSL certificate that you installed in the previous step

HTTP Bindings

To enable multiple worker sites using different domains, we need to bind all the domains to Casewhere Web - `Casewhere.Web.CaseWorker`. Select the website, then select **Bindings** in the **Actions** sidebar.



Here you must create bindings for both HTTP and HTTPS. For HTTPS, make sure you select the right certificate corresponding to your site's domain.

Set up HTTP/HTTPS redirect in IIS

Make sure you create HTTP bindings for all the site domains. Follow this article to set up HTTP/HTTPS redirect in IIS.

5. Configure application settings

Make sure you review and edit the settings in the `web.config` files for all Casewhere sites. You can use the default values for many settings, but you must update the database connection strings, file storage, identity provider, and certificates.

You can get database connection strings here.

If you don't know what a key is about, you can find it in Configuration settings - a complete list of all Casewhere settings.

You can also learn more about the service certificates for Casewhere here.

Configure API key for the background job

Update the following keys in both Casewhere Api (Worker.Api) and Casewhere Job (Hangfire). These keys protect communication between the two sites.

KEY	SAMPLE VALUE	DESCRIPTION
cwjobs:RuntimeApiKey	8sK/3irhASZR/1orkNa50i3vvpSjKn/l7epMRIKSF4g=	A secret string
cwjobs:RuntimeApiEndpoint	https://cw-prod-api.casewhere.com/external/v0.1/	Link to your Casewhere Api

6. Authentication for Casewhere Admin and Casewhere Job

Follow this guideline to configure authentication for Casewhere Admin (Configuration.Web) and Casewhere Job (Hangfire). When this step is done, you can login to Casewhere Admin and start configuring your Casewhere solution.

7. Configure auto pings

To keep the sites always alive, we need to configure the auto-pings as below.

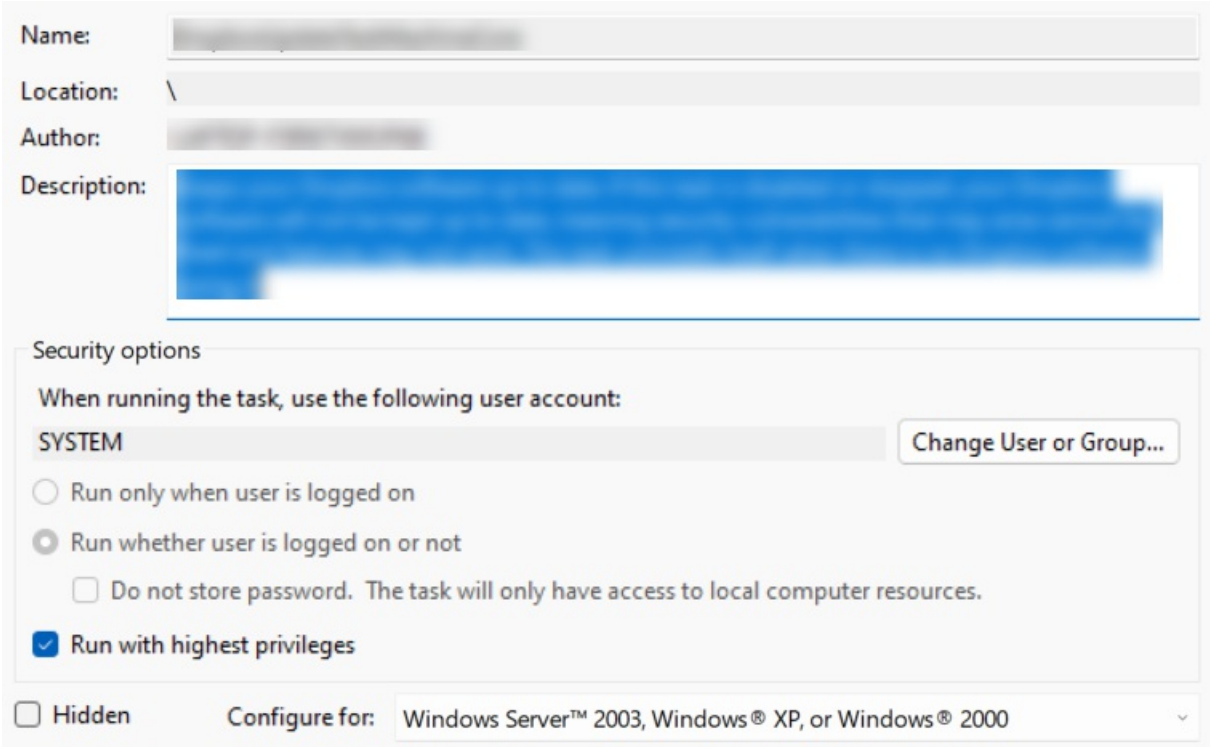
Create a PowerShell script file named **auto-ping.ps1** with below content:

```
Invoke-WebRequest https://<Casewhere Api>/api/v0.1/ping
Invoke-WebRequest https://<Casewhere Job>/ping
Invoke-WebRequest https://<Casewhere Admin>
Invoke-WebRequest https://<Casewhere Web>
```

Open **Task Scheduler** to create a daily task that executes the script every 5 minutes:

- Program/Script: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
- Add arguments: `-windowstyle hidden C:\Casewhere\auto-ping.ps1`

Make sure the task always with **runs whether user is logged on or not**.



Casewhere production deployment checklists

The following checklists ensure that the Casewhere deployment is completed correctly and is fully prepared for production.

Pre-deployment checklist

The following should be prepared before deploying Casewhere to a new environment.

1. Choose the appropriate guideline based on your deployment target:
    - Deploy on Azure
    - Deploy on premise
2. Download the required versions of Casewhere and MongoDB.
3. Prepare certificates as described in Casewhere service certificates. Ensure production environments use distinct certificates from test and development environments.
4. Generate a new `RuntimeApiKey` and store it securely. Ensure production environments use unique keys that are separate from those used in test and development environments.
5. Verify the `web.config` for all sites, ensuring database names and storage container identifiers are appropriate for the specific environment.
6. Set up two distinct database accounts—one for the web application and one for logging—following the MongoDB replica set deployment guideline.

Deployment checklist

Use the following checklist to ensure Casewhere is configured correctly.

1. Each Casewhere site uses a separate IIS application pool.
2. Casewhere Worker API is configured to run in 64-bit mode.
3. All Casewhere sites are set to automatically start and always stay active.
4. Pings are configured to keep sites awake continuously.
5. Application pools are scheduled for recycling during low-traffic periods (e.g., 4:00 AM).
6. All Casewhere sites are running and operational.
7. Confirm successful login to all Casewhere sites.
8. Ensure access to the Log Viewer and the ability to read log data.
9. Verify that TTL indexes are created for Logs and PerfLogs to enable periodic log cleanup.

Post-deployment checklist

The following checklist is to ensure the deployment is finished correctly and ready for production use.

1. Ensure debug mode is turned off for all Casewhere sites.

```
<customErrors mode="On" />
```

1. Enable custom error handling for all Casewhere sites.

```
<compilation debug="false" targetFramework="4.8" />
```

1. Configure an HTTP to HTTPS redirect following this guide.
2. Execute test workflows to confirm that the workflow engine is functioning correctly.
3. Secure the environment by closing any unnecessary ports and access points on the servers.
4. After deployment is complete and before customers begin running smoke tests, back up the production environment.

Important settings

Casewhere.Worker.Web

KEY	RECOMMENDED VALUE	DESCRIPTION
`fcn:StopMonitoringSubFolders`	false	We have a performance issue with it turned on, especially on Azure App Service.

Casewhere.Worker.Api

KEY	RECOMMENDED VALUE	DESCRIPTION
`AllowDiskUse`	false	Do not enable this unless there is a specific instruction.
`dataTrigger:Workers`	2	2 is enough for most cases.
`eventTrigger:Workers`	2	2 is enough for most cases.
`cwjobs:Workers`	3-5	Depending on the size of the web server.
`events:EnableHotAndCold`	false	Do not enable this unless there is a specific instruction.
`ApplyTransaction`	true	For test/production environments, it should be `true`.

Configuration settings

Casewhere provides a lot of settings you can adjust to specifically customize your deployments.

Casewhere Api

Below are the customizable settings for `Casewhere.Worker.Api`.

KEY	TYPE	DESCRIPTION	DEFAULT/SAMPLE VALUE	RELEASE
storage:sysDocumentContainer	`appSettings`	The container for system files. For using with Azure Blob, it's the Blob container. For using with MongoDb GridFs, it's the database collection	systemdocuments	1.0
storage:pluginContainer	`appSettings`	The container for plugins.	plugins	1.0
storage:documentContainer	`appSettings`	The container for runtime files i.e. files added by end-users through workflows	documents	1.0
storage:sysDocument:clientCacheTime	`appSettings`	The cache duration in second	60	2.6
storage:azure:connectionString	`secureAppSettings`	The Azure Storage connection string. This setting is for Azure Queue only	*Azure Storage connection string*	2.0
storage:gridFS:connectionString	`secureAppSettings`	The MongoDb connection string for the file storage database. This setting is for MongoDb GridFs only	*MongoDb connection string*	2.6
cwjobs:RuntimeApiEndpoint	`appSettings`	The API endpoint to process background tasks. In most deployments, we host the background-task engine using the same `Worker Api` instance, but it can be deployed separately from the `Worker Api` if we need to scale them separately	https://api-local.casewhere.com/external/v0.1/	1.0
cwjobs:RuntimeApiKey	`secureAppSettings`	The secret key to protect the communication between `Worker API` and the background task engine	*A secret value*	1.0
cwjobs:Workers	`appSettings`	The number of workers to process background tasks	3	2.0
cwjobs:Timeout	`appSettings`	The timeout for processing background tasks in second	900	2.5
cwJobs:AzureStorageQueue:queuename	`appSettings`	The queue name if you use Azure Queue for the background tasks	cwjobqueue	2.0
cwjobs:AzureStorageQueue:visibleTimeoutSecs	`appSettings`	The visibility timeout in seconds. This setting is for Azure Queue only	3600	2.5
cwjobs:AzureStorageQueue:connectionstrings	`secureAppSettings`	The Azure Storage connection string. This setting is for Azure Queue only	*A secret value*	2.0
UploadTimeout	`appSettings`	The timeout for uploading files from worker sites	00:15:00	2.6
MinifyScript	`appSettings`	If true, Casewhere will minify the generated scripts before responding	false	1.0

KEY	TYPE	DESCRIPTION	DEFAULT/SAMPLE VALUE	RELEASE
WorkerSiteClaimType	`appSettings`	The claim in the token from IdP, which is used to authorize users against the worker site. This setting is `Deprecated`, we recommend using access rules to for user authorization	http://schemas.microsoft.com/identity/claims/tenantdomain	1.0
CacheDuration	`appSettings`	To improve performance, Casewhere cache configuration data and only query MongoDb when needed	00:15:00	2.0
ApplyTransaction	`appSettings`	When true, Casewhere will use transaction when persisting changes to the event source	false	2.6
QueryMaxTime	`appSettings`	The timeout for database queries	00:01:00	2.6
AllowDiskUse	`appSettings`	Utilize disk space for sorting. Read here for more	false	2.6
SearchLimit	`appSettings`	The maximum number of records returned for a search query	5000	2.6
AcceptableDeviationSeconds	`appSettings`	The assertion validation can be failed due to a difference in server time between Casewhere and connected IdP. This setting controls the variation in seconds allowed when validating assertions	3	2.5
EnableBrowserFingerprint	`appSettings`	When true, Casewhere will validate the request signature (IP, Agent) against the issued token	false	2.6
CleanupUserSessionAfterTimeSpan	`appSettings`	User sessions will be deleted in the database after a configurable period to free up disk space	720:00:00	2.6
idpConnection:AllowTokenLifetimeFromConditions	`appSettings`	If true, Casewhere will use the value in `Conditions.NotOnOrAfter` for the token lifetime.	false	2.6
idpConnection:DefaultTokenLifetime	`appSettings`	The default token lifetime. If none is specified in the SAML token from IdP, Casewhere will use this setting.	08:00:00	2.6
dataProtection:KeyProtection:Thumbprint	`appSettings`	The thumbprint of the encryption-key certificate		2.5
dataProtection:KeyProtection:StoreLocation	`appSettings`	The location to store the encryption-key certificate	LocalMachine	2.5
dataProtection:KeyPersistence:MongoDBConnection	`secureAppSettings`	The MongoDb connection string for the key-storage database	*MongoDb connection string*	2.5
dataTrigger:Workers	`appSettings`	The number of workers to process data-change notifications	2	2.5
dataTrigger:ExpireAfterDays	`appSettings`	Old data-change notifications will be deleted in the database to free up disk space	30.00:00:00	2.5
dataTrigger:AvailableFromHours	`appSettings`	It's similar to the visibility timeout in Azure Queue	01:00:00	2.5

KEY	TYPE	DESCRIPTION	DEFAULT/SAMPLE VALUE	RELEASE
dataTrigger:MaxDequeueAttempts	`appSettings`	The maximum number of attempts Casewhere will try to process a data-change notification	10	2.5
jobQueue:ExpireAfterDays	`appSettings`	Old messages will be deleted in the database to free up disk space. This setting is applicable for Mongo Queue only	30.00:00:00	2.6
jobQueue:AvailableFromHours	`appSettings`	It's similar to the visibility timeout in Azure Queue. This setting is applicable for Mongo Queue only	01:00:00	2.6
eventTrigger:ExpireAfterDays	`appSettings`	Old events will be deleted in the database to free up disk space	30.00:00:00	2.6
eventTrigger:InvisibleEventDuration	`appSettings`	It's similar to the visibility timeout in Azure Queue	01:00:00	2.6
eventTrigger:MaxAttemptsForEvent	`appSettings`	The maximum number of attempts Casewhere will try to process an event	10	2.6
eventTrigger:Workers	`appSettings`	The number of workers to process runtime events	2	2.6
serilog:minimum-level	`appSettings`	The minimum logging level	Verbose	2.0
serilog:disableSelfLog	`appSettings`	To debug logging problem	false	2.6
serilog:using:MongoDb	`appSettings`	MongoDb as logging storage	Serilog.Sinks.MongoDb	2.0
serilog:write-to:MongoDb.collectionName	`appSettings`	The MongoDb collection for storing log data	Logs	2.0
serilog:write-to:MongoDb.databaseUrl	`secureAppSettings`	The MongoDb connection string for the log database	*MongoDb connection string*	2.0
perfLog:using:MongoDB	`appSettings`	MongoDb as logging storage	Serilog.Sinks.MongoDB	2.6
perfLog:write-to:MongoDB.collectionName	`appSettings`	The MongoDb collection for storing log data	PerfLogs	2.6
perfLog:write-to:MongoDB.cappedMaxSizeMb	`appSettings`	Casewhere use MongoDb capped collection to store performance logs. The capping size is in Mb	5000	2.6
perfLog:enabled	`appSettings`	Enable or disable performance log	true	2.6
perfLog:write-to:MongoDB.databaseUrl	`secureAppSettings`	The MongoDb connection string for the log database	*MongoDb connection string*	2.6
CasewhereEvents	`connectionStrings`	The MongoDb connection string for the event-source database	*MongoDb connection string*	2.0
CasewhereData	`connectionStrings`	The MongoDb connection string for the snapshot database	*MongoDb connection string*	2.0
Hangfire	`connectionStrings`	The MongoDb connection string for the snapshot database	MongoDb connection string	2.0
CasewhereLog	`connectionStrings`	The MongoDb connection string for the snapshot database	MongoDb connection string	2.0

KEY	TYPE	DESCRIPTION	DEFAULT/SAMPLE VALUE	RELEASE
CasewhereAuditLog	`connectionStrings`	The MongoDb connection string for the snapshot database	MongoDb connection string	2.6

Casewhere Web

Below are the customizable settings for `Casewhere.Web.CaseWorker`.

KEY	TYPE	DESCRIPTION	DEFAULT/SAMPLE VALUE	RELEASE
ApiEndPoint	`appSettings`	The `Worker API` endpoint	https://api-local.casewhere.com/api/v0.1/	1.0
RuntimeApiEndPoint	`appSettings`	The `Runtime API` endpoint	https://api-local.casewhere.com/external/v0.1/	1.0
TextResourceEndPoint	`appSettings`	The endpoint to download text resources	https://api-local.casewhere.com/languages/	1.0
AuthEndPoint	`appSettings`	The API endpoint for authentication	https://api-local.casewhere.com/	1.0
LoginEndpoint	`appSettings`	The login endpoint	https://api-local.casewhere.com/signon?relayState=	1.0
LogoutEndpoint	`appSettings`	The logout endpoint	https://api-local.casewhere.com/signoff	1.0
AdminEndpoint	`appSettings`	The endpoint to Casewhere Admin	https://admin-local.casewhere.com/	2.6
serilog:minimum-level	`appSettings`	The minimum logging level	Verbose	2.7
serilog:disableSelfLog	`appSettings`	To debug logging problem	false	2.7
serilog:using:MongoDb	`appSettings`	MongoDb as logging storage	Serilog.Sinks.MongoDb	2.7
serilog:write-to:MongoDb.collectionName	`appSettings`	The MongoDb collection for storing log data	Logs	2.7
serilog:write-to:MongoDb.databaseUrl	`secureAppSettings`	The MongoDb connection string for the log database	*MongoDb connection string*	2.7

Casewhere Admin

Below are the customizable settings for `Casewhere.Configuration.Web`.

KEY	TYPE	DESCRIPTION	DEFAULT/SAMPLE VALUE	RELEASE
idpConnection:SPCertStoreLocation	`appSettings`	The store location for the service provider (Casewhere) signing certificate	LocalMachine	2.6
idpConnection:SPCertStoreName	`appSettings`	The name of the store for the service provider (Casewhere) signing certificate	My	2.6
idpConnection:IdPCertStoreLocation	`appSettings`	The store location for the IdP signing certificate	LocalMachine	2.6
idpConnection:IdPCertStoreName	`appSettings`	The name of the store for the IdP signing certificate	TrustedPeople	2.6
idpConnection:SPSigningCertificateThumbprint	`appSettings`	The thumbprint of the service provider (Casewhere) signing certificate	*Thumbprint string*	2.6
idpConnection:SignatureAlgorithm	`appSettings`	Signing algorithm	SHA256	2.6
idpConnection:IdPUrl	`appSettings`	The IdP endpoint	https://login-demo.casewhere.com/runtime/	2.6
idpConnection:IdPMetadataFolder	`appSettings`	The folder contains the SAML metadata from the IdP. Make sure there is a metadata.xml inside it	~/Metadata/OnPremiseIdP	2.6
idpConnection:EntityId	`appSettings`	If not specified, the current domain will be used		2.6

KEY	TYPE	DESCRIPTION	DEFAULT/SAMPLE VALUE	RELEASE
idpConnection:LogoutResponseUri	`appSettings`	The Casewhere endpoint to handle logout callback from IdP	https://admin-local.casewhere.com/auth/signoffconsumer	2.6
idpConnection:EnableSSO	`appSettings`	If false, users are always required to login	false	2.6
authorization:EnableBrowserFingerprint	`appSettings`	When true, Casewhere will validate the request signature (IP, Agent) against the issued token	true	2.6
idpConnection:AllowTokenLifetimeFromConditions	`appSettings`	If true, Casewhere will use the value in `Conditions.NotOnOrAfter` for the token lifetime	false	2.6
idpConnection:DefaultTokenLifetime	`appSettings`	The default token lifetime. If none is specified in the SAML token from IdP, Casewhere will use this setting.	08:00:00	2.6
CasewhereAdminClaim	`appSettings`	The claim type is used for authorization. The issued tokens from IdP must have the claim	BackendRoles	2.6
CasewhereAdminClaimValue	`appSettings`	Casewhere will validate the claim against this value to decide if the user is authorized or not. You can specify multiple values separated by commas	CasewhereAdmin	2.6
CMSAdminClaim	`appSettings`	The claim type is used for authorization for the CMS Admin	BackendRoles	2.6
CMSAdminClaimValue	`appSettings`	Casewhere will validate the claim against this value to decide if the user is authorized for the CMS Admin or not. You can specify multiple values separated by commas	CMSAdmin	2.6
storage:sysDocumentContainer	`appSettings`	The container for system files. For using with Azure Blob, it's the Blob container. For using with MongoDb GridFs, it's the database collection	systemdocuments	1.0
storage:pluginContainer	`appSettings`	The container for plugins.	plugins	1.0
storage:documentContainer	`appSettings`	The container for runtime files i.e. files added by end-users through workflows	documents	1.0
storage:azure:connectionString	`secureAppSettings`	The Azure Storage connection string. This setting is for Azure Queue only	*Azure Storage connection string*	2.0
storage:gridFS:connectionString	`secureAppSettings`	The MongoDb connection string for the file storage database. This setting is for MongoDb GridFs only	*MongoDb connection string*	2.6
CacheDuration	`appSettings`	To improve performance, Casewhere cache configuration data and only query MongoDb when needed	00:15:00	2.0
ApplyTransaction	`appSettings`	When true, Casewhere will use transaction when persisting changes to the event source	false	2.6
serilog:minimum-level	`appSettings`	The minimum logging level	Verbose	2.0
serilog:disableSelfLog	`appSettings`	To debug logging problem	false	2.6
serilog:using:MongoDb	`appSettings`	MongoDb as logging storage	Serilog.Sinks.MongoDb	2.0
serilog:write-to:MongoDb.collectionName	`appSettings`	The MongoDb collection for storing log data	Logs	2.0
serilog:write-to:MongoDb.databaseUrl	`secureAppSettings`	The MongoDb connection string for the log database	*MongoDb connection string*	2.0
dataProtection:KeyProtection:Thumbprint	`appSettings`	The thumbprint of the encryption-key certificate		2.5
dataProtection:KeyProtection:StoreLocation	`appSettings`	The location to store the encryption-key certificate	LocalMachine	2.5

KEY	TYPE	DESCRIPTION	DEFAULT/SAMPLE VALUE	RELEASE
dataProtection:KeyPersistence:MongoDBConnection	`secureAppSettings`	The MongoDb connection string for the key-storage database	*MongoDb connection string*	2.5
CasewhereEvents	`connectionStrings`	The MongoDb connection string for the event-source database	*MongoDb connection string*	2.0
CasewhereData	`connectionStrings`	The MongoDb connection string for the snapshot database	*MongoDb connection string*	2.0
Hangfire	`connectionStrings`	The MongoDb connection string for the snapshot database	MongoDb connection string	2.0
CasewhereLog	`connectionStrings`	The MongoDb connection string for the snapshot database	MongoDb connection string	2.0

Casewhere Job

Below are the customizable settings for `Casewhere.Hangfire`.

KEY	TYPE	DESCRIPTION	DEFAULT/SAMPLE VALUE	RELEASE
idpConnection:SPCertStoreLocation	`appSettings`	The store location for the service provider (Casewhere) signing certificate	LocalMachine	2.6
idpConnection:SPCertStoreName	`appSettings`	The name of the store for the service provider (Casewhere) signing certificate	My	2.6
idpConnection:IdPCertStoreLocation	`appSettings`	The store location for the IdP signing certificate	LocalMachine	2.6
idpConnection:IdPCertStoreName	`appSettings`	The name of the store for the IdP signing certificate	TrustedPeople	2.6
idpConnection:SPSigningCertificateThumbprint	`appSettings`	The thumbprint of the service provider (Casewhere) signing certificate	*Thumbprint string*	2.6
idpConnection:SignatureAlgorithm	`appSettings`	Signing algorithm	SHA256	2.6
idpConnection:IdPUrl	`appSettings`	The IdP endpoint	https://login-demo.casewhere.com/runtime/	2.6
idpConnection:IdPMetadataFolder	`appSettings`	The folder contains the SAML metadata from the IdP. Make sure there is a metadata.xml inside it	~/Metadata/OnPremiseIdP	2.6
idpConnection:EntityId	`appSettings`	If not specified, the current domain will be used		2.6
idpConnection:LogoutResponseUri	`appSettings`	The Casewhere endpoint to handle logout callback from IdP	https://admin-local.casewhere.com/auth/signoffconsumer	2.6
idpConnection:EnableSSO	`appSettings`	If false, users are always required to login	false	2.6
authorization:EnableBrowserFingerprint	`appSettings`	When true, Casewhere will validate the request signature (IP, Agent) against the issued token	true	2.6
idpConnection:AllowTokenLifetimeFromConditions	`appSettings`	If true, Casewhere will use the value in `Conditions.NotOnOrAfter` for the token lifetime	false	2.6
idpConnection:DefaultTokenLifetime	`appSettings`	The default token lifetime. If none is specified in the SAML token from IdP, Casewhere will use this setting.	08:00:00	2.6
CasewhereAdminClaim	`appSettings`	The claim type is used for authorization. The issued tokens from IdP must have the claim	BackendRoles	2.6
CasewhereAdminClaimValue	`appSettings`	Casewhere will validate the claim against this value to decide if the user is authorized or not. You can specify multiple values separated by commas	CasewhereAdmin	2.6
serilog:minimum-level	`appSettings`	The minimum logging level	Verbose	2.0

KEY	TYPE	DESCRIPTION	DEFAULT/SAMPLE VALUE	RELEASE
serilog:disableSelfLog	`appSettings`	To debug logging problem	false	2.6
serilog:using:MongoDb	`appSettings`	MongoDb as logging storage	Serilog.Sinks.MongoDb	2.0
serilog:write-to:MongoDb.collectionName	`appSettings`	The MongoDb collection for storing log data	Logs	2.0
serilog:write-to:MongoDb.databaseUrl	`secureAppSettings`	The MongoDb connection string for the log database	*MongoDb connection string*	2.0
cwjobs:RuntimeApiKey	`secureAppSettings`	The secret key to protect the communication between the `Hangfire` instance and the `Worker API`	*A secret value*	1.0
cwjobs:Workers	`appSettings`	The number of workers to process background tasks	3	2.0
cwjobs:Timeout	`appSettings`	The timeout for processing background tasks in second	900	2.5
CasewhereEvents	`connectionStrings`	The MongoDb connection string for the event-source database	*MongoDb connection string*	2.0
CasewhereData	`connectionStrings`	The MongoDb connection string for the snapshot database	*MongoDb connection string*	2.0
Hangfire	`connectionStrings`	The MongoDb connection string for the snapshot database	MongoDb connection string	2.0
CasewhereLog	`connectionStrings`	The MongoDb connection string for the snapshot database	MongoDb connection string	2.0
dataProtection:KeyProtection:Thumbprint	`appSettings`	The thumbprint of the encryption-key certificate		2.5
dataProtection:KeyProtection:StoreLocation	`appSettings`	The location to store the encryption-key certificate	LocalMachine	2.5
dataProtection:KeyPersistence:MongoDBConnection	`secureAppSettings`	The MongoDb connection string for the key-storage database	*MongoDb connection string*	2.5

Casewhere service certificates

Some features in Casewhere require an X.509 certificate to function, such as encrypting data, signing authentication requests, etc. Although you can purchase X.509 certificates from a trusted certification authority, for certain scenarios, it's totally OK to use self-signed certificates for these purposes, so if you're not familiar with them, you can learn to create them here.

To work with Azure App Service, you can learn how to upload a private certificate here. Uploading public certificates can be done as described here.

Key protection certificate

Casewhere encrypts data using symmetric cryptography. It stores the cryptographic keys in MongoDB and protects them at rest using an X.509 - the key-protection certificate.

After installing the certificate, you must update the following settings in three Casewhere applications, i.e., **Worker.Api**, **Configuration.Web** and **Hangfire**:

KEY	DESCRIPTION
dataProtection:KeyProtection:Thumbprint	The certificate thumbprint. For example: `0c456d28086b4a13aa2a11730dc1dab4e8ba2922`
dataProtection:KeyProtection:StoreLocation	The place where you install the certificate. For Azure App Service, you must set it to `CurrentUser`.

Service provider signing certificate

The communication between Casewhere, as a service provider, and identity providers must be secured. It is required that Casewhere must supply a X.509 certificate for signing/verifying authentication requests.

After installing the certificate, you must update the following settings in two Casewhere applications i.e. **Configuration.Web** and **Hangfire**:

KEY	DESCRIPTION
idpConnection:SPSigningCertificateThumbprint	The certificate thumbprint. For example, `d95c06939cb37f18c31ffcfebe76e8d9e0b040c2`
idpConnection:SPCertStoreLocation	The place where you install the certificate. For Azure App Service, you must set it to `CurrentUser`.
idpConnection:SPCertStoreName	The store name where you install the certificate. For most deployments, it is `My`.

Identity provider public certificate

To secure communication between Casewhere and identity providers, the identity provider needs to provide a public certificate. This certificate is crucial for verifying the authenticity of authentication responses sent to Casewhere. When Casewhere receives an authentication response, it uses the public certificate to validate the response's signature, ensuring it comes from a trusted identity provider and hasn't been tampered with during transmission. This method helps maintain the integrity and security of the authentication process, preventing unauthorized access and ensuring that only legitimate responses are accepted.

After installing the certificate, you must update the following settings in two Casewhere applications, i.e., **Configuration.Web** and **Hangfire**:

KEY	DESCRIPTION
idpConnection:IdPCertStoreLocation	The place where you install the certificate. For Azure App Service, you must set it to `CurrentUser`.
idpConnection:IdPCertStoreName	The place where you install the certificate. For Azure App Service, you must set it to `My`. For on-premise deployments, a common practice is installing the certificate in `TrustedPeople`.

Deploy MongoDB replica set

Introduction

Replica sets should always have an odd number of members. This ensures that elections will proceed smoothly. For production deployments, you should maintain as much separation between members as possible by hosting the Mongod instances on separate machines. When using virtual machines for production deployments, you should place each Mongod instance on a separate host server serviced by redundant power circuits and redundant network paths.

When possible, use a logical DNS hostname instead of an IP address, particularly when configuring replica set members or sharded cluster members. The use of logical DNS hostnames avoids configuration changes due to IP address changes.

Before you can deploy a replica set, you must install MongoDB on each system that will be part of your replica set. To save effort and reduce the risk of misconfiguration, we recommend setting up one server and cloning it to create other replica set members. How to clone a VM is beyond the scope of this document.

More about the MongoDB replica set can be found here:

- Replica Set Members — MongoDB Manual
- Replica Set Deployment Architectures — MongoDB Manual
- Deploy Replica Set With Keyfile Authentication – MongoDB Manual

1. Create a replica set key file

To install MongoDB with a replica set, we need to generate a random string and store it in **C:\keys\replicaset-keys.yaml**. We will copy this key file to each MongoDB server for setting the replica set.

A key's length must be between 6 and 1024 characters and may only contain characters in the base64 set. All members of the replica set must share at least one common key. You can generate the key following steps below:

- Download OpenSSl from here.
- Extract OpenSSL Zip and navigate to bin folder.
- Open the PowerShell window and run the script below with **path-to-key-file**, which is the place that you will store the key file.

```
.\openssl rand -base64 756 > "path-to-key-file"
```

Limit access to this file so only **Network Service** and **Administrator** accounts can **read** from it.

2. Install MongoDB on each member

Install MongoDB

Download the binaries from the MongoDB Download Center. Double-click the **.msi** file and follow the Windows Installer steps to install MongoDB.

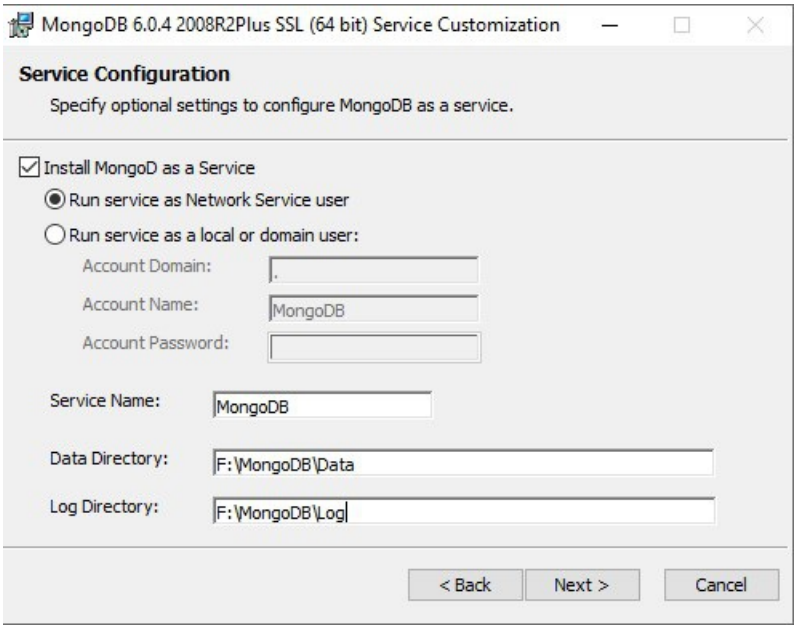
Make sure the following options are selected:

- [x] Install MongoD as a Service.
  - [x] Run service as Network Service user.

Use a **Data Disk** to store logs and data. You **must never use** an **OS Disk** here.

- Data Directory: "F:\MongoDB\Data".

- Log Directory: "F:\MongoDB\Log".



Let's skip the option **Install MongoDB Compass** in the next step.

The following tools are needed to configure MongoDB in the next steps. Please download and install them:

- **MongoDB 6.0+:**
  - Download and install **Studio3T** here. We recommend you set the **Shell-metric** view as preferred when you start using it.
  - Download MongoDB Shell here](https://www.mongodb.com/try/download/shell). Extract the package and copy files in the folder into **C:\Program Files\MongoDB\Server\x.y\bin**.
- **MongoDB < 6.0:**
  - Download and install **Robo3T** here.

Enable access control

Make sure to locate and open the folder where MongoDB is installed, which typically can be found at **C:\Program Files\MongoDB\Server\x.y\bin**.

Open the configuration file **mongod.cfg** and change the settings as below:

- **security.keyFile**: The key file's path. For example, "C:\keys\replicaset-keys.yaml".
- **replication.replSetName**: Replica set name should have the information about the project and environment.
- **net.bindIp**: Since it's hard to foresee how many members and clients to support at this step, it is more flexible to set the binding to **0.0.0.0** and control database access at higher levels such as the machine's Firewall, Network Security Group.

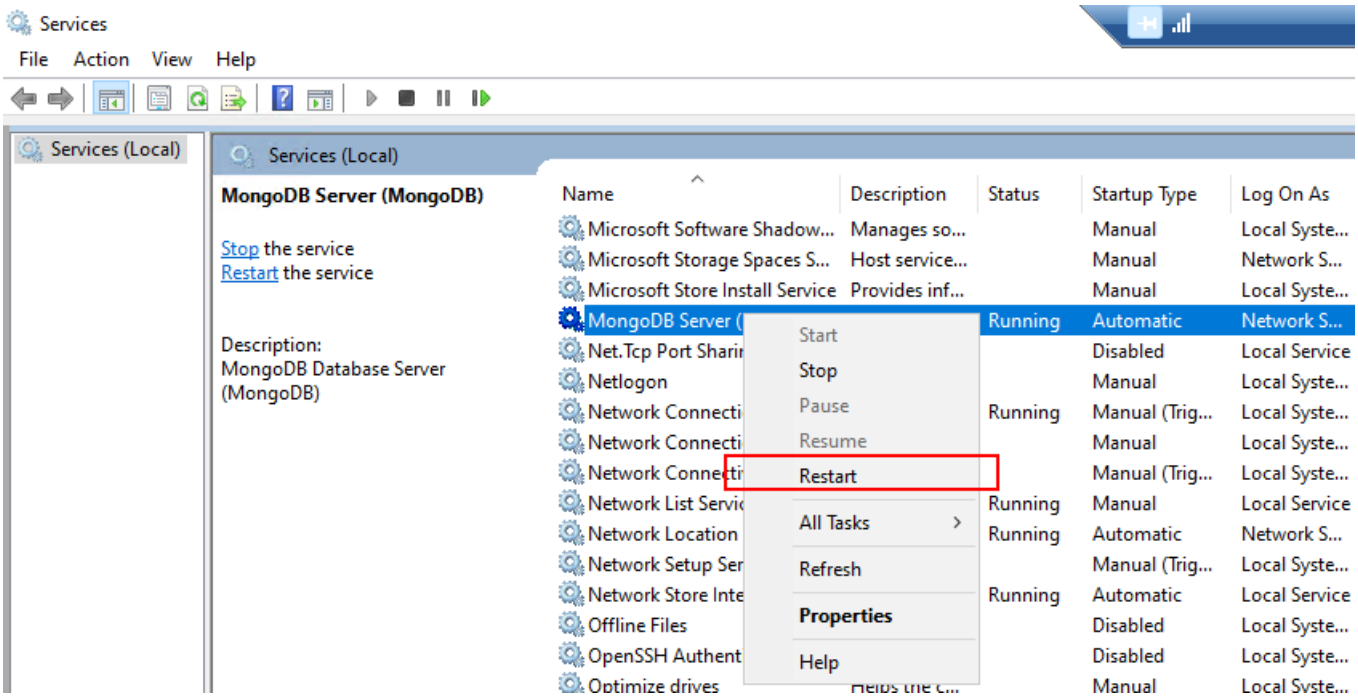
For example:

```
net:
  port: 27017
  bindIp: 0.0.0.0
security:
  keyFile: C:\keys\replicaset-keys.yaml
replication:
  replSetName: "cw-prod"
```

Finally, save your changes and restart the MongoDB service using Powershell:

```
net stop mongodb
net start mongodb
```

You can also restart MongoDB in Windows Services:



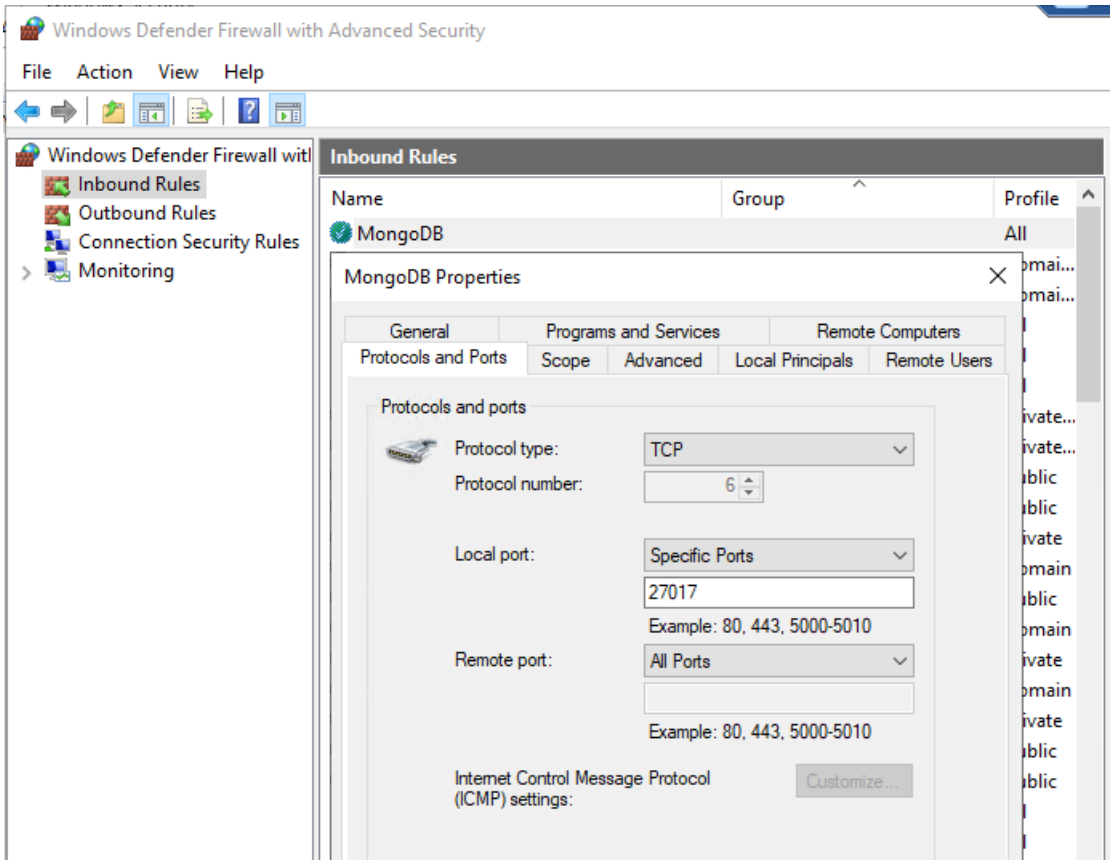
3. Initiate the replica set

Ensure all replica set members are ready and accessible

Do not start this step until you have all servers with MongoDB installed and configured.

For all servers, do the following:

- Install and configure MongoDB as described in the previous steps
- Add an inbound rule to the **Firewall** to have 27017 accessible from outside



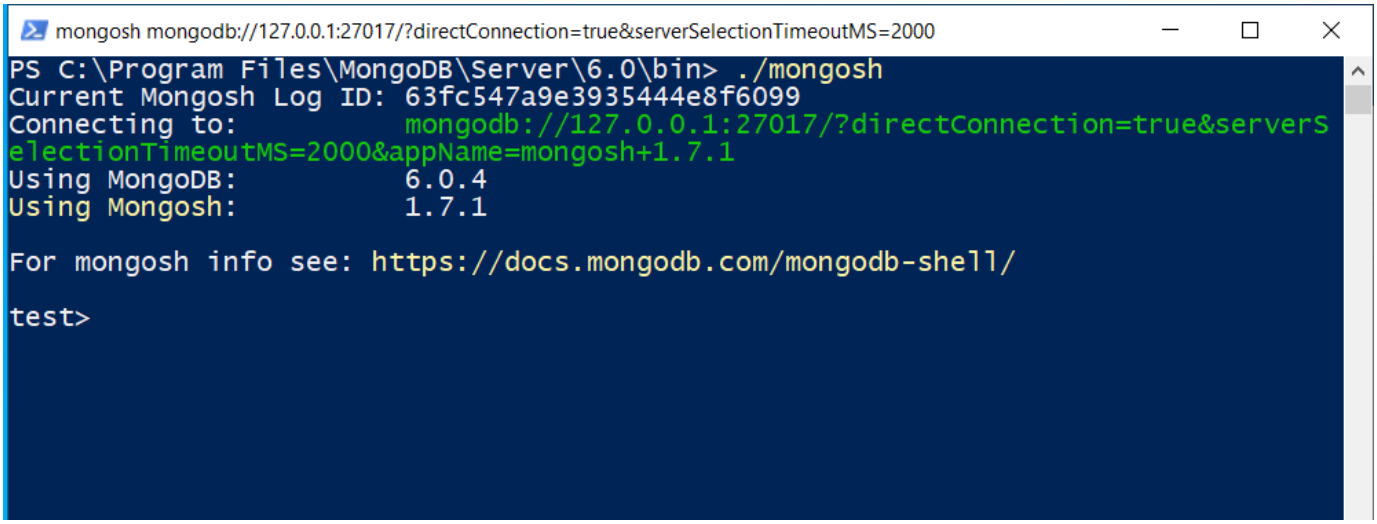
You can set up one server and then duplicate it to create others. Cloning a server is not covered in this document.

Initiate the replica set from the primary member

Navigate to the MongoDB Shell folder, which is typically located at **C:\Program Files\MongoDB\Server\x.y\bin**.

Connect to the MongoDB shell to interact with mongod instances over the localhost interface. You must run the MongoDB shell on the same physical machine as the mongod instance.

- **MongoDB 6.0+:** Use the new mongosh.



- **MongoDB < 6.0:** Use the legacy mongo shell.

```
![image-20211116125919559](../../images/posts/image-20211116125919559.png)
```

The localhost interface is available only until the first user is created for the deployment. It closes after the creation of the initial user

Run the rs.initiate() method

From the mongo shell, run the `rs.initiate()` method. The following example initiates a three-member replica set with an arbiter:

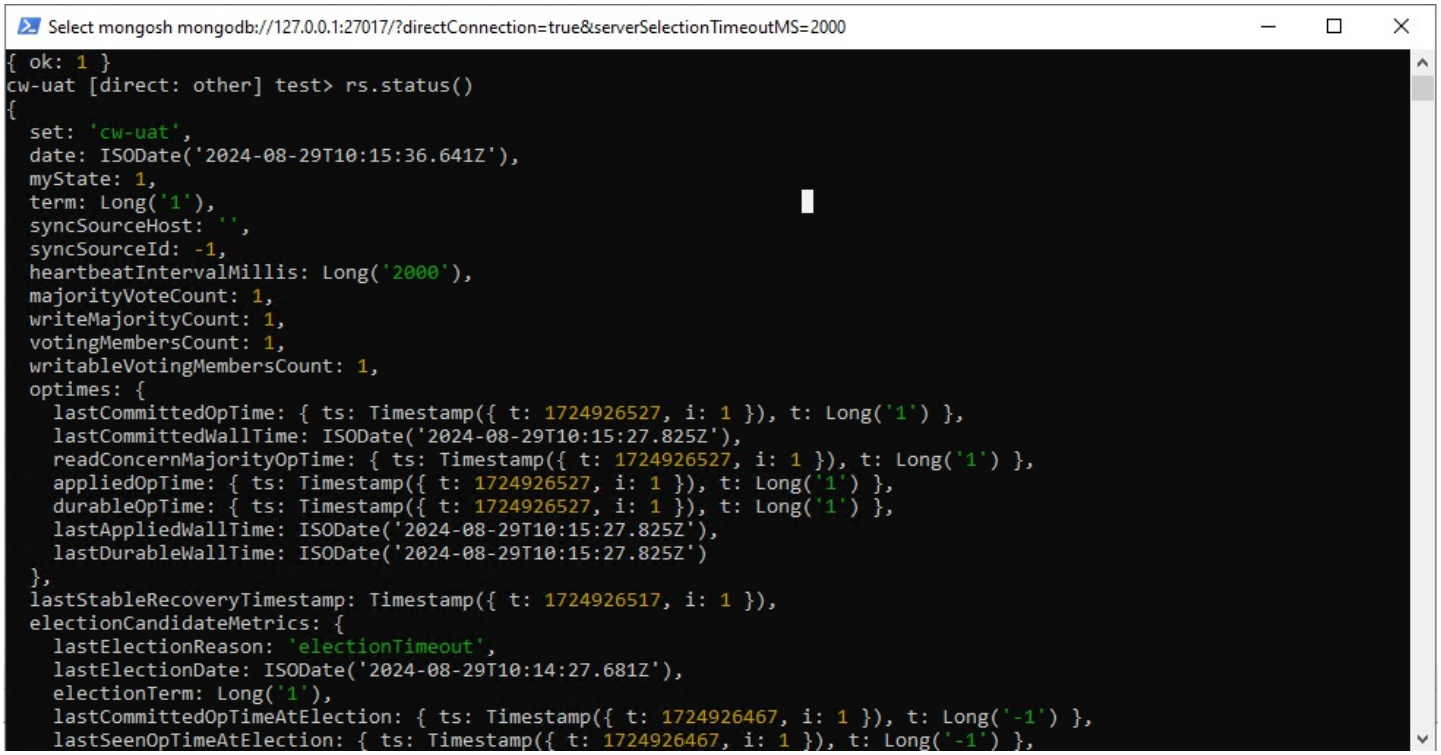
- **_id**: The replica set name that we configured in **mongod.cfg** in the previous step.
- **members[]**: It is recommended to use hostnames for production environments. For development and testing, you can use IP addresses.

If the replica set is created successfully, the following result will be returned.

```
{ ok: 1}
```

Verify replica set

You can run `rs.status()` to verify the status of the replica set.



4. Create users

The localhost interface is only available when no users have been created for the deployment.

Create the admin user

First, use the MongoDB Shell to connect to the primary node. Use the command `rs.status()` to identify the primary member. Then, execute the following command to create the **admin** user.

Be sure to use a **password manager** to generate and store the password for the **admin** user. This user account will be used to administer the database and create other user accounts.

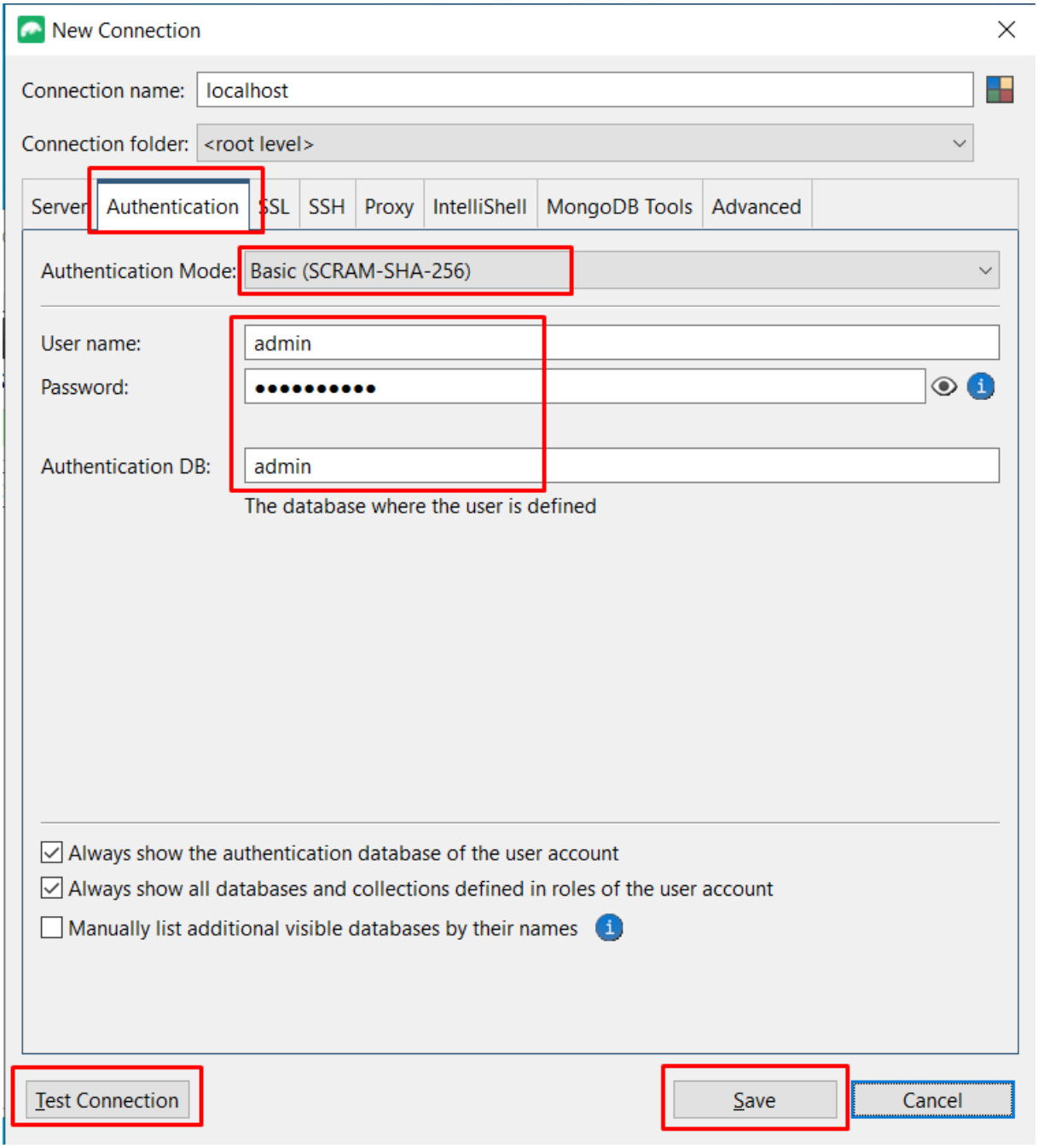
Authenticate as the admin user

- Follow the steps below to authenticate using the **admin** user:

  Using Studio 3T

  1. Click the **Connect** button in the ribbon, then click **New Connection**.
  2. Select **Manually configure my connection settings** and click **Next**.
  3. Enter the **Connection name**. On the **Authentication** tab, select **Basic (SCRAM-SHA-256)** as the authentication

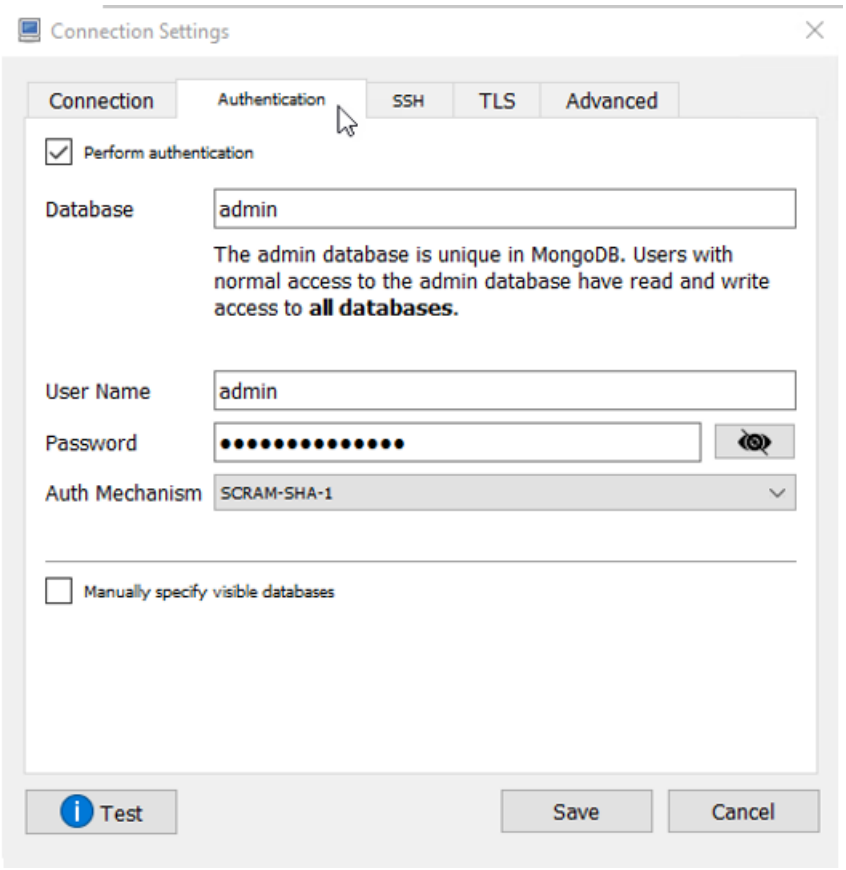
mode, then enter the **Username** and **Password**; also, use **admin** for **Authentication DB**.



- Click **Test Connection** to verify the connection to MongoDB. Ensure that all statuses are **OK**.
- Click **Save** to create the connection.

Using Robo3T

- Edit the **localhost** connection.
- In the **Connection Settings** dialog, navigate to the **Authentication** tab and enable **Perform authentication**.
- Enter **User Name** and **Password** using the **admin** credentials.



Create additional users for your deployment

You should never use the **admin** account for your applications. Follow the least privilege principle by creating roles defining the precise access rights required for specific users. Then, create users and assign them only the roles necessary for their tasks. A user can be either a person or a client application. For the Casewhere deployment, you will need at least two users.

- Web application account: The account is primarily used by Casewhere web applications and should have both read and write access to all databases.
- Logging account: For security purposes, no one should be able to modify the log data once it is saved to the database.

It is recommended to include environment names such as DEV, TEST, and PROD in the database name. Passwords should be generated and managed using a password manager application.

5. Connection strings

Assuming the replica set **cw-prod** has the following members:

- cw-prod-mongo1.casewhere.db:27017 (Data-Bearing)
- cw-prod-mongo2.casewhere.db:27017 (Data-Bearing)
- cw-prod-mongo3.casewhere.db:27017 (Arbiter)

The connection string will look like the example below. As you can see, there is no need to include the **Arbiter** in the connection string.

```
mongodb://username:password@cw-prod-mongo1.casewhere.db:27017,cw-prod-mongo2.casewhere.db:27017/dbname?authSource=admin&replicaSet=cw-prod&retryWrites=true&w=majority
```

- If the **username** or **password** includes the following characters, those characters must be converted using percent-encoding:

```
: / ? # [ ] @
```

- The `dbname` should be replaced with the following values:
    - CasewhereEvents_PROD
    - CasewhereData_PROD
    - CasewhereFileStorage_PROD
    - CasewhereHangfire_PROD
    - CasewhereKeys_PROD
    - CasewhereLog_PROD

Authentication for Casewhere Admin

Introduction

The Casewhere Admin site acts as a SAML2 (Security Assertion Markup Language 2.0) service provider, meaning it relies on identity providers (IdPs) that support the SAML2 protocol for authenticating users. The flexibility of this setup allows Casewhere Admin to integrate with a variety of IdPs, as long as they are compatible with the SAML2 standard.

This guide will specifically cover configuring authentication for the Casewhere Admin site using Identify as the IdP. However, the steps outlined are broadly applicable and can also be used when integrating Casewhere Admin with other SAML2-compatible IdPs. Additionally, these instructions can be applied to configure authentication for the Casewhere Job site using Identify.

Prerequisites

- Casewhere Admin is up and running.
- Identify is up and running.

Configuration steps

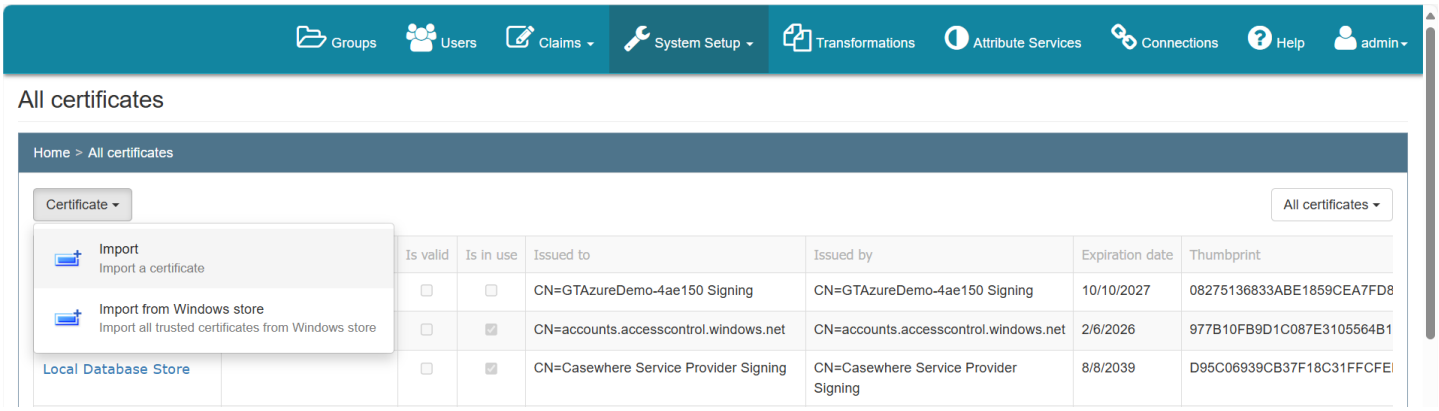
Create SP signing certificate

An SP (Service Provider) signing certificate in SAML (Security Assertion Markup Language) is a digital certificate that the Service Provider uses to sign SAML authentication requests and, in some cases, SAML assertions before sending them to the Identity Provider (IdP). The IdP then uses the public key from the SP's signing certificate to validate the signature, ensuring the integrity and authenticity of the data.

Step 1: Create a signing certificate. You can follow the guideline here to generate a self-signed certificate, which is sufficient for most deployments.

Step 2: Install the certificate, including the private key, on the Casewhere web server, and ensure that **IIS_USRS** has permission to read it.

Step 3: Export the public key and upload it to the Identify server. Log in to Identify Admin, navigate to **System Setup > Manage Certificates**, and select **Certificate > Import** to upload the public key of your signing certificate. Identify will use this key to verify requests sent from Casewhere.



Configure Casewhere

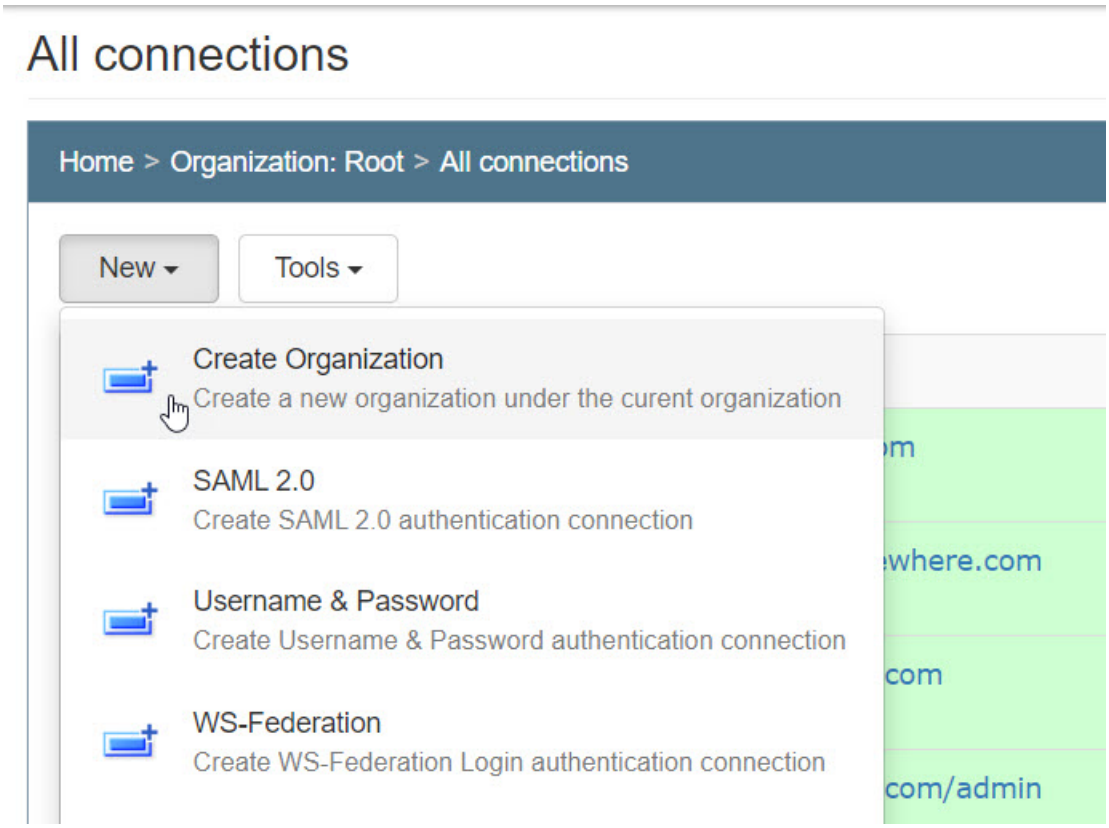
Configure IdP settings in the `web.config` of Casewhere Admin

```
<!-- IdP Connection -->
<add key="idpConnection:SPCertStoreLocation" value="LocalMachine" />
<add key="idpConnection:SPCertStoreName" value="My" />
<add key="idpConnection:IdPCertStoreLocation" value="LocalMachine" />
<add key="idpConnection:IdPCertStoreName" value="TrustedPeople" />
<add key="idpConnection:SPSigningCertificateThumbprint" value="d95c06939cb37f18c31ffcfebe76e8d9e0b040c2" />
<add key="idpConnection:SignatureAlgorithm" value="SHA256" />
<add key="idpConnection:IdPUrl" value="https://login-demo.casewhere.com/runtime/" />
<add key="idpConnection:IdPMetadataFolder" value="~/Metadata/OnPremiseIdP" />
<add key="idpConnection:EntityId" value="" />
<add key="idpConnection:LogoutResponseUri" value="https://job-local.casewhere.com/auth/signoffconsumer" />
```

- **idpConnection:IdPCertStoreLocation** and **idpConnection:IdPCertStoreName**: The location where you install the Identify's signing certificate. The certificate can be downloaded from Identify.
- **idpConnection:SPCertStoreLocation** and **idpConnection:SPCertStoreName**: The location where you install the Casewhere's SP signing certificate.
- **idpConnection:SPSigningCertificateThumbprint**: The private key Casewhere uses to sign requests before sending to Identify.
- **idpConnection:IdPUrl**: URL of the Identify Runtime, e.g., https://login-local.casewhere.com/runtime/.
- **idpConnection:IdPMetadataFolder**: The virtual path to the folder containing the **metadata.xml** provided by the IdP.
- **idpConnection:LogoutResponseUri**: The URI Casewhere uses to handle the logout callback from IdP.
- **idpConnection:EntityId**: The identifier of the Casewhere app as part of the protocol. If not specified, Casewhere will use the application domain name as **EntityId** to communicate with IdP.

Configure Identify connection

Login to Identify Admin and navigate to Connections, select **New**, and a dropdown menu will appear:

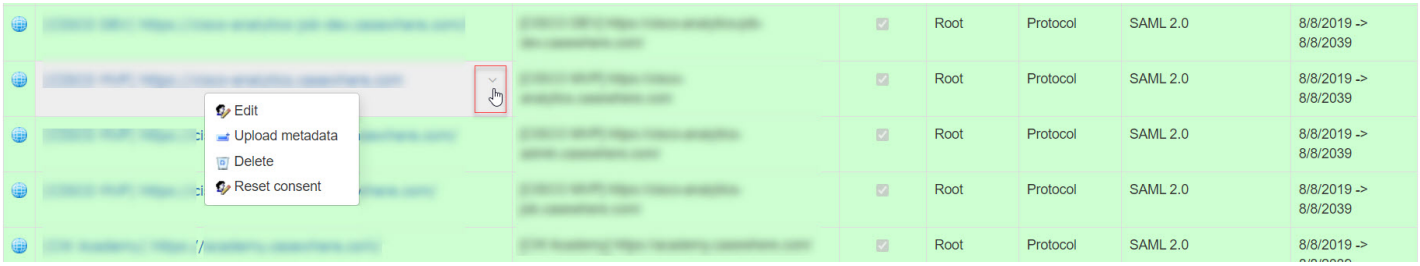


Create a new **SAML 2.0 protocol connection** with the following settings:

- Name: **[service provider name]** e.g. https://admin-local.casewhere.com
- Enabled: **true**
- Authentication connection: e.g., **Username & password**. You can learn how to set up an Identify authentication connection here.

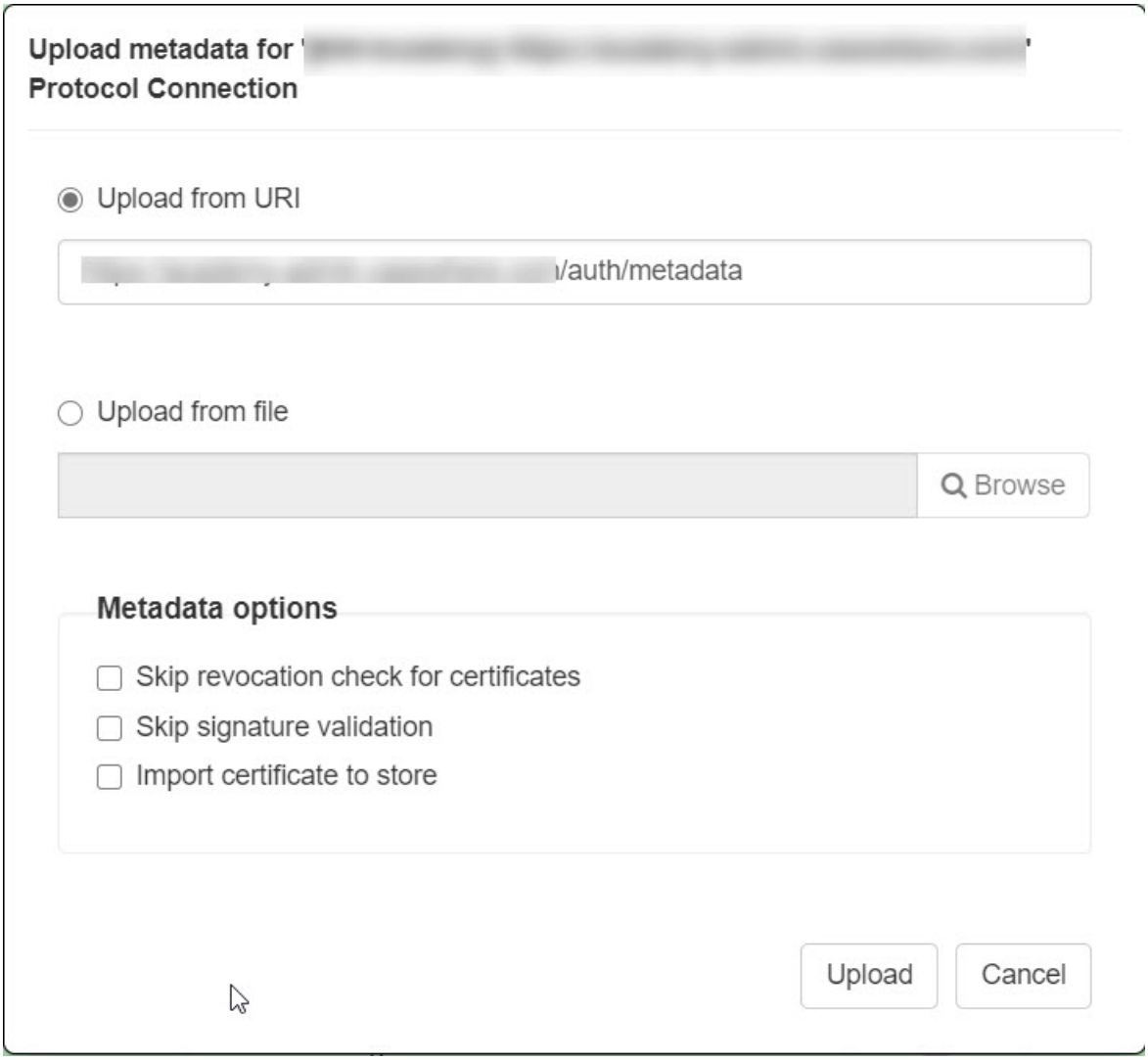
Click **Save and Close**; a connection will be created.

In the connection list, navigate to the connection you just created and select **Upload metadata.**



The upload metadata dialog will appear. Here, you have two options:

- Enter the metadata URI **[Casewhere Admin domain]/auth/metadata**, e.g., https://admin-local.casewhere.com/auth/metadata.
- Upload the file from your machine.



You can skip the revocation check for certificates and signature validation for deployments where self-signed certificates are utilized.

If the configuration is done correctly, when you visit the Casewhere Admin site, you will be directed to Identify where you can enter your login credentials.

Generate self-signed certificates

Creating a self-signed certificate for digital signature purposes using OpenSSL involves a few steps. Below is a step-by-step guide to generate a self-signed certificate:

Step 1: Install OpenSSL

Windows

```
https://www.firedaemon.com/download-firedaemon-openssl
```

Ubuntu

```
sudo apt-get install openssl
```

Step 2: Generate a private key

Generate a private key that will be used to create the certificate. The key length of 2048 bits is commonly used for security.

```
openssl genrsa -out private_key.pem 2048
```

Step 3: Create a configuration file with EKU for `clientAuth` and `serverAuth`

Create an OpenSSL configuration file (e.g., `openssl.cnf`) that includes the necessary `Extended Key Usage` extensions for both client and server authentication. This file will specify how the certificate should be generated.

Create a file named `openssl.cnf` with the following content:

```
[ req ]
default_bits       = 2048
default_keyfile    = CasewhereSigningCertificate_private_key.pem
distinguished_name = req_distinguished_name
x509_extensions    = v3_req
prompt             = no

[ req_distinguished_name ]
C  = DK
ST = Capital
L  = Virum
O  = Globeteam
OU = Casewhere
CN = CasewhereSigningCertificate

[ v3_req ]
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth, clientAuth
```

Update the `C`, `ST`, `L`, `O`, `OU`, and `CN` fields under `[ req_distinguished_name ]` to match your requirements.

Step 4: Generate a certificate signing request (CSR)

Use the private key and configuration file to create a CSR.

```
openssl req -new -key private_key.pem -out csr.pem -config openssl.cnf
```

Step 5: Generate the self-signed certificate

Finally, create a self-signed certificate that includes the specified EKU extensions for both `clientAuth` and `serverAuth`.

```
openssl x509 -req -days 365 -in csr.pem -signkey private_key.pem -out self_signed_cert.pem -extensions v3_req
-extfile openssl.cnf
```

Step 6: Convert the certificate to PFX

```
openssl pkcs12 -export -out self_signed_certificate.pfx -inkey private_key.pem -in self_signed_cert.pem
```

Summary of files created

- `private_key.pem`: Your private key.
- `csr.pem`: Certificate Signing Request.
- `self_signed_cert.pem`: Your self-signed certificate with `clientAuth` and `serverAuth` capabilities.
- `self_signed_certificate.pfx`: The certificate in PFX format.

Now, you have a self-signed certificate that is valid for both client and server authentication and suitable for use in applications requiring mutual TLS (mTLS) or secure communications. Be sure to keep your private key (`private_key.pem`) secure.

These fundamental articles will help you:

- Explore and understand the primary concepts
- Able to use Casewhere to create efficient and sophisticated applications for enterprises

Product overview

Casewhere is a case management, process standardization, and workflow automation system. Casewhere makes it possible to create seamlessly integrated processes that tie together applications, internal and external actors, and data stores. With its extremely open architecture, the user can easily and quickly build up a solution for their organization where all the business systems work and share data. To do this, Casewhere provides a variety of building blocks (products) that you can combine into a solution that serves your needs.

We understand that every organization has its own unique business process. Thanks to the open architecture of Casewhere, we can always customize and modify the products to make them best suit your solution needs. In Casewhere, we have categorized the offered products into the following main types:

- Plugin: .NET libraries developed on top of the Casewhere plugin framework to provide custom functionalities of Casewhere.
- Template: A reusable piece of code, script, or configuration to help you quickly implement a functionality with a minimum customization effort.
- Theme: A package to customize the appearance of the worker sites, including layout, color and typographies.
- Widget: Plug-and-play UI components that display information or provide a specific way for a user to interact with the system.
- Component: A set of Casewhere resources, processes, workflows, data classes, plugins, and widgets, bundled together to help configurator/developers handle specific tasks.
- Application: A set of Casewhere resources, processes, workflow, data classes, even other products, bundled together to help end users to archive some business goals.
- Integration: Applications or components designed for connecting one system to another.

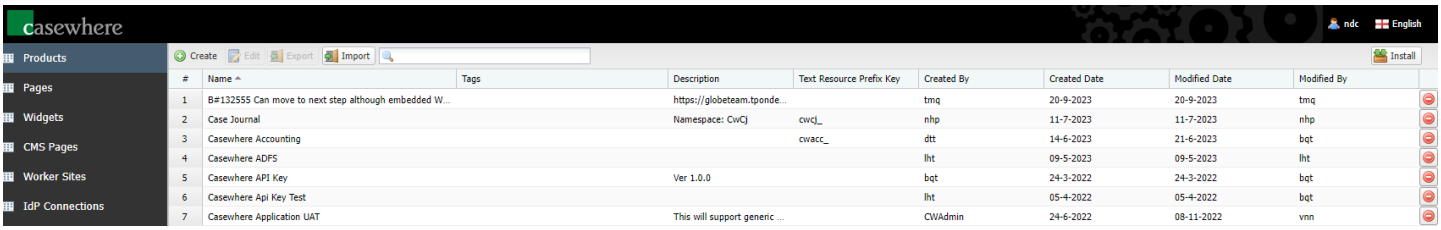
Let us take an example of using products in Casewhere. Email is the most common function in any solution, and in Casewhere you can easily add a product that will manage the whole life cycle of the emails, from creation, queuing to sending using different strategies:

- Define email templates
- Setup queued, scheduled, or manual email sending
- SMTP configuration
- Email monitoring

In conclusion, by using diverse types of products the user can quickly and easily build up a solution in a short amount of time with flexibility and scalability.

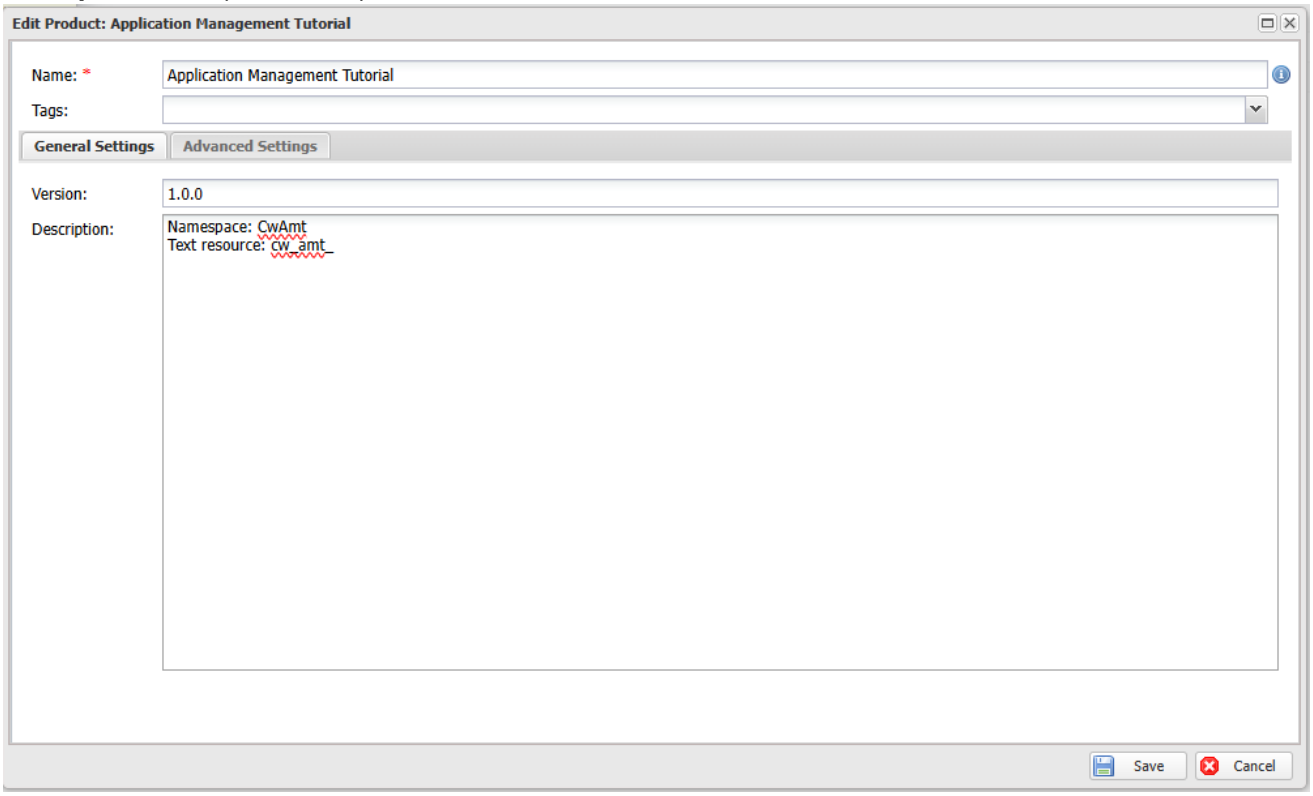
Product management

In Casewhere, a product serves as the top-level resource of a solution, providing a centralized location to contain all solution artifacts. Products are managed through the **Products** function, accessible from the **Products** menu on the left panel of the Casewhere Admin. This page allows users to create, edit, search, import, and export products.

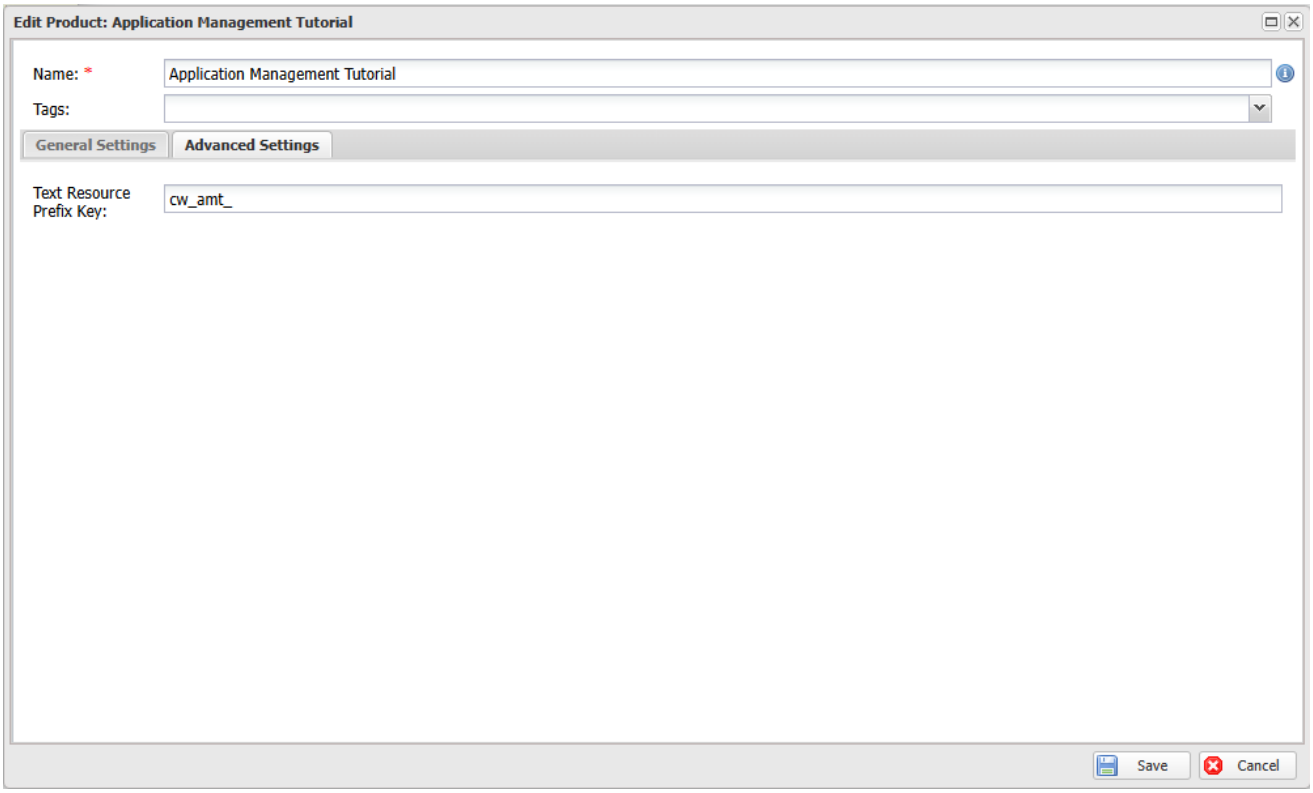


Create a product

To create a new product, click the **Create** button on the toolbar and enter the following information:

- **Name**: Name of the product.
- **Tags**: Tags of the product.
- **General Settings**
  - **Version**: Version of the product.
  - **Description**: Description of the product.
  - 

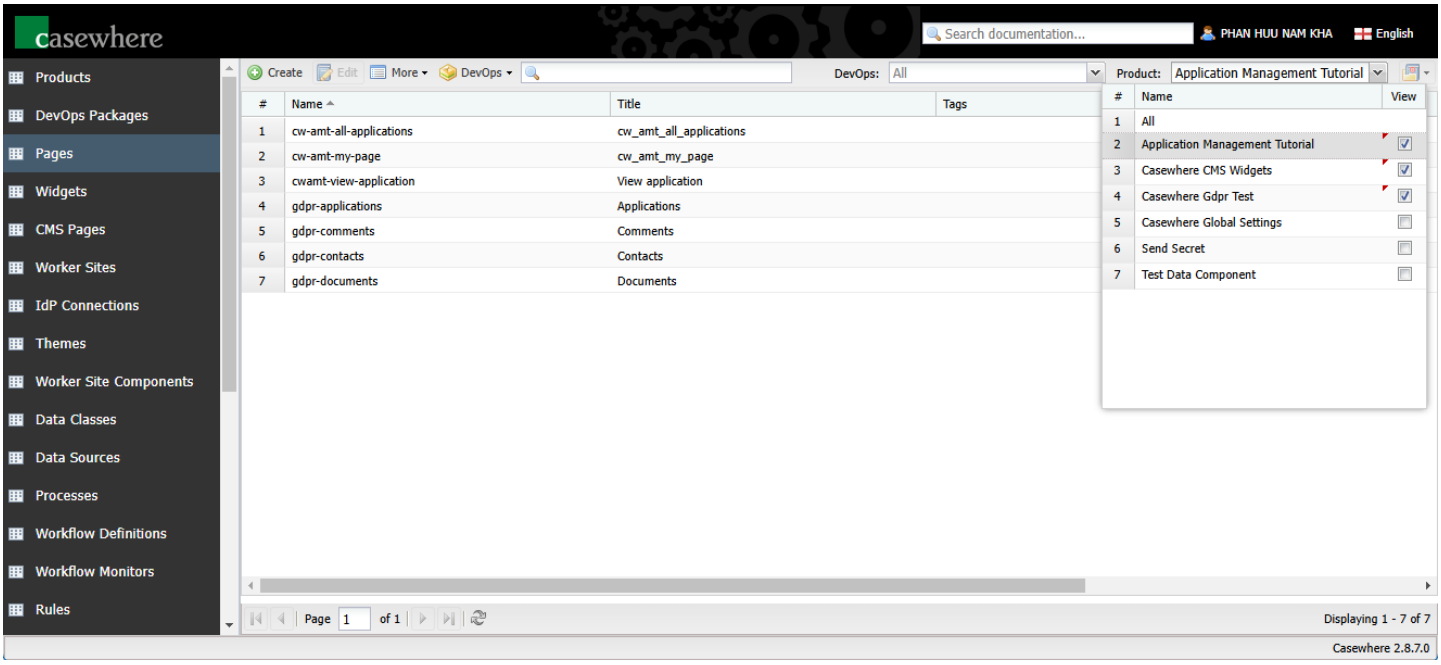
- **Advanced Settings**
  - **Text Resource Prefix Key**: The prefix will be appended to all text resources associated with this product.



Working with products

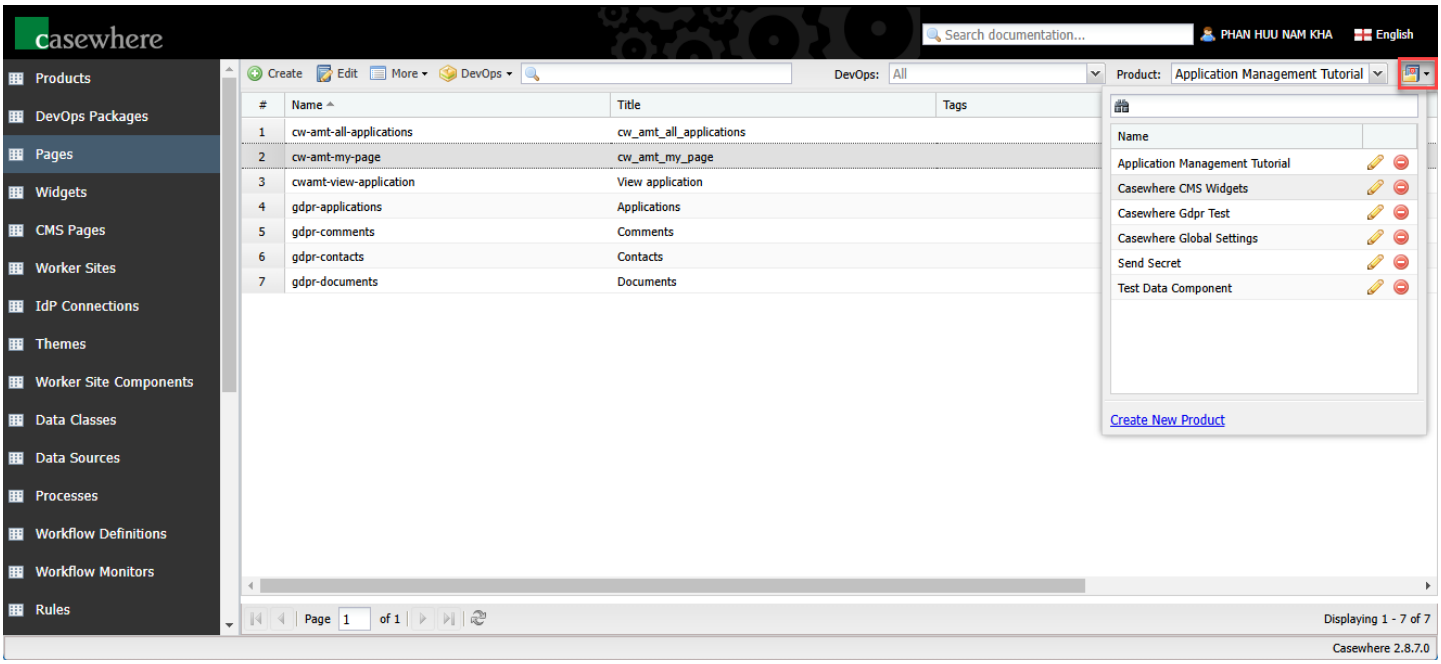
Organizing resources and artifacts using **Products** simplifies management, packaging, and deployment. Therefore, when creating solution resources, it is important to specify the product to which those resources belong.

To specify a product for a resource, select the product name from the **Product** combo box in the top-right corner. Once a product is selected, all newly created resources will automatically be associated with that product. Note that you can choose to **View** multiple products simultaneously, but only one product can be set as the active product in your workspace.



You can also manually add a resource to a product if a product was not previously selected. To do this:

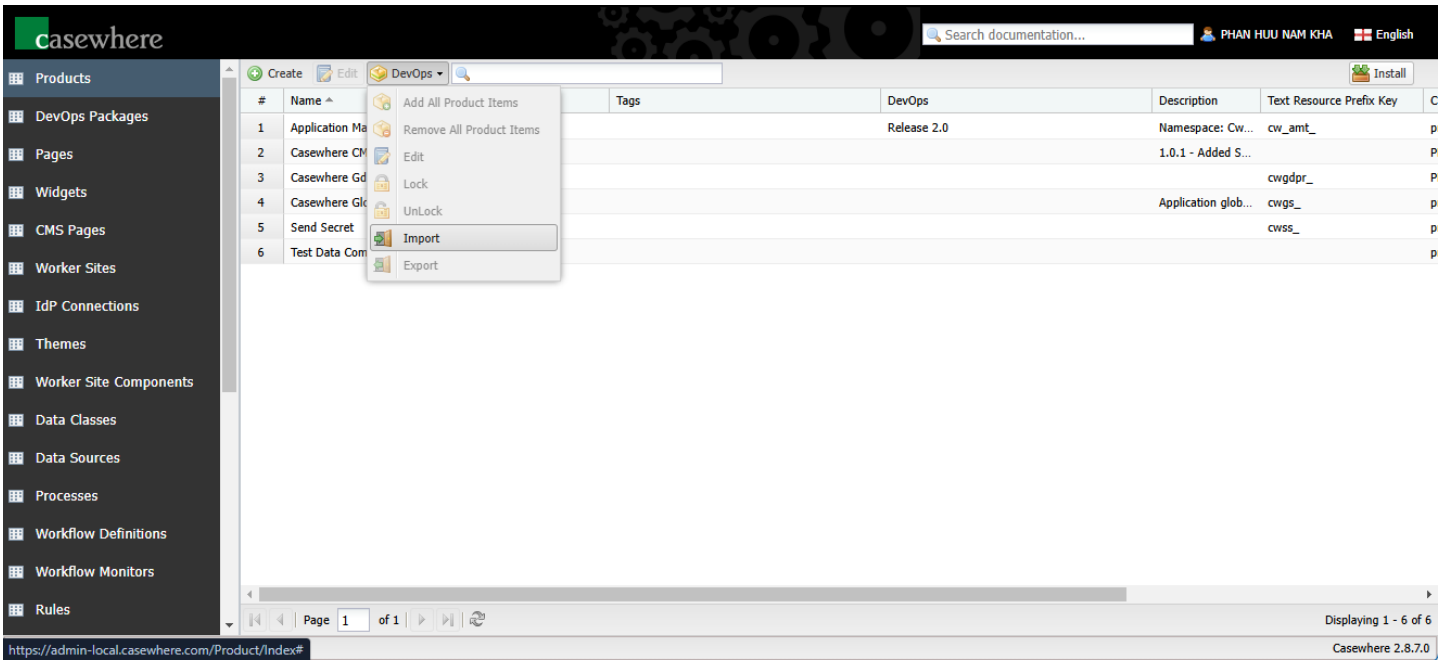
- Select the resource.
- Click the **Move to product** icon .
- Choose the product name from the list or use the search function to find the product.



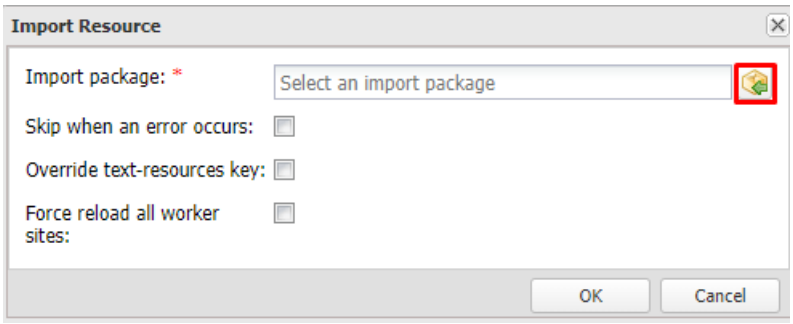
Import and export products

In the **Products** page, do the following steps to import a product:

- Use the menu **DevOps / Import**.



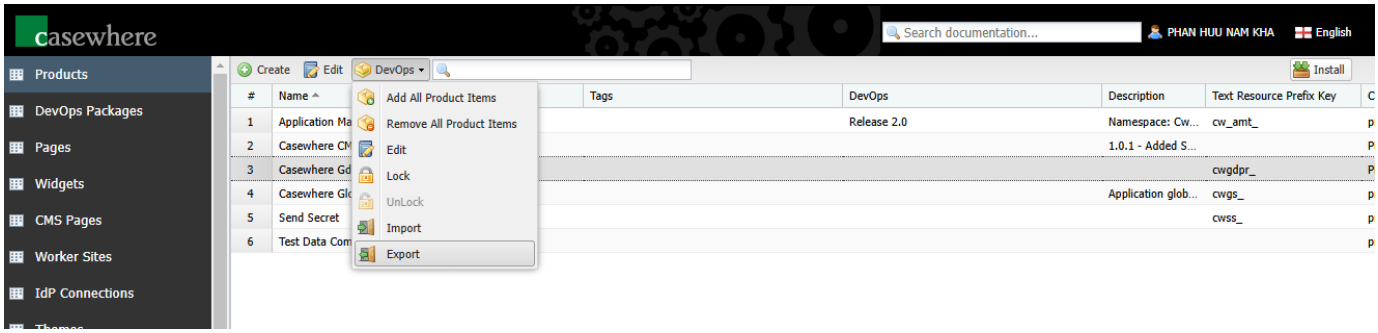
- Select the product package to import. There are some options:
    - **Skip when an error occurs**: By default, Casewhere will stop the import process when having an error while resolving the package. If this option is enabled, Casewhere will bypass the error package and continue the import process.
    - **Override text-resources key**: If this option is enabled, when importing a product, if there is any text-resource existing with the same key, it will be replaced by the new one in the importing product.
    - **Force reload all worker sites**: If this option is enabled, after importing a product, Casewhere will increase the version of worker sites, preventing resource caching



- Click **OK**.

To export a product:

- Select the product to be exported.
- Use the menu **DevOps / Export**.

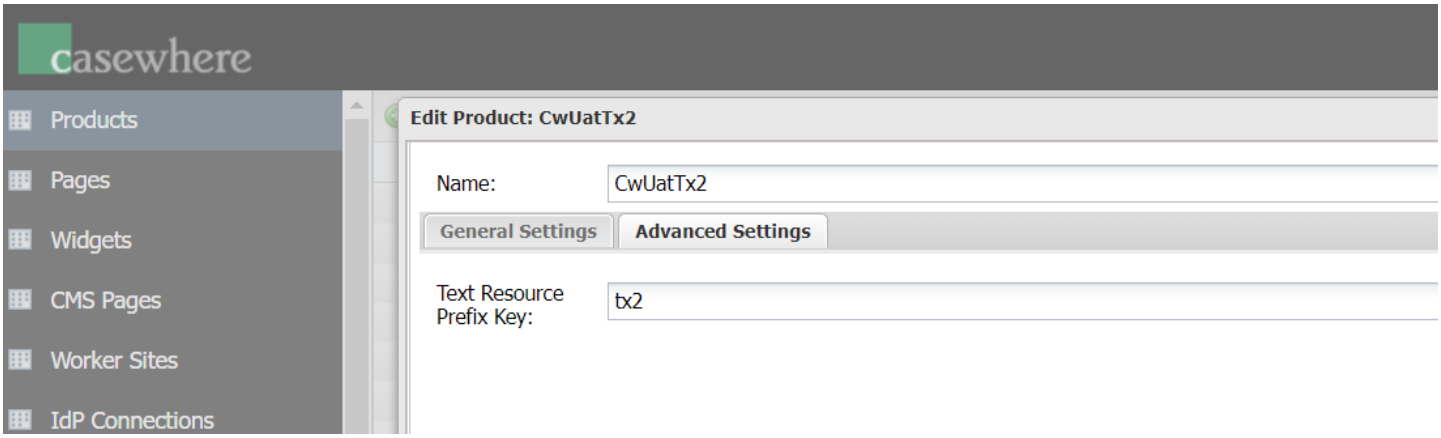


- The product will be exported as a zip file and downloaded to the local machine.

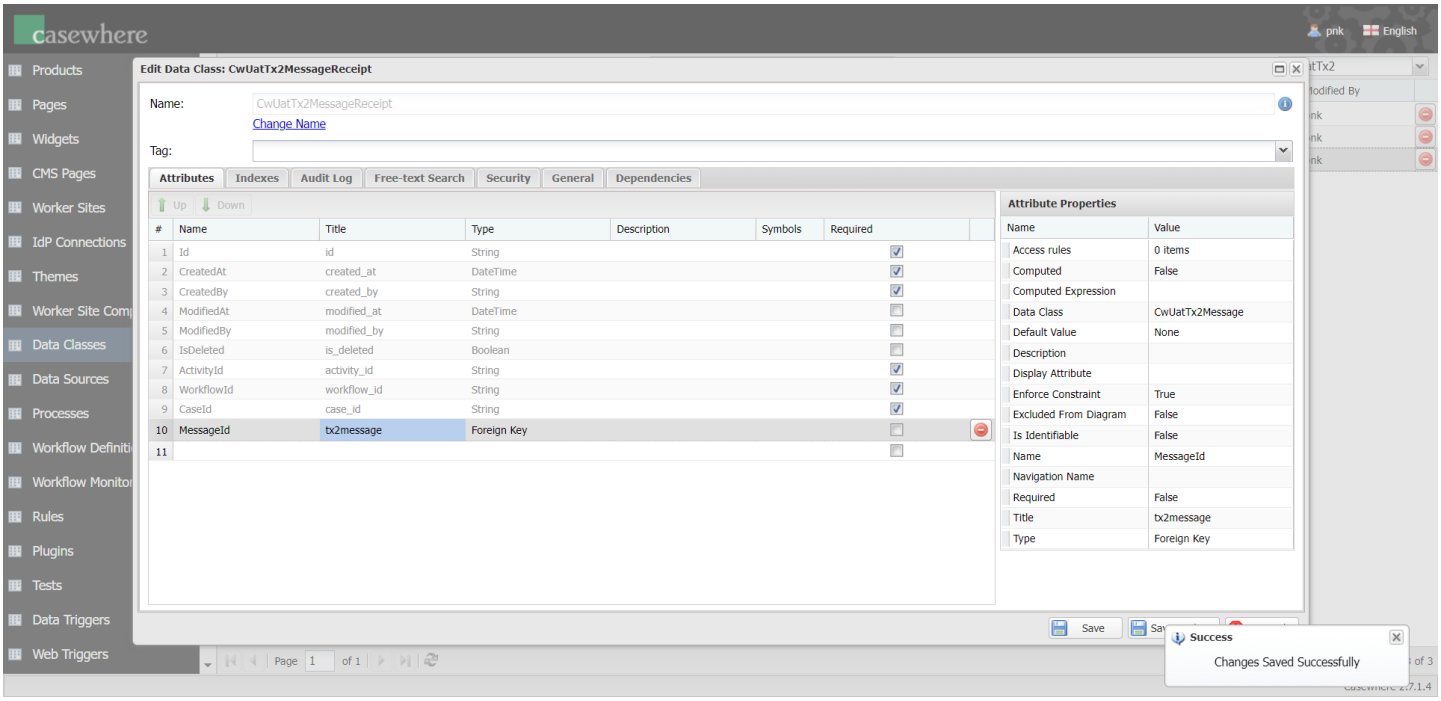
Text resource namespace

> This feature is available in version `2.7.2` or later

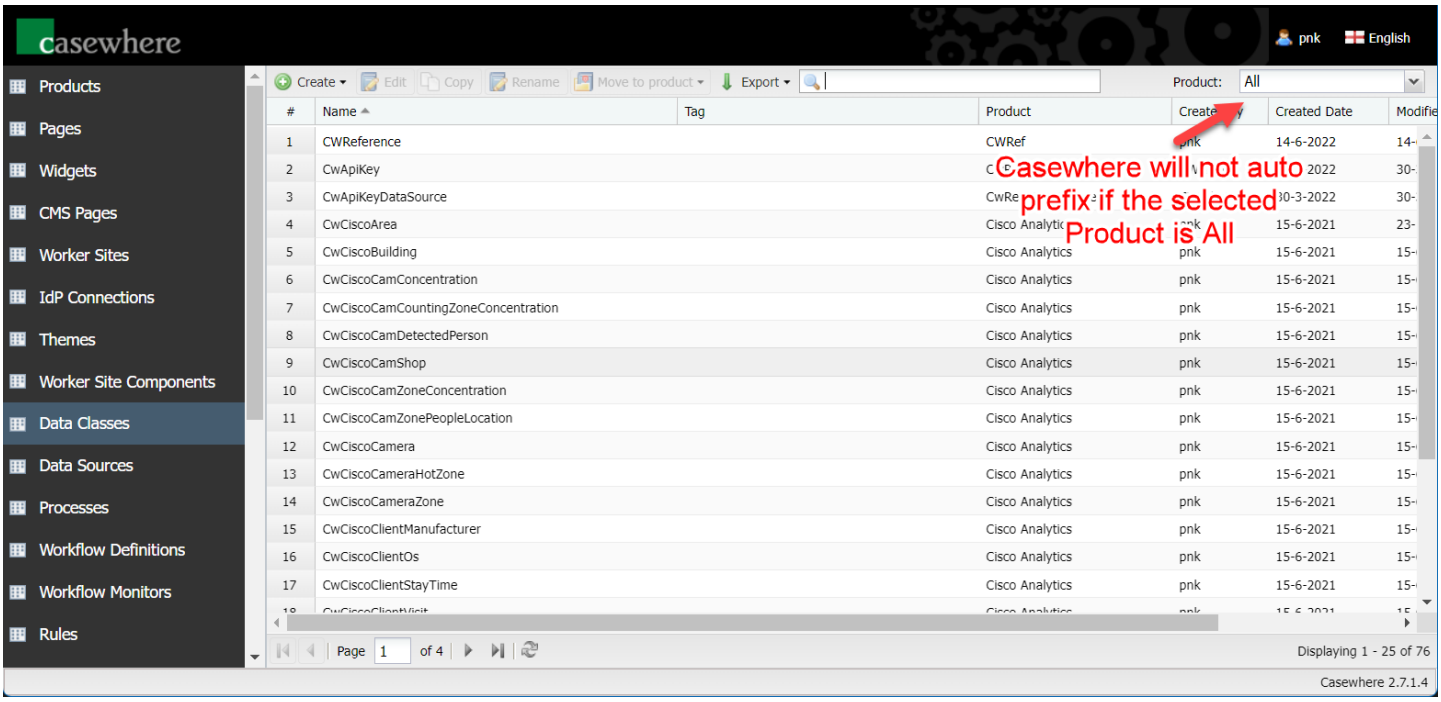
Managing text resources using namespace is a good practice to avoid conflict among different Casewhere products. It's possible to configure the text resource namespace for a Casewhere product.



When you create a text resource in a product context, Casewhere will auto prefix the text resource key with the configured namespace.



Please note that this feature will not work if you're not in a product context because Casewhere does not know which namespace it should used to create the text resource.


casewhere
pnk
English
Products
Pages
Widgets
CMS Pages
Worker Sites
IdP Connections
Themes
Worker Site Components
Data Classes
Data Sources
Processes
Workflow Definitions
Workflow Monitors
Rules
Create
Edit
Copy
Rename
Move to product
Export
Product:
All
Name Tag Product Created Date
1 CWReference CWRef 14-6-2022
2 CwApiKey 2022
3 CwApiKeyDataSource 30-3-2022
4 CwCiscoArea Cisco Analytics 15-6-2021
5 CwCiscoBuilding Cisco Analytics pnk 15-6-2021
6 CwCiscoCamConcentration Cisco Analytics pnk 15-6-2021
7 CwCiscoCamCountingZoneConcentration Cisco Analytics pnk 15-6-2021
8 CwCiscoCamDetectedPerson Cisco Analytics pnk 15-6-2021
9 CwCiscoCamShop Cisco Analytics pnk 15-6-2021
10 CwCiscoCamZoneConcentration Cisco Analytics pnk 15-6-2021
11 CwCiscoCamZonePeopleLocation Cisco Analytics pnk 15-6-2021
12 CwCiscoCamera Cisco Analytics pnk 15-6-2021
13 CwCiscoCameraHotZone Cisco Analytics pnk 15-6-2021
14 CwCiscoCameraZone Cisco Analytics pnk 15-6-2021
15 CwCiscoClientManufacturer Cisco Analytics pnk 15-6-2021
16 CwCiscoClientOs Cisco Analytics pnk 15-6-2021
17 CwCiscoClientStayTime Cisco Analytics pnk 15-6-2021
Casewhere will not auto prefix if the selected Product is All
Page 1 of 4
Displaying 1 - 25 of 76
Casewhere 2.7.1.4

Worker site overview

A worker site is a site that your end users are presented to. A Casewhere solution can have as many worker sites as needed. Each worker site has a unique URL, a different theme, and separate security policies.

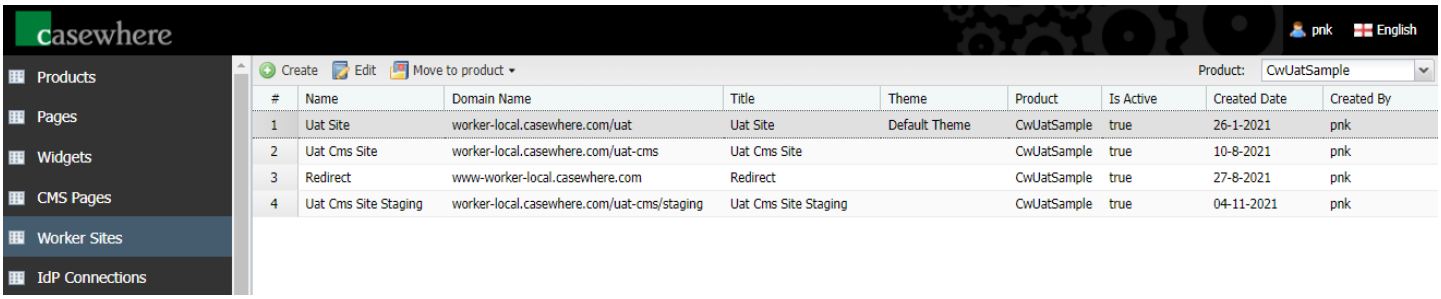
Let's say you're developing a book shop solution. While you can totally manage all business processes using a worker site, it's better to separate them into different worker sites to target different audiences. A typical worker site breakdown can be as below:

- Frontend site: for shopping users
    - URL: book-store.casewhere.com
    - Theme: eCommerce, responsive is needed
    - Security:
        - Allow anonymous visitors
        - Social login, e.g., Facebook, Google, Twitter
- Backend site: for the staff users to manage books, orders, etc.
    - URL: book-store.casewhere.com/admin
    - Theme: eAdmin, no need for responsive
    - Security:
        - Username and password
- Configuration site: for the store manager or IT to configure the solution such as changing logo, translating text resources, adjusting business rules
    - URL: book-store.casewhere.com/config
        - Theme: eAdmin, no need for responsive
        - Security:
        - Username and password
        - IP whitelisted

Organizing worker sites is a critical step when designing an enterprise solution. Doing it properly and at the early phase will make your solution securer while providing the best experience for the end-users. Furthermore, you can put each worker site in a different product so you can have multiple teams develop, deploy and maintain them separately, thus making the overall development process more effective.

Worker site management

In Casewhere Admin, navigate to **Worker Sites** to manage the sites for your solution. From here, you can create, edit and manage worker sites like any product resource.



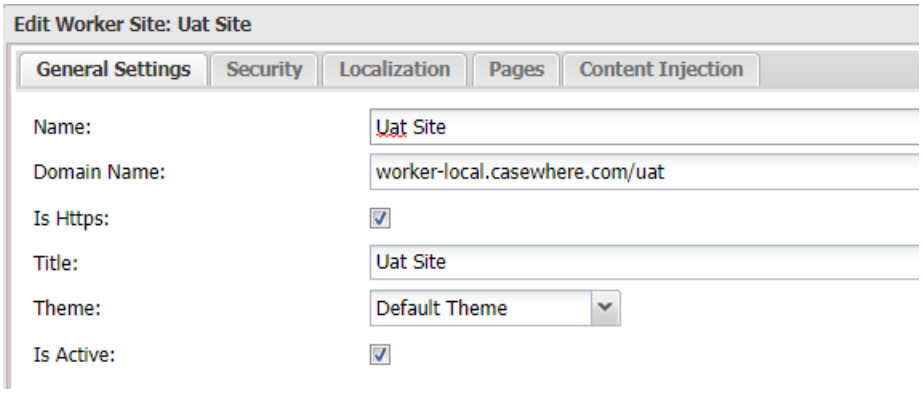
Worker site editor

The worker site editor will give you all the necessary flexibilities to customize your worker sites.

General settings

These are the information required to set up a worker site.

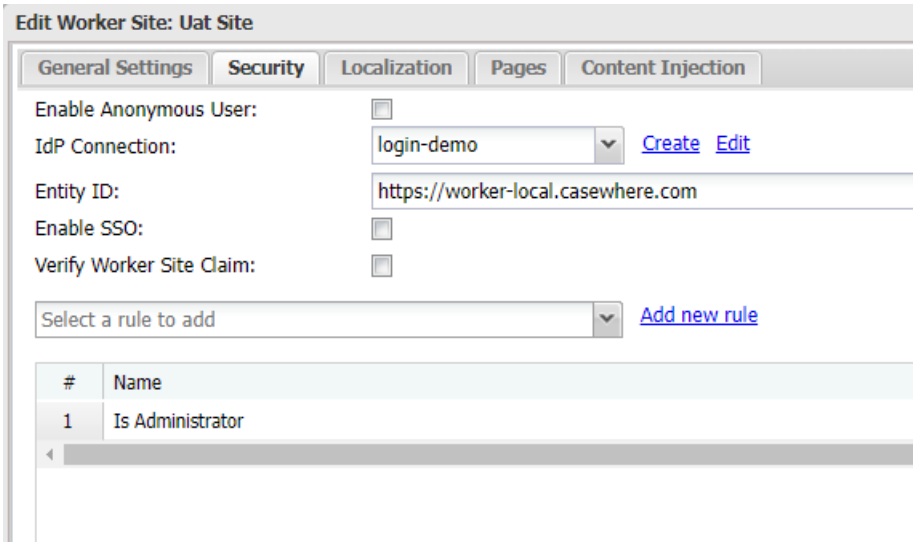
NAME	DESCRIPTION
Name	The name to identify the worker site. The worker site name must be unique.
Domain Name (changed to Uri since v2.7)	The Uri of the worker site. Users will access the worker site via this Uri. See Set up worker site domain to learn more. Uri could be a subdomain (example.casewhere.com) or a path (www.casewhere.com/example).
Is Https	The protocol used for the worker site, i.e., Http or Https.
Title	The title of the worker site. This title will appear on the browser header.
Theme	The theme governs the worker site's looks and feels.
Is Active	If a worker site is inactive, users cannot access it.



Security

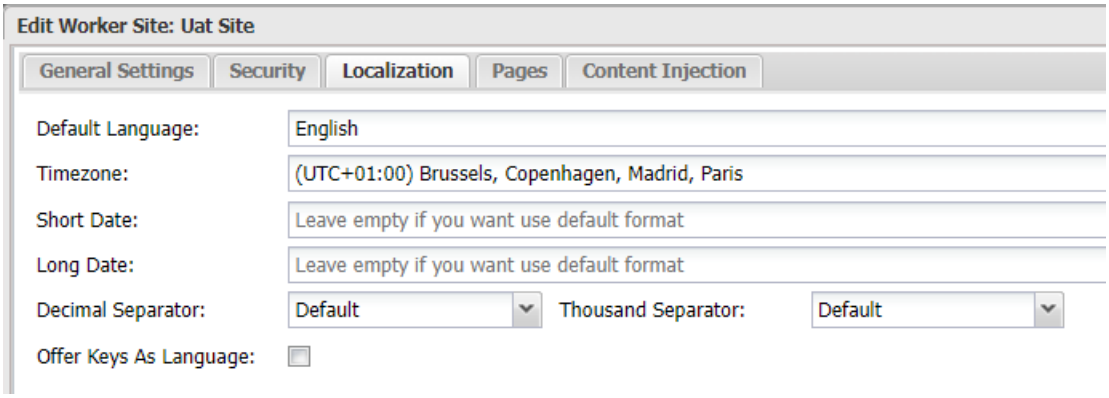
A set of settings and rules to determine how users are authorized when accessing the worker site.

NAME	DESCRIPTION
Enable Anonymous User	When enabled, users can access the worker site without login.
IdP Connection	The identity provider for user authentication. See Worker site authentication to learn more.



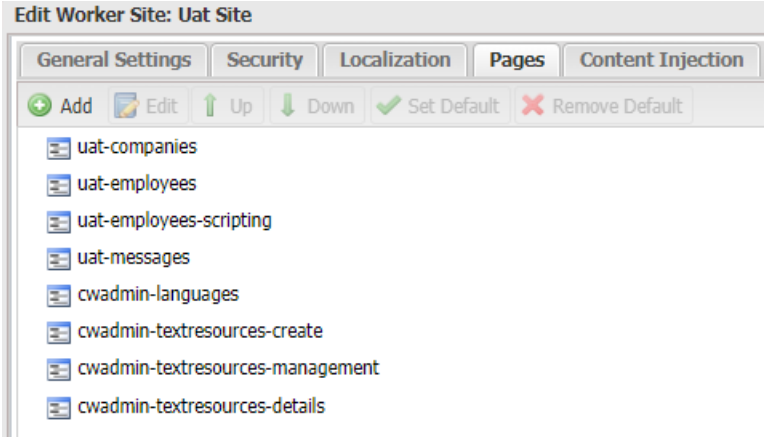
Localization

The settings to customize your worker site to adapt to users' culture, from languages to formats of numbers, date time. See Worker site localization to learn more.



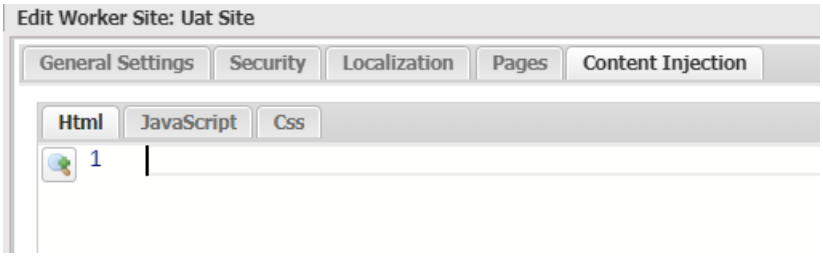
Pages

Where you manage the pages of your worker sites. A page can be linked to multiple worker sites. It is possible to configure the default page for your worker site here.



Content injection

This is where you embed scripts (Html, Javascript, Css) to implement advanced customization for your worker sites.



Worker site authentication

Introduction

The Security Assertion Markup Language (SAML) protocol is an open-standard, XML-based framework for authentication and authorization between two entities without a password:

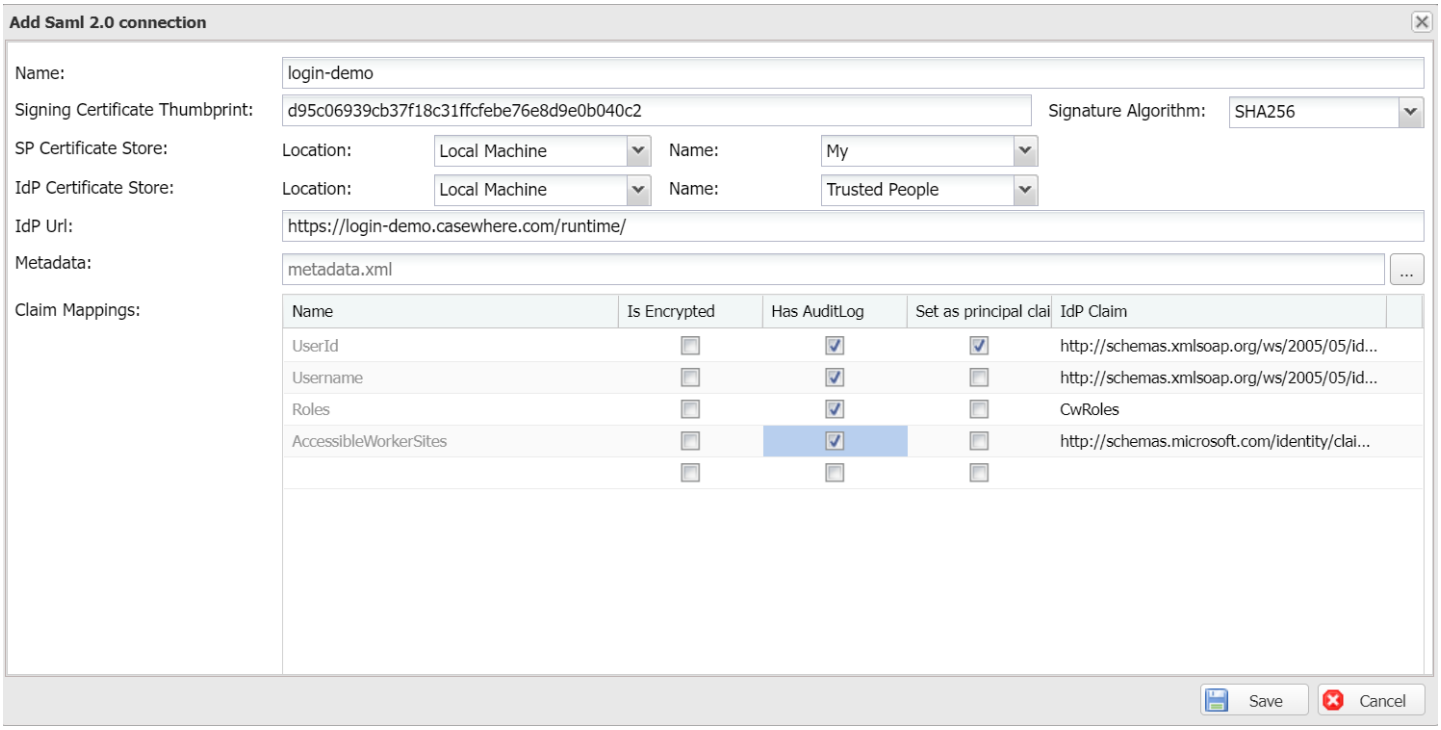
- Service provider (SP) agrees to trust the identity provider to authenticate users
- Identity provider (IdP) authenticates users and provides to service providers an authentication assertion that indicates a user has been authenticated

Casewhere is a service provider using SAML2.0 protocol to integrate with other IdP such as Safewhere, Azure Active Directory, and NemLog-in.

Configure Casewhere authentication with Safewhere Identify

Create IdP connection

- Login to Casewhere Admin
- Navigate to **IdP Connections**
- Create a new Idp Connection with following settings:
  - **Name**: The name of the connection. Example: login-demo
  - **Signing certificate**: The certificate of the connection. Example: d95c06939cb37f18c31ffcfebe76e8d9e0b040c2
  - **Signing Alg**: The signing algorithm. Example: SHA256
  - **SP certificate store**: The location of the certificate. Example: Local Machine/My
  - **IdP Certificate store**: The location of IdP certificate. Example: Local Machine/Trusted People
  - **IdP Url**: The authentication endpoint. Example: https://login-demo.casewhere.com/runtime/
  - **Metadata**: The metadata file from the IdP. For Safewhere Identify, you can download the metadata at https://login-demo.casewhere.com/runtime/saml2/metadata.idp



Configure authentication for worker site

Navigate to **Worker Sites**, you can create a new worker site or select an existing to edit. Below are the required settings:

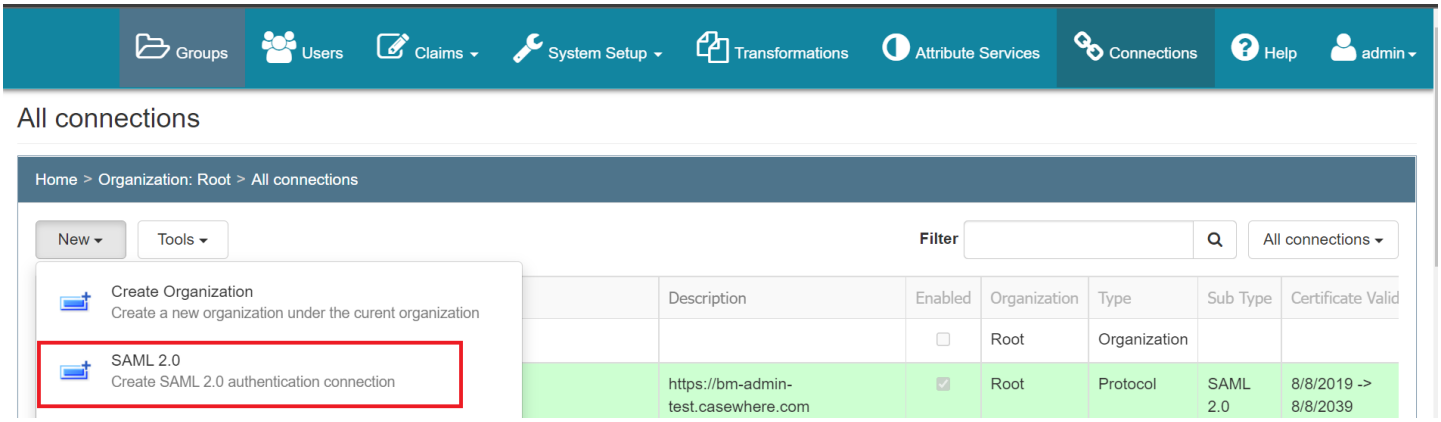
- **Name**: Name of the worker site. Worker site name must be unique

- **Domain Name** (changed to **URI** since version 2.7): Specify the URI for your worker site, without the protocols (https://). Example: worker-local.casewhere.com
- **Title**: The title that appears on the browser window
- **Theme**: Select a theme for your worker site
- **IdP Connection**: Select the connection you created in the previous step

Create connection in Safewhere Identify

Login to Identify Admin and navigate to **Connections**.

Create a new **SAML 2.0** protocol connection with following settings:

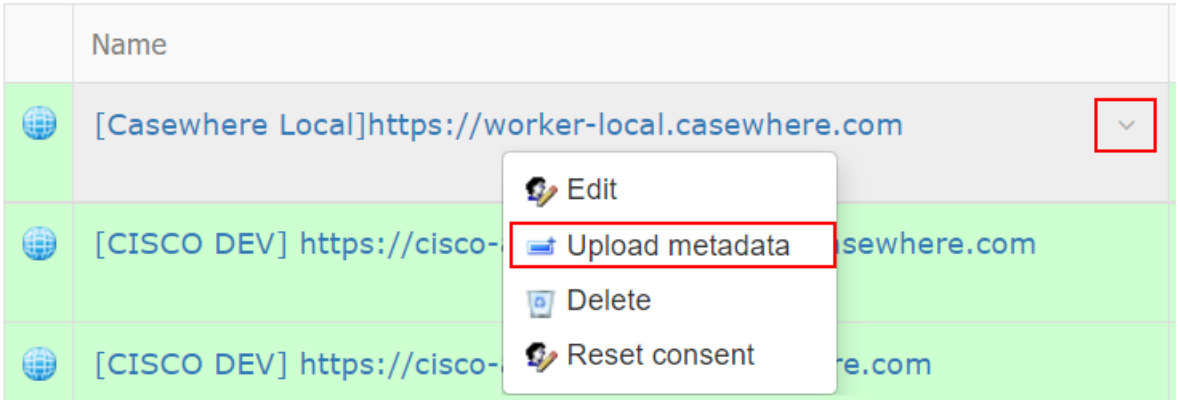


- **Name**: The connection name must be unique. Example: [local]https://worker-local.casewhere.com
- **Enabled**: We must have it checked for the connection to be used
- **Connection Dependency**: Select the login methods you want to enable for this connection

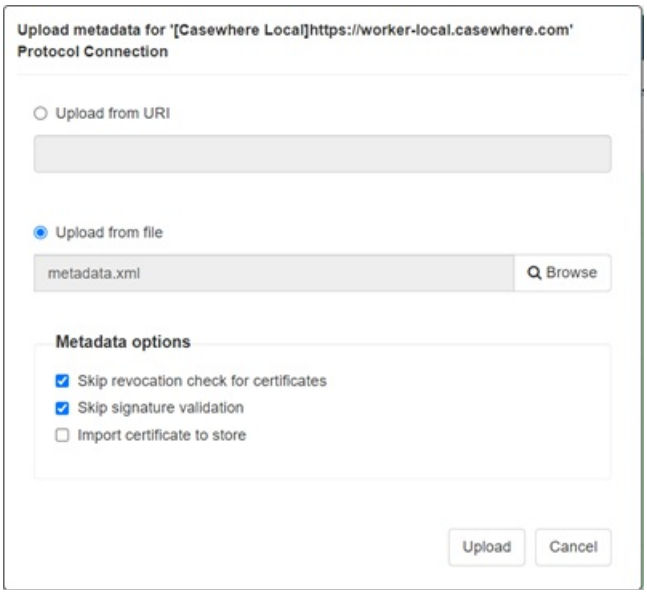
Click **Save & Close**.

Download the worker site metadata from Casewhere using this link format **[Worker Api URI]/metadata?worker=[your worker site URI]**. For example https://api-local.casewhere.com/metadata?worker=worker-local.casewhere.com.

Navigate to the connection that you just created, from the inline menu and select **Upload metadata**:



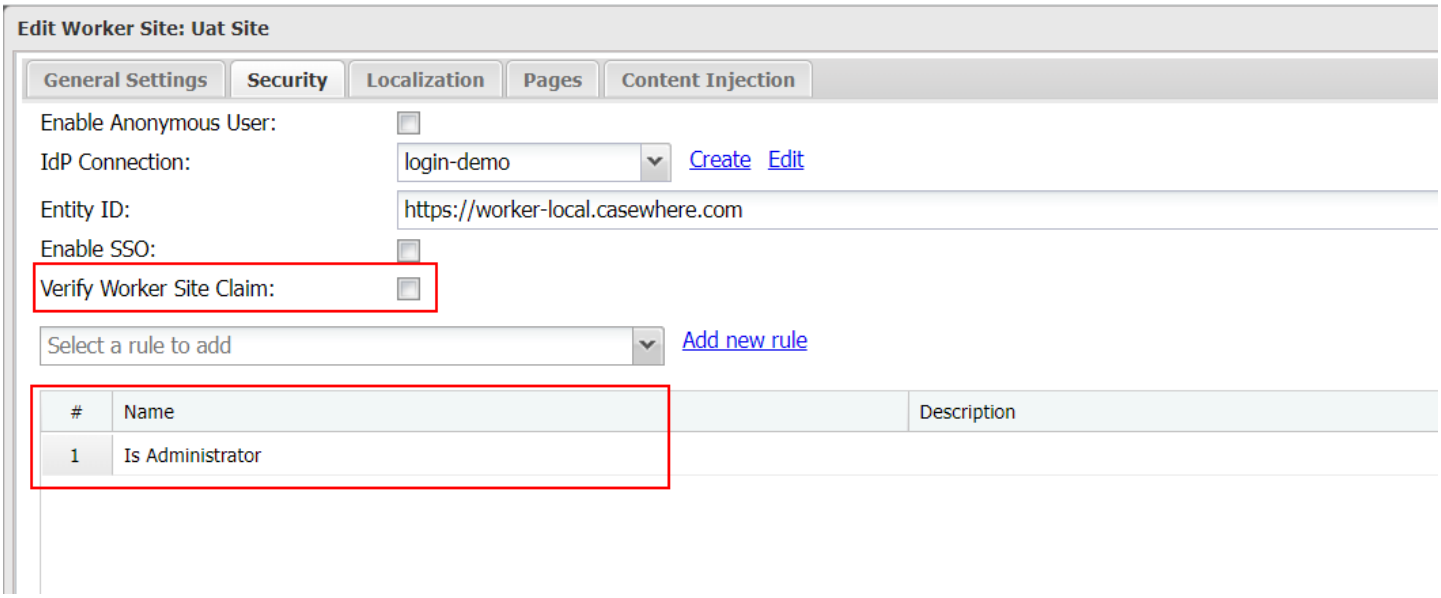
For development environments, you can skip **revocation check for certificates** and **signature validation**. It's not recommended to do so for production environments.



For Casewhere 2.5 or earlier, Casewhere controls access to worker site using a custom claim configured from Identify, so you need to do one extra step:

- In Identify Admin, navigate to **Claims**
- Add a **discrete claim**
    - **Claim type**: http://schemas.microsoft.com/identity/claims/tenantdomain
    - **Friendly name**: CW domains
    - **Options / Add new option**: enter the worker site URI. For example https://worker-local.casewhere.com/
- In Identify Admin, navigate to **Transformations** and add a new transformation rule for your **Connection**
    - Navigate to the discrete claim you just added, e.g., CW domains, and select the appropriate values
    - Make you select appropriate **Conditions** for this new transformation rule

For Casewhere 2.6 and later, you can use access rules to control access for your worker site and ignore this extra step. This can be done easily by turning off the setting **Verify Worker Site Claim** in Casewhere Admin. After that, you will need to assign relevant access rules to protect your worker site.



If the configuration is done correctly, when you visit the Casewhere worker site, you will be directed to Identify where you can enter your login credentials.

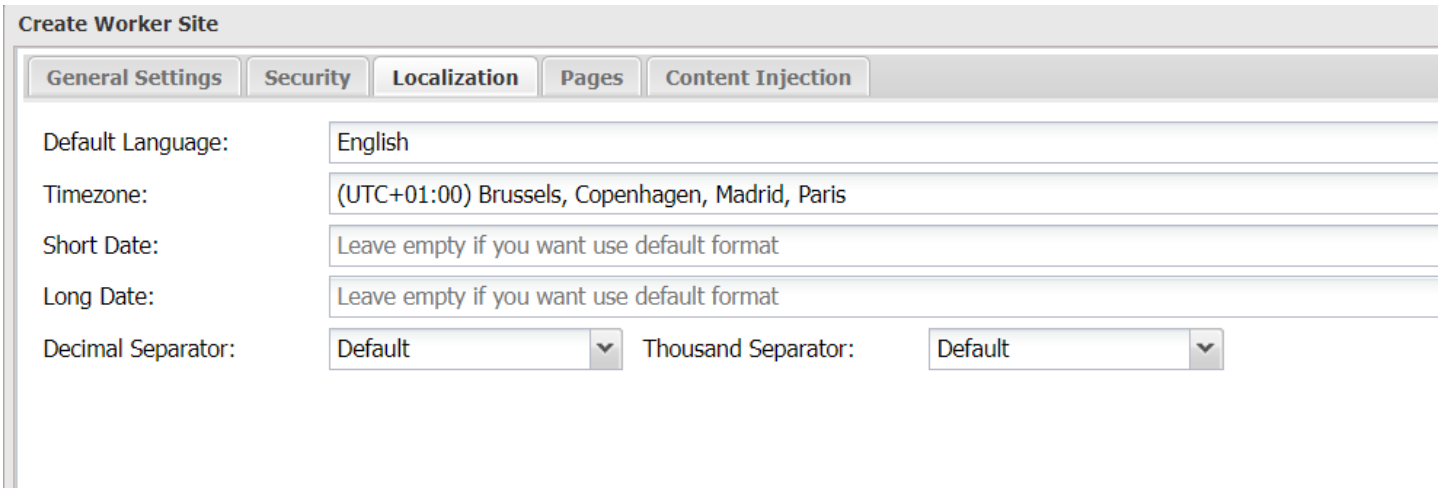
Worker site localization

In Casewhere, localization are the built-in settings and can be setup easily within some simple steps.

Configure Localization settings for worker site

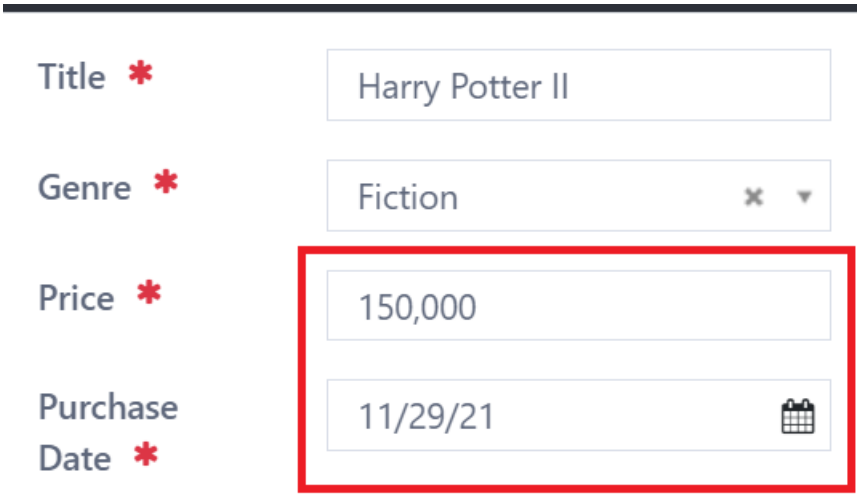
To configure localization for your worker site:

- From Casewhere Admin, navigate to **Worker Sites**.
- Select your worker site and click **Edit**.
- Navigate to **Localization** tab.

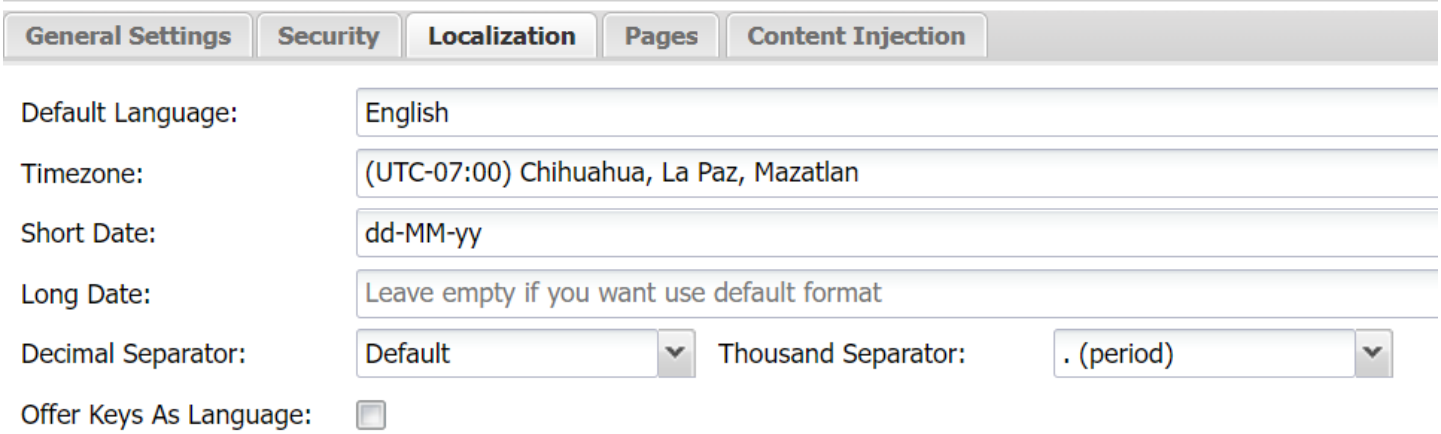


NAME	DESCRIPTION
Default Language	The language to be displayed by default
Timezone	Time zone to display time value
Short Date	Format of short date. Example: dd-MM-yy
Long Date	Format of long date. Example: dd-MM-yyyy hh:mm
Decimal Separator	Decimal separator. Example: 1,4 or 1.4
Thousand Separator	Thousand separator. Example:1.200 or 1,200

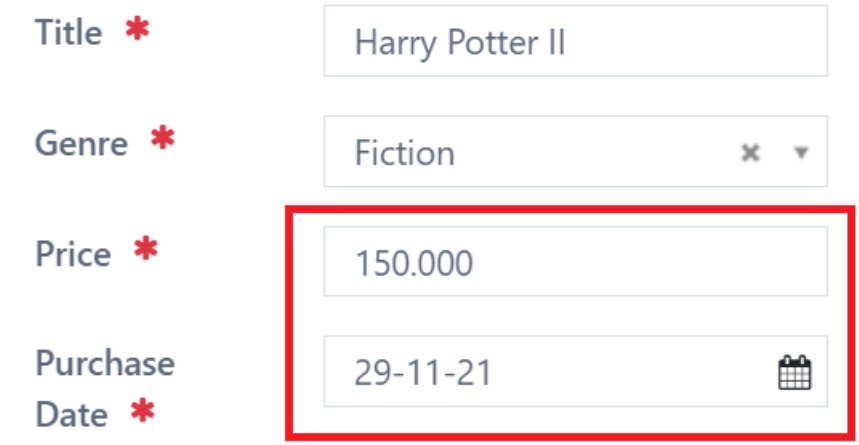
The below example illustrates how these settings works.



Set **Short Date** format to "dd-MM-yy" and **Thousand Separator** to ".(period)".



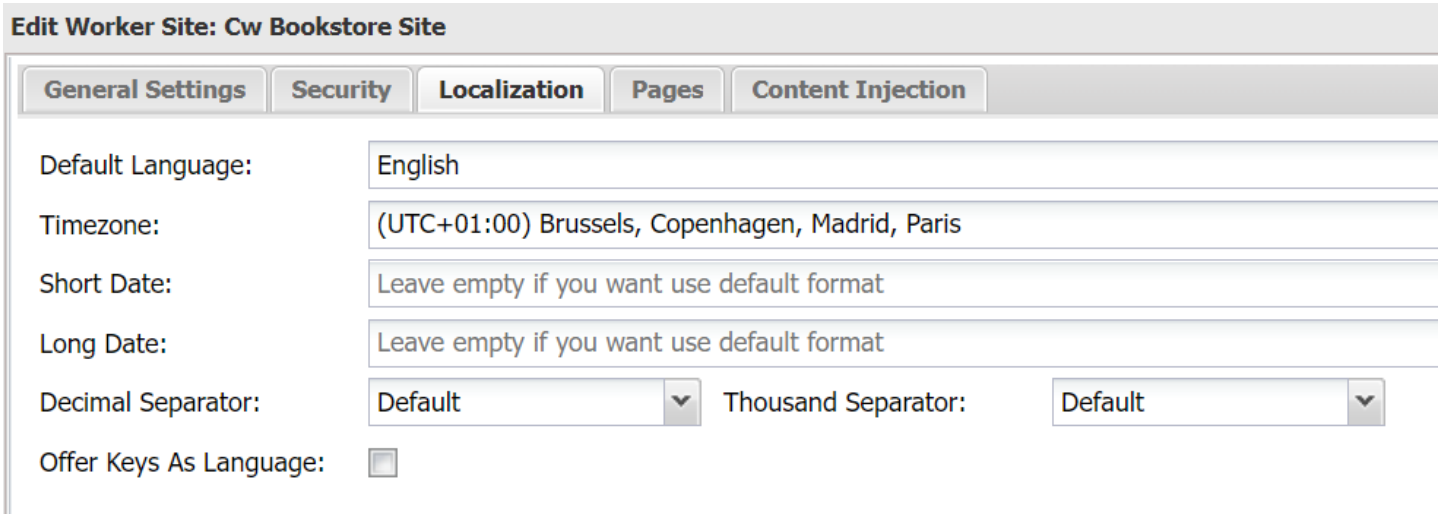
The date will be displayed in "dd-MM-yy" format and thousand separator will be displayed as ".", as we have set.



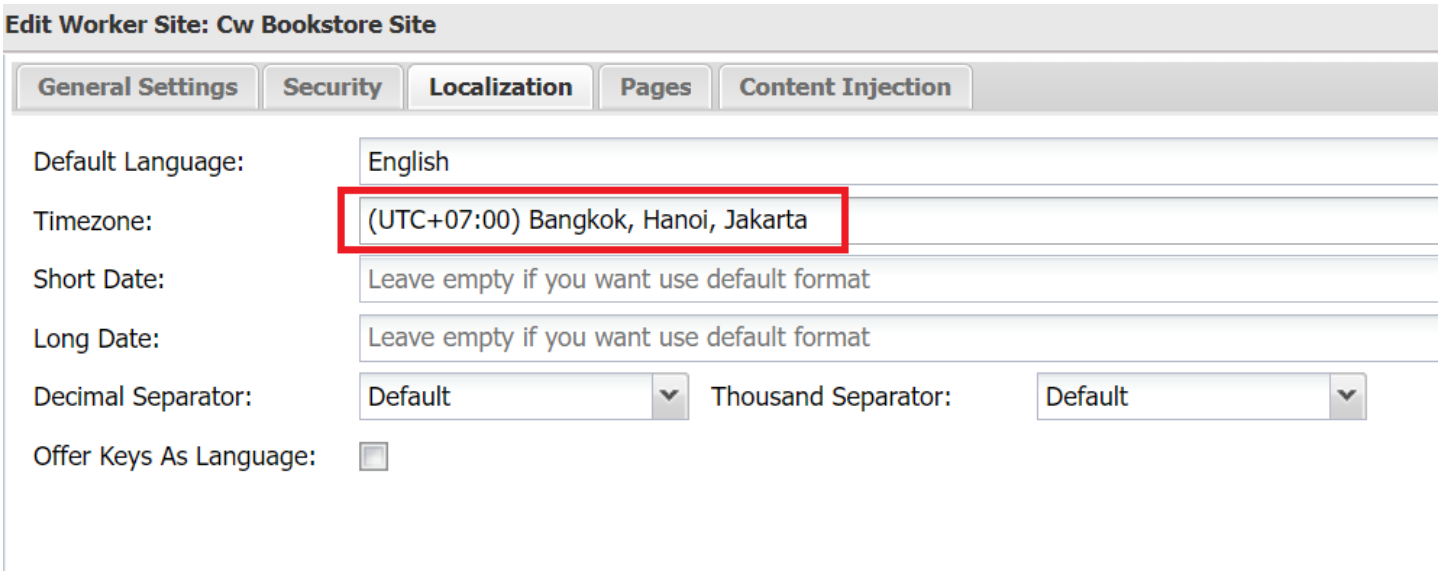
Set **Timezone** of worker site.



In Casewhere Admin, the current time zone of this worker site is set to UTC+1.



Change the time zone to UTC+7.

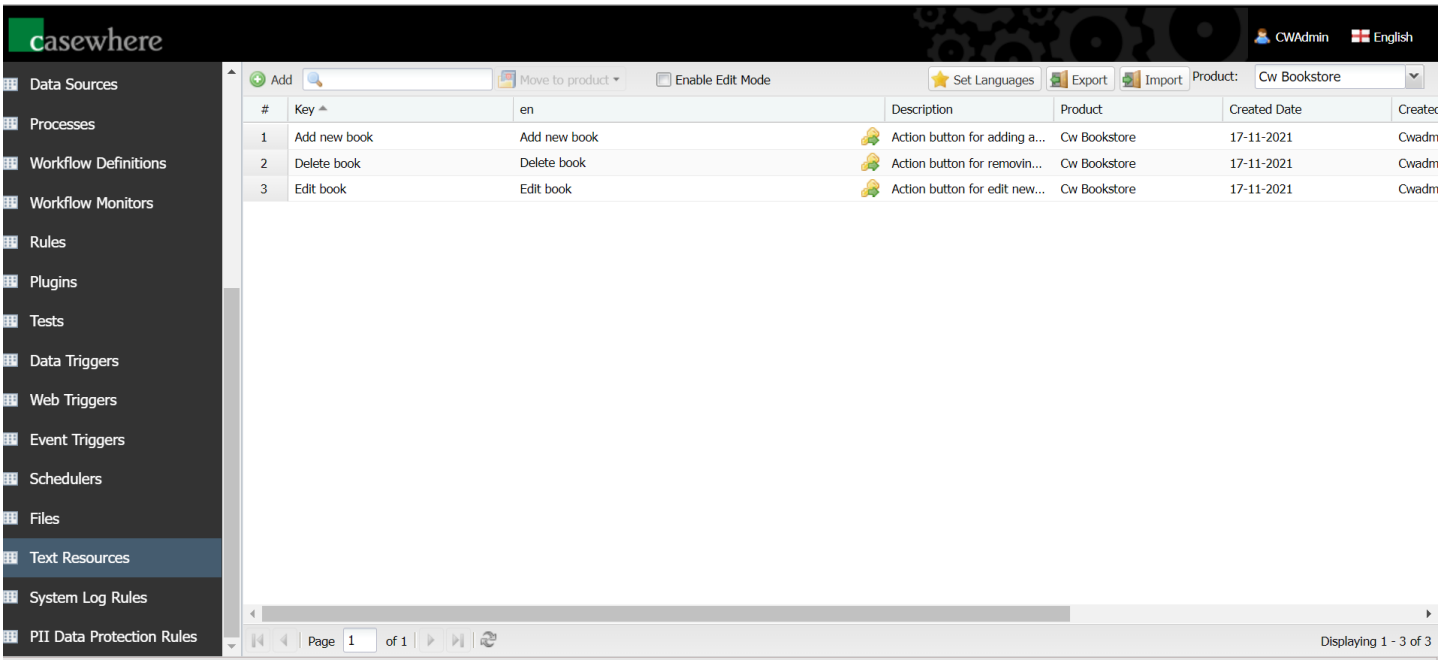


The time will be displayed in time zone UTC+7.

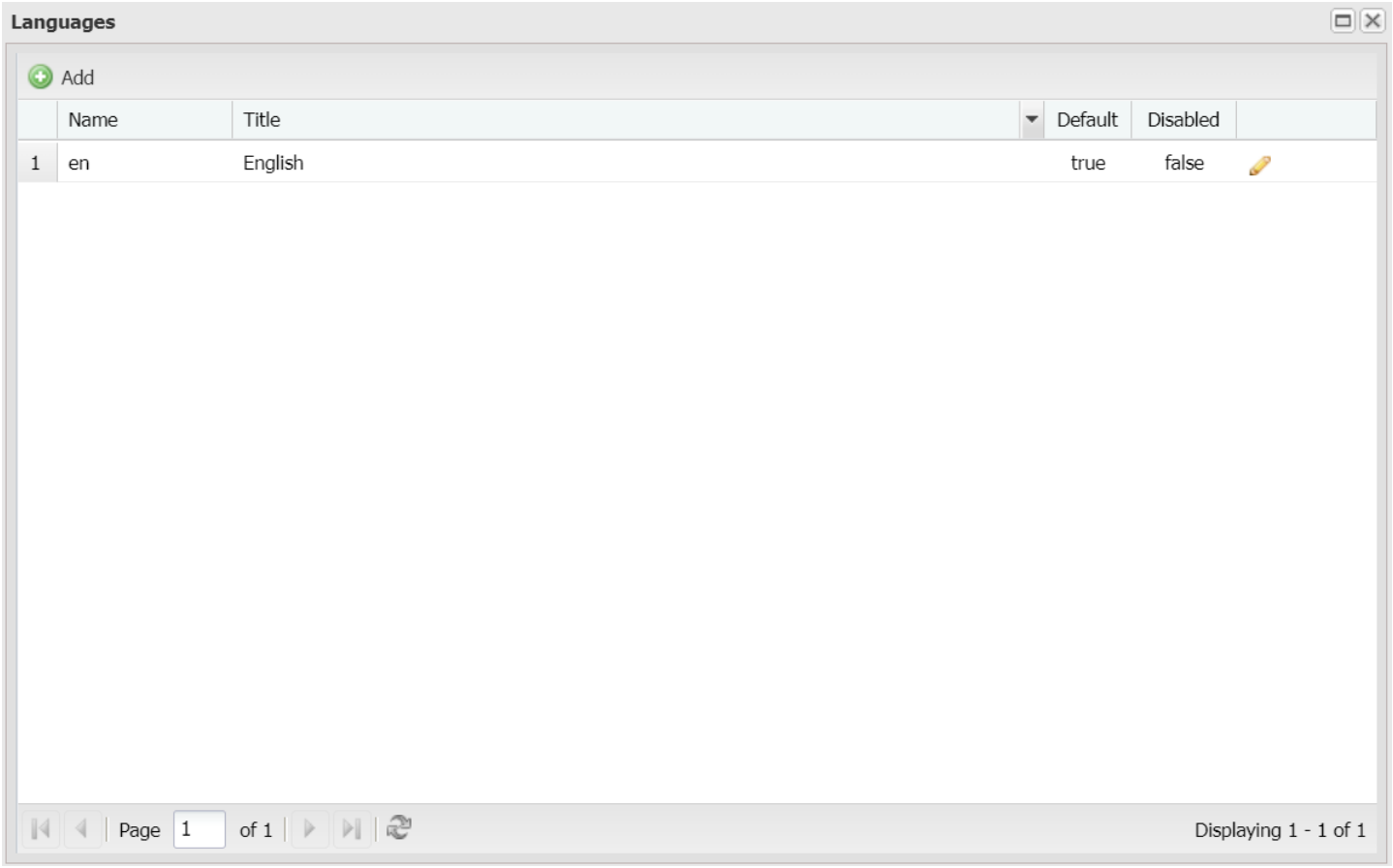


Configure worker site languages

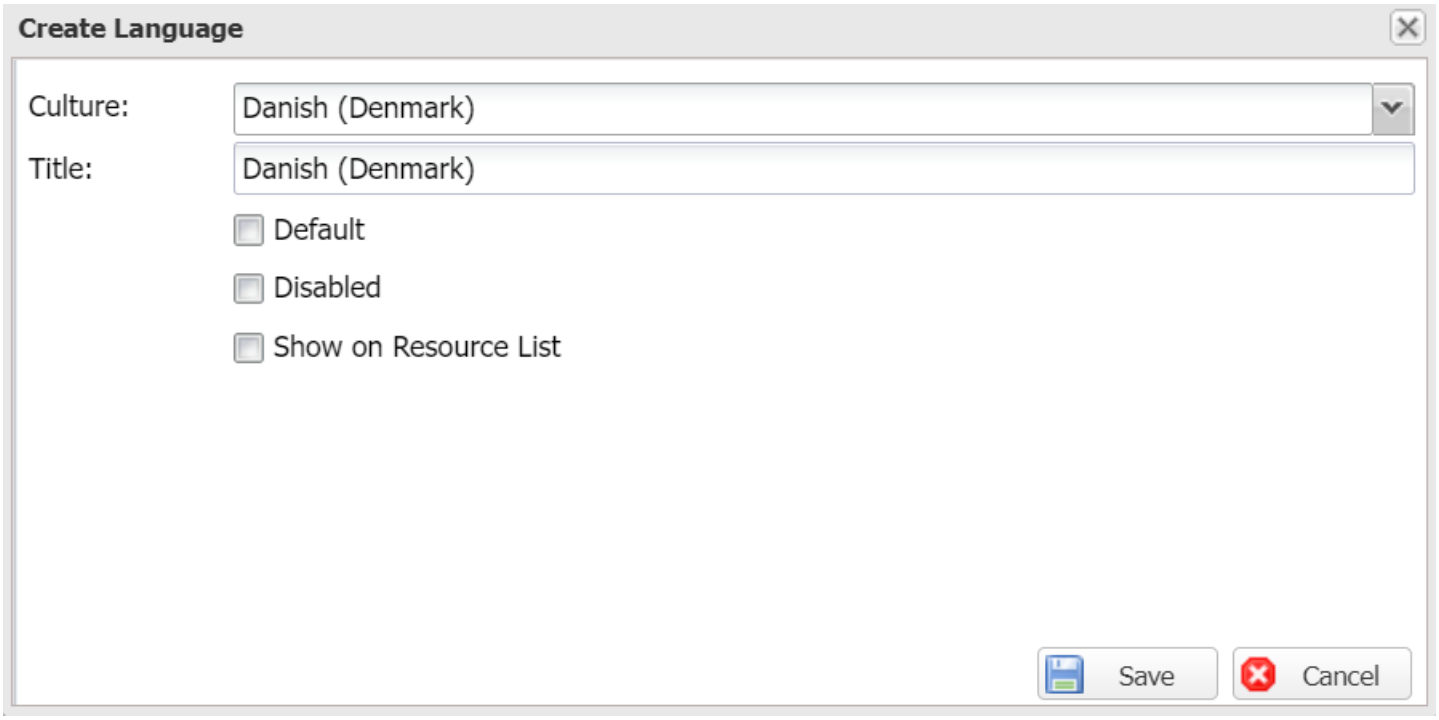
- In Casewhere Admin, click on menu **Text Resources** in the left panel.



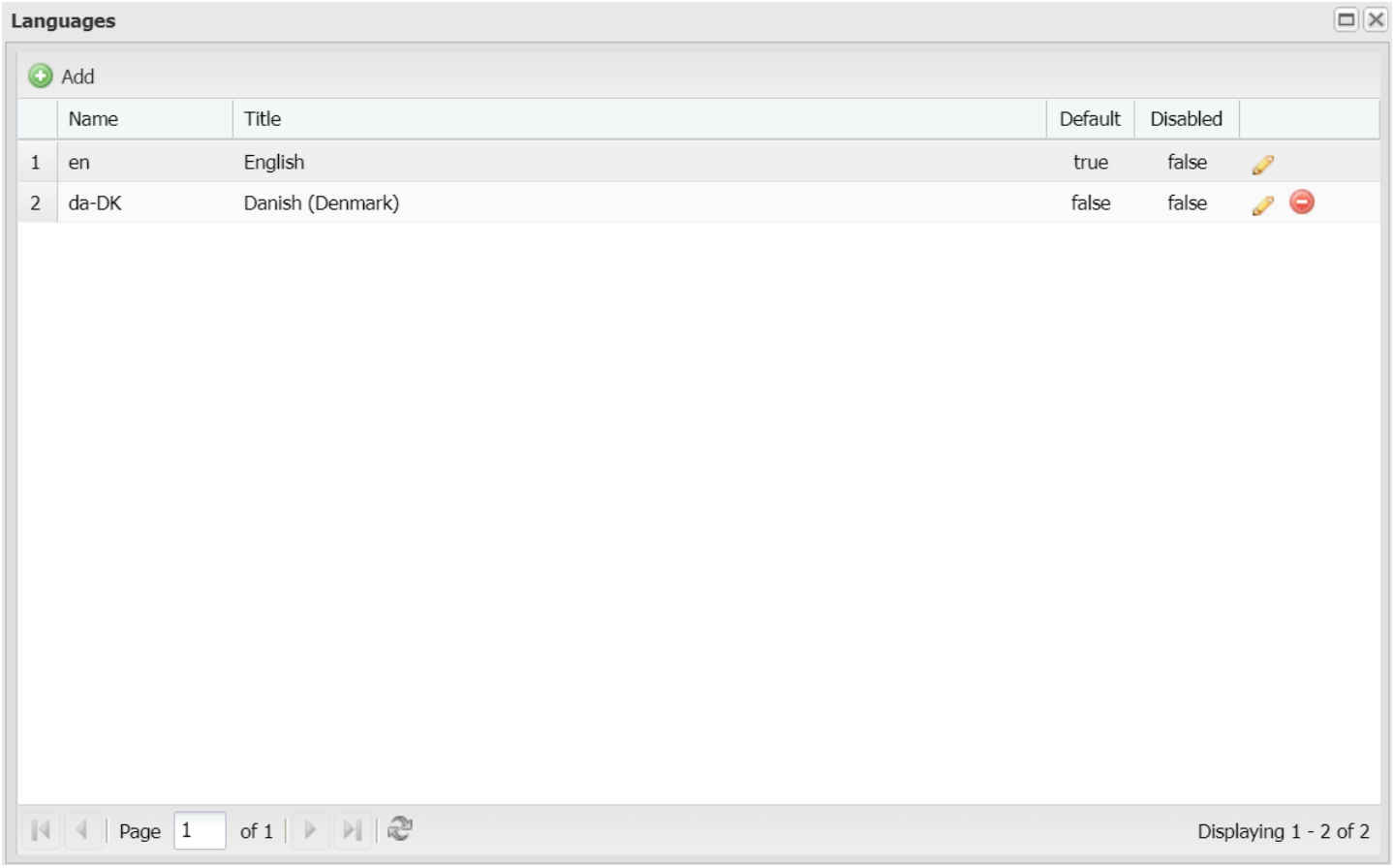
- Click on **Set Languages** toolbar button.



- Click button **Add** to add new language.

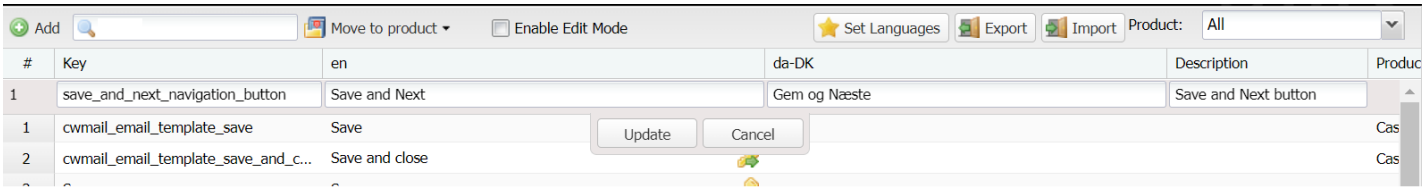


- Click **Save**.

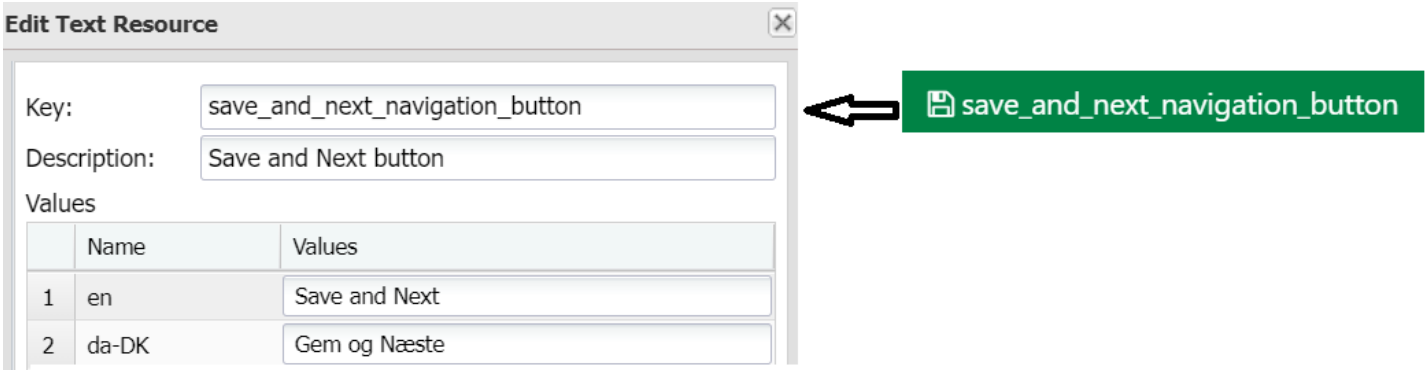


Add key

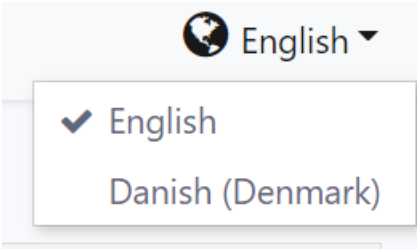
- Click on **Add** toolbar button.
- Enter the key name and its values in different languages.



The key name must be the same as displayed name. Below is an example for a button named "save_and_next_navigation_button".



Now go to the worker site to see how it works. On the top right of your worker site, you will see the list of languages that you have set before.



As set, English is the default language, so the text of button will be displayed in English.



When selecting Danish, the text of button will be displayed in Danish.



Worker site API

Besides integrating with AngularJS built-in services, Casewhere provides a set of APIs to give you more flexibilities to develop worker site components, custom widget, rich-UI form activities, etc.

API list

`AppConfig`

A constant contains all the app's config. For example:

`CurrentUser`

To interact with the user session in the worker site.

`isLoggedIn`()

Deprecated: This method no longer makes sense and will be removed in future versions. Determine whether the user is logged in. This method always returns true if the site allows anonymous users.

Result: bool

`isAuthorized`()

Deprecated: This method no longer makes sense and will be removed in future versions. Determine whether the user is authorized.

Result: bool

`hasClaim(type, value)`

Determine if the user has a specific claim or not.

Result: bool

`getClaims()`

Result: The claim-set object

`loginWithDialog()`

Open the login dialog to authenticate users.

Result: A promise

`forceLogIn(returnUrl)`

Force a login. The user after logins with be reditected to the `returnUrl`. If the `returnUrl` is not provided, Casewhere will redirect the user to the home page.

`logOut(returnUrl)`

Logout and redirect the user to the `returnUrl`. If the `returnUrl` is not provided, Casewhere will redirect the user to `'/logout'`;

`isAuthenticated()`

To determine whether the user is authenticated on the client side.

`DataSourceService`

To get data from a data source. Read more at Restful data source.

`search(dataSourceName, query)`

Search against a data source.

Parameters

- `dataSourceName`: The name of the data source
- `query`: The data query object
  - `filter`: The filter object
  - `fields`: The returned fields
  - `skip`: The number of records to be skipped. This is to support data paging.
  - `take`: The number of records to be returned. This is to support data paging.

Result: A promise returns the data

Example:

```
var query = {
    filter: {
        Kommunenr: '101'
    }
};
DataSourceService.search('People', query).then(function(response) {
    console.log(response.data);
    console.log(response.total);
    console.log(response.metadata);
    console.log(response.isSearchLimitExceeded);
});
```

`loadDataObject(dataSourceName, doId)`

Load a data object by id.

Parameters

- `dataSourceName`: The name of the data source
- `doId`: The data object id

Result: A promise returns the data

Example

```
var query = {
    filter: {
        Kommunenr: '101'
    }
};
DataSourceService.search('People', "person/cf732abc-7a5c-4773-9998-4d4f8ccdb023").then(function(response) {
    console.log(response.data);
});
```

`DocumentService`

To work with the Casewhere file storage.

`generateToken(doId, propertyName)`

Generate a token for downloading a document.

Parameters

- `doId`: The data object id holding the document
- `propertyName`: The data attribute holding the document

Result: a promise returns the token object

```
{ tokenId: "1b2b3e7a5d0544218bcb9495cda043fd" }
```

`generateWorkflowVariableToken(workflowId, variableName)`

Generate a token for downloading a document referenced by a workflow variable.

Parameters

- `workflowId`: The workflow id
- `variableName`: The workflow variable name

Result: a promise returns the token object

```
{ tokenId: "1b2b3e7a5d0544218bcb9495cda043fd" }
```

`download(tokenId)`

Trigger a download given a token id.

Parameters

- `tokenId`: The token id retrieved from `generateToken`

```
DocumentService.download(token.tokenId);
```

`getDownloadUrl`

Return a `GET` download endpoint.

Parameters

- `tokenId`: The token id retrieved from `generateToken`

Result: an URL

`Logger`

To work with application notifications.

```
Logger.log(aMessage);
Logger.error(errorMessage, { positionClass: 'toast-top-right' });
Logger.info(infoMessage); // default position is 'toast-bottom-right'
Logger.warning(warningMessage, { positionClass: 'toast-bottom-right' });
Logger.success(successMessage); // default position is 'toast-bottom-right'
```

`PluginService`

To work with a Casewhere plugins. Read more at Plugin API.

`invoke("pluginName", "pluginMethodName", parameters)`

Parameters

- `pluginName`: The plugin name
- `pluginMethodName`: The method name

- `parameters`: The parameteres

Result: a promise returns the plugin result

Example:

> **Note:** You can only invoke public plugins from worker site.

```
var parameters = { Name: "John Doe" };

PluginService.invoke("pluginName", "pluginMethodName", parameters)
    .then(function (pluginResult) {
        console.log(pluginResult);
    }, function (error) {
        console.log(error);
    });
```

`RuleService`

To evaluate a client-side rule.

`evaluate(ruleName, parameters)`

Parameters

- `ruleName`: The rule you want to evaluate
- `parameters`: The rule parameters

Result: an object

Example:

Imagine you have the following client-side rule configured in Casewhere Admin:

```
// name: get billing address
function (user, parameters)
{
    var isBusiness = user.contains("type", "business")
    if (isBusiness) return parameters.companyAddress;
    else return parameters.personalAddress;
}
```

You can evaluate the rule in the worker site as below:

```
var billingAddress = RuleService.evaluate("get billing address", {
    companyAddress: "Sunset Boulevard xxx",
    personalAddress: "Refshalevej xxx"
});
```

`$translate`

`instant(textResourceKey)`

To translate a text resource.

Parameters

- `textResourceKey`: the text resource key

Result: The translated text

Example:

`WebTriggerService`

To work with web triggers.

`get(webTriggerName, data, config)`

Parameters

- `webTriggerName`: The web trigger name
- `data`: The data to be sent to the server as a query string
- `config`: [Optional] The object describing the request to be made and how it should be processed. Read more at $http

Result: a `HttpPromise` object. Read more at $http

Example:

```
LoaderService.show();
WebTriggerService.get("export", parameters).then(function(response) {
    Logger.info(response.data);
}, (error) => {
    Logger.error(error);
}).finally(function() {
    LoaderService.hide();
});
```

`post(webTriggerName, data, config)`

Parameters

- `webTriggerName`: The web trigger name
- `data`: The payload
- `config`: [Optional] The object describing the request to be made and how it should be processed. Read more at $http

Result: a `HttpPromise` object. Read more at $http

Example:

```
WebTriggerService.post("search", data).then(
    function (response) {
        model.metadata = response.metadata;
        model.gridOpt.data = response.data.data;
        model.gridApi.grid.options.totalItems = response.data.totalItems;
    }
);
```

Filters

`cwFileLinkFilter`

Transform the system document name to a download link.

Example

Input: logo

Output: https://local-api.casewhere.com/api/v0.1/system-documents/logo

```
<img ng-src="{{'logo' | cwFileLinkFilter}}"></img>
```

`workerLink`

Transform a Worker API relative path to an absolute path.

Example

Input: system-documents/logo

Output: https://local-api.casewhere.com/api/v0.1/system-documents/logo

```
<img ng-src="{{'system-documents/logologo' | workerLink}}"></img>
```

`cwWorkerSiteDate`

Convert a date to the worker site time zone and render it using a given format. Learn more about the formats here.

Example

```
{{date | cwWorkerSiteDate:'shortDate'}}
```

`workerSiteLink`

Transform a Worker Site relative path to an absolute path.

Example

Input: resource-name

Output: https://local-worker.casewhere.com/resource-name

Working with worker site APIs

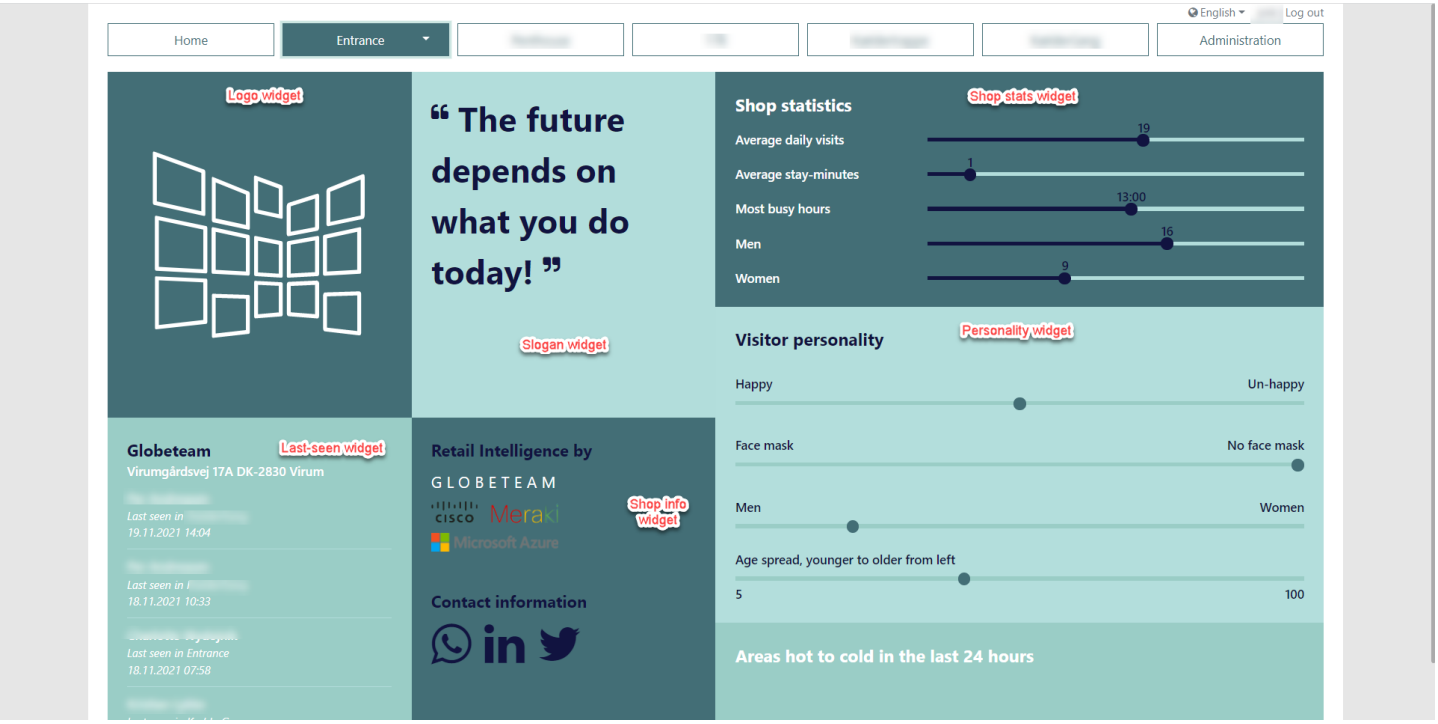
All components in Casewhere Worker Site, such as worker site components, custom widget, etc., are implemented as a AngularJS directive. To work with Casewhere APIs, you will need to learn how to inject a service into a directive. Please follow this instruction and **avoid the Implicit Annotation** so your component can work properly when Casewhere minifies the scripts.

Implicit Annotation

Careful: If you plan to minify your code, your service names will get renamed and break your app.

Page and widget overview

Casewhere page-and-widget is a system designed to help you, as the solution developer, create and manage content efficiently. The system will allow you to design and control the layout of the content, i.e., how a page will look and how it responds on different devices. Pages are more like containers, while widgets are all about how you want to present your data and how your end-users process these data through Casewhere workflows.

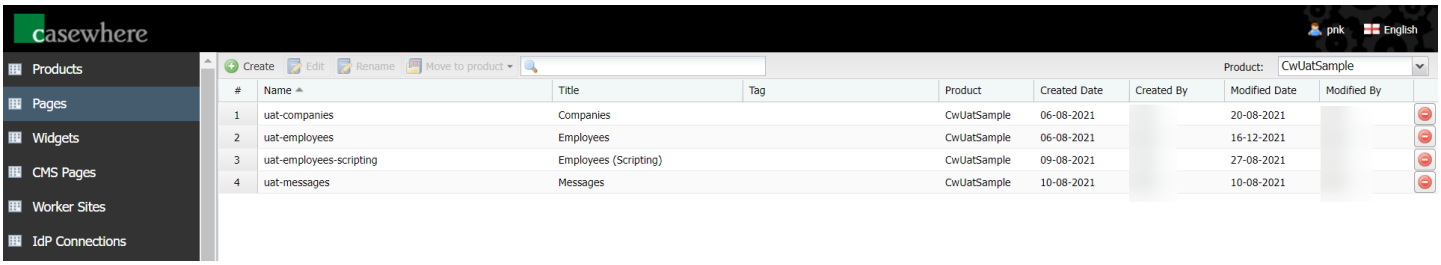


Casewhere page-and-widget is aimed to maximize reusability. A page can be linked to different worker sites, and a widget can be embedded into different pages. Widgets are not responsible for loading and updating data directly. They act as gateways to the underlying data, i.e., widgets will pull data from data sources and send it back to workflows for processing. Data sources and workflows can be shared among widgets, which brings the reusability even further.

Casewhere page-and-widget is built with a strong security mindset. Both pages and widgets are protected by access rules. You can control who can access which pages and see which widgets. Not just that, you can also control how a widget filters data depending on the user's privileges. Which workflows a user is allowed to trigger from the widget itself is also controllable by you as the solution developer. Take the book store tutorial as an example; you can totally configure that children are not allowed to browse books for adults, or only the store owner can add more books.

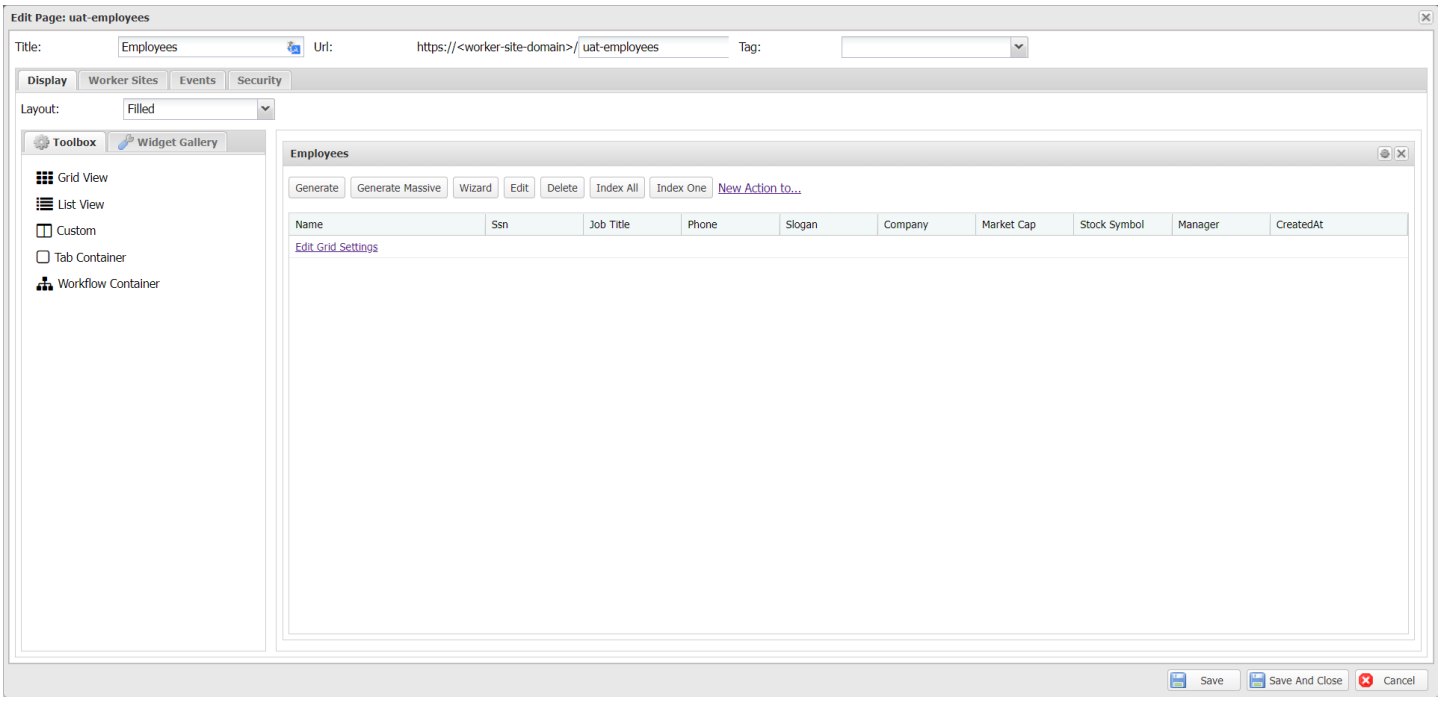
Page management

In Casewhere Admin, navigate to **Pages** to manage the sites for your solution. From here, you can create, edit and manage pages like any product resource.



Page editor

The page editor will give you all the necessary flexibilities to customize your page.



General settings

This is the information required to set up a page:

- **Title**: The page title shows in the worker site and the browser.
- **Url**: The relative path of the page in a worker site. The current limitation is that **Url** must be unique for the entire deployment, even when the pages belong to different worker sites.
- **Tag**: Tagging is useful for resource management.

Display settings

You can design your page by selecting the page layout type first, then dragging and dropping the widgets into the page.

Layout: There are two types of layouts:

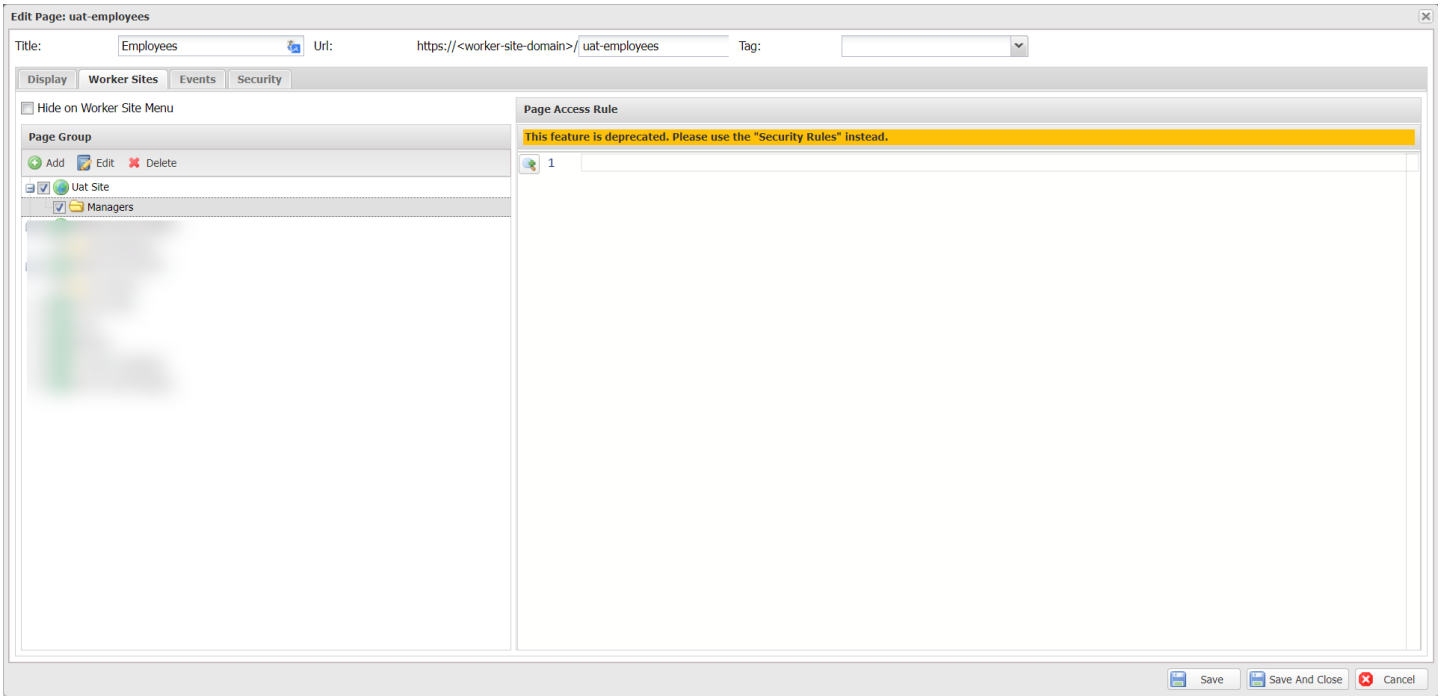
- **Filled**: Widgets are stacked one on top of another in a fixed-size panel.
- **Grid**: Widgets are organized using a grid of cells with the numbers of columns and rows are configurable. A cell can contain more than one widget. We can merge cells together or split them out. The grid layout is best suitable for displaying dashboards.

Widget Toolbox: There are five types of widgets:

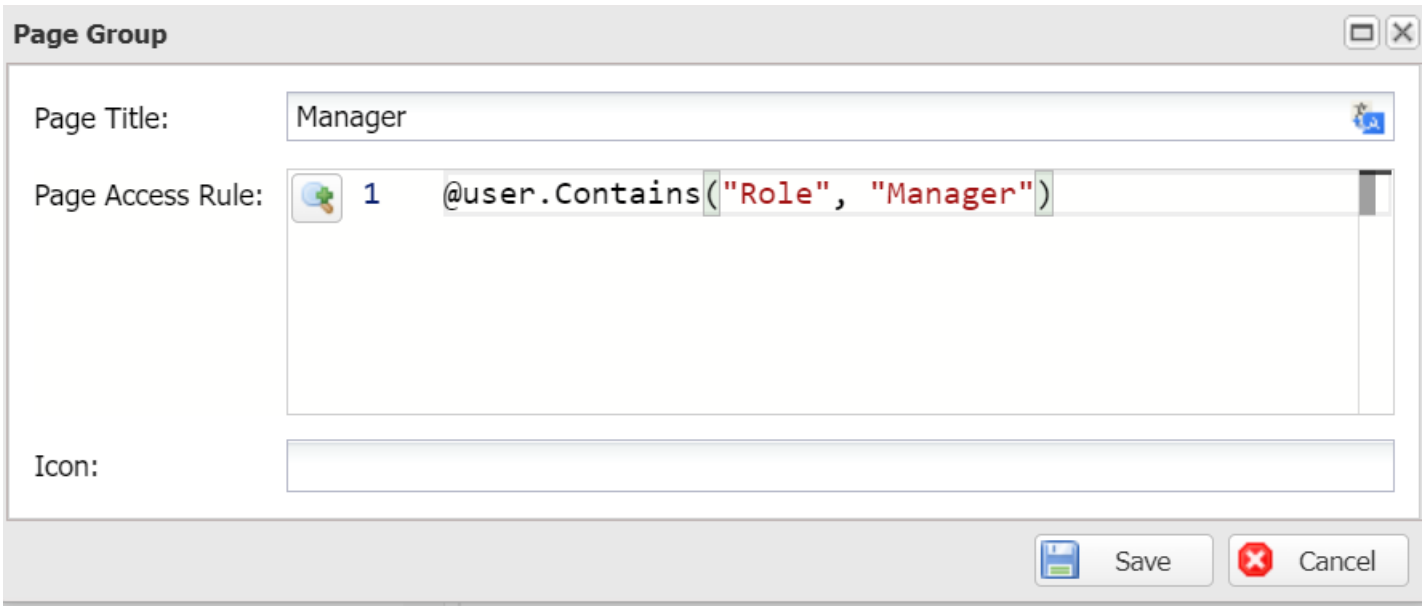
- **Grid View**: Displays data in a tabular structure. Each row usually represents a record in the database, while each column can represent one or more fields.
- **List View**: Display data in a more compact, responsive way.
- **Tab Container**: You can add widgets, e.g., grid views and list views, and organize them in separate tabs.
- **Workflow Container**: You can embed a workflow and show it on a page.
- **Custom**: By understanding that there always will be a logic that the built-in widgets cannot fully handle, Casewhere offers a scripting solution – the powerful custom widget with full scripting flexibilities to design anything we have in mind i.e. from the ability to customize looks and feels to the capability of handling user interaction in your own ways.

Worker sites

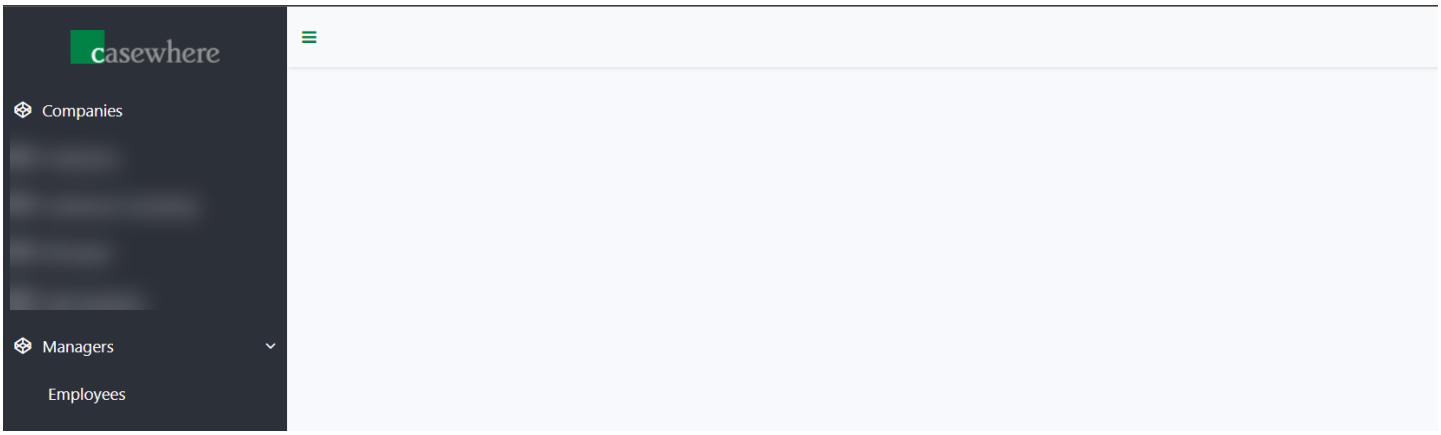
You can link your page to one or many worker sites by selecting the worker sites in the **Page Group** panel.



In order to better organize and secure pages, you can create a page group and define the access rule to protect the group. For example, in the **Page Group** panel, you can **Add** a page group `Managers` to contain all the pages only accessible by manager users and set the rule like below:



In worker sites, pages and page groups are displayed like below.

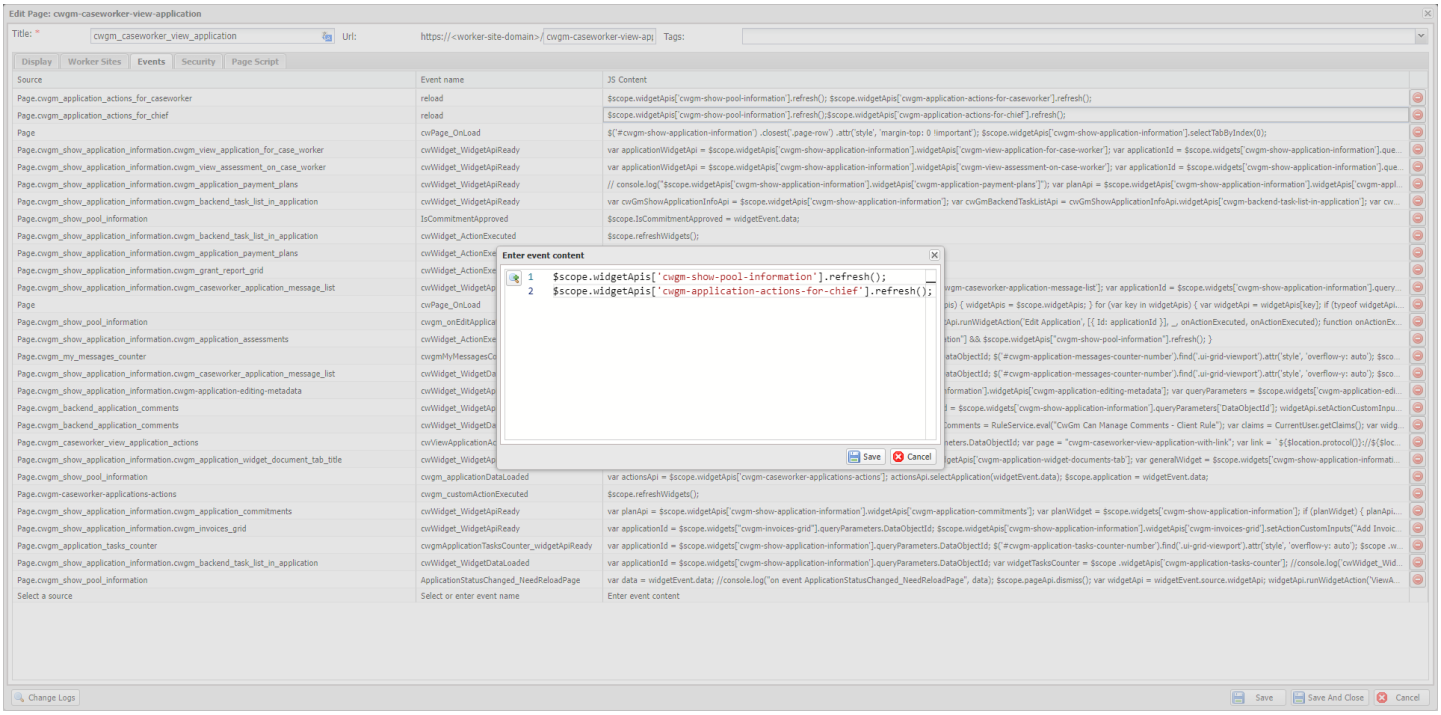


Page events

Casewhere supports widget events so that you can subscribe to an event emitted from a widget and trigger an action in another widget.

To subscribe to an event:

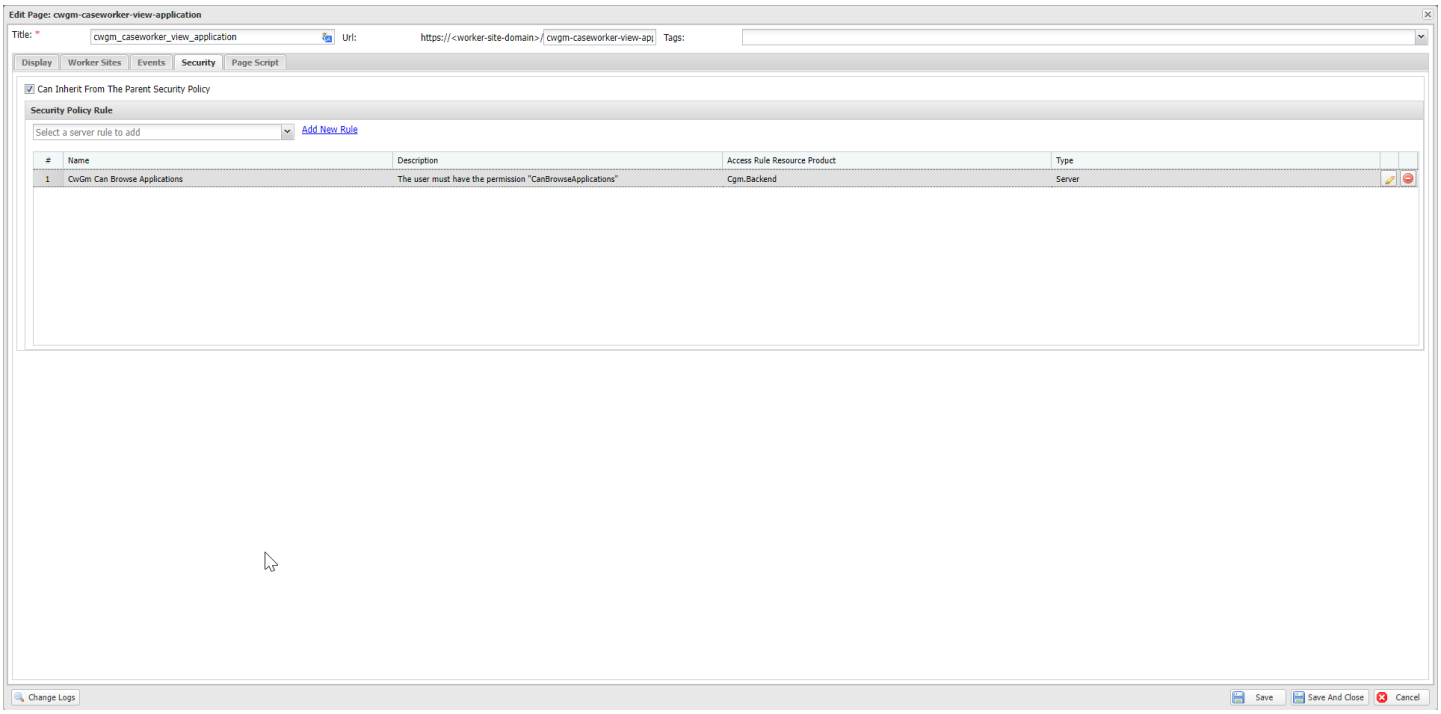
- Select **Source**; this is the widget that emits the event.
- Select the event you want to subscribe to. For custom widgets, you have to type the event name.
- Implement logic for the event handler in **JS Content**.



You can learn more about page events here.

Security

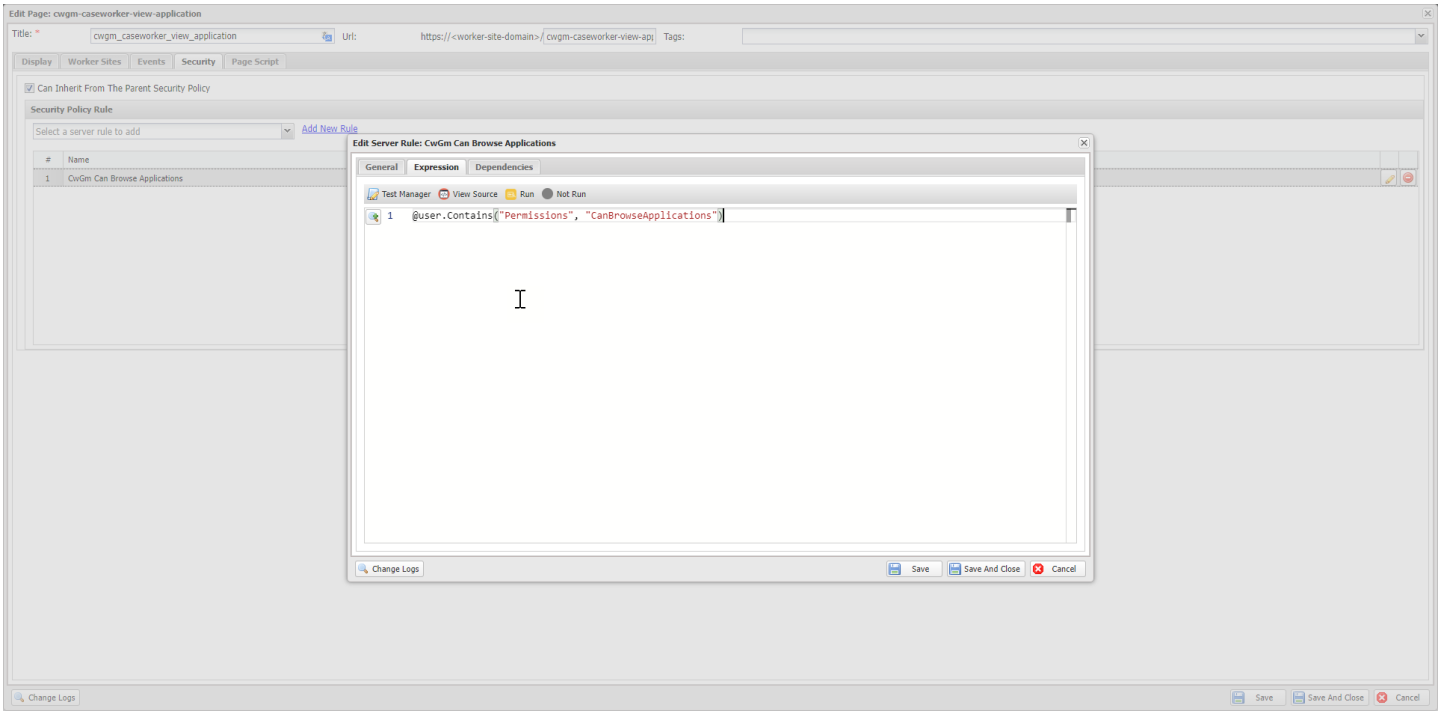
By defining the access rules, you can control who can access the page and what information should be shown. In the page editor, navigate to the **Security** tab to manage access to the page.



A good practice is to inherit access rules from the upper level, i.e., the worker site in this case. Additional access rules can be assigned if the page requires stricter access control.

Example: Only users with "CanBrowseApplications" permission can access the page.

```
@user.Contains("Permissions", "CanBrowseApplications")
```



See Access rule for more information.

Page event

A page may contain one or more widgets, and commonly, for rich-UI applications, there will be a need for a widget to respond to application events automatically. For example, when end-users select a new book in the book list, they want to subscribe to a change in the book list widget and refresh data in the book detail widget. However, widgets in Casewhere are independent entities, so it's not a good practice to have them calling each other. Hence, Casewhere offers the **page events** feature to provide a central place to orchestrate the widget communication while keeping them decoupled.

There are two types of events: built-in events and custom events. Basically, all events will share the information below:

1. Event name: For built-in events, the event name is defined by Casewhere. For custom events, you, as the solution developer, will need to define the event name for your custom widgets.
2. Event source: The entity emits the event. It could be a widget in a page or the page itself.
3. Event data: The data encapsulated by the event. For example, the book object selected by end-users.

Built-in events

Built-in widgets will emit built-in events. An event object looks like below:

```
{
    source: {
        widgetId: '4c0a4156-b307-4a29-8f2e-27d59633c4f6',
        widgetName: 'book-list',
        widgetApi: // each widget has a set of Apis accessible through this object
    },
    data: // anything
}
```

Grid widget / List widget

`cwWidget_ItemSelectChange`

The event is emitted when the end-users select an item in the grid/list widget.

Event data is a list of selected items. For example:

```
[
    { Id: "Books/5b8a8b4a-18ae-496a-bdb3-8bd83da308d2", Name: "Book 1" },
    { Id: "Books/9047a790-7442-48f1-a7ac-3d744b3a151f", Name: "Book 2" }
]
```

`cwWidget_ActionExecuted`

The event is emitted when the end-users click an action button in the grid/list widget.

Event data is an object containing the action name. For example:

```
{
    actionName: "Edit Book"
}
```

`cwWidget_WidgetApiReady`

The event is emitted after the widget object is initiated.

This event does not have data.

`cwWidget_WidgetDataLoaded`

The event is emitted after the widget data is loaded.

This event does not have data.

`cwWidget_ItemDoubleClick`

The event is emitted when the end-user double-clicks on the list item.

Event data for the grid widget is a *ui-grid* row, while it is a data object for the list widget.

Example for grid widget:

```
{
    $$height: // the height of the row,
    element: // a jQuery object that represents the DOM row element,
    entity: {CaseId: 'b03ff96c-5e4d-464e-a58c-ae0500c5db0b', Title: 'Ulysses', Author: 'James Joyce', Year:
'1922', …},
    grid: // the grid that contains this row,
    isFocused: false,
    isSelected: false,
    rowTemplate: "caseListRowTemplate.html",
    uid: "uiGrid-000C",
    visible: true
}
```

Example for list widget:

```
{
    CaseId: "b03ff96c-5e4d-464e-a58c-ae0500c5db0b",
    Title: "Ulysses",
    Author: "James Joyce",
    Year: "1922",
    CaseState: "Created",
    Id: "Book/43634984-7596-4bfc-aaae-ae0500c60d14",
    selected: false
}
```

Tab container widget

`cwWidget_OnTabSelected`

The event is emitted when the end-user select a tab.

Event data contains information of the selected tab. For example:

```
{
    id: "a671bb8a-5242-459f-9809-583d9ac9ff60",
    title: "Adventure books",
    template: {
        html: // an escaped HTML string of a tab
    },
    clientVisibleRule: // an angularJs expression executed in the tab container scope,
    showTab: true
}
```

Workflow container widget

`cwWidget_WorkflowClosed`

The event is emitted when the end-user clicks the button **Save and close**.

Event data contains information of the workflow in context. For example:

```
{
    workflowDefinitionId: "9047a790-7442-48f1-a7ac-3d744b3a151f",
    workflowId: "d6f75834-d380-425a-983d-16cf76f46a4c",
    caseId: "8c43e3a1-ea20-4c12-b58e-25b1ec8b975f"
}
```

`cwWidget_WorkflowCancelled`

The event is emitted when the end-user dismiss the workflow dialog. It could be the **Cancel** button or the `X` icon in the dialog.

Event data contains information of the workflow in context. For example:

```
{
    workflowDefinitionId: "9047a790-7442-48f1-a7ac-3d744b3a151f",
    workflowId: "d6f75834-d380-425a-983d-16cf76f46a4c",
    caseId: "8c43e3a1-ea20-4c12-b58e-25b1ec8b975f"
}
```

`cwWidget_WidgetApiReady`

The event is emitted when the widget is initiated and ready.

This event does not have data.

Page

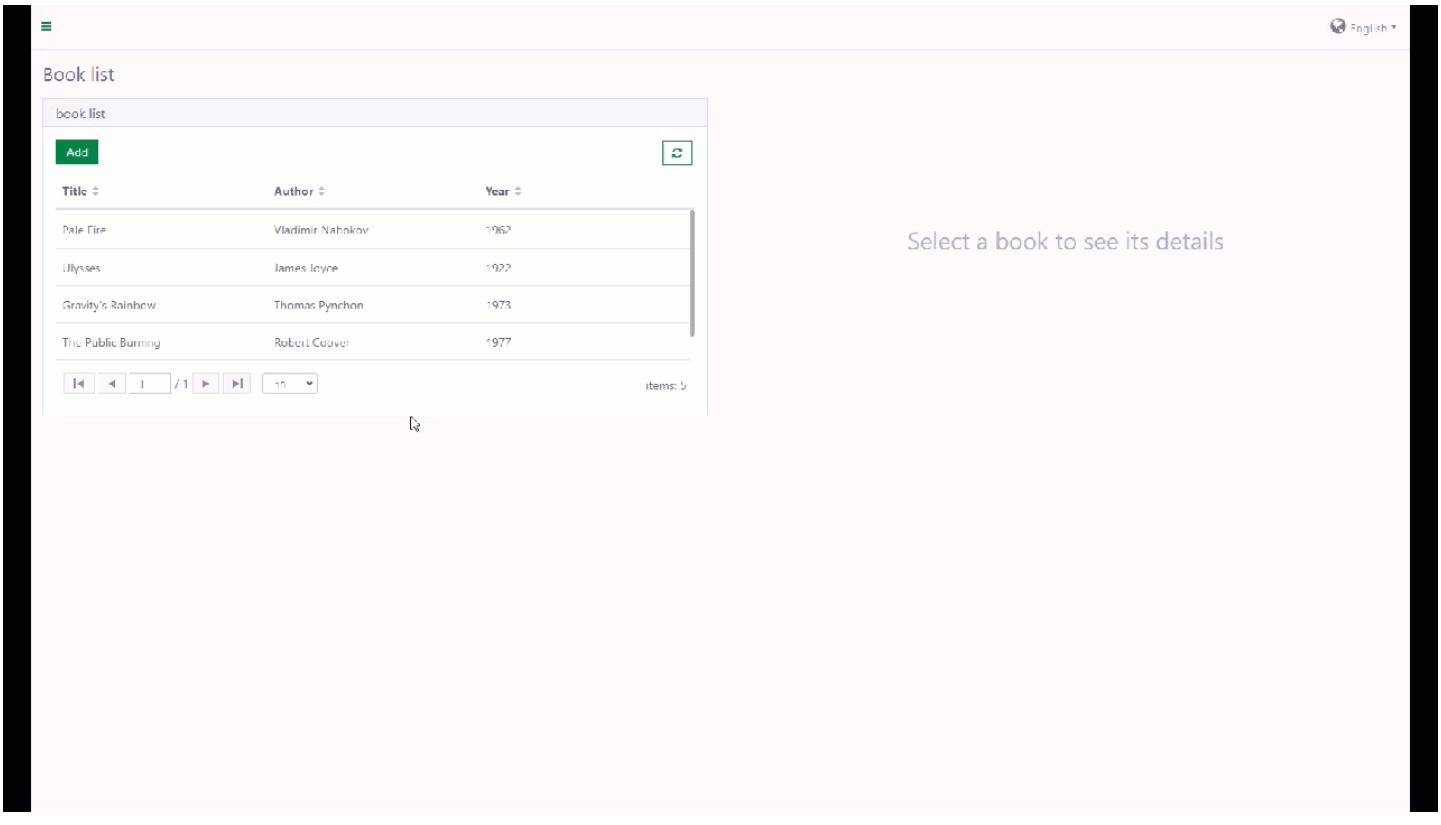
`cwPage_OnLoad`

The event is emitted when the page is loaded.

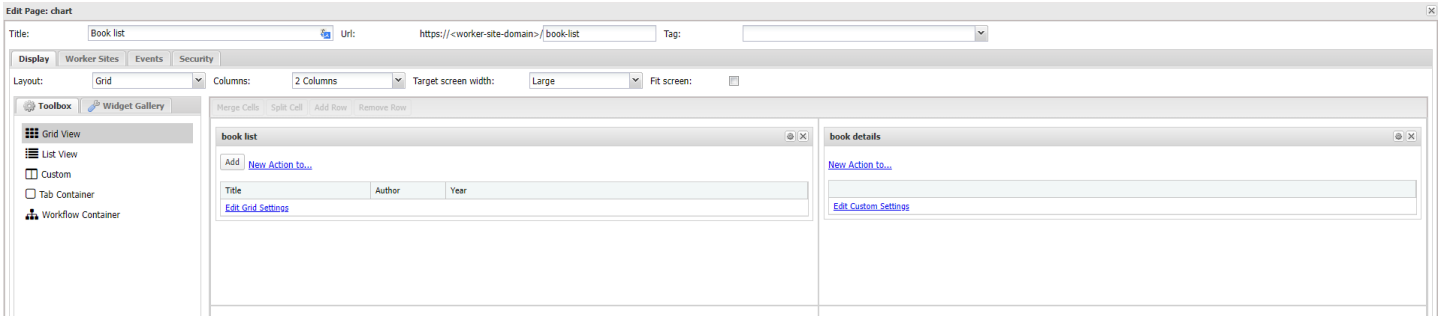
This event does not have data.

Example

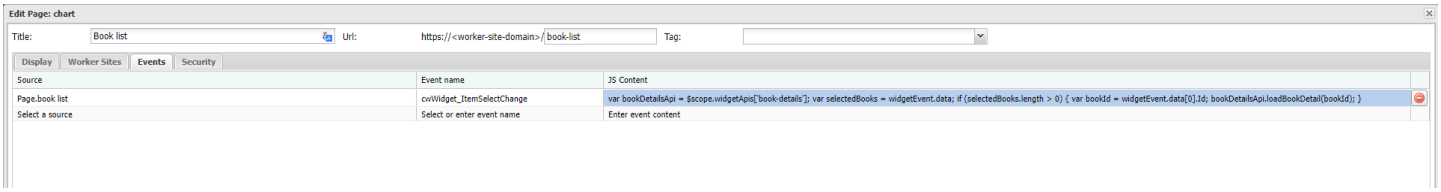
In this example, we have a book-list page, and after the end-user selects a book, the book details will be shown on the right side.



To do that, we set up a page as below:



To handle the event from the book-list grid, we set up a page-event handler as below:



Below is the script for the event handler:

```
var bookDetailsApi = $scope.widgetApis['book-details'];
var selectedBooks = widgetEvent.data;
if (selectedBooks.length > 0) {
    var bookId = widgetEvent.data[0].Id;
    bookDetailsApi.loadBookDetail(bookId);
}
```

Custom events

Custom widgets will emit custom events. An event object looks like below:

```
{
    source: {
        widgetId: '4c0a4156-b307-4a29-8f2e-27d59633c4f6',
        widgetName: 'book-list',
        widgetApi: // each widget has a set of Apis accessible through this object
    },
    data: // anything
}
```

Emit custom event

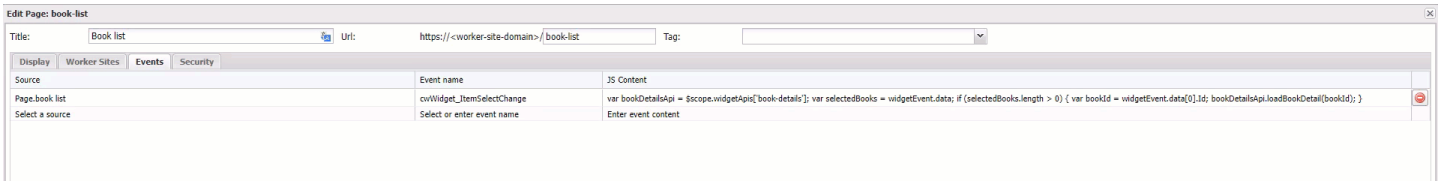
To emit a custom event, you can use this method `publishEvent(eventName, data)` where `eventName` is a free text, `data` is whatever information you think the handler will need. It can be a text, an object, etc.

The below method is embedded in all custom widget templates by default:

```
function publishEvent(eventName, data) {
    var widgetEvent = {
        source: {
            widgetId: $scope.widgetModel.id,
            widgetName: $scope.widgetModel.name,
            widgetApi: $scope.widgetApi
        },
        data: data
    };
    $scope.$emit(eventName, widgetEvent);
}
```

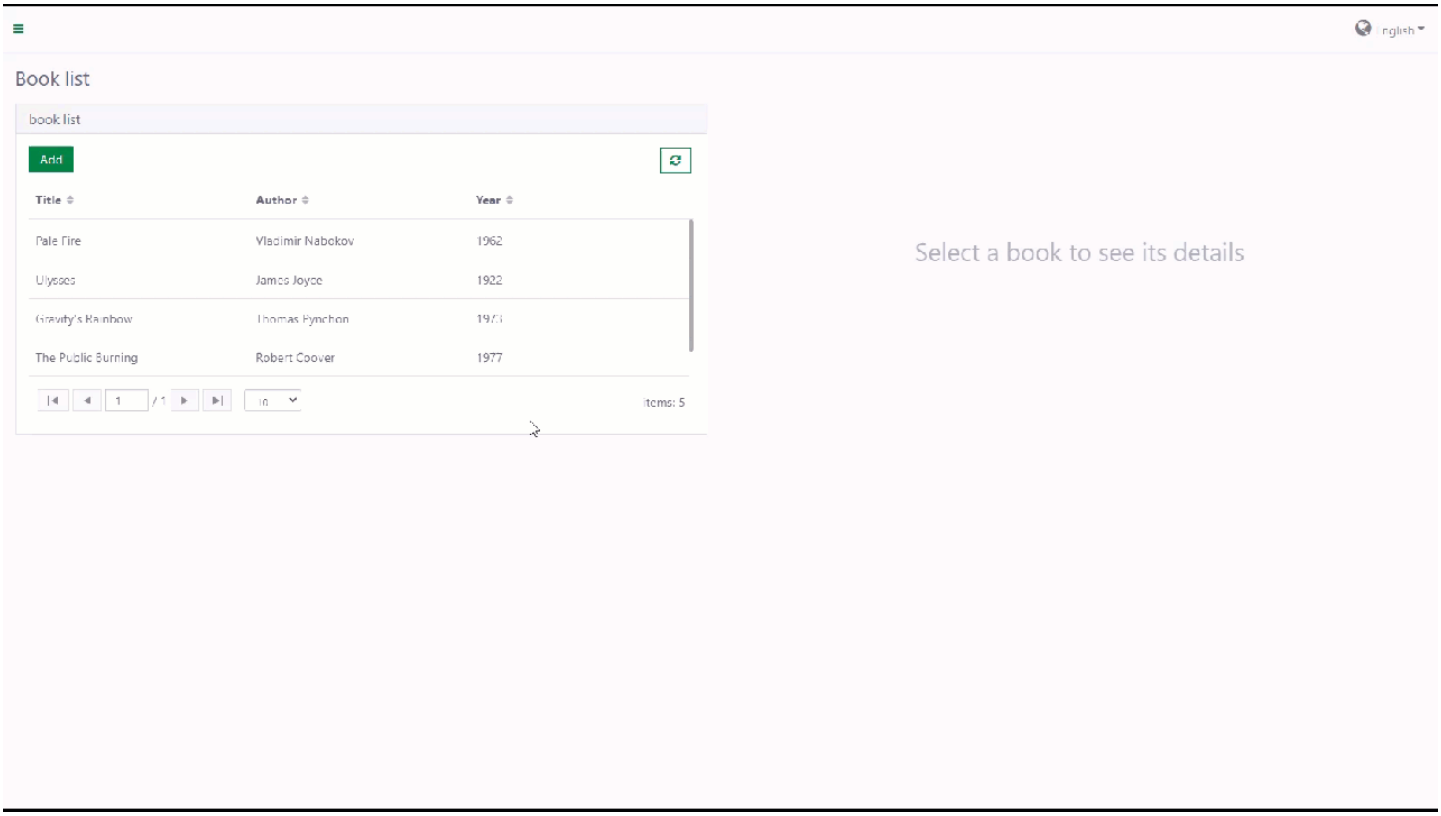
Handle custom event

It's almost the same with handling a built-in event, just a difference, which is instead of selecting an event from a predefined list, you will need to input the custom event name manually in the **Event name** textbox.



Example

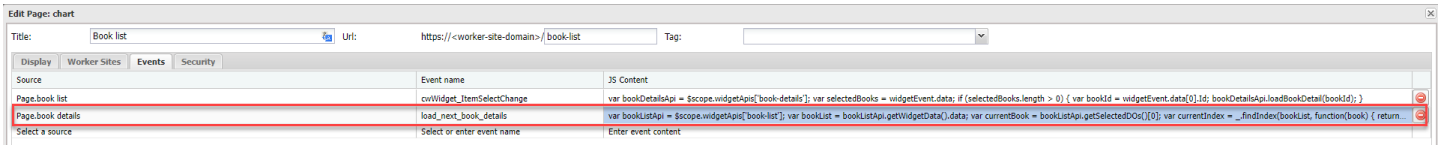
Let's continue the above example. We will add a button in the details pane, so when the end-user clicks this button, Casewhere will load the next book.



The script behinds this button:

```
function loadNextBookDetails() {
    publishEvent("load_next_book_details");
}
```

And finally, handle this custom event in the **page events**.

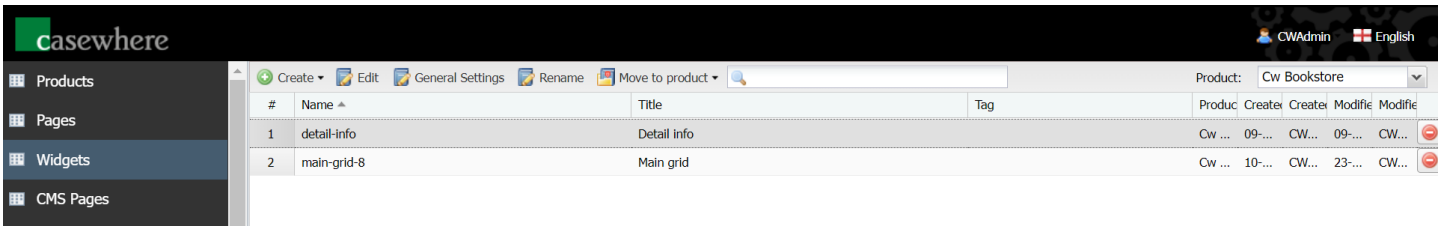


The full script of the event handler is as below:

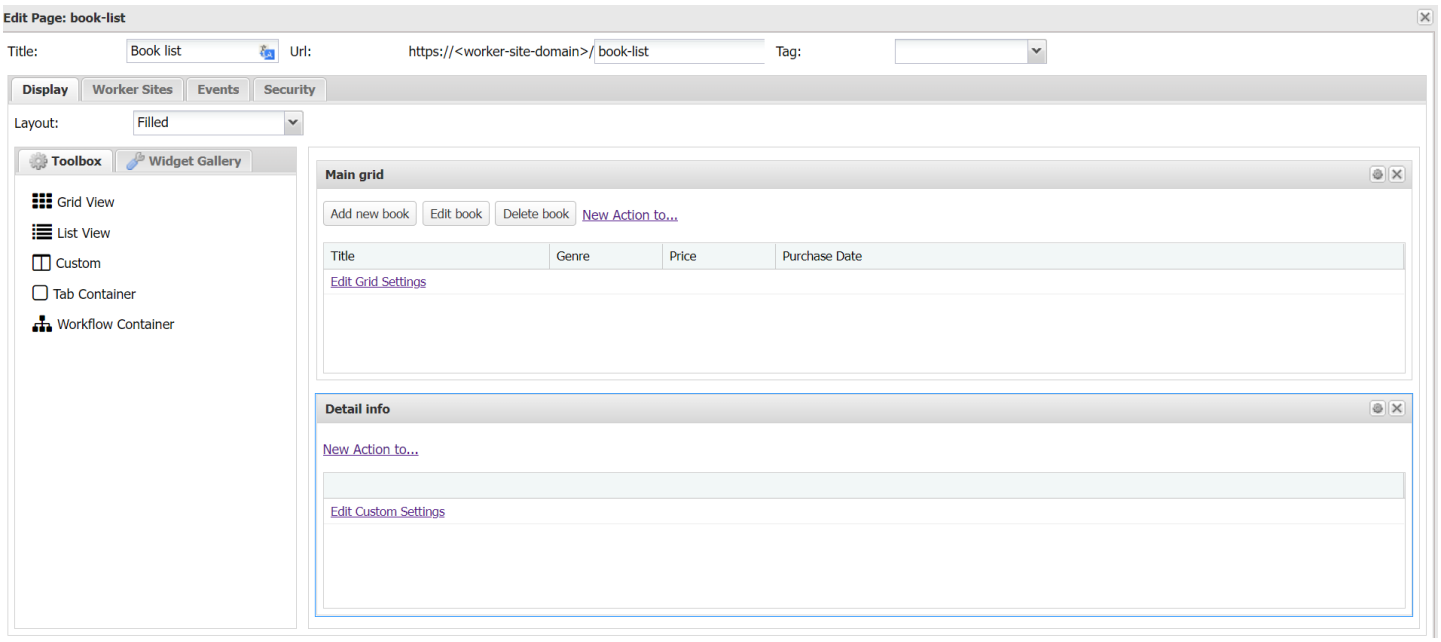
```
var bookListApi = $scope.widgetApis['book-list'];
var bookList = bookListApi.getWidgetData().data;
var currentBook = bookListApi.getSelectedDOs()[0];
var currentIndex = _.findIndex(bookList, function(book) {
    return book.Id === currentBook.Id;
});
var nextBook = bookList[currentIndex + 1];
// Deselect the selected book
bookListApi.setSelectedDOs([]);
// Select the next book
bookListApi.setSelectedDOs([nextBook]);
```

Widget management

In Casewhere Admin, navigate to **Widgets** to manage the widgets for your solution. From here, you can create, edit, and manage widgets like any product resource.

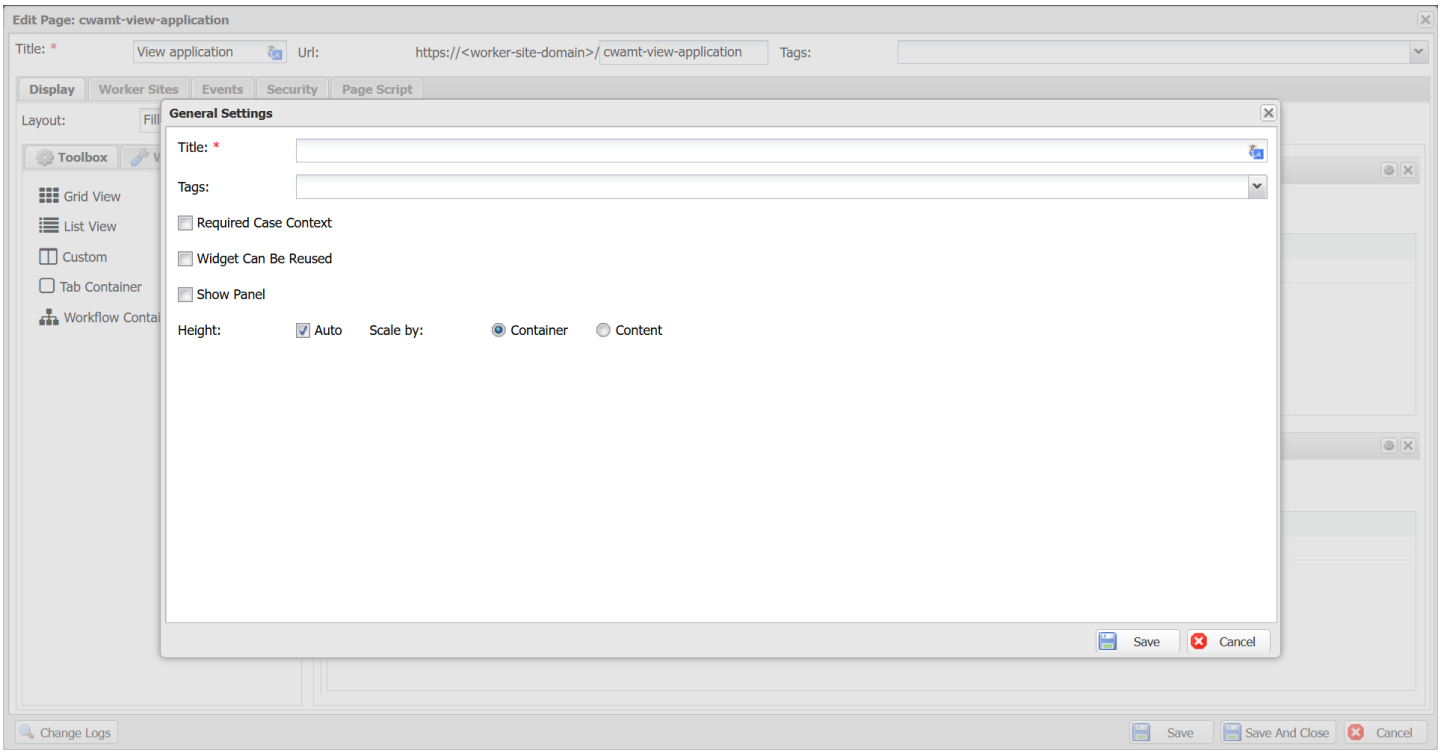


Most Casewhere developers typically use the page editor to work with widgets while designing pages. When you add a widget through the page editor, it will be added to the same product as the page.



Create widget

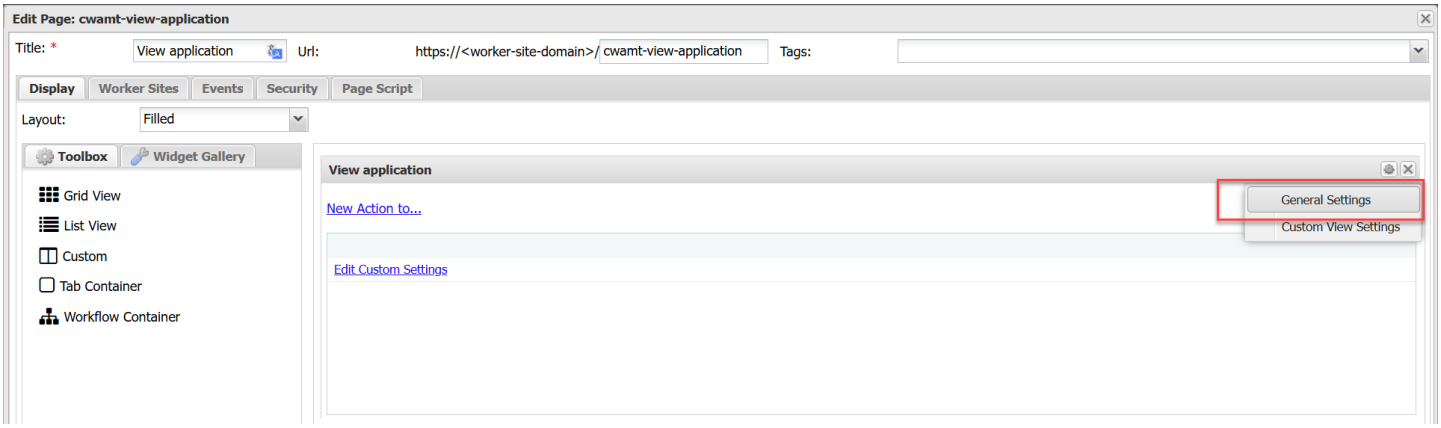
The simplest way to create a widget is by dragging it from the toolbox onto the page.



The below information is needed when creating a new widget:

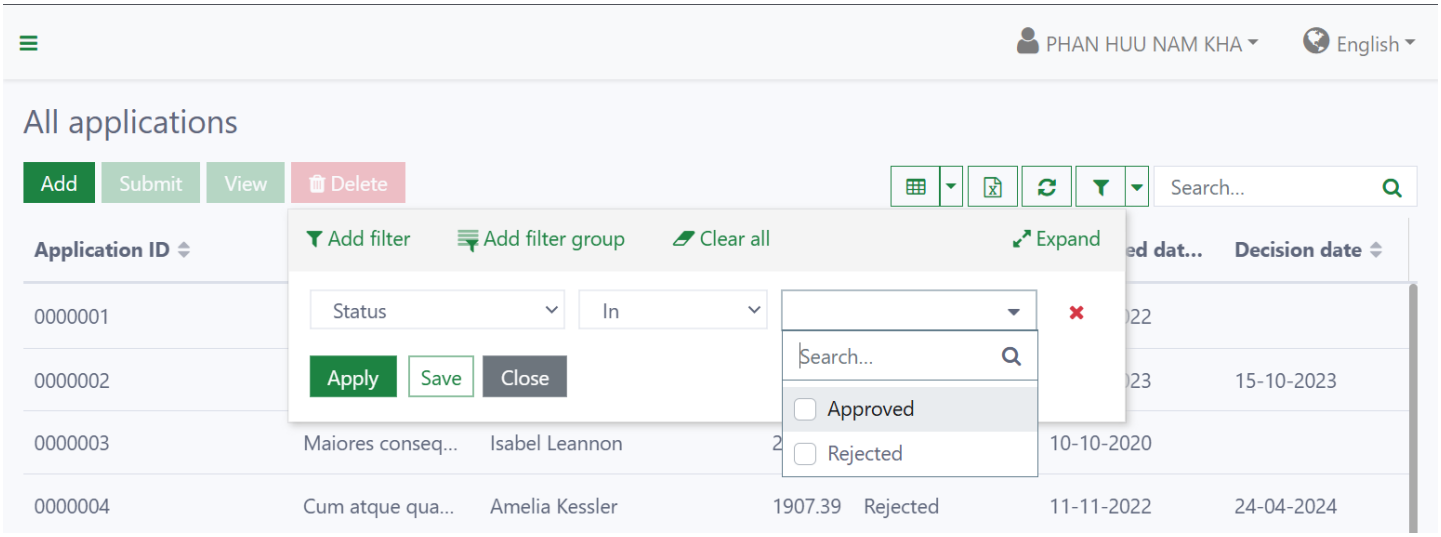
- **Title**: The widget title will only appear on worker sites if you select the **Show Panel** option. Casewhere supports text resource keys for widget titles.
- **Tag**: To ease product management.
- **Required Case Context**: The widget requires a **CaseId** parameter to function, and it shows only data belonging to that specific case. When the setting is enabled, Casewhere will auto-apply the filter `PrimaryDataObject.CaseId == Params.CaseId` to the widget's data query.
- **Widget Can Be Reused**: The reusable widgets can be found in the **Widget Gallery**, located next to the widget toolbox.
- **Show Panel**: Display the widget inside a panel.
- **Height**: You can let Casewhere auto-adjust the widget's height, or you can also specify a fixed number in pixels.
    - **Scale By**: For auto-height widgets, you can configure the widget behavior as follows:
        - **Container**: The widget's height will automatically fill its container.
        - **Content**: The widget's height will automatically resize based on its content.

To edit these information, click on the **Setting** button on the top right corner.

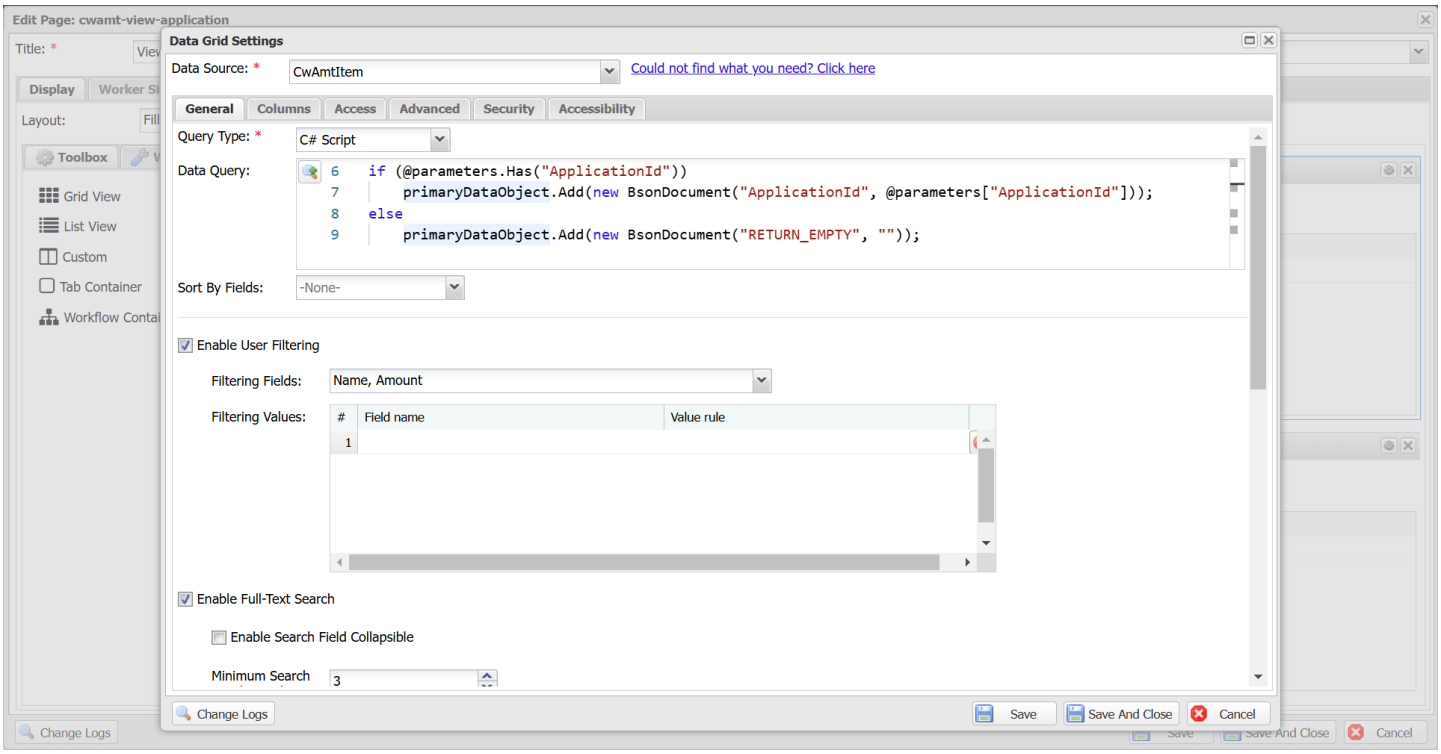


Data source settings

Widgets like Grid, List, or Custom can load and display data from a Casewhere data source. The following screenshot illustrates how data is typically displayed to users on worker sites.



The widget settings will look as follow.



Data Source: The source of data the widget will display. You can read more about data sources here.

Query Type: Casewhere supports two types of query syntaxes: **Bson** and **C#**.

Data Query: When configuring a data widget such as a grid widget or list widget, you will need to instruct Casewhere how to load data from a data source by composing a data query

Sort By Fields: The default sorting order of the widget's data. Casewhere will sort the data before returning it to the worker sites. How the sorting is handled on the worker site depends on the widget type.

Enable User Filtering: Allow end-users to filter data in the worker site.

- **Filtering Fields**: Specify which fields can be filtered.
- **Filtering Values**: In scenarios where you need to limit filtering choices, such as preventing users from selecting "Draft" in a list of submitted applications, you can define acceptable values. You must implement a rule to define the application filtering values.
    - `c# #predicate return new List<string>{ "Rejected", "Approved" };`

Enable Full-Text Search: Allow end-users to search from the worker site.

- **Enable Search Field Collapsible**: The search box can automatically collapse and expand, helping to save space on narrow screens.
- **Minimum Search Word Length**: Specify the minimum length required for users to perform a search. Searching with very short words might result in too much unnecessary data.

Enable Refresh: Allow end-users to refresh the list without refresh the entire page.

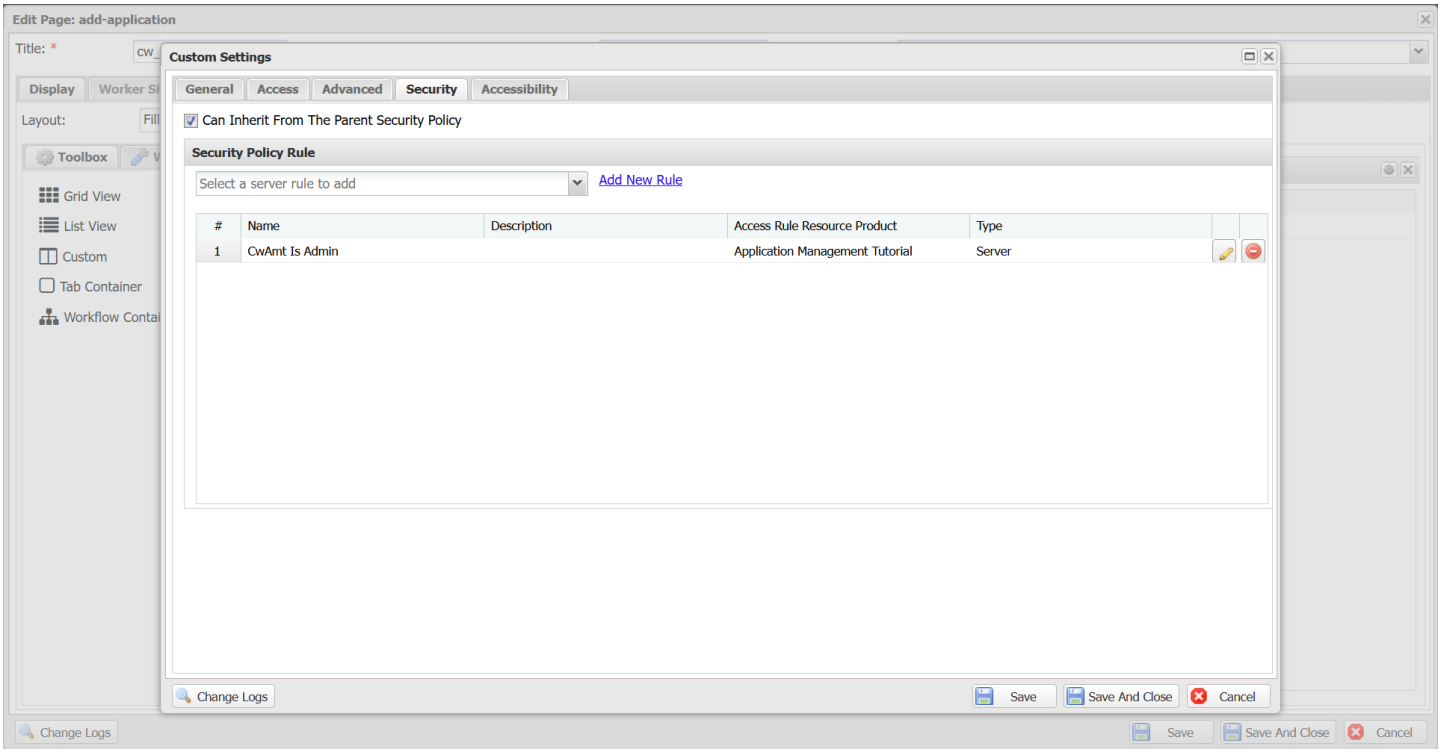
Export to Excel: Allow end-users to export data to an Excel file.

- **Export Limit**: The maximum number of records that can be exported. Ideally, avoid exporting more than 100,000 records.
- **Export Buffer**: The number of records to load at a time. If your data objects are small, increasing the buffer size can speed up the export process.
- **Method**: Determine where the Excel file will be generated: **Server** or **Client**. For optimal performance, it is recommended to use **Server**, which has been the default option since version 2.8.

Empty Message: The message displayed when no data is available to show.

Security

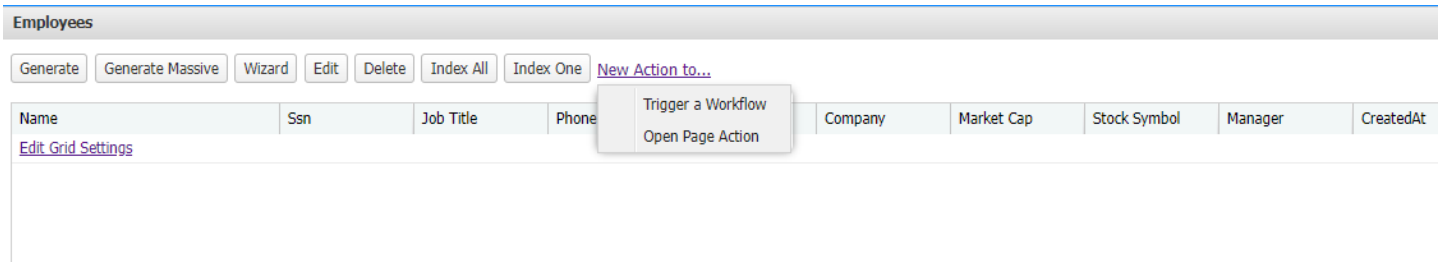
Like other resources, you can control who can see the widget by assigning appropriate access rules.



ypically, a widget should inherit security rules from its parent element, which is the page. This can be achieved by selecting **'Can Inherit From The Parent Security Policy'**, which is enabled by default.

Widget actions

You can have your end-users interact with the widget data using widget actions. Currently, Casewhere supports two types of actions: **Trigger workflow** and **Open page**.

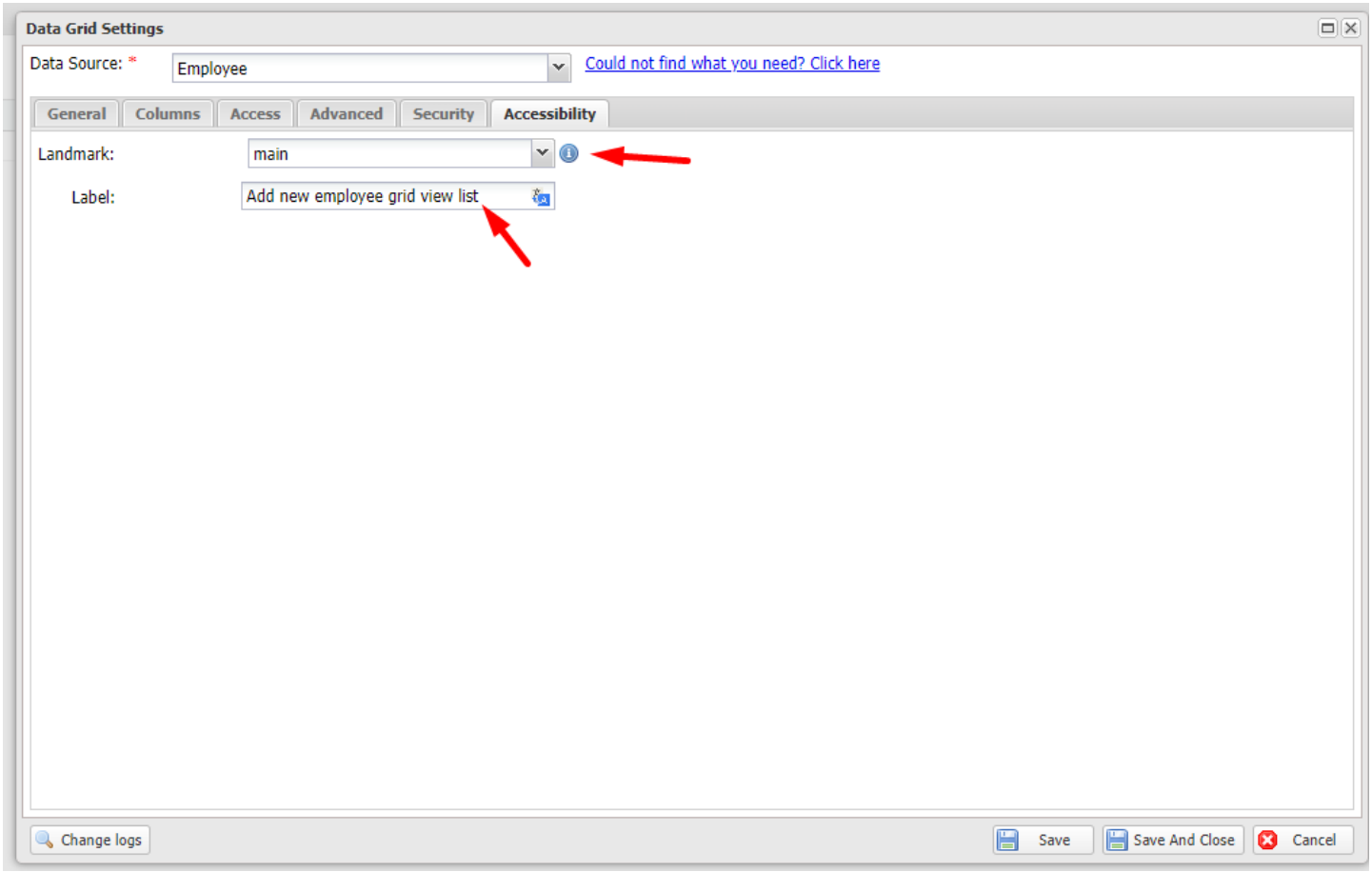


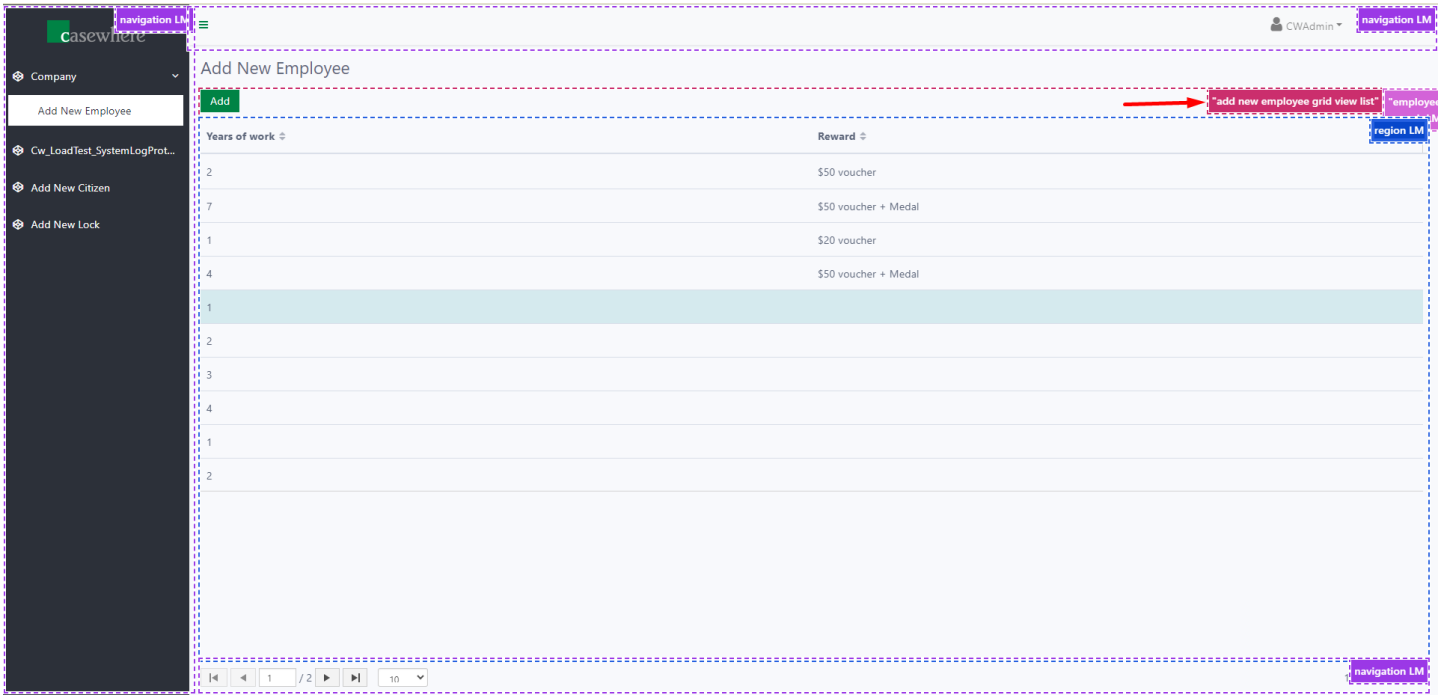
Read here to learn more.

Accessibility

The following settings can be found in the **Accessibility** tab:

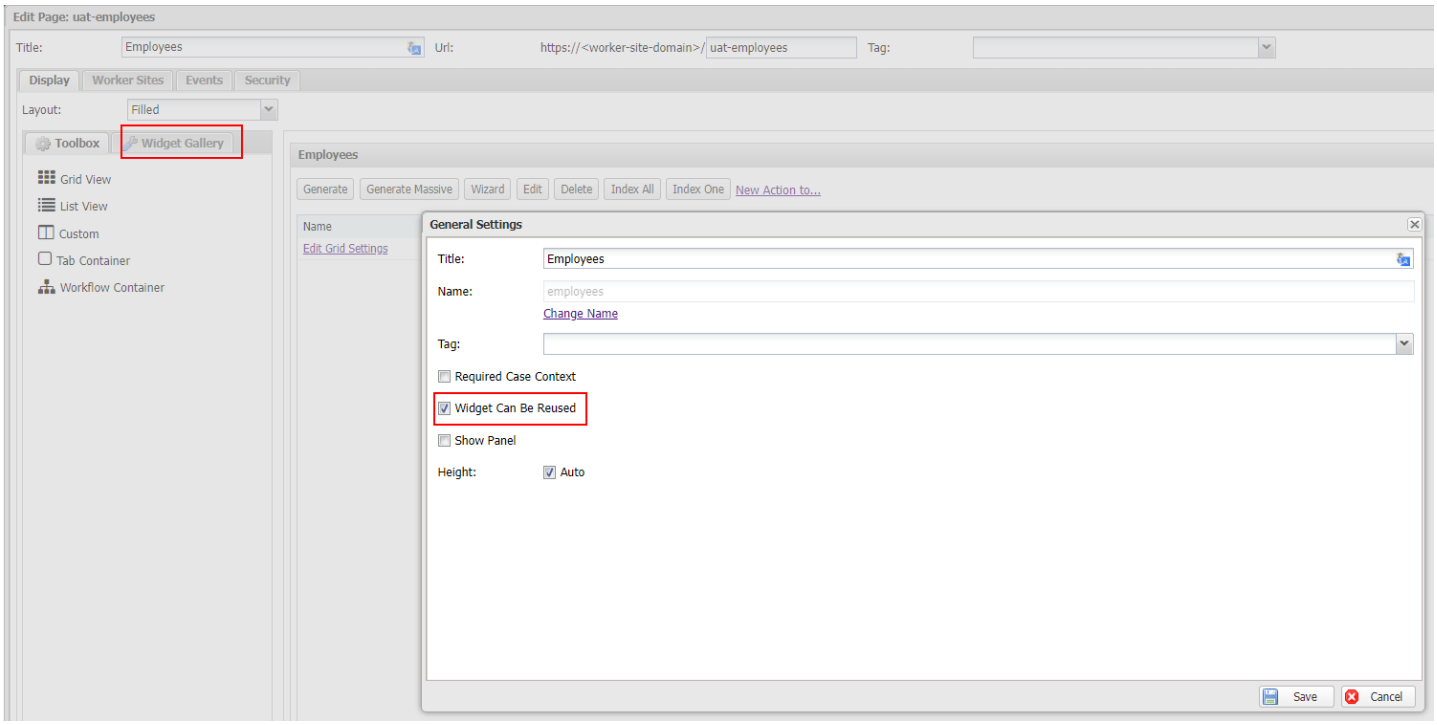
- **Landmark**: Assign the landmark type to the current grid view. The grid view can be detected as a landmark by the Narrator.
- **Label**: The label of the grid view. A small description to describe the grid view for Narrator users.



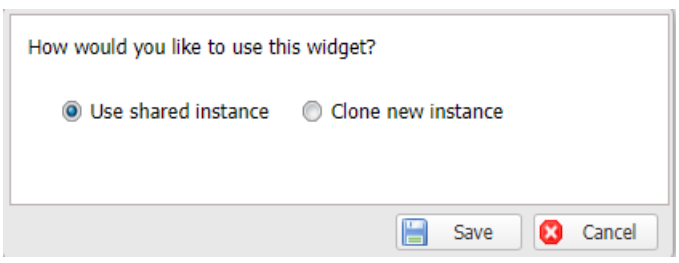


Reuse widgets

When the setting **Widget Can Be Reused** is enabled, the widget will appear in the **Widget Gallery**.



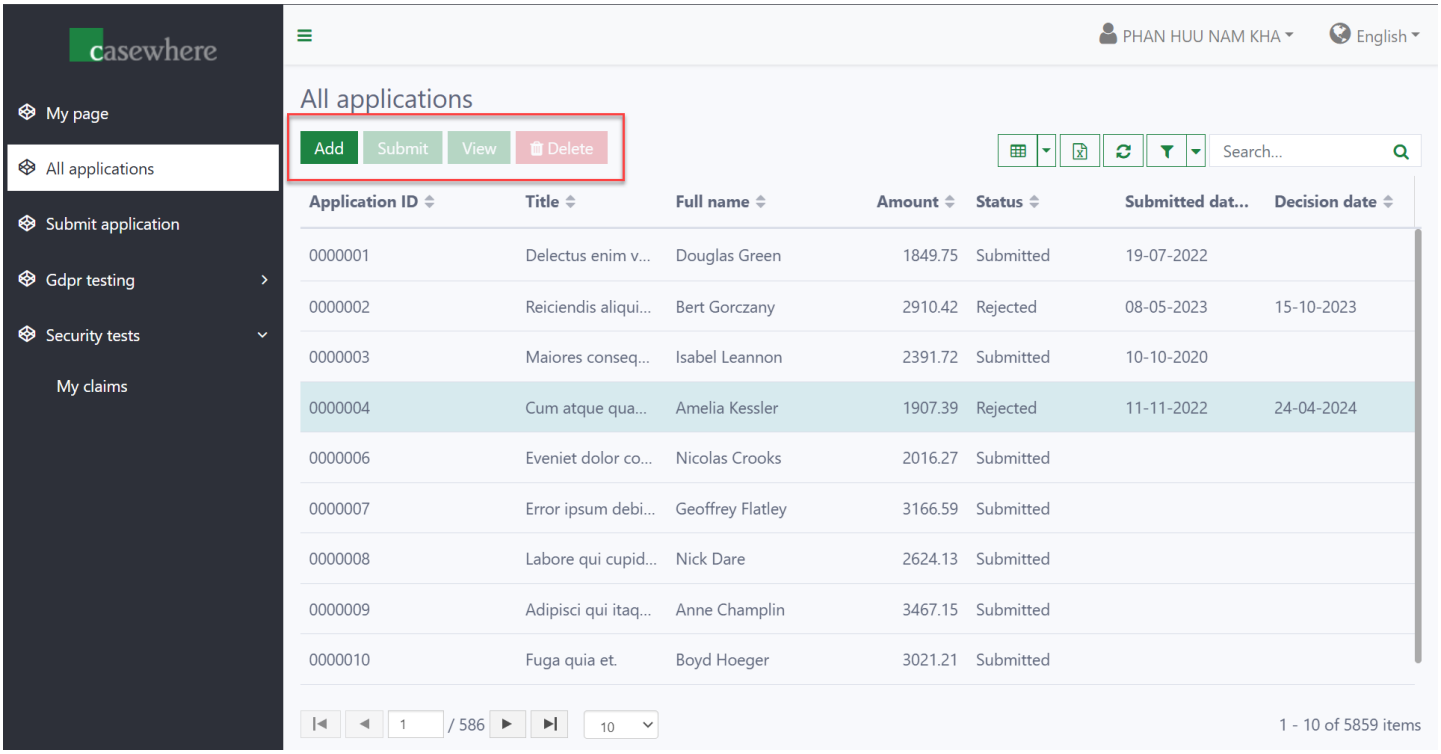
When designing pages, you can drag and drop a widget from the **Widget Gallery**. Casewhere will ask if you want to reuse the same widget instance or clone a new one.



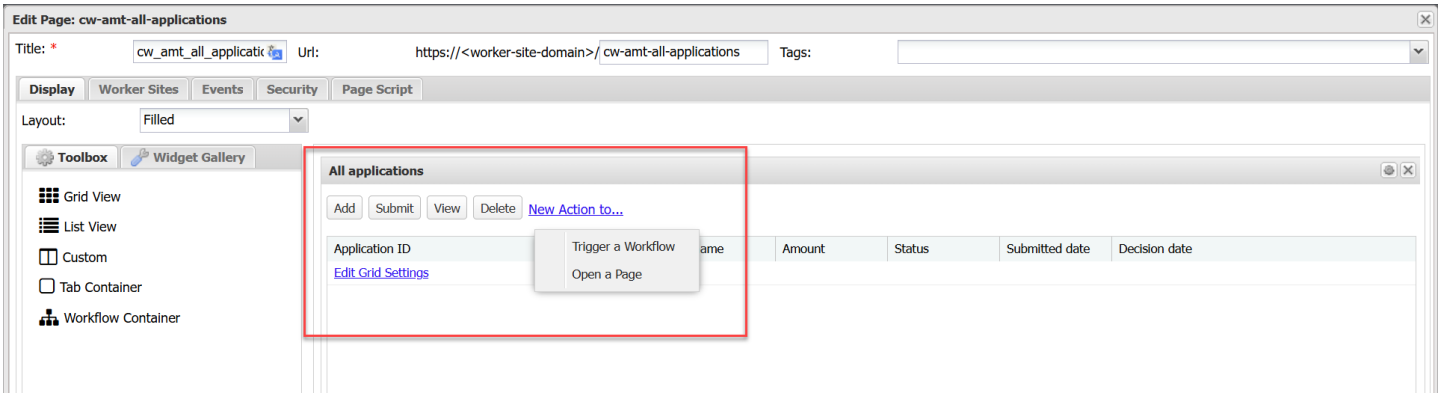
Widget action

While viewing data in Casewhere widgets, end-users may need to interact with the data, such as editing a draft, submitting an application, or approving a budget. These functionalities will be implemented as widget actions in Casewhere.

The screenshot below illustrates how widget actions appear to end users on worker sites.



In Casewhere Admin, the widget actions will appear to developers as shown below. As illustrated, there are two types of widget actions, **Trigger a Workflow** and **Open a Page**.

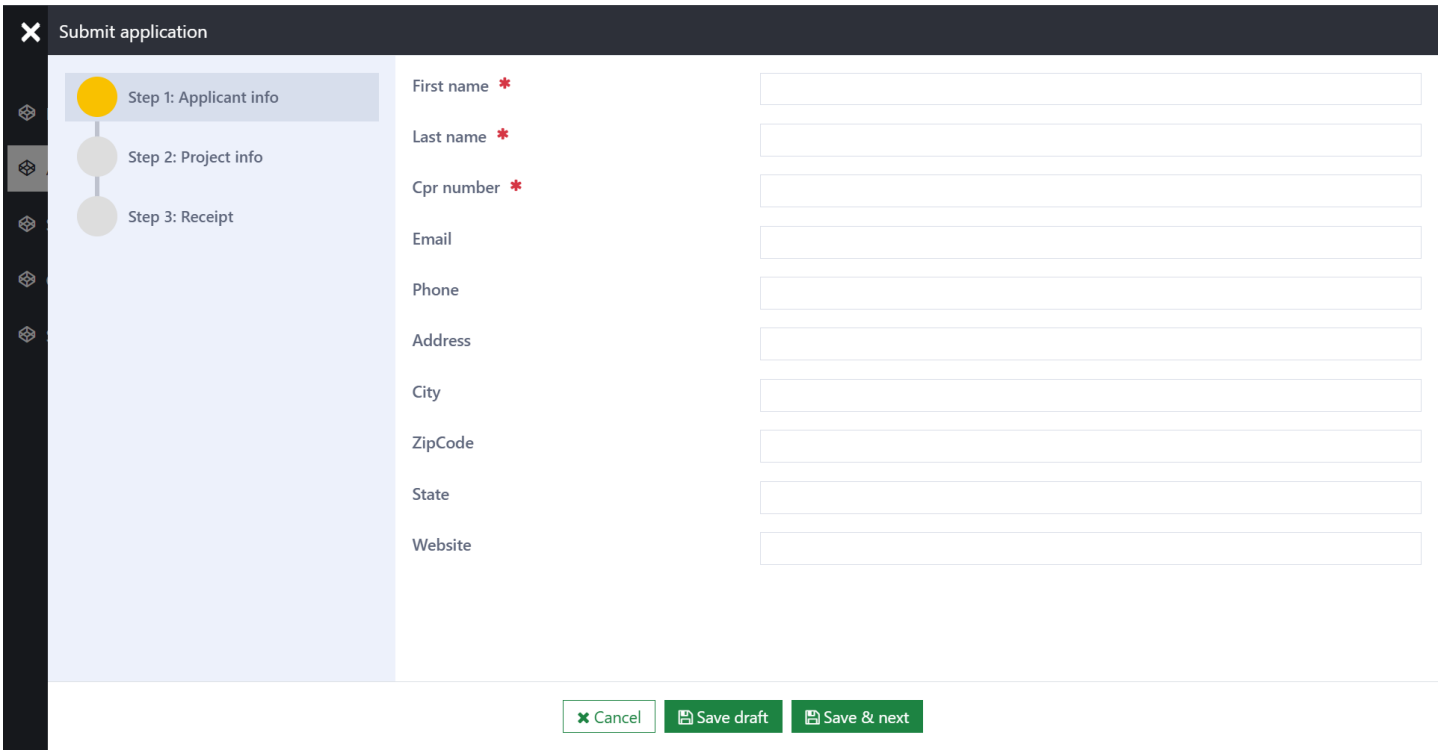


Settings for widget actions are organized into three main areas:

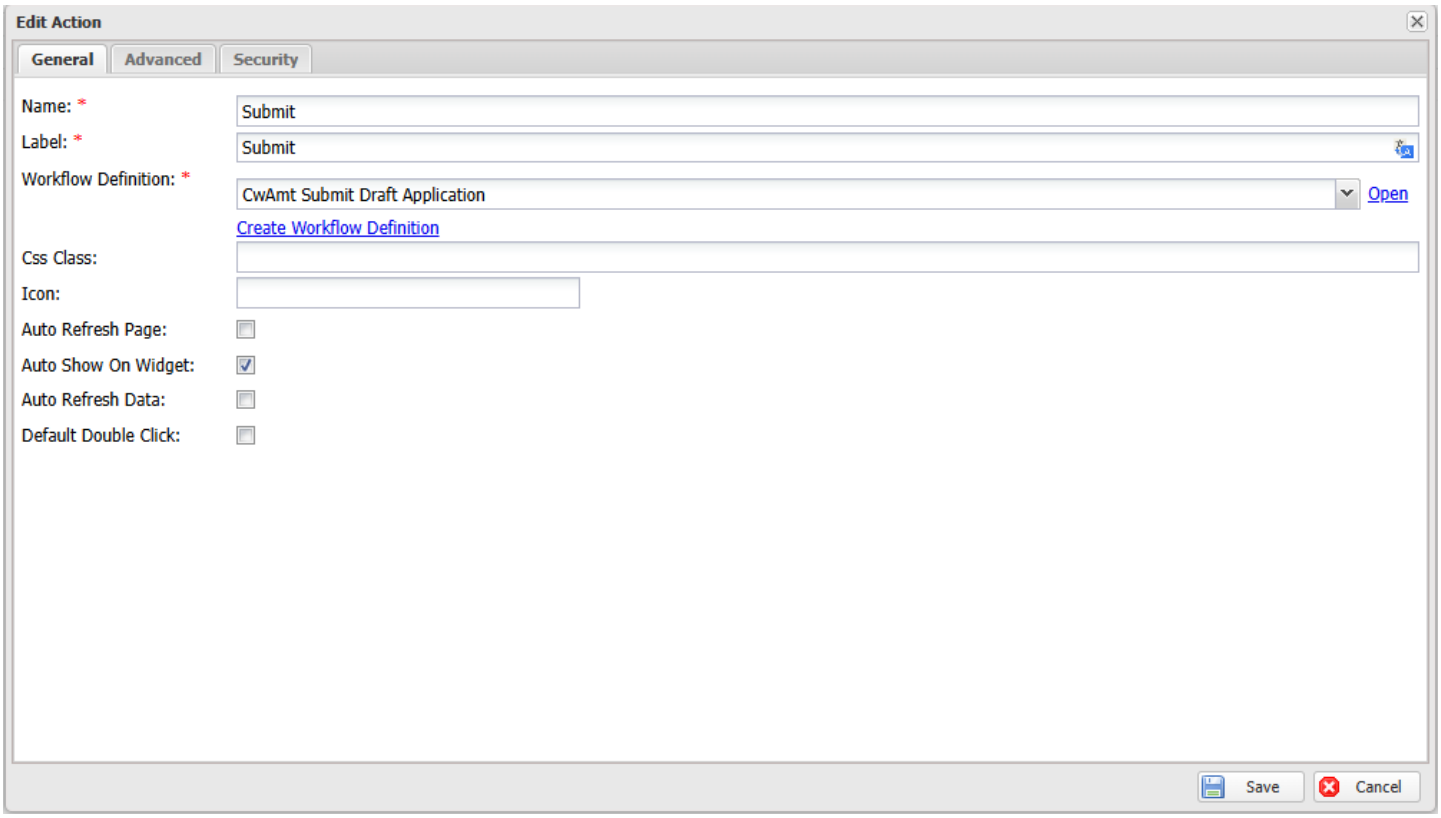
- **General**: Includes key attributes, triggered workflows, appearance, and other general settings.
- **Advanced**: Covers advanced behaviors and rules, often requiring scripting.
- **Security**: Defines security policies to determine who can view and execute the actions.

Trigger-workflow action

The action triggers a workflow and opens it in a dialog. The screenshot below illustrates how a workflow is triggered and loaded on the worker site.



The action settings look as follow.



NAME	MANDATORY	DESCRIPTION
Name	Yes	The unique name referring the action.
Label	Yes	The button label is displayed to end users and accepts a text resource key.
Workflow Definition	Yes	The workflow triggered when the action is executed. If the workflow type is 'Case Creation', a process is also required.

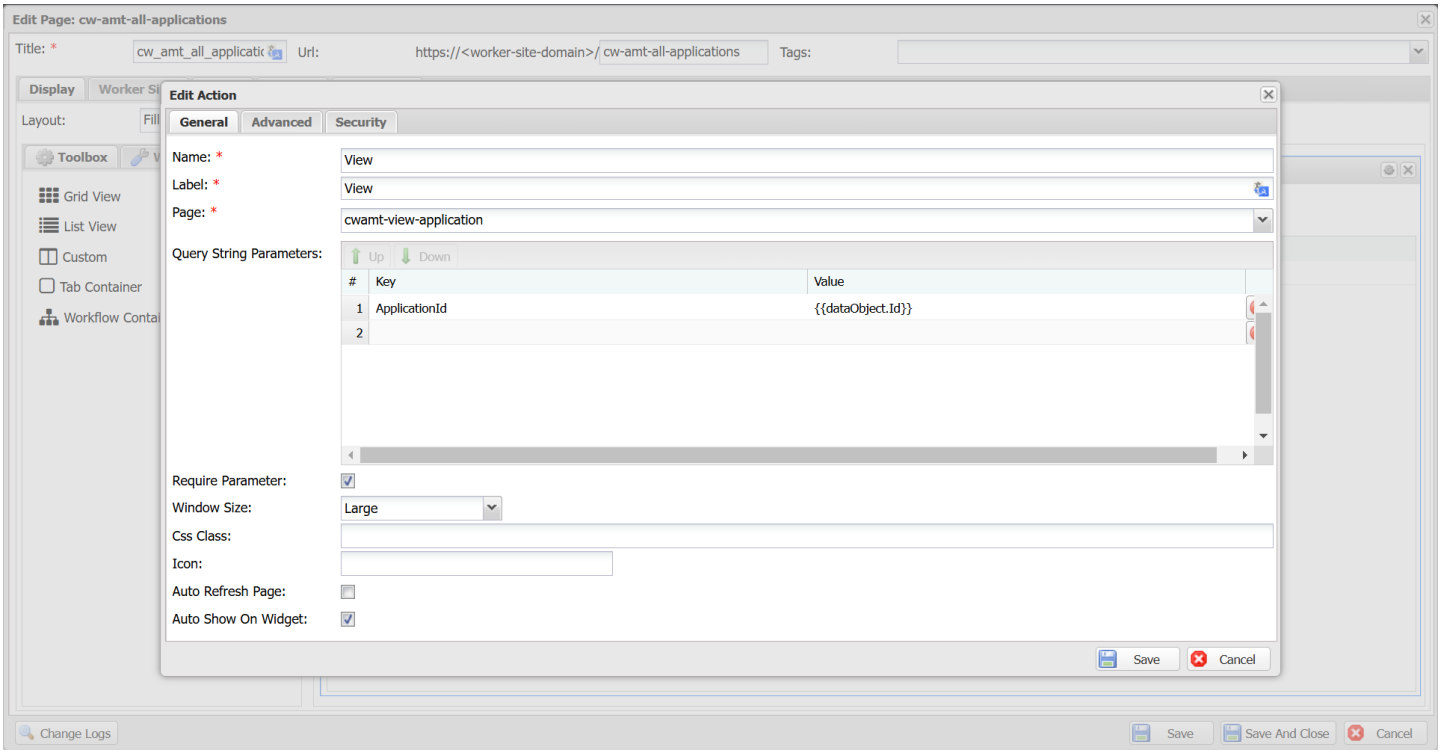
NAME	MANDATORY	DESCRIPTION
Css Class	No	To decorate the button, for example, the "Delete" button can be highlighted using `btn-danger`.
Icon	No	To decorate the button, for example, the icon for a "Delete" button could be `fa fa-trash`.
Auto Refresh Page	No	When selected, Casewhere will automatically refresh the page data, including all widgets within it. This is useful when the action causes changes that need to be reflected across all areas of the page.
Auto Show on Widget	No	By default, the action button is visible. However, if the setting is **not** selected, the button will be hidden, though the action can still be triggered via scripting.
Auto Refresh Data	No	When selected, the action will be triggered when users double-click on a widget data item.

Open-page action

The action will open a new page inside a dialog. The screenshot below illustrates how a page is loaded on the worker site.

TBD

The action settings look as follow.

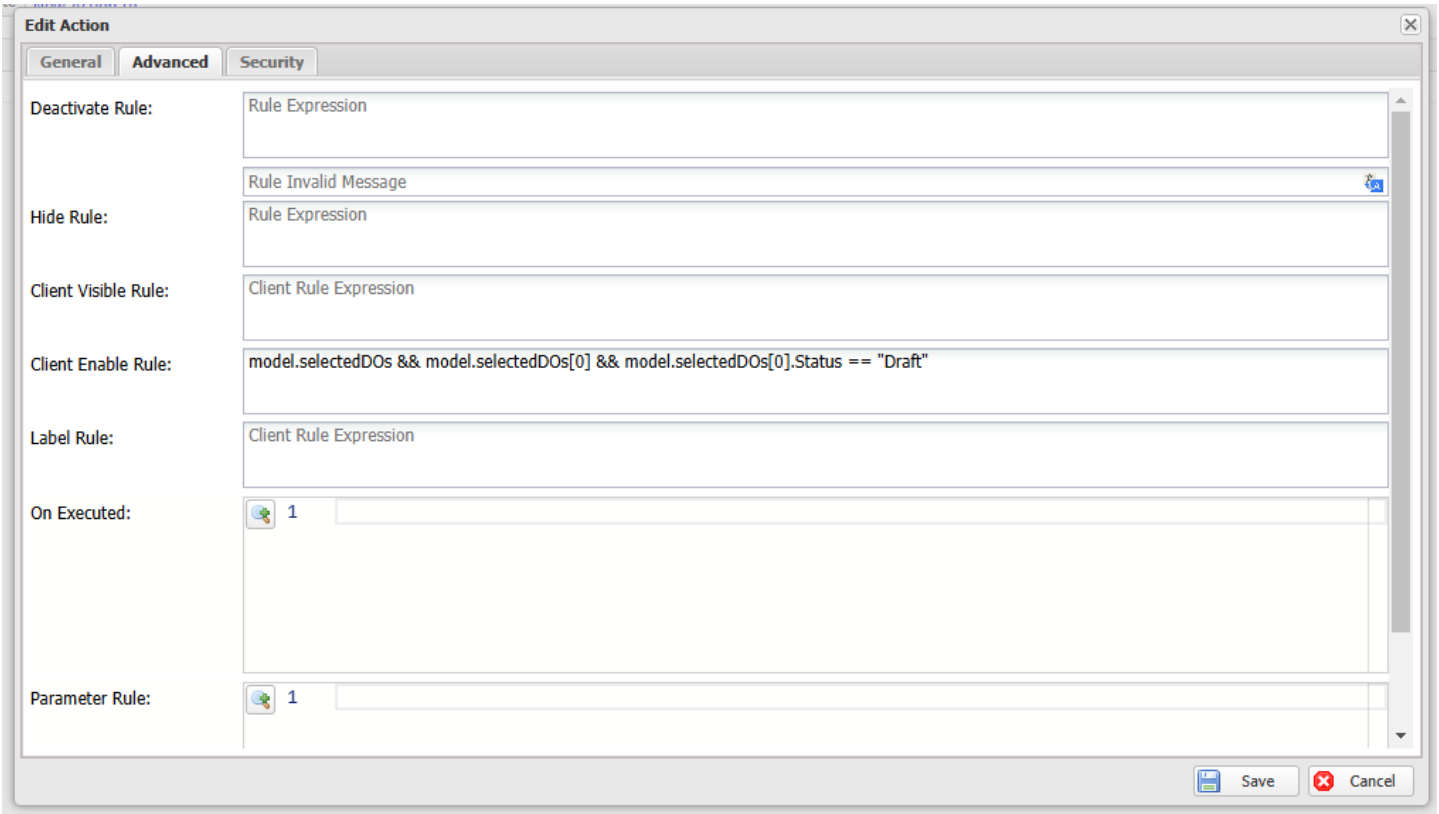


NAME	MANDATORY	DESCRIPTION
Name	Yes	The unique name referring the action.
Label	Yes	The button label is displayed to end users and accepts a text resource key.
Page	Yes	The page to be opened.

NAME	MANDATORY	DESCRIPTION
Query String Parameters	No	A list of key-value pair parameters sent to the page for data loading. The key is used to access the parameters in widget data queries. The value supports a JavaScript expression, such as `dataObject.Id`.
Require Parameter	No	When selected, Casewhere requires parameters to trigger the action. This means end-users must select a data item on the widget to activate the action.
Window Size	No	Large, medium, or small.
Css Class	No	To decorate the button, for example, the "Delete" button can be highlighted using `btn-danger`.
Icon	No	To decorate the button, for example, the icon for a "Delete" button could be `fa fa-trash`.
Auto Refresh Page	No	When selected, Casewhere will automatically refresh the page data, including all widgets within it. This is useful when the action causes changes that need to be reflected across all areas of the page.
Auto Show on Widget	No	By default, the action button is visible. However, if the setting is **not** selected, the button will be hidden, though the action can still be triggered via scripting.

Advanced settings

The advanced settings look as follow.



NAME	DESCRIPTION
Deactivate Rule `deprecated`	This setting controls whether the action is enabled. Note that this action is deprecated; you should use the **Security** function instead.

NAME	DESCRIPTION
Hide Rule `deprecated`	This setting determines whether the action is visible. Note that this action is deprecated; you should use the **Security** function instead.
Client Visible Rule	A JavaScript expression used to determine if the action is visible or not.
Client Enable Rule	A JavaScript expression used to determine if the action is enabled or not.
Label Rule	A JavaScript expression used to dynamically set the button label.
On Executed	A callback triggered when the triggered workflow finishes, and the workflow or page dialog is closed.
Parameter Rule	To send additional data to the triggered workflows.

Example 1: The button will be visible when users select a data object.

Use the following expression for the **Client Visible Rule**.

```
model.selectedDOs && model.selectedDOs.length > 0
```

Example 2: The "Delete" button will be enabled when users select a "Draft" application.

Use the following expression for the **Client Enable Rule**.

```
model.selectedDOs && model.selectedDOs[0] && model.selectedDOs[0].Status == "Draft"
```

Example 3: Open a viewing-data page after the creation workflow is completed.

Use the following expression for the **On Executed**.

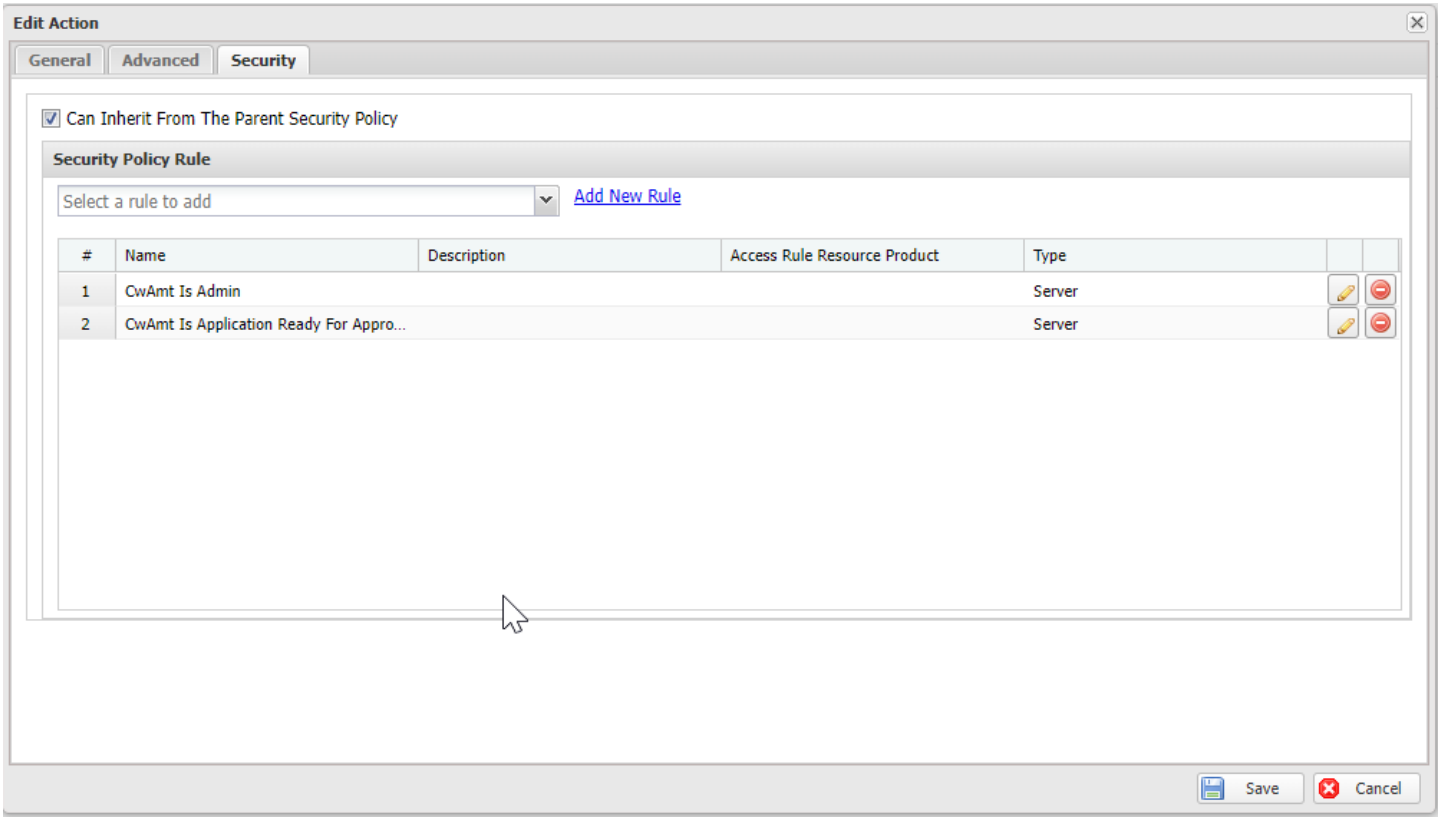
```
dataObjects && dataObjects.length && $scope.widgetApi.runWidgetAction("View", dataObjects)
```

Example 4: Set parameters for widget action, for example, add an item to a parent object.

TBD

Security

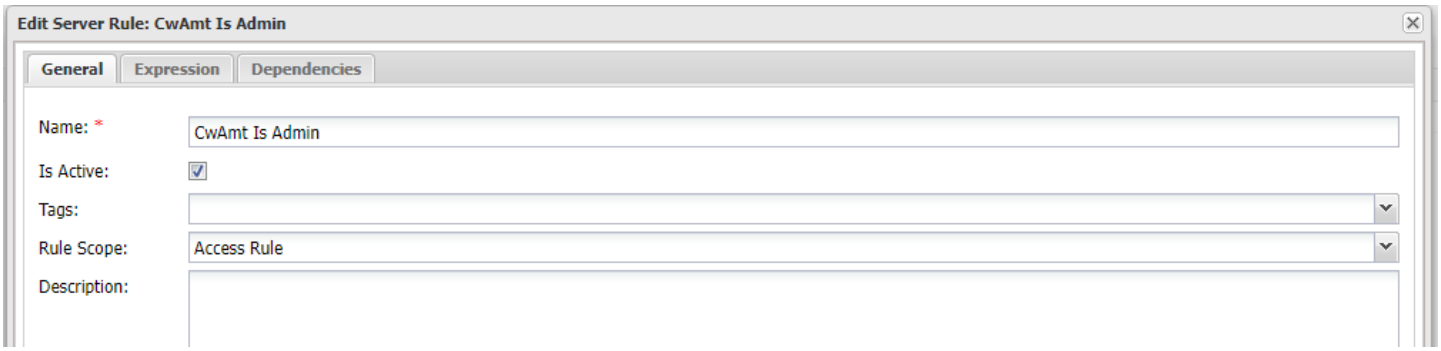
You can manage who can view and trigger widget actions by assigning the appropriate security rules, similar to how rules are applied to protect pages and widgets.



Typically, a widget action should inherit security rules from its parent elements, such as the widget and the page. This is achieved by selecting **"Can Inherit From The Parent Security Policy"**, which is also enabled by default.

Security rules for widget actions have two scopes. The following examples will demonstrate how each scope can be used:

Example 1: Access rules to determine if the widget action should be visible or not.




```
#predicate
return @user.Contains("Roles", "Administrator");
```

Example 2: Widget action rules to determine if the widget action can be executed or not.



```
#predicate

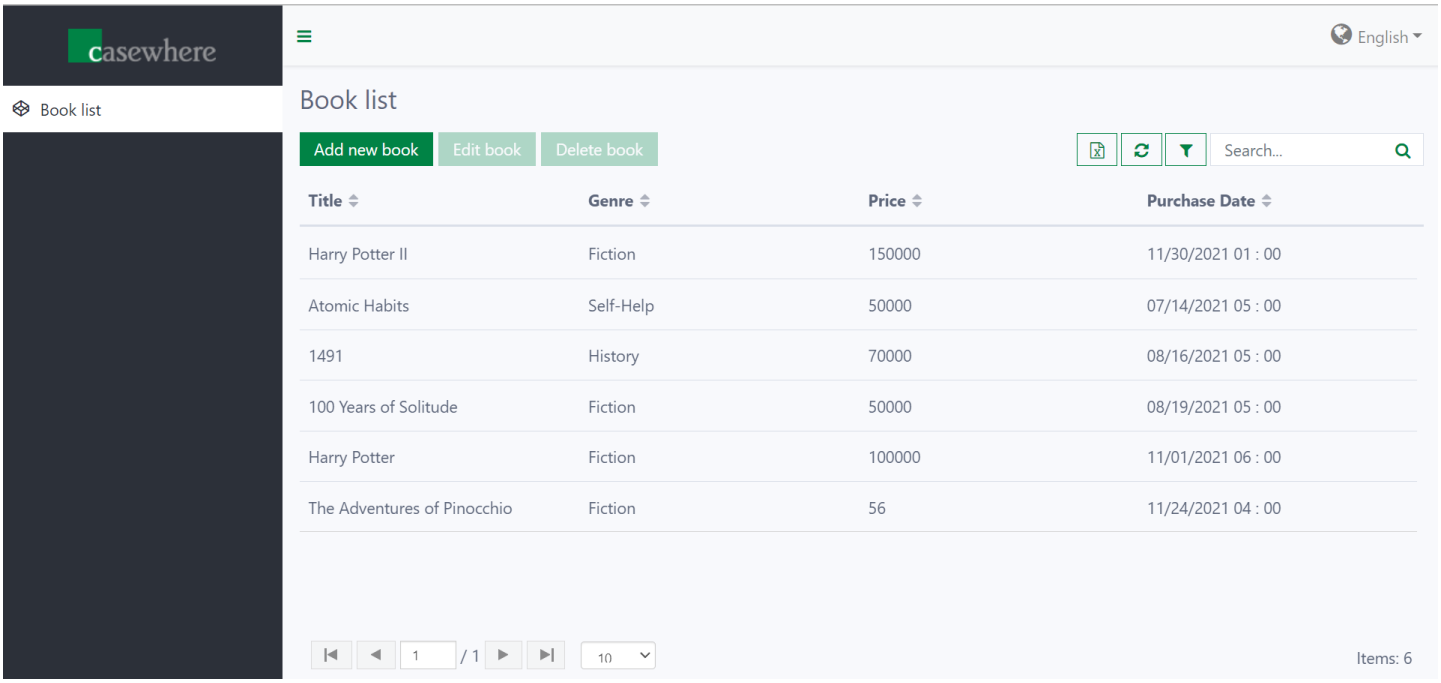
var dataApi = @apiFactory.Get<IDataApi>();
var applicationId = @parameters["DOIds"][0];
var application = dataApi.Load(applicationId);

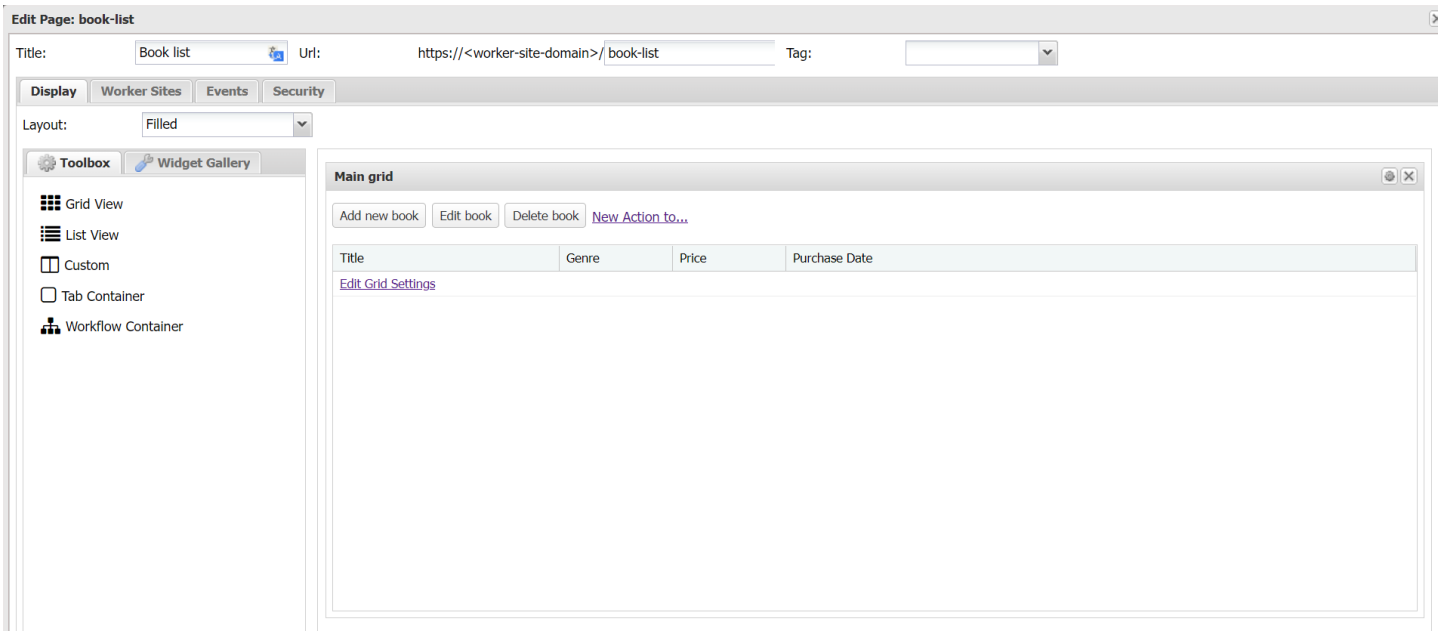
return application["Status"] == "Submitted";
```

Grid widget

Grid widgets are designed to target tabular-structure data. Data is fetched from a data source and displayed in a table. Each row represents a record, while a column can display one or more fields.

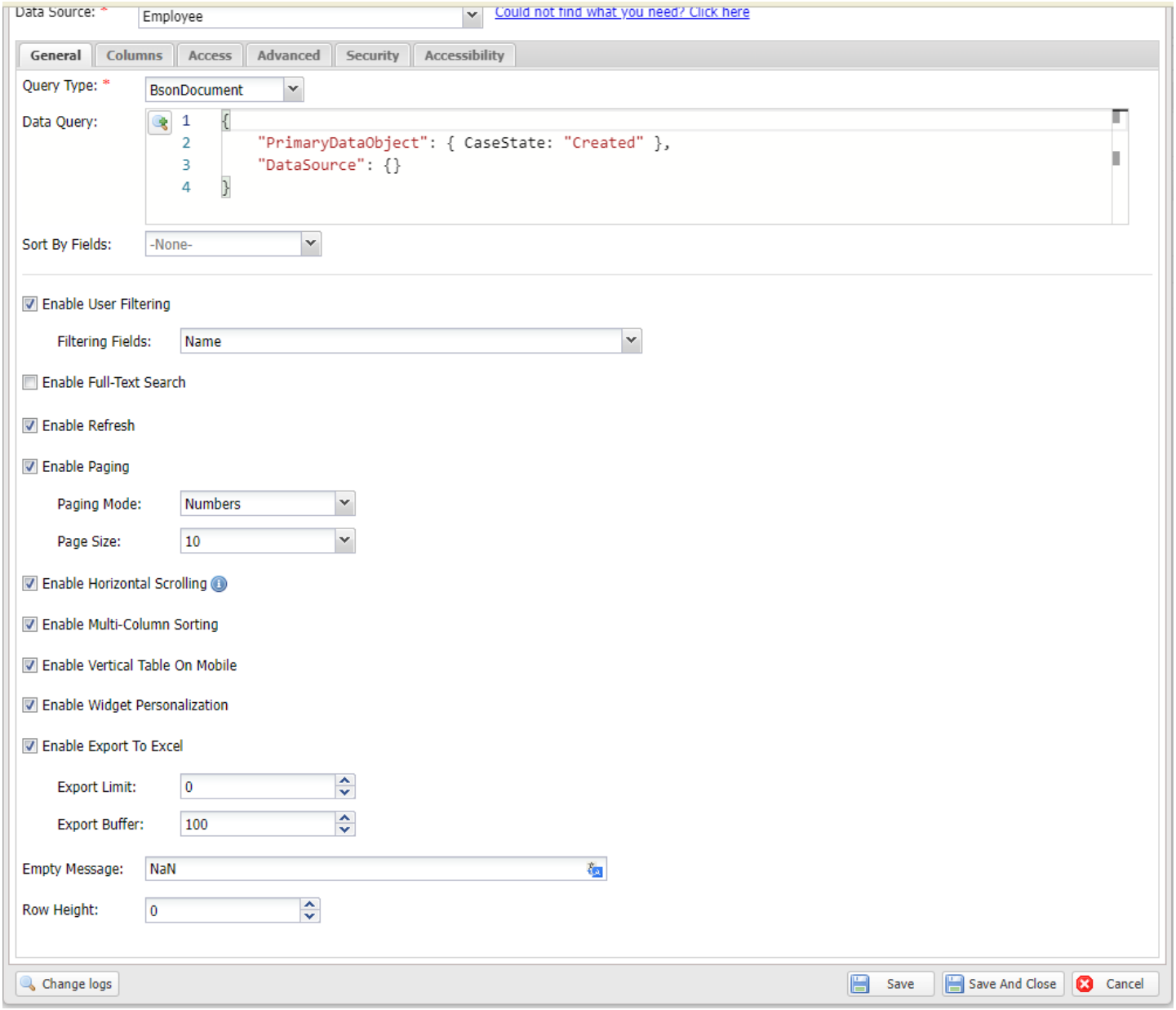
Below is how a grid widget looks in the worker site and in Casewhere Admin.





Configure grid widget

General settings



All data-source widgets will share the same general settings as below:

Data Source: The source of data the widget will display.

Sort By Fields: Which fields are used for sorting data before it is rendered in the worker site.

Enable User Filtering: Allow end-users to filter data in the worker site.

Enable Full-Text Search: Allow end-users to search from the worker site.

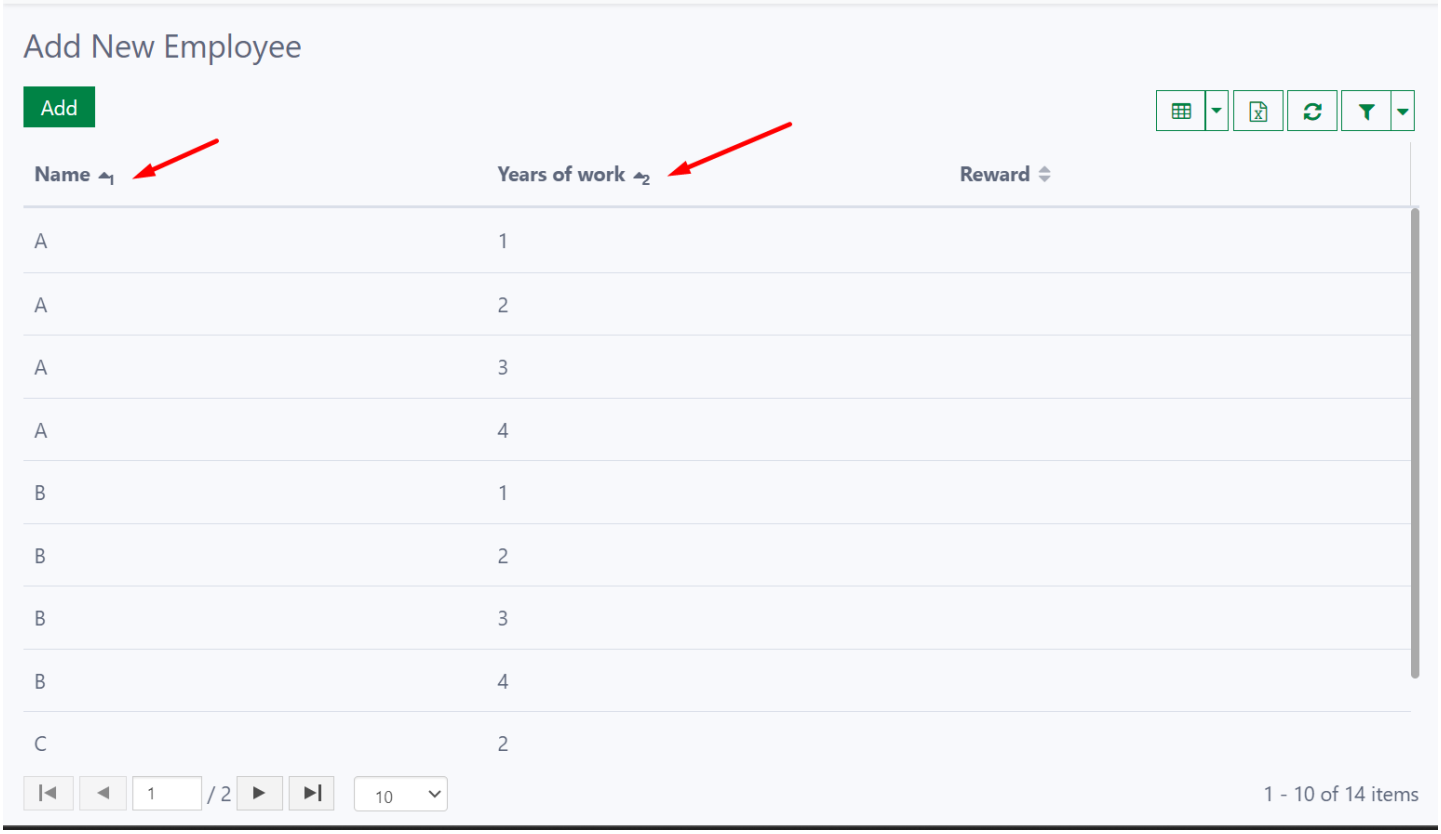
Enable Refresh: Allow end-users to refresh the list without.

Enable Paging: When enabled, Casewhere will load data page by page. For the Grid Widget and List Widget, there will be a paging control at the bottom of the widget that allows the user to navigate between pages. For Custom Widget, you have to handle the paging control yourself.

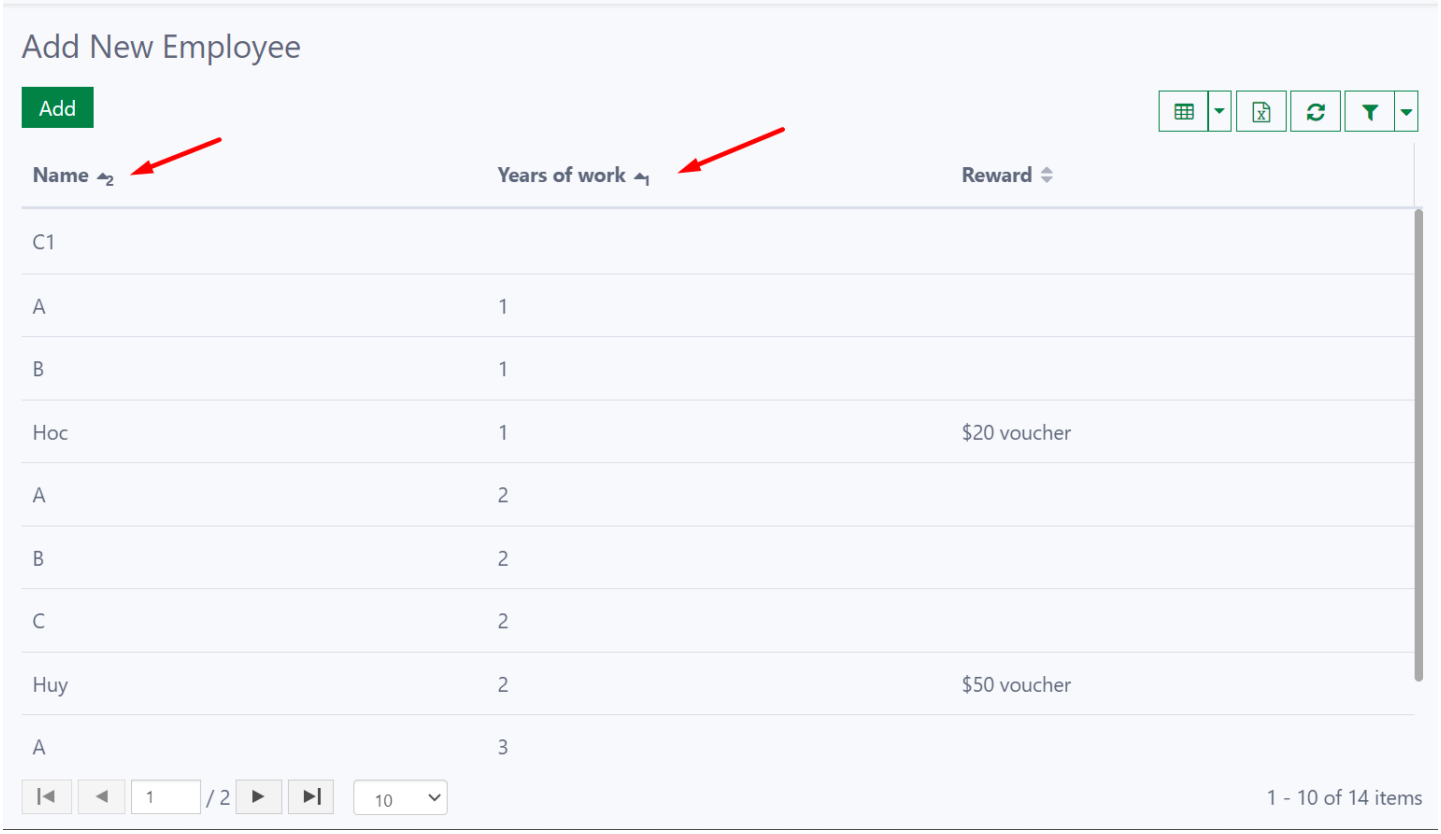
Enable Horizontal Scrolling: Only works when column min-width is set. Allow end-users to scroll horizontally.

Enable Multi-Column Sorting: This feature allows users to not only sort data based on multiple columns simultaneously but also to define the priority of each column in the sort order.

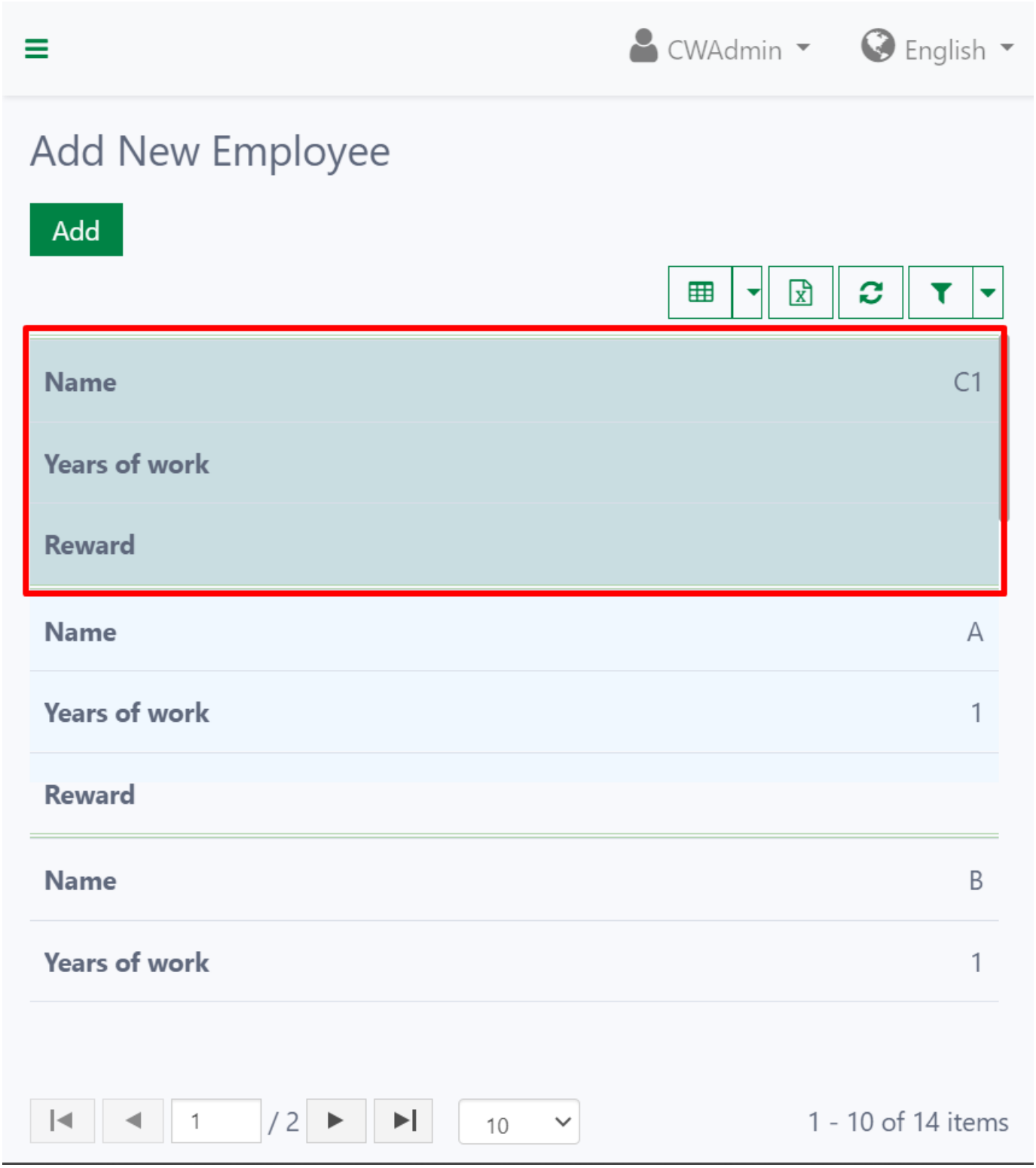
Go to the grid view list, press the "Shift" button, and click the columns to define the priority of each column in the sort order.



As you can see in the picture, the list is sorted by "Name" first, followed by "Year of Work". Now, we are attempting to reverse the order of these two columns.

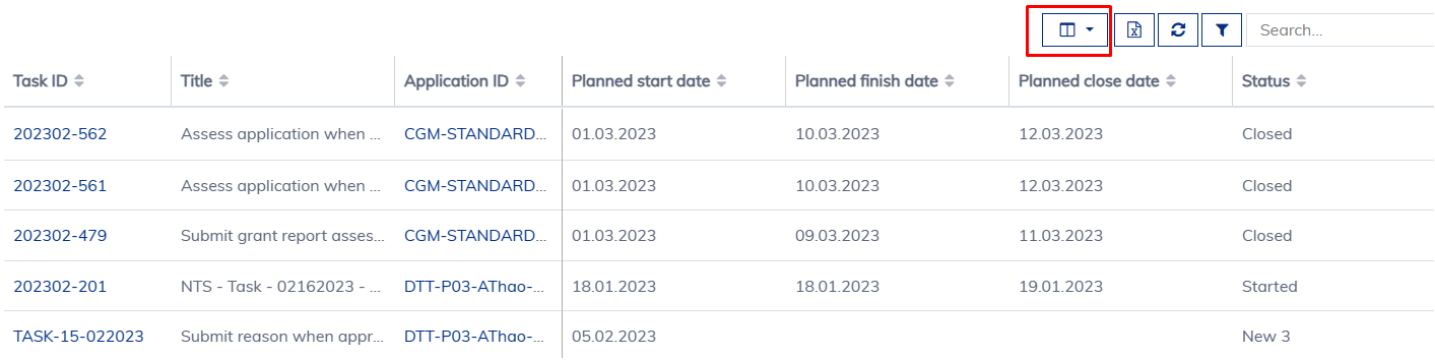


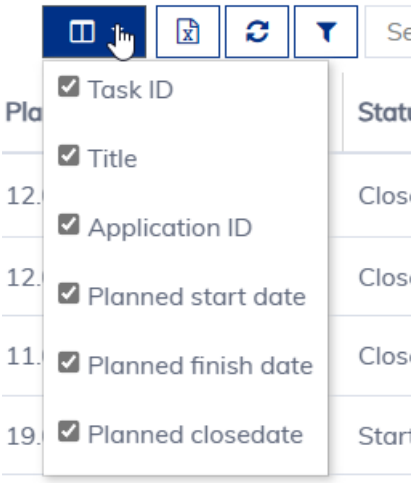
Enable Vertical Table On Mobile: The grid view list has been reconfigured to a vertical layout, specifically optimized for enhanced viewing on smaller screens and mobile devices.



Enable Widget Personalization: The end-users can change the view of the grid based on their preferences. This includes:

- Visible columns: Users can choose which columns are displayed in the grid view, optimizing the interface for their specific tasks.
- Column size: Users can adjust the width of individual columns for optimal readability and information consumption.
- Column position: Users can rearrange the order of columns within the grid view, placing the most relevant information at their fingertips.





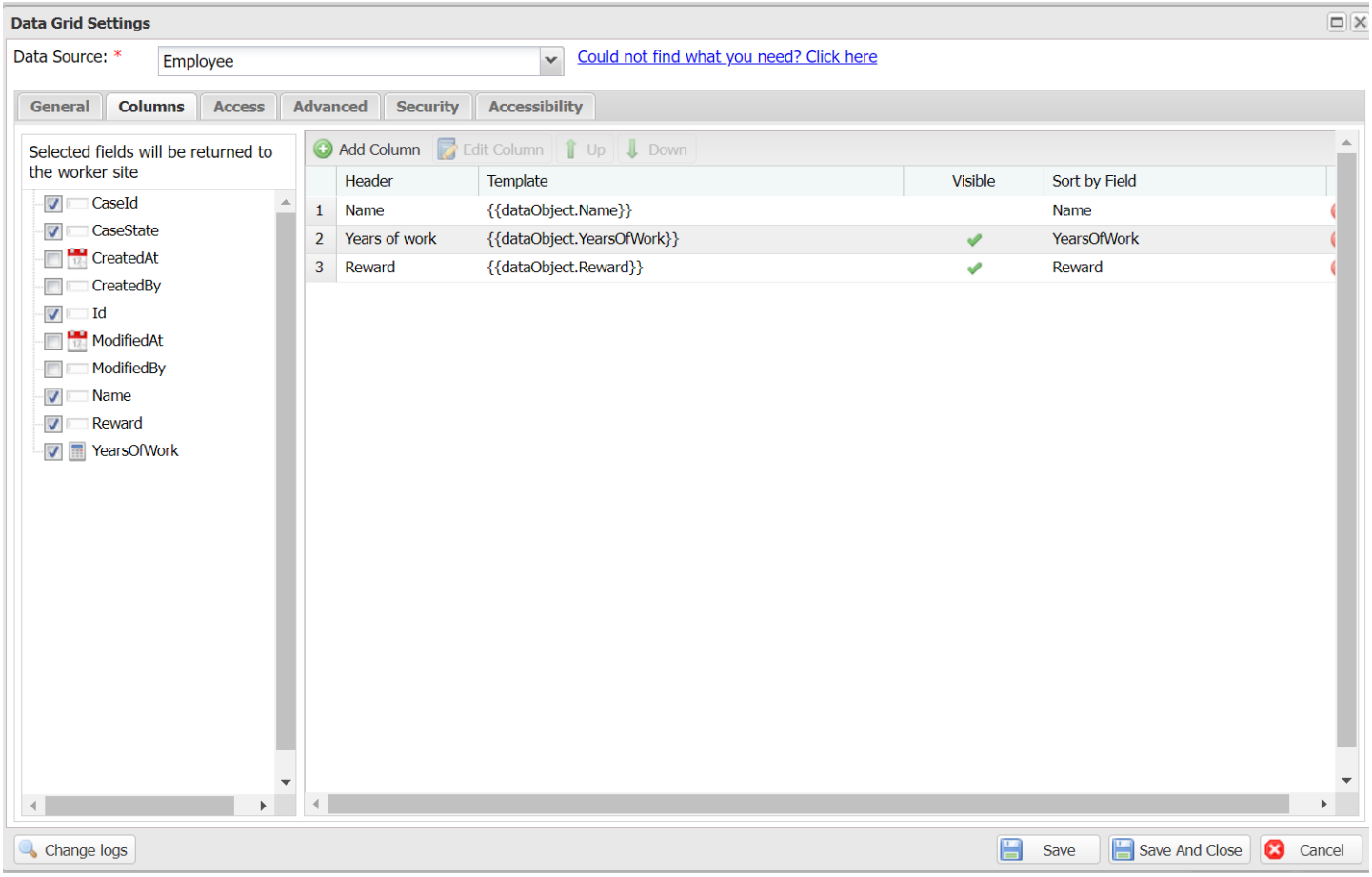
Export to Excel: Allow end-users to export data to an Excel file.

- **Export Limit**: The maximum number of records that can be exported. Ideally, avoid exporting more than 100,000 records.
- **Export Buffer**: The number of records to be loaded at a time. Increasing the buffer size will speed up the export if your data objects are small.

Empty Message: The message to be displayed when there is no data to show.

Columns

You can drag and drop fields from the left-panel to add columns for the widget.



If you have a grid column showing more than one field, make sure you select them in the field list so Casewhere will return them to the worker site.

Below are the settings for a column:

Header: The column header. Casewhere accepts a text resource key here.

Template: The display template. Example: `{{dataObject.PurchaseDate | date : 'MM/dd/yyyy hh : mm'}}`.

Export Expression: The expression to compute data for exporting. Example: `dataObject.Price - dataObject.Cost`.

Export Format: The format of data when exporting. You can select an existing format or customize it yourself.

Responsive Modes: The mode to display the column. By default, the column will be displayed in all modes.

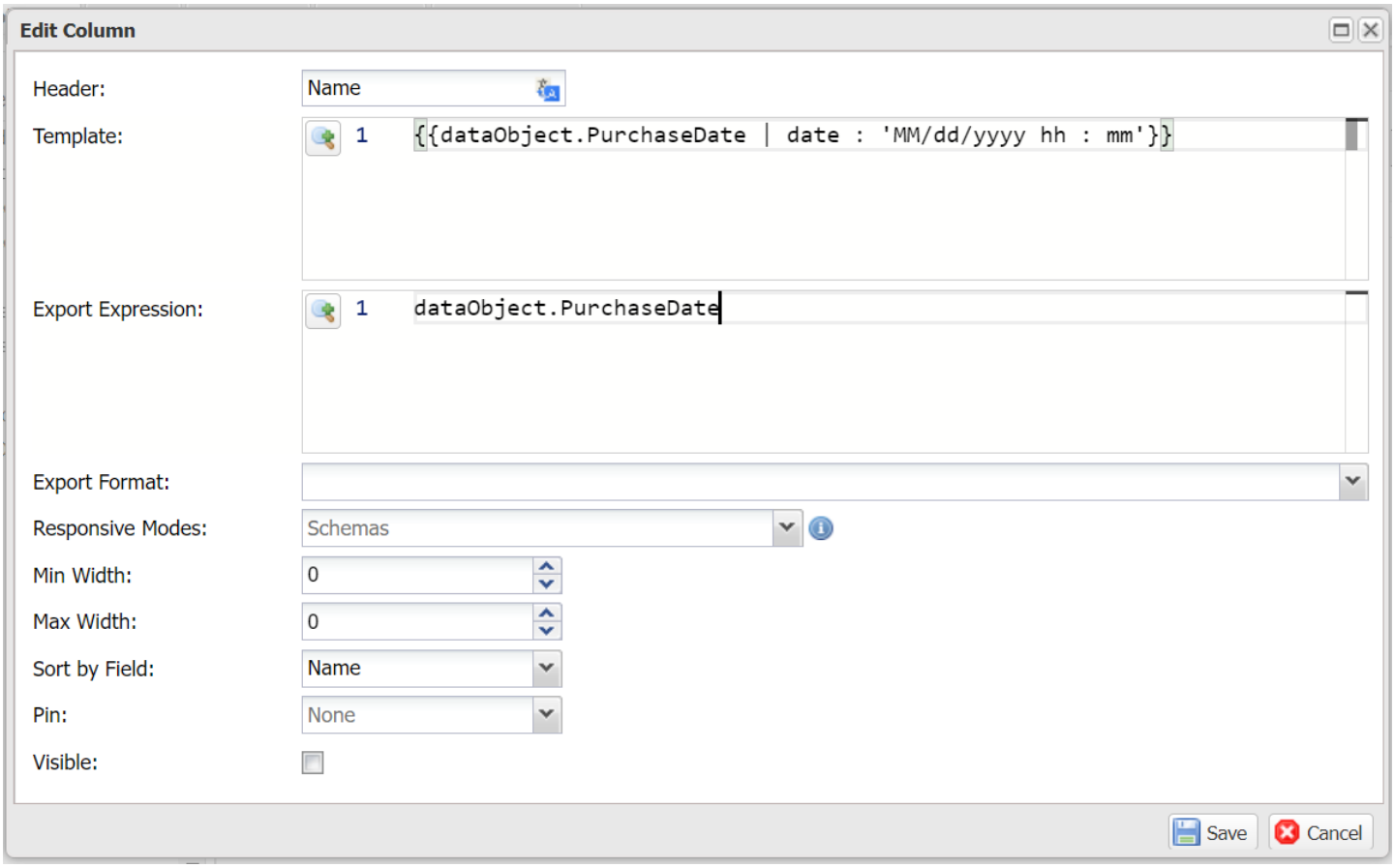
Min Width: Minimum width in pixels.

Max width: Minimum width in pixels.

Sort by Field: The field to sort when users click on the column header. Normally, the sort field is the same as the display field, but there are cases where they will be different, e.g., when your column shows more than one field.

Pin: The position to pin column. This setting is useful when your grid widget has a lot of columns.

Visible: The visibility of the column on the UI.



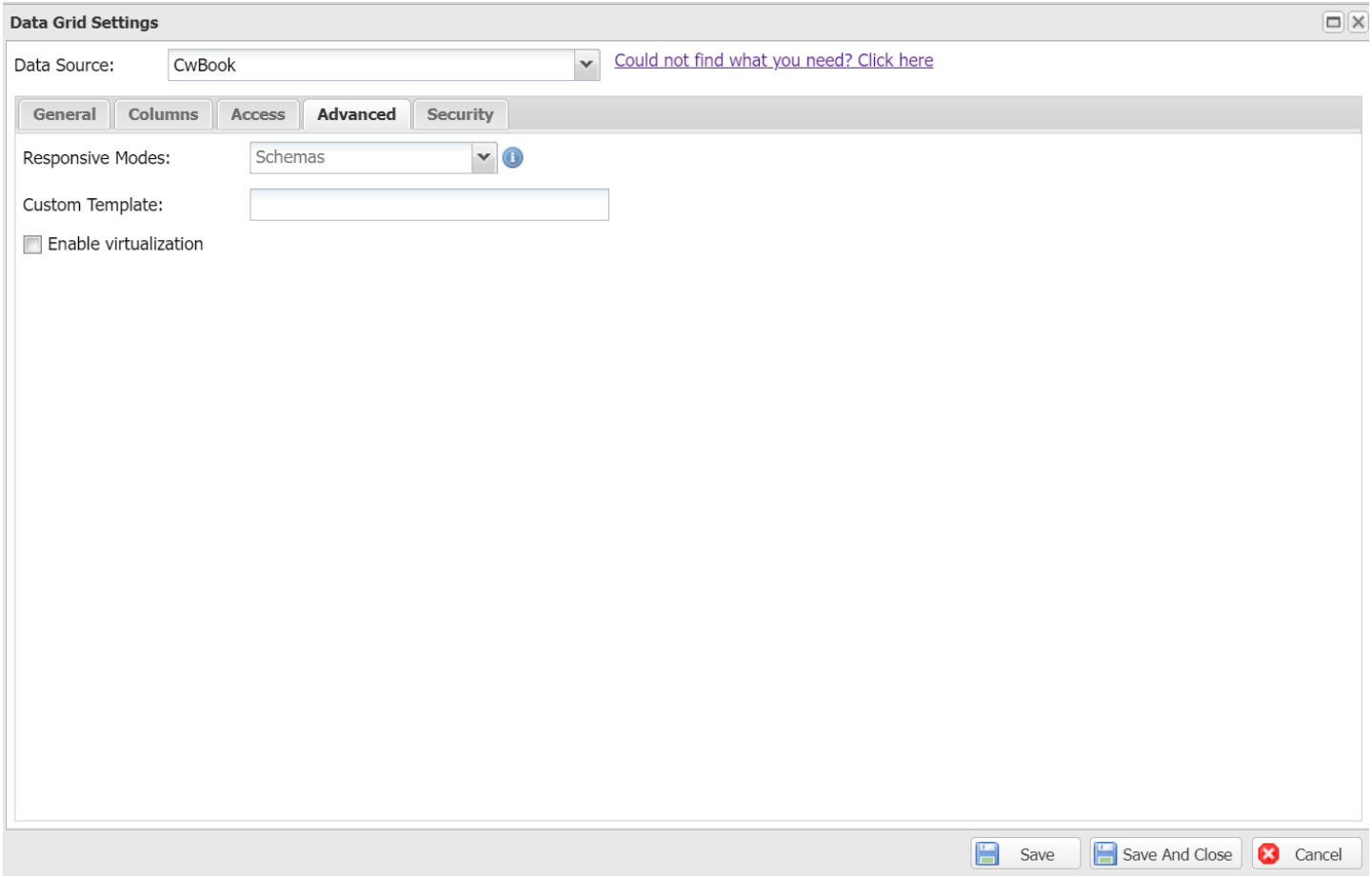
Advanced settings

The below settings are useful in advanced use cases:

- **Responsive Modes**: The device-screen sizes on which the widget will appear. None means the widget will appear on all screen sizes.
- **Custom template**: Use this setting when you want to customize the grid row template. You can define the template in a worker site component and input the template id here. Below is the default template:

```
<div ng-dblclick="grid.appScope.$parent.model.doubleClickHandler(row)"
     ng-repeat="col in colContainer.renderedColumns track by col.colDef.name"
     ng-mouseover="grid.appScope.hoveredIndex = rowRenderIndex"
     ng-mouseleave="grid.appScope.hoveredIndex = null"
     ui-grid-cell
     ng-class="grid.options.rowStyle(row, grid, rowRenderIndex)">
</div>
```

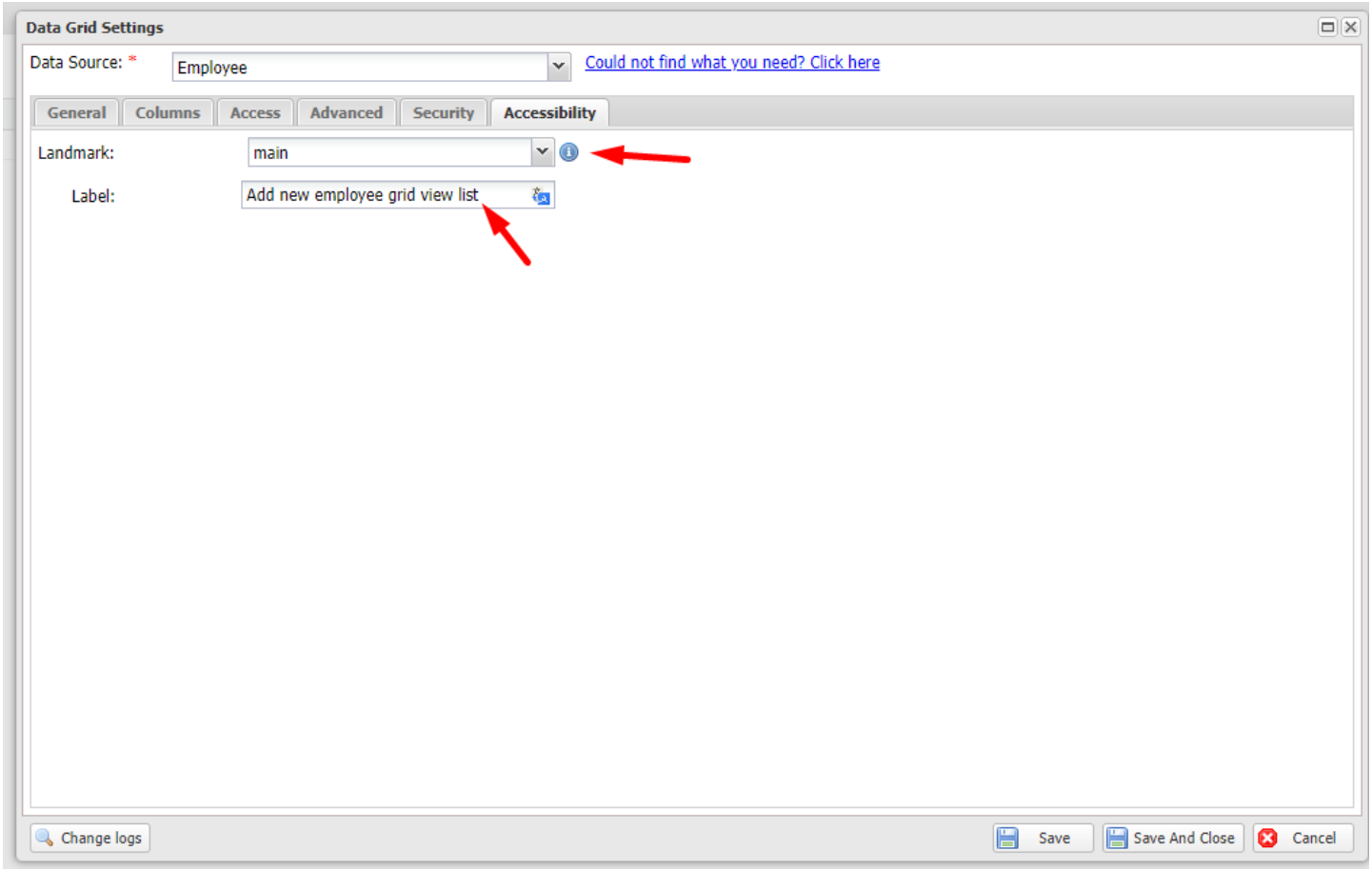
- **Enable virtualization**: Use this setting when you need to display thoudsands of records at the same time without paging data.

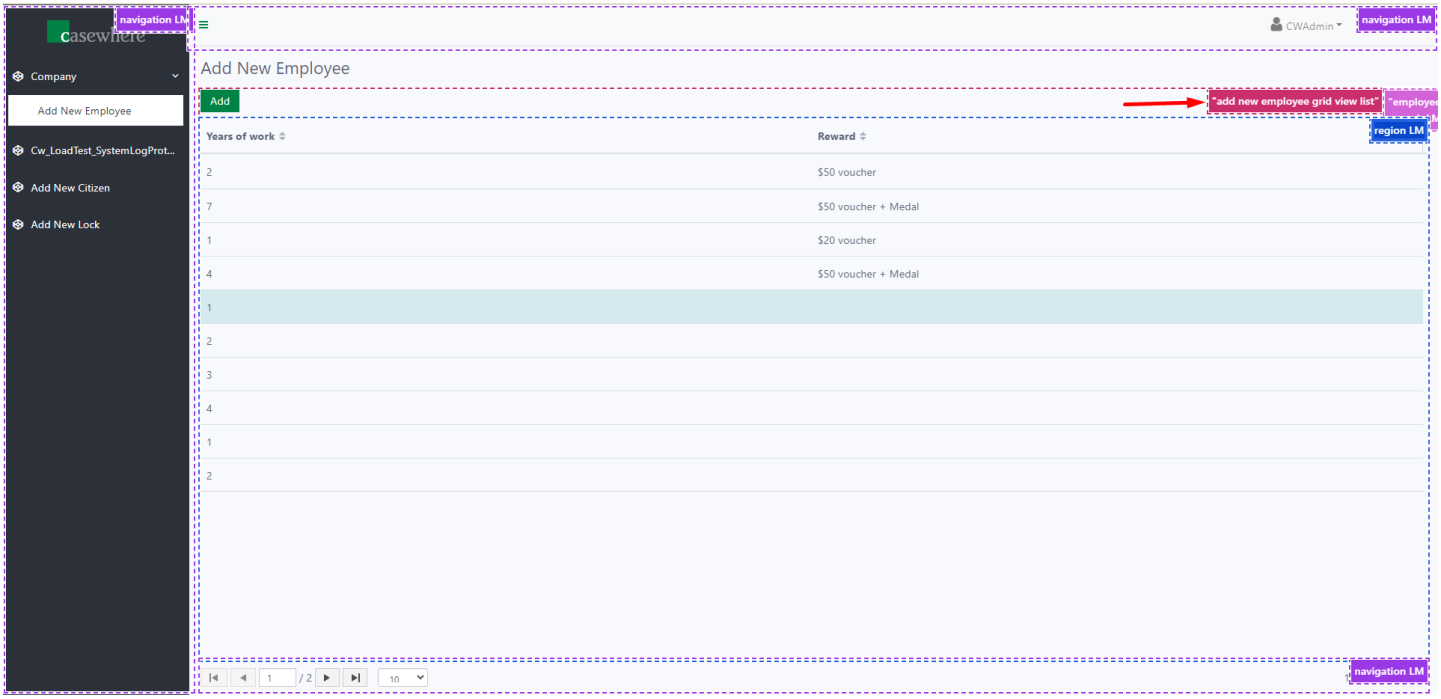


Accessibility

There are 2 fields in this **Accessibility** tab:

- **Landmark**: Assign the landmark type to the current grid view. The grid view can be detected as a landmark by the Narrator.
- **Label**: The label of the grid view. A small description to describe the grid view for Narrator users.


Data Grid Settings
Data Source: *
Employee
Could not find what you need? Click here
General
Columns
Access
Advanced
Security
Accessibility
Landmark:
main
Label:
Add new employee grid view list
Change logs
Save
Save And Close
Cancel


navigation LM
casewhere
Company
Add New Employee
Cw_LoadTest_SystemLogProt...
Add New Citizen
Add New Lock
CWAdmin
navigation LM
Add New Employee
Add
"add new employee grid view list"
region LM
Years of work
Reward
2
$50 voucher
7
$50 voucher + Medal
1
$20 voucher
4
$50 voucher + Medal
1
2
3
4
1
2
navigation LM

List widget

For design scenarios where you want to display data in a more compact layout rather than a table with rows and columns, list widget is the best choice as it provides a wide range of flexibility to customize the widget's appearance and behavior. Below is how list widgets looks in the worker site:

- Card layout



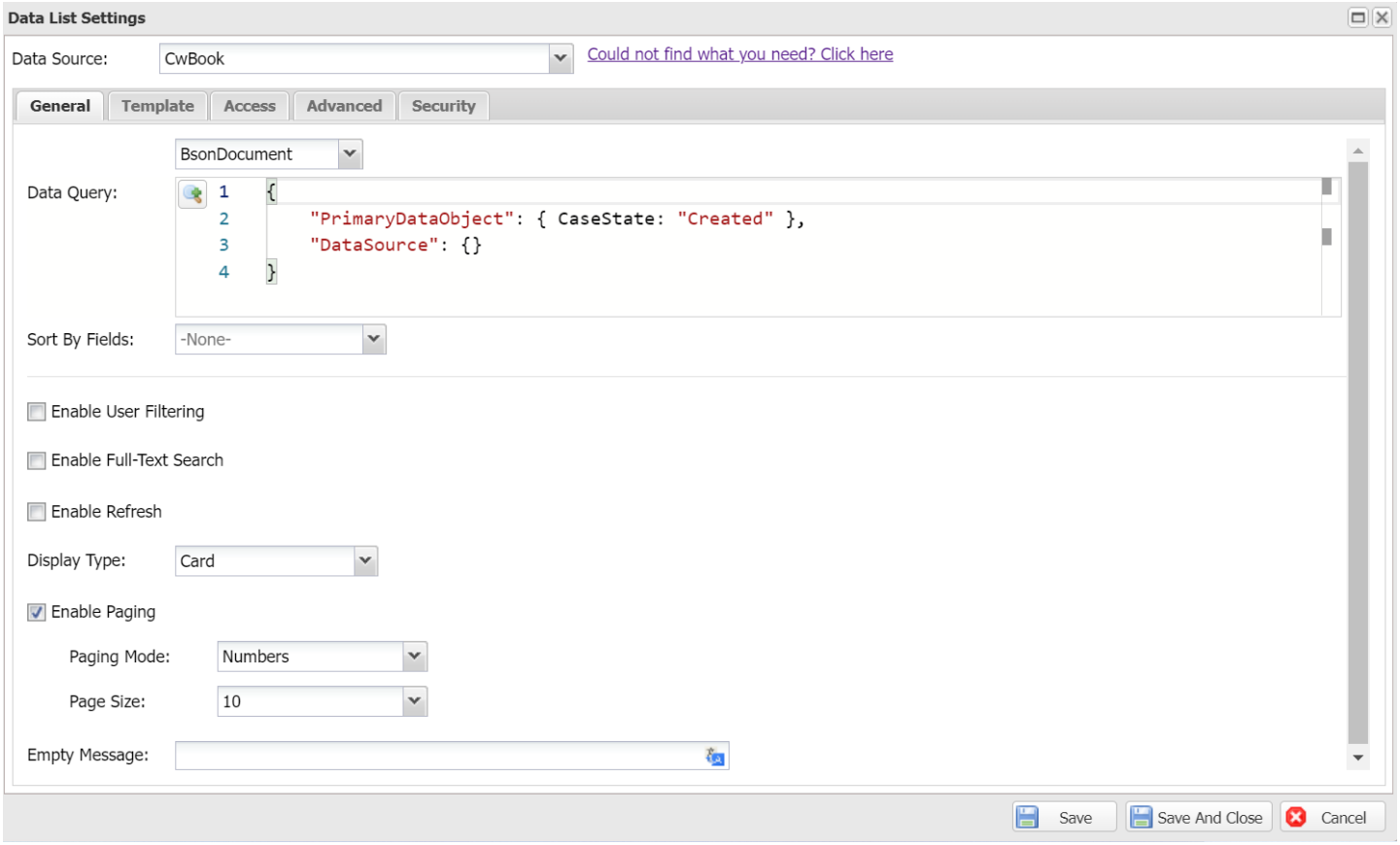
- Row layout



Configure list widget

General

All data-source widgets will share the same general settings for loading data. You can learn more about those settings here.

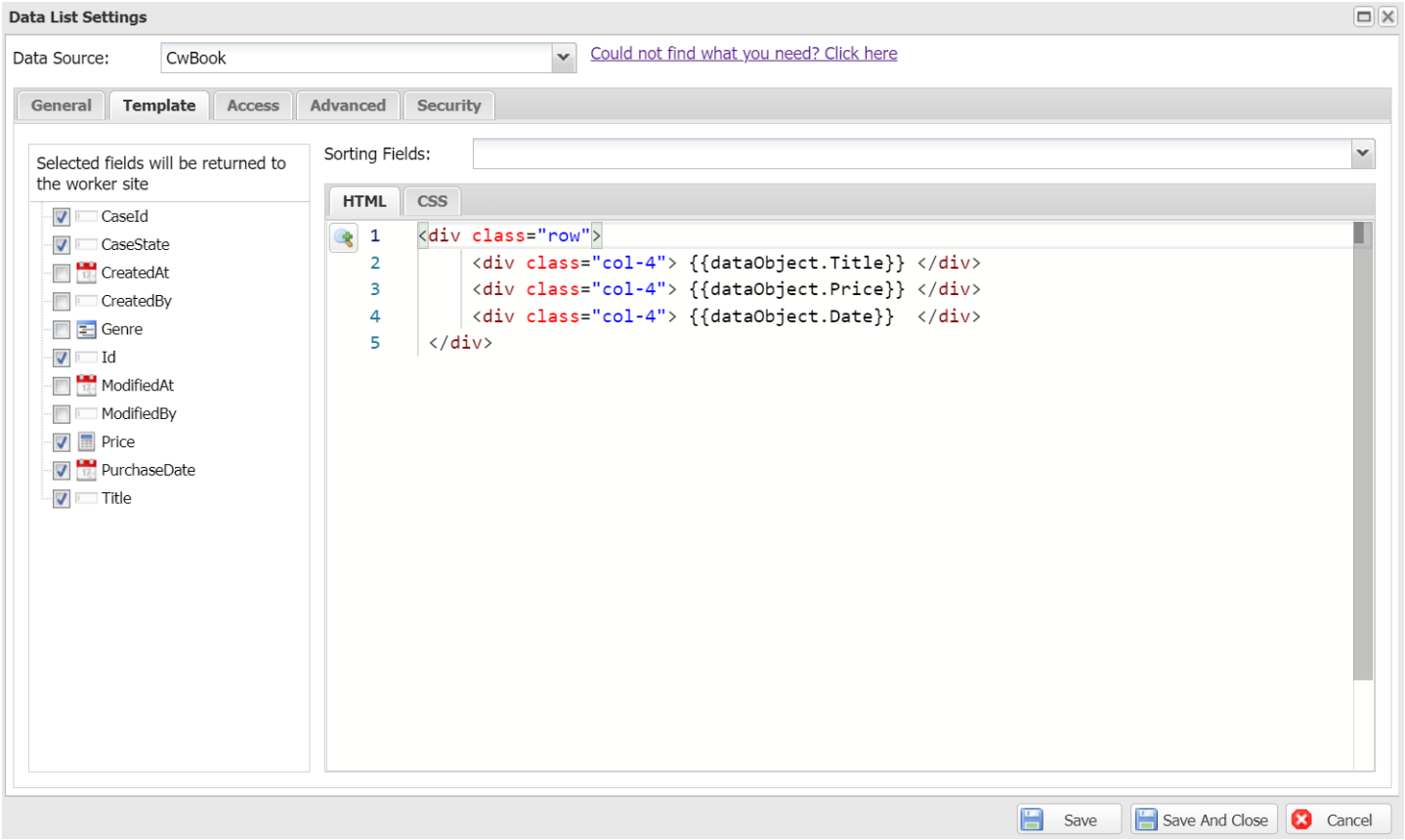


Below are the specific settings for list widget:

- **Display Type**:
  - **Card**: Multiple items per row.
  - **Row**: One item per row.
- **Empty Message**: The message to be displayed when there is no data to show.

Template

You can design the list item template using the below editor. In the field list, make sure you select the fields used in your template so Casewhere will return them to the work site.



Advanced settings

The below settings are useful in advanced use cases:

- **Responsive Modes**: The device-screen sizes on which the widget will appear. None means the widget will appear on all screen sizes.

Custom widget

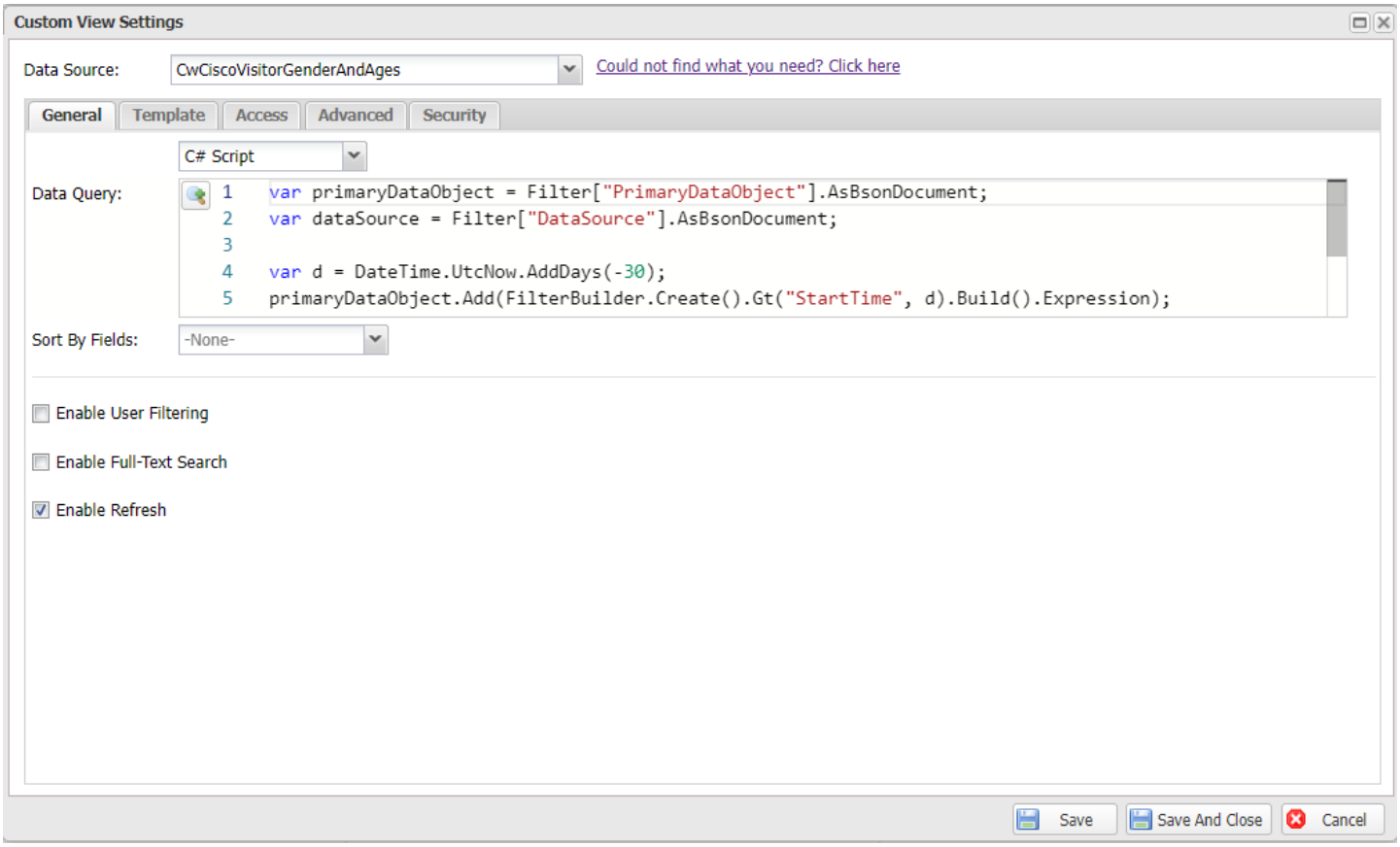
Even though grid widget and list widget are two of the most popular. There will be cases when business needs require developers to build customized widgets.

That's when the **Custom Widget** feature comes handy in Casewhere. Custom widgets are made of HTML, CSS and JavaScript. Like built-in widgets, custom widgets can load and display data from Casewhere data sources.

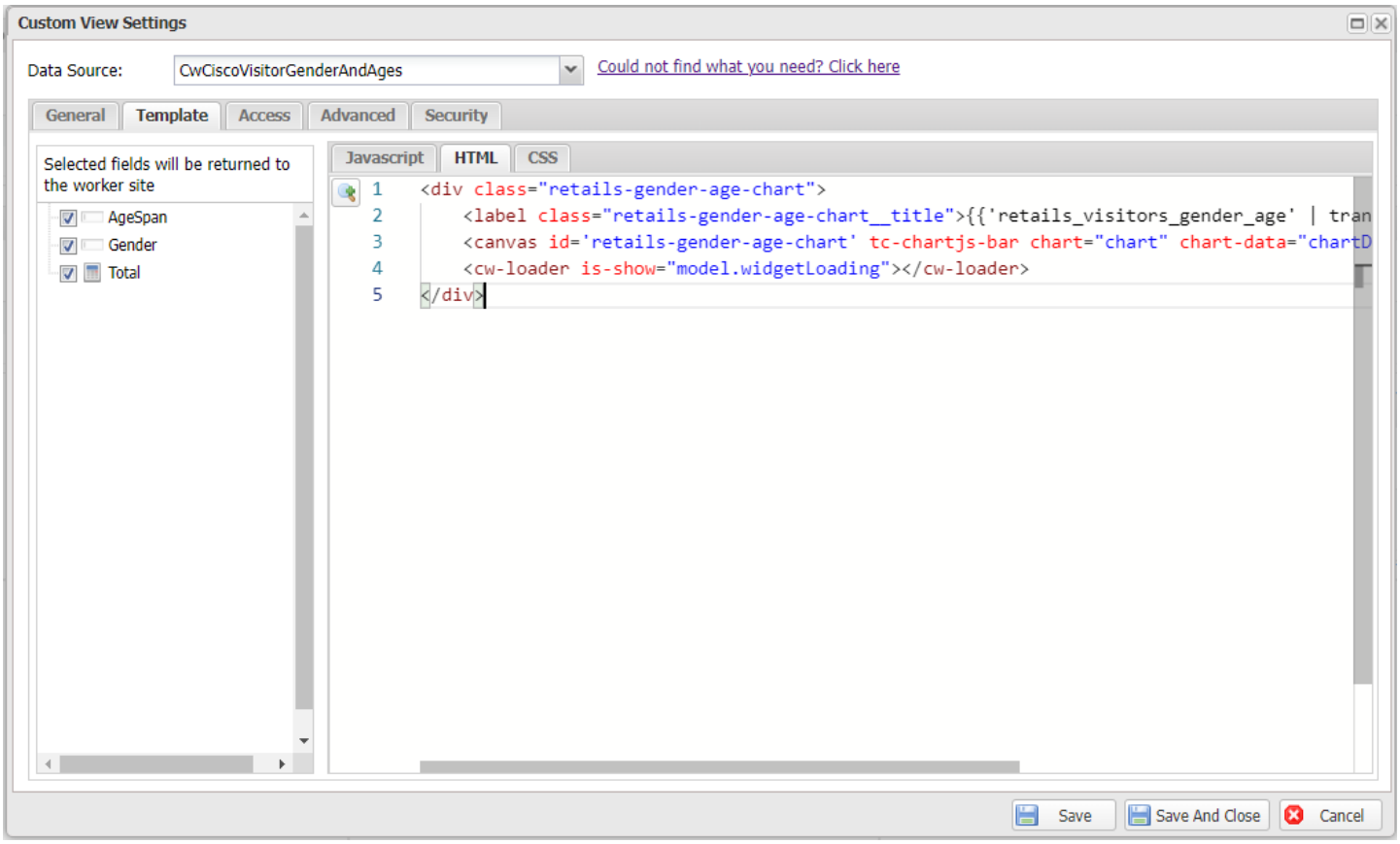
Configure custom widget

There are two important steps to configure a custom widget:

Step 1. Select a data source.



Step 2. Design the widget template. Casewhere will generate a default template you can customize when creating a new widget.



Step 3. Configure other advanced settings. A custom widget shares all common settings with other widget types. Read more in Widget management.

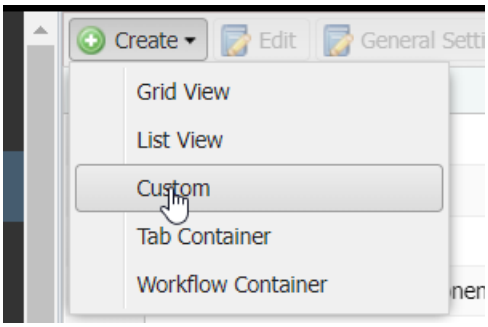
A simple example

Let's say business side asked the develop team to build some simple charts with provided input value.

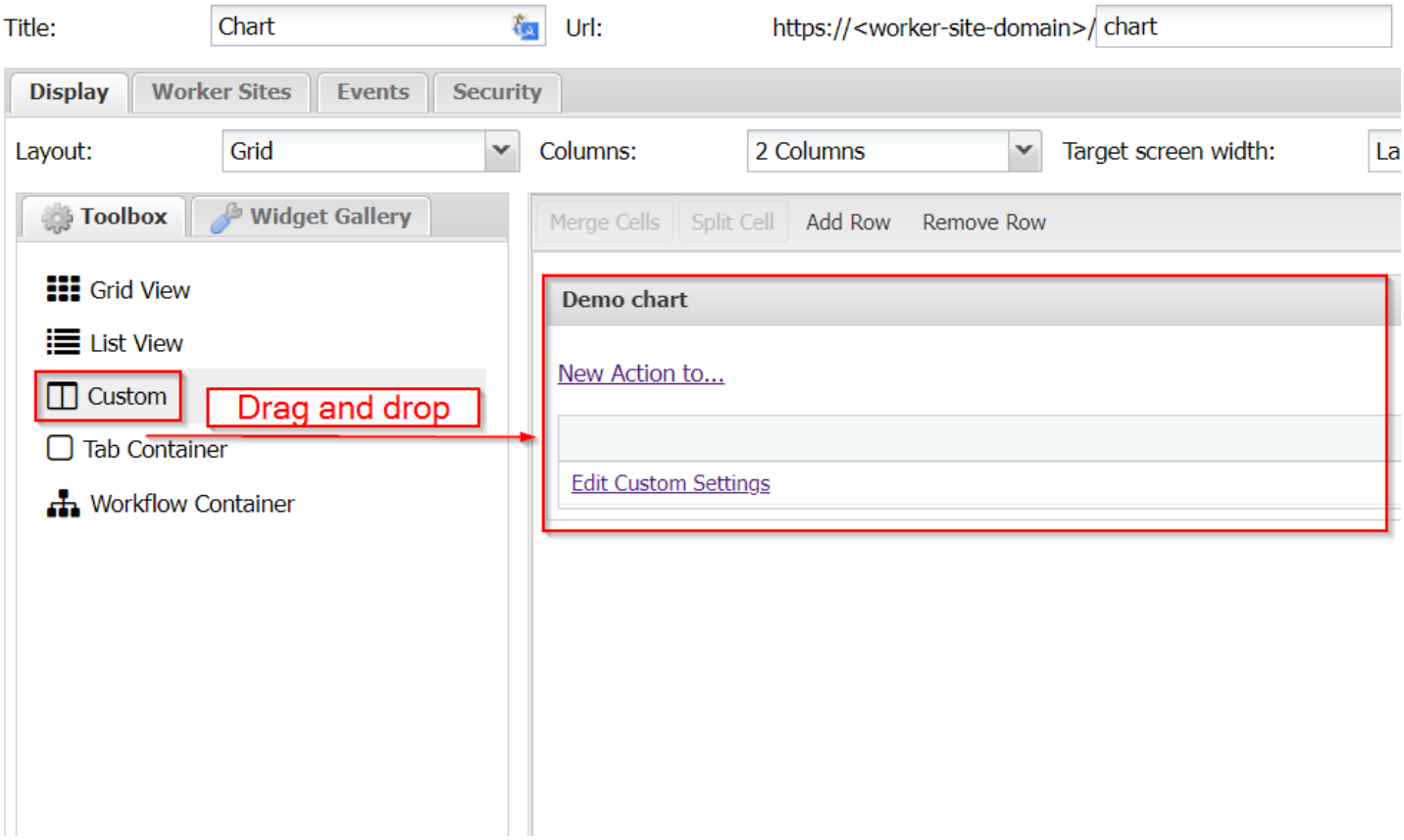
This type of business requirement will not totally fit in with a grid or list widget. But lucky that Casewhere support customize widget where you can call out some visualization library.

This example uses ChartJS. You can implemented more library to Casewhere by using workersite or theme (Navigate to **Theme** > **Custom component** > **Javascript**).

Step 1. Navigate to **Widgets**, Click **Create** button, select **Custom.**.



Then drag and drop **Custom** from **Toolbox** to main working area.



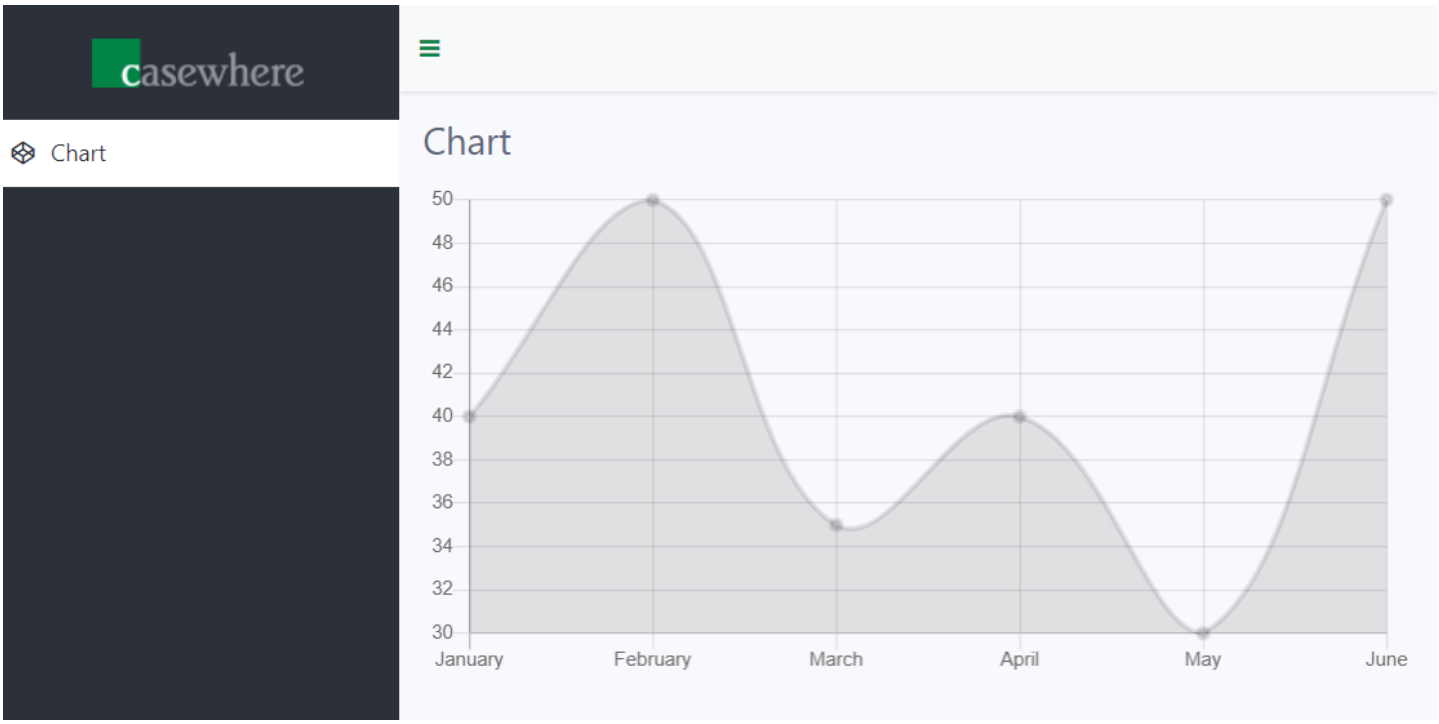
Step 2. Open **Edit Custom Settings** dialog.

Go to **Template** tab and append the following codes to the end of the default codes.

Go to **HTML** tab and paste the following codes.

```
<div class="row d-flex flex-grow">
    <div class="col-12 flex-grow flex-column text-right">
        <canvas id='cisco-chart' tc-chartjs-line chart-data="model.options.data1" chart-
options="model.options.options"></canvas>
    </div>
</div>
```

And that's it. Reload your page to see the chart result.



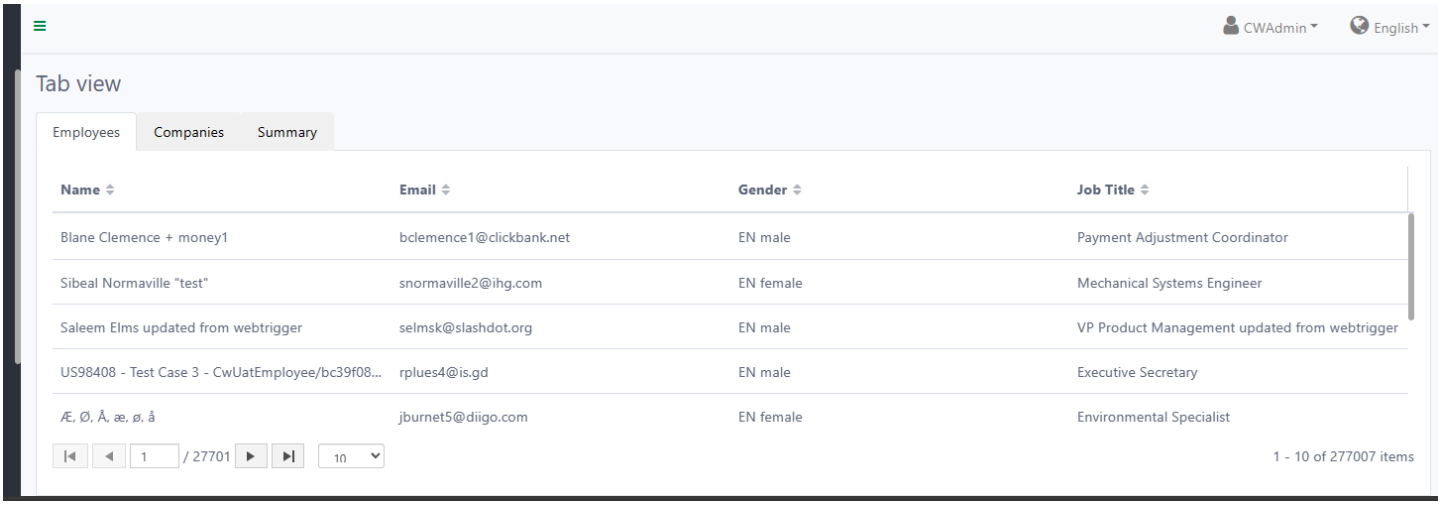
APIs to use with custom widget

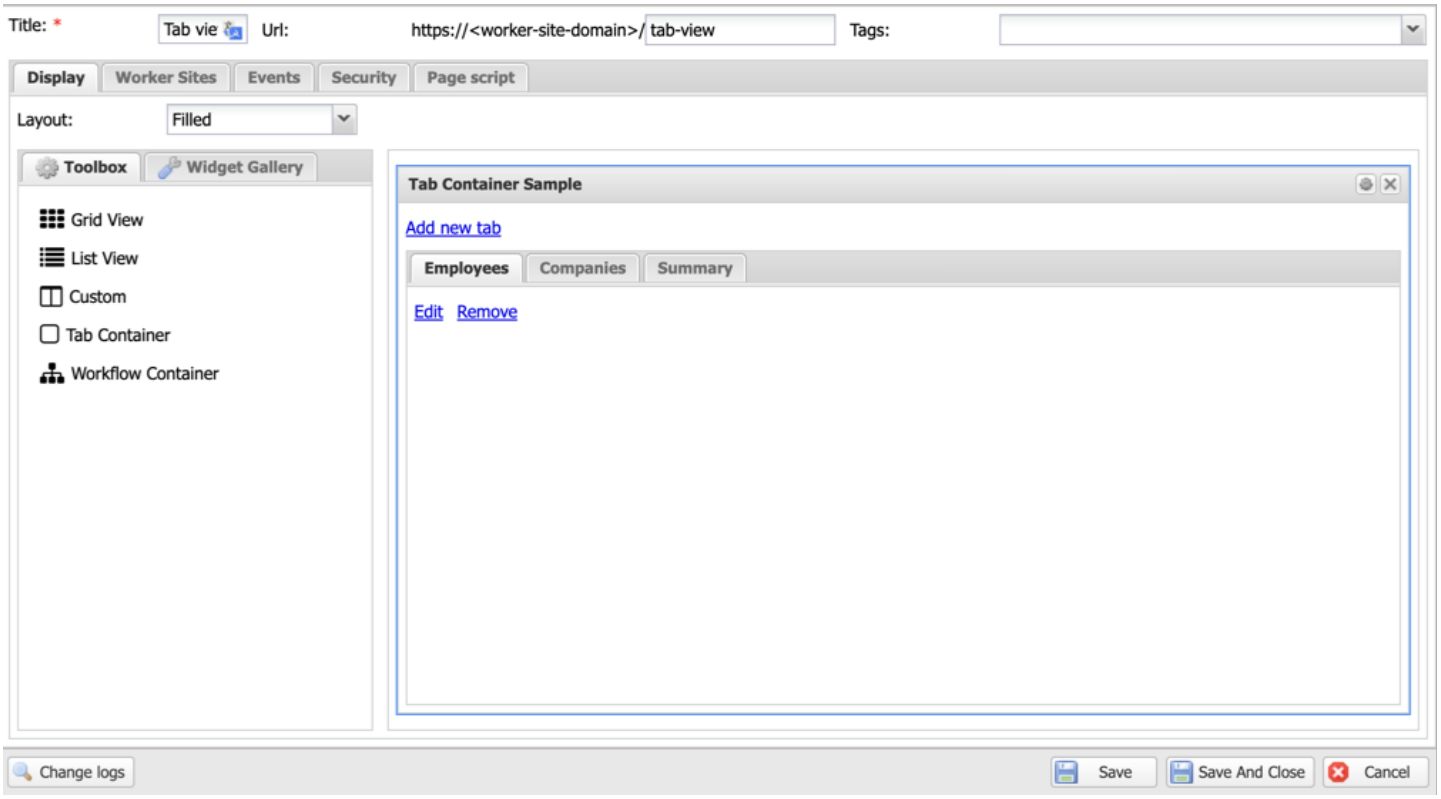
You can inject all AngularJS built-in services to the `customWidgetController`. Besides that, Casewhere provides many worker services you can use to develop your widget.

Tab container

In Casewhere, a **Tab Container** widget typically refers to a UI component that organizes content into tabs. Each tab can contain one or multiple widgets.

Below is how a **Tab Container** widget looks in the worker site and in Casewhere Admin.





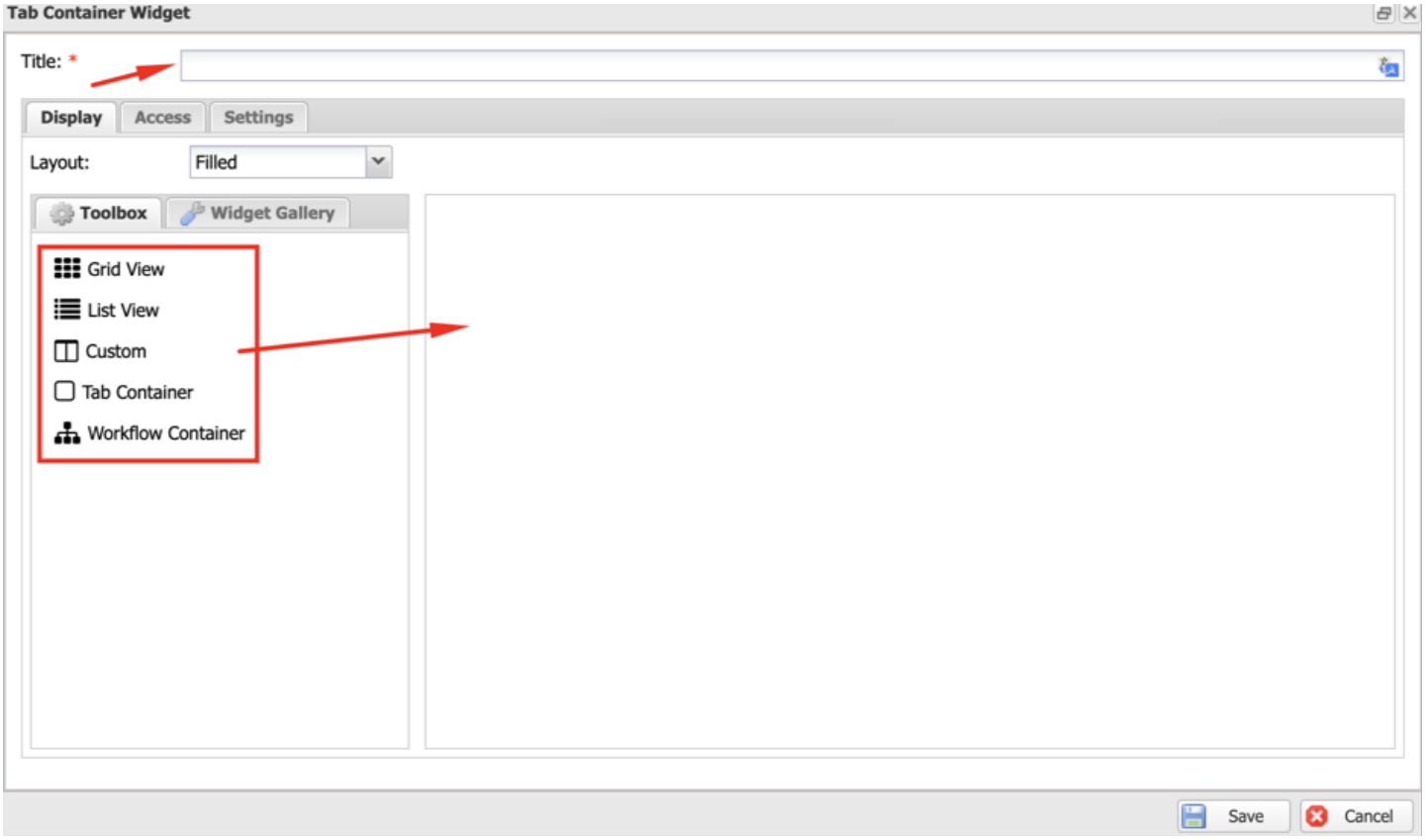
Configure tab container

In the **Tab Container** configuration interface, we can add, edit, or remove tab content via these action links.



Tab content

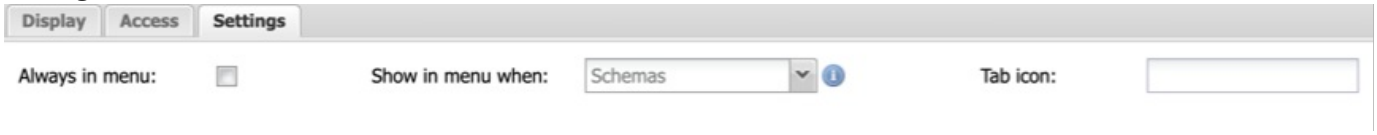
When adding or editing a tab content, we will update these settings below:



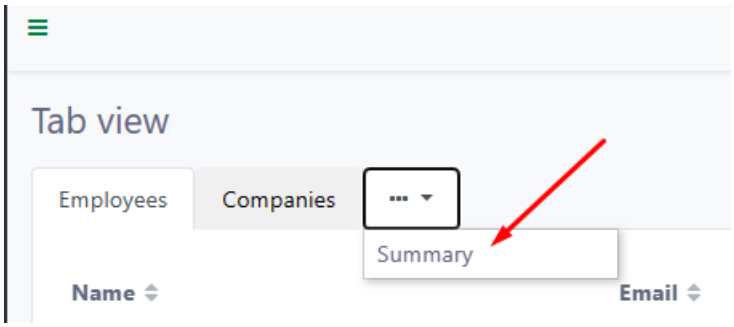
- **Title**: This is a display text when shown on the worker site and it can be a text resource key for supporting multi-language.
- **Display**: This tab allows you can change the kind of layout when rendering at client side. You can also drag and drop many widgets from toolbox to the main content by order follow the arrange of layout.
- **Access**: In this tab, you can write the rule to decide how visible of the tab content at client side and server side.



- **Settings**:



  - **Always in menu**: This setting will let this tab content always in a drop menu when rendering at client side. When it is checked then **Show in menu when** will be disabled.

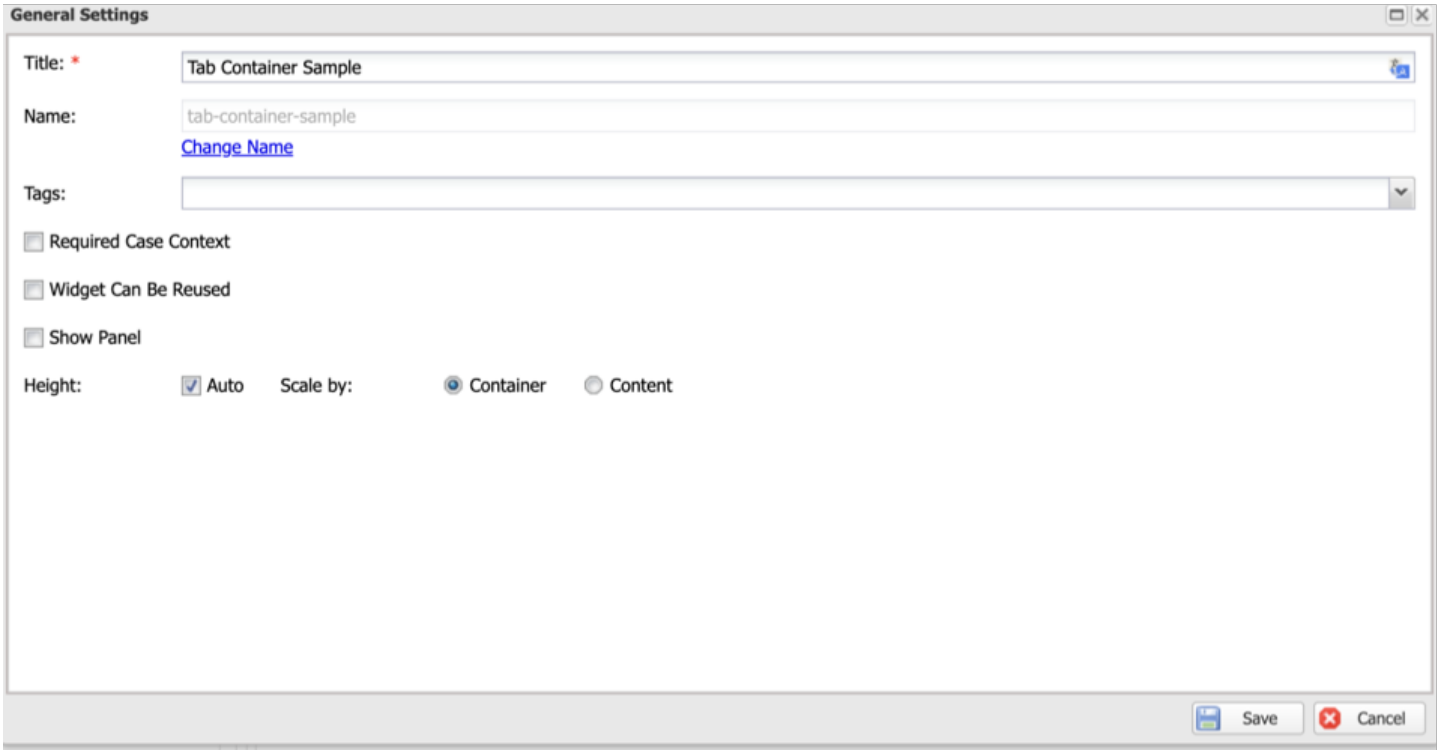


- **Show in menu when**: Only rendering menu in drop menu whenever the client device reaches specified responsive modes.
- **Tab icon**: the custom class to show the icon before the **Title** of tab

Other settings

General settings



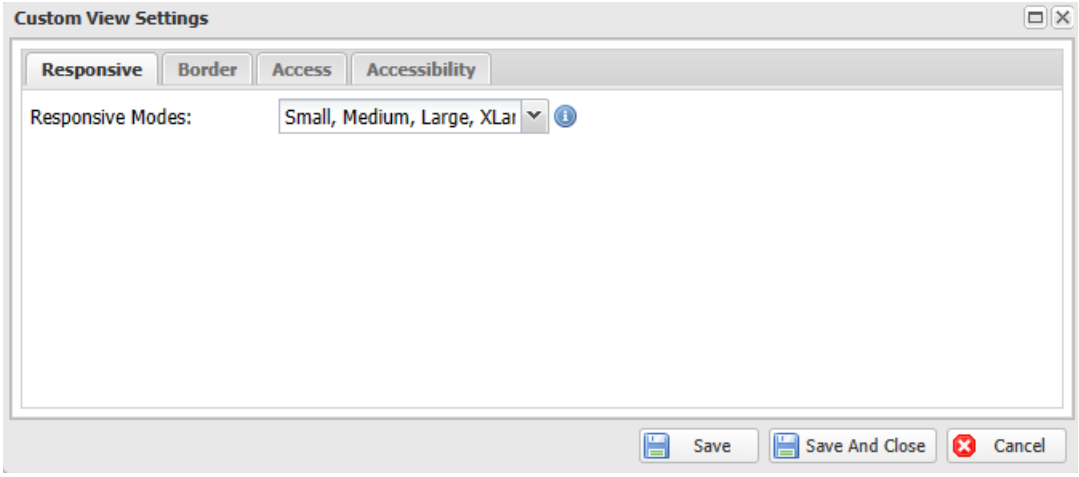


In the general settings, we can change the **Title**, **Name** together with other settings:

- **Required Case Context**: enable if you want your widget only active to use when received a case from outside.
- **Widget Can Be Reused**: enable if you want to share this widget for other page to use. It widget will be available in Widget gallery.
- **Show Panel**: enable if you want to show the panel that contain your widget.
- **Height**:

- **Auto**: if you want your widget auto scale the height by Container or Content.
- **Number**: if un-check **Auto** then you have to manually specify the height by your-self.

Custom view settings

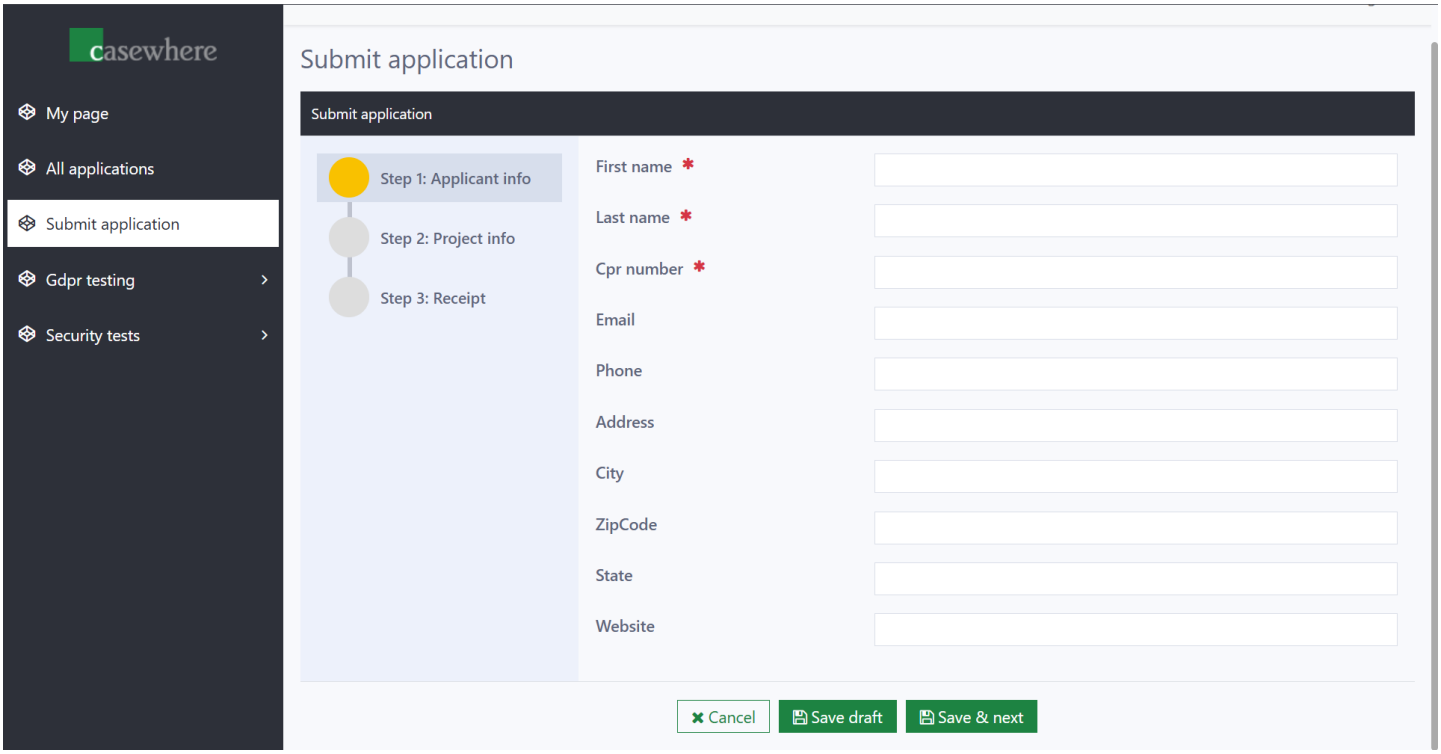


The below settings are useful in custom view use cases:

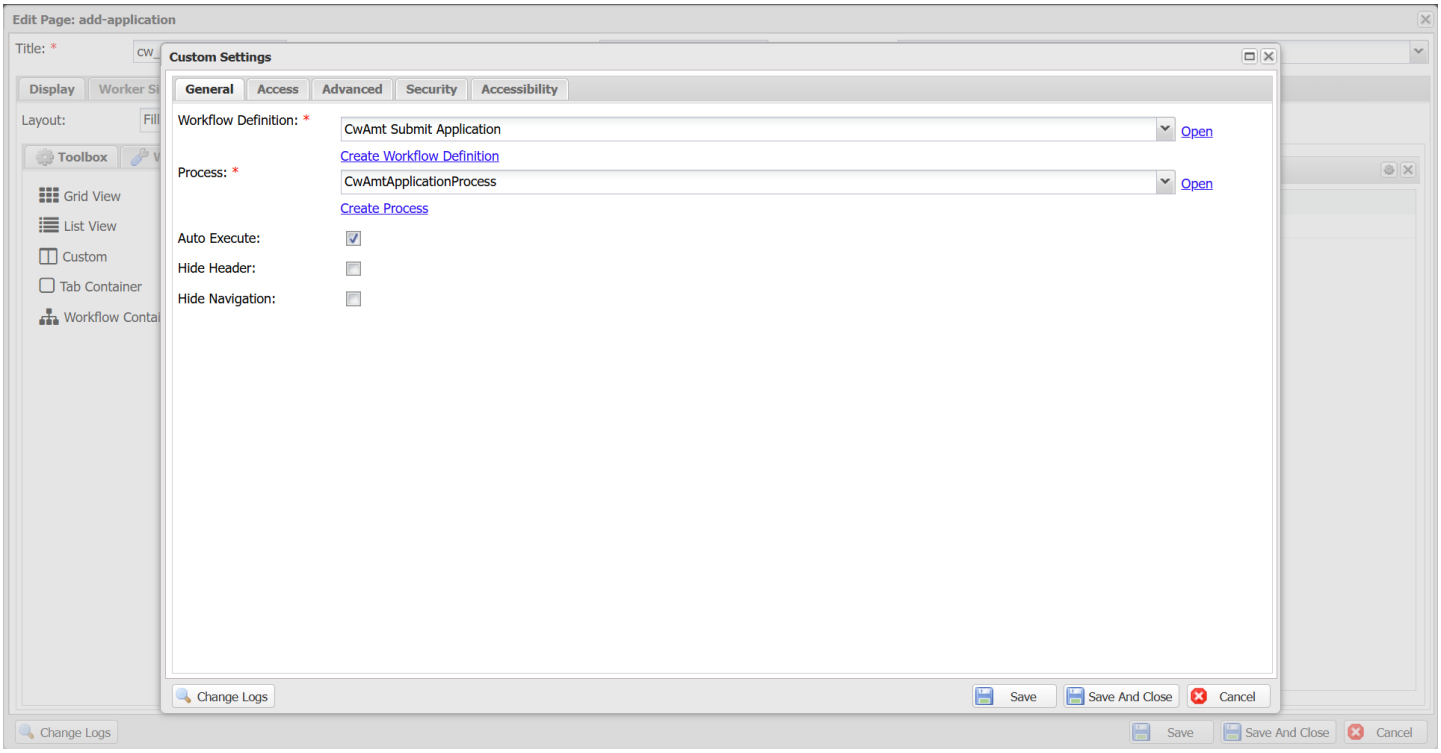
- **Responsive Modes**: The device-screen sizes on which the widget will appear. None means the widget will appear on all screen sizes.
- **Border**: Render the widget with a border.
- **Access**: Only render the widget when the user has access to it via Access Rule.
- **Accessibility**:
  - A label to the widget to make it accessible to screen readers.
  - A landmark role to the widget to make it accessible to screen readers.

Workflow container

In scenarios where you need to load and display a workflow directly within a page, as shown in the following screenshot, the workflow container will be useful.



The settings for a workflow container widget appear as follow.



SETTING	MANDATORY	DESCRIPTION
Workflow Definition	Yes	Specify the workflow to be executed.

SETTING	MANDATORY	DESCRIPTION
Process	Yes	If the selected workflow definition is of the case-creation type, you must specify the process in which the workflow will run.
Auto Execute	No	Casewhere will automatically execute the workflow when the widget is loaded. If unchecked, you must manually execute the workflow using the widget API.
Hide Header	No	By default, Casewhere will display the workflow header, but you can turn it off.
Hide Navigation	No	By default, Casewhere will display the workflow navigation, but you can turn it off.

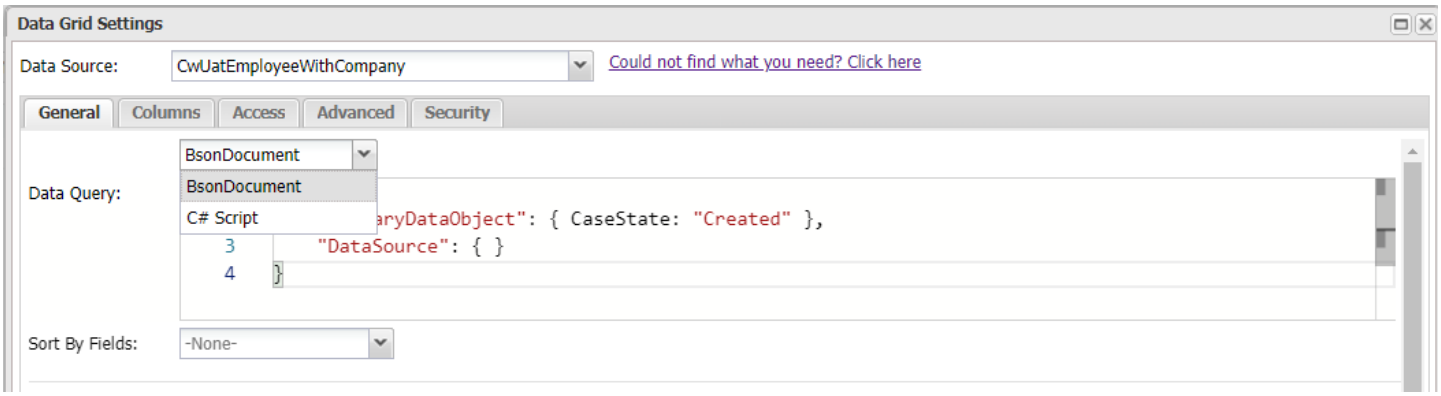
The following script can be used to manually trigger the workflow. You can include this script in a page event handler.

```
var api = $scope.widgetApis['application-form'];
api.triggerWorkflow();
```

Other settings will be the same as other widget types. You can learn more here.

Widget data query

When configuring a data widget such as a grid widget or list widget, you will need to instruct Casewhere how to load data from a data source by composing a data query. Casewhere supports two types of query syntaxes: **Bson** and **C#**.



Bson data query

Bson is the default query syntax, which is sufficient for most cases.

```
{
    "PrimaryDataObject": { CaseState: "Created" },
    "DataSource": {}
}
```

- `PrimaryDataObject`: Filter on the data source's primary data class. Filtering on the primary data class is generally fast because Casewhere can utilize database indexes to speed up the queries.
- `DataSource`: Filter on the data source projected fields. Casewhere cannot utilize database indexes for this kind of filter, so it's not recommended to use on a large data source.

Keywords

`@user`: The user in context. You can learn more about the user data object here.

```
{
    "PrimaryDataObject": { CaseState: "Created", CreatedBy: "{{@user.UserId}}" },
    "DataSource": {}
}
```

`Params`: The widget parameters sent from worker sites. You can learn how to set the parameters here.

Then the data query will be:

```
{
    "PrimaryDataObject": { CaseState: "Created", Author: "{{Params.Author}}" },
    "DataSource": {}
}
```

Examples

The **book** data source has below fields:

```
{
    Id,
    CreatedBy,
    Title,
    Description,
    Author,
    Year,
    ...
}
```

To get a specific book by its ID

```
{
    "PrimaryDataObject": { CaseState: "Created", _id: "{{Params.Id}}" },
    "DataSource": {}
}
```

To get books created by the logged-in user

```
{
    "PrimaryDataObject": { CaseState: "Created", CreatedBy: "{{@user.UserId}}" },
    "DataSource": {}
}
```

Let's extend the book data source a bit, now it also has the library's information.

```
{
    Id,
    CreatedBy,
    Title,
    Description,
    Author,
    Year,
    Library: {
        Id,
        Name,
        ...
    },
    ...
}
```

To get books in the `Library of Congress`:

```
{
    "PrimaryDataObject": { CaseState: "Created" },
    "DataSource": { "Library.Name": "Library of Congress" }
}
```

C# data query

This syntax is designed to handle the most complex scenarios. Here you can query the database or even execute plugins to get more information to enrich the data queries. Similar to the **Bson** syntax, the **C#** syntax also has the two concepts `PrimaryDataObject` and `DataSource`.

```
var primaryDataObject = Filter["PrimaryDataObject"].AsBsonDocument;
var dataSource = Filter["DataSource"].AsBsonDocument;

primaryDataObject.Add(new BsonDocument("CaseState", "Created"));
```

Keywords

`@user`: The user in context. You can learn more about the user data object here.

```
var primaryDataObject = Filter["PrimaryDataObject"].AsBsonDocument;
var dataSource = Filter["DataSource"].AsBsonDocument;

primaryDataObject.Add(new BsonDocument("CaseState", "Created"));
primaryDataObject.Add(new BsonDocument("CreatedBy", @user.UserId);
```

`@parameters`: The widget parameters sent from worker sites. You can learn how to set the parameters here.

```
var primaryDataObject = Filter["PrimaryDataObject"].AsBsonDocument;
var dataSource = Filter["DataSource"].AsBsonDocument;

primaryDataObject.Add(new BsonDocument("CaseState", "Created"));
primaryDataObject.Add(new BsonDocument("Author", @parameters.Get<string>("Author")));
```

`@apiFactory`: To construct Casewhere API objects. Read here for the full list of available Apis.

```
var dataApi = @apiFactory.Get<IDataApi>();
var pluginApi = @apiFactory.Get<IPluginApi>();
var documentApi = @apiFactory.Get<IDocumentApi>();
var cacheApi = @apiFactory.Get<ICacheApi>();
...
```

Examples

Assume you have a data source in the below structure:

```
{
    Id,
    CreatedBy,
    Title,
    Description,
    Author,
    Year,
    Library: {
        Id,
        Name,
        ...
    },

    ...
}
```

To get a specific book by its ID

```
var primaryDataObject = Filter["PrimaryDataObject"].AsBsonDocument;
var dataSource = Filter["DataSource"].AsBsonDocument;

primaryDataObject.Add(new BsonDocument("CaseState", "Created"));
primaryDataObject.Add(new BsonDocument("_id", @parameters.Get<string>("Id")));
```

If the `Id` parameter is not provided, load all books. Else, load the book by its Id.

```
var primaryDataObject = Filter["PrimaryDataObject"].AsBsonDocument;
var dataSource = Filter["DataSource"].AsBsonDocument;

primaryDataObject.Add(new BsonDocument("CaseState", "Created"));
if (@parameters.Has("Id"))
{
    primaryDataObject.Add(new BsonDocument("_id", @parameters.Get<string>("Id")));
}
```

To get books created by the logged-in user

```
var primaryDataObject = Filter["PrimaryDataObject"].AsBsonDocument;
var dataSource = Filter["DataSource"].AsBsonDocument;

primaryDataObject.Add(new BsonDocument("CaseState", "Created"));
primaryDataObject.Add(new BsonDocument("CreatedBy", @user.UserId));
```

To get books in the `Library of Congress`:

```
var primaryDataObject = Filter["PrimaryDataObject"].AsBsonDocument;
var dataSource = Filter["DataSource"].AsBsonDocument;

primaryDataObject.Add(new BsonDocument("CaseState", "Created"));
dataSource.Add(new BsonDocument("Library.Name", "Library of Congress"));
```

To find books at all libraries in Denmark:

```
var primaryDataObject = Filter["PrimaryDataObject"].AsBsonDocument;
var dataSource = Filter["DataSource"].AsBsonDocument;

primaryDataObject.Add(new BsonDocument("CaseState", "Created"));

var dataApi = @apiFactory.Get<IDataApi>();
var libraryFilter = FilterBuilder
    .Create()
    .Eq("Country", "DK")
    .Build();

var libraries = dataApi.Search(DataObjectApiQuery
    .For("Library")
    .FilterBy(libraryFilter)).Data;

var libraryIds = libraries.Select(lib => lib.Get<string>("Id")).ToList();

if (libraryIds.Count > 0)
{
    primaryDataObject.Add(new BsonDocument("LibraryId", new BsonDocument("$in", new BsonArray(libraryIds))));
}
```

Widget API

Widget API is designed to facilitate the communication between page widgets, i.e., you can subscribe events from a widget to update another one. All Casewhere widgets provide a set of APIs so you can, in page event handlers, call these APIs to interact with the widgets. You can event implement custom API for custom widgets to handle your custom logic.

Built-in APIs

All built-in widgets provide a set of APIs so you can, in page event handlers, call these APIs to interact with the widgets.

Grid widget

`refresh()`

Refresh the grid widget. Data will be reloaded from the server.

Example

```
var api = $scope.widgetApis['book-list'];
api.refresh();
```

`getSelectedDOs()`

Retrieve the user's selected items.

Example

```
var api = $scope.widgetApis['book-list'];
api.getSelectedDOs();
```

`setSelectedDOs(items)`

Set the selected items, the old ones will be replaced with the new ones.

Parameters

NAME	TYPE	DETAILS
items	object[]	Each item must have the `Id` field. Casewhere will ignore the items which are not found in the current widget's data.

Example

```
var api = $scope.widgetApis['book-list'];
var bookList = bookListApi.getWidgetData().data;

// Find the current book
var currentBook = api.getSelectedDOs()[0];

// Clear user's selection
api.setSelectedDOs([]);

var currentIndex = _.findIndex(bookList, function(book) {
    return book.Id === currentBook.Id;
});

// Select the next one
var nextBook = bookList[currentIndex + 1];
api.setSelectedDOs([nextBook]);
```

`runWidgetAction(widgetActionName, doIds, data)`

Run the widget action configured for the widget.

Parameters

NAME	TYPE	DESCRIPTION
widgetActionName	string	Name of the action
doIds	object[]	Data objects as input for running the action. Each data object must have the `Id` field
data	object	Additional parameters for running the action

Example

```
var api = $scope.widgetApis['book-list'];
var parameters = {
    Name: "The Da Vinci Code",
    Author: "Dan Brown",
    Year: "2003"
};

api.runWidgetAction("Add book", null, parameters);
```

`setQueryParameters(parameters)`

Set the parameter for the widget data query.

Parameters

NAME	TYPE	DESCRIPTION
parameters	object	A key/value object

Example

```
var api = $scope.widgetApis['book-list'];
var parameters = {
    Author: "Dan Brown"
};

api.setQueryParameters(parameters);
```

`setActionCustomInputs(widgetActionName, doIds)`

Use this method to programmatically set input for widget actions, which is useful when you want to load data to dynamically reflect the change in UI.

Parameters

NAME	TYPE	DESCRIPTION
widgetActionName	string	Name of the action
doIds	object[]	Inputs for running the action

Example

```
var api = $scope.widgetApis['book-list'];
var bookIds = [
    {
        "Id": "Book/3ae2aa8c-7937-4d02-ba17-ae0500c5b426"
    }
];

api.setActionCustomInputs("Edit book", bookIds);
```

`show()`

Show the widget.

Example

```
var api = $scope.widgetApis['book-list'];
api.show();
```

`hide()`

Hide the widget.

Example

```
var api = $scope.widgetApis['book-list'];
api.hide();
```

`setState(name, value)`

Save data to the widget internal storage.

Parameters

NAME	TYPE	DESCRIPTION
name	string	The name to reference the state value
value	*	The value could be anything

Example

```
var api = $scope.widgetApis['book-list'];
api.setState("version", 1);
```

`getState(name)`

Get data from the widget internal storage.

Parameters

NAME	TYPE	DESCRIPTION
name	string	Name of the state

Example

```
var api = $scope.widgetApis['book-list'];
var version = api.getState("version"); // version: 1
```

`getWidgetData()`

Get the widget data in the format:

```
{
    data: [], // an array of items
    total: 100, // total items
    metadata: {} // the metadata of the data source used by the widget
}
```

Example

```
var api = $scope.widgetApis['book-list'];
var widgetData = api.getWidgetData();
var books = widgetData.data;
var totalBook = widgetData.total;
```

`updateWidgetData(widgetData)`

Update the widget data.

Parameters

NAME	TYPE	DESCRIPTION
widgetData	object	An object in the following format: `{ data: // an array of items, total: // total items }`

Example

```
var api = $scope.widgetApis['book-list'];
var books = getBooksByGenre("Novel");
var totalBook = 1000;
var widgetData = {
    data: books,
    total: totalBook
};
api.updateWidgetData(widgetData);
```

`refreshByWorkflowId(workflowId)`

Gets the data changes by a workflow and then, reflects them in the grid.

Parameters

PARAM	TYPE	DETAILS
workflowId	string	ID of the workflow to get changes.

Example

```
var api = $scope.widgetApis['book-list'];
var workflowId = "4aaf6240-11a3-4422-9509-ae1400d56039";
api.refreshByWorkflowId(workflowId);
```

`onDelete(dataObject)`

Remove the data object from the current view.

> From 2.7.2, the name for this method should be `delete` however, you can still use `onDelete`.

Parameters

PARAM	TYPE	DETAILS
dataObject	object	The object must have the `ID` field.

Example

```
var api = $scope.widgetApis['book-list'];
var widgetData = api.getWidgetData();
var books = widgetData.data;
api.onDelete(books[0]);
```

`export()`

Export the grid data to an excel file.

Example

```
var api = $scope.widgetApis['book-list'];
api.export();
```

`addColumns(columns, index)`

Add a set of columns at a specific position.

Parameters

PARAM	TYPE	DETAILS
columns	object/object[]	Pass an object if you want to add a single column, pass an array if you want to add multiple columns. The format of each column: `{ "name": <string>, // name of a column "header": <string>, // header of a column "path": <string>, // a column name used for sorting a column" "template": <string>, // an AngularJS binding expression, e.g. " {{dataObject.Title}}" "minWidth": <number>, // column's minimum width in pixel "maxWidth": <number>, // column's maximum width in pixel "modes": <string[]>, // screen sizes to display a column, leave it empty or null to display on all screens "exportExpression": <string>, // a JS expression e.g. "dataObject.Title" "exportFormat": <string>, // the default formats are listed in ECMA-376 18.8.30 "pin": <string>, // Possible values: "Left", "Right", null }`
index	number	[Optional] A zero-based index position where the columns are added. If not defined, Casewhere will append the columns to the last position.

Example

```
var api = $scope.widgetApis['book-list'];
var newColumn = {
    name: "Author",
    header: "Author",
    path: "Author",
    template: "{{dataObject.Author}}",
    exportExpression: "dataObject.Author"
};
api.addColumns(newColumn, 0);
```

`getQueryOptions()`

Get the current query options.

```
{
    QueryParameters:, // the parameters used for getting data
    Pagination: {
        PageSize:, // the number of items in each page
        PageIndex: // the current page number
    },
    clauses:, // the clauses for filtering data
    sortOptions:, // the current sorting options
    Search: // the full text search text
}
```

Example

```
var api = $scope.widgetApis['book-list'];
var queryOptions = api.getQueryOptions();
```

`setQueryOption(option)`

Set the data query options.

Parameters

NAME	TYPE	DETAILS
option	object	The object described at getQueryOptions.

Example

```
var api = $scope.widgetApis['book-list'];
var option = {
    Search: "Advanture"
};
api.setQueryOption(option);
```

`extendGridOptions(option)`

Customize the grid settings.

Parameters

NAME	TYPE	DESCRIPTION
option	object	Learn more about the grid settings here.

Example

```
var api = $scope.widgetApis['book-list'];
var option = {
    rowHeight: 50
};
api.extendGridOptions(option);
```

List widget

`refresh()`

Refresh the list widget. Data will be reloaded from the server.

Example

```
var api = $scope.widgetApis['book-list'];
api.refresh();
```

`getSelectedDOs()`

Retrieve the user's selected items.

```
[
    {
        "Id": // first selected item's ID,
        ...
    },
    {
        "Id": // second selected item's ID,
        ...
    }
]
```

Example

```
var api = $scope.widgetApis['book-list'];
api.getSelectedDOs();
```

`setSelectedDOs(items)`

Sets the selected items, the old ones will be replaced with the new ones.

Parameters

NAME	TYPE	DETAILS
items	object[]	Each item must have the `Id` field. Casewhere will ignore the items which are not found in the current widget's data.

Example

```
var api = $scope.widgetApis['book-list'];
var bookList = api.getWidgetData().data;
var currentBook = api.getSelectedDOs()[0];
var currentIndex = _.findIndex(bookList, function(book) {
    return book.Id === currentBook.Id;
});
var nextBook = bookList[currentIndex + 1];
// Deselect the selected book
api.setSelectedDOs([]);

api.setSelectedDOs([nextBook]);
```

`runWidgetAction(widgetActionName, doIds, data)`

Run the widget action configured for the widget.

Parameters

NAME	TYPE	DETAILS
widgetActionName	string	Name of the action configured in the widget
doIds	object[]	Data objects as input for running the action. Each data object must have the `Id` field
data	object	Additional parameters for running the action

Example

```
var api = $scope.widgetApis['book-list'];
var newBook = {
    Name: "The Da Vinci Code",
    Author: "Dan Brown",
    Year: "2003"
};

api.runWidgetAction("Add book", null, newBook);
```

`setQueryParameters(parameters)`

Set the parameter for the widget data query.

Parameters

NAME	TYPE	DESCRIPTION
parameters	object	A key/value object

Example

```
var api = $scope.widgetApis['book-list'];
var parameters = {
    Author: "Dan Brown"
};

api.setQueryParameters(parameters);
```

`setActionCustomInputs(widgetActionName, doIds)`

Sometimes, the inputs for an action are not available in the widget itself, but from an external place. In those cases, use this method to manually set the inputs for the action.

Parameters

PARAM	TYPE	DETAILS
widgetActionName	string	Name of the action
doIds	object[]	Inputs for running the action

Example

```
var api = $scope.widgetApis['book-list'];
var bookIds = [
    {
        "Id": "Book/3ae2aa8c-7937-4d02-ba17-ae0500c5b426"
    }
];

api.setActionCustomInputs("Edit book", bookIds);
```

`show()`

Show the widget.

Example

```
var api = $scope.widgetApis['book-list'];
api.show();
```

`hide()`

Hide the widget.

Example

```
var api = $scope.widgetApis['book-list'];
api.hide();
```

setState(name, value)`

Save data to the widget internal storage.

Parameters

NAME	TYPE	DESCRIPTION
name	string	The name to reference the state value
value	*	The value could be anything

Example

```
var api = $scope.widgetApis['book-list'];
api.setState("version", 1);
```

`getState(name)`

Get data from the widget internal storage.

Parameters

NAME	TYPE	DESCRIPTION
name	string	Name of the state

`getWidgetData()`

Gets the widget data in the following format.

```
{
    data:, // an array of items
    total:, // total items
    metadata: // the metadata of the data source used by the widget
}
```

Example

```
var api = $scope.widgetApis['book-list'];
var widgetData = api.getWidgetData();
var books = widgetData.data;
var totalBook = widgetData.total;
```

`updateWidgetData(widgetData)`

Sets the widget data.

Parameters

NAME	TYPE	DESCRIPTION
widgetData	object	An object in the format: `{ data: // an array of items, total: // total items }`

Example

```
var api = $scope.widgetApis['book-list'];
var books = getBooksByGenre("Novel");
var totalBook = 1000;
var widgetData = {
    data: books,
    total: totalBook
};
api.updateWidgetData(widgetData);
```

`refreshByWorkflowId(workflowId)`

Gets the data changes by a workflow and then, reflects them in the list.

Parameters

NAME	TYPE	DESCRIPTION
workflowId	string	ID of the workflow to get changes.

Example

```
var api = $scope.widgetApis['book-list'];
var workflowId = "4aaf6240-11a3-4422-9509-ae1400d56039";
api.refreshByWorkflowId(workflowId);
```

`getQueryOptions()`

Get the current query options.

```
{
    QueryParameters:, // the parameters used for getting data
    Pagination: {
        PageSize:, // the number of items in each page
        PageIndex: // the current page number
    },
    clauses:, // the clauses for filtering data
    sortOptions:, // the current sorting options
    Search: // the full text search text
}
```

Example

```
var api = $scope.widgetApis['book-list'];
var queryOptions = api.getQueryOptions();
```

`setQueryOption(option)`

Set the data query options.

Parameters

NAME	TYPE	DESCRIPTION
option	object	An object in the format described at getQueryOptions.

Example

```
var api = $scope.widgetApis['book-list'];
var option = {
    Search: "Advanture"
};
api.setQueryOption(option);
```

Tab container widget

`selectTab(tabName)`

Select a tab by name.

Parameters

NAME	TYPE	DESCRIPTION
tabName	string	Name of the tab you want to select

Example

```
var api = $scope.widgetApis['book-categories'];
api.selectTab("Advanture");
```

`selectTabByIndex(tabIndex)`

Selects a tab by index.

Parameters

NAME	TYPE	DESCRIPTION
tabIndex	number	A zero-based index

Example

```
var api = $scope.widgetApis['book-categories'];
api.selectTabByIndex(0);
```

`clearCache()`

Remove the cached content of inactive tabs, so Casewhere will reload them when they are activated.

Example

```
var api = $scope.widgetApis['book-categories'];
api.clearCache();
```

`showTab(tabName)`

Show a tab by name.

Parameters

NAME	TYPE	DESCRIPTION
tabName	string	Name of a tab.

Example

```
var api = $scope.widgetApis['book-categories'];
api.showTab("Advanture");
```

`hideTab(tabName)`

Hide a tab by name.

Parameters

NAME	TYPE	DESCRIPTION
tabName	string	Name of a tab.

Example

```
var api = $scope.widgetApis['book-categories'];
api.hideTab("Advanture");
```

`showTabByIndex(tabIndex)`

Show a tab by index.

Parameters

PARAM	TYPE	DESCRIPTION
tabIndex	number	A zero-based index

Example

```
var api = $scope.widgetApis['book-categories'];
api.showTabByIndex(0);
```

`hideTabByIndex(tabIndex)`

Hides a tab by index.

Parameters

NAME	TYPE	DESCRIPTION
tabIndex	number	Index starts from 0

Example

```
var api = $scope.widgetApis['book-categories'];
api.hideTabByIndex(0);
```

`widgetApis`

To reference the child widgets by their names.

Example

```
var bookCategoriesApi = $scope.widgetApis['book-categories'];
var adventureBookListApi = bookCategoriesApi.widgetApis["adventure-book-list"];
adventureBookListApi.refresh();
```

`show()`

Show the widget.

Example

```
var api = $scope.widgetApis['book-categories'];
api.show();
```

`hide()`

Hides the widget.

Example

```
var api = $scope.widgetApis['book-categories'];
api.hide();
```

`setState(name, value)`

Save data to the widget internal storage.

Parameters

NAME	TYPE	DESCRIPTION
name	string	The name to reference the value
value	*	The value could be anything

Example

```
var api = $scope.widgetApis['book-categories'];
api.setState("version", 1);
```

`getState(name)`

Get data from the widget internal storage.

Parameters

NAME	TYPE	DESCRIPTION
name	string	Name of the state

Example

```
var api = $scope.widgetApis['book-categories'];
var version = api.getState("version"); // version: 1
```

`setTabCounterNumber(tabName, counterNumber)`

Set counter number (total list items) displayed on a tab header.

Example

```
var counterNumber = 12
$scope.widgetApis['cwgm-my-pagev2'].setTabCounterNumber('cwgm_tasks', counterNumber)
```

Workflow container widget

`triggerWorkflow(options)`

Trigger and load the associated workflow.

Parameters

PARAM	TYPE	DETAILS
options	object	[Optional] An object in the following format: `{ "caseId": <string>, // case to run the workflow "inputs": <object[]>, // inputs for running the workflow "parameters": <object>, // external data for the workflow "showHeaderCloseButton": <boolean>, // false to hide the close button "widgetId": <string>, // the tab container widget ID "container": <object>, // e.g. { containerId: "7e28d777-0723-424e-837d-ac39004337cd", containerType: "Page" } }`

Example

```
var api = $scope.widgetApis['edit-book-workflow-container'];
api.triggerWorkflow();
```

`closeWorkflow()`

Close the workflow container.

Example

```
var api = $scope.widgetApis['edit-book-workflow-container'];
api.closeWorkflow();
```

`setWorkflowParameters(caseId, inputs, parameters)`

Set the parameters to trigger workflow.

Parameters

NAME	TYPE	DESCRIPTION
caseId	string	The case to run the workflow. It is required if the workflow definition is **Edit Case**
inputs	object[]	[Optional] An array of data objects. All item must have the field `Id`
parameters	object	[Optional] Additional data to run the workflow

Example

```
var api = $scope.widgetApis['edit-book-workflow-container'];
var caseId = "413e1752-ed1a-4c87-96b9-ae1500ad5d5f";
api.setWorkflowParameters(caseId);
api.triggerWorkflow();
```

`setCaseId(caseId)`

Set the case ID to trigger workflow. It is required if the workflow definition is **Edit Case**.

Parameters

NAME	TYPE	DESCRIPTION
caseId	string	[Required] A case to run the workflow

Example

```
var api = $scope.widgetApis['edit-book-workflow-container'];
var caseId = "413e1752-ed1a-4c87-96b9-ae1500ad5d5f";
api.setCaseId(caseId);
api.triggerWorkflow();
```

`setInputs(inputs)`

Set inputs for triggering workflow.

Parameters

PARAM	TYPE	DETAILS
inputs	object[]	[Optional] An array of data objects. All item must have the field `Id`

Example

```
var api = $scope.widgetApis['edit-book-workflow-container'];
var inputs = [{ Id: "Book/55e601e6-8566-49e4-94fe-ae0500c677bb" }];
api.setInputs(inputs);
api.triggerWorkflow();
```

`setParameters(parameters)`

Set additional data for triggering workflow.

Parameters

PARAM	TYPE	DETAILS
parameters	object	[Optional] Additional data to run the workflow

Example

```
var api = $scope.widgetApis['edit-book-workflow-container'];
var parameters = {
    publishedYear: 2022
};
api.setParameters(parameters);
api.triggerWorkflow();
```

`workflowShellApi`

Access to the workflow shell API object.

Example

```
var api = $scope.widgetApis['edit-book-workflow-container'];
var workflowShellApi = api.workflowShellApi;
workflowShellApi.saveAndClose();
```

`show()`

Show the widget.

Example

```
var api = $scope.widgetApis['edit-book-workflow-container'];
api.show();
```

`hide()`

Hide the widget.

Example

```
var api = $scope.widgetApis['edit-book-workflow-container'];
api.hide();
```

Custom widget

`runWidgetAction(widgetActionName, doIds, data, refreshCallback, reloadDataCallback)`

Run the widget action.

Parameters

PARAM	TYPE	DETAILS
widgetActionName	string	[Required] Name of the widget action
doIds	object[]	[Optional] Inputs for running the action. The data item must have the `Id` field
data	object	[Optional] Additional data for running the action.
refreshCallback	function	[Optional] This method is called after an automatic workflow is run.
reloadDataCallback	function	[Optional] Calls after an UI workflow or a page action is run.

Example

```
var bookListApi = $scope.widgetApis['book-list'];
var newBook = {
    Name: "The Da Vinci Code",
    Author: "Dan Brown",
    Year: "2003"
};

bookListApi.runWidgetAction("Add book", null, newBook);
```

`setQueryParameters(parameters)`

Set the parameter for the widget data query.

Parameters

NAME	TYPE	DESCRIPTION
parameters	object	A key/value object

Example

```
var api = $scope.widgetApis['book-list'];
var parameters = {
    Author: "Dan Brown"
};

api.setQueryParameters(parameters);
```

`setActionCustomInputs(widgetActionName, doIds)`

Use this method to programmatically set input for widget actions, which is useful when you want to load data to dynamically reflect the change in UI.

Parameters

NAME	TYPE	DESCRIPTION
widgetActionName	string	Name of the action
doIds	object[]	Inputs for running the action

Example

```
var api = $scope.widgetApis['book-list'];
var bookIds = [
    {
        "Id": "Book/3ae2aa8c-7937-4d02-ba17-ae0500c5b426"
    }
];

api.setActionCustomInputs("Edit book", bookIds);
```

`show()`

Show the widget.

Example

```
var api = $scope.widgetApis['book-list'];
api.show();
```

`hide()`

Hide the widget.

Example

```
var api = $scope.widgetApis['book-list'];
api.hide();
```

`setState(name, value)`

Save data to the widget internal storage.

Parameters

NAME	TYPE	DESCRIPTION
name	string	The name to reference the value
value	*	The value could be anything

Example

```
var bookListApi = $scope.widgetApis['book-list'];
bookListApi.setState("version", 1);
```

`getState(name)`

Get data from the widget internal storage.

Parameters

NAME	TYPE	DESCRIPTION
name	string	Name of the state

Example

```
var bookListApi = $scope.widgetApis['book-list'];
var version = bookListApi.getState("version"); // version: 1
```

`getQueryOptions()`

Get the query options for loading data.

```
{
    QueryParameters:, // the parameters used for getting data
    Pagination: {
        PageSize:, // the number of items in each page
        PageIndex: // the current page number
    },
    clauses:, // the clauses for filtering data
    sortOptions:, // the current sorting options
    Search: // the full text search text
}
```

Example

```
var bookListApi = $scope.widgetApis['book-list'];
var queryOptions = bookListApi.getQueryOptions();
```

`setQueryOption(option)`

Set the data query options.

Parameters

NAME	TYPE	DESCRIPTION
option	object	An object in the format described at getQueryOptions.

Example

```
var bookListApi = $scope.widgetApis['book-list'];
var option = {
    Search: "Advanture"
};
bookListApi.setQueryOption(option);
```

Implement custom API for custom widget

Casewhere allows you to implement custom API for custom widgets. In the below example, we're going to add two new methods: `getABook` and `getAllBooks`

```
$scope.$watch('widgetApi', function () {
    if (!$scope.widgetApi) {
        $scope.widgetApi = {};
    }

    $scope.widgetApi.getABook = function(name) {
        return $scope.books[name];
    };

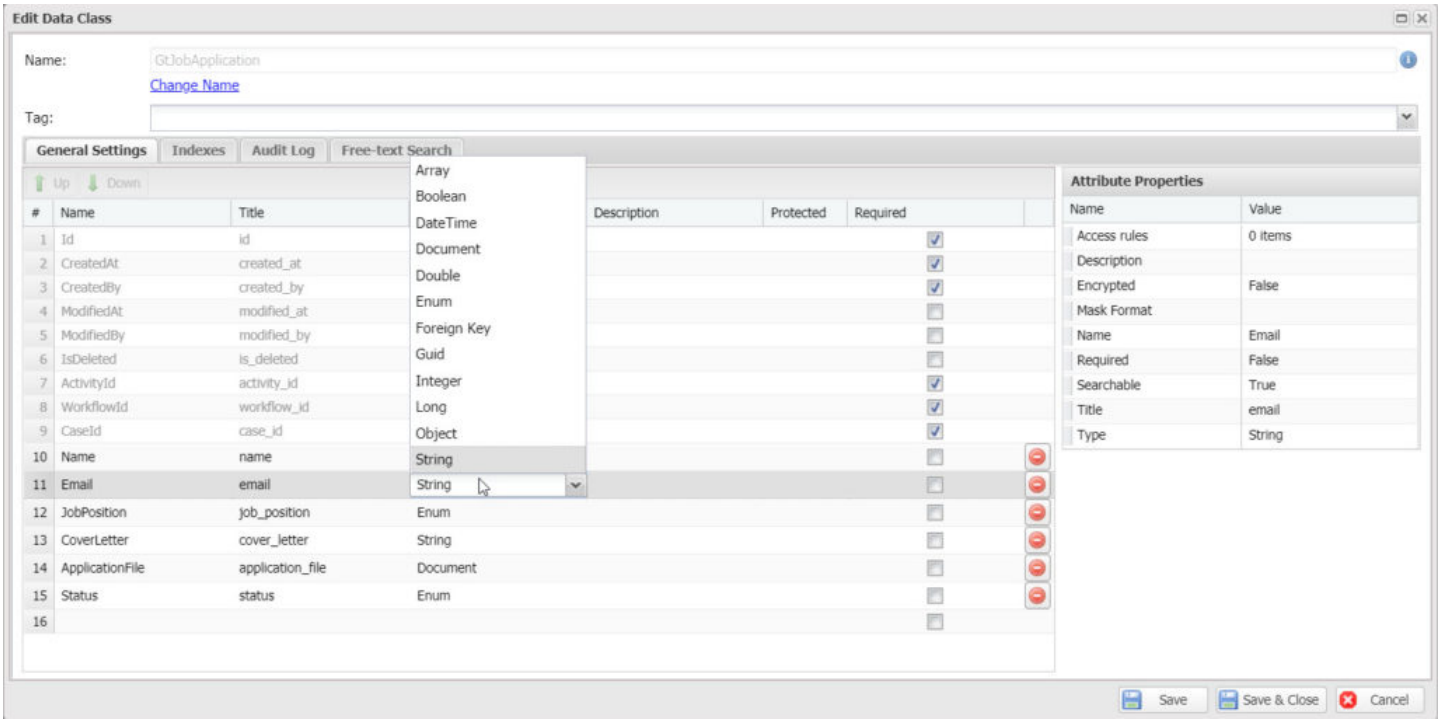
    $scope.widgetApi.getAllBooks = function() {
        return $scope.books;
    };
});
```

Data class overview

Casewhere is not designed for any specific business domain, so it does not offer a fixed, built-in data model. Instead, as a Casewhere designer, you will define how data should be stored by creating data classes and specifying the relationships between them. Below is the information you need to provide when adding a new data class in Casewhere:

- The name of the data entity you are working on, such as Employee, JobApplication, Book, Movie, etc.
- How the data entity is related to others, if applicable. Casewhere allows you to define relationships using foreign keys.
- The attributes that describe your data entity include:
    - What is the data type? Casewhere offers a variety of attribute types to give you maximum flexibility in modeling your business scenarios, no matter how complex.
    - Is the attribute required or optional? If required, the information must be provided when creating data objects.
    - Does the attribute contain sensitive data that needs to be encrypted?
    - Can users perform a free-text search on this attribute?
    - Many more.

All data objects created from a data class will have the same data structure. For example, consider the data class in the picture below:

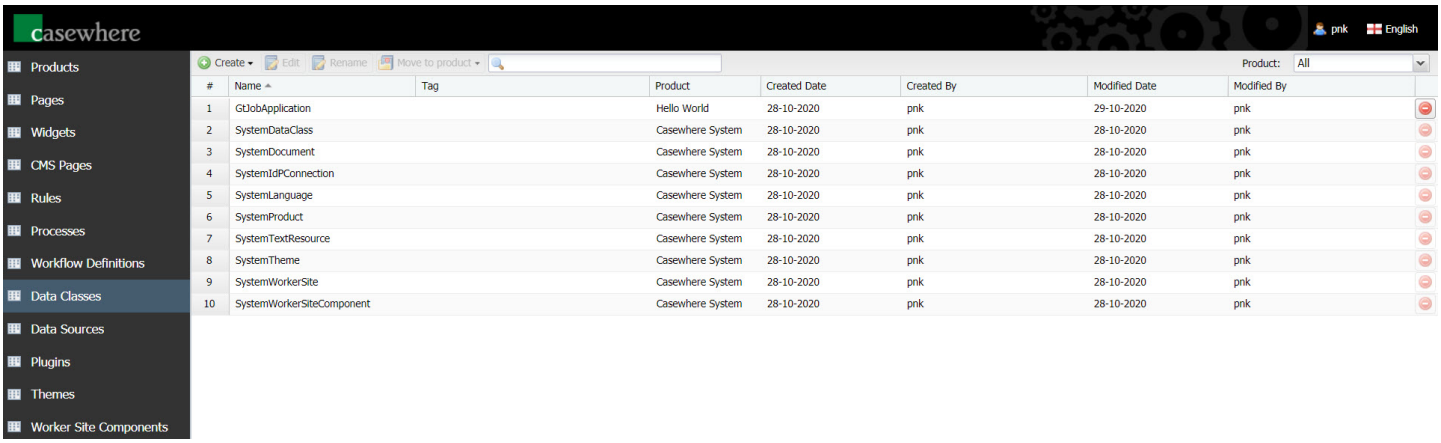


It will have data objects created in the database as shown below:

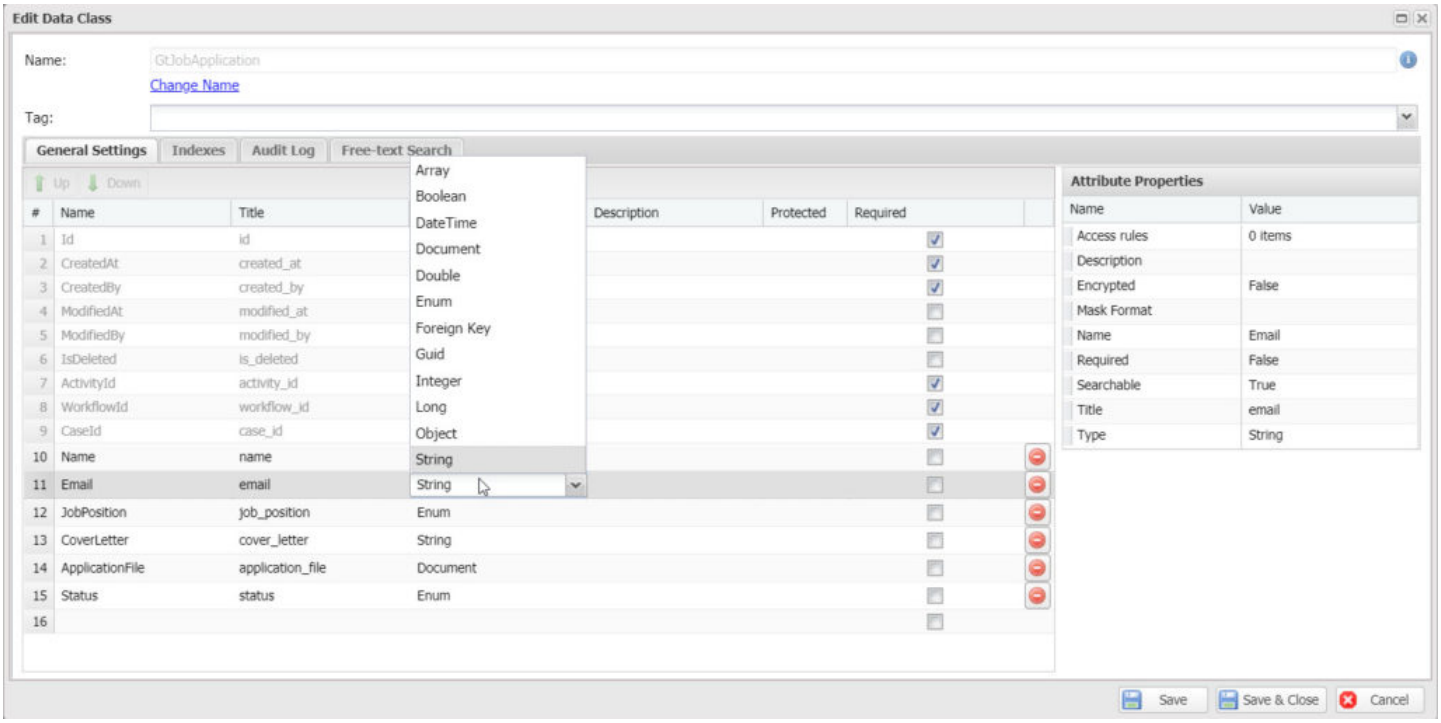
```
{
    "_id" : "GtJobApplication/5222dda9-6bea-465d-ace3-ac69008638ca",
    "IsDeleted" : false,
    "CreatedAt" : ISODate("2020-11-04T08:08:41.207Z"),
    "CreatedBy" : "Anonymous User - 1b936074-6988-4583-b316-ac6900721b79",
    "Version" : 2,
    "ModifiedAt" : ISODate("2020-11-04T08:10:53.877Z"),
    "ModifiedBy" : "pnk",
    "ActivityId" : LUUID("8e8ae0a8-6580-4e43-82a2-ac690085f3ea"),
    "WorkflowId" : LUUID("924884b8-d923-9847-bd69-ac690085f3e3"),
    "CaseId" : LUUID("672acea7-7b83-7541-9ae6-ac690085f3a4"),
    "Name" : "PNK",
    "Email" : "pnk@globeteam.com",
    "JobPosition" : "developer",
    "CoverLetter" : "The Model-View-Controller (MVC) architectural pattern separates an application into three main groups of components: Models, Views, and Controllers. This pattern helps to achieve separation of concerns. Using this pattern, user requests are routed to a Controller which is responsible for working with the Model to perform user actions and/or retrieve results of queries. The Controller chooses the View to display to the user, and provides it with any Model data it requires.\n\nThe following diagram shows the three main components and which ones reference the others:",
    "ApplicationFile" : {
        "_t" : "DocumentInfo",
        "_id" : LUUID("6f1d68af-ebcb-d842-93d3-528a21864ed4"),
        "FileName" : "Lorem ipsum dolor sit amet.pdf"
    },
    "Status" : "approved",
    "CaseState" : "Created"
}
```

Data class management

In Casewhere Admin, navigate to **Data Classes** to design data model for your solution. From here, you can create, edit and manage data classes like any product resource.



The data class editor is where you define the attributes for your data class and configure other advanced settings such as optimizing data queries using indexes, configuring audit log, enabling free-text search for your users, etc.



Data attributes

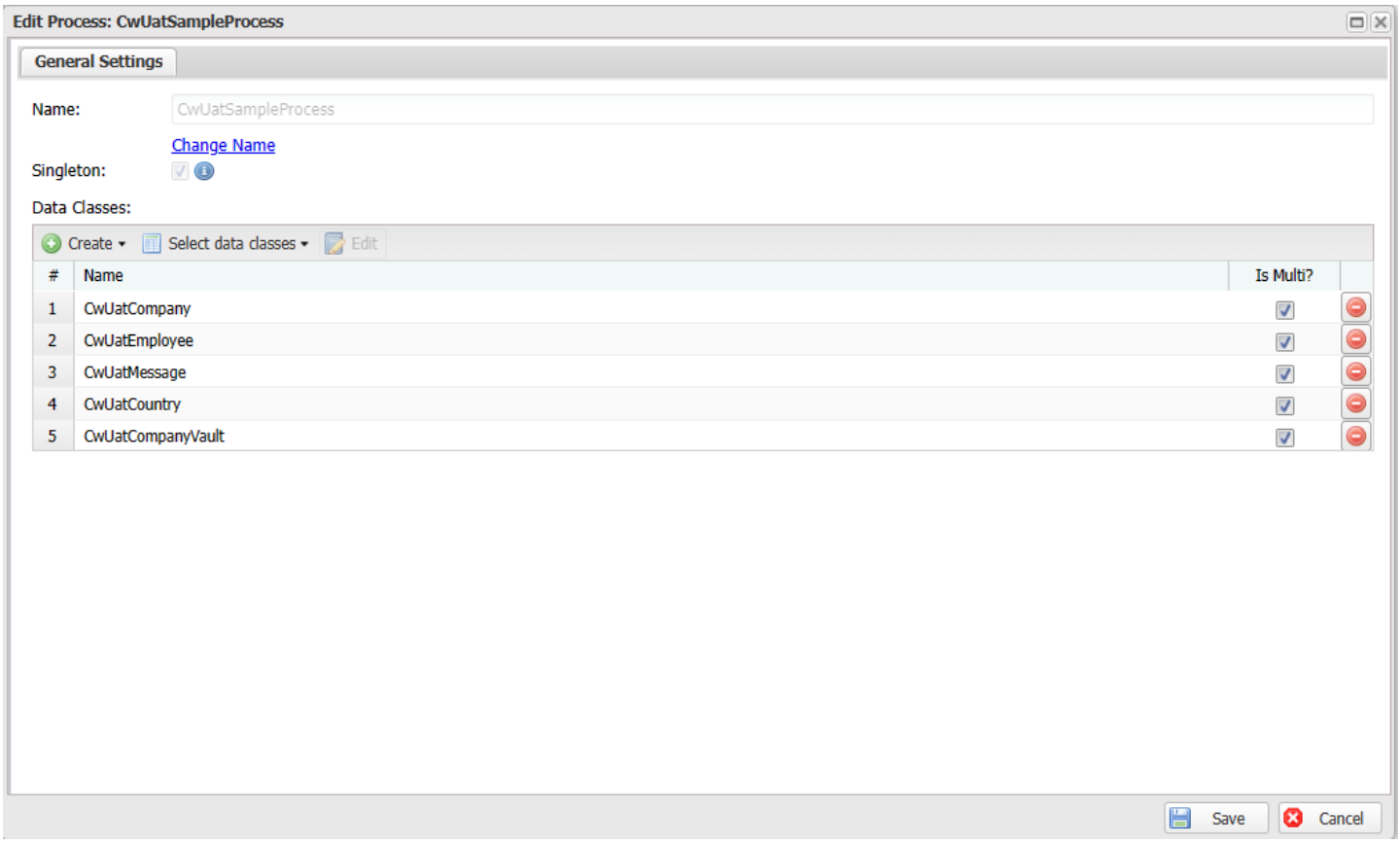
Casewhere supports the following attribute types:

NAME	DESCRIPTION
Boolean	Represents a Boolean value, which can be either `true` or `false`.
String	Represents a sequence of zero or more Unicode characters.
Guid	Represents a globally unique identifier (`Guid`).
Integer	Represents 4-byte integer numbers from `-2,147,483,648` to `2,147,483,647`.

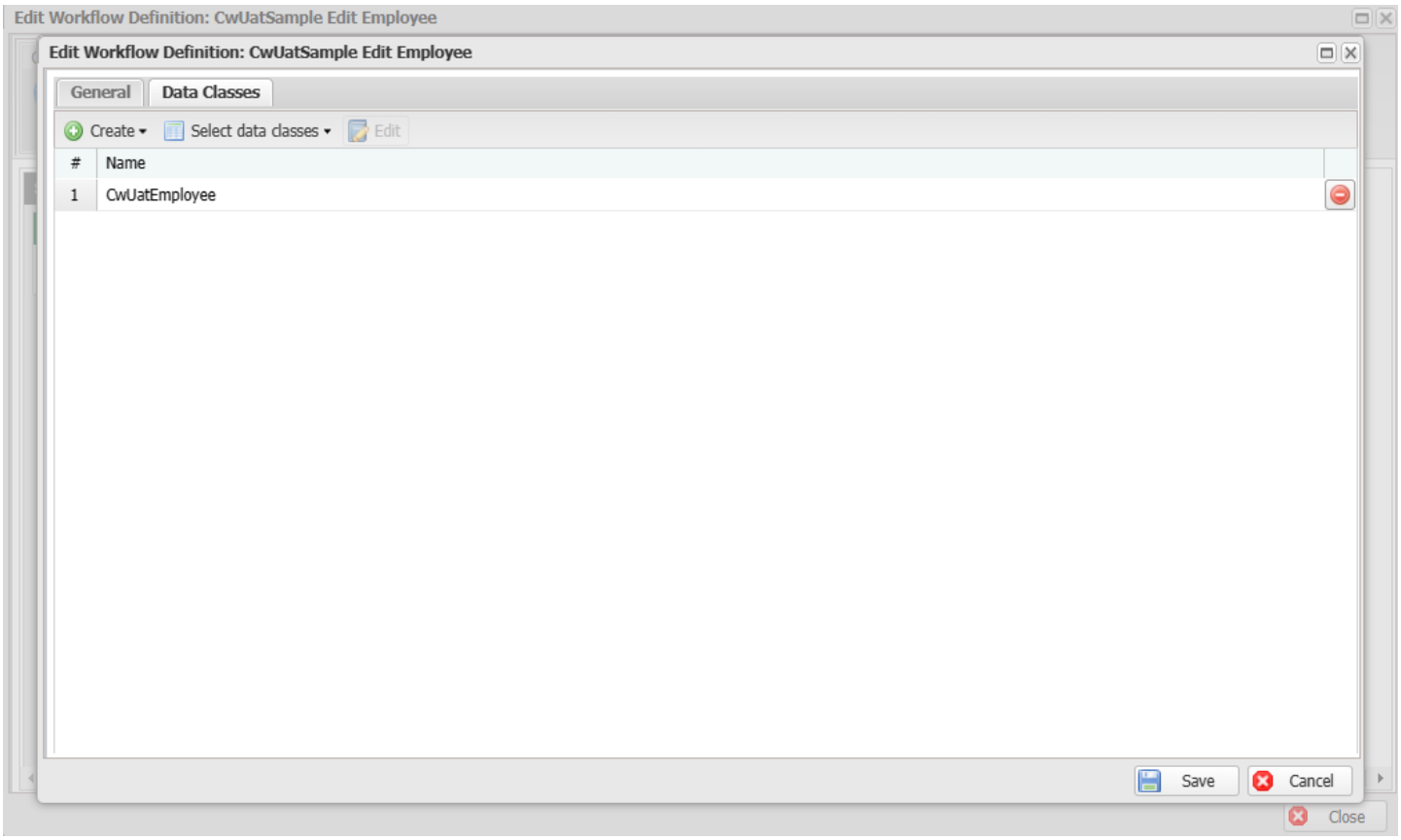
NAME	DESCRIPTION
Long	Represents 8-byte integer numbers from `-9,223,372,036,854,775,808` to `9,223,372,036,854,775,807`
Double	Represents 8-byte real numbers from `±5.0 × 10−324` to `±1.7 × 10308`
DateTime	Represents a value of time, typically expressed as a date and time of day.
Enum	Represents a combination of choices. You will need to define the options as part of the configuration.
Document	Represents a binary file.
Object	Represents a dynamic object.
Foreign Key	Represents a reference to another data class.
Array	Represents an array of elements. Element type could be any singular data type supported by Casewhere. An array of array currently is not supported yet.

Working with data classes

There are two main places where we add data classes; namely processes and workflows definitions. When data classes are added to a process, it specifies the type of data that can be managed in all cases of that process.



Workflows have the responsibility of administering the data of cases, so by adding data classes to a workflow definition, we are making it easy to know, which workflow definitions can be used for which processes.

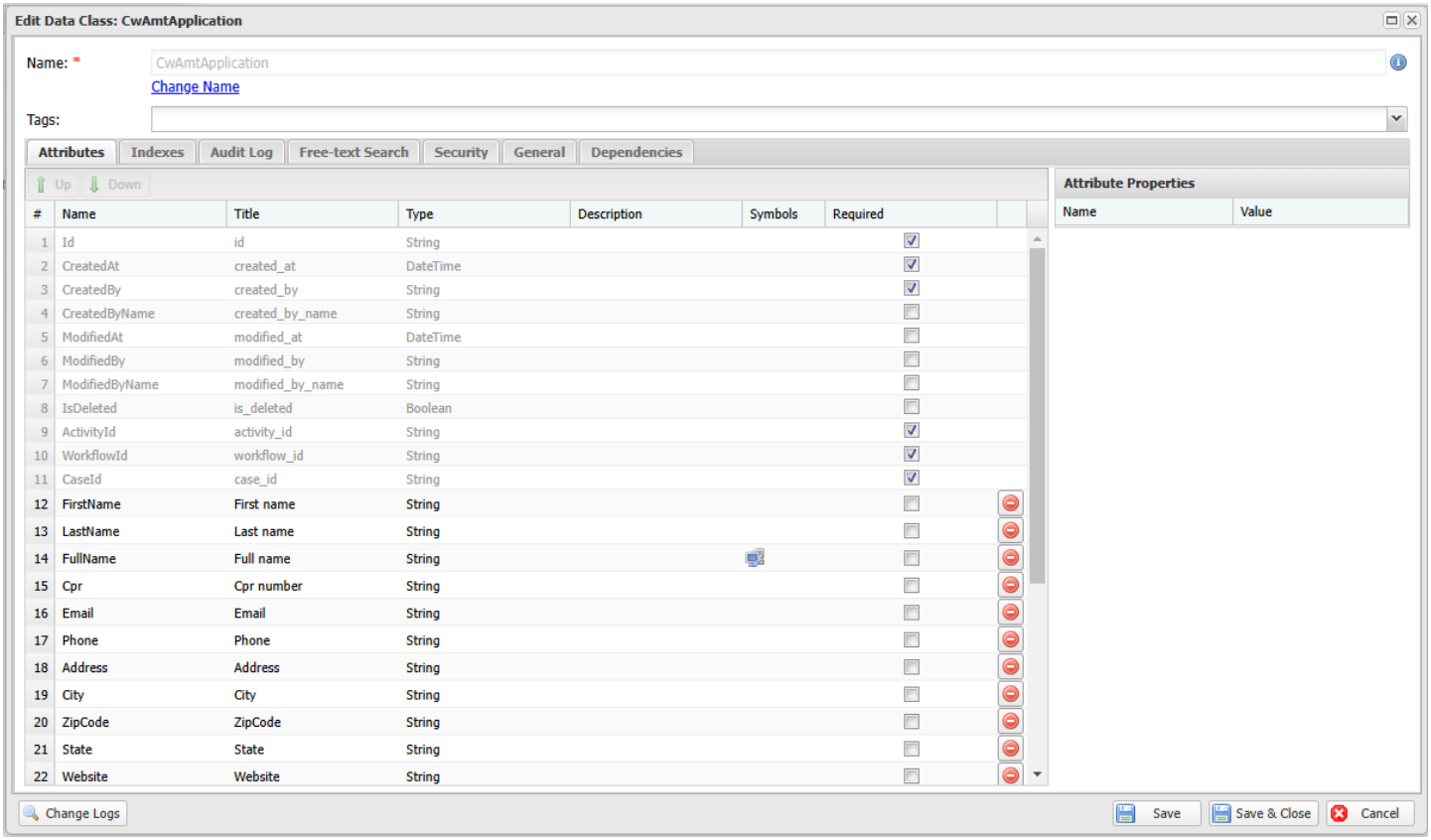

Edit Workflow Definition: CwUatSample Edit Employee
Edit Workflow Definition: CwUatSample Edit Employee
General
Data Classes
Create
Select data classes
Edit
#
Name
1
CwUatEmployee
Save
Cancel
Close

Data objects

Data storage

Casewhere stores information in data objects, each constructed by a set of data attributes follows the definition in the corresponding data class.

The screenshot below illustrates how a data class is configured in Casewhere:



Each data object is then respresented by a MongoDB document, shown as below:

```
{
    "_id" : "CwAmtApplication/eebf8351-adc8-4b33-be3f-b1f6009dff03",
    "IsDeleted" : false,
    "CreatedAt" : ISODate("2024-09-25T09:35:14.779+0000"),
    "CreatedBy" : "CWRuntime",
    "CreatedByName" : "CWRuntime",
    "Version" : NumberInt(1),
    "ModifiedAt" : ISODate("2024-09-25T09:35:15.741+0000"),
    "ModifiedBy" : "CWRuntime",
    "ModifiedByName" : "CWRuntime",
    "ActivityId" : CSUUID("257A1B62-62A8-47FE-AE69-B1F6009DFED2"),
    "WorkflowId" : CSUUID("A39DAA11-7FAC-429A-A6D3-B1F6009DFED1"),
    "CaseId" : CSUUID("BC4271BE-9199-4BAD-AB14-B1F6009DFED1"),
    "FirstName" : "Amber",
    "LastName" : "Steuber",
    "FullName" : "Amber Steuber",
    "Cpr" : "2212681914",
    "Email" : "Amber.Steuber@hotmail.com",
    "Phone" : "(949) 693-3229",
    "Address" : "05703 Kenya Island",
    "City" : "Rodriguezmouth",
    "ZipCode" : "60496-1955",
    "State" : "New Jersey",
    "Website" : "camila.info",
    "Avatar" : "https://cloudflare-
ipfs.com/ipfs/Qmd3W5DuhgHirLHGVixi6V76LhCkZUz6pnFt5AJBiyvHye/avatar/32.jpg",
    "ApplicationTitle" : "Tempore et dolorem aut sed rerum temporibus cupiditate fuga.",
    "ApplicationContent" : "Modi qui ea blanditiis iusto repellat illum. Nam asperiores veniam et repudiandae
cumque. Voluptatum sed facere voluptatem velit quam sunt. Quos quo est et soluta saepe debitis maxime eum.
Magni sapiente eum. Est in dolorum voluptas cupiditate porro consequatur reprehenderit odio.\n\nNam neque
magnam assumenda id blanditiis. Et voluptatem voluptates et. Similique dolores deserunt temporibus magnam
eveniet in dolor. Sequi est non dolor ea similique eveniet at. Aut in eos accusantium sit qui fuga aut dolor
mollitia. Nemo ea sapiente doloribus repudiandae ab impedit non non natus.\n\nMagni nam excepturi iste quia
deserunt velit rerum exercitationem. Aperiam aut numquam aut ut sed eos neque ad. Voluptatibus nulla et.
Facilis itaque ut ex eos delectus et et laborum.",
    "ApplicationAmount" : 2248.98,
    "Status" : "Approved",
    "SubmittedDate" : ISODate("2021-06-16T13:03:44.764+0000"),
    "DecisionDate" : ISODate("2023-02-11T17:32:08.536+0000"),
    "ApplicationID" : "0000251",
    "CaseState" : "Created"
}
```


Data attributes

There are two main types of data attributes:

- System-managed attributes: These are automatically set by Casewhere.
  - **Id**: A unique string generated by Casewhere when a new data object is created. The format is `{DataClass}/{Guid}`. For example: `Application/ff87b1a8-fa67-4fbf-a0b7-8b7f07d63670`.
  - **CreatedAt**: The UTC timestamp indicating when the data object was created.
  - **CreatedBy:** The identifier of the user who created the data object. This value is determined based on the claim mapping in the **IdP Connection**.
  - **CreatedByName**: The display name of the user who created the data object. This value is determined based on the claim mapping in the **IdP Connection**.
  - **ModifiedAt**: The UTC timestamp indicating when the last time the data object was updated.
  - **ModifiedBy**: The identifier of the last user who updated the data object. This value is determined based on the claim mapping in the **IdP Connection**.
  - **ModifiedByName**: The display name of the last user who updated the data object. This value is determined based on the claim mapping in the **IdP Connection**.

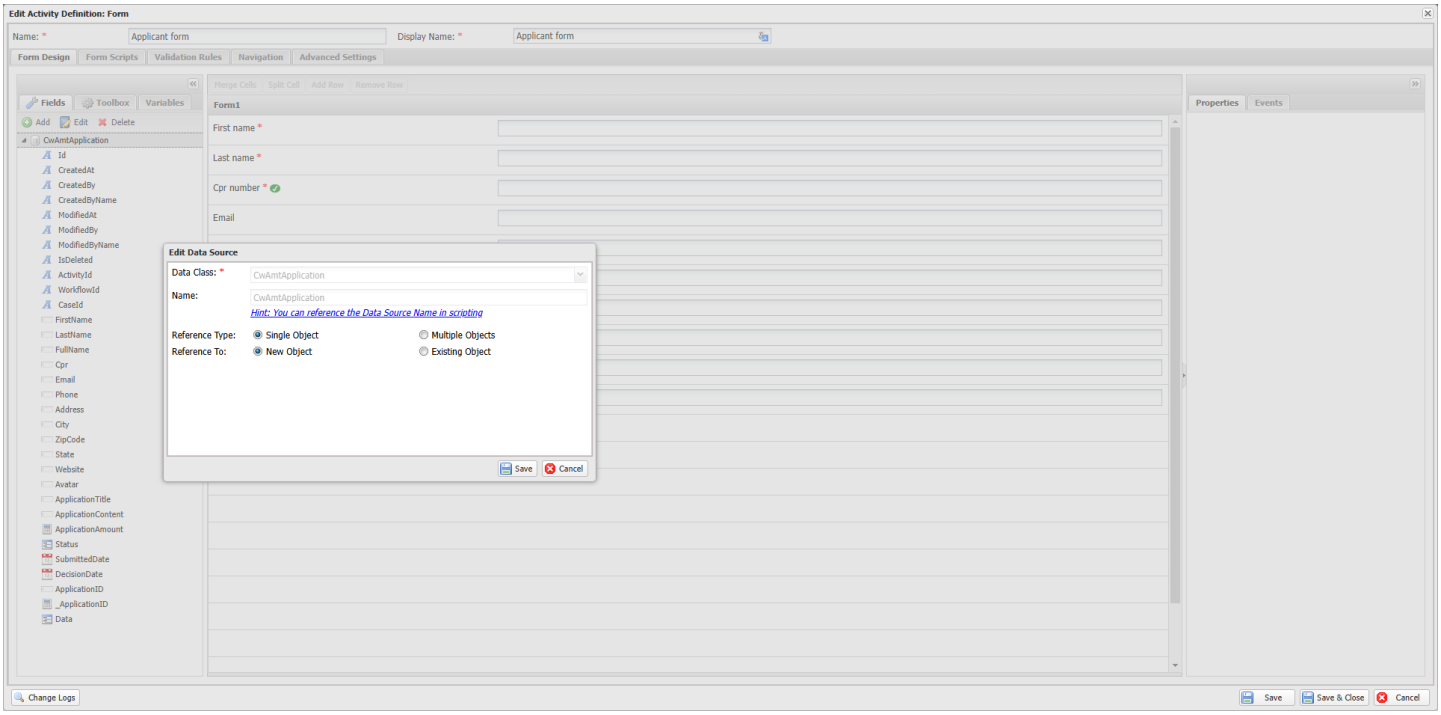
- **IsDeleted**: To determine if the data object is deleted or not.
- **ActivityId**: The ID of the workflow activity that created the data object.
- **WorkflowId**: The ID of the workflow that created the data object.
- **CaseId**: The ID of the case to which the data object belongs.

- Custom attributes: These are defined by your solution, which are responsible for setting their values. You can find the complete list of data class attribute types here.

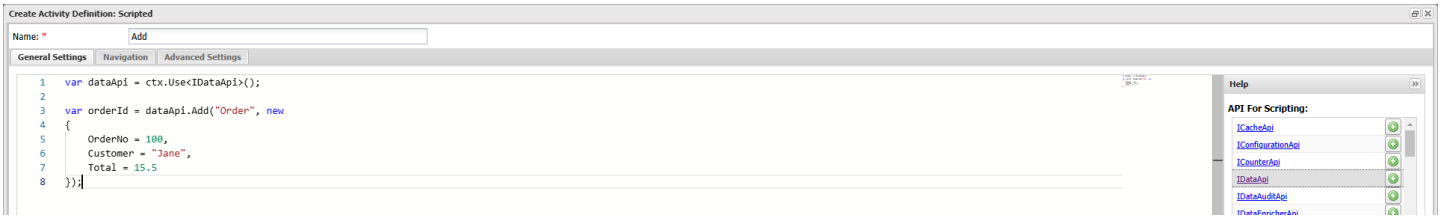
Create and update data objects

All data objects in Casewhere must be created through a workflow, either via a form activity or a scripted activity.

Form Activity: When end users submit data through a form, Casewhere creates or updates data objects for the associated data class configured in the form's data source, provided that all data validations are successfully passed. Learn more about form activity here.



Scripted Activity: Casewhere creates and updates data objects by executing C# scripts and provides a comprehensive set of APIs for interacting with these objects. For more details, refer to the **DataApi** documentation. Learn more about scripted activity here.



Delete data objects

Casewhere supports two types of deleting data, soft deletion and hard deletion.

Soft deletion: The data object is marked as deleted but remains in the database. This is the default deletion method. Below is an example of deleting data using a Scripted Activity.

```
var dataApi = ctx.Use<IDataApi>();
dataApi.Delete(ctx.Input.Id);
```

Hard deletion: Casewhere provides a separate API for permanently deleting data objects. This API is only available when the configuration is enabled in the Worker API `web.config`.

```
<add key="db:EnableHardDelete" value="false" />
```

The deletion script might look like the following example:

```
var dataApi = ctx.Use<IDataApi>();
dataApi.HardDelete(customerId);
```

Load and search data objects

If you load data through widgets or form activities, Casewhere automatically retrieves the data based on your data source configuration. In this post, we focus more on manual loading data using Scripted Activity.

Example 1: Load a single data object by ID

```
var dataApi = ctx.Use<IDataApi>();
var application = dataApi.Load(applicationId);

Log.Info("Applicant name: {ApplicantName}", application["ApplicantName"]);
Log.Info("Project title: {ProjectTitle}", application.Get<string>("ProjectTitle"));
```

Example 2: Search all active employees in a given company

```
var dataApi = ctx.Use<IDataApi>();
var filter = FilterBuilder.Create()
    .Eq("CompanyId", ctx.Input.Id)
    .Eq("Active", true)
    .Build();

var result = dataApi.Search(DataObjectApiQuery
    .For("Employee") // Data Class Name
    .FilterBy(filter));

Log.Info("Total: {TotalItems}", result.TotalItems);
foreach(var emp in result.Data)
{
    Log.Info("Employee {Name} {Skills}",
         emp.Get<string>("Name")
         emp.GetList<string>("Skills"));
}
```

Example 3: Iterate through a huge collection

```
var dataApi = ctx.Use<IDataApi>();

var filter = FilterBuilder.Create().Eq("Active", true).Build();
var query = DataEnumerationQuery.For("Employee").FilterBy(filter);

foreach (var employee in dataApi.Enumerate(query))
{
    Log.Info("Employee {FirstName} {LastName}",
         emp.Get<string>("FirstName")
         emp.Get<string>("LastName"));
}
```

Read data objects

The Casewhere DataApi returns data objects of the type `DynamicDataObject`. This type encapsulates the raw data and provides convenient methods for accessing and reading it. Below are some commonly used methods:

Example 1: Read data attributes using `T Get<T>(string key)`

```
var name = dataObject.Get<string>("Name");
var amount = dataObject.Get<double>("Amount");
var isActive = dataObject.Get<bool>("IsActive");
```

Example 2: Read list of data attributes using `IList<T> GetList<T>(string key)`

```
var skills = dataObject.GetList<string>("Skills");
var ratings = dataObject.GetList<double>("Ratings");
var categoryIds = dataObject.GetList<int>("CategoryIds");
```

Example 3: Verify if a data object has value for a given attribute

```
// Returns true if the attribute value is neither null nor empty.
var hasSkillSet = dataObject.Has("Skills");
```

Example 4: Verify if a data attribute exists or not. The attribute value can be null or empty.

```
// Returns true if the attribute is present, regardless of its value.
var hasSkillSet = dataObject.HasKey("Skills");
```

Configure audit logging

Introduction

Casewhere audit logging provides essential information about data changes made within the system, which can be crucial for diagnosing potential or actual security issues. This feature is designed to address key questions in a user-friendly manner:

- Who made the change?
- What was changed?
- When the changes were made?
- Who can access the log data?

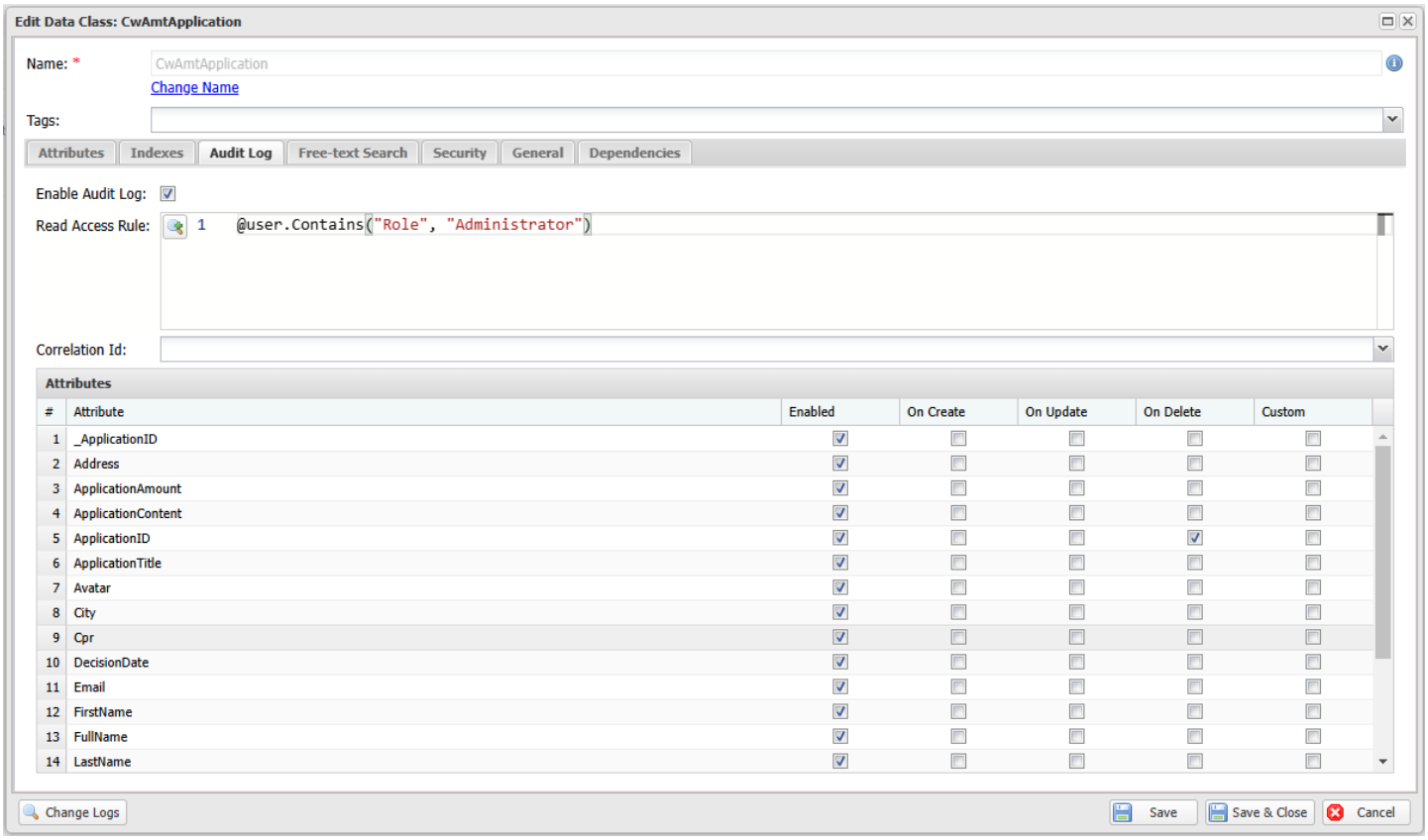
Logging is performed at the field level, allowing you not only to configure audit logs for specific data classes but also to track changes in the values of selected fields.

Configuration

Enable audit log in data class

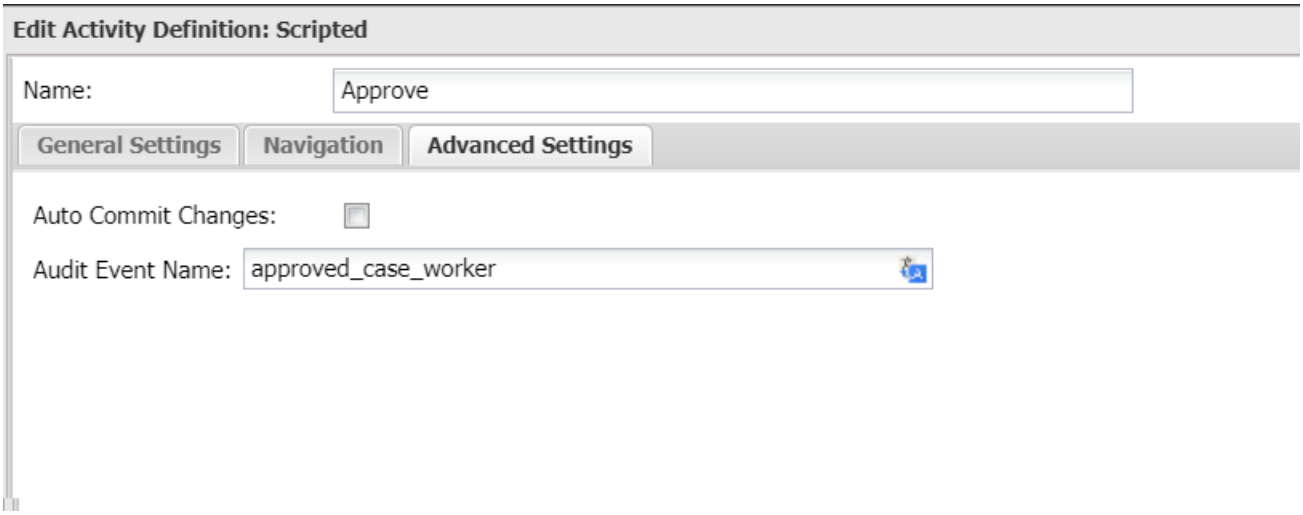
In the **Data Class** editor, navigate to the tab **Audit Log**, and configure the following:

- **Read Access Rule**: to decide who can access the log data. See more here for instructions on how to configure access rules.
- **Correlation Id**: to identify related log entries.
- **Attributes**: select fields to enable the audit log. You can control who can access the log data by specifying the access rule.
    - Log data can be enriched with additional information to make it easier to interpret. For instance, logging the `Application ID` and `Application Title` when an application is deleted allows users to identify what was removed, as it can otherwise be difficult to determine. This can be done by enabling the **On Delete** option for the relevant attributes.



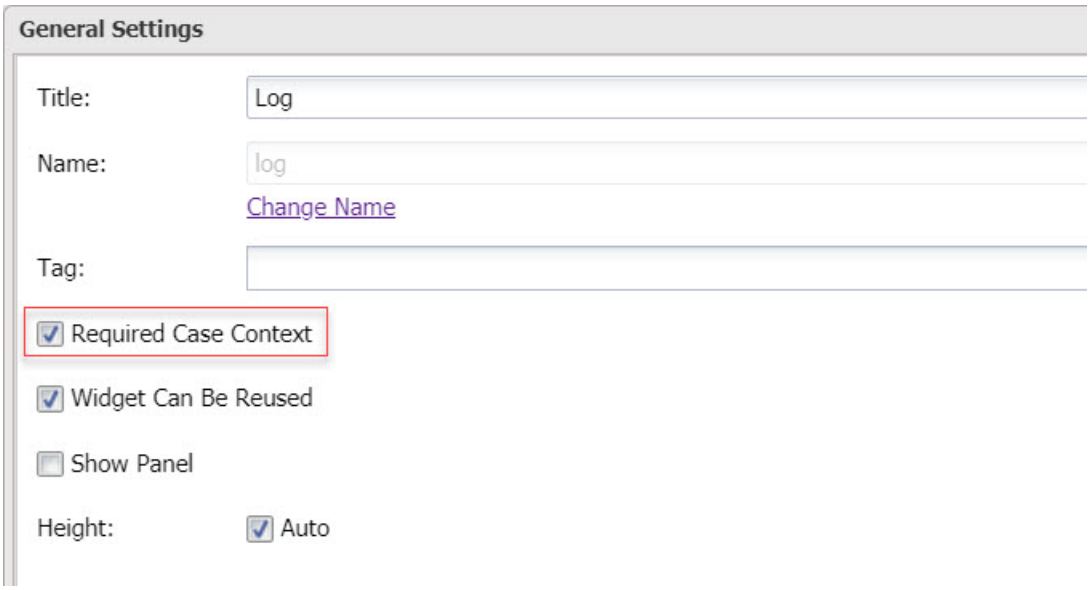
When the audit log setting is enabled for a data class, all changes to its data objects will be logged. By default, Casewhere uses the activity name for the log event, but this can be customized. To do so, in the **Workflow Definition**, go to the **Advanced Settings**

tab and enter a custom **Audit Event Name**.



View audit log in worker site

Create a page with a custom widget **Log**. The widget needs a case id to query data, so the setting **Required Case Context** must be enabled.



The widget template is as follows:

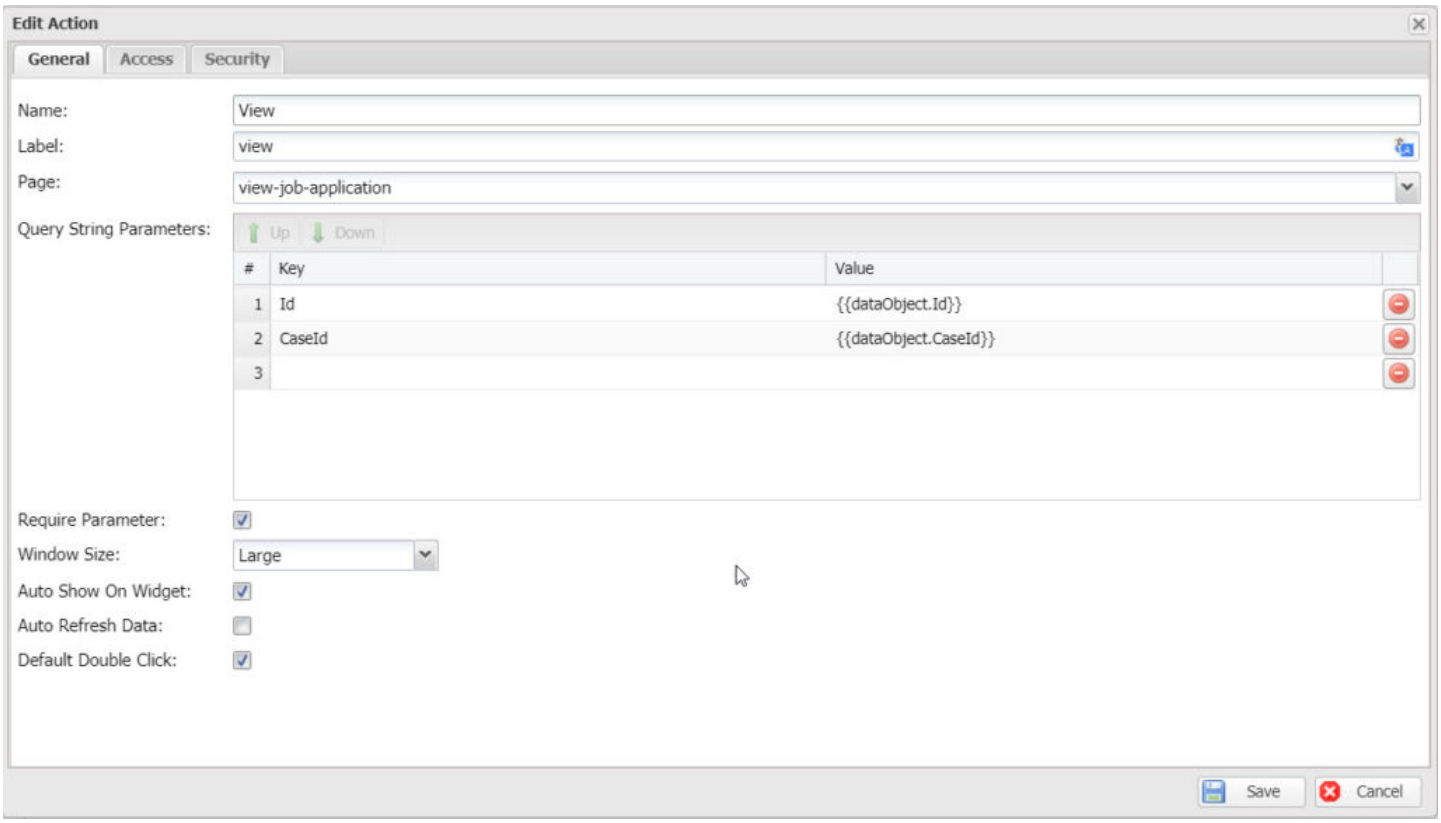
- Html

```
<div class="row flex-grow">
  <div class="col-12">
      <cw-audit-log data-config="auditLogConfig"></cw-audit-log>
  </div>
</div>
```

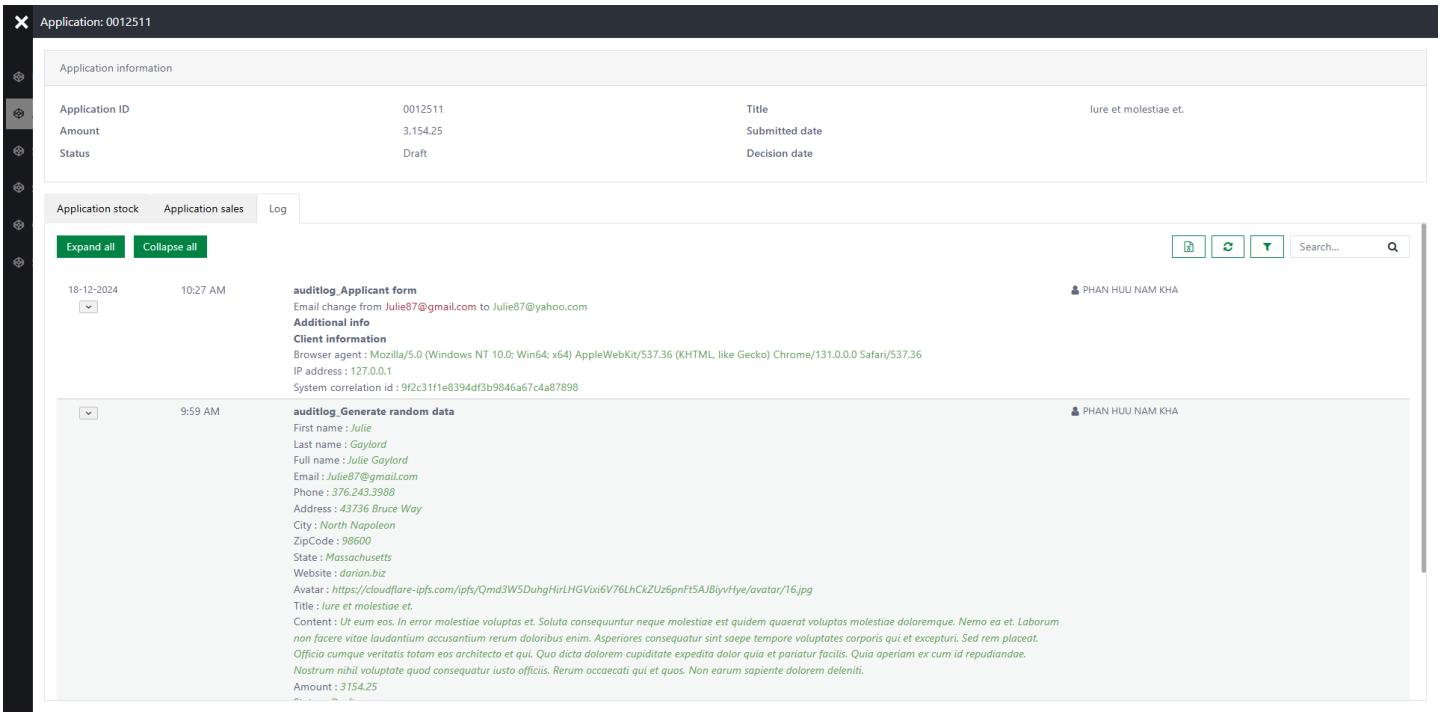
- Javascript: in the scope of **customWidgetController,** append the following code:

Select a case from Application List to open the log viewer using **Open Page** action. Following parameters are required for the log viewer to work:

- Id: {{dataObject.Id}}
- CaseId: {{dataObject.CaseId}}



The logging page will appear as shown below:

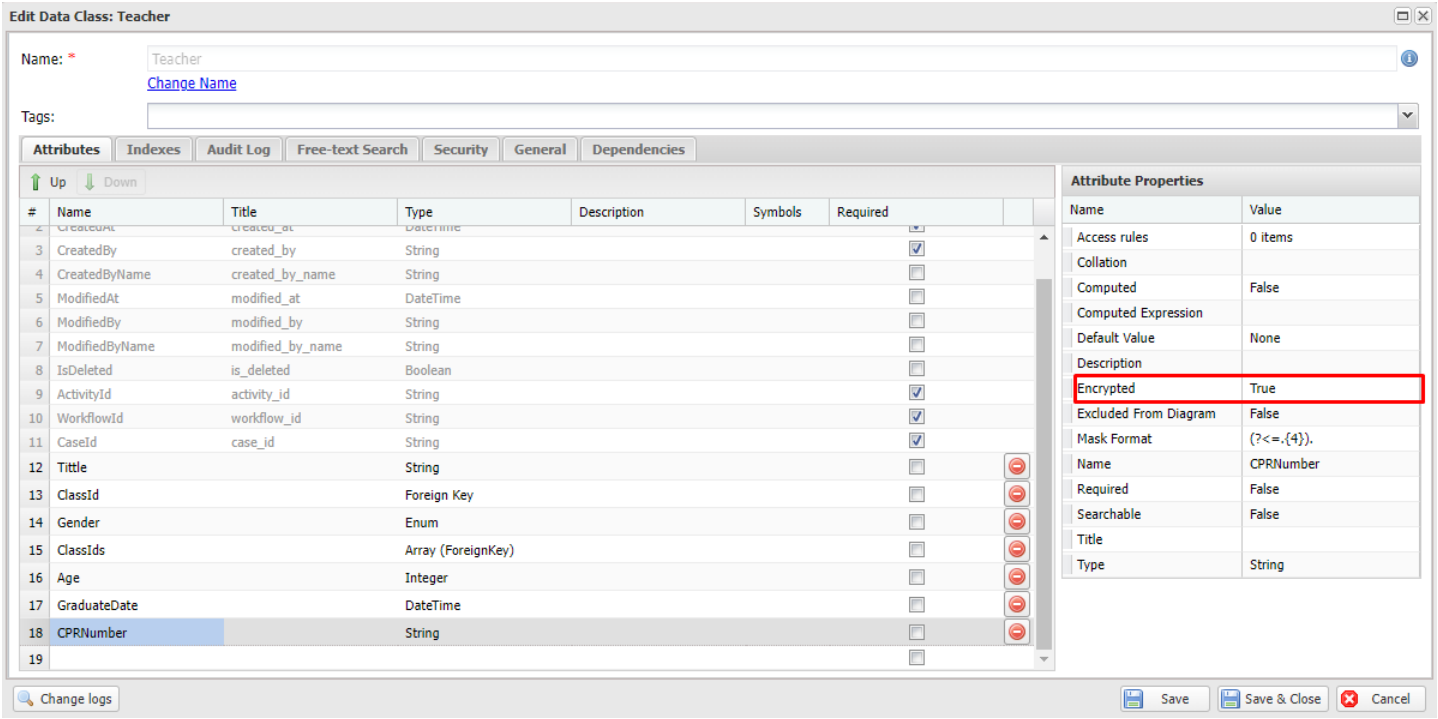


How to protect the sensitive attributes from viewing by user

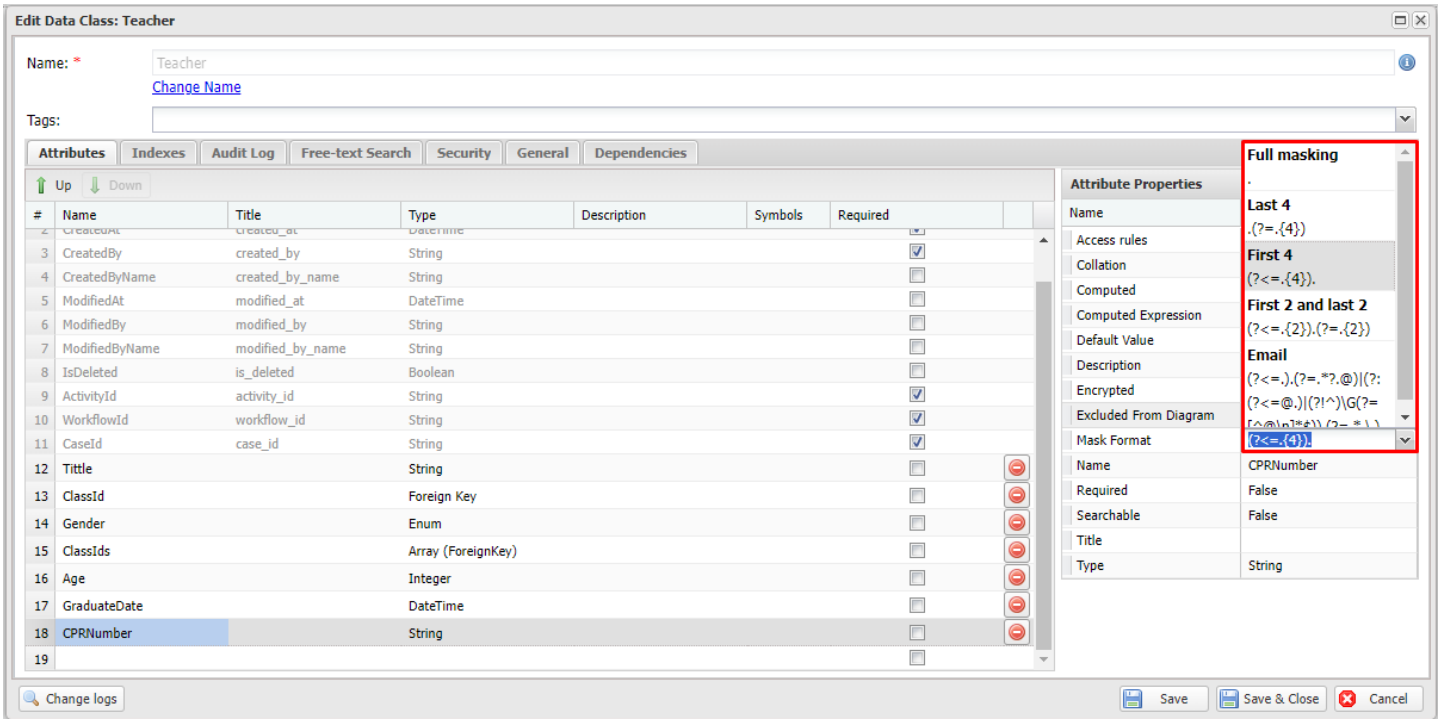
In real-world scenarios, sensitive data like credit card numbers or CPR numbers should not be visible to users. Casewhere includes a feature to protect this data from being displayed on the audit log page.

For example, if there is a sensitive attribute called **CPRNumber** in the **Teacher** data class that needs protection, you can safeguard it as follows:

1. Open the **Teacher** data class in the configuration. Enable the **Encrypted** property of the **CPRNumber** attribute.



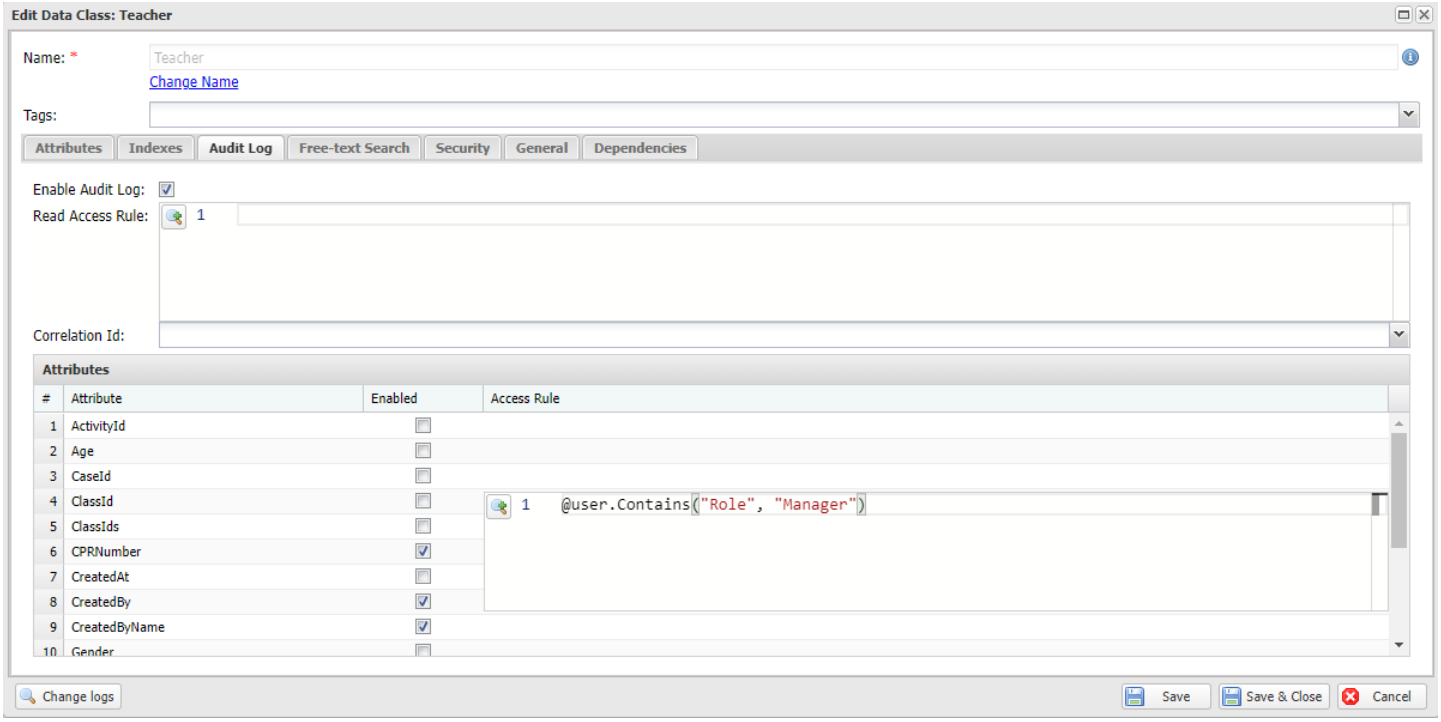
1. Then, choose the **MaskFormat** for it. The mask format is the property to configure how we want to display the value of the encrypted attribute on the audit log page. In this example, we choose the **First 4** option. With this option, if the value is `123456789`, it will be displayed as `1234*****`.



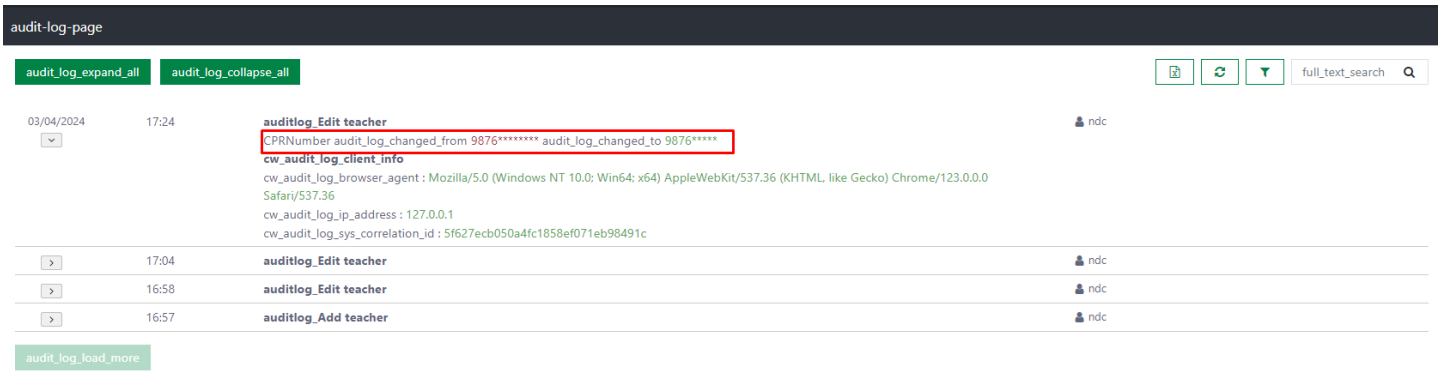
1. Next, navigate to the **Audit Log** tab. Ensure we have already checked the **Enabled** option of the **CPRNumber** property. Then we have to define an access rule for the encrypted attribute to determine when this attribute should be protected. In the image below, we define an access rule to ensure only the user who has the manager role can view the whole value of the attribute on the audit log page.

```
@user.Contains("Role", "Manager")
```

You can learn more about the Casewhere rule here.



1. Open the audit log viewer page, observe the audit log contains the **CPRNumber** value. With the user does not have the Manager role, the value of the encrypted attribute has been protected properly.



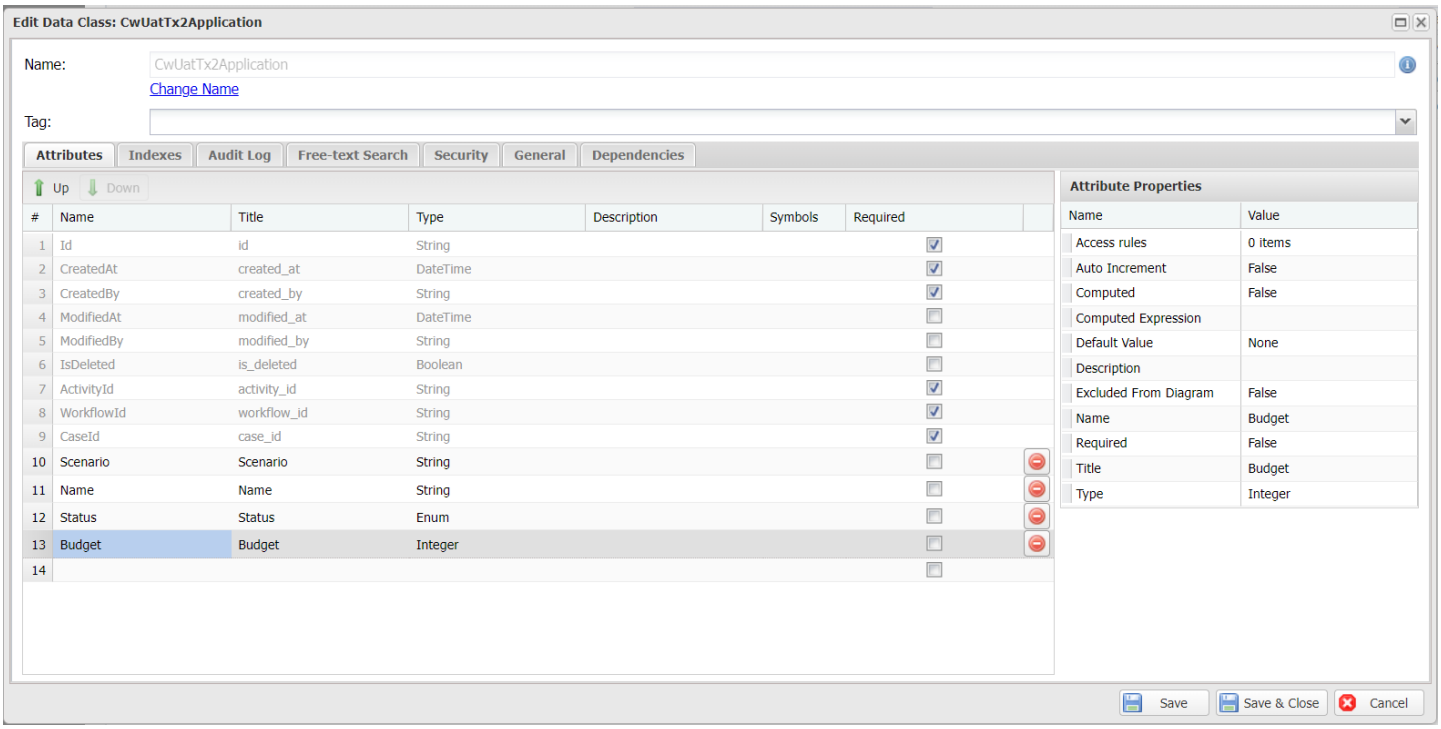
Data validation

Before persisting data objects to DB, Casewhere will validate data against the configuration in the data class and the data attributes. If the validation fails, Casewhere will throw an error and reject the data.

General validation

Data conversion

When persisting a data object, Casewhere will try to convert the data fields to the corresponding data attributes. If for some reasons the data conversion fails, Casewhere will throw an error and reject the data. Take the `Application` data class below as an example:



If you try to run this script in a workflow,

```
var dataApi = ctx.Use<IDataApi>();
var id = dataApi.Add("CwUatTx2Application", new
{
    Budget = "test"
});
```

you will get an exception `FormatException`.

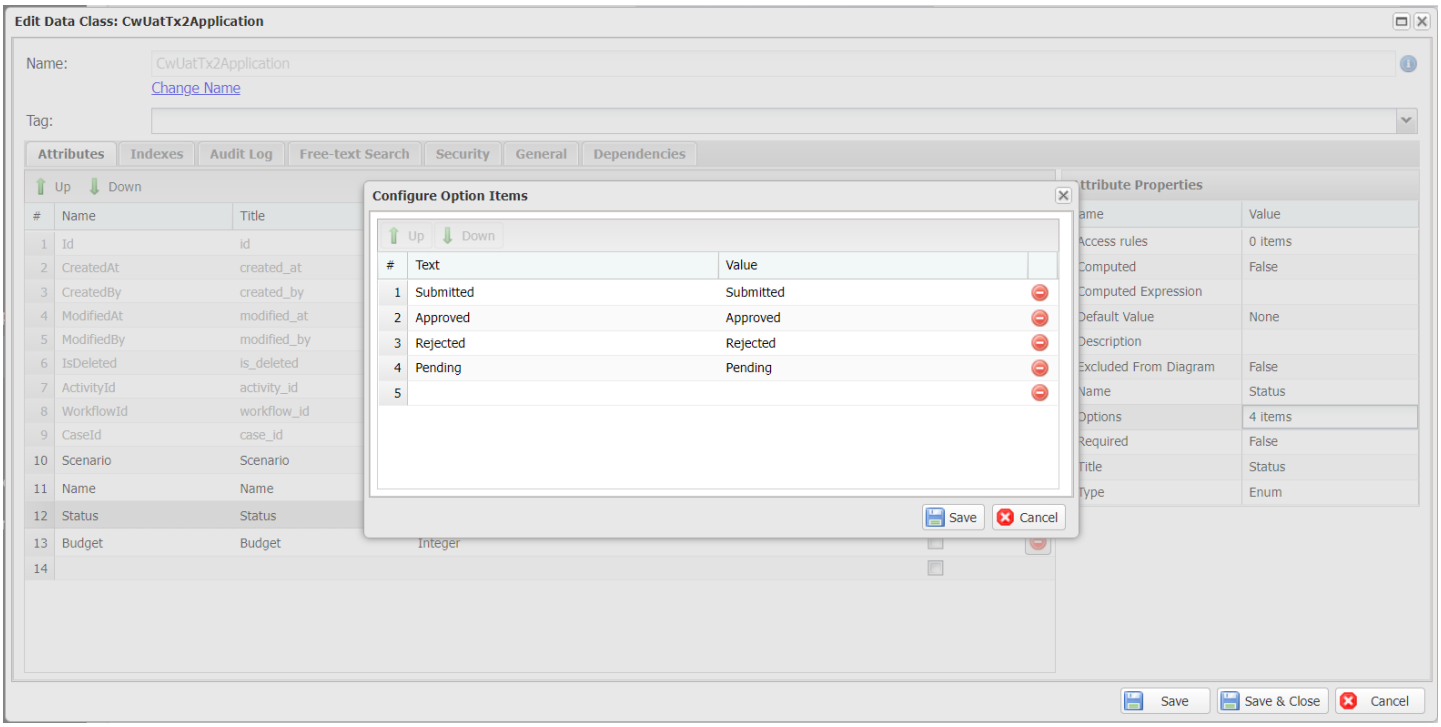
Required

All data attributes in Casewhere have a property named **Required**. If the property is set to `true` for a data attribute, a value must be specified for this attribute when Casewhere creates a new data object. Otherwise, it will throw an error and reject the data.

Attribute-specific validation

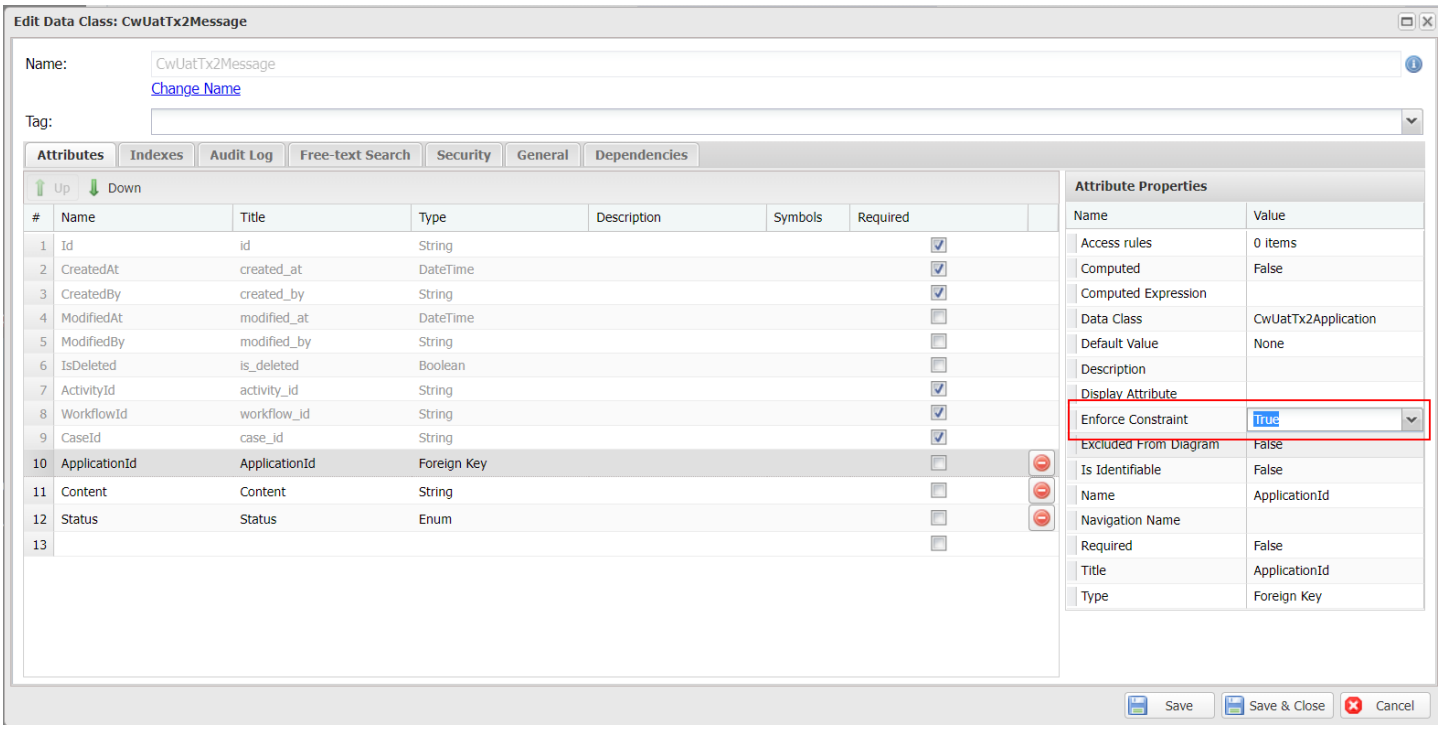
Enum

Casewhere will reject all data persisting requests that attempt to set a value outside of the **Enum** predefined values.



Foreign Key

When the **Enforce Constraint** is turned on for a **Foreign Key** attribute, Casewhere will verify the data object reference. Casewhere will throw an error and reject the data if the referenced data object does not exist. This feature is available since `v2.7.2`.



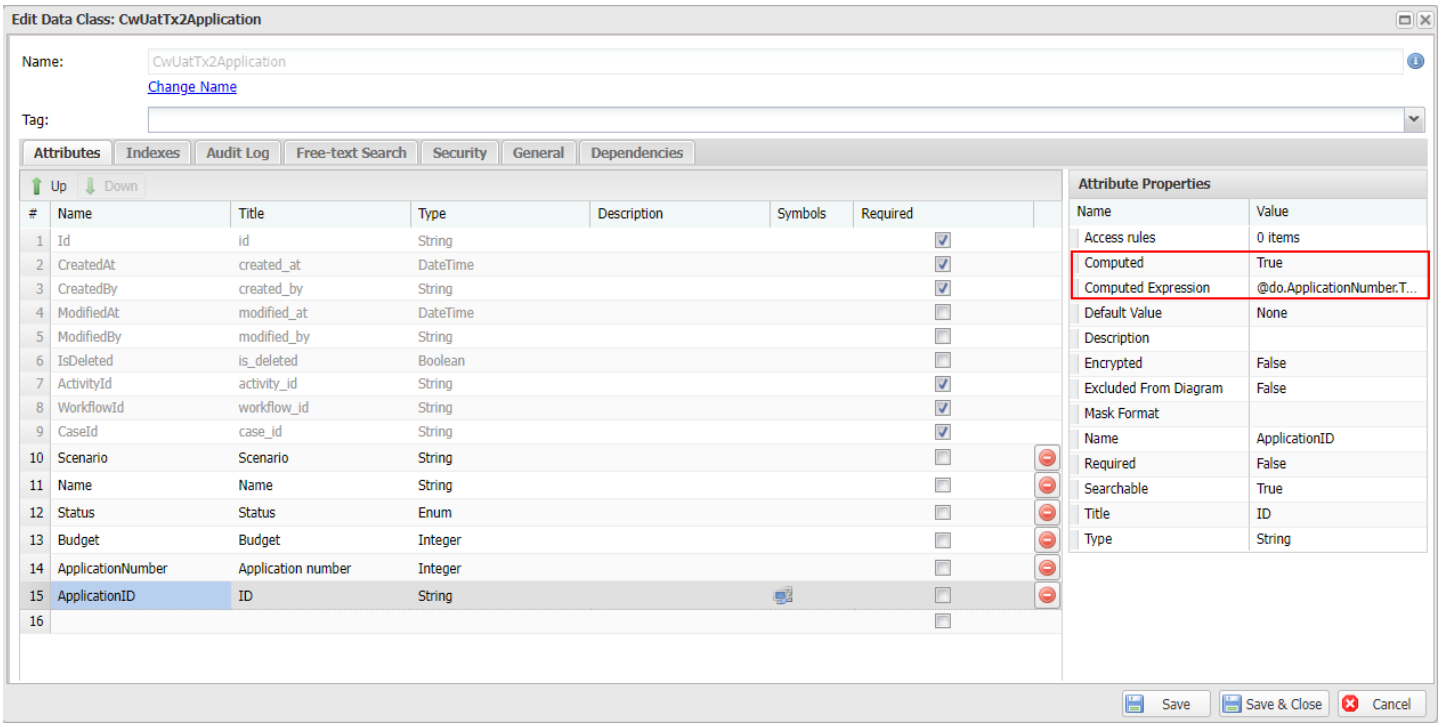
Computed attribute

> This feature is available in version `2.7.2` or later

Like many storage solutions, Casewhere supports computed attributes, for which, instead of accepting values set by applications, Casewhere will compute values from given expressions. Casewhere still persists computed attributes in the database, so features like indexing, searching still work as designed for these attributes.

Configuration

In the data class editor, select an attribute and set **Computed** to `True`. It's possible to enable **Computed** for any data attribute.



After turning on the **Computed** setting, you will need to provide a **Computed Expression**, which Casewhere will use to compute the value. The expression is a simple C# expression, accepting the keyword `@do`, which is used for accessing the data object attributes. Please note that you can only compute data from other existing data attributes.

Example 1: Compute full name from first name and last name

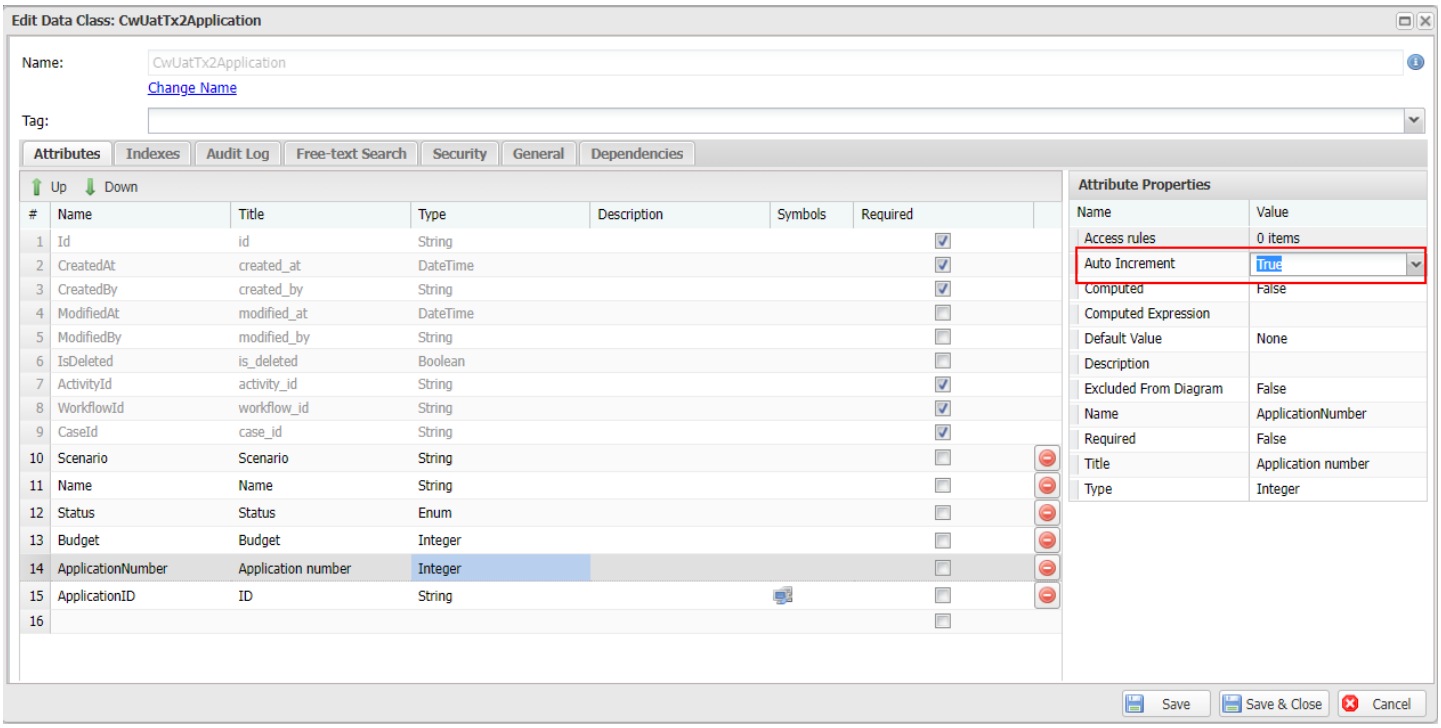
```
@do.FirstName + ", " + @do.LastName // "Jone, Doe"
```

Example 2: Compute a friendly id from an auto-increment value

```
@do.ApplicationNumber.ToString().PadLeft(5, '0') // "00018", "00019", ...
```

Auto-increment attribute

Auto-increment is a special computed attribute, which is applied for **Integer** and **Long** only. In the data class editor, select an **Integer** or **Long** attribute and set **Auto Increment** to `True`. When the setting is turned on, Casewhere will assign a new unique number when a new data object is created. The starting value is 1, and it will increment by 1 for each new data object.


Edit Data Class: CwUatTx2Application
Name:
CwUatTx2Application
Change Name
Tag:
Attributes
Indexes
Audit Log
Free-text Search
Security
General
Dependencies
Up
Down
Name Title Type Description Symbols Required
1 Id id String
2 CreatedAt created_at DateTime
3 CreatedBy created_by String
4 ModifiedAt modified_at DateTime
5 ModifiedBy modified_by String
6 IsDeleted is_deleted Boolean
7 ActivityId activity_id String
8 WorkflowId workflow_id String
9 CaseId case_id String
10 Scenario Scenario String
11 Name Name String
12 Status Status Enum
13 Budget Budget Integer
14 ApplicationNumber Application number Integer
15 ApplicationID ID String
16
Attribute Properties
Name Value
Access rules 0 items
Auto Increment True
Computed False
Computed Expression
Default Value None
Description
Excluded From Diagram False
Name ApplicationNumber
Required False
Title Application number
Type Integer
Save
Save & Close
Cancel

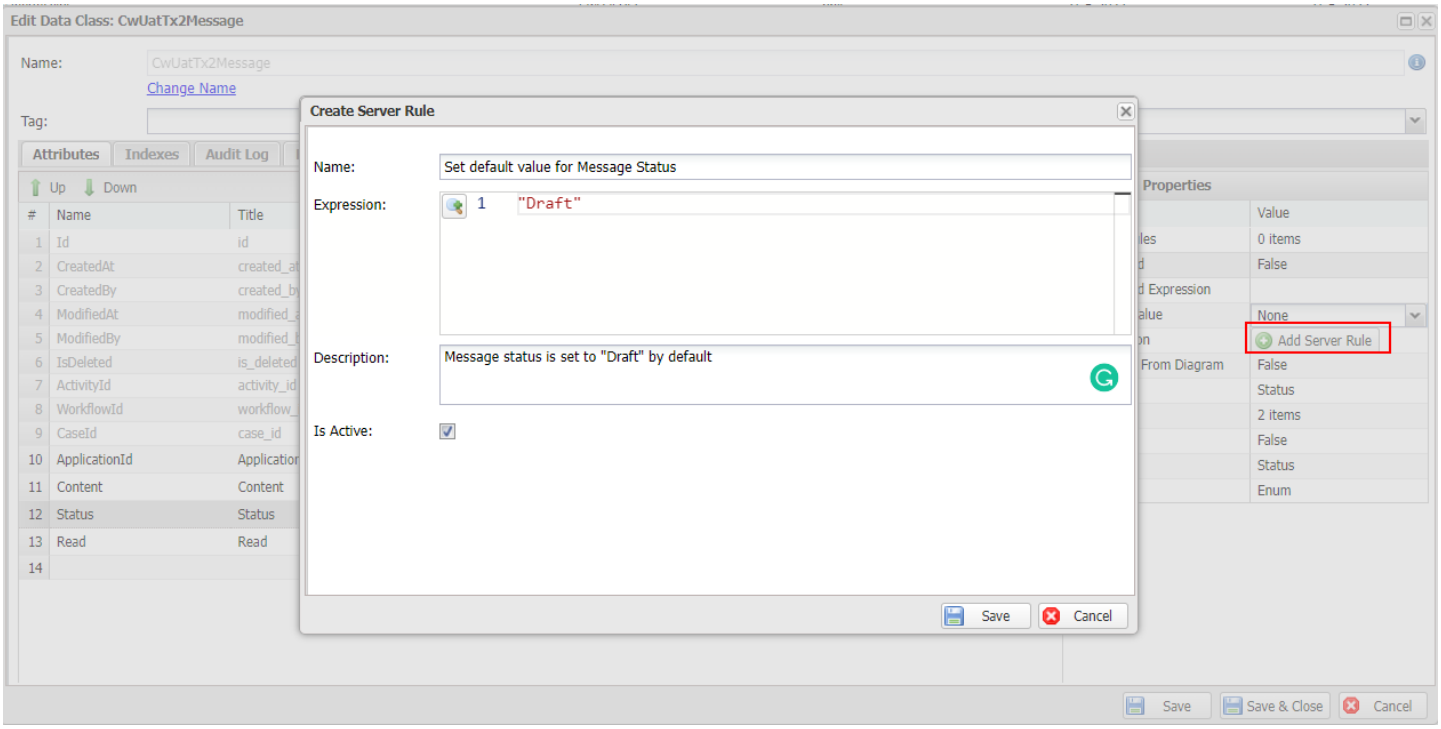
Attribute default value

> This feature is available in version `2.7.2` or later

Object creation, in many cases, is complex. Generally, repeating the object creation logic in different workflow definitions is not a good practice because we might forget to update this logic when it changes. Casewhere allows configuring object creation logic by specifying rules for setting default values for data attributes.

Configuration

Casewhere uses scalar-function rules to calculate default values. In the data class editor, select the data attribute you want to configure default value, then select a rule or writing a new one.



Please note that if you specify a value for the attribute when creating a new object, Casewhere will persist this value and ignore the rule. In the below example, the created message will have status `Sent` instead of the default value `Draft`.

```
var dataApi = ctx.Use<IDataApi>();
dataApi.Add("CwUatTx2Message", new
{
    Status = "Sent",
    Content = $"Application {ctx.Input.Name} is {ctx.Input.Status}"
});
```

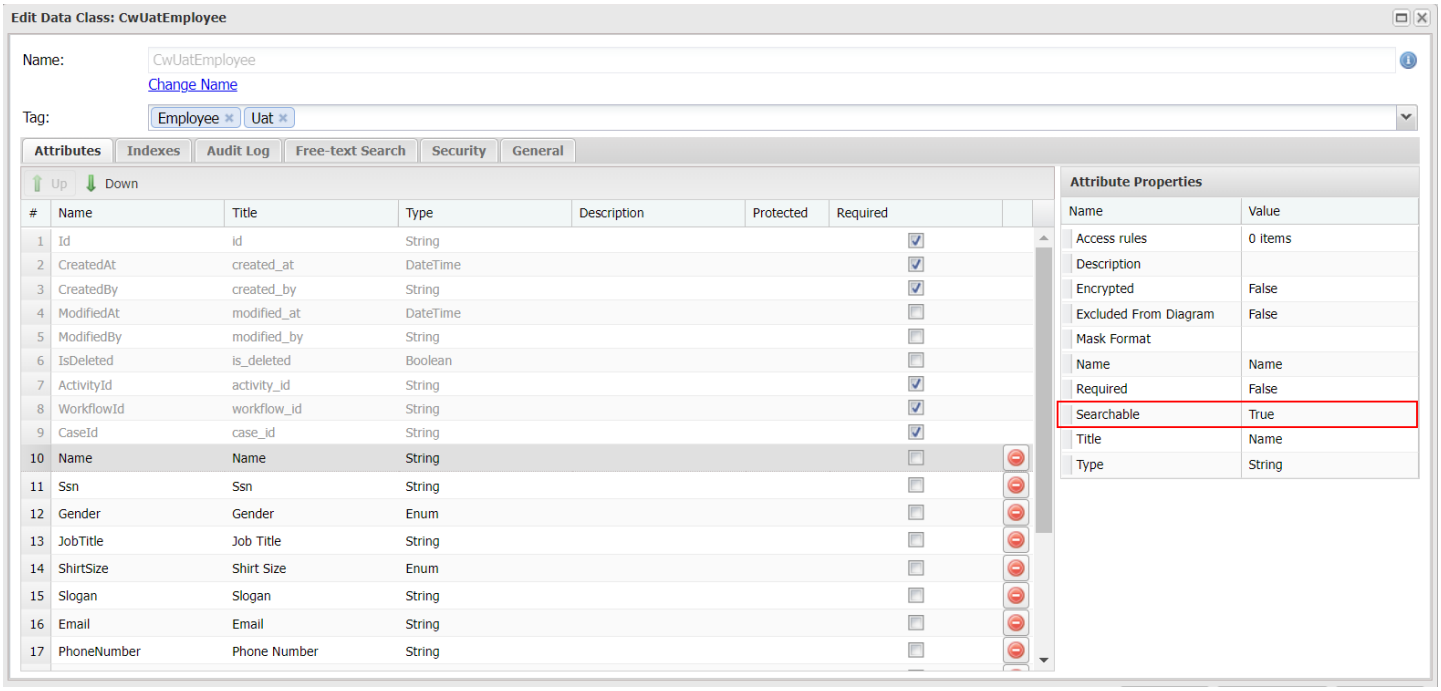
Free-text search

Free-text search is a technique for searching a single document or a collection in a free-text database. In a free-text search, a search engine examines all of the words in every stored document as it tries to match search criteria.

Free-text search is enabled in Casewhere by default. There are also advanced settings so you can enrich and customize your search data easily.

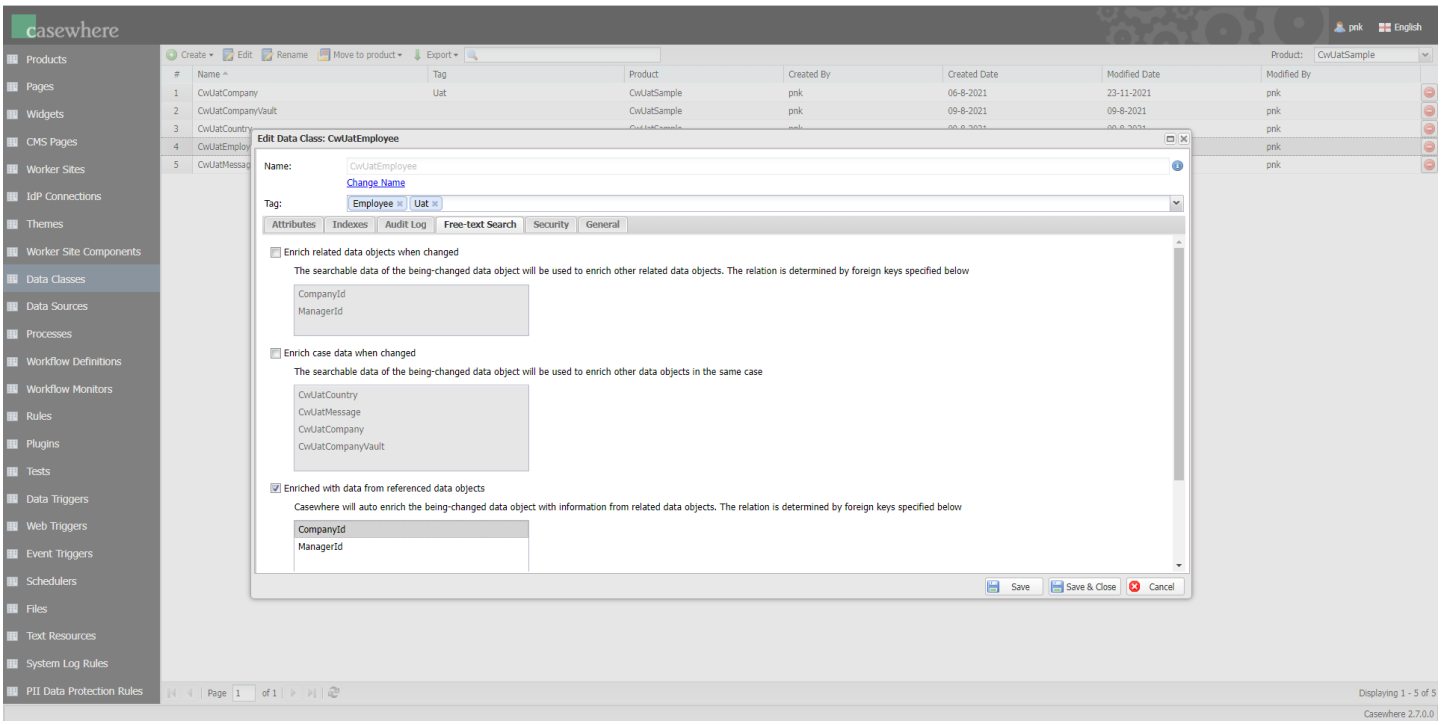
Configuration

By default, Casewhere auto enables free-text search for all string attributes in your data classes. However, you can disable free-text search for a specific field by setting `Searchable` to `False`.



Advanced settings

In the data class editor, navigate to **Free-text search** tab. Here you can find all advanced settings to customize your search strategy.



Data enriching

Data enriching is the process of extending your already existing data objects by providing supplementary searchable information. The additional data can be retrieved from related data objects.

Enrich related data objects when changed

The searchable data of the being-changed data object will be used to enrich other related data objects. The relations are determined by foreign keys.

Example: The relation between `Employee` and `Company` data objects is defined by the foreign key `Employee.CompanyId`. By enabling this setting on `Employee`, every time the data object is changed, the related `Company` will be enriched with the employee's information.

Enrich case data when changed

The searchable data of the being-changed data object will be used to enrich other data objects in the same case.

Example: By enabling this setting on `Employee`, every time the data object is changed, the data objects in the same case, e.g. `Contract`, `Email`, etc., will be updated.

Enriched with data from referenced data objects

The being-changed data object will be enriched with the information from related data objects. The relation is determined by foreign keys.

Example: The relation between `Employee` and `Company` data objects is defined by the foreign key `Employee.CompanyId`. By enabling this setting on `Employee`, every time the `Company` data object is changed, the `Employee` data object in context will be enriched with the company's information.

Tokenizer

> **Note:** Available since Casewhere 2.6.10

The built-in search engine in MongoDB performs text searches in a collection, it tokenizes the search string using whitespace and most punctuation as delimiters. You can search documents with individual words in the given text but not with partial words. To solve this problem, Casewhere implements the N-gram tokenizer to break a text down into words. Each word based on a

continuous sequence of characters of the specified length. It is useful for querying languages that don't use spaces or that have long compound words, like German.

To enable N-gram tokenizer:

- Go to **Tokenizer** section and check on **Enable n-gram tokenizer**.
- Set **Min-gram** and **Max-gram** value.

Example:

- Text = "Quick Fox"
- Min-gram = 1
- Max-gram = 2
- Output = ["Q", "Qu", "u", "ui", "i", "ic", "c", "ck", "k", "k ", " ", " F", "F", "Fo", "o", "ox", "x"]

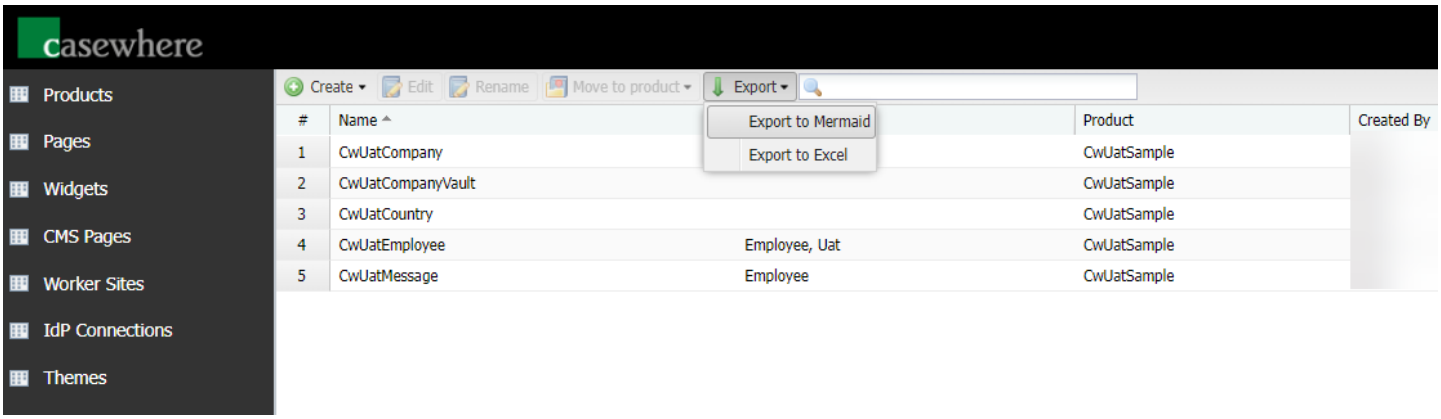
Data diagram

Mermaid is one of the most popular visualization languages that let you quickly create diagrams and visualization using text and code. Since version **2.6.14**, Casewhere enables you to export data classes to Mermaid diagrams.

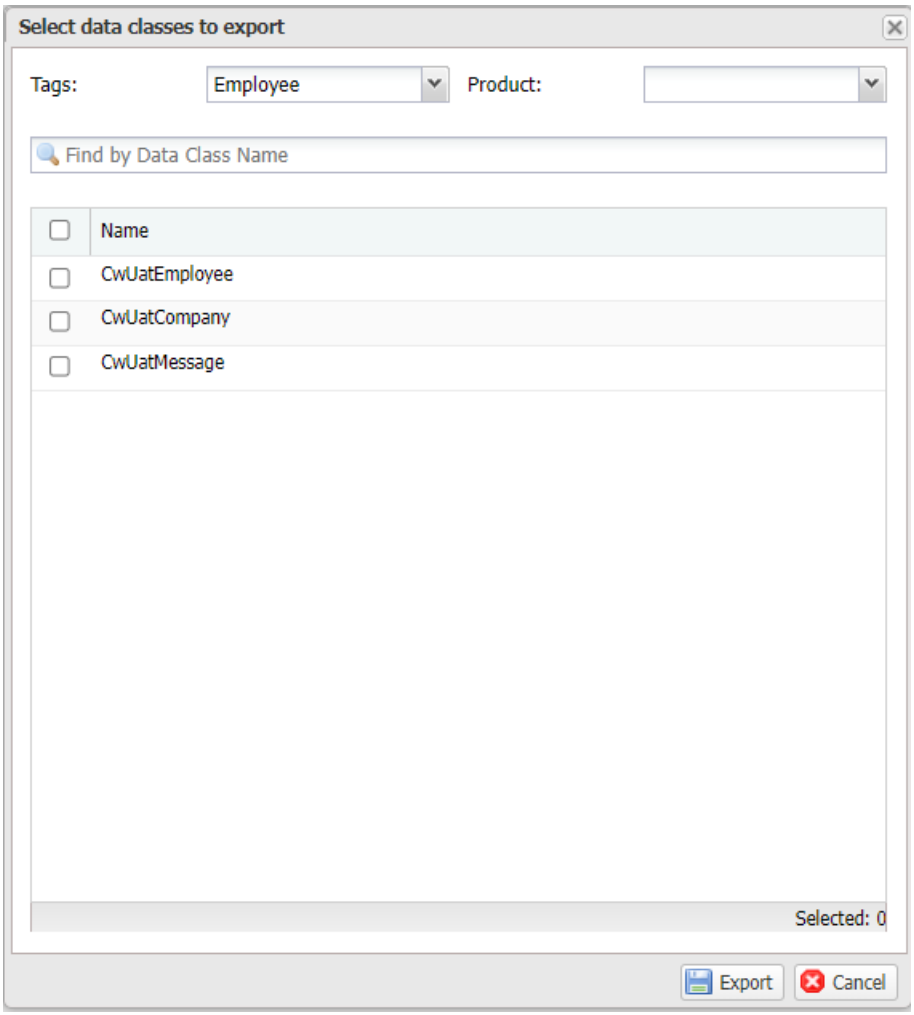
Generate diagram

In Casewhere Admin, navigate to **Data Classes**:

- Click **Export**
- Select **Export to Mermaid** from the drop-down list



- Select data classes you want include in the diagram. Here you can filter data classes by tags, products



- Click **Export** to generate a diagram file with the **.mmd** extension

```
erDiagram

CwUatMessage }o--|| CwUatEmployee : ""
CwUatCompany }o--o| CwUatEmployee : "CEO"
CwUatCompany }o--o{ CwUatEmployee : "Managers"
CwUatEmployee }o--o| CwUatCompany : "Work For"
CwUatEmployee }o--o| CwUatEmployee : "Manager"

CwUatMessage {
    ForeignKey EmployeeId
    String Content
    Enum Status
}
CwUatCompany {
    String Name
    String StockSymbol
    String StockMarketCap
    String Introduction
    String Address
    Boolean Active
    Array CountryIds
    Document Logo
    ForeignKey CeoId
    Array ManagerIds
    DateTime StartDate
    Double AuthorizedCapital
}
CwUatEmployee {
    String Name
    String Ssn
    Enum Gender
    String JobTitle
    Enum ShirtSize
    String Slogan
    String Email
    String PhoneNumber
    String Address
    ForeignKey CompanyId
    String Signature
    ForeignKey ManagerId
    Double Salary
    DateTime Dob
}
```

```
erDiagram

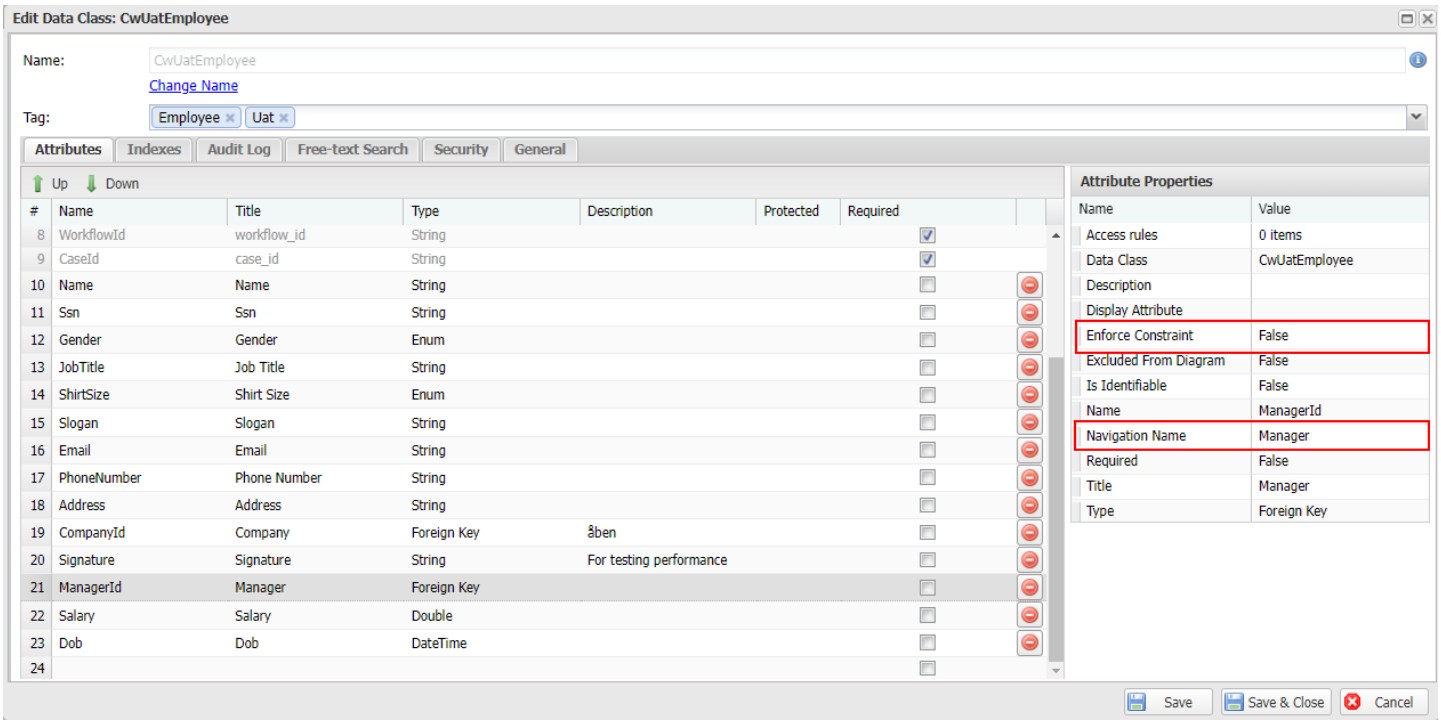
CwUatMessage }o--|| CwUatEmployee : ""
CwUatCompany }o--o| CwUatEmployee : "CEO"
CwUatCompany }o--o{ CwUatEmployee : "Managers"
CwUatEmployee }o--o| CwUatCompany : "Work For"
CwUatEmployee }o--o| CwUatEmployee : "Manager"

CwUatMessage {
    ForeignKey EmployeeId
    String Content
    Enum Status
}
CwUatCompany {
    String Name
    String StockSymbol
    String StockMarketCap
    String Introduction
    String Address
    Boolean Active
    Array CountryIds
    Document Logo
    ForeignKey CeoId
    Array ManagerIds
    DateTime StartDate
    Double AuthorizedCapital
}
CwUatEmployee {
    String Name
    String Ssn
    Enum Gender
    String JobTitle
    Enum ShirtSize
    String Slogan
    String Email
    String PhoneNumber
    String Address
    ForeignKey CompanyId
    String Signature
    ForeignKey ManagerId
    Double Salary
    DateTime Dob
}
```

Advanced settings

The name of the data class relationship is determined by the **Navigation Name** property of the **Foreign Key Data Attribute**.

It's not a good idea to display all data class attributes in the diagram. You can limit what to include or exclude in the diagram using the property **Excluded from Diagram**.


Edit Data Class: CwUatEmployee
Name:
CwUatEmployee
Change Name
Tag:
Employee
Uat
Attributes
Indexes
Audit Log
Free-text Search
Security
General
Up
Down
Name Title Type Description Protected Required
8 WorkflowId workflow_id String
9 CaseId case_id String
10 Name Name String
11 Ssn Ssn String
12 Gender Gender Enum
13 JobTitle Job Title String
14 ShirtSize Shirt Size Enum
15 Slogan Slogan String
16 Email Email String
17 PhoneNumber Phone Number String
18 Address Address String
19 CompanyId Company Foreign Key åben
20 Signature Signature String For testing performance
21 ManagerId Manager Foreign Key
22 Salary Salary Double
23 Dob Dob DateTime
24
Attribute Properties
Name Value
Access rules 0 items
Data Class CwUatEmployee
Description
Display Attribute
Enforce Constraint False
Excluded From Diagram False
Is Identifiable False
Name ManagerId
Navigation Name Manager
Required False
Title Manager
Type Foreign Key
Save
Save & Close
Cancel

Indexes

Introduction

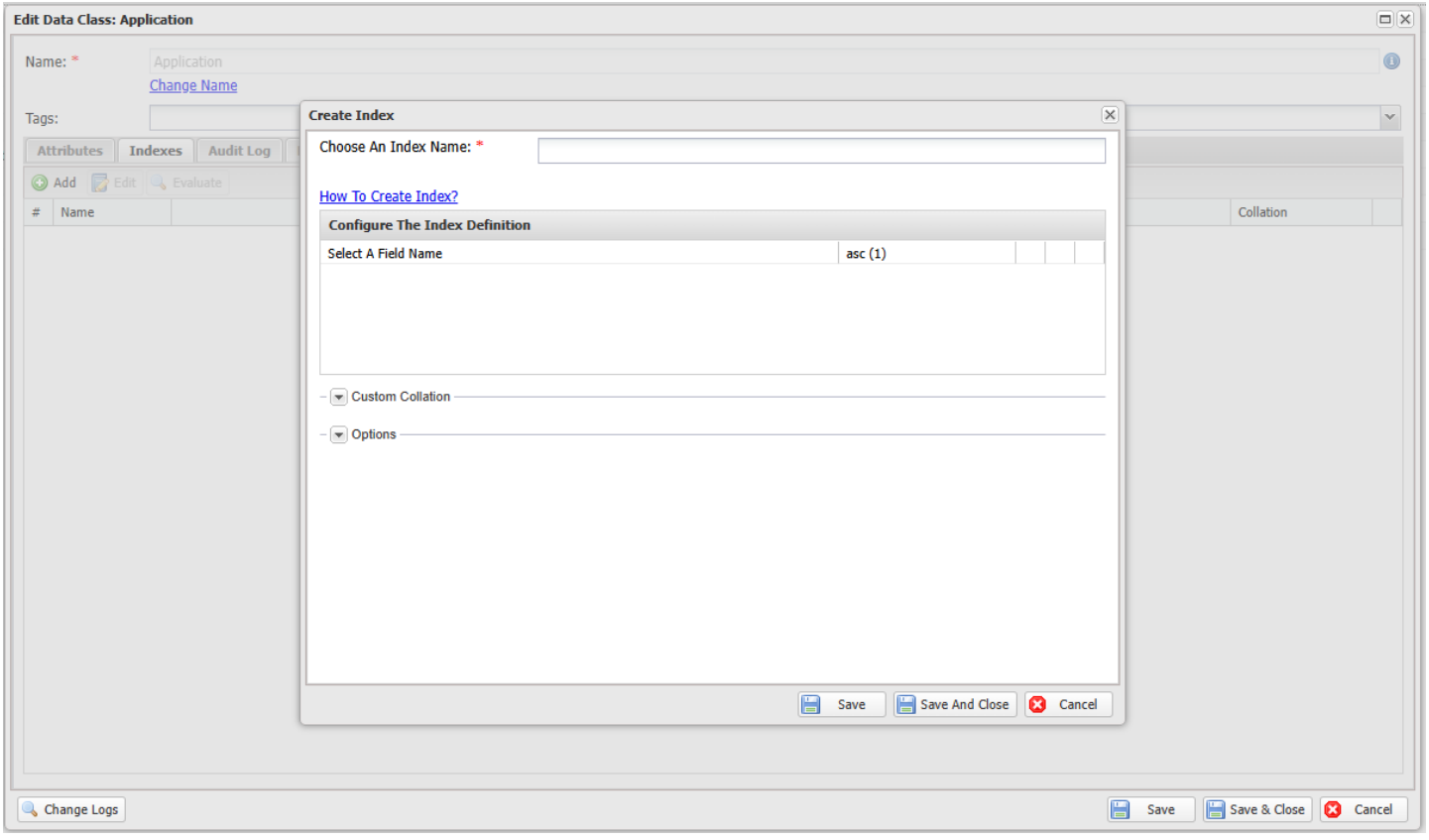
Casewhere employs MongoDB for its data storage. By default, MongoDB will go through all the documents to select the documents that match the query filter criteria, which is not ideal for dealing with large data collections. With proper indexes, MongoDB can narrow down the documents it must scan, thereby speeding up queries.

This guide provides a complete overview of setting up effective indexes in MongoDB for the Casewhere platform. You will learn:

- The importance of indexes in optimizing database queries.
- Step-by-step instructions for configuring and managing indexes in Casewhere.
- Best practices for maintaining indexes to ensure long-term efficiency.

Create indexes

In the data class editor, go to the **Indexes** tab to manage the indexes for the data class. Click **Add** to create a new index. The index editor appears as follows:



The following fields are needed to create an index:

- **Index Name**: Index names must be unique. See here for popular index name conventions.
- **Index Definition:** Specify the index fields and their sorting order. The order of these fields is crucial as it impacts how the query planner utilizes the index.
- **Custom Collation**:
    - If no custom collation is specified, the index will be created with the default collation. The default collation is configurable per Casewhere deployment.
    - You can create an index without specifying collation by setting **Locale** to 'None'. Non-collation indexes generally offer better performance, but they do not support case-insensitive text filtering, such as filtering for users with an address containing 'virum'.
        - Non-collation indexes are highly useful for aggregation queries on large data collections, particularly when

grouping and counting are required.

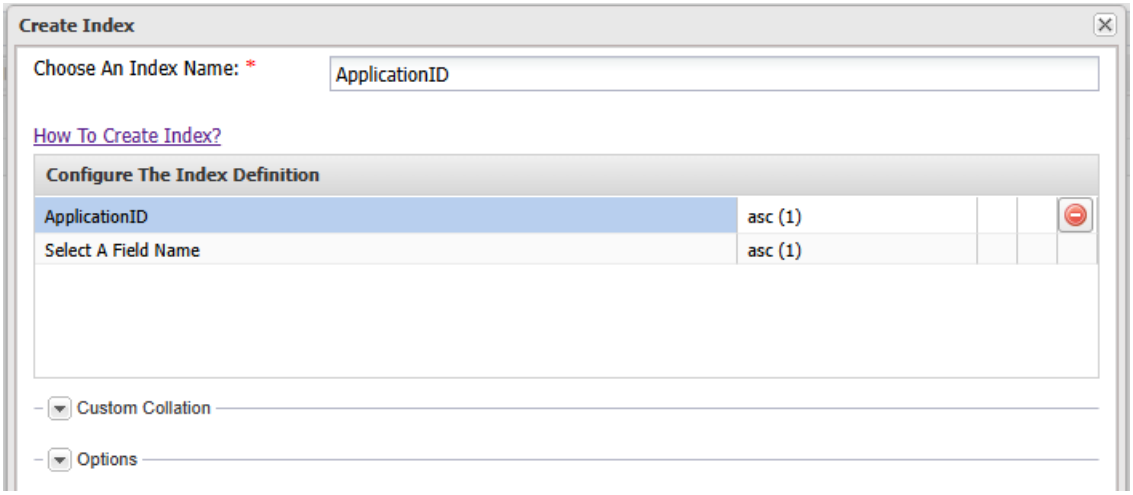
- **Options:**
  - **Build Index In The Background**: Building an index can be time-consuming, so Casewhere shouldn't wait for the process to complete. It's recommended to use the default setting here.
  - **Create Unique Index**: A unique index ensures that the indexed fields do not store duplicate values.
  - **Create TTL**: TTL indexes are special single-field indexes that MongoDB can use to automatically remove documents from a collection after a certain amount of time or at a specific clock time. Data expiration is useful for certain types of information like machine generated event data, logs, and session information that only need to persist in a database for a finite amount of time.

To demonstrate how MongoDB indexes work in Casewhere, we'll use the following data class **Application** as an example.

NAME	TYPE
ApplicationID	String
SubmittedDate	DateTime
Status	Enum
ApplicantId	ForeignKey
ApplicantCpr	String, Encrypted
IsDeleted	Casewhere built-in field to determine if an object is deleted or not
CaseState	Casewhere built-in field to indicate the data object's creation state

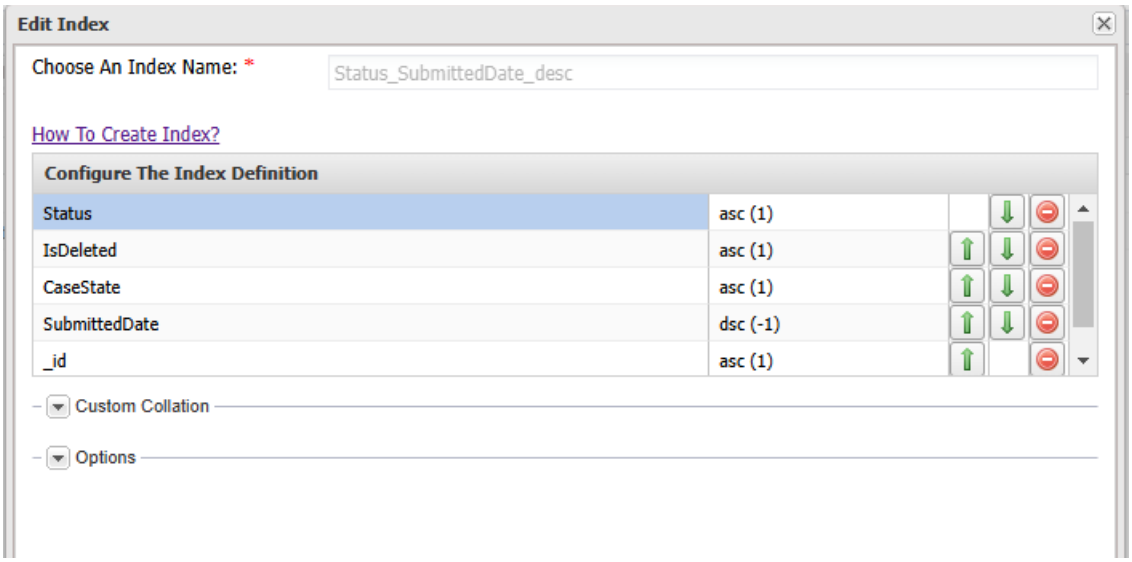
Example 1: Find applications by ApplicationID

`ApplicationID` is a high-cardinality field since each application has a unique value. Although Casewhere automatically appends conditions for `IsDeleted` and `CaseState` for all queries by default, there's no need to include these or any sorting criteria in the index, keeping it small and efficient.



Example 2: Filter by Status and sort by SubmittedDate in descending order

This is a common query used in most data widgets. Please note that Casewhere automatically appends conditions for `IsDeleted` and `CaseState` for all queries by default. You should stick to the ESR rule starting with high cardinality field first, in this case, the `Status` field.



Casewhere automatically appends the sorting condition `{_id: 1}` to ensure correct paging, so you should also include it in the index.

Example 3: Optimize aggregation queries with projection covered

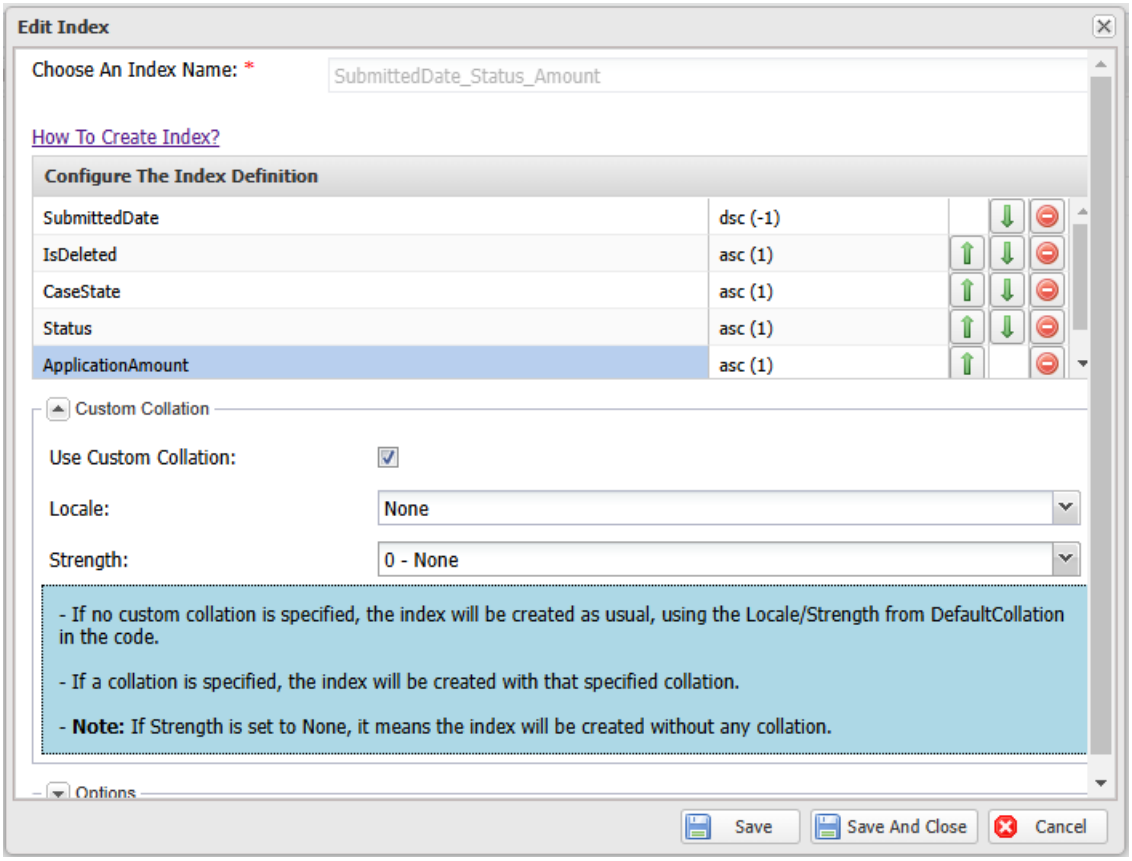
For scenarios where aggregation operations, such as summing, needed to be done against large data collections, consider optimizing performance by ensuring PROJECTION_COVERED. This means retrieving all necessary data directly from the indexes, without the need for additional fetching.

For example, you are using the following query to filter applications submitted within a specific period and calculating the total amount for each application status.

```
db.getCollection("CWApplications").aggregate(
[
    {
        $match: {
            IsDeleted: false,
            CaseState: 'Created',
            SubmittedDate: {
                $gt: ISODate("2022-01-01 00:00:00.000Z"),
                $lt: ISODate("2024-01-01 00:00:00.000Z")
            }
        }
    },
    {
        $group: {
            _id: "$Status",
            TotalAmount: {
                $sum: "$ApplicationAmount"
            }
        }
    }
])
```

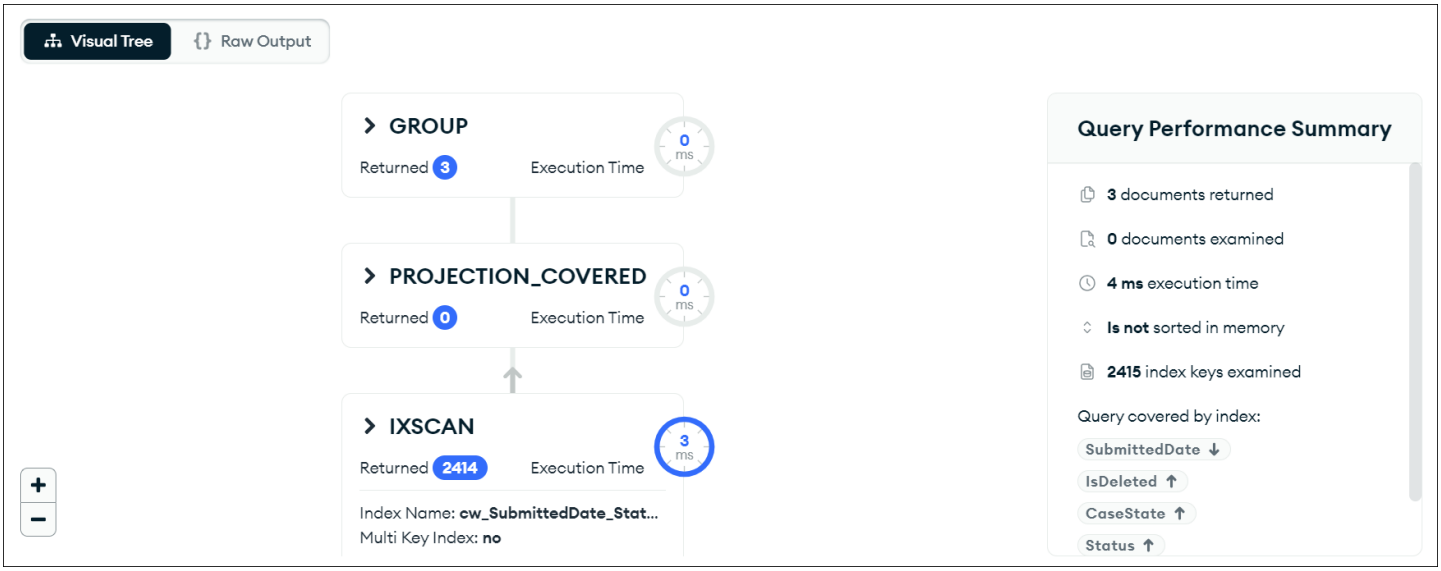
The query may run very slowly if the filter condition matches several million applications. To improve the performance, you can apply the following **simple-collation** index. This ensures that `Status` and `ApplicationAmount` are retrieved directly from the index, even when these fields are not part of the filter criteria.

```
{
    "SubmittedDate" : -1,
    "CaseState" : 1,
    "IsDeleted" : 1,
    "Status" : 1,
    "ApplicationAmount" : 1
}
```

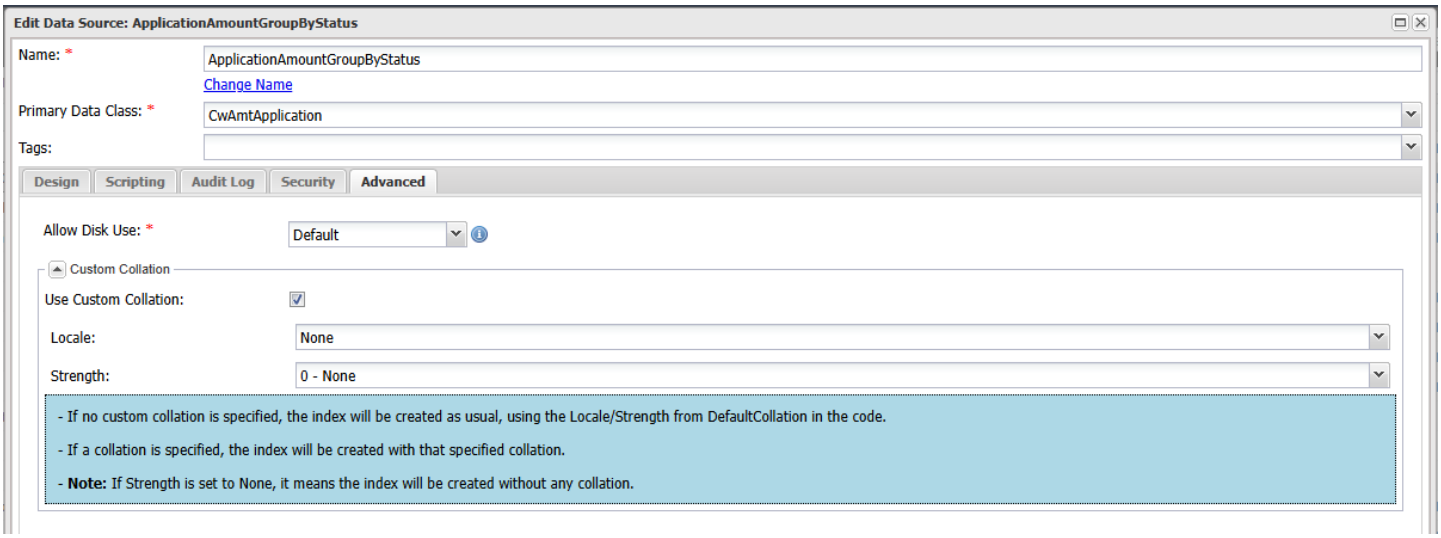


Since `IsDeleted` and `CaseState` are low-cardinality fields, placing them at the beginning of the index won't have a significant impact in this case. Additionally, having too many indexes prefixed with these fields may cause MongoDB to evaluate numerous index plans unnecessarily.

Execute the query and you can observe that PROJECTION_COVERED has been achieved. Please note that your query must be non-collation.



To enable simple-collation queries in Casewhere, ensure your custom data source is set to simple collation by configuring **Locale** to **None**.



Best practices

Naming conventions

While MongoDB doesn't enforce a specific naming convention for index names, it's a good practice to adopt a clear and consistent format to improve readability and maintainability. Here are two common conventions:

1. General: Name the index based on the fields it includes, separated by underscores. When creating compound indexes, you could specify the order (asc or desc) in the name.
    - Example: `general_Status_SubmittedDate_desc`. The index name should begin with the prefix `general`, and built-in fields such as `CaseState` and `IsDeleted` should be excluded.

2. Specific: In some cases, it's useful to include the index's purpose if it's for a specific query.
    - Example: `specific_report_total_amount` for an index created specifically to calculate total amount for reports. The index should be prefixed with `specific`.

The ESR rule

An index that references multiple fields is a compound index. Compound indexes can dramatically improve query response times. In most cases, applying the ESR (Equality, Sort, Range) rule to arrange the index keys helps to create a more efficient compound index. This page introduces the ESR Rule. Here are the key principles:

1. **Equality**: Prioritize equality conditions in your query, as MongoDB is most efficient when filtering documents based on equality. When designing your indexes, place fields with equality conditions at the beginning of the index, and ensure that high-cardinality fields come first.
2. **Sort**: Following the fields used for equality, include fields that are used in sort operations. This arrangement allows MongoDB to use the index both for filtering and for sorting the results, which is more efficient than sorting the data after filtering.
3. **Range**: Lastly, include fields with range queries (e.g., greater than, less than operations). Range queries are less selective compared to equality filters, so they are placed last in the index.

Example: Assuming you have the following query:

```
var filter = FilterBuilder.Create()
    .Gt("ApplicationAmount", 1000) // Range query
    .Eq("Status", "Submitted")
    .Eq("CaseState", "Created").Build();
var query = DataObjectApiQuery.For("Application")
    .FilterBy(filter)
    .OrderBy("SubmittedDate", true);
var result = ctx.Use<IDataApi>().Search(query);
```

Your compound index in this case should be ordered as below:

1. Status - *Equality* - High-cardiniality field comes first
2. IsDeleted - *Equality*
3. CaseState - *Equality*
4. SubmittedDate - *Sort*
5. _id - *Sort* - Casewhere always appends _id when sorting to ensure the order is consistent.
6. ApplicationAmount - *Range*

Query planner: Minimize the number of candidate plans

To evaluate the efficiency of query plans, the query planner runs all candidate plans during a trial period. In general, the winning plan is the query plan that produces the most results during the trial period while performing the least amount of work. If a query has too many candidate plans, MongoDB may take considerable time and resources to determine the optimal plan, which can lead to unpredictable slow response times.

For instance, most Casewhere data queries include the following filter criteria:

```
{
    IsDeleted: false,
    CaseState: 'Created'
}
```

If there are many indexes starting with these two fields, MongoDB must evaluate each one when updating the query plan, which can lead to significantly slower response times. To prevent this, it's important to avoid creating indexes that begin with `IsDeleted` and `CaseState`.

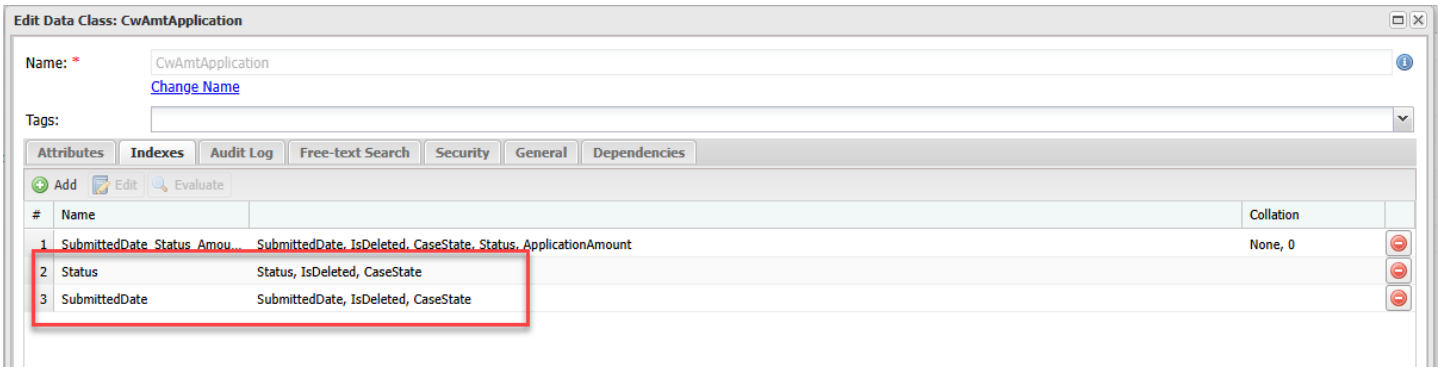
Working with `$or`

When evaluating the clauses in the `$or` expression, MongoDB either performs a collection scan or, if all the clauses are supported by indexes, MongoDB performs index scans.

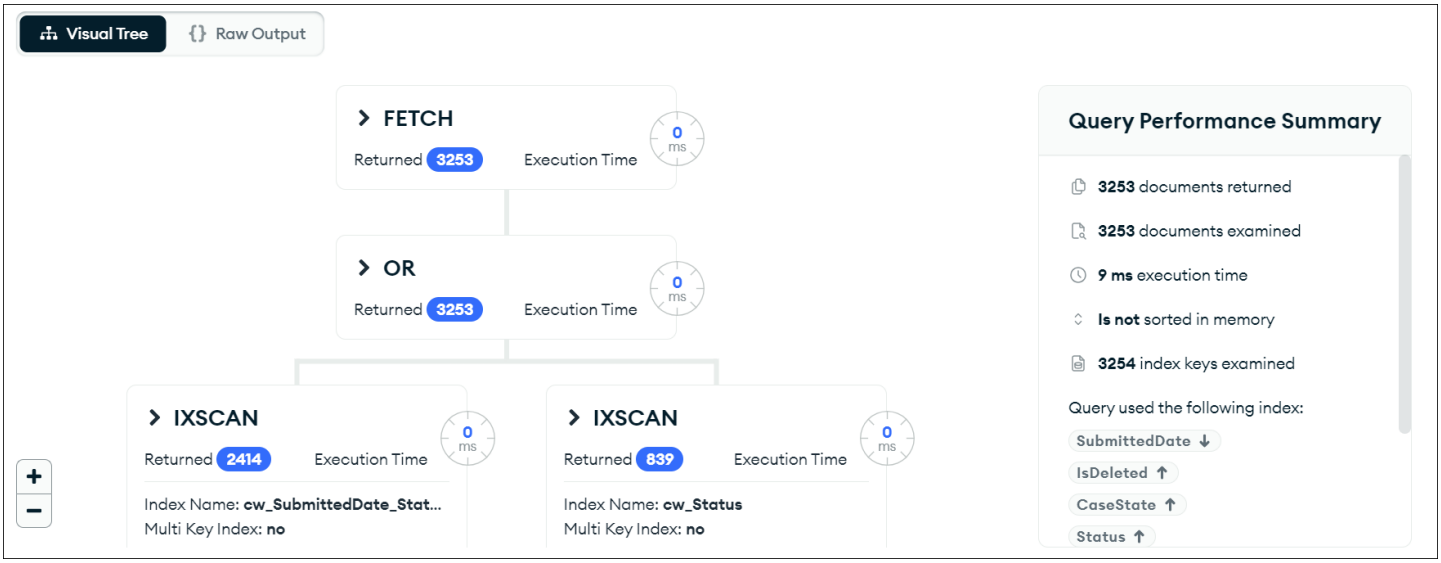
```
[
  {
    $match: {
      IsDeleted: false,
      CaseState: "Created",
      $or: [
        {
          SubmittedDate: {
            $gt: ISODate("2020-01-01 00:00:00.000Z"),
            $lt: ISODate("2024-01-01 00:00:00.000Z")
          }
        },
        {
          Status: 'Draft'
        }
      ]

    }
  }
]
```

To support this query, rather than a compound index, you would create one index on `Status` and another index on `SubmittedDate`:



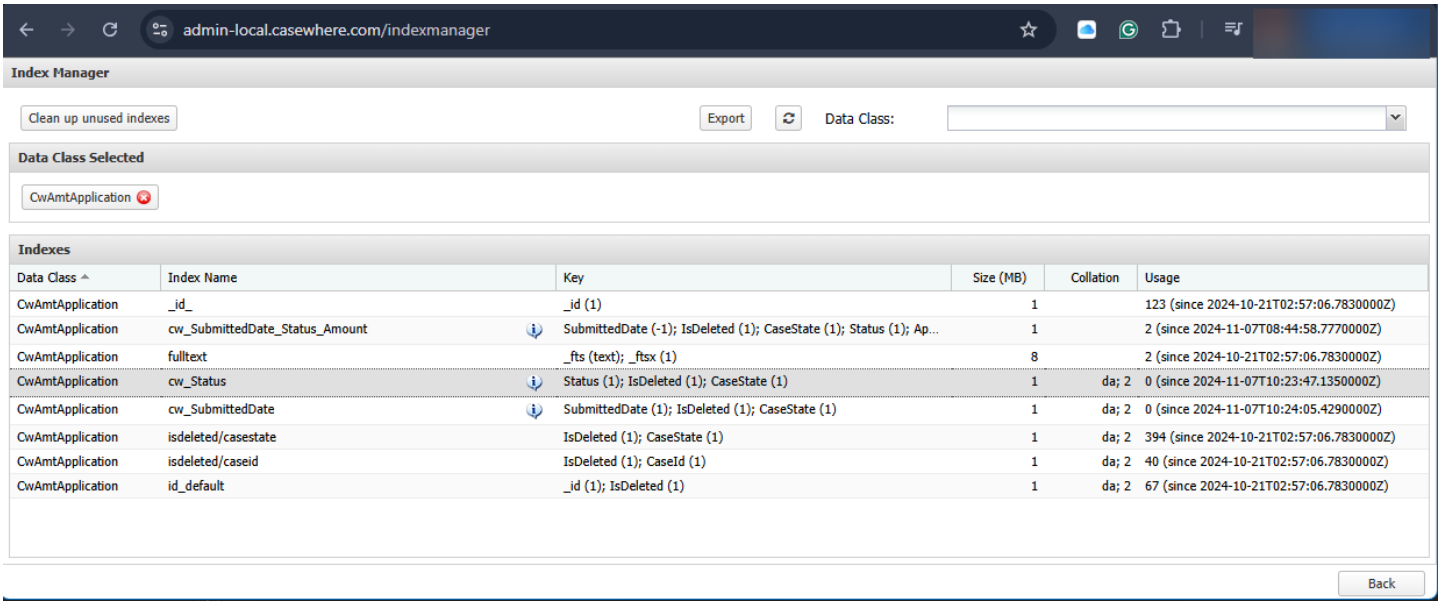
The query plan will look as follows:



Remove duplicated indexes

Building and maintaining indexes are costly, so unused indexes should be removed to optimize performance. The index (A: 1, B: 1, C: 1) is redundant and can be omitted because it is fully encompassed by (A: 1, B: 1, C: 1, D: 1). However, it won't be completely covered by (A: 1, B: -1, C: 1, D: 1) due to the opposite sorting direction of B.

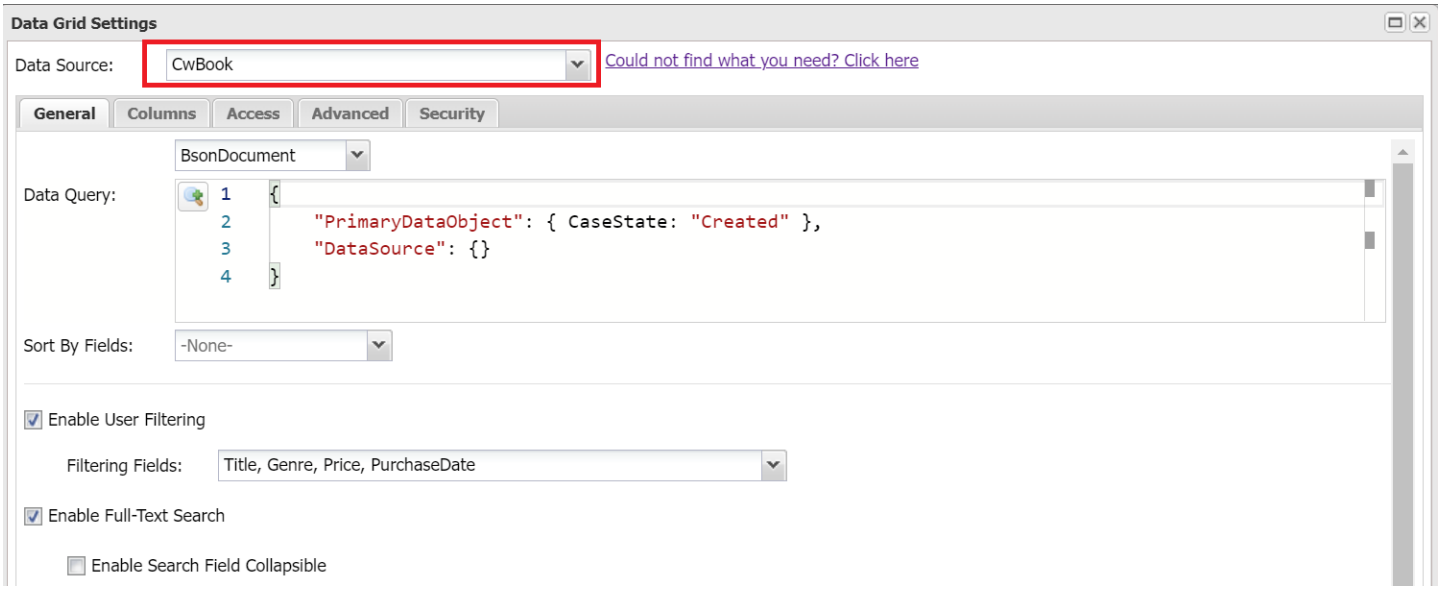
You can use the Index Manager to monitor index usage.



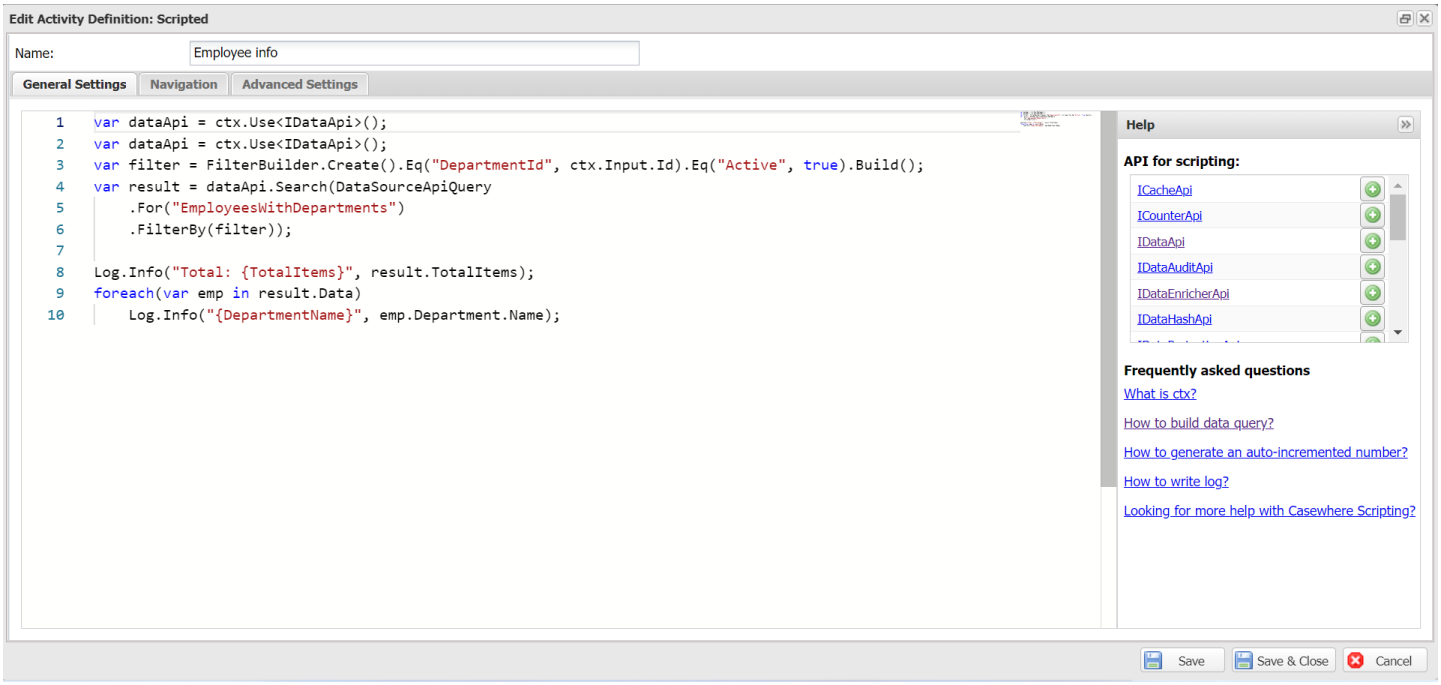
Data source overview

In Casewhere, applications and components fetch data from data sources instead of opening connections directly to the database. This is to hide the complexity of working with the database while maximizing reusability. The data consumers don't need to worry about where data is loaded or computed. They only need to know what the data source offers - the output data structure. A data source can serve different purposes. You can access data sources from widgets, workflows, or even from worker sites.

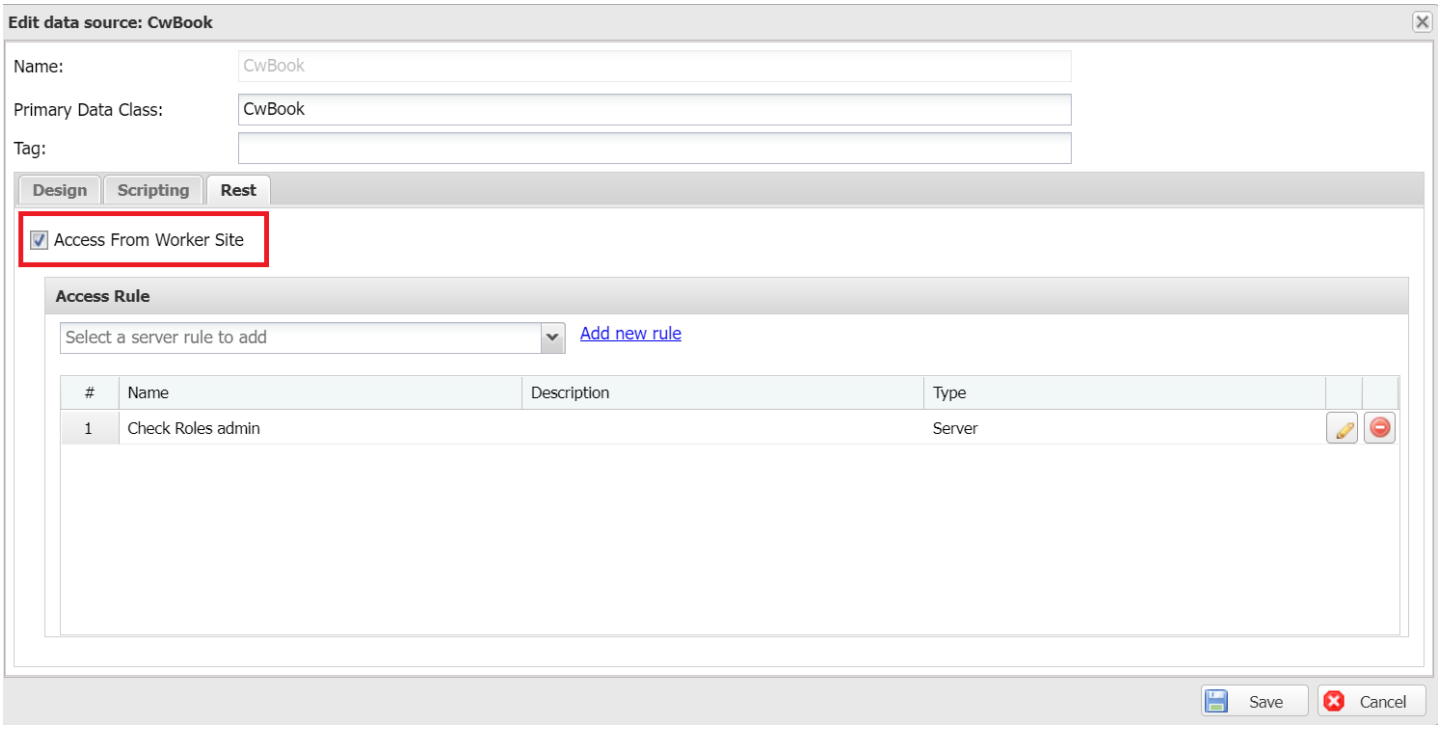
Data sources are used in Casewhere widgets:



or in Casewhere workflows:

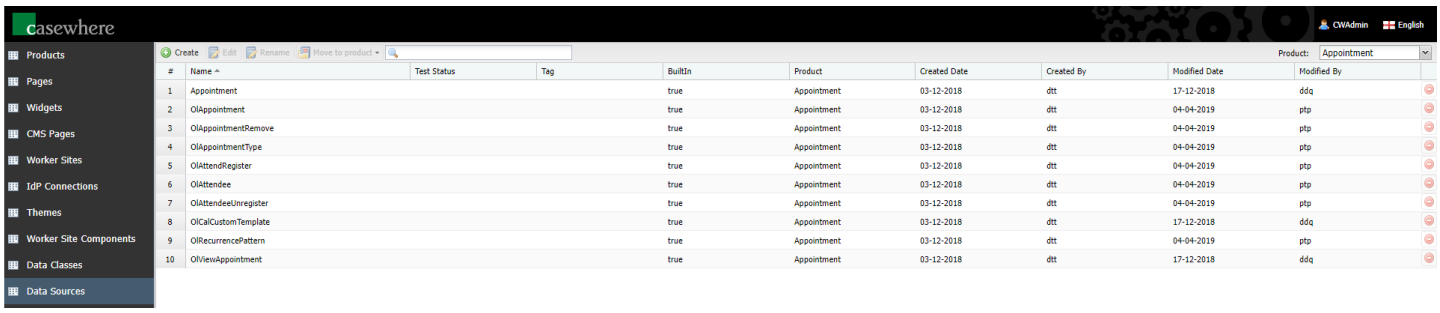


It's also possible to access data sources from worker site components:


Edit data source: CwBook
Name:
CwBook
Primary Data Class:
CwBook
Tag:
Design
Scripting
Rest
Access From Worker Site
Access Rule
Select a server rule to add
Add new rule
Name Description Type
1 Check Roles admin Server
Save
Cancel

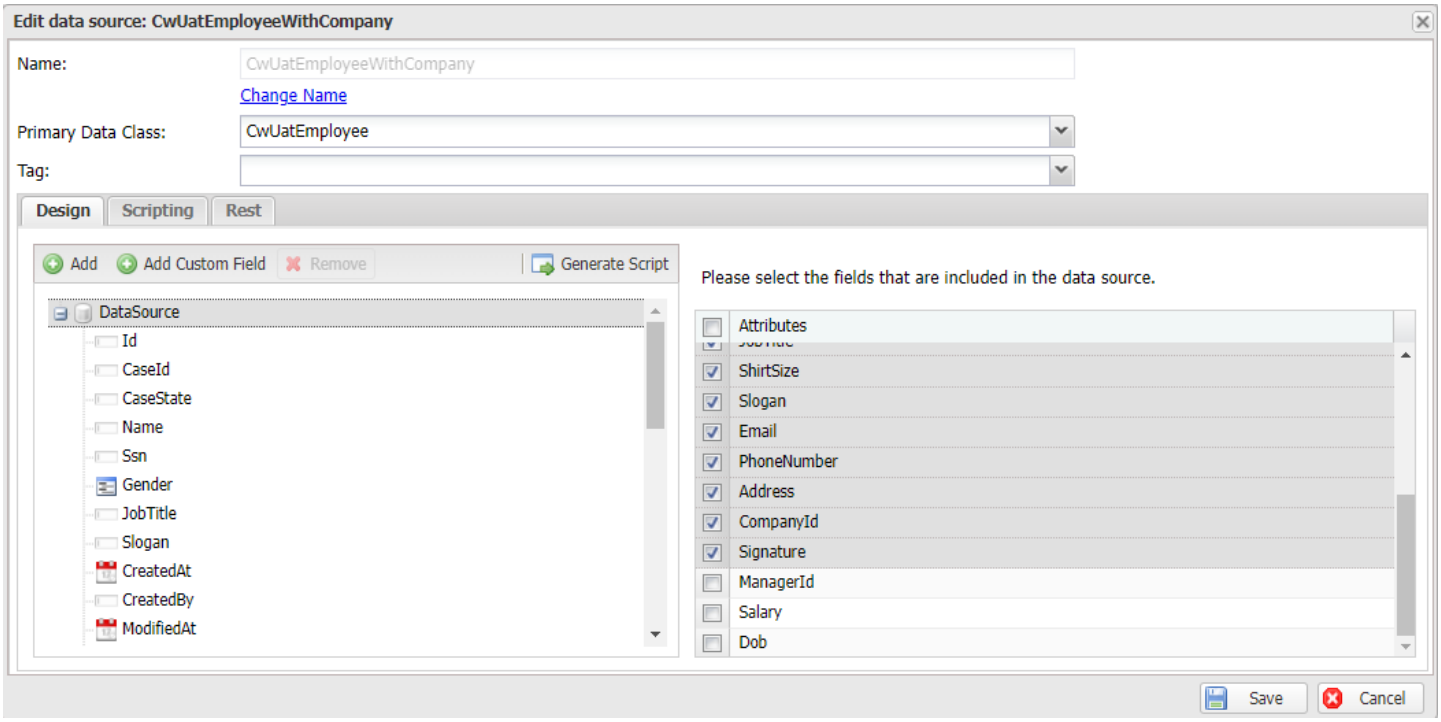
Data source management

In Casewhere Admin, navigate to **Data Sources** to create, edit and manage data sources like any product resource.



Data source editor

Casewhere provides an editor to design and configure data sources.



General information

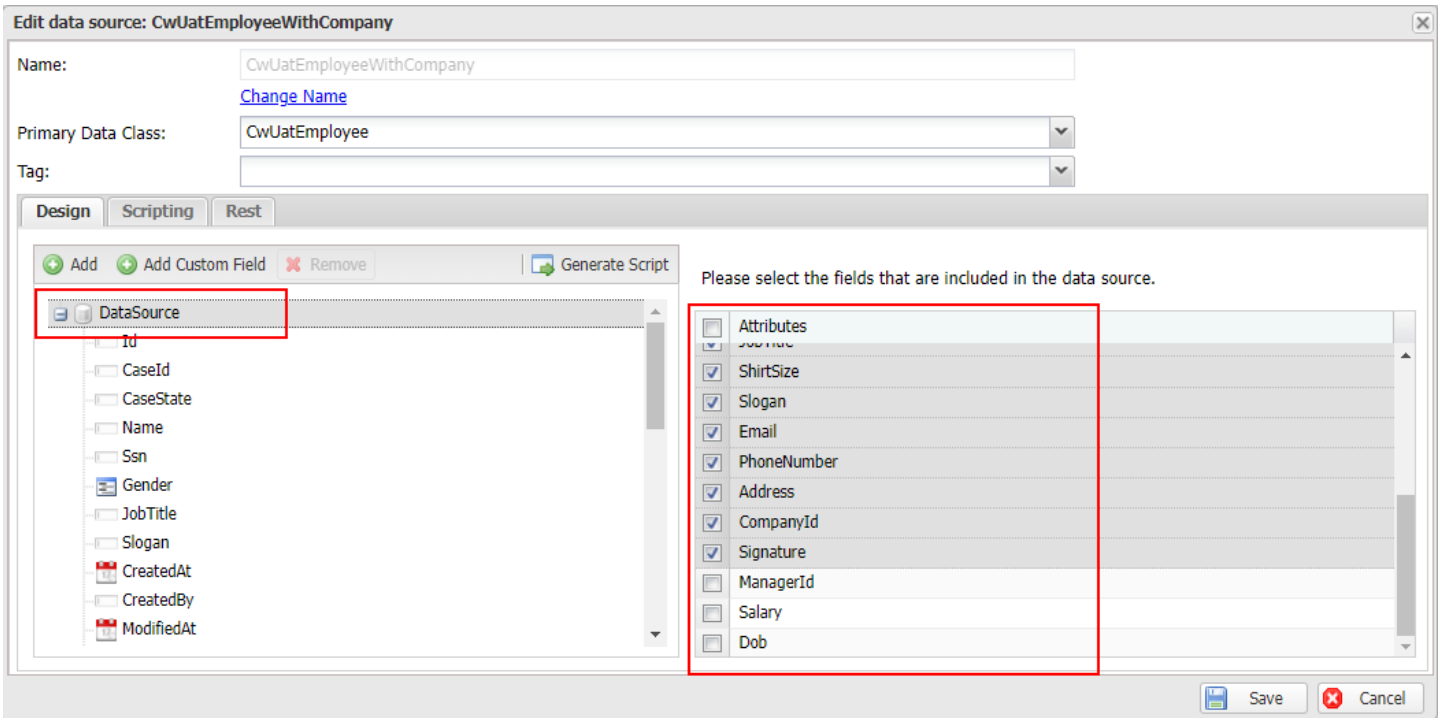
Name: Data source name must be unique

Primary Data Class: You must select a primary data class for the data source. From here, you can link to different related data classes to build a rich data source

Tag: Tags are used to facilitate searching, resource management

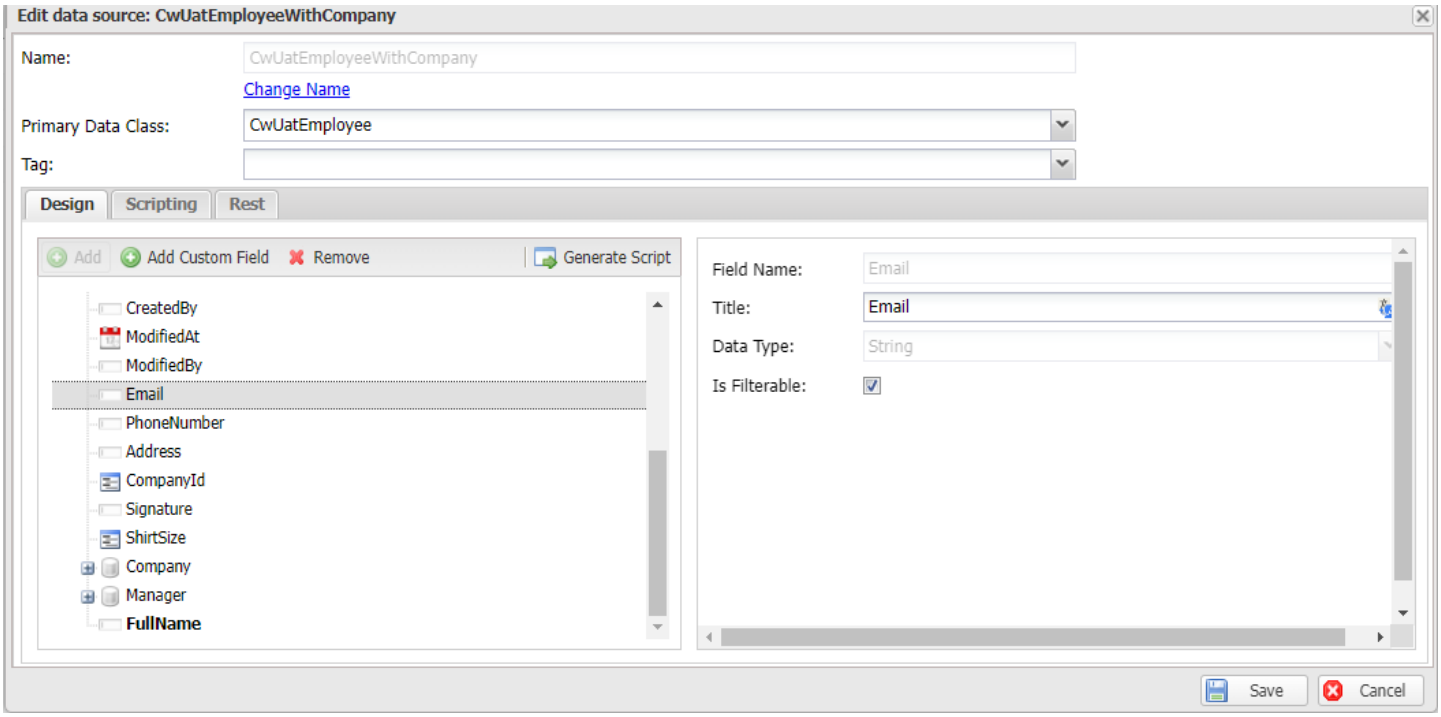
Design data source

Click on the data node to select what to include in the data source output. Note that a data source can have multiple data nodes; each represents a data class. The root node represents the primary data class.



Click on the field node to edit. You can easily see that the field icon reflects the field data type.

- **FieldName**: The name of the data class attribute. You can only edit the field name for custom fields
- **Title**: The title of the field shown on UI. Casewhere accepts a text resource key here
- **Data Type**: The data attribute type. You can only edit for custom fields
- **Is Filterable:** Determine if the data consumers can filter by this field

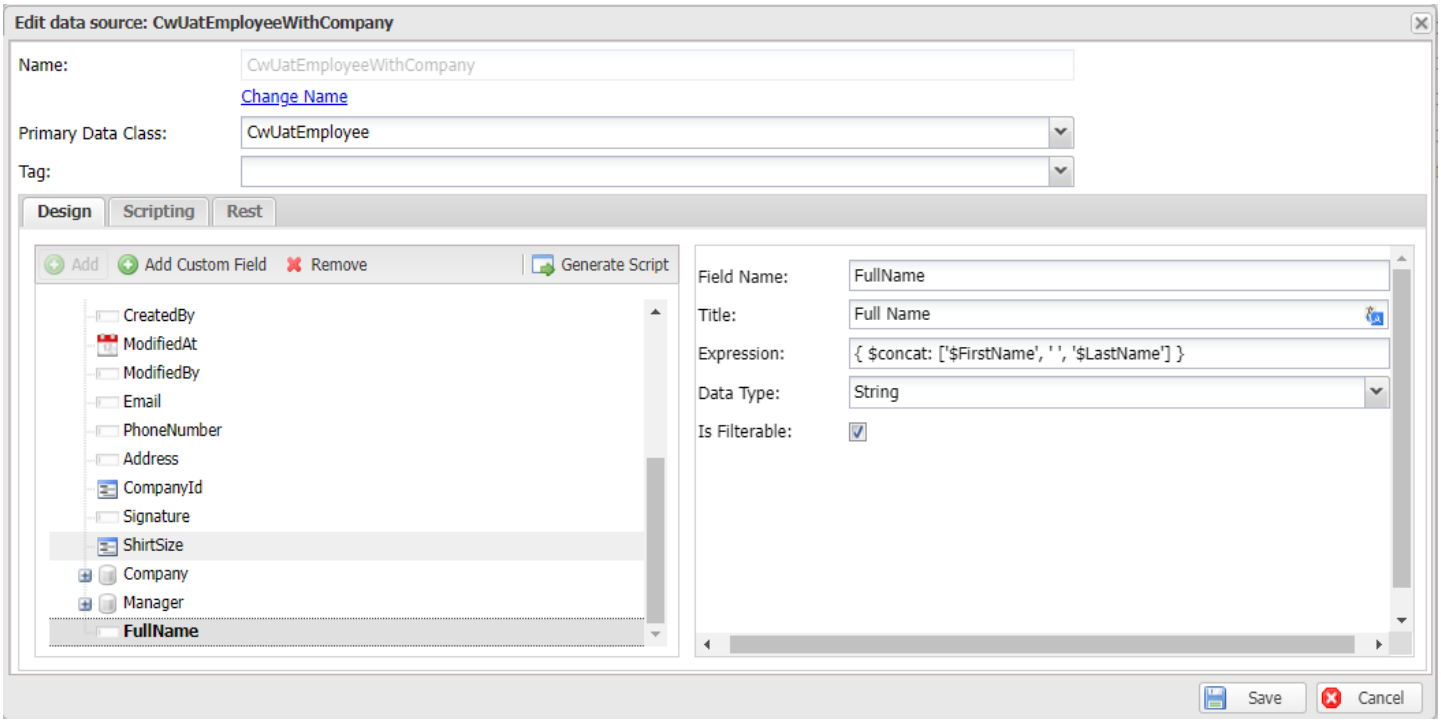


You can enrich the data source by joining a data class related to the primary. To do that, click the **Add** button.

- **Related Data Class**: Select the data class to join. As you can easily see, the relationships are determined by foreign keys.
- **Alias**: Name of the field holding the information of the joined data class. The default name is the data class name.



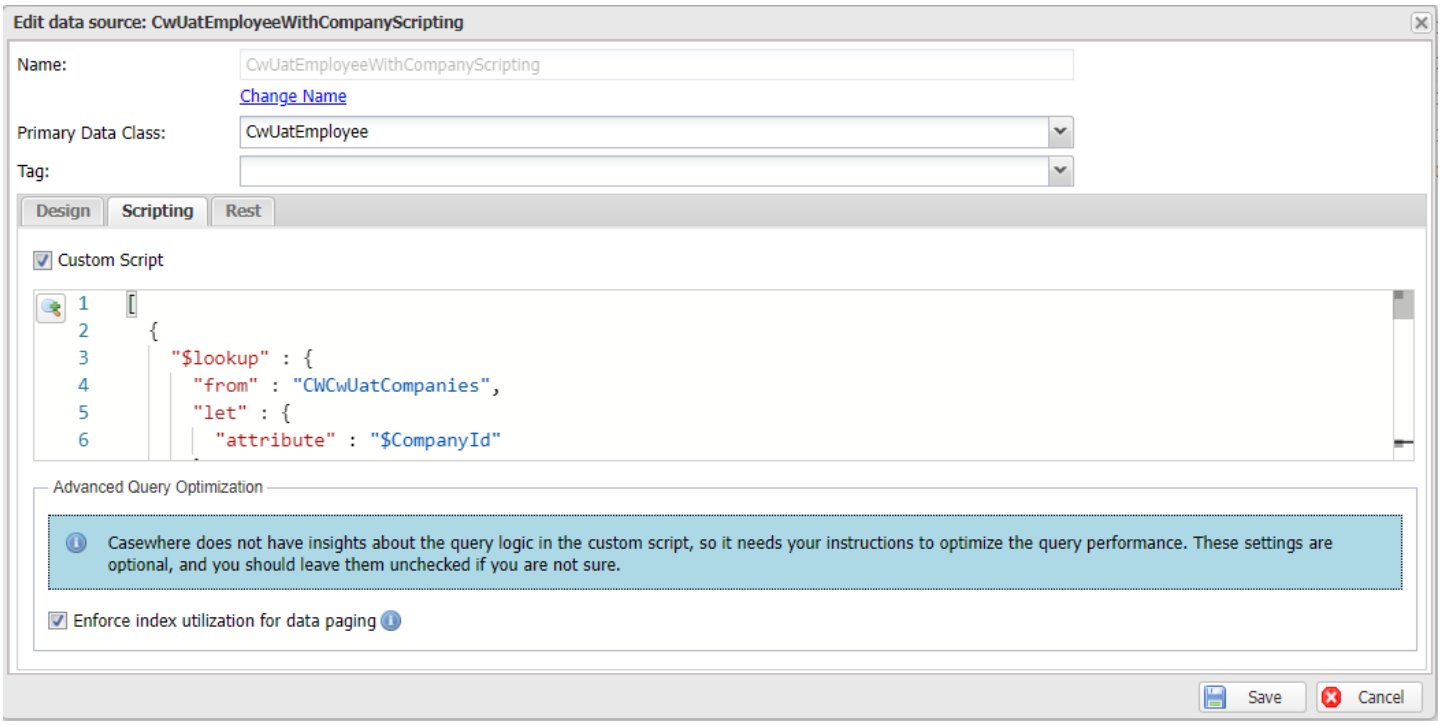
You can add a custom field by clicking on **Add Custom Field** button, and define the field logic by providing a MongoDB expression. The below example shows how to add a custom field for `FullName` computed from the two base fields `FirstName` and `LastName`.



Advanced settings

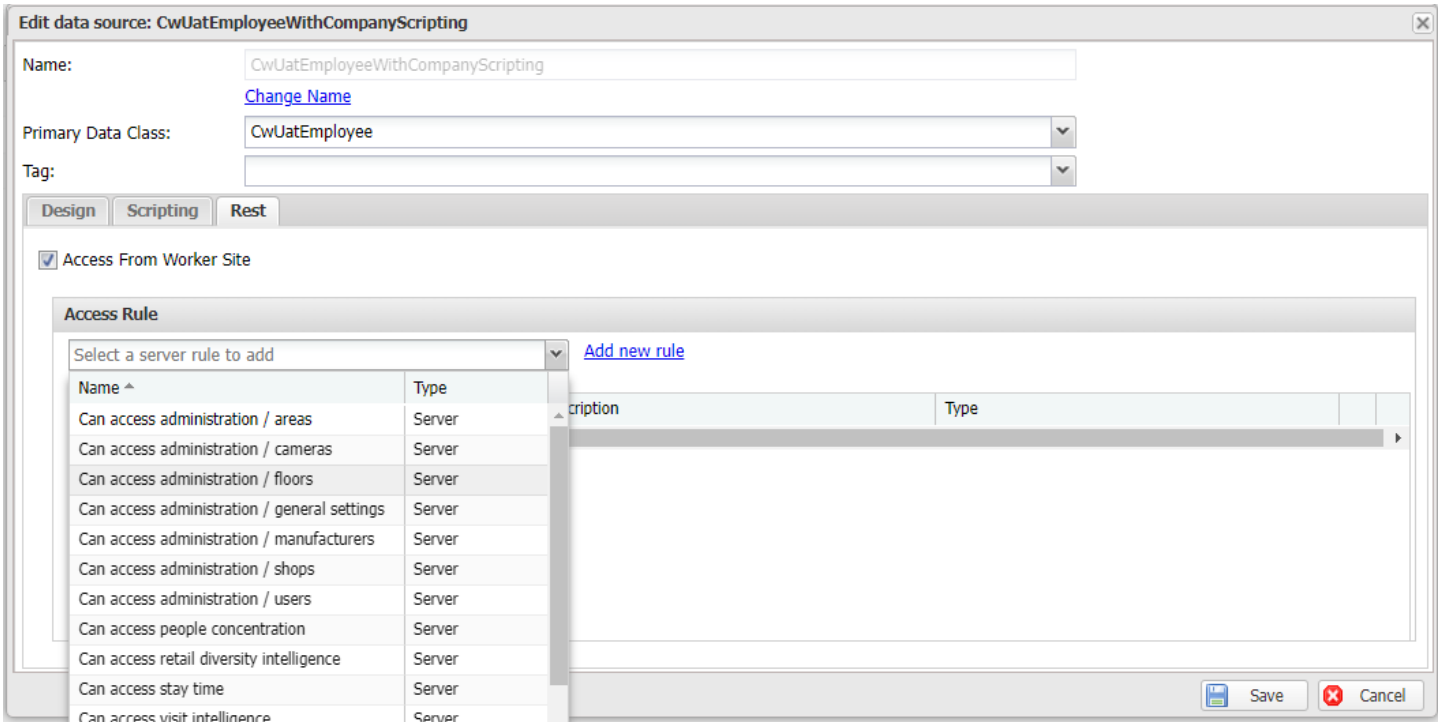
Scripting

Scripting is useful when you need to calculate aggregate values for the data in a collection, e.g., monthly income, sales per employee, etc. To start building a data source using scripting, navigate to the **Scripting** tab, enable **Custom scripting**. After that, define the aggregation pipeline using MongoDB Aggregation Framework. Click here to learn more about scripting data source.



REST

It's possible for the frontend components in worker sites to query the data source directly. For that, navigate to the **Rest** tab and enable **Access From Worker Site**. You can protect the data source by specifying the access rules. Click here to learn how to use restful data sources in Casewhere.



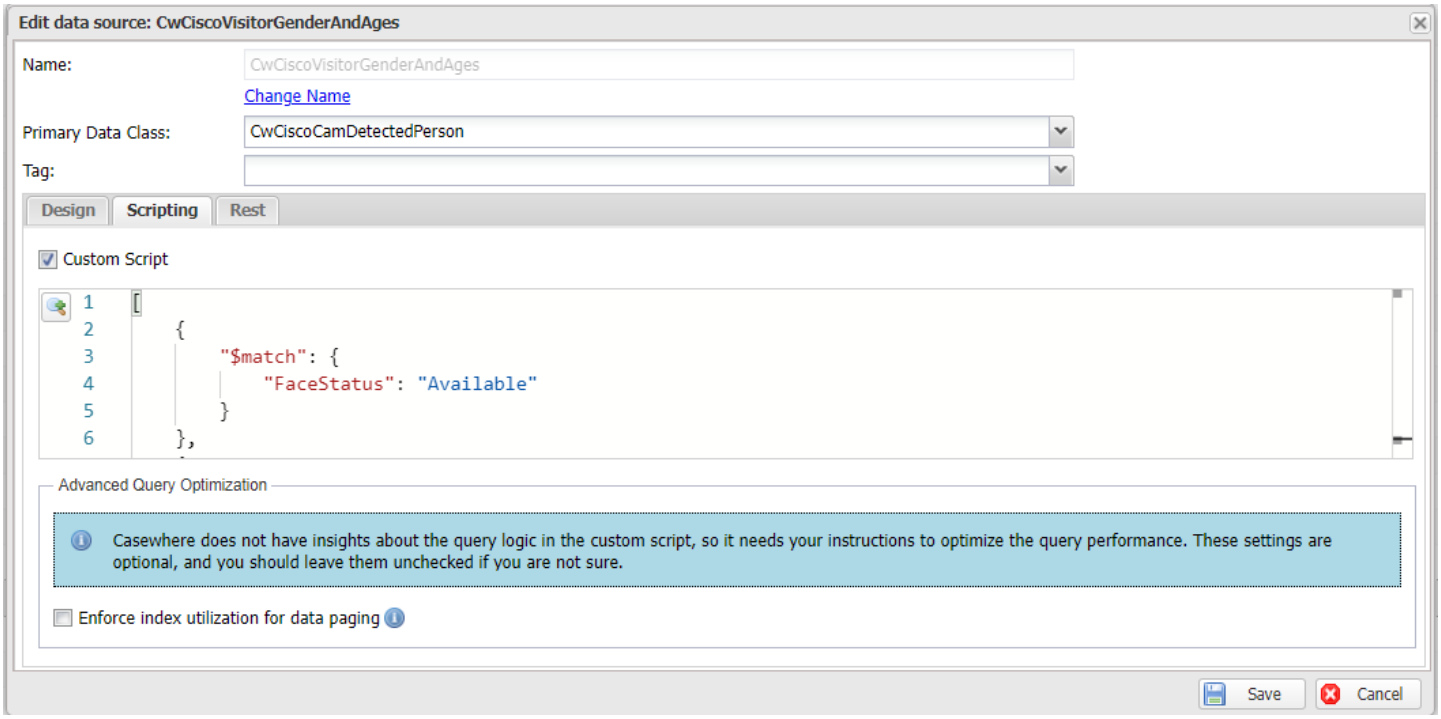
Scripting data source

It's easy to design data sources with the current GUI editor. However, for complex reporting scenarios where you need to calculate aggregate values, scripting is a better choice as it gives you all the flexibilities of the MongoDB Aggregation Framework.

Configure scripting data source

Enable scripting

Navigate to the **Scripting** tab and enable **Custom Script**. After that, you can start to compose the script for your data source.



Below are the typical aggregation operations used in most data sources:

- `$match`
- `$sort`
- `$group`
- `$project`

You can learn more at Practical MongoDB Aggregations Book.

Data source parameters

The inline variable called @params provides dynamic and flexible customization for a custom data source. It allows users to define input parameters that adapt a pipeline dynamically based on user-provided input. Overall enhances the reusability and responsiveness of data queries.

Key Concepts

- @params variable:
  - A special variable that holds user-defined parameters.
- Input flexibility:
  - Parameters can be passed to the data source from two distinct points:
    - Widget data query
    - SetParameters method (from scripted IDataApi)

A typical pipeline might look like the following when parameters are used:

```
[
    ...
    "BalanceAsOfToDate": {
        "$sum": {
            "$cond": {
                "if": {
                    "$and": [
                        { "$gte": @params.startDate },
                        { "$lte": @params.endDate }
                    ]
                },
                "then": {
                    "$cond": {
                        "if": { "$eq": [ "$TypeOfPosting", "Credit" ] },
                        "then": { "$multiply": [ -1, "$Amount" ] },
                        "else": "$Amount"
                    }
                },
                "else": 0
            }
        }
    }
    ...
]
```

This snippet demonstrates:

- The use of @params inside a data source pipeline.
- It can use anywhere in the pipeline, a nested conditional flow like an example, not just in the $match stage.

Parameters injection points

- Widget data query
    - User-defined parameters can be initialized in the widget using the dataSourceParameters variable. Below is an example implementation:

```
DateTime startDate = rangeDateFilter["StartDate"];
DateTime endDate = rangeDateFilter["EndDate"];

var startDateBson = new BsonArray();
startDateBson.Add("$PaymentDate");
startDateBson.Add(BsonValue.Create(startDate));

var endDateBson = new BsonArray();
endDateBson.Add("$PaymentDate");
endDateBson.Add(BsonValue.Create(endDate));

dataSourceParameters["startDate"] = startDateBson;
dataSourceParameters["endDate"] = endDateBson;
```

- SetParameters method
    - Input can also be provided using the SetParameters method from various query types, such as DataEnumerationQuery or DataObjectApiQuery. Below is an example implementation:

```
var startDateParameter = new BsonArray();
startDateParameter.Add("$PaymentDate");
startDateParameter.Add(BsonValue.Create(DateTime.MinValue));

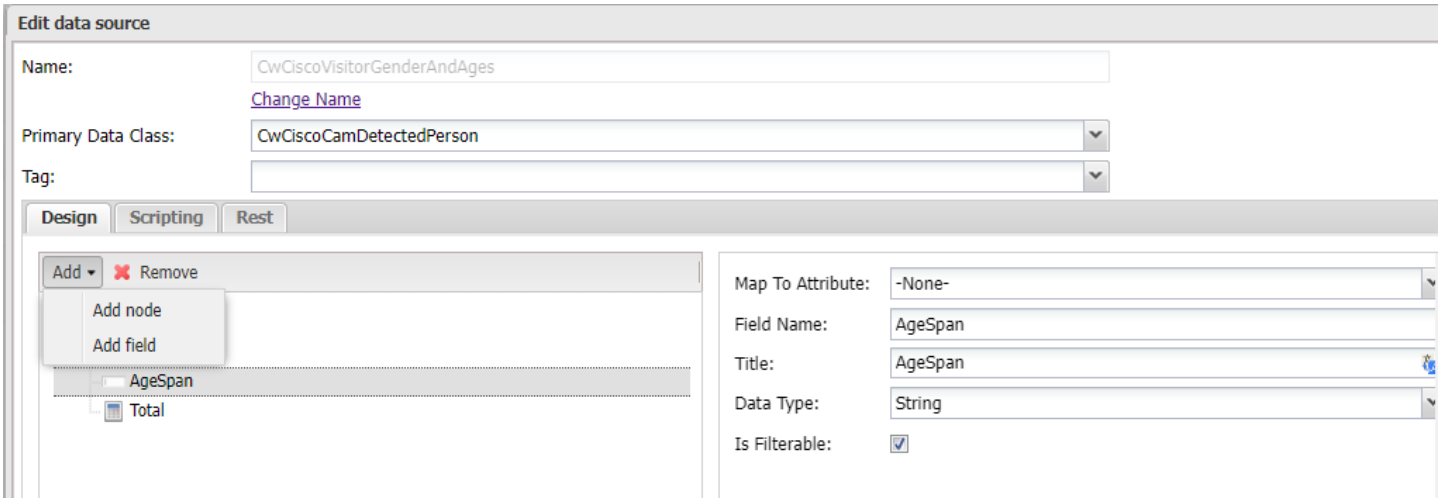
var endDateParameter = new BsonArray();
endDateParameter.Add("$PaymentDate");
endDateParameter.Add(BsonValue.Create(lastDayOfPreviousMonth));

var parameters = new Dictionary<string, object>
{
  { "startDate", startDateParameter },
  { "endDate", endDateParameter }
};

var query = DataSourceEnumerationQuery
  .For("CwAcParkingFeesWithCurrentBalancePerToDateForASpecificPeriodv2")
  .FilterBy(filter)
  .SetParameters(parameters);
```

Define the output structure

Casewhere cannot know the output structure for scripting data sources, so it needs you to specify the information. In the tab **Design**, select **Add**, and map the fields and nodes to the projection fields in your data source script.

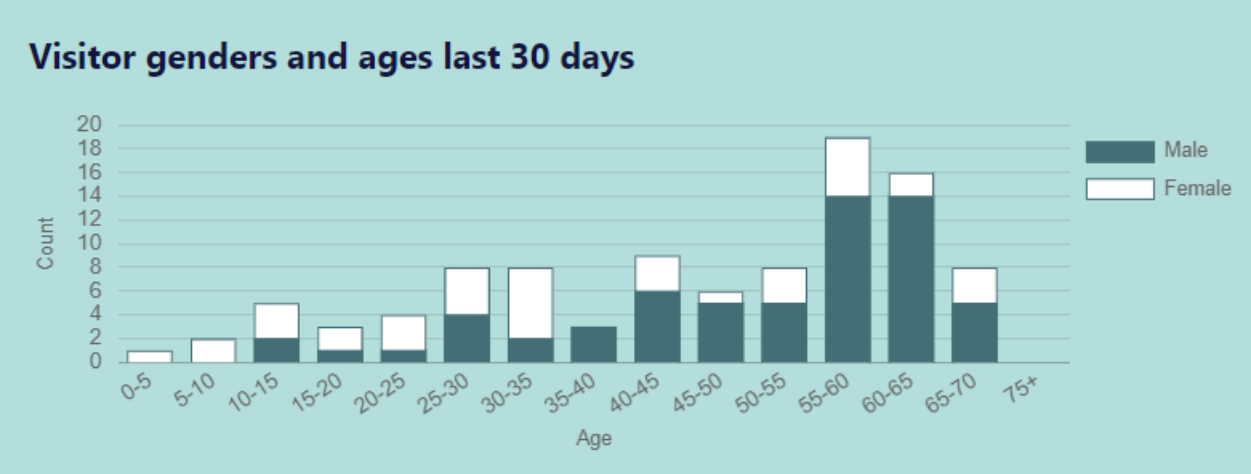


Hint: You should add the fields defined in the projection stage of your data source

```
[
    ...
    {
        "$project": {
            "_id": 0,
            "Gender": "$_id.Gender",
            "AgeSpan": "$_id.AgeSpan",
            "Total": "$Total"
        }
    }
    ...
]
```

A simple example

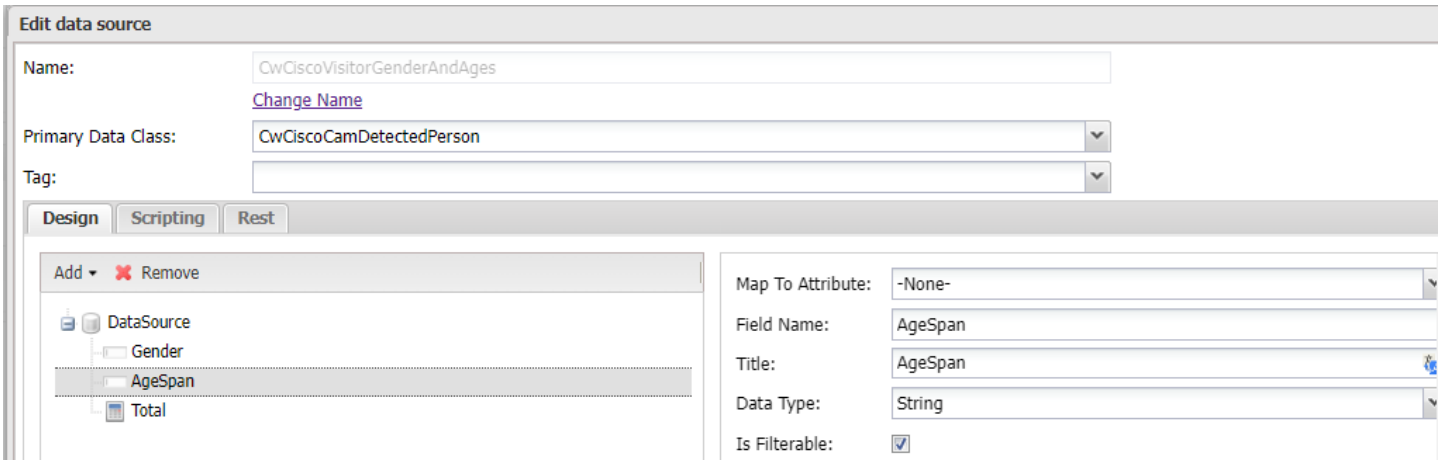
Imagine you're building an intelligent-shopping application that every time a person appears in front of the camera, the application will register a record `DetectedPerson (Behavior, Age, Gender)` in the database. The customer now asks you to build a report visualizing shop visitors across gender and age like below:



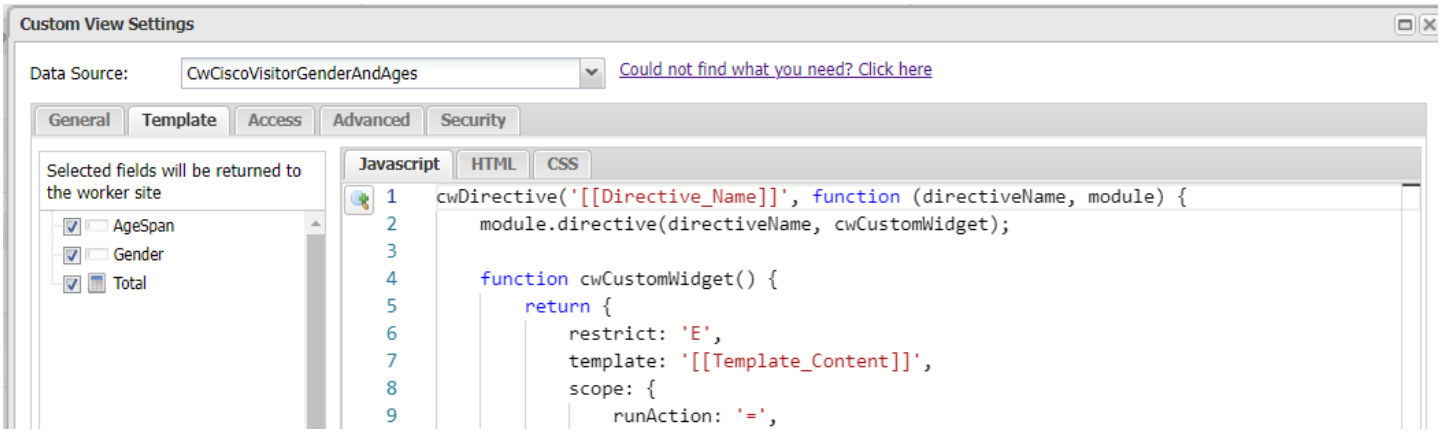
For the report, you will need to make a data source to aggregate the data in `DetectedPerson`.

```
[
    {
        "$match": {
            "Behavior": "GoInto"
        }
    },
    {
        "$group": {
            "_id": {
                "Gender": "$Gender",
                "AgeSpan" : {
                    "$switch": {
                        "branches": [
                                { "case": { "$lte": [ "$Age", 5 ] }, "then": "0-5" },
                                { "case": { "$lte": [ "$Age", 10 ] }, "then": "5-10" },
                                { "case": { "$lte": [ "$Age", 15 ] }, "then": "10-15" },
                                { "case": { "$lte": [ "$Age", 20 ] }, "then": "15-20" },
                                { "case": { "$lte": [ "$Age", 25 ] }, "then": "20-25" },
                                { "case": { "$lte": [ "$Age", 30 ] }, "then": "25-30" },
                                { "case": { "$lte": [ "$Age", 35 ] }, "then": "30-35" },
                                { "case": { "$lte": [ "$Age", 40 ] }, "then": "35-40" },
                                { "case": { "$lte": [ "$Age", 45 ] }, "then": "40-45" },
                                { "case": { "$lte": [ "$Age", 50 ] }, "then": "45-50" },
                                { "case": { "$lte": [ "$Age", 55 ] }, "then": "50-55" },
                                { "case": { "$lte": [ "$Age", 60 ] }, "then": "55-60" },
                                { "case": { "$lte": [ "$Age", 65 ] }, "then": "60-65" },
                                { "case": { "$lte": [ "$Age", 70 ] }, "then": "65-70" },
                                { "case": { "$lte": [ "$Age", 75 ] }, "then": "70-75" }
                            ],
                            "default": "75+"
                        }
                    }
                },
                "Total": { "$sum": 1}
            }
        },
    {
        "$project": {
            "_id": 0,
            "Gender": "$_id.Gender",
            "AgeSpan": "$_id.AgeSpan",
            "Total": "$Total"
        }
    },
    {
        "$sort": {
            "AgeSpan": 1
        }
    }
]
```

After that, define the field mapping in the **Design** tab.



Finally, create a custom widget using the data source. When you select the data source, the corresponding fields will be loaded and ready to use. How to make the widget is out of the scope of this example.



Restful data source

Coming soon...

Process overview

A process in Casewhere defines the type of data a case can have and the workflows connected to it. When a process is started, Casewhere creates a case to manage and track its progress. To make the difference between a process and a case clearer, consider this example:

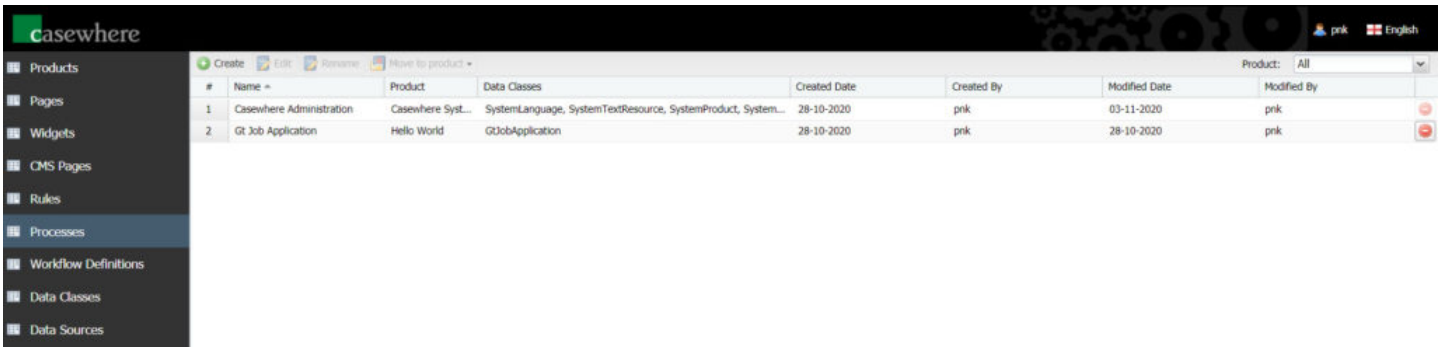
A process could be "Applying for a job." When a user applies, a case is created that includes the application, a cover letter, related documents, messages, and more. If the user later triggers a workflow to upload a document, they are updating the case by adding the document to it.

To learn more about working with processes and cases in Casewhere, please refer to the articles below:

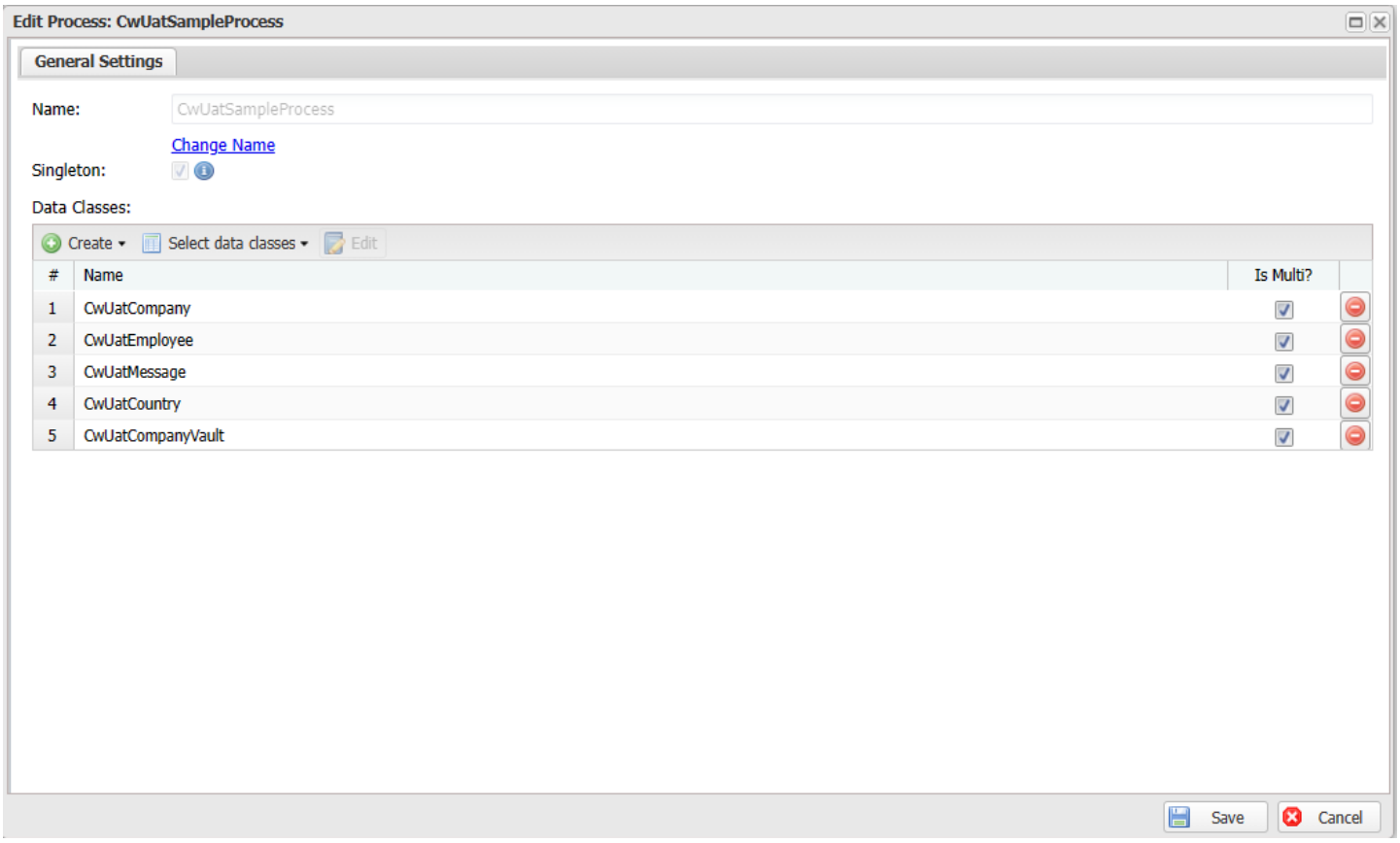
- Process management: Learn how to create and manage processes in Casewhere.
- Case creation logic: Understand the case creation process and its associated constraint logic.

Process management

In Casewhere Admin, navigate to **Processes** to set up processes for your solution. From here, you can create, edit and manage processes like any product resource.



The process editor is where you configure the process, specify which data classes can be stored in the cases.



A process has the following properties/settings:

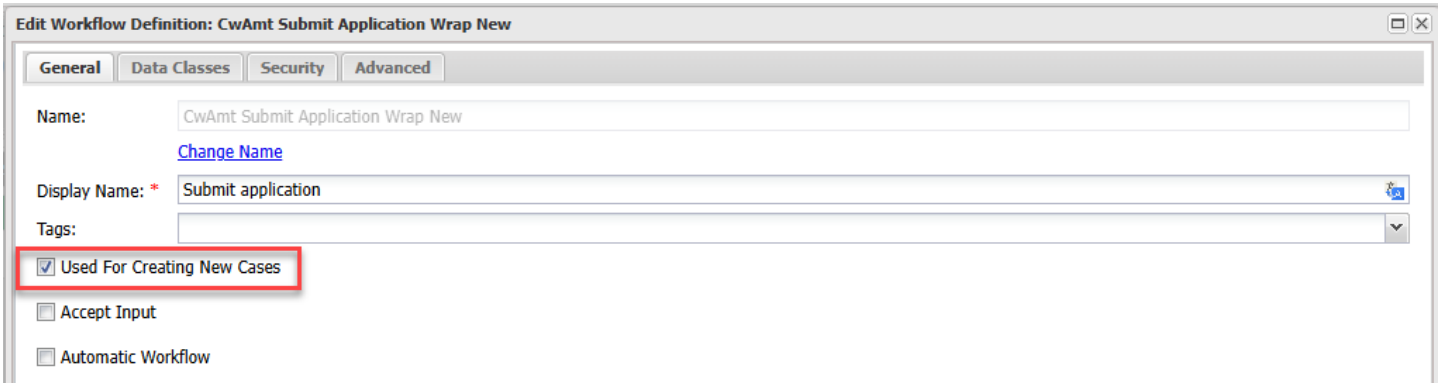
NAME	TYPE	DESCRIPTION
Name	string	The name of the process. Process names must be unique.
Singleton	bool	if enabled, the process will only have one case. Please note that this setting is only enabled when you create a new process.
Data classes	A list of data classes	Manage which data classes can be stored in the case. If you enable **Is Multi** for a data class, you can have many data objects created in the same case.

Case creation logic

To work efficiently with Casewhere, developers and configurators must understand the case creation logic and its constraints on workflows and data.

Workflow constraints

When creating a new workflow, you must decide whether it will create a new case or modify an existing one.

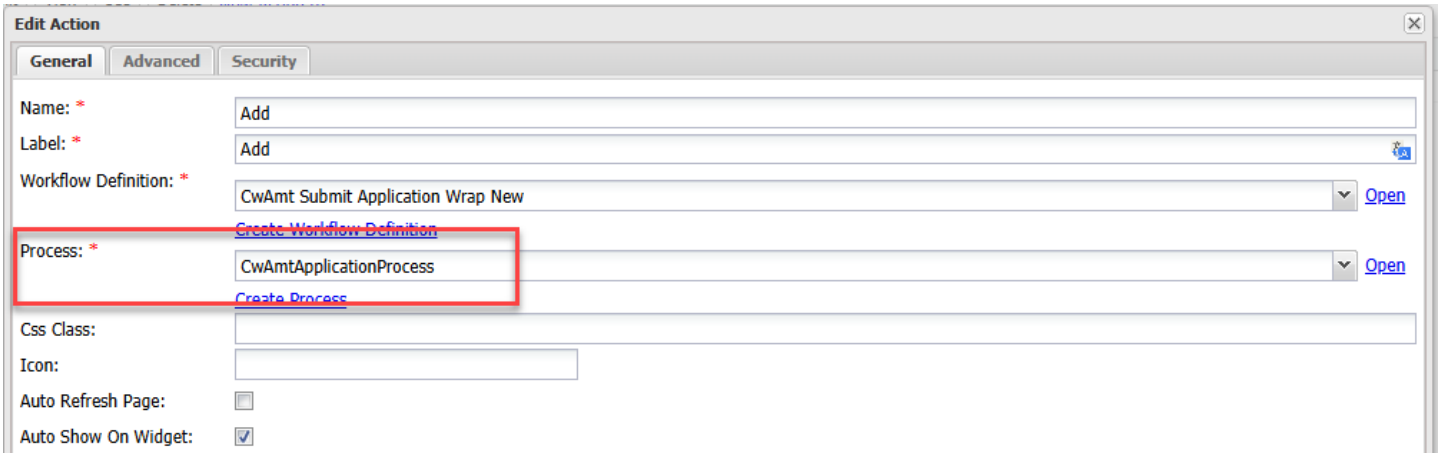


When triggering a create-case workflow, Casewhere creates a new case object with the state `Draft`. All data objects generated during case creation will also have their `CaseState` set to `Draft`. Access to a draft case and its associated data is restricted as follows:

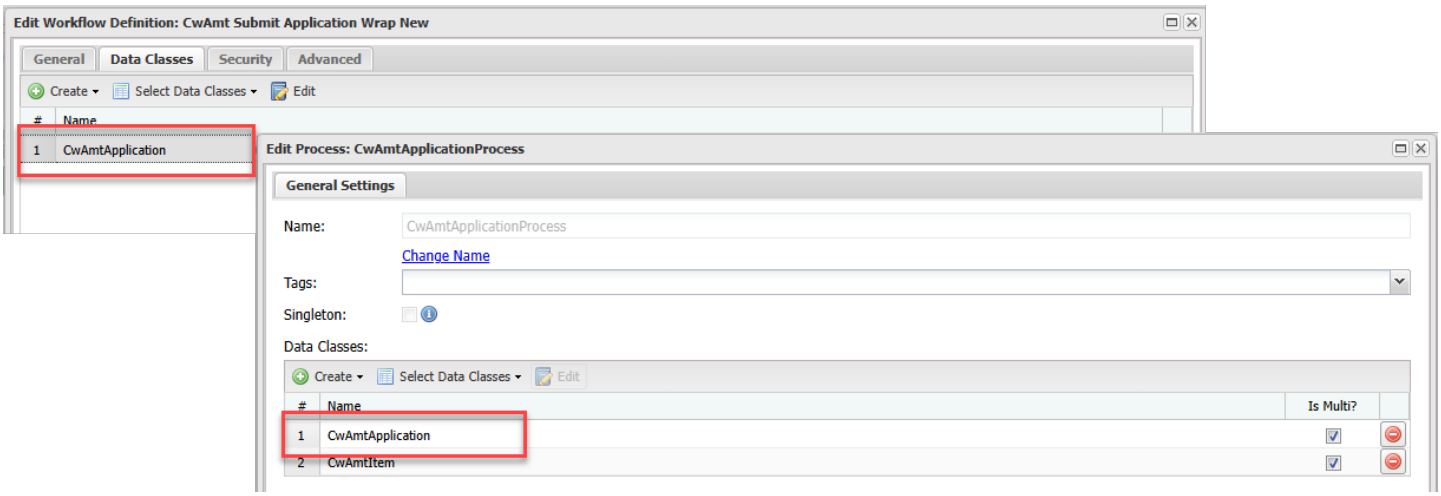
- Draft cases are visible only in data widgets and queries that explicitly include them in their filtering logic.
- Cases cannot be edited until they are fully created. For example, end users cannot trigger edit-case workflows while the case is in the `Draft` state. Therefore, they must first complete the create-case workflow before executing any further workflows on the case. To do so, end users must finish all required activities defined in the workflow. Once the create-case workflow is completed, Casewhere updates the case state to `Created`.

Configuration constraints

Whenever you set up a trigger for a "create-case" workflow—such as from a widget action on a webpage or an API call via a web trigger—you must specify the process in which the workflow will run.



Please note that Casewhere will only display processes related to the selected workflow definition. In other words, the workflow must work with the data classes supported by the process.



Data constraints

Casewhere only allows adding new data objects to a case if the case process supports the corresponding data classes. Additionally, Casewhere controls whether a process accepts a single or multiple data objects for a specific case. For example, the "Applying for job" process should only allow one "Application" data object.

When a data object is created in a draft case, the `CaseState` of the data object is also set to `Draft`. Access to draft data objects will be restricted similar to the draft cases. When the create-case workflow is completed, Casewhere will set the state of the case and all associated data objects to `Created`.

If the workflow enables draft saving, Casewhere will always create the case and its data objects as `Created`.

Rule overview

Casewhere is a rule-based system, meaning rules are used throughout the entire system for making business decisions such as triggering processes, workflow navigation, data validation, access control, etc. Rules in Casewhere are scripted in CSharp. On top of that, Casewhere provides a set of APIs for interacting with Casewhere objects, e.g., you can make a rule to prevent users not in a configurable whitelist from accessing a specific web page.

Casewhere has the following types of rules:

- Data trigger rule: Decide when to trigger a workflow based on data conditions.
- Web trigger rule: Decide when to trigger a workflow based on input from the web request.
- Access rule: Control user access to worker sites, pages, and widgets.
- Data protection Rule: Securing data access, control data ownership, etc.
- Workflow navigation rule: Control the navigation of workflow.
- Scalar-function rule: Rule is used as a computing function.

Rule syntax

There are two syntaxes of scripting rules.

Expression syntax

This syntax is most suitable for simple rules.

```
@do.Status != "ReadyForSend"
```

Function syntax

This syntax is suitable for more complex rules, which are better organized using multiple script lines. You have to begin with the `#predicate` keyword and `return` the value explicitly.

```
#predicate

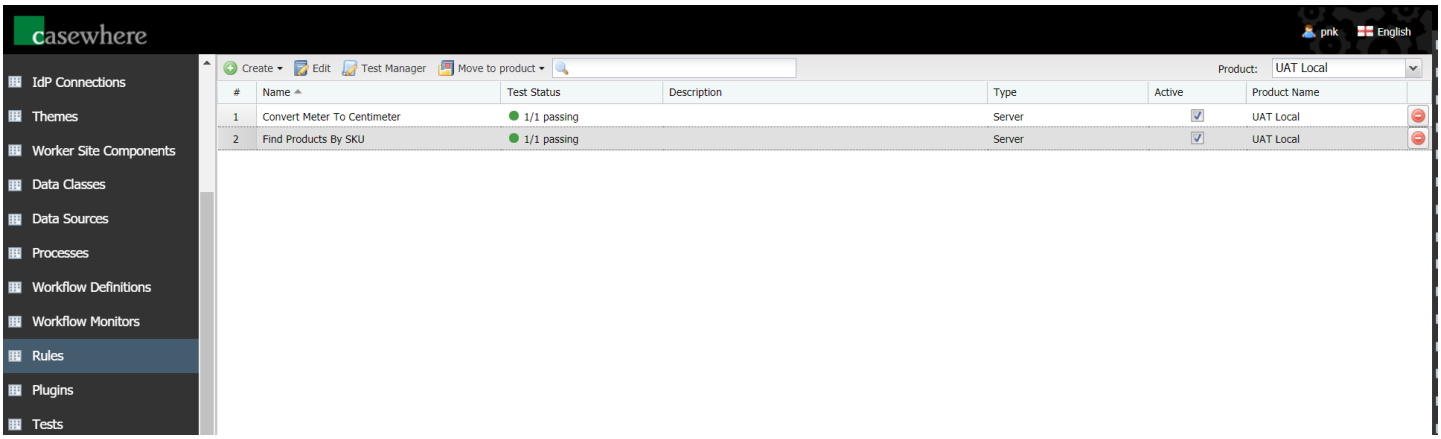
// Parse the request header
var dataApi = @apiFactory.Get<IDataApi>();
if (!@headers.ContainsKey("Authorization"))
{
    return false;
}

// Query Api Keys in DB
var headerKey = @headers["Authorization"].First();
var filter = FilterBuilder.Create().Eq("ApiKey", headerKey).Eq("Active", true).Build();
var found = dataApi.Any("Applications", filter);

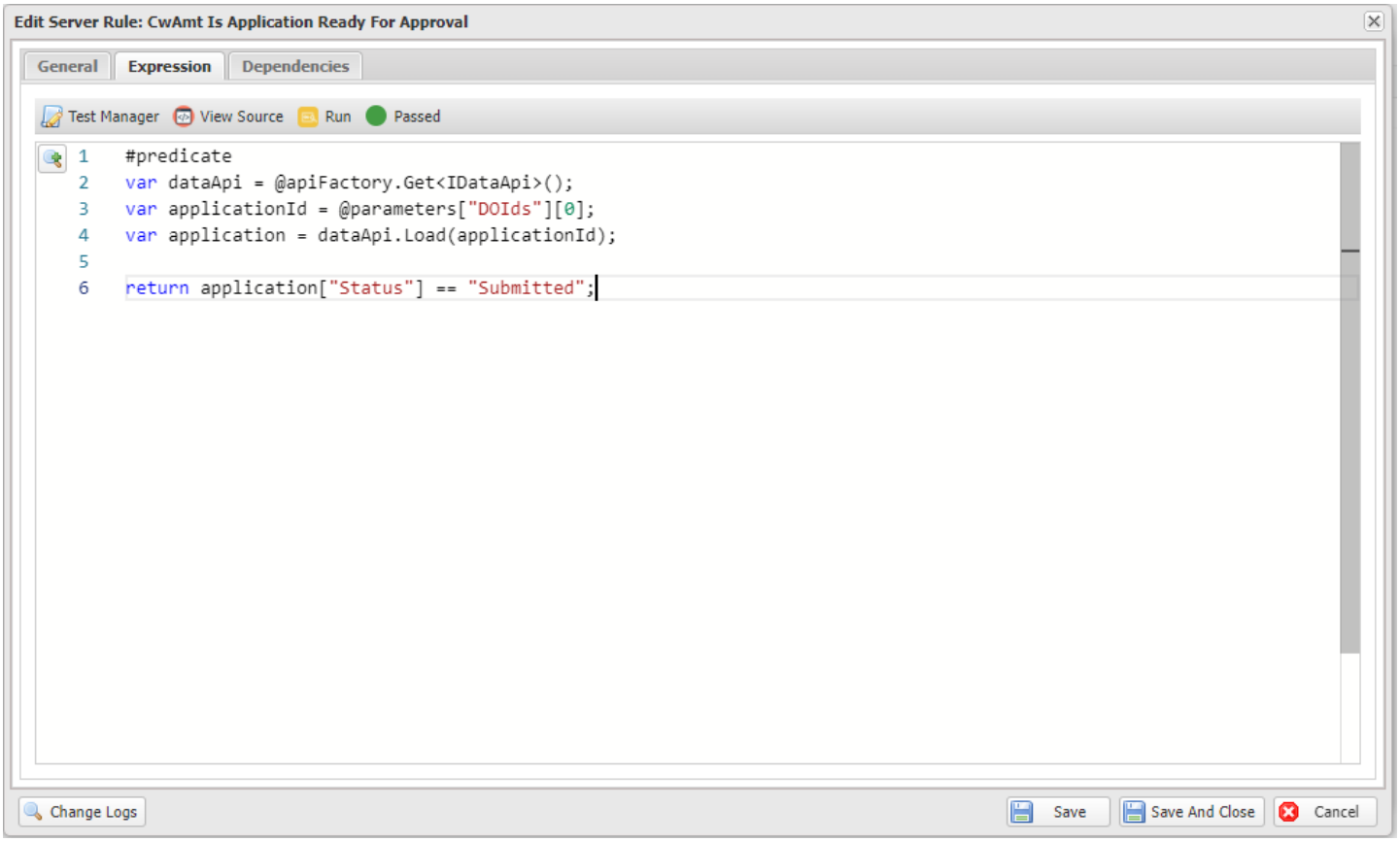
return found;
```

Rule management

In Casewhere Admin, navigate to **Rules** to set up rules for your solution. From here, you can create, edit and manage rules like any product resource.

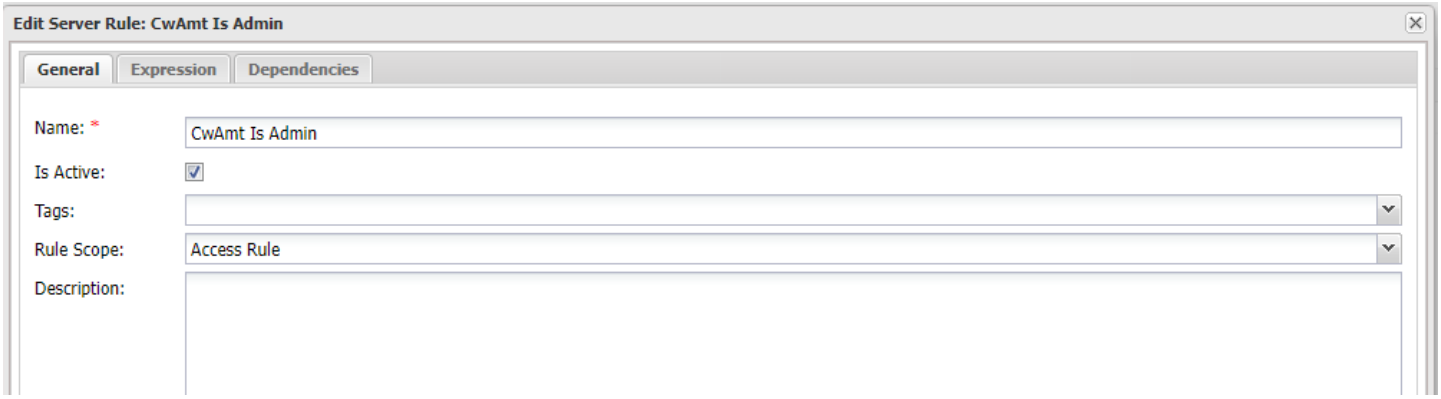


The rule editor is where you compose the rule expression. Here you can also see which resources are using the rule.



Rule information

A rule has the following properties/settings:

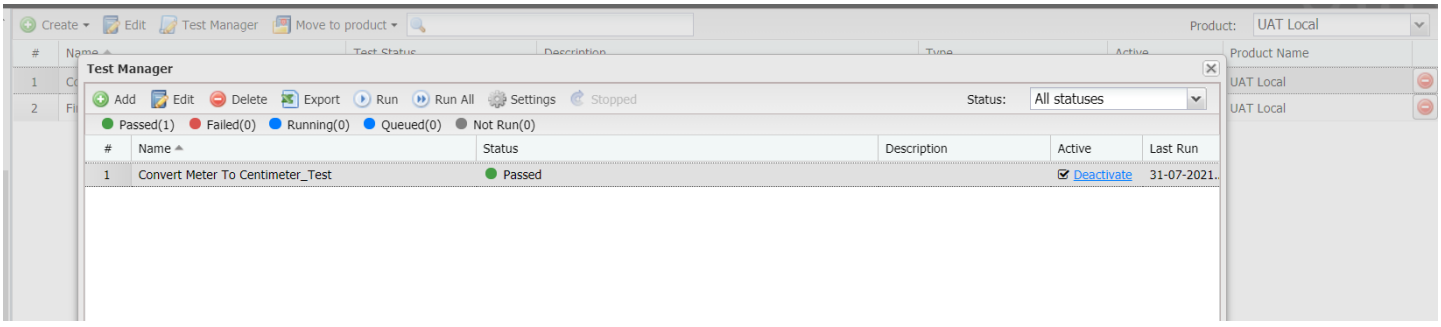


NAME	TYPE	DESCRIPTION
Name	string	The name of the rule. Rule names must be unique.
Expression	string	A C# expression. Learn more about the rule syntax here.
Description	string	Describe more what the rule is for, what can of parameters it accepts, etc.
Is Active	bool	Determine if the rule is active or not. Casewhere will ignore inactive rules, but it will throw an error if an inactive rule is used in a workflow at runtime.
Rule Scope	enum	Specify where the rules are applied. For example, **Access Rule** is used to protect your solutions, while **Computation Rule** serves a general purpose, such as performing an action and returning a value.

Rule testing

You can write automation test for rules in Casewhere. It's recommended to have tests for every rule you write and package them in the same product. This way, your rules will look more reliable.

Select the rule you want to write automation tests, then select **Test Manager**. From here, you can manage all tests related to the rule.



Access rule

Introduction

Access rules are designed to control user access to worker sites, pages, and widgets.

Rule syntax

Keywords

You can use the following keywords to work with access rules:

NAME	TYPE	DESCRIPTION
@user	DslUser	The user in context.
@apiFactory	IManagedApiFactory	To construct Casewhere API objects. Read here for the full list of available Apis

A `DslUser` object has the following properties:

NAME	TYPE	DESCRIPTION
UserId	string	Id of the user.
Username	string	Username used for login.
Has	bool (string claimType)	A method used to determine if the user has a claim or not.
Contains	bool (string claimType, string value)	A method used to determine if the user has a claim value or not.
WorkerSite	DslWorkerSite	The worker site in context.

A `DslWorkerSite` object has the following properties:

NAME	TYPE	DESCRIPTION
Name	string	Name of the worker site. Worker site name must be unique.
DomainName	string	Domain of the worker site.
TimezoneId	string	Standard name of the timezone object retrieved from TimeZoneInfo.GetSystemTimeZones. For example, Romance Standard Time,
TimezoneOffset	double	A number indicates the time difference in minutes between the current time zone's standard time and Coordinated Universal Time (UTC).
ShortDate	string	Short date format. For example: dd.MM.yyyy
LongDate	string	Long date format. For example: dd.MM.yyyy HH:mm
DecimalSeparator	string	It could be "," or "." depending on the configuration.

NAME	TYPE	DESCRIPTION
ThousandSeparator	string	It could be "," or "." depending on the configuration.

Examples

Example 1

Role-based access control – Only users having a specific role can access the resource.

```
@user.Contains("Role", "Manager")
```

Example 2

Only whitelisted users can access the resource.

```
#predicate

var dataApi = @apiFactory.Get<IDataApi>();
var filter = FilterBuilder.Create().Eq("Username", @user.Username).Eq("Active", true).Build();
var found = dataApi.Any("WhitelistedUsers", filter);

return found;
```

Example 3

```
#predicate

var dataApi = @apiFactory.Get<IDataApi>();
var filter = FilterBuilder.Create().Eq("Username", @user.Username).Eq("Active", true).Build();
var found = dataApi.Any("WhitelistedUsers", filter);

return found;
```

Scalar-function rule

Introduction

Casewhere rule is not only about boolean predicates. You can use rule to implement a scalar function that does some logic and returns a value. It is always a good practice to extract computing logic in workflows into rules that can be reused in different places.

Rule syntax

Keywords

You can use the following keywords to work with scalar-function rules:

NAME	TYPE	DESCRIPTION
@users	DslUser	The user object in context. Read access rule for more information.
@parameters	DynamicDataObject	You can access attributes using the "dot" operator. For example, "@parameters.address".
@apiFactory	IManagedApiFactory	To construct Casewhere API objects. Read here for the full list of available Apis.

Examples

Example 1

Assuming you have a rule "Convert Meter To Centimeter" that converts centimeter to meter:

```
@parameters.meterValue * 100
```

You can evaluate the rule in a workflow scripted activity using `RuleApi`:

```
var ruleApi = ctx.Use<IRuleApi>();

var parameters = new Dictionary<string, object>();
parameters["meterValue"] = 1;

var cm = ruleApi.Eval<int>("Convert Meter To Centimeter", parameters);
```

Example 2

Assuming you need to implement a function to query products by SKU and you want to reuse it in many places. You can create a rule "Find Products By SKU" like below:

```
#predicate

var dataApi = @apiFactory.Get<IDataApi>();
var filter = FilterBuilder.Create().Eq("SKU", @parameters.SKU).Build();
var result = dataApi.Search(DataObjectApiQuery
    .For("Product")
    .FilterBy(filter));

return result.Data;
```

You can evaluate the rule in a workflow scripted activity using RuleApi:

```
var ruleApi = ctx.Use<IRuleApi>();

var parameters = new Dictionary<string, object>();
parameters["SKU"] = "LAPTOP";

var products = ruleApi.Eval<List<DynamicDataObject>>("Find Products By SKU", parameters);
```

Workflow navigation rule

Introduction

Workflow navigation rule controls Workflow navigation. At this point, the workflow navigation rule is embedded in the workflow definition and cannot be shared among different workflow definitions, unlike other rule types.

Rule syntax

Keywords

You can use the following keywords to work with workflow navigation rules:

NAME	TYPE	DESCRIPTION
@ctx	DslCompilerContext	Provides APIs to access data of the running workflow. Read more about @ctx here.
@apiFactory	IManagedApiFactory	To construct Casewhere API objects. Read here for the full list of available Apis

Examples

Example 1: Imagine you are implementing a "Delete object" workflow that has two activities in the following order:

1. Form activity: show a confirmation message. The user's decision is recorded in the variable "agreed".
2. Scripted activity: delete the data object if the user agrees.

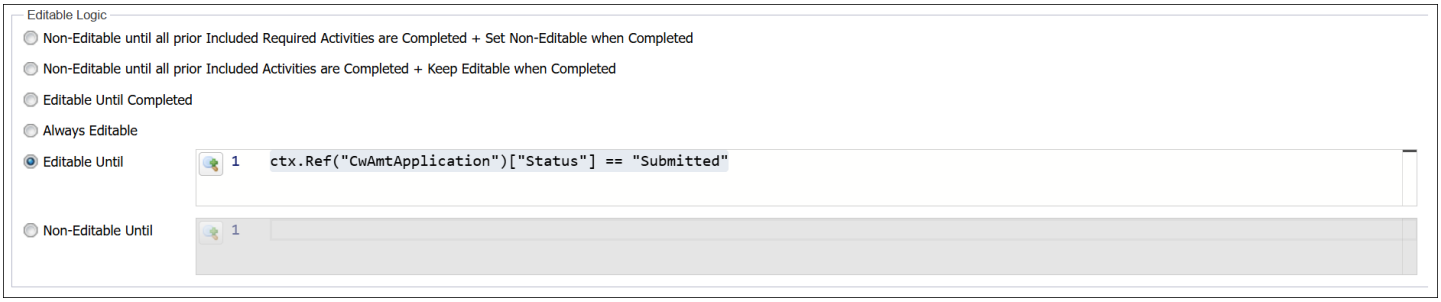
You will need to configure the inclusion rule for the scripted activity using the option "Included When" with the rule below:

```
@ctx.Get<bool>("agreed")
```



Example 2: The submission form will be disabled and become non-editable after the user submits their application.

```
ctx.Ref("CwAmtApplication")["Status"] == "Submitted"
```



Coming soon...

Data trigger rule

Introduction

Data trigger rule is used to determine whether a new workflow should be triggered when a data object is created, updated, or deleted. You can learn more about data trigger here.

Rule syntax

Keywords

You can use the following keywords to work with data trigger rules:

NAME	TYPE	DESCRIPTION
@do	DynamicDataObject	The data object in context, which is triggering the workflow. You can access the attributes using the "dot" operator.
@event	DataChangeNotification	The change event in context, which is triggering the workflow.
@changes	DynamicDataObject	The changed values. This keyword is valid for update events only. You can access the attributes using the "dot" operator.

A `DataChangeNotification` object has the following properties:

NAME	TYPE	DESCRIPTION
IsCreated	bool	Determine whether the event is about creating a new data object or not.
IsDeleted	bool	Determine whether the event is about deleting a data object or not.
CaseId	string	Id of the case in context.
WorkflowName	string	Name of the workflow emitted the event.
ActivityName	string	Name of the workflow activity emitted the event.
DataClass	string	Data class name.
DataObjectId	string	Id of the data object in context.
Timestamp	DateTime	The time the event is emitted.

Examples

Example 1: Integrate independent components

Applications can trigger the 'Sending Email' workflow independently of the Email Component by simply updating the status of the Email object.

```
!@event.IsCreated && @changes != null && @changes.Status != "ReadyForSend"
```

Example 2: Avoid duplication

If we have logic like this “All objects created must be activated by default”, in many cases, it’s a good idea to use a data trigger to

avoid repeating the logic every time we create a new object.

```
@event.IsCreated
```

Web trigger rule

Introduction

Web trigger rule is used to determine whether a new workflow should be triggered when Casewhere receives an HTTP request. You can learn more about web trigger here.

Rule syntax

Keywords

You can use the following keywords to work with web trigger rules:

NAME	TYPE	DESCRIPTION
@parameters	IDictionary<string, string>	Query string parameters.
@headers	IDictionary<string, IEnumerable>	Header key-value pairs.
@body	DynamicDataObject	A data object represents the request payload. You can access the attributes using the “dot” operator. For example, “@body.phoneNumber”.
@apiFactory	Factory object	To construct Casewhere API objects. Read here for the full list of available Apis.

Examples

Example 1

For the request “api/product?category_id=1” we can access the syntax below.

```
@parameters["category_id"] == "1"
```

Example 2

Verify the Api Key in the request headers.

```
#predicate

// Parse the request header
var dataApi = @apiFactory.Get<IDataApi>();

if (!@headers.ContainsKey("Authorization"))
{
    return new
    {
        StatusCode = 400,
        Message = "Authorization header is missing"
    };
}

// Query Api Keys in DB
var headerKey = @headers["Authorization"].First();
var filter = FilterBuilder.Create().Eq("ApiKey", headerKey).Eq("Active", true).Build();
var found = dataApi.Any("Applications", filter);

if (!found)
{
    return new
    {
        StatusCode = 401,
        Message = "Invalid Api key"
    };
}

return true;
```

Security policy rule

Introduction

We already know about how to use the Access rule to control user access to worker sites, pages, and widgets. However, the result of the Access rule is a boolean value (true/false), which is not enough to control user access to data. For example, we may want to control user access to data based on the user's department or other criteria. In such cases, we need to use the Security policy rule.

The Security policy rule provides more flexibility to control user access with the following features:

- Support for inheritance: The children's security policy must be inherited from the parent security policy.
- The children can override the parent security policy.
- The security policy can be applied to many types of resources.

Rule syntax

Keywords

You can use the following keywords to work with Security policy rules:

NAME	TYPE	DESCRIPTION
@user	DslUser	The user in context.
@apiFactory	IManagedApiFactory	To construct Casewhere API objects. Read here for the full list of available APIs.
@parameter	dynamic	Dictionary of parameter objects passed into the rule for verifying context.
@parentAccessMode	AccessMode?	Available for access rules, to the determine access right of the container element.

A `DslUser` object has the following properties:

NAME	TYPE	DESCRIPTION
UserId	string	The ID of the user.
Username	string	The username used for login.
Has	bool (string claimType)	A method used to determine if the user has a claim or not.
Contains	bool (string claimType, string value)	A method used to determine if the user has a claim value or not.
WorkerSite	DslWorkerSite	The worker site in context.

A `DslWorkerSite` object has the following properties:

NAME	TYPE	DESCRIPTION
Name	string	The name of the worker site. Worker site names must be unique.
DomainName	string	The domain of the worker site.

NAME	TYPE	DESCRIPTION
TimezoneId	string	The standard name of the timezone object retrieved from TimeZoneInfo.GetSystemTimeZones. For example, "Romance Standard Time".
TimezoneOffset	double	A number indicating the time difference in minutes between the current time zone's standard time and Coordinated Universal Time (UTC).
ShortDate	string	The short date format. For example: "dd.MM.yyyy".
LongDate	string	The long date format. For example: "dd.MM.yyyy HH:mm".
DecimalSeparator	string	The decimal separator. It could be "," or "." depending on the configuration.
ThousandSeparator	string	The thousand separator. It could be "," or "." depending on the configuration.

In Security policy rules, we can return these access modes:

NAME	TYPE	DESCRIPTION
Read	string	The user can read the resource.
Write	string	The user can read and write the resource.
Deny	string	The user cannot access to the resource

Compatibility and effect

ACCESS RULE	SECURITY POLICY RULE	CONTEXT	DESCRIPTION
true	AccessMode.Read/Write	Pages	Show on UI
false	AccessMode.Deny	Pages	Un-authorized error
true	AccessMode.Read/Write	Widget	Show on UI
false	AccessMode.Deny	Widget	Hide on UI
true	AccessMode.Write	Widget Action	Show UI and executable
	AccessMode.Read	Widget Action	Show UI and non-executable
false	AccessMode.Deny	Widget Action	Hide on UI

Examples

Example 1

Role-based access control – Only users having a specific role can access the resource.

Single line

```
@user.Contains("Roles", "admin") ? AccessMode.Write : AccessMode.Read;
```

Multi-line

```
#predicate
if (@user.Contains("Roles", "admin"))
{
    return AccessMode.Write;
}
// other logic
return AccessMode.Read;
```

Example 2

Only whitelisted users can access the resource.

```
#predicate

var dataApi = @apiFactory.Get<IDataApi>();
var filter = FilterBuilder.Create().Eq("Username", @user.Username).Eq("Active", true).Build();
var found = dataApi.Any("WhitelistedUsers", filter);

return found ? AccessMode.Read : AccessMode.Deny;
```

Example 3

Read input parameter via `@parameters` variable to verify users can access the resource.

```
#predicate

if (!@parameters.Has("UserName"))
{
    return AccessMode.Deny;
}

var dataApi = @apiFactory.Get<IDataApi>();
var userName = @parameter["UserName"];
var filter = FilterBuilder.Create().Eq("Username", username).Eq("Active", true).Build();
var found = dataApi.Any("WhitelistedUsers", filter);

return found ? AccessMode.Read : AccessMode.Deny;
```

Example 4

Access the security policy rule result of parent resource via `@parentAccessMode` variable.

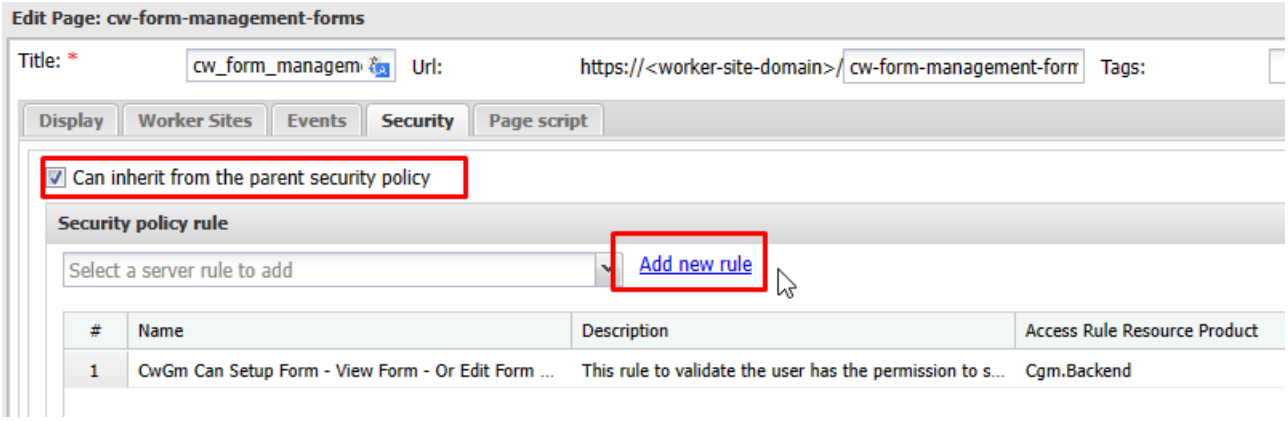
```
#predicate

if (@parentAccessMode != null && @parentAccessMode == AccessMode.Deny)
{
    // if parent access mode is Deny, the child will be denied access to the resource.
    return AccessMode.Deny;
}

// handle other logic
// ...
return AccessMode.Read;
```

How to apply Security policy rules

To apply Security policy rules for Pages, Widgets and Widgets actions, you can go to **Security tab** in the Edit resource panel.



- If you allow "Can inherit from parent", the children resources will inherit the parent security policy. The children resources can override the parent security policy if necessary.
- You can **Add new rule** or **Select** multiple rules from the list. The rules will be combined by the logical operator **AND** including the result of the parent security policy if inherited.
    - If any of the rules returns **Deny**, the user will be denied access to the resource.
    - If none of the rules assign to the resource, the resource will have **Write** access mode.

Page

Use the security policy rule is not only to control access to the page but also to control how the page's widgets and widget actions show on UI and can be executable or not. The page's security policy rule will be applied to all widgets and widget actions on the page.

Example

A security policy rule sample to control the user can read, write or deny access to the page.

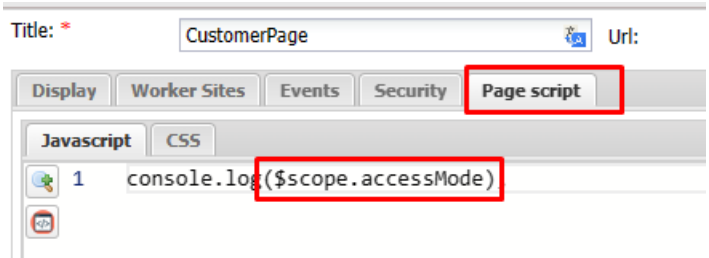
```
#predicate

if (@user.Contains("Roles", "Admin") ||
    @user.Contains("Roles", "Manager"))
{
    return AccessMode.Write;
}

if (@user.Has("Roles", "User"))
{
    return AccessMode.Read;
}

return AccessMode.Deny;
```

You can also access to Access Mode of Page at client site via Page script or events. This way allow you can customize the UI logic base of that access mode.



Widget and Widget Actions

Using the security policy rule for **Widgets** or **Widget actions** is similar to using it for **Pages**. The widget's security policy rule

will be applied to all widget actions on the widget.

It is more flexible to control the widget's security policy rule by using the widget's context. For example, you can control the use can view or do edit/delete Widget actions base on the Access mode of the Widget.

To access the widget's access mode at client side, you can use the Page events and read it from

Workflow overview

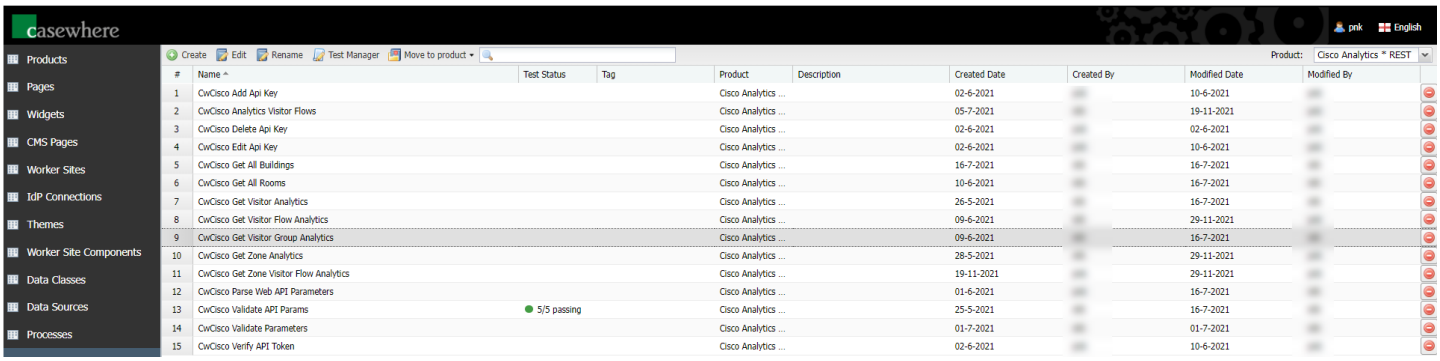
In Casewhere, all actions related to cases require a workflow to be executed, whether it involves purchasing parking tickets, applying for jobs, or sending newsletters. Casewhere offers two types of workflows:

- UI workflows: Display data to your users or collect information from them using UI elements. UI workflows are triggered by end-users through widgets. Casewhere provides a workflow definition editor with a form builder, allowing you to design complex and dynamic UI activities.
- Automatic workflows: Triggered automatically by events in the system, changes in the database, or other workflows. Casewhere offers a wide range of workflow triggers, providing the flexibility needed to design solutions for a variety of complex scenarios.

To define how workflows operate in Casewhere, you must create a workflow definition in the Casewhere Admin. This will specify the data classes (and thus the processes and cases) with which the workflows can interact, as well as the activities that will execute under specific conditions. Since workflows do not execute automatically, you will also need to define the workflow triggers, specifying who or what can initiate them.

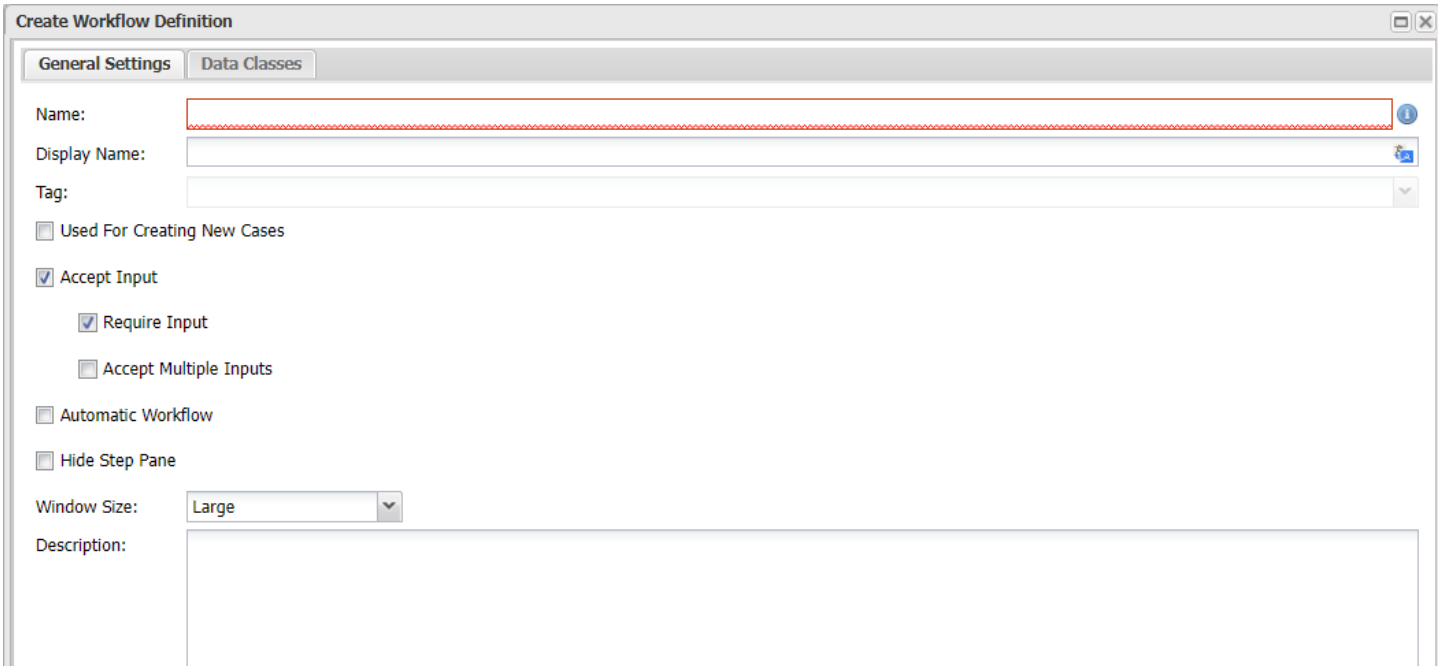
Workflow definition

You need to create a workflow definition in order to define how workflows work. In Casewhere Admin, navigate to **Workflow Definitions** to set up workflow definitions for your solutions. From here, you can create, edit and manage workflow definitions like any product resource.



Create workflow definition

Click **Create**, the **Create Workflow Definition** editor will appear as below.



Configure general settings

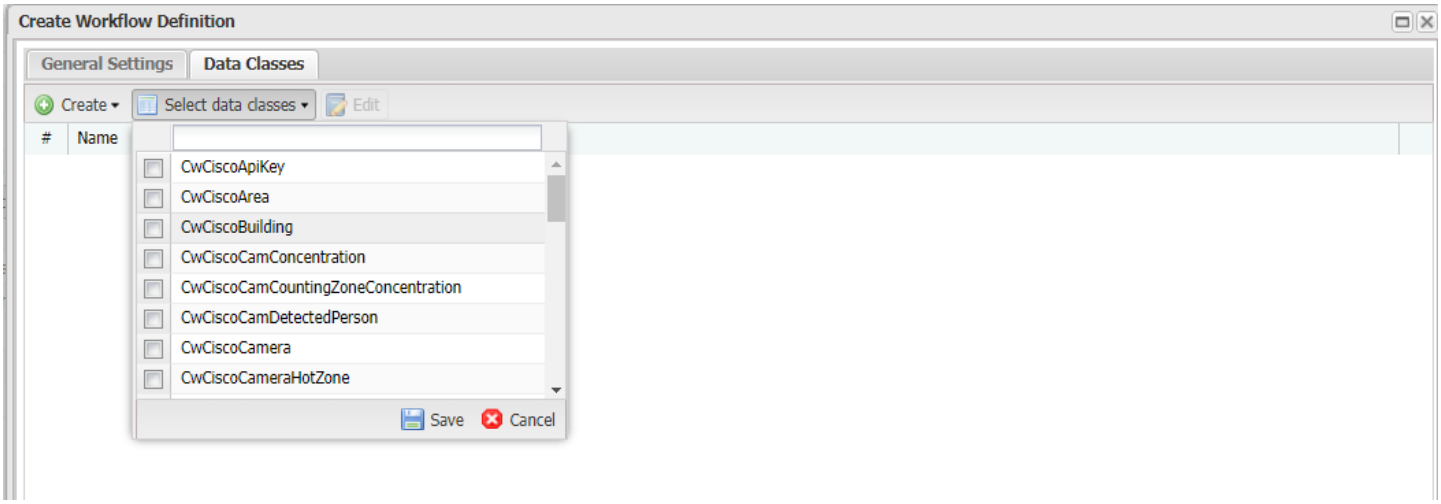
You need to set up the following when creating a new workflow:

FIELD	DESCRIPTION
Name	Workflow name must be unique
Display Name	The name appear on the workflow dialog in the worker site
Tag	To facilitate searching. Like any other Casewhere resources, you can use tags to manage workflows. This field is optional
Used For Creating New Cases	Check if the workflow creates a new case. Learn more about the case creation logic here.

FIELD	DESCRIPTION
Accept Input	Check if the workflow can accept data object(s) as input
Require Input	Check if the workflow requires input to function
Accept Multiple Inputs	Check if the workflow accepts multiple data objects as input
Automatic Workflow	Automatic or UI workflow. See here for more information
Non-persistent Workflow	When enabled, the workflow is created in memory only and will not be persisted in the database. This setting can help improve performance significantly, especially when maintaining the workflow state is unimportant, e.g., read-only workflows. This field is available only for automatic workflows.
Hide Step Pane	When enabled, Casewhere will hide the step pane in the workflow wizard. This field is available only for UI workflows
Window Size	Large, medium, or small. Though it is totally possible to customize the theme for workflow size, using these three options is sufficient for most cases. This field is available only for UI workflows
Description	This field is optional. You can use it for documentation

Configure data classes

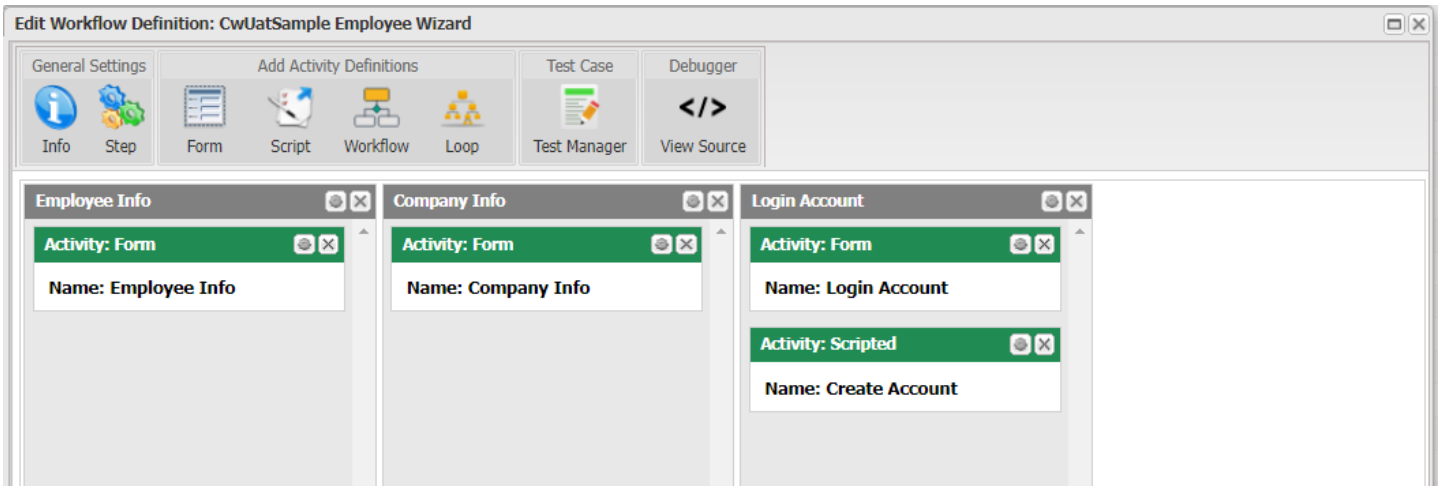
You must specify which data classes your workflow will create or update. Workflow definitions are not tightly coupled to processes. You can use the same workflow definition on cases from different processes. That means for a workflow to be executed on a case successfully, this case must support the same data classes.



Note: At the moment of writing (Casewhere v2.7.0 or earlier), a workflow requires at least one data class configured to run. If your workflow does not create or update anything, you still need to select a data class in the process your workflow is supposed to run in.

Edit workflow definition

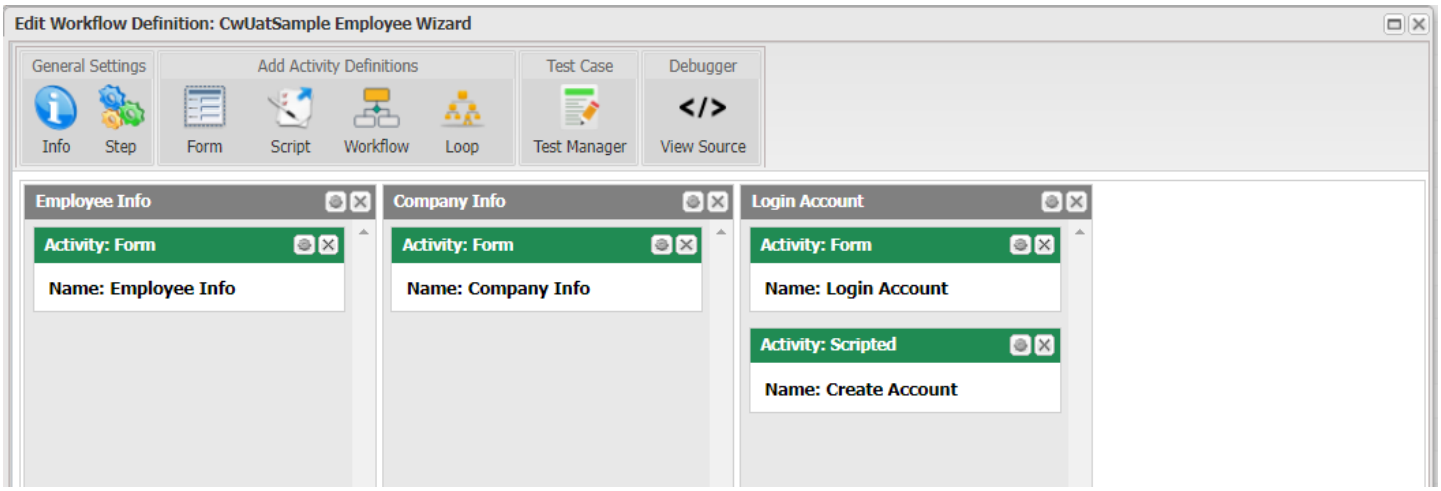
Select a workflow definition and click **Edit**. The **Edit Workflow Definition** editor will appear as below:

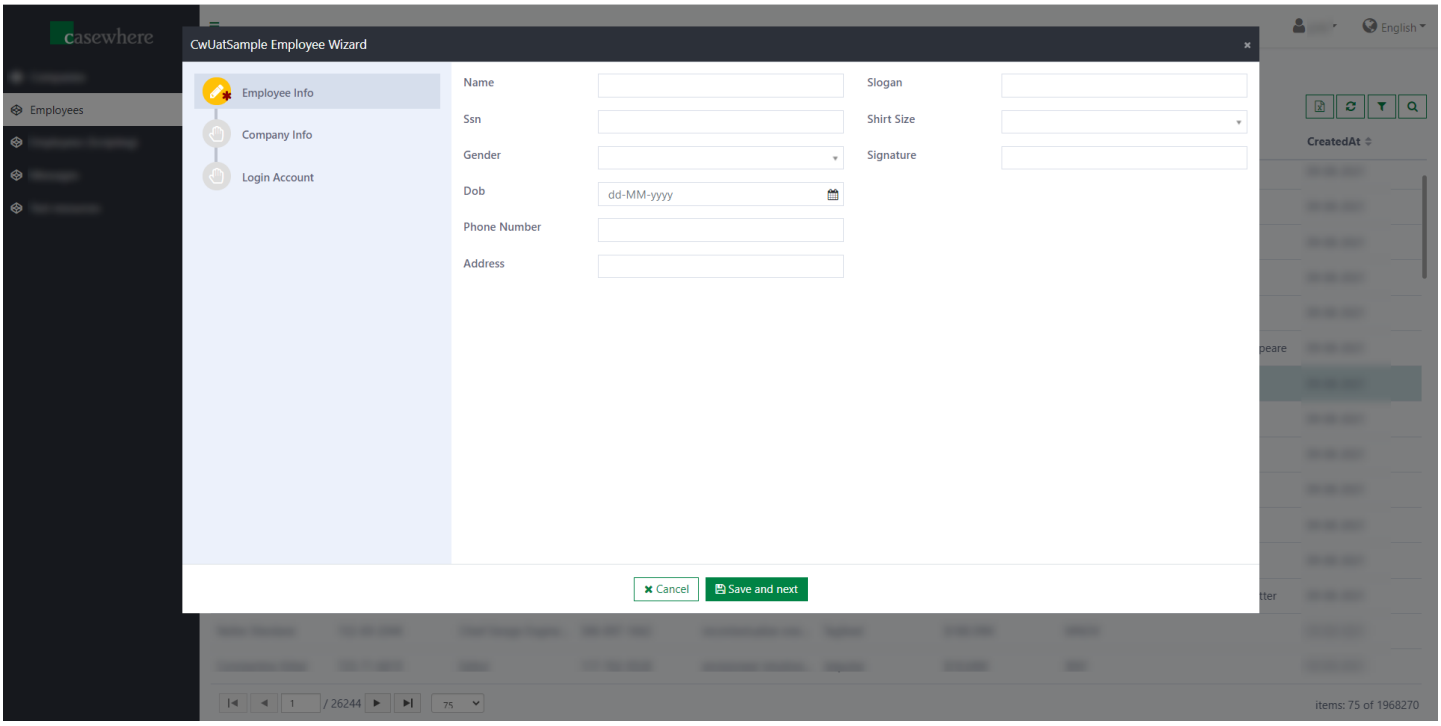


This editor is where you design the workflow's steps and activities and control how Casewhere navigates and executes them. See here to learn more about workflow navigation.

Workflow steps and activities

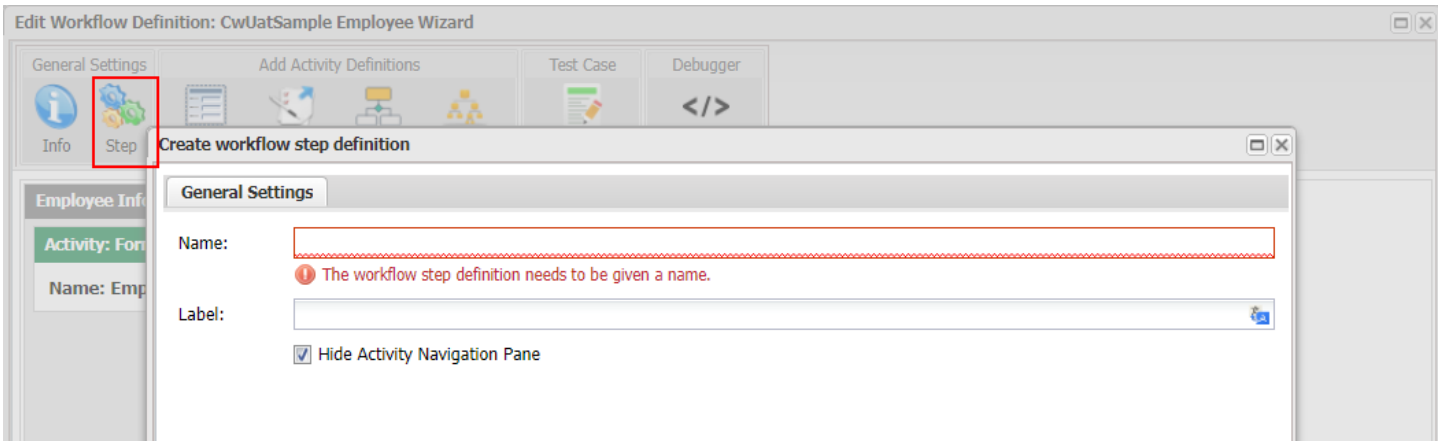
A Casewhere workflow is composed of steps and activities. Steps and activities are executed in the order they were designed. The below example shows how a workflow looks in Casewhere Admin and in worker site. Here you can see that a workflow is generally a sequence of steps, and a step can have one or more activities.





Configure steps

In the **Edit Workflow Definition** dialog, click **Step** to add in the toolbar to add a new step.



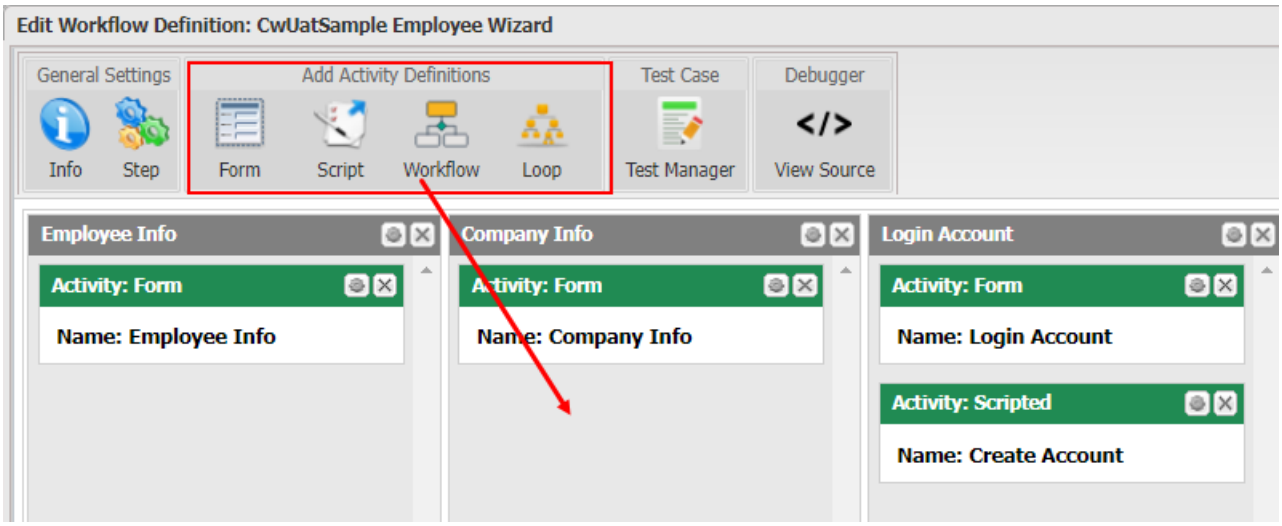
A step has the following fields:

NAME	DESCRIPTION
Name	Step name in a workflow must be unique
Label	The step title is displayed in the workflow wizard. You can use a text resource key here
Hide Activity Navigation Pane	If a step has more than one activity, you can enable a tab-pane to support your users in navigating between activities.

For automated workflows, steps don't have much meaning as they do in UI workflows and are mostly used for organizing activities in the workflow definitions. It's still a good practice to group related activities in the same step.

Configure activities

You can add one or more activities to a step by dragging items in the **Add Activity Definitions** section into the step panel.



Currently, Casewhere supports the following activity definitions:

NAME	TYPE	DESCRIPTION
Form	UI Activity	To collect information from or display information to end-users
Script	Automated Activity	Execute a C# script
Workflow	Automated Activity	Execute another workflow

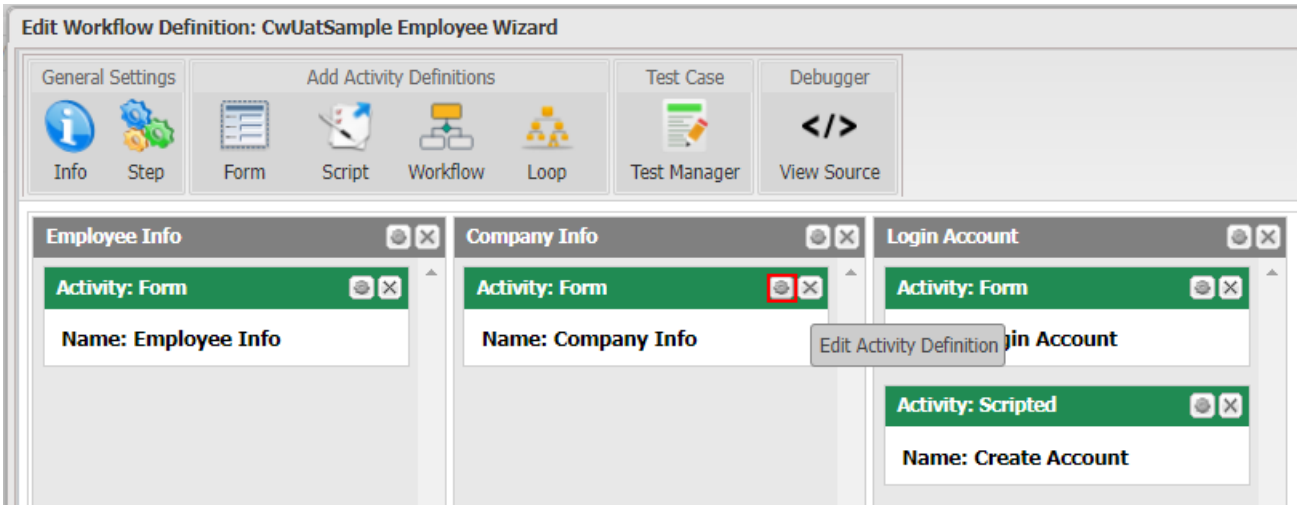
NAME	TYPE	DESCRIPTION
Loop	Automated Activity	Loop through the data objects filtered by a given condition and trigger a new workflow per data object

The key difference between a UI and an automated activity is that when the workflow engine navigates to a UI activity, it will idle while awaiting end-user interaction. On the other hand, no end-user interaction is needed for automated activities. The workflow engine will continue after executing the activity.

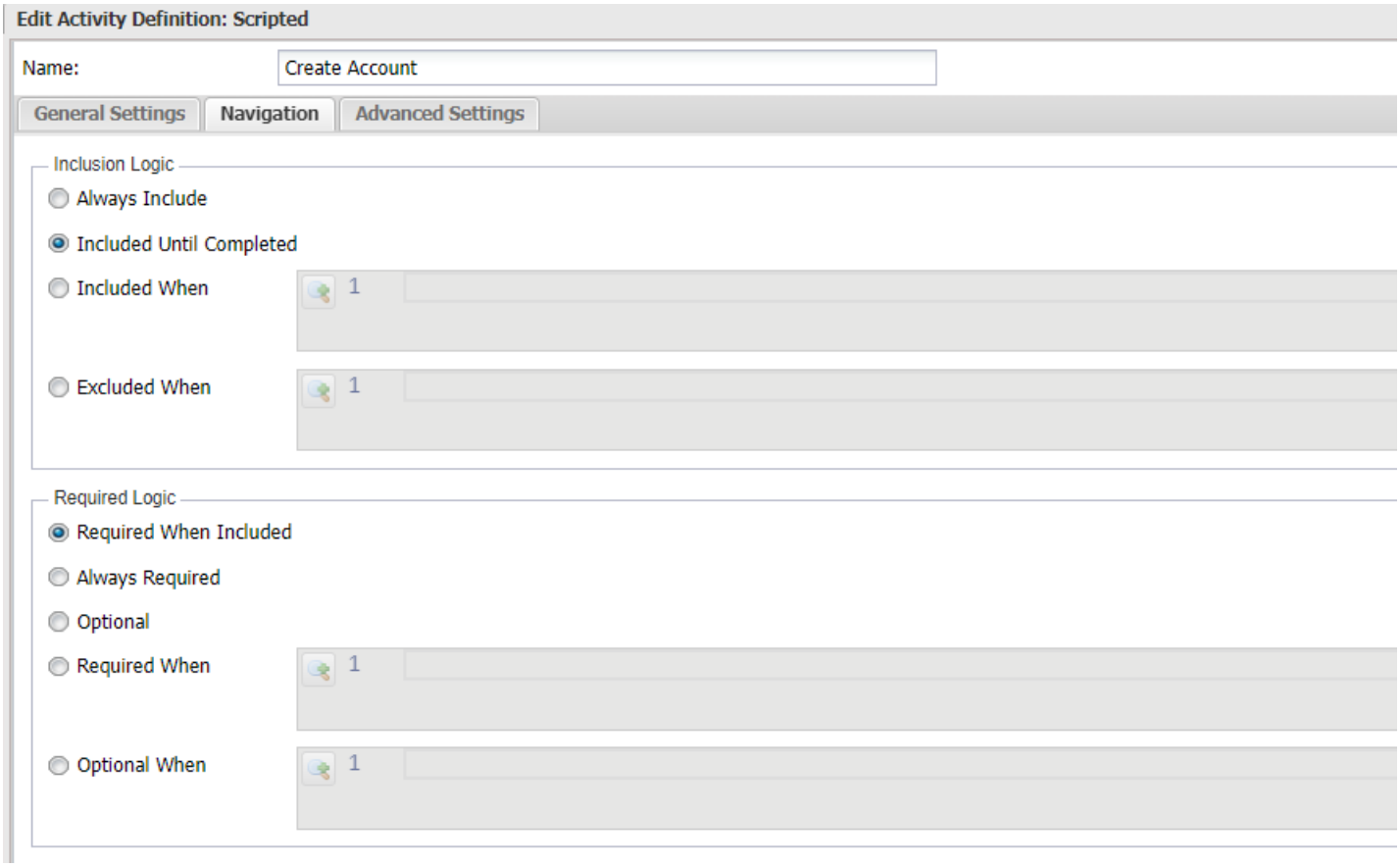
Workflow navigation

In order to work with Casewhere workflows efficiently, you, as the solution developer, need to understand how the workflow engine works, especially how it navigates from one activity to another.

After adding an activity to a step, click **Edit Activity Definition** to set up the activity definition.



The navigation settings can be found in the **Navigation** tab, which is available for all activity definitions.



Activity navigation

Inclusion logic

The inclusion logic controls whether the workflow engine should execute the activity or not. You can configure by choosing one of the following options:

- **Always Include**: The workflow engine will always execute the activity.

- **Included Until Completed**: After executing the activity, the workflow engine will exclude it from the execution path. If the activity is a UI activity, you will see it disappear in the workflow wizard.
- **Included When**: The workflow engine will execute the activity when the expression returns true. You can learn more about the syntax here.
- **Excluded When**: The workflow engine will skip the activity when the expression returns true. You can learn more about the syntax here.

For UI activities, the default setting is **Always Include** while it is **Included Until Completed** for automated activities.

Required logic

The required logic determines whether the activity is required to complete the workflow. A workflow completes when all required activities are completed. You can configure the logic by choosing one of the following options:

- **Required When Included**: The activity is only required when it is included.
- **Always Required**: The activity is always required.
- **Optional**: The activity is optional and does not impact the workflow completion.
- **Required When**: The activity is required when the expression returns true. You can learn more about the syntax here.
- **Optional When**: The activity is optional when the expression returns true. You can learn more about the syntax here.

The default value is **Required When Included**.

Editable logic

The editable logic controls whether or not the activity is available for user input. This setting is applicable for UI activities only. You can configure the logic by choosing one of the following options:

- **Non-Editable until all prior Included Required Activities are Completed + Set Non-Editable when Completed**
- **Non-Editable until all prior Included Activities are Completed + Keep Editable when Completed**
- **Editable Until Completed**: The activity is set to non-editable after it is completed.
- **Always Editable**: The activity is always editable.
- **Editable Until**: The activity is editable until the expression returns true. You can learn more about the syntax here.
- **Non-Editable Until:** The activity is non-editable until the expression returns true. You can learn more about the syntax here.

The default value is **Non-Editable until all prior Included Required Activities are Completed + Set Non-Editable when Completed**.

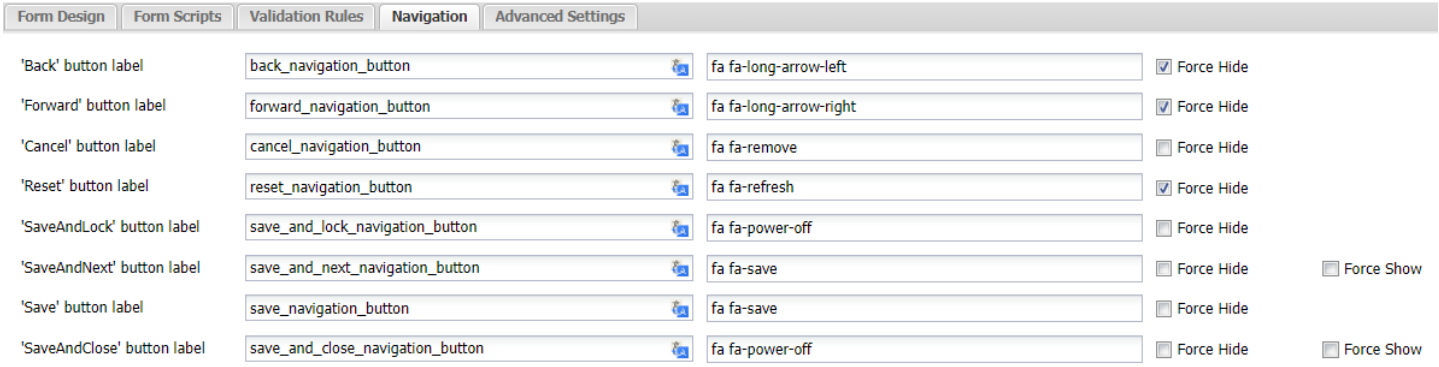
Completion logic

> **Notes**: These settings are rarely used and kept for the compatible reason only.

Auto Complete Activity Without User Interaction: When enabled, the workflow engine will evaluate the validation rules and automatically complete the form when all rules are passed.

Workflow wizard buttons

The workflow wizard provides a set of buttons for end-users to submit data, navigate back and forth. You can customize these navigation buttons from Casewhere Admin as part of the **Navigation** logic of the **Form Activity Definition**.



Below are the navigation buttons' default behaviors, but you can always override them by using **Force Hide** or **Force Show**.

- **Back**: Allow the user to move to the previous activity. This button does not save data.
- **Forward**: Allow the user to move to the next activity in the current execution path. This button does not save data.
- **Cancel**: Close the workflow wizard and discard all unsaved changes. This button always appears.
- **Reset**: Clear the unsaved changes and reload the activity.
- **SaveAndLock** (`deprecated`): This button only appears at the last required activity in the current execution path. After saving data successfully, Casewhere will lock the workflow from running again. Locking workflows have been deprecated since version 2.5.
- **SaveAndNext**: After saving data successfully, Casewhere will navigate the user to the next activity in the current execution path. This button will disappear at the last activity.
- **Save**: This button will appear at the last activity in the current execution path instead of the **SaveAndNext** button.
- **SaveAndClose**: After saving data successfully, the workflow engine will close the workflow wizard. This button will only appear at the last activity of the current execution path.

Here you can also customize the buttons' appearance:

- Label: Casewhere accepts text resource keys for button's label.
- Icon: CSS classes. For example, you can use Font Awesome to customize the icon.

Workflow navigation - More examples

Introduction

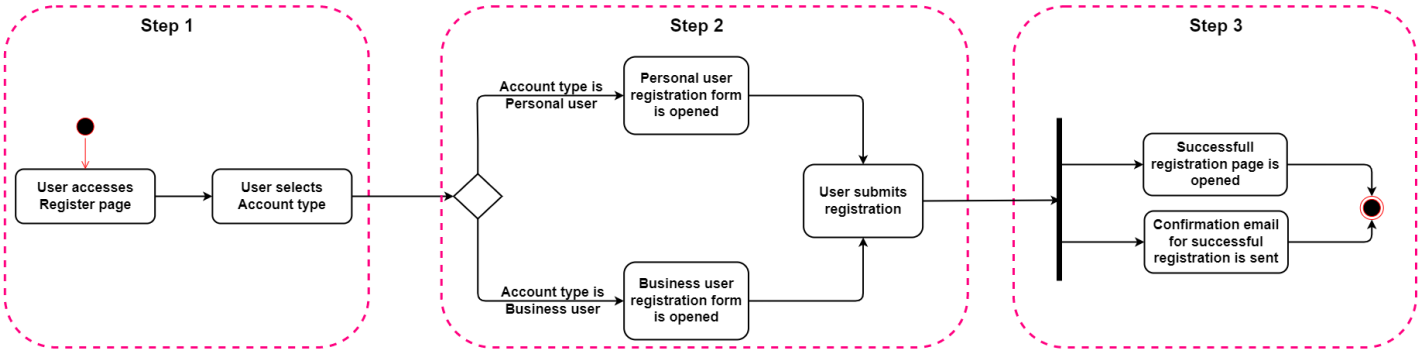
The purpose of this article is to bring up more examples about how the Casewhere workflow navigation is set up in proportion to the project business rules. We also describe other logic configurations relating to the workflow navigation in this article.

Examples

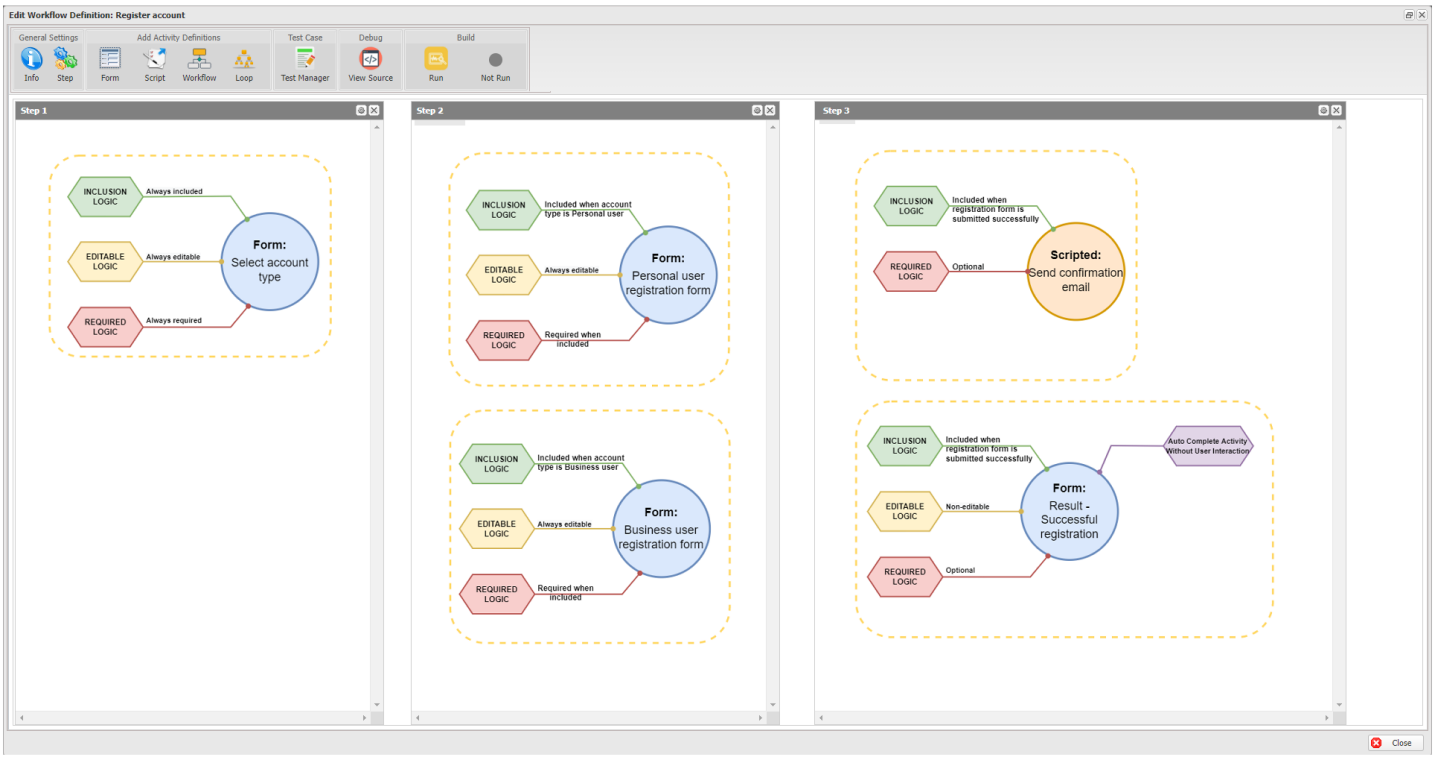
Example 1 - Register new account

The Registration flow is used to register new account by any new user. The business flow includes 3 steps: "Select account types", "Fill registration form", and "Return results" which are in proportion to 3 steps in Casewhere workflow. The flow starts when a user opens the Register form.

Diagram describing the business rule:



Workflow definition in Casewhere:



Step 1 - Select account types

First of all, a user must select the account type s/he want to register: Personal user or Business user. This flow is handled in Step 1 of Casewhere workflow with the form activity called **Select account type**. In this activity, the logic configurations are as below:

- **Inclusion logic**: At this step, the user must define which account type s/he wants to register, so the activity must be included. Then, the **Inclusion logic** here is **Always included**.
- **Editable logic**: The user can edit the account type, so the form must be editable. Then, the **Editable logic** is **Always editable**
- **Required logic**: The user must complete the form to go to the next step. Then, the **Required logic** is **Always required**

Step 2 - Fill registration form

After step 1 is completed, based on the selected account type, the corresponding form will be opened. In Step 2 of Casewhere workflow, there are 2 form activities handling this business rule: Personal user registration form, Business user registration form. The logic configurations are as below

Form: Personal user registration

- **Inclusion logic**: The form is only opened when the selected account type is **Personal user**, so the **Inclusion logic** is **Included when account type is Personal user**
- **Editable logic**: The user can edit this form, so the **Editable logic** is **Always editable**
- **Required logic**: When the form is displayed, the user must complete the form to go to the next step. Then, the **Required logic** is **Required when included**

Form: Business user registration

- **Inclusion logic**: The form is only opened when the selected account type is **Business user**, so the **Inclusion logic** is **Included when account type is Business user**
- **Editable logic**: The user can edit this form, so the **Editable logic** is **Always editable**
- **Required logic**: When the form is displayed, the user must complete the form to go to the next step. Then, the **Required logic** is **Required when included**

Step 3 - Return results

After the user submits the registration form, the result page will be opened and a confirmation email will be sent to the user. This part will be handled in Step 3 of Casewhere workflow with 1 scripted activity called **Send confirmation email** and 1 form activity called **Result-Successful registration**. The logic configurations are as below

Script: Send confirmation email

- **Inclusion logic**: The script is run to send email to the user immediately after the user submits the registration successfully. Therefore, the **Inclusion logic** is **Included when registration form is submitted successfully**.
- **Required logic**: This step is the ending step, it is not required to complete to go to next step. Therefore, the **Required logic** is **Optional**

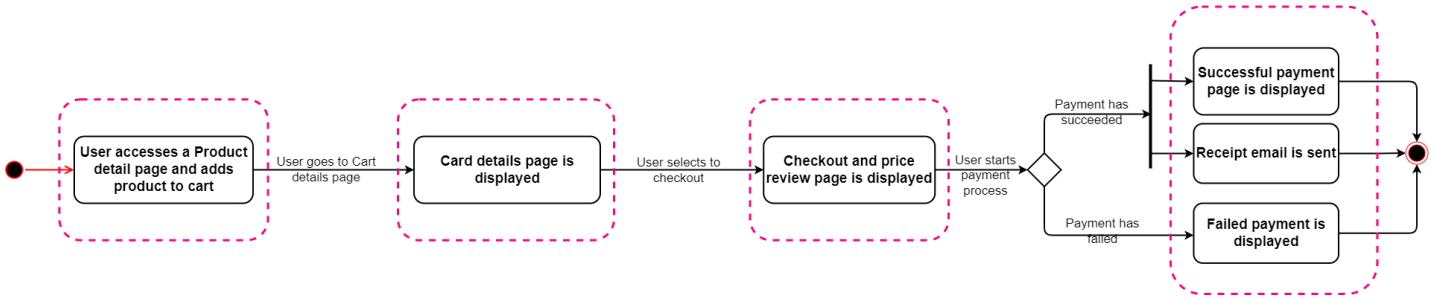
Form: Result-Successful registration

- **Inclusion logic**: The form is only opened when the registration form is submitted successfully, so the **Inclusion logic** is **Included when registration form is submitted successfully**
- **Editable logic**: The user cannot edit this form, so the **Editable logic** is **Non-editable**
- **Required logic**: When the form is displayed, it is the ending step, it is not required to complete to go to the next step. Then, the **Required logic** is **Optional**
- **Completion Logic**: When the form is displayed, it means the flow has ended, the user does not need to make any interaction to complete the flow. Therefore, the **Completion Logic** is **Auto complete activity without user interaction**

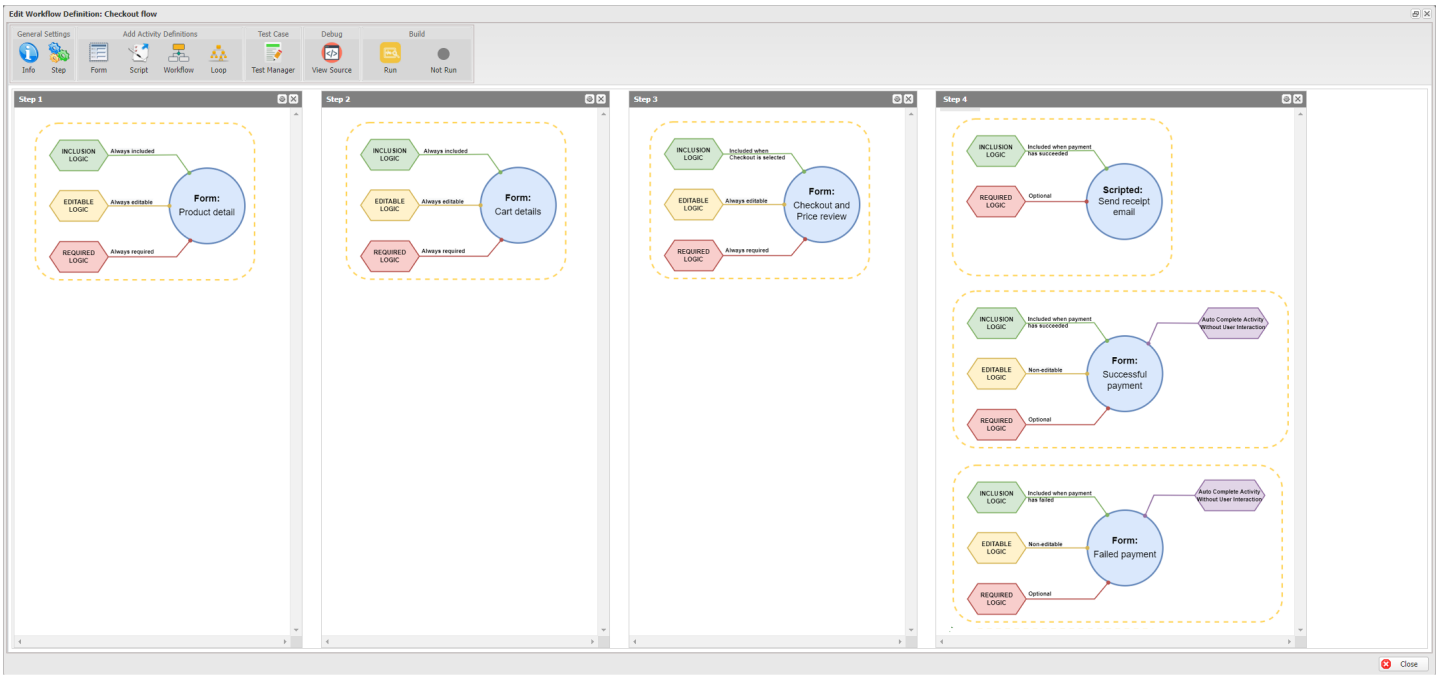
Example 2 - Add product to Cart and Check out

This flow is about the process of adding products to the cart and checkout that is often used in the e-commerce product. The business flow includes 4 steps: "Add product to cart", "Open Cart details", "Checkout", and "Returned result" that are in proportion to 4 steps in Casewhere workflow. The flow starts when a user opens the Product detail page.

Diagram describing the business rule:



Workflow definition in Casewhere:



Step 1 - Add product to cart

At first, the user must access the Product detail page to add product to cart. This step is handled in Step 1 of Casewhere workflow with the **Form activity** called **Product detail**. In this activity, the logic configurations are as below:

- **Inclusion logic**: To add product to cart and go to the next steps, the user must access the Product detail page at first. Therefore, this activity must be included, and the **Inclusion logic** is **Always included**.
- **Editable logic**: The user can edit the amount of product to add to cart, so the form must be editable. Then, the **Editable logic** is **Always editable**
- **Required logic**: The user must add product to cart before go to the step of viewing cart details and checkout. Then, the **Required logic** is **Always required**

Step 2 - Open Cart details

After products were added to cart meaning step 1 has completed, the user must go to Cart details page to continue the flow. This step is handled in Step 2 of Casewhere workflow with 1 form activity called **Cart details** . The logic configurations are as below

- **Inclusion logic**: To go to the next step of checkout, the user must access the **Cart details** page , so the **Inclusion logic** is **Always included**
- **Editable logic**: The user can remove or edit amount of product in **Cart details** page, so the **Editable logic** is **Always editable**
- **Required logic**: The user must complete the changing on Cart details page to be able to go to the Checkout step. So, the **Required logic** is **Always required**

Step 3 - Checkout:

After the user selects to checkout the products in Cart details page, the Checkout page will be opened that allows the user to review the total price and list of checkout products. This step is handled in Step 3 of Casewhere workflow with 1 form activity called **Checkout**. The logic configurations are as below

- **Inclusion logic**: This activity is only included when the user clicks on Checkout button on the Cart details page. So the **Inclusion logic** is **Included when Checkout is selected**
- **Editable logic**: The user can apply coupon or changing delivery address on this page, so the **Editable logic** is **Always editable**
- **Required logic**: This activity must be completed before moving to the payment process. So, the **Required logic** is **Always required**

Step 4 - Return results

After the step of Checkout, the payment process will be handled by a third party. The payment result will be returned from the third party and Casewhere will handle the next actions based on the payment result with 1 scripted activity called **Send receipt email** and 2 form activities called **Successful payment**, **Failed payment**. The logic configurations are as below

Script: Send receipt email

- **Inclusion logic**: The script is run to send email to the user after Casewhere receives the successful payment notification . Therefore, the **Inclusion logic** is **Included when payment has succeeded**
- **Required logic**: This step is the ending step, it is not required to complete to go to next step. Therefore, the **Required logic** is **Optional**

Form: Successful payment

- **Inclusion logic**: The form is only opened after Casewhere receives the successful payment notification, so the **Inclusion logic** is **Included when payment has succeeded**
- **Editable logic**: The user cannot edit this form, so the **Editable logic** is **Non-editable**
- **Required logic**: When the form is displayed, it is the ending step, it is not required to complete to go to the next step. Then, the **Required logic** is **Optional**
- **Completion Logic**: When the form is displayed, it means the flow has ended, the user does not need to make any interaction to complete the flow. Therefore, the **Completion Logic** is **Auto complete activity without user interaction**

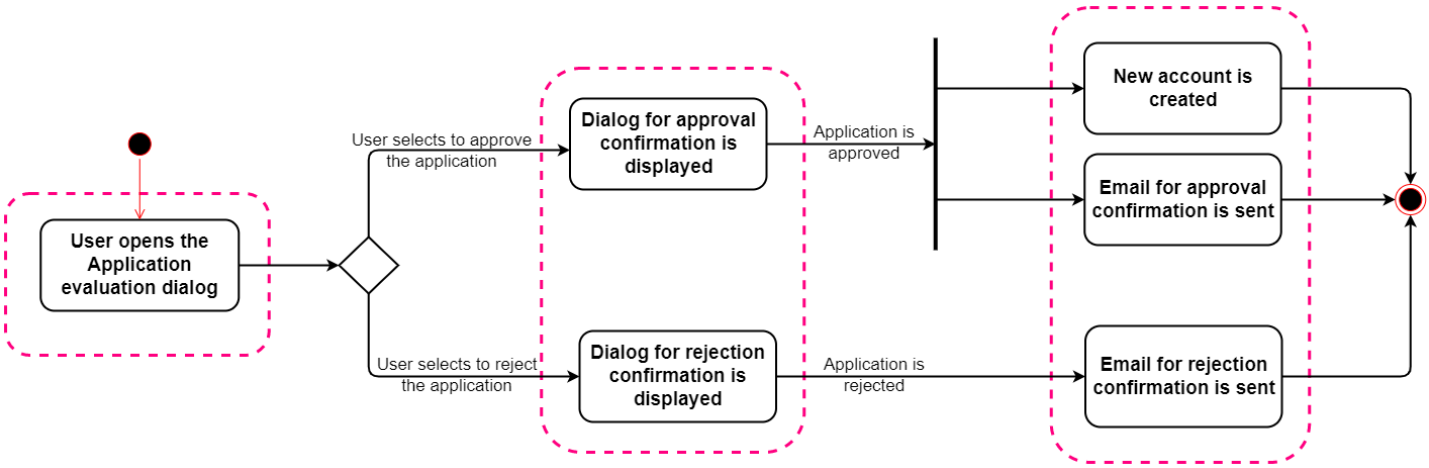
Form - Failed payment

- **Inclusion logic**: The form is only opened after Casewhere receives the failed payment notification, so the **Inclusion logic** is **Included when payment has failed**
- **Editable logic**: The user cannot edit this form, so the **Editable logic** is **Non-editable**
- **Required logic**: When the form is displayed, it is the ending step, it is not required to complete to go to the next step. Then, the **Required logic** is **Optional**
- **Completion Logic**: When the form is displayed, it means the flow has ended, the user does not need to make any interaction to complete the flow. Therefore, the **Completion Logic** is **Auto complete activity without user interaction**

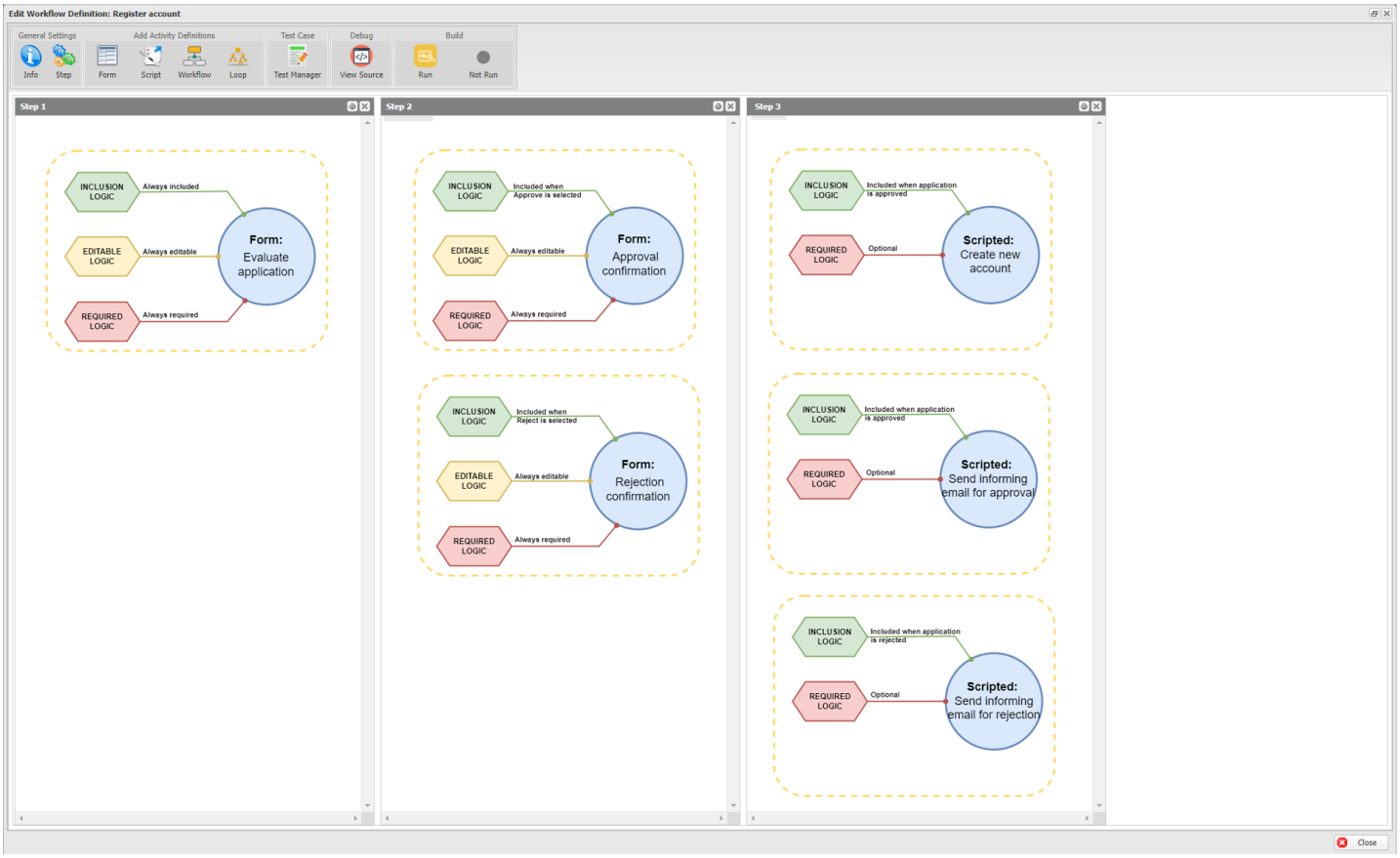
Example 3 - Evaluate registration application

This flow is about the function of evaluating the applications of the external user via the flow of Register new account. This flow is often used in the admin worker site. The business flow includes 3 steps: "Evaluate an application (Approve or Reject the application)", "Confirm the evaluation decision", "Results". The flow starts when a user opens the Application evaluation dialog.

Diagram describing the business rule:



Workflow definition in Casewhere:



Step 1 - Evaluate an application

First of all, the user must open the evaluation dialog for an application s/he wants to evaluate . This flow is handled in Step 1 of Casewhere workflow with the **Form activity** called **Evaluate application**. In this activity, the logic configurations are as below:

- **Inclusion logic**: The evaluation process is only able to start when this form is opened. Then, the **Inclusion logic** here is **Always included**.
- **Editable logic**: The user can choose to approve or reject the application. Then, the **Editable logic** is **Always editable**
- **Required logic**: The user must complete the evaluation to trigger the next step. Then, the **Required logic** is **Always required**

Step 2 - Confirm the evaluation decision

After evaluate the application, based on the decision, the corresponding confirmation dialog will be opened. If the decision is **Approve**, the Approval confirmation dialog will be opened. If the decision is **Reject**, the Rejection confirmation dialog will be

opened. In Step 2 of Casewhere workflow, there are 2 form activities handling this business rule: **Approval confirmation**, **Rejection confirmation**. The logic configurations are as below

Form: Approval confirmation

- **Inclusion logic**: The form is only opened when the user selects to approve the application, so the **Inclusion logic** is **Included when Approve is selected**
- **Editable logic**: The user can choose to agree with the confirmation or decline it, so the **Editable logic** is **Always editable**
- **Required logic**: When the form is displayed, the user must complete the form to go to the next step. Then, the **Required logic** is **Required when included**

Form: Rejection registration

- **Inclusion logic**: The form is only opened when the user selects to reject the application, so the **Inclusion logic** is **Included when Reject is selected**
- **Editable logic**: The user can choose to agree with the confirmation or decline it, so the **Editable logic** is **Always editable**
- **Required logic**: When the form is displayed, the user must complete the form to go to the next step. Then, the **Required logic** is **Required when included**

Step 3 - Results

After the user confirms the evaluation decision, the corresponding action will be executed. If the application is approved, a new account will be created and an informing email for approval will be sent to the registrant. If the application is rejected, an informing email for rejection will be sent to the registrant. This part will be handled in Step 3 of Casewhere workflow with 3 scripted activities: **Create new account, Send informing email for approval, Send informing email for rejection.** The logic configurations are as below

Script: Create new account

- **Inclusion logic**: The script is run to create new account immediately after the application is approved. Therefore, the **Inclusion logic** is **Included when application is approved**.
- **Required logic**: This step is the ending step, it is not required to complete to go to next step. Therefore, the **Required logic** is **Optional**

Script: Send informing email for approval

- **Inclusion logic**: The script is run to send email to the registrant after the application is approved. Therefore, the **Inclusion logic** is **Included when application is approved**.
- **Required logic**: This step is the ending step, it is not required to complete to go to next step. Therefore, the **Required logic** is **Optional**

Script: Send informing email for rejection

- **Inclusion logic**: The script is run to send email to the registrant after the application is rejected. Therefore, the **Inclusion logic** is **Included when application is rejected**.
- **Required logic**: When the form is displayed, it is the ending step, it is not required to complete to go to the next step. Then, the **Required logic** is **Optional**

Workflow engine

This article explains the execution of Casewhere workflows, how they are triggered, the workflow states' meaning, how they are transitioned, and more importantly, how workflows work with processes, cases and data objects.

Workflow triggers

There are many ways a workflow can be triggered in Casewhere. Here we only briefly introduce the workflow triggers. There are separate articles explaining how they work in detail and how to configure them.

- UI trigger: End-users can trigger workflows by clicking widget buttons. This is the only way to trigger a UI workflow while it is totally possible to trigger an automatic workflow here.
- Data trigger: You can configure a rule so that a change in database can trigger an automatic workflow.
- Time trigger: You can configure a scheduler to trigger an automatic workflow at a specific time.
- Event trigger: You can configure Casewhere to listen to a specific event and trigger an automatic workflow.
- Web trigger: You can expose a Web API endpoint for external systems to trigger a workflow in Casewhere.

Workflow states

A workflow has four states:

- `Open`: When the workflow starts, it sets the workflow's state to `Open`.
- `Completed`: A workflow completes when all required activities are completed.
- `Locked`: When a workflow is locked, it cannot be run again. Locking workflows have been deprecated since version 2.5.
- `Queued`: Casewhere can defer the workflow's execution by putting them in queues. Queueing is an important technique to ensure that Casewhere can handle many workflows without being overloaded. When a workflow is placed in a queue successfully, it has the state `Queued`.

The diagram below explains how the state transition works:

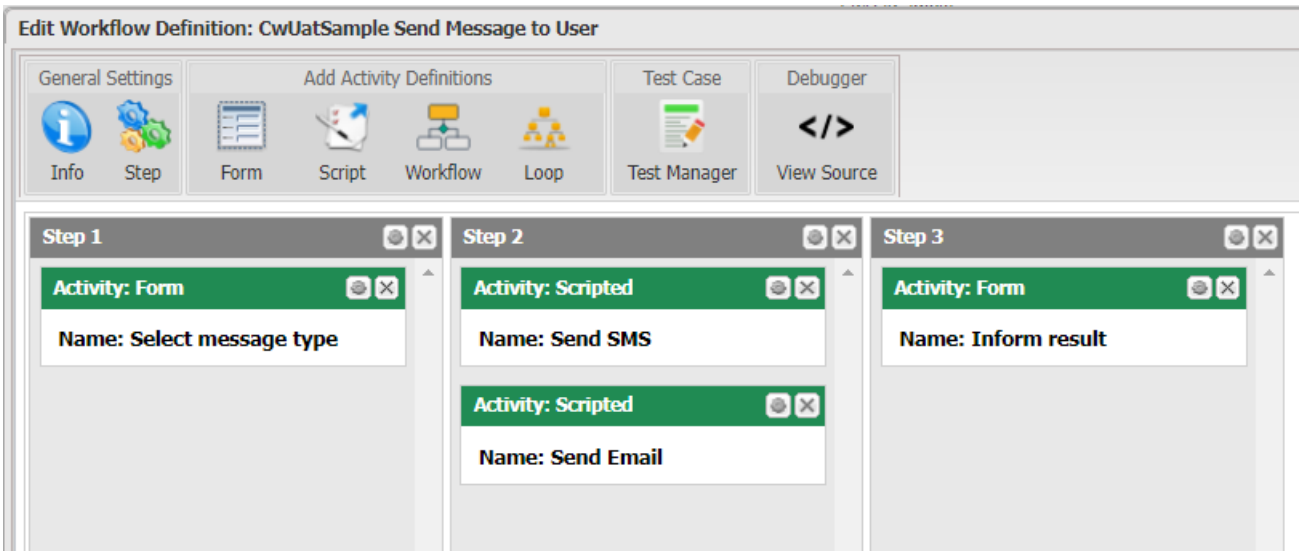
```
stateDiagram-v2
    [*] --> Open: Engine starts
    [*] --> Queued: Place in queue
    Queued --> Open: Engine starts
    Open --> Completed: If all required activities completed
    Completed --> [*]
    Open --> [*]:  If any required activity incompleted
```

Workflow execution paths

A workflow can have many execution paths that are activated depending on the current data and user input. The workflow engine will evaluate the activity navigation rule to decide which execution path to take next. This happens every time the workflow engine executes an activity.

To help you understand better, let's set up a simple workflow for sending message as below:

1. Form activity: Select message type
2. Script activity: Send SMS
    1. Included When: `message.Type == "SMS"`
3. Script activity: Send email
    1. Included When: `message.Type == "Email"`
4. Form activity: Inform result
    1. Included When: `message.IsSent == true`



In one scenario, the workflow can execute as below:

1. After triggered, the workflow stops at the first activity and waits for user input. The current execution path is `Select message type`, and it's the only activity you can see in the workflow wizard.
2. The user selects the message type `Email` and clicks **Save**.
3. The workflow navigates to the scripted activity `Send SMS`. Since the user selected `Email` in the previous activity, the inclusion rule fails, so Casewhere will skip the activity.
4. The workflow navigates to the scripted activity `Send Email`. The inclusion rule passes since the user selected `Email`, so Casewhere will execute the activity and set `IsSent` to `true`. At this point, the execution path is `Select message type` `->` `Send Email`.
5. The workflow navigates to the scripted activity `Send Email`. Since the message is sent, the inclusion rule passes, Casewhere will execute the activity. At this point, the execution path is `Select message type` `->` `Send Email` `->` `Inform result`, and the user will see the form `Inform result` appear as the last step of the workflow wizard.

In another scenario where the user selects `SMS`, the execution path will be `Select message type` `->` `Send SMS` `->` `Inform result`.

Working with cases and data objects

When setting up a workflow definition, you must specify whether it will be "Case creating" or "Case editing" via the setting **Used For Creating New Cases**.

Casewhere always requires a case context when triggering new workflows. If a workflow is "Case creating", Casewhere will create a `Draft` case when it starts the workflow. Casewhere will place all data objects created during the workflow execution in that case. These data objects will have the state `Draft` until the workflow completes. If the workflow fails to complete, the case and all data will remain to be `Draft` and thus are inaccessible from other workflows.

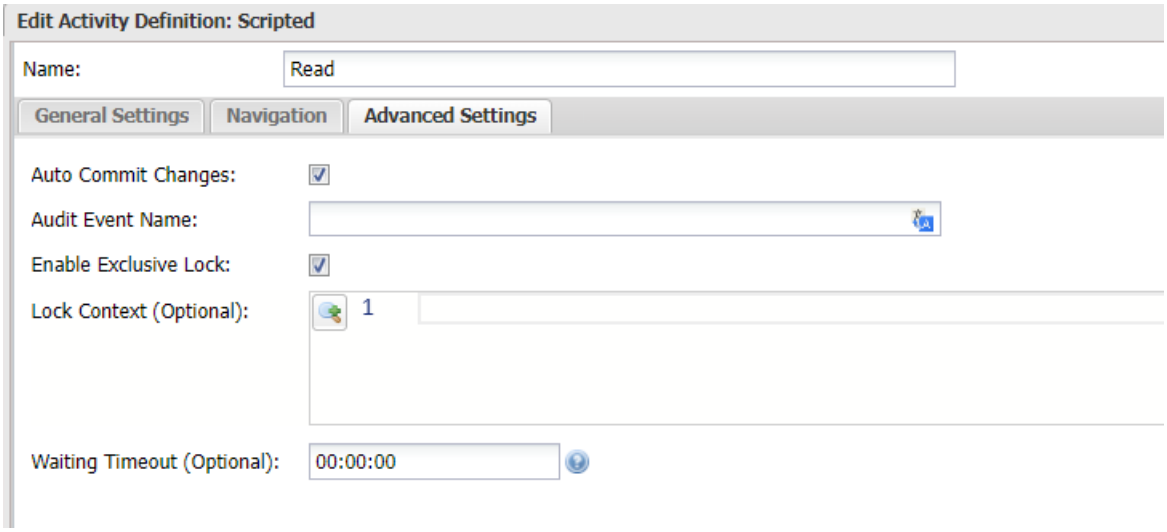
Workflow concurrency control

> This feature is available in version `2.7.2` or later

A race condition in Casewhere occurs when two or more workflows share the same context (e.g., access shared data objects) and try to change the context state simultaneously. Having race conditions for critical business scenarios such as payment, booking, etc., is unacceptable. For instance, it must not be possible for two users to purchase the same product, or an invoice should not be paid more than once. Understand the challenge of dealing with race conditions, Casewhere provides a feature to control concurrency that works out of the box for developers.

Configuration

Casewhere allows you to control concurrency at activity level using exclusive lock. When editing an activity, you can navigate to **Advanced Settings** to turn on the setting **Enable Exclusive Lock**.



Other settings:

- **Lock Context (optional)**: When the exclusive lock is enabled, Casewhere will only allow one activity to execute at a time, in the entire system. This is obviously not optimized for many scenarios. By providing a lock context, you can control the concurrency logic on a more granular level. For example, you can enable the lock on specific data objects to execute more activities as long as they're on different objects. The setting accept an expression using the same syntax with navigation rule. You can learn more about the rule syntax here.
- **Waiting Timeout**: Casewhere will release the lock if it lasts longer than the configured timeout, even when the activity is not finished.

Example

Imagine we are implementing a simple feature to count application message readers. The data model looks like below:

Edit Data Class: CwUatTx2Message

Name: CwUatTx2Message

Change Name

Tag:

Attributes | Indexes | Audit Log | Free-text Search | Security | General | Dependencies

Up | Down

#	Name	Title	Type	Description	Symbols	Required
1	Id	id	String			☑
2	CreatedAt	created_at	DateTime			☑
3	CreatedBy	created_by	String			☑
4	ModifiedAt	modified_at	DateTime			☐
5	ModifiedBy	modified_by	String			☐
6	IsDeleted	is_deleted	Boolean			☐
7	ActivityId	activity_id	String			☑
8	WorkflowId	workflow_id	String			☑
9	CaseId	case_id	String			☑
10	ApplicationId	ApplicationId	Foreign Key			☐
11	Content	Content	String			☐
12	Status	Status	Enum			☐
13	Read	Read	Integer	**Message read counter**		☐
14						☐

A message is accessible to many users, and every time someone reads a message, Casewhere will increase the counter as below:

Edit Activity Definition: Scripted

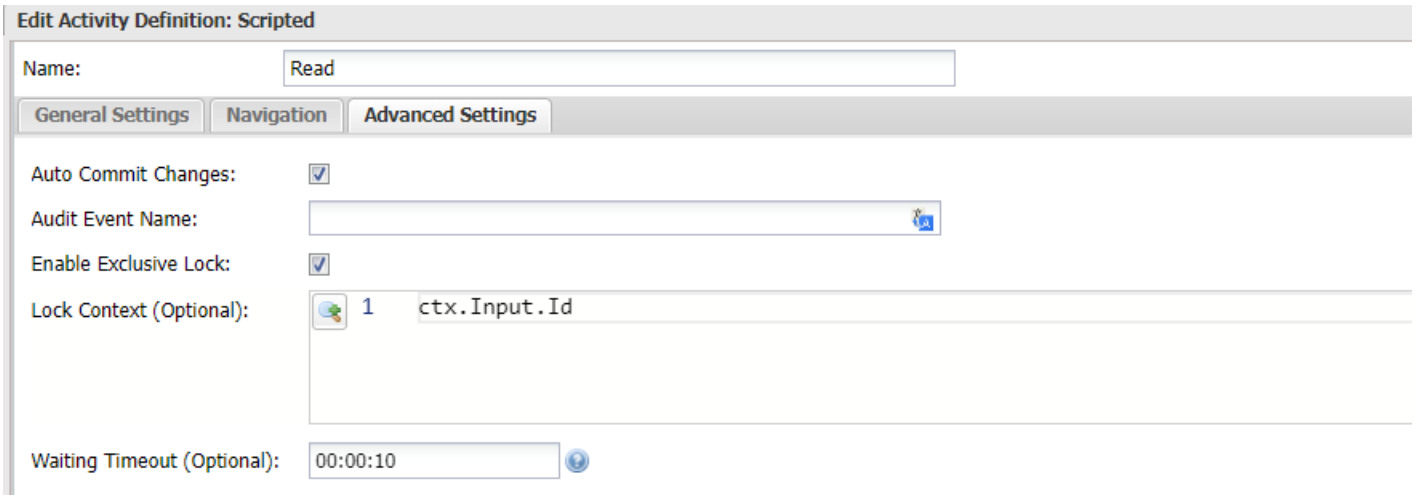
Name: * Read

General Settings | Navigation | Advanced Settings

```
1  var dataApi = ctx.Use<IDataApi>();
2
3  // We must load data from DB because ctx.Input is fetched before this script runs
4  var filter = FilterBuilder.Create().Eq("_id", ctx.Input.Id).Build();
5  var result = dataApi.Search(DataObjectApiQuery
6      .For("CwUatTx2Message")
7      .FilterBy(filter));
8
9  if (result.Data.Count > 0)
10 {
11     var read = result.Data[0].Has("Read") ? result.Data[0].Get<int>("Read") : 0;
12     read = read + 1;
13
14     // We expect message.Read to be equal to the total number of receipt objects
15     dataApi.Update(ctx.Input.Id, new { Read = read }); // Increase the counter for message
16     dataApi.Add("CwUatTx2MessageReceipt", new { MessageId = ctx.Input.Id }); // Create a receipt everytime someone reads a message
17 }
```

The feature will not count correctly if many people are reading the same message at the same time. The value `Read` will always be lower than the number of readers in reality due to the race condition, i.e., the counter is modified after being loaded by another workflow.

By enabling the exclusive lock on the **Read** activity definition, the feature will work as expected because there is only one activity is executed at a time. However, that also means the system only allow one person to read message at a time, even when they are reading different messages. To improve the user experience here, you can configure the **Lock Context** as below:



By configuring the **Lock Context** as described above, multiple users can read messages simultaneously, provided they are accessing different messages. For this setup to function correctly, ensure the **Auto Commit Changes** setting is enabled. This allows the executing activity to read the latest value of the `Read` counter, which was saved by previously executed activities.

Workflow transaction

> This feature is available in version `2.7.2` or later

Casewhere provides the transaction concept to represent a unit of work, which is made up of multiple operations, performed and treated in a reliable and coherent way. A Casewhere workflow transaction, by definition, must be

- Atomic: It must either be complete in its entirety or have no effect whatsoever
- Consistent: It must conform to existing constraints in Casewhere
- Isolated: It must not affect other transactions
- Durable: It must get written to persistent storage

Transactional activity

Form activity and Scripted activity are transactional activities in Casewhere. Casewhere will reject all changes created during the activity execution if these activities fail to execute for some reason.

Example 1: All changes are rejected if the scripted activity throws an exception

```
dataApi.Add("Message", new { Status = "Draft", Content = "Message 1" });
dataApi.Add("Message", new { Status = "Draft", Content = "Message 2" });
throw new Exception("Error");
```

In this example, both `Message 1` and `Message 2` will not be persisted.

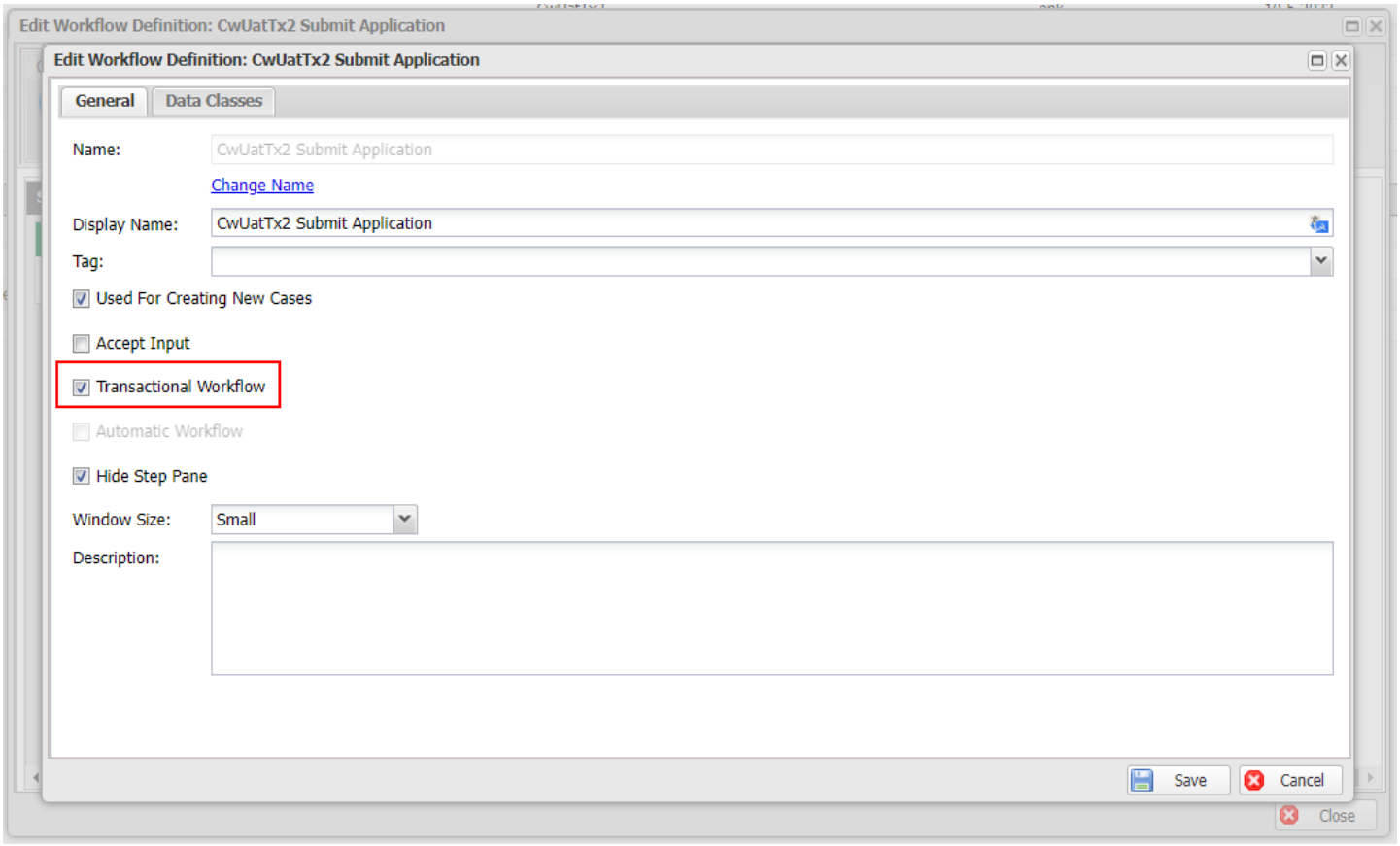
Example 2: Commit transaction

```
dataApi.Add("Message", new { Status = "Draft", Content = "Message 1" });
ctx.CommitChanges();
dataApi.Add("Message", new { Status = "Draft", Content = "Message 2" });
throw new Exception("Error");
```

In this example, `Message 1` is persisted while `Message 2` is not.

Transactional workflow

Casewhere allows to enable transaction for a specific workflow definition. For **Automatic Workflow**, transaction is enabled by default.



Every time Casewhere persists the workflow state, it will commit all opening transactions and flush all changes to database. These changes once committed will not be rolled back even if the workflow fails to execute the next activities. When you enable transaction for a UI workflow, you will see Casewhere creates a new transaction every time the user click "Save" in the workflow wizard. For automatic workflows where no user interaction is required, Casewhere will run all activities in the same transaction.

When a transaction rolls back, besides the changes in data objects, Casewhere will also discard the following information tracked by the transaction:

- Workflow queue jobs
- Data trigger notifications
- Event trigger events
- Audit logs

In order to understand how transactions work for Casewhere workflows, let's go through the following examples.

Example 1: Error is thrown from a required activity

Assuming we have the following workflow definition:

- `Workflow 1 (Transaction enabled)`
  - `Scripted Activity 1`
    - The activity saves data successfully
  - `Scripted Activity 2 (Required)`
    - The activity throws an error

Since the `Scripted Activity 2` is required, the `Workflow 1` will stop. Since the workflow is transactional, Casewhere will discard all data changes including the ones in `Scripted Activity 1`.

Example 2: Error is thrown from an optional activity

Assuming we have the following workflow definition:

- `Workflow 1 (Transaction enabled)`
  - `Scripted Activity 1`
    - The activity saves data successfully
  - `Scripted Activity 2 (Optional)`
    - The activity throws an error
  - `Scripted Activity 3`
    - The activity saves data successfully

Since the `Scripted Activity 2` is optional, the `Workflow 1` will run to the end. Though the workflow is transactional, Casewhere still persists on changes in `Scripted Activity 1` and `Scripted Activity 3`, only discard the change if any in `Scripted Activity 2`.

Example 3: Error is thrown from nested workflow

Assuming we have the following workflow definition:

- `Workflow 1 (Transaction disabled)`
  - `Scripted Activity 1`
    - The activity saves data successfully
  - `Workflow Activity 2 (Required, Executed immediately)`
    - `Workflow 2 (Transaction enabled)`
      - `Scripted Activity 2.1 (Required)`
        - The activity throws an error
  - `Scripted Activity 3`

The `Workflow 1` will stop at `Workflow Activity 2` because the activity is required. Casewhere will discard all changes in `Workflow 2` but still accepts the changes in `Scripted Activity 1` because `Workflow 1` is not transactional.

Example 4: Error is thrown from a synchronous triggered workflow

Assuming we have the following workflow definition:

- `Workflow 1 (Transaction enabled)`
  - `Scripted Activity 1 (Required)`
    - Trigger a synchronous workflow `Workflow 2` (through data trigger or event trigger)
      - `Workflow 2` executes and throws an error
  - `Scripted Activity 2`

The `Scripted Activity 1` will fail to complete because of the error thrown from `Workflow 2` . The `Workflow 1` will stop at `Scripted Activity 1` because the activity is required. Casewhere will discard all changes in `Workflow 1` and `Workflow 2` because they share the same transaction context.

UI trigger

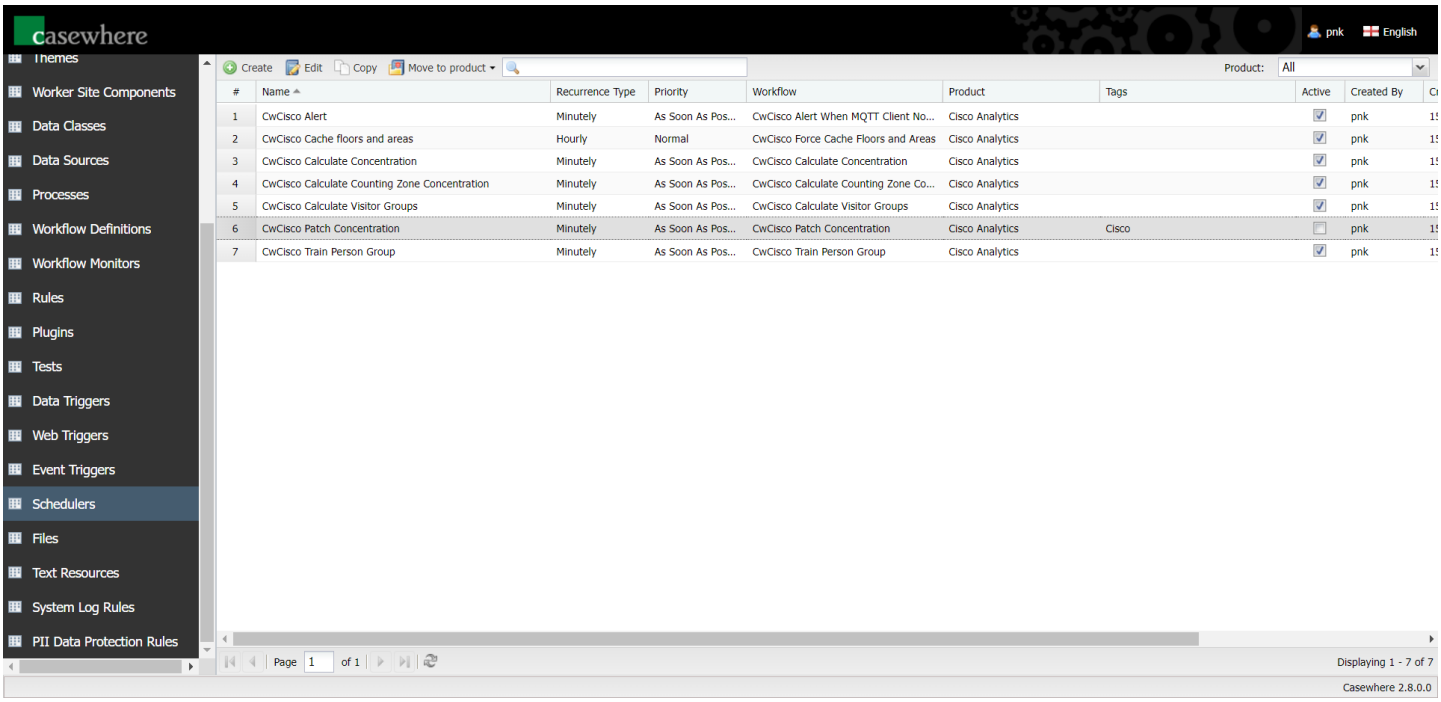
Coming soon...

Scheduler - Time trigger

Introduction

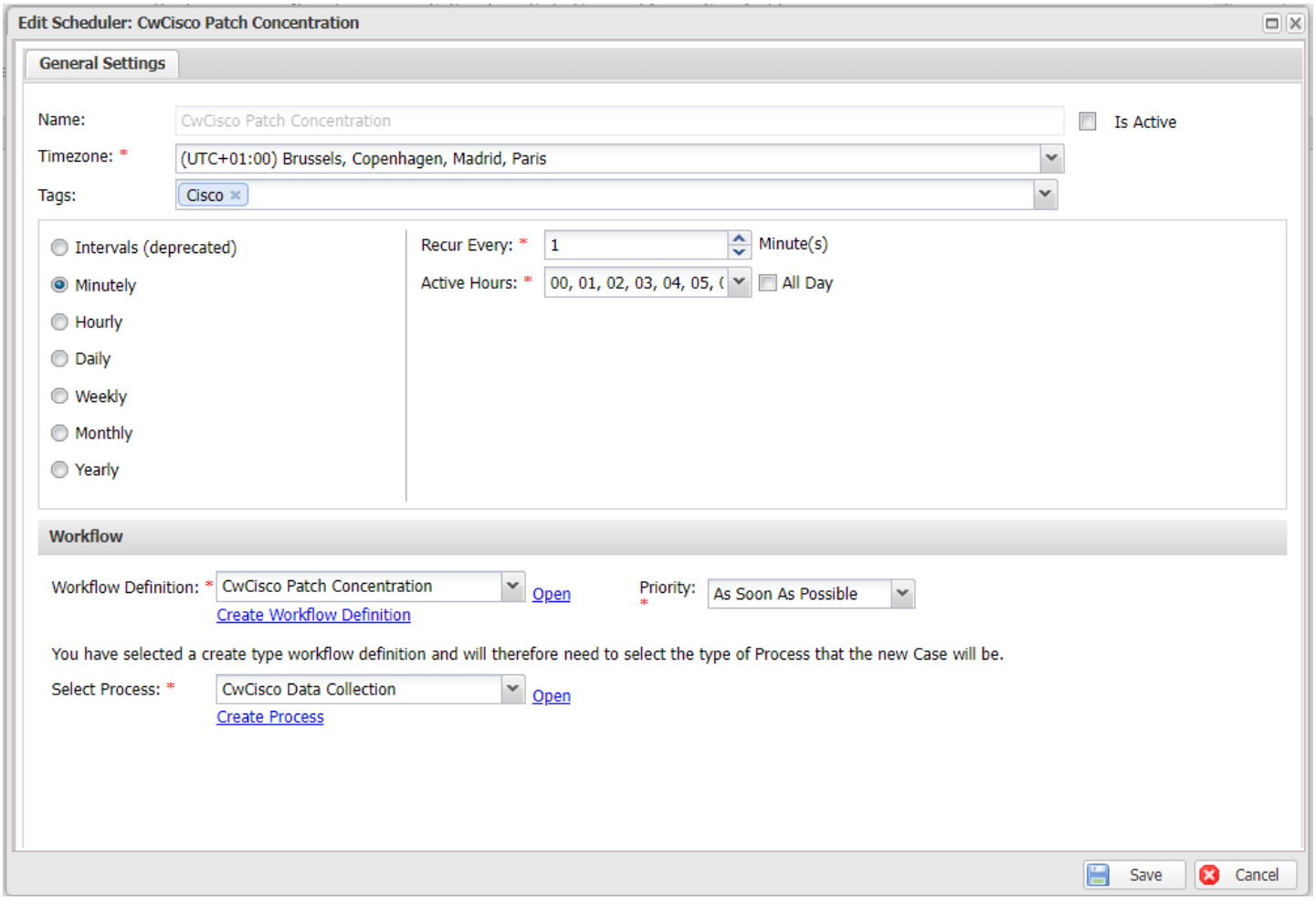
Casewhere scheduler is your best solution for handling automated, recurring tasks. You can configure to trigger a workflow at a particular time or interval. It's also possible to specify the time zone in which your scheduler will run.

In Casewhere Admin, navigate to **Schedulers**. From here, you can create, edit and manage schedulers like any product resource.



Configuration

To set up a scheduler, you need the following information:



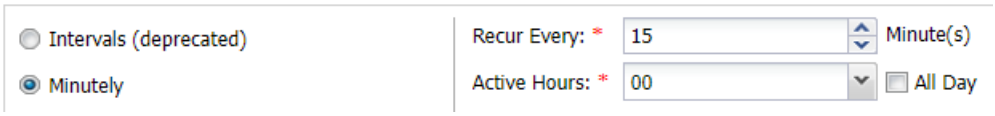
Name: Scheduler name must be required and unique.

Is Active: If a scheduler is inactive, it will not run.

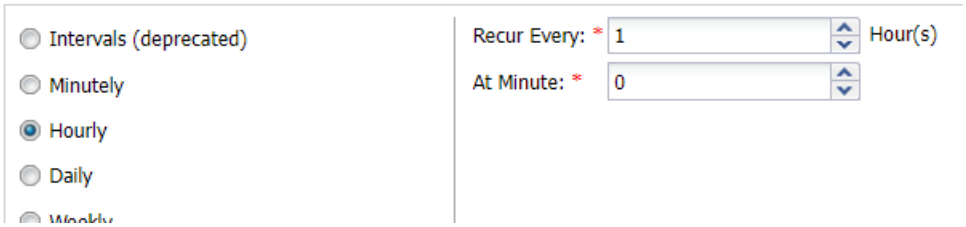
Timezone: The time zone in which the schedulers will run.

Schedule: Casewhere provides the following options to schedule workflows:

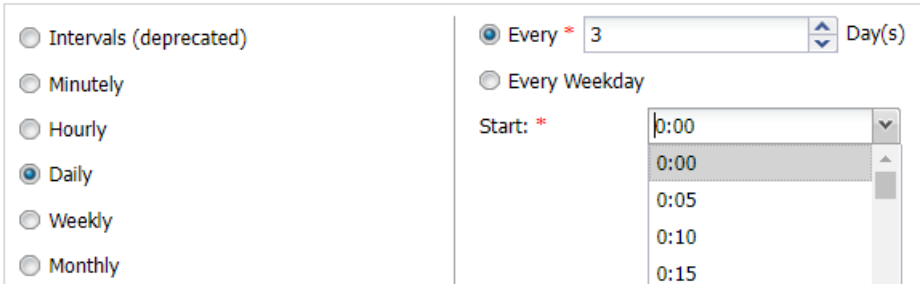
Minutely: Casewhere will execute a workflow every *xx* minutes, where *xx* ranges from 0 to 59. You can also choose to enable the scheduler in some specific hours or have it runs all day. For example, if you configure **Recur Every: 15 Minutes** and **Active Hours: 00:00**, the trigger will be on from 00:00 to 01:00. During this time, the workflow will run once every 15 minutes.



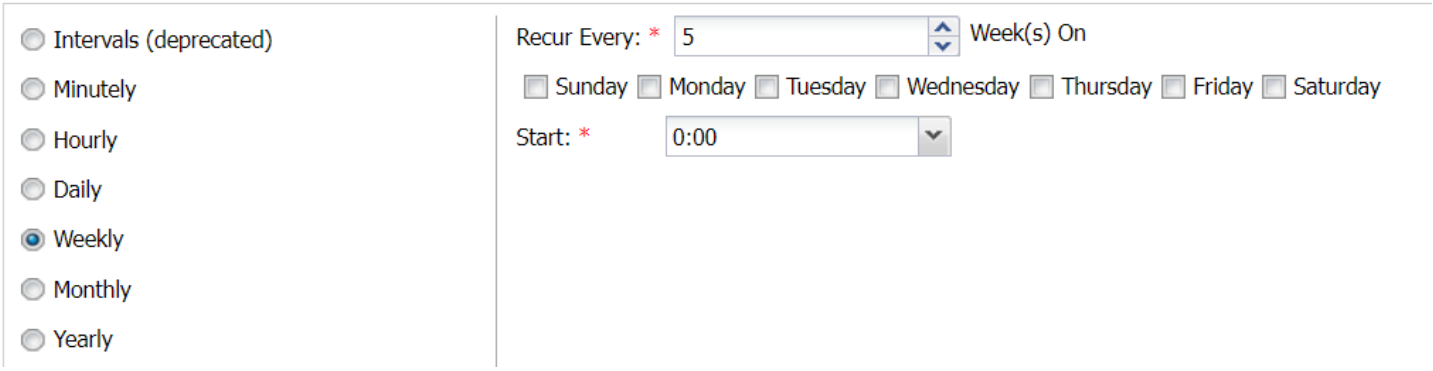
Hourly: Casewhere will execute a workflow every *xx* hours, where *xx* ranges from 0 to 23. You also specify the minute (0-59) the workflow will run.



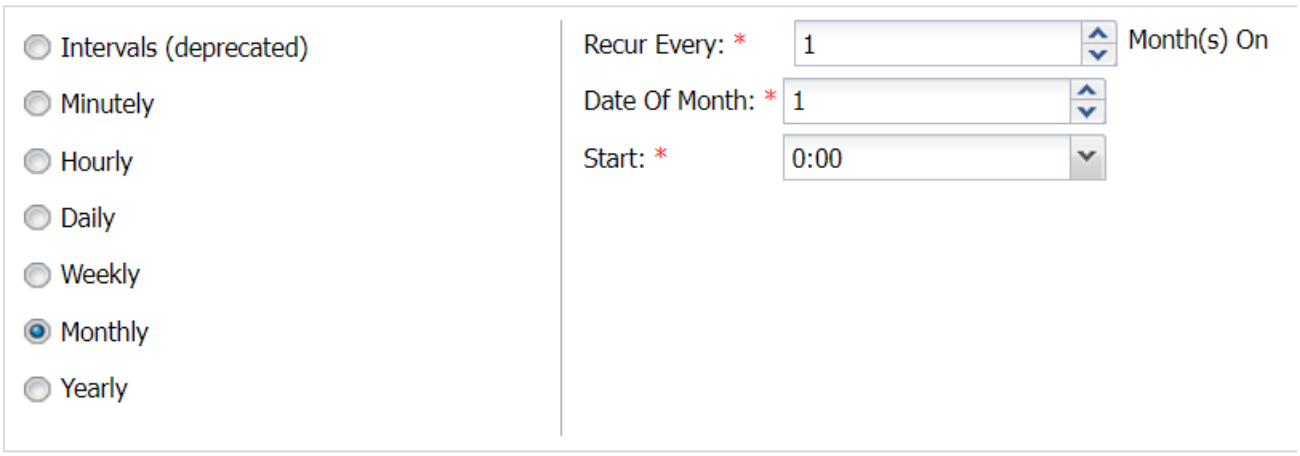
Daily: Casewhere will execute a workflow every *xx* days, where xx ranges from 1 to 31. You can choose to only execute workflows during weekdays and start at a specific time during a day.



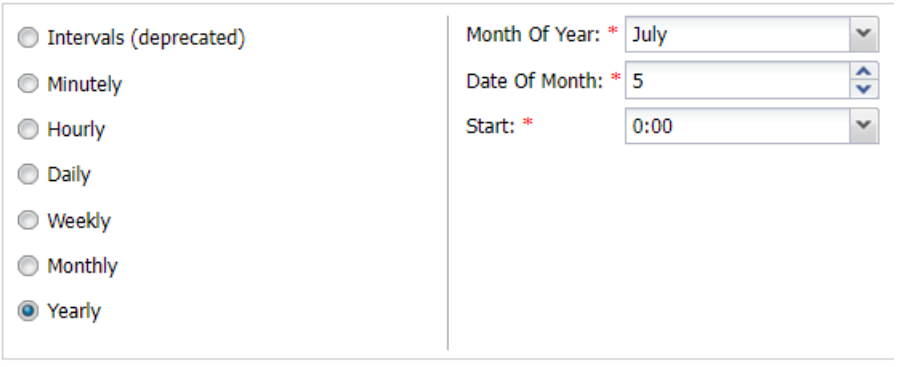
Weekly: Casewhere will execute a workflow after certain weeks. You can specify which days and times during the day to execute workflows.



Monthly: Casewhere will execute a workflow after certain months. You can specify which days and times during the month to execute workflows.



Yearly: Casewhere will execute a workflow every year at a specific date and time.



Workflow definition: Casewhere only accepts automatic workflow for using with schedulers.

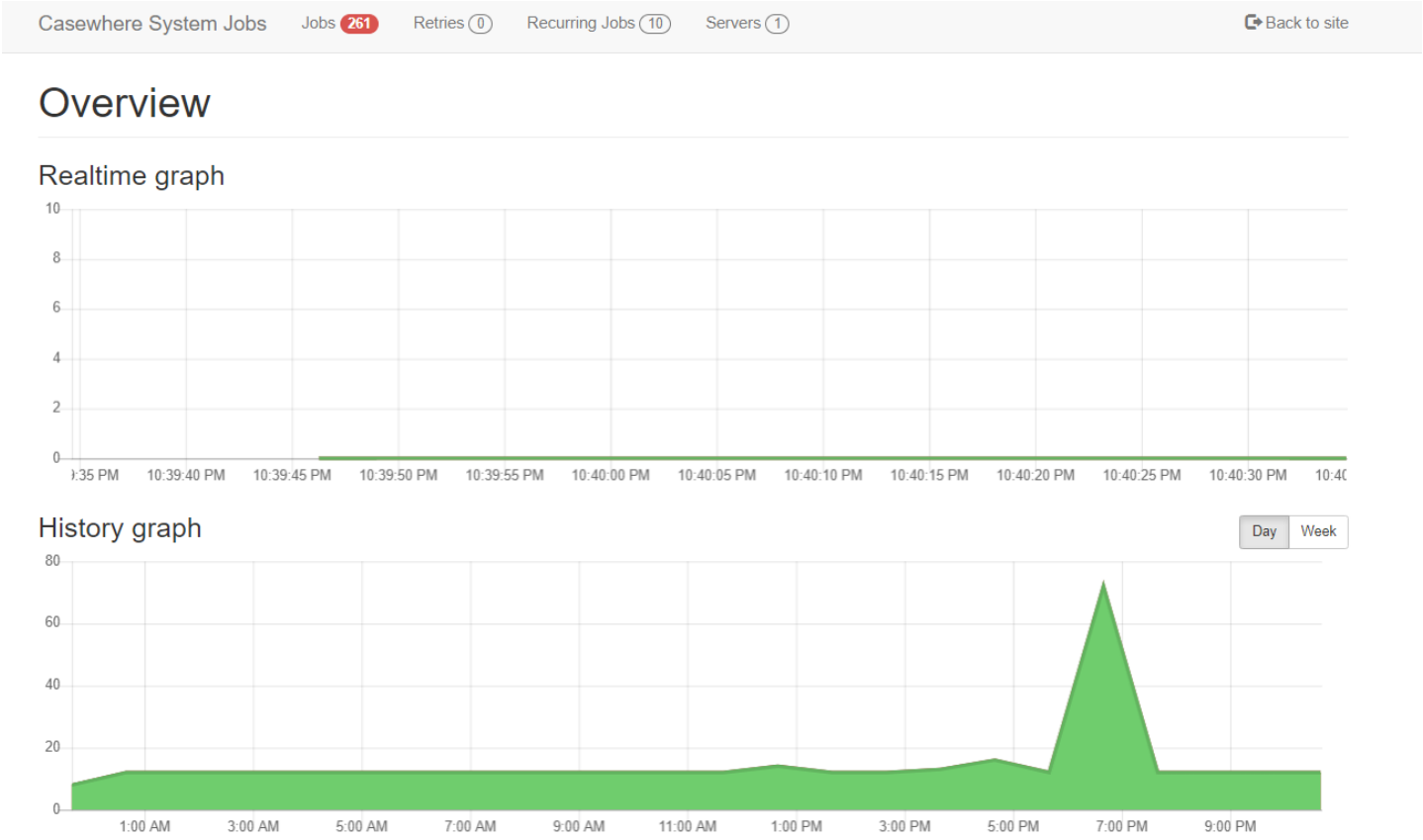
Priority: There are many workflows that can run in background at the same time, so it is important to tell Casewhere which one should run before the others. Priority order: Urgent > As Soon As Possible > Normal > Low.

Process: The process in which the triggered workflow will run.

Trigger rule: Rules only apply to edit-case workflows.

How schedulers work

Behind the scene, Casewhere relies on Hangfire to manage scheduled jobs. Every active scheduler will have a recurring job created in Hangfire. You can visit the Hangfire Dashboard at https://your-deployment-domain/hangfire.



At the scheduled time, Casewhere will create a workflow and queue it. The workflows are then executed based on their priorities.

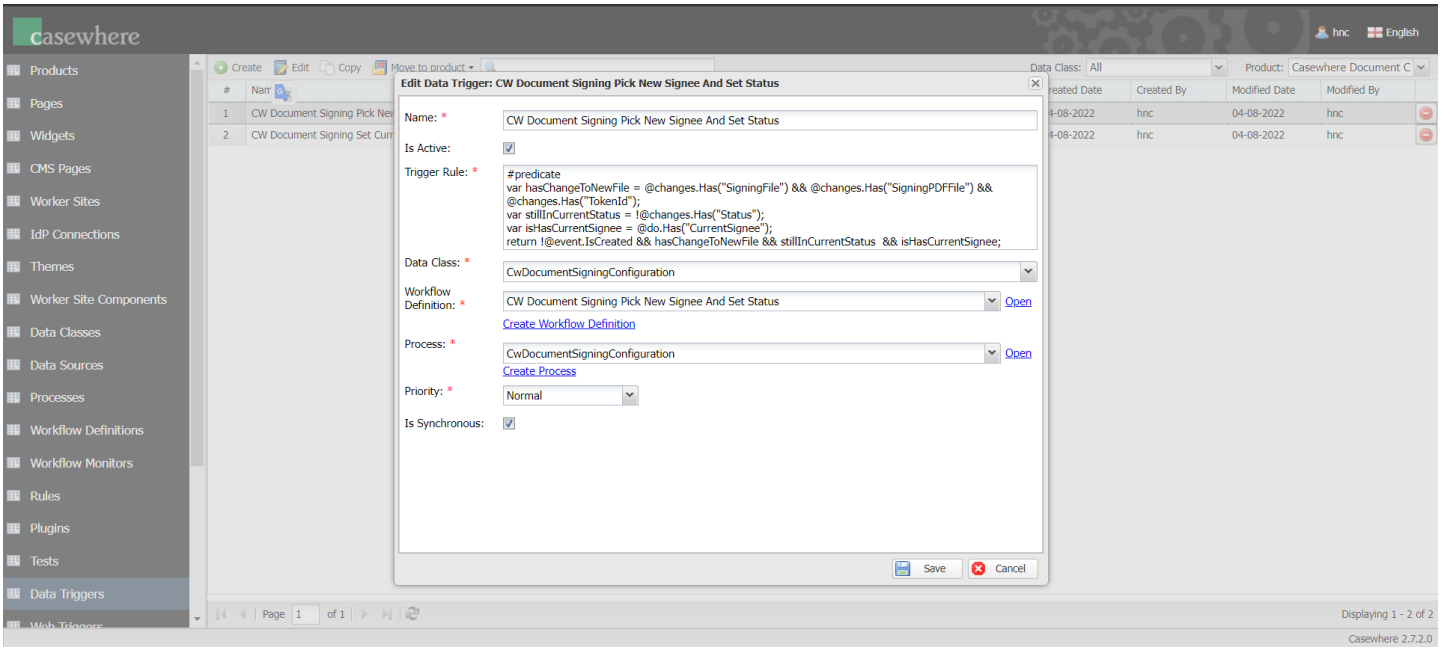
Data trigger

Introduction

There are multiple ways to trigger a workflow in Casewhere. One is **data trigger**, which allows Casewhere to execute a workflow when data changes.

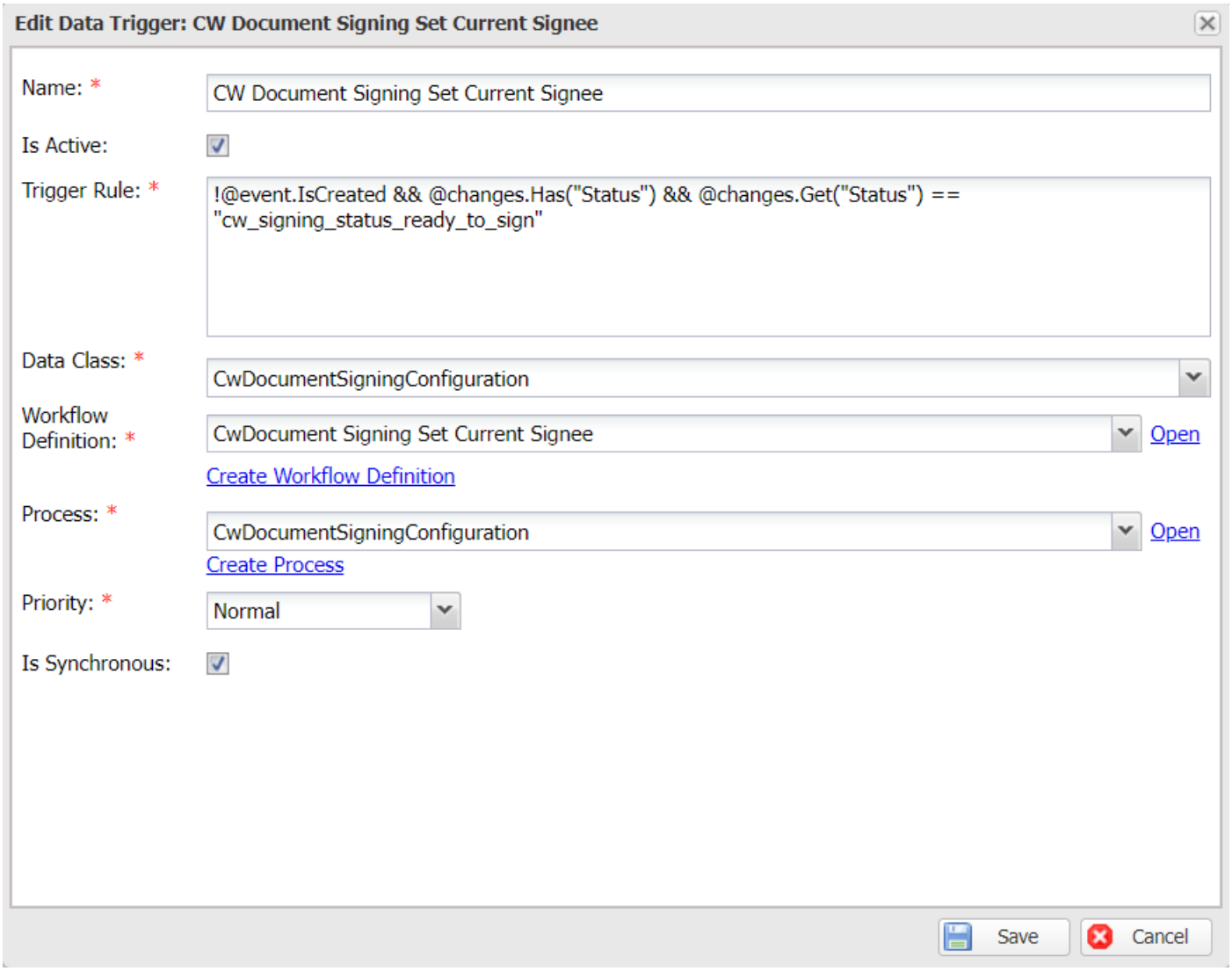
Imagine your solution must send an email every time a user is added. There are many ways to add a user, and repeating the sending email logic in all user creation workflows is never a good approach. A better one is to create a data trigger that listens to the user creation event and triggers a workflow to send an email.

In Casewhere Admin, navigate to **Data Triggers**. From here, you can create, edit and manage data triggers like any product resource.

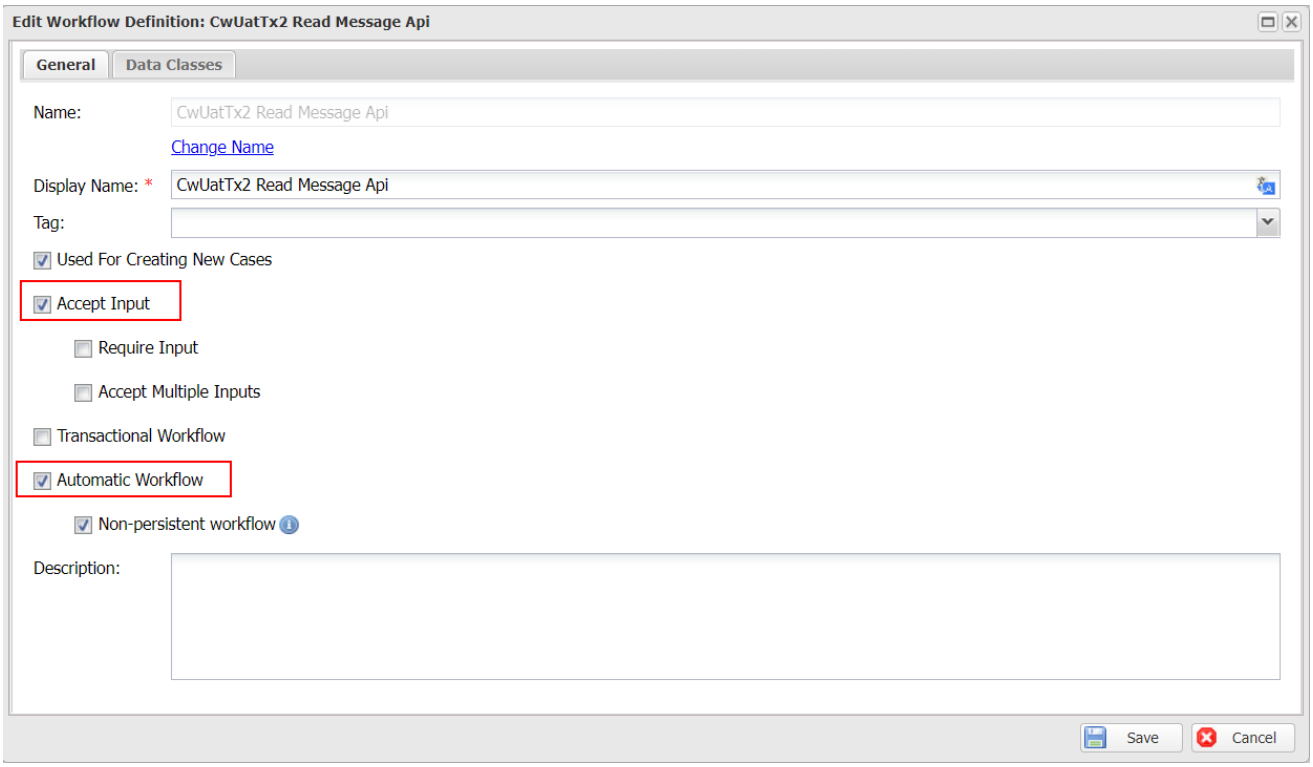


Configuration

The following information is required to configure a data trigger:



- **Name**: Name must be required and unique.
- **Trigger Rule**: Rule to determine whether a new workflow should be triggered upon the creation, modification, or deletion of a data object. Here you can learn more about it.
- **Data Class**: The data class the trigger runs on.
- **Is Synchronous**: When checked, the data trigger, including the rule evaluation and workflow execution will run immediately in the same context of the workflow emitting the change event. Otherwise, the data trigger will be queued and executed in the background.
- **Workflow Definition**: The triggered workflow definition will be run when the data object meets the trigger rules. Casewhere only accepts automatic workflow definitions for using with data triggers. The workflow definition must also accept input in order to access the changed data object.



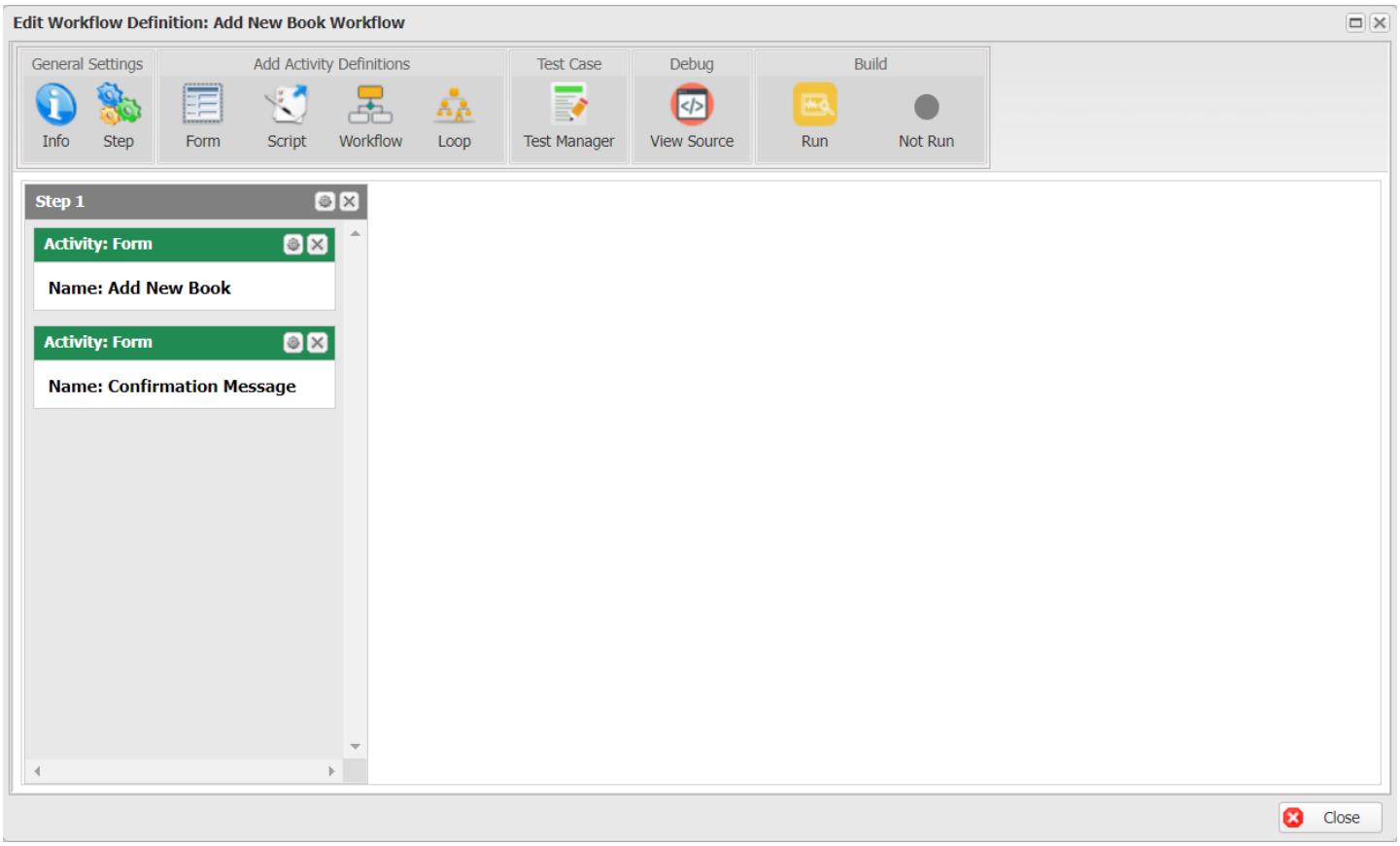
- **Process**: The process Casewhere executes the data trigger. If the workflow definition is case creation, a new case will be created.
- **Is Active**: Based on this value, a trigger can run or not run. If you don't want a trigger to run, uncheck this box.
- **Priority**: Triggers will run in order, so you should set their priority. Priority order: *Urgent* > *As Soon As Possible* > *Normal* > *Low*.

How it works

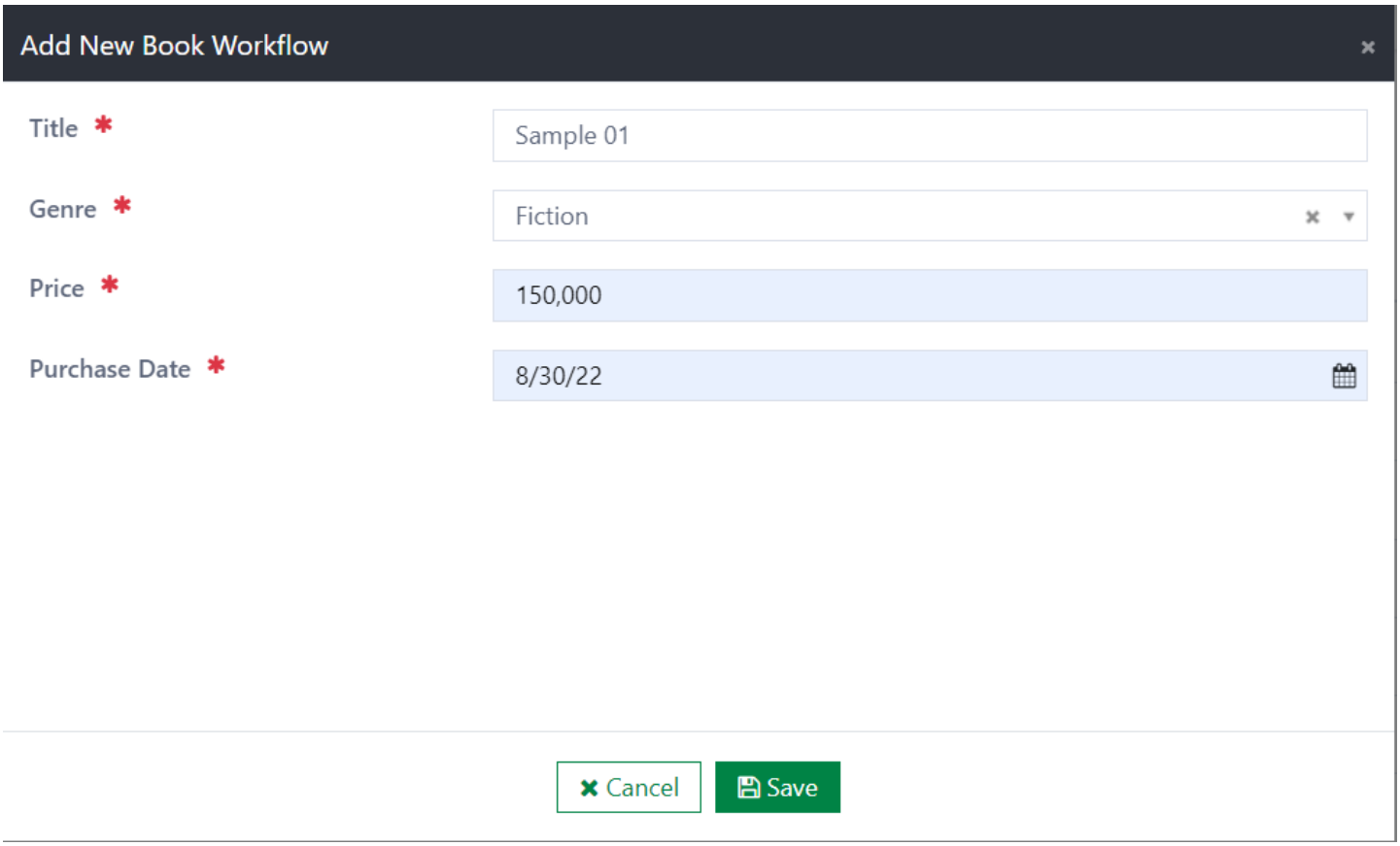
When a data object changes, Casewhere will examine all active data triggers associated with that data class and execute them in priority order. The workflow can be executed if the data object satisfies the trigger rules. Depending on the configuration, it can be run immediately or in the background.

Let's have a look at an example below:

In the workflow "Add New Book Workflow", we have 2 activities: "Add New Book" and "Confirmation Message".

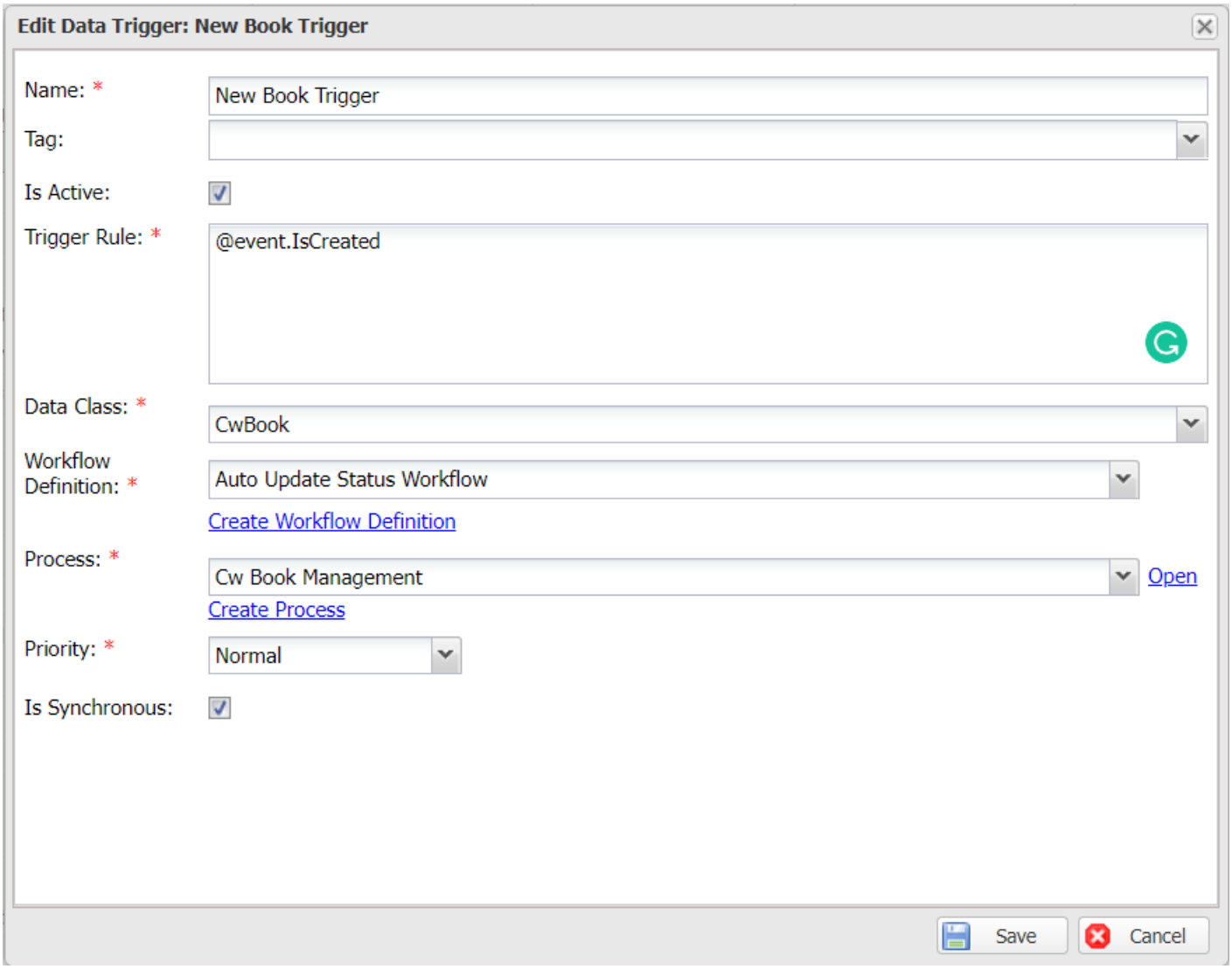


"Add New Book" will show a dialog, in which a user can fill out the information for a new book.



"Confirmation Message" will show a message to confirm that adding is complete.

A trigger "New Book Trigger" will execute a workflow to update "status".



When a new book is added, the trigger will run. It executes the workflow to update the status "available".

If the "Is Synchronous" setting is checked, the trigger immediately updates the status and allows a user to view it on the form.



Otherwise, the workflow must be delayed and run in the background. Therefore, the status just shows the default value "unavailable".



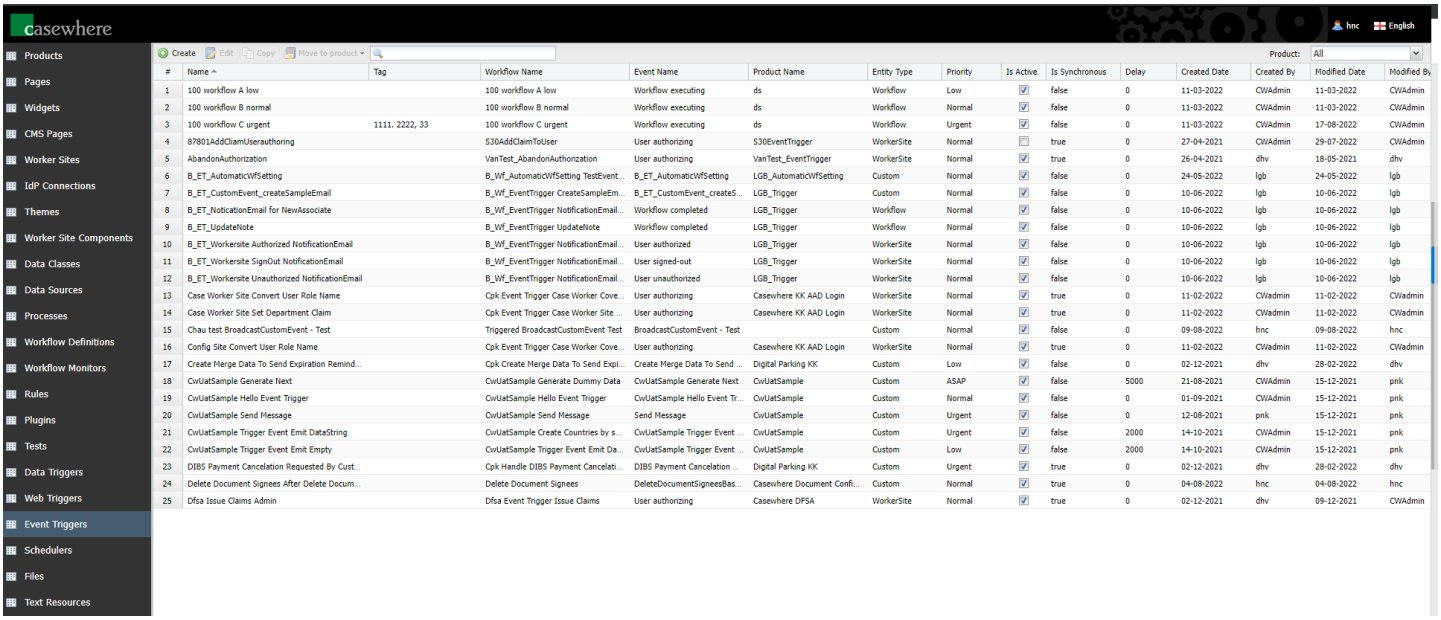
Event trigger

Introduction

The **Event Trigger** feature enables applications and components to subscribe to built-in events and trigger workflows. For instance, a Casewhere application can subscribe to the "User Authorizing" event to execute additional authorization logic, determining whether to accept or reject the request.

Additionally, Casewhere applications or components can emit custom events within a workflow, allowing other applications or components to subscribe or intercept these events to handle their own business processes.

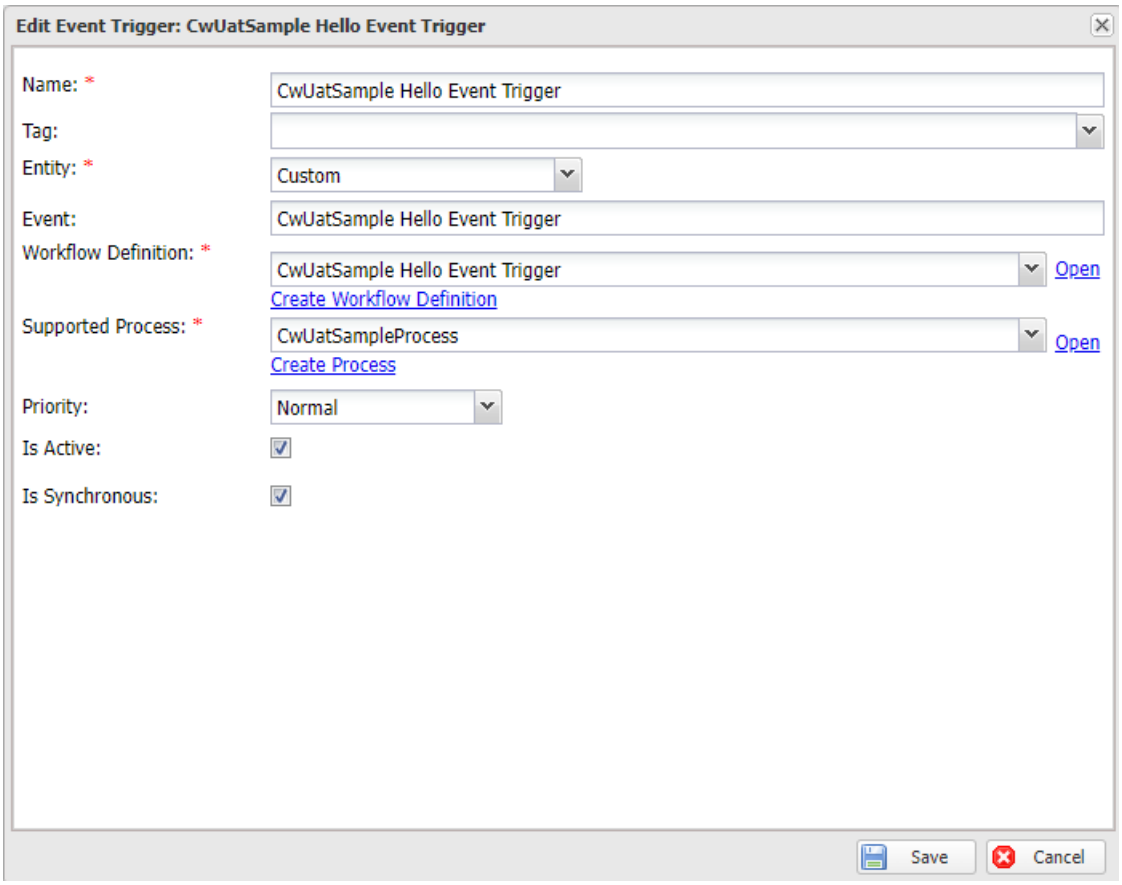
In Casewhere Admin, navigate to **Event Triggers**. From here, you can create, edit and manage event triggers like any product resource.



#	Name	Tag	Workflow Name	Event Name	Product Name	Entity Type	Priority	Is Active	Is Synchronous	Delay	Created Date	Created By	Modified Date	Modified By
1	100 workflow A low		100 workflow A low	Workflow executing	ds	Workflow	Low	☑	false	0	11-03-2022	CWAdmin	11-03-2022	CWAdmin
2	100 workflow B normal		100 workflow B normal	Workflow executing	ds	Workflow	Normal	☑	false	0	11-03-2022	CWAdmin	11-03-2022	CWAdmin
3	100 workflow C urgent	1111. 2222, 33	100 workflow C urgent	Workflow executing	ds	Workflow	Urgent	☑	false	0	11-03-2022	CWAdmin	17-08-2022	CWAdmin
4	87801AddClaimUserauthoring		S30AddClaimToUser	User authorizing	S30EventTrigger	WorkerSite	Normal	☐	true	0	27-04-2021	CWAdmin	29-07-2022	CWAdmin
5	AbandonAuthorization		VanTest_AbandonAuthorization	User authorizing	VanTest_EventTrigger	WorkerSite	Normal	☑	true	0	26-04-2021	dhv	18-05-2021	dhv
6	B_ET_AutomaticWfSetting		B_Wf_AutomaticWfSetting TestEvent...	B_ET_AutomaticWfSetting	LGB_AutomaticWfSetting	Custom	Normal	☑	false	0	24-05-2022	lgb	24-05-2022	lgb
7	B_ET_CustomEvent_createSampleEmail		B_Wf_EventTrigger CreateSampleEm...	B_ET_CustomEvent_createS...	LGB_Trigger	Custom	Normal	☑	false	0	10-06-2022	lgb	10-06-2022	lgb
8	B_ET_NoticationEmail for NewAssociate		B_Wf_EventTrigger NotificationEmail...	Workflow completed	LGB_Trigger	Workflow	Normal	☑	false	0	10-06-2022	lgb	10-06-2022	lgb
9	B_ET_UpdateNote		B_Wf_EventTrigger UpdateNote	Workflow completed	LGB_Trigger	Workflow	Normal	☑	false	0	10-06-2022	lgb	10-06-2022	lgb
10	B_ET_Worksersite Authorized NotificationEmail		B_Wf_EventTrigger NotificationEmail...	User authorized	LGB_Trigger	WorkerSite	Normal	☑	false	0	10-06-2022	lgb	10-06-2022	lgb
11	B_ET_Worksersite SignOut NotificationEmail		B_Wf_EventTrigger NotificationEmail...	User signed-out	LGB_Trigger	WorkerSite	Normal	☑	false	0	10-06-2022	lgb	10-06-2022	lgb
12	B_ET_Worksersite Unauthorized NotificationEmail		B_Wf_EventTrigger NotificationEmail...	User unauthorized	LGB_Trigger	WorkerSite	Normal	☑	false	0	10-06-2022	lgb	10-06-2022	lgb
13	Case Worker Site Convert User Role Name		Cpk Event Trigger Case Worker Cove...	User authorizing	Casewhere KK AAD Login	WorkerSite	Normal	☑	true	0	11-02-2022	CWadmin	11-02-2022	CWadmin
14	Case Worker Site Set Department Claim		Cpk Event Trigger Case Worker Site ...	User authorizing	Casewhere KK AAD Login	WorkerSite	Normal	☑	true	0	11-02-2022	CWadmin	11-02-2022	CWadmin
15	Chau test BroadcastCustomEvent - Test		Triggered BroadcastCustomEvent Test	BroadcastCustomEvent - Test		Custom	Normal	☑	false	0	09-08-2022	hnc	09-08-2022	hnc
16	Config Site Convert User Role Name		Cpk Event Trigger Case Worker Cove...	User authorizing	Casewhere KK AAD Login	WorkerSite	Normal	☑	true	0	11-02-2022	CWadmin	11-02-2022	CWadmin
17	Create Merge Data To Send Expiration Remind...		Cpk Create Merge Data To Send Expi...	Create Merge Data To Send ...	Digital Parking KK	Custom	Low	☑	false	0	02-12-2021	dhv	28-02-2022	dhv
18	CwUatSample Generate Next		CwUatSample Generate Dummy Data	CwUatSample Generate Next	CwUatSample	Custom	ASAP	☑	false	5000	21-08-2021	CWAdmin	15-12-2021	pnk
19	CwUatSample Hello Event Trigger		CwUatSample Hello Event Trigger	CwUatSample Hello Event Tr...	CwUatSample	Custom	Normal	☑	false	0	01-09-2021	CWAdmin	15-12-2021	pnk
20	CwUatSample Send Message		CwUatSample Send Message	Send Message	CwUatSample	Custom	Urgent	☑	false	0	12-08-2021	pnk	15-12-2021	pnk
21	CwUatSample Trigger Event Emit DataString		CwUatSample Create Countries by s...	CwUatSample Trigger Event ...	CwUatSample	Custom	Urgent	☑	false	2000	14-10-2021	CWAdmin	15-12-2021	pnk
22	CwUatSample Trigger Event Emit Empty		CwUatSample Trigger Event Emit Da...	CwUatSample Trigger Event ...	CwUatSample	Custom	Low	☑	false	2000	14-10-2021	CWAdmin	15-12-2021	pnk
23	DIBS Payment Cancelation Requested By Cust...		Cpk Handle DIBS Payment Cancelati...	DIBS Payment Cancelation ...	Digital Parking KK	Custom	Urgent	☑	true	0	02-12-2021	dhv	28-02-2022	dhv
24	Delete Document Signees After Delete Docum...		Delete Document Signees	DeleteDocumentSigneesBas...	Casewhere Document Confi...	Custom	Normal	☑	true	0	04-08-2022	hnc	04-08-2022	hnc
25	Dfsa Issue Claims Admin		Dfsa Event Trigger Issue Claims	User authorizing	Casewhere DFSA	WorkerSite	Normal	☑	true	0	02-12-2021	dhv	09-12-2021	CWAdmin

Configuration

The following details are necessary to configure an event trigger:



Name: Name must be required and unique.

Entity: There are 3 main entities: Worker Site, Workflow, and Custom. It's possible to subscribe to a custom event emitted from a workflow. The event name must be provided.

Event: Select a built-in event or enter the custom event name.

Workflow: The workflow triggered when the event emitted. Casewhere only accepts automatic, case-creation workflow definitions for using with event triggers. To access the event's data, the workflow definition must also accept input.

Process: The process in which the triggered workflow will run. A new case is created every time Casewhere executes an event trigger.

Priority: When an event happens, triggers will run in the order *Urgent* > *As Soon As Possible* > *Normal* > *Low*.

Is Active: Inactive event triggers will not be executed.

Is Synchronous: To determine whether Casewhere should execute the workflow immediately in the current context or if the workflow should be queued. You can also choose how long the workflow should be delayed.

Built-in events

Worker site

`User authorizing`

The event is emitted when Casewhere starts authorizing users after they login. It is possible to intercept the authorization flow here with your custom logic to accept or reject the user session accordingly.

Event data:

PROPERTY NAME	PROPERTY TYPE	DESCRIPTION
EventName	`string`	"User authorizing"
WorkerDomainName	`string`	The domain name of the worker site
WorkerSiteId	`Guid`	The id of the worker site
Username	`string`	The `Username` claim
SPIDPSessionID	`Guid`	The session id to be used with `IdentityApi`
UserId	`string`	The `UserId` claim

`User authorized`

The event is emitted after Casewhere authorizes a user login session. It's possible to subscribe to this event and enrich the user session with additional claims.

Event data:

PROPERTY NAME	PROPERTY TYPE	DESCRIPTION
EventName	`string`	"User authorized"
WorkerDomainName	`string`	The domain name of the worker site
WorkerSiteId	`Guid`	The id of the worker site
Username	`string`	The `Username` claim
UserSessionId	`Guid`	The user session id managed by Casewhere
UserId	`string`	The `UserId` claim

`User sign-out`

The event is emitted when users log out.

Event data:

PROPERTY NAME	PROPERTY TYPE	DESCRIPTION
EventName	`string`	"User signed-out"
WorkerDomainName	`string`	The domain name of the worker site
WorkerSiteId	`Guid`	The id of the worker site
Username	`string`	The `Username` claim
UserSessionId	`Guid`	The user session id managed by Casewhere

PROPERTY NAME	PROPERTY TYPE	DESCRIPTION
UserId	`string`	The `UserId` claim

`User unauthorized`

The event is emitted when Casewhere rejects a request due to the user's insufficient permission.

Event data:

PROPERTY NAME	PROPERTY TYPE	DESCRIPTION
EventName	`string`	"User unauthorized"
WorkerDomainName	`string`	The domain name of the worker site
WorkerSiteId	`Guid`	The id of the worker site
Username	`string`	The `Username` claim
UserSessionId	`Guid`	The user session id managed by Casewhere
UserId	`string`	The `UserId` claim

Workflow

`Workflow executing`

The event is emitted right before Casewhere executes a workflow.

Event data:

PROPERTY NAME	PROPERTY TYPE	DESCRIPTION
EventName	`string`	"Workflow executing"
CaseId	`Guid`	The id of the case in which the workflow is executed
WorkflowId	`Guid`	The id of the worker site
WorkflowDefinitionId	`Guid`	The id of the workflow definition
WorkflowName	`string`	The name of the workflow
ActivityId	`Guid?`	The id of the activity that triggers the workflow
Input	`IList<string>`	The list of data object ids as workflow input

`Workflow executed`

The event is emitted right after Casewhere executes a workflow.

Event data:

PROPERTY NAME	PROPERTY TYPE	DESCRIPTION
EventName	`string`	"Workflow executed"
CaseId	`Guid`	The id of the case in which the workflow is executed
WorkflowId	`Guid`	The id of the worker site
WorkflowDefinitionId	`Guid`	The id of the workflow definition
WorkflowName	`string`	The name of the workflow
Input	`IList<string>`	The list of data object ids as workflow input

`Workflow completed`

The event is emitted when Casewhere completes a workflow. A workflow is completed when all required activities are completed. Read more here.

Event data:

PROPERTY NAME	PROPERTY TYPE	DESCRIPTION
EventName	`string`	"Workflow completed"
CaseId	`Guid`	The id of the case in which the workflow is executed
WorkflowId	`Guid`	The id of the worker site
WorkflowDefinitionId	`Guid`	The id of the workflow definition
WorkflowName	`string`	The name of the workflow
Input	`IList<string>`	The list of data object ids as workflow input

Emit custom events

You can emit a custom event from a workflow **Scripted Activity** using `ctx.Emit`.

```
ctx.Emit("A custom event", new
{
    TextData = "data",
    NumericData = 2,
    BooleanData = true
});
```

Please note that the event is sent to the event queue only if the **Scripted Activity** completes without errors. If the activity crashes, Casewhere will reject all events created during its execution. To ensure that events are sent to the queue regardless of the activity's status, you can use `ctx.ForceEmit`, available since version 2.8.

Examples

Learn more about working with event triggers here.

Event trigger - Examples

Introduction

This article aims to show you how to use event triggers to handle different scenarios in reality. It also covers important configurations of workflow definition to send and receive data from an event trigger. Before starting, it may be worth ensuring you know how to add a worker site and a workflow definition.

Examples

Example 1: Subscribe to the `User unauthorized` event to log suspicious accesses

Introduction

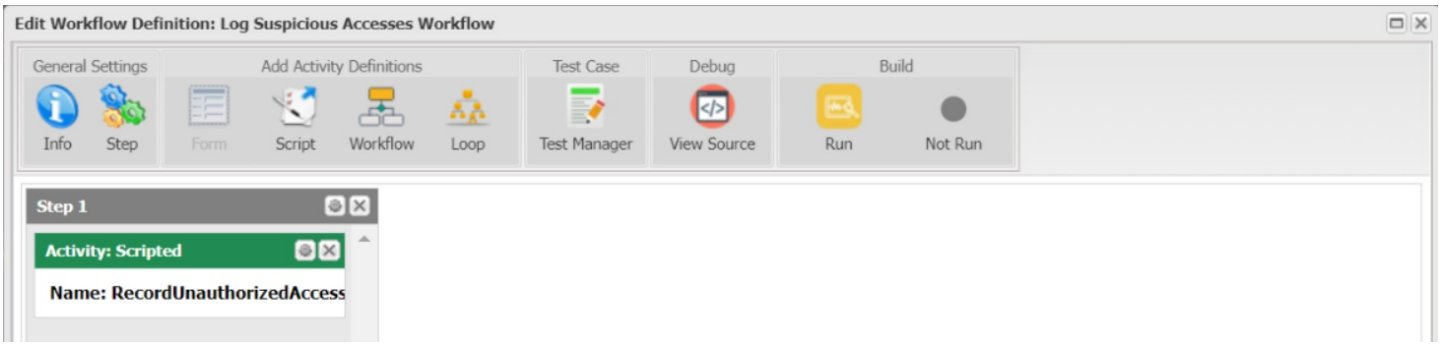
When a user attempts to access a restricted page, Casewhere emits an event called `User unauthorized`. In this example, you will configure an event trigger to listen to the event and starts a workflow, which extracts data from the event and add a log entry in database.

Configuration

Add the following resources in Casewhere Admin.

Workflow definition: Log Suspicious Accesses

The workflow to record unauthorized accesses.

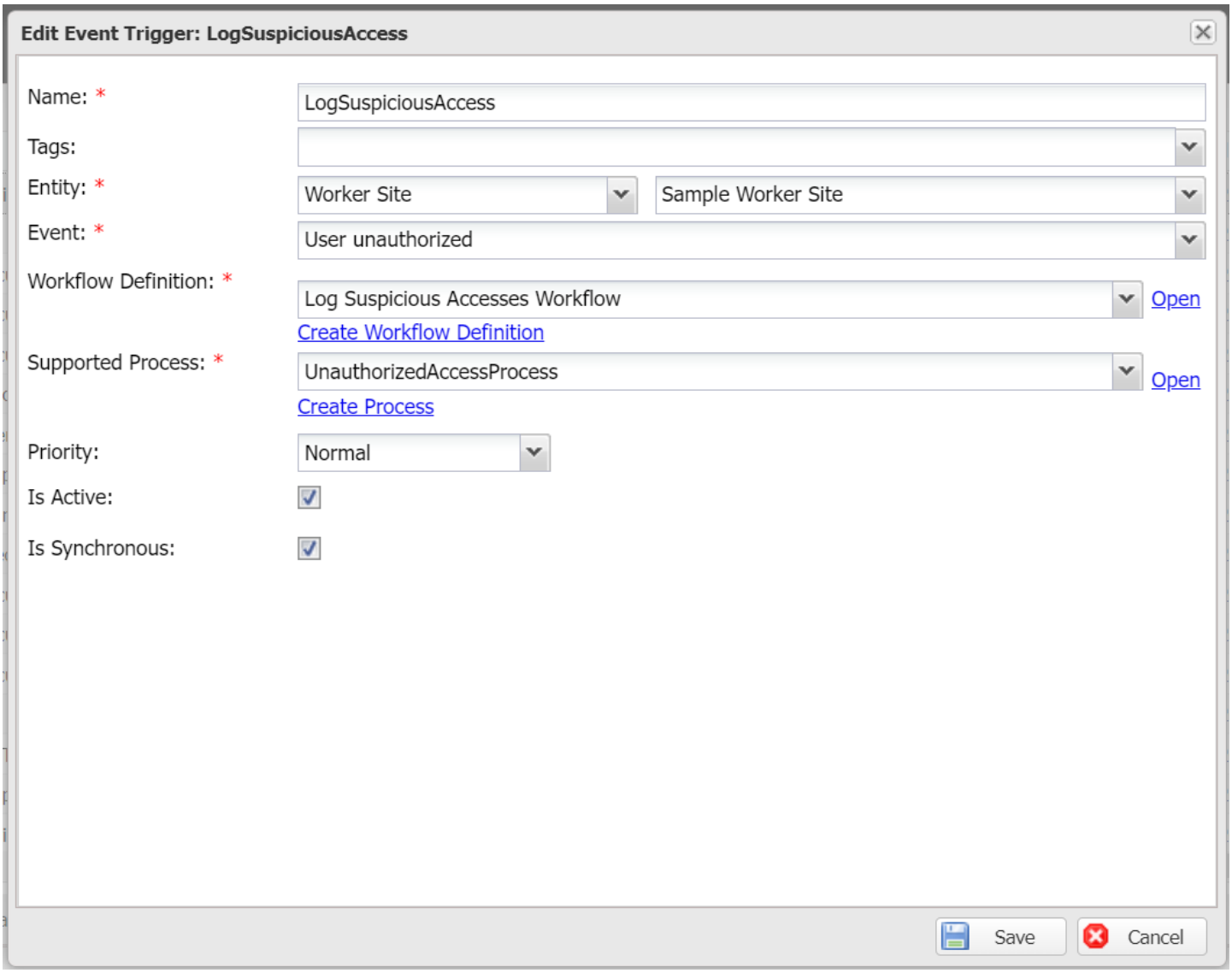


The below script demonstrates how to access the event data.

```
var dataApi = ctx.Use<IDataApi>();
var username = ctx.Input.Username;
// Add a new log entry with data from the event
dataApi.Add("UnauthorizedAccessLog", new { Username = Username });
```

Event trigger: Log Suspicious Access

The event trigger that listens to the `User unauthorized` event on the worker site "Sample Worker Site" and executes the workflow "Log Suspicious Access" when users access a page they are not allowed to.



The workflow should start immediately when the event is emitted if **Is Synchronous** is set. Otherwise, the workflow will be added to the queue and run afterwards in the background.

Example 2: Emit and subscribe to custom events

Introduction

Imagine you're developing a component to send emails in Casewhere. You want to provide your consumers with a convenient way to intercept and modify the email content, e.g., adding an email signature, before it is sent without modifying the component. In this example, you will learn how to emit an event from a workflow, and then handle the event using event trigger.

Configuration

Emit event

Assuming you have a workflow sending email with the following script:

```
var dataApi = ctx.Use<IDataApi>();
var pluginApi = ctx.Use<IPluginApi>();

var emailInfo = new
{
    Recipient = "example@casewhere.com",
    Body = "Welcome to custom event"
};

var emailId = dataApi.Add("CwUatEmailMessage", emailInfo);

// emit the event so others can subscribe or intercept
ctx.Emit("message sending", emailInfo);

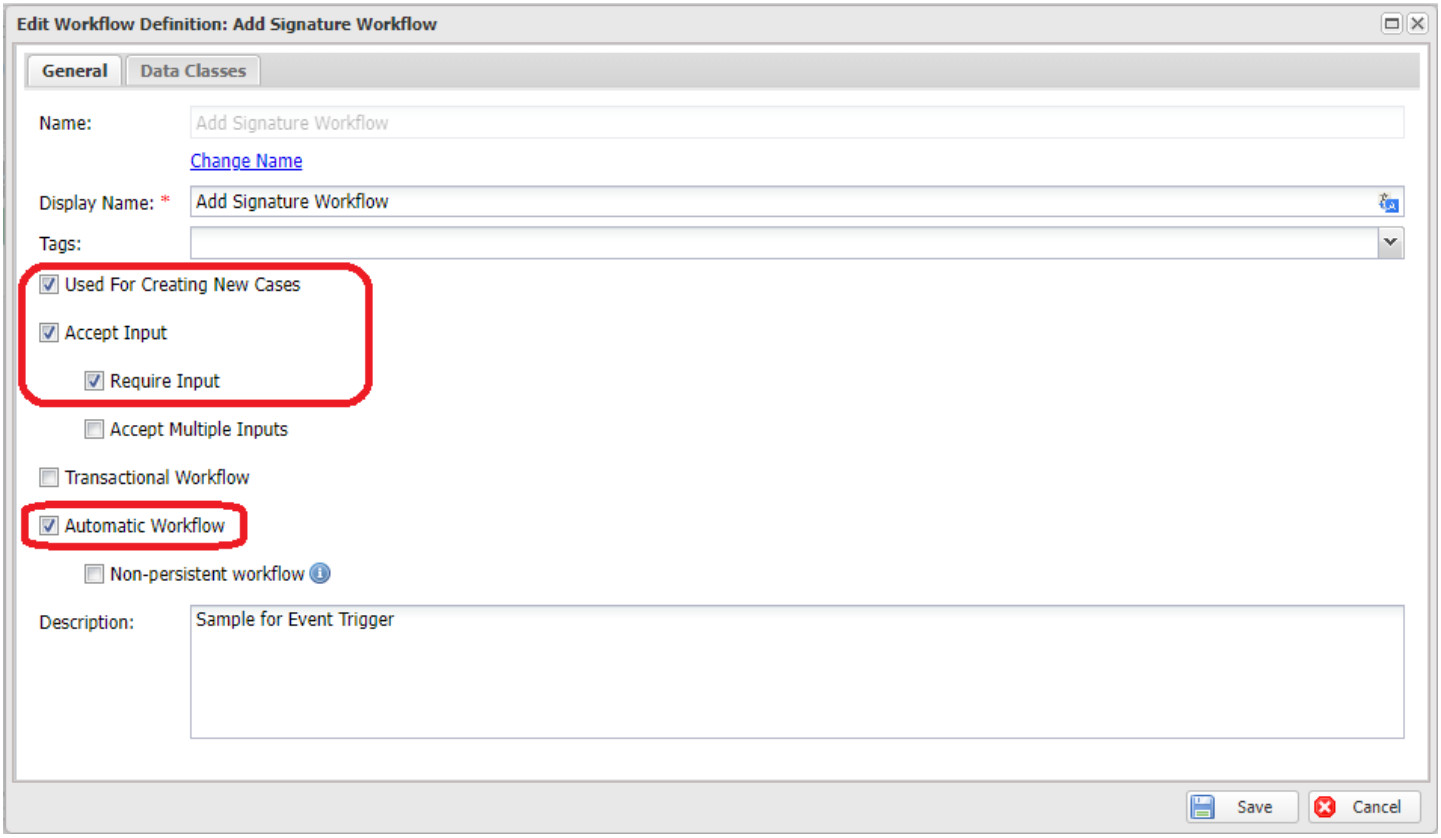
// load the email again because someone might modify it by subscribing to the event
var email = dataApi.Load(emailId);

// send email using plugin
var mailParameters = new Dictionary<string, object>()
{
    { "body", email["Body"] },
    { "to", email["Recipient"] },
};

var pluginResult = pluginApi.Invoke("Mail plugin", "SendEmail", mailParameters);
```

Handle event

Add a workflow for adding the email signature. Casewhere only accepts automatic, create-case workflow definitions for using with event triggers. We also need to set **Accept Input** to receive the event data as workflow input.



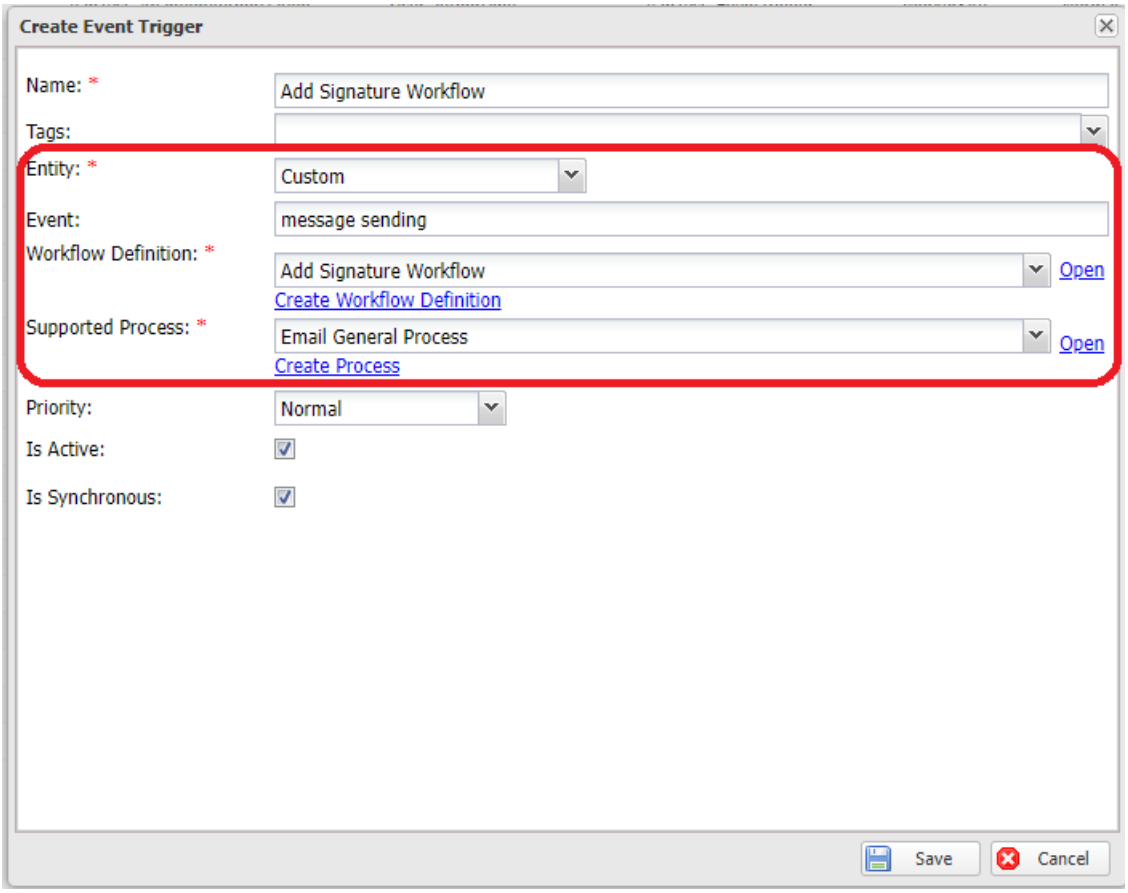
Add a scripted activity to append the signature to the email content.

```
var dataApi = ctx.Use<IDataApi>();

var emailId = ctx.Input["Data"]["Id"];
var emailBody = ctx.Input["Data"]["Body"];
emailBody += "<br/>Cheers, your Casewhere team.";

dataApi.Update(emailId, new { Body = emailBody });
```

Add a event trigger to subscribe to the custom event `message sending`. Make sure you enable **Is Synchronous** so that the workflow can be executed before the email is sent.



Example 3: Intercept the `User authorizing` event to handle additional authorization logic

Introduction

The scenario is that your solution needs to verify the user contract when he or she logins into your system. If the contract is no longer valid, the user will not be allowed to access.

Configuration

Workflow definition: Validate User Contract

Casewhere provides the API `IWebApi.AbandonAuthorization` for canceling the user login session.

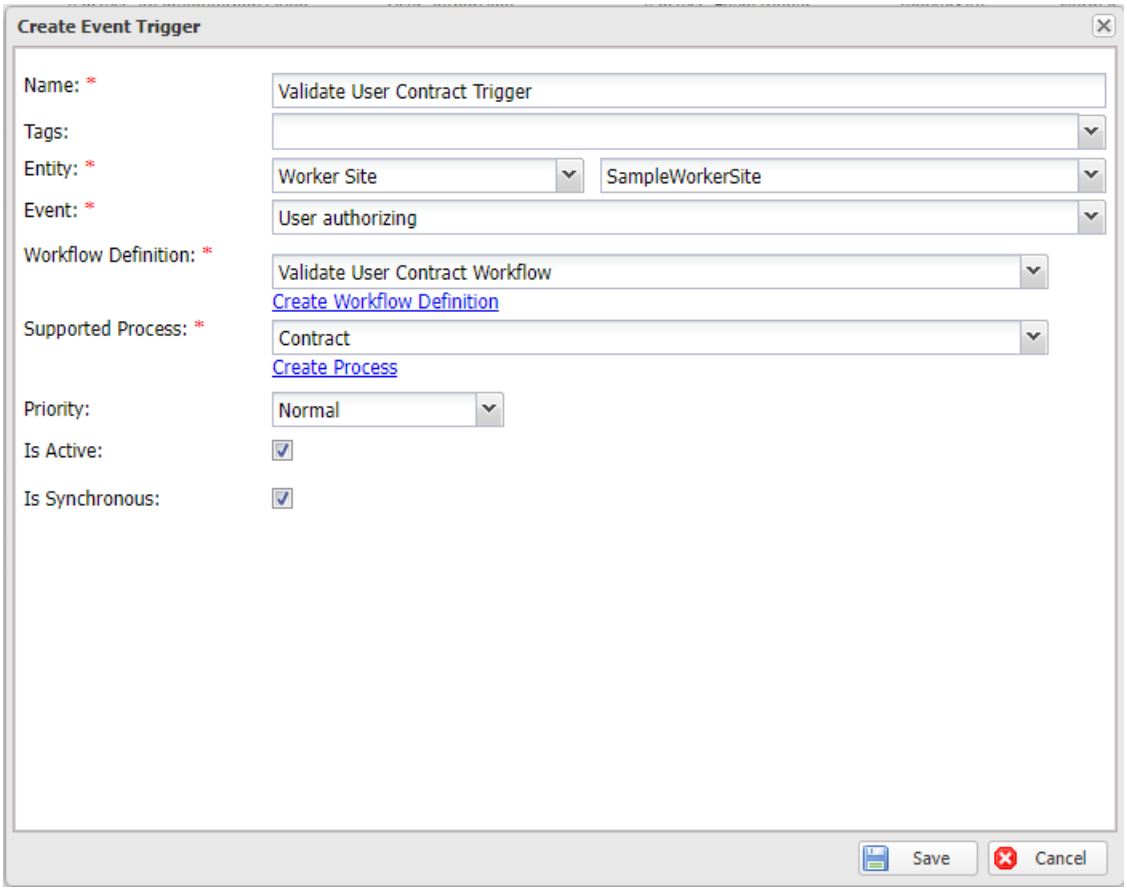
In this example, a plugin is used to validate the user contract and determine whether to accept or reject the login request.

```
var webApi = ctx.Use<IWebApi>();
var pluginApi = ctx.Use<IPluginApi>();
var username = ctx.Input.Username;

var parameters = new Dictionary<string, object>();
parameters["Username"] = username;
var pluginResult = pluginApi.Invoke("ContractManagement", "Validate", parameters);
var isValid = pluginResult.Get<bool>("IsValid");
if(!isValid)
{
    webApi.AbandonAuthorization();
}
```

Event trigger: Validate User Contract

Add an event trigger to subscribe to the `User authorizing` event. Make sure you enable **Is Synchronous** so that the workflow can be executed in the context of the user authentication.

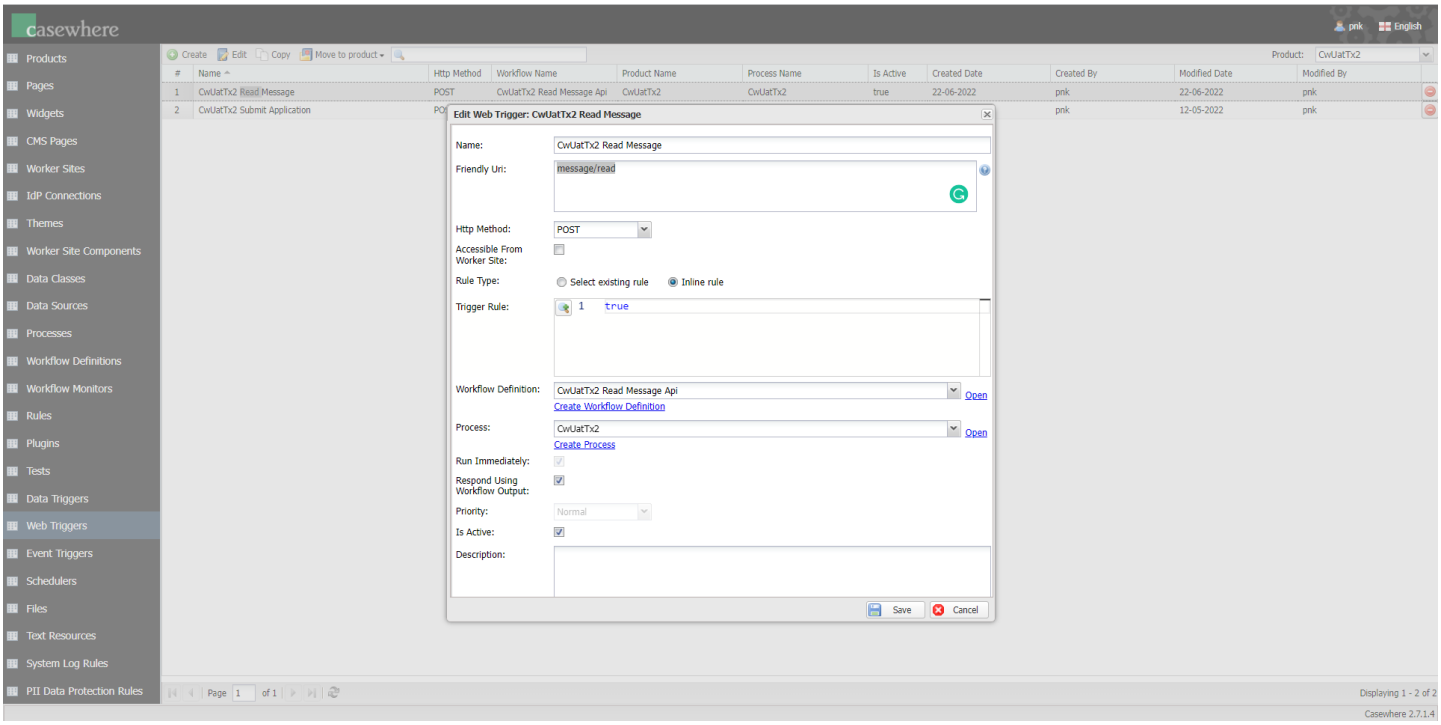


Web trigger

Introduction

There are many ways a workflow can be triggered in Casewhere. With web triggers, you can expose a Web API endpoint for your external systems to trigger a workflow in Casewhere. Using web triggers is one of the most popular ways to integrate other systems with Casewhere.

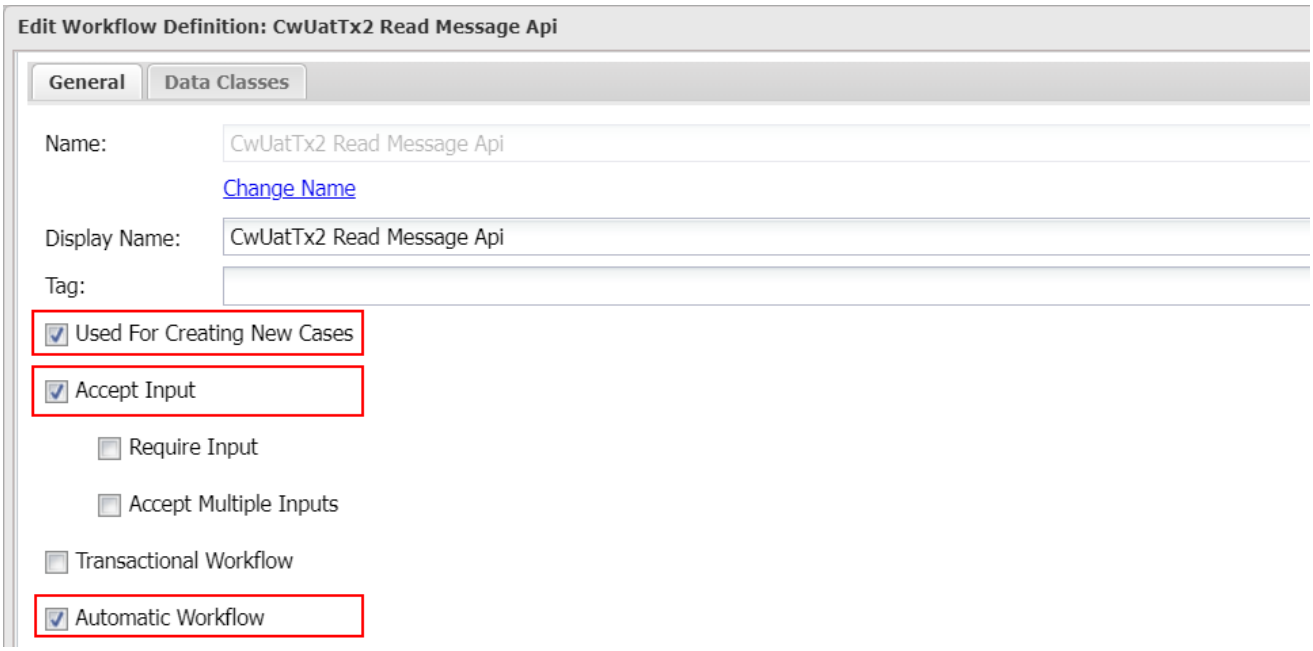
In Casewhere Admin, navigate to **Web Triggers**. From here, you can create, edit and manage web triggers like any product resource.



Basic configuration

The following information is required to configure a web trigger:

- **Name**: The web trigger name must be unique. It's possible to execute a web trigger using its name through the endpoint `https://{your-casewhere-api-domain}/external/v0.1/{web-trigger-name}`.
- **Http Method:** The HTTP request methods. You can read more here.
    - Initially, Casewhere only supports POST and GET. From version `2.7.2`, Casewhere supports two more methods, PUT and DELETE.
- **Rule**: To validate the web request and determine where or not a workflow should be executed. If the validation fails, Casewhere will respond with status code 400. Learn to write web trigger rule here. While you can write an expression using **Inline rule**, it's recommended to create a separate rule for reuse.
- **Workflow Definition**: The triggered workflow definition. Casewhere only accepts automatic, create-case workflow definitions for using with web triggers. The workflow definition must also accept input in order to access the posted data.



- **Process**: The process in which the triggered workflow will run. A new case is created every time Casewhere executes a web trigger.
- **Run Immediately**: To determine whether or not Caseware should execute the workflow in the same context with the web request or should Casewhere queue the workflow. Please note that if you choose to queue the workflow, it's not possible to send response data to clients from the workflow because Casewhere will respond immediately right after it queues the workflow.

Friendly Uri

It's possible to access web triggers in a more friendly way using custom URIs. A web trigger can accept many URIs as needed.



You then can access the web trigger using the endpoint `https://{your-casewhere-api-domain}/dynamic/{friendly-uri}`. For example, you can access the above example via:

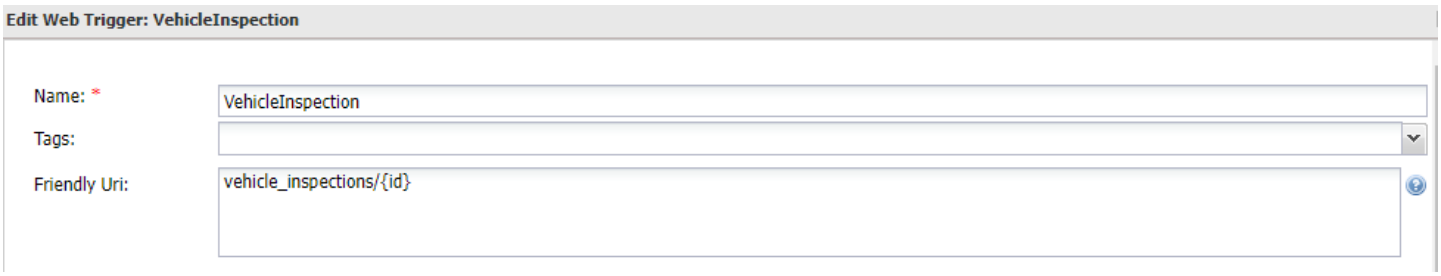
- https://api-local.casewhere.com/dynamic/post/
- https://api-local.casewhere.com/dynamic/post/all
- https://api-local.casewhere.com/dynamic/post/casewhere/all
- https://api-local.casewhere.com/dynamic/post/casewhere/platform/all

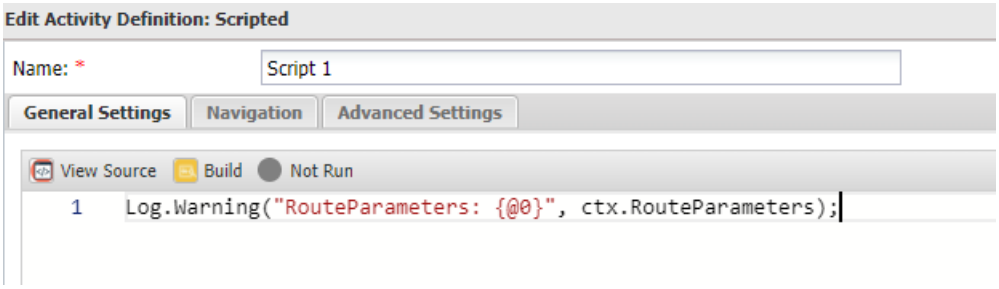
Currently, Casewhere supports the friendly URI of the web trigger with up to four segments.

Parameterized URI

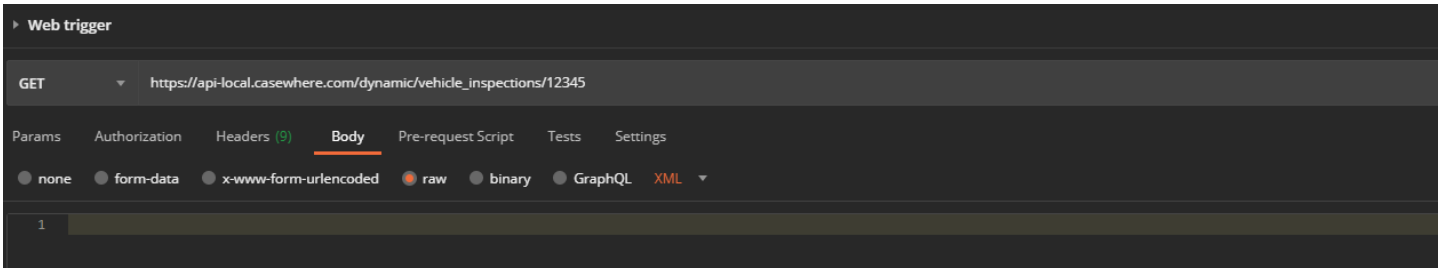
Casewhere provides the solution to work with the URL parameters. For example, if a pattern `/dynamic/vehicle_inspections/{id}` is configured for a web trigger, when you request to endpoint `/dynamic/vehicle_inspections/12345`, Casewhere will trigger that web trigger, and set the parameter `id` with value is `12345` in the `ctx.RouteParameters`.

1. Create a web trigger with `id` as its parameter. This web trigger will trigger a workflow with a simple scripted activity use a warning log to write the `ctx.RouteParameters` into the System Log.

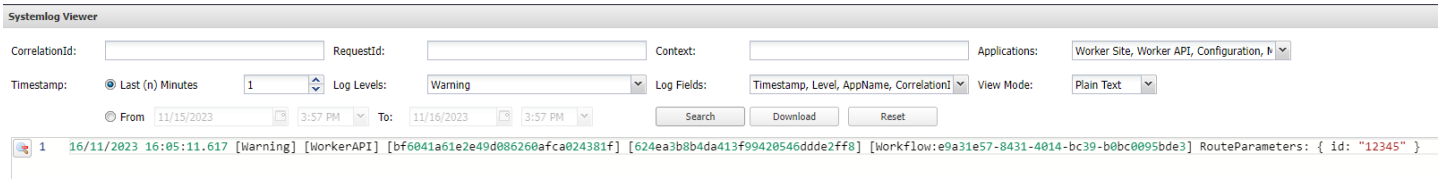




1. Make a request to the endpoint `/dynamic/vehicle_inspections/12345`



1. Observe the `ctx.RouteParameters` value in the System Log



Invoke web triggers from a worker site

Turning on the setting **Accessible From Worker Site** will enable Casewhere to run the web-trigger rule in the worker-site context when the request is sent from a worker site. You then can use the `@user` keyword to verify user's claims. Read more about rules and user context here.

Casewhere provides a web-trigger client so you can use to invoke web triggers from worker site widgets. Please read more here.

Work with query string

There are two ways to retrieve data in Casewhere.

Approach 1: Using Web API

> Please note that the `IWebApi` is available only when the setting **Run Immediately** is on.

The below code can be used in a scripted activity to retrieve data from the request query string.

```
var webApi = ctx.Use<IWebApi>();
// assuming we have the query string "?categoryIds=2&categoryIds=3&productId=1"
var queryString = webApi.GetQueryString();
var categoryId1 = queryString["categoryIds"].ToArray()[0]; // 2
var categoryId2 = queryString["categoryIds"].ToArray()[1]; // 3
var productId = queryString["productId"].FirstOrDefault(); // 1
```

Approach 2: Using `ctx.Parameters`

Form version `2.7.2`, Casewhere will parse the query string and store the values in the workflow, which is persisted in database, so it's possible to retrieve the query string value using `ctx.Parameters` even when the workflow runs in background.

```
var categoryId1 = ctx.Parameters["categoryIds"][0]; // 2
var categoryId2 = ctx.Parameters["categoryIds"][1]; // 3
var productId = ctx.Parameters["productId"]; // 1
```

Respond data

There are two ways to set web response in the triggered workflow.

Approach 1: Using Web API

> Please note that the `IWebApi` is available only when the setting **Run Immediately** is on.

You can set the response from the triggered workflow using `IWebApi.CreateResponse` in a scripted activity.

```
var webApi = ctx.Use<IWebApi>();
var data = new { FirstName = "John", LastName = "Doe" };
webApi.CreateResponse(data);
```

Furthermore, you can also set the file stream as the response of the triggered workflow using `IWebApi.CreateFileResponse` in a scripted activity as the code below:

```
var documentApi = ctx.Use<IDocumentApi>();
var webApi = ctx.Use<IWebApi>();

string content = "File created by scripted DOWNLOAD FILE BY STREAM";
string filePath =  Path.GetDirectoryName(Path.GetRandomFileName());
string fileName = "FileByStream.txt";
string downloadFileName = "FileByStream-Download.txt";
string fullPath = Path.Combine(filePath, fileName);

bool existsFolder = System.IO.Directory.Exists(filePath);
if(!existsFolder) System.IO.Directory.CreateDirectory(filePath);

using (StreamWriter sw = new StreamWriter(fullPath))
{
    sw.WriteLine(content);
    sw.Close();
}

Stream fs = File.OpenRead(fullPath);
webApi.CreateFileResponse(downloadFileName, fs);
```

> To work with the file stream response, you need to enable the **Run Immediately** setting and disable the **Respond Using Workflow Output** setting.

Approach 2: Using workflow output

Enable the setting **Respond Using Workflow Output**, so Casewhere will respond to clients using the output from the triggered workflows.

You can set the output for the triggered workflow using the code below in a scripted activity.

```
ctx.Output = data;
```

Queue triggered workflows

When the setting **Run Immediately** is turned off, Casewhere will queue the workflow and execute it in the background, separated from the context of the web request. You can control the order the workflow is dequeued by configuring **Priority**.

When a workflow runs in the background, that workflow can't interact with web-context APIs, including `IWebApi`.

Web API

Casewhere provides a set of APIs for developers to work with the HTTP requests and responses directly. You can read more here. Please note that these APIs are available only when the setting **Run Immediately** is on.

Support XML input/output

Casewhere also support XML type for input and output of the Web trigger. We can access the XML input via `ctx.Input`.

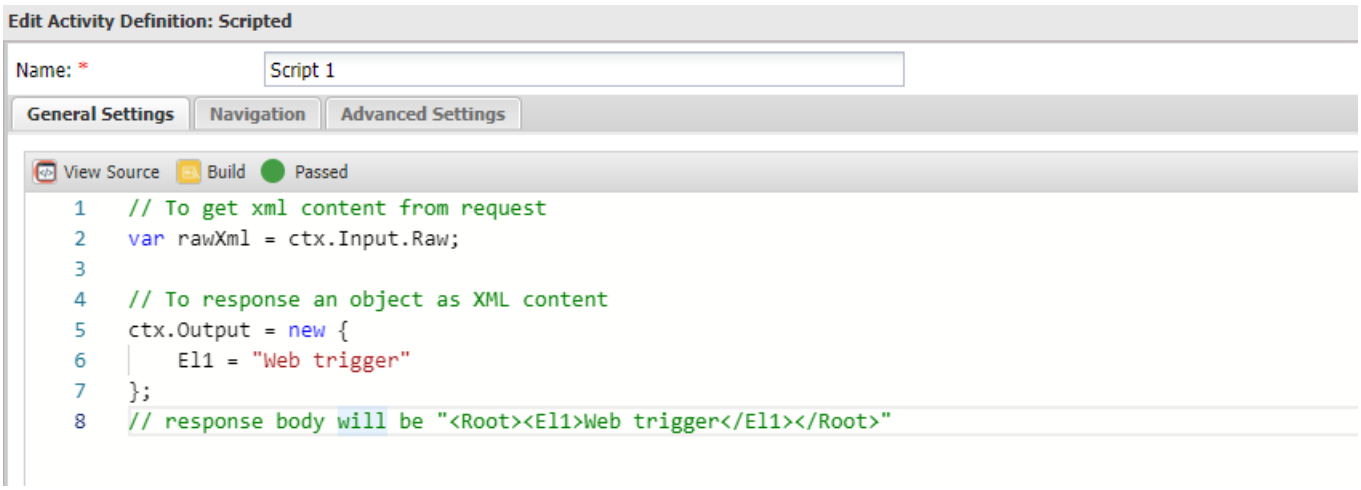
1. Create a request to a web trigger with the **Accept** header is **application/xml**.



1. Then create a web trigger to trigger a workflow with a simple scripted activity as below:

```
// To get xml content from request
var rawXml = ctx.Input.Raw;

// To response an object as XML content
ctx.Output = new {
    El1 = "Web trigger"
}
// response body will be "<Root><El1>Web trigger</El1></Root>"
```

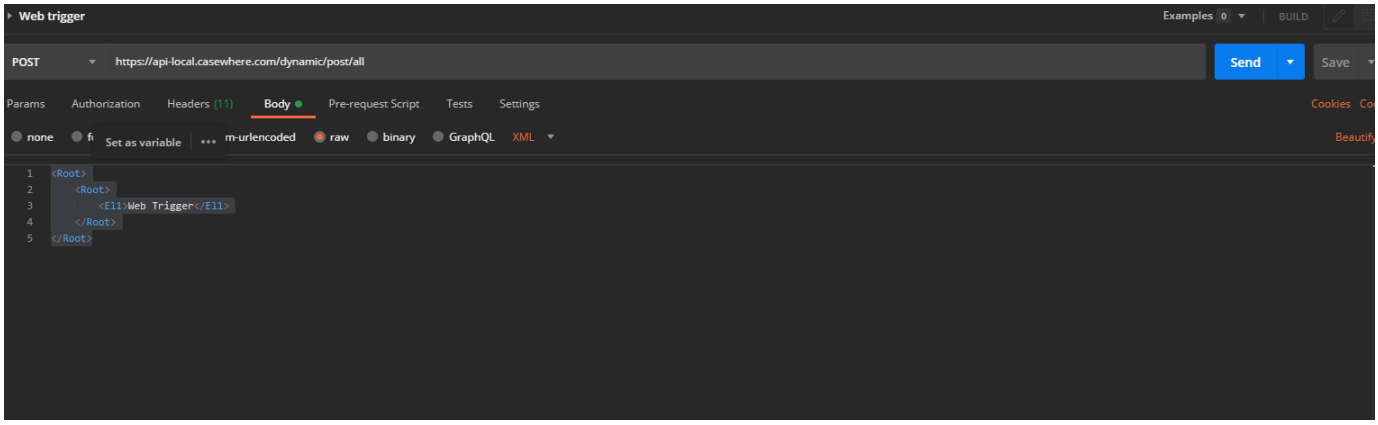


Please note that Casewhere will always wrap the XML output inside a root element as:

```
<Root>
    {Output content}
</Root>
```

1. Request to above web trigger with the input:

```
<Root>
 <Root>
     <El1>Web Trigger</El1>
 </Root>
</Root>
```



2. The response will be:

```
<Root>
 <Root>
     <El1>Web trigger</El1>
 </Root>
</Root>
```

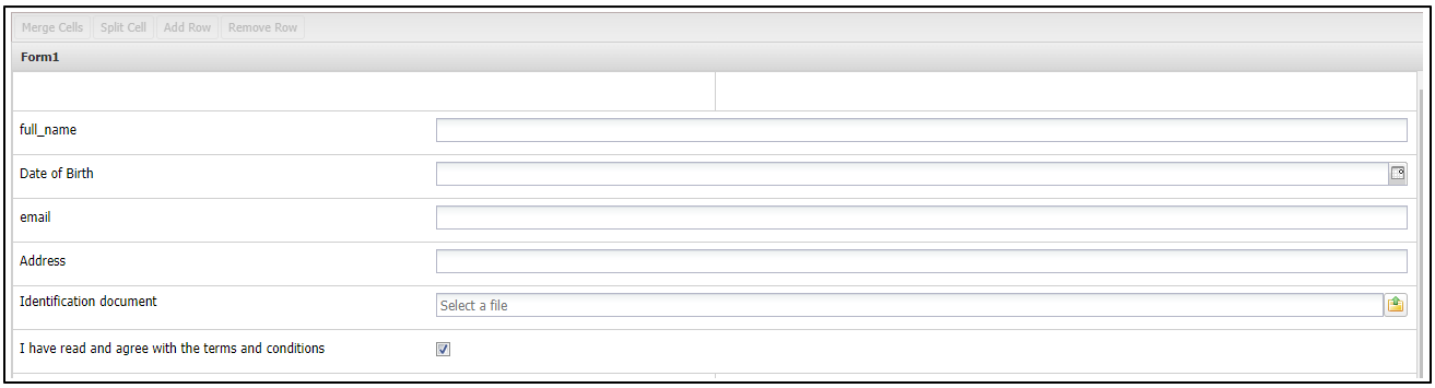


Form Activity

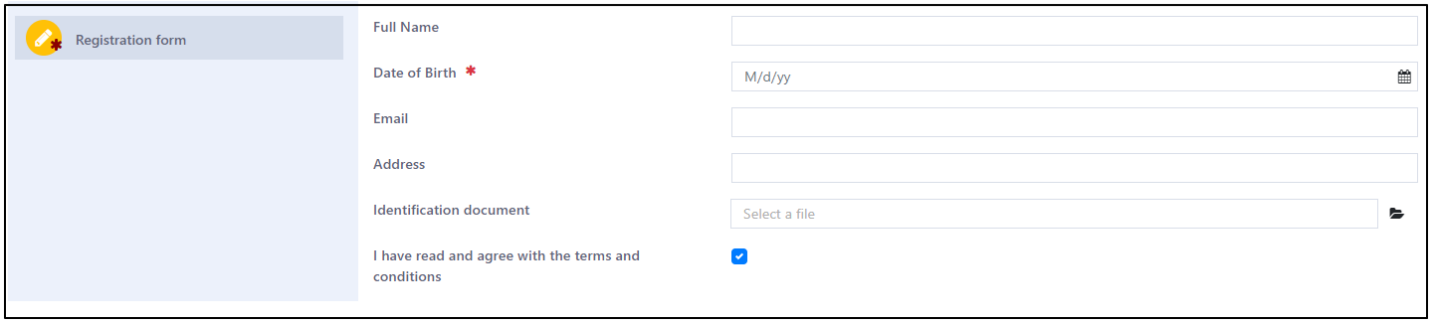
Introduction

Form Activity is designed to collect data from users. When a workflow reaches a **Form Activity**, it becomes idle until the user submits the form. Casewhere offers a visual editor for seamless form design and configuration. The following screenshots illustrate how the form appears in Casewhere Admin and how it is displayed to end-users on the worker sites:

- In Casewhere Admin:

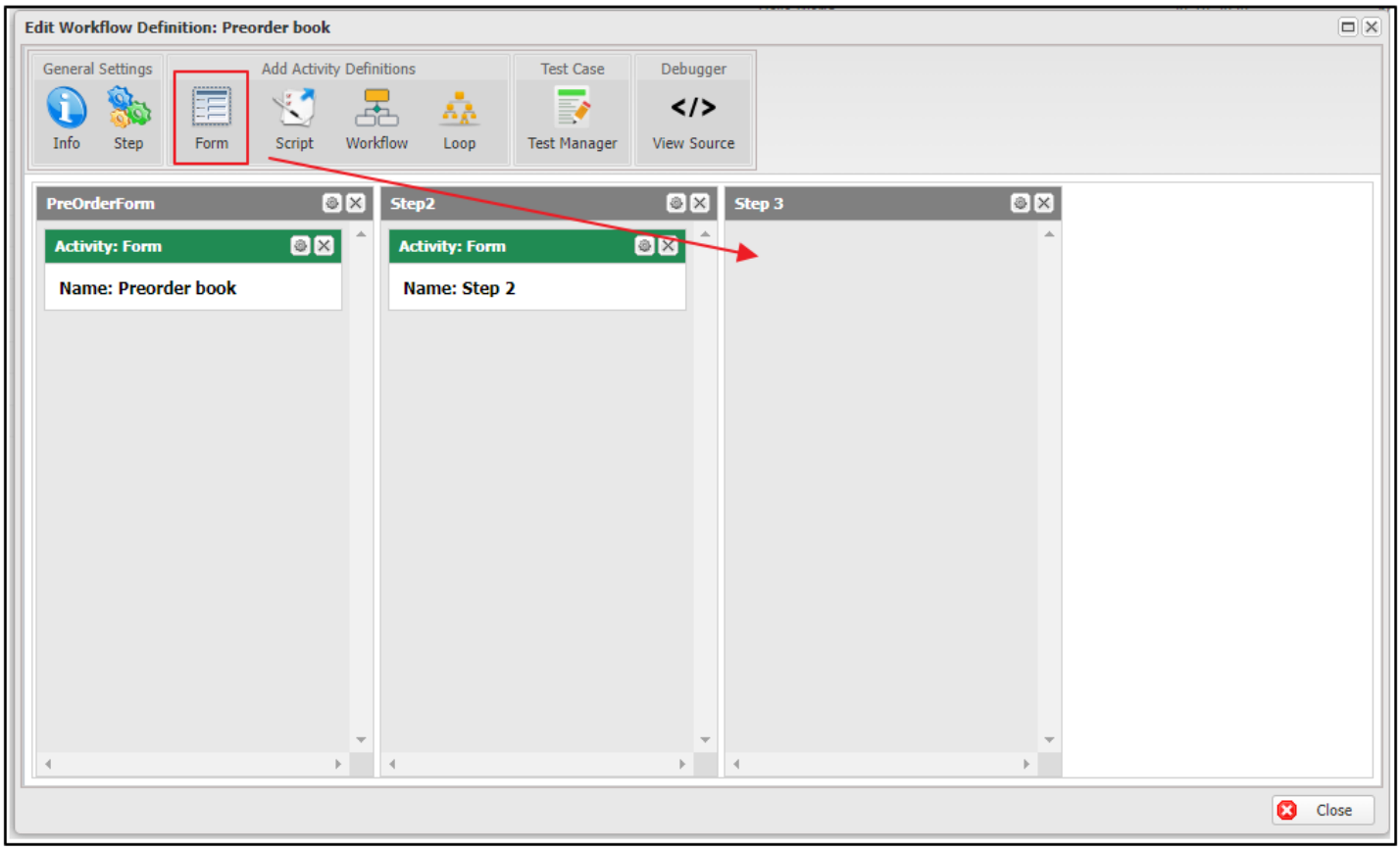


- In Worker site

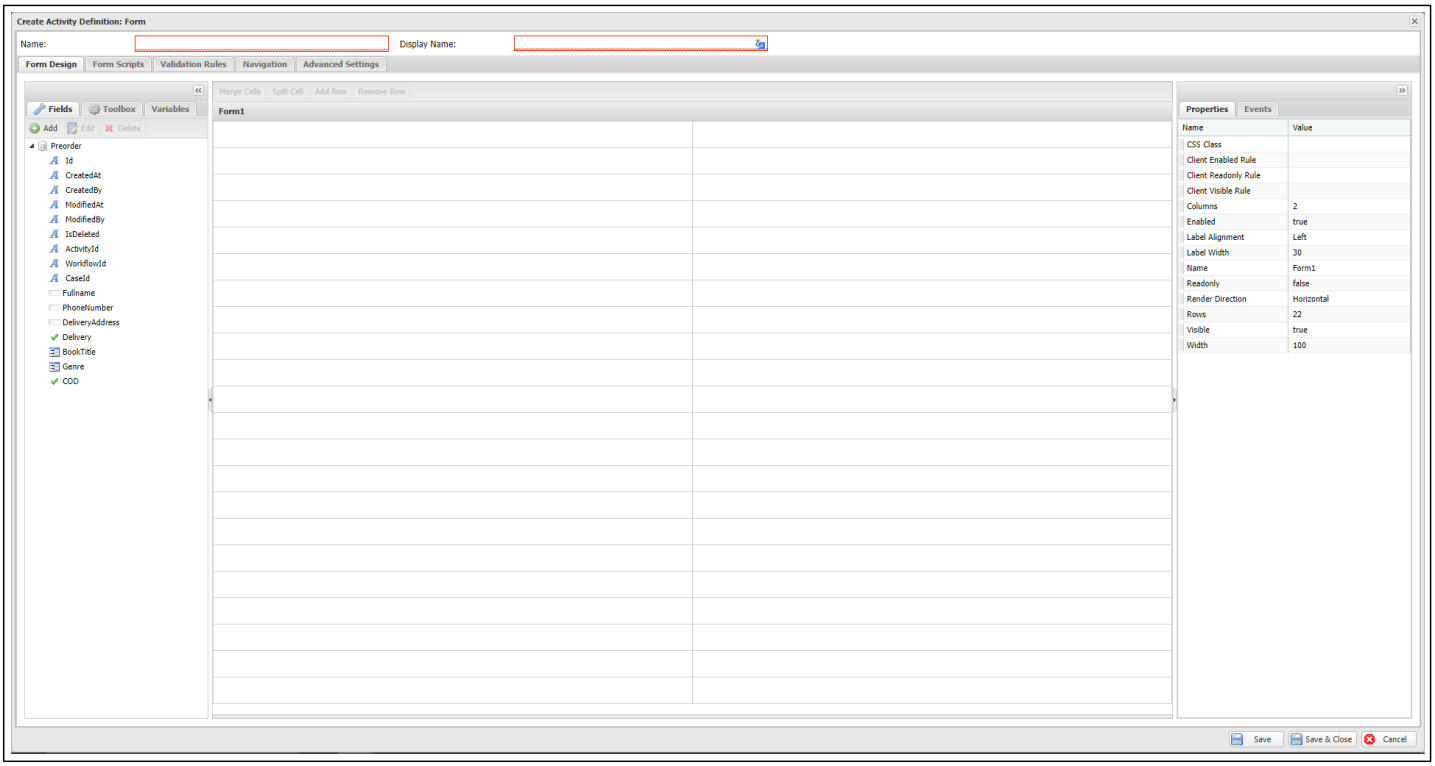


Add a new Form Activity

In the **Edit Workflow Definition** step, click on the **Form** icon in the **Add Activity Definitions** section and drag it into the **Step** panel you added.



After the drag-and-drop step, the **Form Activity** creation dialog will open. There are two required fields in this dialog: **Name** and **Display Name**. You must fill in both fields to save the form.



Other configurations for the form are optional and can be completed either during the creation step or later in the editing step.

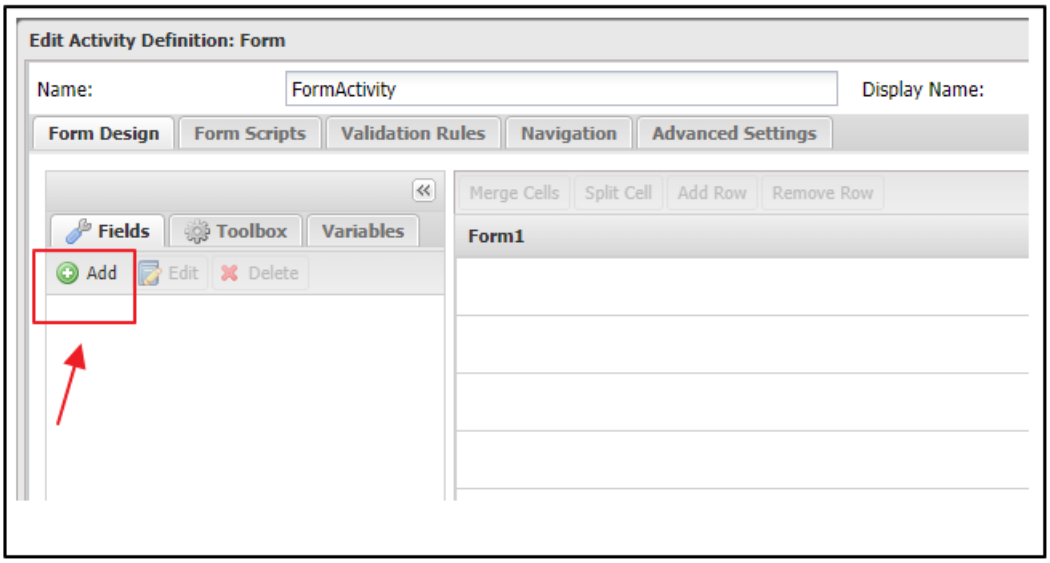
Form Design

Form Design is the section where you can configure the user interface of a form. In this section, you can add fields to the form from either a data source or the toolbox. Additionally, you can configure the properties and events of each field, as well as add variables for a field in the form.

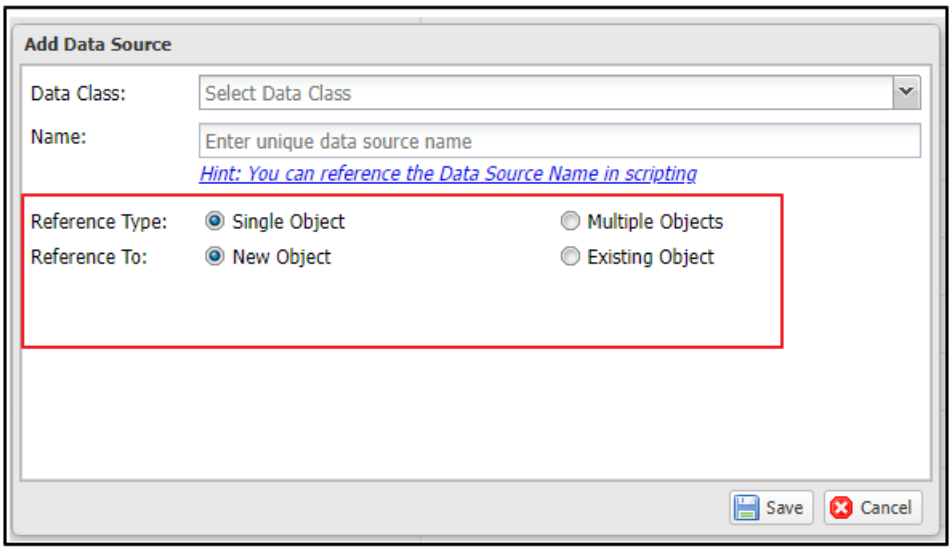
Add fields from Data Source

Step 1: Add a data source.

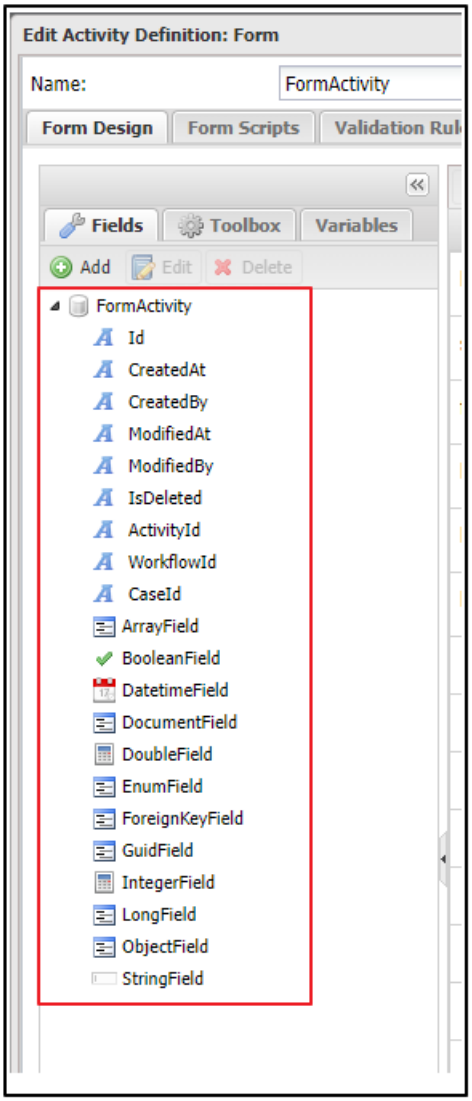
- In the **Fields** tab of the left menu, click on the **Add** button:



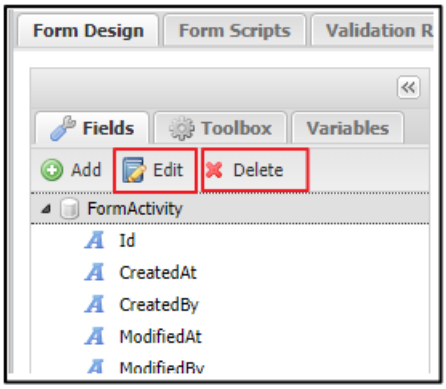
- Then, the **Add Data Source** dialog will open, allowing you to configure the necessary fields.
  - **Data Class**: This field is required. You must specify the data class with which your form will interact. If you do not see any data classes listed, you will need to enable them in your workflow definition. For more information, see here.
  - **Name**: This field is required. The data source name must be unique. By default, Casewhere uses the data class name, but you can modify it. You can reference the data source by its name in scripting.
  - **Reference Type**: This field is required. Currently, Casewhere only supports **Single Object**.
    - **Single Object**: Only one data object will be created after the form is submitted on the worker site.
  - **Reference To**: This field is required. You need to select one of the following two options:
    - **New Object**: Use this option if you want to create a new data object.
    - **Existing Object**: Use this option if you want to edit an existing data object.



- After filling in all required fields and clicking the **Save** button, the list of fields for the data source will be displayed in the left menu. You can also add multiple data sources if needed.

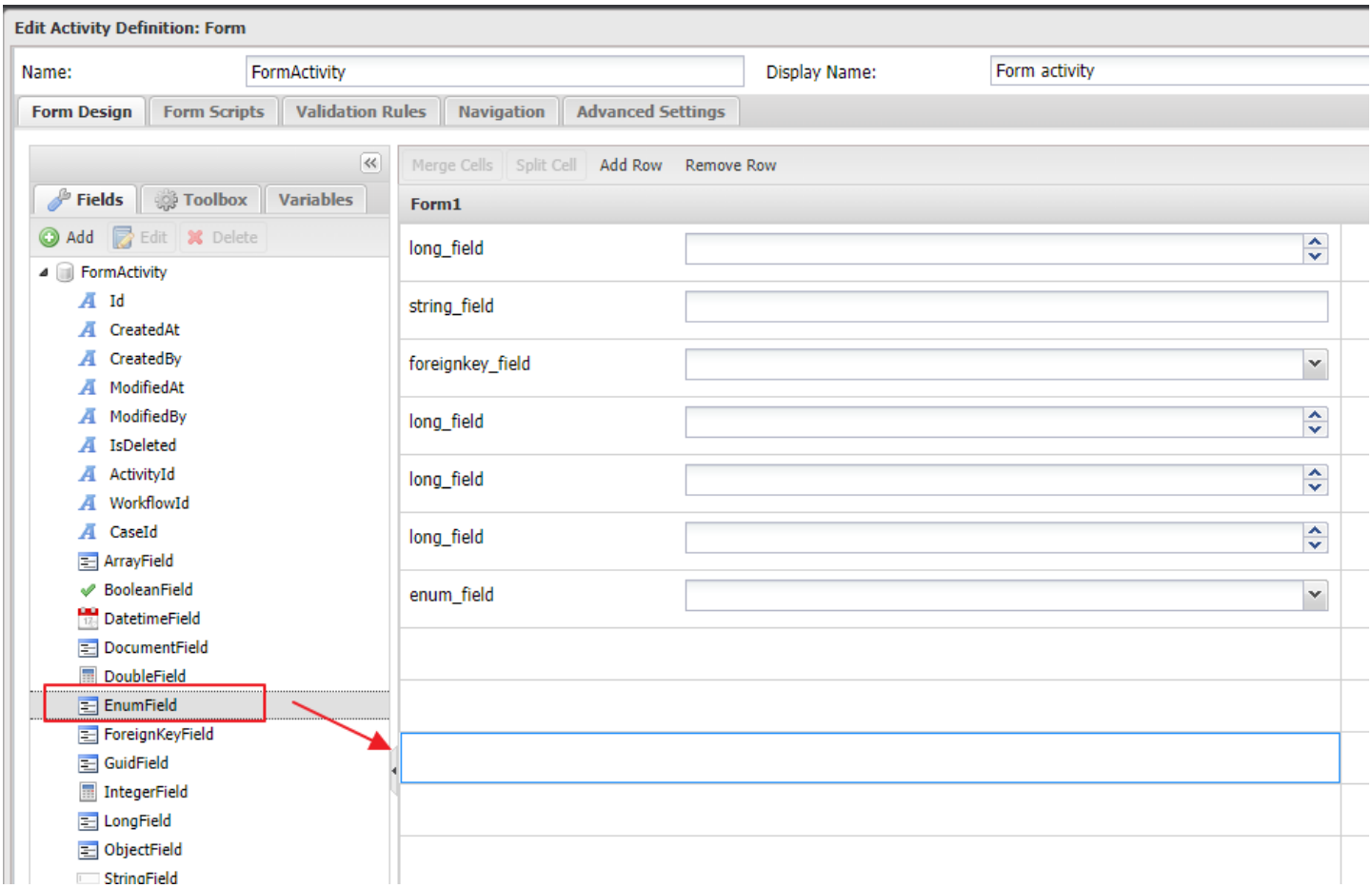


- You can also edit or delete the added data source by selecting a data source and clicking the **Edit** or **Delete** button.

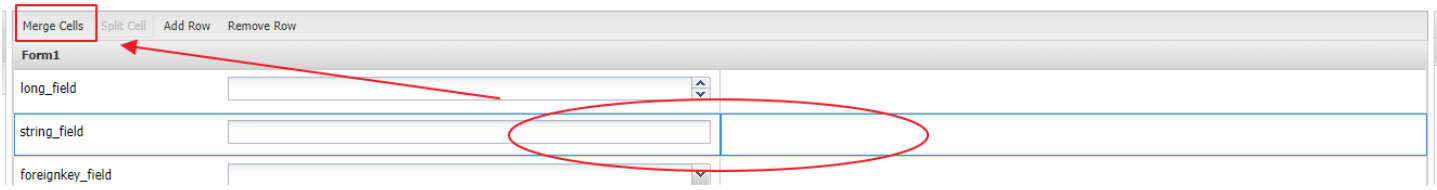


Step 2 : Add fields to the form and configure them.

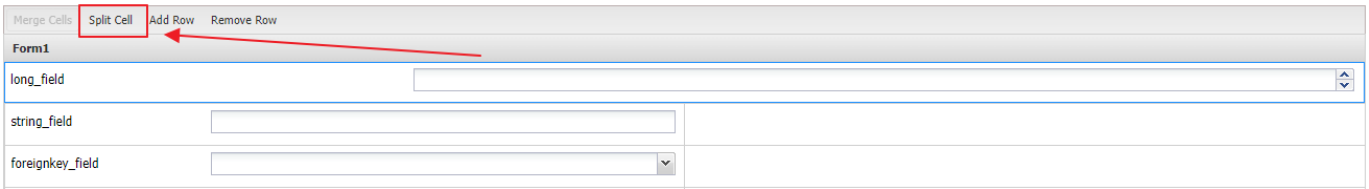
- In the data source list, click on the field you want to add to the form. Then, drag the field to the form area and drop it into the desired cell. You can also reposition the field by dragging and dropping it into another cell. Additionally, multiple fields can be added to a single cell.



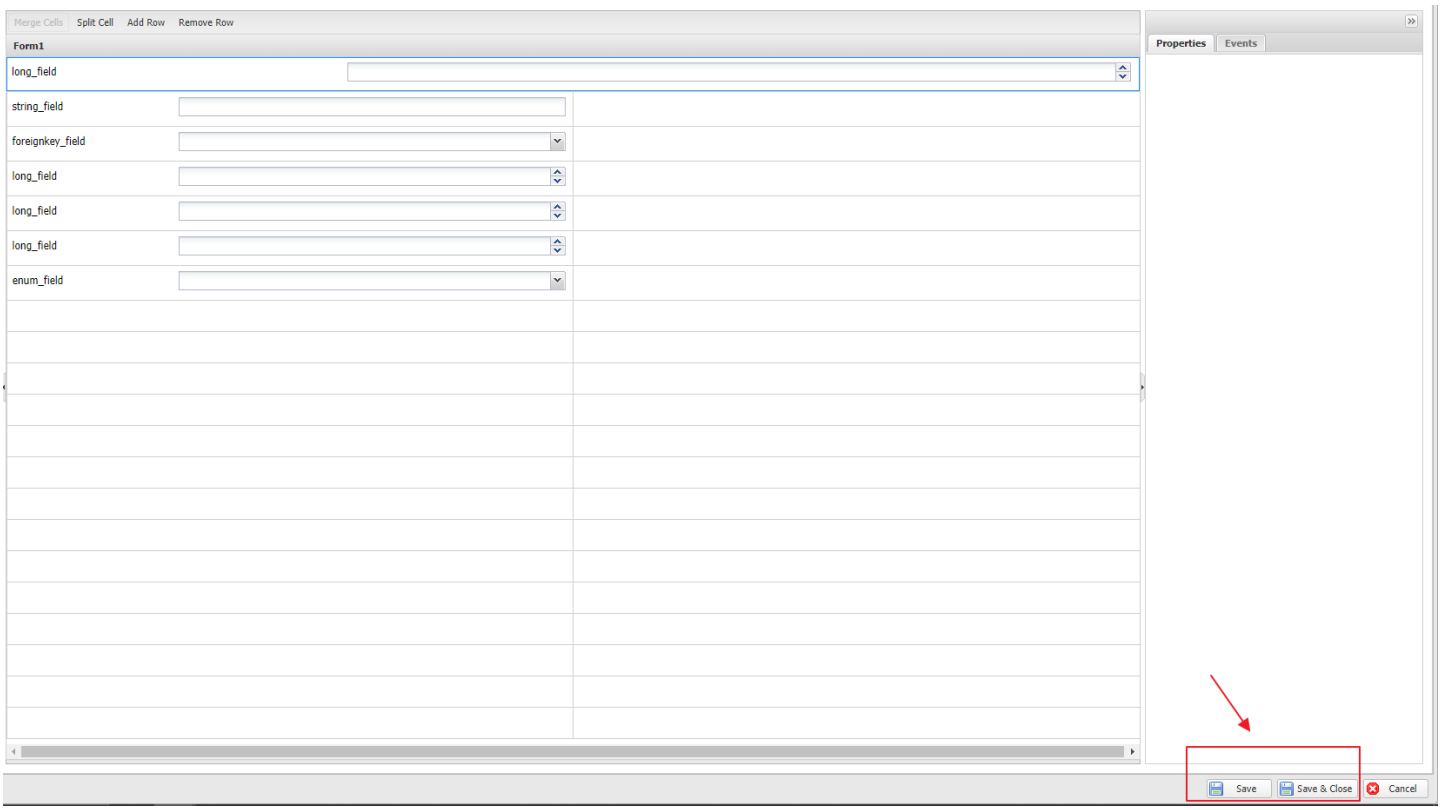
- To delete an added field from the form, click on the field and press the **Delete** key on your keyboard. To delete all fields in a row, click on a cell in the row and select **Remove row**.
- You can also merge two cells horizontally by selecting them in a row and clicking on **Merge Cells**. To select two cells, hold the **Shift** key while clicking.



- To split a merged cell, select it and click on **Split Cell**.



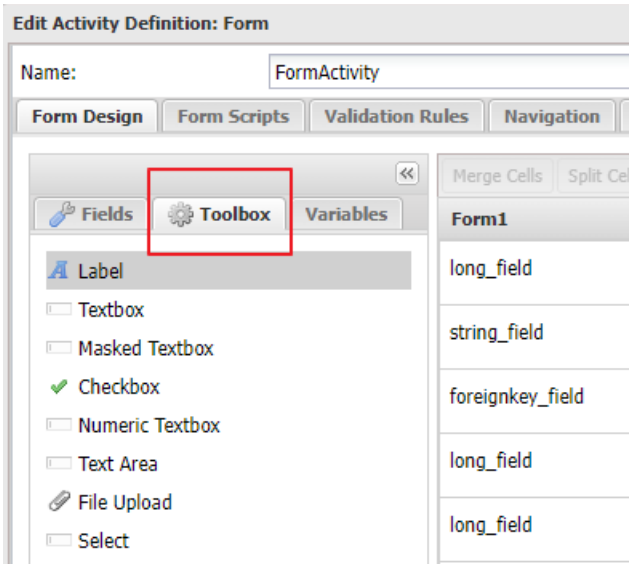
Step 3 : Save the form by clicking the **Save** or **Save and Close** button. Your changes will be saved.



Add fields from toolbox

In addition to adding fields from the data source, you can also add fields from the **Toolbox**. The **Toolbox** is a list of components that you can directly add to the form.

Step 1: Open **Toolbox** list.



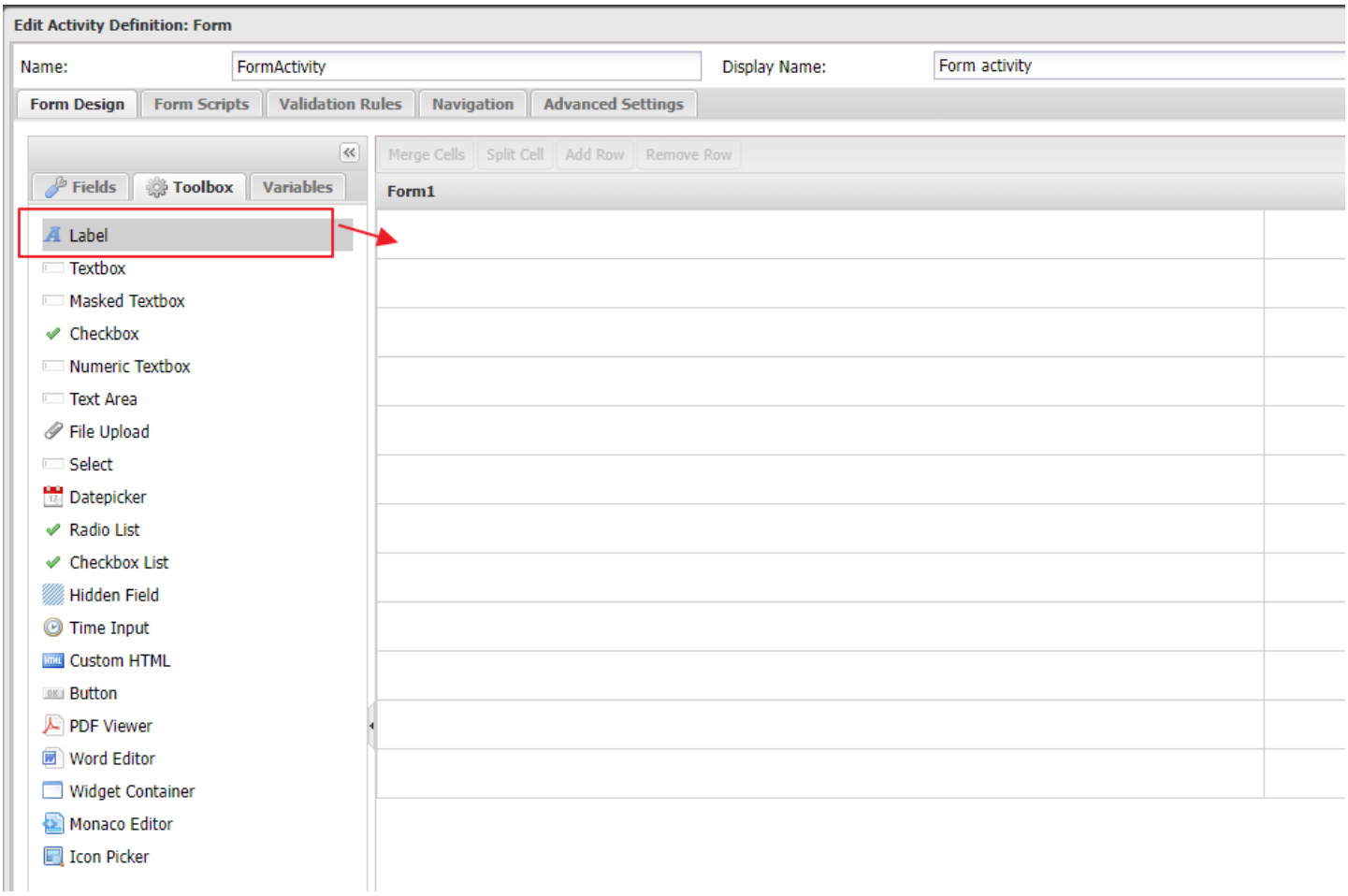
Click on the **Toolbox** tab, located next to the **Fields** tab, to open the **Toolbox** list. The components in this list include:

COMPONENT NAME	DESCRIPTION
Label	To add a label element
Textbox	To add a textbox field
Masked Textbox	To add a textbox field with specific validation of input values

COMPONENT NAME	DESCRIPTION
Checkbox	To add a checkbox field
Numeric Textbox	To add a textbox field that allows to input numeric values only
Text Area	To add a text-area field
File Upload	To add a file upload field
Select	To add a drop-down list field
Datepicker	To add a date time field
Radio List	To add a radio button field with multiple options
Checkbox List	To add a checkbox field with multiple options
Hidden Field	To add a hidden field
Time Input	To add a time selection field
Custom HTML	To add a field which you can customize HTML
Button	To add a button field
PDF Viewer	To add a PDF Viewer field
Word Editor	To add a Word Editor field
Widget Container	To add a widget container which you can create a new widget or reuse another widget
Monaco Editor	To add Monaco editor
Icon Picker	To add an icon picker field with pre-set icons.

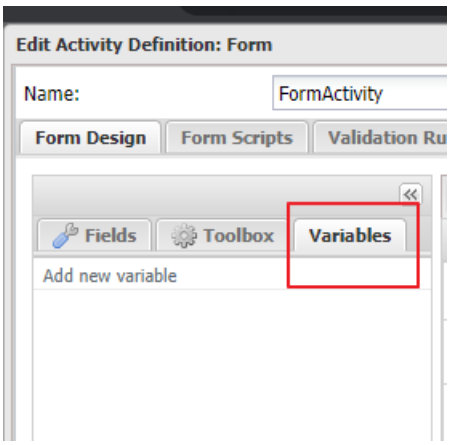
Step 2 : Add component to the form.

Click on a field in the **Toolbox** list, drag it to the form area, and drop it into the desired cell.

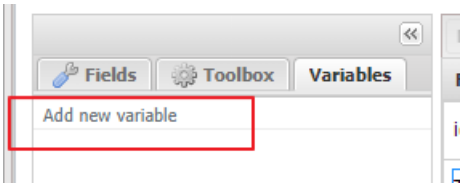


Add workflow variables

Step 1: Open **Variables** tab.

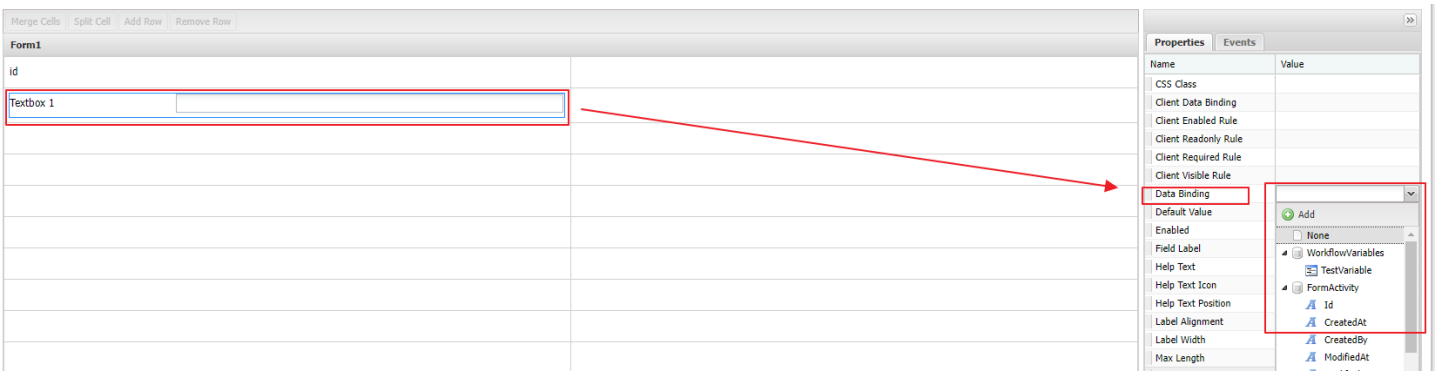


Step 2: Click on **Add new variable** field.



Step 3: Enter the variable name and press the **Enter** key to add it.

Step 4: Select an added field in the form area, then choose the **Properties** menu item. In the **Properties** list, expand the **Data Binding** section and select the variable.



Field properties configuration

After adding a field to the form, select the field to open the **Properties** menu on the right side of the form area. You can configure specific properties for the field. The available properties are:

PROPERTY NAME	DESCRIPTION
CSS Class	It is used to specify the name of a CSS class applied to the field's UI. This can be a CSS class from a component in the Worker site, a theme, or the Bootstrap library.
Client Data Binding	Use this to bind data to a variable on the client side.
Client Enabled Rule	Configure a client-side rule to determine whether the field is enabled or disabled.
Client Readonly Rule	Configure a client-side rule to determine whether the field is readonly or not.
Client Required Rule	Configure a client-side rule to determine whether the field is required or optional.
Client Visible Rule	Configure a client-side rule to determine whether the field is visible or hidden.
Data Binding	Use this to bind a data-source field or a workflow variable to the field.
Default Value	Use this to set the default value of the field in the UI.
Enabled	If you choose **true**, the field will always be enabled. If you choose **false**, the field will always be disabled. You can also customize the field's enabled state by selecting **Custom** and adding an expression.
Field Label	The field label accepts a text resource key.
Help Text	The help content provides a more detailed explanation of the field.
Help Text Icon	The icon indicating help text. For example, `fas fa-question-circle`.

PROPERTY NAME	DESCRIPTION
Help Text Position	To determine the position of the help icon. The help text appears when you hover over the help text icon. - **Label**: The icon is displayed right next to the field label. - **Input**: The icon is displayed right next to the field input. - **Inline**: The help content is displayed inline.
Label Alignment	Determines the alignment of the label with the field input. The available options are: **Left**, **Right**, and **Top**.
Label Width	The width of the field label.
Name	The field name, which can be referenced in scripts.
Min value	Sets the minimum value for the field. The user's input cannot be less than this value.
Max value	Sets the maximum value for the field. The user's input cannot be greater than this value.
Placeholder	Use this to provide inline explanatory text for the field.
Range message	Used to define the error message when the input value does not fall within the min/max range.
Readonly	If you select **True**: The field is always read-only. If you select **False**: The field is always readable and editable. If you select **Custom**: You can define specific rules for reading or editing the field.
Required	If you select **True**: The field is mandatory. If you select **False**: The field is optional. If you select **Custom**: You can define specific rules for the field's mandatory status.
Required message	Used to define the error message when the required field value is missing.

Field events configuration

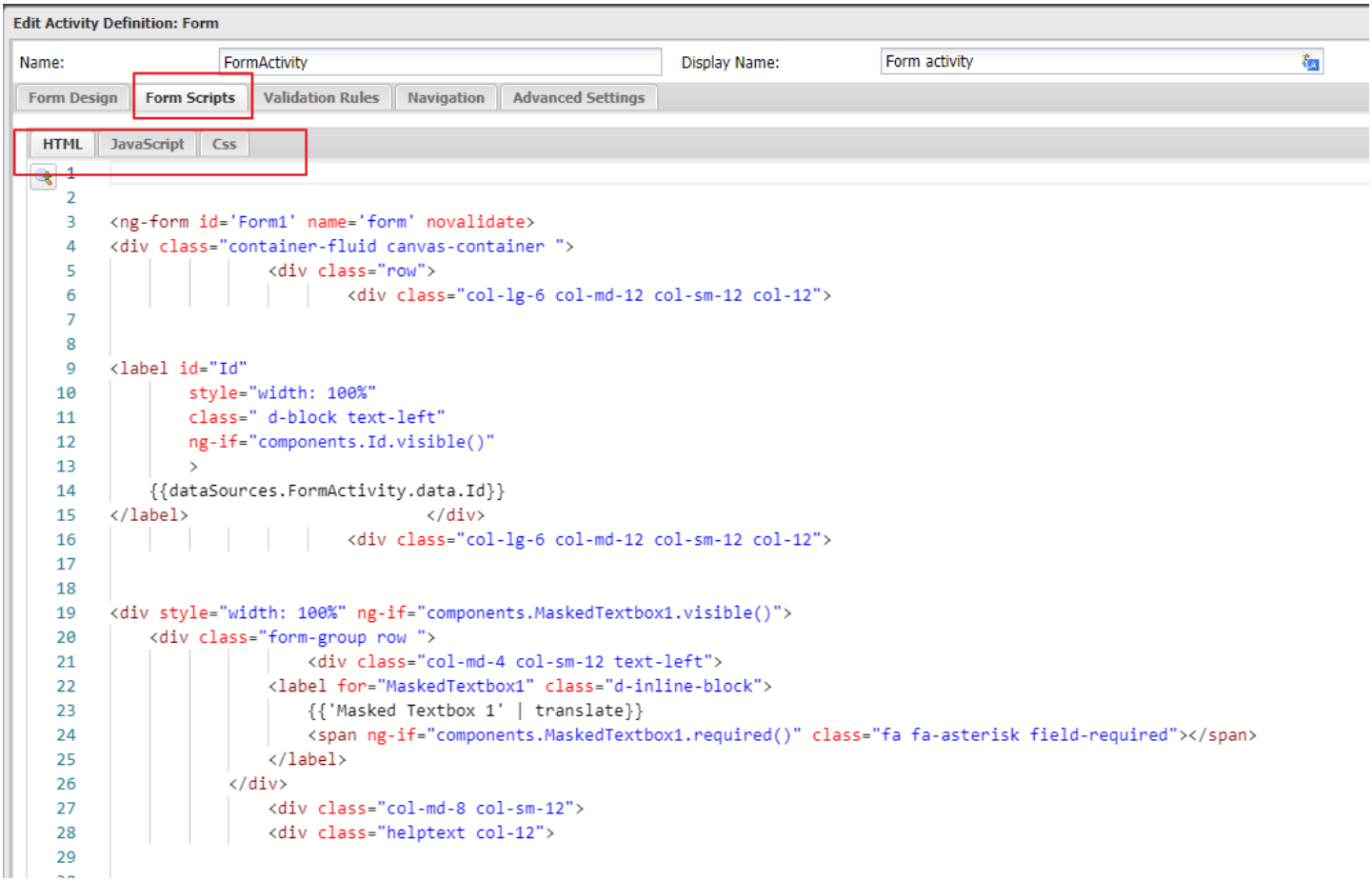
In Properties & Events section, by clicking on **Events** tab, list of events will be displayed. You can configure action of an event for the field. Those events are:

EVENT NAME	DESCRIPTION
Blur	Define the event handler for when the "Blur" event.
Change	Define the event handler for when the "Change" event.
Click	Define the event handler for when the "Click" event.
DbClick	Define the event handler for when the "DbClick" event.
Focus	Define the event handler for when the "Focus" event.
KeyDown	Define the event handler for when the "KeyDown" event.
KeyPress	Define the event handler for when the "KeyPress" event.

EVENT NAME	DESCRIPTION
KeyUp	Define the event handler for when the "KeyUp" event.
MouseDown	Define the event handler for when the "MouseDown" event.
MouseEnter	Define the event handler for when the "MouseEnter" event.
MouseLeave	Define the event handler for when the "MouseLeave" event.
MouseMove	Define the event handler for when the "MouseMove" event.
MouseOver	Define the event handler for when the "MouseOver" event.
MouseUp	Define the event handler for when the "MouseUp" event.

Form Scripts

Form Scripts is the section where you can view the executable scripts generated for the form. The script is read-only, and Casewhere will automatically regenerate the script whenever you modify the form design.

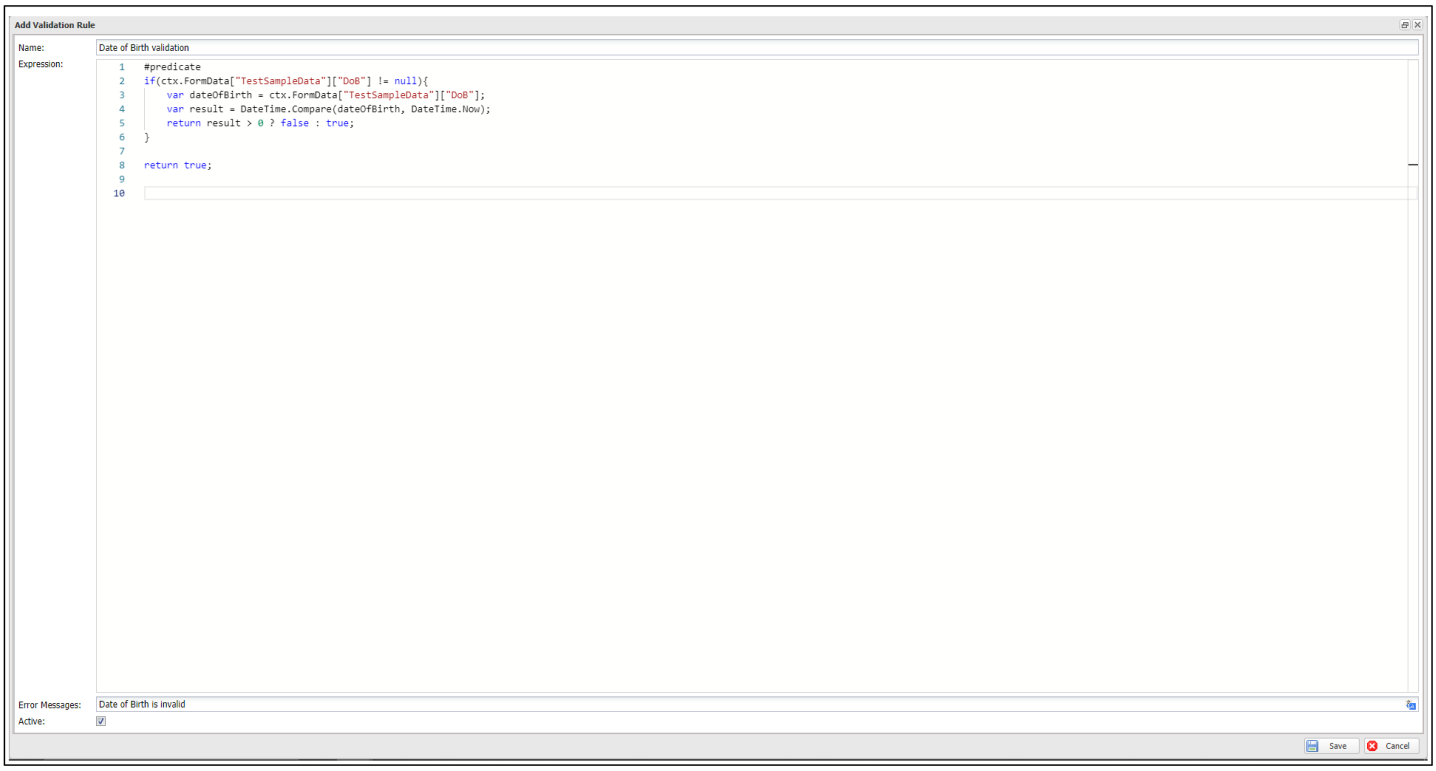


Validation Rules

Validation Rules is a section in the **Form Activity** dialog where you can create and edit validation rules.

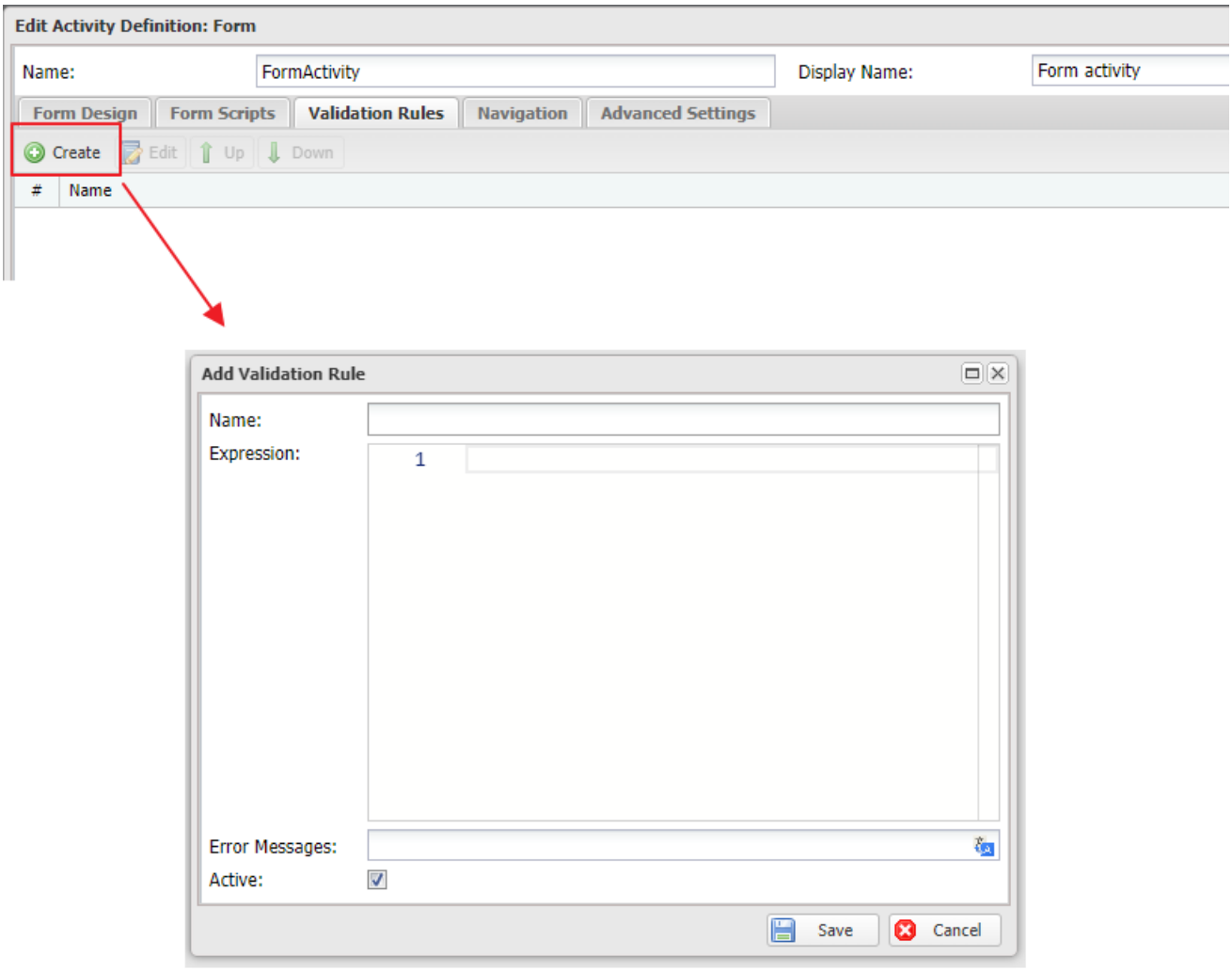


For example, the following is a validation rule for the user's date of birth:



Create a new Validation Rule

Clicking the **Create** button opens a dialog to add a validation rule:

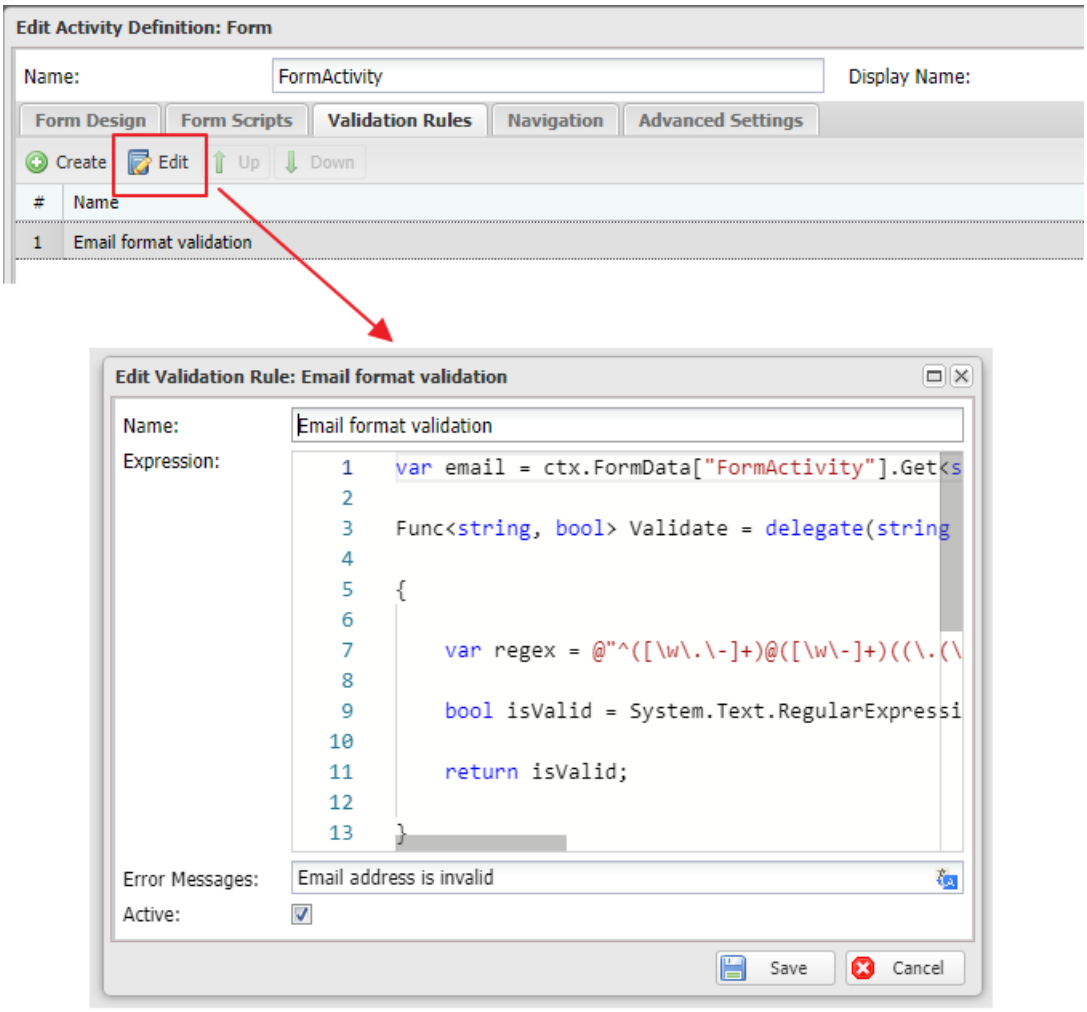


In this dialog, there are fields:

FIELD NAME	DESCRIPTION
Name	Is used to fill name of the validation rule
Expression	Is used to fill expression of the validation rule
Error Message	Is used to fill error message when field value does not match validation rule
Active	Is used to activate or deactivate the validation rule. If the checkbox is unticked, the rule will be deactivated

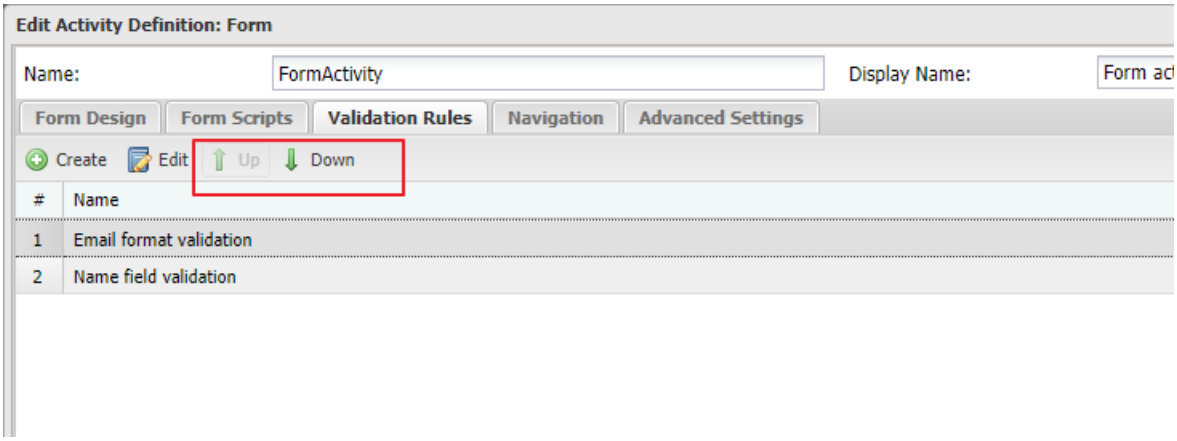
Edit a Validation Rule

By selecting an existing validation rule, then clicking on **Edit** button, the dialog of editing validation rule will be opened. You can change the name, expression, error message, active state of the validation rule.



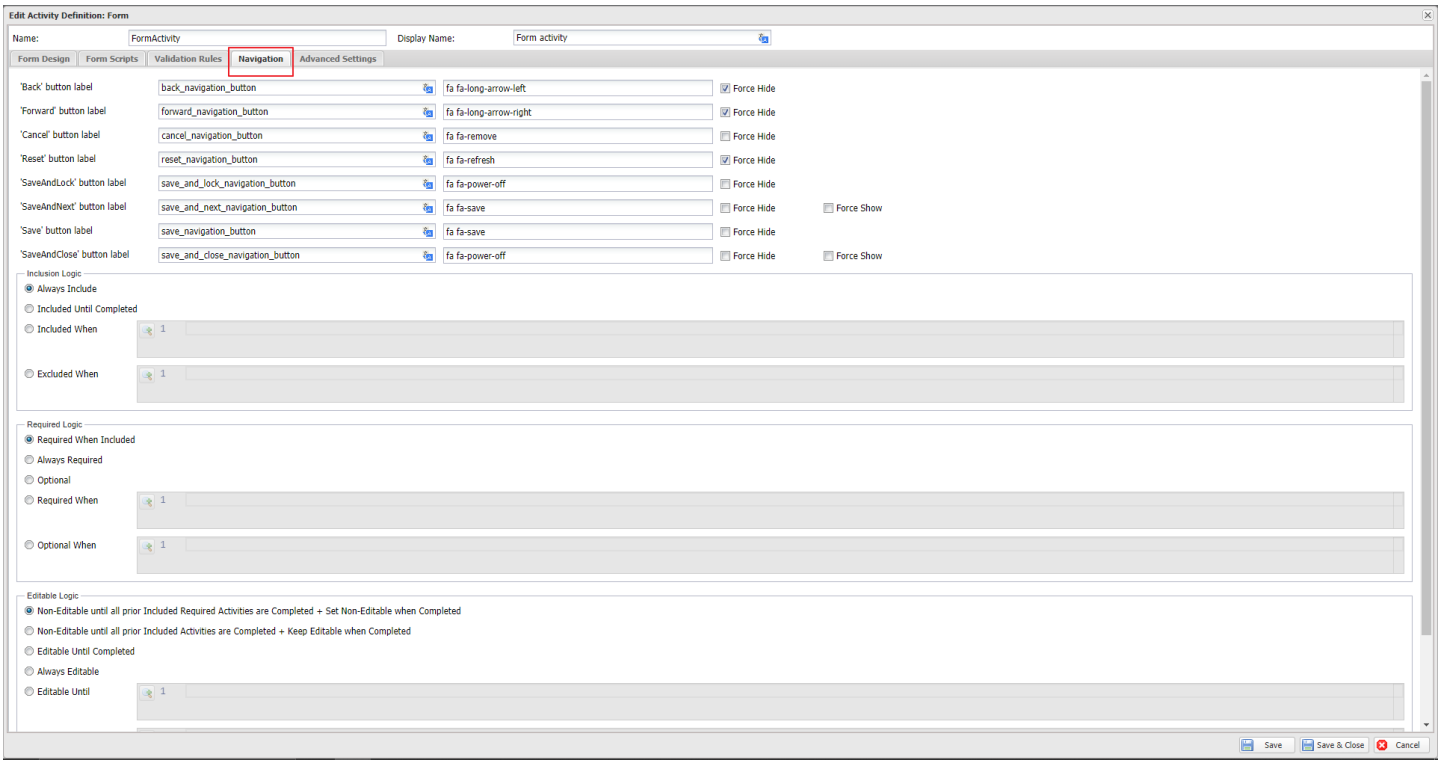
Edit position of a validation rule in the list

You can use **Up** and **Down** button to change the position of a validation rule in the list. First, select a rule, then click on **Up**/**Down** button move it up/down.



Navigation

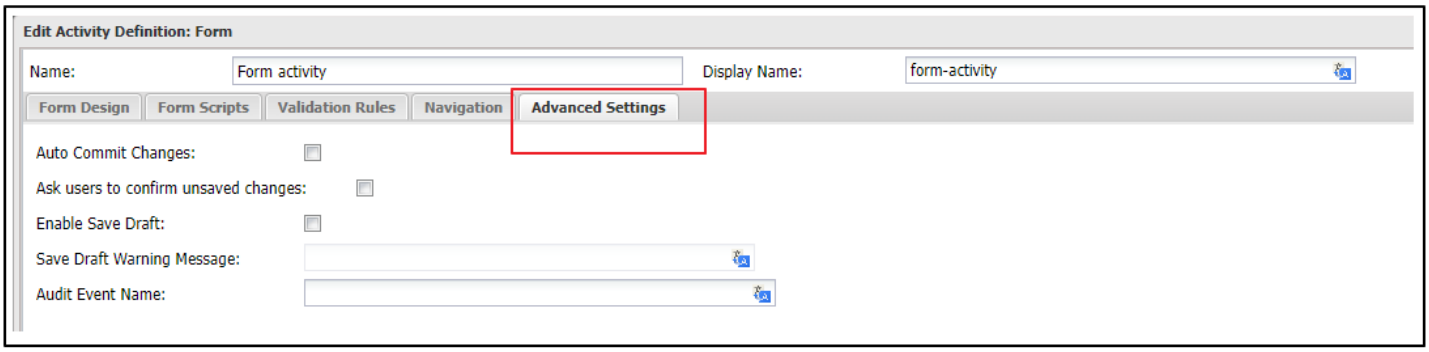
Navigation section in **Form activity** is where you can configure the navigation logic for your activity.



More details can be found in: Workflow-navigation

Advanced Settings

Advanced Settings is the section in the **Form activity** where you can configure some additional advanced settings for the form.



SETTINGS NAME	DESCRIPTION
Auto Commit Changes	If this checkbox is selected, Casewhere will save all unsaved data to the database after the activity is completed. By default, Casewhere only saves data when the entire workflow is finished.
Audit Event Name	Starting from version 2.7, Casewhere automatically audits all events that alter the data state. By default, the activity name is used for auditing, but you can customize it using this setting.
Enable Exclusive Lock	Casewhere enables you to control concurrency at the activity level using an exclusive lock. To learn more, refer to the article on "Workflow Concurrency Control".

SETTINGS NAME	DESCRIPTION
Lock Context (optional)	When the exclusive lock is enabled, Casewhere allows only one activity to execute at a time across the entire system. This may not be ideal for many scenarios. By providing a lock context, you can control concurrency more granularly. For example, you can enable the lock on specific data objects, allowing multiple activities to run as long as they target different objects. The setting accepts an expression that uses the same syntax as the navigation rule.
Waiting Timeout	Casewhere will release the lock if it exceeds the configured timeout, even if the activity is not yet complete.

Once all configurations are complete, clicking **Save** will save all the changes.

Form validation

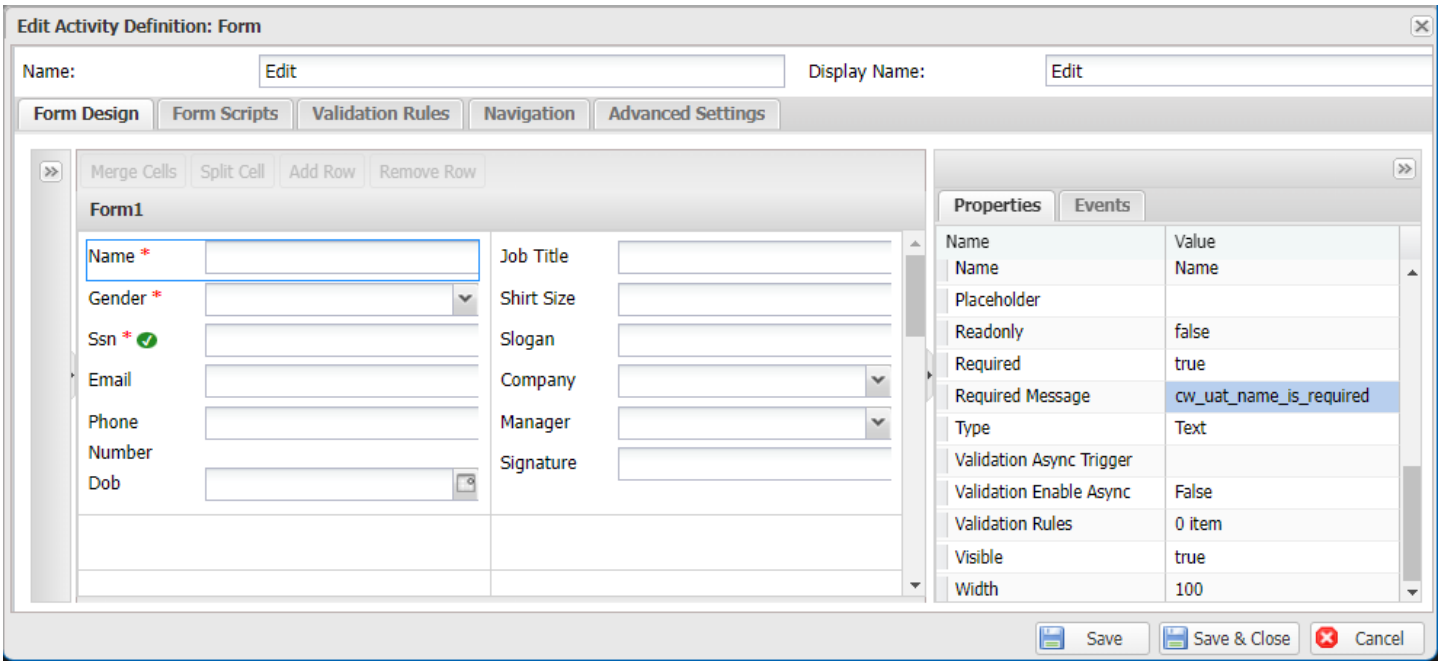
Validating data is always the most critical step in every business workflow, especially where it requires user interaction. Understanding the challenge, Casewhere offers many validation methods that work out of the box for developers while still giving them the flexibility to implement the most complex business requirements.

Basic validation configuration

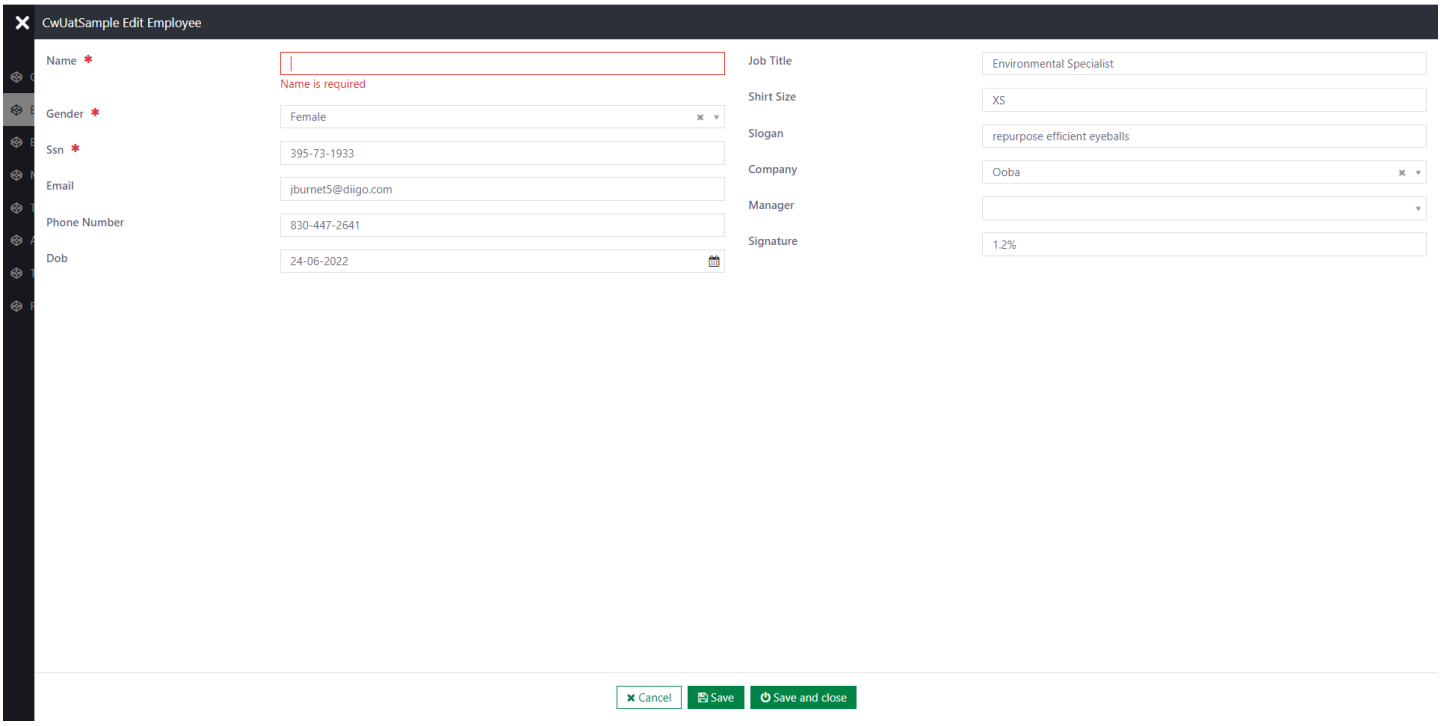
Required validation

All form component has a property **Required**. When the property is enabled, users must enter a value in order to submit the form successfully. It's possible to customized the **Required Message**, which will be displayed to the end users. The **Required Message** accepts a text resource key.

The below example shows how the property is configured in Casewhere Admin,



and here is how the validation works for end users in worker sites.



Component-specific validation

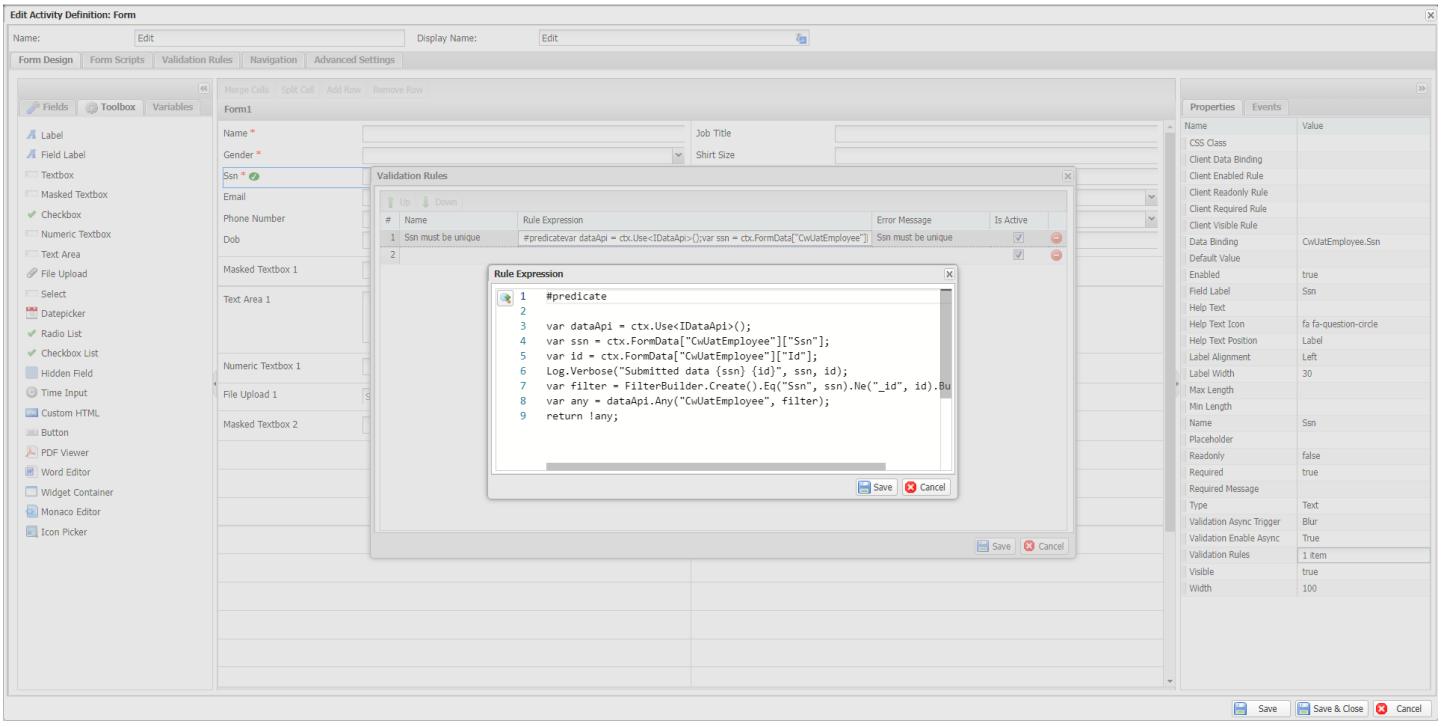
Each form component has a different set validation properties. Below are some typical components and their validation properties you might usually work with.

COMPONENT	PROPERTY	DESCRIPTION
Textbox	Max Length	The maximum number of characters allowed
Textbox	Min Length	The minimum number of characters required in an input field
Numeric Textbox	Max Value	The maximum value
Numeric Textbox	Min Value	The minimum value
Datepicker	Max Date	The maximum value
Datepicker	Min Date	The minimum value
File Upload	File Extensions	The acceptable file extensions
File Upload	File Size	The maximum size in MB allowed to be uploaded
Masked Textbox	Mask	The pattern of input

Component validation rules

> Since version `2.7.2`, we recommend configuring validation rules at the component-level instead of managing them at the form-level in previous versions.

Casewhere allows configuring validation rules for form components. Select the component you want to validate, in the **Properties** tab pane, select **Validation Rules**. It's possible to add multiple validation rules for a component. Components having validation rules will have a green tick icon in the form editor.



A validation rule, basically, is a boolean expression. The validation is passed if the rule returns `true`. The rule accepts the `@ctx` keyword to access the workflow data. Read more about @ctx here. You can learn more about writing rule here.

Example: Verify if users enter a unique Ssn

```
#predicate

var dataApi = ctx.Use<IDataApi>();

var ssn = ctx.FormData["CwUatEmployee"]["Ssn"];
var id = ctx.FormData["CwUatEmployee"]["Id"];

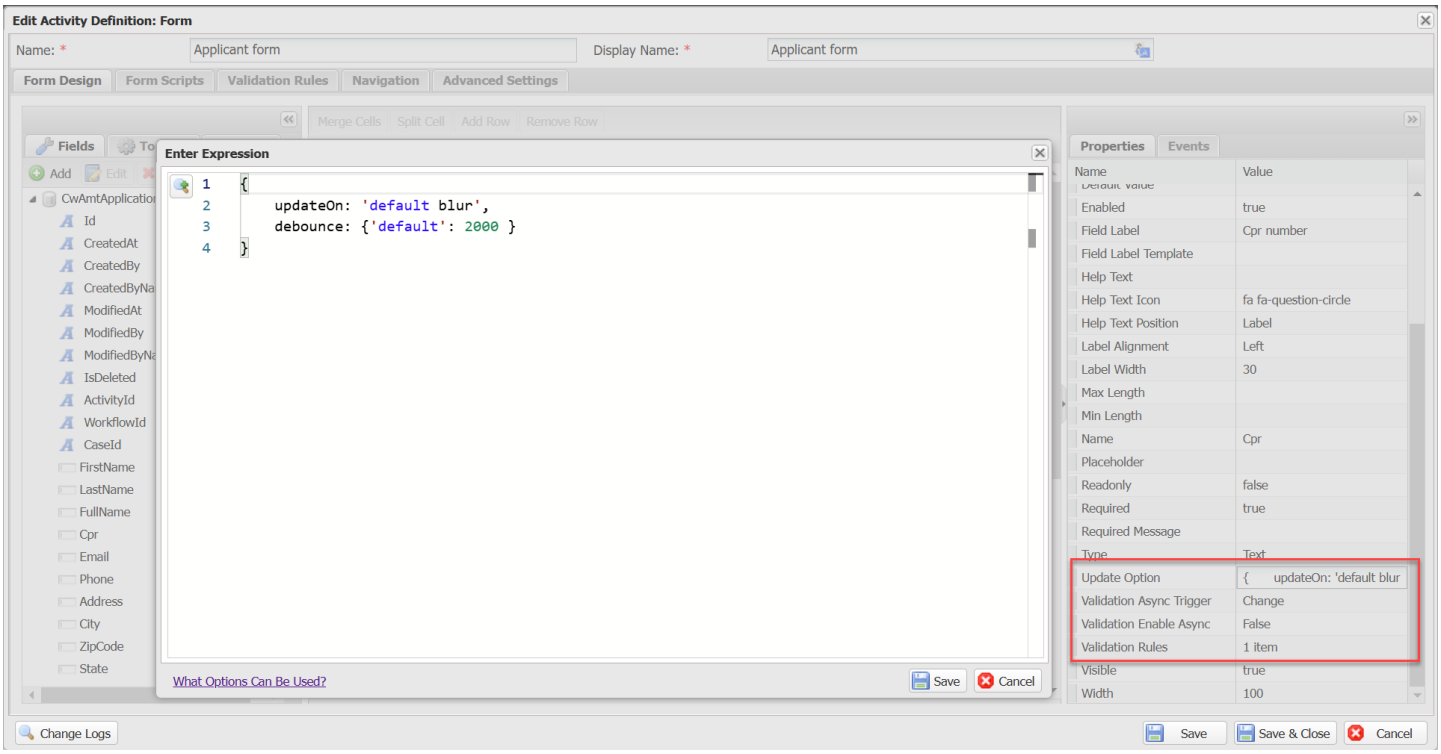
var filter = FilterBuilder.Create().Eq("Ssn", ssn).Ne("_id", id).Build();
var any = dataApi.Any("CwUatEmployee", filter);
return !any;
```

Instant validation

By default, Casewhere only validates when users submit data, i.e., click the save button in the workflow wizard. Many applications trigger validation right after users input data to improve user experience. Understanding the need, Casewhere supports instant validation that triggers validations based on various component events by utilizing the same validation rules.

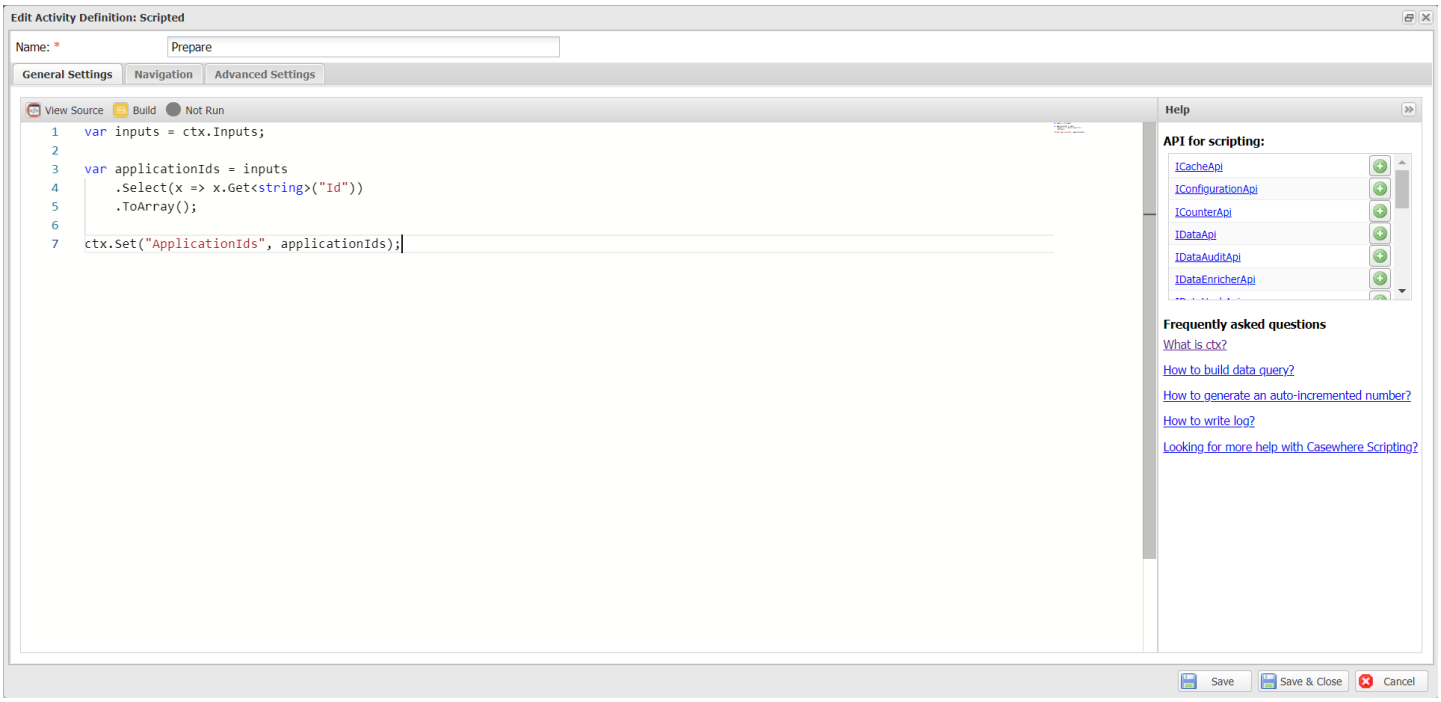
To enable instant validation, set **Validation Enable Async** to **True** and then specify a value for the property **Validation Async Trigger**. For example, if **Blur** is selected, when users focus out the component, Casewhere will immediately trigger the data validation on that component. An error message will be displayed if the validation fails, just like when submitting form data.

Please note that sending excessive validation requests while users are typing can be frustrating for end users and place unnecessary strain on the server. To address this, it's recommended to implement a debounce for value updates using the **Update Option** setting.



Scripted Activity

Casewhere Scripted Activity enables the execution of custom C# scripts within the workflow context. It offers a comprehensive range of APIs to support application development, including functionalities for working with data objects, evaluating rules, invoking plugins, and more. For detailed information about these APIs, consult the help links available in the editor.



In this article, we will explore some commonly used techniques in **Scripted Actitivity**.

Workflow variables

Persisted variables

Workflow persisted variables are stored in the database as part of the workflow data.

Setting a workflow variable: Workflow variables can accept any data type.

```
ctx.Set("Confirmed", true);
ctx.Set("Error", "Invalid data");
ctx.Set("Count", 999);
ctx.Set("SomeDataObject", aDataObject);
```

Getting a workflow variable: A workflow variable can be accessed throughout the workflow where it is created, including in scripted activities, navigation rules, validation rules, and more.

```
var isConfirmed = ctx.Get<bool>("Confirmed");
var error = ctx.Get<string>("Error");
var count = ctx.Get<int>("Count");
var aDataObject = ctx.Set("SomeDataObject"); // Read an object
```

Determine if a variable is created or not: The check disregards the variable's value. For instance, `ctx.Has` will return `true` even if the variable's value is `null` or empty.

```
if (ctx.Has("Error")) Log.Error("An error occurred.");
```

Determine if a variable has a value or not: The check considers the variable's value and returns `true` only when the value is neither null nor empty.

```
// A log event is created only if the error is not null or empty.
if (ctx.HasValue("Error")) Log.Error("An error occurred.");
```

Temporary variables

Storing large objects in workflow variables is not ideal, especially if there is no need to persist them, such as for display on the UI. To optimize performance, Casewhere introduces temporary variables for storing and accessing data in memory only. Temporary data will be discarded when Casewhere persists the workflow to the database.

Setting a temporary variable

```
ctx.SetTemp("Confirmed", true);
ctx.SetTemp("Error", "Invalid data");
ctx.SetTemp("Count", 999);
ctx.SetTemp("SomeDataObject", aDataObject);
```

Getting a temporary variable

```
var isConfirmed = ctx.GetTemp<bool>("Confirmed");
var error = ctx.GetTemp<string>("Error");
var count = ctx.GetTemp<int>("Count");
var aDataObject = ctx.GetTemp<DynamicDataObject>("SomeDataObject"); // Read an object
```

Determine if a temporary variable is created or not: The check disregards the variable's value. For instance, `ctx.HasTempKey` will return `true` even if the variable's value is `null` or empty.

```
if (ctx.HasTempKey("Error")) Log.Error("An error occurred.");
```

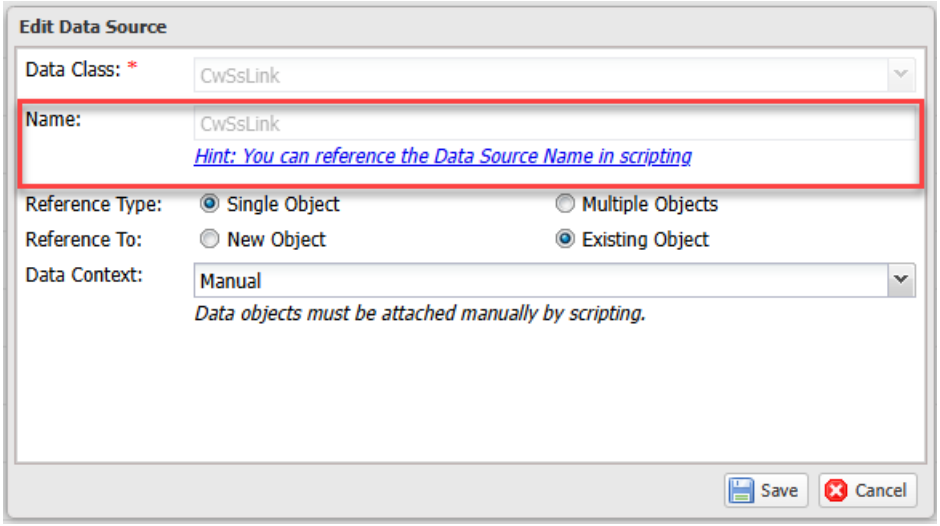
Data object references

You can create a variable to store a reference to a data object and later use that reference to access the object. Casewhere stores only the data object ID and always fetches the latest data when accessing the referenced object.

```
// Activity 1: Set reference to newly created data objects
var applicationId = dataApi.Add("Application", new { ... });
ctx.Ref("Application", applicationId);

// Activity 2: Read the referenced data object
var application = ctx.Ref("Application");
Log.Info("Created application {@application}", application);
```

Access form data: Please note that Casewhere automatically creates references for data objects generated in form activities. You can access them using the form data source names.



It's also possible in **Scripted Activity** to set the data reference for **Form Activity**. This can be useful if you have a form with complex logic of data loading.

```
// Some complex validation before setting form data sources
if (!link.Get<bool>("IsActive"))
{
    ctx.Set("Error", "cwss_error_not_found");
    return null;
}

if (link.Get<bool>("RequiresLogin") && ctx.User.Has("IsAnonymousUser"))
{
    ctx.Set("Error", "cwss_error_requires_login");
    return null;
}

if (link.Get<DateTime>("ExpiresAt") <= DateTime.UtcNow)
{
    Log.Warning("The link {linkID} is expired", linkID);
    ctx.Set("Error", "cwss_error_expired");
    return null;
}

//...
// Setting form data sources to be displayed on the UI in the next activity.
ctx.SetDataSource("CwSsLinkSecret", linkSecret["Id"]);
ctx.SetDataSource("CwSsLink", link["Id"]);
```

Working with events

You can emit custom events from a **Scripted Activity** using `ctx.Emit`.

```
ctx.Emit("DeactivateLinkEvent", new
{
    LinkId = "a_link_id"
});
```

In a event-handle workflow, you then can read data from the event:

```
var linkId = ctx.Input["Data"]["LinkId"];
dataApi.Update(linkId, new
{
    IsActive = false
});
```

Please note that the event is sent to the event queue only if the **Scripted Activity** completes without errors. If the activity crashes, Casewhere will reject all events created during its execution. To ensure that events are sent to the queue regardless of the activity's status, you can use `ctx.ForceEmit`, available since version 2.8.

Working with data objects

Create data

```
// Activity 1: Create new product with data from a plugin
var pluginApi = ctx.Use<IPluginApi>();
var dataApi = ctx.Use<IDataApi>();

var product = pluginApi.Invoke("TestDataPlugin", "GetProduct", new Dictionary<string, object>
()).GetObject("Product");

var productData = new DynamicDataObject();
productData["Name"] = product["Name"];
productData["Picture"] = product["Picture"];
productData["Material"] = product["Material"];
productData["Description"] = product["Description"];
productData["Amount"] = product["Price"];

var productId = dataApi.Add("Product", productData);
ctx.Set("productId", productId); // Store the product ID to a workflow variable
```

Update data

```
// Activity 2: Update product with data from a plugin
var pluginApi = ctx.Use<IPluginApi>();
var dataApi = ctx.Use<IDataApi>();

var product = pluginApi.Invoke("TestDataPlugin", "GetProduct", new Dictionary<string, object>
()).GetObject("Product");

var productData = new DynamicDataObject();
productData["Name"] = product["Name"];
productData["Picture"] = product["Picture"];
productData["Material"] = product["Material"];
productData["Description"] = product["Description"];
productData["Amount"] = product["Price"];

var productId = ctx.Get<string>("productId"); // Store the product ID to a workflow variable
dataApi.Update(productId, productData);
```

Delete data

```
// Delete the data object from the workflow input
var dataApi = ctx.Use<IDataApi>();
dataApi.Delete(ctx.Input.Id);
```

Read data

```
var product = ctx.Input;

var name = product.Get<string>("Name");
var price = product.Get<double>("Price");
var isActive = product.Get<bool>("IsActive");
var categories = product.GetList<string>("Categories");
```

Workflow Activity

Introduction

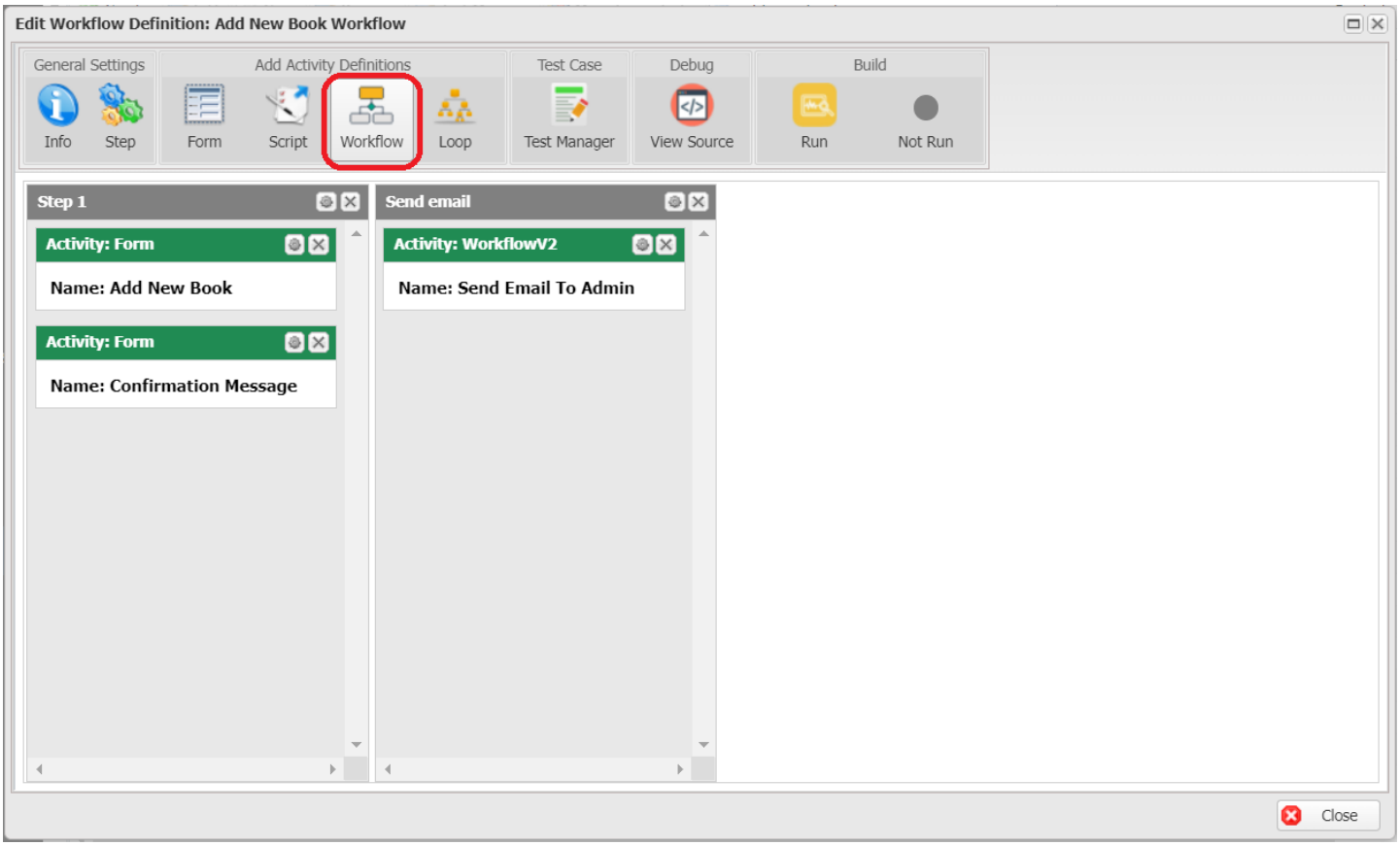
Workflow Activity is a solution designed to streamline the reuse of functionalities across different workflows. It enables us to share common forms or scripts, enhancing efficiency and consistency.

For instance, "Sending Email" is a frequently used feature that can serve various purposes, such as creating a new user account, updating an email address, or sending notifications.

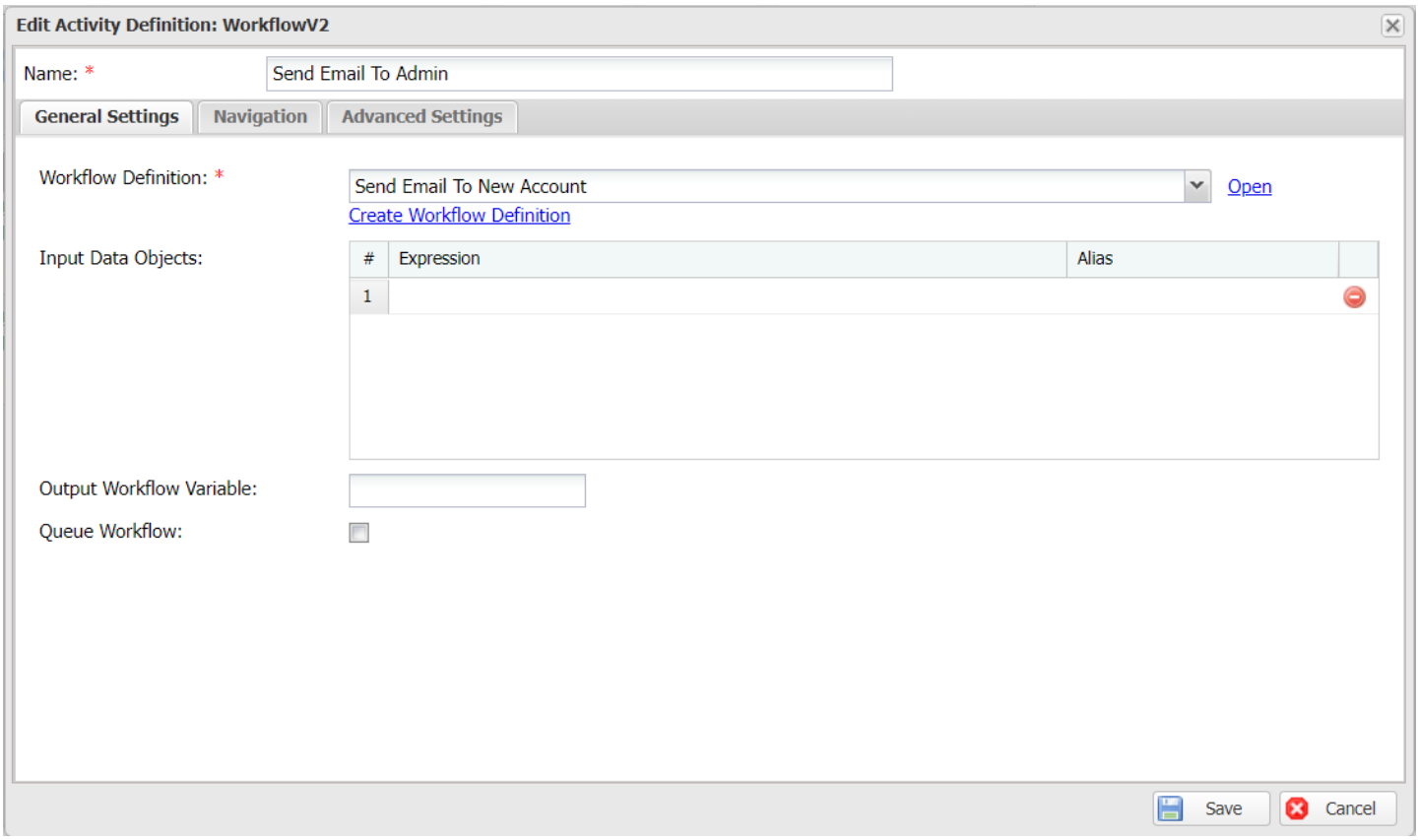
By leveraging **Workflow Activity**, we can create a dedicated workflow called "Sending Email" once and seamlessly incorporate it into other workflows as needed.

Configuration

To add a new workflow activity in Casewhere, navigate to the **Add Activity Definitions** section and drag the activity into the appropriate step of the workflow definition.



The following information is required to set up a **Workflow Activity**:

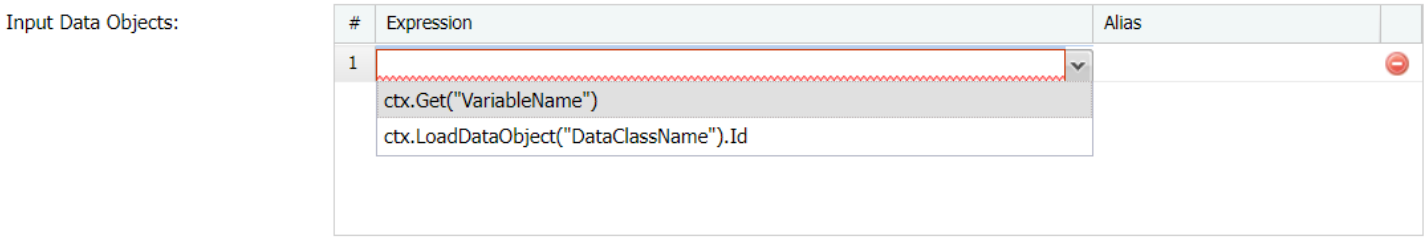


Name: The activity name must be unique and meaningful.

Workflow Definition: Select the workflow definition you want to run, also known as the "embedded workflow".

Input Data Objects: There are two options for sending data to the triggered workflow. For example:

- `ctx.Get("VariableName")`: Send data using a workflow variable.
- `ctx.Input.Id`: Send data using the workflow input.

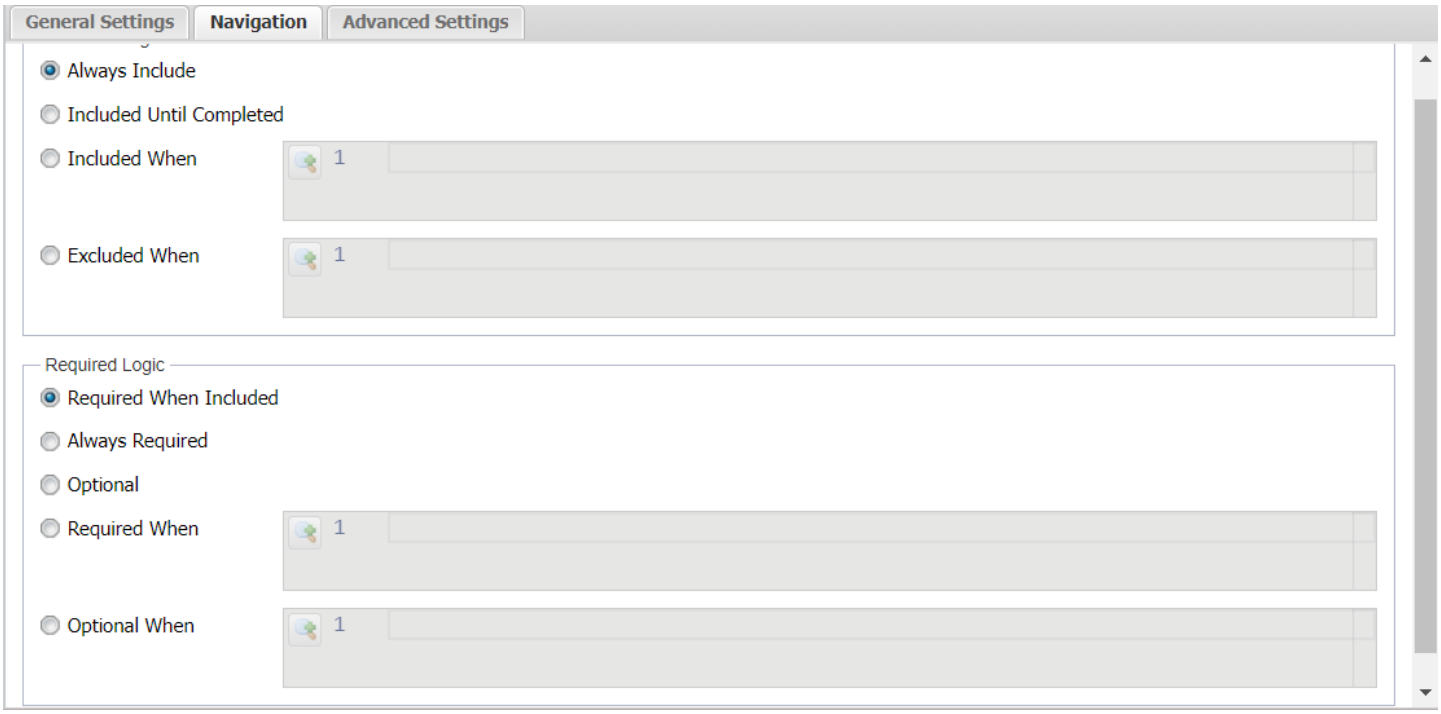


Process: A new case is created each time Casewhere executes the workflow.

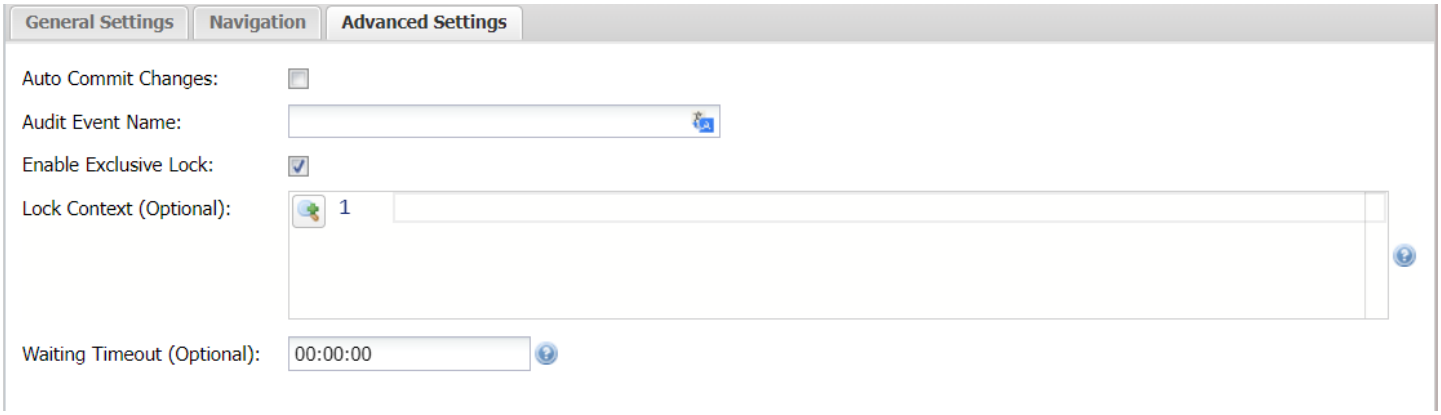
Output Workflow Variable: Defines a variable to store the value returned by the workflow.

Queue Workflow: When enabled, the workflow is queued for background execution.

Navigation: Setting for navigation between workflow activities. To learn more, read the article Activity Navigation.



Advanced Settings: Standard advanced settings applicable to all workflow activity types.

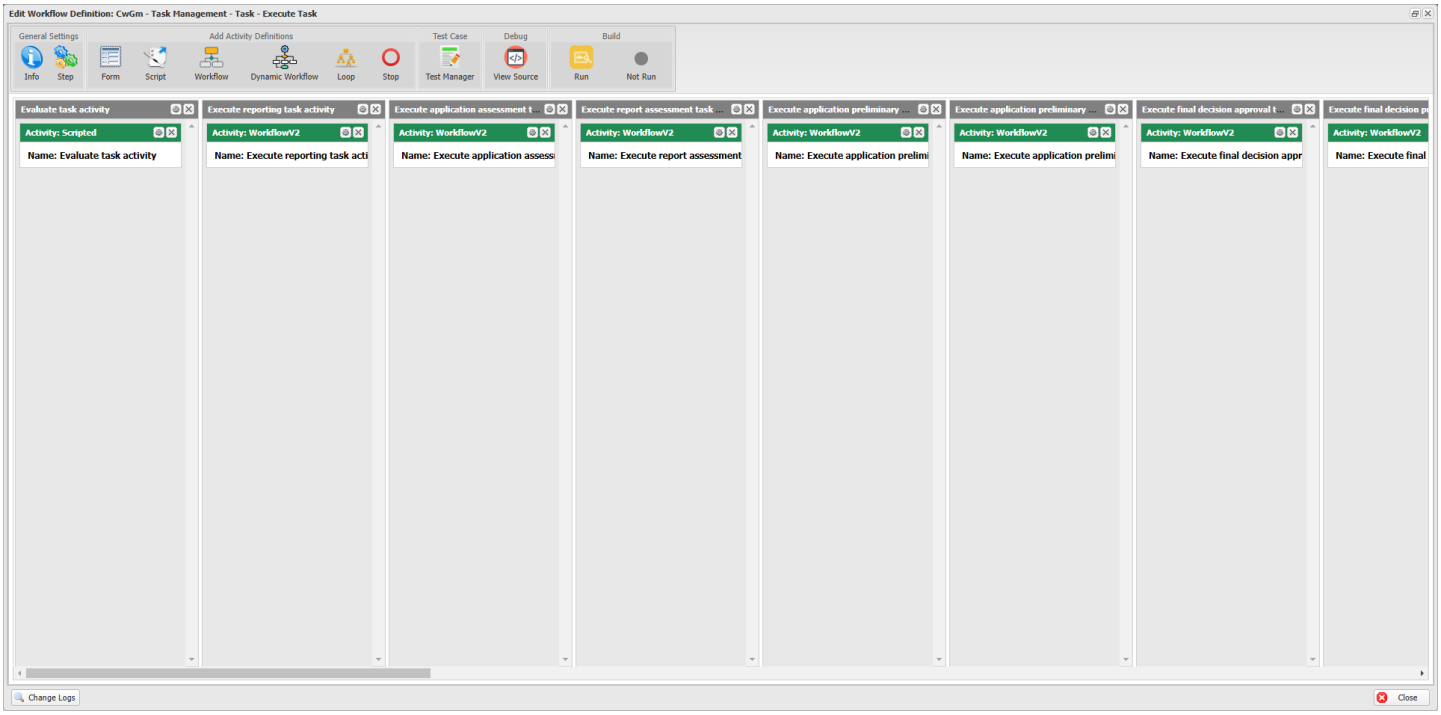


SETTINGS NAME	DESCRIPTION
Auto Commit Changes	If this checkbox is selected, Casewhere will save all unsaved data to the database after the activity is completed. By default, Casewhere only saves data when the entire workflow is finished.
Audit Event Name	Starting from version 2.7, Casewhere automatically audits all events that alter the data state. By default, the activity name is used for auditing, but you can customize it using this setting.
Enable Exclusive Lock	Casewhere enables you to control concurrency at the activity level using an exclusive lock. To learn more, refer to the article on "Workflow Concurrency Control".
Lock Context (optional)	When the exclusive lock is enabled, Casewhere allows only one activity to execute at a time across the entire system. This may not be ideal for many scenarios. By providing a lock context, you can control concurrency more granularly. For example, you can enable the lock on specific data objects, allowing multiple activities to run as long as they target different objects. The setting accepts an expression that uses the same syntax as the navigation rule.
Waiting Timeout	Casewhere will release the lock if it exceeds the configured timeout, even if the activity is not yet complete.

Dynamic Workflow Activity

The **Workflow Activity** is designed to trigger a fixed, preconfigured workflow definition. However, when the workflow to be triggered depends on specific conditions, managing multiple Workflow Activities and their inclusion logic can become inefficient—especially when the logic is complex. To address this challenge, Casewhere introduces the **Dynamic Workflow Activity**, enabling workflows to be triggered dynamically based on runtime conditions.

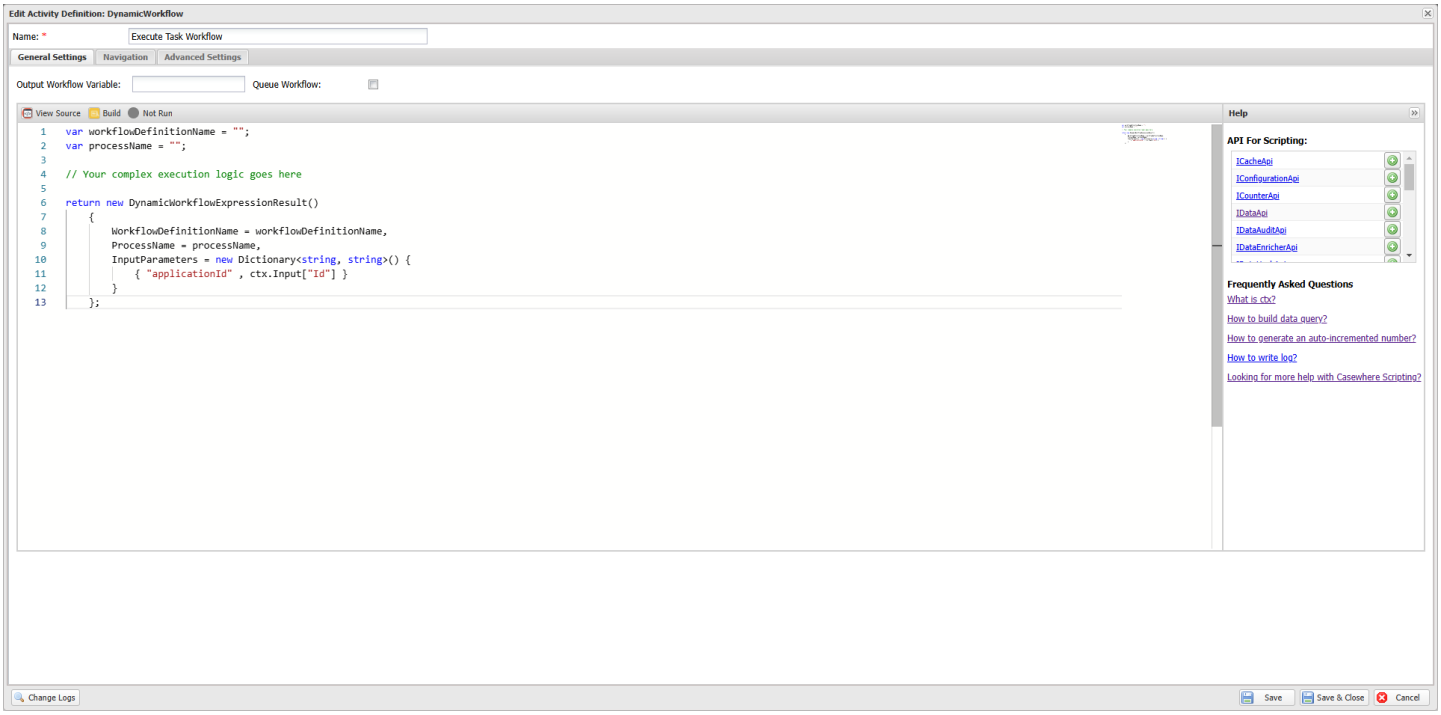
The following screenshot demonstrates how complex the setup can become without the **Dynamic Workflow Activity**.



Configuration

The **Dynamic Workflow Activity** can be added to the workflow definition just like any other workflow activity.

The activity configuration appears as follows:



Output Workflow Variable: Defines a variable to store the value returned by the workflow.

Queue Workflow: When enabled, the workflow is queued for background execution.

- **Priority**: Used to determine the execution order, applicable only to queued workflows.

Script: A script that executes custom logic to determine the workflow definition, process and input, returning a `DynamicWorkflowExpressionResult`. Casewhere will use this information to trigger the corresponding workflow.

```
var workflowDefinitionName = "CwAmt Runtime Approve Application";
var processName = "CwAmtApplicationProcess";

// Your complex execution logic goes here

return new DynamicWorkflowExpressionResult()
    {
        WorkflowDefinitionName = workflowDefinitionName,
        ProcessName = processName, // No need to specify the "ProcessName" if the triggered workflow
definition is "Edit-Case"
        InputParameters = new Dictionary<string, string>() {
            { "applicationId" , ctx.Input["Id"] }
        }
    };
```

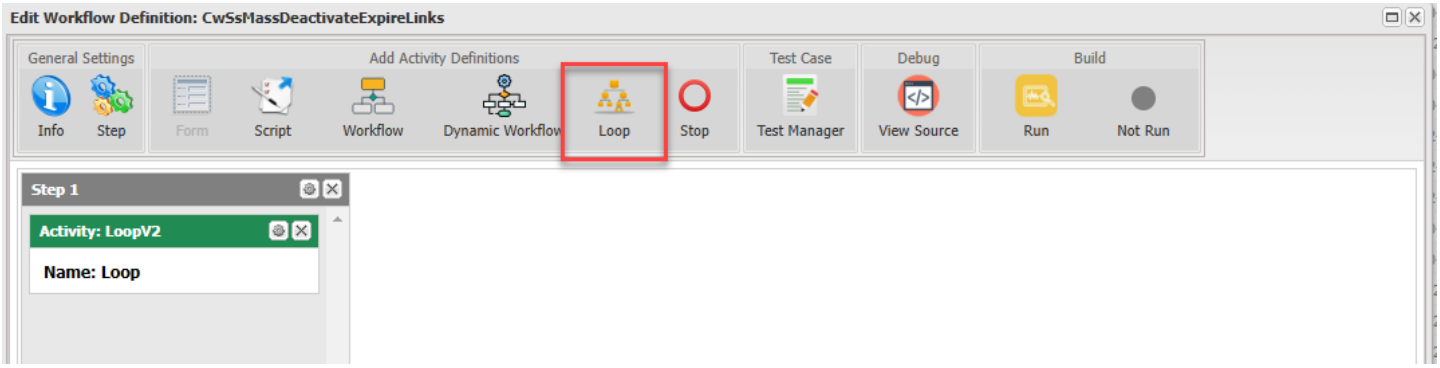
In the example above, the triggered workflow can access the passed data using `ctx.Input` or `ctx.Ref("applicationId")`.

Loop Activity

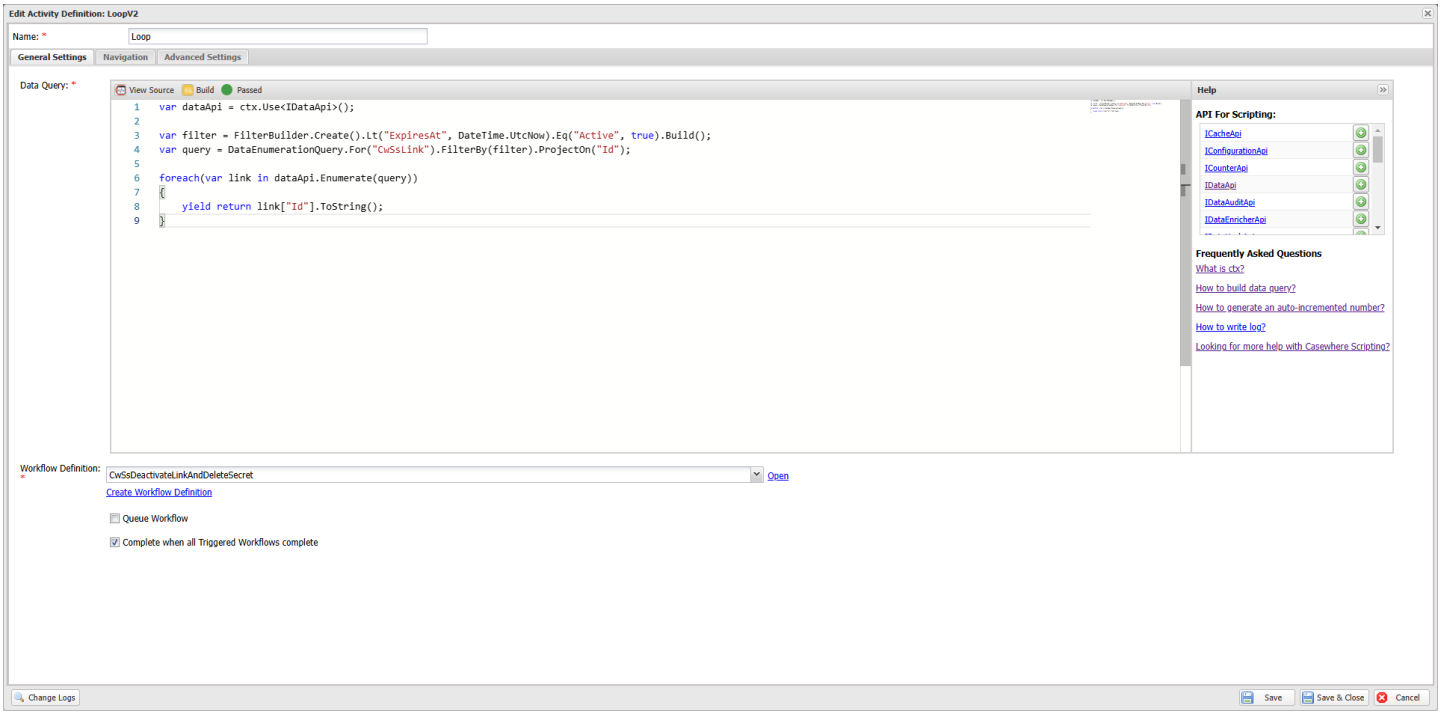
If a business scenario requires processing a large volume of work that a single workflow cannot handle, the **Loop Activity** can be a suitable solution, particularly when executing a large, dynamic number of workflows.

Configuration

The **Loop Activity** can be added to the workflow definition just like any other workflow activity.



The activity configuration appears as follows:



Data Query: A script used to determine the data for looping. The script must return an `IEnumerable<string>`.

Workflow Definition: The workflow for data looping iteration, which accepts a data object as workflow input.

Queue Workflow: When selected, the workflow will be queued for execution in the background.

- **Priority**: Used to determine the execution order, applicable only to queued workflows.
- **Complete when all Triggered Workflows Complete**: If selected, the loop activity completes only when all looping workflows complete with out any error. This setting is only available when the **Queue Workflow** option is not selected.

Data query examples

Example 1: Looping through a large data collection using `Enumerate` API

```
var dataApi = ctx.Use<IDataApi>();

var filter = FilterBuilder.Create().Lt("ExpiresAt", DateTime.UtcNow).Eq("Active", true).Build();
var query = DataEnumerationQuery.For("CwSsLink").FilterBy(filter).ProjectOn("Id");

foreach(var link in dataApi.Enumerate(query))
{
    yield return link["Id"].ToString();
}
```

Example 2: Looping through a normal data collection using `Search` API

```
var dataApi = ctx.Use<IDataApi>();

var filter = FilterBuilder.Create()
    .Lt("ExpiresAt", DateTime.UtcNow)
    .Eq("Active", true)
    .Build();
var result = dataApi.Search(DataObjectApiQuery
    .For("CwSsLink")
    .FilterBy(filter)
    .ProjectOn("Id")).Data as List<DynamicDataObject>;

return result.Select(item => item["Id"]).Cast<string>();
```

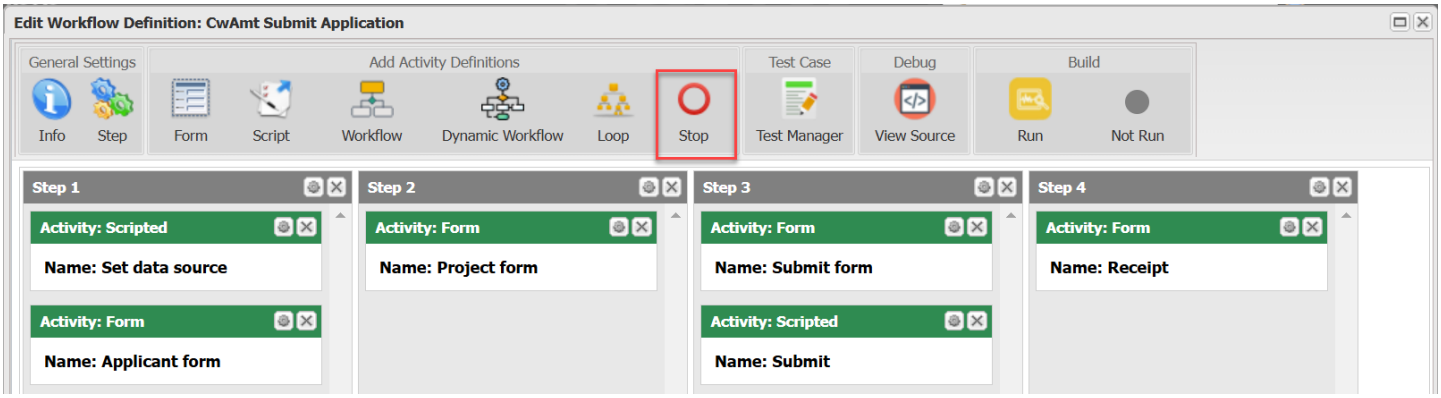
Stop Activity

A workflow progresses to the final included activity before stopping. In complex workflows with intricate navigation rules, identifying potential exit points and understanding their stopping logic can be challenging, complicating debugging and maintenance.

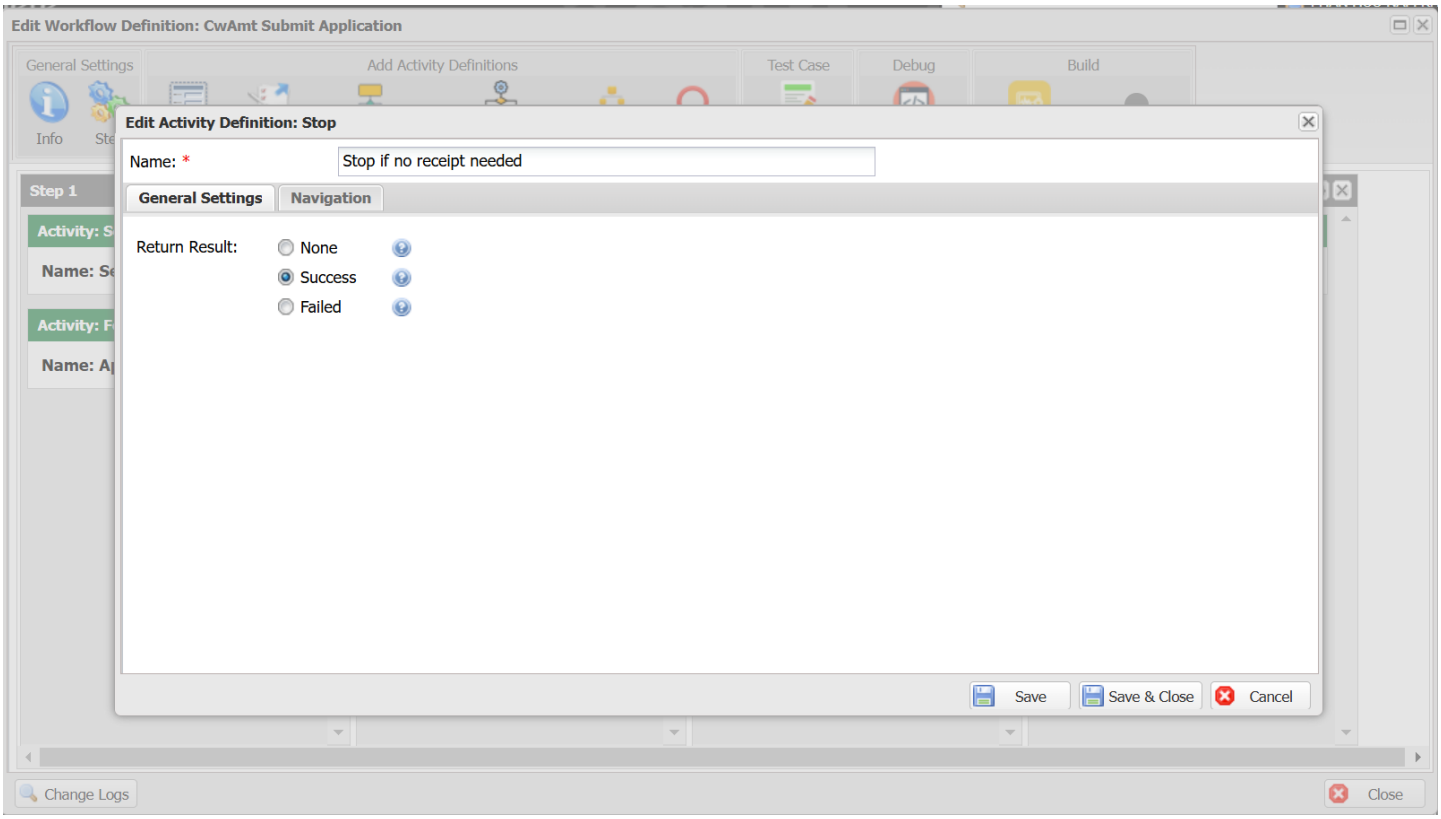
Since version 2.8, Casewhere introduces the Stop Activity feature, allowing workflows to be explicitly stopped at defined points.

Configuration

The **Loop Activity** can be added to the workflow definition just like any other workflow activity.



The activity configuration appears as follows:



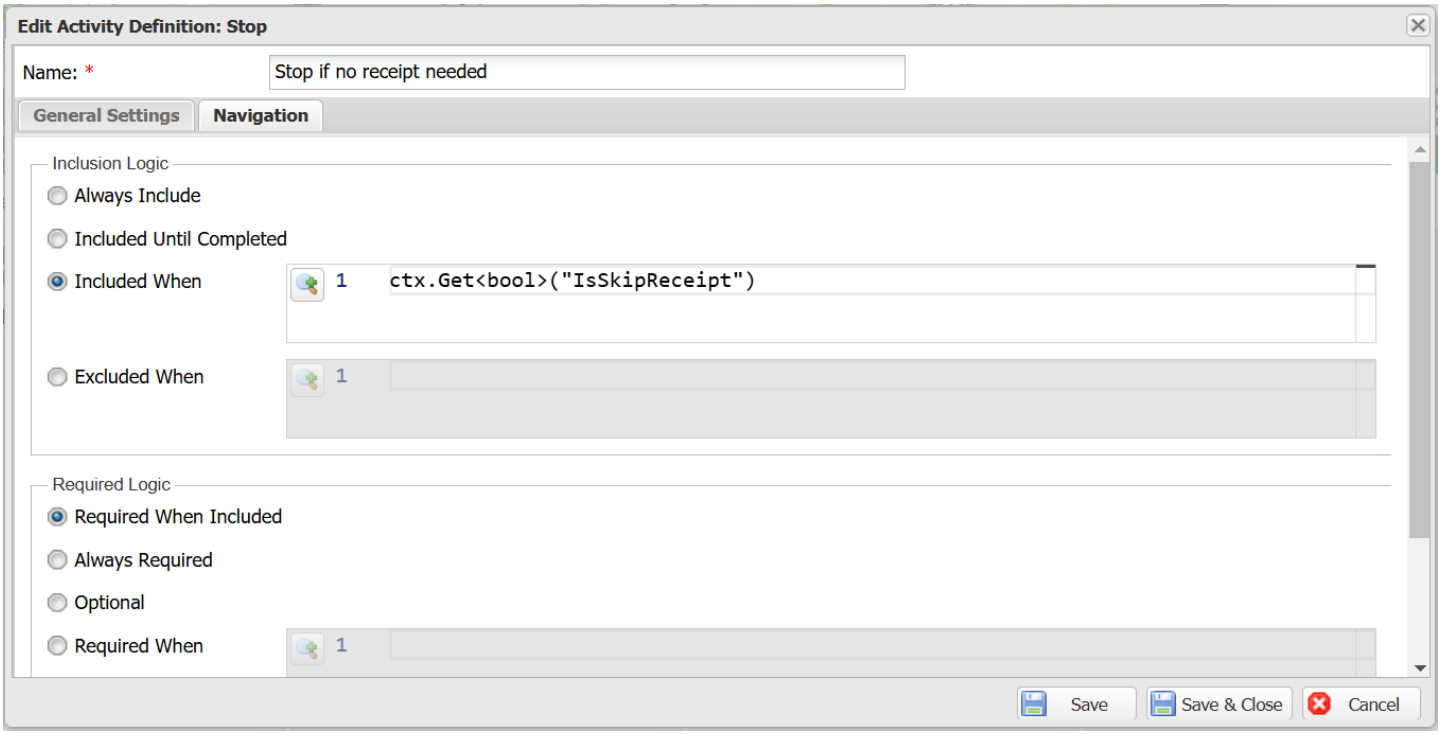
Return Result: Defines the workflow state upon stopping.

- **None**: Casewhere uses the existing logic—based on whether all required activities have been completed successfully—to determine if the workflow state is **Open** or **Complete**.
- **Success**: Forces the workflow state to **Complete**, regardless of activity statuses.
- **Failed**: Forces the workflow state to **Open**, regardless of activity statuses.

Similar to other activities, the **Stop Activity** adheres to the workflow's navigation logic and only executes if it is included in the

workflow.

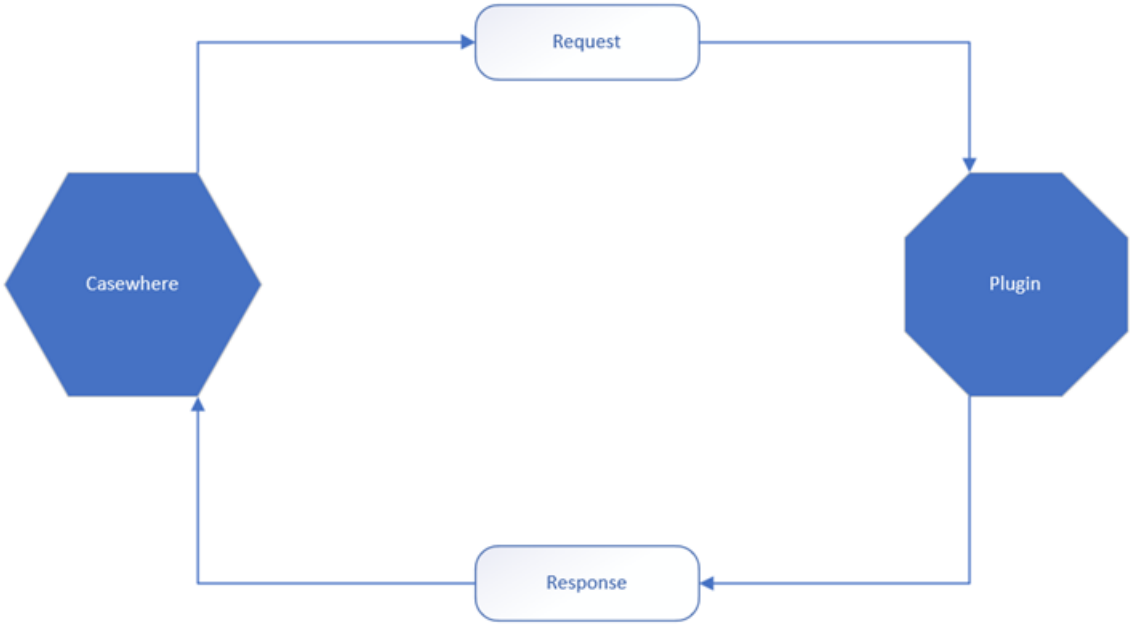
The following screenshot illustrates how a **Stop Activity** is included when users choose to skip the receipt. The workflow stops and does not proceed to the next step, bypassing the receipt display.



Plugin overview

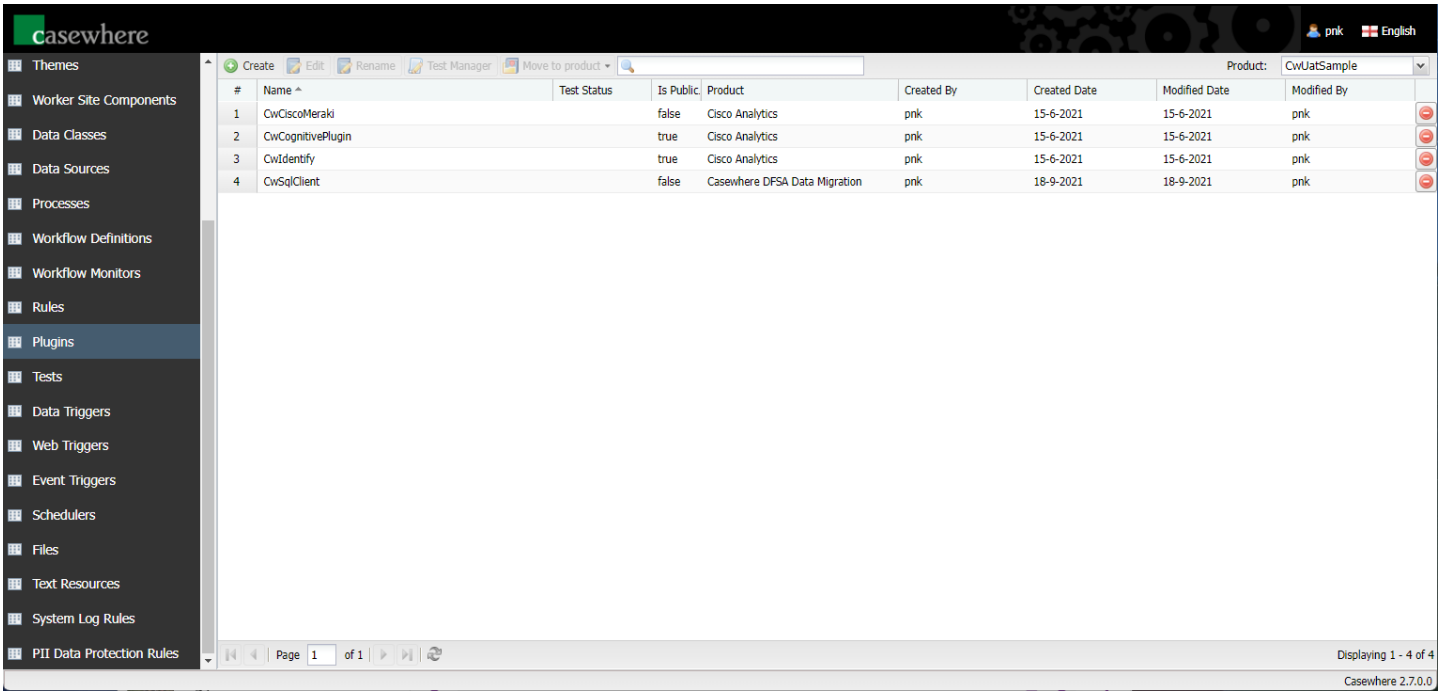
Casewhere is a standard platform, so it does not have business-specific functionalities such as communicating with payment gateways, sending emails, signing PDFs, etc. However, Casewhere comes with a flexible plugin framework to make it possible to add features to the platform and extend your business solution. Casewhere plugins can be developed independently, installed on-demand, and invoked at runtime.

A request/response protocol is used for the communication between Casewhere and plugins. Basically, plugins will receive requests from Casewhere, process the request, format the result and finally send the response back to Casewhere.



Plugin management

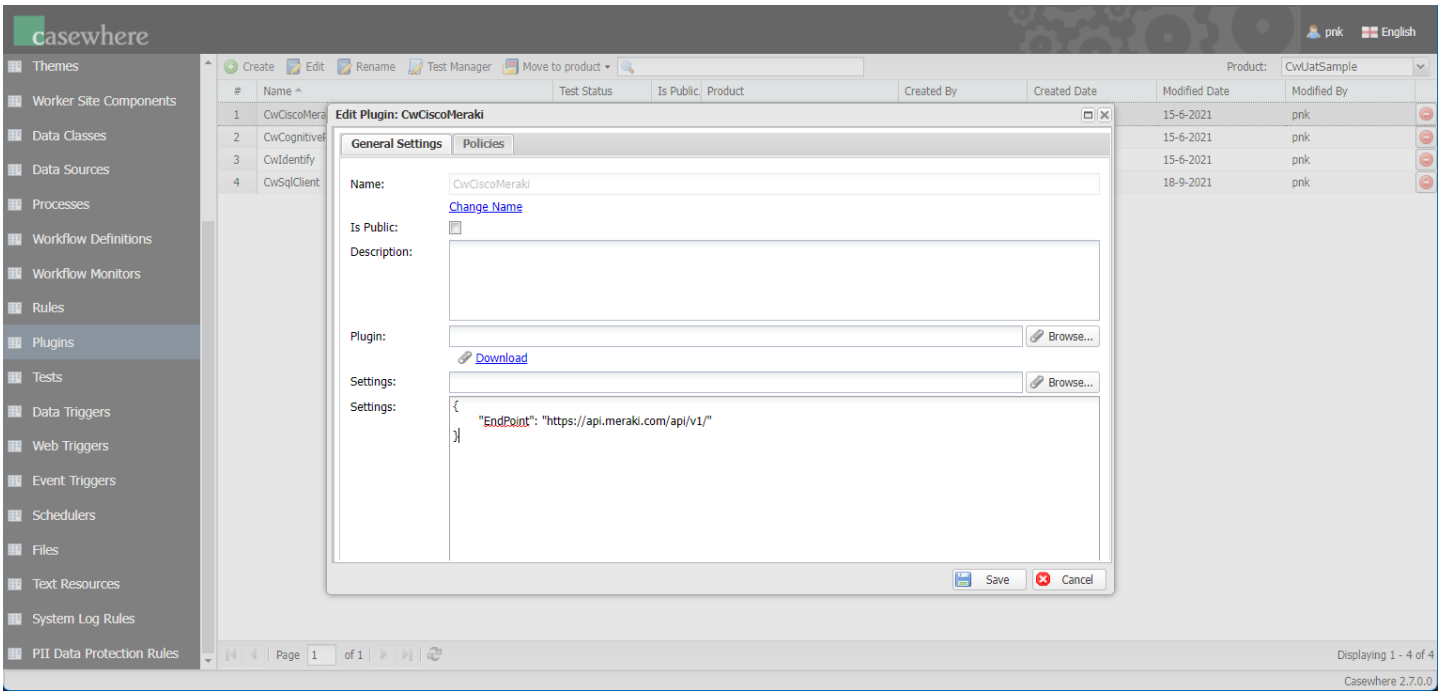
In Casewhere Admin, navigate to **Plugins** to create, edit and manage plugins like any product resource.



Configure plugin

Click **Create** and input the following into the plugin editor:

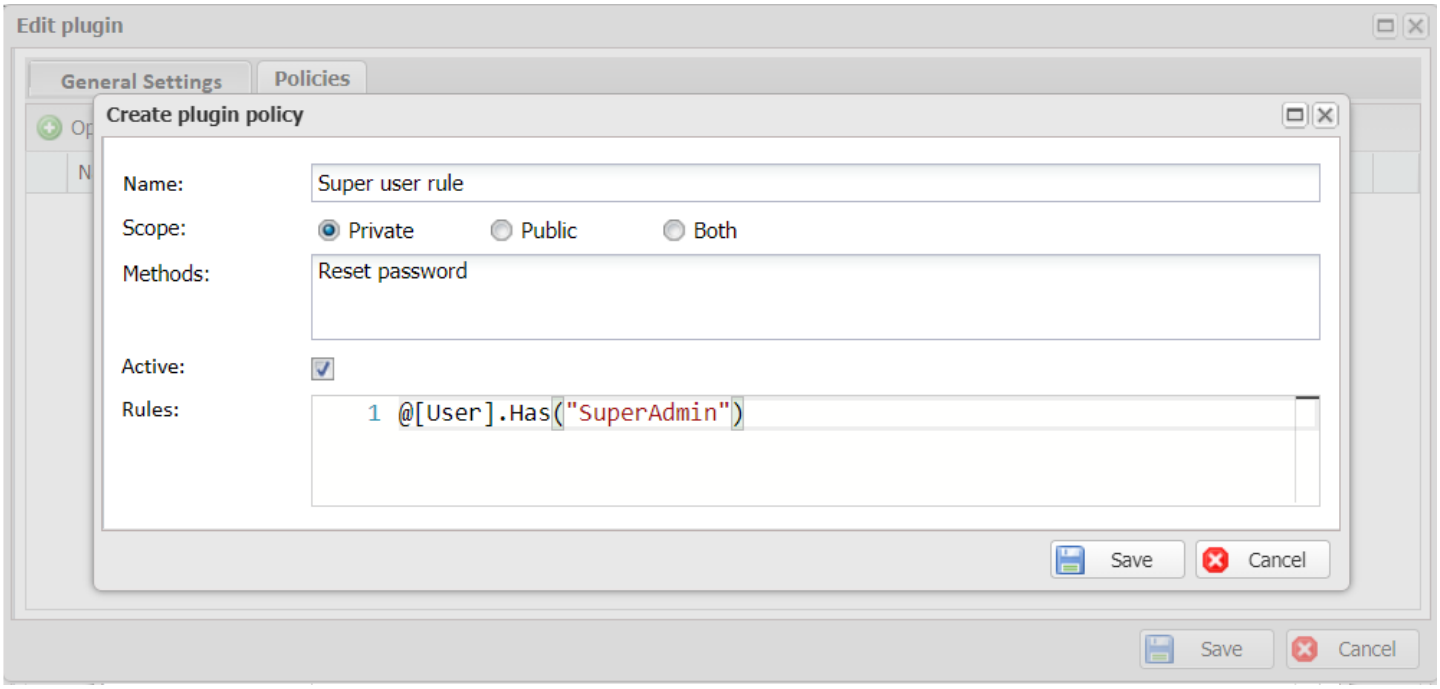
- **Name**: Plugin name must be unique.
- **Is Public**: Determines if the plugin is available for access from Worker API or not. It's not recommended to turn on this settings for sensitive plugins.
- **Desctription**: Here you can describe more what the plugin is about.
- **Plugin**: Upload the plugin package - a zip package containing DLL files required for the plugin to execute.
- **Settings**: Plugin configuration in a text format, e.g., JSON, XML.



Plugin security

Plugin security is vital. Casewhere executes plugins in a partially trusted AppDomain - separate from Casewhere's core - to mitigate the risk of plugins being exploited to execute malicious code. The 'Is Public' setting is optional because some plugins are too sensitive to be exposed to a web interface.

Additionally - for even more extensive security - Casewhere allows you to define rules to protect the plugin methods.



Plugin developement

Casewhere provides a flexible plugin framework to make it easy to extend your Casewhere solution with more functionalities. From simple tasks like sending emails to complex ones such as executing a SQL query, almost everything can be done using a Casewhere plugin. Casewhere plugins can be developed independently, installed on-demand, and invoked at runtime.

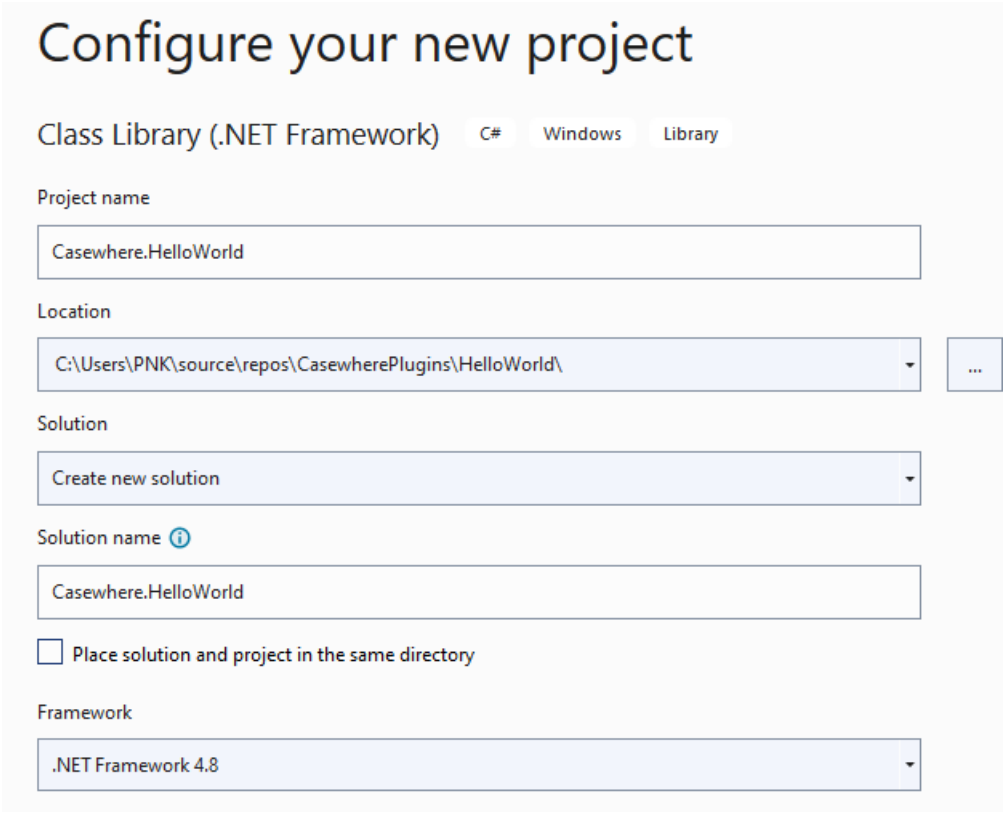
Plugin method

`PluginMethod` is an interface designed to ease the communication between Casewhere and plugins. Through this interface, Casewhere will send data to plugin and retrieve the result after the plugin executes.

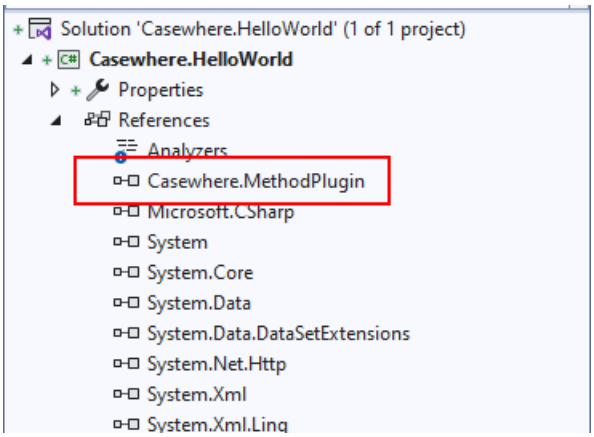
```
public interface IPluginMethod
{
    PluginResult Invoke(PluginRequest request);
}
```

Develop plugin

Create a .**Net Framework Class Library** project.

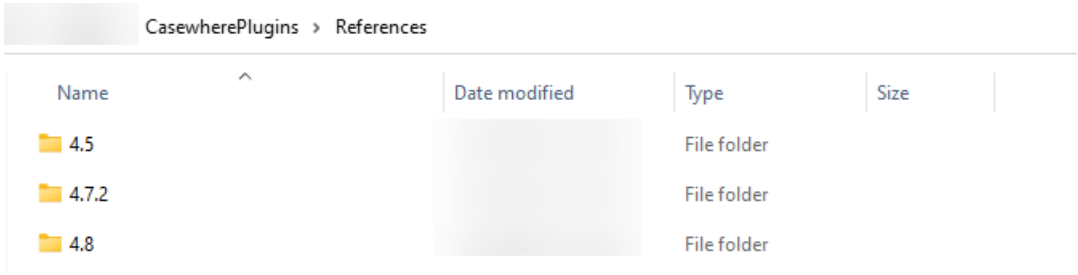


Add a reference to the assembly `Casewhere.MethodPlugin`. Make sure you refer to the correct assembly targeted to the .Net framework version you're working on.



Hint: You can find the file **Casewhere.MethodPlugin.dll** in the folder **Casewhere.Worker.Api/bin**. Make sure you copy the file to your project folder.

Important: If you're a Casewhere Team developer, make sure you follow the team's guidelines to reference the file.



Add a `class` and implement the interface `IPluginMethod`.

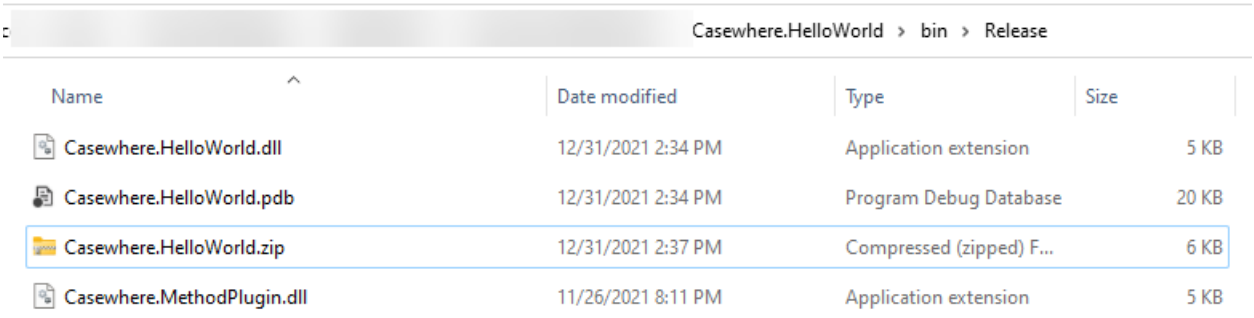
```
public class HelloWorldMethod : IPluginMethod
{
    public PluginResult Invoke(PluginRequest request)
    {
        var name = request.Parameters["Name"]?.ToString();

        var result = new PluginResult();
        result.Data["Greeting"] = $"Hi {name}";

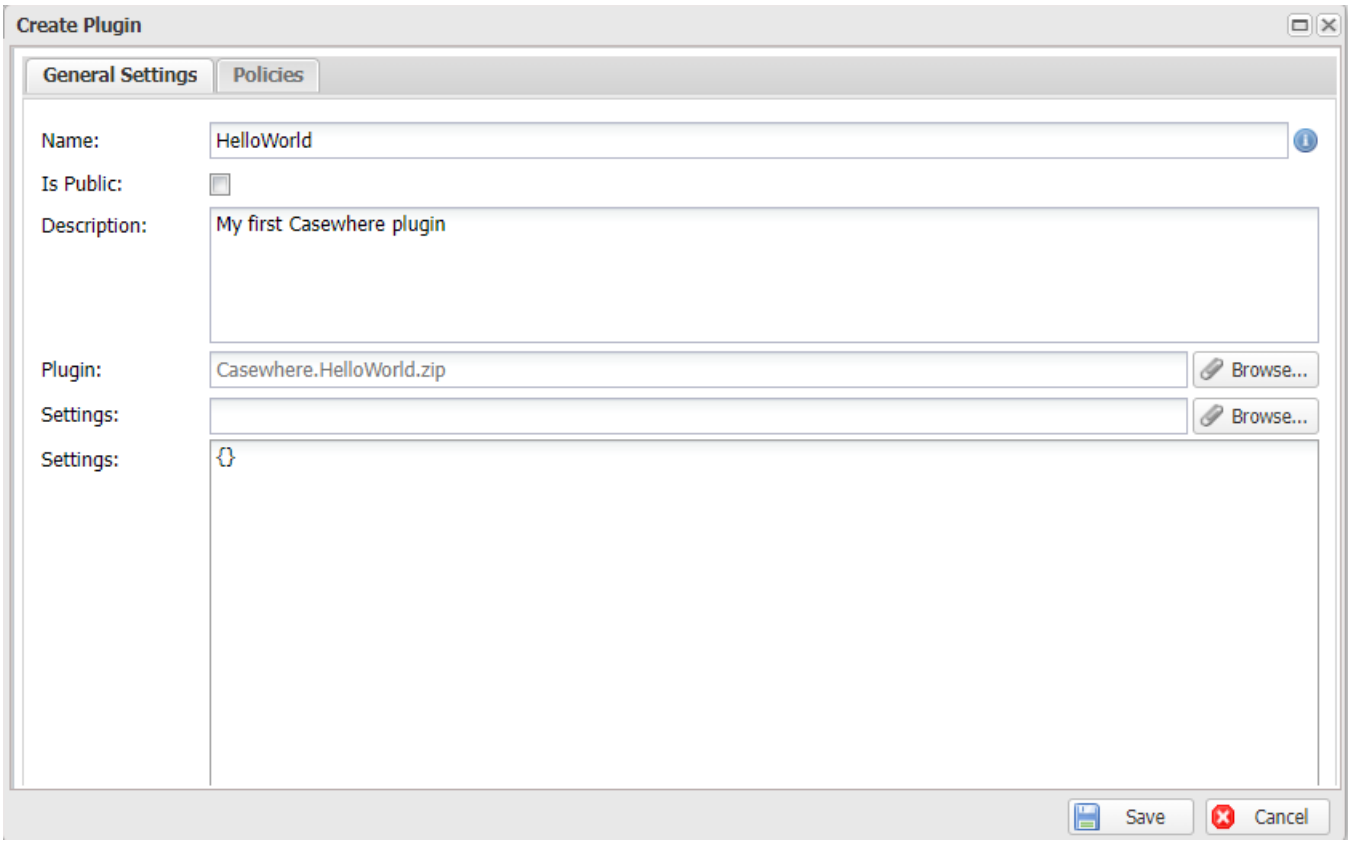
        return result;
    }
}
```

Install plugin

When you finish coding, build the project in **Release** mode. Navigate to **bin/Release** in your project folder and create a zip file for all the files.



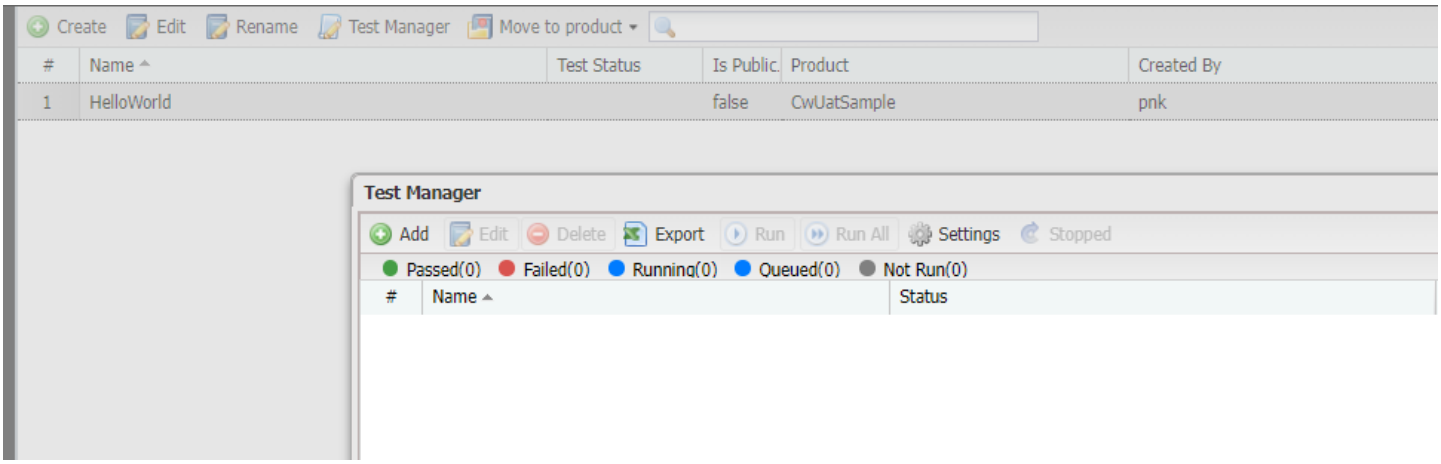
In Casewhere Admin, navigate to **Plugins**, click **Create** and fill in the information as the example below:



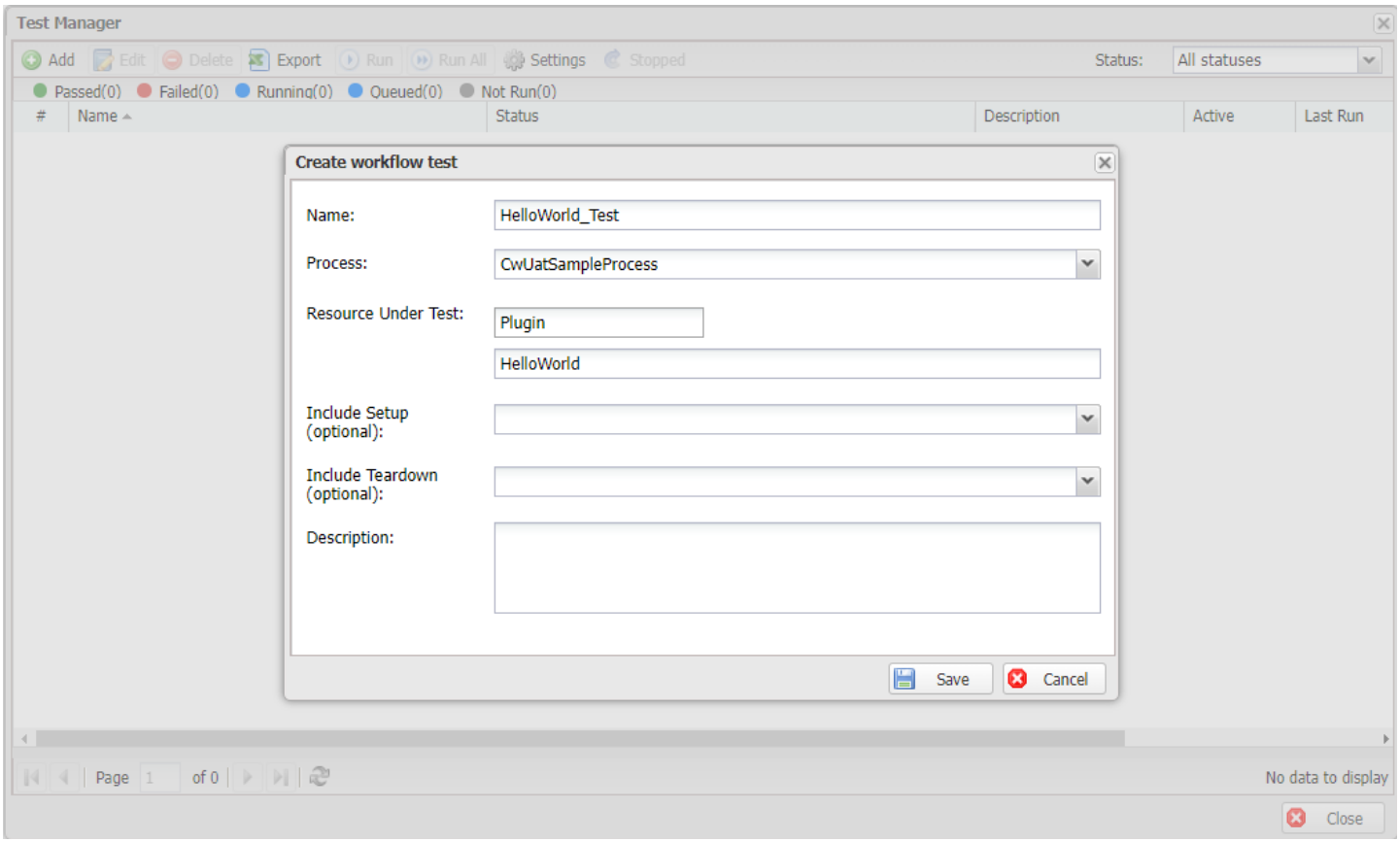
Test plugin

After creating the plugin, you can use the built-in testing framework to test if the plugin works as expected.

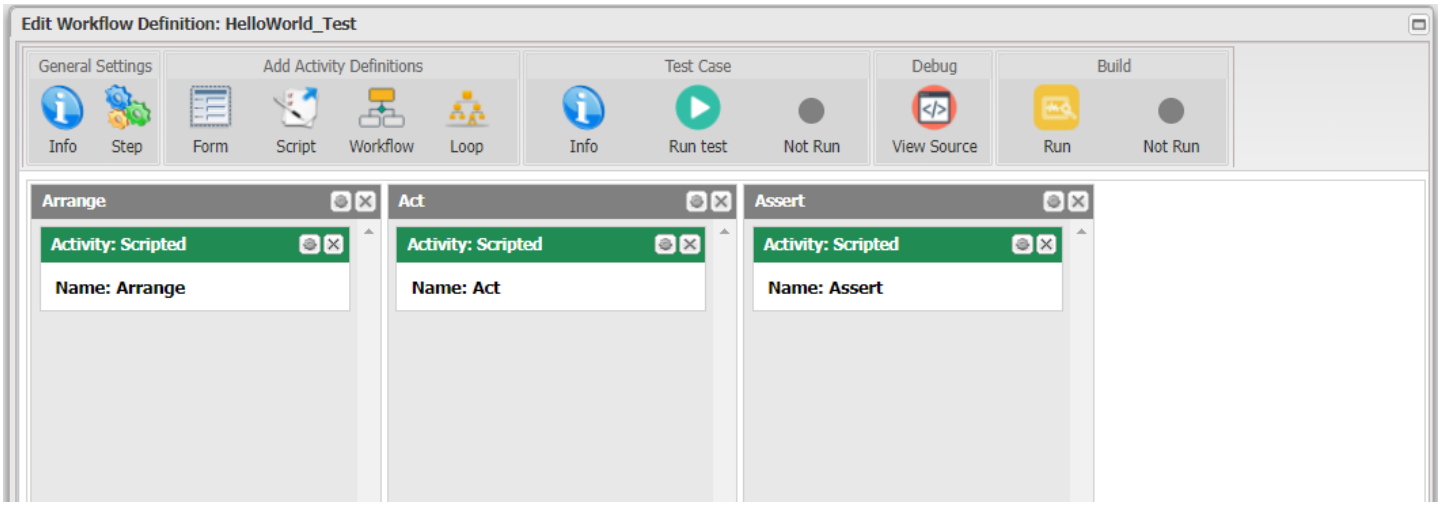
Select the plugin you just created, click **Test Manager**.



In the **Test Manager**, click **Add**, fill-in the information as well as select a **Process** for testing. Finally, click **Save**.



A new test workflow will be created as below:



In **Arrange**, you can use the following script to prepare the test data.

```
var testApi = ctx.Use<ITestApi>();
testApi.Set("Name", "John Doe");
```

The **Act** step is where you execute the plugin and retrieve the reult.

```
var pluginApi = ctx.Use<IPluginApi>();
var testApi = ctx.Use<ITestApi>();

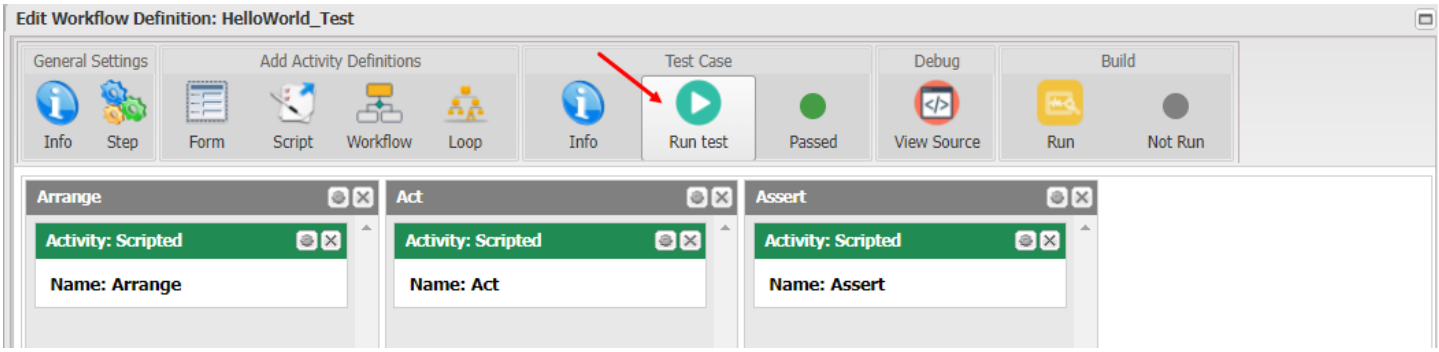
var parameters = new Dictionary<string, object>();
var result = pluginApi.Invoke("HelloWorld", "HelloWorldMethod", parameters);
testApi.Set("result", result.Get<string>("Greeting"));
```

In **Assert**, you will verify if the result is what you are expecting.

```
var testApi = ctx.Use<ITestApi>();

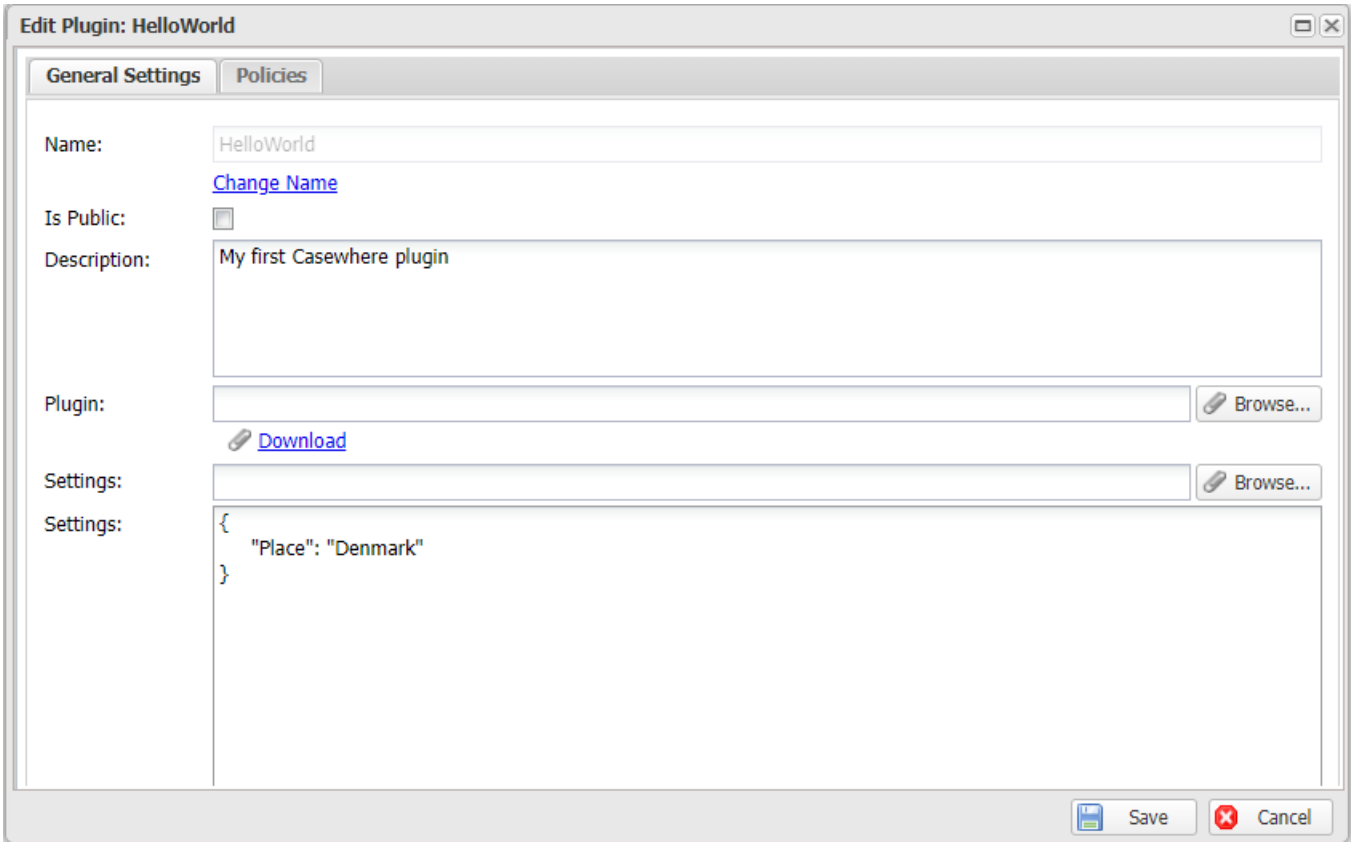
var result = testApi.Get<string>("result");
testApi.True(result == "Hi John Doe", "The output is not correct.");
```

Finally, click **Run test** and examine the test result. You can learn more about the Casewhere automation test framework here.



Plugin configuration

When installing the plugins, Casewhere users must configure the **settings** that the plugin requires to function. The plugin settings must be text-based, using a format specified by the plugin. A typical format for plugin settings is JSON as the example below:



The Casewhere plugin framework provides an interface for plugins to load the configuration from Casewhere Admin and use it in the plugin methods.

```
public interface IPluginConfigure
{
    void Configure(string settings);
}
```

The above example is showing that the plugin requires a `Place` attribute to function. You can implement the plugin configuration by adding a `class` with the below code:

```
public class PluginConfiguration : IPluginConfigure
{
    public void Configure(string settings)
    {
        var obj = JObject.Parse(settings);
        Endpoint = obj["Place"].Value<string>();
    }

    public static string Place { get; set; }
}
```

You then can use the setting in your plugin method like this:

```
public PluginResult Invoke(PluginRequest request)
{
    var name = request.Parameters["Name"]?.ToString();

    var result = new PluginResult();
    result.Data["Greeting"] = $"Hello {name} from {PluginConfiguration.Place}";

    return result;
}
```

Plugin API

Casewhere provides a set of APIs to interact with plugins from different places in the solution. A typical place to invoke plugin methods is from inside a workflow. You can also invoke plugins from the worker site.

Execute plugin from workflow

From a workflow scripted activity, you can invoke plugin methods using the interface `IPluginApi`.

```
PluginMethodResult Invoke(string pluginName, string methodName, IDictionary<string, object> parameters);
```

Here is an example how it can be done:

```
var pluginApi = ctx.Use<IPluginApi>();
var testApi = ctx.Use<ITestApi>();

var parameters = new Dictionary<string, object>();
var result = pluginApi.Invoke("HelloWorld", "HelloWorldMethod", parameters);
testApi.Set("result", result.Get<string>("Greeting"));
```

The `PluginMethodResult` class provides properties and methods to help retrieve data from the plugin:

- `Data`: A key-value data collection returned from the plugin.
- `T Get<T>(string name)`: Get data for a given key. Use this method when you know exactly the type of the data.
- `JObject GetObject(string name)`: Return data as a JSON object. Return null if the item is not in JSON format.
- `JArray GetArray(string name)`: Return data as a JSON array. Return null if the item is not in JSON format.

Execute plugin from worker site

> **Note:** You can only invoke public plugins from worker site.

From a custom widget, you can invoke plugin methods using the `PluginService`.

```
var parameters = { Name: "John Doe" };

PluginService.invoke("pluginName", "pluginMethodName", parameters)
    .then(function (pluginResult) {
        console.log(pluginResult);
    }, function (error) {
        console.log(error);
    });
```

Casewhere CMS overview

Introduction

Casewhere has Pages and Widgets, which are very powerful for building enterprise web applications and provide a solid foundation in security and workflow.

However, if the website admin (the customers) wants to actively update website content on a regular basis, an application built solely with Page and Widget may not be adaptable enough.

Case in point: CMS (referred to as CMS) was created to address this customer need, even better, CMS was designed to bridge the gap between business processes and content management.

Everyone who is in charge of creating and managing a website will benefit from CMS.

CMS provides website administrators with an intelligent user interface for creating and managing webpage content. CMS boosts designers' creativity, allowing them to create the best website layouts possible. CMS allows developers with the flexibility to build and focus on creating solutions to complex problems.

CMS helps businesses streamline the website upgrade process, resulting in lower opportunity costs. CMS also allows for consistency, which is essential for developing users engagement and providing a better user experience.

A standout feature of Casewhere CMS is the ability to embed widget templates, as well as plug in with other Casewhere products, and even integrate with the classic "Page and Widget".

Features

- Content editing
- Widget template
- Content publishing
- Content searching

Access Casewhere CMS

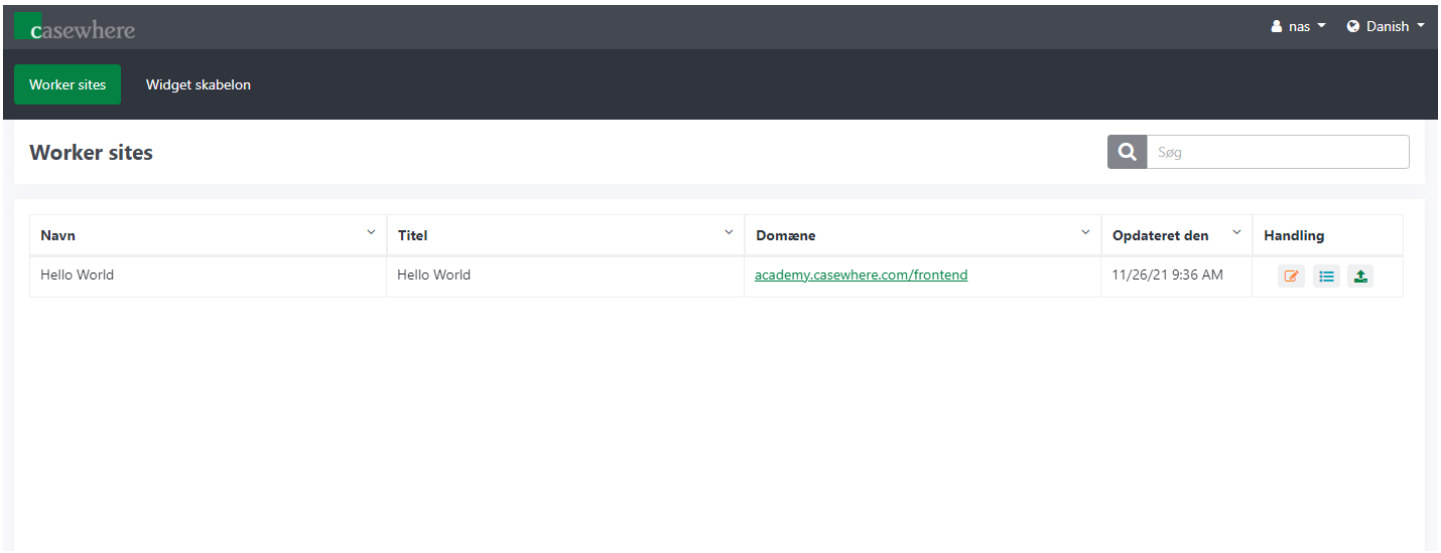
You can access Casewhere CMS by adding "/admin" in the Url of your Casewhere Admin instance.

Example: https://academy-admin.casewhere.com **/admin**

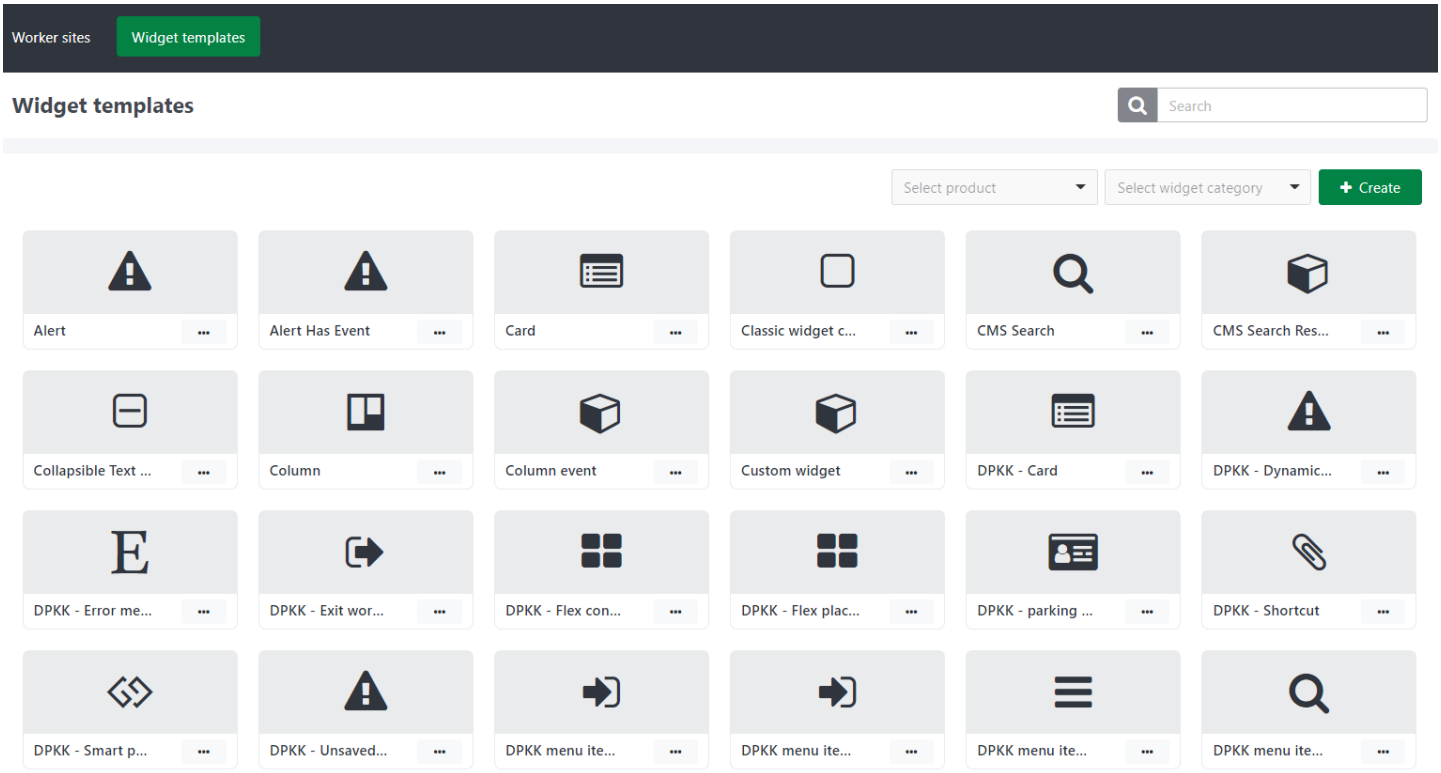
Interface

Casewhere CMS interface has a menu at the very top bar. Use this bar to navigate between 2 pages:

- Worker sites: a list of all worker sites existing in the system. Note that the worker sites need to be created in Casewhere Admin (Read more in Content editing - Add a worker site to CMS)



- Widget template: a list of pre-built widgets that can be customized in terms of appearance and functionality.



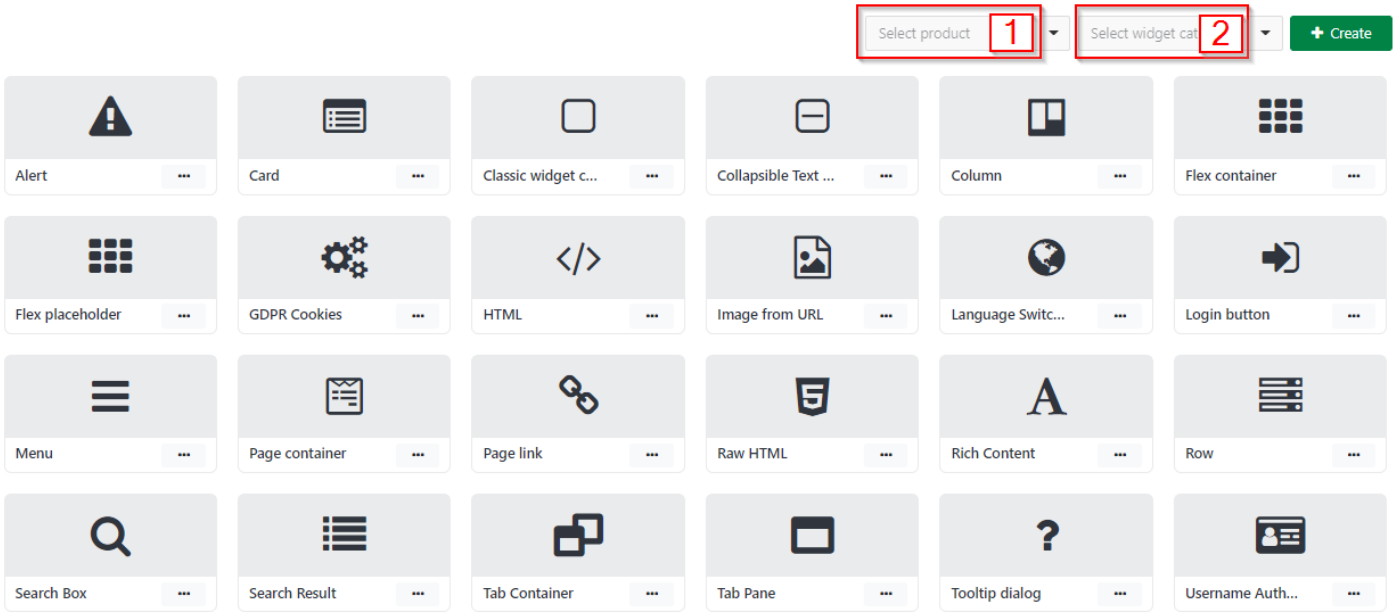
CMS widget template

Introduction

Widget templates is a modest framework for creating and configuring widgets right inside CMS.

Filter widget template list

The initial view of **Widget templates** will list out all available widgets, you can filter by (1) Product name and (2) Widget category.



Design widget template tool

Click **Create** button  to open **Create widget template** interface, which contains:

1. General setting menu
2. Coding area
3. Configuration menu
4. Content menu



1. General setting menu

Where widget developer can set up all general settings of the widgets

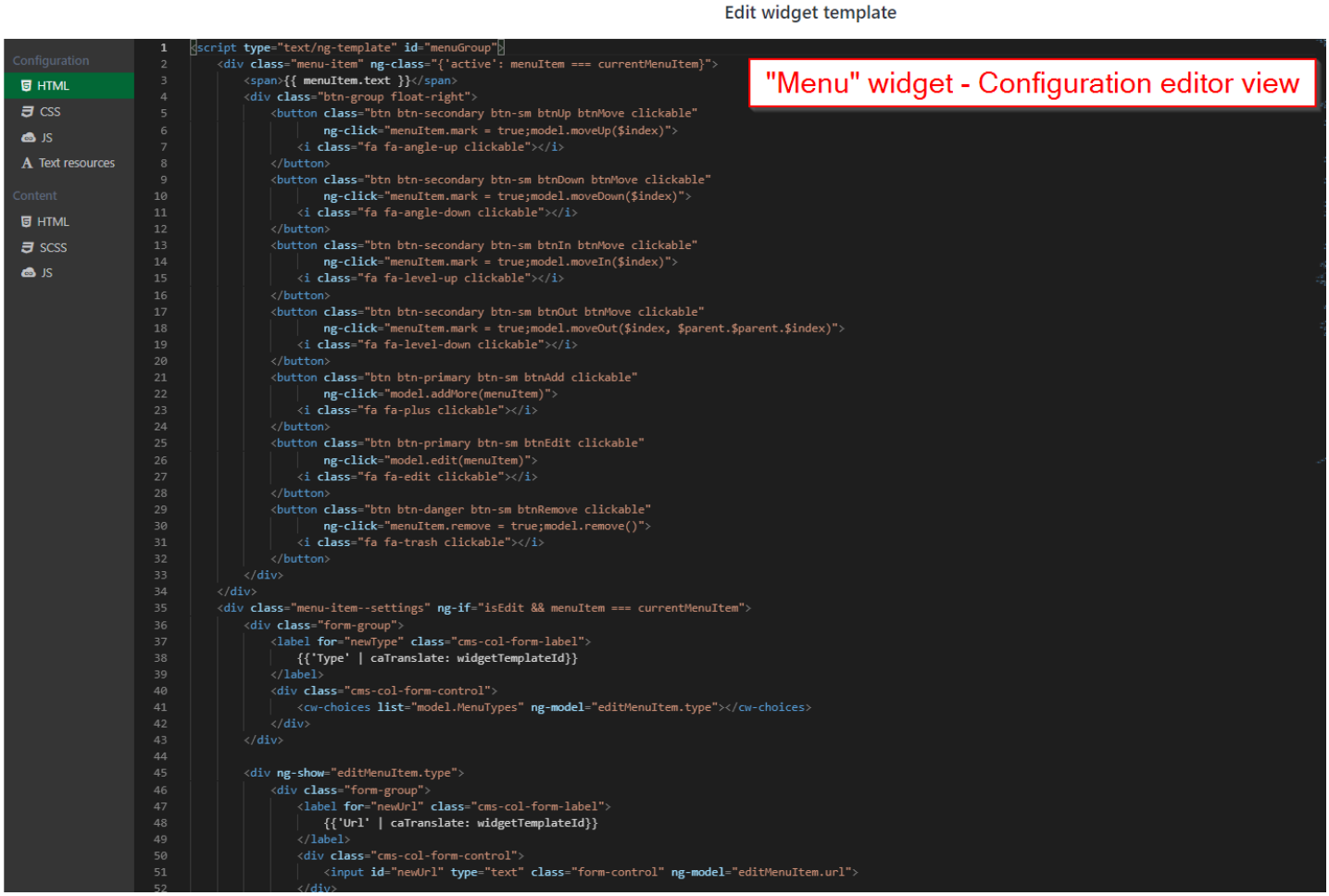
- **Name**: set widget template name
- **Only included in**: include this widget template only inside another parent widget template
    - If this field is empty: the widget has no placing limitation, it can be added in anywhere in the site
    - If a widget name (refers as parent widget) is selected: the widget can only appear inside the parent widget
- **Window size**: set widget template working window size
- **Icon**: choose an icon represent this widget in a library of 700+ icons
- **Is container**: check if allow this widget template to be a container itself
- **Product**: select the product that this widget template belongs to
- **Widget categories**: select the category that this widget template belongs to
    - Add category: create more widget category
- **Require user's configuration**: check if you want to force user to config the widget right after it is created. A modal config will appear when user select the widget's name
- **Lock deletion**: check if you want to prevent others developer to delete this widget template
    - Lock deletion will automatically turn on if the widget template is being used
- **Searchable Content**: check if you want to allow the content of this widget to be searchable. Read CMS content searching for more on how to use this field

2. Coding area

Where widget developers write down the codes.

3. Configuration editor

This is where developers implement the widget configuration UI, which is shown to the CMS editors.



4. Content editor

This is where developers implement the widget display, which is shown to end-users in the worker sites.

Edit widget template




```
<script type="text/ng-template" id="dropdownTemplate">
    <div class="cms-menu__nav-content"
         ng-class="'cms-menu__nav-content--' + menuItem.imagePosition"
         ng-if="$root.canAccess(menuItem.id) && !model.checkMenuItemsValid(menuItem)"
         ng-style="{'color': model.configuration.TextColor}"
         ng-mouseover="model.changeColor($event)"
         ng-mouseleave="model.resetColor($event)">

        <a class="dropdown-item cms-menu-item cms-menu-text"
           ng-class="{'reset-pointer': !menuItem.lastUrl}"
           ng-href="{{menuItem.lastUrl}}"
           target="{{menuItem.target}}">
            <img ng-if="menuItem.image && menuItem.image.Url"
                 ng-src="{{menuItem.image.Url | cwFileLinkFilter}}"
                 ng-attr-alt="{{menuItem.image.AltText | translate}}"
                 ng-style="menuItem.imageStyle"
                 class="cms-menu-item__img" />
            <span class="cms-menu-text" ng-bind-html="menuItem.text | translate"></span>
        </a>
    </div>

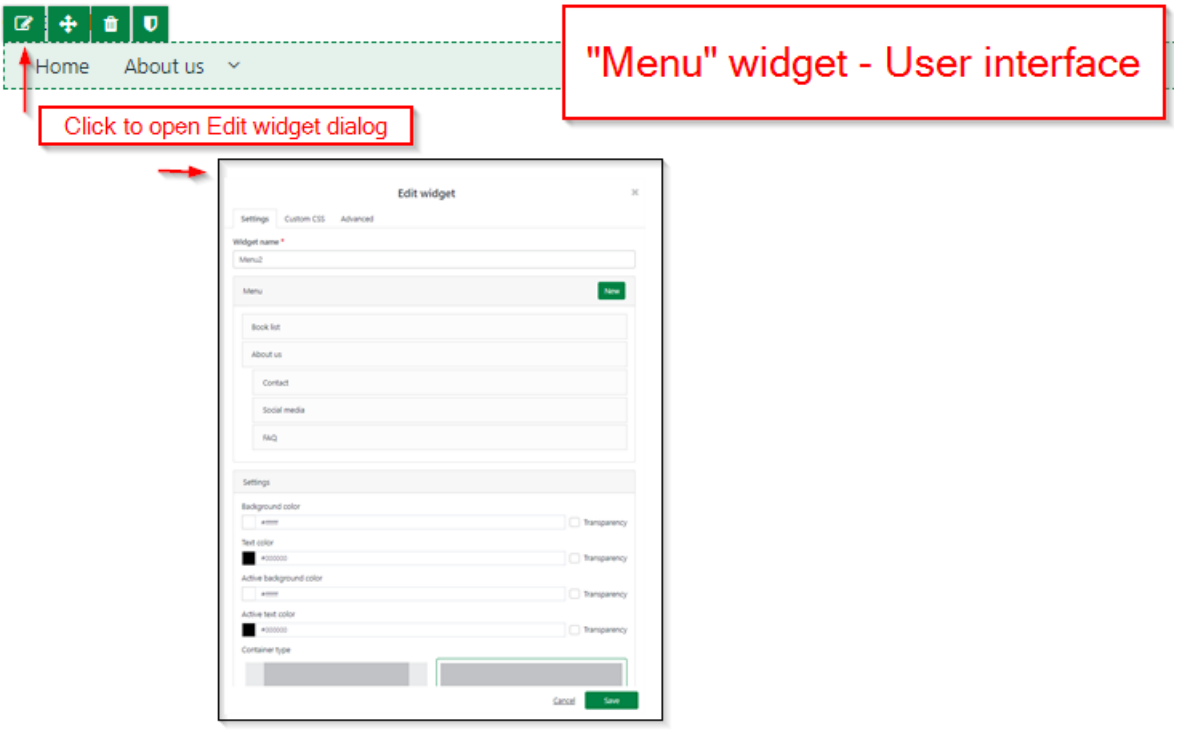
    <div class="cms-dropdown-submenu">
        <div class="cms-menu__nav-content"
             ng-class="'cms-menu__nav-content--' + menuItem.imagePosition"
             ng-if="$root.canAccess(menuItem.id) && model.checkMenuItemsValid(menuItem)"
             ng-style="{'color': model.configuration.TextColor}"
             ng-mouseover="model.changeColor($event)"
             ng-mouseleave="model.resetColor($event)">

            <a role="button"
               class="dropdown-item cms-menu-item"
               ng-class="{
                    'cms-menu-item--reversed': model.configuration.Align === 'right',
                    'reset-pointer': !menuItem.lastUrl
                    }"
               id="{{'navbarDropdownMenuLink' + $index}}"
               ng-href="{{menuItem.lastUrl}}"
               ng-attr-target="{{menuItem.target}}"
               ng-click="model.parentClick($event, menuItem)">

                <span class="d-flex align-items-center flex-1">
                    <img ng-if="menuItem.image && menuItem.image.Url"
                         ng-src="{{menuItem.image.Url | cwFileLinkFilter}}"
                         ng-attr-alt="{{menuItem.image.AltText | translate}}"
                         ng-style="menuItem.imageStyle"
                         class="cms-menu-item__img" />

                    <span class="cms-menu-text"
                          ng-bind-html="menuItem.text | translate"
                          ng-class="{'active': model.activeMenuItemMap[menuItem.ParentId] === menuItem}"></span>
                </span>
```

The image below illustrates how the "Menu" widget will appear in the user interface. Application designer can access **Edit widget** dialog to customize the widget features such as menu levels, background color, text color.

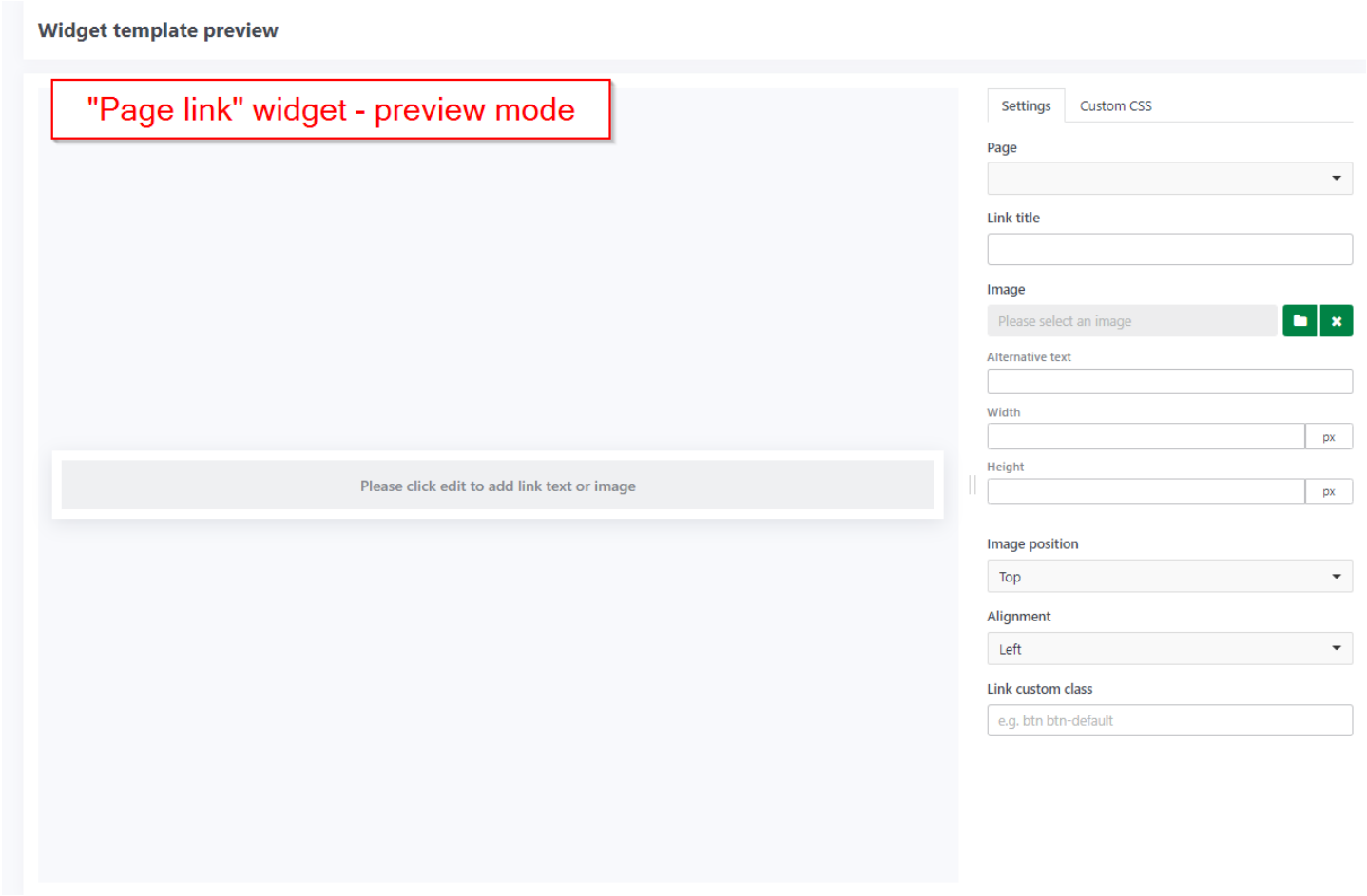


Preview and push changes

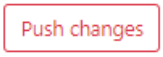
After you've finished designing a widget template, you can update and edit the widget template by preview and push changes functions.

Preview

Click **Preview** button  to enter **Widget template preview** mode. You will be able to view the interface as if the widget template was active.



Push changes

Click **Push changes** button  to public the widget template.

In case you are editing a widget template that been already used in some pages. This push changes action will also affect those pages. The pages status will also be updated as draft. Go to the page and publish it again to initialize the changes.

Getting started widget package

There are several widgets that may be used to create site content. All widgets include user-friendly configuration interfaces that allow CMS Designers to edit the widget content, implement security restrictions, and customize the style and feel, among other things.

Casewhere provides a set of standard widgets to help boost your website creation. You can totally customize these widget templates, make new ones, or reuse widget templates from other developers.

- **Search**: search box, search results, enable multiple languages content configuration
- **Alert**: add alert to inform or warning actions
- **Card**: add cards to your site
- **Row**: add rows to layout
- **Column**: add columns to row
- **Tab container**: add tabs to layout
- **Tab pane**: add tab pane to layout
- **Tooltip dialog**: widget to create tooltip dialog

- **Flex container**: add flexible container
- **Flex placeholder**: add flexible placeholder
- **GDPR Cookies**: accommodate GDPR requirements
- **Language Switcher**: enable multiple languages content configuration
- **Login button**: add login button to your page
- **Menu**: add and manage main menu
- **Page container**: all application will need this widget (read more at CMS content editing - Edit site design section)
- **HTML**: add HTML content fragment directly into a layout

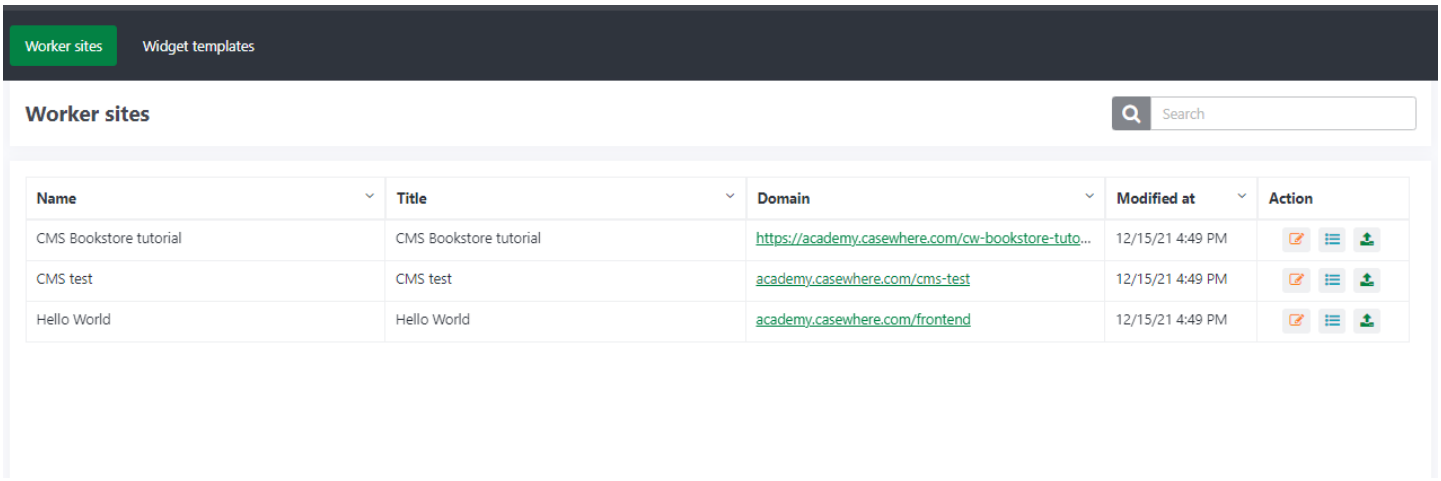
CMS content editing

Introduction

Understanding how to edit site content and manage website pages is important for application designer who works with Casewhere. These tasks are accomplished using the Casewhere CMS admin interface.

Casewhere CMS admin (refers as CMS) interface contains 2 main functioning areas:

- Worker sites: where you manage all of your worker sites and pages, we will discuss this functioning area in this article
- Widget template: where you work with the widgets by creating, modifying or even importing and exporting, read more in Widget template



You can edit CMS content easily, whether it's published or in a draft, without changing what's live on your site.

Worker sites

Add a worker site to CMS

First thing first, let go to Casewhere admin to create a new worker site. Think of Casewhere admin like a place to manage the backend of your application, meanwhile CMS is the place to manage the frontend a.k.a how your application will look like.

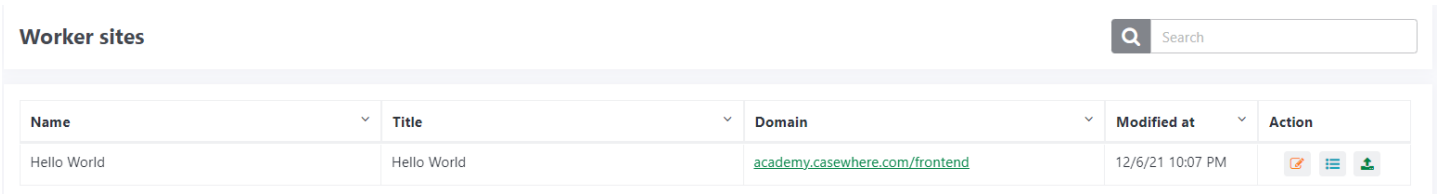
> Tip: switching from CMS (for application's frontend) to Casewhere admin (for application's backend) is actually very simple, just remove the "/admin" in your CMS's url

Step 1. Read Create a worker site for a guideline if you don't know how to create a worker site in Casewhere, notice a slight change that in step 3 you must choose the theme name "none"

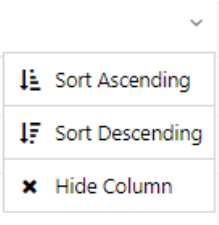
Step 2. Reload CMS, if you see your newly created worker site in the homepage list, congratulation, you are all set!

Manage worker sites

In the CMS **Worker sites** page, you will see a list of all worker sites with their name, tile, domain, modified time and action lists.



There are a few things that you can do with this homepage to make ease when managing worker sites:

- Sort any column 

- Click the domain name to quickly view the site 

- Design your worker site with action buttons

Design your worker site

Edit site design

Intro

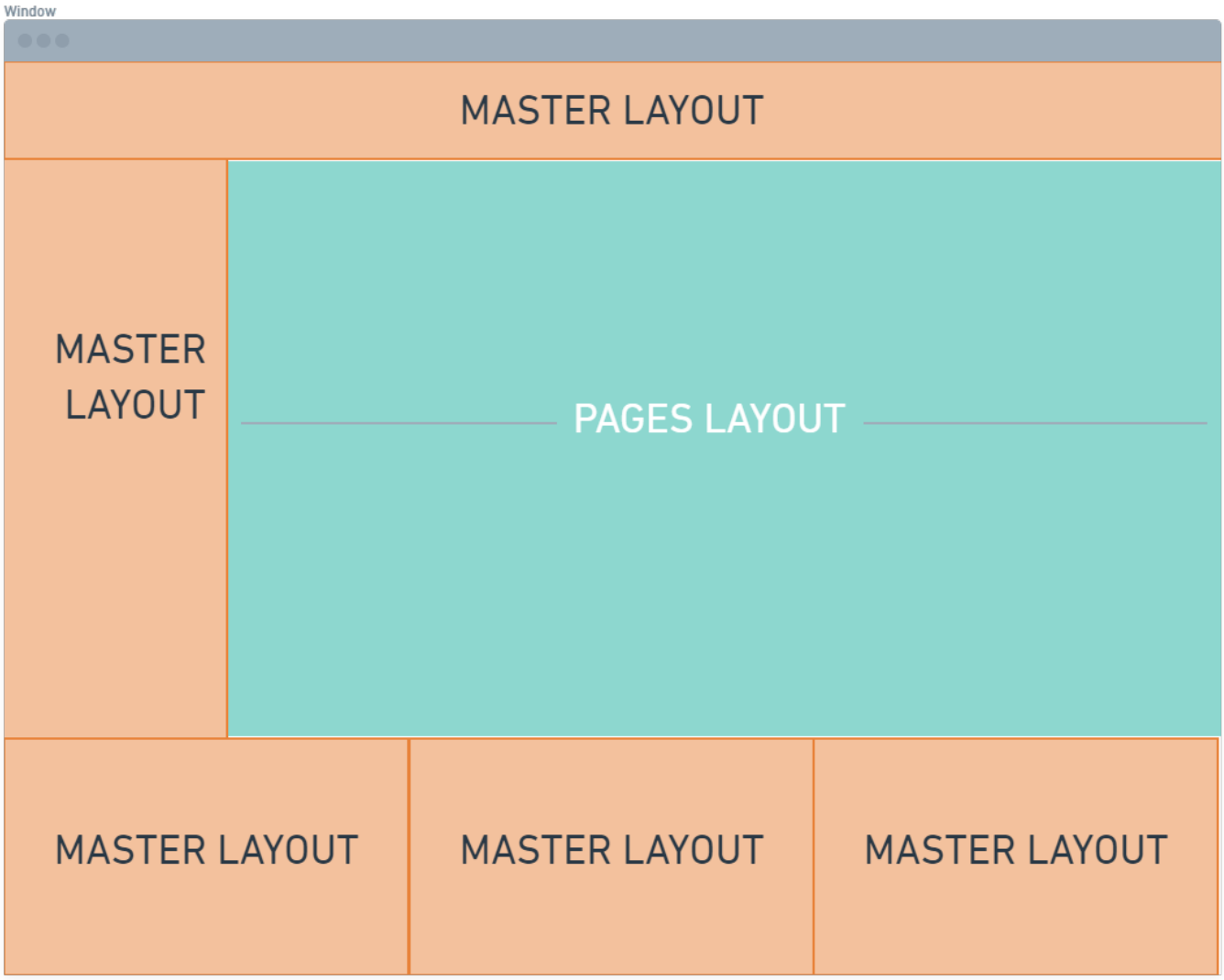
This function will help you design the master layout of your site.

Click **Edit site design** button  to enter the master layout editing mode.

What is master layout?

The master layout is an important building block in the construction of a worker site. Master layouts often contain graphical elements common to the theme. A master layout normally will have a header, logo area, main menu, and footer, but you actually can define as many elements as you want for your master layout.

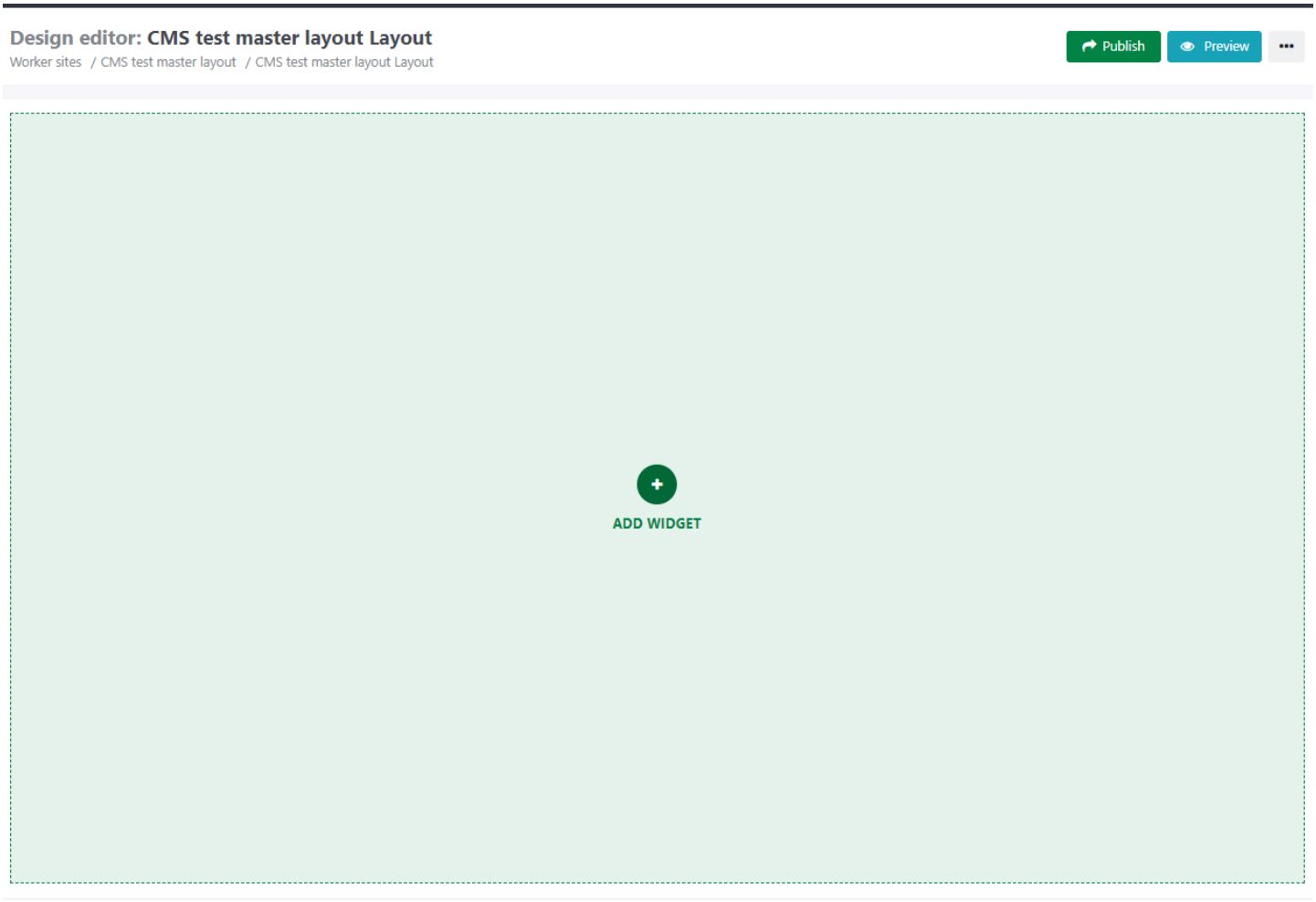
Each worker site has one master layout. It is not only helps to maintain a consistent appearance through all the pages of your application, but also help to save your time and effort since the designer only have to define it one time.



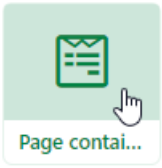
Guideline

A brand new master layout is totally blank. You can reuse an old layout, or in this article, you will learn how to create a new layout.

Step 1. Click **ADD WIDGET**

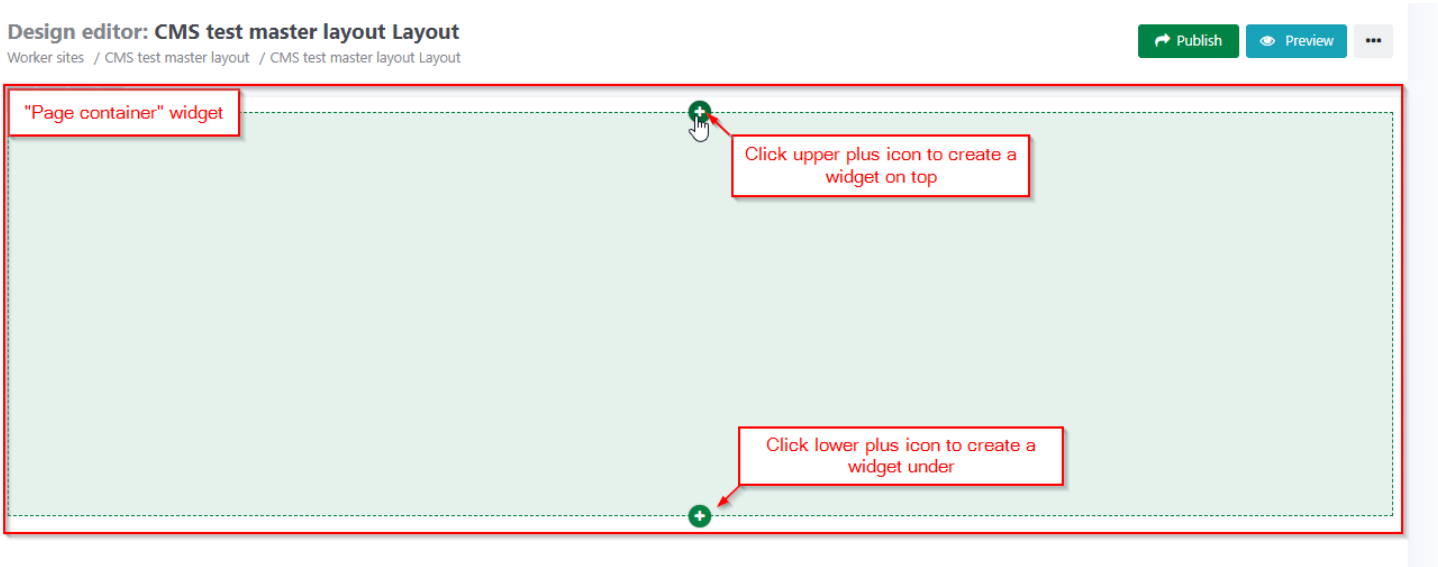


Step 2. Select the widget **Page container**

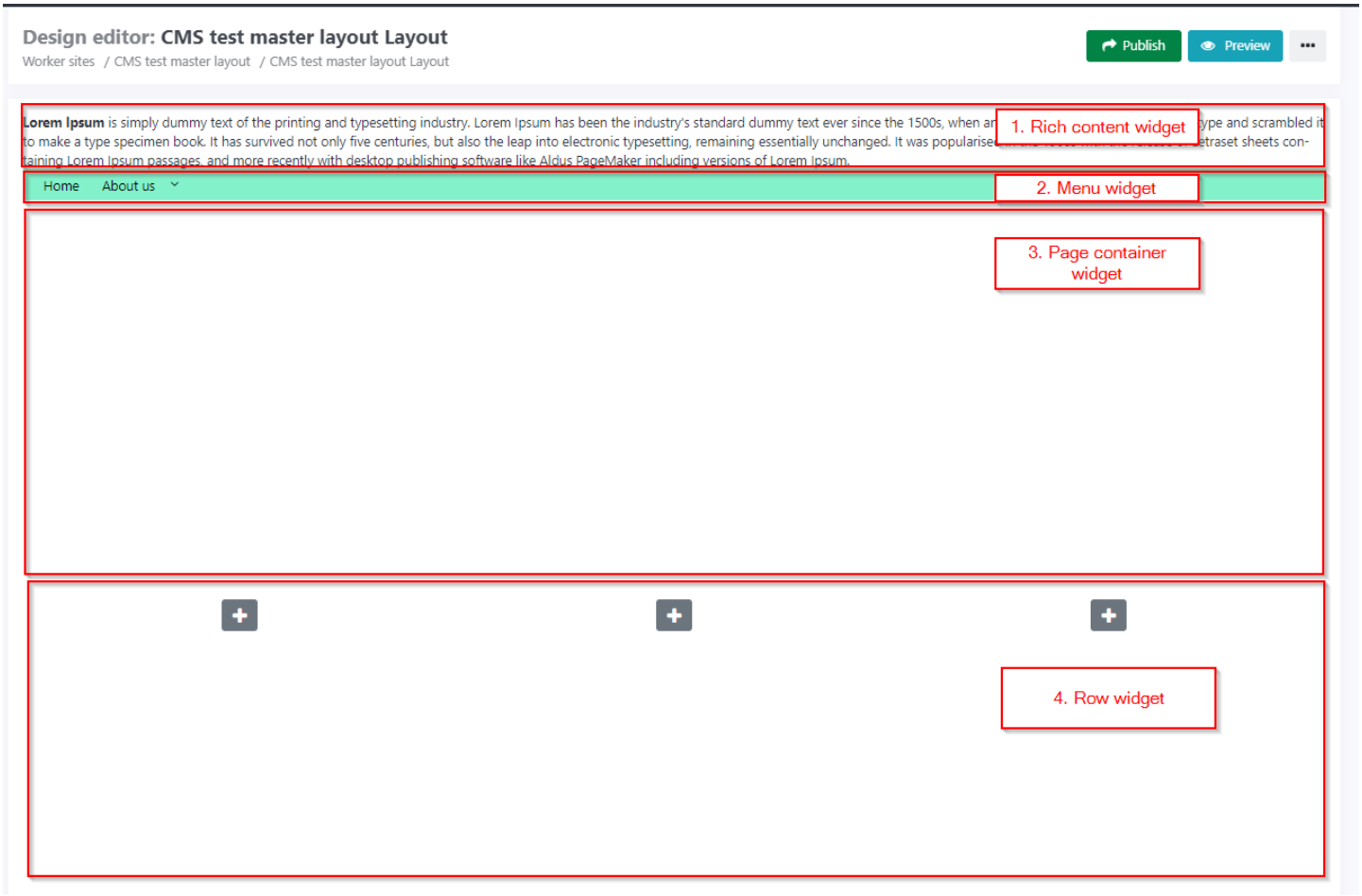


Page container widget is a placeholder for your pages, that you will learn how to config in the next sections.

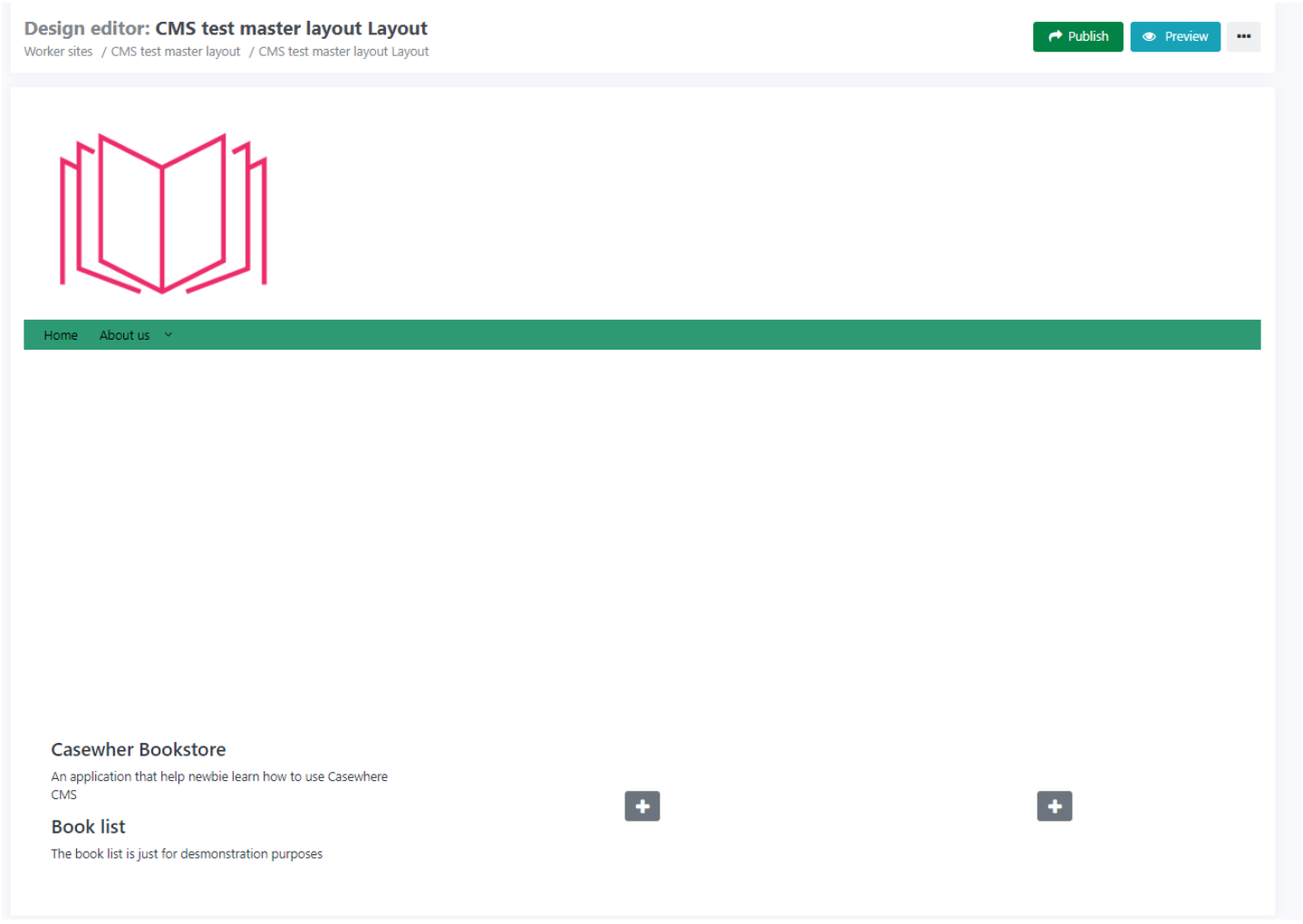
Step 3. Click the upper or lower plus icon  to add more widgets.



Step 4. We will add a **Rich Content** and a **Menu** widget on top of, and a **Row** widget under the **Page container** widget.



Now you can fill in the content placeholder with image and text.



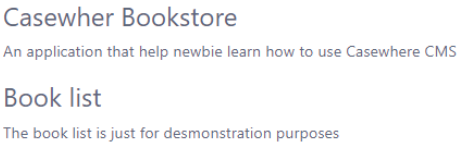
Go to page list 

This button will get you enter the page managements interface, read section **CMS page management** for more.

Publish all 

When you click this button, your master layout will be published, and you can see how it looks in a real site.

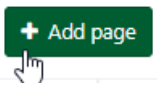




CMS pages management

Before you start create a new page, make sure your **Page Container** widget has been added to your master layout as a placeholder.

Create a new page

Click **Add page** 

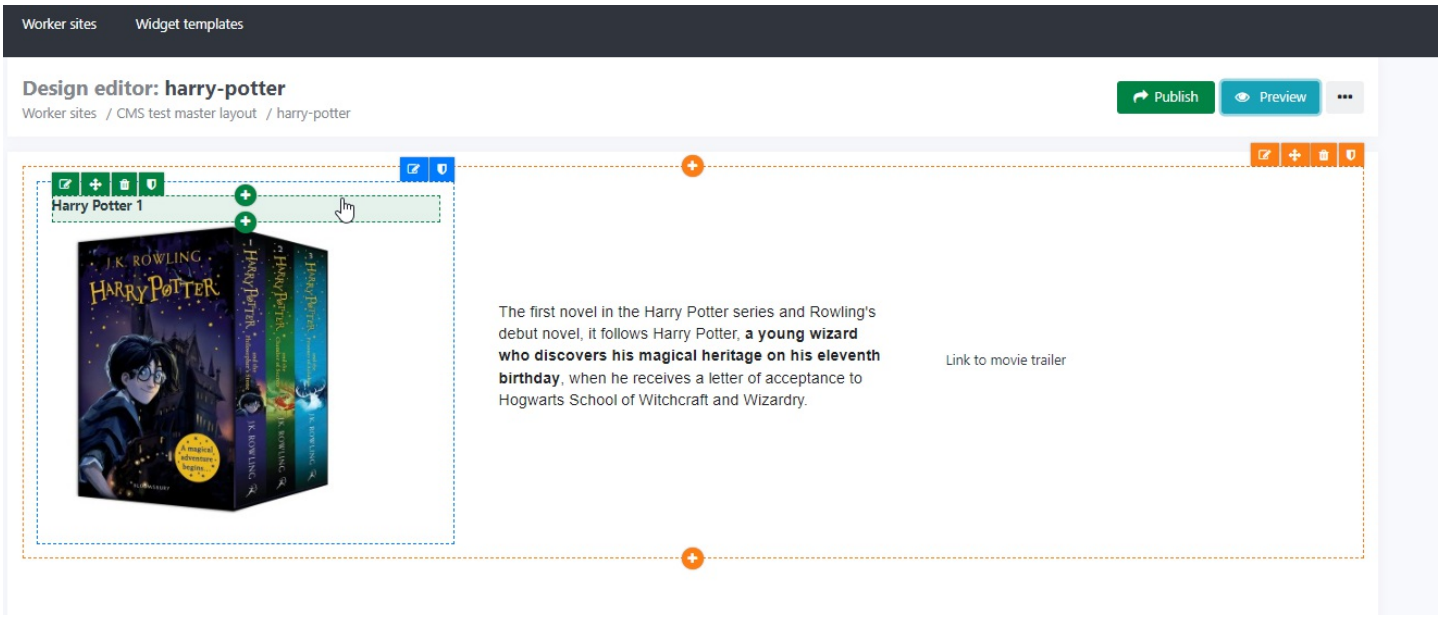
to create a totally new page.

Edit page design

Click **Edit page design** button  to enter page editing mode.

The designing concept of CMS is not changed even you are working with a master layout or a page.

Reference **Design your worker site** section to learn how to add and arrange widgets.



Publish

You can click **Publish** button  to immediately publish your page.

Preview

You can click **Preview** button  to preview the page appearance before publish.

Read more about publish and preview in Content publishing.

Copy a page

Click **Copy this page** button  to quickly duplicate a page and inherit from that page's design.

Setting

Click the **Setting** button  to edit page info such as title, parent page, and order.

Set a page as homepage

Click **Set at homepage**  button to set the page as homepage. Each worker site only have 1 homepage.

Access rule 

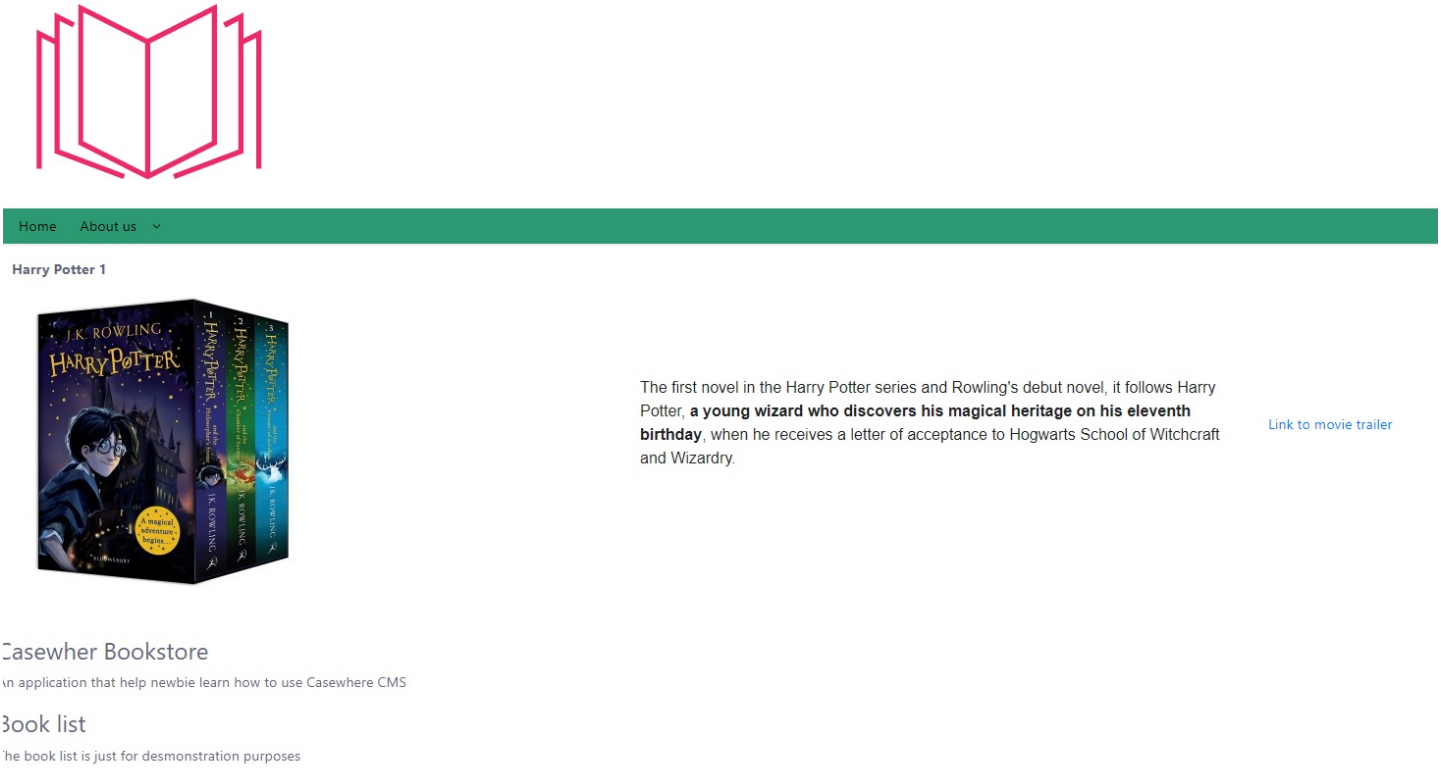
There are sites that should only be accessed by a specific group of identities.

For example, the application designers can set up access rules to show or hide specific widgets for certain user roles.

To set such restriction to a site, users need to add some rules, read more in CMS access rule (coming soon)

Publish a page

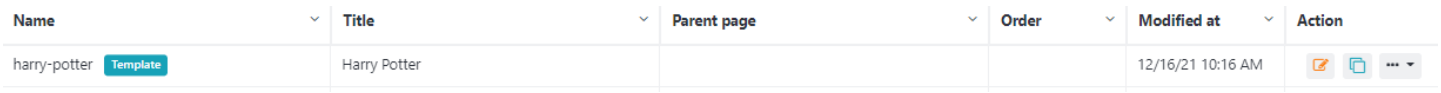
Click **Publish** button  to publish. The published page will have both master layout and the page content.



Mark a page as template

Click **Mark as template** button to add a page to the page template library.

So when you create a new page for the same worker site, you can inherit from another page's template and do not have to do double-works, especially if you want to keep the same page layout across some pages.



You can also unmark a page as a template Unmark as template . This action will remove the page from the page template library.

Unlink from worker site

You can click **Unlink from worker site** to seperate the page from its worker site. This action will not delete the page but will made it no longer accessible from the worker site.

Delete

You can click **Delete** to delete a page. Be careful with this action since the page once deleted can not be recovered.

CMS content publishing

Introduction

"Drafting, reviewing, updating... and repeat"

Creating website content from draft to finish is a long process.

Most of the time, the website administrator or designers must do the tasks and return to it several times before it is defined as finished.

In this case, CMS can help by keeping the site in draft mode Draft. This feature allows users to return to work on the application as many times as they want until the designer decides it is ready for publication.

The publishing action will make the content public on the internet and allow others to view it.

While the CMS handles everything in the backend, the user does not have to worry about file management, version overrides, or hosting issues and can instead focus on polishing the contents.

Casewhere CMS content can be shared via a direct link with clients or coworkers immediately once they are published.

Features

- Publish a page
- Publish a worker site
- Unpublish your site
- Scheduled publishing

Publish a page Publish

A page could be under multiple worker sites. However, publishing this page from one worker site does not make it public in the other.

Publish a worker site

Publish a worker site will make all the changes in the worker site, including all its pages, to be public. This action cannot be reverted, so there will be a confirmation popup to ask for the user's final approval.

Unpublish the site

You can unpublish the site. Unpublishing will work immediately, but you can always republish by access the publishing feature once again.

Scheduled publishing

Administrators can use scheduled publishing to specify the exact day and time that the publishing or unpublishing will proceed.

This feature prevents the need for the administrator to be physically present at the computer when the application already has the plan to start or to expire.

Publishing date

From date | Hour | Minute

M/d/yy

To date | Hour | Minute

M/d/yy

CMS searching

Introduction

The ability to search for static content across a whole website is a standard feature of many digital applications.

Casewhere CMS provides built-in widget that help you set up the feature with 4 main steps:

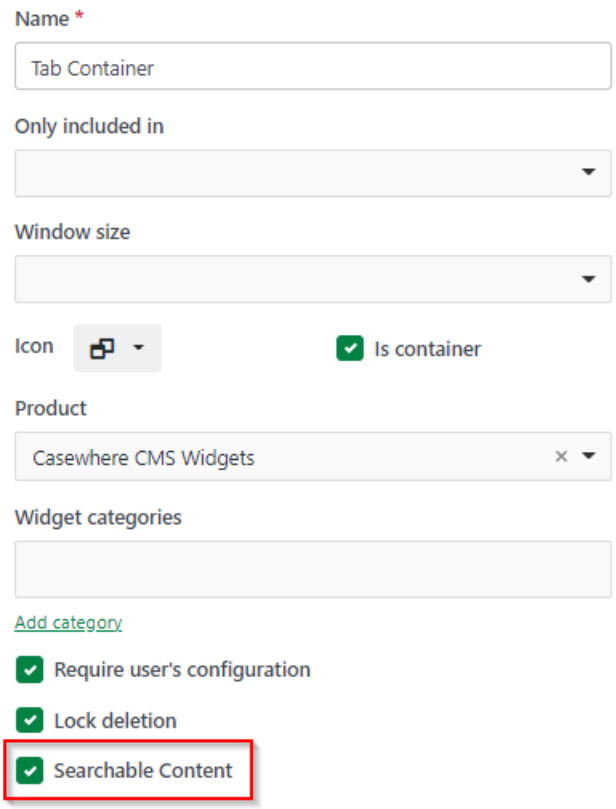
- Define searchable widget template - global
- Define searchable contents - on page
- Configure search box
- Configure search result page

Prerequisites

1. Casewhere CMS Widgets v1.0.1 or later

Define searchable widget template - global

Step 1. As widget template designers, you need to decide if the content of your widget can be searchable.



> **Hint:** If you're editing an existing widget template, remember to **push changes** to globally enable the search feature through all available widgets. **Push changes** will also update the widgets that already been added to your sites.

Step 2. You must tell Casewhere how to extract the searchable information from the widget's content.

If your widget template offers multi-language content editor, the simplest way is using the built-in directive `ca-multi-lang-editor`. You can learn from the **Rich Content** widget template.

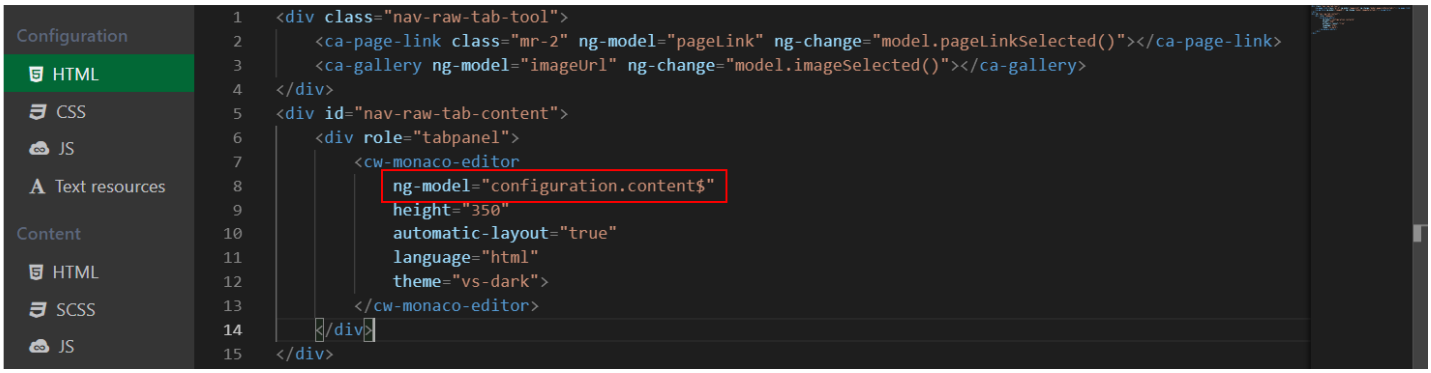
Edit widget template



You can use the suffix `$` to tell Casewhere which fields in your widget template are searchable. The field value could simply be plain-text, or a key-value object in case it has multi-language content:

```
{
    "en": "This is a test",
    "da": "Dette er en test"
}
```

Edit widget template



The widget searchable fields may contain text resources as part of their content. CMS automatically extracts text resource values from the content to enrich data for searching. Currently, the CMS indexer supports two syntaxes as below:

- `<ANY>{{'text_resource_key' | translate}}</ANY>`
- `<p ng-bind-html="'text_resource_key' | translate"></p>`

Define searchable contents - on page

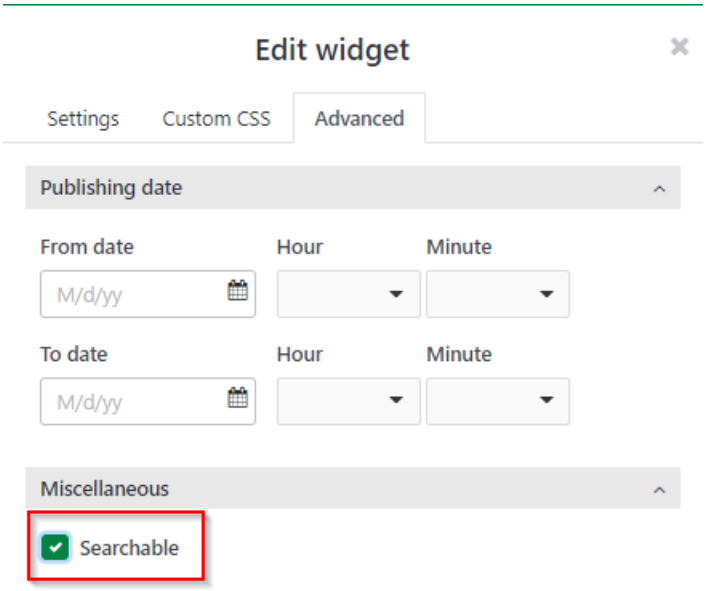
Your CMS page contains multiple widgets. CMS has a setting to help you define which widget of your website will be searchable. This setting is only available if the corresponding widget template supports it.

Step 1. Go to the page, select a widget, click **Edit** button 

Edit widget dialog will open

Step 2. Switch to **Advanced** tab, scroll to **Miscellaneous** section

Check **Searchable**

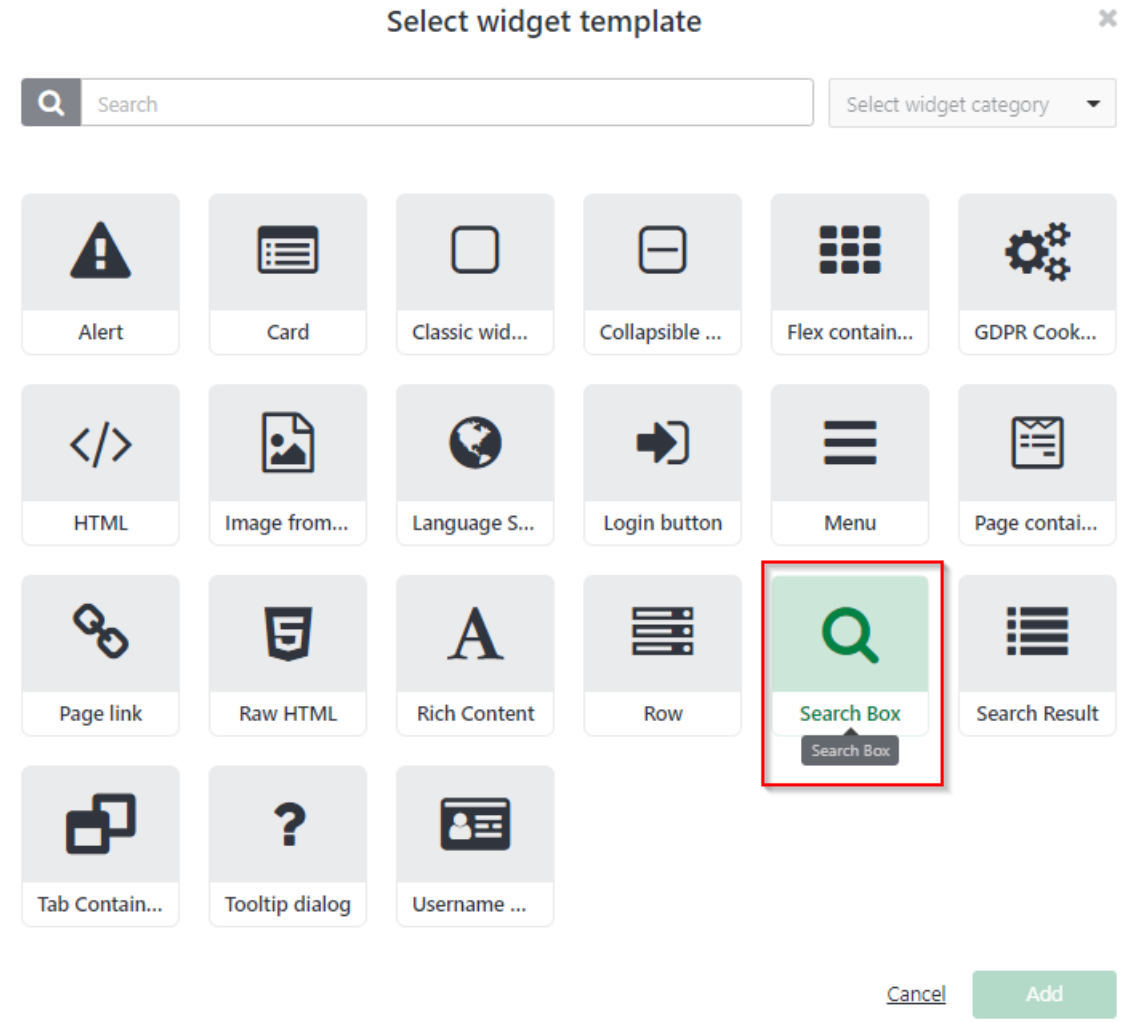


Configure search box

You can add a search box anywhere in your page. Whether it is on top of the home page or inside a row, column, panel, or flexbox widget.

In **Select widget template** dialog, just looking for **Search Box** widget.

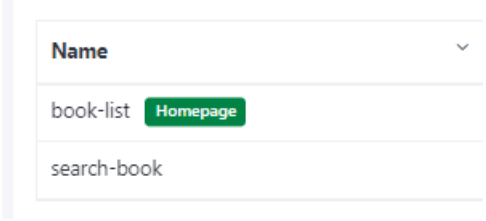
Select the widget, and click **Add** to include to your page.



Configure search result page

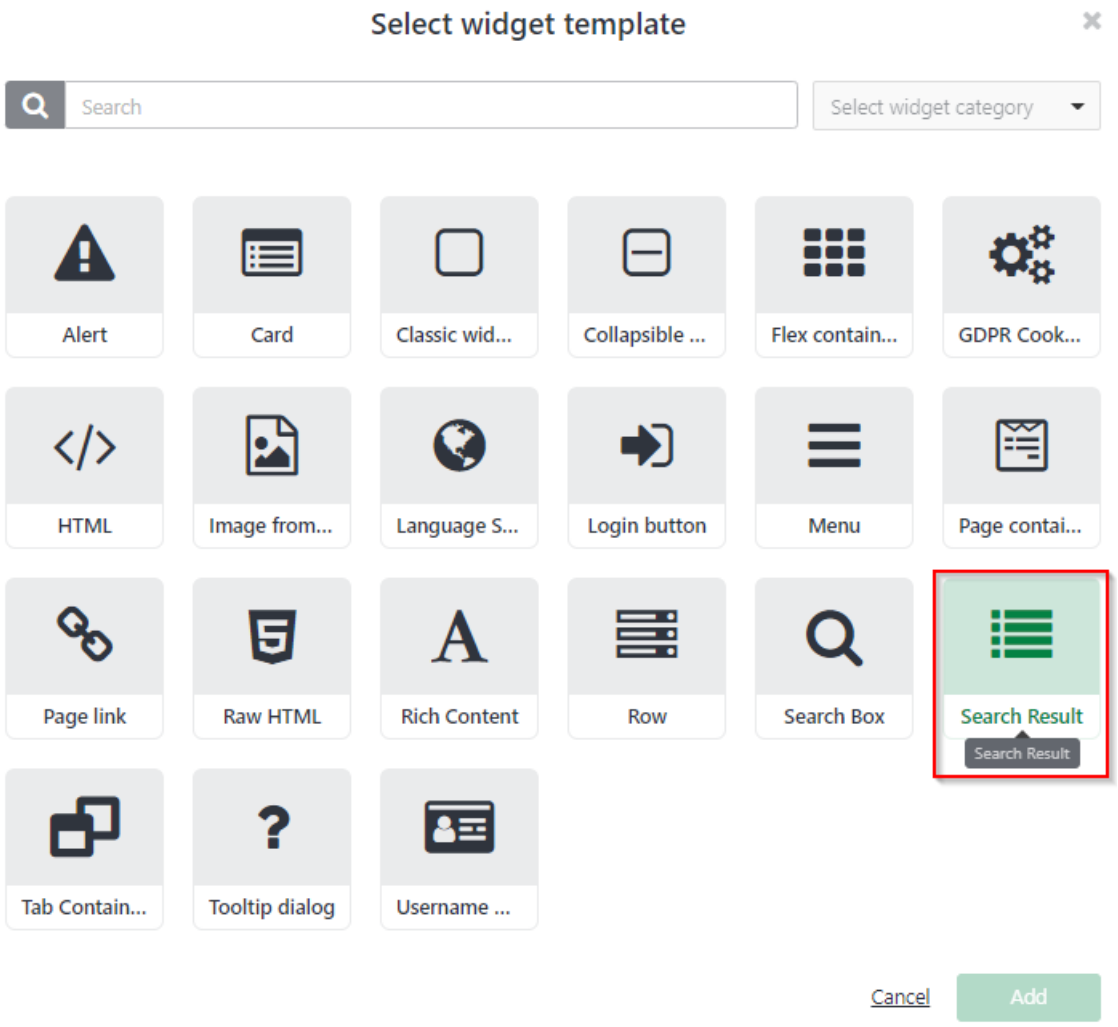
You will need to create an individual page to display the search results.

> Take a CMS page of the bookstore tutorial as an example. The application will need at least 2 pages:
>
> **book-list** page contains all the book data and the search box
>
> **search-book** page to present the search results



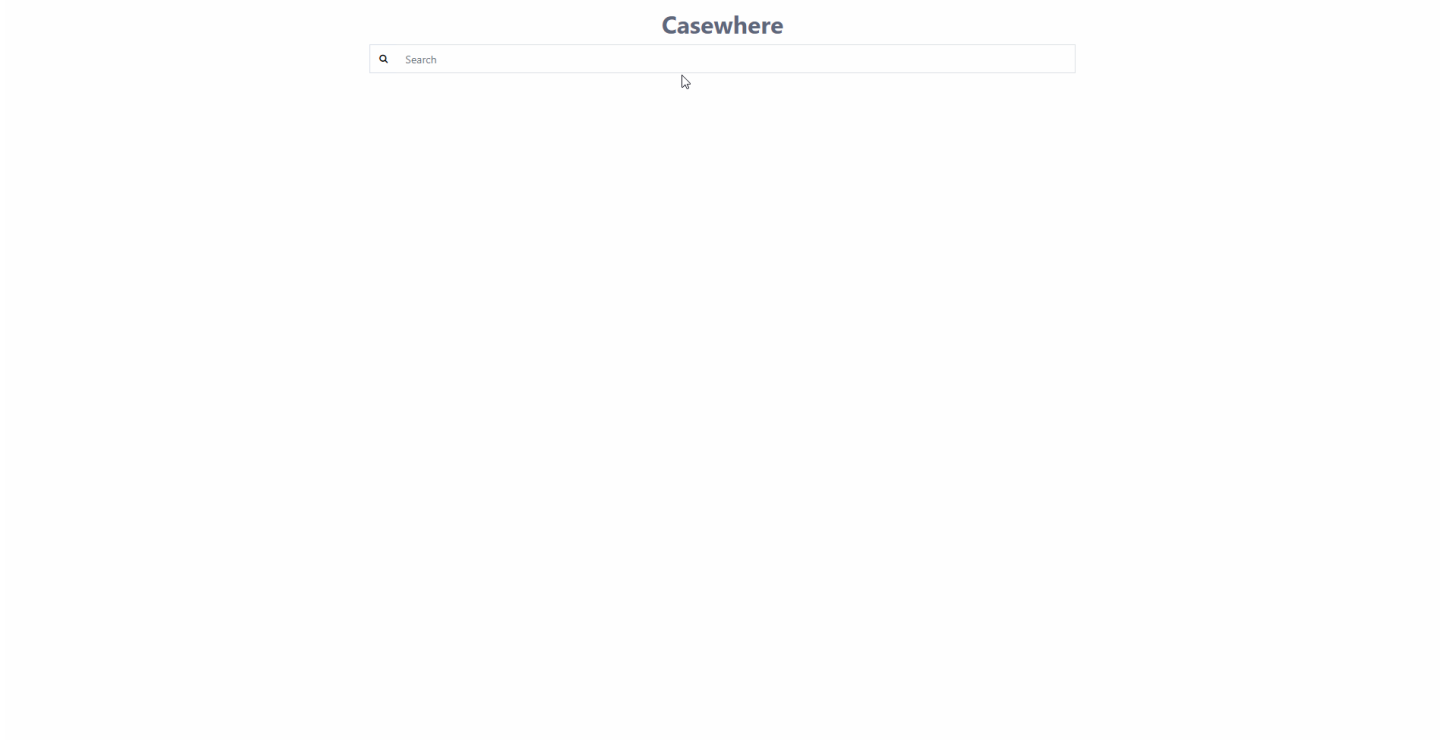
In **Select widget template** dialog, just looking for **Search Result** widget.

Select the widget, and click **Add** to include to your page.



Test

The image below shows a simple example to test the search function. When the user searches for "Lorem" in the **Search Box** widget, the search engine will display all matching results in the **Search Result** widget. By default, Casewhere will search content with the language users selected in the worker site.


Casewhere
Search

Theme overview

A theme is a collection of template files that define an application's appearance. A theme is more than just graphics; it's where an application developer alters the application's appearance (graphics, styles), rearranges the layout, overrides default behaviors, and adds additional components to extend functionality.

Casewhere worker site can always be customized using a theme. The theme eliminates the need for the applications interface to be hard-coded. An application can create its own theme library or inherit from another application.

Config theme

In your Casewhere instance, navigate to **Theme**

Click **Create** button 



Create New Theme dialog will open with 3 main working areas:

- Theme info
  - **Name** (required): allow users to define a unique name for the theme
  - **Description** (optional): allow users to add details description about the theme
- Menu
  - **Casewhere Components:** list of all Casewhere default components
  - **Custom Components:** list of all Casewhere custom components, where user can create their own template
- Coding area
  - **Use default template**: check to enable read-only mode, uncheck to start editing the code

Themes management

Introduction

Theme is a package of customized files that define the appearance of worker sites. This includes styling of form controls, buttons, text formats, modals, utilities,.. basically everything that your users will see and feel.

Theme components

Theme components uses HTML, CSS and JavaScript together to form the front-end design of a worker site.

Theme components depends on its purpose can be designed to manipulate content, style or interactivity of the site.

There are two type of theme components in Casewhere:

- Casewhere components
- Custom components

Both could be edited and managed by Casewhere admin site.

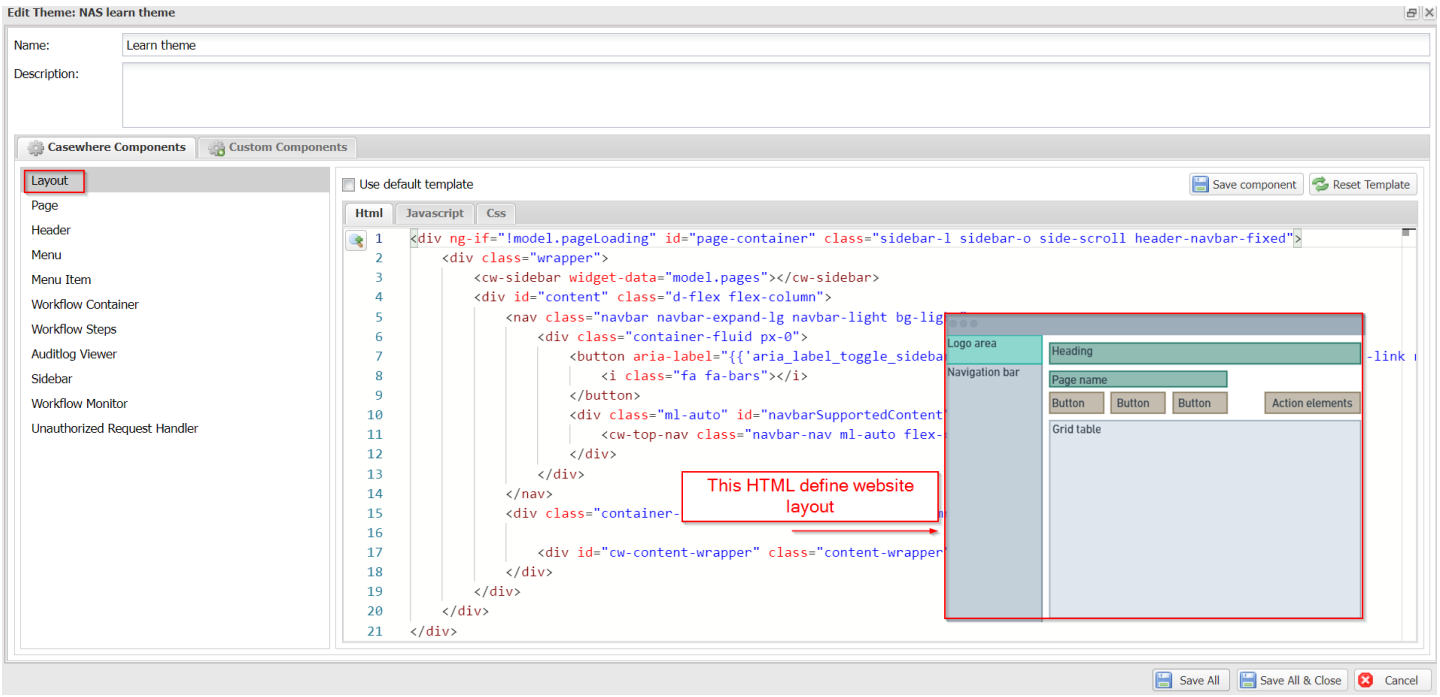
Casewhere components

Casewhere theme component is a default package.

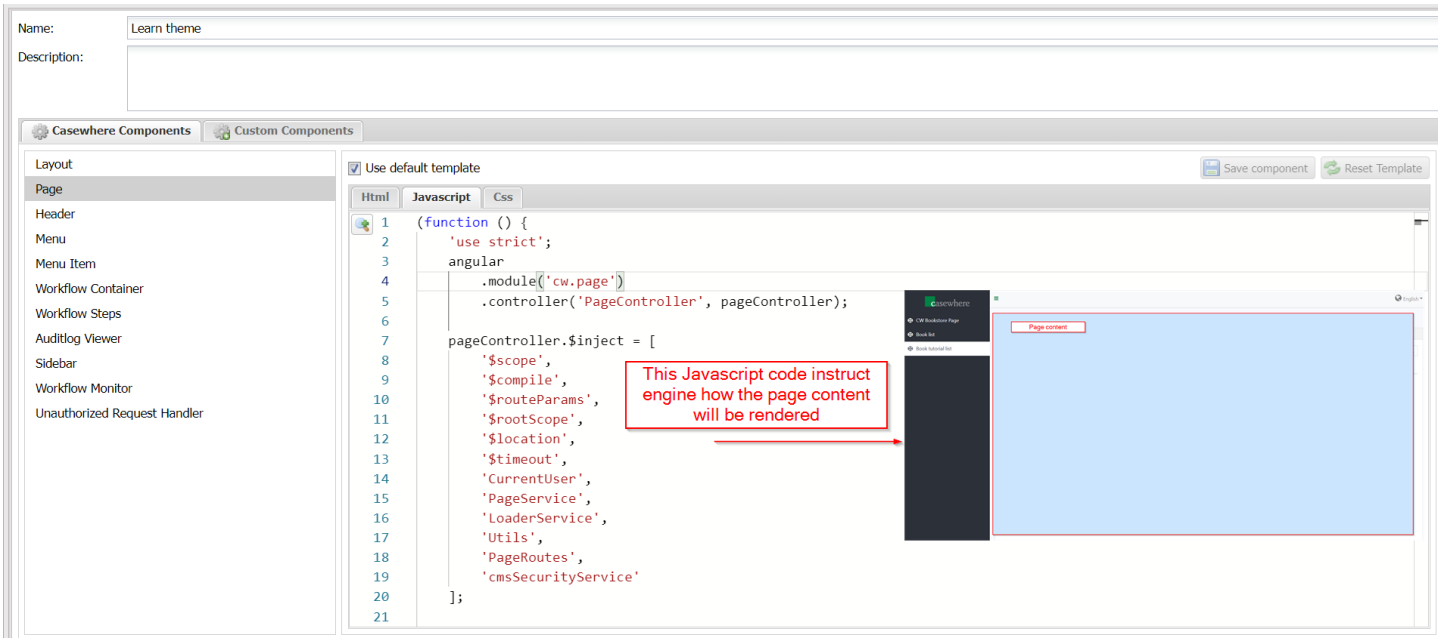
This package includes all necessary components that create a working worker site.

Check below for a walk through all this default components and how they work.

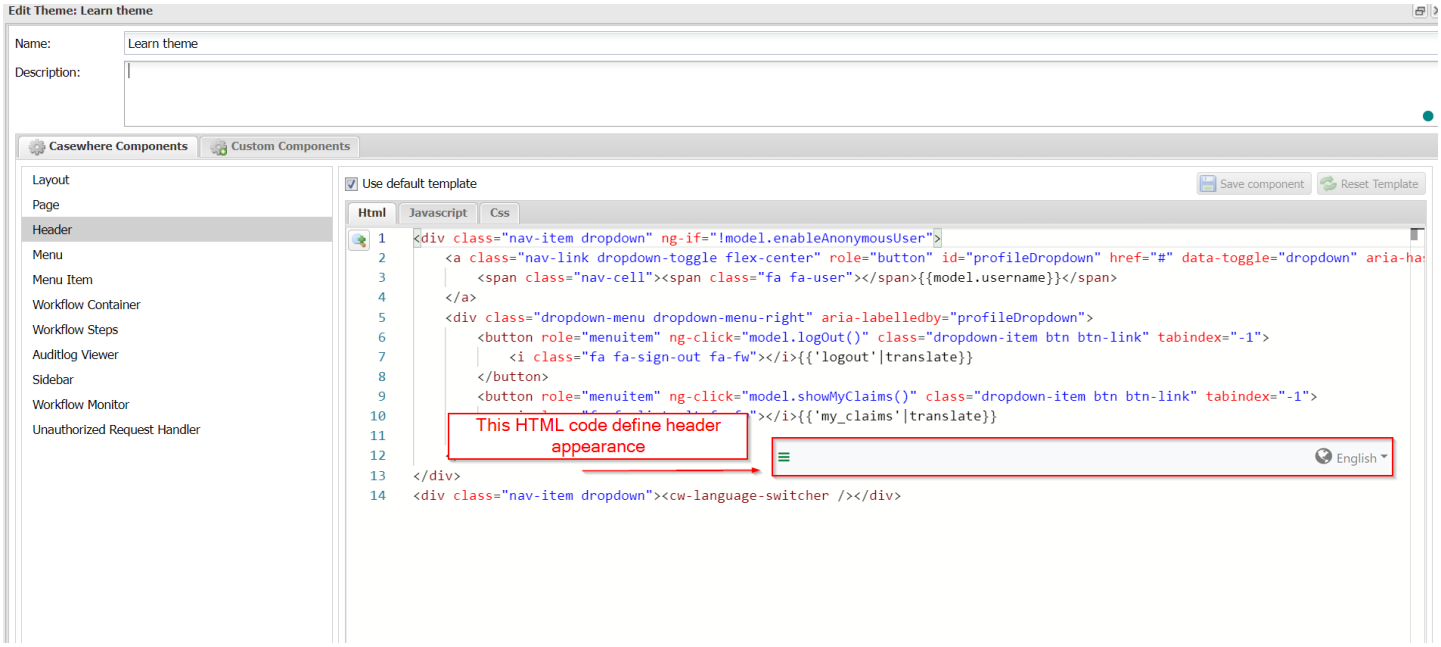
Layout



Page


Name:
Learn theme
Description:
Casewhere Components
Custom Components
Layout
Page
Header
Menu
Menu Item
Workflow Container
Workflow Steps
Auditlog Viewer
Sidebar
Workflow Monitor
Unauthorized Request Handler
Use default template
Save component
Reset Template
Html
Javascript
Css
(function () {
'use strict';
angular
.module('cw.page')
.controller('PageController', pageController);
pageController.$inject = [
'$scope',
'$compile',
'$routeParams',
'$rootScope',
'$location',
'$timeout',
'CurrentUser',
'PageService',
'LoaderService',
'Utils',
'PageRoutes',
'cmsSecurityService'
];
This Javascript code instruct engine how the page content will be rendered
Page content

Header

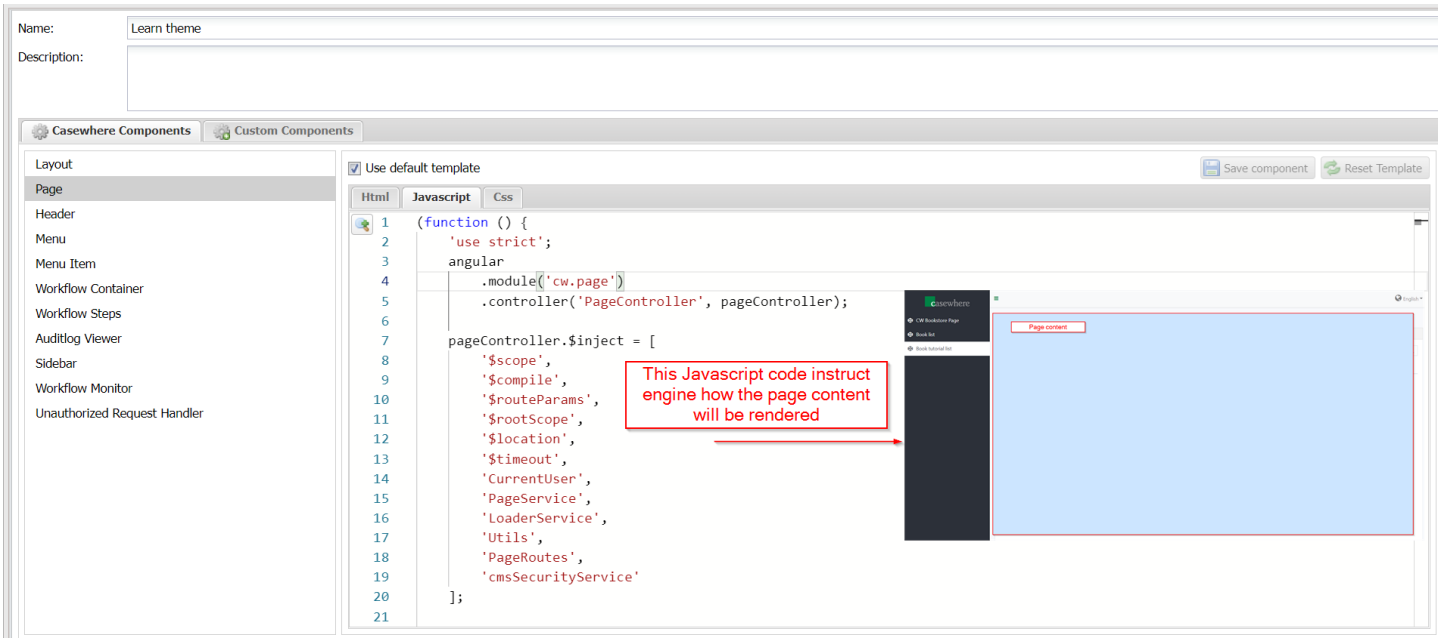

Edit Theme: Learn theme
Name:
Learn theme
Description:
Casewhere Components
Custom Components
Layout
Page
Header
Menu
Menu Item
Workflow Container
Workflow Steps
Auditlog Viewer
Sidebar
Workflow Monitor
Unauthorized Request Handler
Use default template
Save component
Reset Template
Html
Javascript
Css

<a class="nav-link dropdown-toggle flex-center" role="button" id="profileDropdown" href="#" data-toggle="dropdown" aria-ha
{{model.username}}

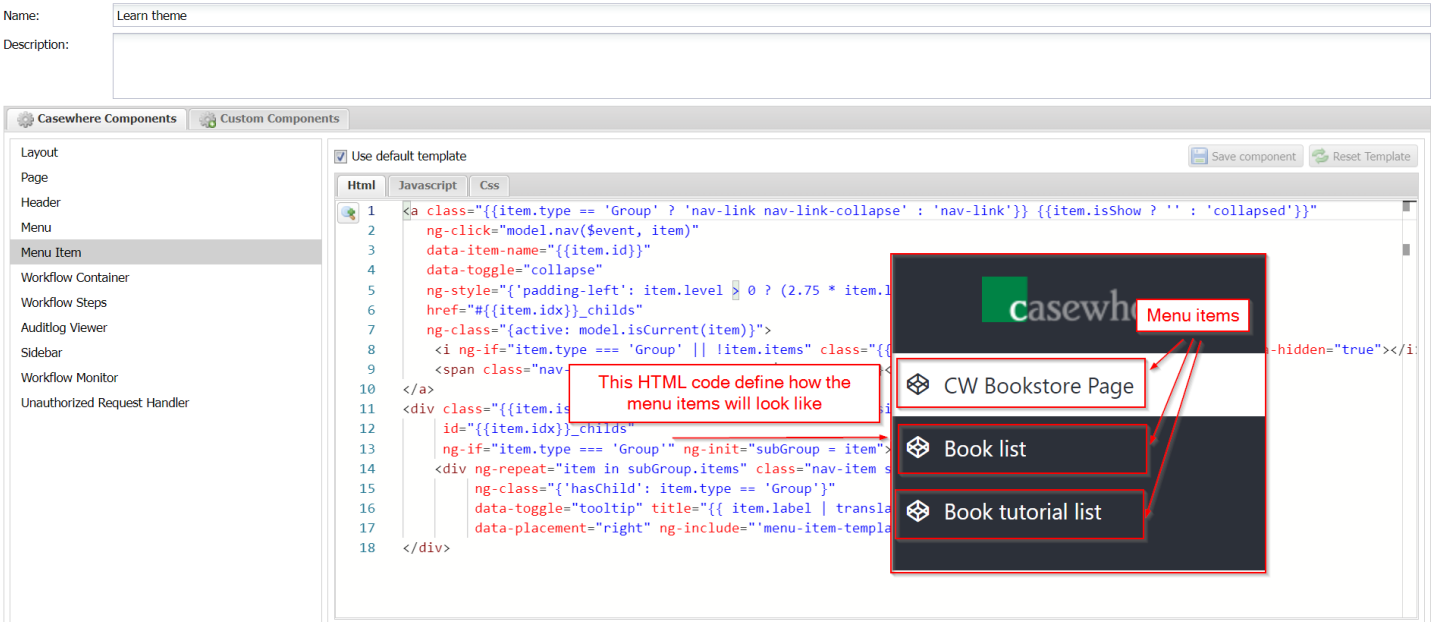
<button role="menuitem" ng-click="model.logOut()" class="dropdown-item btn btn-link" tabindex="-1">
{{'logout'|translate}}
</button>
<button role="menuitem" ng-click="model.showMyClaims()" class="dropdown-item btn btn-link" tabindex="-1">
>{{'my_claims'|translate}}
This HTML code define header appearance
English

<cw-language-switcher />

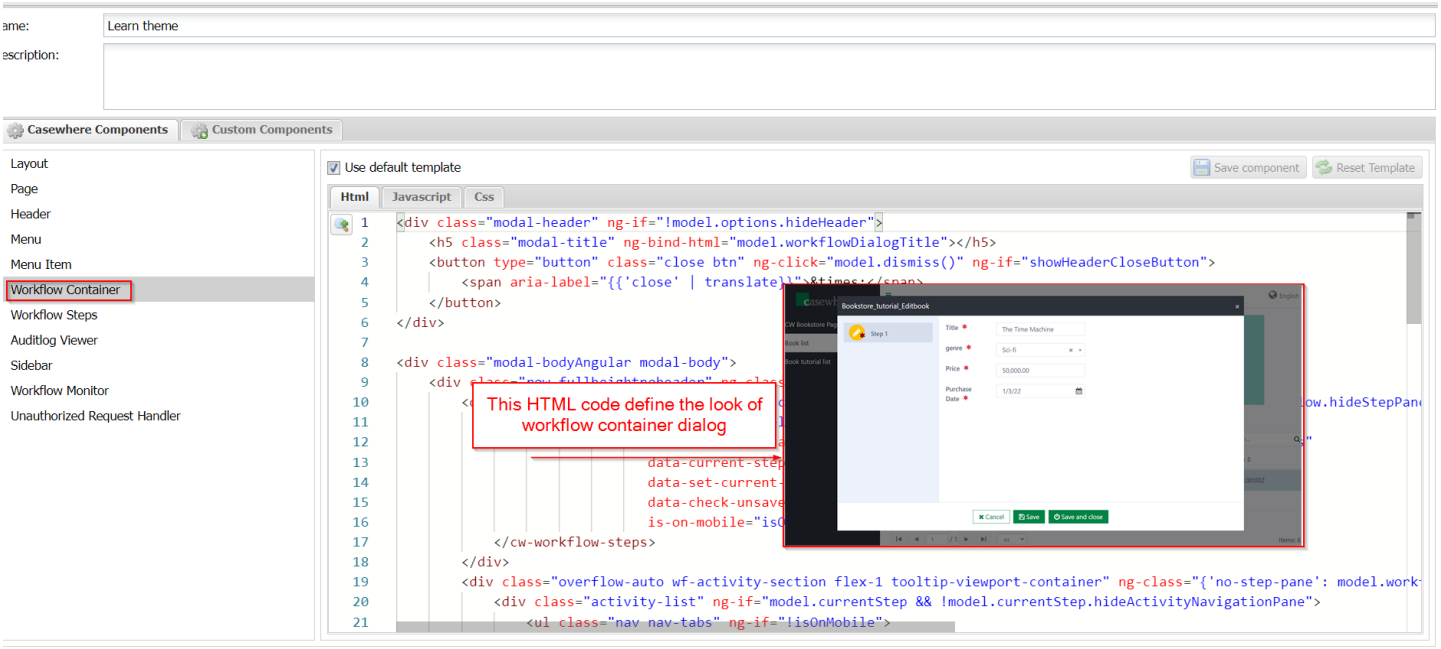
Menu



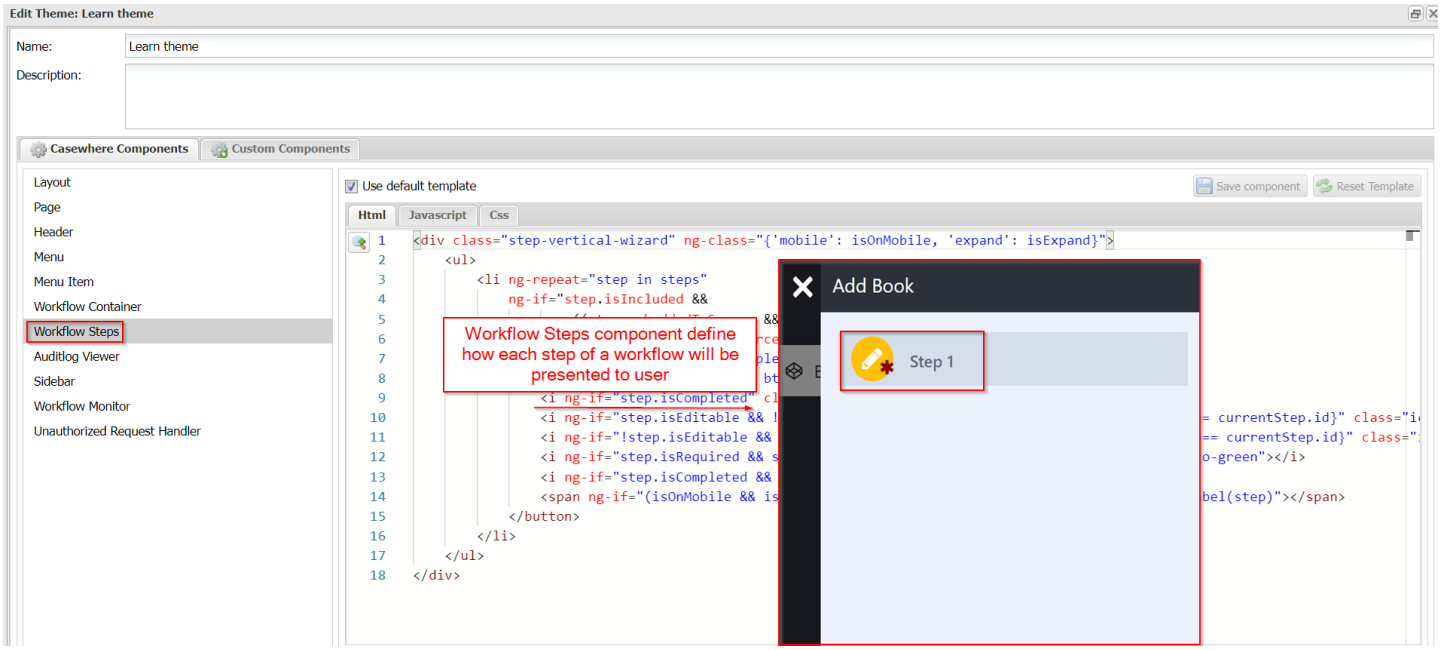
Menu items



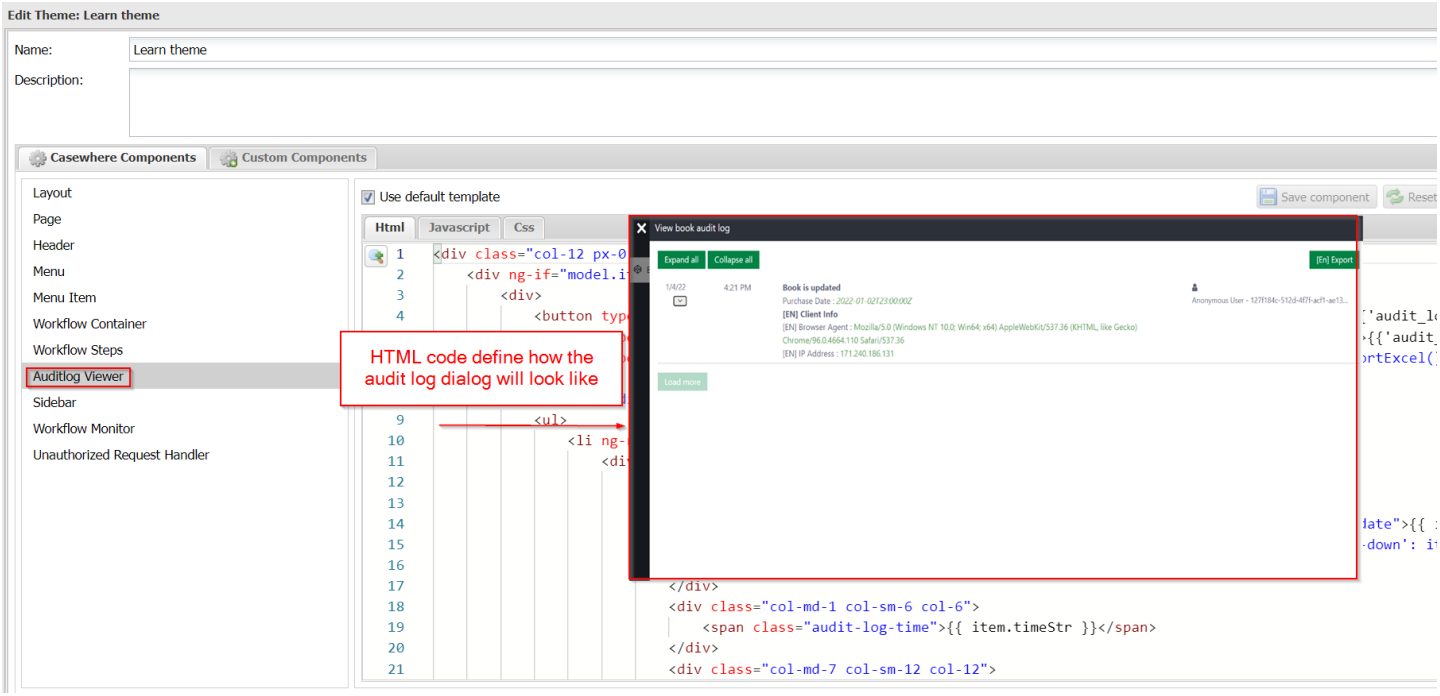
Workflow Container



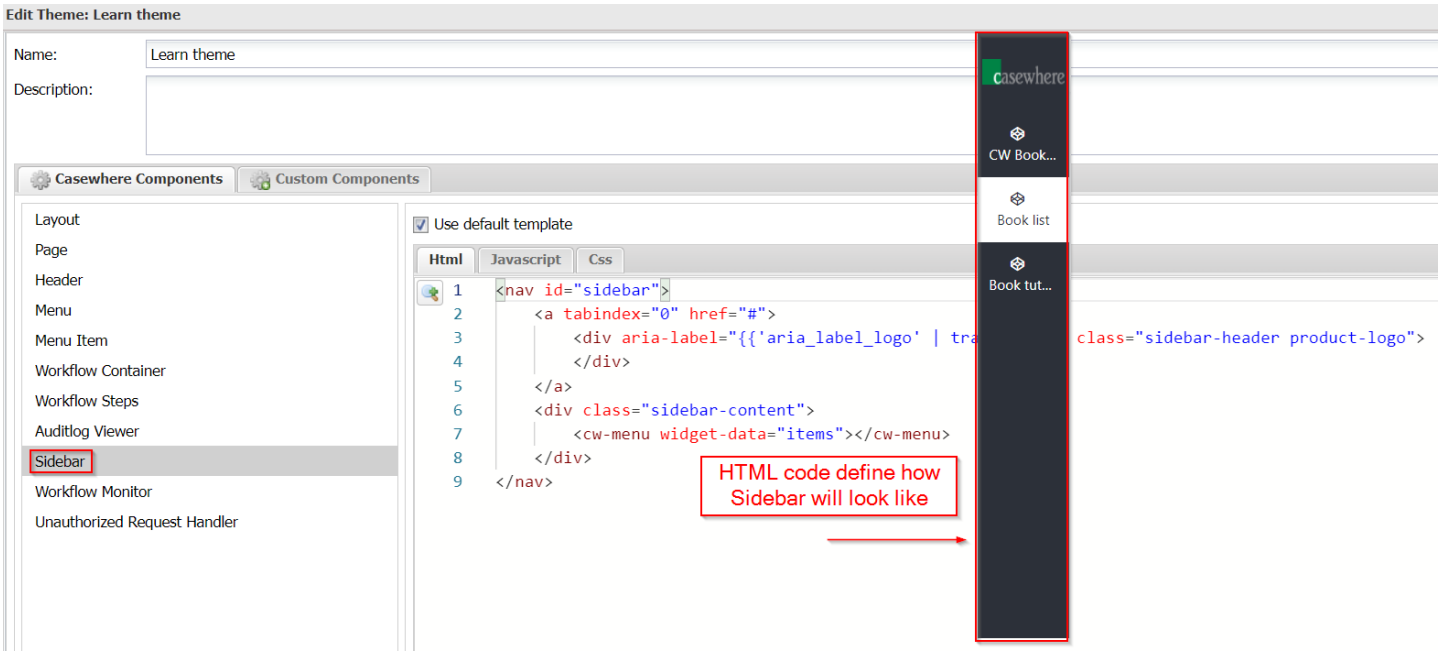
Workflow Steps



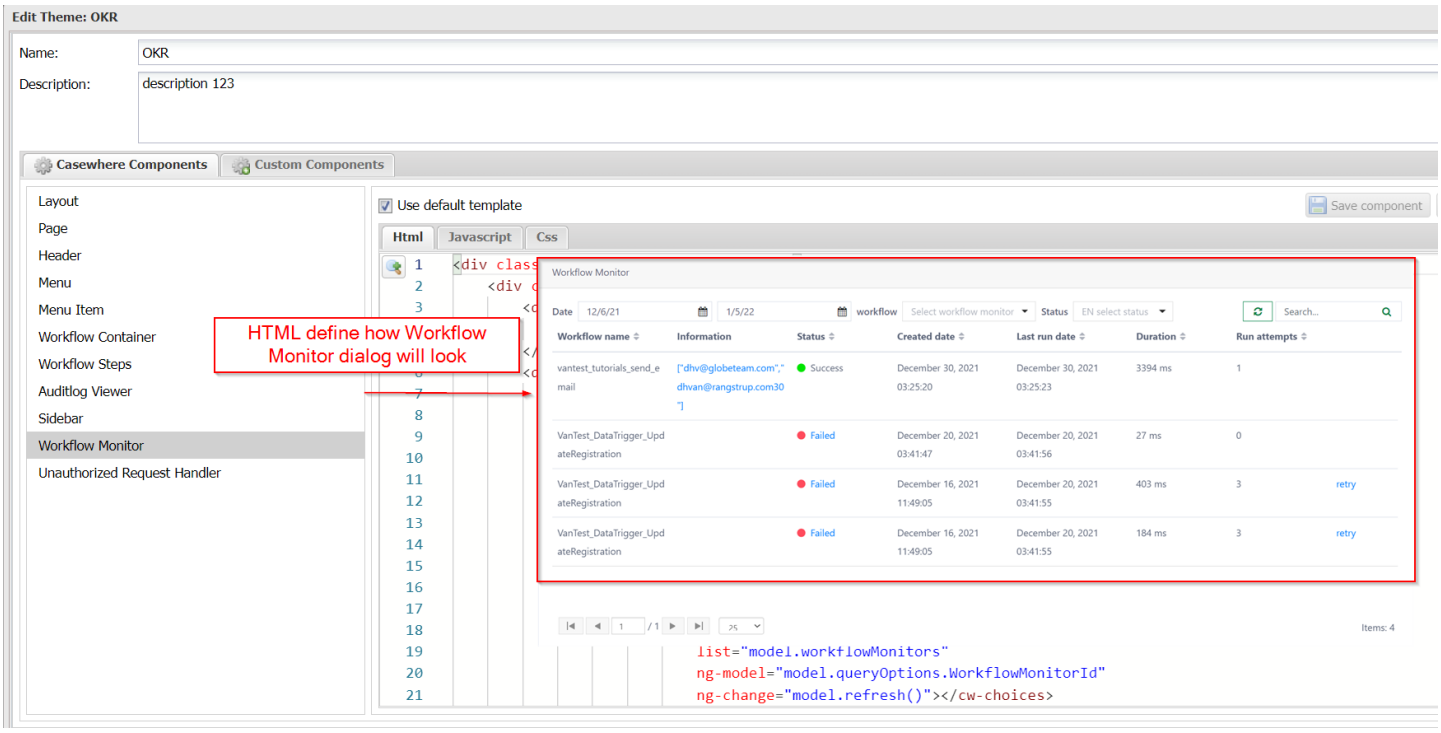
Auditlog Viewer



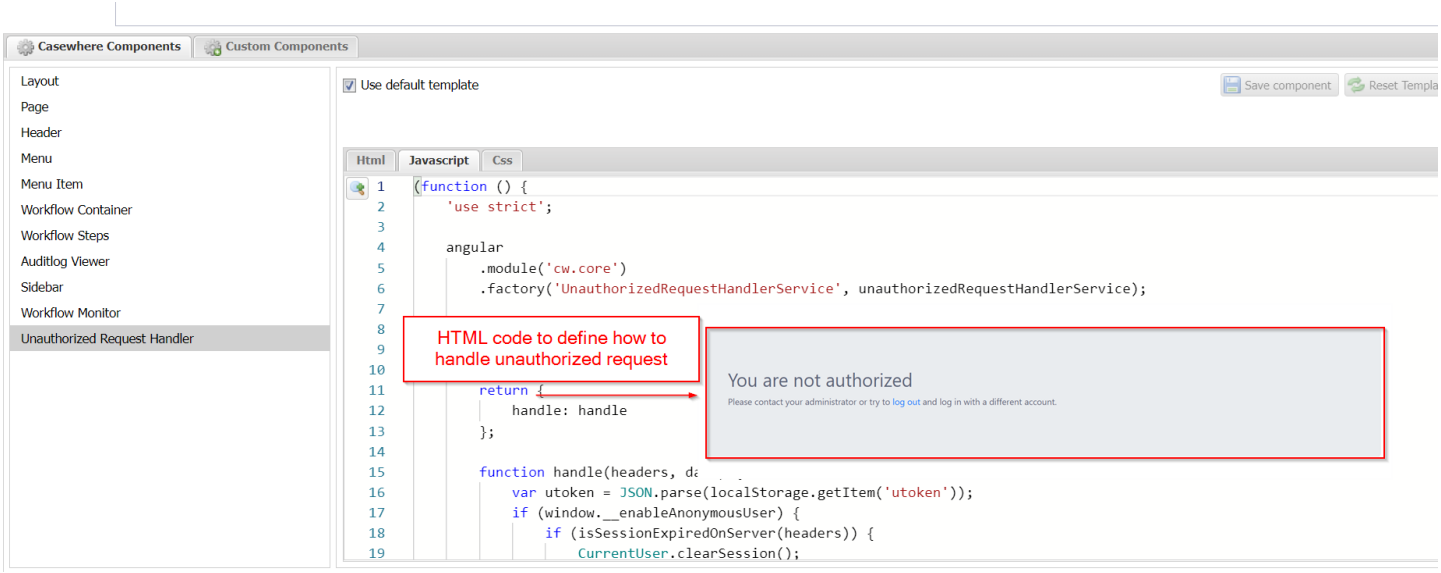
Sidebar



Workflow Monitor



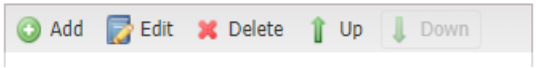
Unauthorized Request Handler



Custom component

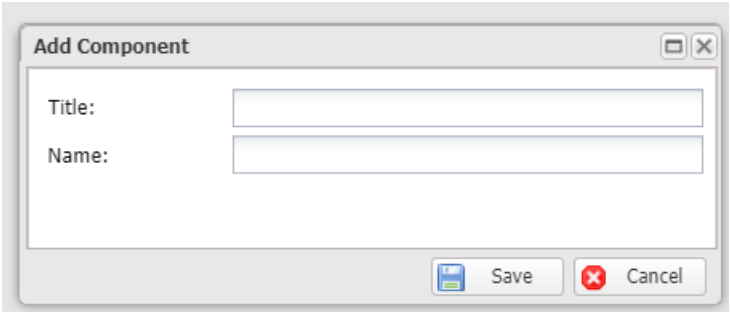
Casewhere has a built in editor, allow developer to create theme custom components.

The editor support **Add**/ **Edit**/**Delete** and re-ordered the component list with **Up** and **Down** buttons.

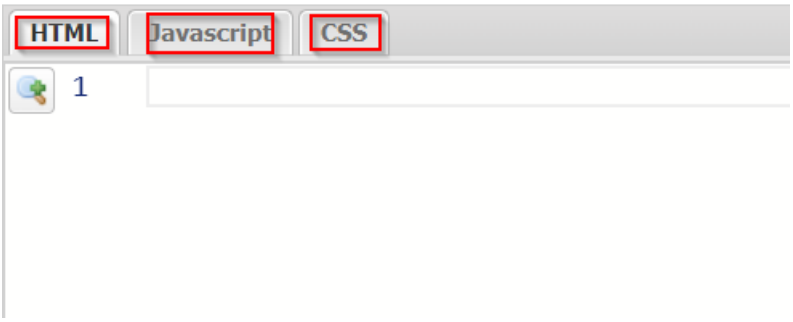


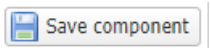
Custom component guide

- Step 1. Navigate to **Custom Components** tab
- Step 2. Click **Add** button, fill in component **Title**, the field **Name** will be auto generated



- Step 3. Fill in HTML/Javascript/CSS code

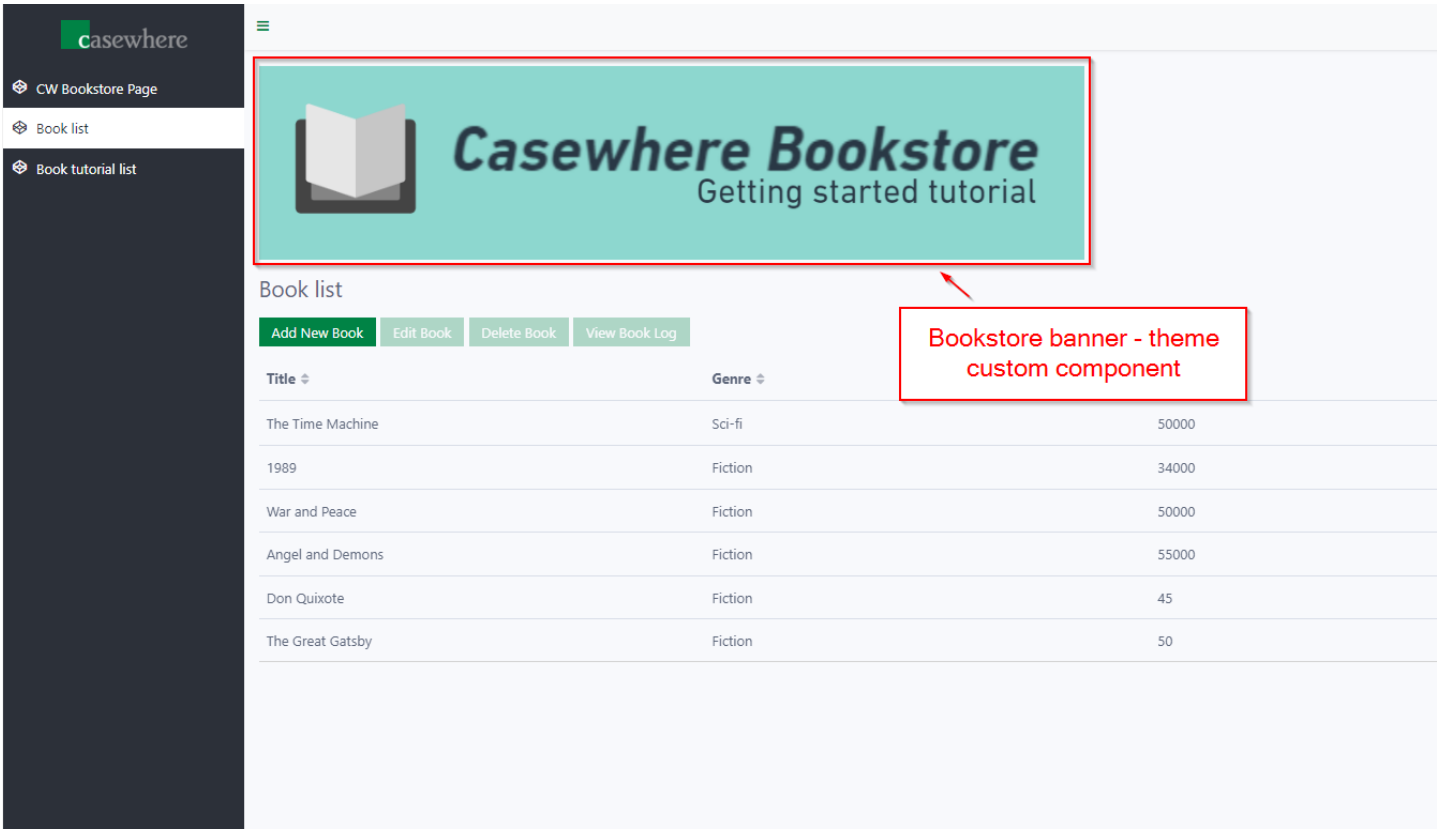


Click **Save component **button 

Click **Save All & Close**

Custom component example

Says you want to make a banner for your bookstore tutorial site.



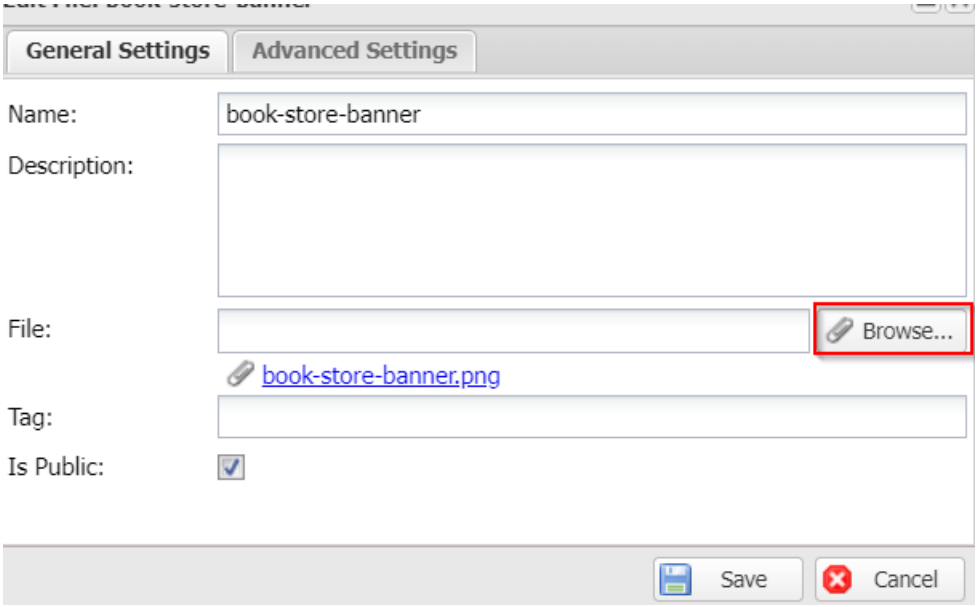
- Step 1. In Casewhere instance, navigate to **Files**



Click **Create** button to open **Create file** dialog

Fill in "book-store-banner" in **Name** field

Click **Browse** to browse to the banner image file



Click **Save**

- Step 2. Follow the **Custom component guide** in the above section. For the codes, please copy and paste the following:

HTML

```
<div class="bookstore-banner__container">
    <img class="bookstore-banner__img" src="[[ApiEndPoint]]system-documents/book-store-banner" alt="Book store banner" />
</div>
```

Javascript

CSS

```
.bookstore-banner__container {
    margin-bottom: 1rem;
}
```

- Step 3. Navigate to **Layout** in **Casewhere component**

Find the following div:

```
<div class="container-fluid flex-grow d-flex flex-column">
```

Add this 1 line of code into div:

```
<bookstore-banner></bookstore-banner>
```



Click **Save component**

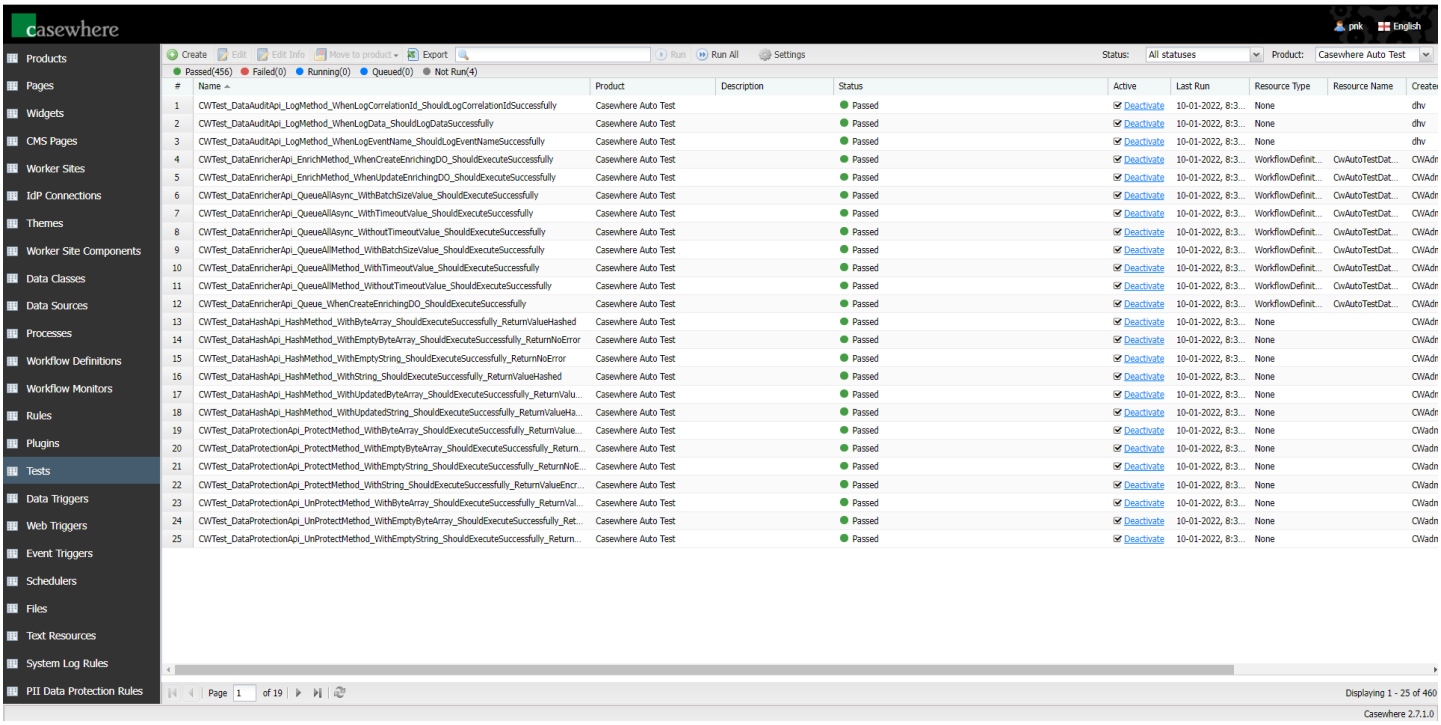
Click **Save All & Close**

Automation test overview

Tests have to be repeated often during development to ensure quality. Every time configuration or script is modified, tests must be repeated. For each release of the application, manually repeating the tests is costly and time-consuming. By understanding the need to have automated tests, Casewhere provides a testing framework integrated deeply into Casewhere Admin where you can write test for complex components such as workflow, rule, plugin, etc. Below are the highlights of Casewhere automation test:

Friendly interface for managing tests

You can manage tests like any product resource. In fact, it's a good practice to have tests included as part of the application solution. In Casewhere Admin, you can quickly review the testing status and run all test cases with one click.

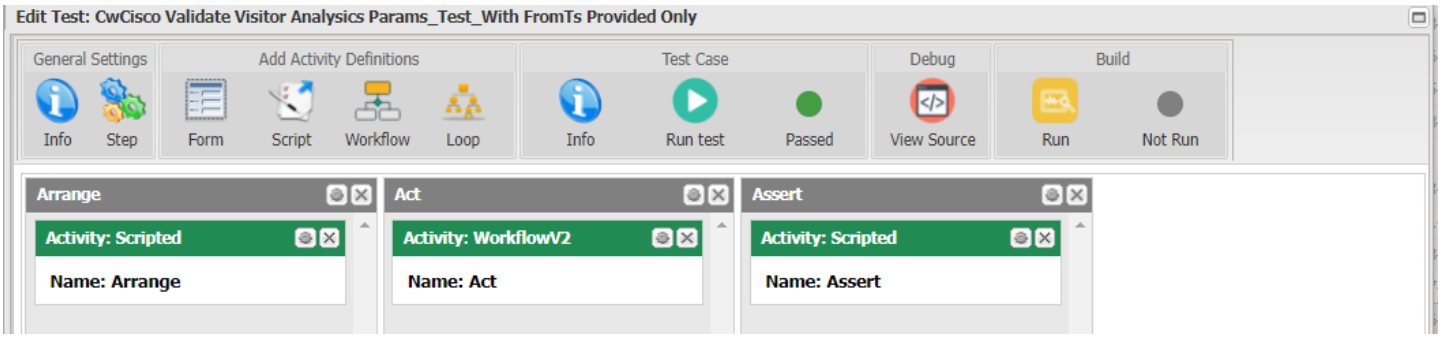


Casewhere is test-driven

You can write tests for different features and components in Casewhere. You can quickly review the test coverage of your solution, e.g., how many test cases a workflow or a rule has, how many of them are **Failed**, **Passed**, or **Not Run**. When you modify a resource, all tests related to it will be changed to **Not Run** immediately, so you know you need to run the tests again.

Test templates

Casewhere provides different test templates depending on the resource under test. For example, when you create a test for a workflow, Casewhere will generate the test using the below **AAA** template. **AAA** stands for **Arrange-Act-Assert**. As you can see, the workflow under test is embedded in the test workflow in the step **Act**.



Test APIs

Casewhere provides a rich set of APIs to help you make tests more efficient such as generating test data, simulating user interaction, impersonating users with different roles, verifying test results, etc.

DevOps overview

A Casewhere solution typically comprises multiple Casewhere products. While exporting a single Casewhere product and importing it into another environment is feasible, deploying an entire solution or a portion of it across different environments can be challenging.

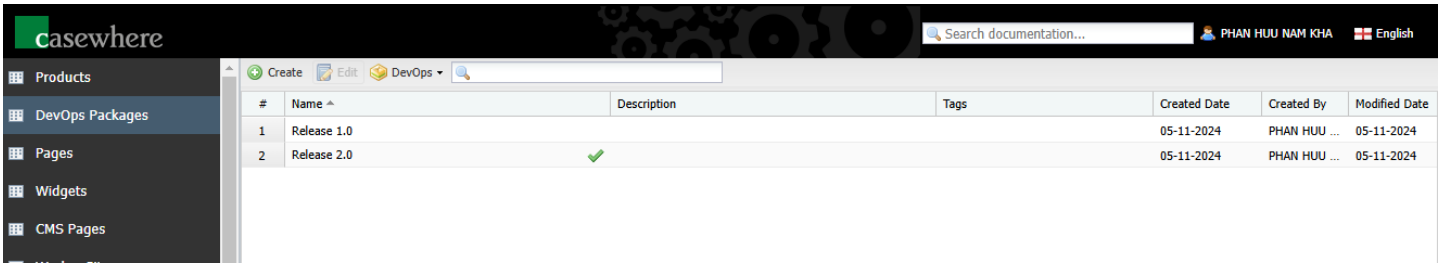
Casewhere DevOps is designed to facilitate solution deployment by enabling your development team to organize deployments into a single package. This package can be handed off to an IT operations team without requiring detailed knowledge of its contents, enabling them to deploy it quickly through Casewhere Admin with minimal effort. This functionality helps you manage and track your solution releases efficiently. Read more here to learn about working with Casewhere DevOps.

Below is an example of a typical deployment pipeline. A single deployment package progresses through multiple environments before reaching production.

```
flowchart LR
    DEV-->|Internal test|QA
    QA-->|User acceptance test|PRE-PROD
    PRE-PROD-->|Golive|PROD
```

DevOps packages

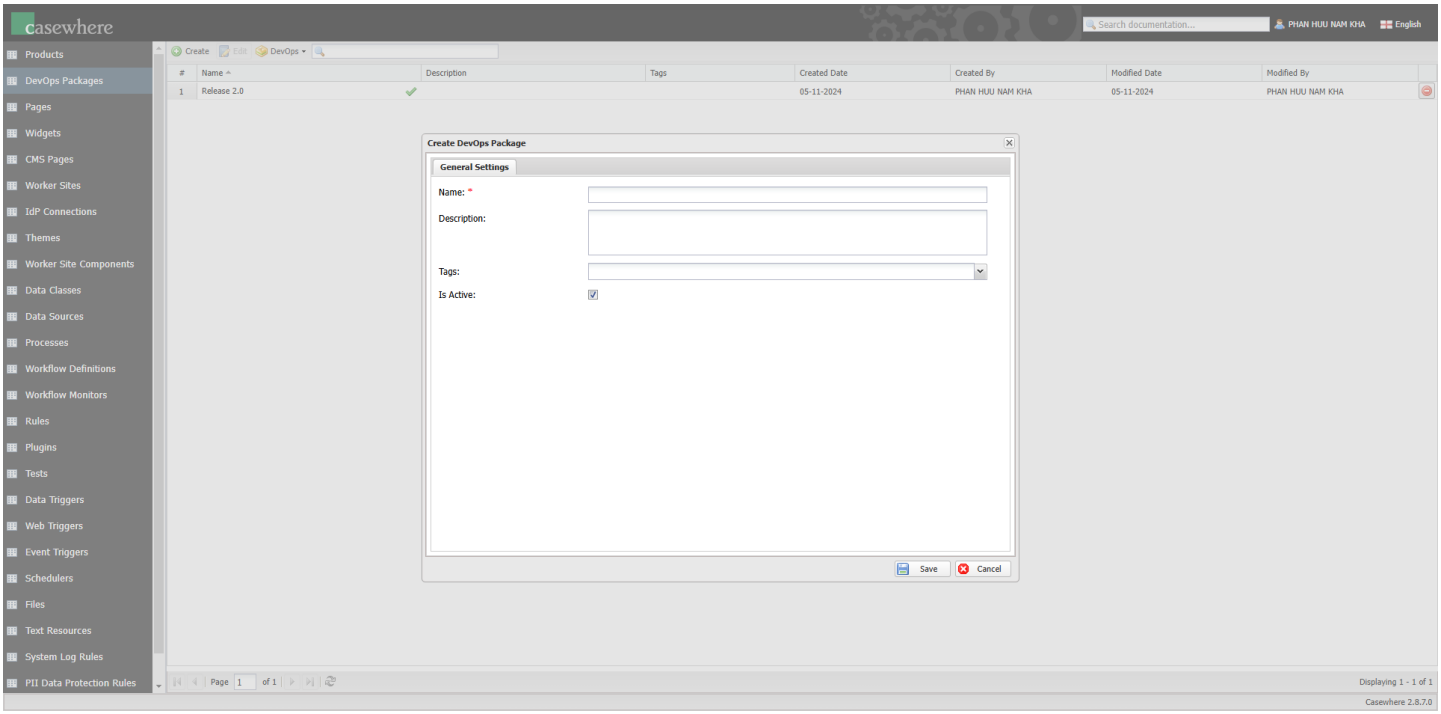
In Casewhere Admin, navigate to **DevOps Packages** to create and manage packages for your solution deployment.



Create a DevOps package

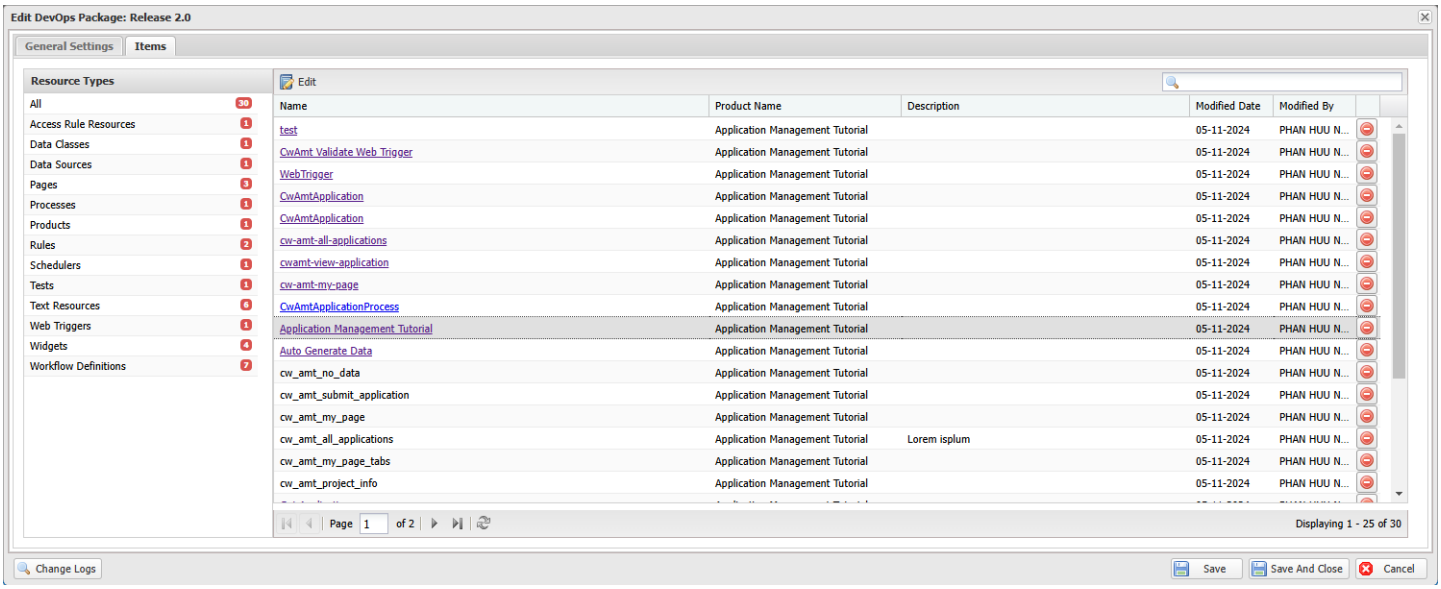
On the **DevOps Packages** page, click **Create** and fill in the following fields:

- **Name**: This field is required for the package name.
- **Description**: Provide additional details about the package, such as its purpose.
- **Tags**: Use tags to facilitate management.
- **Is Active**: Only active packages are accessible within Casewhere Admin.



Edit a DevOps package

On the **DevOps Packages** page, select a package and click **Edit**. The following dialog will appear, allowing you to edit the package.



In the dialog, you can view all items added to the package. You can also add comments to individual items and delete items that are no longer part of the deployment. For adding items to the package, read here to learn more.

Delete or deactivate a package

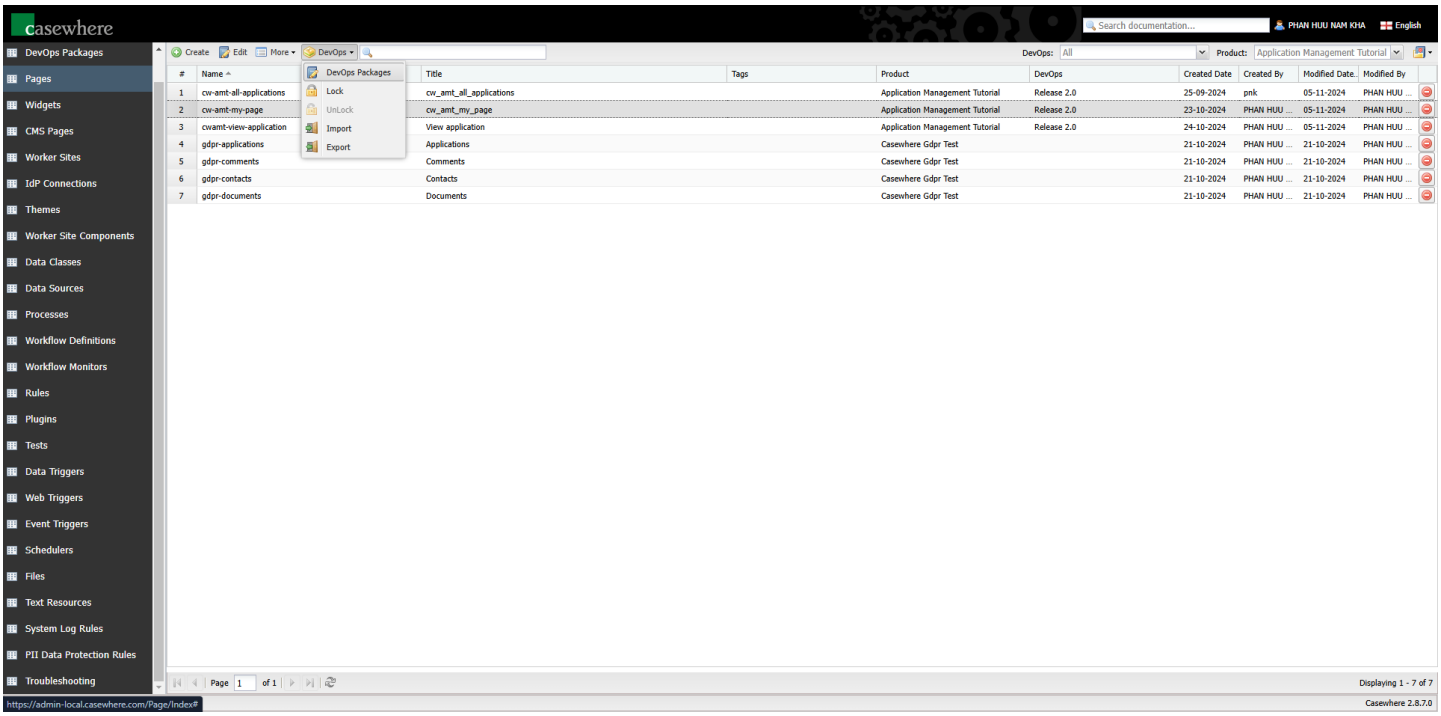
While you can delete a package, it is recommended to retain all deployed packages for historical purposes. Deactivating these packages is a more appropriate choice, as deactivated packages will be inaccessible from other functionalities in Casewhere Admin.

Working with Casewhere DevOps

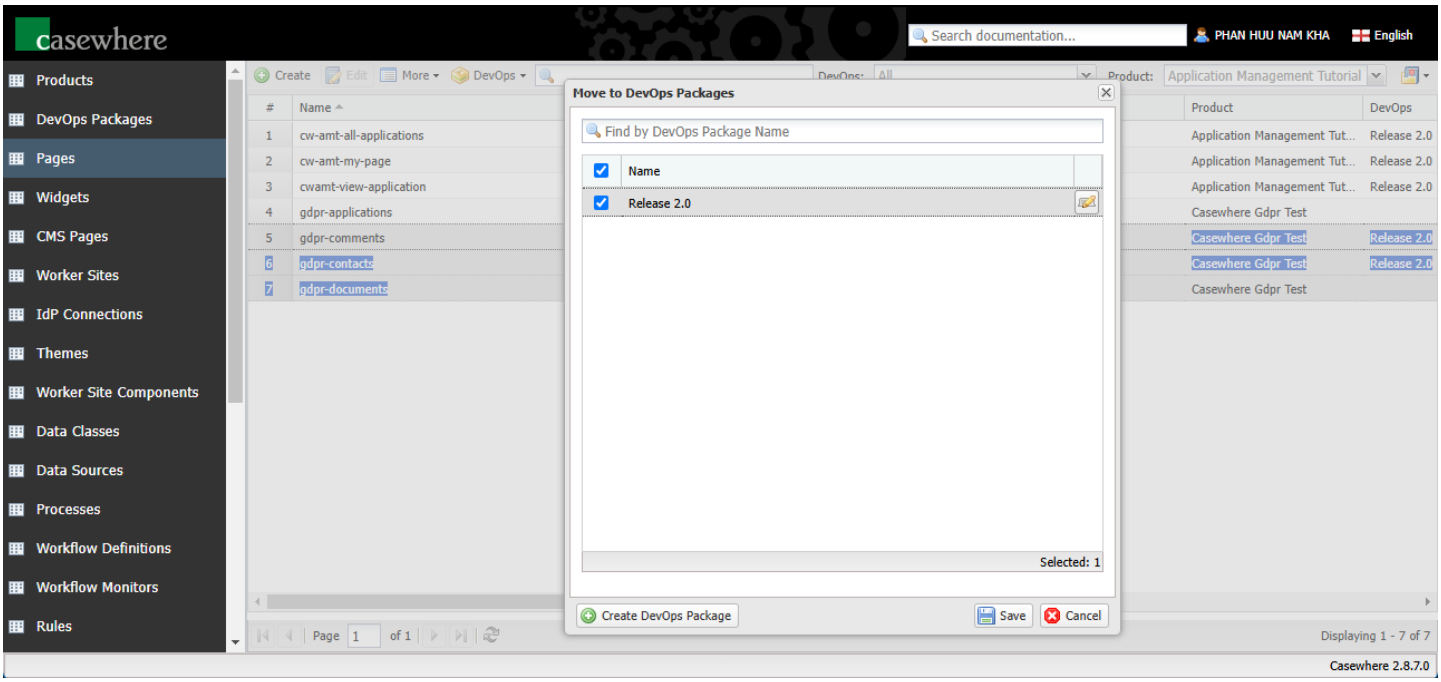
This article will explain how Casewhere DevOps works in practice and highlight some best practices to consider. For creating DevOps packages, read here for more information.

Add/remove items

Since Casewhere 2.8.7, you can find the DevOps menu in all management pages under Casewhere Admin.



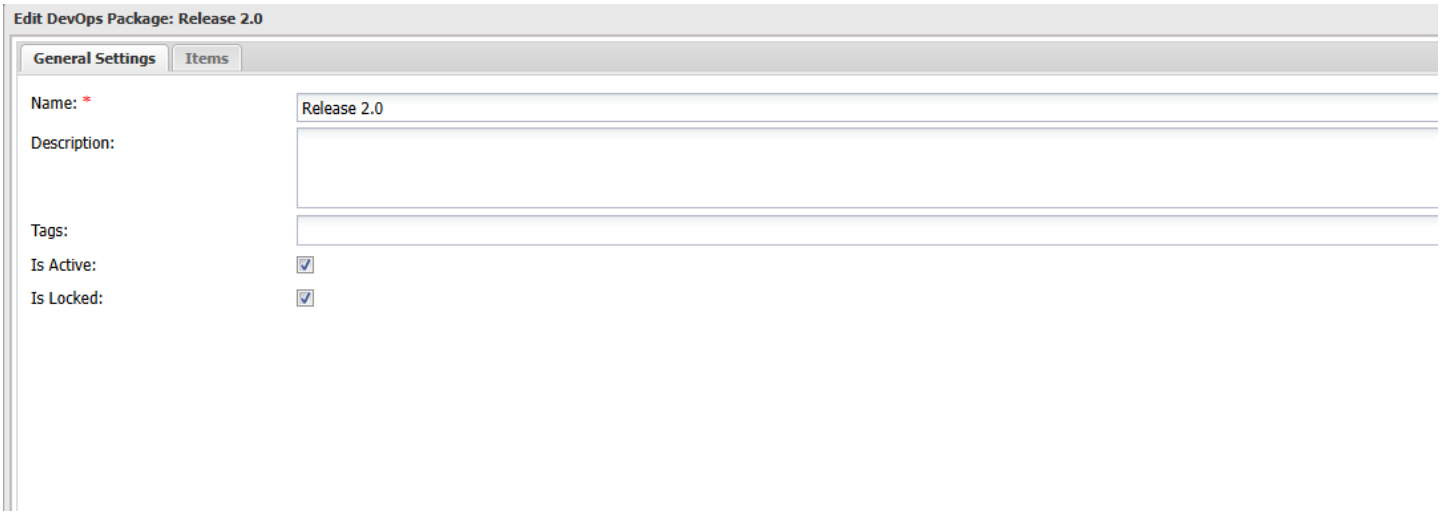
To add items to DevOps packages, simply select one or more resource items and click **DevOps / DevOps Packages**. A dialog will appear, allowing you to select the DevOps packages to which you want to add the items.



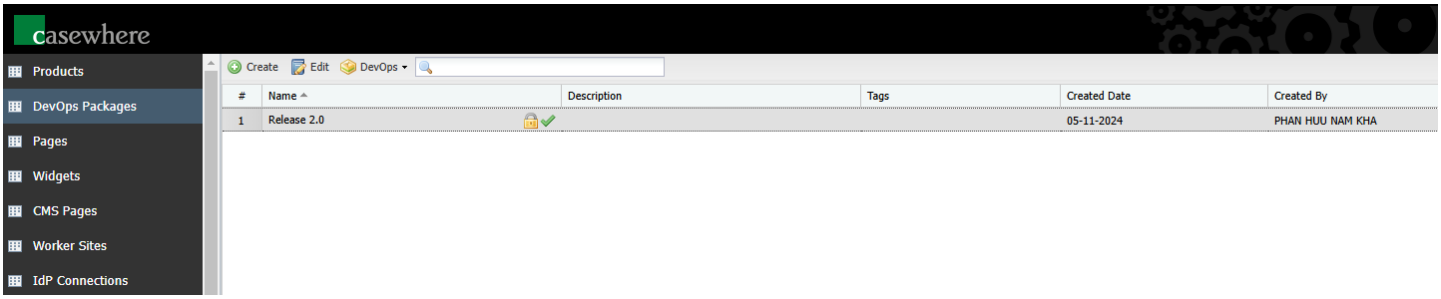
Please note that only active DevOps packages are available for selection. After you click **Save**, all DevOps packages associated with the resource item will appear in the **DevOps** column on the page.

Lock a DevOps package

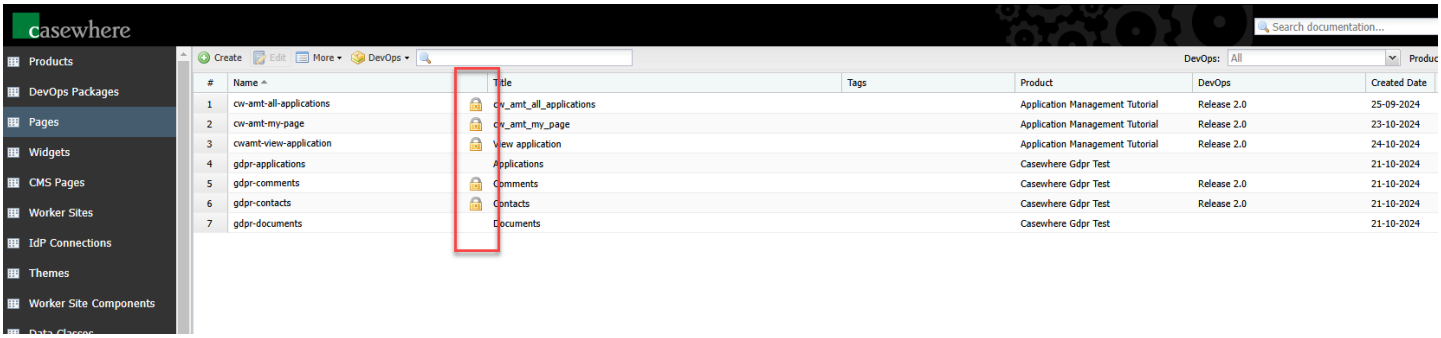
Once you have completed a release and are preparing for production deployment, it is a good practice to freeze the package while it is under user acceptance testing to prevent unwanted modifications. To freeze a package, navigate to "DevOps Packages" to edit it. In the editor dialog, check **Is Locked**, and finally click **Save**.



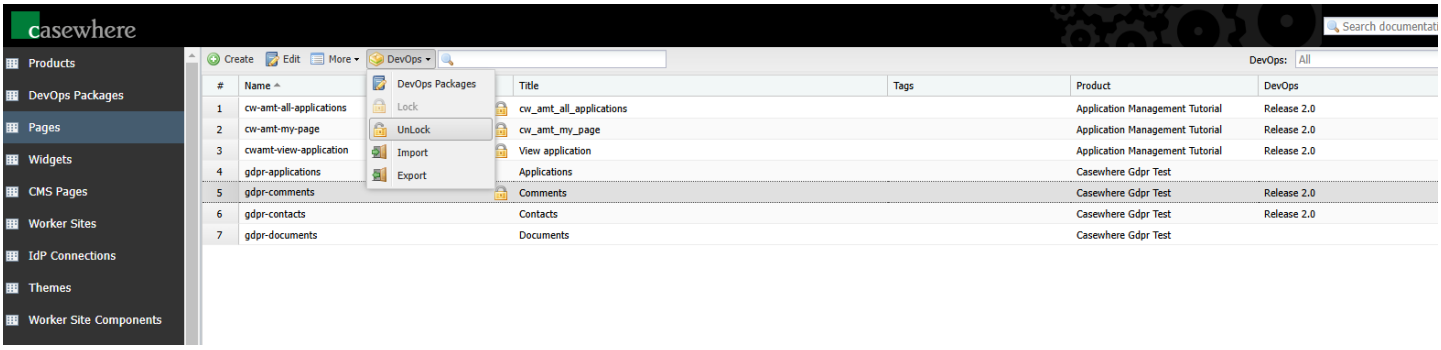
You will notice a "lock" icon appearing for locked packages.



Once a package is locked, all resource items associated with it will also be locked from modification.

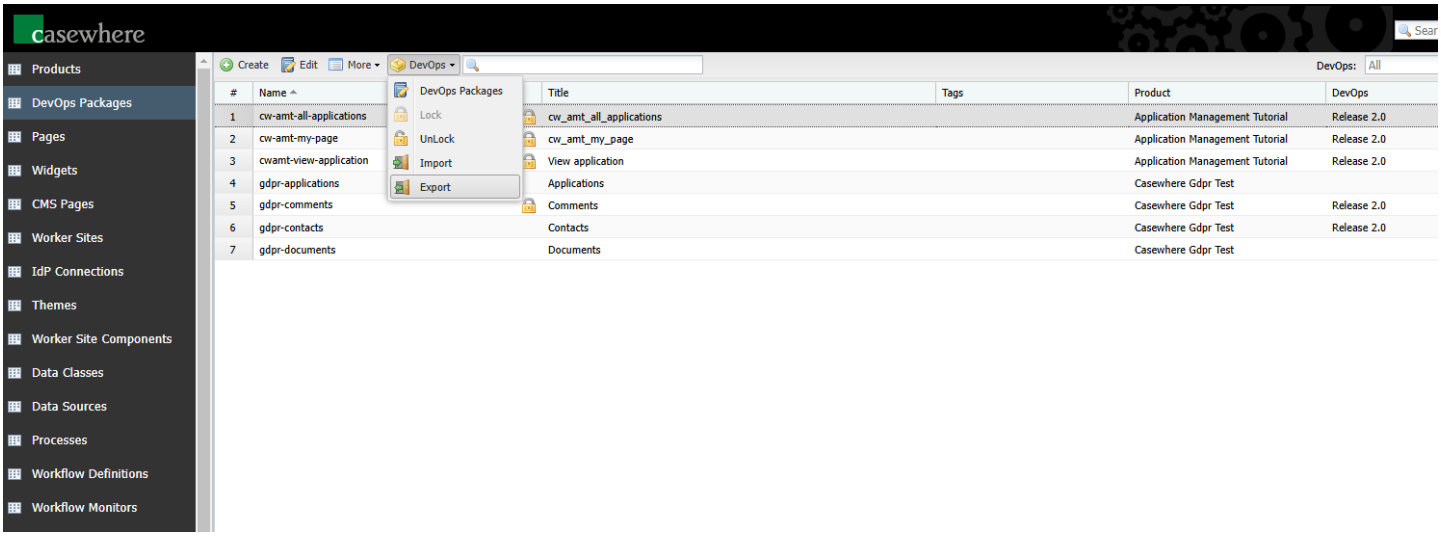


Even when the package is locked, you can still **Lock** or **Unlock** individual resource items. This can be useful when managing multiple releases simultaneously.

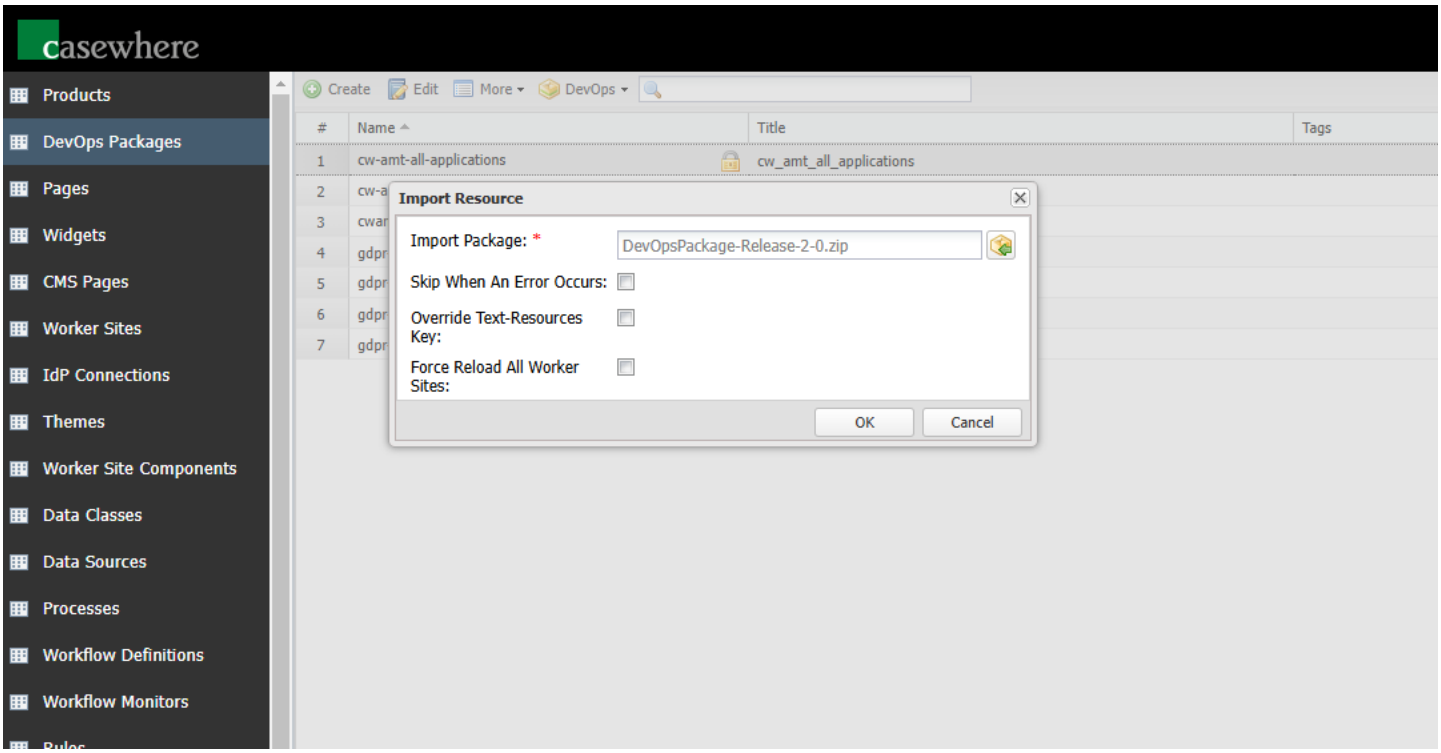


Import/export DevOps packages

In Casewhere Admin, import and export functionalities can be accessed from the **DevOps Packages** page. To export a DevOps package, select a package and click **DevOps / Export**.



To import a package, use **DevOps / Import**. The import options are similar to those for importing products.



Overview

These advanced articles will give you a solid foundation on the Casewhere platform by providing an in-depth guide tour of all the features available in the Casewhere Platform and common practices.

Deployment

These advanced deployment articles will guide you to optimize Casewhere for different scenarios.

Operation and troubleshooting

These articles provide recommendations to help you avoid issues in your production Casewhere deployment. Here we will also explain how to determine, diagnose, and troubleshoot operational problems that you might encounter when working with Casewhere.

Security

These articles introduce how Casewhere targets different aspects of security and provides guidelines and best practices to protect your applications.

Use cases

You can learn how others are using Casewhere and how to apply their experiences to solve your problems.

Best practices

Explore a set of best practices for working with Casewhere. Learn and apply these practices to your specific use cases when they are appropriate.

Hot and Cold: transferring events

When your Event database grows to hundreds of millions of events, there could be some potential performance hits when Casewhere writes data. To optimize performance for those scenarios, Casewhere supports a feature called "**Hot and cold**". When the feature is enabled, Casewhere will maintain two collections for storing events:

- **Hot** collection: Casewhere writes directly to this collection. The performance is significantly improved when Casewhere only has to write to a smaller data collection.
- **Cold** collection: All events will eventually be stored in this collection. There will be a background job running at a scheduled time to periodically transfer events from **Hot** to **Cold**.

Enable Hot and Cold

Configure Casewhere deployments

In the Casewhere Admin (project `Casewhere.Configuration.Web`), update the bellow setting in `web.config`:

```
<add key="events:EnableHotAndCold" value="true" />
```

Similarly, enable the same setting in the `web.config` of the Worker API (project `Casewhere.Worker.Api`).

If your deployment is using Azure App Service, just add the key `events:EnableHotAndCold` in the **Application Settings**.

Set up the background job for transferring events

Step 1. Download and install

Download the tool here (login is required). After that, copy the tool to the server where you want it to run. The tool is lightweight, so to save cost, you can just run it on the same server with the Web Server or the Database Server.

Install the following dependencies on the server:

1. https://download.visualstudio.microsoft.com/download/pr/69a4b292-6680-46b6-a067-eaf5b4a7ed1e/a14fa32031441812288c113bee953c8d/aspnetcore-runtime-5.0.10-win-x64.exe
2. https://download.visualstudio.microsoft.com/download/pr/9f93e474-1bad-4c66-8445-57a2ef6983d6/b49e4837fc4851a0c85da41d70495c13/dotnet-runtime-5.0.10-win-x64.exe
3. https://download.visualstudio.microsoft.com/download/pr/a0da9621-68f0-439a-b617-4697ee0669e3/38eb4aa6e879b9f06b73599ea2e1535f/dotnet-hosting-5.0.10-win.exe

Step 2. Configure the `appsettings.json`

1. Update the `MongoDBSettings`
    1. `ConnectionString`: The connection string of the **Casewhere Events **database
    2. `DatbaseName`: The name of **Casewhere Events** database
    3. `SourceEventCollectionName`: The **Hot** collection. The value must be `HotEvents`
    4. `TargetEventCollectionName`: The **Cold** collection. The value must be `Events`
2. Configure logging at `Serilog\WriteTo\Args`
    1. Update `databaseUrl` using the same connection string of the **Casewhere Log** database used in your deployment.
3. Configure the transferring parameters
    1. `BatchSize`: The number of events transferred per batch. Default is `1000`
    2. `MaxDuration`: The tool will stop after a configurable period of time. Default is `01:00:00`
    3. `DelayDurationWhenNoEvents`: The tool will keep listen for new events to transfer until it stops. The default delay is `00:00:05`

Step 3. Schedule the task

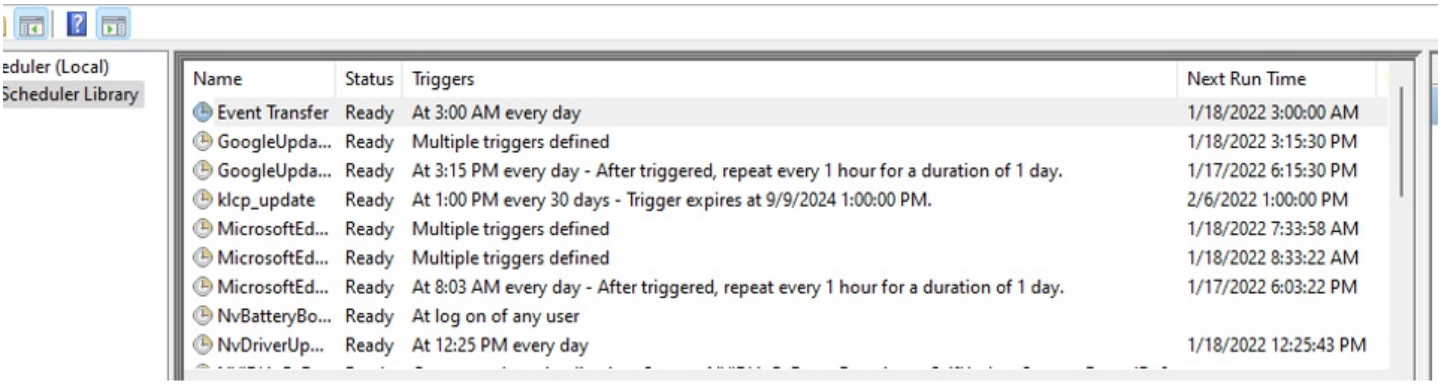
Edit the **CreateEventTransferScheduler.ps1** script to:

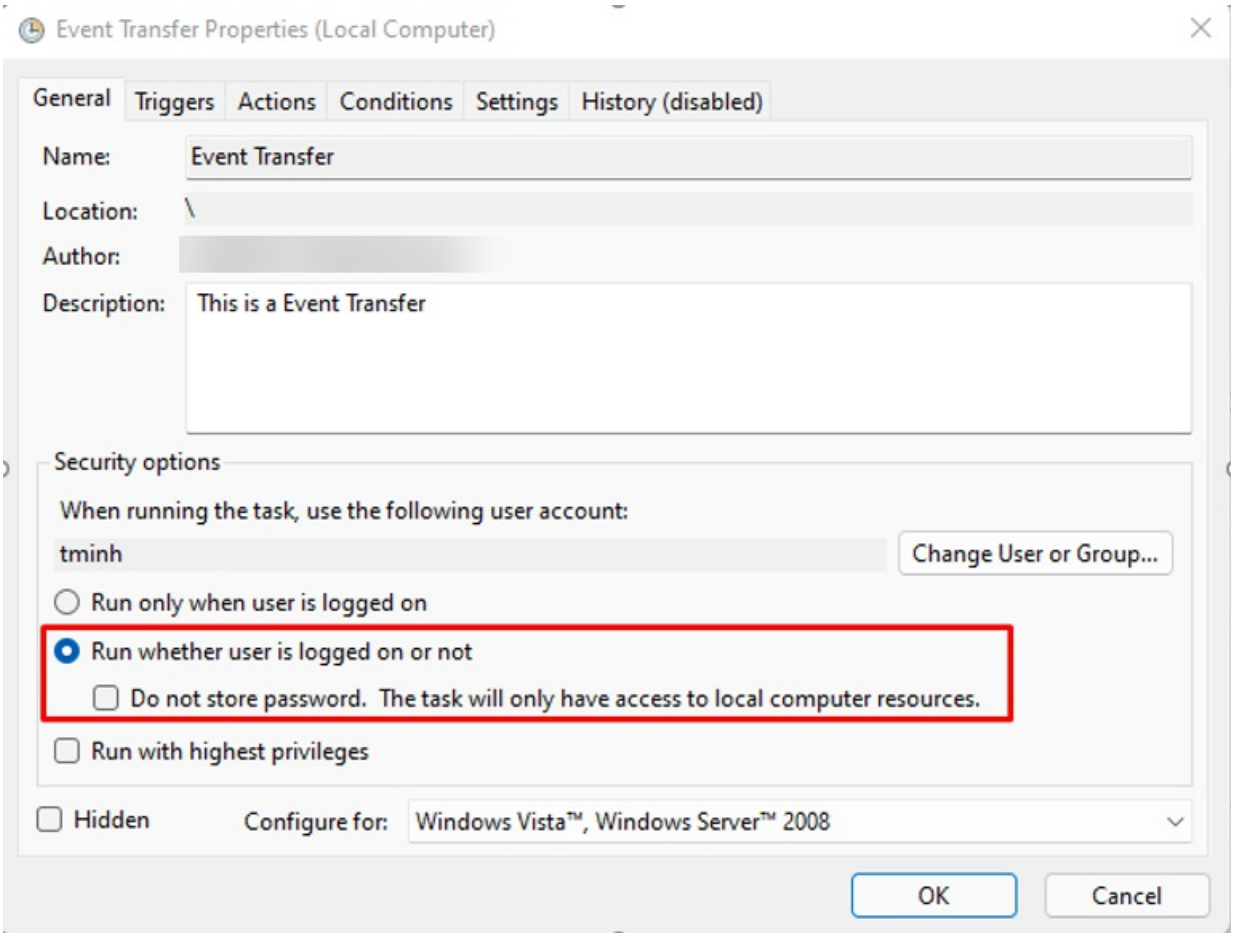
- Set the `$path` parameter to the location of the tool
- Provide the `$password` of the current user, i.e., the user account you use to run this script
- (Optional) Change the `$runTime` parameter. Default is `3AM`

CreateEventTransferScheduler.ps1

```
$path = "Enter the path to the Casewhere.EventTransfer.App.exe tool"
$runTime = "3AM"
$password = "Enter your password"
$action = New-ScheduledTaskAction -Execute $path
$trigger = New-ScheduledTaskTrigger -At $runTime -Daily
Register-ScheduledTask -TaskName "Event Transfer" `
                       -Description "This is a Event Transfer" `
                       -Action $action `
                       -Trigger $trigger `
                       -User "$env:USERDOMAIN\$env:USERNAME" `
                       -Password $password
```

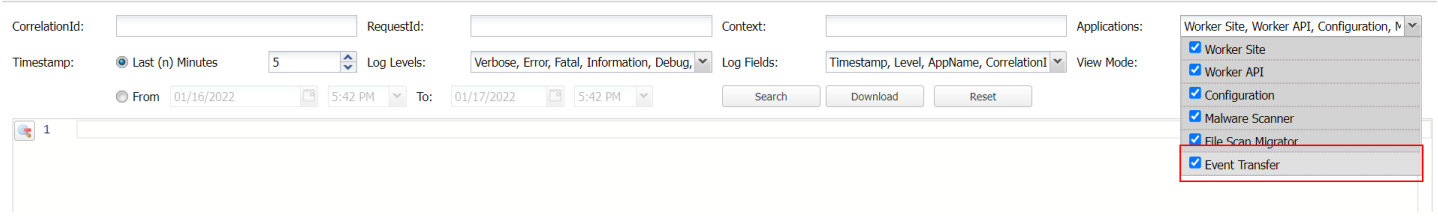
Verify the result in **Task Scheduler**. Make sure the task will run regardless of whether the user logins or not.





Step 4. Monitor the tool

When the tool runs, it will write log to Casewhere database so you can monitor it using Log Viewer.



Enable Azure Application Insight codeless integration

To enable the Azure Application Insights codeless integration:

- Please remove **System.Diagnostics.DiagnosticSource.dll** and **System.Diagnostics.DiagnosticSource.xml** out of **bin** folder inside `Casewhere.Worker.Api` and `Casewhere.Configuration.Web` before you deploy to Azure App Service.
- In Azure Portal, open the **Worker Api App Service** and **Configuration App Service**, navigate to **Configuration / Application Settings** and make sure the following key:

  `InstrumentationEngine_EXTENSION_VERSION : disabled`
- Stop and start these Azure App Services and verify the Azure Application Insight status following the link:

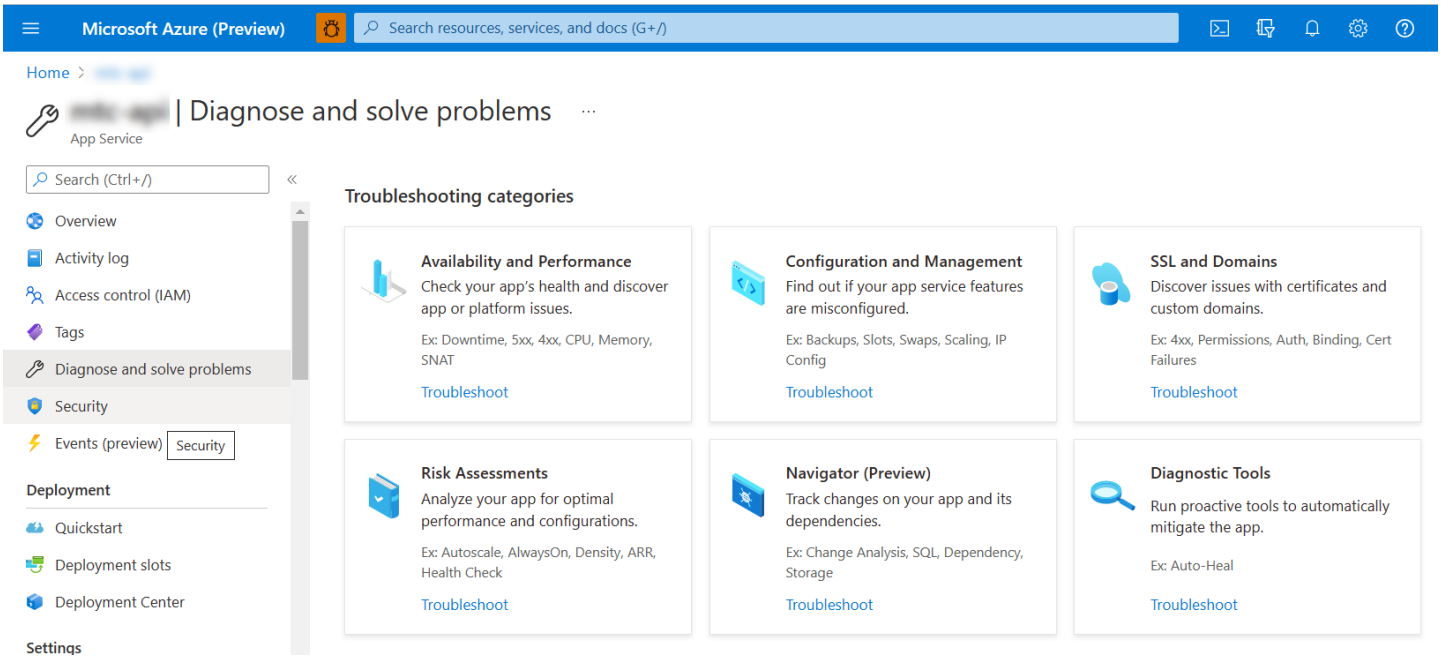
  `https://{yourwebappname}.scm.azurewebsites.net/ApplicationInsights`

Troubleshoot Casewhere on Azure App Service

Azure App Service enables you to host Casewhere without managing infrastructure. It offers auto-scaling, high availability, and specifically, provides built-in diagnostics tools to assist troubleshooting. In this article, you will learn to use these tools to determine, diagnose and debug the problems you might encounter when working with Casewhere on Azure.

App Service diagnostics

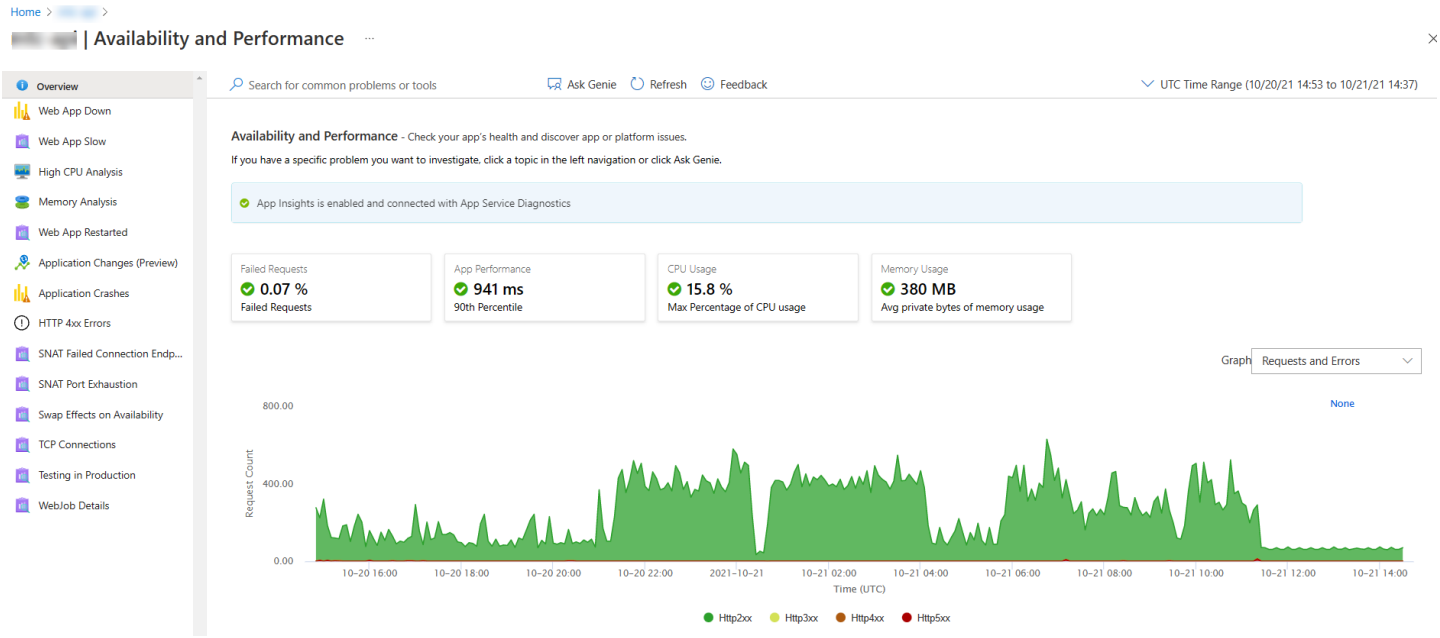
To access App Service diagnostics, navigate to your App Service web app or App Service Environment in the Azure portal. In the left navigation, click on **Diagnose and solve problems**.



Availability and Performance tools

If your application is performing unstable, this is where you should start. Below are the analysis tools for diagnosing different problems:

- **Web App Down**: Troubleshoot app performance and availability
- **Web App Slow**: Check app latency and analyze common patterns that cause your app to respond slowly
- **High CPU Analysis**: Analyze CPU usage of your app on all instances and CPU usage of all apps in your app service plan
- **Memory Analysis**: Analyze high memory usage for your app and drill down further to identify app or process consuming high memory
- **Web App Restarted**: See the timeline of restarts as well as the cause of the restart
- **Application Crashes**: Detects application crashes and related events for your application
- *... and many more*

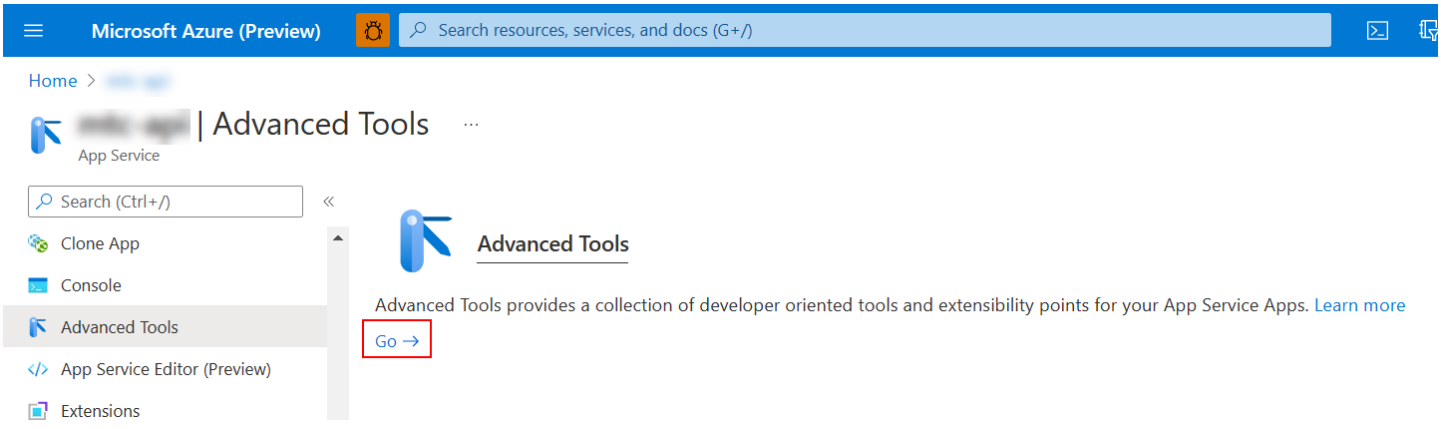


Diagnostic tools

- **Collect a Profiler Trace**: If your app is down or performing slow, you can collect a profiling trace to identify the root cause of the issue. Profiling is lightweight and is designed for production scenarios.
- **Collect a Memory Dump**: If your app is slow or not responding at all, you can collect a memory dump to identify the root cause of the issue.
- **Application Event Logs**: View Event Logs containing exceptions and errors generated by your app
- **Failed Request Tracing Logs**: Drill down to view Failed Request tracing logs for your app
- **Metrics**: A wide range of metrics for monitoring your app

Kudu

Kudu is very useful when troubleshooting Azure App Service. Navigation to **Advanced Tools**, click **Go** to open the Kudu site.



From here, you can:

- Navigate to **Tools/Diagnostic dump** to download the diagnostic dump packaged in a zip file.
- Navigate to **Process explorer** to review the memory status and run profiling if necessary.

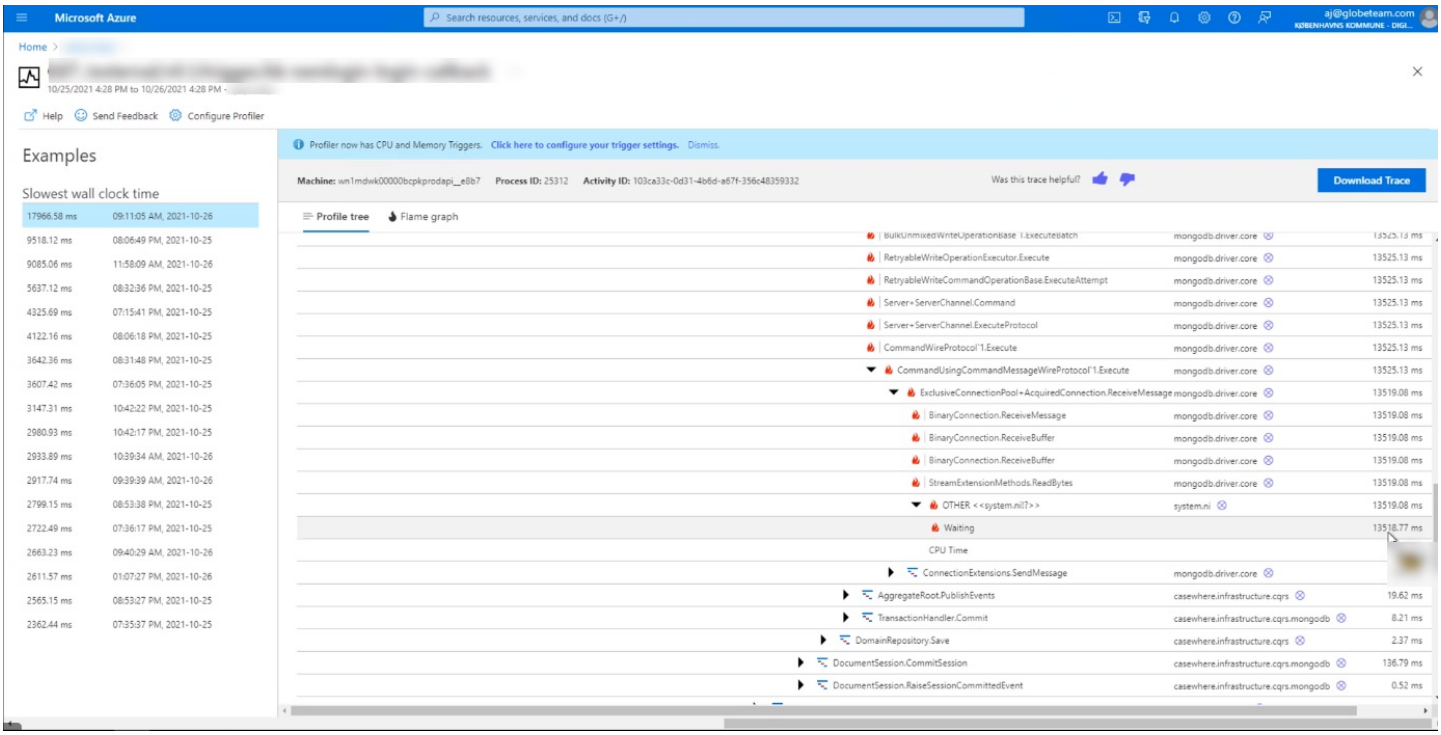
Application Insights

In Application Insights, you can see the following data:

- **Availability**: To see if your SLA is being met
- **Failures**: Failed requests, dependency failures, exceptions, and server requests
- **Performance**: To see how fast or slow your pages or APIs are

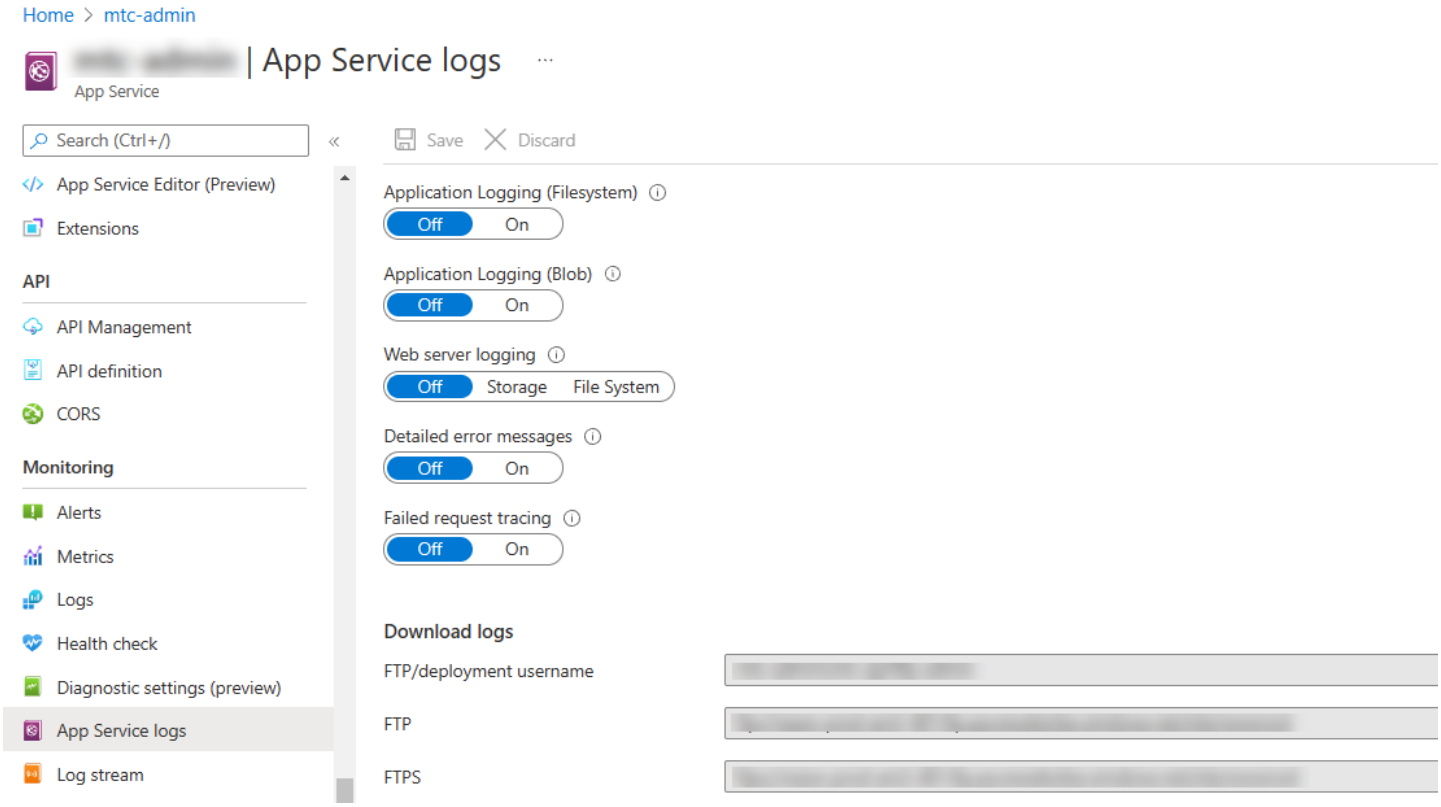
- **Server metrics**: To see the server metrics of all the servers on which your app runs, like CPU and RAM usage
- **Custom events**: Define your own events by calling Application Insights from your app and writing a custom message

The profiler in Application Insights is very useful when detecting bottlenecks. Application Insights generate traces for slow requests, which you can download or view directly on Azure Portal.



App Service logs

If you are enabling diagnostics logging for your app, it's always a good idea to download and examine these data.



We recommend using **Blob** storage for storing these log data. Click here to learn how to enable the **App Service logs**.

Casewhere logs

Casewhere logs are one of the most crucial tools you need to get familiar with to troubleshoot problems on production. It's specifically handy for debugging Casewhere applications.

Casewhere currently provides three types of logs:

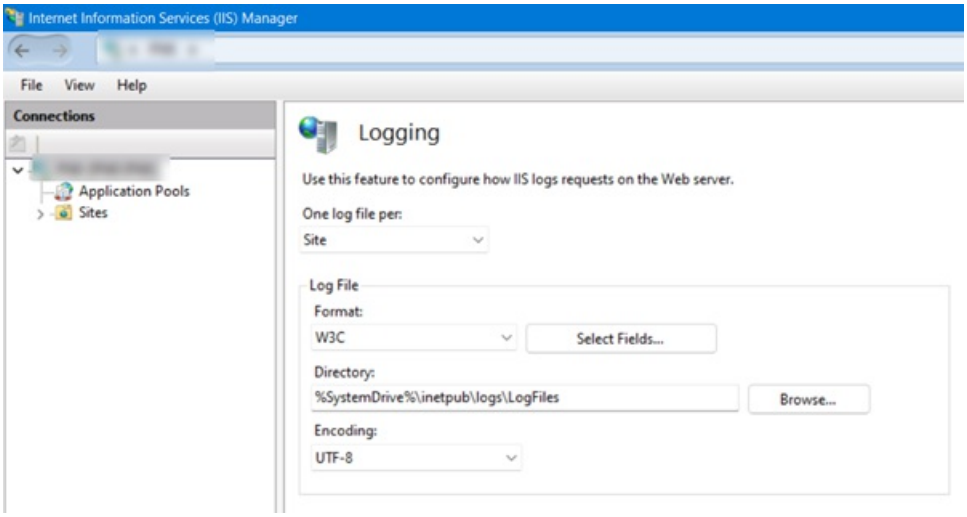
- System log: Platform logs and application logs
- Performance log: The response time of all HTTP requests
- Audit log: Data changes, who, what, and when

You can learn more about Casewhere logs here.

Troubleshoot Casewhere on-premise

Standard IIS logs

Standard IIS logs will include all web requests that flows through Casewhere sites.

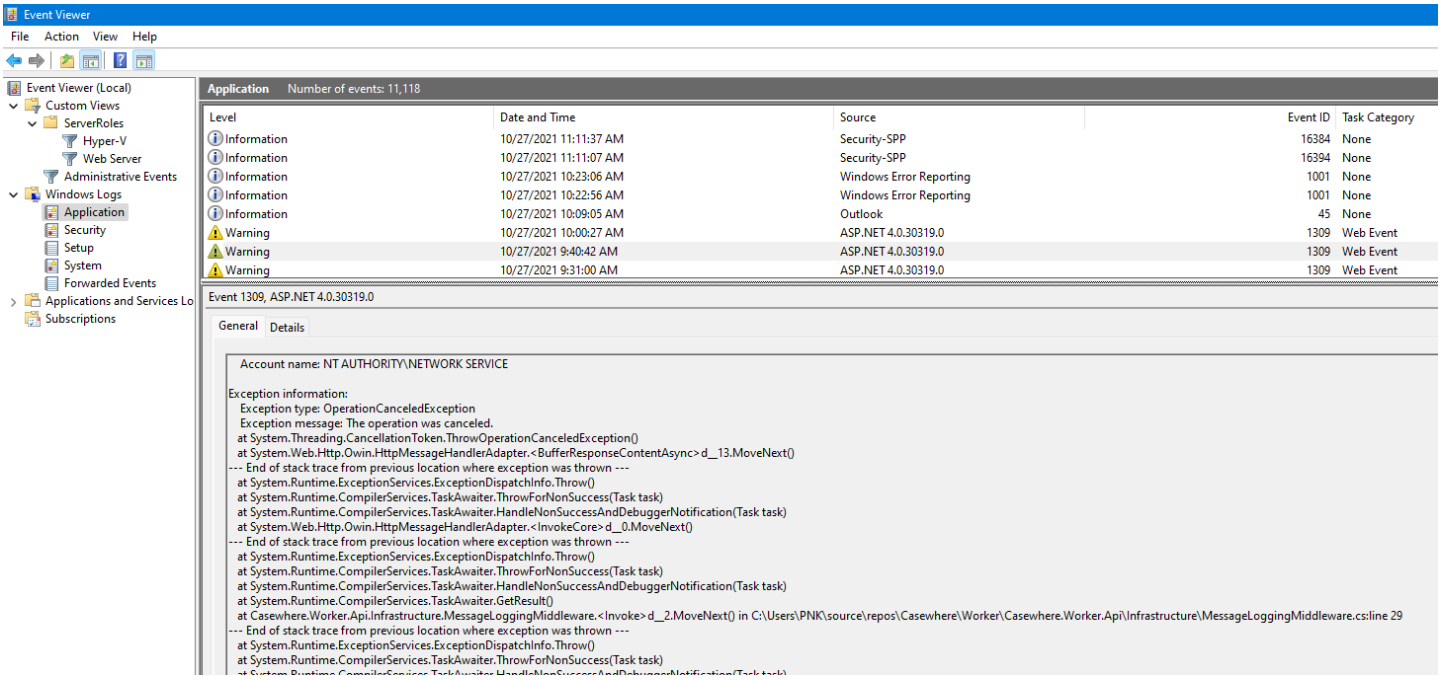


HTTPERR logs

Incoming requests to your server first route through HTTP.SYS before being handed to IIS. If single web request doesn't show in your IIS log, it is possible that the request never made it to IIS, or IIS wasn't running. The HTTPERR logs can be found at `C:\Windows\System32\LogFiles\HTTPERR`.

ASP.NET exceptions in Windows Event Viewer

By default, ASP.NET will log unhandled 500 level exceptions to the Windows Application EventLog.



Casewhere logs

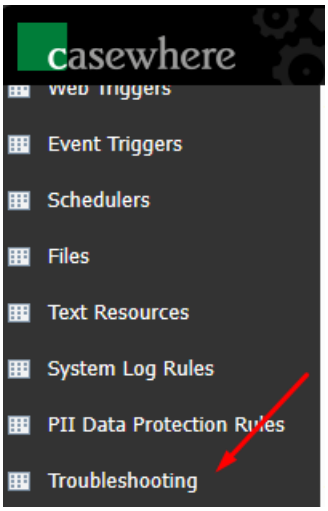
Casewhere logs are one of the most crucial tools you need to get familiar with to troubleshoot problems in production. Casewhere currently provides the following types of logs:

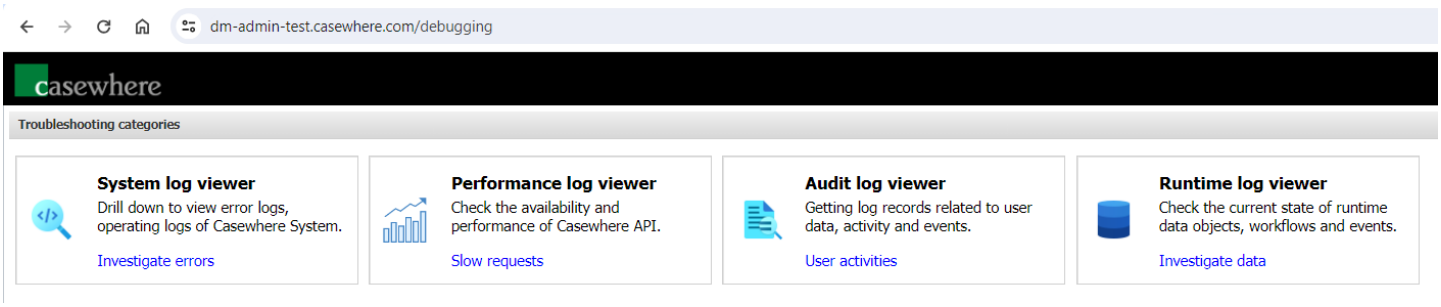
- System log: Platform logs and application logs
- Performance log: The response time of all HTTP requests
- Audit log: Data changes, who, what, and when

You can learn more about Casewhere logs here.

Troubleshoot with Casewhere logs

Casewhere logs are one of the most crucial tools you need to get familiar with to troubleshoot problems in production. The Casewhere Troubleshoot Tools can be found at `/debugging` or via the **Troubleshooting** menu in Casewhere Admin. The menu is only available for users with the **Administrator** role.

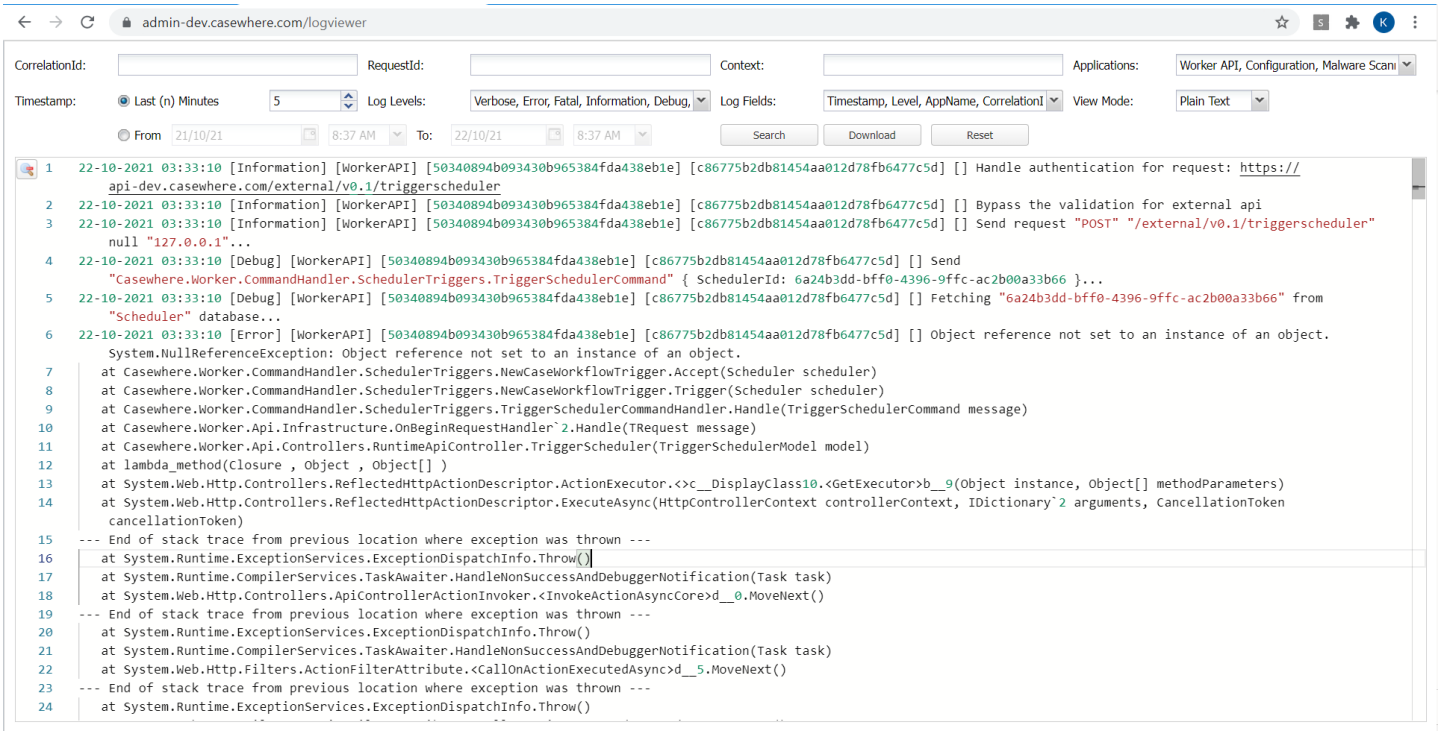




System log viewer

By default, Casewhere stores logs in MongoDB. From Casewhere Admin, you can access the log data using Log Viewer at the path "/logviewer". For example, https://admin-dev.casewhere.com/logviewer.

In this page, you can drill down to view error logs, operating logs of Casewhere System.



It's possible to filter log data using the below fields:

FIELD NAME	DESCRIPTION
`CorrelationId`	All HTTP requests in the same context will share a **CorrelationId**. For example, workflow A creates data object B, that triggers a workflow C through a data trigger. Workflow A and workflow C will share the same **CorrelationId**. Casewhere automatically set the **CorrelationId**, which can be found in the HTTP response header.
`RequestId`	Casewhere will assign each HTTP request a unique ID, which can be found in the HTTP response header. From v2.7.0, you can query log data by **RequestId**.
`Context`	The logging context in which Casewhere writes log. One typical use case for the **Context** is to view log for a specific workflow. You can use this syntax `Workflow:<workflow_guid_id>` to filter log data. For example, `Workflow:6a24b3dd-bff0-4396-9ffc-ac2b00a33b66`.
`Applications`	The sources generate log data.
`Timestamp`	The time log data is created.
`Log Levels`	Casewhere write logs at different levels and each level has its own purposes. You can read more here.
`Log Fields`	You can use this setting to choose which fields you want to see in the **Log Viewer**. This setting is available from v2.7.0 and it's applicable for the **Plain Text** view mode only.
`View Mode`	**Plain Text** or **Json**.

Logging levels

Below are the logging levels Casewhere is writing logs with. It's possible to configure the minimum level specifically for a Casewhere deployment.

LEVEL	USAGE
`Verbose`	Verbose is the noisiest level, rarely (if ever) enabled for a production app.

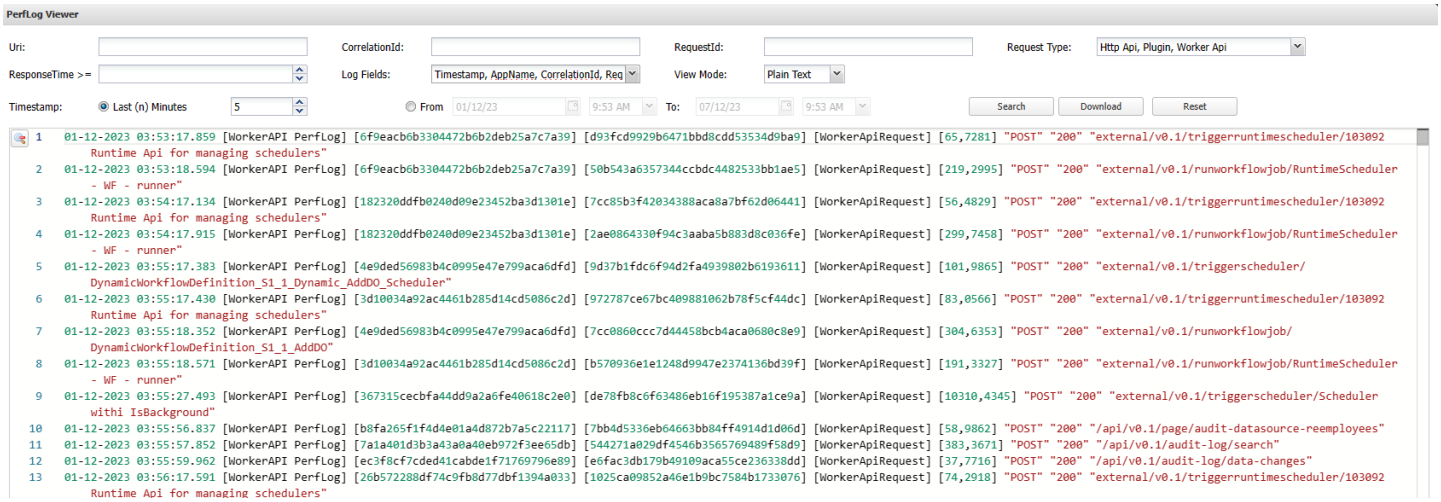
LEVEL	USAGE
`Debug`	Debug is used for internal system events that are not necessarily observable from the outside, but useful when determining how something happened.
`Information`	Information events describe things happening in the system that correspond to its responsibilities and functions. Generally these are the observable actions the system can perform.
`Warning`	When service is degraded, endangered, or may be behaving outside of its expected parameters, Warning level events are used.
`Error`	When functionality is unavailable or expectations broken, an Error event is used.
`Fatal`	The most critical level, Fatal events demand immediate attention. This level is available from v2.7.0.

Tips

- Where to find the CorrelationId and RequestId?

Performance log viewer

Casewhere tracks the response time of all HTTP requests and stores the data in MongoDB by default. If you can access the log database, you will find the data in the `PerfLogs` collection.



In this page, you can check the availability and performance of Casewhere API. The filtering of performance logs same as System log viewer.

Below are the properties of a performance-log entry:

NAME	DESCRIPTION
`Timestamp`	The time log data is created. Use this field if you want to query data by time.
`Properties.RequestUri`	The full URI of the HTTP request.
`Properties.Uri`	The URI of the HTTP request without the query string.
`Properties.ResponseTime`	Response time in milliseconds.
`Properties.HttpMethod`	The HTTP method. For example, POST, GET, PUT, DELETE.

NAME	DESCRIPTION
`Properties.StatusCode`	The HTTP response status code. For example, 200 means success.
`Properties.CorrelationId`	The unique Id you can use to trace data in the system log. All related HTTP requests will share the same **CorrelationId**.
`Properties.RequestId`	The unique Id you can use to trace data in the system log. Each HTTP request has an Id assigned by Casewhere.
`Properties.Type`	Type of the request. Possible values are: WorkerApiRequest, PluginRequest, HttpApiRequest. The last two represent external calls.
`Properties.AppName`	The sources generate log data.

Here is an example of performance-log data:


```
{
    "_id" : ObjectId("615a7aaef9d398600cc8bd79"),
    "Timestamp" : ISODate("2021-10-04T03:53:17.597Z"),
    "Level" : "Information",
    "MessageTemplate" : "api/v0.1/worker-site/worker-site-components/worker-local.casewhere.com",
    "RenderedMessage" : "api/v0.1/worker-site/worker-site-components/worker-local.casewhere.com",
    "Properties" : {
        "RequestUri" : "/api/v0.1/worker-site/worker-site-components?workerSiteId=989d20c0-c0ff-4a4b-ad8f-
ad81010c4292",
        "ResponseTime" : 7.3167,
        "HttpMethod" : "GET",
        "StatusCode" : "200",
        "Uri" : "api/v0.1/worker-site/worker-site-components/worker-local.casewhere.com",
        "Type" : "WorkerApiRequest",
        "CorrelationId" : "58be16a31f0347e5936d6054e83f4502",
        "RequestId" : "f82caaaea4114c9eb4e5ad9a58f0bcb7",
        "RequestTimestamp" : "2021-10-04T03:53:17.5908466Z",
        "AppName" : "WorkerAPI PerfLog"
    },
    "UtcTimestamp" : "2021-10-04 03:53:17Z"
}
```


Tips

- How to query the top slowest endpoints in Casewhere?

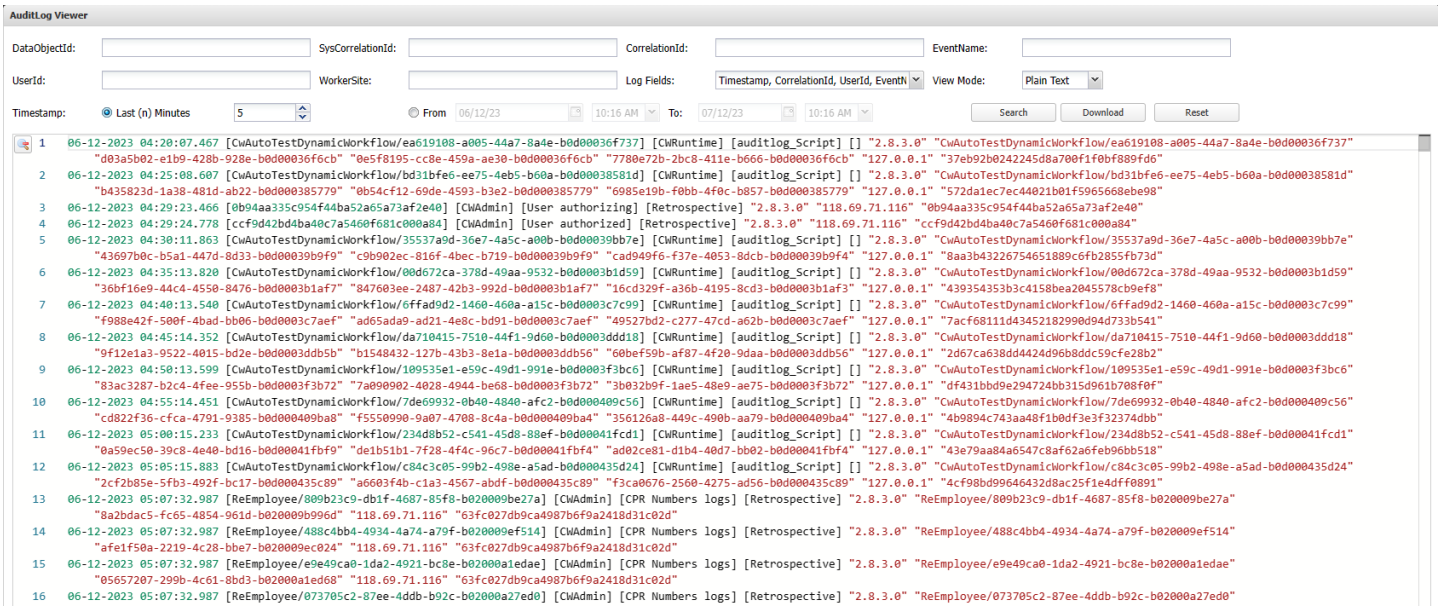
Audit log viewer

Audit logs are a valuable resource for admins and auditors who want to examine suspicious activity or diagnose and troubleshoot issues. It's possible to configure Casewhere to track data changes and important events that happened in the system and store them in a more consistent, readable format.

Currently, Casewhere audits the following events:

- Data change events: All changes are logged. It's possible to specify what to log and what to ignore at the field level
- Read events: It’s possible to enable logging for read access to critical data sources
- Authentication: All events related to authentication are logged by default
- Custom logging: It’s possible to hook into any workflow/process for custom logging

Using the tool, you can get log records related to user data, activity, and events



Below are the properties of an audit-log entry:

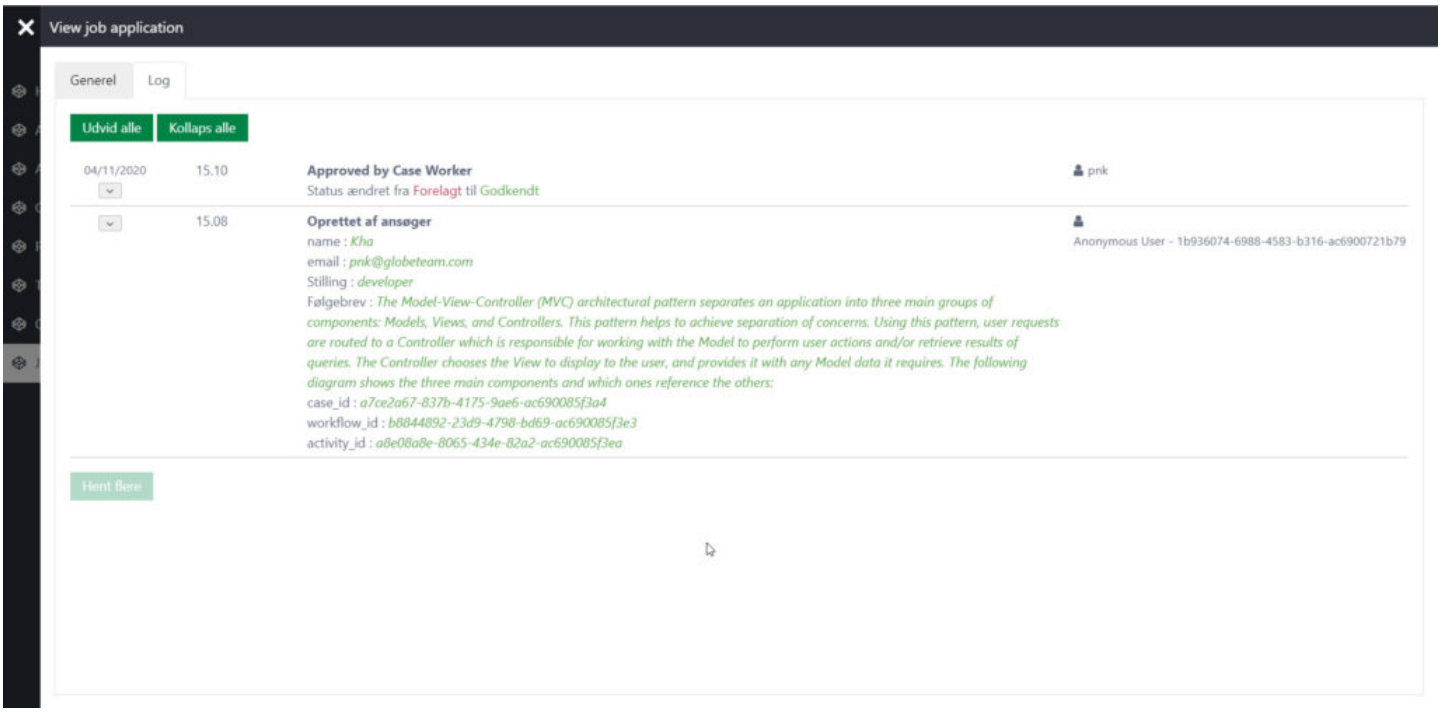
FIELD NAME	DESCRIPTION
`EventName`	The event being logged. For example, `auditlog_Send`, `User authorized`
`UserId`	Id of the user
`UserClaims`	Username, roles, and other claims. It's possible to select which claims to log
`WorkerSite`	The site in which the event happened
`CorrelationId`	It's possible to use a business value, e.g., `CustomerID`, to track all related events
`SysCorrelationId`	All HTTP requests in the same business context will share a **SysCorrelationId** generated by the system
`Timestamp`	The time the event happened
`IsCustomLog`	The event is built-in or custom
`PlatformVersion`	The version of the Casewhere platform
`IpAddress`	The client's IP address
`BrowserAgent`	The browser sending the request
`Data`	For data change events, the system will track changed information in the following format `[{AttributeName: "Status", FromValue: "Submitted", ToValue: "Approved"}]`
`DataObjectId`	The data object being changed, which emits the event
`DataSourceId`	The data source being read, which emits the query event
`ActivityId`	The activity that emits the event

FIELD NAME	DESCRIPTION
`WorkflowId`	The workflow that emits the event
`CaseId`	The case that emits the event

Below is an example of an audit log record:

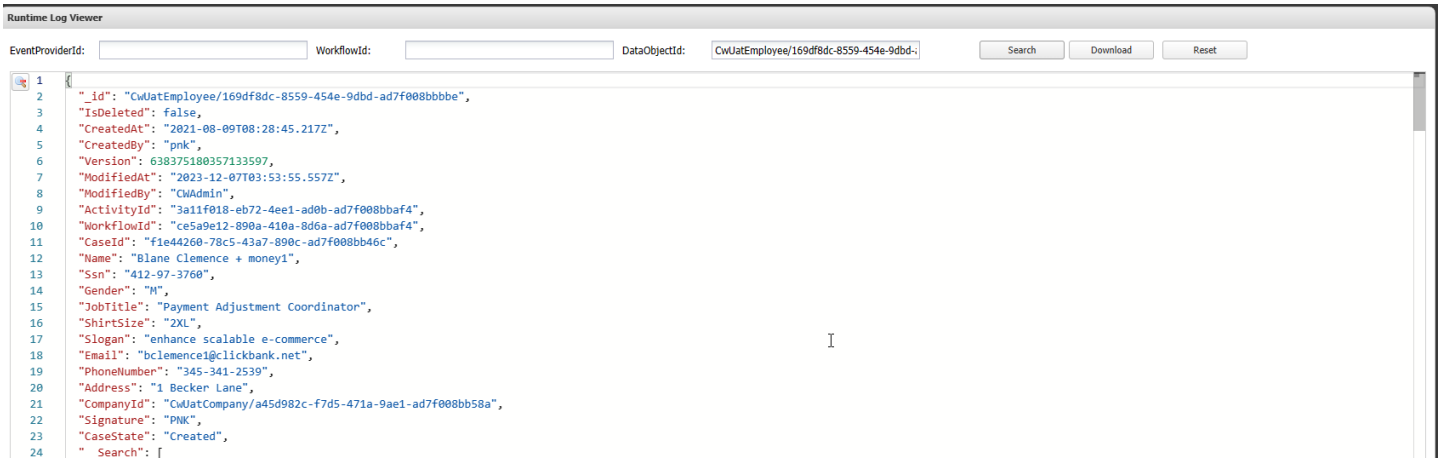

```
{
    "_id" : NUUID("f0458910-ec3b-4e68-a271-d348668aad6a"),
    "_t" : [
        "AuditLog",
        "DataObjectAuditLog"
    ],
    "EventName" : "auditlog_Auto approve",
    "UserId" : "pnk",
    "WorkerSite" : "Caseworker Site",
    "UserClaims" : [
        {
            "Key" : "Username",
            "Values" : [
                "pnk",
                "GLOBETEAM\\pnk"
            ]
        },
        {
            "Key" : "Roles",
            "Values" : [
                "Employee",
                "Administrator"
            ]
        }
    ],
    "CorrelationId" : null,
    "SysCorrelationId" : "cc645a2cba0f45eeb8df32a6a98085f1",
    "Timestamp" : ISODate("2022-08-25T07:41:38.249Z"),
    "IsCustomLog" : false,
    "PlatformVersion" : "2.8.0.0",
    "DataObjectId" : "CwUatTx2Application/ab77d7ee-6e33-4984-879b-aefc007ecada",
    "ActivityId" : NUUID("2e97482e-82f8-401c-bd62-aefc007ecadc"),
    "WorkflowId" : NUUID("d43210ff-71d6-4101-9292-aefc007ecadb"),
    "CaseId" : NUUID("c6ab7fd3-8adc-4f42-8539-ae91006484ad"),
    "Data" : [
        {
            "AttributeName" : "Status",
            "FromValue" : "Submitted",
            "ToValue" : "Approved"
        }
    ],
    "IpAddress" : "127.0.0.1",
    "BrowserAgent" : "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/104.0.0.0 Safari/537.36"
}
```


Casewhere also provides an audit-log viewer that developers can embed in worker sites.



Runtime log viewer

You can use this tool to examine the state of a workflow, a data object, or Casewhere events. This is useful in many cases



- **EventProviderId**: The data object ID.
- **WorkflowId**: The workflow ID.
- **DataObjectId**: The data object ID.

Note: Sensitive information will be masked following the rules defined in the **System log rules**.

Maintaining Casewhere solutions - Daily operations

Introduction

This document serves as a comprehensive resource tailored for IT administrators, system operators, and any personnel involved in the daily maintenance to ensure the seamless operation and optimal performance of Casewhere solutions.

Documentation and communication

Personnel

Creating a personnel list for daily operations is essential for maintaining transparency and assigning responsibilities effectively. Below is a template for a personnel list that you can customize according to your organization's structure and needs.

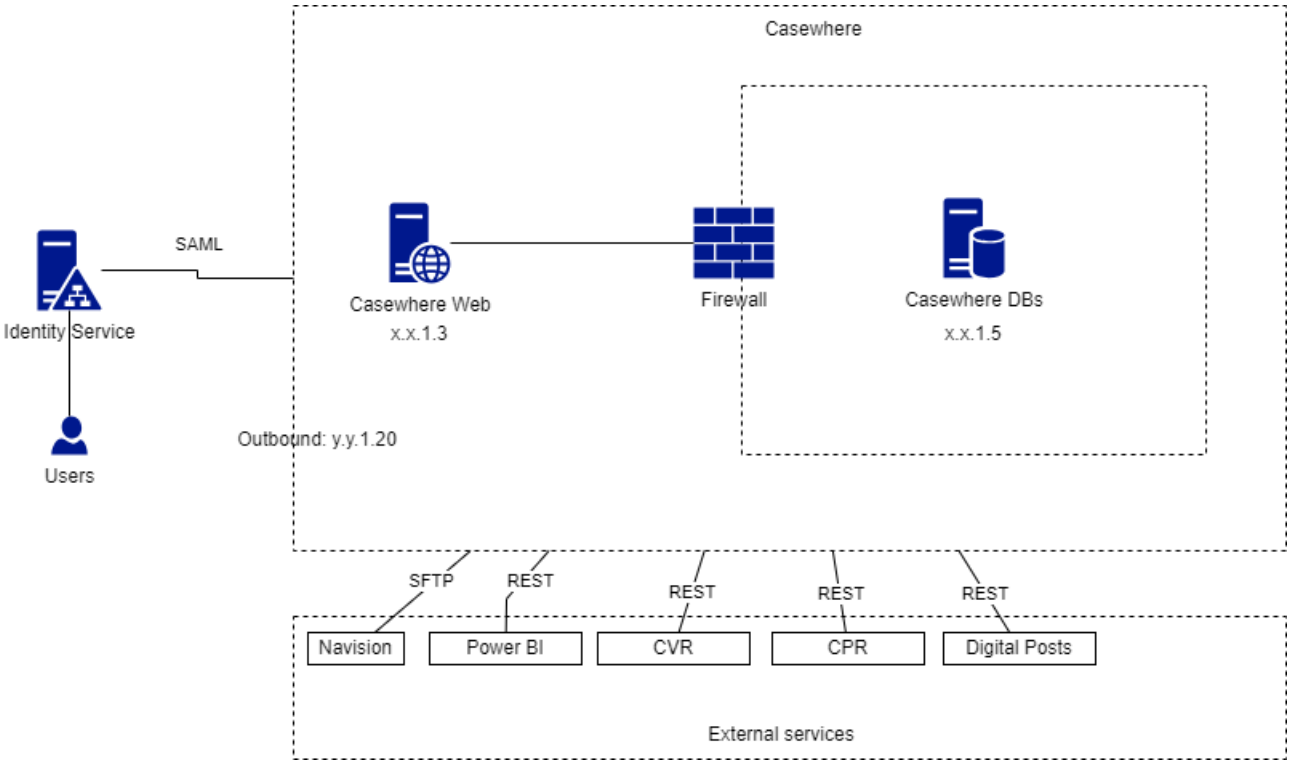
NAME	ROLES	RESPONSIBILITIES	CONTACT INFORMATION
Jone Doe	System administrators	Be responsible for system health monitoring, server performance, and database connectivity	jone-doe@example.com
Jane Doe	Casewhere specialists	Provide IT supports, addresses technical issues related to the Casewhere platform	jane-doe@example.com

Solution components

It's recommended to document all system components and their relations. This will aid in debugging and troubleshooting.

Solution diagram

Diagrams are always the best tools for describing solution architecture. A typical Casewhere solution can be illustrated as follows:



Component list

There is still a need to list all system components with detailed profiles, configurations, etc. For example:

NAME	TYPE	DESCRIPTION
Casewhere Web	VM	Cores: 8. RAM: 16GB. SSD 256GB. IP: x.x.1.3. Outbound IP: y.y.1.20
Casewhere DB	VM	Cores: 8. RAM: 32GB. SSD 512GB. IP: x.x.1.5
Identity Service	VM	Safewhere Identify
CVR	Datafordeler	Restful API
CPR	Service platform	Restful API
Digital Posts	Service platform	Restful API

Others

Other things, such as domains and SSLs, must be documented. It's recommended to create reminders for SSL renewal.

Solution monitoring

Solution monitoring should be automated as much as possible. Daily reviews of notifications and alerts are essential to ensure seamless availability and performance. Detailed analysis can be conducted on suspicious or potentially high-risk threats.

Monitoring availability

Casewhere provides a health check endpoint at `{worker_api}/api/v0.1/ping`, which verifies the availability of Casewhere, including its web applications and associated databases. If your solution depends on external services, it is recommended to implement availability tests for publicly accessible services at a minimum.

The following screenshot demonstrates the use of Azure Monitor Standard Test to set up tests for monitoring the availability of Casewhere.



For on-premise solutions, Casewhere provides an external tool that continuously performs health checks and sends alerts to designated recipients when the system becomes unavailable.



Monitoring custom metrics

Each project has unique concerns and may require defining and monitoring specific metrics to ensure the solution meets expectations.

Casewhere offers standard components for monitoring system performance and generating insightful SLA reports. These components allow you to monitor individual resources (e.g., pages, workflows) or groups of resources, define service goals, and configure alert rules. Learn more here.

The following screenshot illustrates how Casewhere can manage and monitor various types of metrics.



Monitoring system resources

It is recommended to use the built-in monitoring tools provided by hosting providers. If these options are limited, you can also consider setting up alerts using Windows Performance Counters on both the Web Server and Database Server. Some typical monitoring metrics include:

- CPU utilization. For example, every 5 minutes, check and alert if the average CPU percentage in the last 30 minutes is greater than or equal to 85%.
- Memory consumption. For example, every 5 minutes, check and alert if the average available RAM percentage in the last 30 minutes is greater than or equal to 85%.
- Disk consumption. For example, every day, check and alert if the available space of the data disk is less than 10%.
- Error rate: For instance, every 5 minutes, check and trigger an alert if the total HTTP 5xx errors in the last 30 minutes exceed 10.
- Unusual slow: For example, check every minute and trigger an alert if the average response time over the last 5 minutes exceeds 10 seconds.

Monitoring applications

It is advisable to monitor critical components within the solution, including the payment gateway, mail service, CPR service, user authentication module, among others. Casewhere provides standard components for capturing custom events generated by the solution. You can integrate a variety of logging technologies to store events, ranging from cloud services like Azure Application Insights to local tools such as Windows Event Log. It's crucial to establish monitoring objectives and set up alerts accordingly.

The screenshot below shows how Casewhere collects custom events in Application Insights:

If integration with a cloud-managed service like Application Insights isn't possible, Casewhere provides a self-monitoring component that store and manage events within Casewhere and triggers custom alerts.

System backups and recovery

Backup procedures

The backup strategy should be able to provide multiple restore points for data recovery. It’s helpful to recover from a data corruption disaster. For example:

RESOURCE	FREQUENCY	TIME	RETENTION
MongoDB VM	Daily	02:00 AM, UTC	7 days
MongoDB VM	Monthly	02:00 AM, UTC	3 months
MongoDB VM	Weekly	02:00 AM, UTC	5 weeks

Data verification

The IT personnel must regularly test if the backup is restorable to make sure there is no surprise during the disaster recovery.

Disaster recovery plan

The primary goals of the disaster recovery plan:

- To minimize interruptions to normal operations.
- To limit the extent of disruption and damage.
- To minimize the economic impact of the interruption.

- To establish alternative means of operation in advance.
- To train personnel with emergency procedures.
- To provide for smooth and rapid restoration of service

For any disaster scenario, the following elements should be addressed:

- Emergency response
  - Notify customers, responsible personnel, and concerned parties immediately
  - Consult with related parties to determine the degree of disaster and its impact on the business.
  - Constantly monitor the disaster progress and keep parties updated.
- Backup data: Even when you have regular backups, an instant backup should be taken at the time of disaster if it's still possible.
- Recovery actions
  - Follow the strategic actions designed for the happening scenario.
  - Ensure that all personnel involved know their tasks.
  - Monitor the recovery progress and keep related parties updated.
  - Send a service report to concerned parties after recovery.

Troubleshooting

When you need to conduct a thorough analysis to investigate an incident or a bug, familiarizing yourself with Casewhere logs is essential for troubleshooting problems in production. Casewhere currently provides the following types of logs:

- System log: Platform logs and application logs
- Performance log: The response time of all HTTP requests
- Audit log: Data changes, who, what, and when

You can learn more about Casewhere logs here.

GDPR - Right to be forgotten

As an application development platform for enterprises, a top priority of Casewhere is to ensure that Casewhere applications developed for or by our customers comply with GDPR. Under Article 17 of the GDPR, individuals have the right to have personal data erased. This is also known as the "right to be forgotten". This article will introduce you to how Casewhere helps developers handle this requirement by configuring anonymization rules. But first, you will need to understand how the anonymization process works in Casewhere.

Anonymization process

Data anonymization is the process of protecting private or sensitive information by removing identifiers that connect an individual to stored data. For example, you can erase Personally Identifiable Information (PII) such as names, social security numbers, and addresses through a data anonymization process that retains the data but keeps the source anonymous. The anonymization process in Casewhere will ensure that after anonymized, it's impossible to identify the person in all Casewhere databases:

- Event source
- Snapshot database
- Audit log
- System log
- Encrypted data: It's possible to protect sensitive data in Casewhere by enabling encryption on some specific fields. What's great is that the anonymization process will also anonymize these encrypted data, so even when the attacker somehow gets the encryption key of Casewhere, he still cannot identify the person.

The Casewhere anonymization process will ensure that the application can still function properly on the anonymized data. That is, the caseworker can still view and even edit the case or part of it though he can no longer identify to who the case is related.

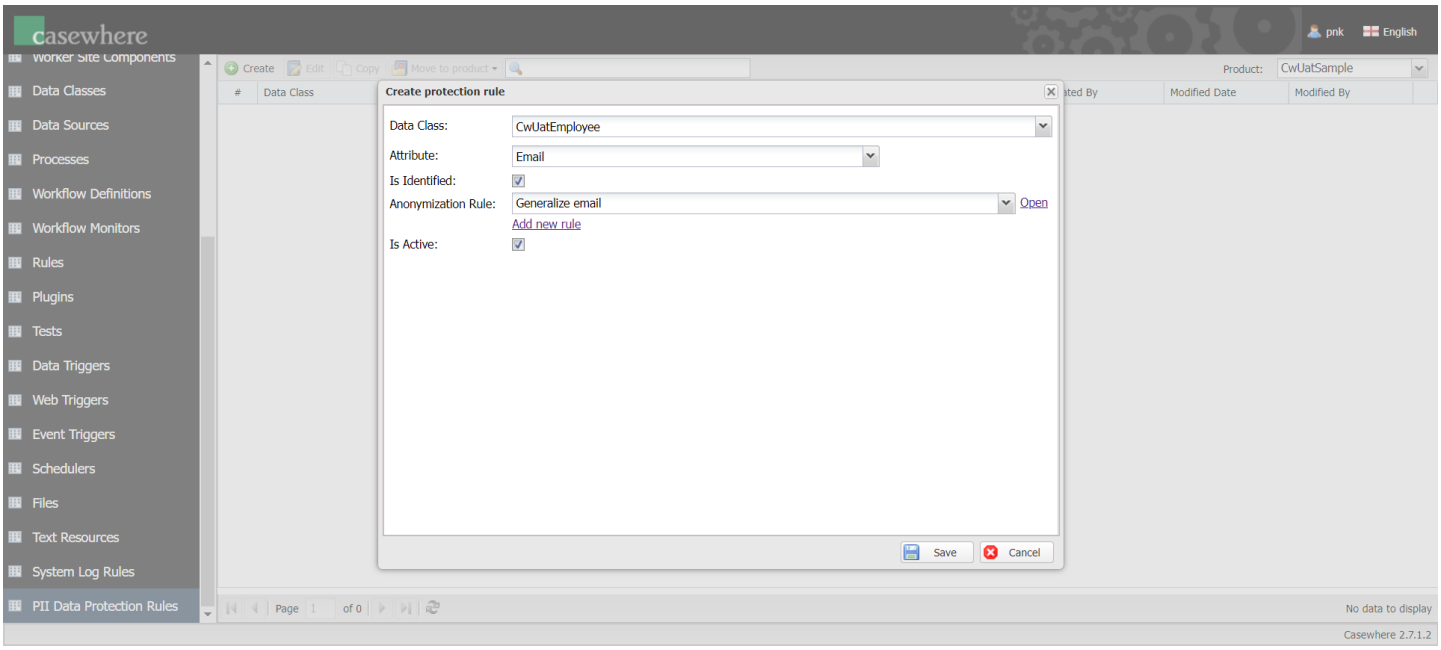
Casewhere does not offer any built-in method for anonymizing data. Instead, the platform allows you to define rules to anonymize data. That is, how data is anonymized is really up to your business, and you have the full freedom to decide what and how data is anonymized. There are a few techniques you can consider for your project:

- Data masking: You can configure a rule to hide data with altered values. For example, you can replace phone number digits with the symbol "*". Data masking makes reverse engineering or detection impossible.
- Pseudonymization: You can configure a rule to replace real names with fake names defined from a given dataset. Pseudonymization preserves statistical accuracy and data integrity, allowing the modified data to be used for training, development, testing, and analytics while protecting data privacy.
- Generalization: You can configure a rule to remove some of the data to make it less identifiable. For example, john@globeteam.com can be transformed to anonymous19001570@globeteam.com, which is still a valid, unique email. We still know the user comes from Globeteam, but it's impossible to identify who he is.

Anonymization rules can be packaged as standard products for reuse. Casewhere provides APIs so you can easily customize the process of anonymization to fulfill your business requirements. In this article, we will guide you through configuring a simple anonymization process, but you can totally find and install similar applications in the Casewhere standard package library to save effort. In reality, the business requirement can be a bit more complex. You might want to create a data class to keep track of all anonymization requests. You might also want to employ an approval process here as your business demands. Anonymizing data could be a heavy task if your system has a large database; you might need to schedule the jobs to run in the background at midnight.

Configuration

In Casewhere Admin, navigate to **PII Data Protection Rules**, here you can start create and manage anonymization rules for your applications.



When defining a new rule, you need to specify the followings:

- **Data Class**: Specify the data class to which the rule will apply
- **Attribute**: Specify the attribute to which the rule will apply
- **Is Identified:** When individuals request to be forgotten, they need to provide their identity (CPR, Email, etc.) for matching and verification. This identity information will later be searched against the Casewhere database. This data collection step is a part of the anonymization process, and as a developer, you need to tell Casewhere which fields it can use to search.
- **Anonymization Rule**: Select or create a new server rule to transform data. You can also learn more about rule development here. Below are some examples:
    - Generalize email, e.g., convert "john@domain.com" to "Anonymous190014570@domain.com"

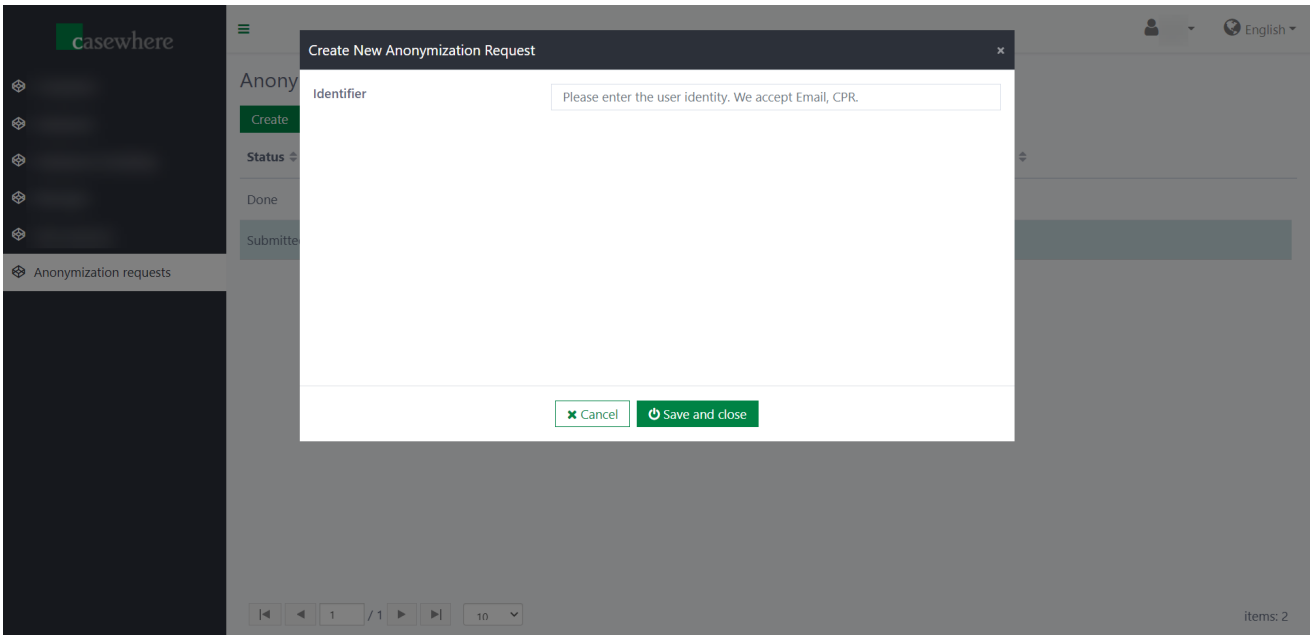
```
#predicate
var parts = @parameters.OriginalValue.Split(new char[]{ '@' },
StringSplitOptions.RemoveEmptyEntries);
var id = Guid.NewGuid().ToString().Replace("-", "");
parts[0] = $"Anonymous{id}@";
return String.Join("", parts);
```

    - Mask phone number, e.g., convert "50-12-34-56" to "50-xx-xx-xx"

```
#predicate
return @parameters.OriginalValue.Substring(0, 2) + "-xx-xx-xx";
```

After defining rules, the next step is making a workflow to anonymize data for your application. It's up to your business how the workflow should be but typically, there should be 2 activities:

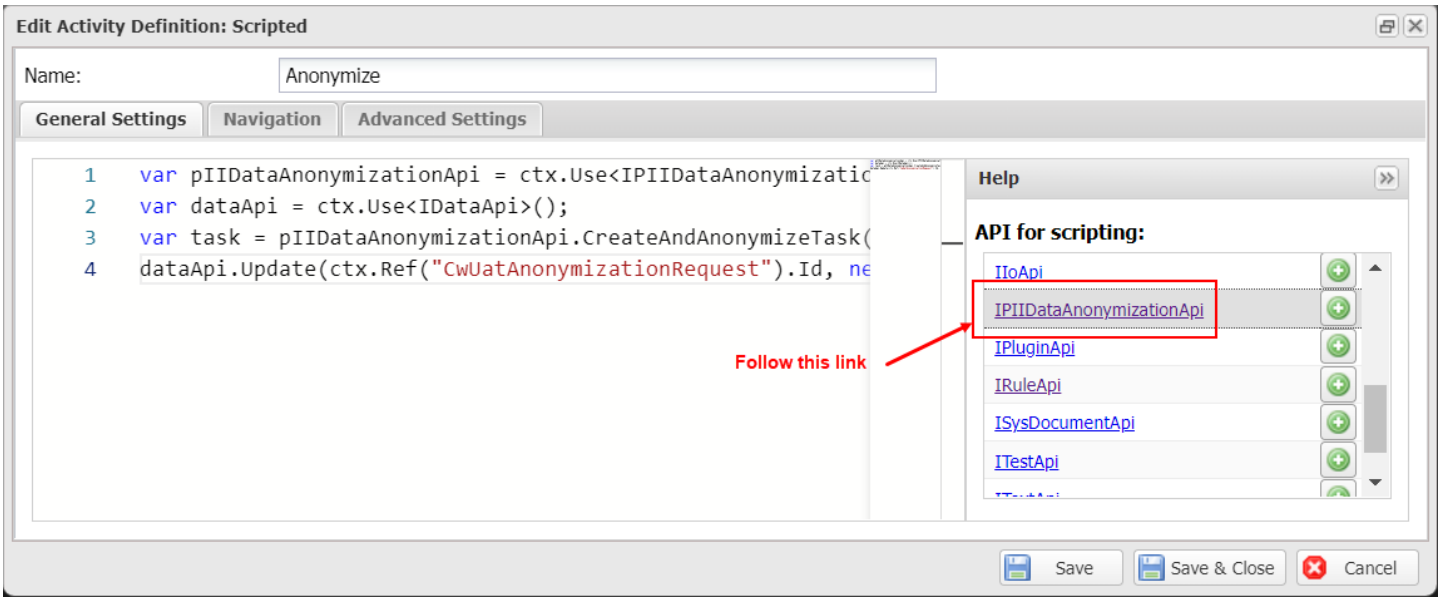
1. Form activity: to collect the user identity

- 

2. Scripted activity: Anonymize data using Casewhere API.

```
c# var pIIDataAnonymizationApi = ctx.Use<IPIIDataAnonymizationApi>(); var dataApi = ctx.Use<IDataApi>
(); var task =
pIIDataAnonymizationApi.CreateAndAnonymizeTask(ctx.Ref("CwUatAnonymizationRequest").Identifier);
```

You can learn more about `IPIIDataAnonymizationApi` by visiting this in-app document.



OWASP Top Ten - 2021

One of the most critical missions of Casewhere is to provide applications a strong security foundation to protect their data and users. We adopts OWASP Top Ten as one of standard security guidelines and constantly evaluate Casewhere against these security threats. We also understand that OWASP Top Ten is just a starting point.

A01 – Broken Access Control

Except for public resources, Casewhere denies all accesses by default. Access control in Casewhere can be done at multiple layers. Casewhere embraces the principle of least privilege and the defense-in-depth approach by enabling the ability to assign access rules to all types of resources:

1. Worker site: This is the highest level of protection. Rules assigned to a worker site are always enforced when users access a resource in the worker site
2. Page: To protect the page itself, including widgets and workflow actions inside the page
3. Widget: To protect the widget itself, including widget data and workflow actions inside the widget
4. Widget action: To protect the triggered workflow
5. Data source: To protect all data queries on the data sources
6. Data class: To protect read/write access to data at the field level
7. Custom access control: It's always possible to implement custom access control logic in data filters and workflows

Last but not least, Casewhere logs all unauthorized access attempts by default.

A02 – Cryptographic Failures

First of all, all Casewhere deployments enforce HTTPS by default. Secondly, data at rest will be protected by disk encryption. If the customer can adopt MongoDB Enterprise, there will be an extra encryption layer at rest. At the application level, Casewhere employs Microsoft Data Protection API with strong encryption algorithms; encryption keys are also encrypted at rest and rotated frequently. We have very strict policies to store the key-encryption key (i.e., the key to encrypt the encryption keys). It's possible in Casewhere to enable field-level encryption for sensitive data such as credit card numbers, social security numbers, etc.

A03 – Injection

Casewhere applies many techniques to prevent injection threats:

1. All data queries are parameterized
2. User-supplied data is always validated against the data schema
3. Hostile APIs such as excuting script on server side are turned off by default

A04 – Insecure Design

Security is a feature in Casewhere, as our top mission is to provide developers with a solid security toolbox to develop secure applications. Security requirements go side by side with business requirements in the whole development lifecycle, from the early feature analysis, functional designing with threat modeling, and data modeling to implementation, testing, and even documentation. The development process of Casewhere embraces secure-by-design principles:

- Minimize attack surface area: Every time a developer adds a feature, he is increasing the risk of a security vulnerability
- Establish secure defaults: The application must be secure by default. That means a new user must take steps to obtain higher privileges and remove additional security measures
- The principle of least privilege: A user should have the minimum set of privileges required to perform a specific task
- The principle of defence in depth: Multiple security controls that approach risks in different ways are the best option for securing an application
- Fail securely: Applications should fail in a secure way. Failure should not give the user additional privileges, and it should not

show the user sensitive information like database queries or logs
- Zero trust: The application should always check the validity of data sent by users or third-party services
- Avoid security by obscurity: If an application requires its administration URL to be hidden so it can remain secure, then it is not secure at all. There should be sufficient security controls in place to keep your application safe without hiding core functionality or source code.
- Keep security simple: Developers should avoid the use of very sophisticated architecture when developing security controls for their applications. Having mechanisms that are very complex can increase the risk of errors.

A05 – Security Misconfiguration

Casewhere is an application platform, so it's a top priority to provide documents and guidelines for deploying and configuring the platform most securely.

Besides guidelines and documents, Casewhere's import/export feature enables quickly migrating configuration across environments with different credentials used in each environment. This process should minimize the effort required to set up a new secure environment.

It's best practice to have a task to review and update the configurations appropriate to all security notes and updates. Many of these can be done automatically by using the Casewhere automation test feature. For example, you can write tests to verify if unnecessary features/endpoints in Casewhere are turned off and if the setting values have the expected value.

Finally, it's still the developer's responsibility to use Casewhere securely and also improve their awareness of different security aspects.

A06 – Vulnerable and Outdated Components

Casewhere has an integration to continuously monitor for vulnerabilities in the components. Our release model facilitates quick response to critical security issues while also providing periodical security patches.

A07 – Identification and Authentication Failures

Casewhere does not have built-in authentication, but the platform can integrate with almost wellknown IdPs such as Safewhere Identify, AD, AAD, which support all kinds of best practices for identification and authentication.

A08 – Software and Data Integrity Failures

To mitigate the threats, the Casewhere development enforces the following rules when using 3rd parties:

- Libraries and dependencies are from trusted repositories
- Review the code of 3rd parties for major components/integrations
- Integration in CI/CD pipelines to continuously monitor for vulnerabilities in the components

The platform also provides developers with essential APIs and components for protecting and verifying data integrity.

A09 – Security Logging and Monitoring Failures

Logging is the area Casewhere really puts its heart and soul into. Below are the highlights:

- Our Event Sourcing architect ensures that all data changes are recorded as events in the database. It's always possible to tell who did what and when
- The Casewhere auditing tracks both read/write accesses with high flexibility for customization. All unauthorized accesses are logged by default
- Logs and events are stored in a consistent, readable format
- Logs and events databases are managed separately with stricter policies. The applications can only write logs and have no

access to modify log entries once they are stored in the database
- The platform provides APIs to enable custom logging when needed
- Casewhere can write logs using different storage technologies (plug-and-play). The platform also provides components for exporting logs to other storage, as we understand that storing logs locally alone is not a good practice

Casewhere monitoring functionality makes it possible to:

- Monitor failed workflows
- Subscribe to failure events to trigger custom workflows. For example, you can implement a workflow to send an alert email when a payment transaction fails to execute. You can also implement a workflow to send the events to analytics platforms if your system demands a more sophisticated alert policy
- Configure rules to automatically retry failed workflows

A10 – Server-Side Request Forgery (SSRF)

Casewhere strictly adheres to the zero-trust principle and validate all data sent from users or external services. The platform does not have APIs that fetch remote resources based on user-supplied URLs. However, the risk is still high for applications on top of Casewhere if developers are not aware of this threat, especially when implementing RESTful API using web trigger. To mitigate the threat, we encourage Casewhere application developers to:

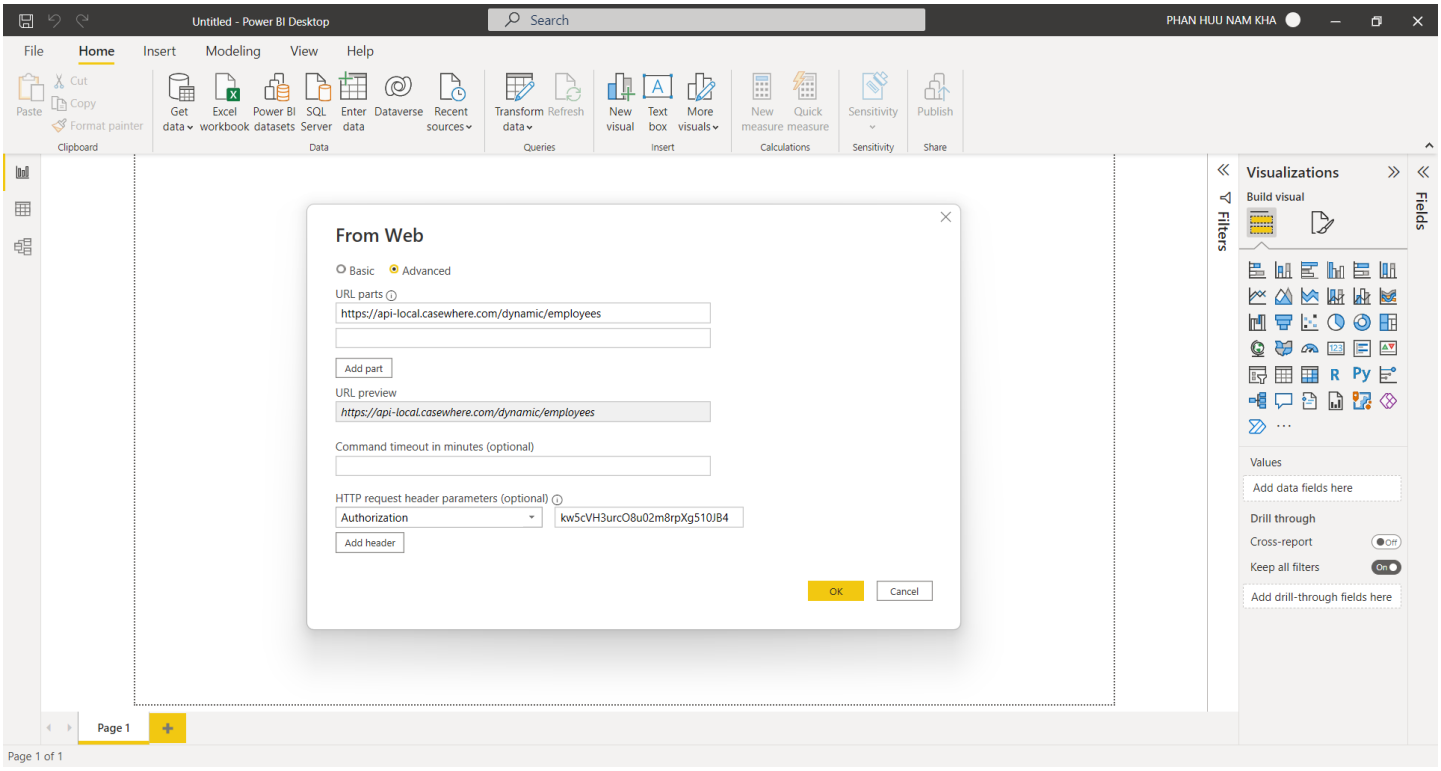
- Sanitize and validate all client-supplied input data
- Enforce the URL schema, port, and destination with a positive allow list. Do not mitigate SSRF via the use of a deny list or regular expression. Attackers have payload lists, tools, and skills to bypass deny lists
- Do not send raw responses to clients

Integrate with Power BI

Integrating with business intelligence platforms like Power BI is one of the best ways to visualize your data and gain more insights about the business. This article demonstrates the solutions to connect Casewhere data from Power BI, but the same techniques can be applied to other platforms.

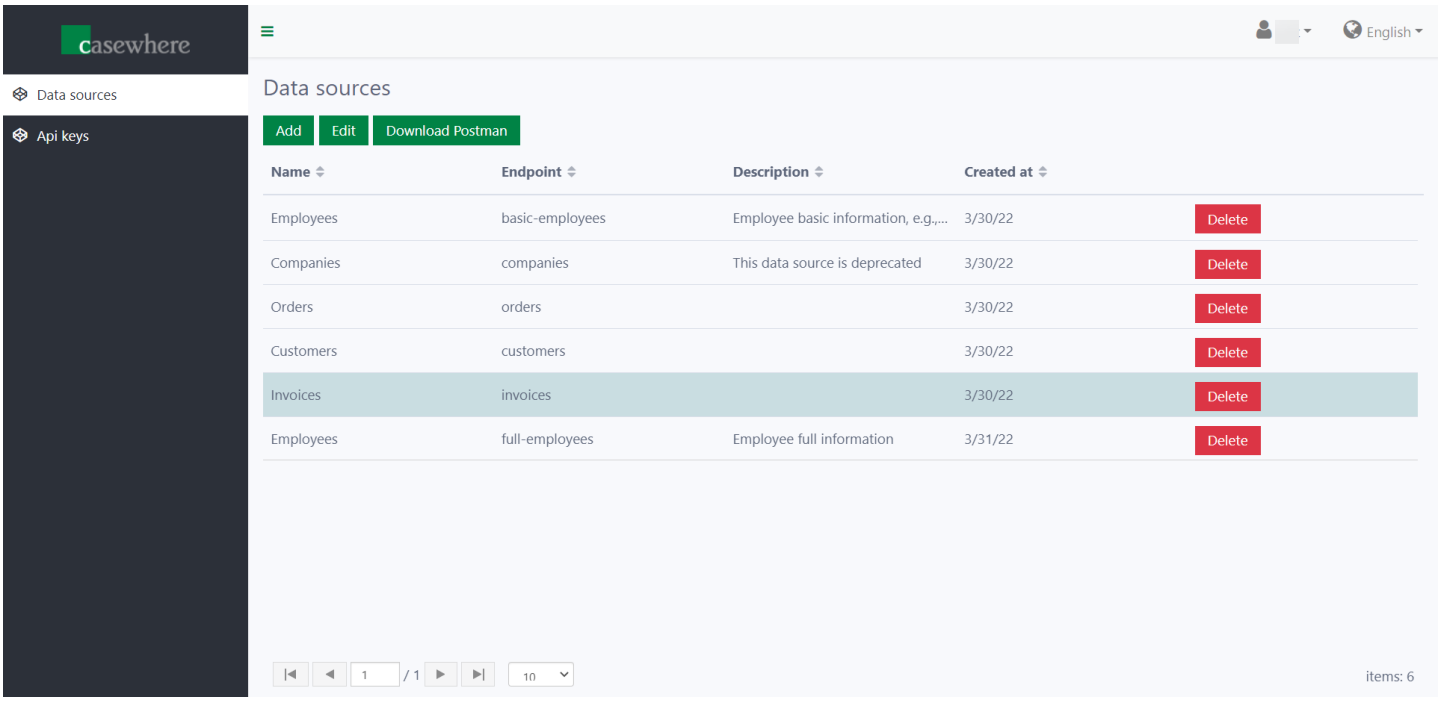
REST API

Like most business intelligence platforms, Power BI supports a Web Connector to load data from a web endpoint. Here you can use an API Key to protect the communication.

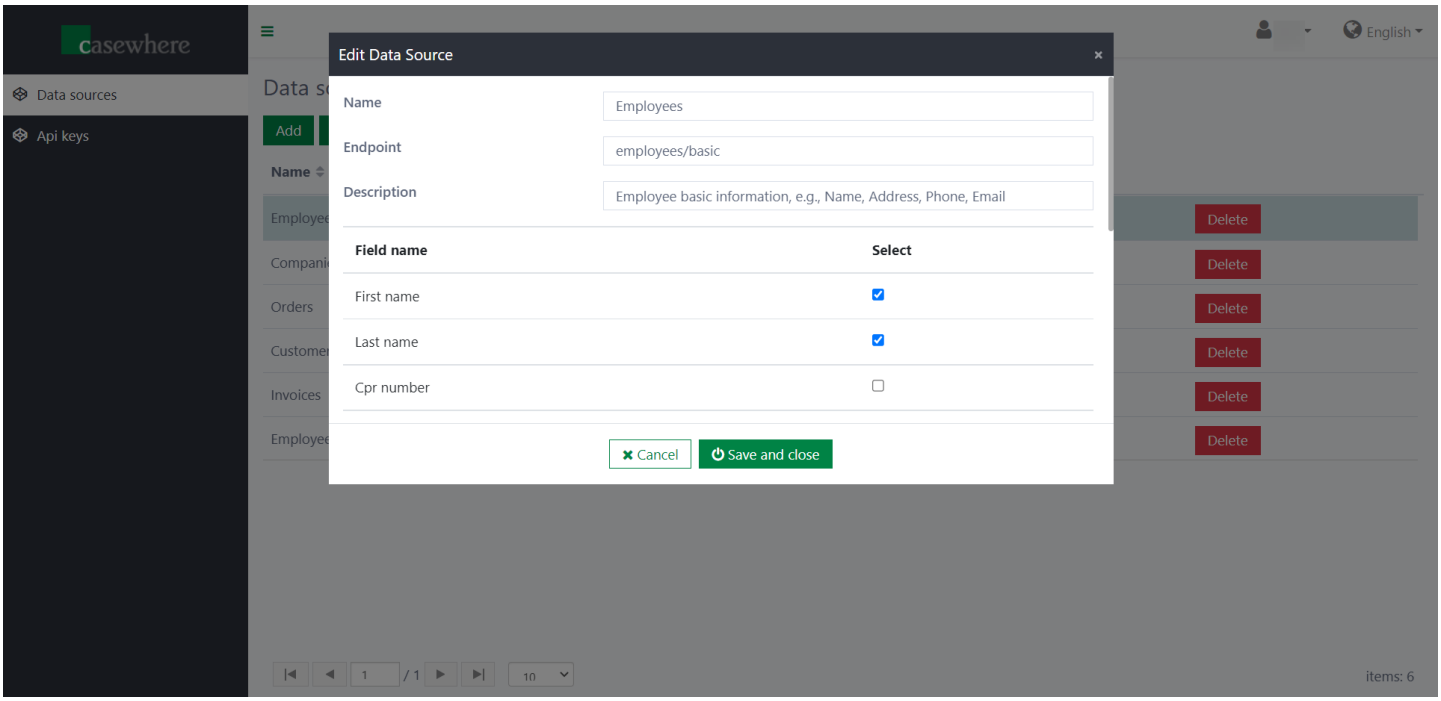


While you can totally implement the REST API yourselves with Casewhere web trigger, in this article, we want to recommend the component Rest data source, which is designed to optimize for scenarios like this. The component provides many friendly tools to simplify the configuration and speed up the development.

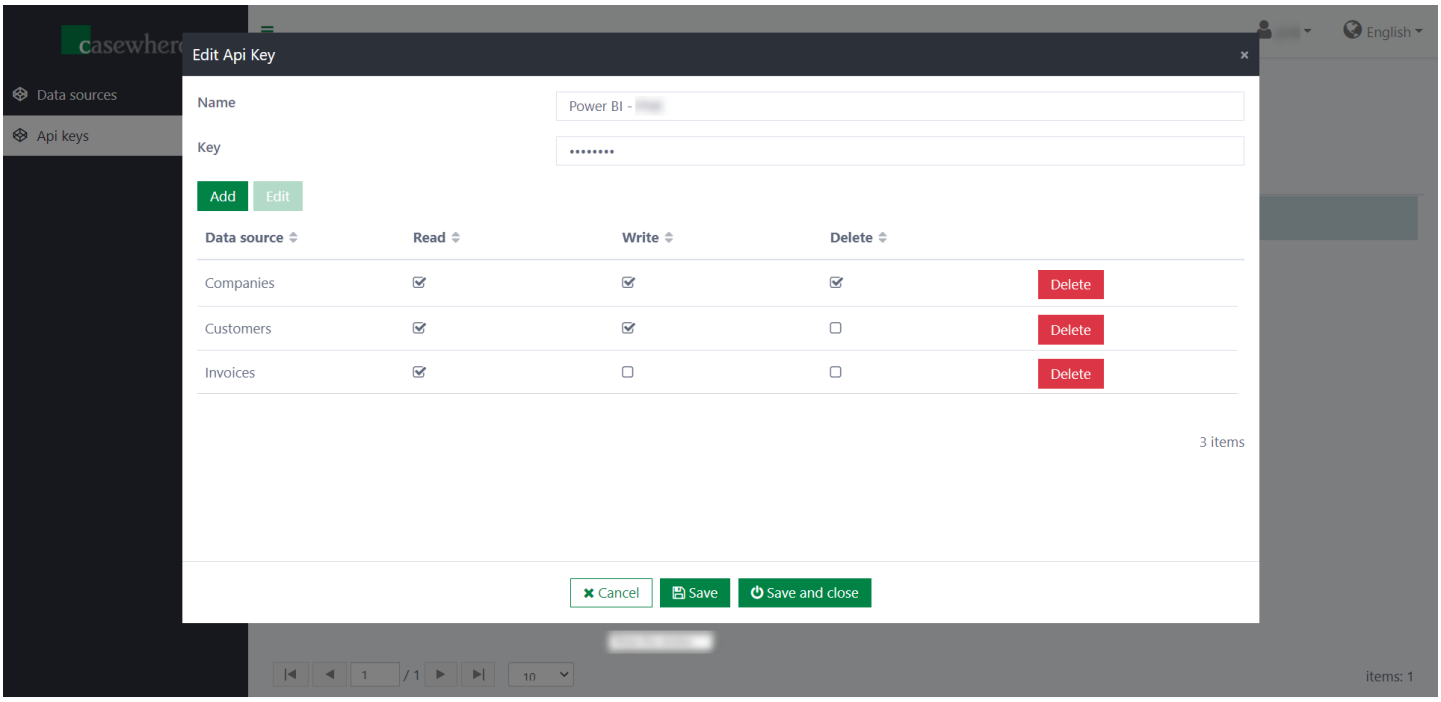
There is a friendly interface for managing data sources.



It's possible to create multiple endpoints for a single data source to return different portions of data. For example, you can create an endpoint `employees/full` to return all fields of employees, while having another one `employees/basic` just to return essential information.

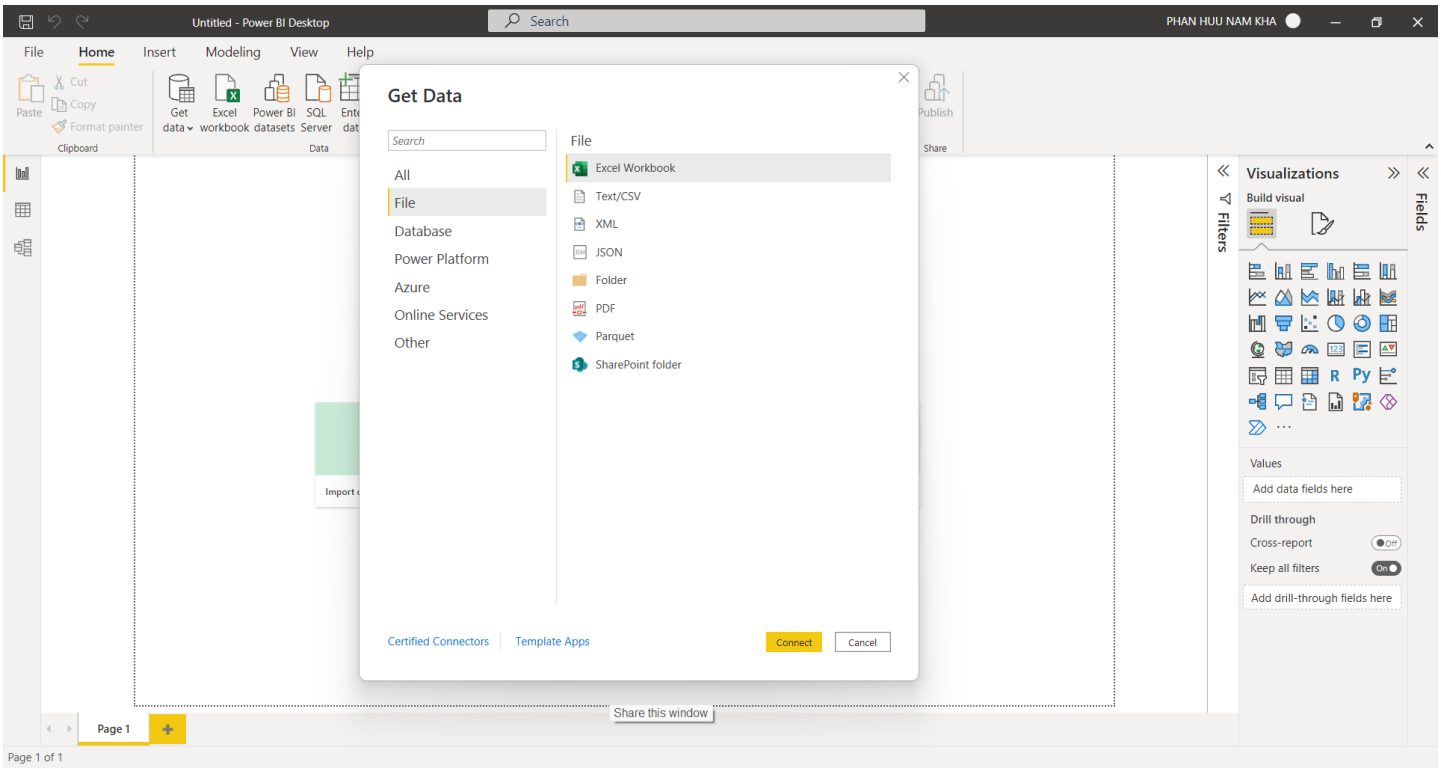


The component provides an interface for managing API keys. Here you can decide which data sources are enabled for a specific API key and which permissions (read, write, delete) the user (using the API key) has on each data source.



File extraction

Another popular approach is exporting data to a file and uploading it to storage where Power BI can read from.



Instead of implementing the integration from scratch, you can save a lot of effort by utilizing the component Data export. You can configure which and how data is exported, the location where the files are uploaded. The component supports SFTP by default, but it's totally possible to integrate with different file storages like FTP, Azure Storage, SharePoint, etc.

Integrate with Azure Active Directory (AAD)

Introduction

The authentication is an essential part in all applications. It's not a problem with Casewhere because Casewhere can be easy to integrate with all types of Identity Providers (IdP) that support the SAML 2.0 authentication. This guide shows you how to integrate Casewhere with an Azure AD tenant. There are two ways to do it

- Use many applications in Azure AD to connect with a Casewhere instance
- Reuse one application in Azure AD to connect with a Casewhere instance

Basically, a Casewhere instance include four main sites

- Admin site
- Job site
- Worker site
- Worker API site

No matter the way you choose, integrating with Azure AD require configuring the settings at two sides: Azure side and Casewhere side. It helps to build their trust together. You always start to integrate the Admin site with an application in Azure AD first, then integrate the other sites with Azure AD.

Integrate with AAD by using many applications

You need to create three applications to connect three sites in the Casewhere instance like Admin site, Job site and Worker site

Admin site

Configure the settings in Azure AD

Register an application for the Admin site

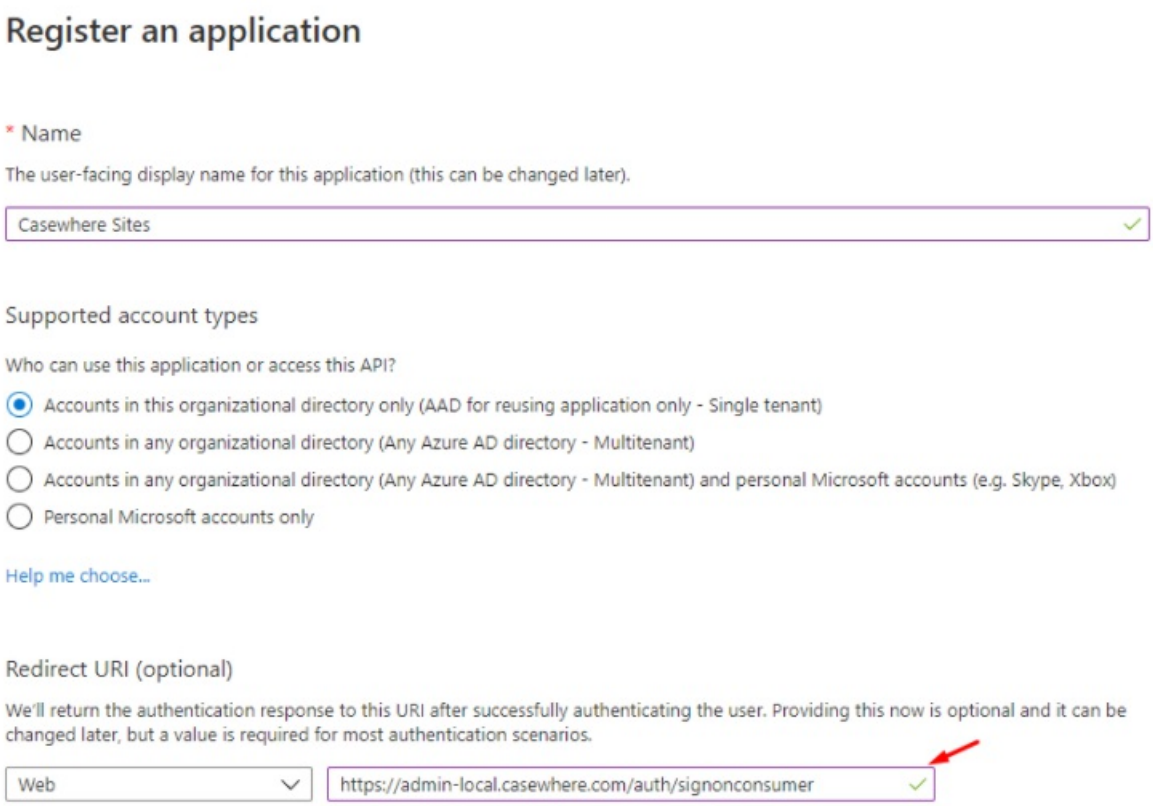
You can follow the following link to register an application in Azure AD

Quickstart: Register an app in the Microsoft identity platform | Microsoft Docs

In the **Redirect URI (optional)** section, enter the link with the format below

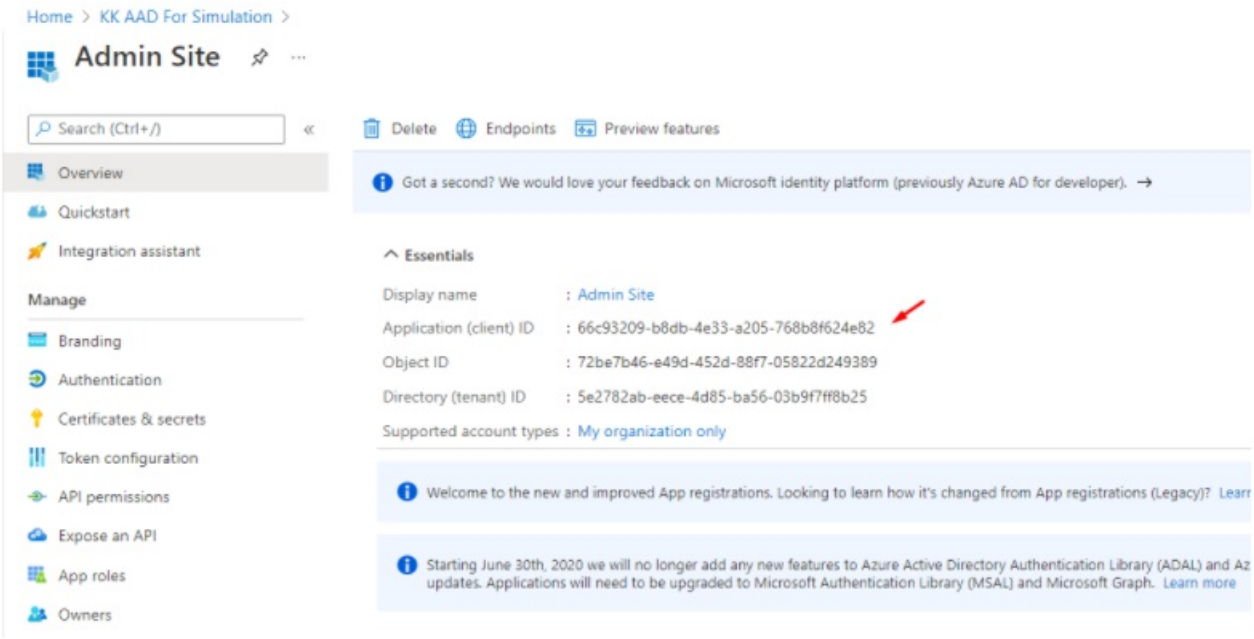
https://[admin site domain]/auth/signonconsumer

E.g. https://admin-local.casewhere.com/auth/signonconsumer



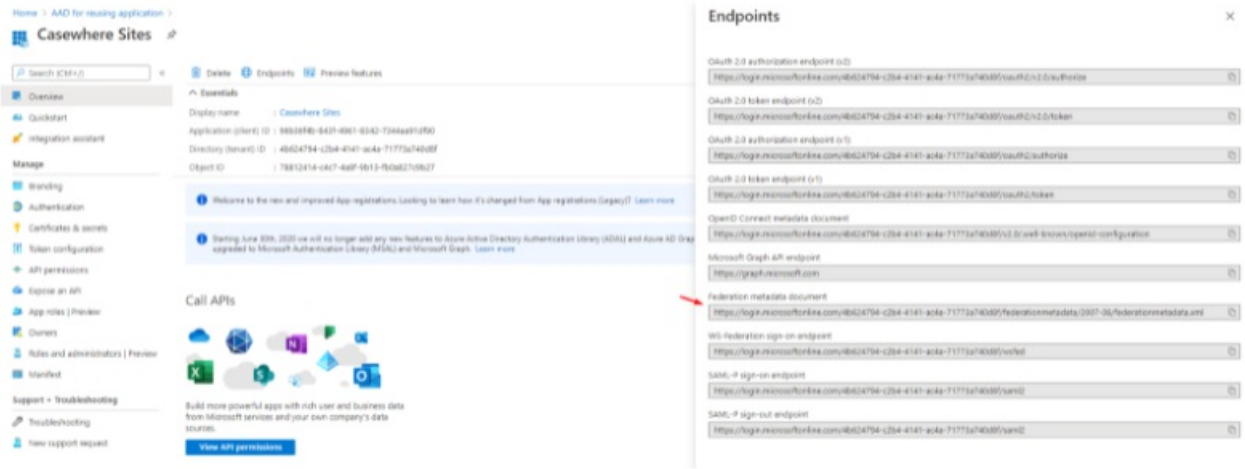
Get the application ID

After creating the application, you must write down the application ID because it will used later.



Download the metadata

On **Overview** page, click the **Endpoints** button and get the **Federation metadata document** link. Then open this link and save it as the xml file.



Configure authentication

- Select the **Authentication** menu
- Configure Redirect URIs to the Admin site:

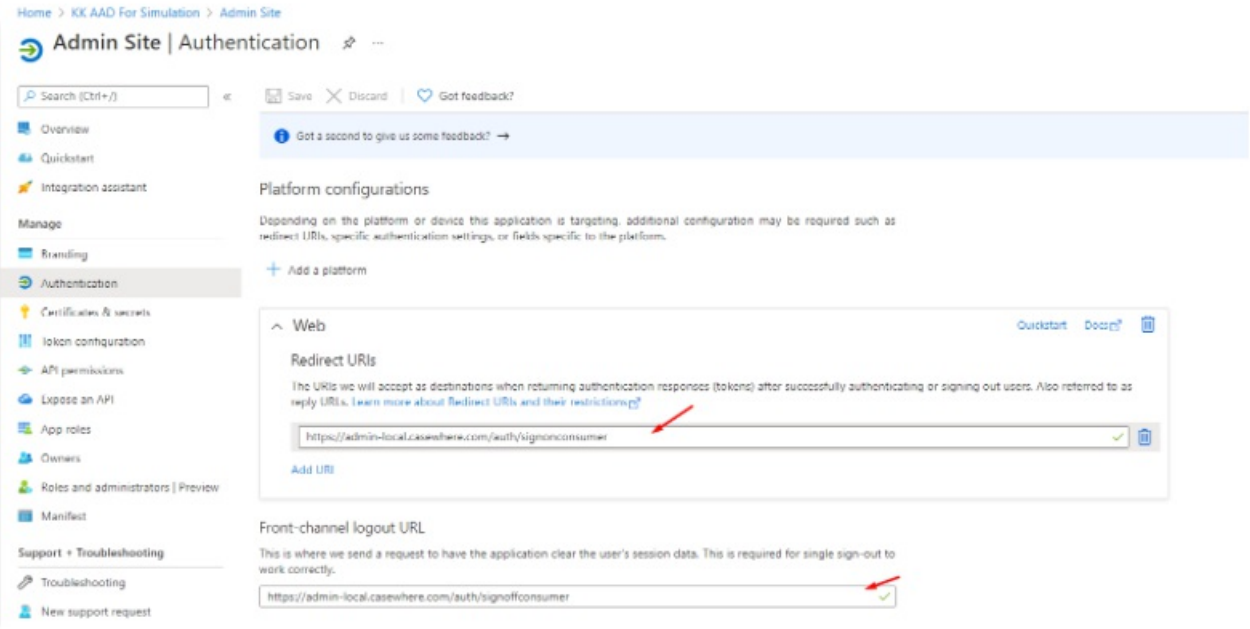
  https://[admin site domain]/auth/signonconsumer

E.g. https://admin-local.casewhere.com/auth/signonconsumer

- Configure Front-channel logout URL to the Admin site

  https://[admin site domain]/auth/signoffconsumer

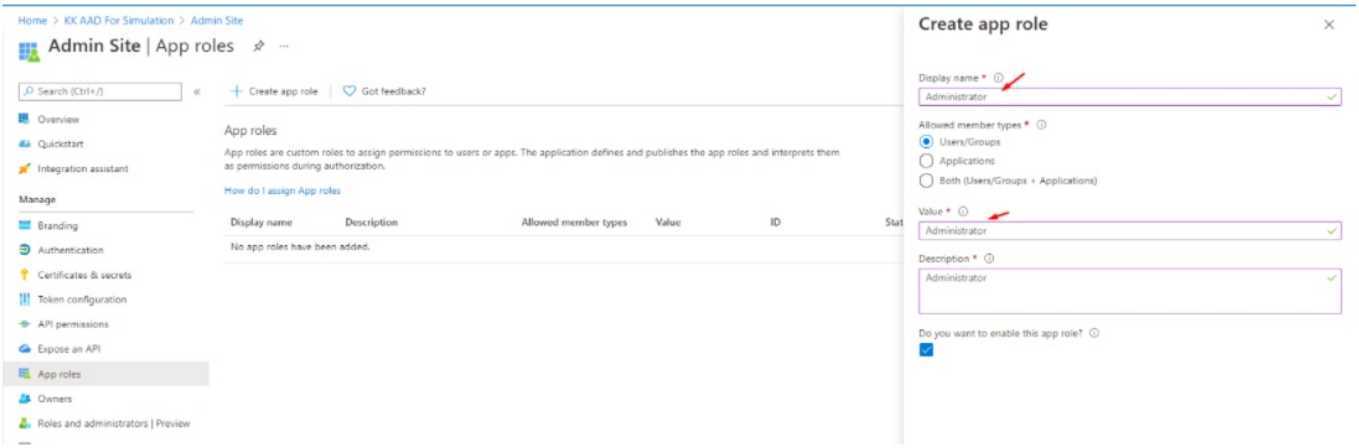
E.g. https://admin-local.casewhere.com/auth/signoffconsumer



Create App roles in the application

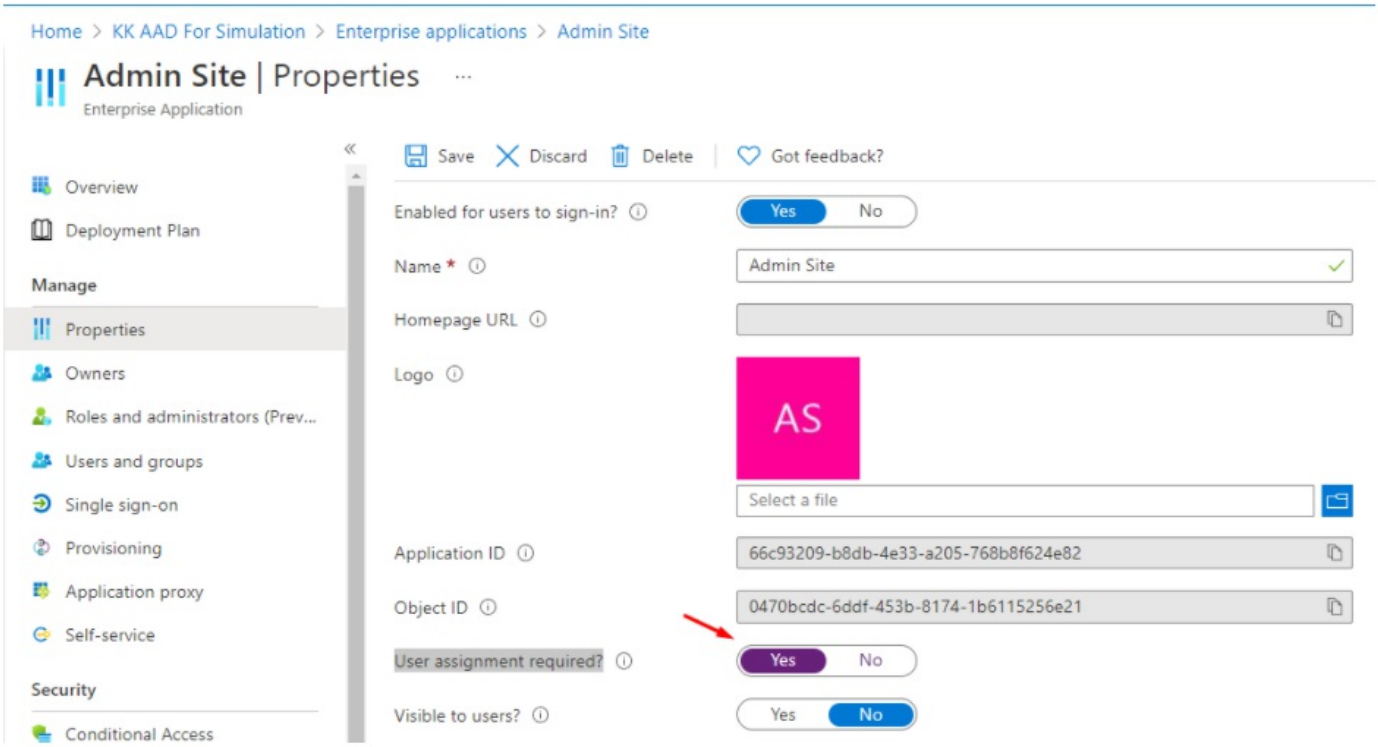
- Select the **App roles** menu
- Click the **Create app role** button
- Create a role with the display name = **Administrator,** the value = **Administrator**
- Select and click the **Save** button

- Select the **Users/Groups** option in the **Allowed member types** field
- Click the **Save** button



Enable user assignment

- Go back to the **Overview** page of the AAD tenant
- Select the **Enterprise applications** menu
- Select the application you created
- Select the **Properties** menu
- Check the **Yes** mode in the **User assignment required?** Field
- Click the **Save** button

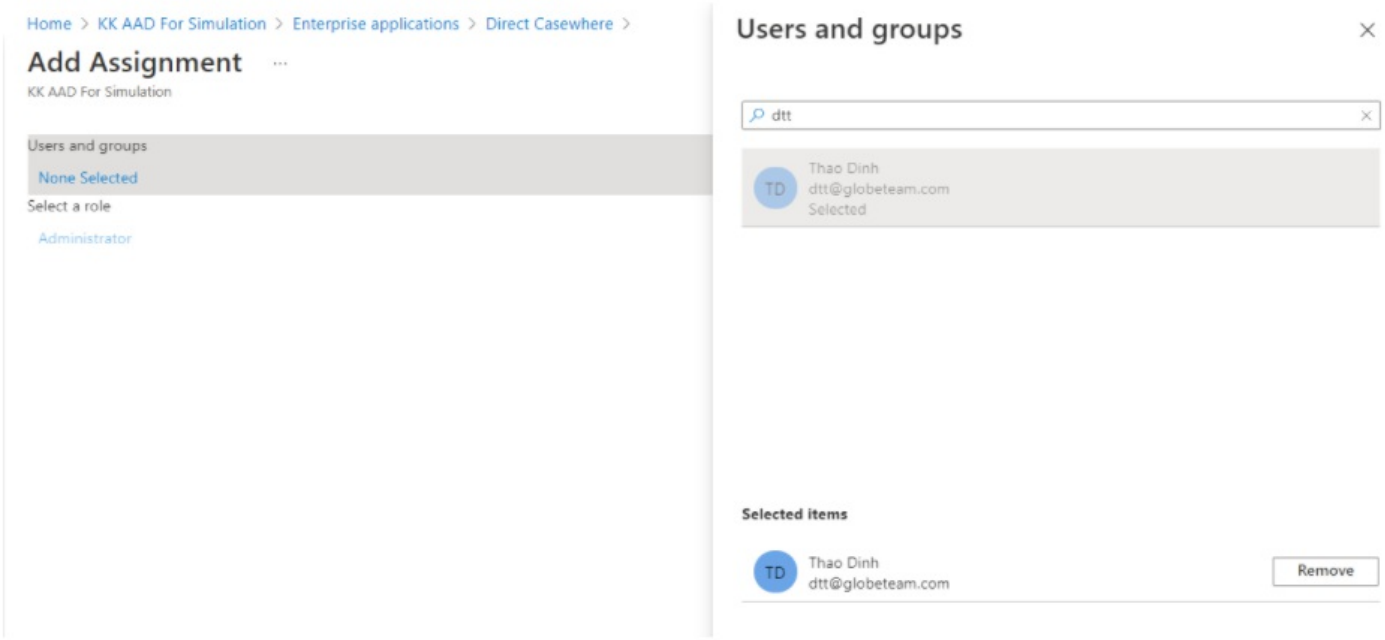


Assign App roles to a user

Assume that there is at least one user in the AAD tenant. If not, follow this link Add or delete users - Azure Active Directory | Microsoft Docs

- Go back to the **Overview** page of the AAD tenant
- Select the **Enterprise applications** menu

- Select the application you created
- Select the **Users and groups** menu
- Click the **Add user/group** button
- Click the **Selected** link under the **Users** section, then select the users you want to assign App roles
- Similarly, click the **Selected** link under **Select a role** section, then select the **Administrator** role
- Finally, click the **Save** button



Configure the settings in Admin site

Install public signing certificate

- Open the metadata of the application on the browser like Edge
- Copy the text content of the element EntityDescriptor/Signature/KeyInfo/X509Data/X509Certificate
- Open the text editor like Notepad
- Paste it into the editor
- Save it under the ***.cer** extension file

On the on premise environment, you need to install it into the **LocalMachine/TrustedPeople** location on the host machine. Otherwise, on the cloud environment like Azure portal, you need to upload it into the App Service. This guide focuses on the on premise environment.

***Note:** Please this step because you repeat it at the other sites.

Update metadata

- Navigate to the **Casewhere.Configuration.Web** folder in the Casewhere instance
- Create a **Metadata** folder, go into this folder and create an **AAD** folder
- Then copy the metadata of the application into the **AAD** folder

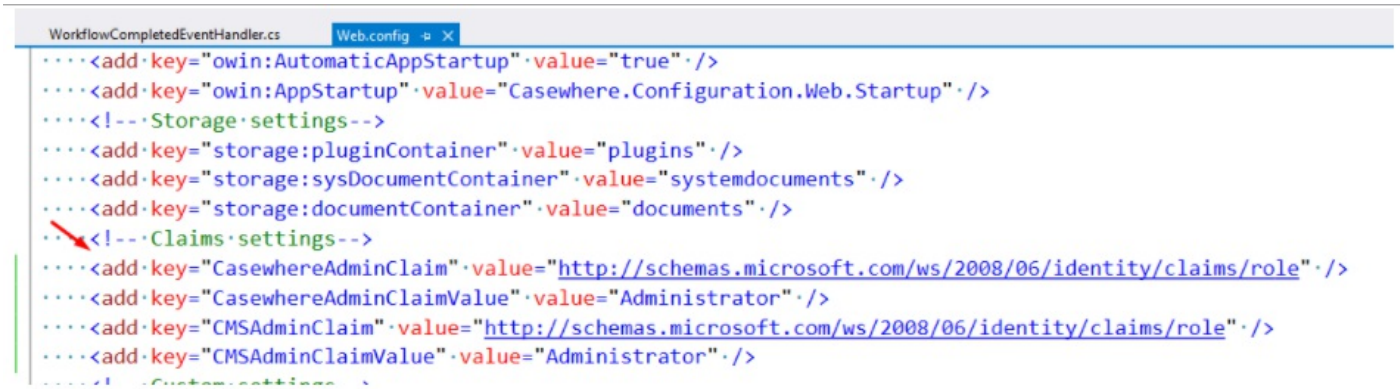
ChangeWeb.config in Admin site

You need to change the following settings in the **appSettings** element of the Web.config file

In **Claims settings** section

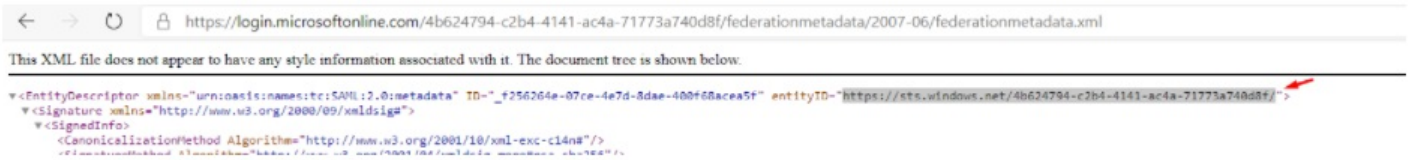
- CasewhereAdminClaim: Set it to *http://schemas.microsoft.com/ws/2008/06/identity/claims/role*
- CasewhereAdminClaimValue: Set it to **Administrator**
- CMSAdminClaim: Set it to *http://schemas.microsoft.com/ws/2008/06/identity/claims/role*

- CMSAdminClaimValue: Set it to **Administrator**



In the **IdP Connection** section

- idpConnection:IdPUrl: Open the metadata file and get it from the **entityID** attribute of the **EntityDescriptor** element



- idpConnection:IdPMetadataFolder: Enter the relative path **~/Metadata/AAD**
- idpConnection:EntityId: Put the application ID of the application in Azure AD into it.
- idpConnection:LogoutResponseUri: Put the sign-off endpoint of the Worker API site. It looks like

  https://[admin site domain]/auth/signoffconsumer

E.g. https://admin-local.casewhere.com/auth/signoffconsumer

Job site

Similar to the Admin site, you need configure both sides

Configure the settings in Azure AD

Register an application for the Job site

Create another application in Azure AD to work with the Job site. In the **Redirect URI (optional)** section, enter the link with the format below

https://[job site domain]/auth/signonconsumer

E.g. https://job-local.casewhere.com/auth/signonconsumer

Then, go to the **Overview** menu and write down the application ID because it will used later. Next, click the **Endpoints** button and get the **Federation metadata document** link and download the metadata file

Configure authentication

- Select the **Authentication** menu
- Configure Redirect URIs to the Job site

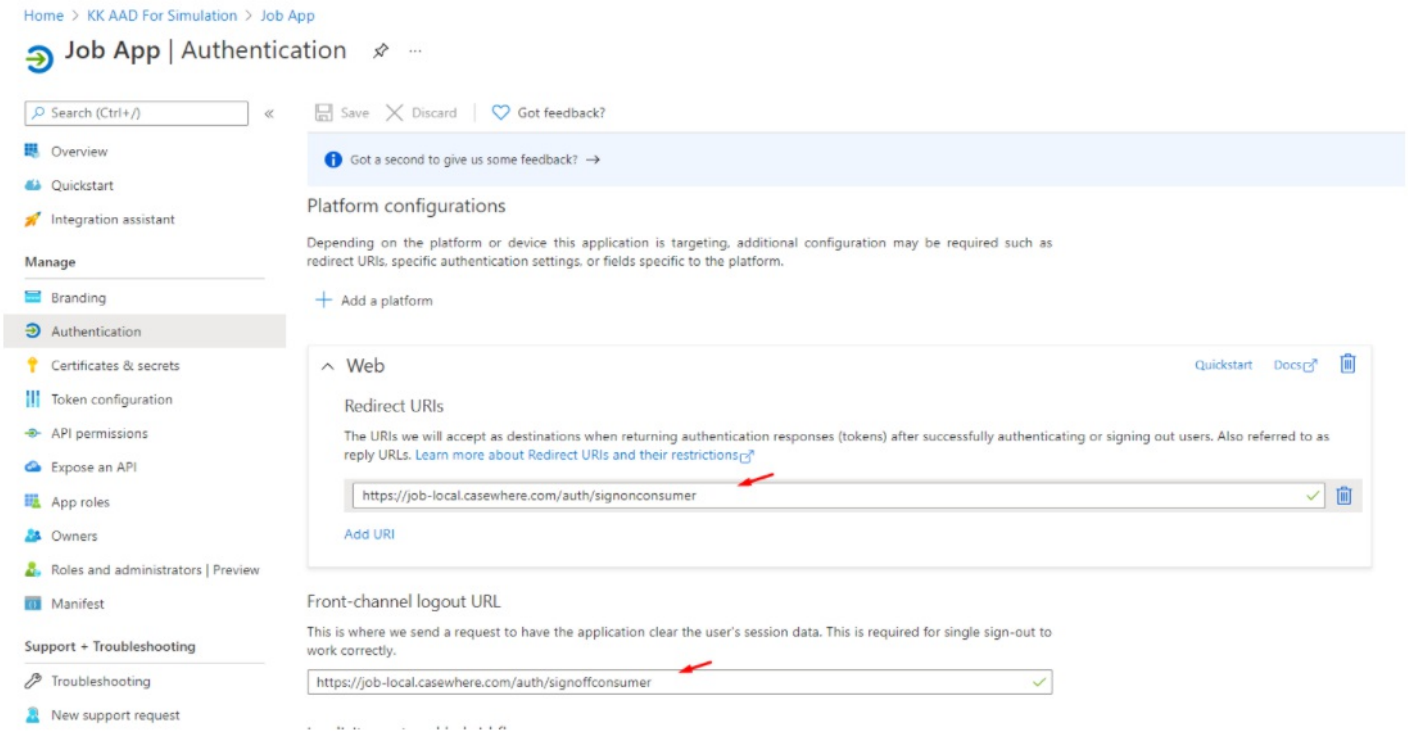
  https://[job site domain]/auth/signonconsumer

E.g. https://job-local.casewhere.com/auth/signonconsumer

- Configure Front-channel logout URL to Job site

https://[job site domain]/auth/signoffconsumer

E.g. https://job-local.casewhere.com/auth/signoffconsumer



\

Then, follow the following steps in the **Admin site** section and do it similarly to the new application.

- Create App roles in the application
- Enable user assignment
- Assign App roles to a user

Configure the settings in Job site

Install public signing certificate

Similar to the Admin site

Update metadata

- Navigate to the **Casewhere.Hangfire** folder
- Create a **Metadata** folder, go into this folder and create an **AAD** folder
- Then copy the metadata of the application into the **AAD** folder

Change Web.config in Job site

You need to change the following settings in the **appSettings** element of the Web.config file

In **Claims settings** section

- CasewhereAdminClaim: Set it to *http://schemas.microsoft.com/ws/2008/06/identity/claims/role*
- CasewhereAdminClaimValue: Set it to **Administrator**
- 

In the **IdP Connection** section

- idpConnection:IdPUrl: Open the metadata xml file and get the value of the **entityID** attribute of the **EntityDescriptor** element
- idpConnection:IdPMetadataFolder: Enter the relative path **~/Metadata/AAD**
- idpConnection:EntityId: Put the application ID into it.
- idpConnection:LogoutResponseUri: Put the sign-off endpoint of the Worker API site. It looks like

https://[job site domain]/auth/signoffconsumer

E.g. https://job-local.casewhere.com/auth/signoffconsumer

Worker site

You need to use the Worker API site to perform the configuration

Configure the settings in Azure AD

Register an application for the Worker site

Create another application in Azure AD to work with the Worker site. In the **Redirect URI (optional)** section, enter the link with the format below

https://[worker api domain]/signonconsumer

E.g. https://api-local.casewhere.com/signonconsumer

Then, go to the **Overview** menu and write down the application ID because it will used later. Next, click the **Endpoints** button and get the **Federation metadata document** link and download the metadata file.

Configure authentication

- Select the **Authentication** menu
- Configure Redirect URIs to the Admin site

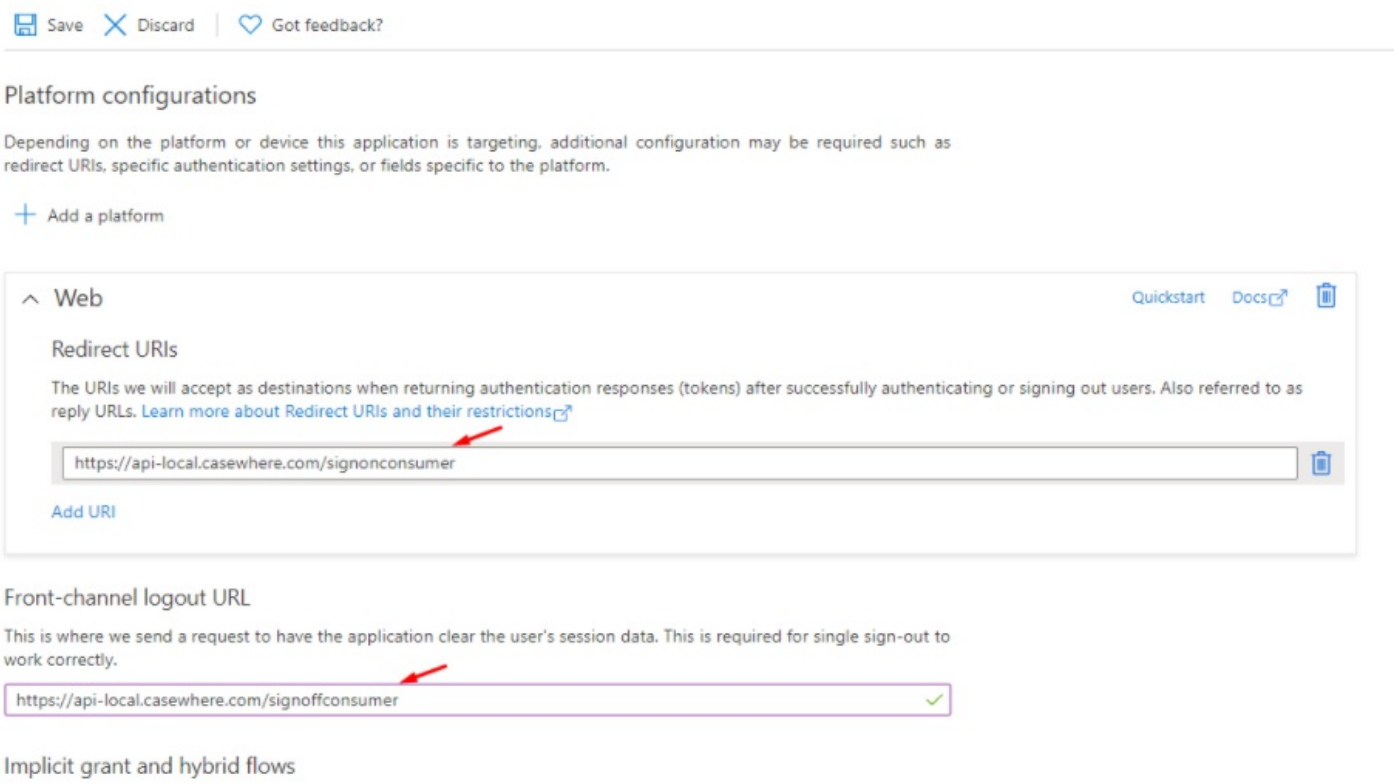
  https://[worker api site domain]/signonconsumer

E.g. https://api-local.casewhere.com/ signonconsumer

- Configure Front-channel logout URL to the Admin site

  https://[worker api site domain]/signoffconsumer

E.g. https://api-local.casewhere.com/signoffconsumer



Next, follow the following steps in the **Admin site** section and do it similarly to the new application.

- Create App roles in the application
- Enable user assignment
- Assign App roles to a user

Configure the settings in Worker site

These steps below assume that you can log in the Admin site successfully and on it.

Install public signing certificate

Similar to the Admin site

Create a IdP Connection

- Navigate to the **IdP Connections** menu
- Click the **Create** button, choose the **SAML 2.0** option
- In the **Name** field, enter the name like **Login Azure For API**
- In the **Signing Certificate Thumbprint** field, enter the thumbprint of the Casewhere signing certificate (Ask NTU if you need the support)
- In the **IdPUrl** field, enter the link you get from the value of the **entityID** attribute of the **EntityDescriptor** element when you open the metadata of the application.
- Upload the metadata file from Azure AD into the **Metadata** field
- In the Claim Mapping section,

UserId: http://schemas.microsoft.com/identity/claims/objectidentifier

Username: http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name

Roles: http://schemas.microsoft.com/ws/2008/06/identity/claims/role

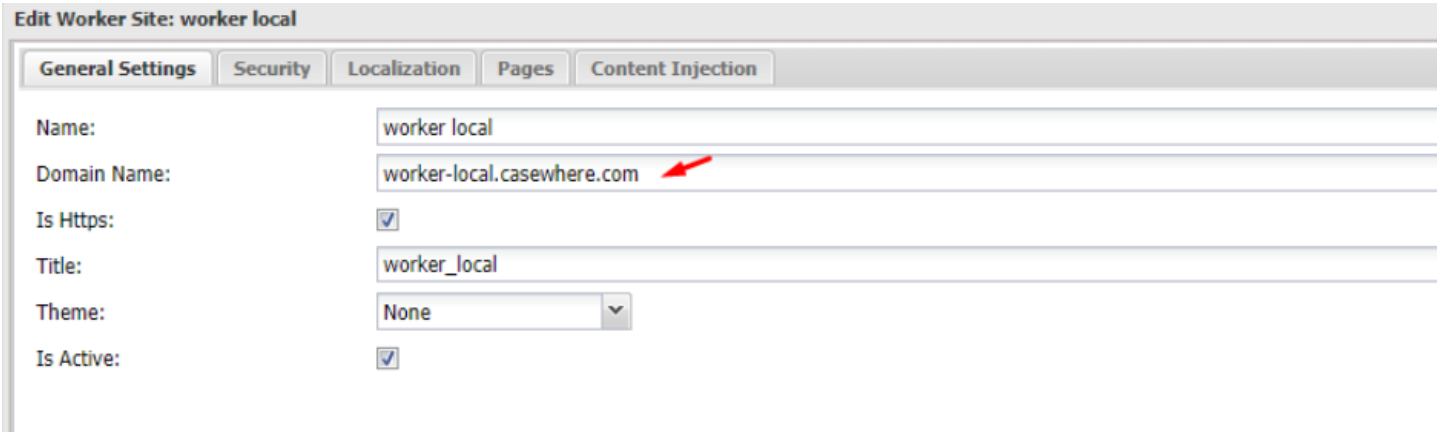
- Finally save it.

Create a public worker site

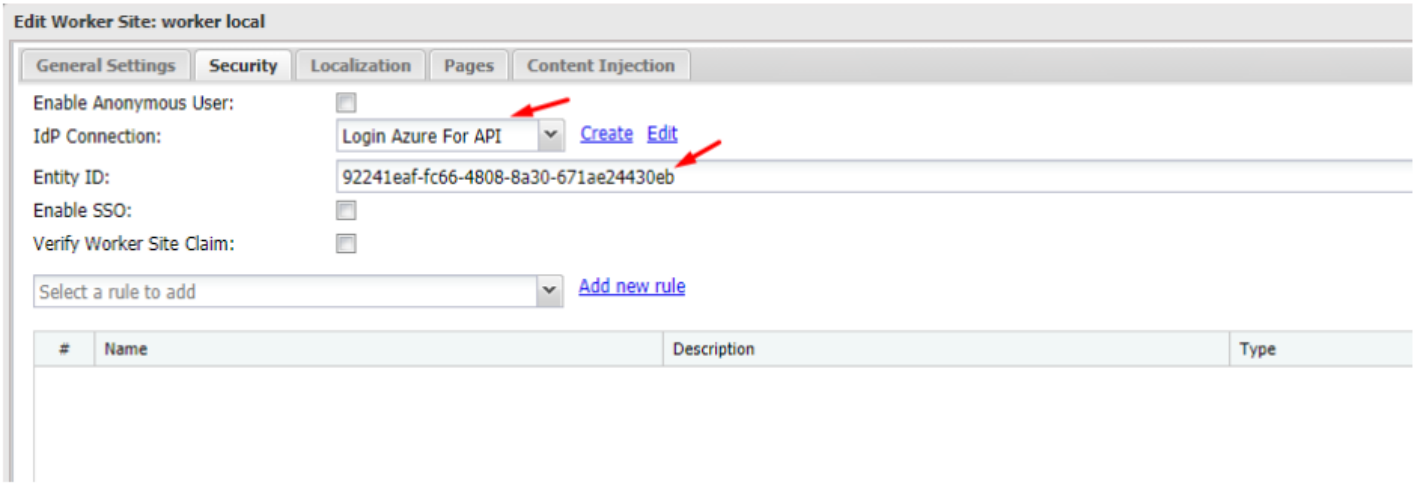
- Navigate to the **Worker Sites** menu, click the **Create** button
- In the **Name** field, enter a suitable name in the business context, like CPK Caseworker site
- In the **Domain Name** field, enter the domain of that site.

E.g. worker-local.casewhere.com

- In the **Theme** field, select a theme or None
- Check **Is active** field



- Go to the **Security** tab, select the **Login Azure For API** option in the **IdP Connection** dropdown
- In the **Entity ID** field, enter the application ID
- Uncheck the **Verify Worker Site Claim** checkbox
- Click the **Save** button



Integrate with AAD by reusing one only application

Configure the settings in Azure AD

Register an application for

Create another application in Azure AD to work with the sites of the Casewhere instance. In the **Redirect URI (optional)** section, enter the link with the format below

https://[admin site domain]/auth/signonconsumer

E.g. https://admin-local.casewhere.com/auth/signonconsumer

Then, go to the **Overview** menu and write down the application ID because it will used later. Next, click the **Endpoints** button and

get the **Federation metadata document** link and download the metadata file

Configure authentication

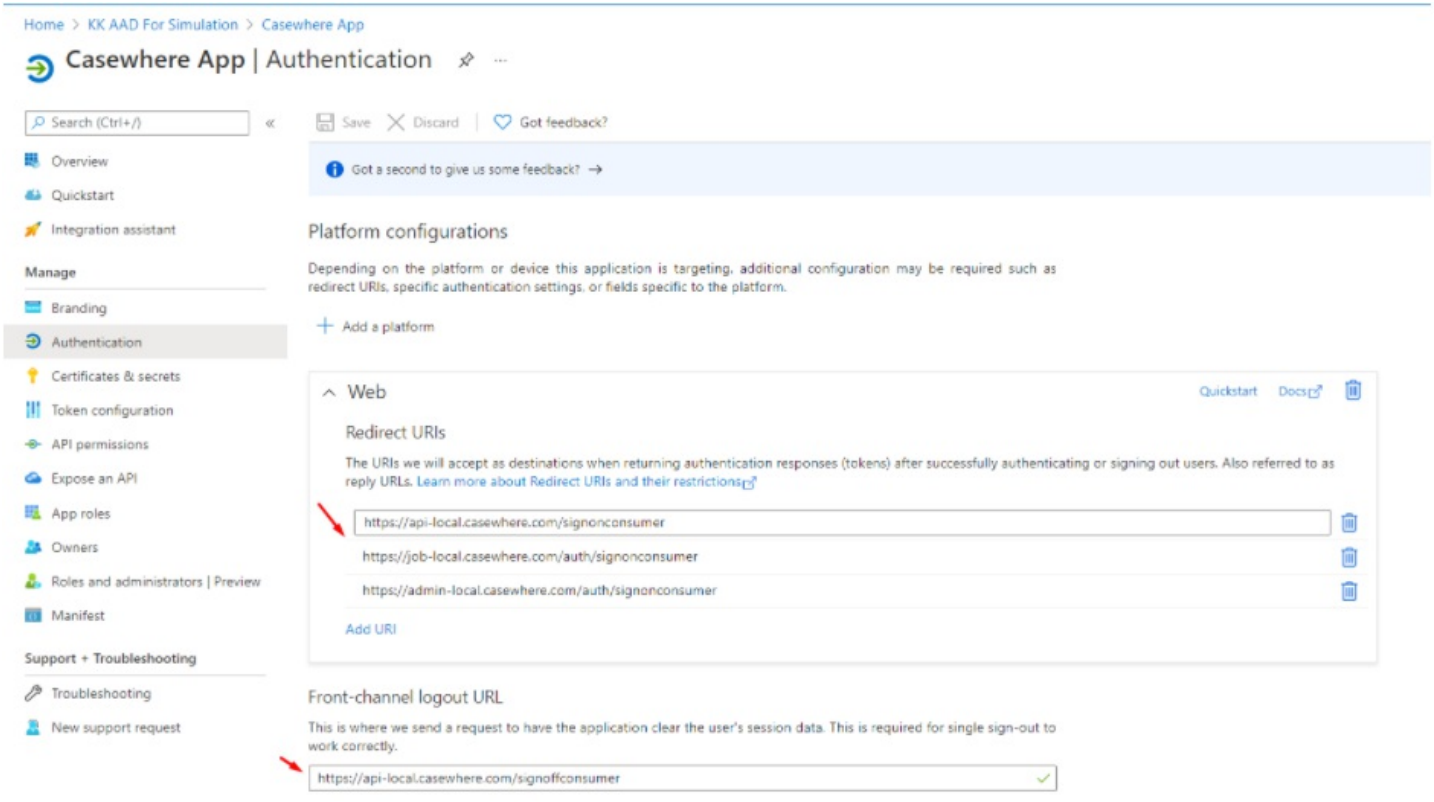
- Select the **Authentication** menu
- Configure **Redirect URIs** for three sites in the Casewhere instance. Take a look at the image below to see what it looks like
    - Admin site: https://[admin site domain]/auth/signonconsumer

      E.g. https://admin-local.casewhere.com/auth/signonconsumer
    - Job site: https://[job site domain]/auth/signonconsumer

      E.g. https://job-local.casewhere.com/auth/signonconsumer
    - Worker API site: https://[worker api site domain]/signonconsumer

      E.g. https://api-local.casewhere.com/signonconsumer
- Configure Front-channel logout URL and you must use the domain of Worker API
    - Worker API site: https://[worker api site domain]/signoffconsumer

      E.g. https://api-local.casewhere.com/signoffconsumer



Then, follow the following steps in the **Admin site** section and do it similarly to the new application.

- Create App roles in the application
- Enable user assignment
- Assign App roles to a user

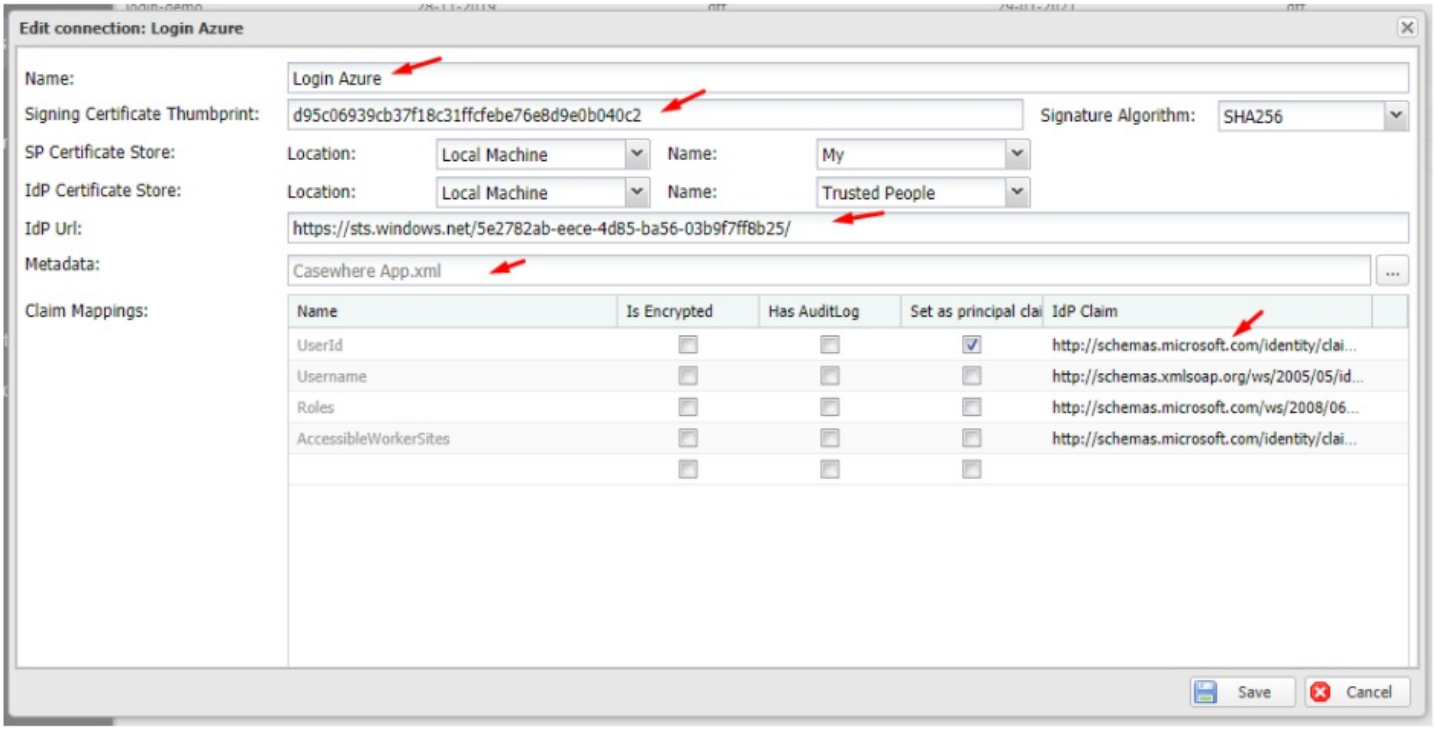
Configure the settings for three sites in Casewhere

Install public signing certificate

Similar to the Admin site.

Create an IdP connection

- Navigate to the **IdP Connections** menu
- Click the **Create** button, choose the **SAML 2.0** option
- In the **Name** field, enter the name like **Login Azure**
- In the **Signing Certificate Thumbprint** field, enter the thumbprint of the Casewhere signing certificate
- In the **IdPUrl** field, enter the link you get from the **entityID** attribute of the **EntityDescriptor** element when you open the metadata of the application. For example,
- Upload the metadata of the application into the **Metadata** field
- In the Claim Mapping section,
    - UserId: http://schemas.microsoft.com/identity/claims/objectidentifier
    - Username: http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name
    - Roles: http://schemas.microsoft.com/ws/2008/06/identity/claims/role
- Finally save it.



Next, you need to change the settings for the specific sites in Casewhere. Make sure you can access the folder that contains the Casewhere instance.

Configure the settings for the Admin site

Update metadata

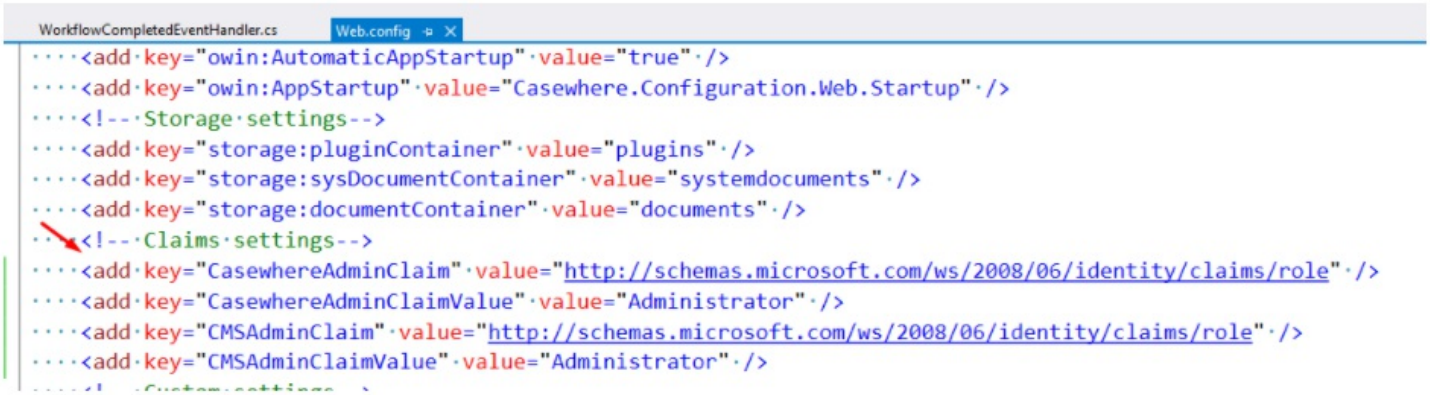
- Navigate to the **Casewhere.Configuration.Web** folder
- Create a **Metadata** folder, go into this folder and create an **AAD** folder
- Then copy the metadata of the application into the **AAD** folder

Change the settings in *Web.config*

Change the following settings in the appSettings element of the Web.config file

In **Claims settings** section

- CasewhereAdminClaim: Set it to *http://schemas.microsoft.com/ws/2008/06/identity/claims/role*
- CasewhereAdminClaimValue: Set it to **Administrator**
- CMSAdminClaim: Set it to *http://schemas.microsoft.com/ws/2008/06/identity/claims/role*
- CMSAdminClaimValue: Set it to **CMSAdmin**



In the **IdP Connection** section

- idpConnection:IdPUrl: Open the metadata file and get the value of the **entityID** attribute in the **EntityDescriptor** element



- idpConnection:IdPMetadataFolder: Enter the relative path **~/Metadata/AAD**
- idpConnection:EntityId: Put the application ID into it.
- idpConnection:LogoutResponseUri: Put the sign-off endpoint of the Worker API site. It looks like

  https://[worker api site domain]/signoffconsumer

  E.g. https://api-local.casewhere.com/signoffconsumer



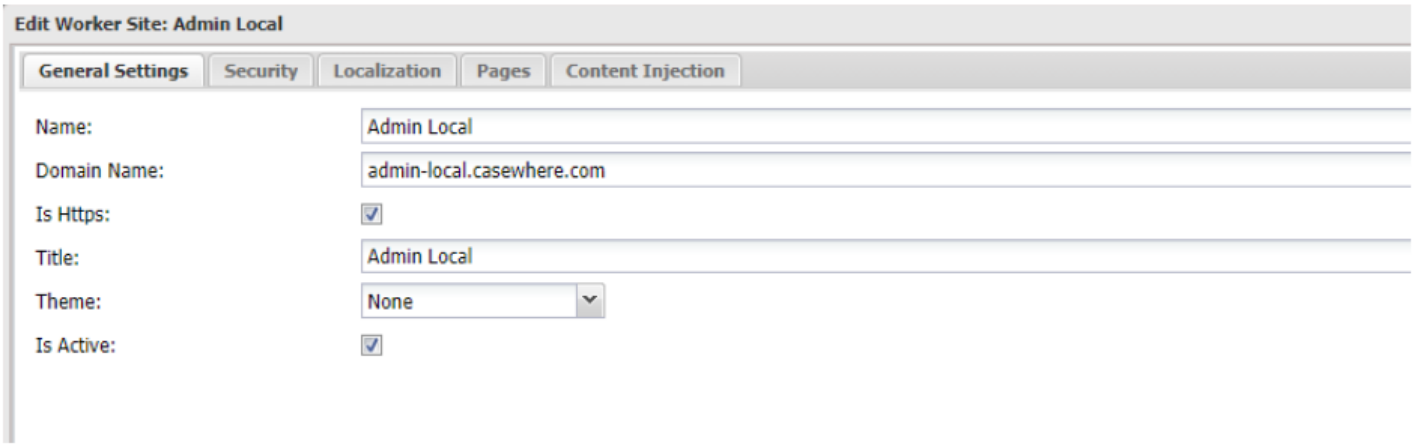
Create an internal worker site for the Admin site

This worker site is used to support the logout action.

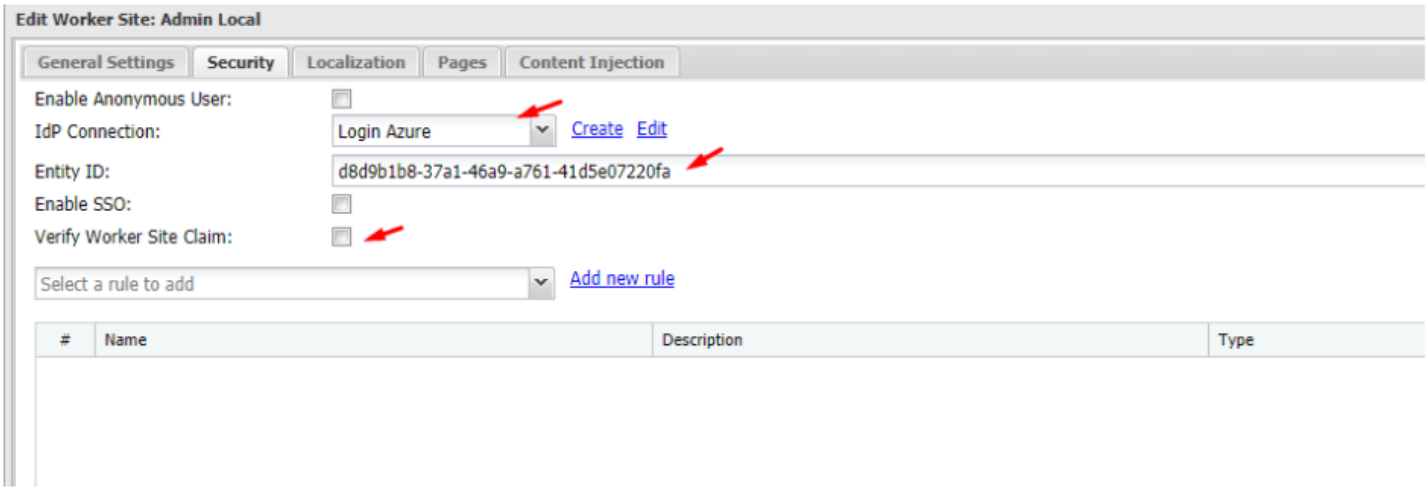
- Navigate to the **Worker Sites** menu, click the **Create** button

- In the **Name** field, enter a suitable name in the business context, like CW Admin site
- In the **Domain Name** field, enter the domain of the Admin site.

  E.g. admin-local.casewhere.com
- In the **Theme** field, select a theme or None
- Check **Is active** field



- Go to the **Security** tab, select the **Login Azure** option in the **IdP Connection** dropdown
- In the **Entity ID** field, enter the application ID
- Uncheck the **Verify Worker Site Claim** checkbox
- Click the **Save** button



Configure the settings for the Job site

Update metadata

- Navigate to the **Casewhere.Hangfire** folder
- Create a **Metadata** folder, go into this folder and create an **AAD** folder
- Then copy the metadata of the application into the **AAD** folder

Change the settings in Web.config

Change the following settings in the appSettings element of the Web.config file

In **Admin Roles** section

- CasewhereAdminClaim: Set its value to *http://schemas.microsoft.com/ws/2008/06/identity/claims/role*
- CasewhereAdminClaimValue: Set its value to **Administrator**



In the **IdP Connection** section

- idpConnection:IdPUrl: Open the metadata file and get it from the **entityID** attribute of the **EntityDescriptor** element
- idpConnection:IdPMetadataFolder: Enter the relative path **~/Metadata/AAD**
- idpConnection:EntityId: Put the application ID into it.
- idpConnection:LogoutResponseUri: Put the sign-off endpoint of the Worker API site. It looks like

https://[worker api site domain]/signoffconsumer

E.g. https://api-local.casewhere.com/signoffconsumer

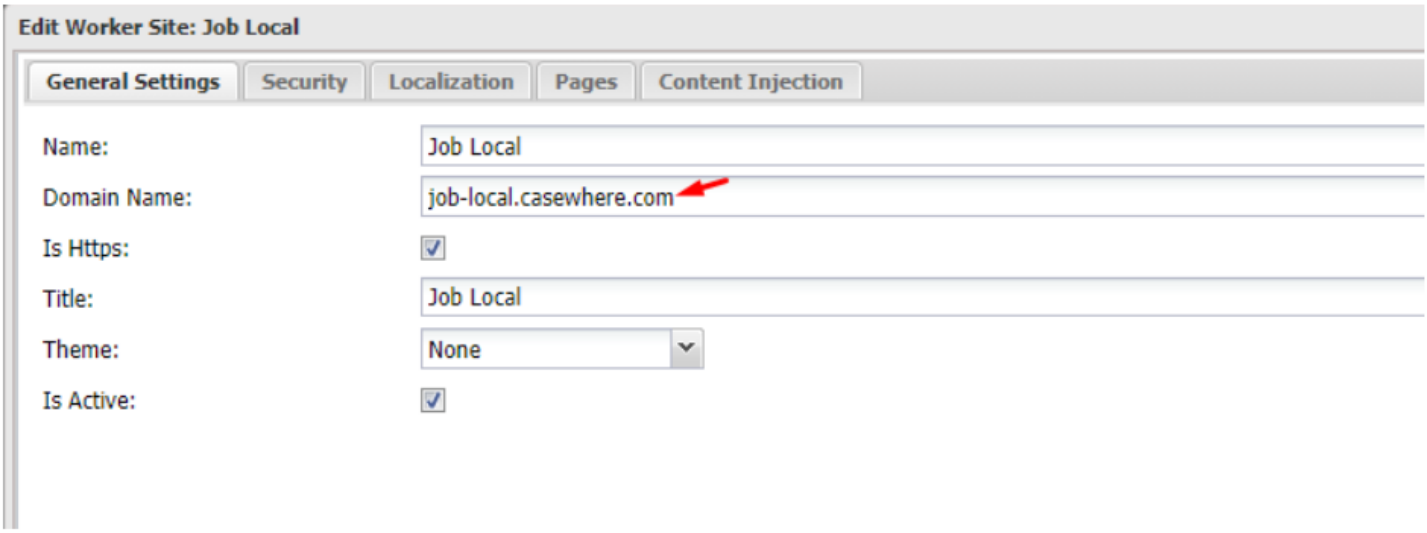


Create an internal worker site for the Job site

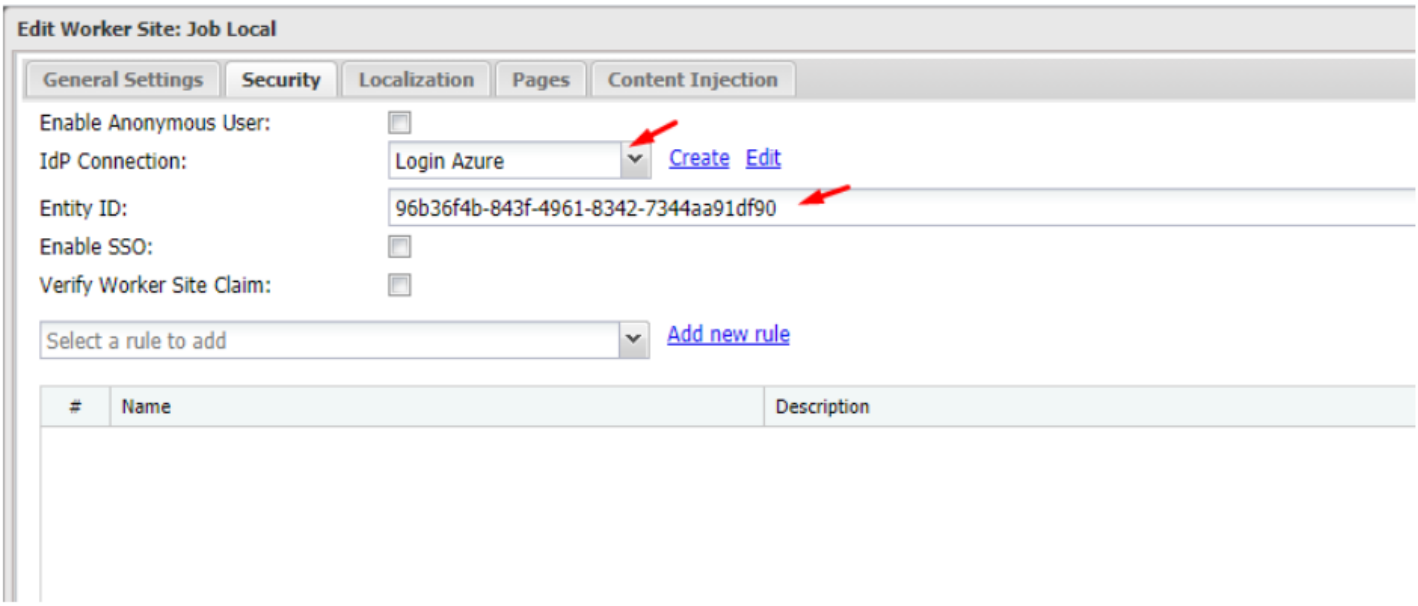
This worker site is used to support the logout action.

- Navigate to the **Worker Sites** menu, click the **Create** button
- In the **Name** field, enter a suitable name in the business context, like Job Local
- In the **Domain Name** field, enter the domain of the job site.

  E.g. job-local.casewhere.com
- In the **Theme** field, select a theme or None
- Check **Is active** field



- Go to the **Security** tab, select the **Login Azure** option in the **IdP Connection** dropdown
- In the **Entity ID** field, enter the application ID
- Uncheck the **Verify Worker Site Claim** checkbox
- Click the **Save** button

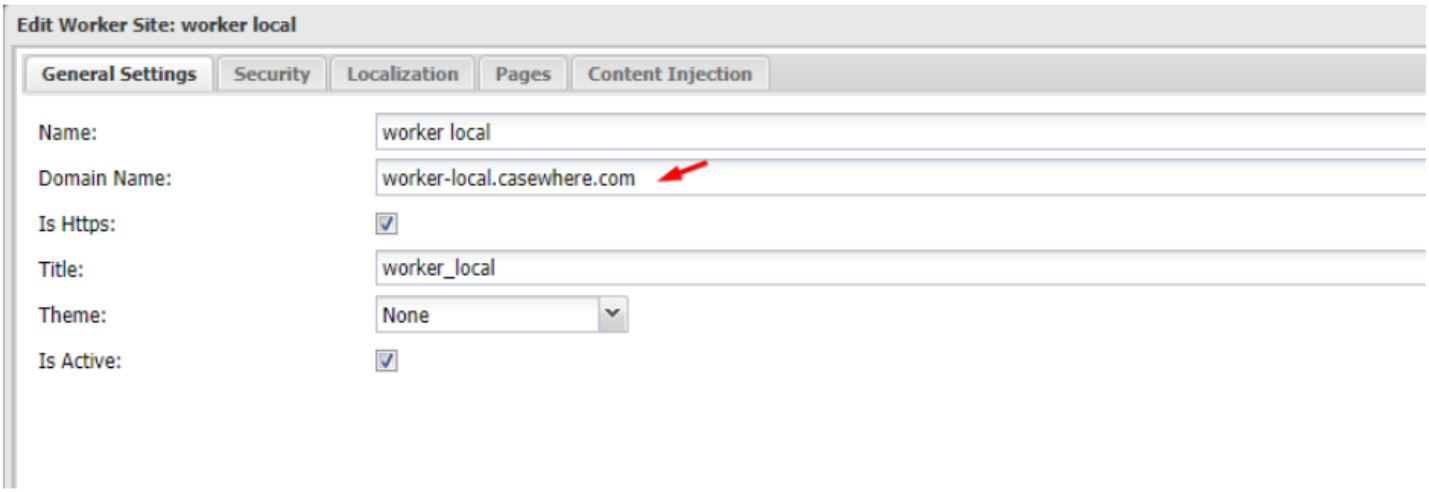


Configure the settings for Worker site

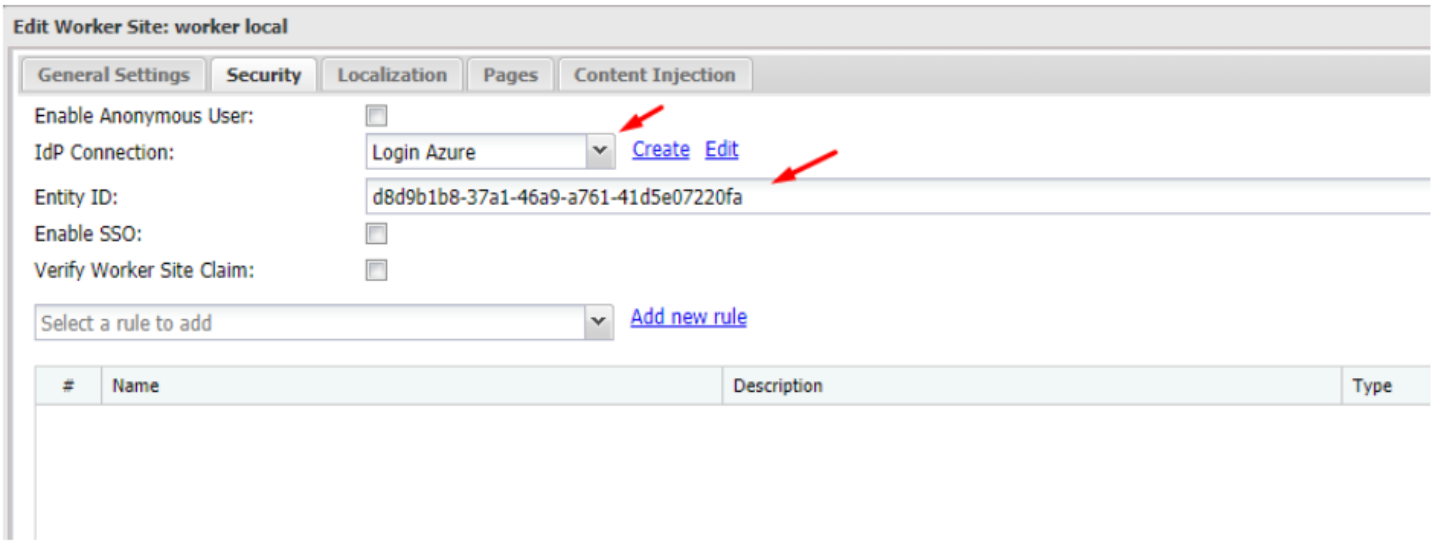
Create a public worker site

- Navigate to the **Worker Sites** menu, click the **Create** button
- In the **Name** field, enter a suitable name in the business context, like CPK Caseworker site
- In the **Domain Name** field, enter the domain of that site.

  E.g. worker-local.casewhere.com
- In the **Theme** field, select a theme or None
- Check **Is active** field



- Go to the **Security** tab, select the **Login Azure** option in the **IdP Connection** dropdown
- In the **Entity ID** field, enter the application ID
- Uncheck the **Verify Worker Site Claim** checkbox
- Click the **Save** button

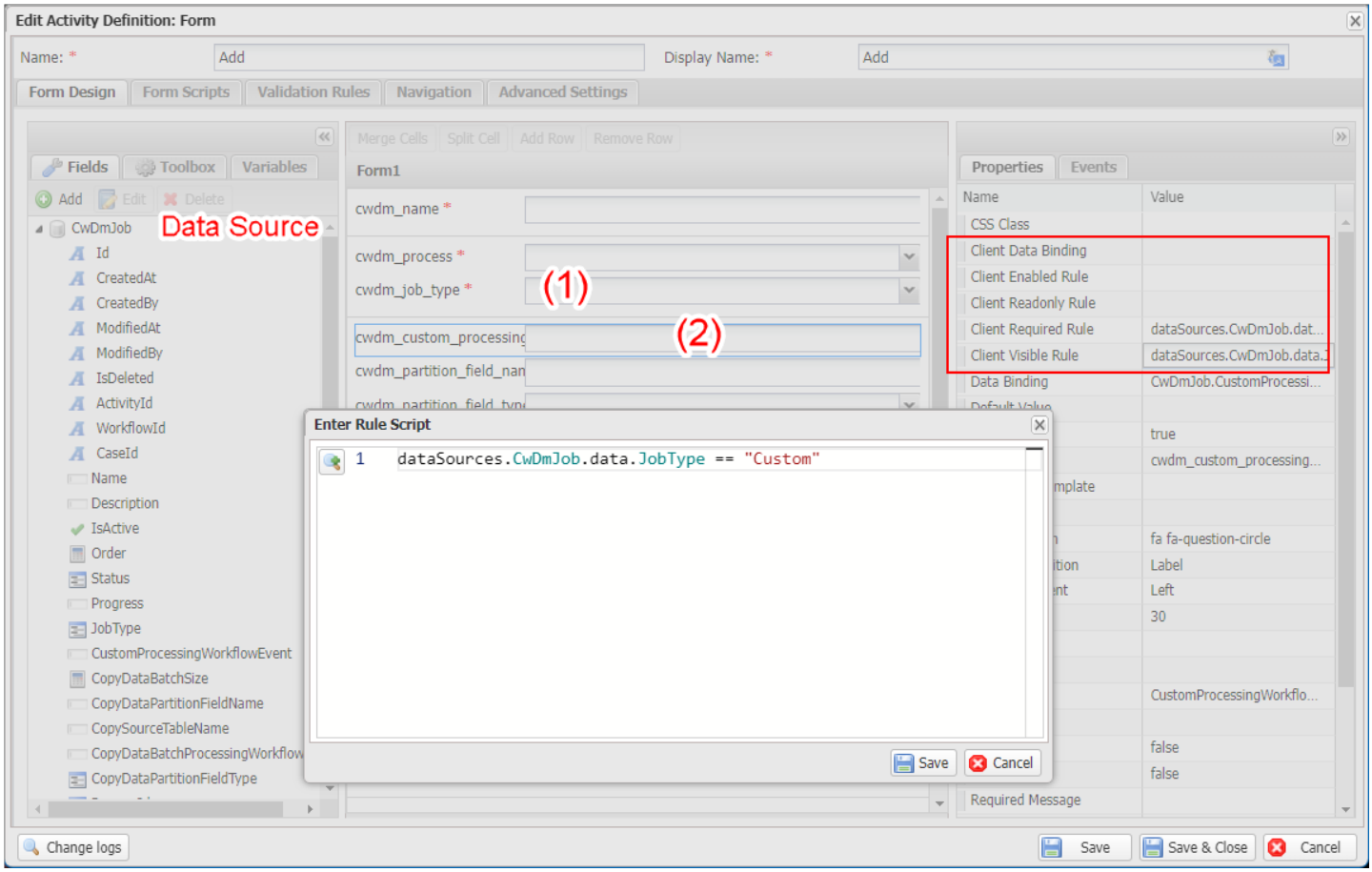


Form activity - best practices

Explore a set of best practices for working with Casewhere form activities. Learn and apply these practices to your specific use cases when they are appropriate.

Form client rules

These rules prove valuable in enhancing the responsiveness and user-friendliness of forms. Consider a scenario where you need to display or hide a component based on specific conditions. In this example, we will only show (2) when users select 'Custom' for (1).

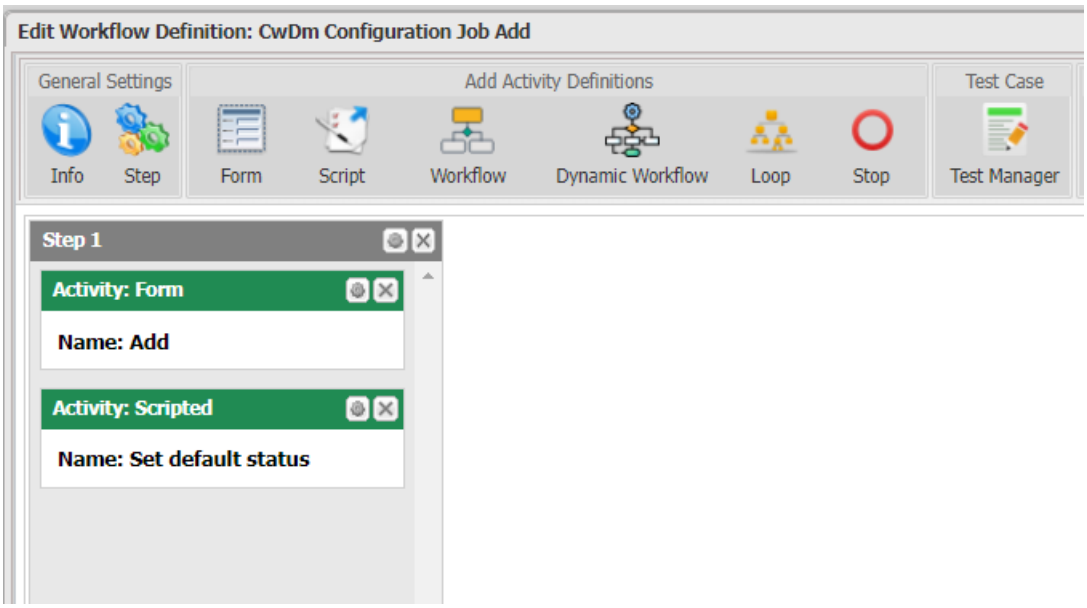


In this case, you can configure the **Client Visible Rule** and **Client Required Rule** as shown below:

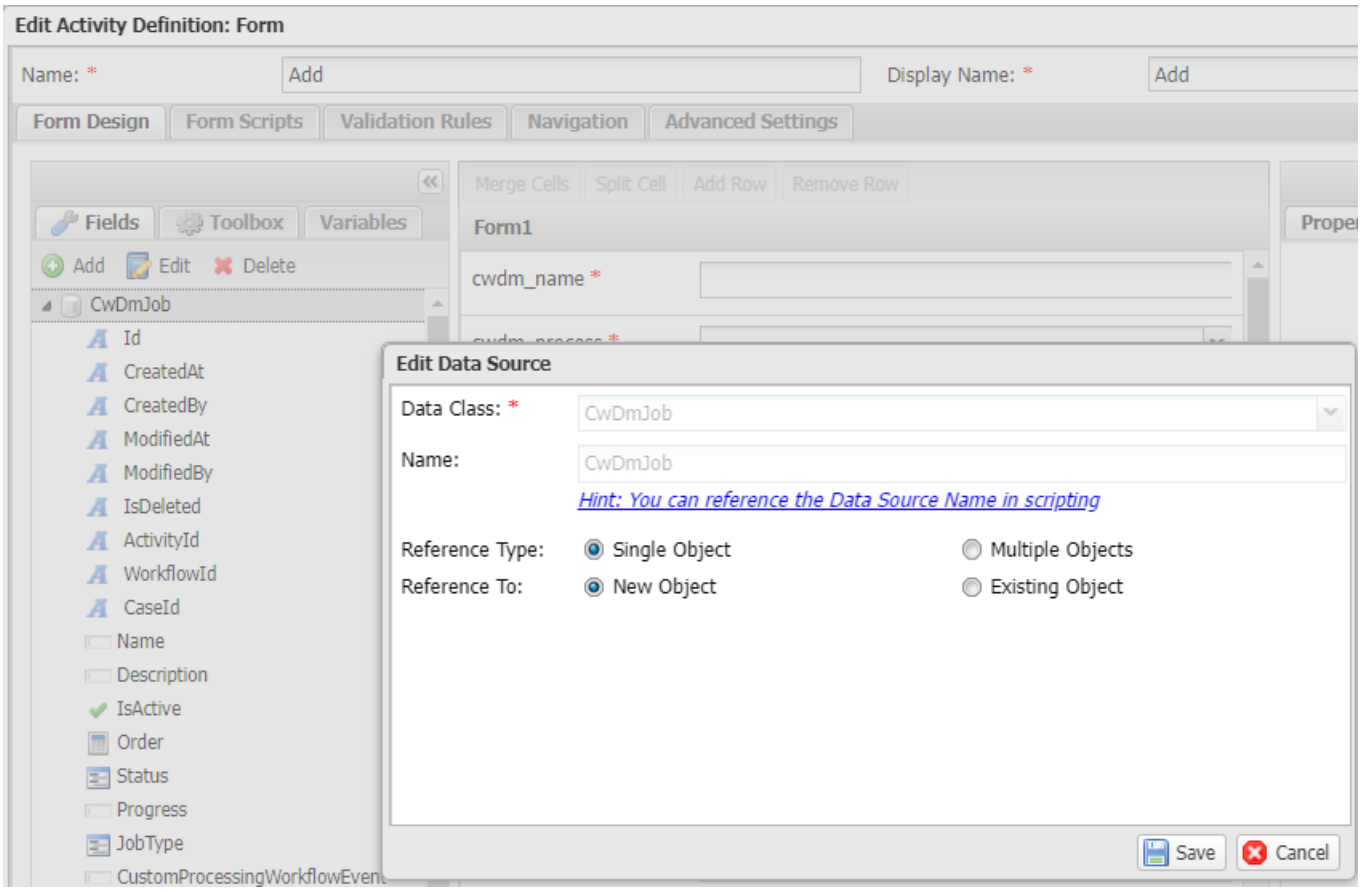
```
dataSources.CwDmJob.data.JobType == "Custom"
```

References form data sources in scripts

Imagine you're setting a workflow as shown below, and you want configure the **Scripted** activity to update the data object created in the preceding **Form** activity.



The form has a data source **CwDmJob** as shown below:



In the **Scripted** activity, you can use `ctx.Ref` to reference the data object:

```
var dataApi = ctx.Use<IDataApi>();
dataApi.Update(ctx.Ref("CwDmJob").Id, new { Status = "Ready", Progress = "" });
```

Translate text with parameters

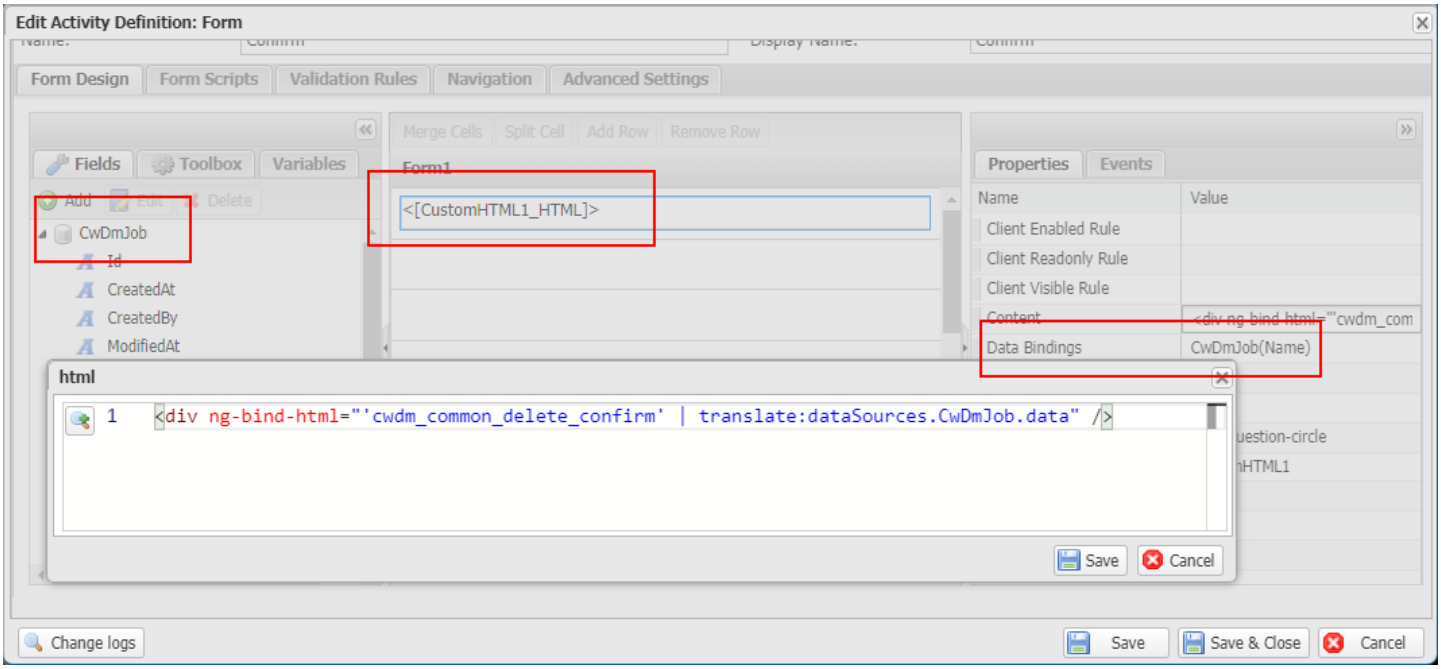
Imagine you need to set up a simple workflow illustrated below:



The first step involves creating a text resource key with parameter:

- Key: `cwdm_common_delete_confirm`
- Value: `Are you sure you want to delete <b>{{Name}}</b>?`

Next, in the form activity, add a new **CustomHTML** component, bind it to the data source and then update the content




```
<div ng-bind-html="'cwdm_common_delete_confirm' | translate:dataSources.CwDmJob.data" />
```

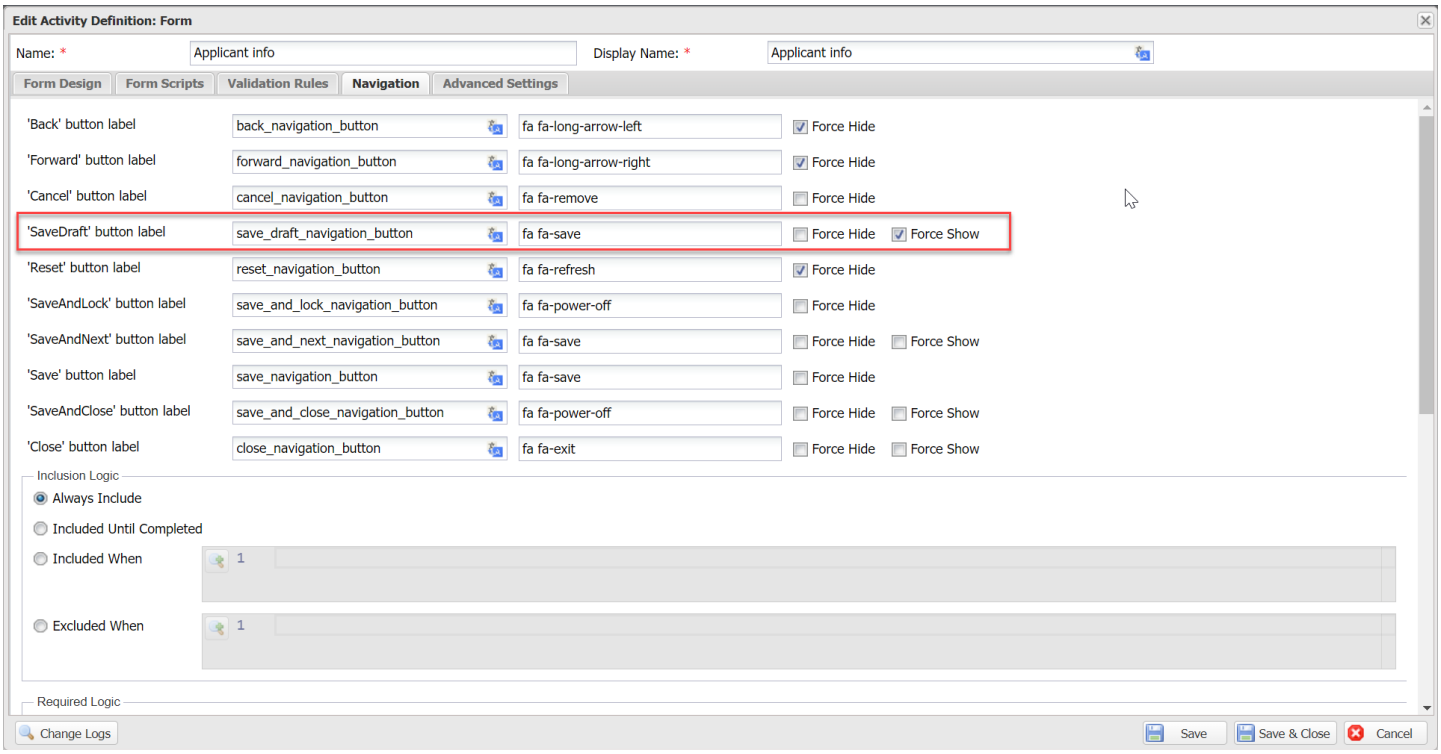
Please note that you can pass any object to the translate filter; it's not limited to a form data source as long as the object has the data needed for translation. For example:

```
<div ng-bind-html="'cwdm_common_delete_confirm' | translate:{Name: dataSources.CwDmJob.data.Name}" />
```

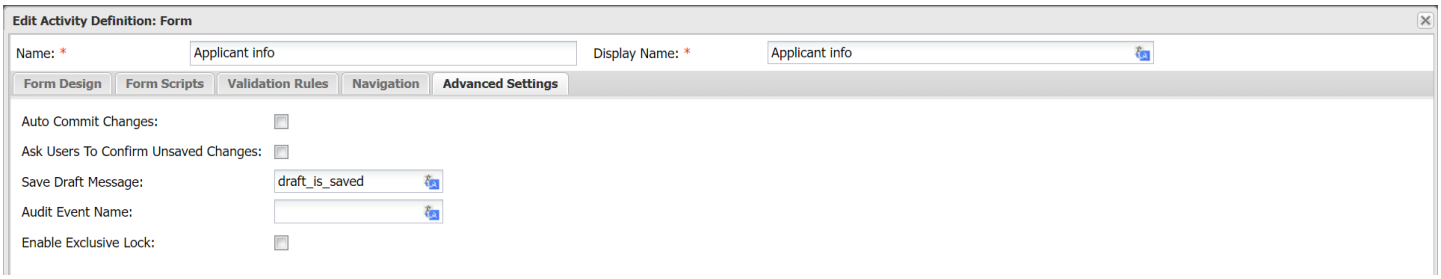
Enable save draft

For complex forms, many solutions support end users in saving a draft so that they can later go back and continue to edit it.

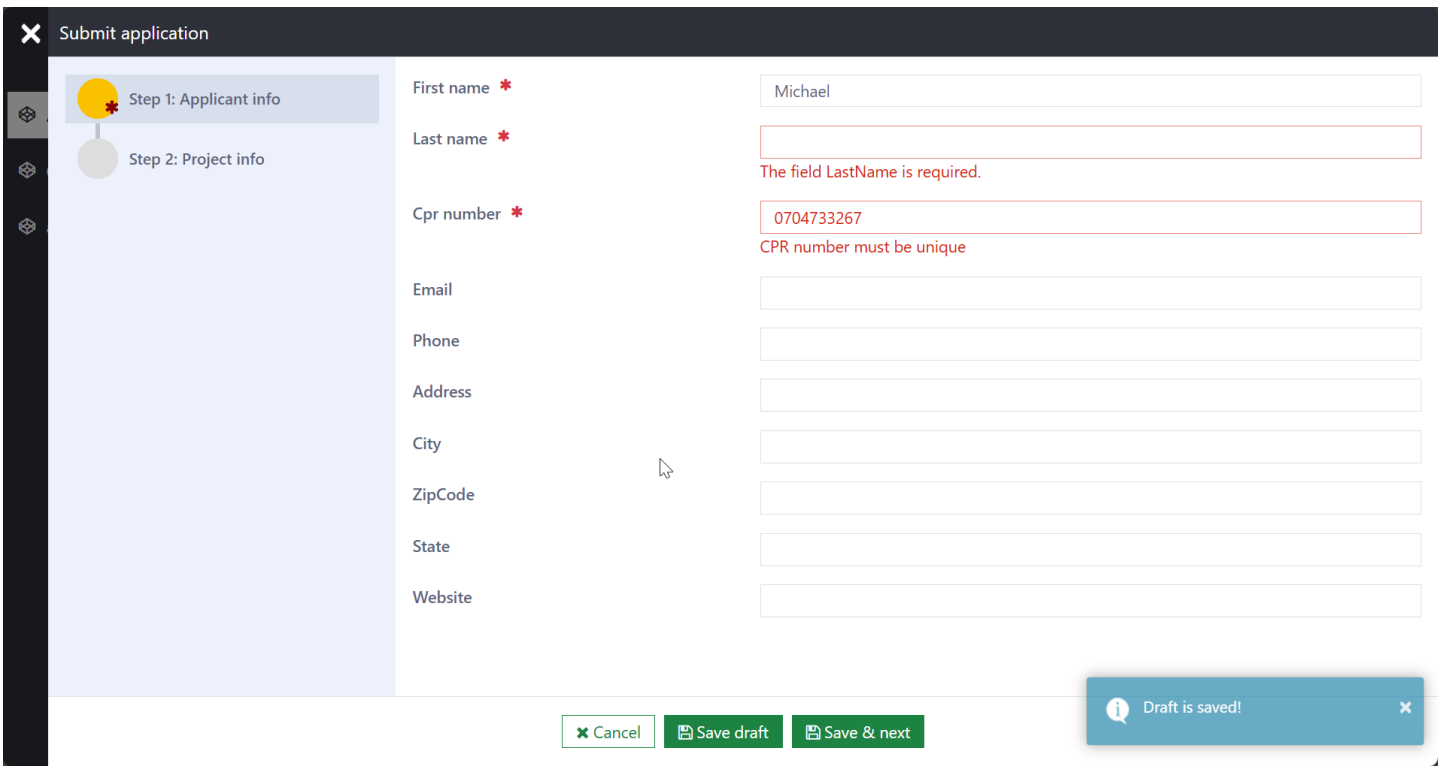
By default, Casewhere doesn't display the **Save draft** button, but you can enable it in the form activity's **Navigation** settings. To do so, locate **SaveDraft** and select the **Force Show** option.



It is also recommended to configure the **Save Draft Message** to notify users that their draft has been saved successfully.



On the worker site, the form with a save draft button will appear as follows:



When users click **Save draft**, Casewhere will bypass validation to save the data, but any validation errors will still be displayed, ensuring users are aware of what needs to be completed. It's also important to note that when draft saving is enabled, Casewhere

will automatically set the `CaseState` of data objects to `Created`, ensuring that end users can access and edit data until it is ready to be submitted.

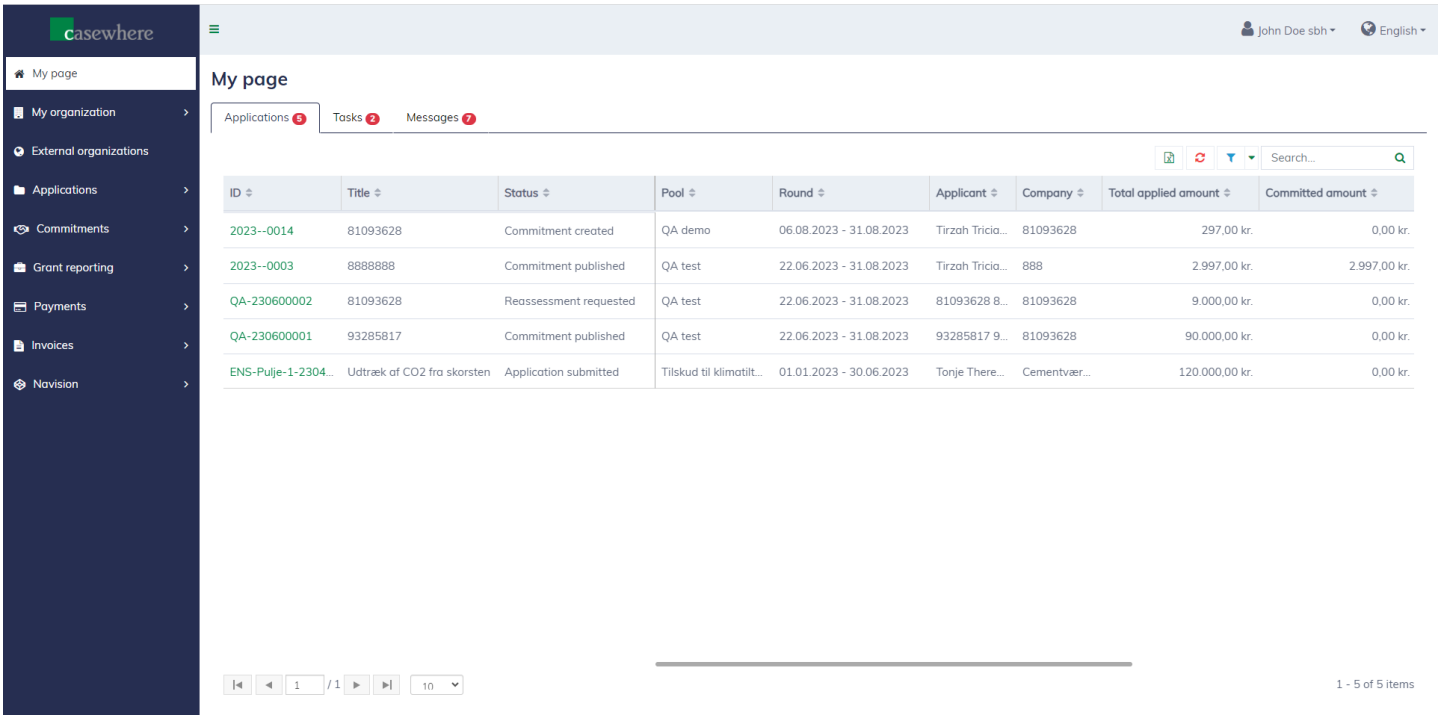
Grid widget - best practices

Explore a set of best practices for working with Casewhere grid widgets. Learn and apply these practices to your specific use cases when they are appropriate.

Standard grid

A typical grid widget should be configured with the following settings:

- **Search**: Enable by default
- **Export**: Enable by default
- **Sort**: Enable for all visible columns
  - You can choose to disable sorting for columns like `Description` that are not optimized for sorting.
- **Filter**: Enable for all visible columns
  - You can choose to disable filtering for columns like `Description` that are not optimized for filtering.
- **Look and feel**
  - **Column width:**
    - Must provide enough space for important columns
    - Use a fixed size for columns where the type/size is certain, such as datetime or boolean.
  - **Format**:
    - Number/money: Right-aligned; utilize the workersite's localization setting.
    - Datetime: Use the workersite's localization setting.
  - If the grid has too many columns, consider "freezing" some important columns.



Administration grid

Administration grid is used for manage configuration, master data, etc. A typical administration grid widget should be configured with the following settings:

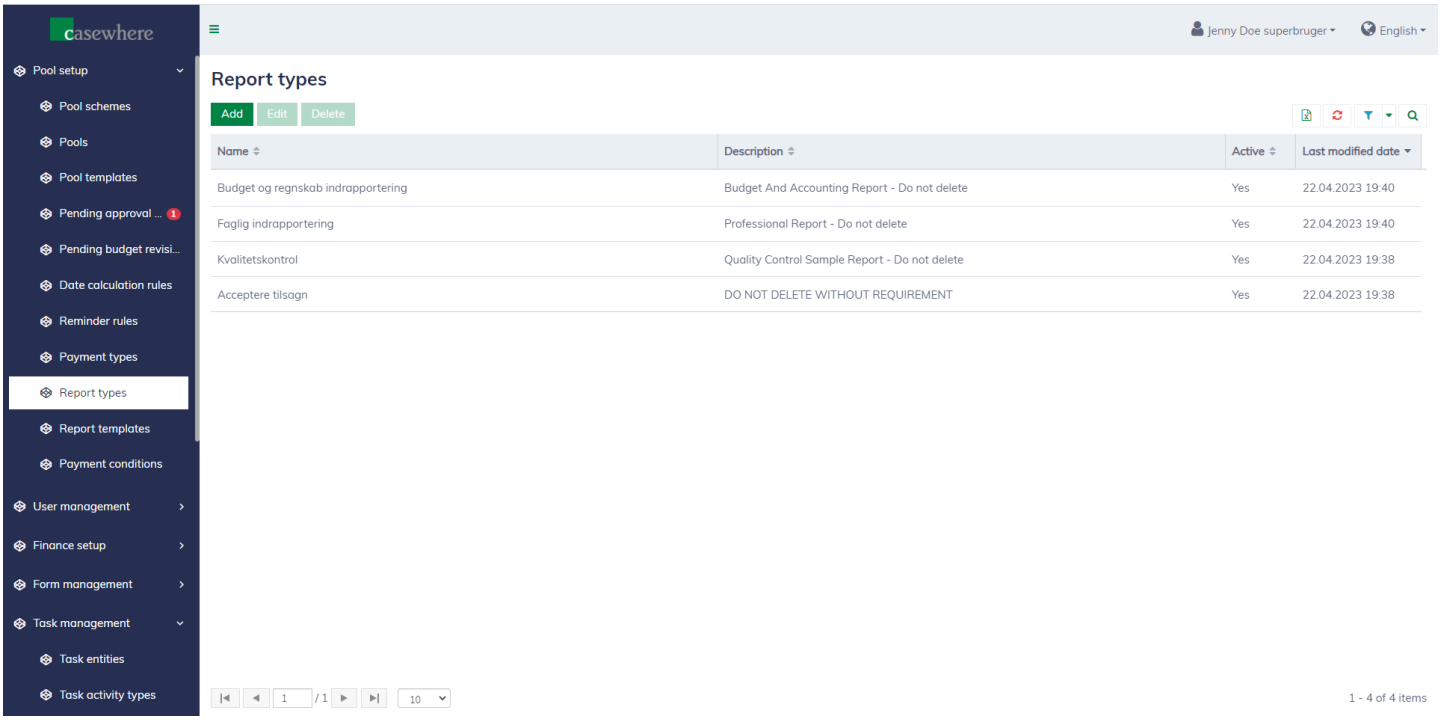
Action name:

- Utilize verbs like "Add," "Edit," and "Delete," along with common text resource keys.

- Avoid duplication: Eliminate unnecessary context specification to avoid duplication. For instance, use "Add user" in the context of the list of users.

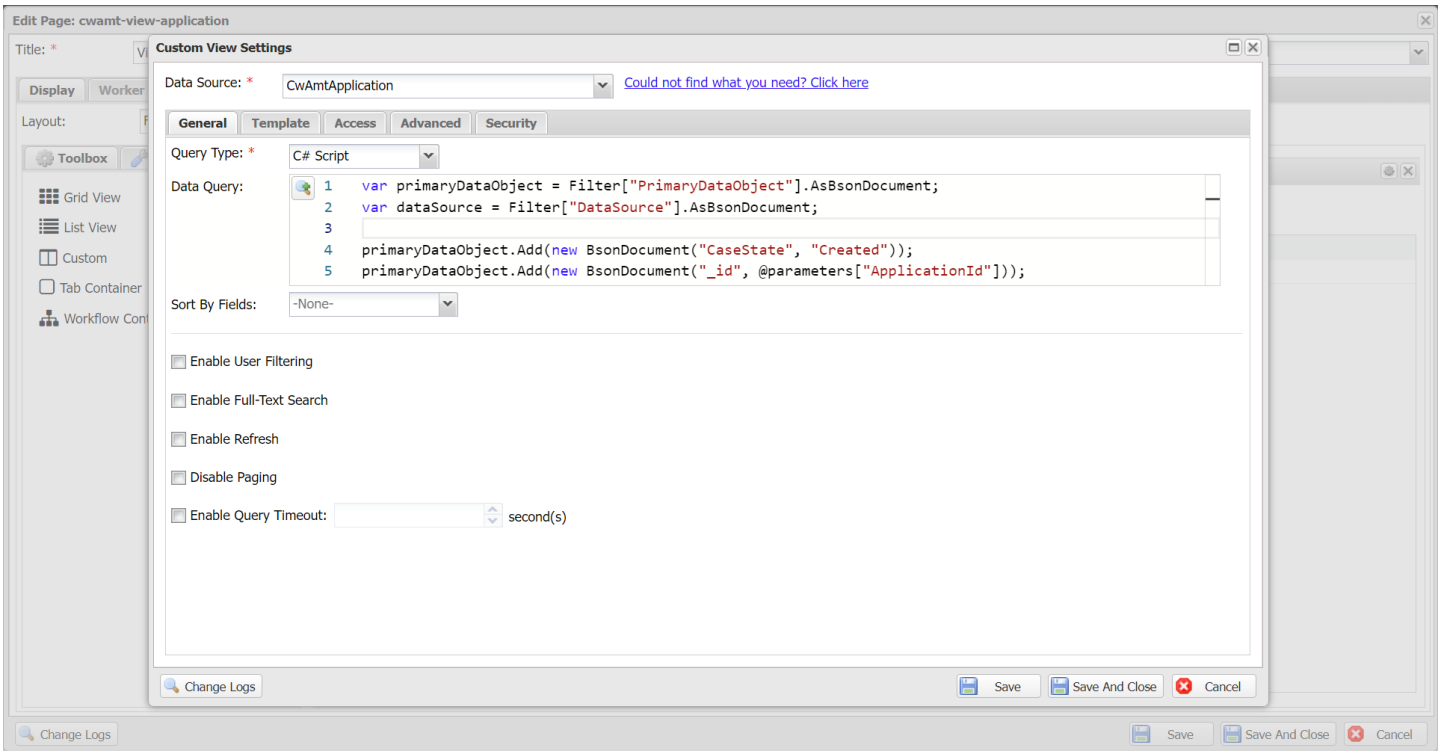
Administration columns:

- "Active" column: Represent as yes/no and utilize common text resource keys.
- "Last modified date" column: Display the time and adhere to the workersite's localization setting.
- Ensure "Active" and "Last modified date" are the last two columns.



Widget data queries: best practices

Implementing efficient widget data queries can be challenging. This article highlights common real-world scenarios and the best approaches to address them.



Return empty data

In scenarios when certain conditions are not met, you can use the keyword `RETURN_EMPTY` to efficiently return an empty response instead of making a roundtrip to the database or throwing exceptions.

```
if (@parameters.Has("ApplicationId"))
    primaryDataObject.Add(new BsonDocument("_id", @parameters["ApplicationId"]));
else
    primaryDataObject.Add(new BsonDocument("RETURN_EMPTY", ""));
```

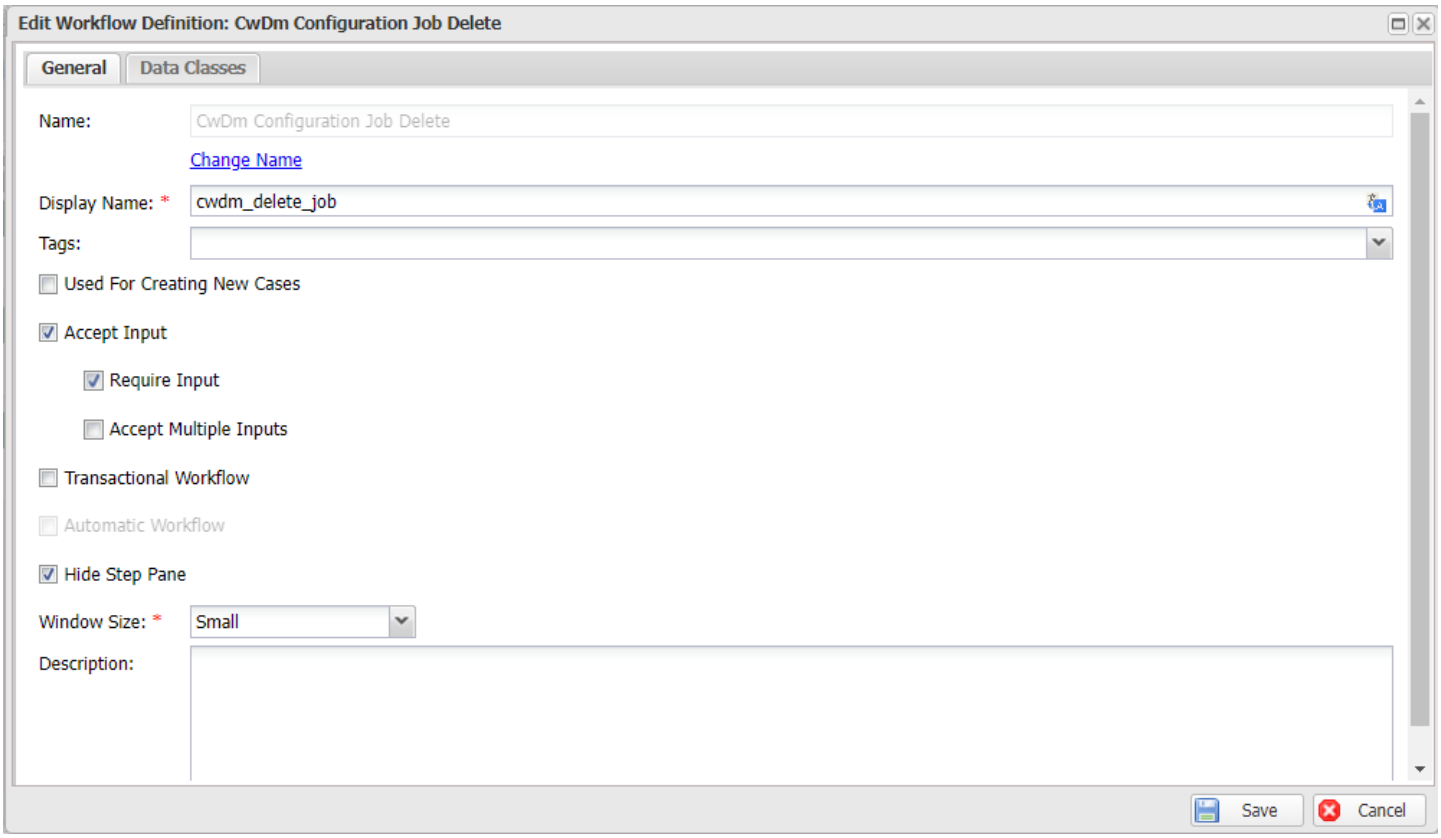
Workflow delete data - best practices

A typical workflow setup for deleting data.

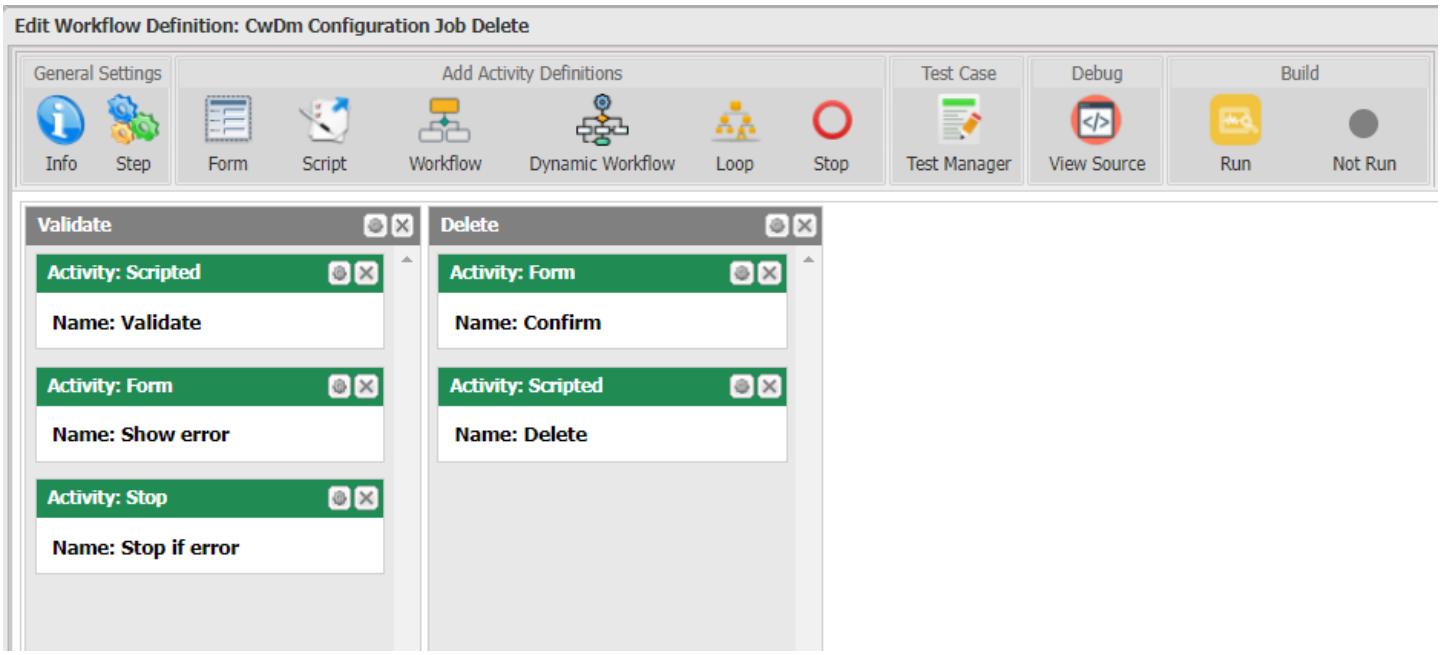
Workflow setup

The workflow setup should be:

- **Accept Input:** Yes
  - **Required Input:** Yes
- **Hide Step Pane:** Yes
  - **Window Size**: Small



The activities should be configured as shown below:



Scripted: Validate data

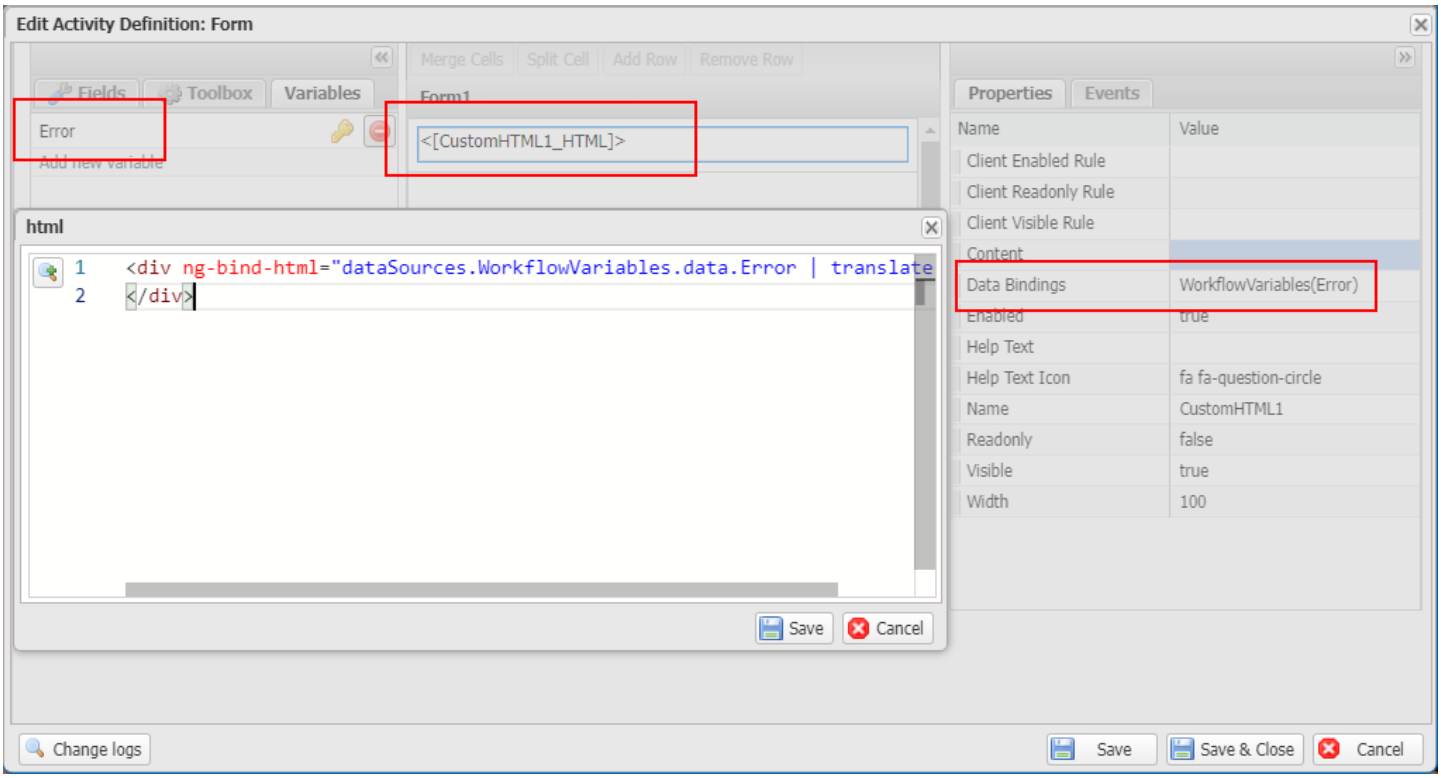
The workflow must validate whether the data is deletable. Many systems have constraints for deleting data, such as it must not be possible to delete a setting that is currently in use or an active user.

The activity should record validation errors in a workflow variable. In many cases, the error should be represented by a text resource key so that it can be displayed on the UI.

```
// Example: the data must be deactivated before deleting
if (ctx.Input.Get<bool>("IsActive"))
{
    // Set the error message to a workflow variable "Error"
    ctx.Set("Error", "cwdm_cannot_delete_active_job");
}
```

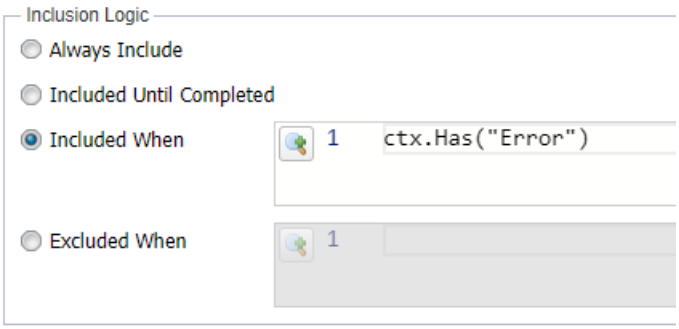
Form: Show error

Declare the variable to be used in the form, and then bind it to a **CustomHTML** component. Make sure you remember to apply the `translate` filter.



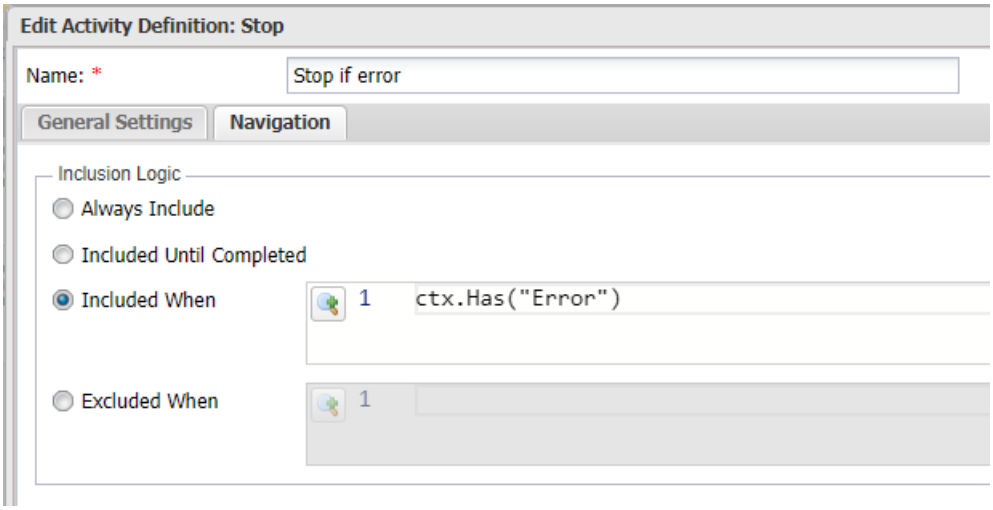

```
<div ng-bind-html="dataSources.WorkflowVariables.data.Error | translate"></div>
```

You must configure the **Inclusion Logic** so that the activity only executes when there is any error.



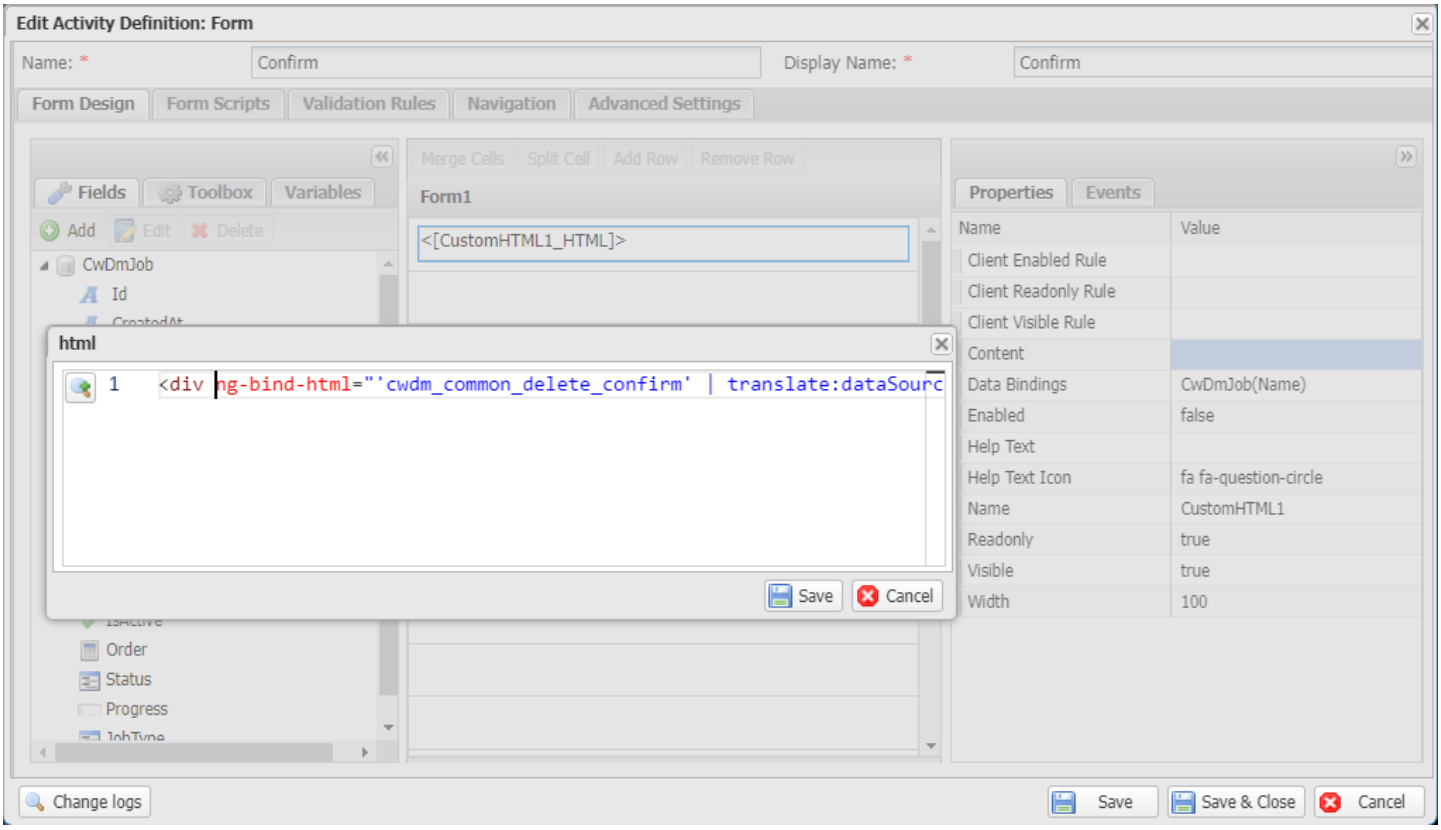
Stop: Stop if error

Use the **Stop** activity to stop the workflow. You must configure the **Inclusion Logic** so that the activity only executes when there is any error.



Form: Confirm

A form for confirming user decision. This is recommended for critical actions like deleting data.



Remember to apply translation for the message:

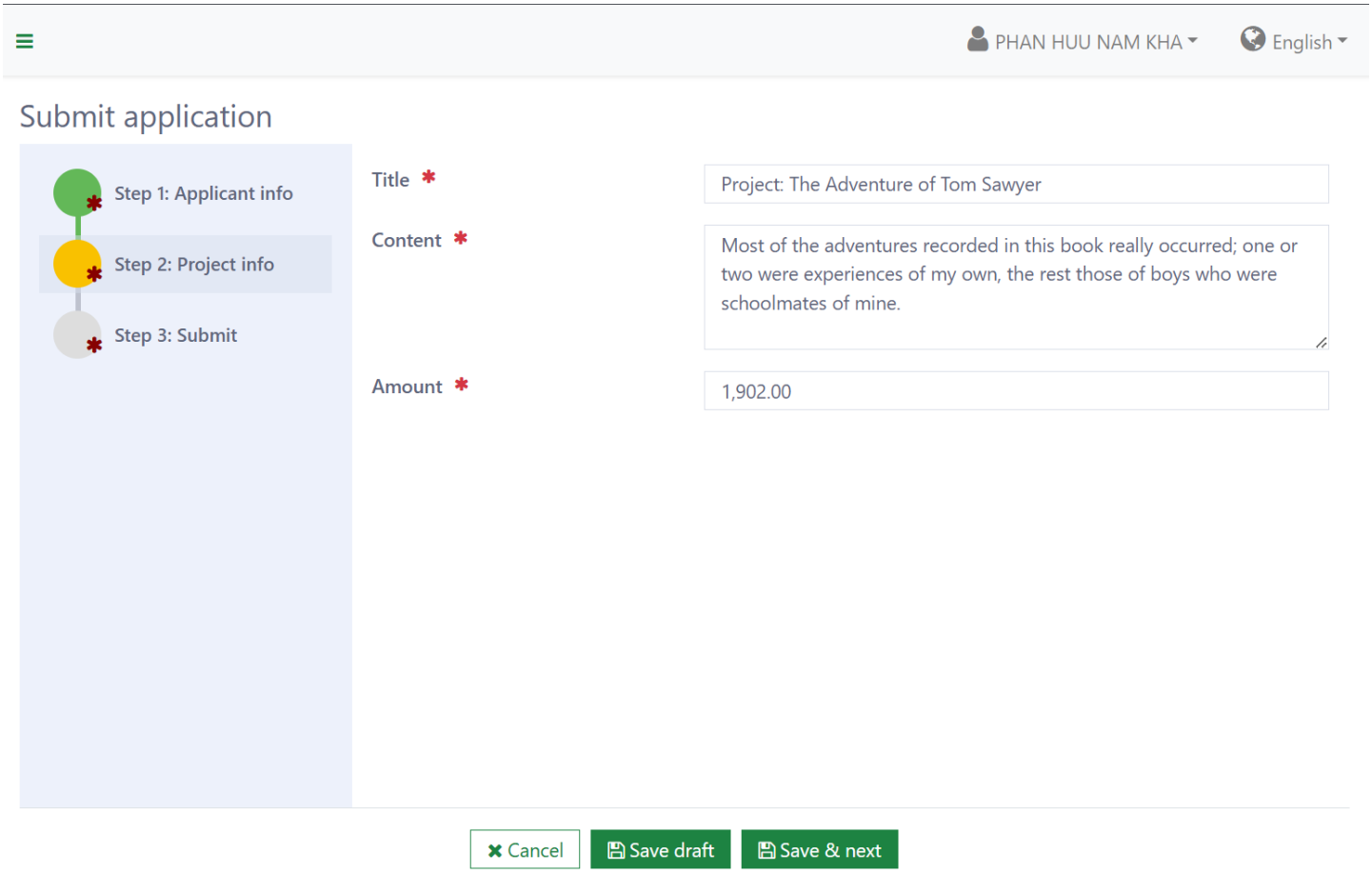
```
<div ng-bind-html="'cwdm_common_delete_confirm' | translate:{Name: dataSources.CwDmJob.data.Name}" />
```

Scripted: Delete

```
var dataApi = ctx.Use<IDataApi>();
dataApi.Delete(ctx.Input.Id);
```

Multiple-step workflows

The article aims to provide guidelines for setting up a long workflow where users must complete multiple steps before submitting their work. It should also allow users to save drafts along the way. Below is an example of what such a workflow might look like.



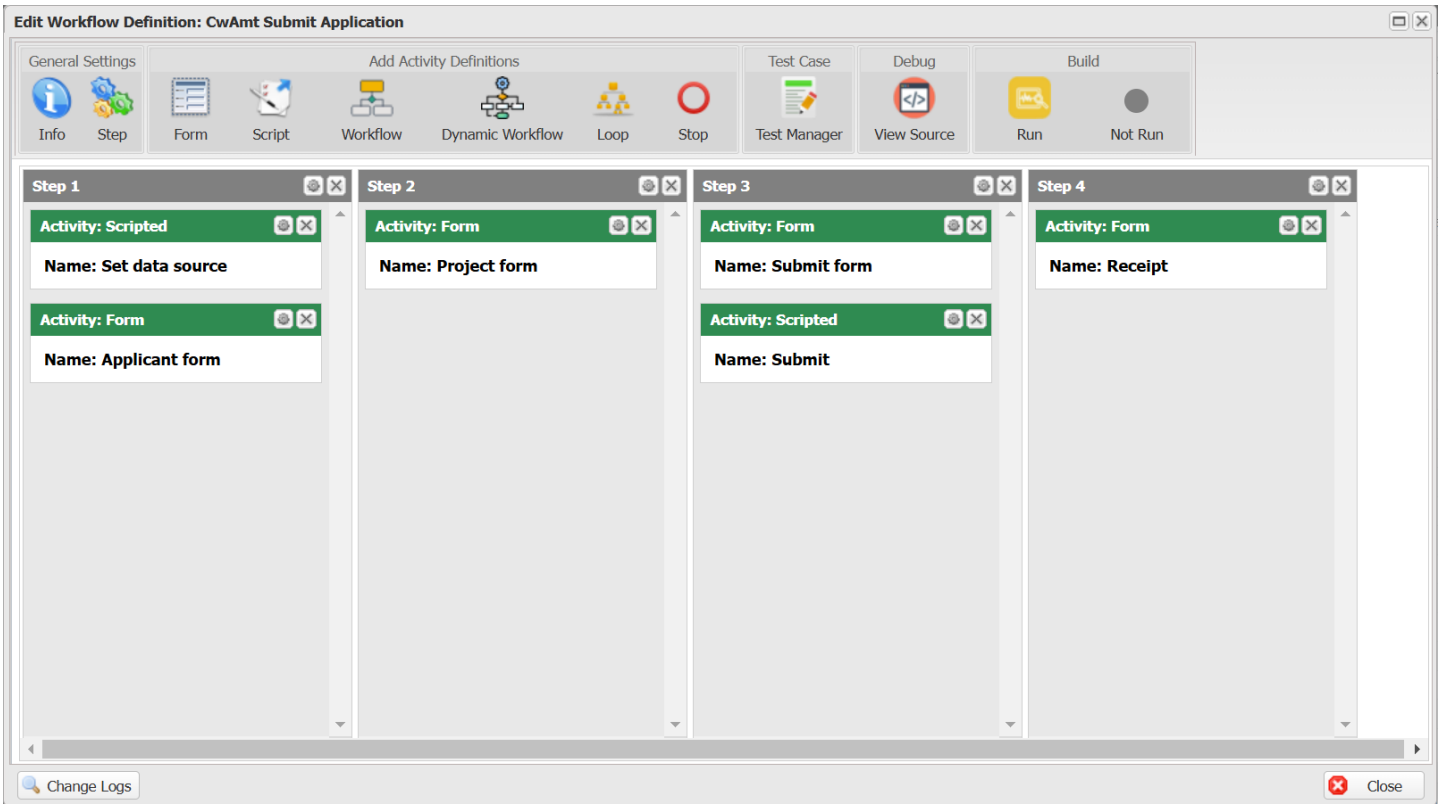
In this article you will learn:

- Enable draft saving
- Reuse workflows for both creating and editing cases
- Configure complex workflow navigation

Workflow definition for case editing

Since it's a long workflow with multiple steps and a lot of logic, we should aim to reuse as much code and configuration as possible to save cost for development and maintenance.

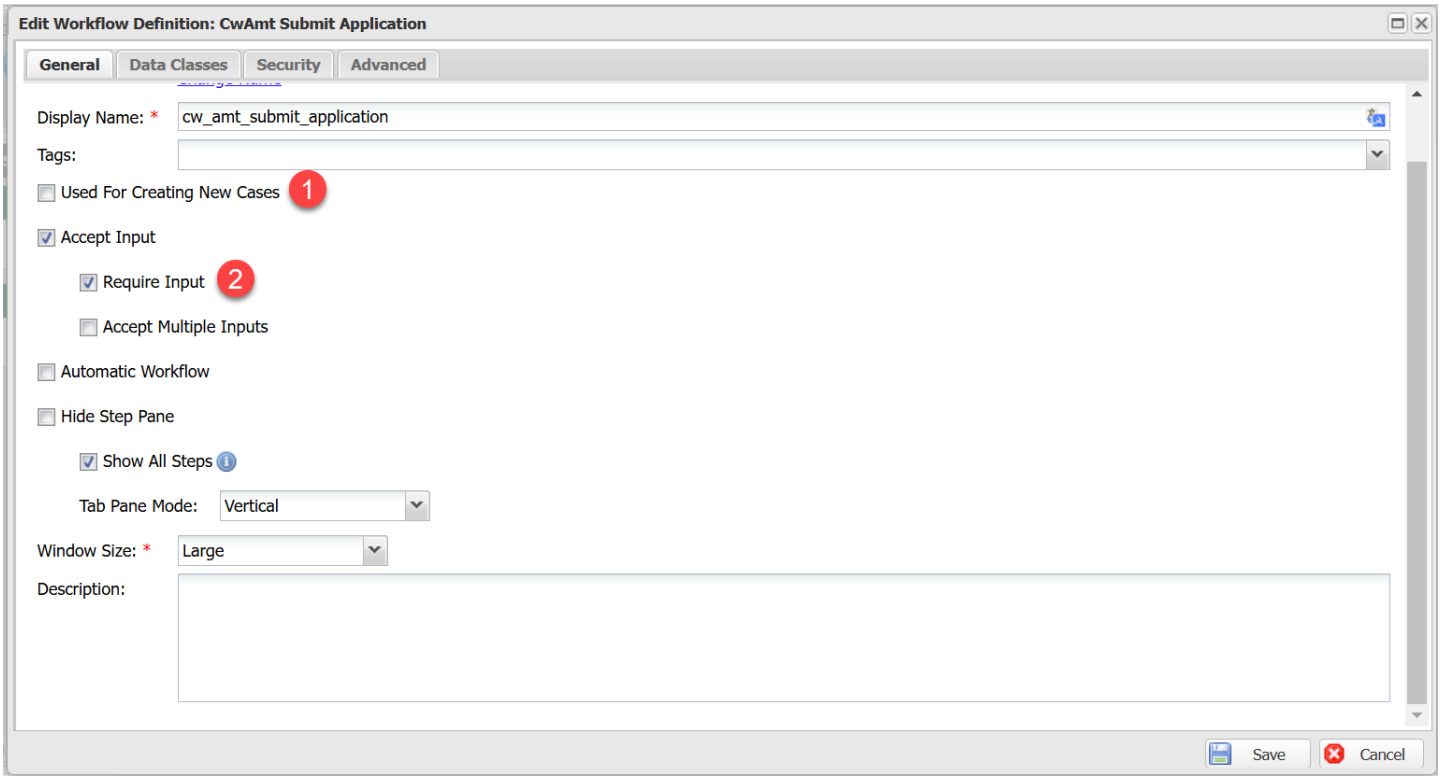
The first step is to create a workflow definition that encapsulates the majority of the logic. The workflow outline might look like this:



It is beyond the scope of the article to over every detail of the workflow setup; only the important steps are highlighted below.

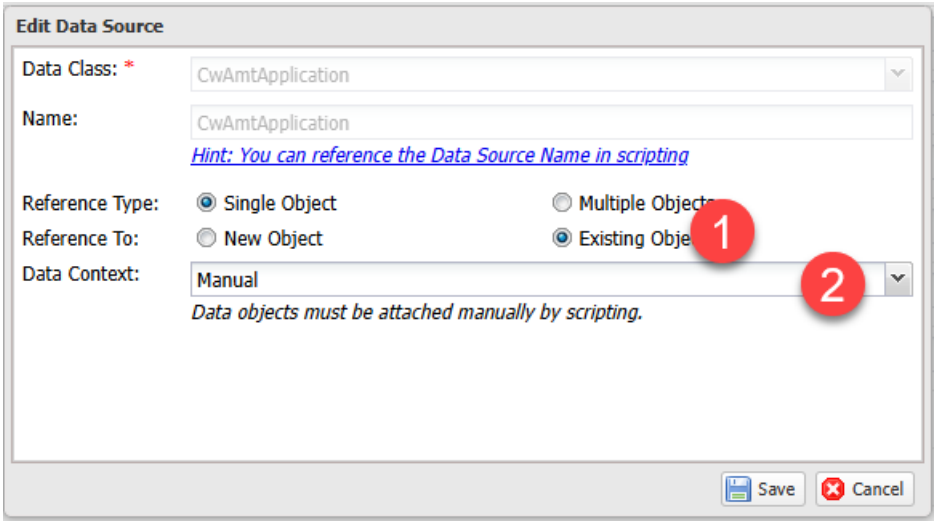
Step 1: Ensure that the workflow type is set to 'Case Editing' and that it requires user input.

1. **Used For Creating New Cases**: Off
2. **Require Input**: On

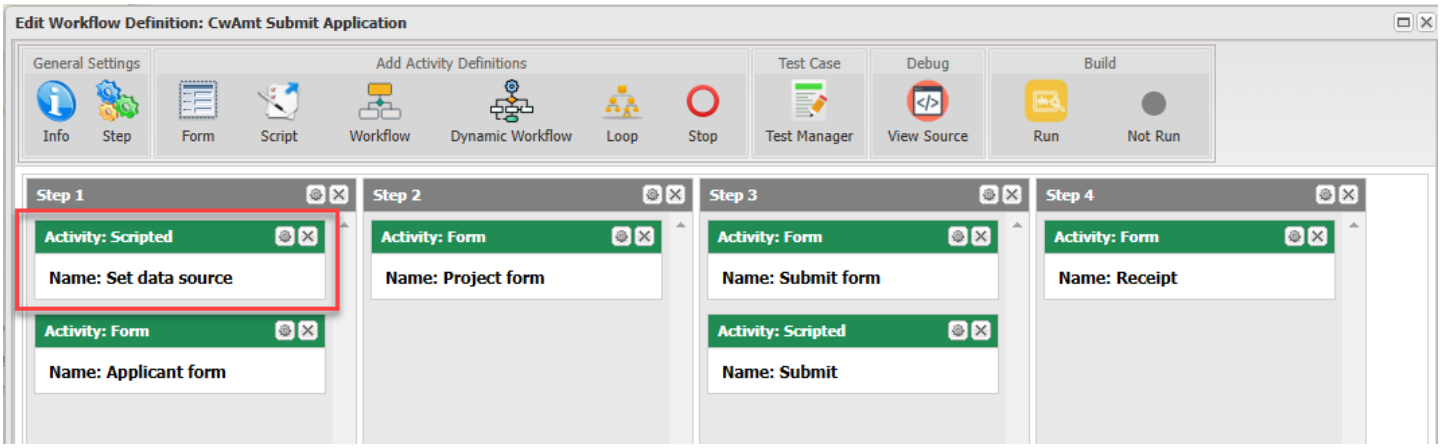


Step 2: Ensure that the form's data source accommodates both object creation and editing, and supports attaching objects at runtime.

- **Reference To**: Set to **Existing Object**
- **Data Context**: Set to **Manual**



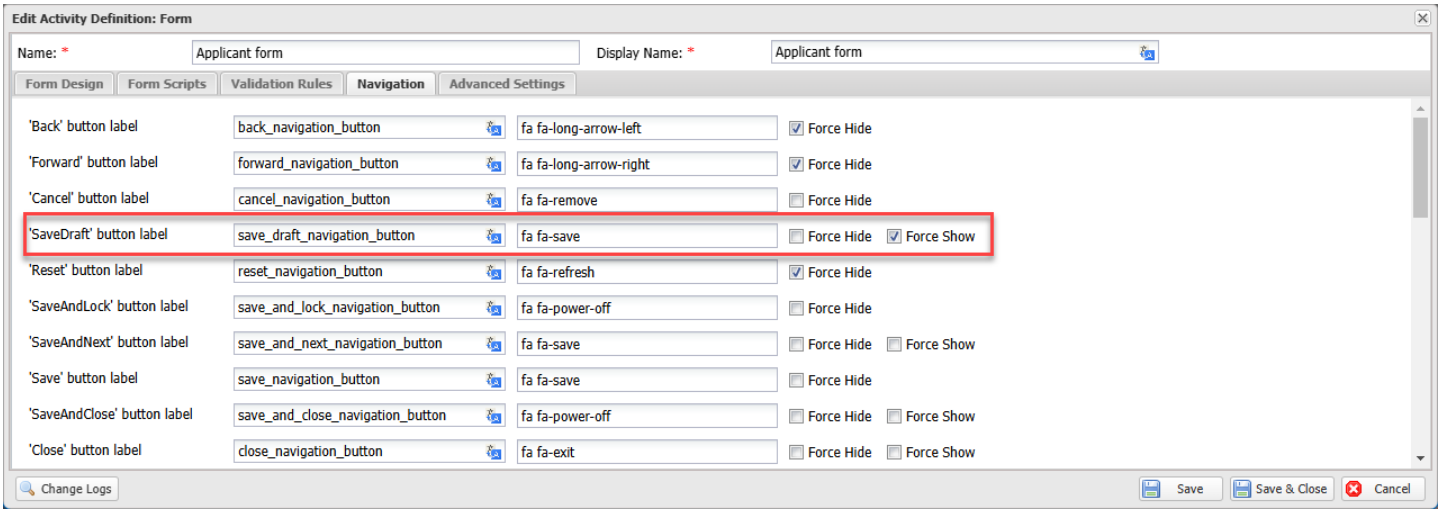
Step 3: Bind the form's data source to the workflow input, if available, to ensure the workflow loads the input object for editing. If no input is set, the workflow will create a new object based on the user's submitted data.

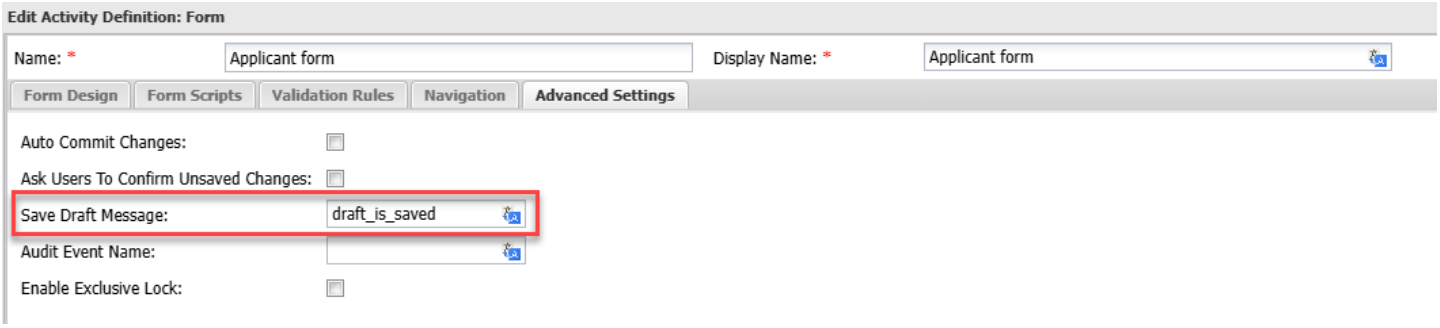



```
if (ctx.Input != null)
{
    ctx.SetDataSource("CwAmtApplication", ctx.Input.Id);
}
```

Step 4: Enable the **"Save Draft"** option in all intermediate forms

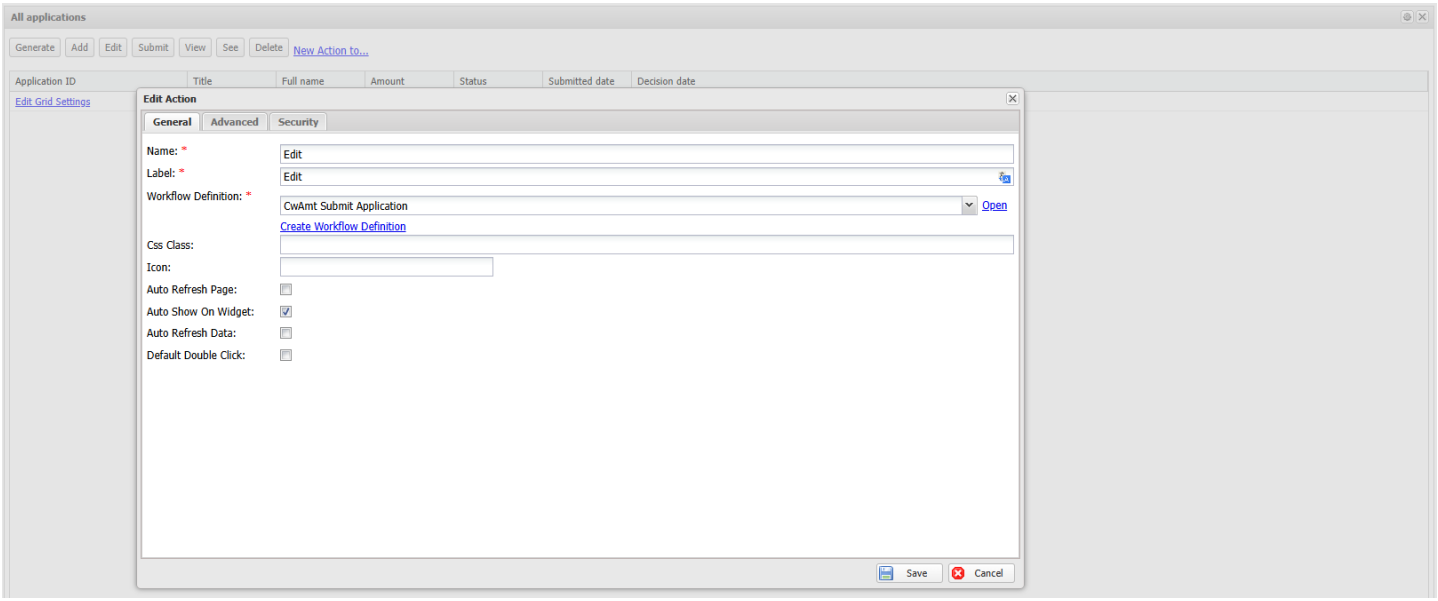
- **SaveDraft**: Set to **Force Show**
- **Save Draft Message**: A text resource key indicating that the draft has been saved successfully.





Follow the guideline here to learn more about form activity and draft saving.

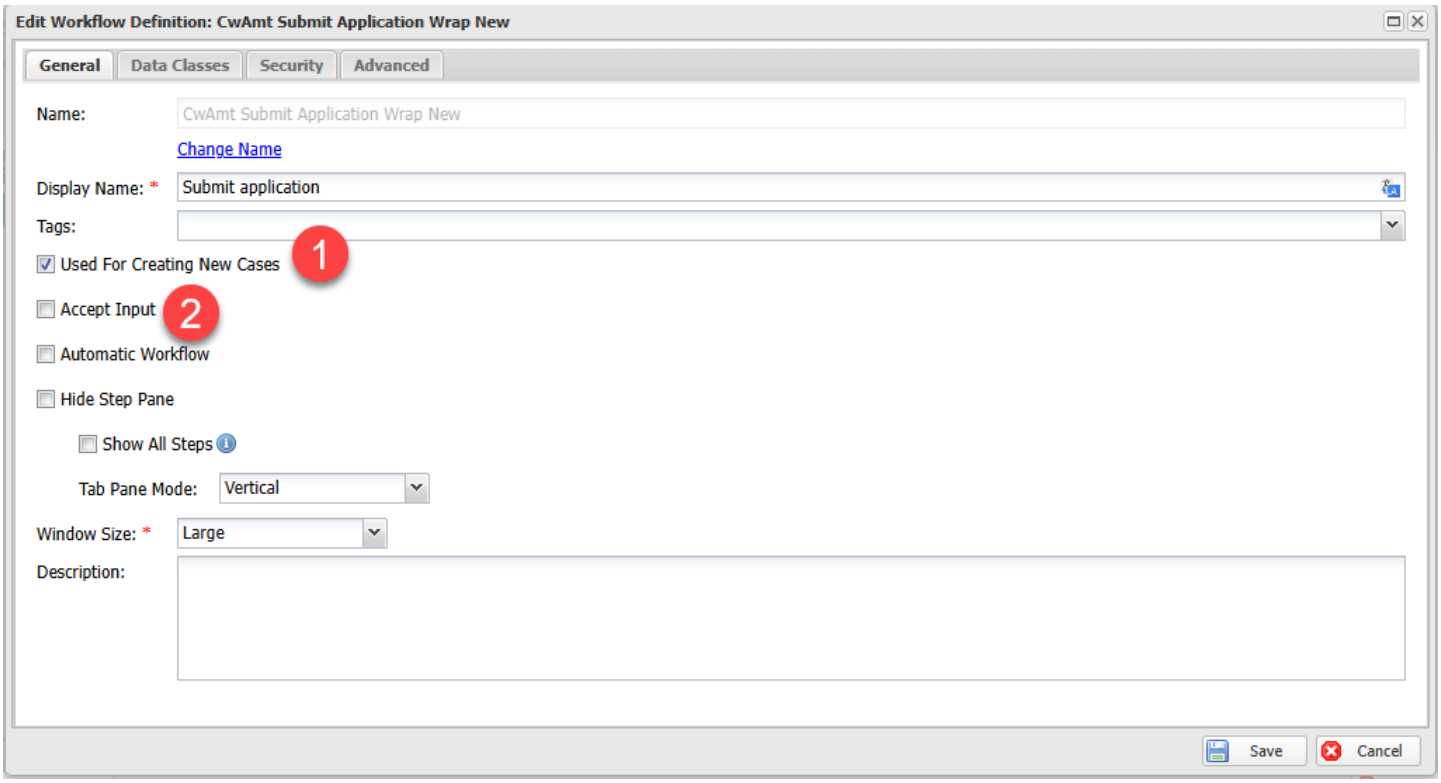
After these steps, you now can use the workflow for data editing. The screenshot below illustrates how to use the create-case workflow in a widget action.



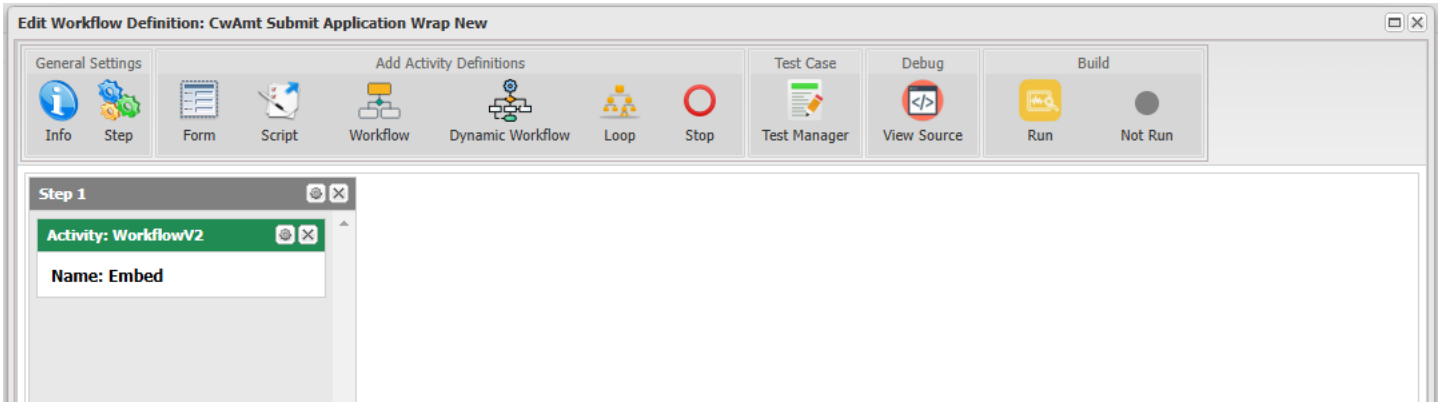
Workflow definition for case creation

We'll attempt to reuse the case-editing workflow definition created in the previous step. However, first, let's create a new workflow definition for creating cases, as shown below:

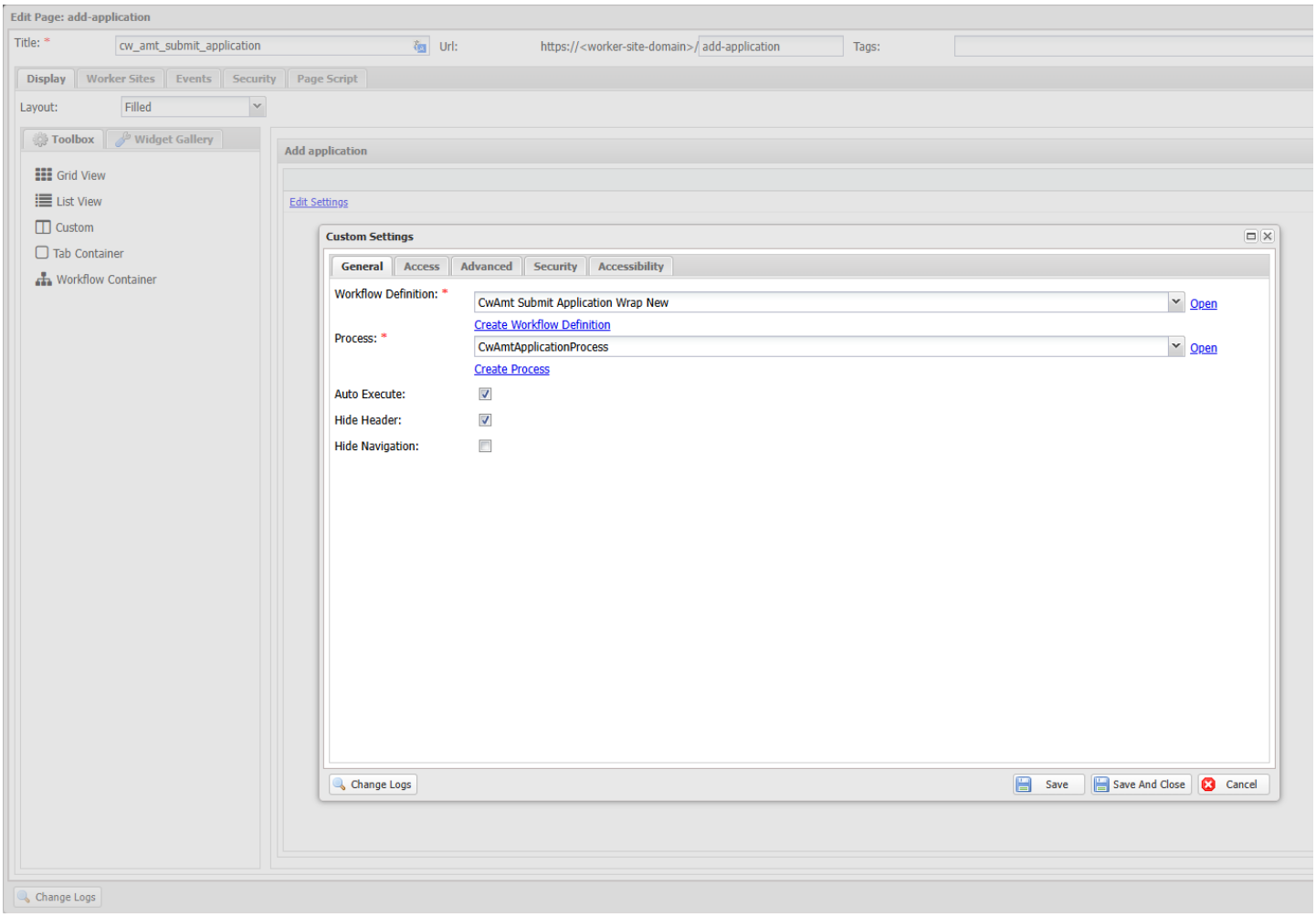
- **Used For Creating New Cases**: On
- **Accepy Input**: Off. You can still enable it if your workflow definition requires input for creating new cases.



The create-case workflow definition will then incorporate the edit-case workflow.



You can now use the workflow definition. The screenshot below illustrates how to use the create-case workflow in a page workflow container widget.


Edit Page: add-application
Title: *
cw_amt_submit_application
Url:
https://<worker-site-domain>/
add-application
Tags:
Display
Worker Sites
Events
Security
Page Script
Layout:
Filled
Toolbox
Widget Gallery
Grid View
List View
Custom
Tab Container
Workflow Container
Add application
Edit Settings
Custom Settings
General
Access
Advanced
Security
Accessibility
Workflow Definition: *
CwAmt Submit Application Wrap New
Open
Create Workflow Definition
Process: *
CwAmtApplicationProcess
Open
Create Process
Auto Execute:
Hide Header:
Hide Navigation:
Change Logs
Save
Save And Close
Cancel
Change Logs

Utilizing existing components invariably constitutes the most rapid and straightforward method for developing your Casewhere solutions. Casewhere offers a wide range of ready-to-install applications, components, plugins, and other resources, from which you may select the most suitable ones for your business solutions.

Applications

Casewhere resources, such as processes, workflows, data classes, and even other products, are bundled together to assist end users in achieving specific business goals.

- Form management
- Language
- Mailbox
- User management

Components

Casewhere resources, such as processes, workflows, data classes, plugins, and widgets, are bundled together to help configurators and developers handle specific tasks.

- Application Changelog
- Audit Log Export
- Automatic logout
- Comment Box
- Data migration
- Data export
- IdP sync
- Location data
- OData URI Parser
- REST data source
- SFTP hub
- Performance report
- Document management
- API key management
- Global settings
- Self-service theme

Integrations

Applications or components designed for connecting one system to another.

- Navision
- CPR Update
- Digital Post

Plugins

.NET libraries developed on top of the Casewhere plugin framework to provide custom functionalities of Casewhere.

- Aspose Word
- Aspose PDF
- CVR
- File compression
- JWT

- Query Filter String Parser
- SQL client
- SFTP
- SSH
- CSV
- SMTP

Template

A reusable piece of code, script, or configuration to help you quickly implement a functionality with a minimum customization effort.

- Mobile app template

Themes

To customize the appearance of the worker sites, including layout, color and typographies.

- Case administration
- Self-service portal

Application: Mailbox

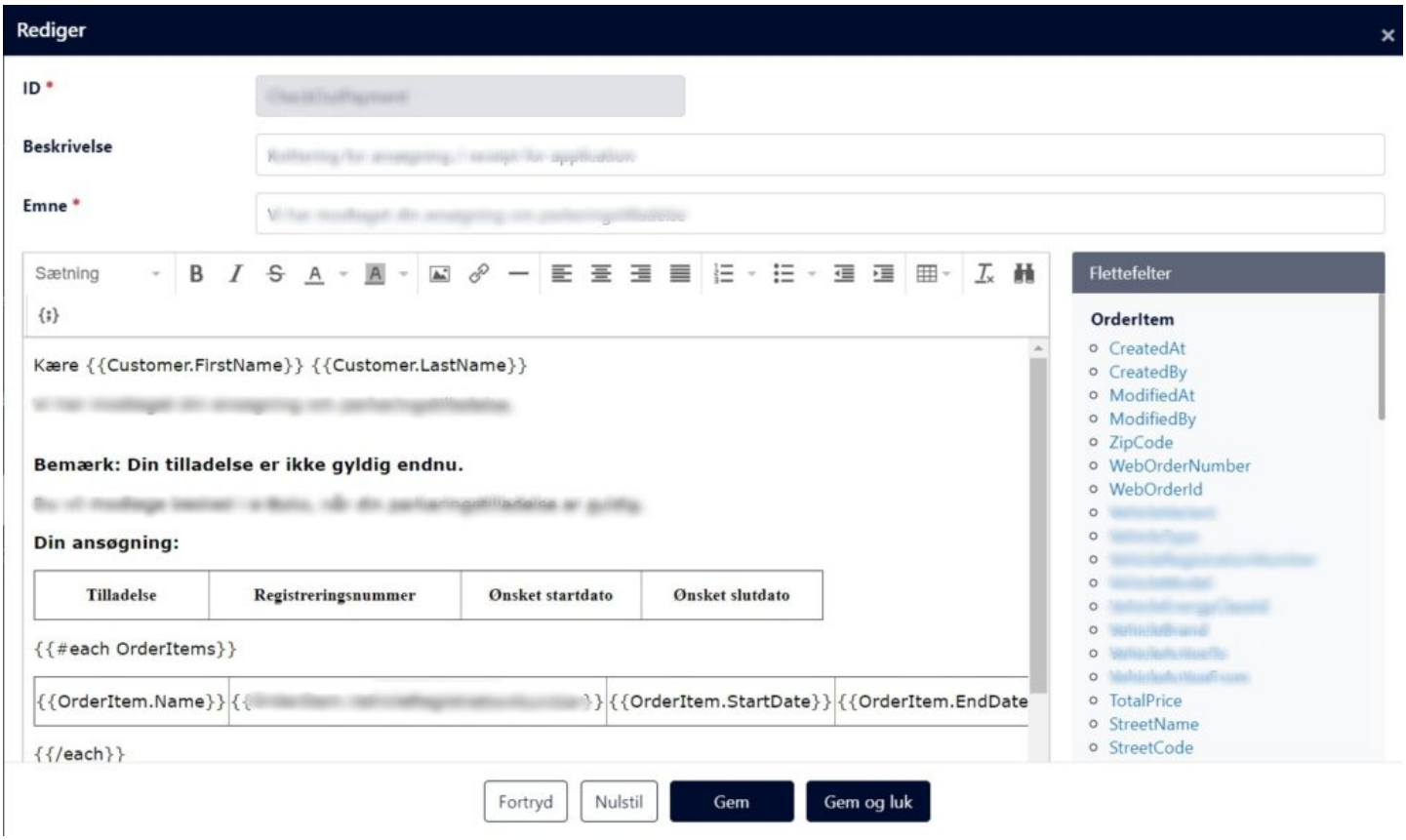
Introduction

Email is a typical communication channel needed in most solutions. This application with many end-to-end features will help you quickly integrate email functionality to your Casewhere solution and enable your end-users to efficiently manage the whole life cycle of the emails, from creation, queuing to sending with different strategies.

Features

Email templates

You can define templates with merge fields to standardize email throughout your Casewhere solution. The merge fields will be replaced with context-specific data at runtime before sending the mail to receivers. The email templates are composed in Casewhere's standard HTML editor that supports various predefined formatting tools to make it easy to set up well-structured emails for the receivers. Handlebars (https://handlebarsjs.com) is used as the template engine, which provides the power necessary to let you build semantic templates effectively with no frustration. It is also possible to enable your end-users to edit the email templates.



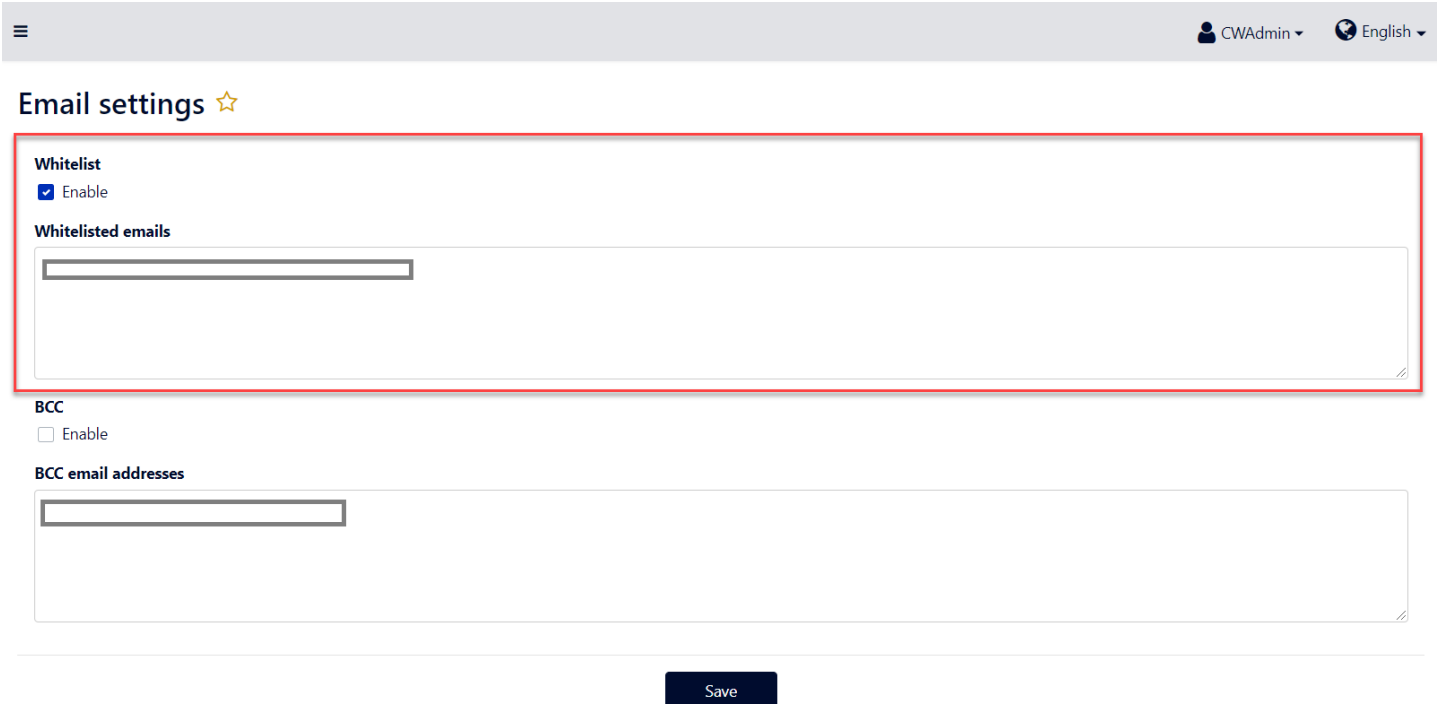
Email sending

The application provides three methods for sending emails:

- **Queued**: When an email object is created with the type **Queued**, through Data Trigger, a **Sending email** workflow is immediately created and queued. Email sending is based on the configured priority.
- **Manual**: When an email object is created with the type **Manual**, it is up to your solution to create the **Sending email** workflow and trigger it as you want it sent.

Email sending whitelist

An email whitelist is a list of trusted contacts. Messages from contacts on this list will never end up in your junk or spam folders. You can setup email whitelist by going to **Email settings**



SMTP configuration

The application includes an SMTP plugin, which is responsible for sending emails from Casewhere. You need to provide the SMTP server information in the Plugin Setting page in Casewhere Admin for Casewhere to send emails. Furthermore, the application provides a Worker Sites page that you can utilize to delegate the configuration to your end-users.

Installation

Requirements

- Casewhere 2.6 or later
- SMTP server

Configuration

After installing, please follow the below steps to configure the application:

1. Configure the SMTP server in the **Plugin Setting** in Casewhere Admin. You can later enable your end-users to override these settings from a **Worker Sites** page.
2. Define your email templates. Make sure you preview all templates before publishing them.
3. Integrate the email templates with corresponding transactional workflows. Make sure you specify the correct **Template Code** when creating email objects in your workflows.
4. Test your email templates and sending logic.
5. Assign security rules to control access so only relevant users can edit email templates and monitor emails from the **Worker Sites**.

Releases

1.1.0 - 08/12/2021

Changelog

- Email sending whitelist

Download (login required): Casewhere Mailbox v1.1.0

1.0.0 - 13/04/2021

Changelog

- Email template
- Email sending

Download (login required): Casewhere Mailbox v1.0.0

Application: User management

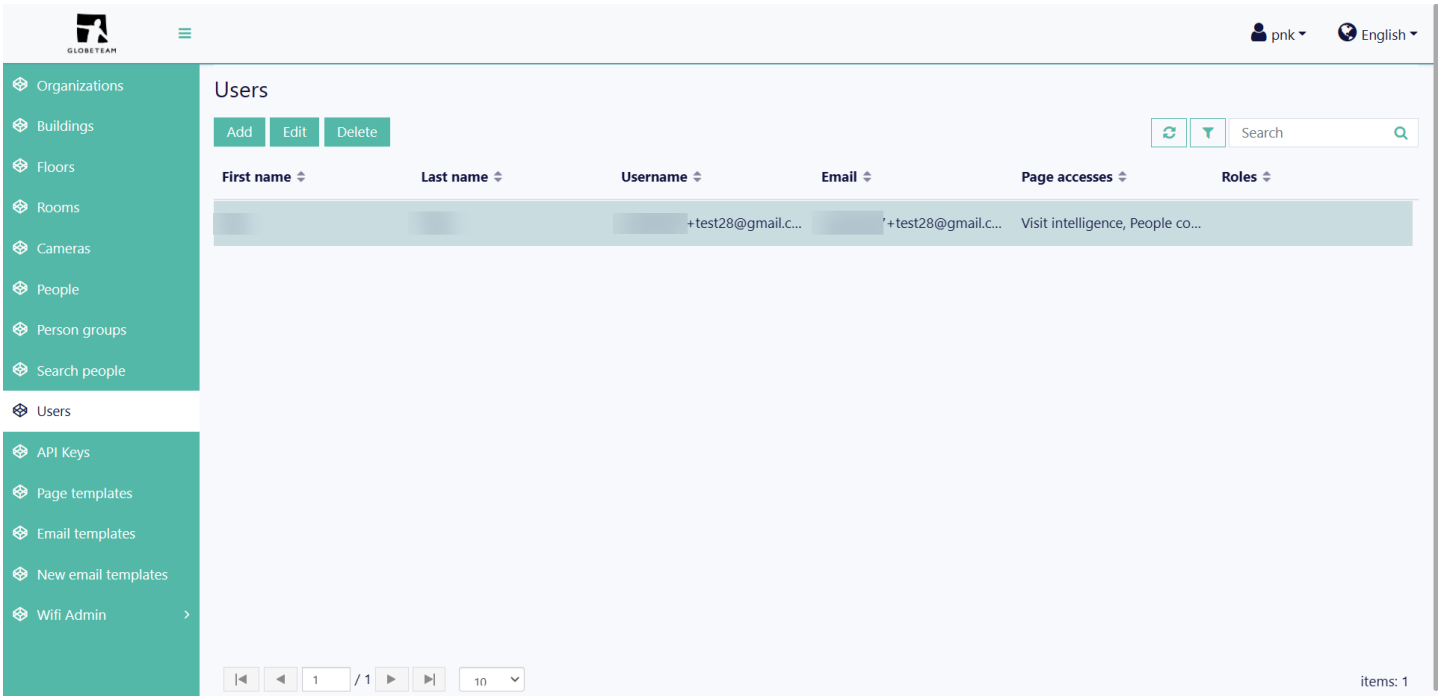
Introduction

Casewhere makes it possible to manage your user base both within Casewhere as well as rely on an external user repository like Active Directory or Safewhere*Identify. It is even possible to combine a Casewhere managed user repository with an external one. The user profile fields are fully configurable and integrates seamlessly with other modules like Access Rules, CPR, CVR, eBoks, Workflow, and many more. Various default fields come out of the box like name, position, address, organization, username, password, and many more. User management can be departmentalized to share/divide responsibility and access to users across the organization.

Features

User management

The application provides an interface for administrators to manage users and assign permissions.



Role-based access control

Role-based access control is an approach that uses the job functions performed by individual users within the organization to determine their appropriate access levels. Grouping common access permissions into roles provides a secure and efficient way to manage access, while simplifying the process for both administrators and users.

We support both role-based and permission-based access control. It's possible to assign various permissions to various users or various roles.

Integrate with standard IdPs

The application provides a standard interface to integrate with different IdPs such as AD, Safewhere*Identify.

Installation

Requirements

- Casewhere 2.6 or later
- IdP - Active Directory / Safewhere*Identify

Configuration

1. Import the product
2. Configure the plugin settings
3. Grant permissions for admin users

Releases

1.0.0 - 20/09/2020

Changelog

- User management
- Role-based access control
- Integrate with Identify

Download (login required): User management application v1.0.0

Application: Text resources management

Introduction

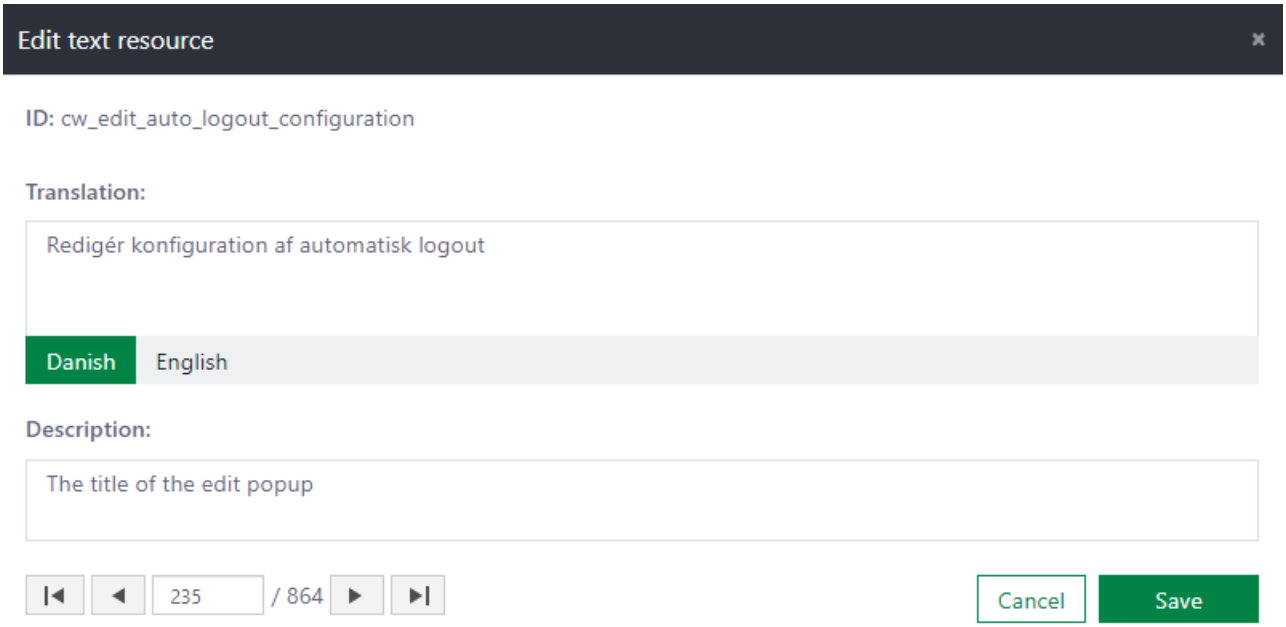
Supporting multiple languages is now popular for most of the applications. To support that demand, Casewhere provides a central place to manage text resources. It has many features: add, edit, delete a text resource, import text resources from an excel file, export current text resources to an excel file...

Features

Text resources

You can do the CRUD functions for text resources used in your solution. Each text resource will have the following settings:

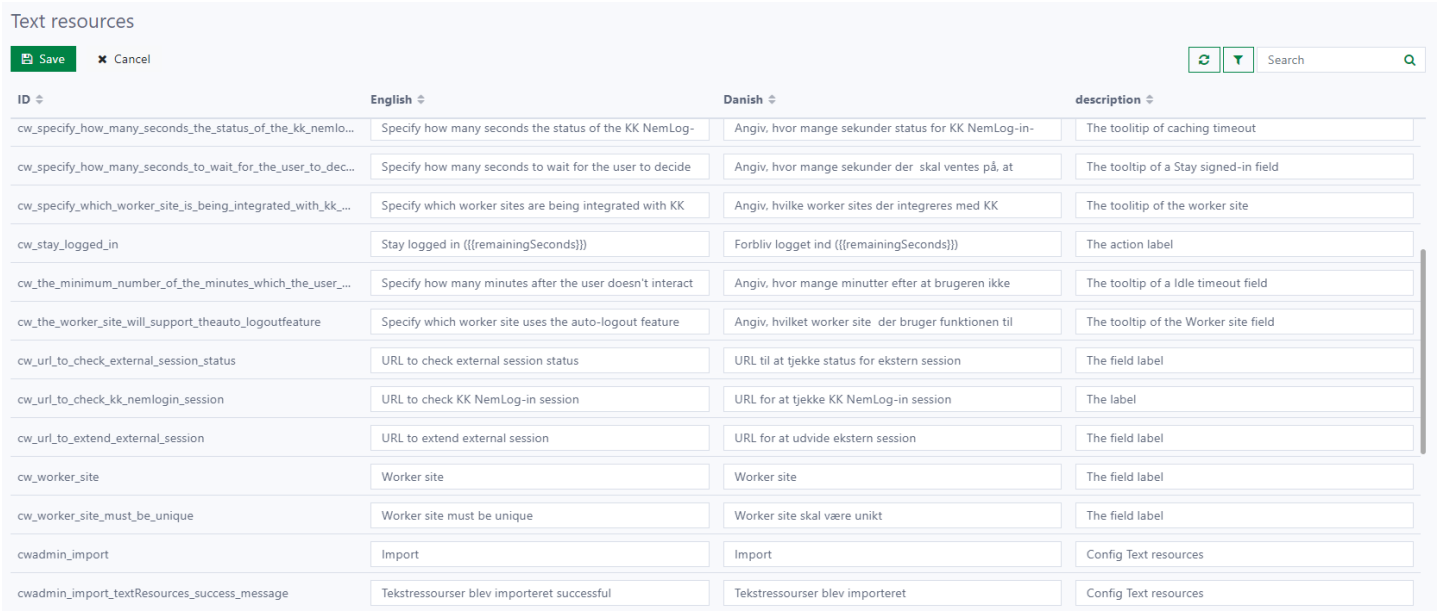
- ID: the text resource key
- Translation:
  - All configurated languages will be shown here, you can input the content for each language.
  - More information about configurating languages can be found here.
- Description: the description of a text resource.



Moreover, you can also prepare an excel file in the following format and import it to your system. Exporting text resources will share the same format.

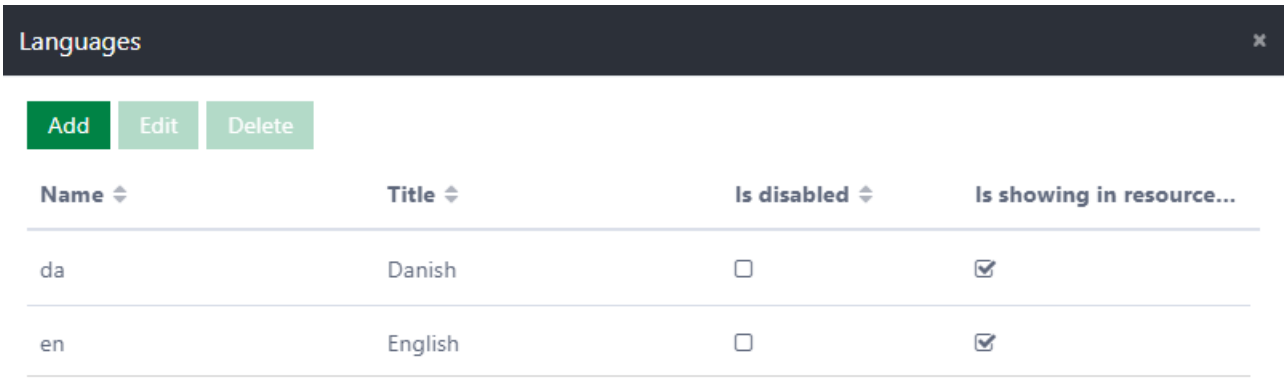
	A	B	C	D
1	ID	English	Danish	description
2	cw_edit_auto_logout_configuration	Edit auto-logout configuration	Redigér konfiguration af automatisk logout	The title of the edit popup
3	cw_edit_kk_nemlogin_session_configuration	Edit KK NemLog-in session configuration	Rediger KK NemLog-in session konfiguration	The title of the edit popup
4				
5				
6				

In case you want to edit multiple text resources, you can use the mass editing function:



Languages

You can add/edit/delete languages, depends on your solution demand.



Installation

Requirements

- Casewhere 2.6 or later

Configuration

1. Import the product.
2. Assign all pages to your proper worker site.

Releases

1.0.0 - 12/05/2021

Changelog

- CRUD
- Import/Export
- Mass editing
- Set languages

Download (login required): Casewhere Text resources v1.0.0

Roadmap

- Support text resource type.
- Support WYSIWYG editor.

Application: Jira to Targetprocess

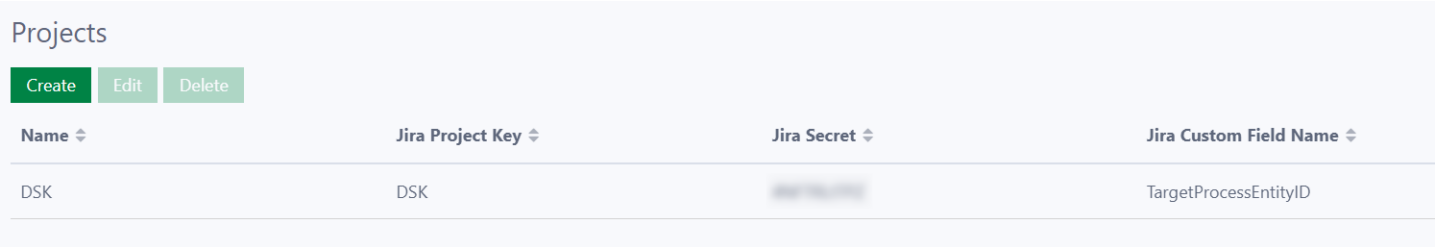
Introduction

Certain projects are managed using the Targetprocess, which is used for planning, tracking, and other administrative functions. However, because customer requests are typically submitted through Jira, it's difficult for us to keep track of which requests have been submitted and which comments have been added. To close this gap, we created the component Jira2Tp Synchronization, which allows us to automatically synchronize requests, comments, and other data from Jira to Targetprocess.

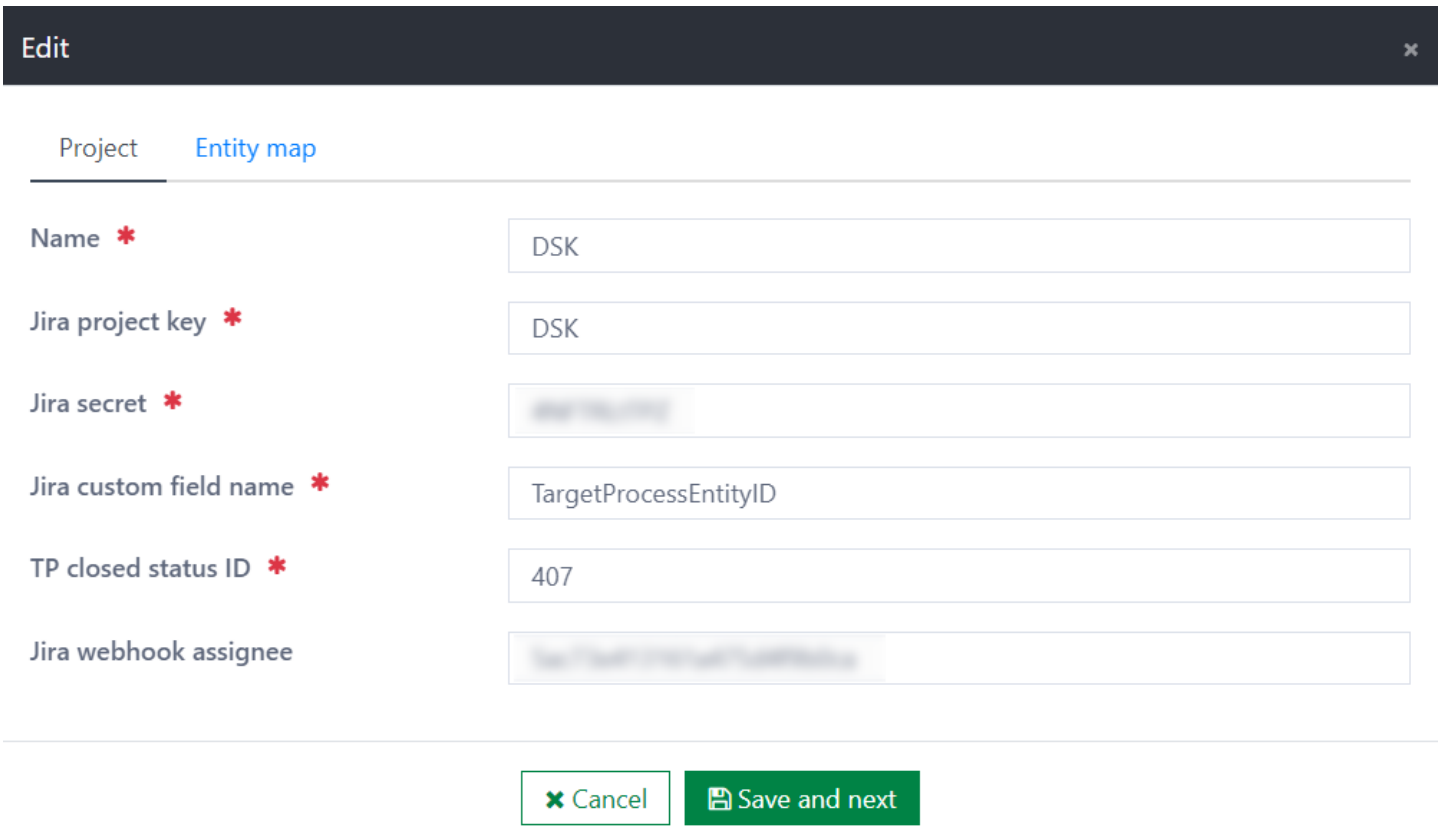
Features

Projects

To let the tool know which projects need to be sync'ed, we need to add them to Casewhere first.



We need to configure some more fields as following:



- **Name**: the name of project to manage in Casewhere.
- **Jira project key**: the key of the project in Jira system.
- **Jira secret**: a secret key added in the Jira webhook, this is to let Casewhere knows that the requests coming to the Casewhere system are exactly from the Jira.
- **Jira custom field name**: we need to have a field to display the sync'ed TP-ID for each Jira request, this field is to provide the name for this field.
- **TP closed status ID**: each project may have a different set of statuses, to run some background jobs, we need to provide

the closed status ID.
- **Jira webhook assignee**: the user-id in the Jira system. This is to double-check that we only sync requests which are assigned to this user.

Entity map

We have done the first step. The next one is also the most important part, where we need to specify the logic for mapping fields between Jira and Targetprocess. This is in the JSON format, each object will have the following attributes:

Example:

```
[
  {
    "targetFieldName": "AssignedTeams",
    "targetFieldValue": {
      "Items": [
        {
          "Team": {
            "Id": 63942
          }
        },
        {
          "Team": {
              "Id": 70528
          }
        }
      ]
    },
    "isRequiredJiraChange": true,
    "targetIsCustomField": true
  },
  ...
]
```

- **sourceFieldName**: [Optional] the field name from Jira. This can be a JSONPath query, more about JSONPath can be found here.
- **targetFieldName**: [Required] the field name in Targetproccess.
- **targetFieldValue**: [Optional] the fixed value for the **targetFieldName**, it means if it defined, the value of it will always be set to the **targetFieldName**.
- **values**: [Optional] an object contains a pair key/value, which equals to Jira's value/Target process'value.
  - Example:
    `json "sourceFieldName": "status.name", "targetFieldName": "EntityState", "values": { "Open": { "Id": 64 }, "Reopened": { "Id": 1477 } }`
    It means, for the status **Open** on Jira, it will be mapped to "Id: 64" on Targetprocess, which is Open. The same logic applied for **Reopened**.

- **isRequiredJiraChange**: [Optional] if set to true, the field will only be synchronized when the targetFieldName on Jira changes its value. Otherwise, it will always be synced.
- **targetIsCustomField**: [Optional] set it to true to let Casewhere knows that it's a custom field on the Targetprocess.

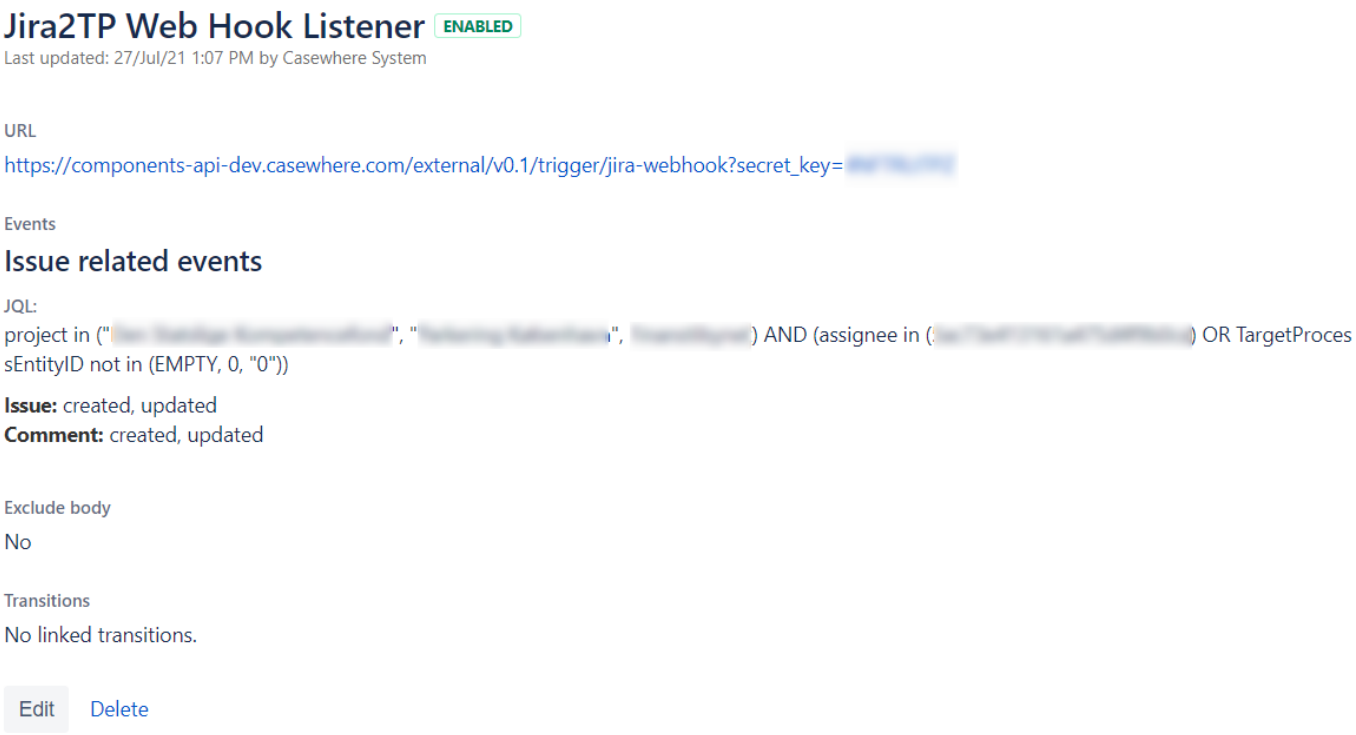
Installation

Requirements

- Casewhere 2.6 or later

Configuration

1. Import the product.
2. If you have access to the Jira, set up a webhook with the following information:



3. Replace the URL with your API endpoint, also remember to input the secret key configured in the project.

Releases

2.0.1 - 02/03/2022

Changelog

- Update mappings, allow ignoring specific values of a Jira field.
- Automatically adding tags in Targetprocess for some fields.

Download (login required): Jira2Tp v2.0.1

2.0.0 - 06/05/2021

Changelog

- Configure projects, entity mappings.
- Sync requests, comments, attachments.

Download (login required): Jira2Tp v2.0.0

1.0.0 - 10/04/2018 (Deprecated)

Changelog

- Sync requests, comments.

Download (login required): Jira integration v1.0.0

Roadmap

- No plan as of now.

Application: Form management

Introduction

Form management is your solution to manage, build and deploy web forms. With the form management component, you can flexibly customize the form based on your own business solutions. The component also adapts well to the complex data management and business process applications. It provides an easy-to-use Form builder interface designed specifically for non-technical users, you can control and manage the forms without the assistance of the solution provider.

This video will give you a brief introduction about our form management.

Feature

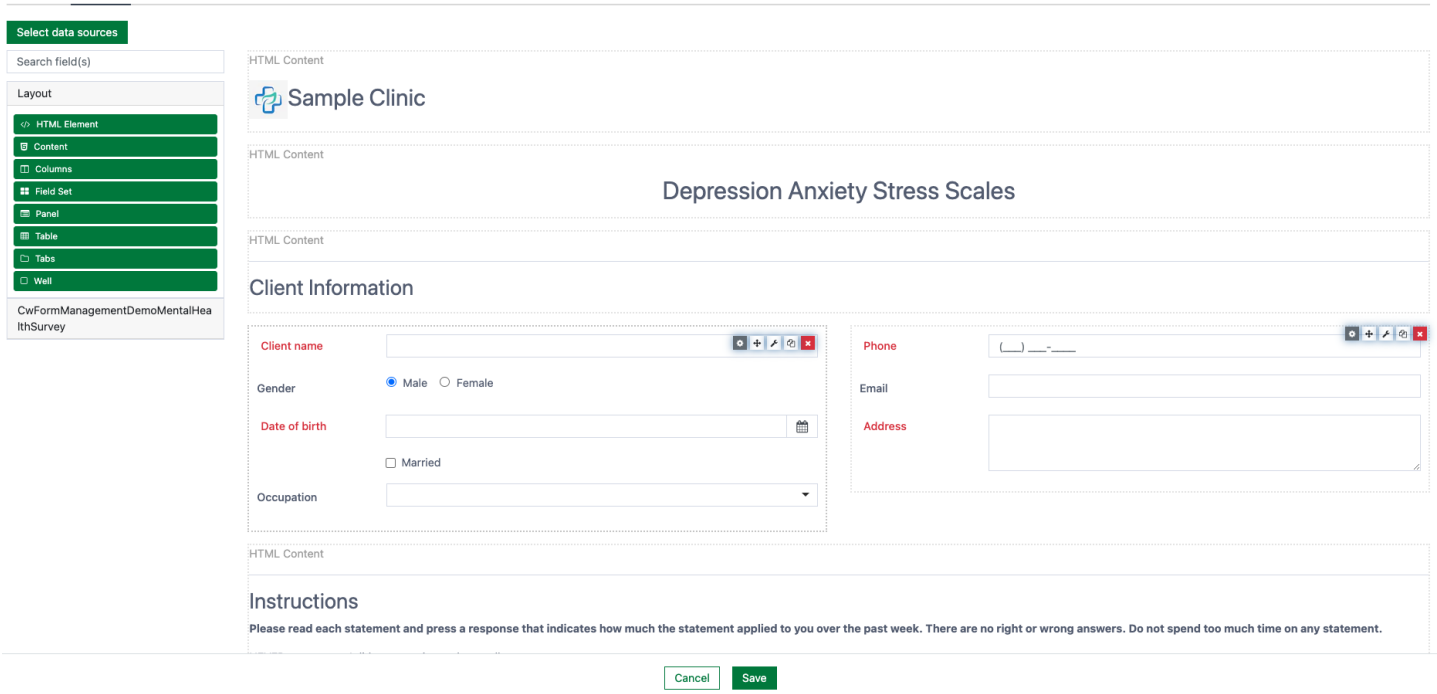
Form builders

Form builders is a visual form designer which allows you to build and deploy forms very quickly.

You as the form admin can create as many forms as needed. Once you publish a form, it will be ready for your end-users to access and submit data.

With Form builder, you can define the validation rules for your form fields, and specify the error messages to display. You can also define whether fields are visible or editable by users; and create rules to automatically calculate and populate their values, for example, the dependency rules about populating field data between fields.

The following is an example of a Form builder. When you build the form, it is a good practice to define the form layout at first. You can make your form a table with rows and columns, and sometimes it is best to organize content into tabs, etc. You can even compose a form with multiple sections having different layouts. After settling the form layout, you can drag and drop fields into the defined placeholders.



To explore more, please read the Form guide.

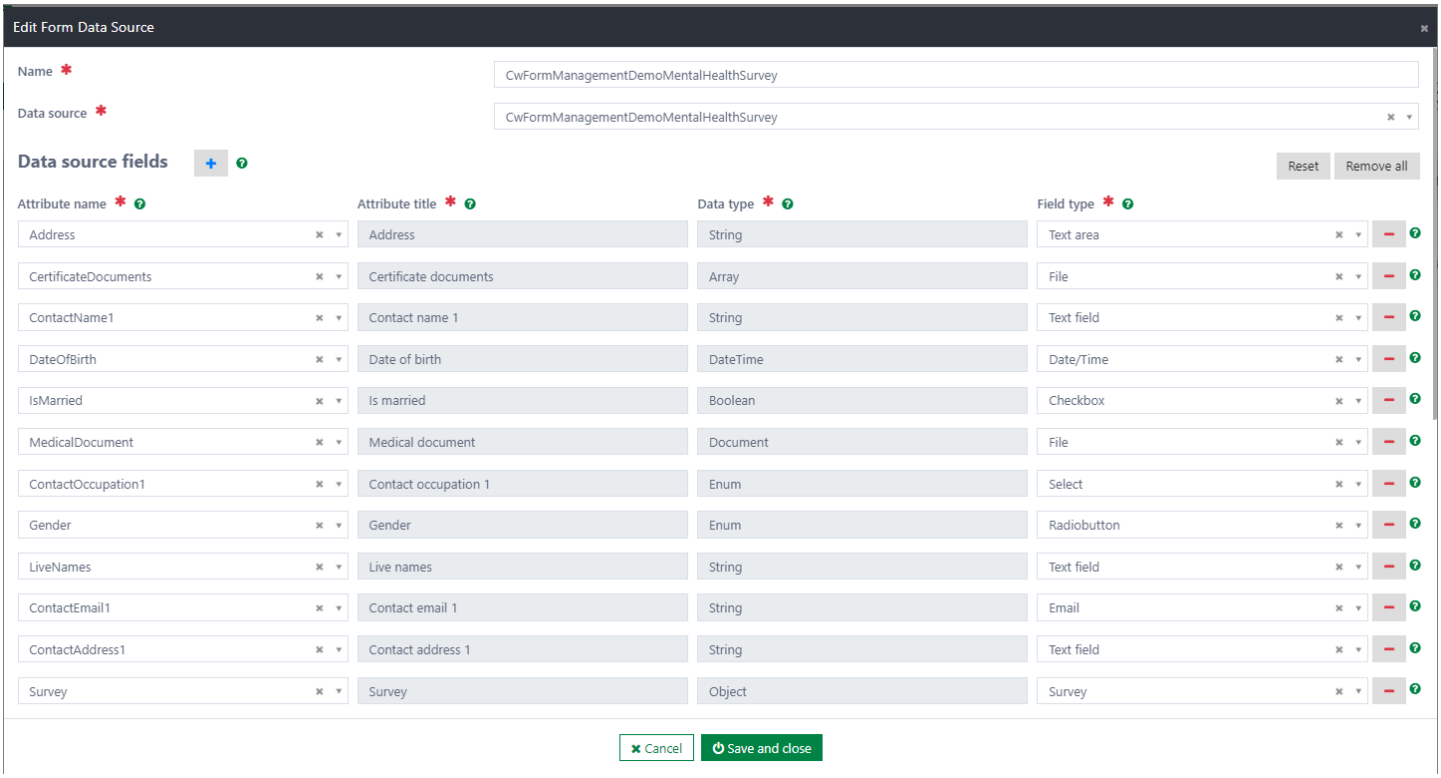
Data management

Data management provides an efficient way to organize the interaction between Form and Casewhere data models. The form admin can configure the field mappings between Casewhere data models and the Form, in which the data model defines the data

object's structure and the Form determines how the data object looks like to the user.

Once the mapping has been configured, the data object can be created, read, deleted, and queried via the Form.

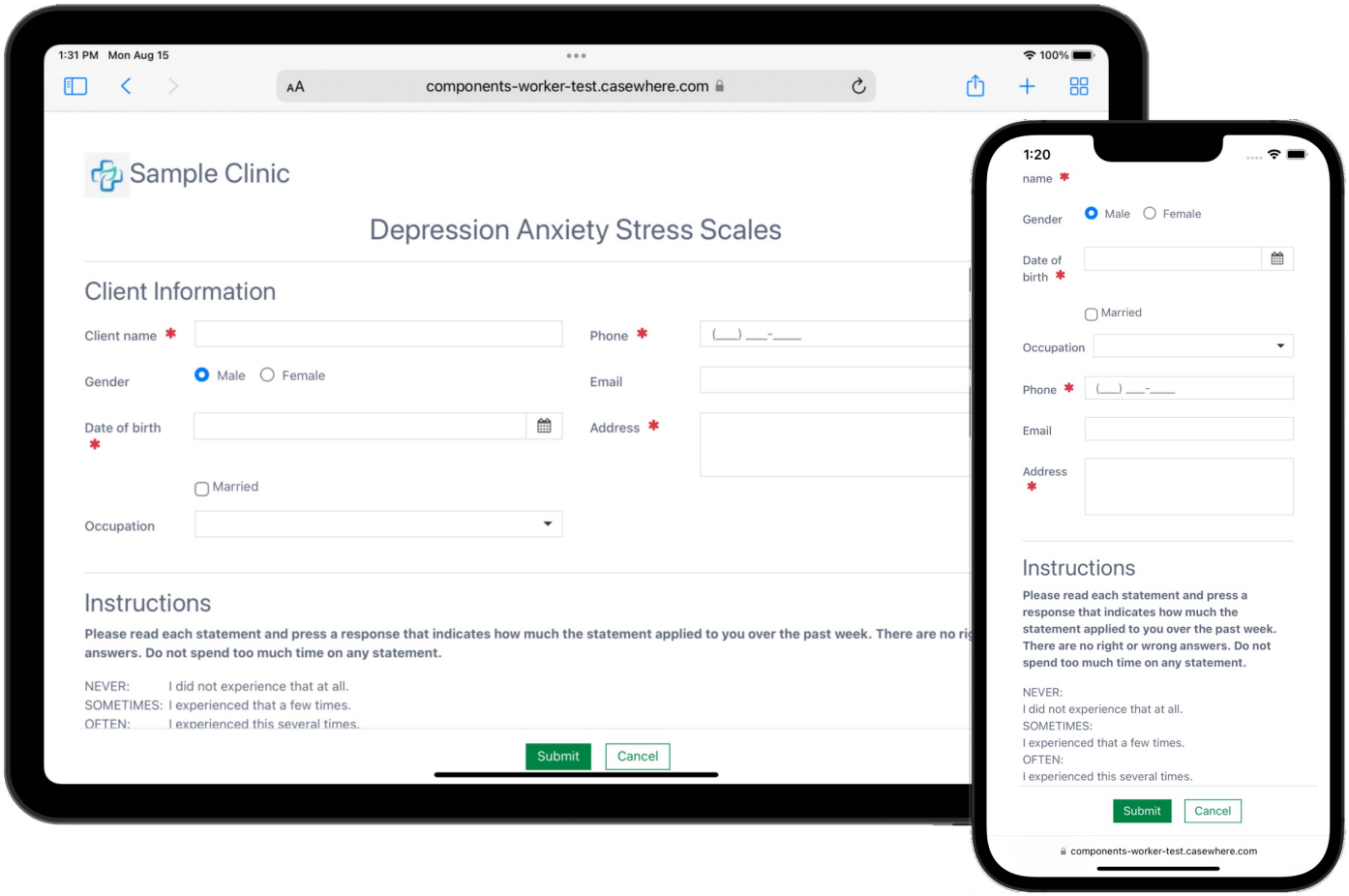
Besides, the feature provides a fast and friendly method to generate Form fields from the existing data models. Here you can flexibly select what entities and fields should be available for use in the Form builder.



For an in-depth explanation of how Data management work, please read the Data management guide.

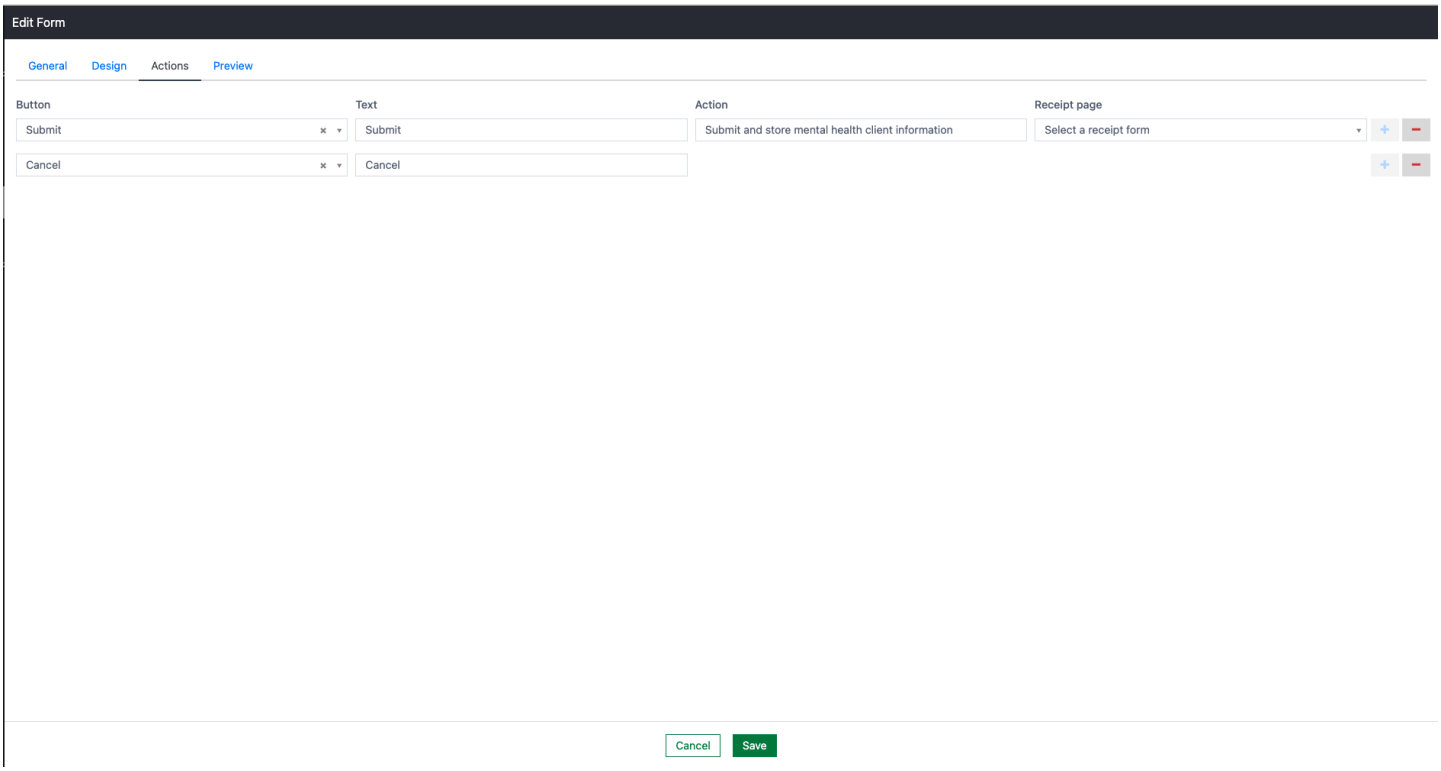
One form, multi-platform

The forms you created in the Form builder are web forms. They are implemented using standard web technologies and can be utilized for mobile apps. Hence, it saves us cost and effort by avoiding making different form engines for web and apps.



Actions

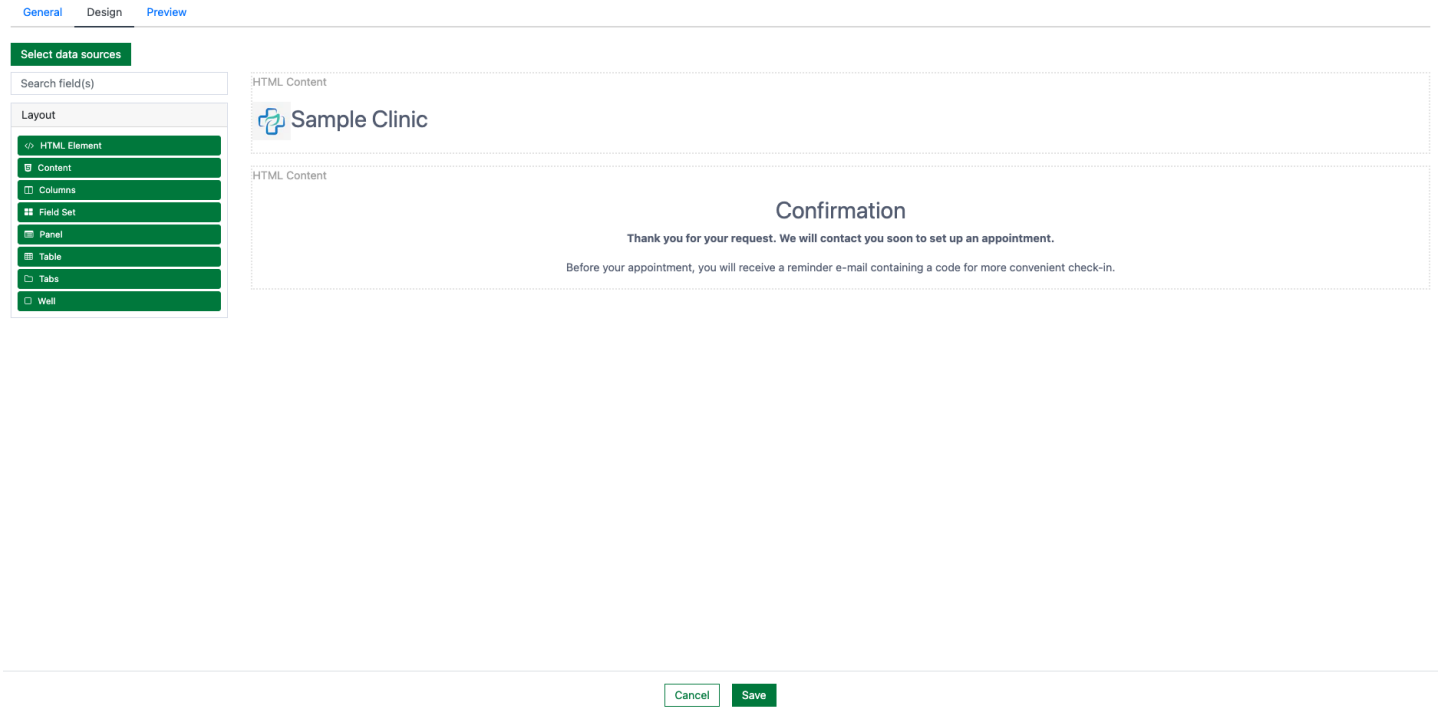
You can configure which workflow (selected from a predefined list) handles the form submission when creating a form. For that, you can decide how to process data for each form and where the users are directed to after the data is submitted, e.g., you can send a notification email after the registration application form is submitted.



The detail guideline is described in Form guide.

Receipt forms

Normally, a receipt will be displayed after a form is submitted, and the Receipt Forms configuration will handle that business. The admin can create, design, and then attach the receipt form to the main form.



To explore more, please read the Receipt form guide.

Form render

The form is rendered the same as how you see it in the Form builder.

- The form is friendly, responsive, and complies with WCAG 2.1
- The form can be translated in multiple languages. It allows your users to choose the language they want to use. The component also is integrated with Casewhere Text Resource so that you can easily manage all of the text resource in a single place.
- The form will be designed to support the access control management in the future versions. The idea is supporting to define the roles and the permissions for the form users that allows them to perform the certain actions as well as to access and submit form.

Installation

Requirements

- Casewhere 2.6.0 or later.

Configuration

- To get started, please go to the Installation guide

Dependencies

- Component: Casewhere Common ## Release ### 2.0.0 - 31/12/2024 **Changelog**
- Validation rules
- Form dependency rules

Download (login required): Form management v2.0.0

Roadmap

- Support multiple languages
- Support dynamic fields
- Integrate with mobile apps
- Access control management
- Offline forms

Component: Form management - Data management

Introduction

Data management provides an efficient way to organize the interaction between Form and Casewhere data models. The form admin can configure the field mappings between Casewhere data models and the Form, in which the data model defines the data object's structure and the Form determines how the data object looks like to the user.

Once the mapping has been configured, the data object can be created, read, deleted, and queried via the Form.

This article will show you how to manage the data models as well as how to use them in the Form builder.

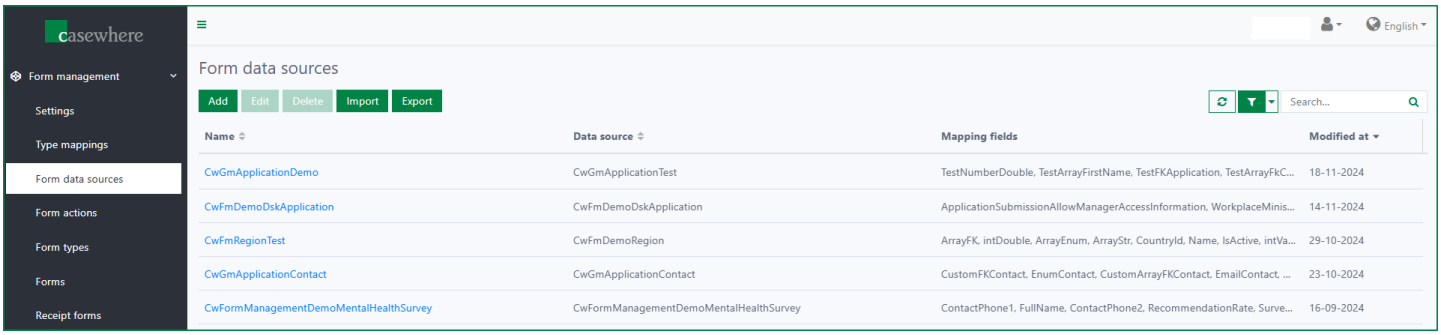
Guideline

Form data sources

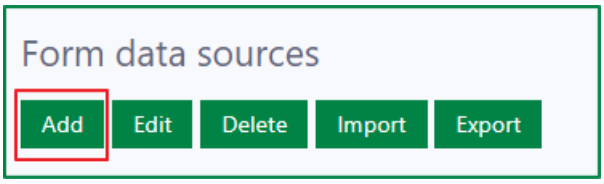
The **Form data sources** page is used to manage the data sources which are used for building the forms.

The Form data source is configured based on the original data sources in admin site. Instead of using all attributes from the original data sources, you can customize the form data source according to the using purpose.

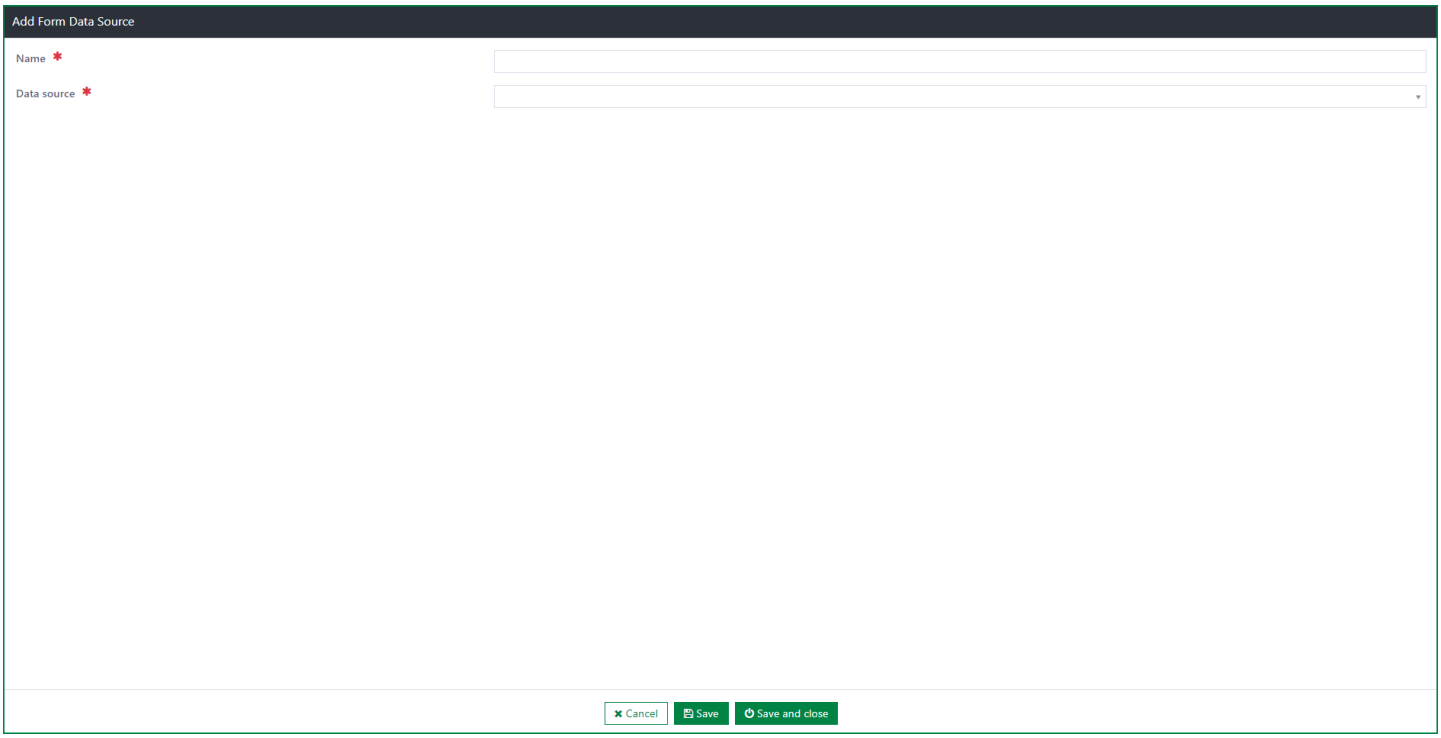
The data sources are displayed on the list with the basic information, like: **Name**, **Data source**, and **Mapping fields.** In those, **Data source** is name of the original data source from admin site, **Mapping fields** is list of the custom field of the form data source.



Add new form data sources

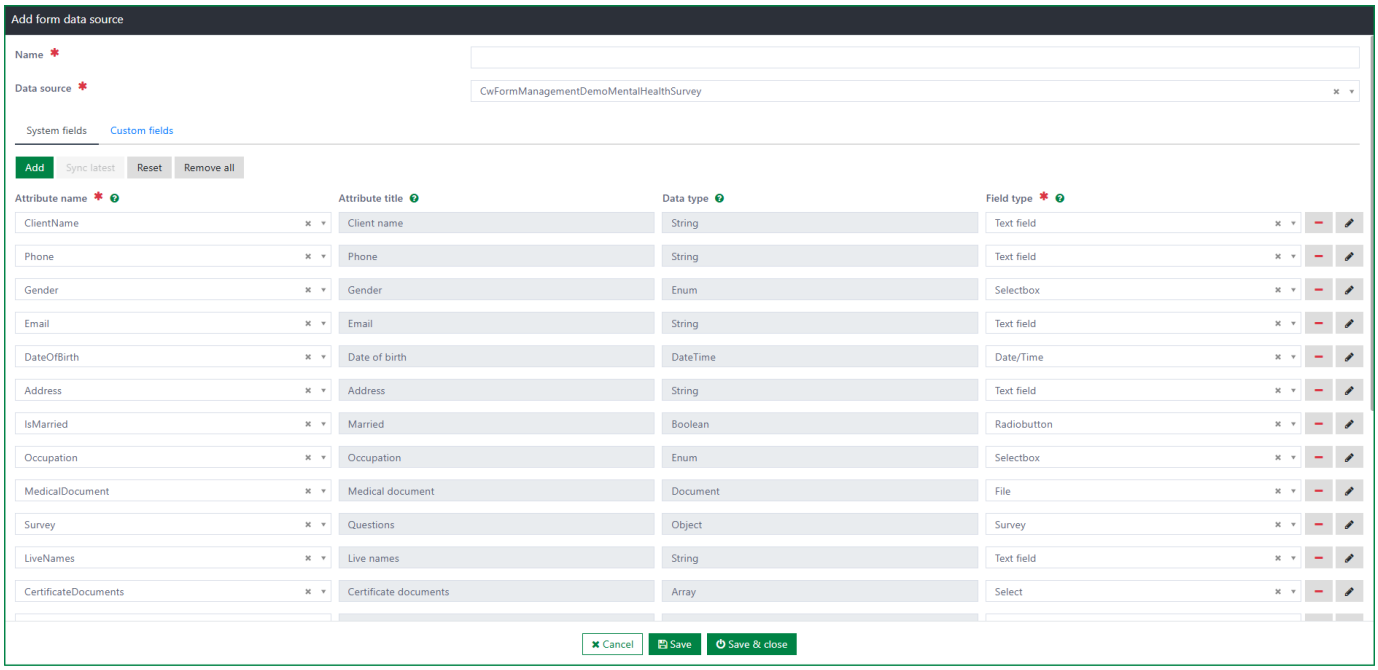


First, click on the **Add** button to open the **Add Form Data Source** dialog.



The dialog includes the following fields:

- **Name**
  - This field is required.
  - Form data source name must be unique and must not contain space and special character.
- **Data source**
  - This field is required.
  - You must select a data source from the list. List of data sources is retrieved from the **Data sources** page in admin site.
- **Data source fields**



  - The section of data source fields is only displayed after the Data source is selected
  - After the data source is selected, by default, the corresponding list of attributes will be displayed in the **System fields** section except for some CW built-in fields.
  - The section includes 2 parts, those are **System fields**, and **Custom fields**

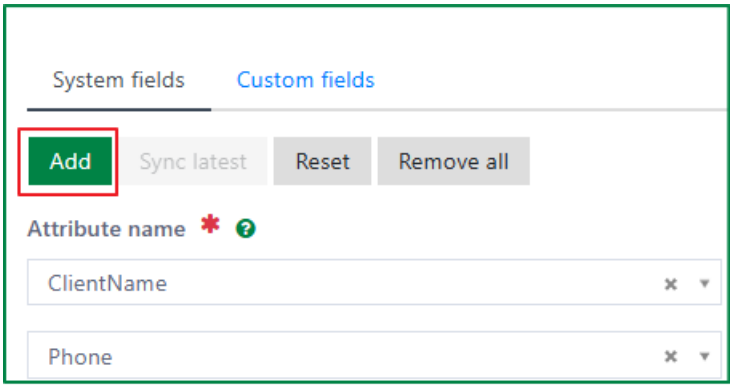
System fields

The system fields are displayed with the information, like: **Attribute name**, **Attribute title**, **Data type**, and **Field type**. Additional information about the attribute is displayed in the **Configure system field** dialog which is opened via the **Edit** button at the right of each row.

The available actions in **System fields** section are:

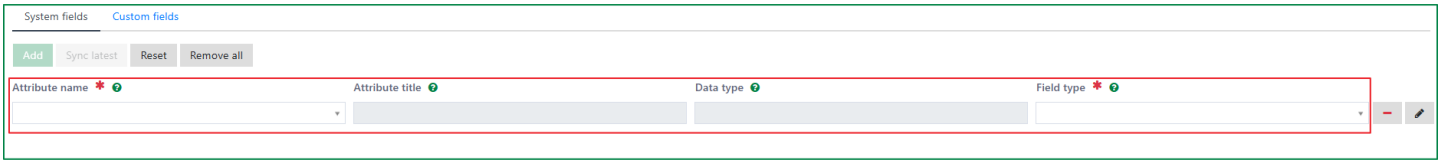
- Add system field from the Casewhere data source
- Edit field type of the system field
- Configure additional information for the system field
- Remove system fields from the form data source
- Reset all changes made on the system fields list

Add system field from Casewhere data source



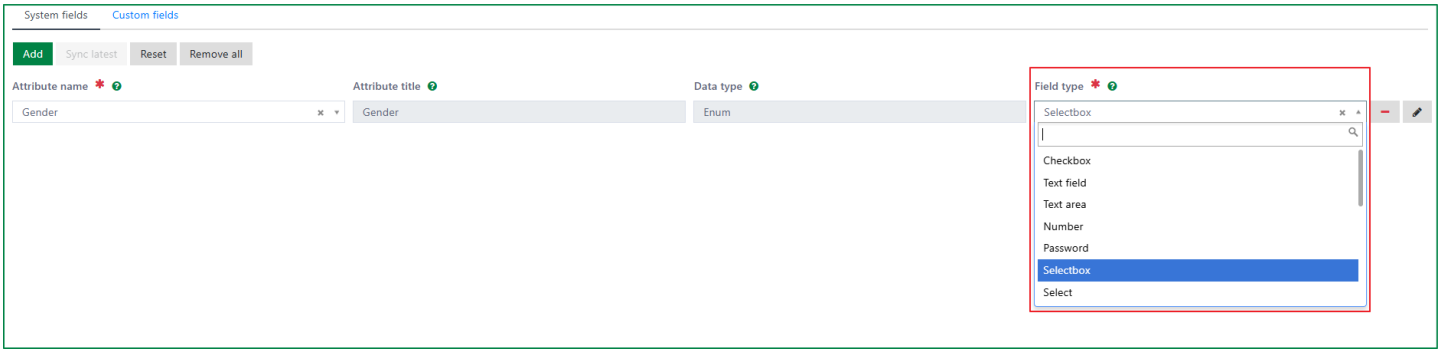
After the Casewhere data source is selected, all the system fields (except for the built-in ones) under it will be displayed at the **System fields** list. You can manually add the built-in fields, or add the removed ones back.

By clicking on the **Add** button, an additional row for new system field will be displayed. You must select the field in **Attribute name** column and select the **Field type** for the field.



You can only add new system field into the form data source if the required information of all existing system fields in the form data source has been filled.

Edit field type for the system field

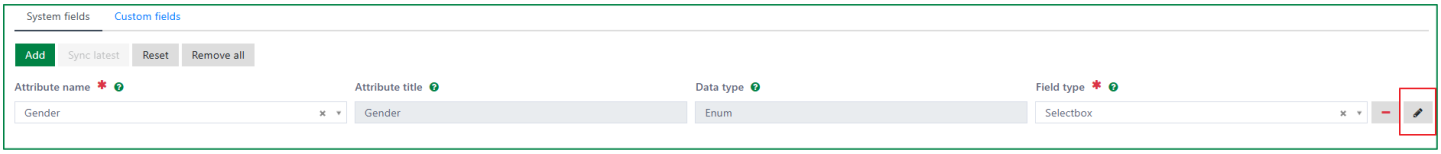


Each system field must have the field type, you can select the field type at the **Field type** column.

If the data type of the system field was already mapped with a field type in the **Type mappings** page, the field type will be pre-filled automatically. If the data type was not mapped with any field type in the **Type mappings** page, the field type will be empty and you must select a field type for the system field.

You can also edit the field type of the system field in the **Configure system field** dialog.

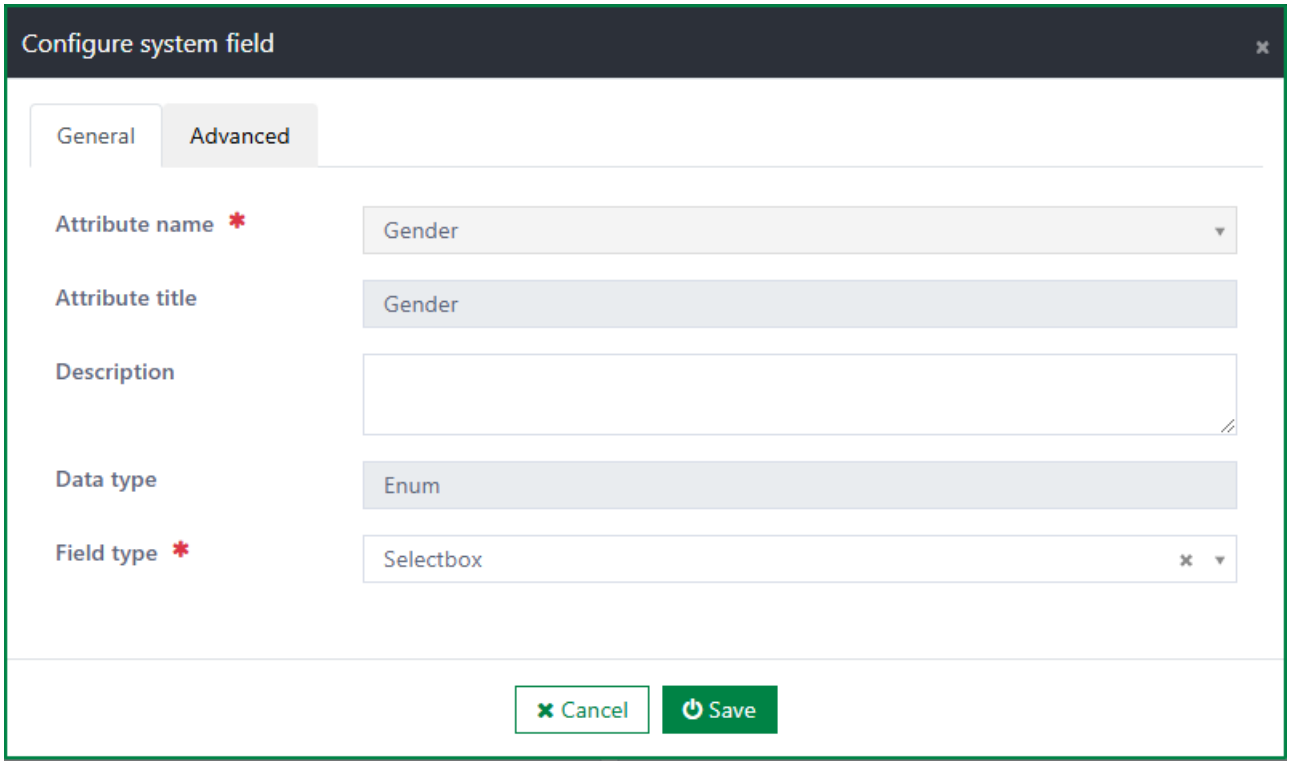
Configure additional information of the system field



By clicking on the **Edit** button at each field row, the **Configure system field** dialog of the system field will be opened.

The dialog includes 2 tabs: **General**, and **Advanced**.

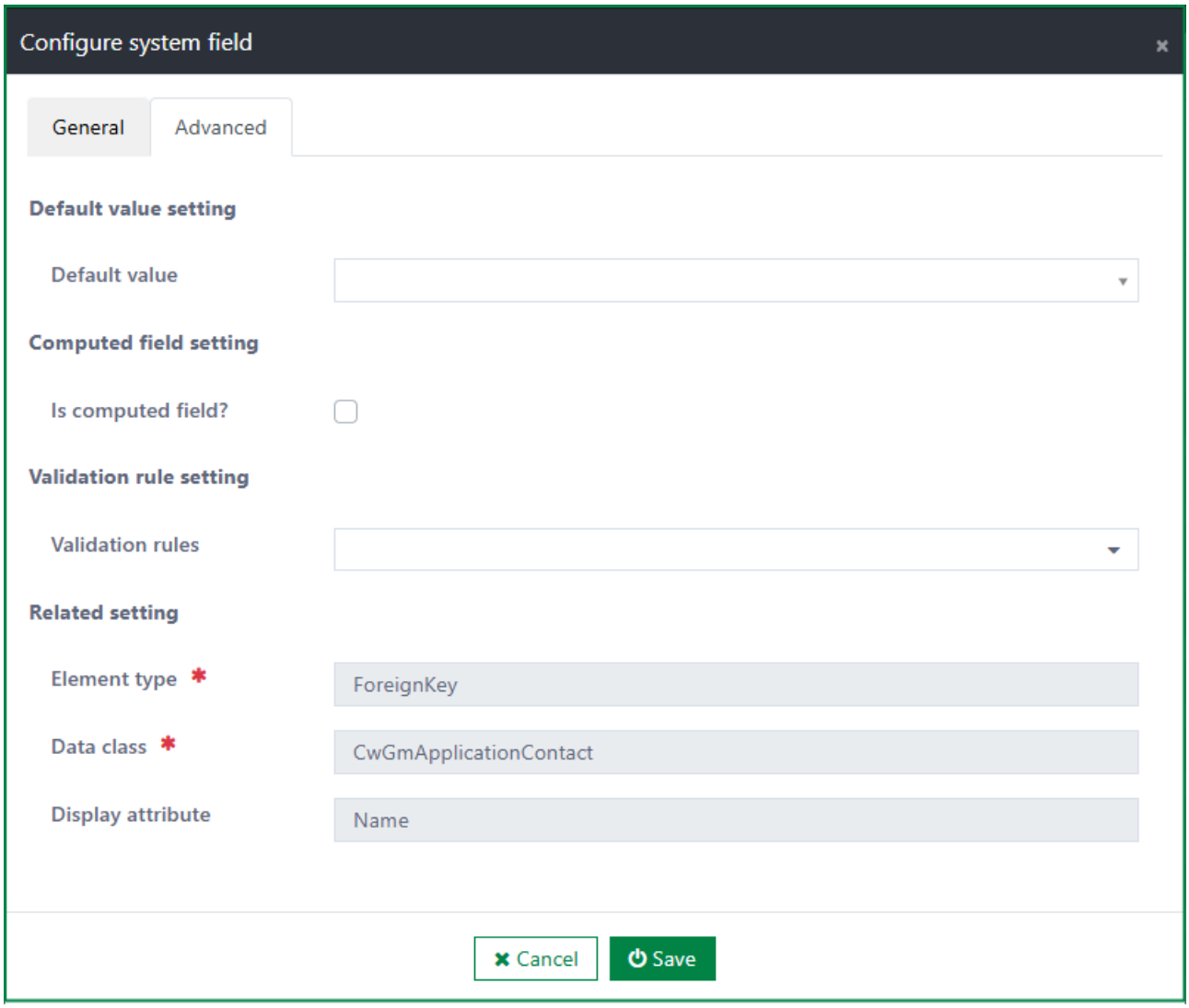
- **General** tab



  The **General** tab includes the following fields:

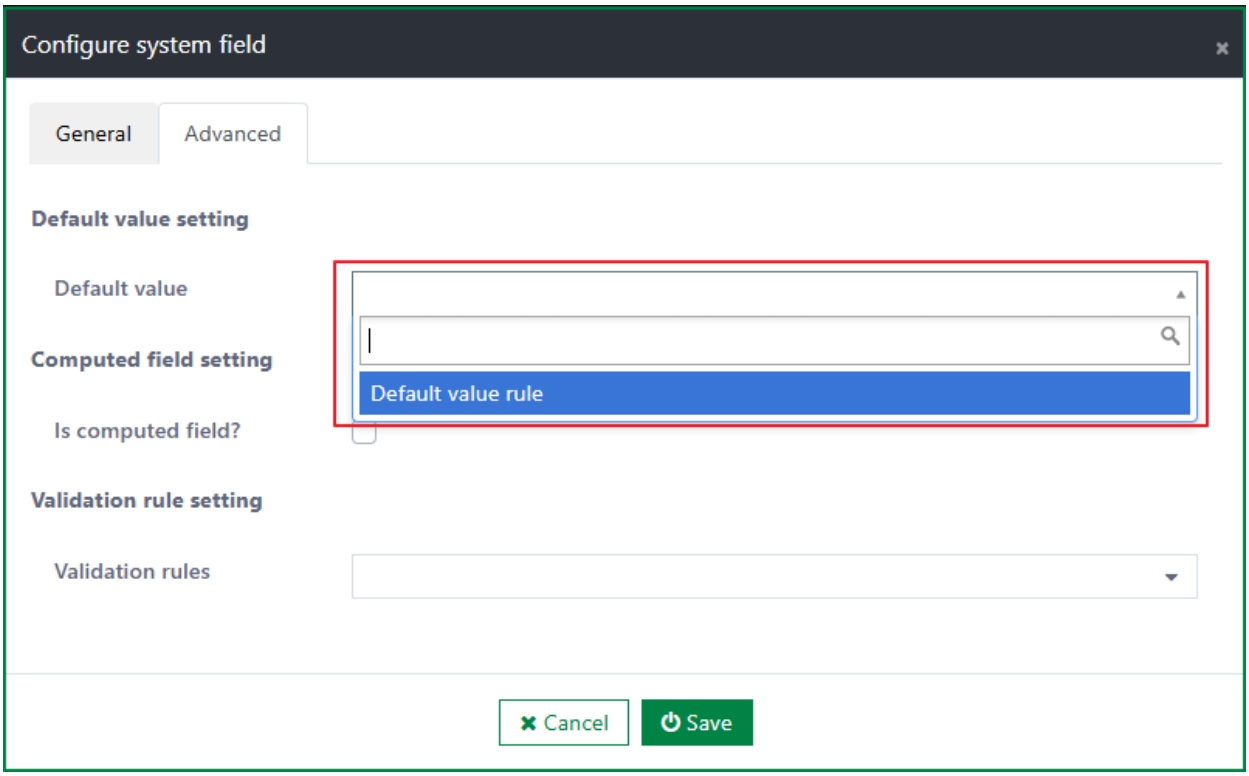
  - **Attribute name**
    - This field is pre-filled and read-only.
    - **Attribute title**
      - This field is pre-filled and read-only.
    - **Description**
      - You can describe more about the system field via this field.
    - **Data type**
      - This field is pre-filled and read-only.
    - **Field type**
      - If the field type was filled in the **System fields** list, the value will be displayed properly in the dialog.
      - You can change the field type for the system field.

- **Advanced** tab



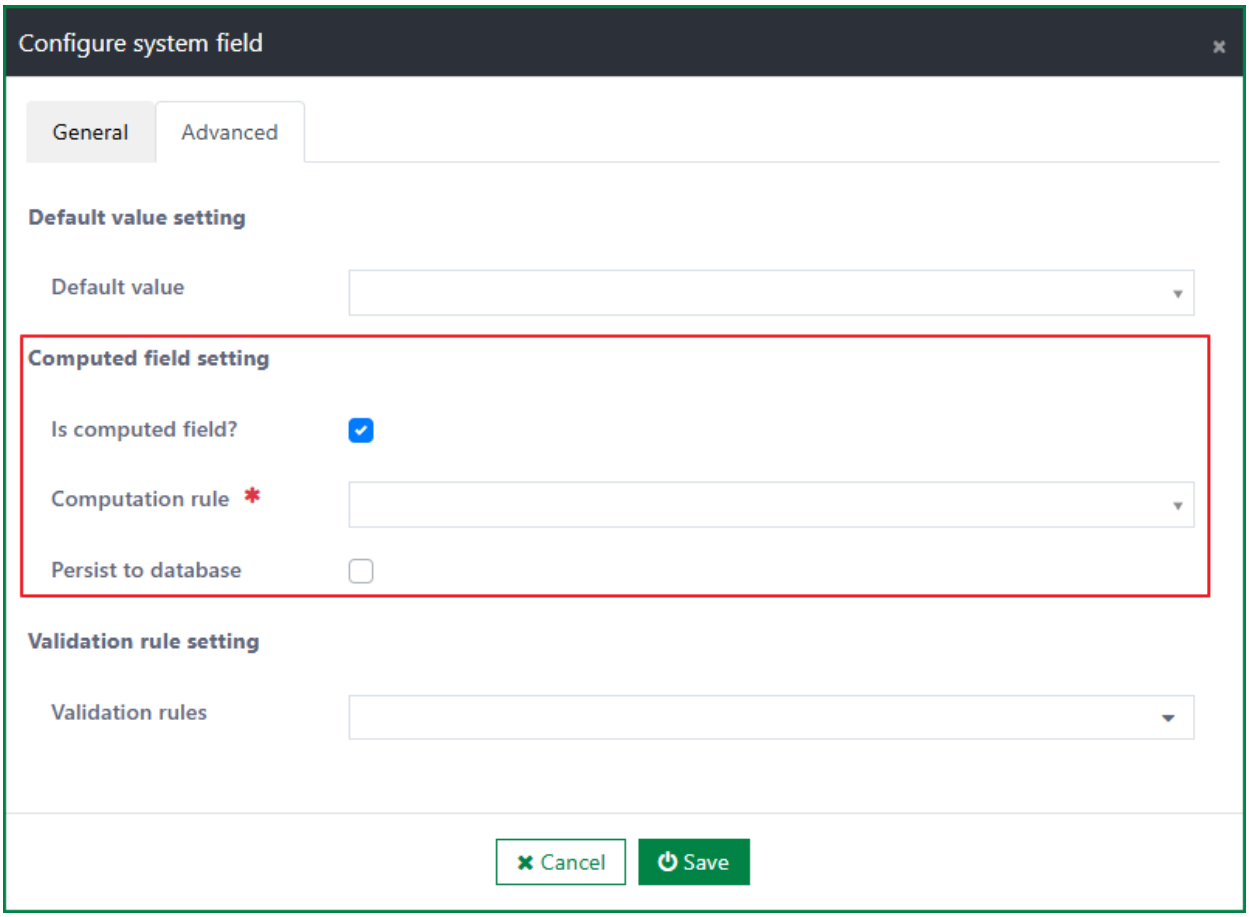
To all data types, the Advanced tab includes the rule settings for the field like: **Default value setting**, **Computed field setting**, and **Validation rule setting**. Besides that, to several data types including **Array, Enum,** and **Foreign key**, the tab also includes the **Related setting** section showing the specific information of the data type.

- **Rule settings in the Advanced tab**
  - ***Default value setting***



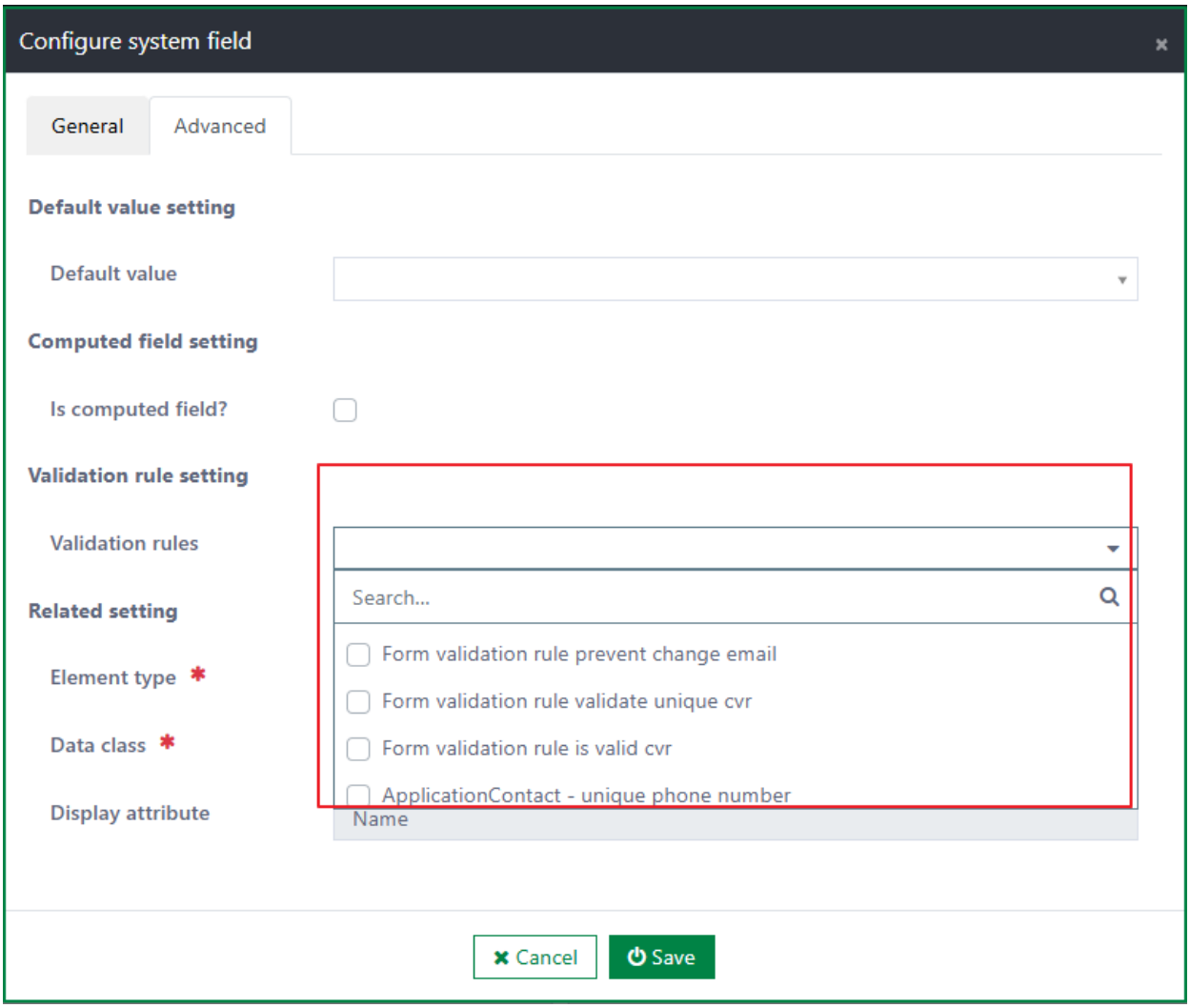
- This setting is for selecting a rule to calculate and indicate the default value of the field in the submission form. *For example*: In submission form, the field **Email** must be pre-filled with the email address in master data of the applicant as default value.
- The section includes the following field:
  - **Default value**
    - The field includes a list of active rules with rule type **Form default value** and configured in the **Rules** page.
    - You can select a rule from the list, then in the submission form, the field will be pre-filled with the default value which is calculated according to the selected rule.

- ***Computed field setting***



- This setting is for selecting the rule to calculate value of the field based on the saved values of other fields. *For example*: In the receipt form of online course registration, value of the field **Total cost** must be calculated from the filled values of the fields like: **Tuition**, **Course materials**, **Exam fee**, and **Registration fee**.
- The setting section includes the following fields:
  - **Is computed field?**
    - When you ticks on this checkbox, the additional fields likes **Computation rule**, and **Persist to database** are displayed.
  - **Computation rule**
    - The field includes a list of active rules with rule type **Form computed field** and configured in the **Rules** page.
    - You can select a rule from the list.
    - In the submission form, the field will be auto-filled with the value which is calculated according to the selected rule.
  - **Persist to database**
    - If this checkbox is ticked, the calculated value of the field on the form will be for viewing only and will not be saved to the database.

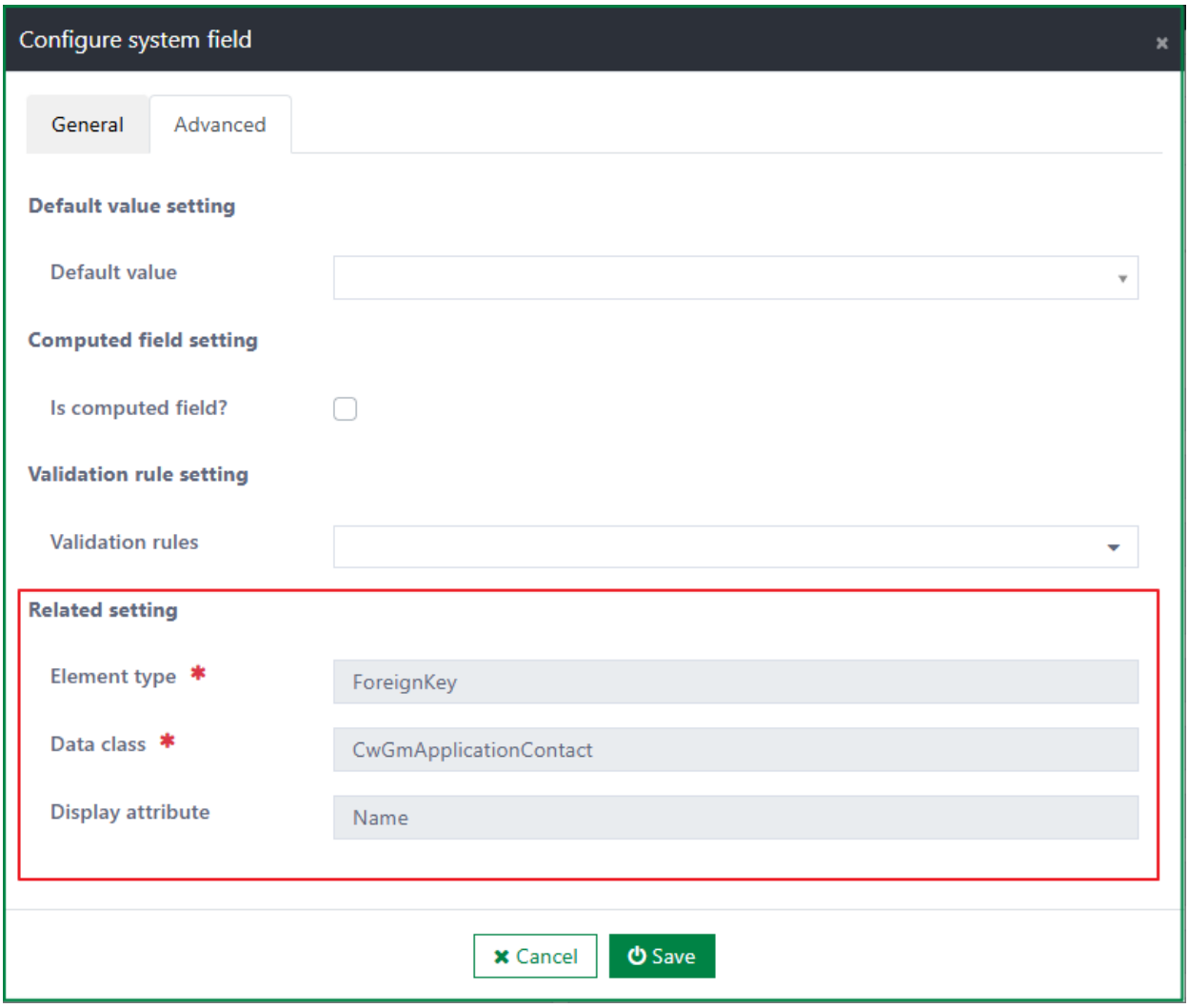
- ***Validation rule setting***



- This setting is for selecting the rule for the specific validations of the field. *For example*: In submission form, the input value of the field **CVR number** must be unique, have correct format, and exist in the CVR data.
- The setting section includes the following fields:
  - **Validation rules**
    - The field includes a list of active rules with rule type **Form validation rule**
    - You can select multiple rules from the list.
    - In the submission form, when you fills the field, the validations will be executed according to the selected rules.

- **Specific information for data types Array, Enum, and Foreign key:**

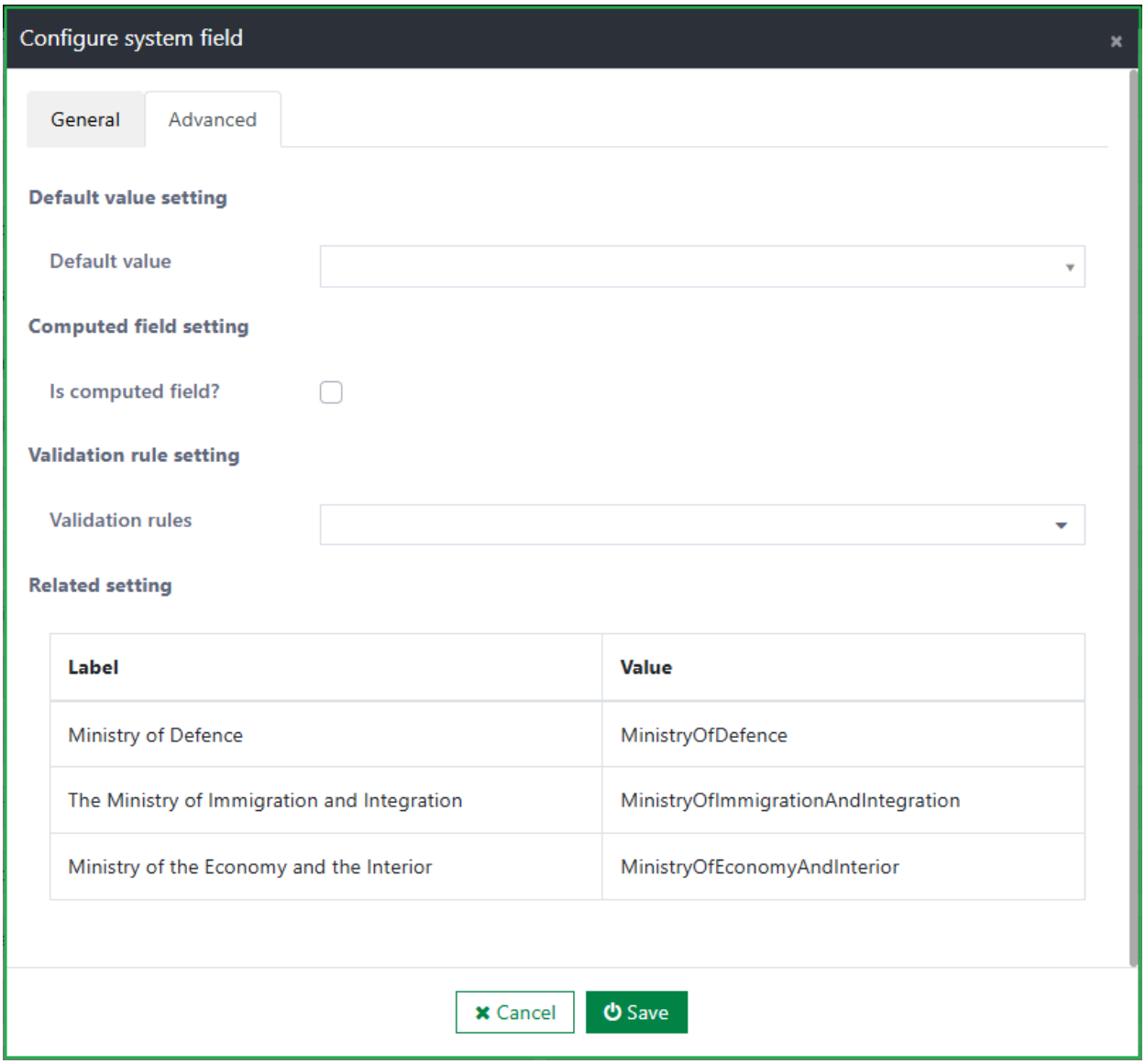
  - ***Array* data type*



In the **Advanced** tab of this data type, the **Related setting** section includes the following fields with pre-filled value and the value cannot be changed:

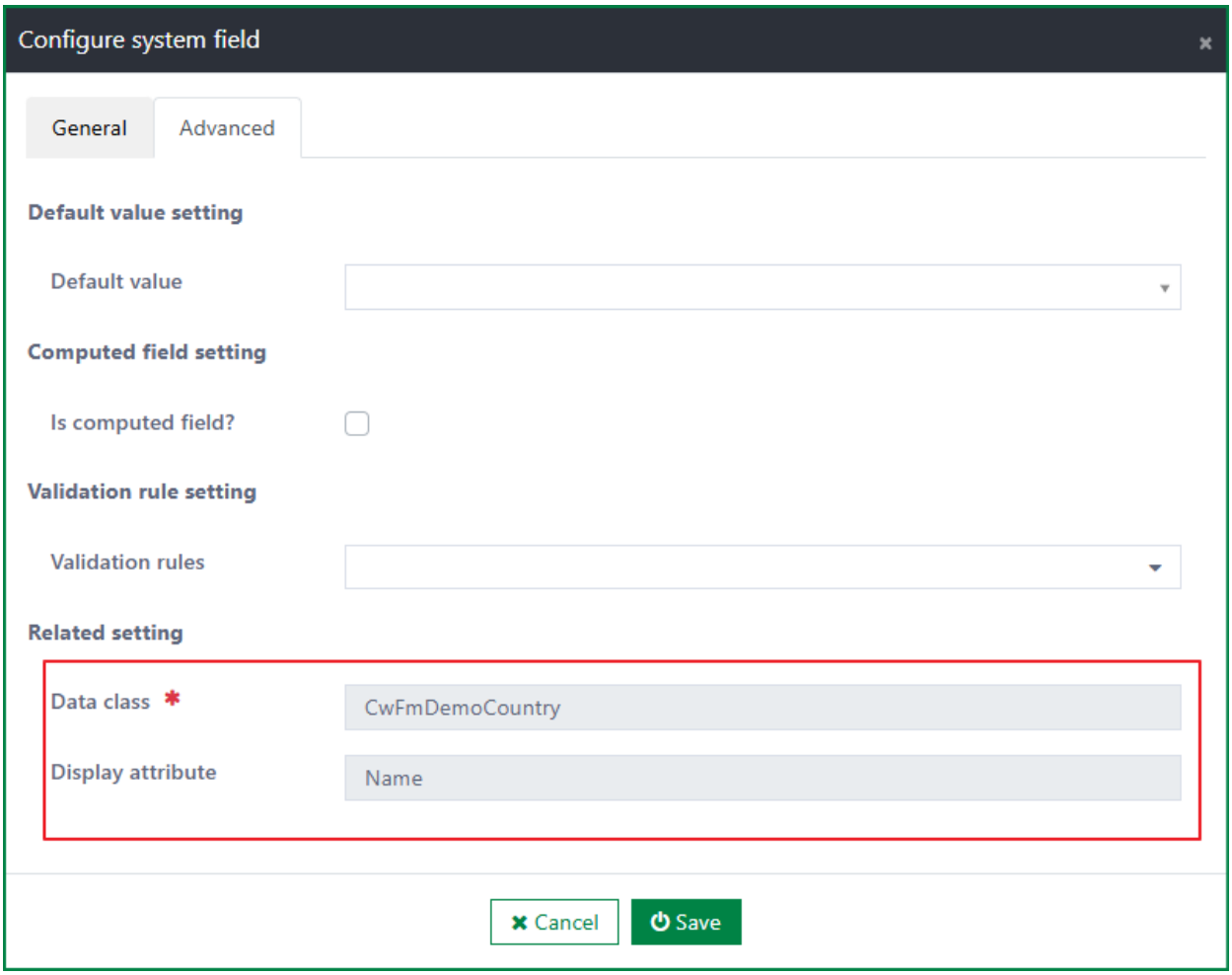
- **Element type**
- **Data class**
- **Display attribute**

- ***Enum*** data type*



    - In the **Advanced** tab of this data type, the **Related setting** section includes a list of pre-defined **Enum** values.

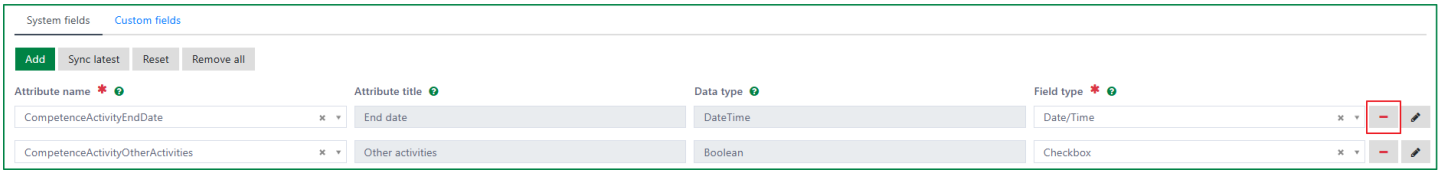
- **_Foreign Key_* data type*



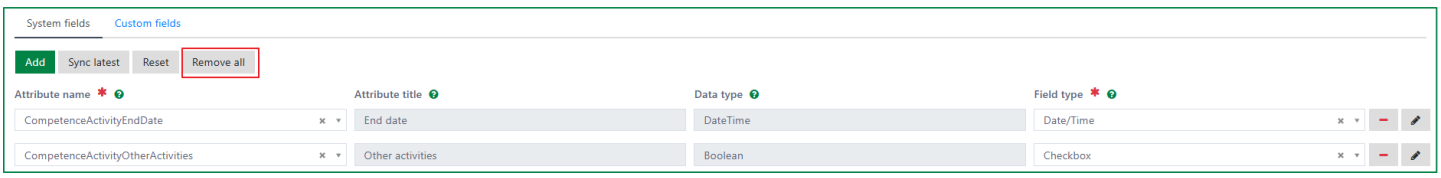
- In the **Advanced** tab of this data type, the **Related setting** section includes the following fields with pre-filled value and the value cannot be changed:
  - **Data class**
  - **Display attribute**

Remove system fields from the form data source

- **Remove each system field**
  - You can remove system field one by one by clicking on the **Remove** button at the field row.
  - After being removed, the field will be hidden from the **System fields** list. To add the field again, you must add it manually.
  - The field is only removed from the form data source and will not be removed from Casewhere data source.

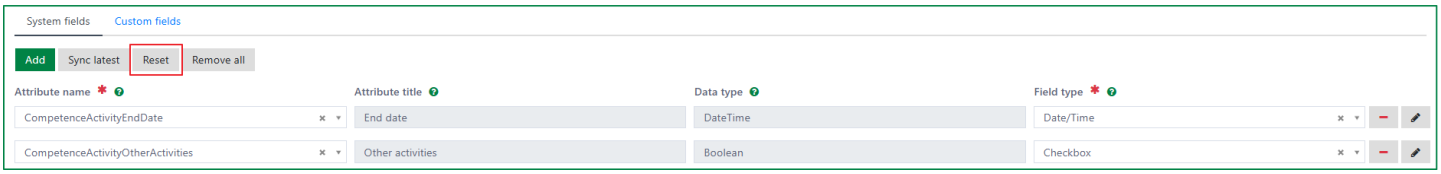


- **Remove all fields**
  - You can remove all fields at once by clicking on the **Remove all** button.



Reset all changes made on the system fields list

By clicking on the **Reset** button, all system fields (except for the Casewhere built-in ones) under the selected data source will be displayed in **System fields** list.



Custom fields

Besides the list of system fields retrieved from CW data source, you can also add custom fields to be used in building the form.

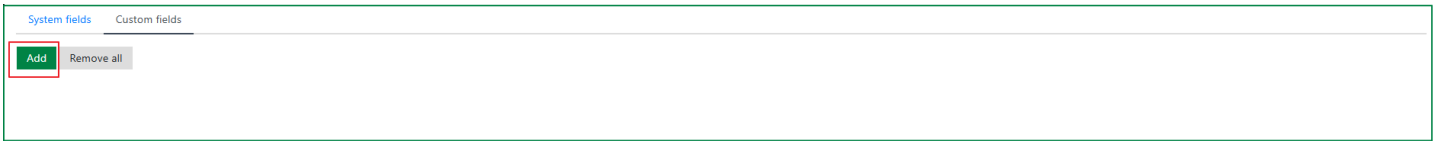
The **Custom fields** list includes all created custom fields under the selected data source.

The custom fields are displayed with the information, like: **Attribute name**, **Attribute title**, **Container**, **Data type**, and **Field type**.

The available actions in **Custom fields** section are:

- Add new custom field
- Edit custom field
- Configure additional information of the custom fields
- Remove custom fields

Add new custom field



By clicking on the **Add** button, new row of the custom field will be displayed in the list of custom fields with the following fields:



- **Attribute name**
  - This field is required.
  - The **Attribute name** must be unique.
  - The **Attribute name** must not contain spaces and special characters.
- **Attribute title**
  - This field is required.
  - The **Attribute** title is used as displayed name for the field in building form.
- **Container**
  - This field is required.
  - The field is a list of container which is the field with **Object** data type belonging to the selected data source. You must select an item from the list.
  - The submitted data of the custom fields via form submission will be stored in the container.
- **Data type**
  - This field is required.
  - You must select a data type for the custom field.
  - List of the data types is pre-defined with 12 types. Those are: **Array, Boolean, DateTime, Document, Double, Enum, ForeignKey, Guid, Integer, Long, Object,** and **String**.

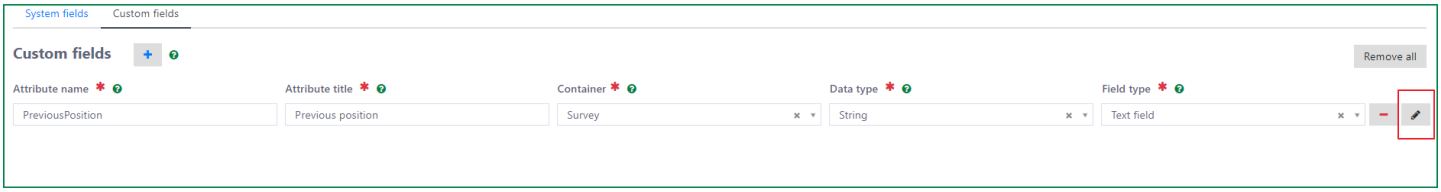
- **Field type**
    - This field is required.
    - You must select a field type that works properly with the selected data type of the custom field.
    - List of the field types is pre-defined with 17 field types. Those are: **Checkbox**, **Text field**, **Text area**, **Number**, **Password**, **Select box**, **Select**, **Radio button**, **Email**, **Url**, **Phone number**, **Date/Time**, **Day**, **Currency**, **File**, **Survey**, **Data grid**, **Data map**, and **Multiple text fields**.

Edit custom field

You can edit the following fields of a custom field directly in the **Custom field**s list:

- **Attribute name**
- **Attribute title**
- **Container**
- **Data type**
- **Field type**

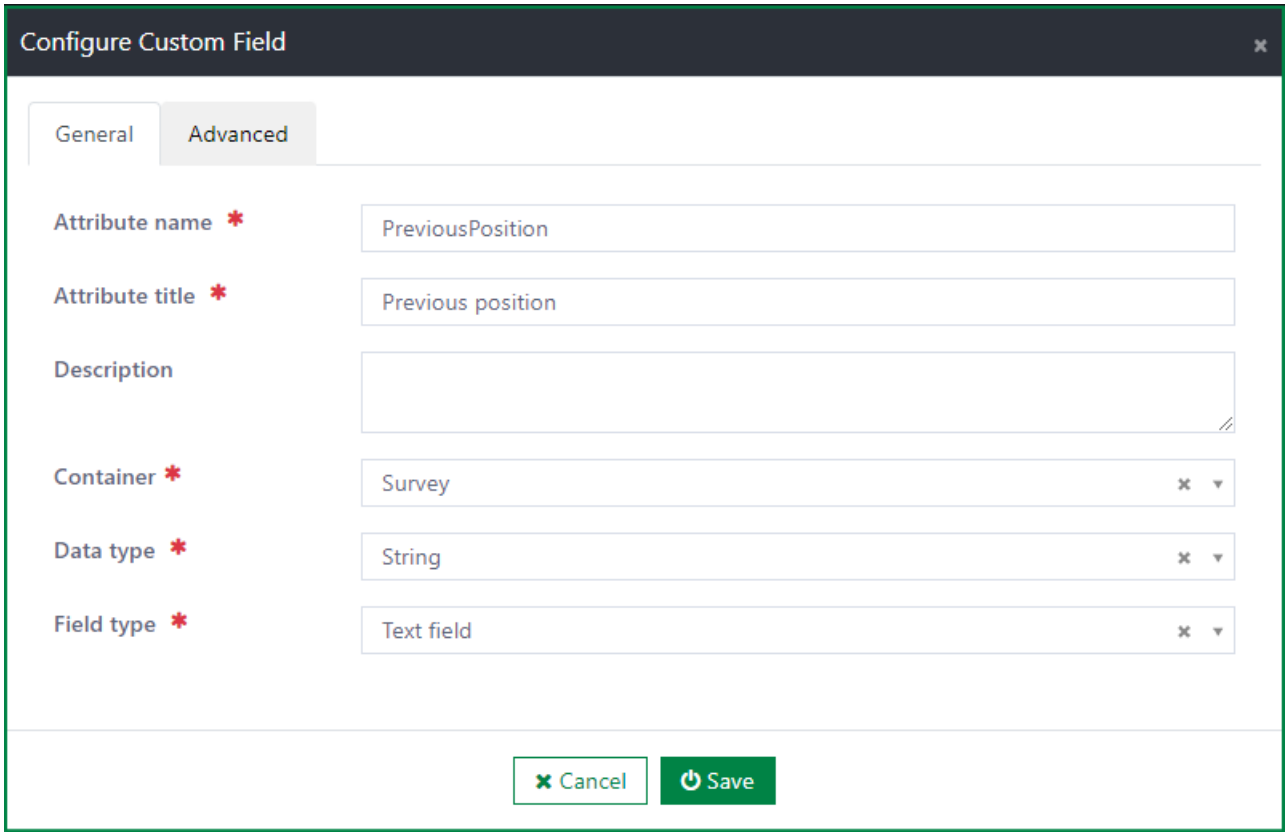
Configure additional information of the custom fields



By clicking on the **Edit** button at each field row, the **Configure custom field** dialog will be opened.

The dialog includes 2 tabs: **General**, and **Advanced**.

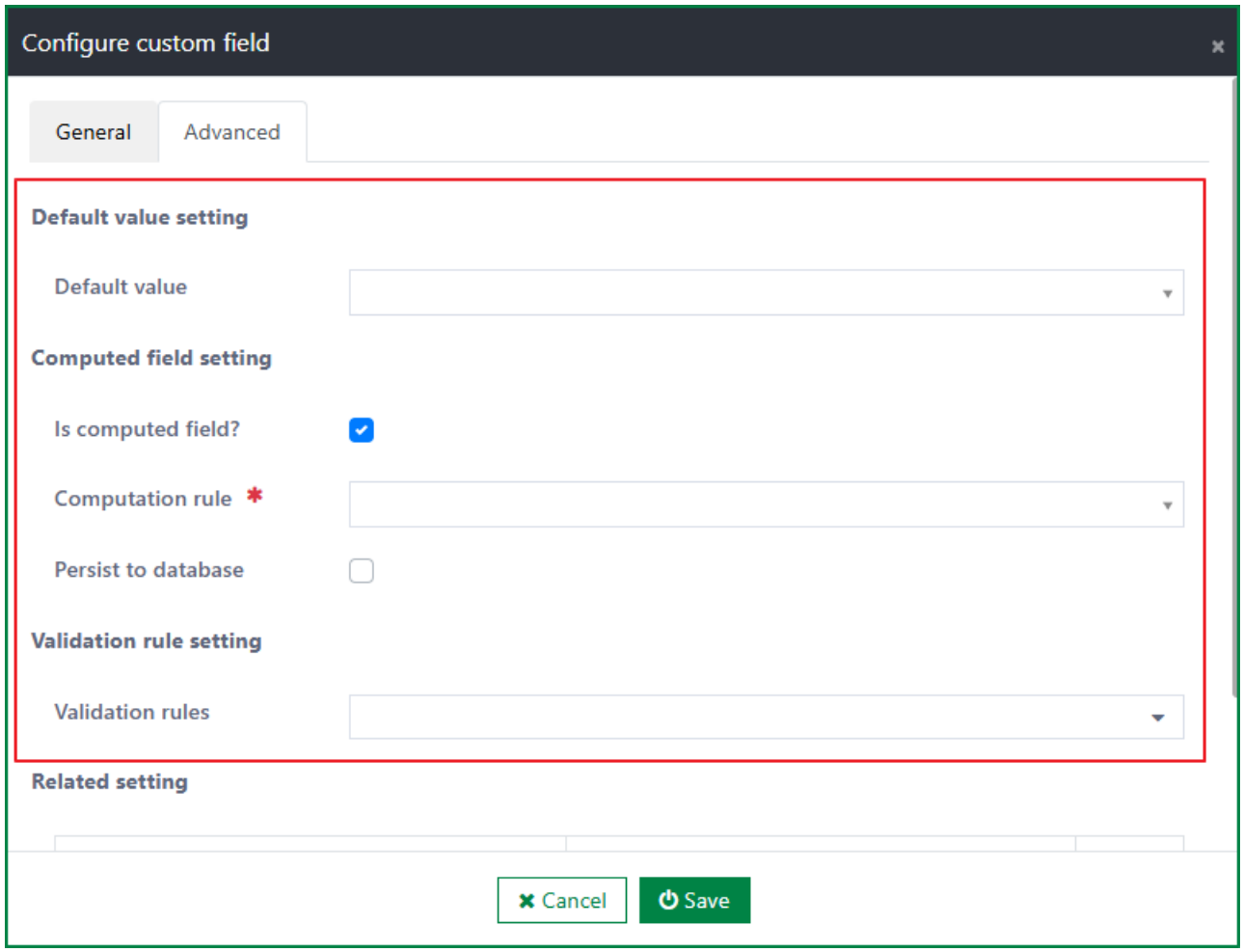
- **General** tab



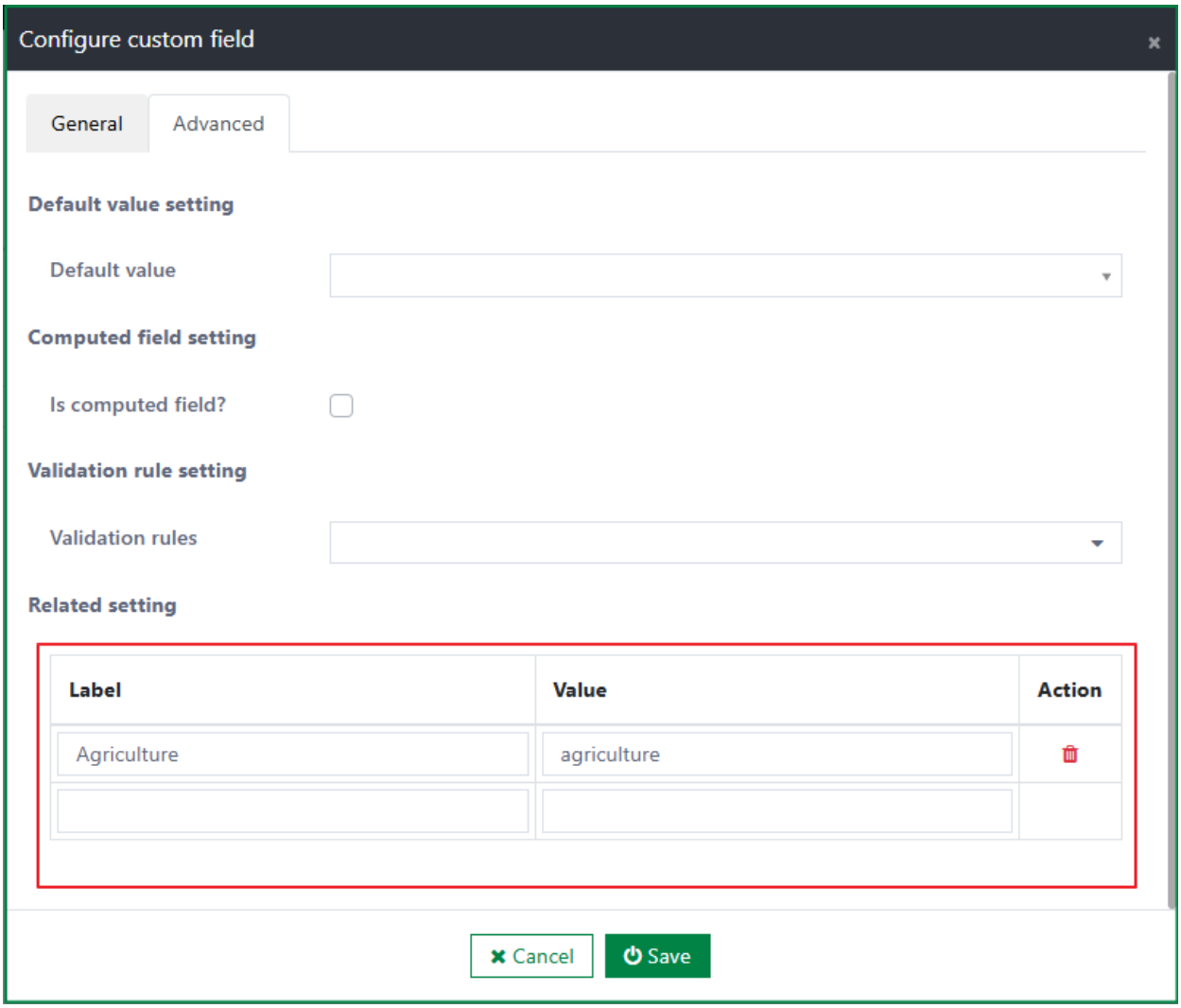
    - The **General** tab is available for all data types.
    - The tab includes the basic information of the custom field similarly to the **Custom fields** list, like: A**ttribute name, Attribute title, Container, Data type, Field type**. All these fields are editable.

- Besides that, you can also describe more about the custom field via the **Description** field.

- **Advanced** tab
  - Similarly to the system field, the Advance tab of the custom field with all data types includes the rule settings like: **Default value setting**, **Computed field setting**, and **Validation rule setting**.

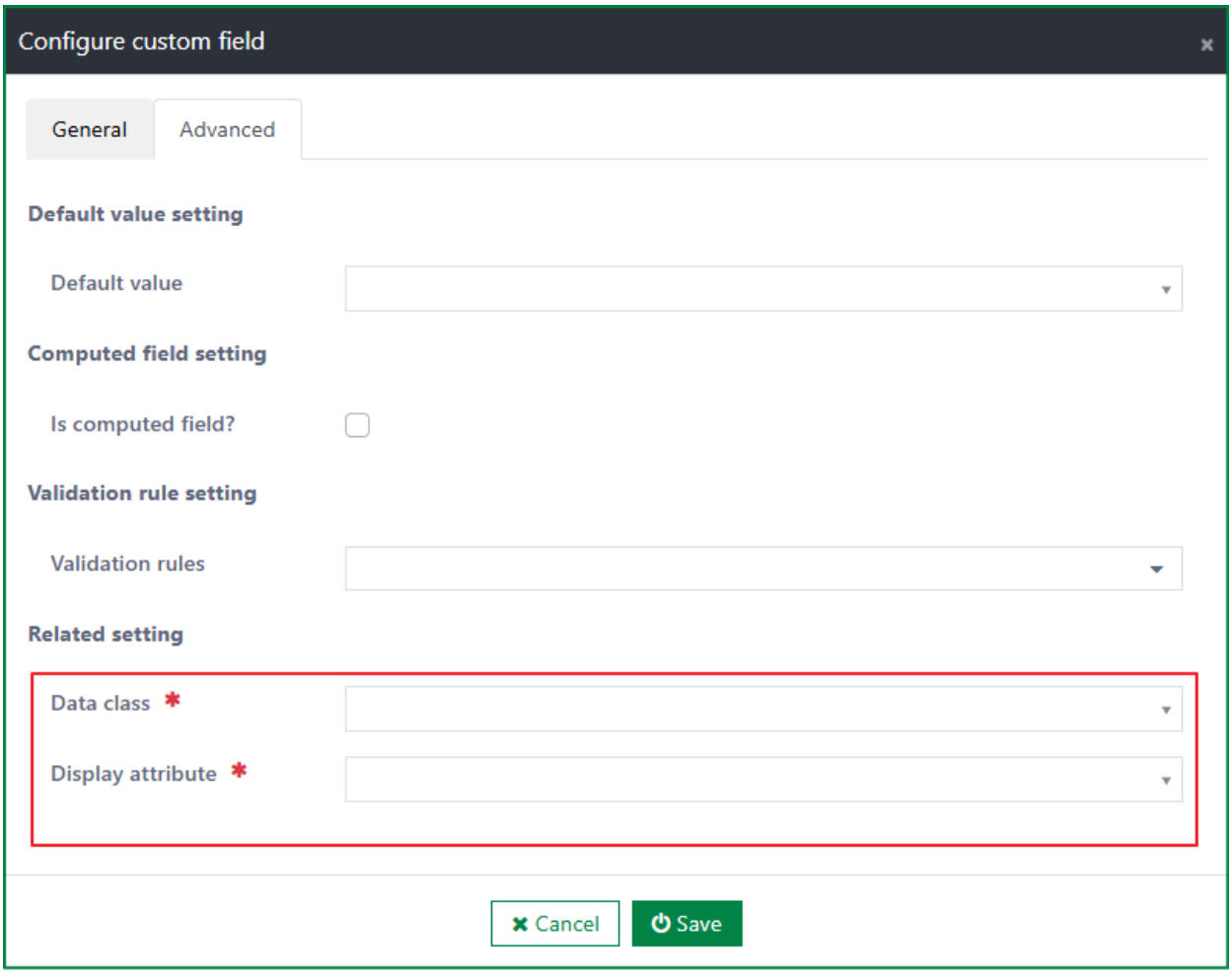


  - The tab also contains the **Related setting** section for some specific data types including **Array**, **Enum**, and **Foreign Key**.
    - **Enum data** type



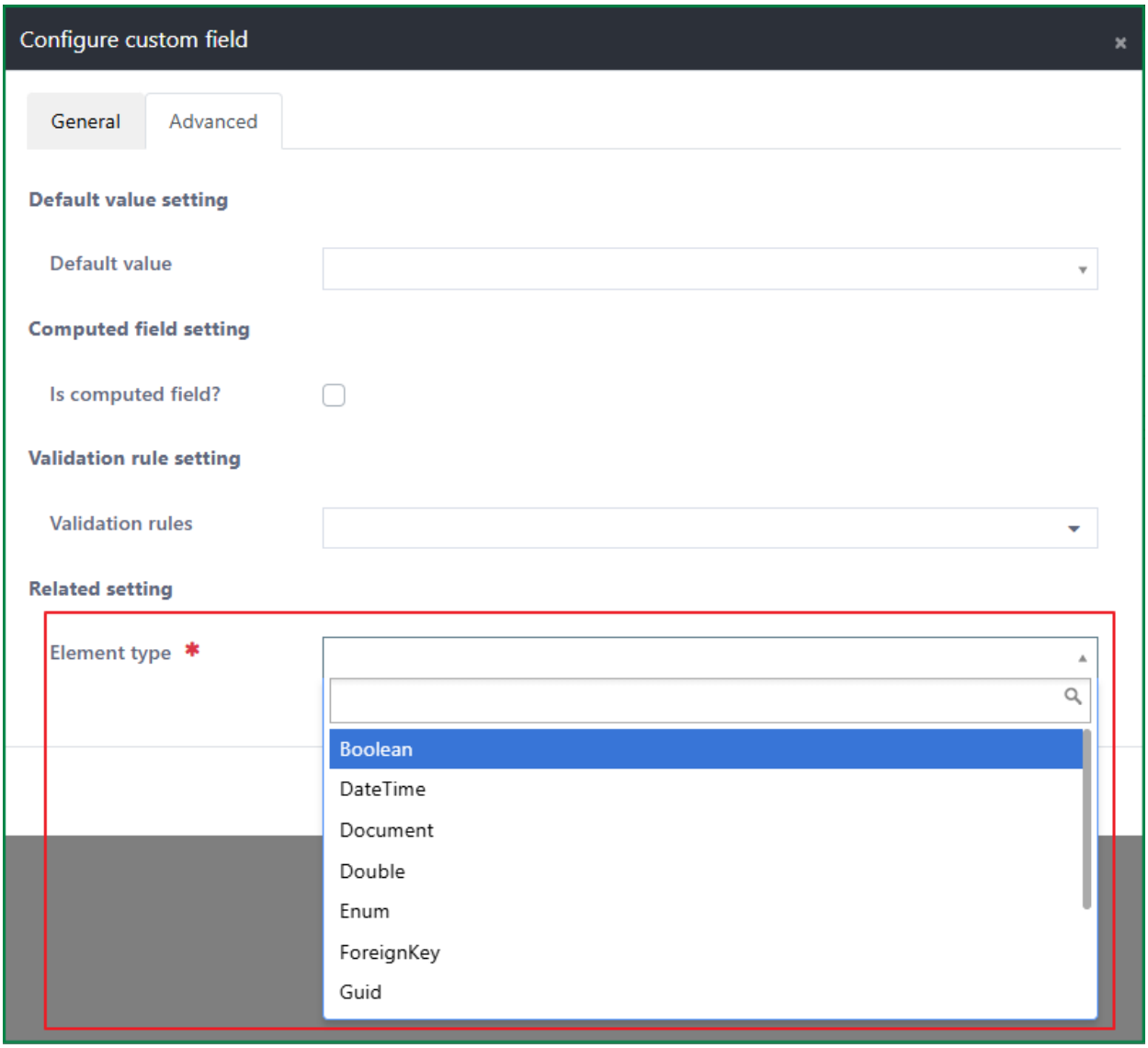
- In case data type of the custom field is Enum, in the **Related setting** section, you must define the set of **Enum** values.
- The set of values is a list with 2 columns to input the value for: **Label**, and **Value**.
- **Label** is used as displayed name of the Enum value on UI of the rendered form.
- **Value** is used as saved data in backend. Value must not contain spaces, and special characters.
- For example, it is a field called **Industry** in an application form. The field is a dropdown list field with values, like: **Agriculture, Resource, Engineering and high-tech**. So, when configuring the custom field, the field should have **Enum** data type, **Select** field type. And in advanced settings, it should have the **Enum** values as the following:
  - (Label) Agriculture, (Value) agriculture
  - (Label) Resource, (Value) resource
  - (Label) Engineering and high-tech, (Value) engineeringAndHightech

- **Foreign Key** data type



- In case the data type of the custom field is Enum, you must define the following data for the field:
  - **Data class**
    - You must select a data class from the list. List of data classes is retrieved from the **Data class** page in admin site,
  - **Display attribute**
    - You must select an attribute from the list. The list includes all attributes belonging to the data class. List of attributes is only enabled after the data class is selected.

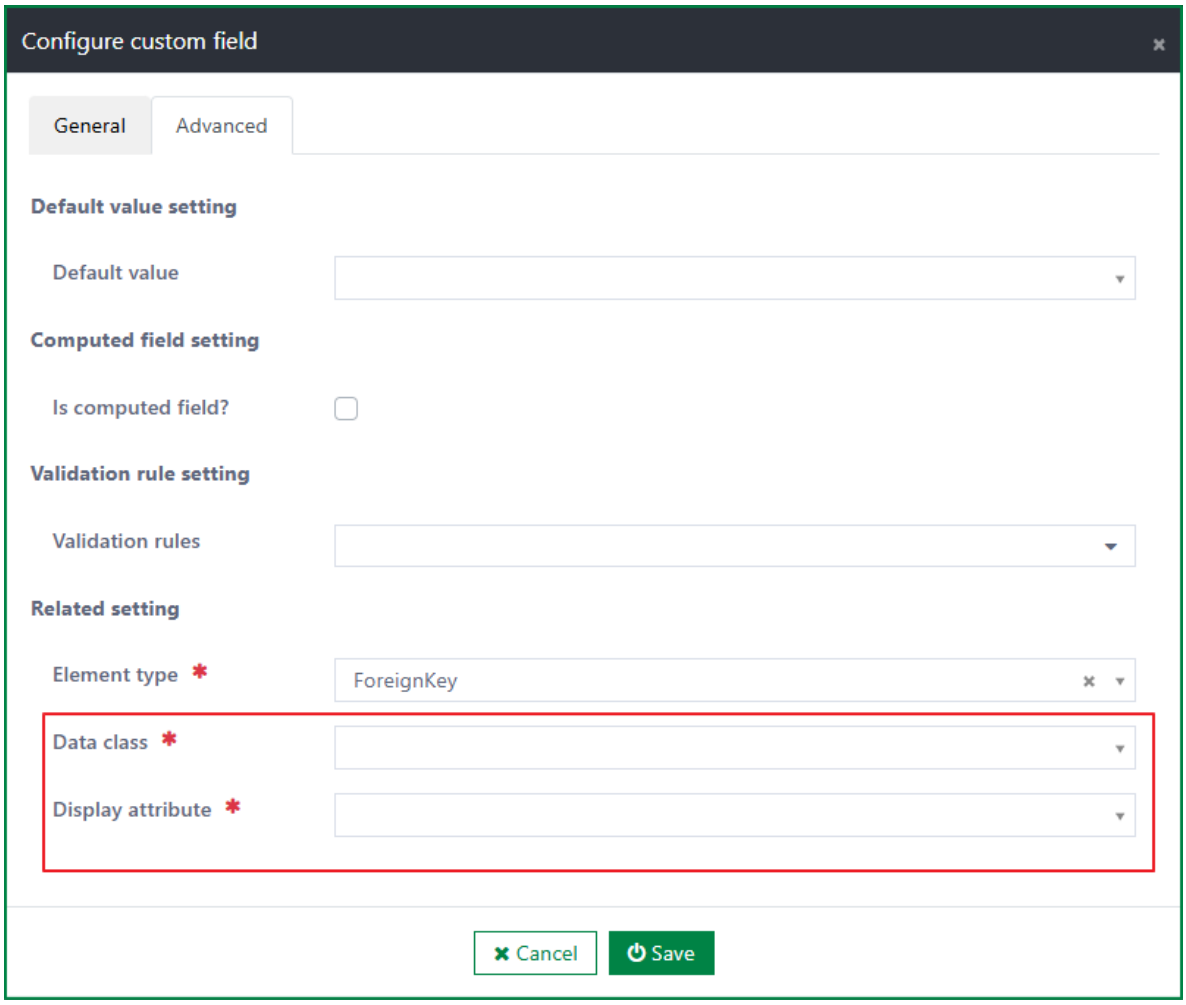
- **Array** data type



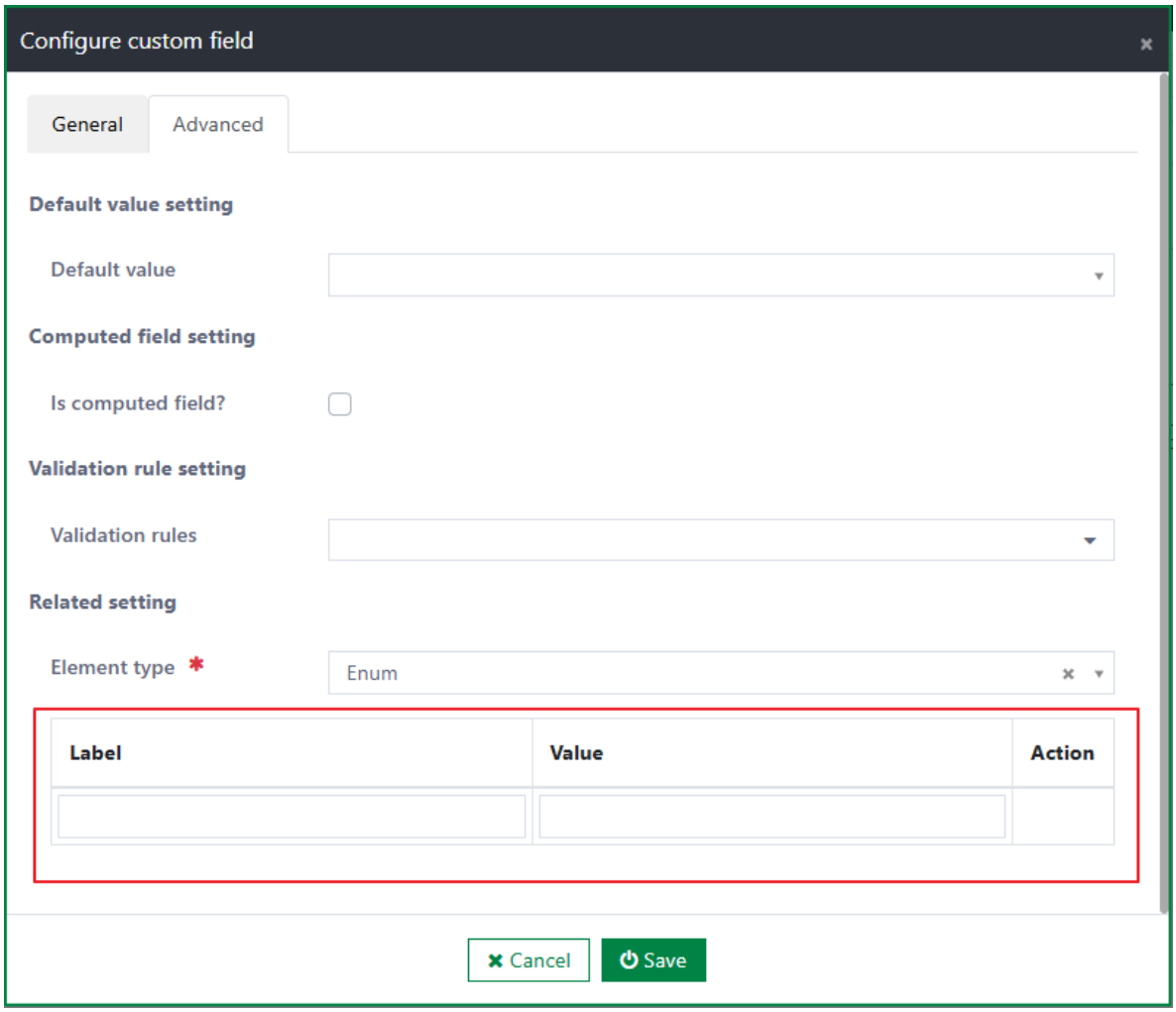
- To the **Array** data type, the **Related setting** section includes the field **Element type** where you can select data type for the elements of the array from the list of 11 data types.

  For example: it is a document field in a form which allows uploading multiple files. So, the data type of the custom field must be **Array**, and its element data type must be **Document**.

- If the **Element type** is **Foreign key**, you must select the **Data class** and **Display attribute** for the element.

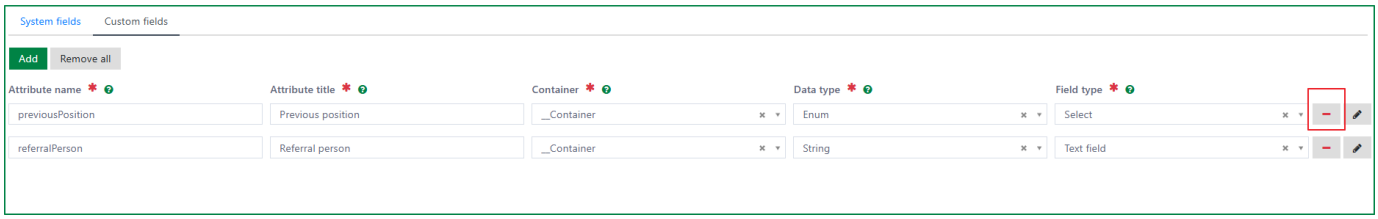


- If the **Element type** is **Enum**, you must define a set of **Enum** values for the element.



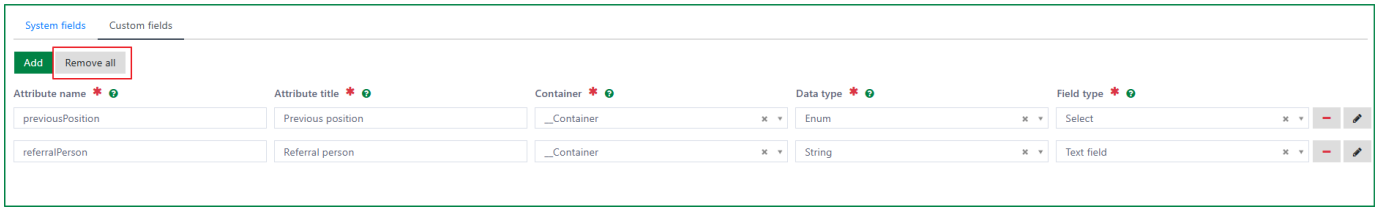
Remove custom fields

- **Remove each field**



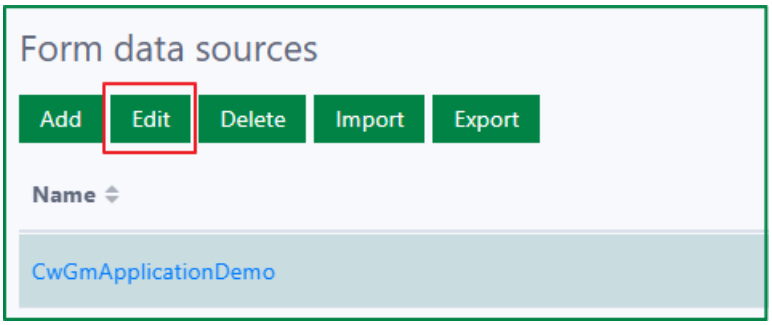
  - You can remove each custom field by clicking on the **Remove** button at the field row.
  - After being removed, the field will be hidden from the **Custom fields** list. The removed custom field will never be able to use again, however, if it was any submitted data for the field, the data is still stored in the relevant container.

- **Remove all custom fields**

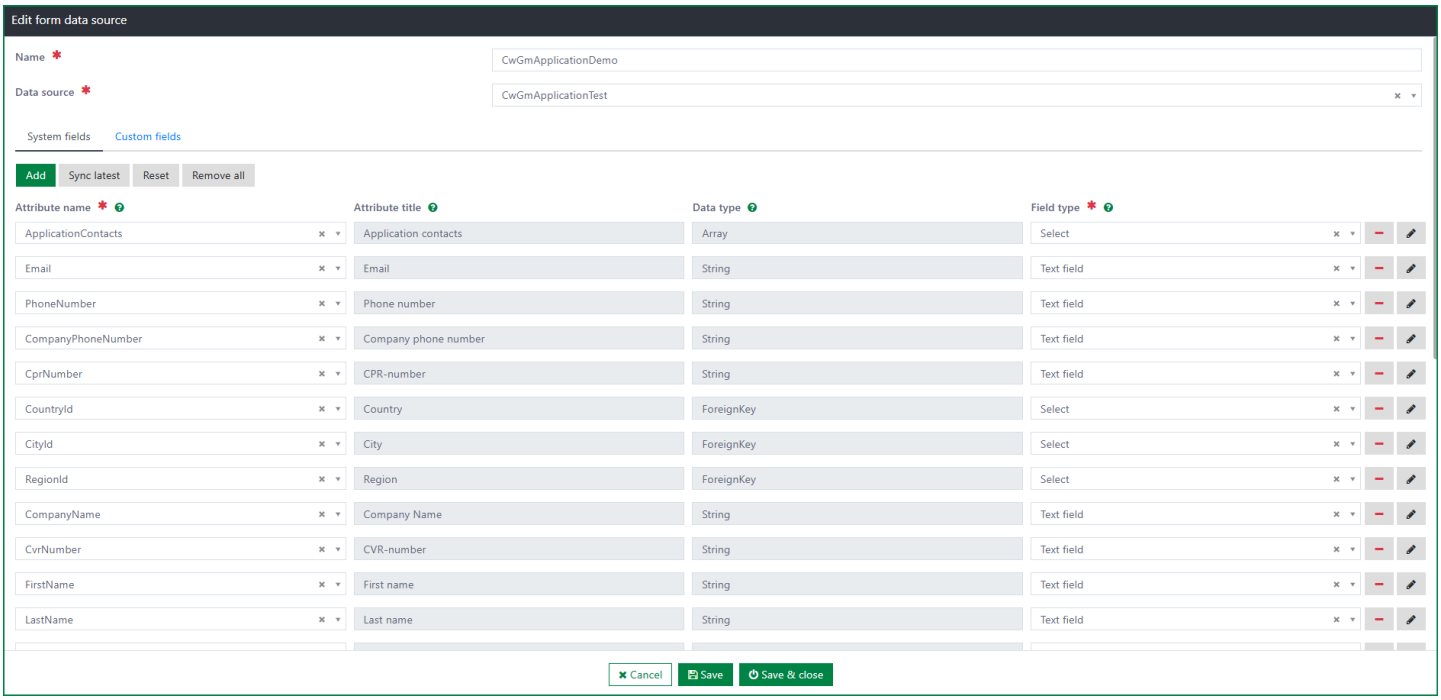


  - You can also remove all fields at once by clicking on the **Remove all** button.

Edit form data sources



By clicking on the **Edit** button, the **Edit form data source** dialog will be opened.

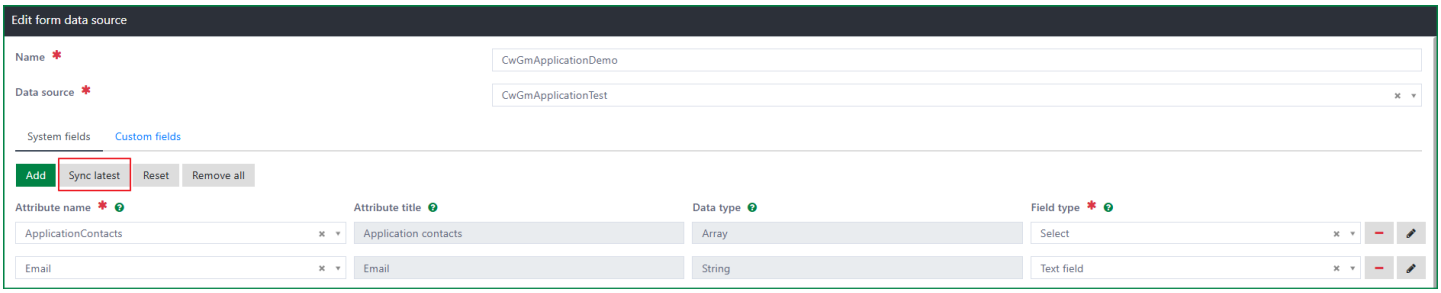


The dialog includes 3 parts:

- Configuration section for Name of form data source, Casewhere data source selection
- Data source system fields
  - In this section, the available actions are:
    - Add system field from Casewhere data source
    - Edit field type for the system field
    - Configure additional information for the system field
    - Remove one/all system fields from the form data source
    - Reset the list of system fields
    - Sync latest changes on system fields from the Casewhere data source
- Data source custom fields
  - In this section, the available actions are:
    - Add new custom field
    - Edit custom field
    - Configure additional information for the custom field
    - Remove one/all custom fields from the form data source

Most of the actions are similar to the corresponding actions in the flow of adding new form data source. Only the **Sync latest** action is different, you can only execute the syncing action when editing the form data source. The function of syncing changes will be mentioned in the following section.

Sync latest changes on system fields from the CW data source



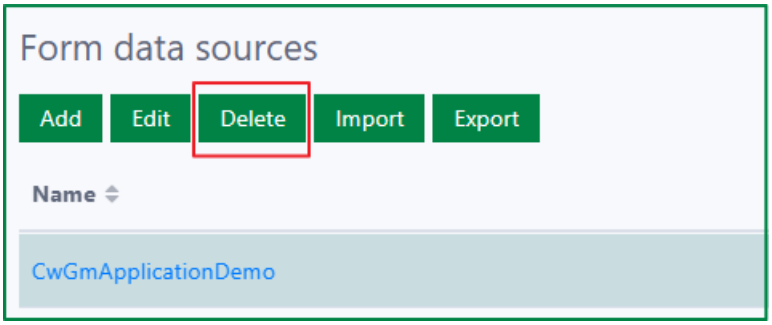
By clicking on the **Sync latest** button, the syncing between Casewhere data source and the form data source is executed based on the **Attribute name**.

The syncing rules are:

- If the attribute name exists in Casewhere data source and has not been added to the form data source yet, after syncing, the system field with the corresponding attribute name will be added to the Form data source.
- If the attribute name no longer exists in the Casewhere data source and the relevant system field exists in the form data source, after syncing, the field will be removed from the form data source.
- If the attribute name exists in Casewhere data source and the related system field was removed from the form data source, after syncing, the field will not be added to the form data source again.

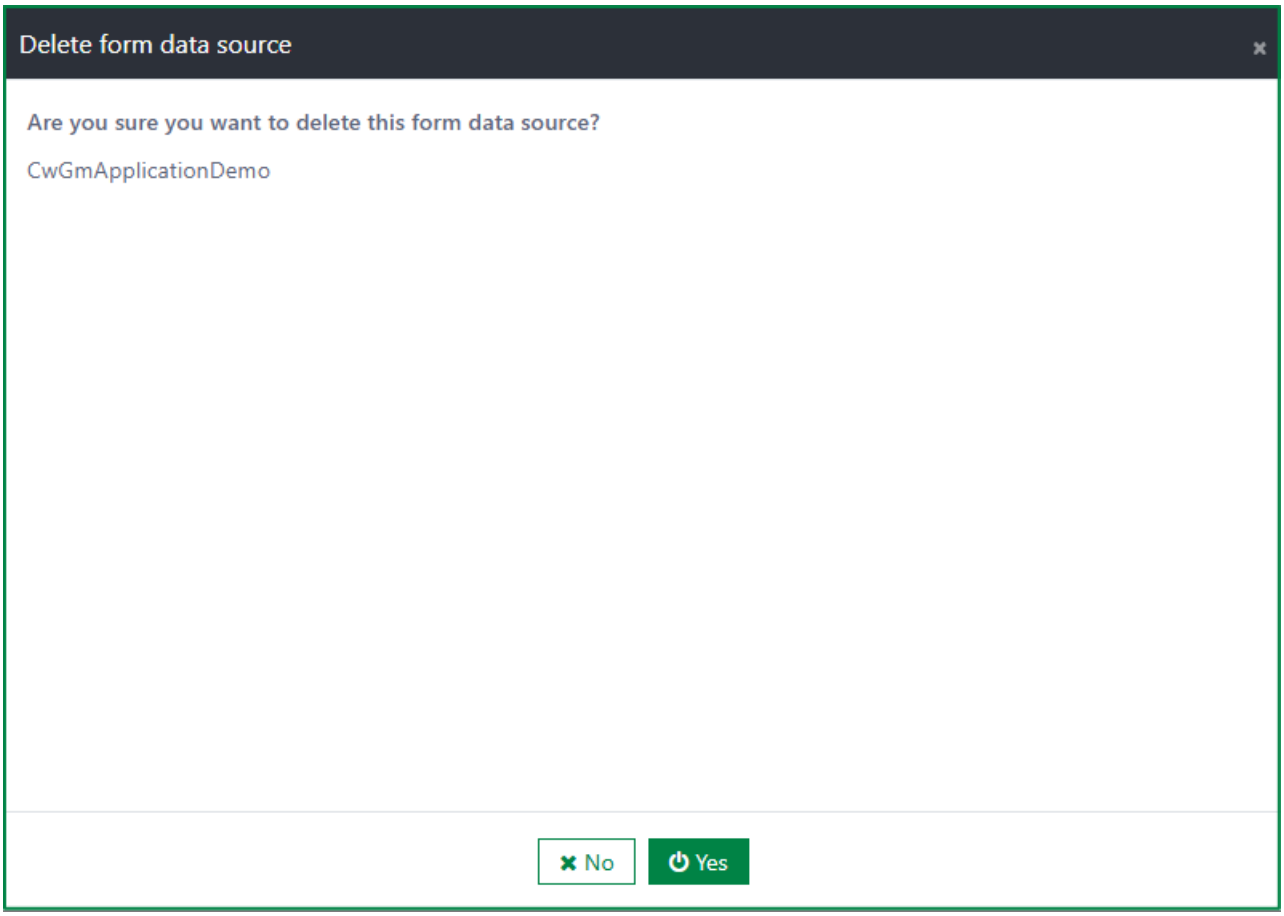
If the attribute name is changed, after syncing, the relevant system field will be added to the form data source as a new field. Meanwhile, the field with old name will be also removed from the form data source according to the 2nd rule above.

Delete form data sources



The **Delete** button is only enabled after a form data source is selected.

By clicking on on the **Delete** button, the deletion confirmation dialog will be opened.



- If you select the Cancel button, the deletion will be cancelled.
- If you select the Delete button, the validation for in-use form data source will be executed.
    - If the form data source is not used anywhere, the form data source will be deleted immediately.
    - If the form data source is in use, the deletion will be disallowed, and an error message will be displayed.

Next section

Component: Form Management - Form Actions

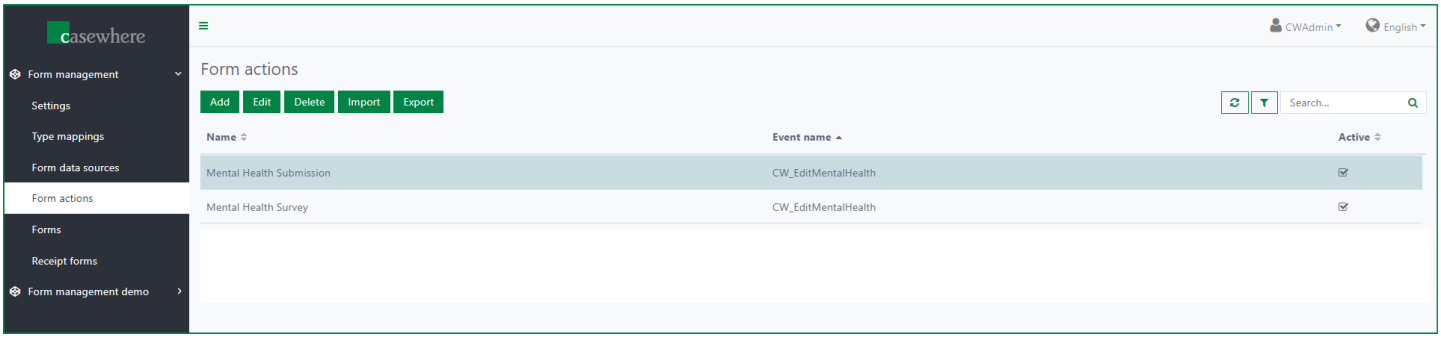
Component: Form management - Form actions

Introduction

The Form actions page is used to manage the form actions which are used in the form, e.g: submit form, save form as draft, etc.

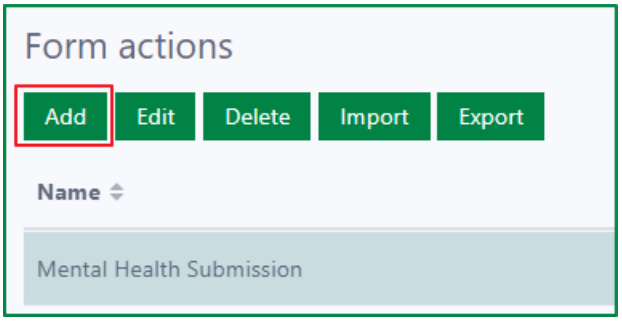
The list displays the basic information of the form actions, like: **Name**, **Event name**, **Active** status.

You can add, edit, delete the form actions . User can also import form actions from JSON file, or export it to a JSON file.

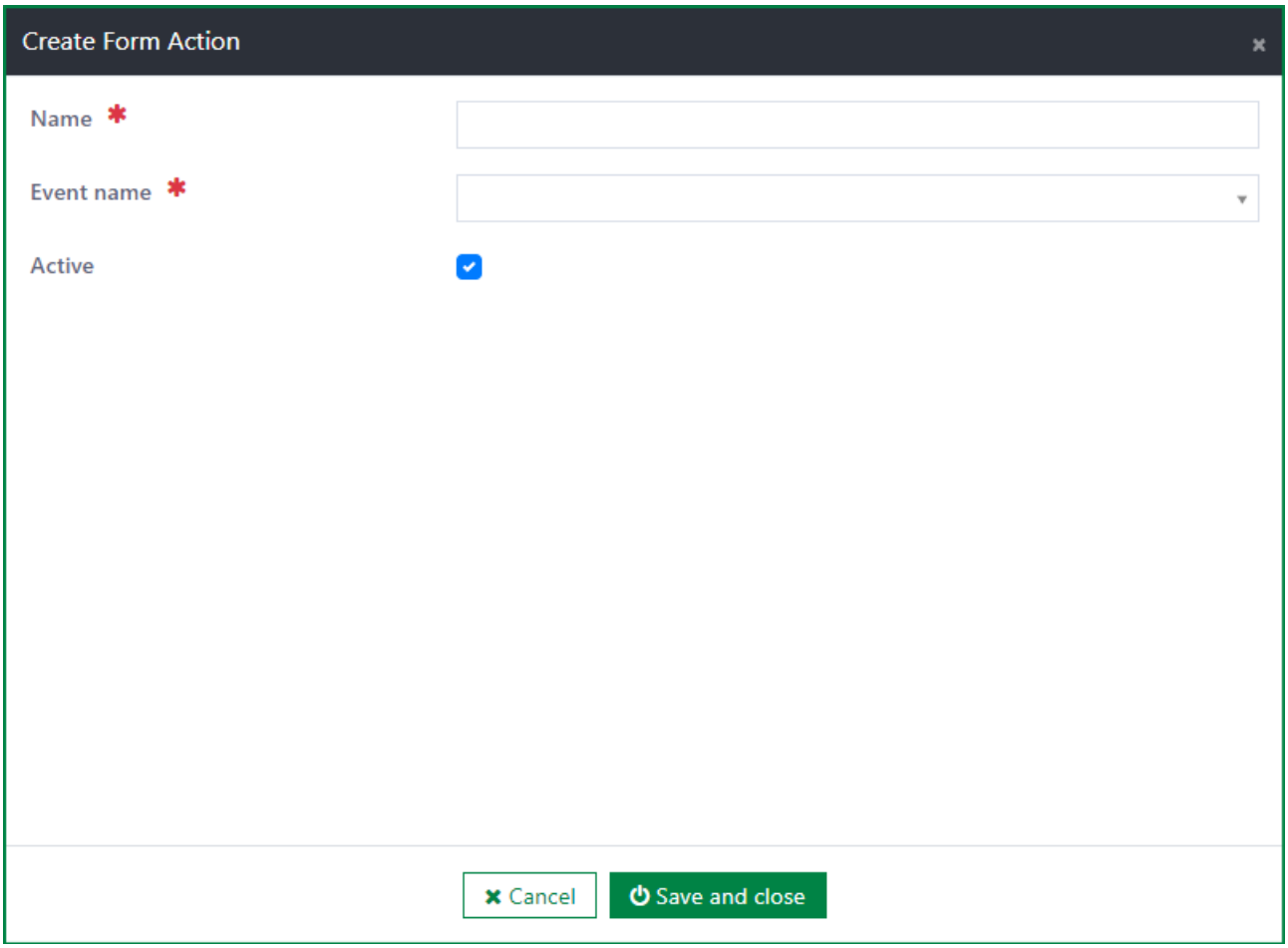


Guideline

Add form action



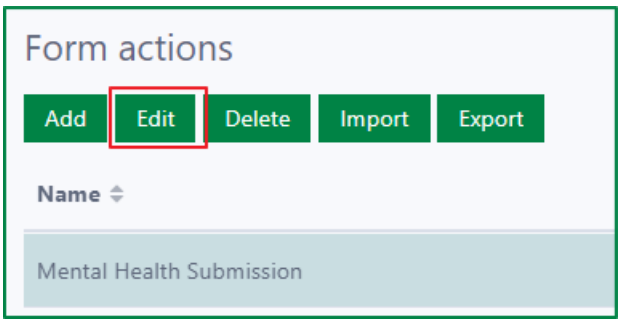
By clicking on the **Add** button, the **Add form action** dialog is opened.



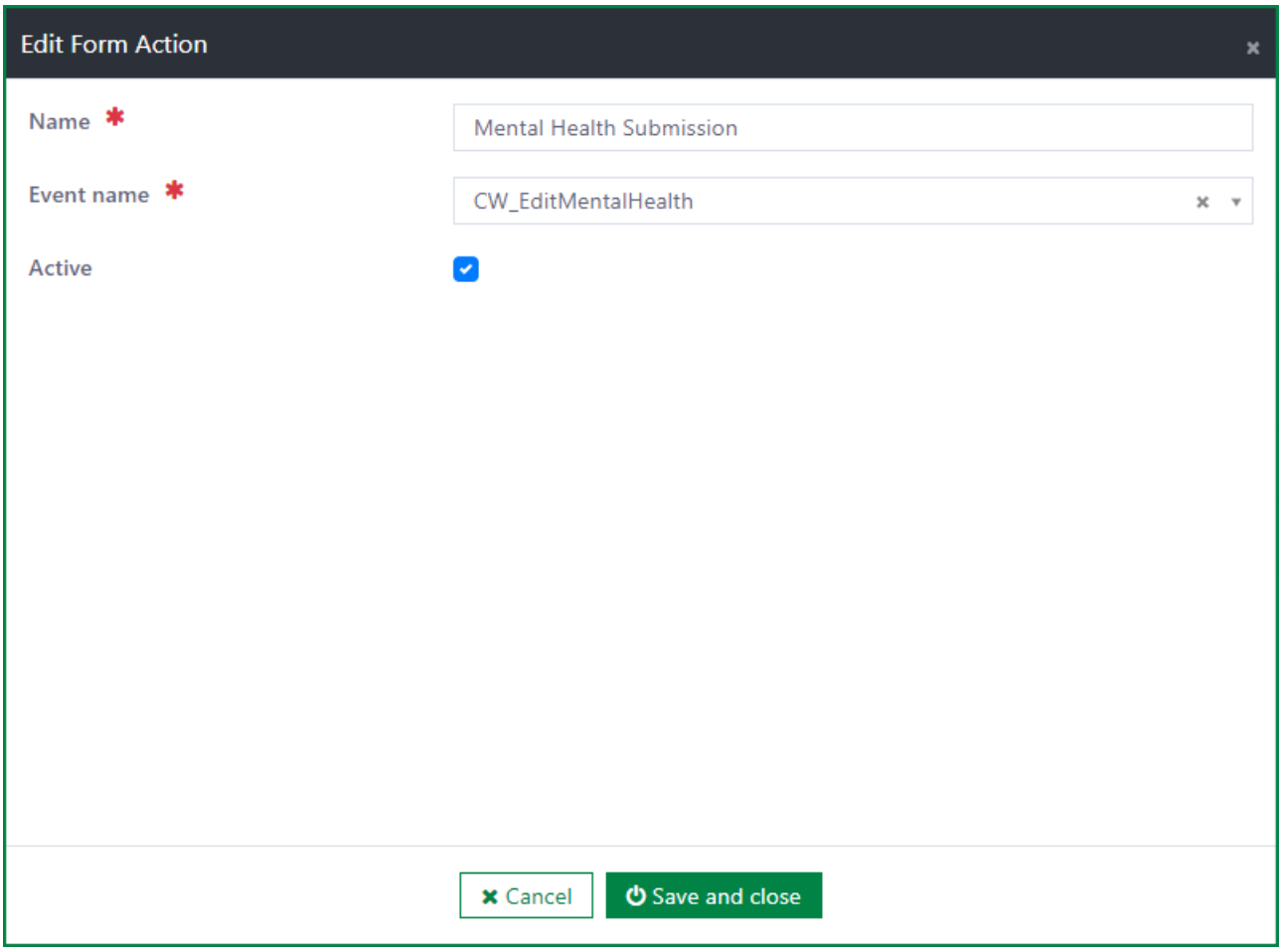
The dialog include the following fields:

- **Name**
  - This field is required.
  - The name of form action must be unique.
- **Event name**
  - This field is required.
  - User must select an event from the list.
  - List of event name were pre-defined by developers.
- **Active**
  - The form action is active by default.
  - User can deactivate it by unticking the checkbox.
  - Only the active form actions are available in the relevant settings of Form configuration.

Edit form action

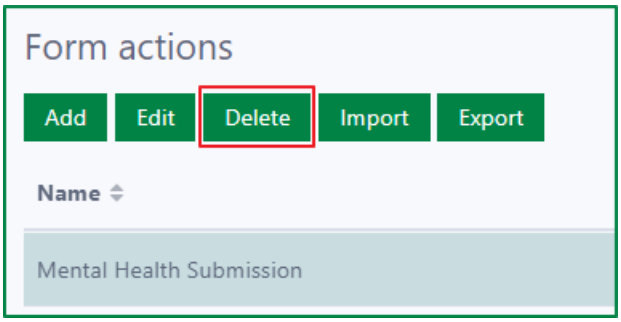


By selecting an item on the list and clicking on the **Edit** button, the **Edit form action** dialog will be opened.



The dialog displays the saved data of the form action, like: **Name, Event name,** and **Active** status. And you can update the saved data.

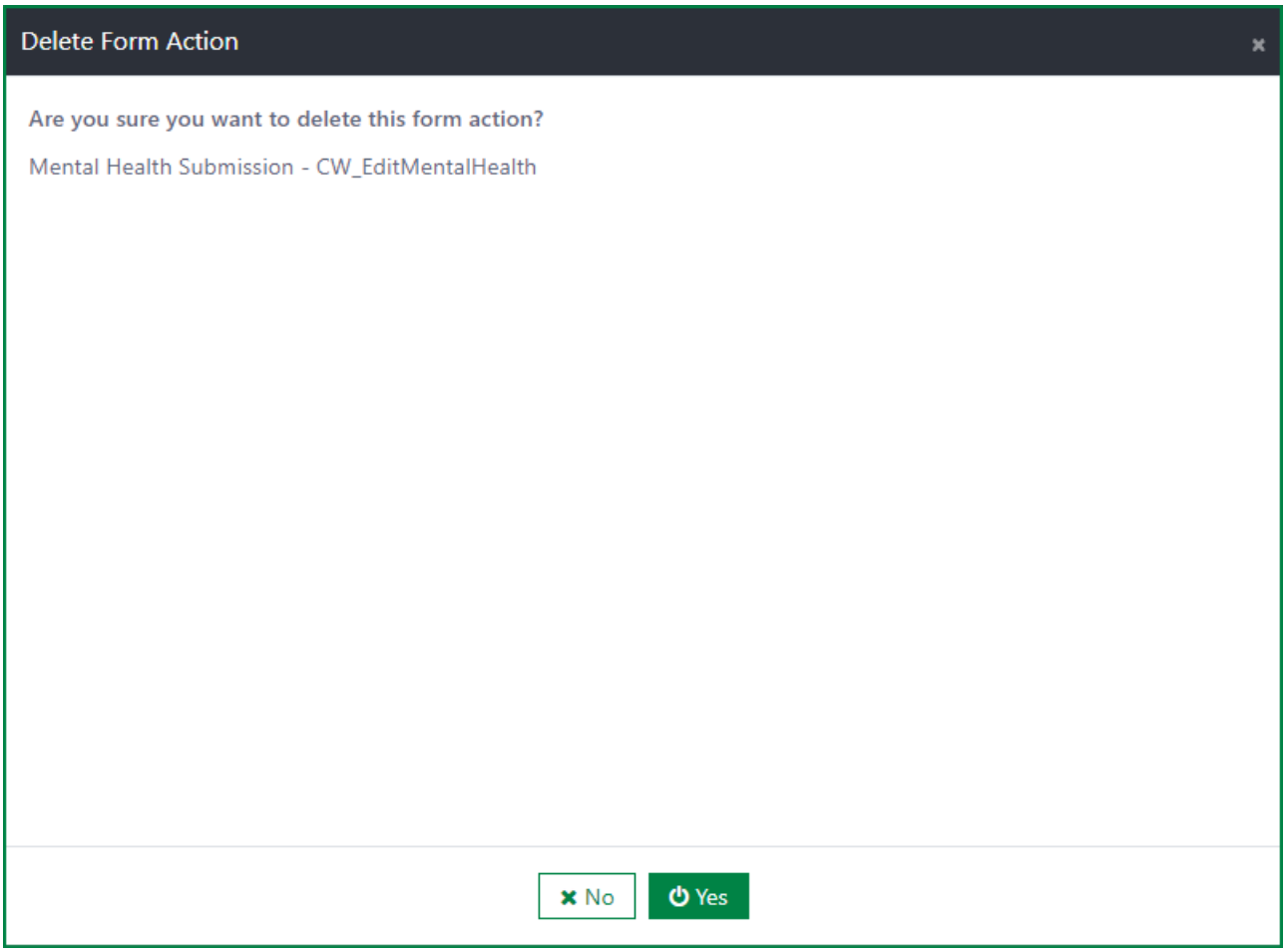
Delete form action



The **Delete** button is only enabled when a form action is selected. Only the form action which is not in use can be deleted.

If you deletes a form action which is already in use, the deletion will be disallowed, and a warning message will be displayed.

If you deletes a form which is not in use, the deletion confirmation dialog will be opened. By selecting the **Yes** button, the form action will be deleted immediately. If user selects the **No** button, the deletion will be cancelled.



Next section

Component: Form Management - Forms

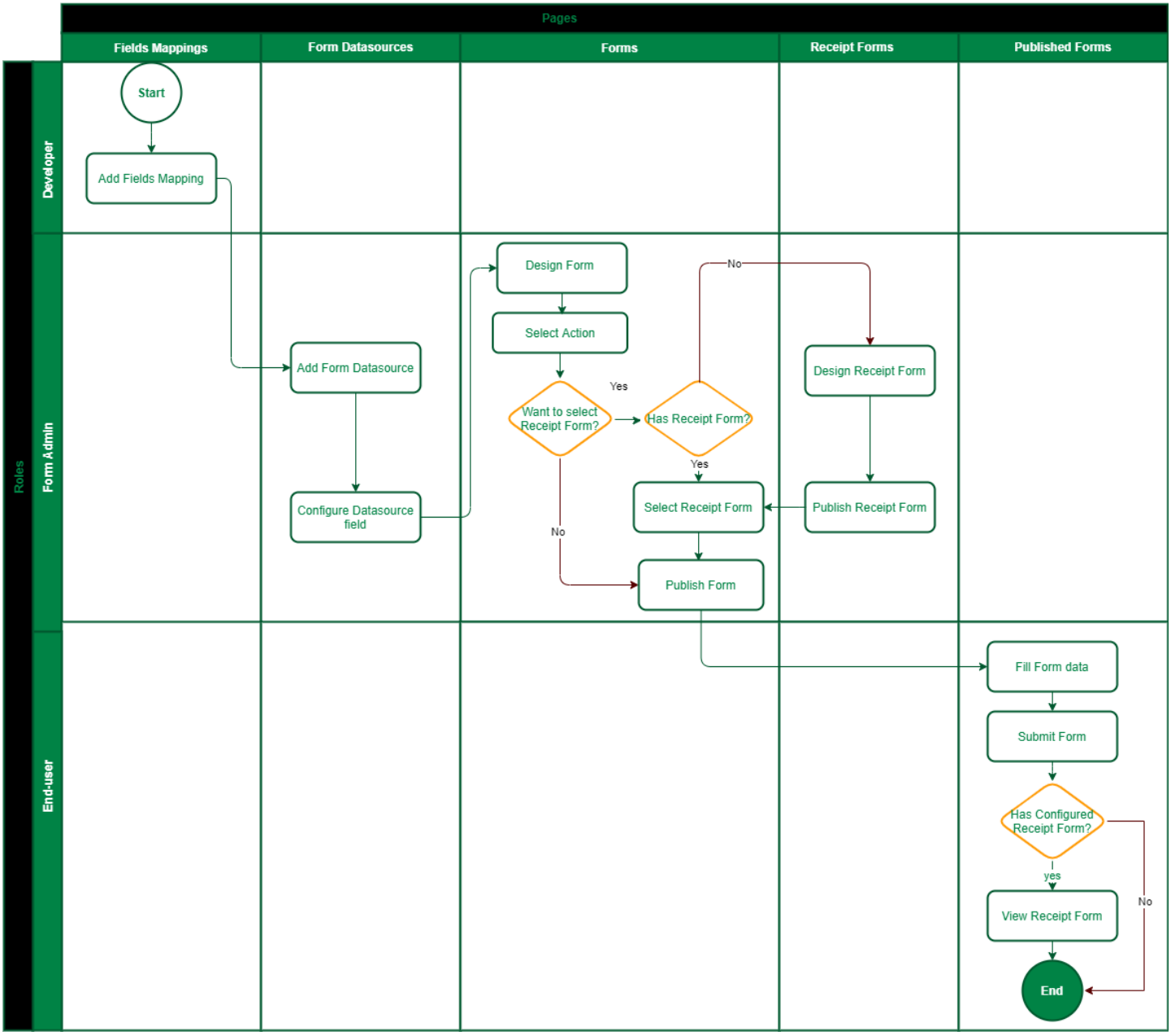
Component: Form management - Form design

Introduction

Form design let the form admin drag and drop independent components into the body of the form builder. The form builder provides a UI, which makes form admin easy to visualize and modify the form.

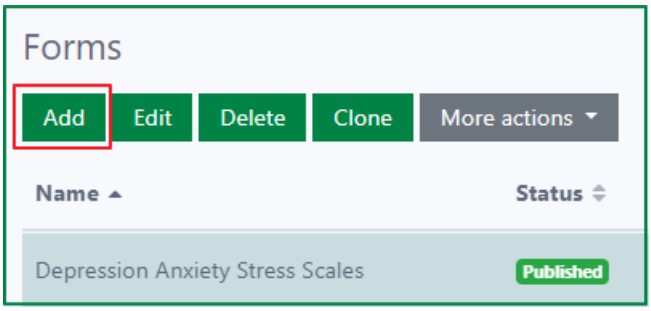


Workflow of form management

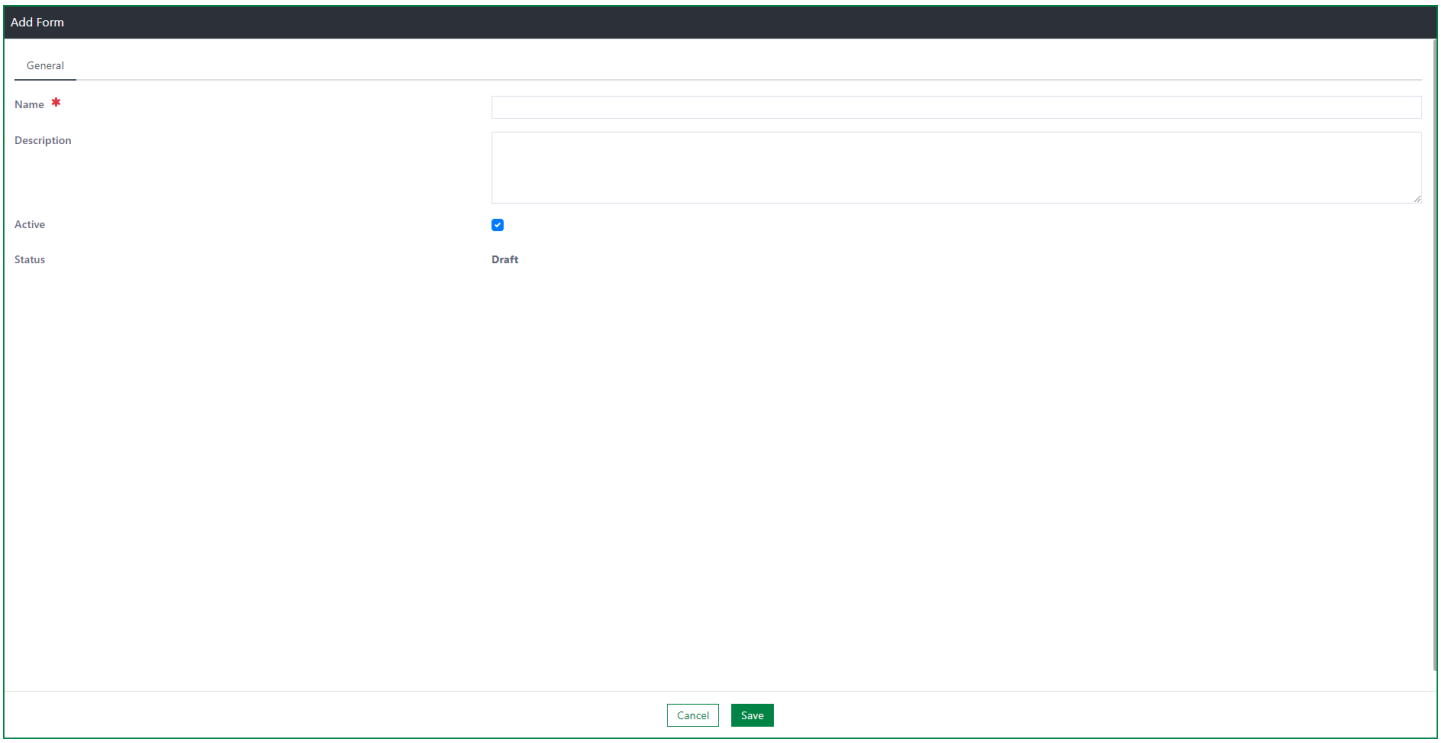


Guideline

Add new form



By clicking on the Add button, the Add form dialog will be opened.



The dialog includes the following fields:

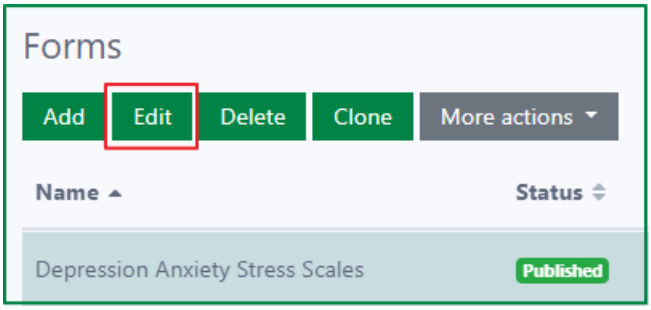
- **Name**
    - This field is required.
    - Form name must be unique.
- **Description**
    - This field is optional.
    - You can describe more information of the form.
- **Active**
    - By default, the form is active. You can deactivate it by unticking the checkbox.
    - An inactive form will be unable to access in public site even if it was published.
- **Status**
    - By default, the form status is **Draft** when it is created.
    - A form can has one of these status: **Draft**, and **Published**.
        - **Draft**

- A form has **Draft** status when it is newly created, or it is edited after being published.

- **Published**
  - A form has **Published** status after it is published successfully.

You can explore different configurations in different field components here.

Edit a form

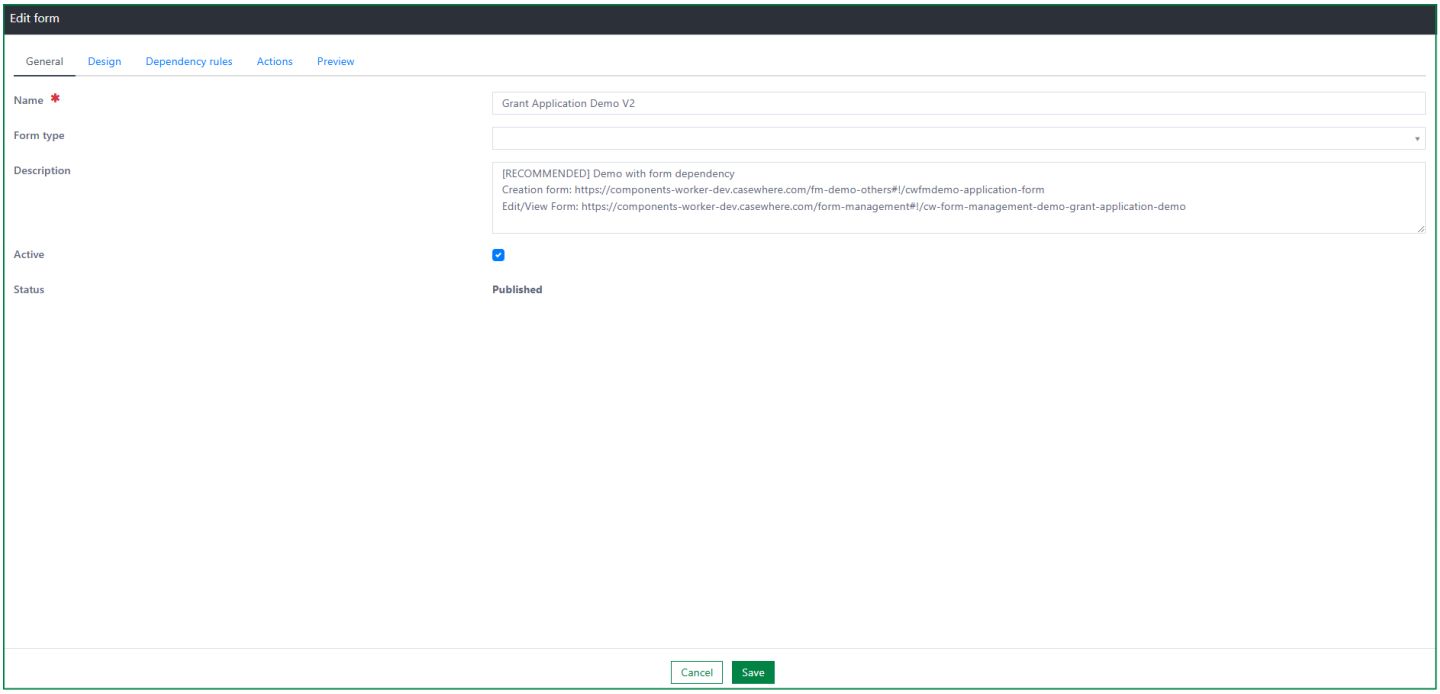


The **Edit form** dialog will be opened after a new form is created successfully or by selecting a form and clicking on the **Edit** button.

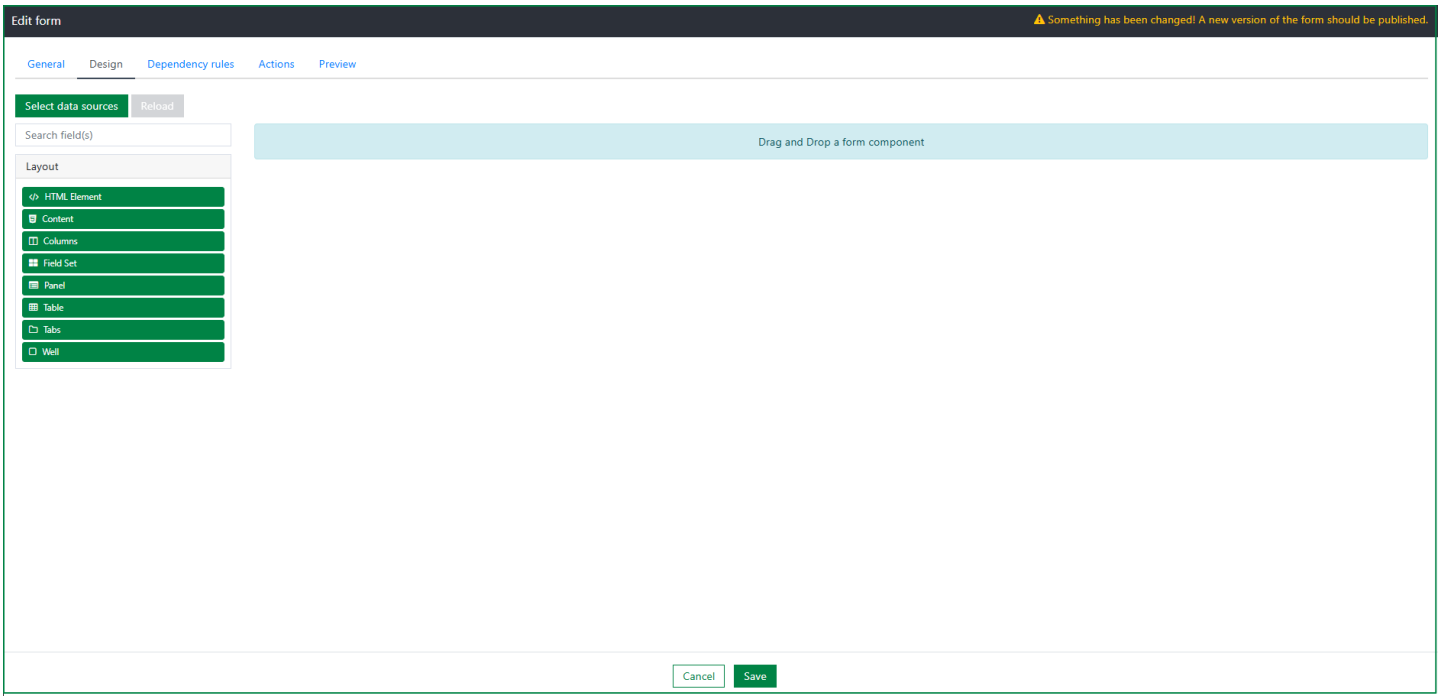
The **Edit form** dialog includes 5 tabs: **General, Design, Dependency rules, Actions,** and **Preview**.

General

This tab displays all saved general information of the form, like: **Name, Form type, Description, Active,** and **Status.**



Design



The **Design** tab includes 2 sections: fields list, and form design area.

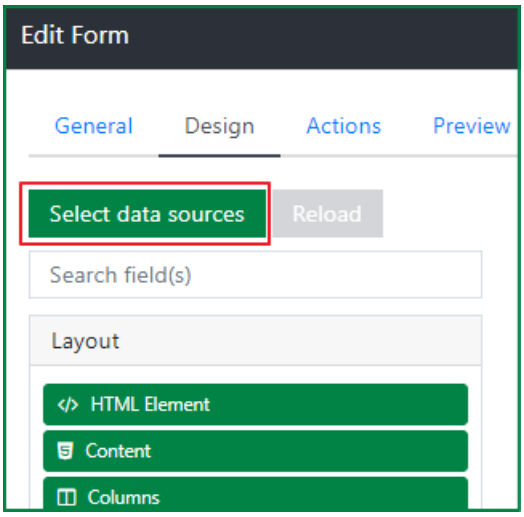
You can drag and drop the fields from the fields list to the form design area.

The fields list includes 2 types of field: layout fields from **Form.io**, and fields from form data source. There are a lot of form data sources to be used for various forms, so, you must select the specific form data source for your form design.

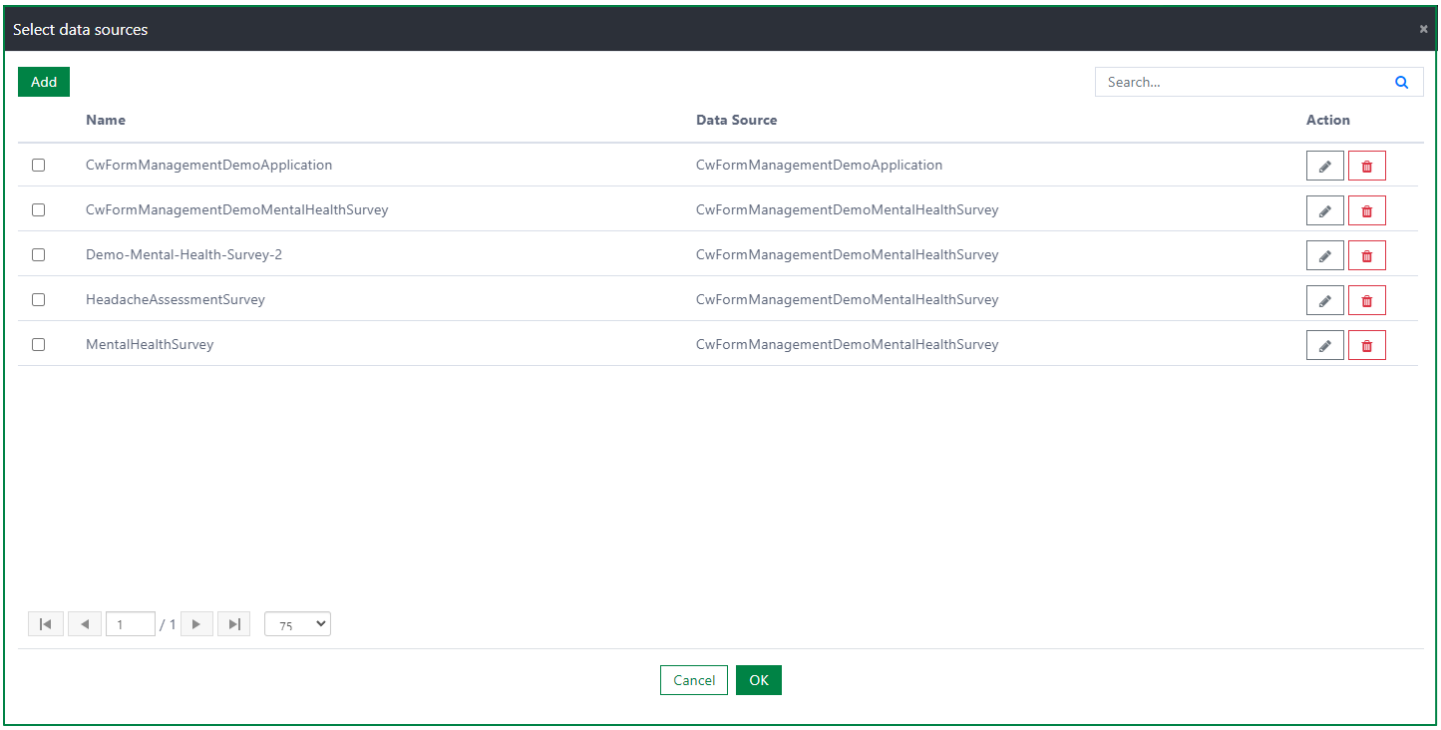
In this part, we separate it to 2 main actions:

- Select form data source
- Design form

Select form data source



By clicking on the **Select data sources** button, the **Select data source** dialog will be opened.

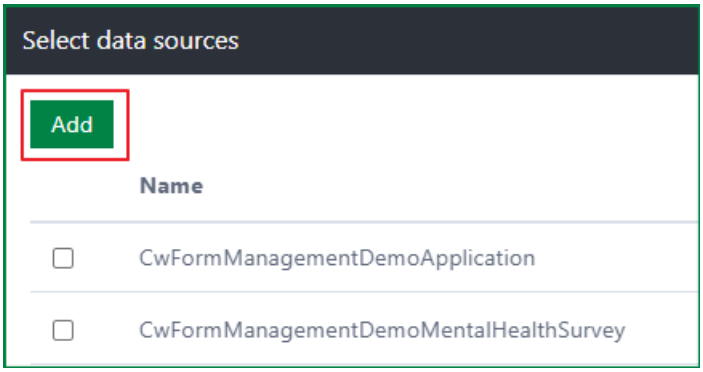


The dialog includes list of all created form data sources from Form data sources page. The form data sources are displayed with basic information like: **Name**, and **Data source**.

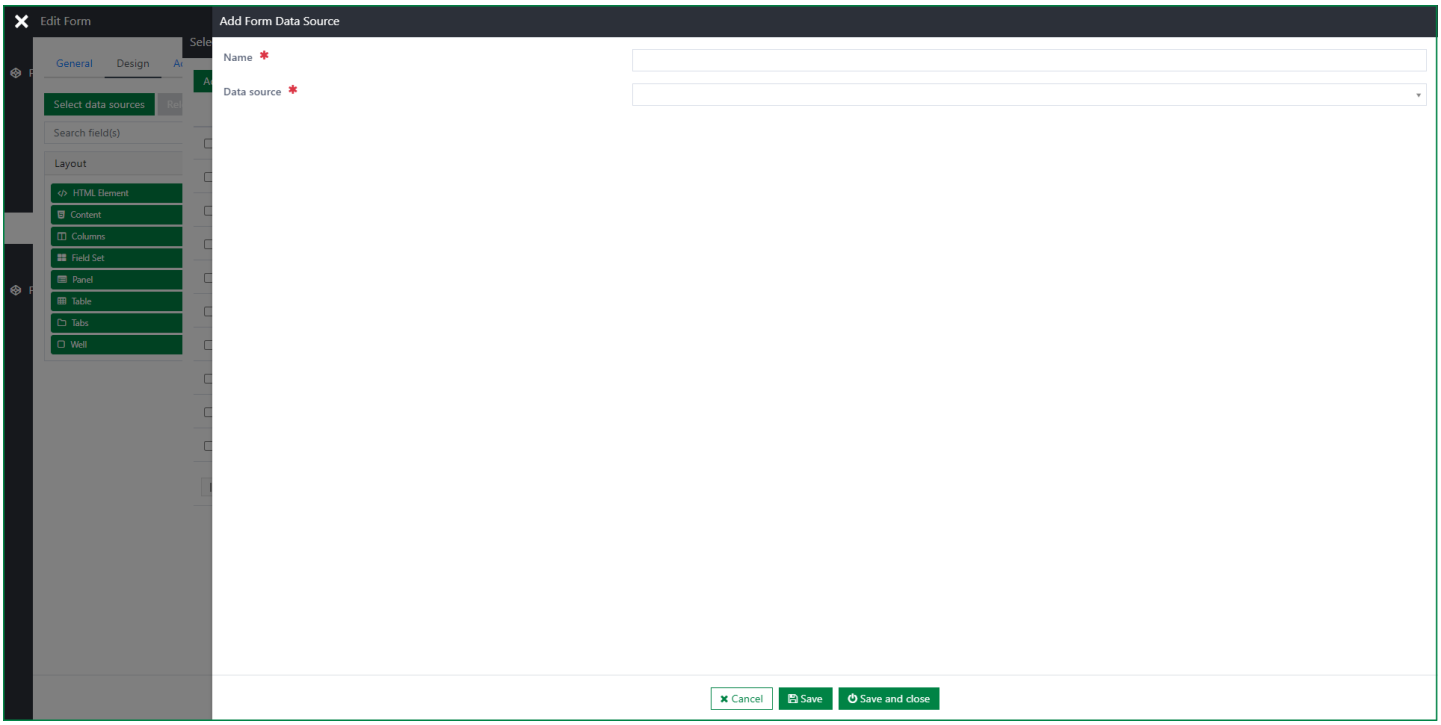
You can select one or multiple data sources to be used in designing form. After the data source is selected, list of fields under the data source will be displayed in the fields list section.

Besides selecting the existing form data sources, you can also manage the form data source in this dialog, like: add new form data source, edit form data source, and delete the form data source.

- **Add new form data source**

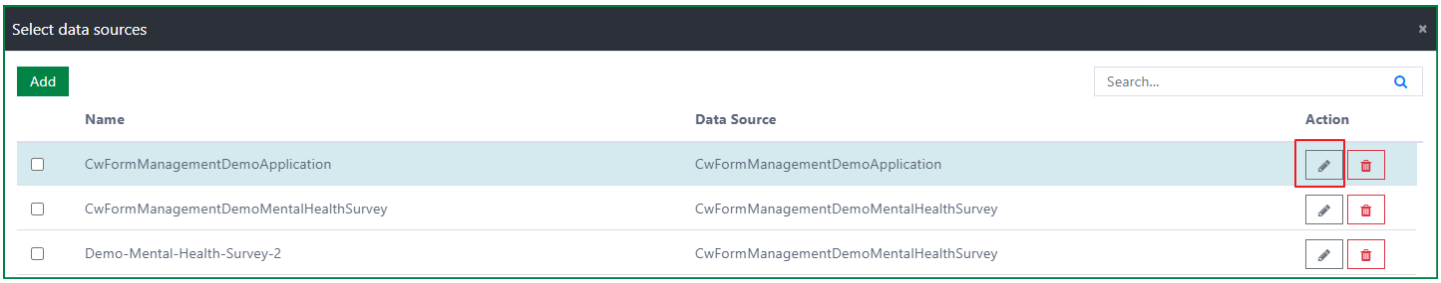


By clicking on the **Add** button, the **Add new form data source** dialog will be opened.

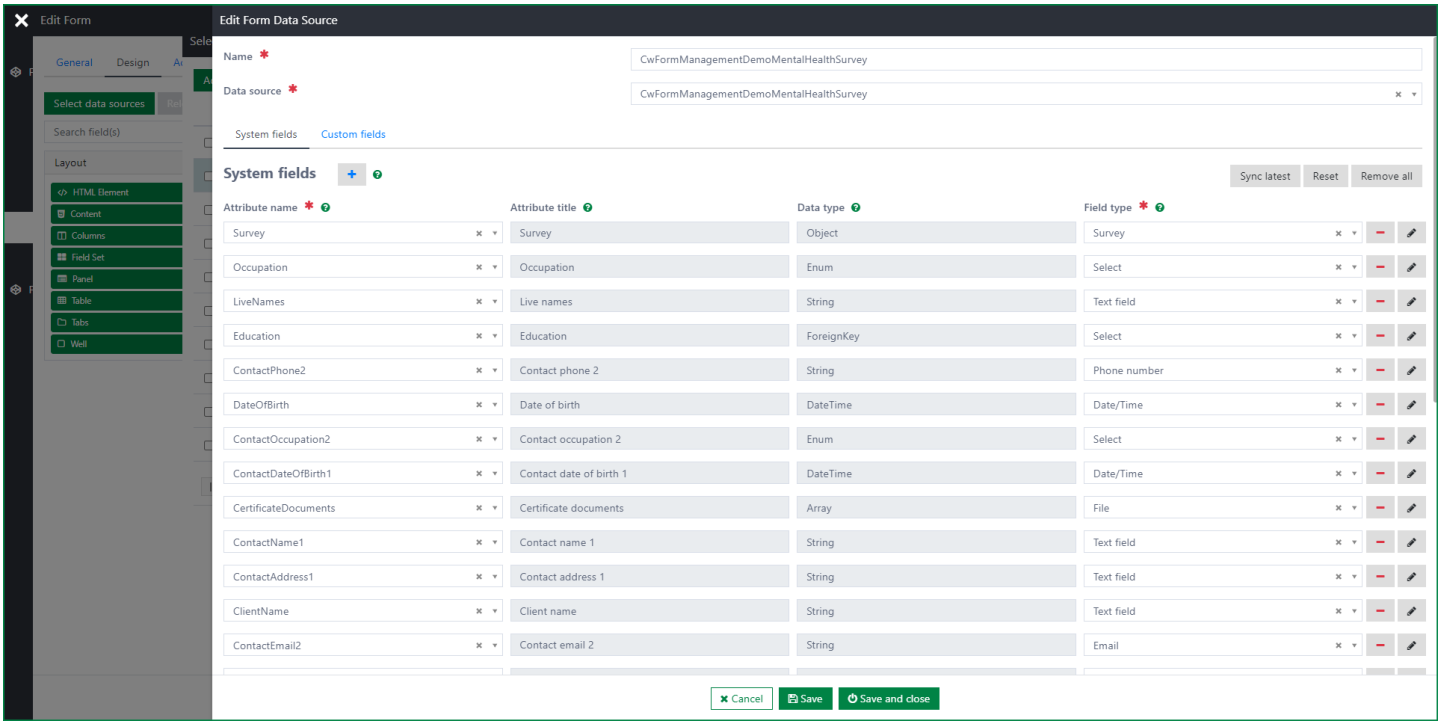


The steps to add new data source is the same to the **Add new form data source** flow mentioned in [Data management][data-management.md] page. The added form data source will be stored in the **Form data sources** page and displayed in the **Select data source** dialog properly.

- **Edit form data source**

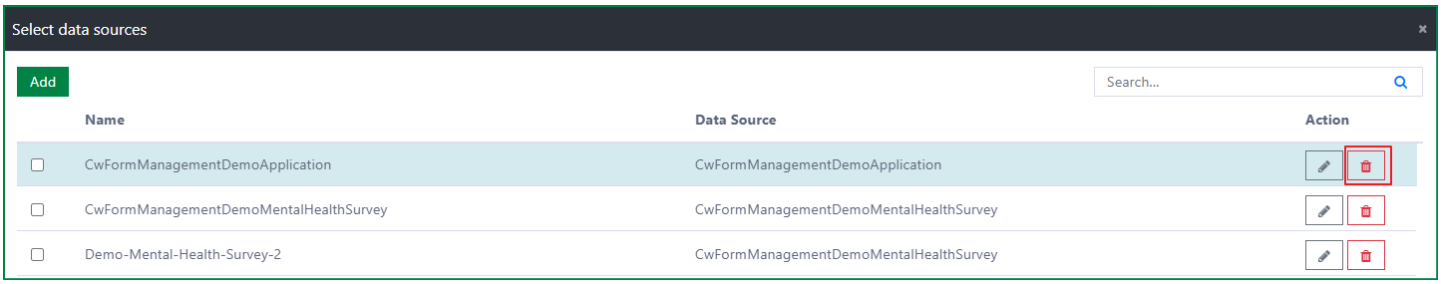


By clicking on the **Edit** button at every form data source record, the **Edit form data source** dialog will be opened.

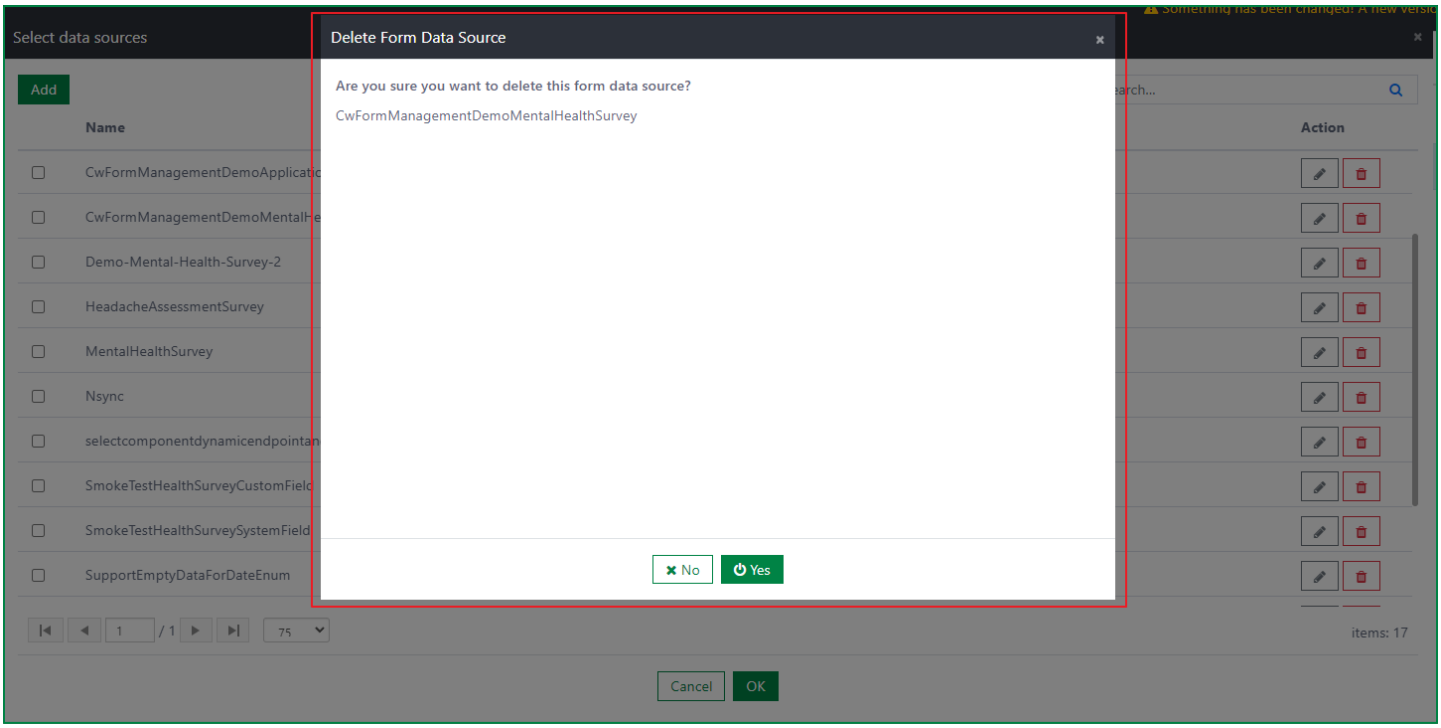


You can edit form data source like the **Edit form data source** flow mentioned in [Data management][data-management.md] page.

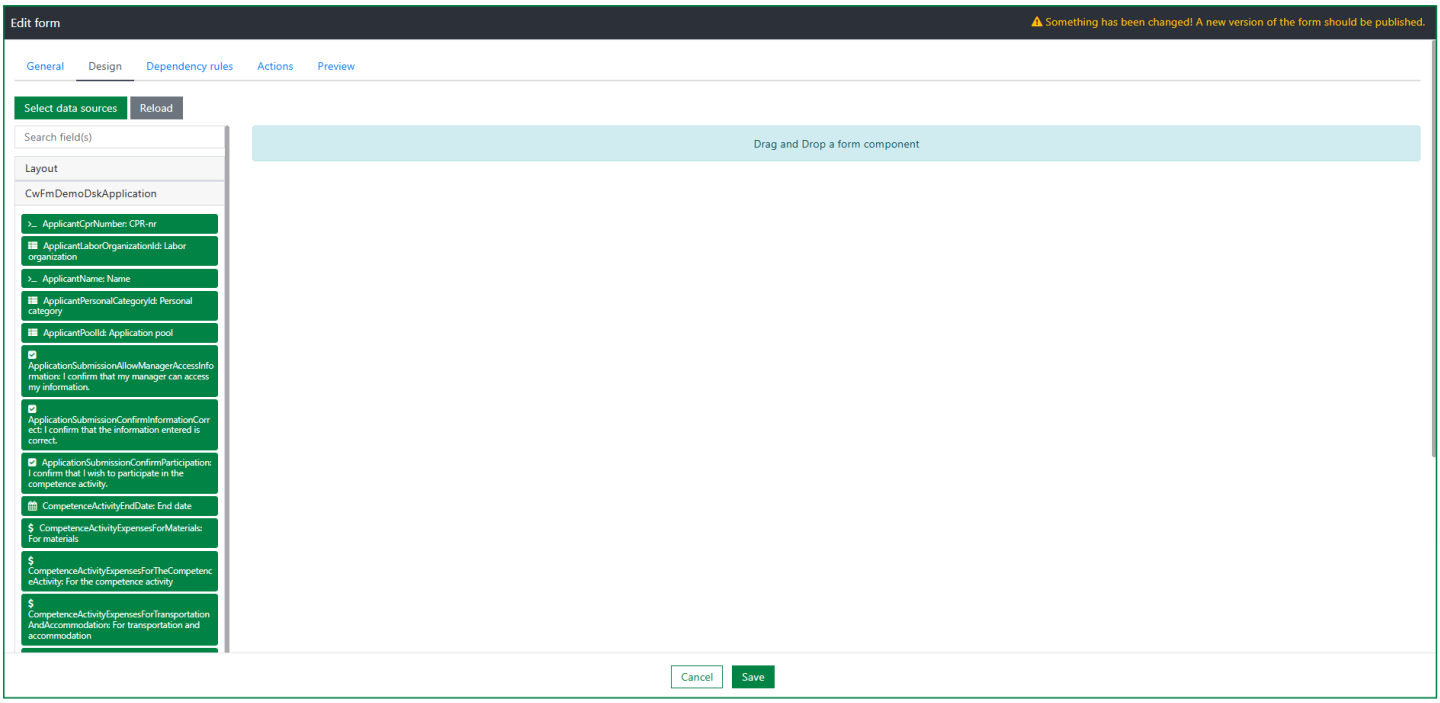
- **Delete form data source**



By clicking on the **Delete** button at every form data source record in the list, a deletion confirmation dialog will be opened. You can choose to cancel, or continue the deletion for the form data source. Only the form data source which is not in use can be deleted. The data source which is deleted in the **Select data sources** dialog will be also deleted from **Form data sources** page.

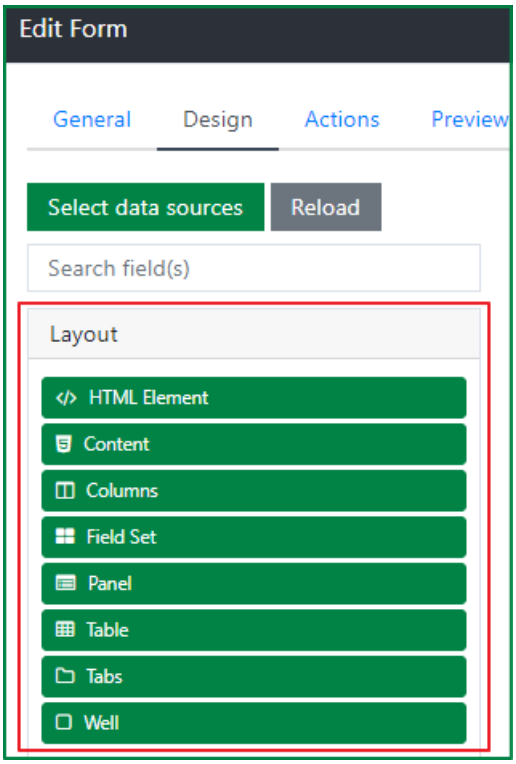


Design form



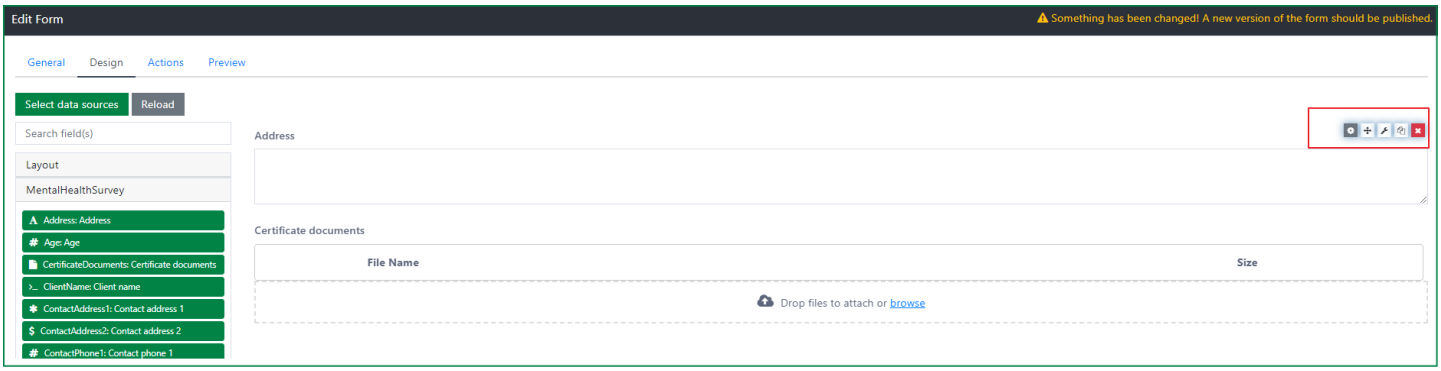
After the data sources are selected, all fields under the data sources will be displayed in the fields list section. You can drag and drop the fields to the design area to design form.

You can customize UI of the form (columns, table, etc.) by using the layout fields supported by **Form.io**. The supported layout fields are: **HTML element, Content, Columns, Field set, Table, Tabs,** and **Well.**

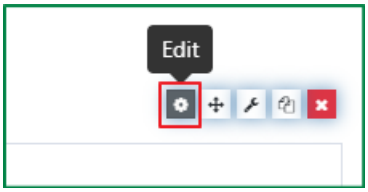


After adding the field to the form, you can edit the component settings, change the field position in the form, edit JSON of the component, copy new field from the field, remove the field from the form.

The relevant functional buttons will be displayed when you hover over the added field.



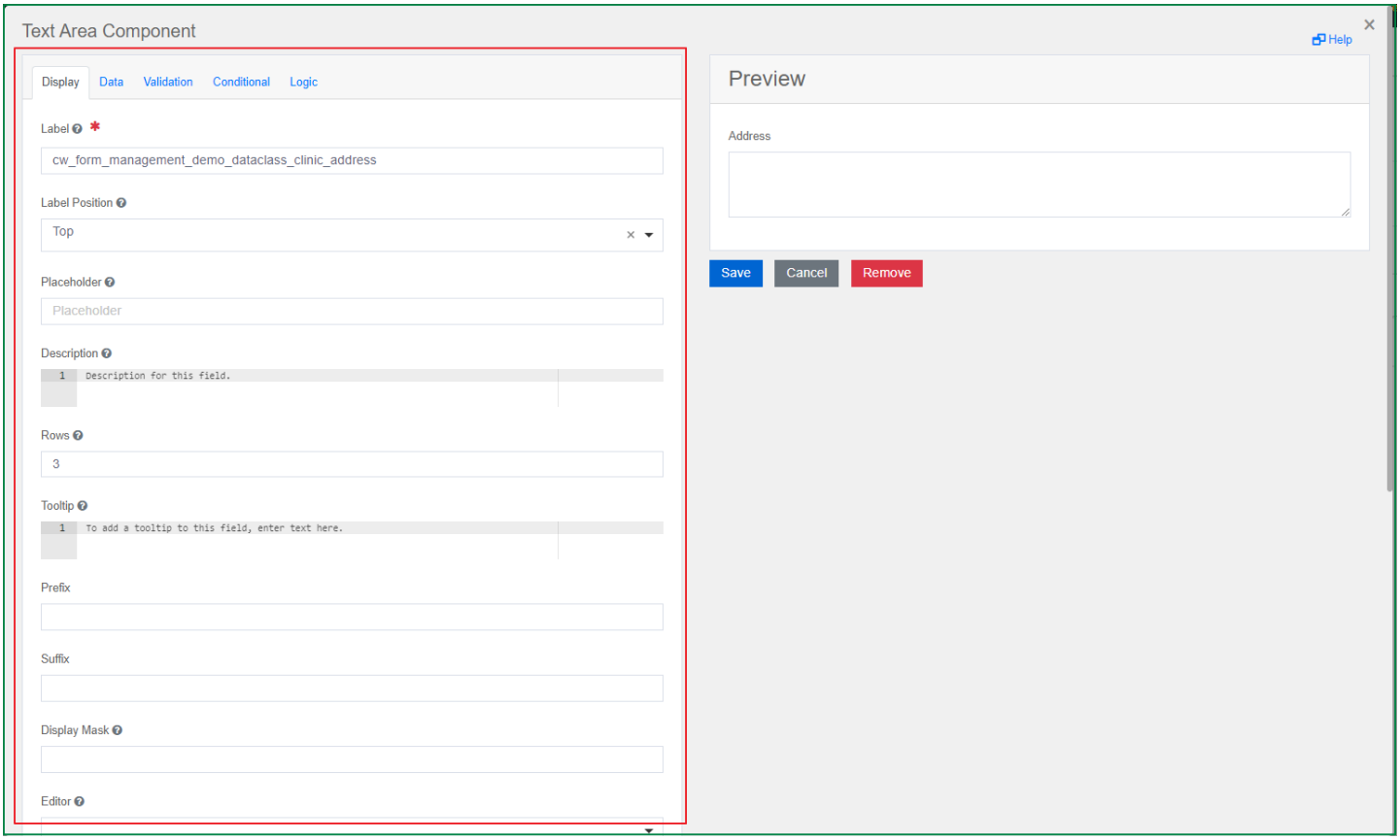
- **Edit component settings**



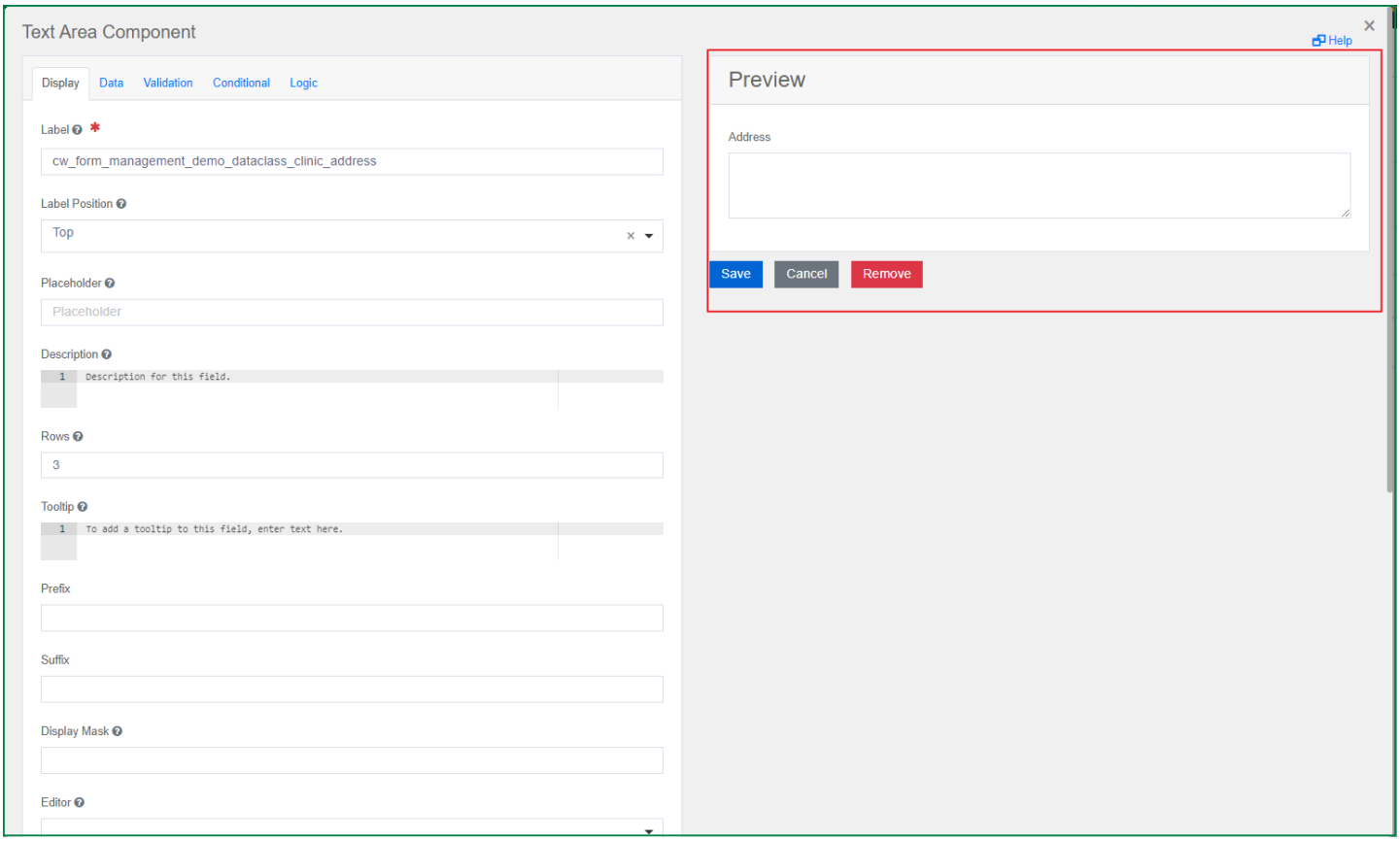
By clicking on the **Edit** button, according to field type of the field, the corresponding component settings dialog will be opened.

The dialog includes 2 sections: configuration section, and preview section.

The configuration section includes 5 tabs: **Display settings**, **Data**, **Validation**, **Conditional**, and **Logic**.

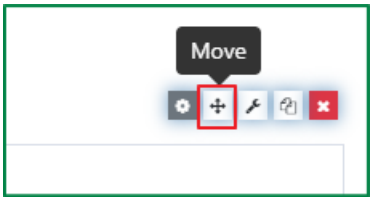


After configuring the settings for the field, you can preview it in the preview section before saving the changes.

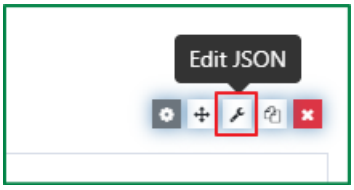


- **Change the field position in the form**

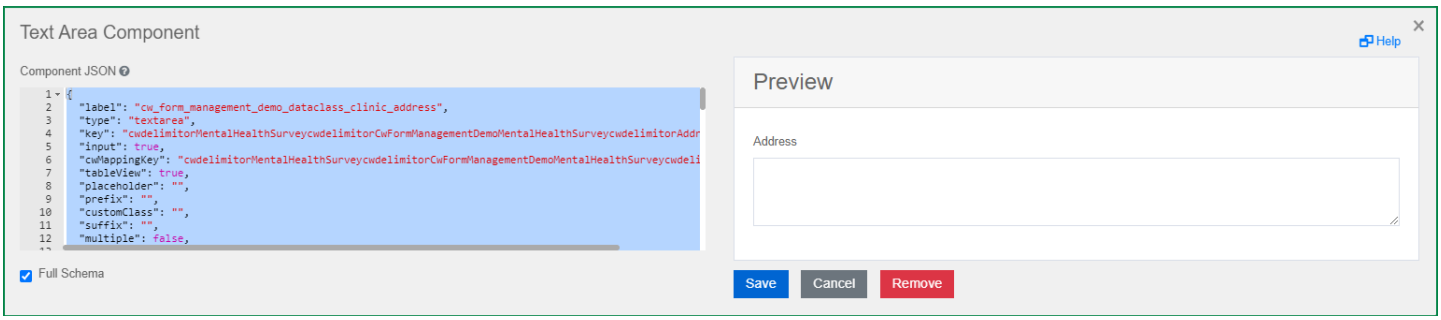
You can change position of the field in the form by holding the **Move** button and move the field to the expected position.



- **Edit JSON of component**



By clicking on the **Edit JSON** button, the corresponding JSON dialog of the component will be opened. You can directly edit the component settings by configuring the JSON.



The JSON Schema depends on the configuration of the component.

If you tick on the **Full Schema** checkbox, it will show the full schema configurations, and the schema of setting which was not

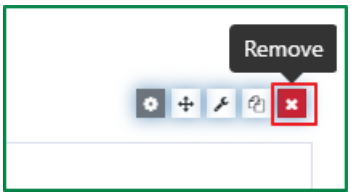
configured for the component will be also displayed.

- **Copy new field from the current field**



You can copy the field by clicking on the **Copy** button and paste it under another field by clicking on the **Paste below** button.

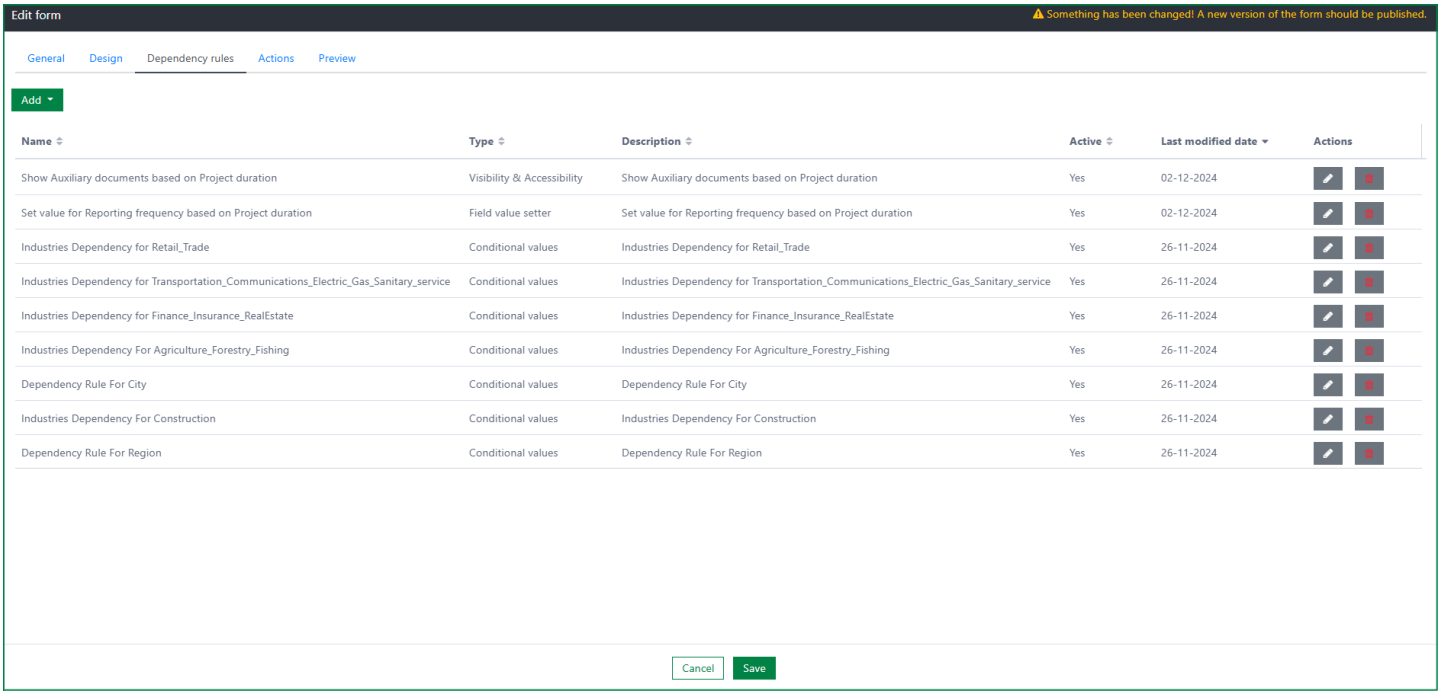
- **Remove the field from the form**



When you click on the **Remove** button of the field, the field will be removed from the form immediately.

After completing designing form, you must save the form design before moving to the other tabs.

Dependency rules



The **Dependency rules** tab includes a list of configured dependency rules for fields in the form. You can add, edit, delete the dependency rules.

The changes on dependency rules are only activated on the submission form when the form is published after the changes.

There are 3 types of dependency rules: **Conditional values** , **Visibility & Accessibility**, and **Field value setter**.

- **Conditional values dependency rule**
  - This rule defines the dependency of values list of a field on the value of other fields. This rule is only available for the dependent fields with field type **Select**, **Select boxes**, **Radio button**.
  - Following is the example for the dependency of **Region** values list on the value of the **Country** field:
    - If the selected value of the **Country** field is **Denmark**, the values list of the Region field includes **Hovedstaden**, **Midtjylland**, and **Nordjylland**.

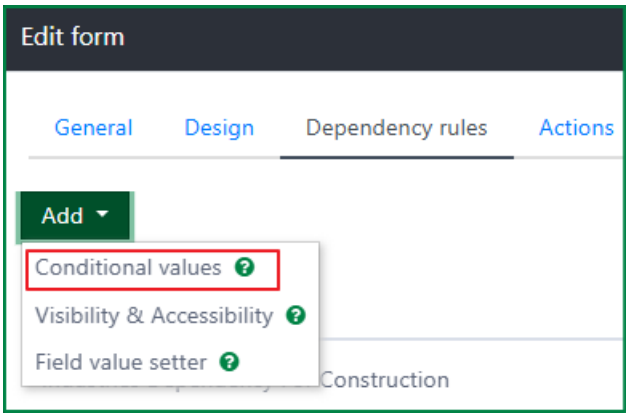
        - If the selected value of the **Country** field is **Germany**, the values list of the Region field includes **Dresden**, **Bautzen**, and **Leipzig**.

- **Visibility & Accessibility dependency rule**
    - This rule defines whether a field should be shown/hidden, required/not required, enabled/disabled based on the value of other fields.
    - Following is the example for the dependency of visibility and accessibility of the **Auxiliary documents** field on value of the **Project duration** field:
        - If the value of the **Project duration** field is greater than or equal to 6 months, the **Auxiliary documents** field is shown, required, and enabled.
        - If the value of the **Project duration** field is less than 6 months, the **Auxiliary documents** field is hidden, and not required.

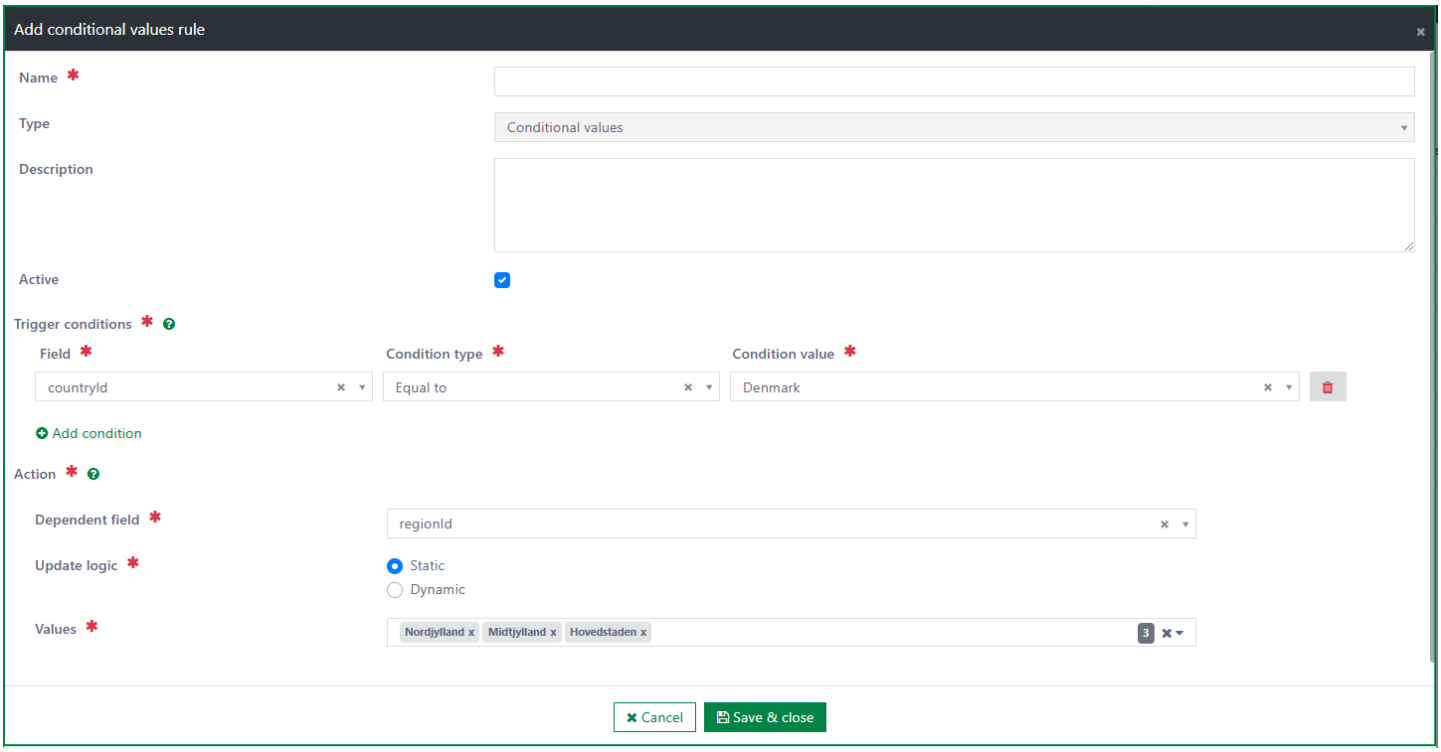
- **Field value setter dependency rule**
    - This rule defines the value indication of a field based on value of other fields.
    - Following is the example of the indication for value of the **Reporting frequency** field based on the value of **Project duration** field. Given that the **Reporting frequency** field cannot be inputted manually:
        - If the value of the **Project duration** field is less than 12 months, value of the **Reporting frequency** field is set to **Monthly** automatically.
        - If the value of the **Project duration** field is equal to or greater than 12 months, value of the **Reporting frequency** field is set to **Quarterly** automatically.

Add new dependency rule

Conditional values dependency rule

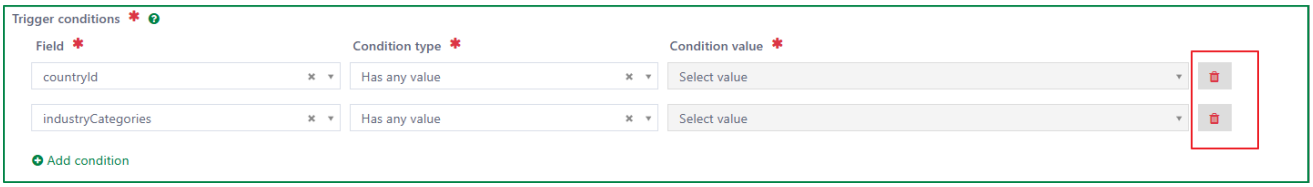


By clicking on the **Add** button and selecting the option **Conditional field values**, the **Add conditional field values rule** dialog is displayed.



The dialog includes the following fields:

- **Name**
  - This field is required.
  - Name of the dependency rule must be unique.
- **Type**
  - The field is read-only and prefilled with type **Conditional values**.
- **Description**
  - This field is optional.
  - You can describe more information of the dependency rule.
- **Active**
  - By default, the dependency rule is active. You can deactivate it by unticking the checkbox.
  - In submission form, only the active dependency rule is affective.
- **Trigger conditions**
  - This section is to define a list of conditions which triggers the execution of the dependency rule.
  - *Given the example (1)*: ***If the selected country is Denmark, the list of regions includes Hovedstaden, Midtjylland, and Nordjylland***; in this example, the **Trigger conditions** will handle the part of ***If the selected country is Denmark***.
  - A dependency rule can have one or multiple trigger conditions. If multiple conditions are added, the relation between the conditions is **AND**, it means that the rule is only triggered if all conditions are met.
  - You can also delete the added trigger conditions by clicking on the **Delete** button at the end of the condition row, but the rule must have at least a trigger condition.

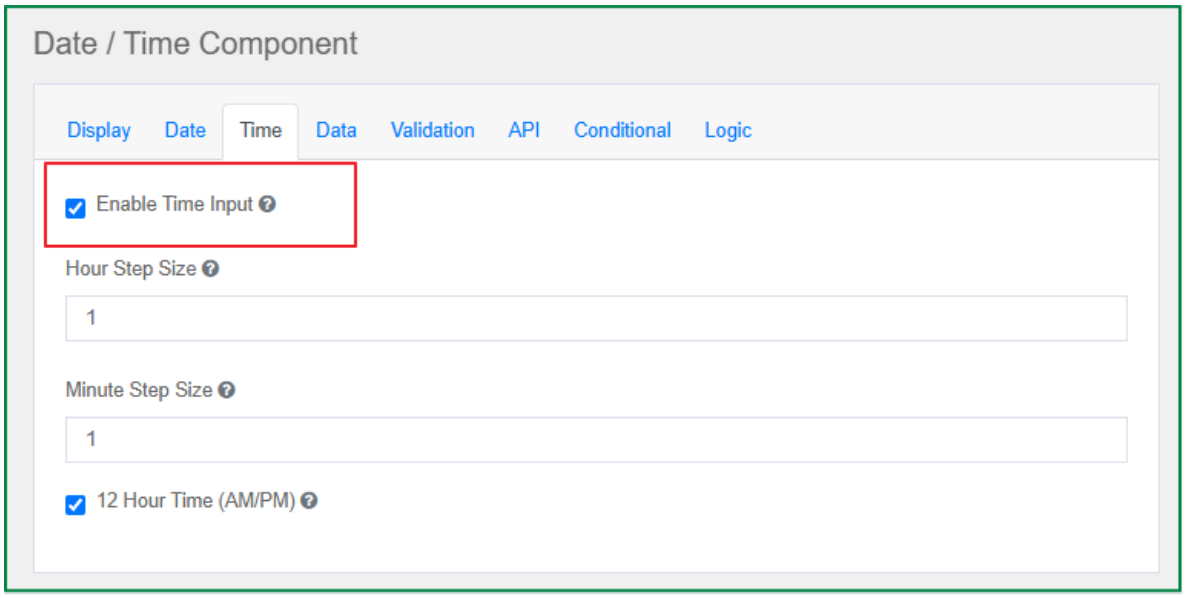


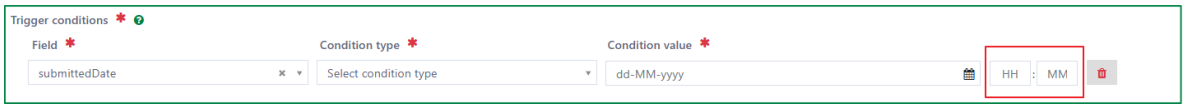
- By clicking on the **Add condition** button, new record of condition is added, each record includes the following fields:
  - **Field**
    - This field is used to select the field which is the object of the condition.
    - This field is required.
    - The dropdown list values are the fields added to the form in the Design tab. You can select a field from the list.
    - If there are multiple trigger conditions, the condition field must be unique. You cannot add 2 conditions with the same condition field.
    - The field with data type **Document** or **Guid** is not supported.
    - *According to the example (1) above*, the value should be **Country**
  - **Condition type**
    - This field is used to select the type of the condition.
    - This field is required.
    - The field includes a list of condition types.
    - There are 9 condition types, the list of condition types is displayed according to the data type of the selected field.
    - The mappings between condition type and data type are as the following table.

DATA TYPE	CONDITION TYPE
Array	In, Not in, Has any value
Boolean	Equal to, Not equal to, Has any value
DateTime	Equal to, Not equal to, Greater than, Greater than or equal, Less than, Less than or equal, Has any value
Double	Equal to, Not equal to, Greater than, Greater than or equal, Less than, Less than or equal, Has any value
Enum	In, Not in, Equal to, Not equal to, Has any value
ForeignKey	In, Not in, Equal to, Not equal to, Has any value
Integer	Equal to, Not equal to, Greater than, Greater than or equal, Less than, Less than or equal, Has any value
Long	Equal to, Not equal to, Greater than, Greater than or equal, Less than, Less than or equal, Has any value
String	Equal to, Not equal to, Has any value
Guid	Equal to, Not equal to, Has any value

    - *According to the example (1) above*, the **Condition type** should be **Equal to**
  - **Condition value**
    - This field is used to define value of the condition field.

- This field is required.
- If the selected **Condition type** is **Has any value**, the field is disabled. It means that the rule is triggered with any value of the condition field except for the **null** value.
- If the selected field having data type **DateTime** and the setting **Enable Time Input** of the component was enabled, the **Condition value** field format is date time. If the setting **Enable Time Input** of the component was disabled, the **Condition value** field format is date only.





- *According to the example (1) above*, the **Condition value** should be **Denmark**.

- **Action**
  - This section is to define the action will be executed when the dependency rule is triggered.
    - *According to the example (1) above*, this **Action** section will handle the part of ***the list of regions includes Hovedstaden, Midtjylland, and Nordjylland***.
  - The section includes the following fields:
    - **Dependent field**
      - This field is used to select the field which is object of the action when the dependency rule is executed.
      - This field is required.
      - The dropdown list values are the fields with field types **Select**, **Select boxes**, **Radio button** added to the form in the Design tab. You can select a field from the list.
      - *According to the example (1) above*, the **Dependent field** value should be **Region**.
    - **Update logic**
      - This field is used to select the logic for updating the dependent field.
      - There are 2 types of the update logic: **Static**, and **Dynamic**.
      - If the selected update logic is **Static**, the field **Values** is displayed.
        - With this option, you can select one or multiple items from the list of defined values of the dependent field to be shown on UI when the dependency rule is triggered.
      - If the selected update logic is **Dynamic**, the fields to configure the filter condition is displayed.

        - With this update logic, only the values matching the filter condition will be shown on UI when the dependency rule is triggered.
        - The **Dynamic** option is only enabled if the data type of the selected dependent field is **ForeignKey**, or **Array-ForeignKey**.

    - *According to the example (1) above* , the **Update logic** field value should be **Static**

- **Values**

    - This field is only displayed if the update logic is **Static**.
    - This field is used to select the value lists of the dependent field which will be displayed when the dependency rule is triggered.
    - This field is required after it is displayed.
    - The dropdown list values are the values of the selected dependent field. You can select one or multiple items in the list

    - *According to example (1) above*, the **Values** field values should be **Hovedstaden, Midtjylland**, and **Nordjylland**.



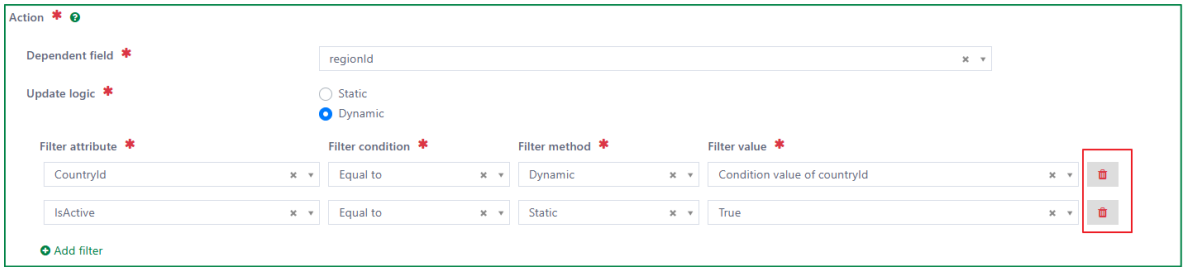
- **Filter condition**

    - This section is only displayed if the update logic is **Dynamic.**

    - This section is used to configure the condition to filter the values of the dependent field which will be displayed when the dependency rule is triggered based on the attributes of the field.

    - You can add one or multiple conditions. If there are multiple conditions are added, the relation between the conditions is **AND**.

    - You can also delete the added conditions by clicking on the **Delete** button at the end of the condition row, but the rule must have at least 1 filter condition if the update method is **Dynamic**.



    - By clicking on the **Add filter button**, new record of filter condition is displayed with the following fields:

        - **Filter attribute**

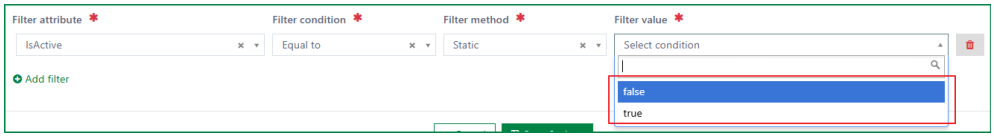
            - This field is used to select an attribute of the dependent field. The attribute is the object of the filter condition
            - This field is required.
            - The dropdown list of the field includes a list of the attributes of the selected dependent field.

        - **Filter condition**

- This field is used to select the type of the condition.
- This field is required.
- The field includes a list of condition types.
- The list of condition types is displayed according to the data type of the selected attribute.
- The mappings between condition type and data type are as the following table.

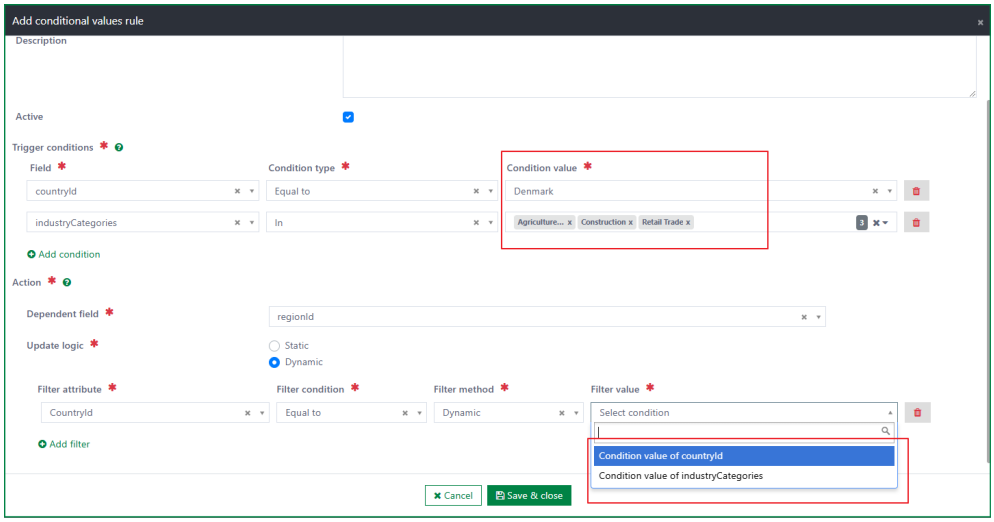
DATA TYPE	CONDITION TYPE
Array	In, Not in
Boolean	Equal to, Not equal to
DateTime	Equal to, Not equal to
Double	Equal to, Not equal to
Enum	In, Not in, Equal to, Not equal to
ForeignKey	Based on the data type of the field that the attribute refers to
Integer	Equal to, Not equal to
Long	Equal to, Not equal to
String	Equal to, Not equal to
Guid	Equal to, Not equal to

- **Filter method**
  - This field is used to select the method of the filtering.
  - This field is required.
  - There are 2 methods: **Static**, **Dynamic**.
    - With the method Static, you can select the existing values of the selected filter attribute
    - With the method Dynamic, you can select the
  - The values list of the **Filter value** field depends on the selected method.
- **Filter value**
  - This field is required.
  - If the filter method is **Static,** the values list of the **Filter value** field is the values of the selected attribute.
    - For example: If the filter attribute is **IsActive** (boolean), the list of filter value is **true**, **false**.

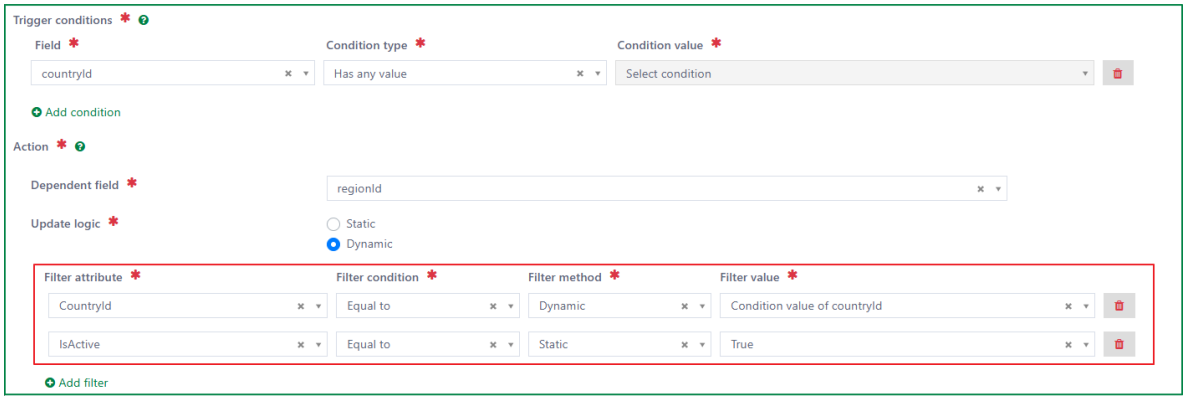


  - If the filter method is **Dynamic**, the values list of the **Filter value** field is the condition values of the configured the trigger conditions above.

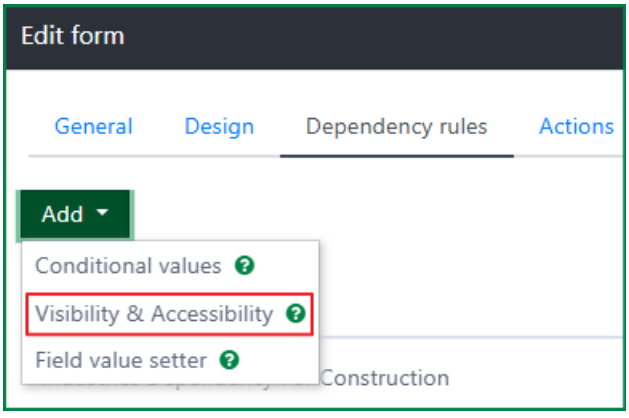
- *For example:* There are trigger conditions of field **countryId**, **industryCategories**, then the values of the Filter value field includes **Condition value of countryId**, **Condition value of industryCategories**



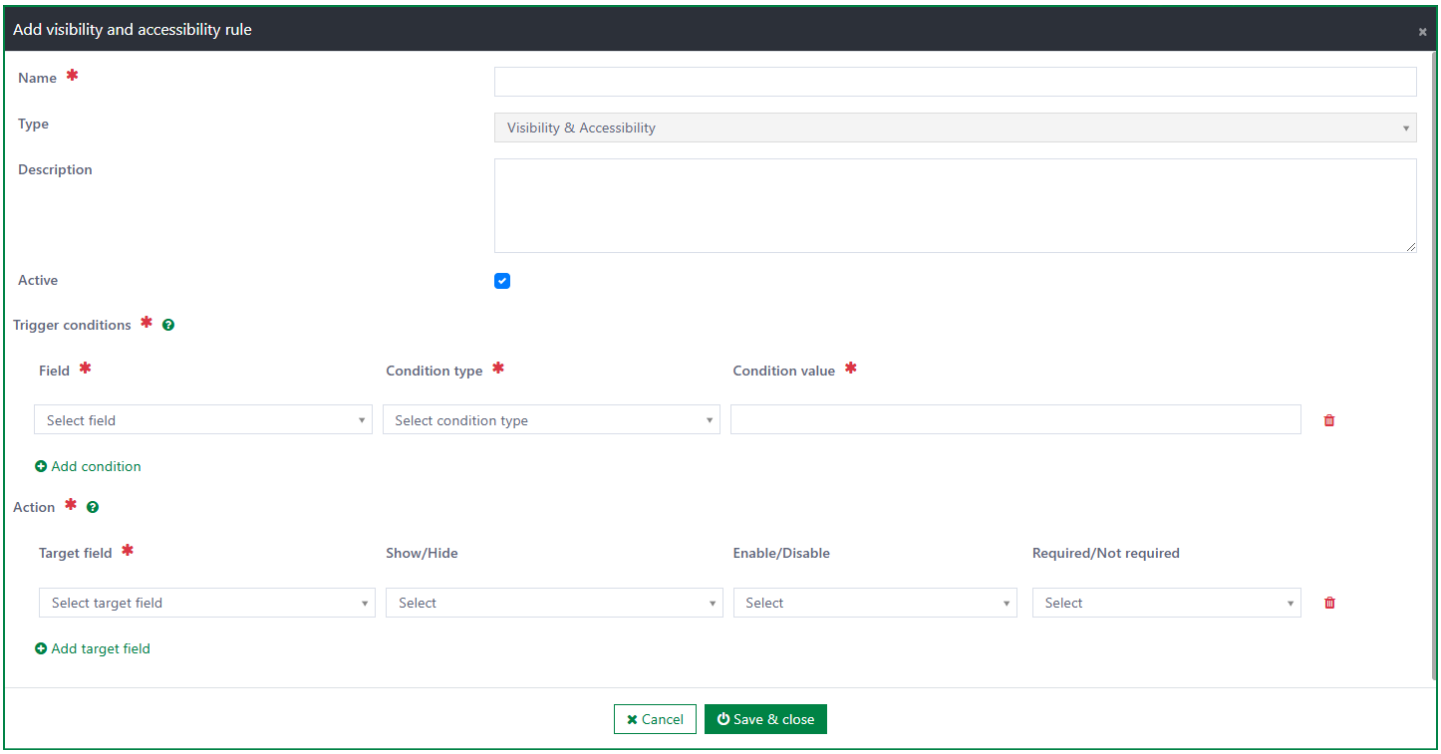
- *Following is an example of the conditional value dependency rule with filter condition:* ***If a country selected, then the values list of the Region must includes the active regions of the selected country***



Visibility & Accessibility dependency rule

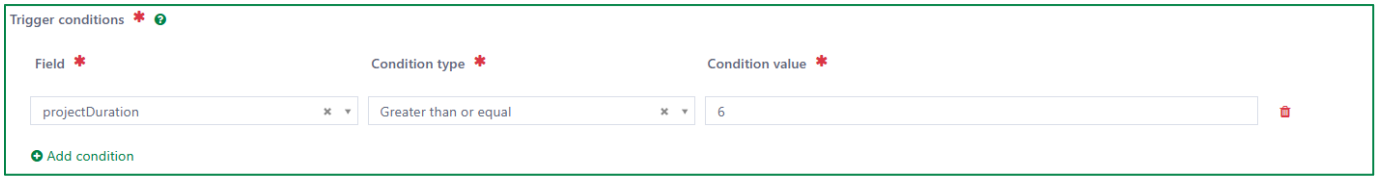


By clicking on the **Add** button and selecting the option **Visibility & Accessibility**, the **Add visibility and accessibility rule** dialog is displayed.



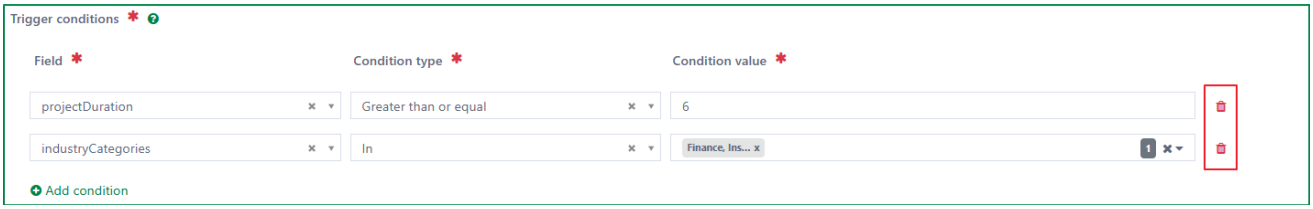
The dialog includes the following fields:

- **Name**
  - This field is required.
  - Name of the dependency rule must be unique.
- **Type**
  - The field is read-only and prefilled with type **Visibility & Accessibility**.
- **Description**
  - This field is optional.
  - You can describe more information of the dependency rule.
- **Active**
  - By default, the dependency rule is active. You can deactivate it by unticking the checkbox.
  - In submission form, only the active dependency rule is affective.
- **Trigger conditions**



  - This section is to define a list of conditions which triggers the execution of the dependency rule.
  - *Given the example (2)*: **If the Project duration is greater than or equal to 6 months, the Auxiliary documents field is shown, required, and enabled**; in this example, the **Trigger conditions** will handle the part of **If the Project duration is greater than or equal to 6 months**.
  - A dependency rule can have one or multiple trigger conditions. If multiple conditions are added, the relation between the conditions is **AND**, it means that the rule is only triggered if all conditions are met.

- You can also delete the added trigger conditions by clicking on the **Delete** button at the end of the condition row , but the rule must have at least a trigger condition

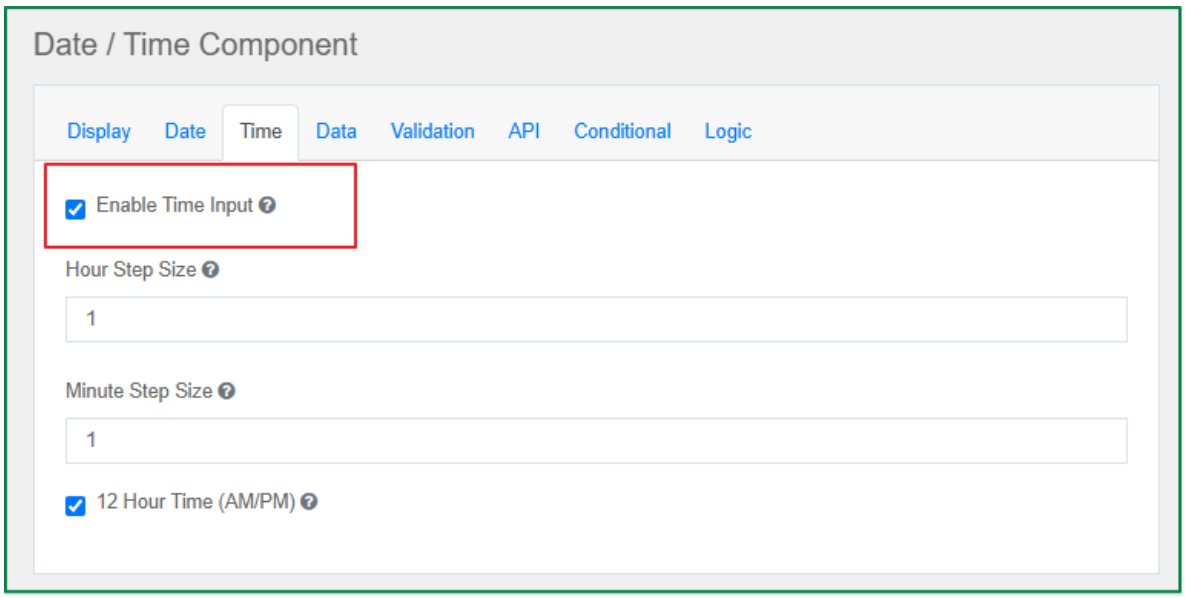


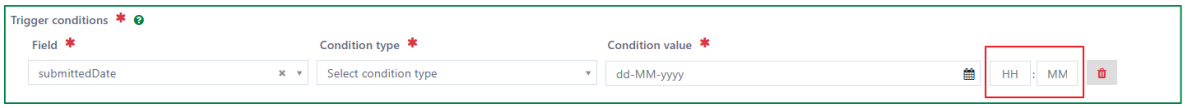
- By clicking on the **Add condition** button, new record of condition is added, each record includes the following fields:
  - **Field**
    - This field is used to select the field which is the object of the condition.
    - This field is required.
    - The dropdown list values are the fields added to the form in the Design tab. You can select a field from the list.
    - If there are multiple trigger conditions, the condition field must be unique. You cannot add 2 conditions with the same condition field.
    - The field with data type **Document** or **Guid** is not supported.
    - *According to the example (2) above*, the value should be **Project duration**
  - **Condition type**
    - This field is used to select the type of the condition.
    - This field is required.
    - The field includes a list of condition types.
    - There are 9 condition types, the list of condition types is displayed according to the data type of the selected field.
    - The mappings between condition type and data type are as the following table.

DATA TYPE	CONDITION TYPE
Array	In, Not in, Has any value
Boolean	Equal to, Not equal to, Has any value
DateTime	Equal to, Not equal to, Greater than, Greater than or equal, Less than, Less than or equal, Has any value
Double	Equal to, Not equal to, Greater than, Greater than or equal, Less than, Less than or equal, Has any value
Enum	In, Not in, Equal to, Not equal to, Has any value
ForeignKey	In, Not in, Equal to, Not equal to, Has any value
Integer	Equal to, Not equal to, Greater than, Greater than or equal, Less than, Less than or equal, Has any value
Long	Equal to, Not equal to, Greater than, Greater than or equal, Less than, Less than or equal, Has any value
String	Equal to, Not equal to, Has any value
Guid	Equal to, Not equal to, Has any value

    - *According to the example (2) above*, the **Condition type** should be **Greater than or equal**

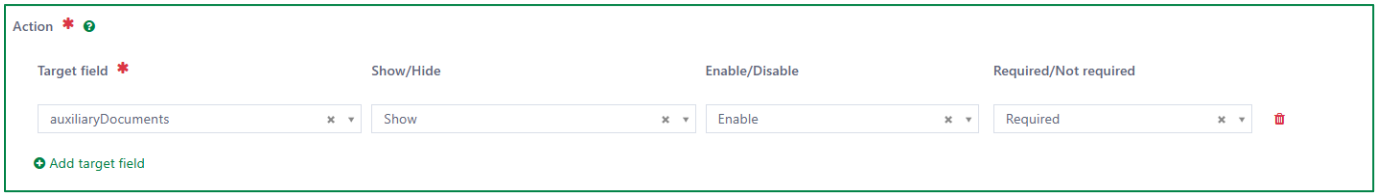
- **Condition value**
  - This field is used to define value of the condition field.
  - This field is required.
  - If the selected **Condition type** is **Has any value**, the field is disabled. It means that the rule is triggered with any value of the condition field except for the **null** value.
  - If the selected field having data type **DateTime** and the setting **Enable Time Input** of the component was enabled, the **Condition value** field format is date time. If the setting **Enable Time Input** of the component was disabled, the **Condition value** field format is date only.



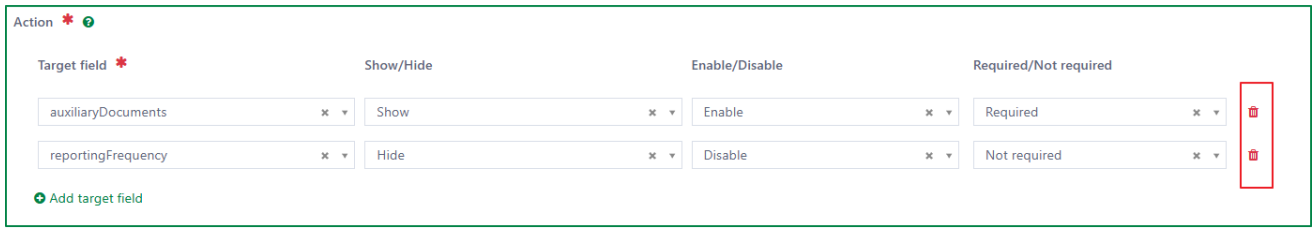


  - *According to the example (2) above*, the **Condition value** should be **6**.

- **Action**

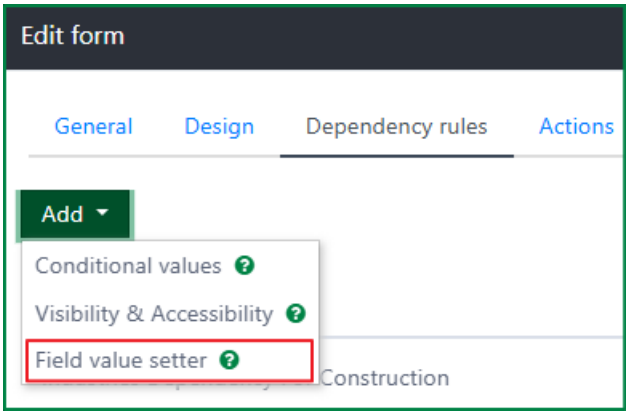


  - This section is to define the action will be executed when the dependency rule is triggered.
    - *According to the example (2) above*, this **Action** section will handle the part of ***the Auxiliary documents field is shown, required, and enabled.***
  - You can add one or multiple target fields.
  - You can also delete the added target fields by clicking on the **Delete** button at the end of the row, but the rule must have at least a target field.

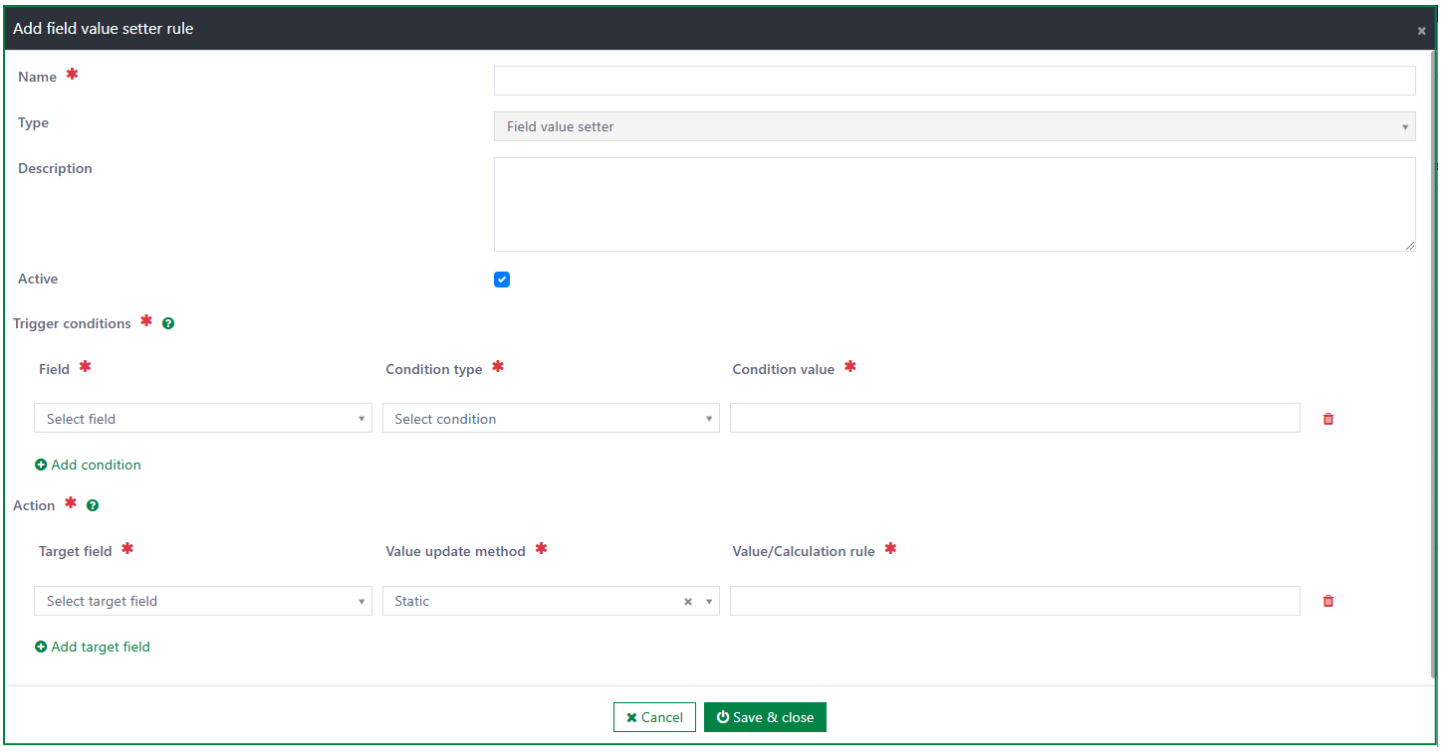


- By clicking on the **Add target field** button, new record of action is added, each record includes the following fields:
  - **Target field**
    - This field is used to select the field which is object of the action when the dependency rule is executed.
    - This field is required.
    - The dropdown list values are the fields added to the form in the Design tab. You can select a field from the list.
    - *According to the example (2) above*, the **Target field** value should be **Auxiliary documents**.
  - **Show/Hide**
    - This field is used to select whether the target field should be shown/hidden when the dependency rule is executed
    - This field is required
    - The dropdown list values include: **Show**, **Hide**
    - *According to the example (2) above,* the **Show/Hide** field value should be **Show**
  - **Enable/Disable**
    - This field is used to select whether the target field should be enabled/disabled when the dependency rule is executed
    - This field is required
    - The dropdown list values include: **Enable**, **Disable**
    - *According to the example (2) above,* the **Enable/Disable** field value should be **Enable**
  - **Required/Not required**
    - This field is used to select whether the target field should be required/not required when the dependency rule is executed
    - This field is required
    - The dropdown list values include: **Required**, **Not required**
    - *According to the example (2) above,* the **Required/Not required** field value should be **Enable**

Field value setter dependency rule

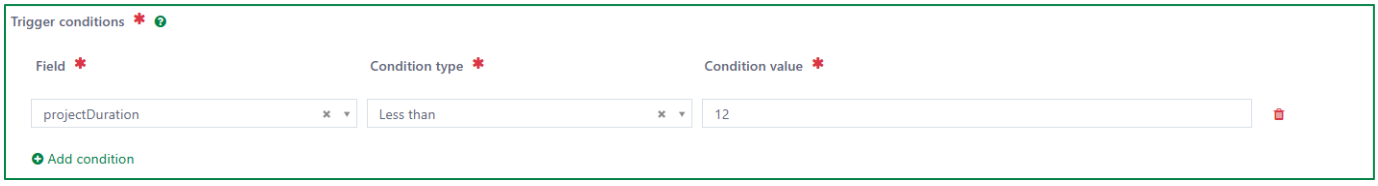


By clicking on the **Add** button and selecting the option **Field value setter**, the **Add field value setter rule** dialog is displayed.



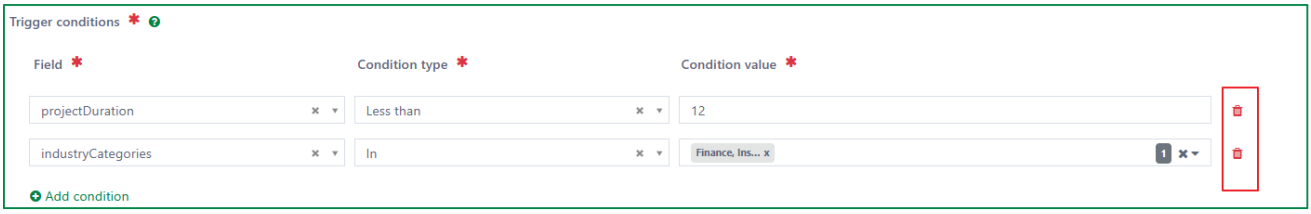
The dialog includes the following fields:

- **Name**
  - This field is required.
  - Name of the dependency rule must be unique.
- **Type**
  - The field is read-only and prefilled with type **Field value setter**.
- **Description**
  - This field is optional.
  - You can describe more information of the dependency rule.
- **Active**
  - By default, the dependency rule is active. You can deactivate it by unticking the checkbox.
  - In submission form, only the active dependency rule is affective.
- **Trigger conditions**



  - This section is to define a list of conditions which triggers the execution of the dependency rule.
  - *Given the example (3)*: ***If the value of the Project duration field is less than 12 months, value of the Reporting frequency field is set to Monthly automatically***; in this example, the **Trigger conditions** will handle the part of ***If the value of the Project duration field is less than 12 months***.
  - A dependency rule can have one or multiple trigger conditions. If multiple conditions are added, the relation between the conditions is **AND**, it means that the rule is only triggered if all conditions are met.

- You can also delete the added condition by clicking on the **Delete** button at the end of the condition row, but the rule must have at least a trigger condition.

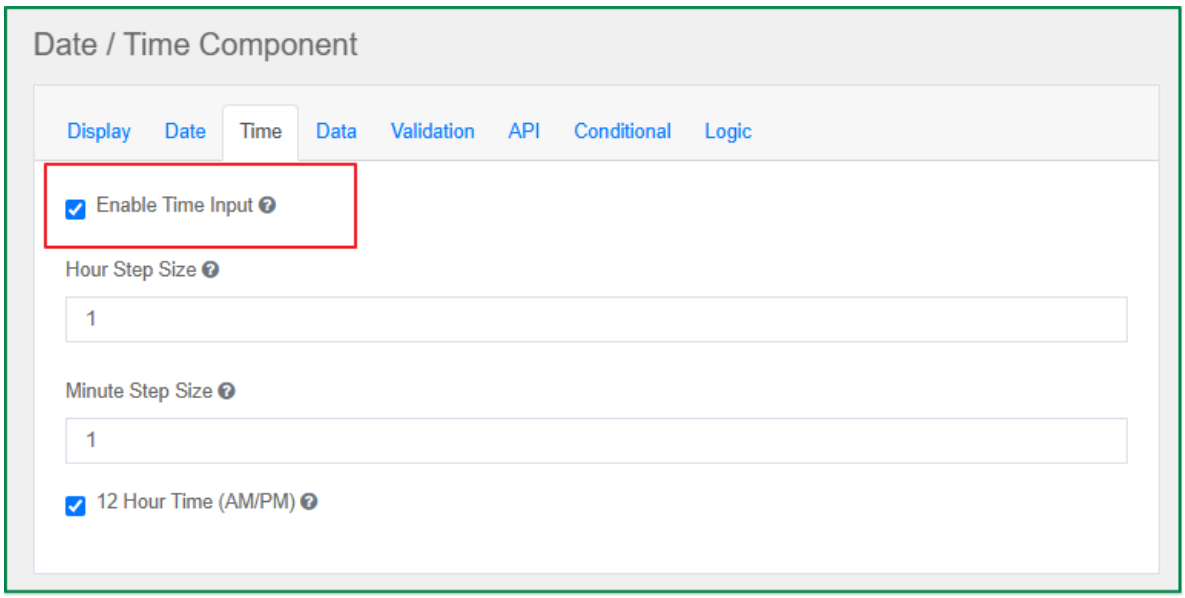


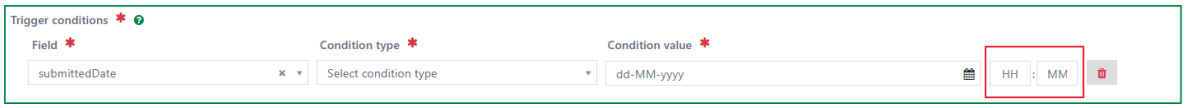
- By clicking on the **Add condition** button, new record of condition is added, each record includes the following fields:
  - **Field**
    - This field is used to select the field which is the object of the condition.
    - This field is required.
    - The dropdown list values are the fields added to the form in the Design tab. You can select a field from the list.
    - If there are multiple trigger conditions, the condition field must be unique. You cannot add 2 conditions with the same condition field.
    - The field with data type **Document** or **Guid** is not supported.
    - *According to the example (3) above*, the value should be **Project duration**
  - **Condition type**
    - This field is used to select the type of the condition.
    - This field is required.
    - The field includes a list of condition types.
    - There are 9 condition types, the list of condition types is displayed according to the data type of the selected field.
    - The mappings between condition type and data type are as the following table.

DATA TYPE	CONDITION TYPE
Array	In, Not in, Has any value
Boolean	Equal to, Not equal to, Has any value
DateTime	Equal to, Not equal to, Greater than, Greater than or equal, Less than, Less than or equal, Has any value
Double	Equal to, Not equal to, Greater than, Greater than or equal, Less than, Less than or equal, Has any value
Enum	In, Not in, Equal to, Not equal to, Has any value
ForeignKey	In, Not in, Equal to, Not equal to, Has any value
Integer	Equal to, Not equal to, Greater than, Greater than or equal, Less than, Less than or equal, Has any value
Long	Equal to, Not equal to, Greater than, Greater than or equal, Less than, Less than or equal, Has any value
String	Equal to, Not equal to, Has any value
Guid	Equal to, Not equal to, Has any value

    - *According to the example (3) above*, the **Condition type** should be **Less than**

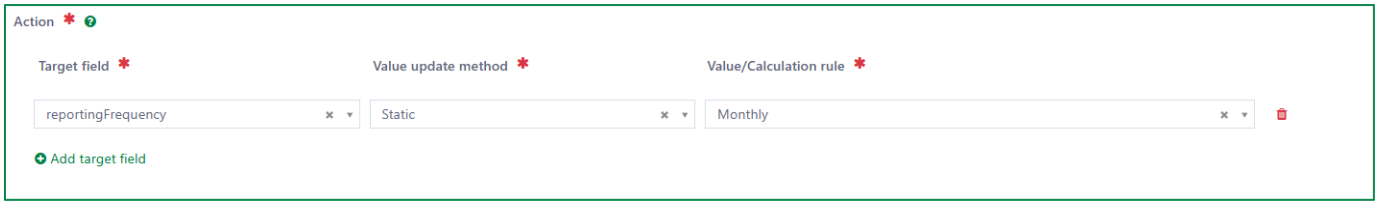
- **Condition value**
  - This field is used to define value of the condition field.
  - This field is required.
  - If the selected **Condition type** is **Has any value**, the field is disabled. It means that the rule is triggered with any value of the condition field except for the **null** value.
  - If the selected field having data type **DateTime** and the setting **Enable Time Input** of the component was enabled, the **Condition value** field format is date time. If the setting **Enable Time Input** of the component was disabled, the **Condition value** field format is date only.



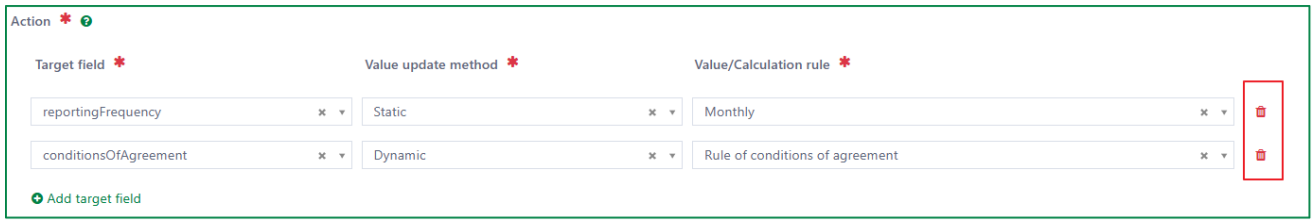


  - *According to the example (3) above*, the **Condition value** should be **12**.

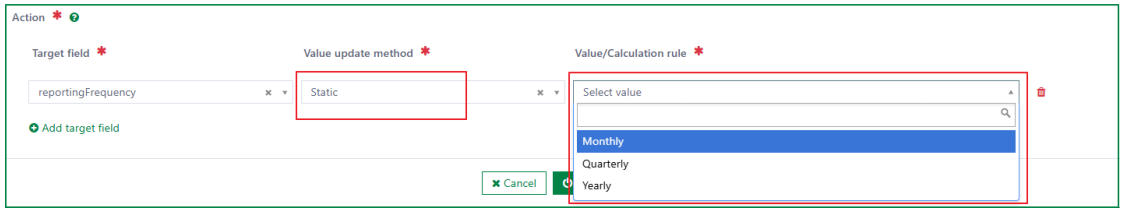
- **Actions**



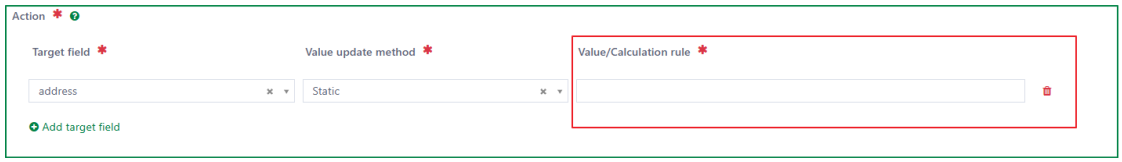
  - This section is to define the action will be executed when the dependency rule is triggered.
    - *According to the example (3) above*, this **Action** section will handle the part of ***value of the Reporting frequency field is set to Monthly automatically.***
  - You can add one or multiple target fields.
  - You can also delete the added target fields by clicking on the **Delete** button at the end of the row, but the rule must have at least a target field.



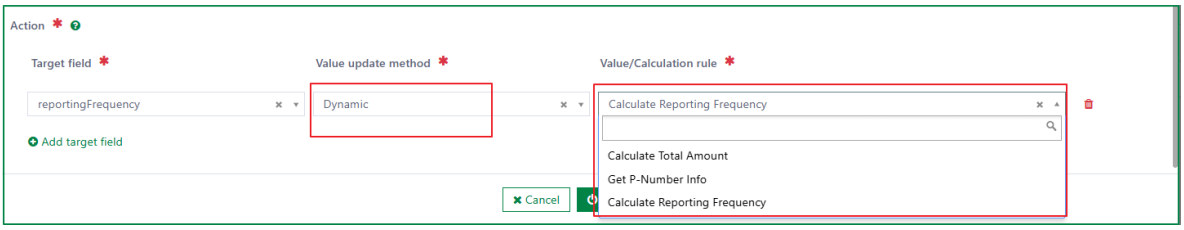
- By clicking on the **Add target field** button, new record of action is added, each record includes the following fields:
  - **Target field**
    - This field is used to select the field which is object of the action when the dependency rule is executed.
    - This field is required.
    - The dropdown list values are the fields added to the form in the **Design** tab. You can select a field from the list.
    - The field with data type **Document** or **Guid** is not supported
    - *According to the example (3) above*, the **Target field** value should be **Reporting frequency**.
  - **Value update method**
    - This field is used to select the method of updating value for the target field when the dependency rule is triggered.
    - This field is required.
    - The dropdown list values include: **Static**, **Dynamic**.
    - Values of the **Value/Calculation rule** field depends on the selected update method.
    - *According to the example (3) above*, the **Value update method** value should be **Static**.
  - **Value/Calculation rule**
    - This field is used to select the static value which will be set for the target field, or to select the calculation rule which calculates the value of the target field when the dependency rule is triggered.
    - If the **Value update method** field value is **Static**, you can input/select a value of the target field according to the data type of the field:
      - If the data type is **Array**, **Boolean**, **Enum**,the **Value update method** field is a a dropdown list including values of the target field, and you can select a value from the list.



      - If the data type is **String**, **DateTime**, **Integer**, **Long**, **Double**, **Guid**, the **Value update method** field is a input field, you can input the value.

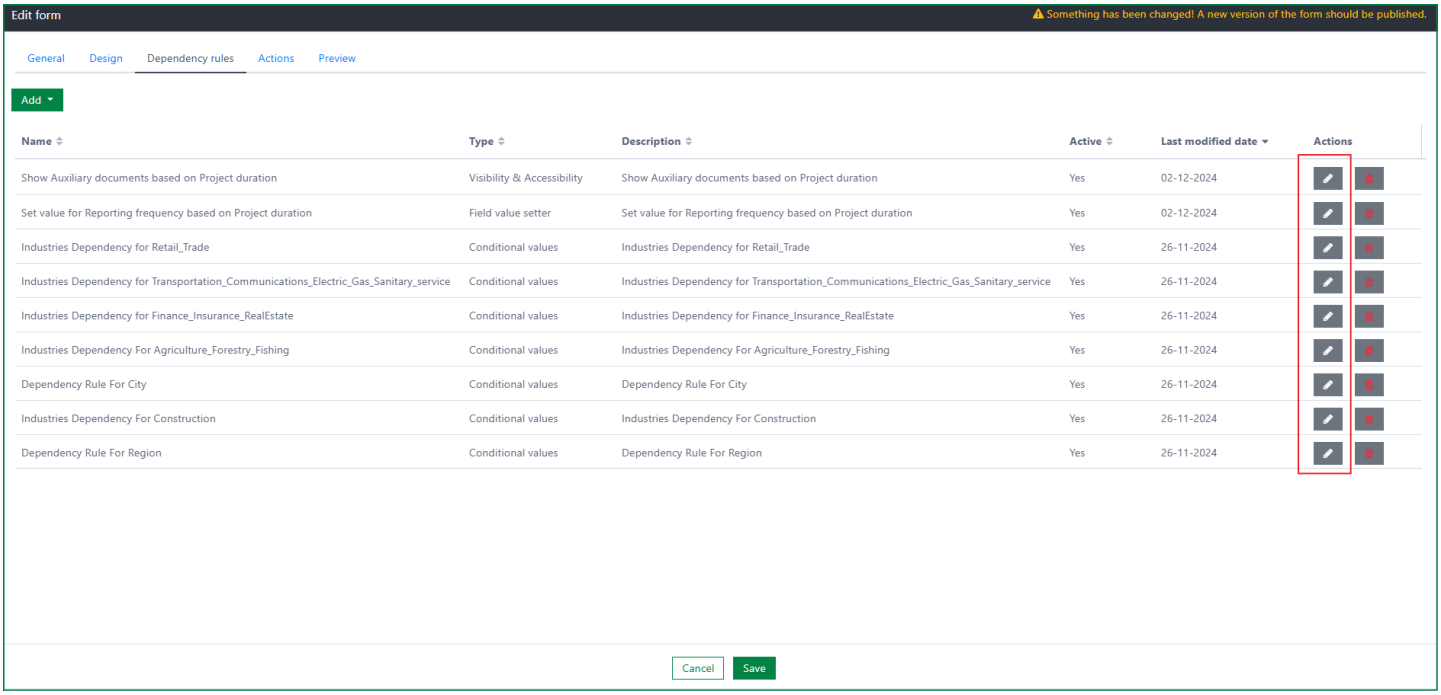


      - If the data type is **ForeignKey**, it depends on the field that it refers to.
    - If the **Value update method** field value is **Dynamic**, the **Value update method** field is a dropdown list including a list of active rules with type **Form field value setter dependency rule** configured in the **Rules** page.



- *According to the example (3) above*, with the update method as **Static**, the **Value/Calculation rule** field value should be **Monthly**.

Edit dependency rule

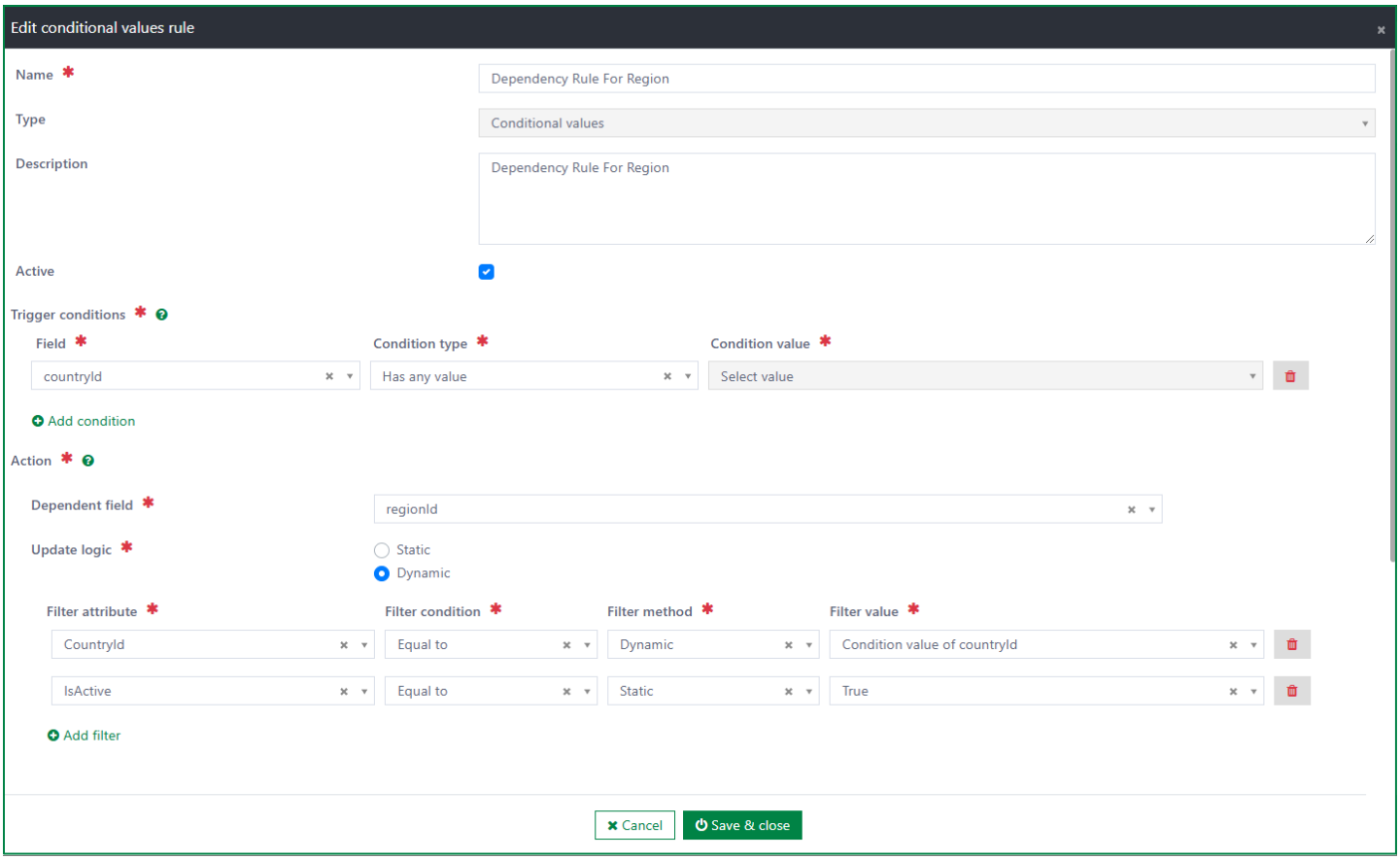


By clicking on the **Edit** button at the row of an item in the list of dependency rules in the **Dependency rules** tab, the corresponding dialog of editing dependency rule is opened according to the dependency rule type.

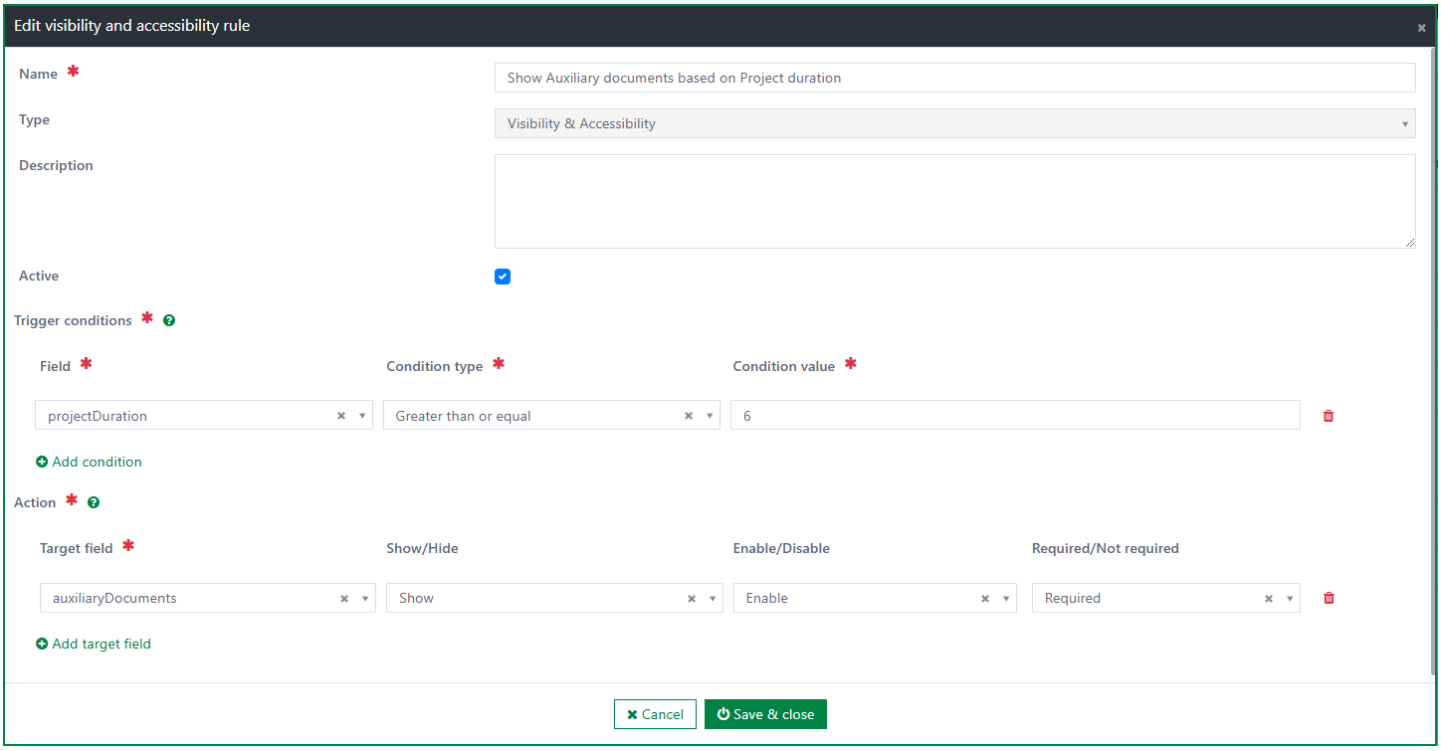
You can edit the following fields in the dialog:

- **Name**
- **Description**
- **Active**
- **Trigger conditions**
- **Action**

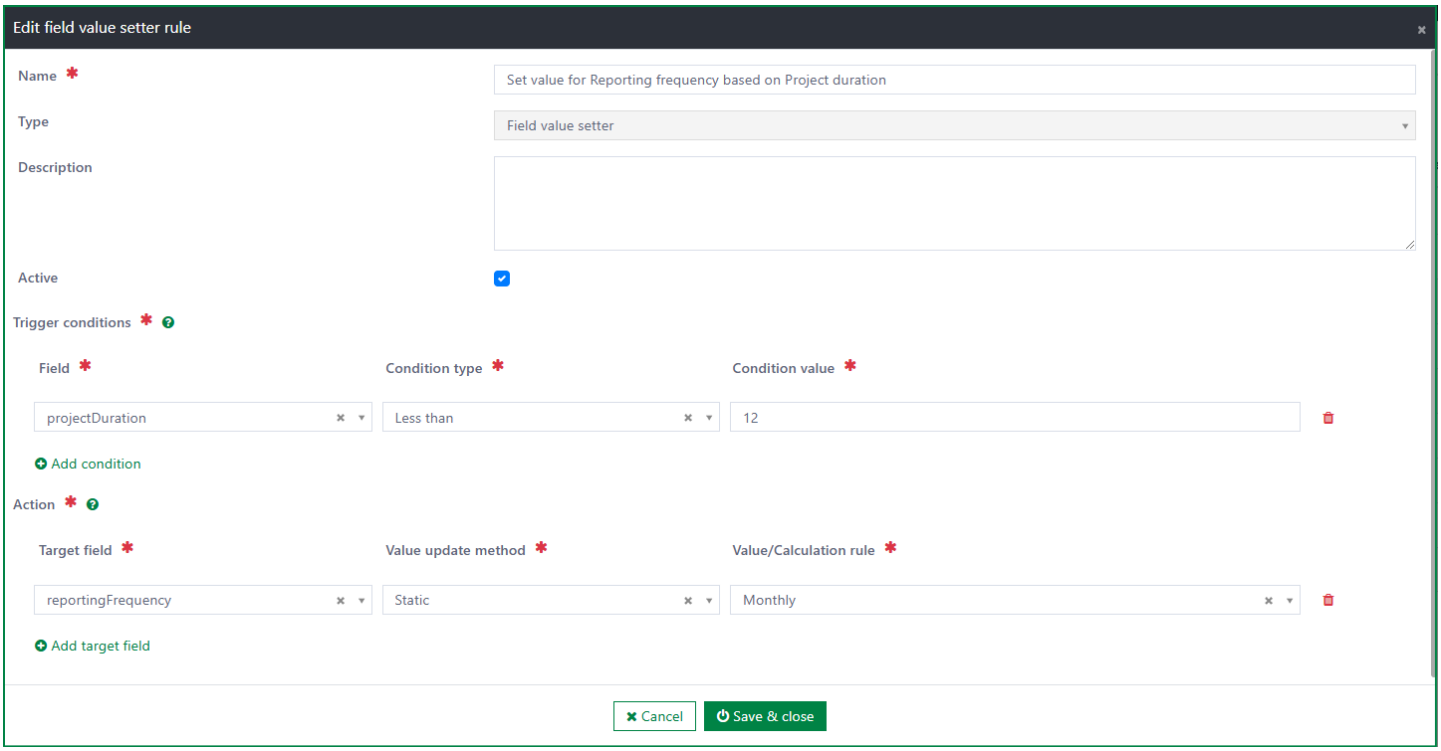
***Edit condition values rule** dialog*



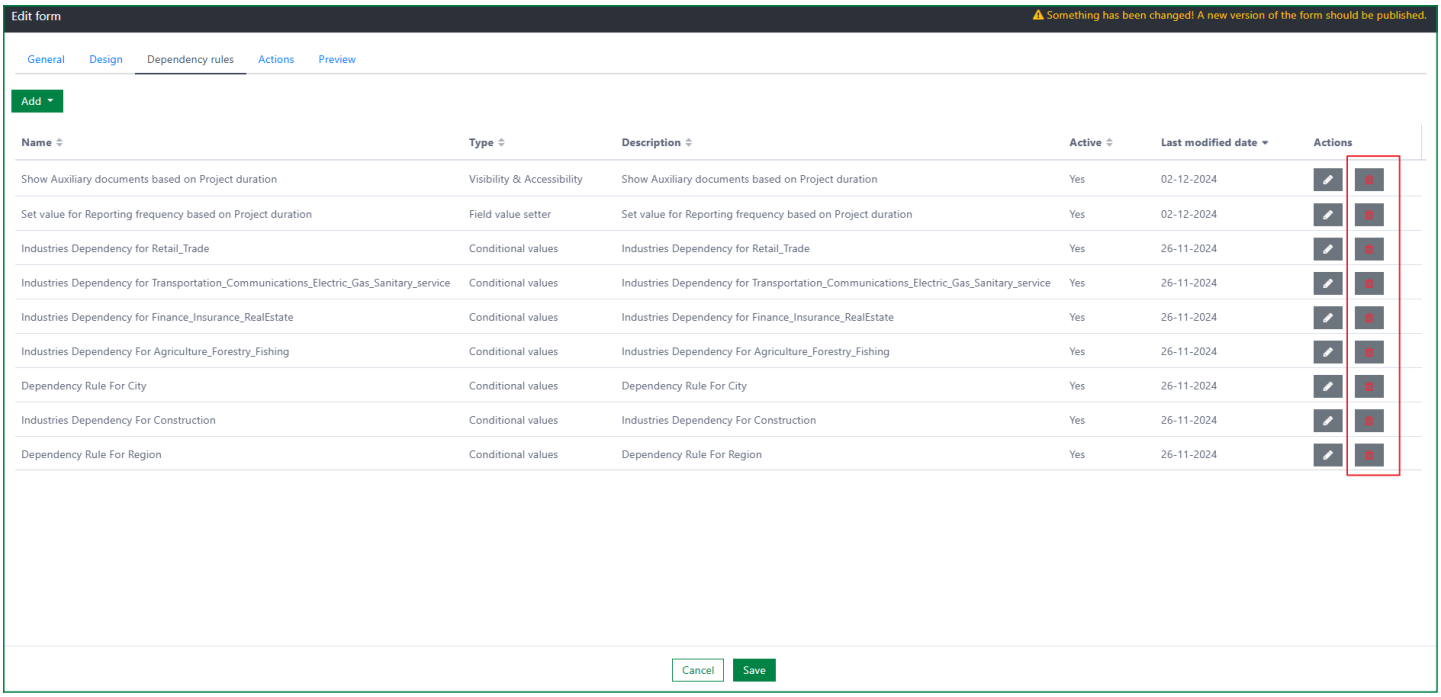
Edit visibility and accessibility rule dialog



Edit field value setter rule dialog

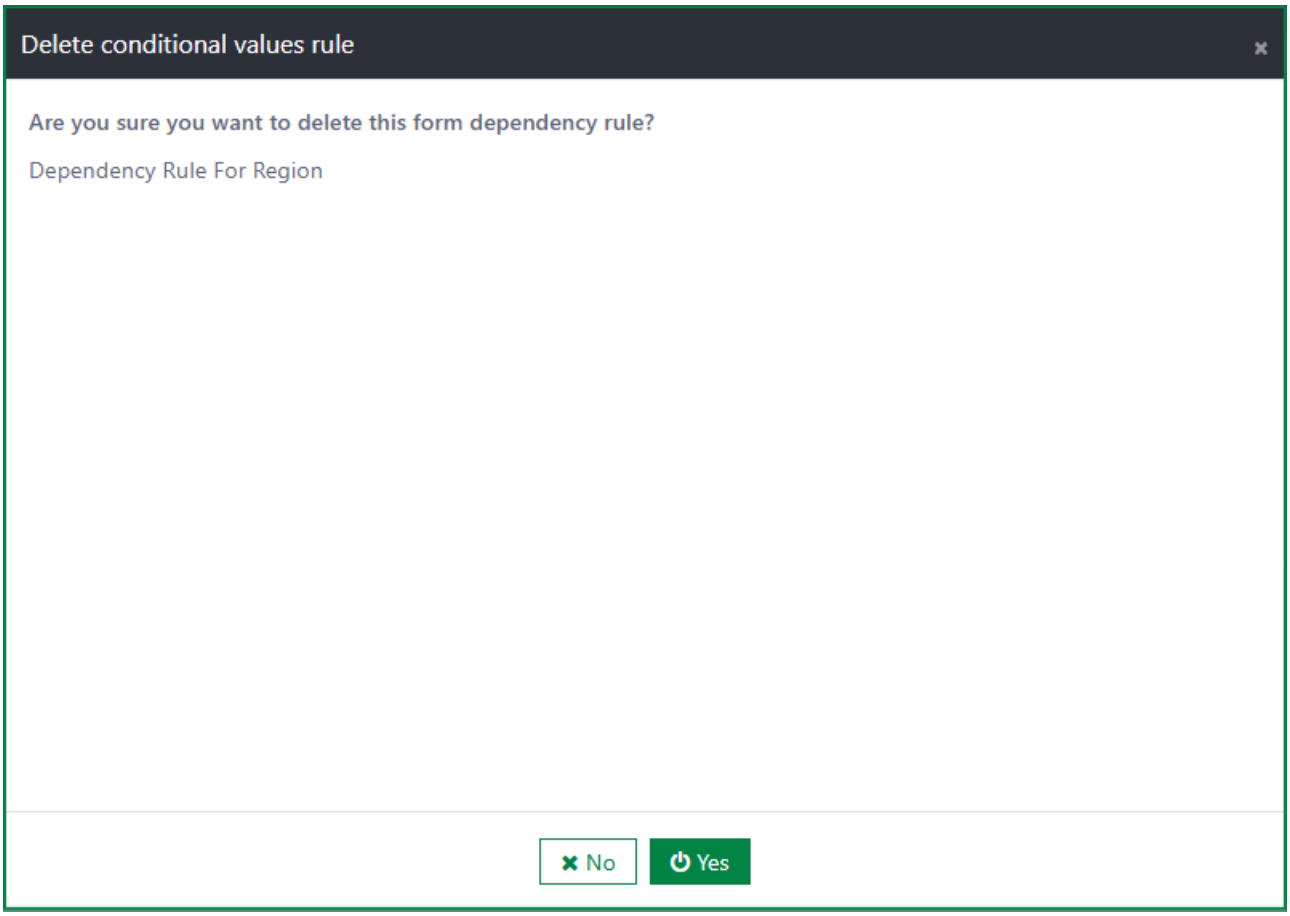


Delete dependency rule



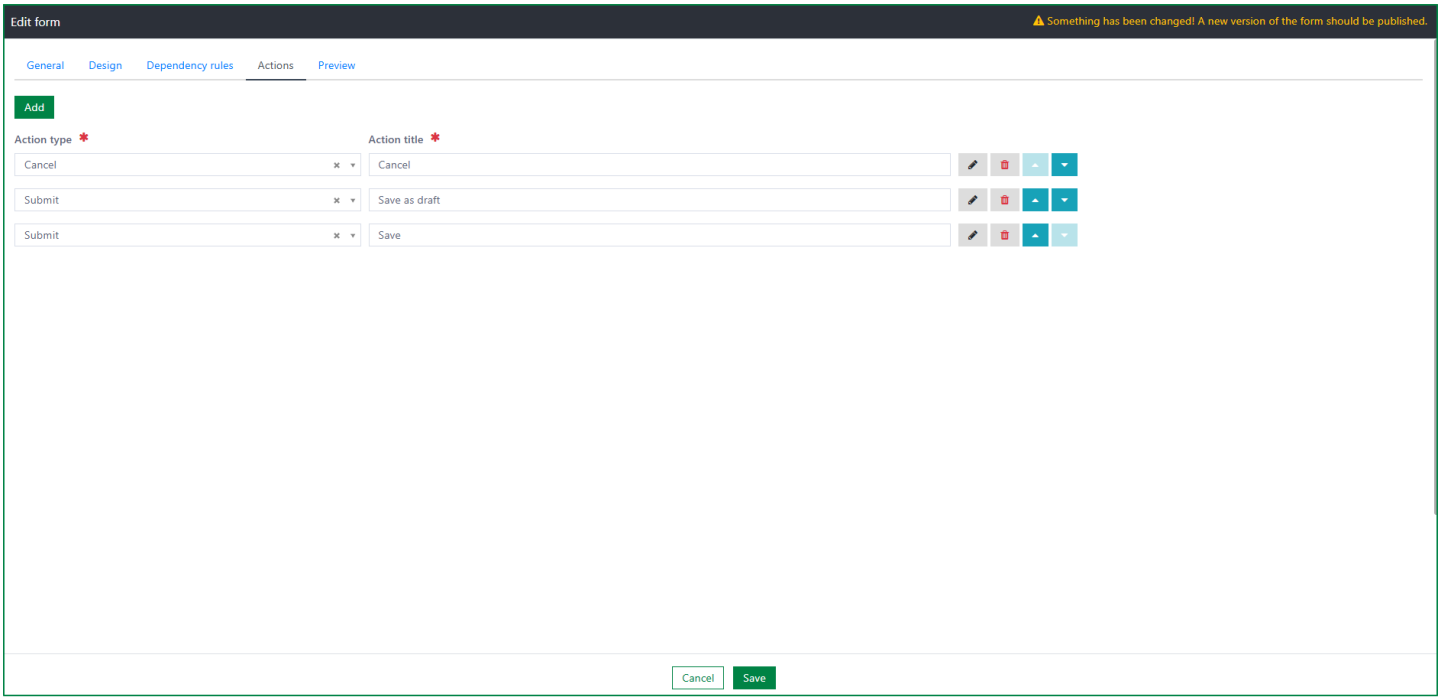
By clicking on the **Delete** button at the row of an item in the list of dependency rules in the **Dependency rules** tab, the confirmation dialog for the deletion is displayed.

- If you select the **No** button, the deletion will be cancelled.
- If you select the **Yes** button, the dependency rule is deleted immediately.



After being deleted, the deleted dependency rule is still effective on the submission form until you publish the form again.

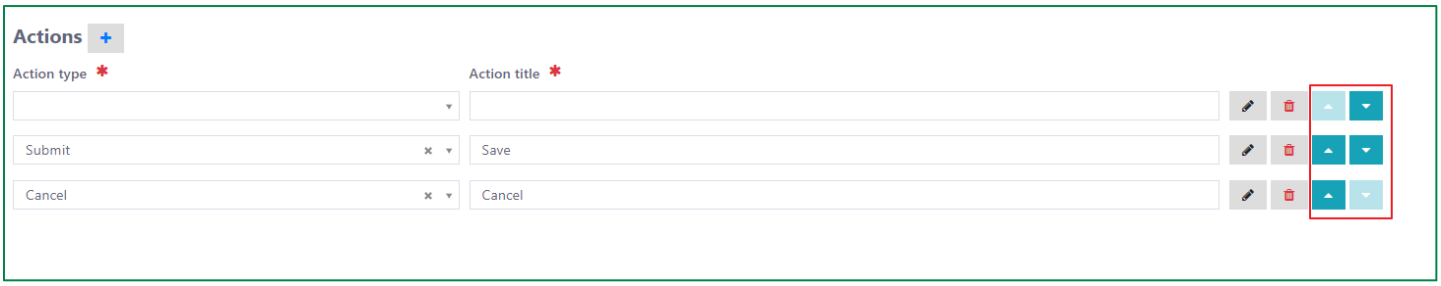
Actions



The Actions tab includes a list of created actions which are used for the form.

On the list, the form action is displayed with information like: **Action type**, and **Action title**.

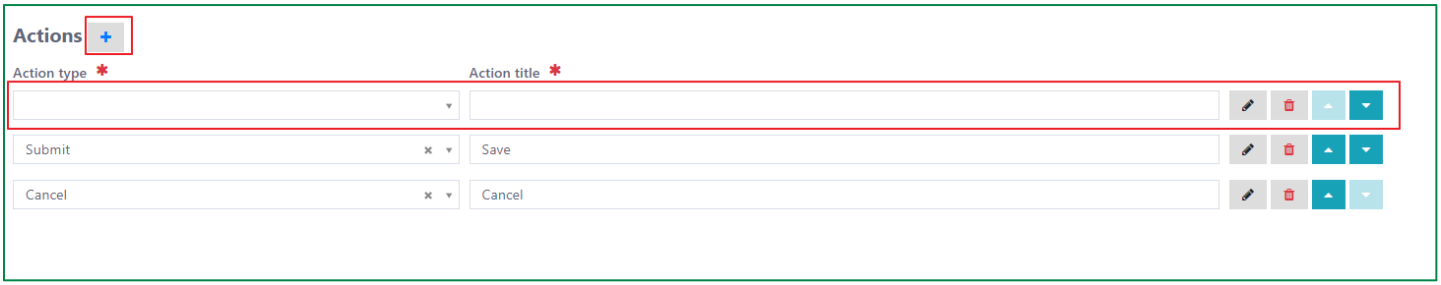
Order of action buttons on the render form from left to right depends on the order of actions on the list from top to bottom. To changing order of the form actions, you can use the up/down arrow at the right of each action.



You can also add new form action, edit form action, and delete form action.

Add new form action

By clicking on the **Add** button, a new row for new form action will be added at the top of the form actions list.

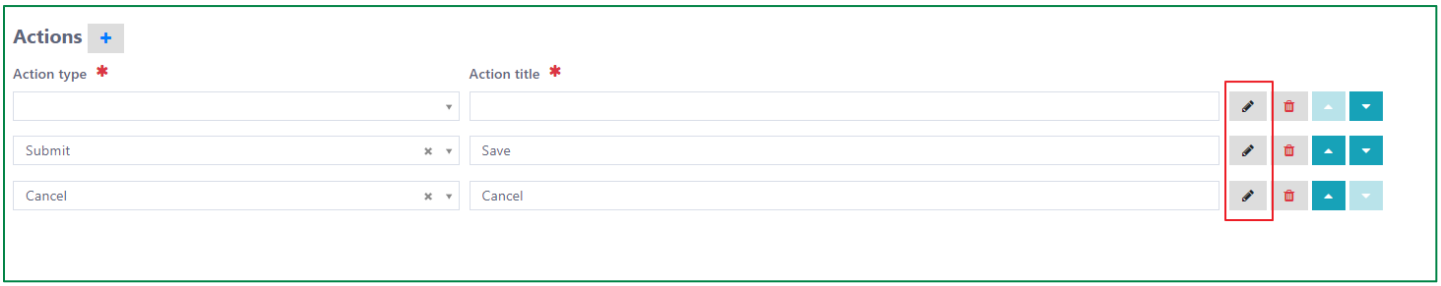


You must select the **Action type**, and fill in the **Action title** for the new action.

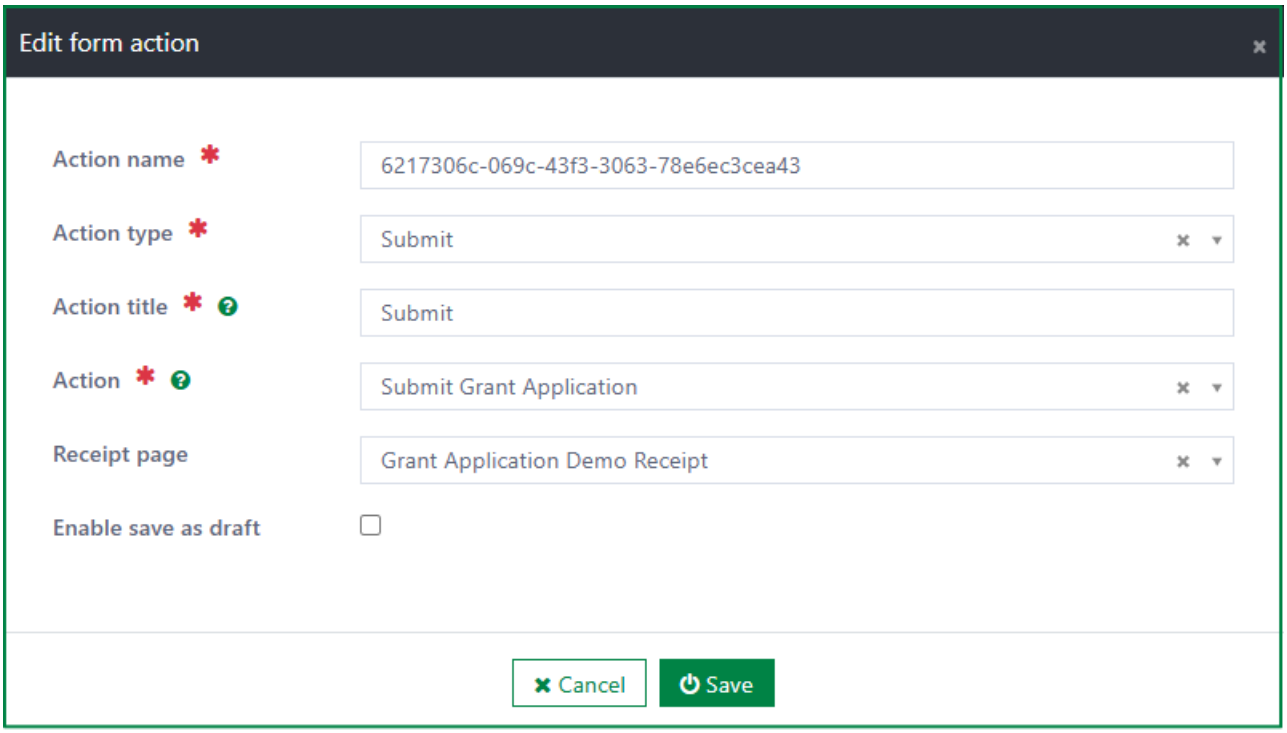
The Action type list includes 2 types: **Submit**, and **Cancel**.

According to the selected **Action type**, other relevant configurations will be displayed in the **Edit form action** dialog.

Edit form action



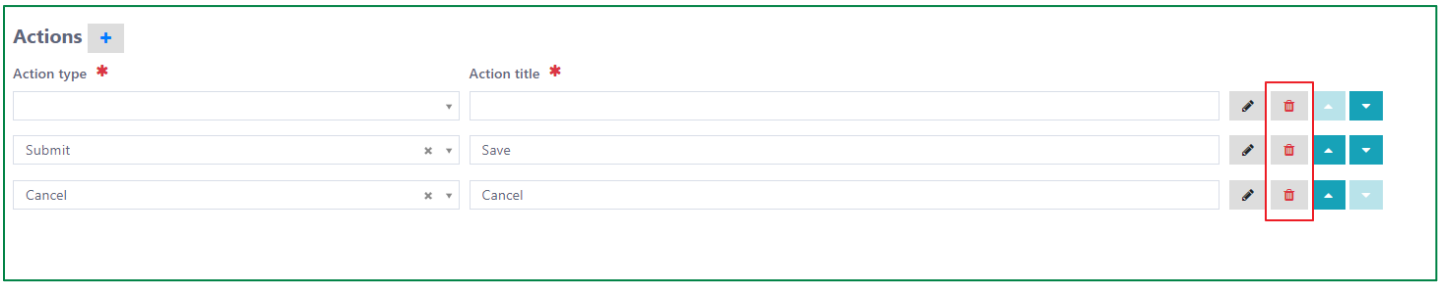
By clicking on the **Edit** button at each form action row, the **Edit form action** dialog will be opened.



The dialog includes the following fields:

- **Action name**
  - This field is required.
  - Action name must be unique.
- **Action type**
  - This field is required.
  - You must select an item from the list. There are 2 action types in the list: **Submit**, and **Cancel.**
- **Action title**
  - This field is required
  - Action title is name of the button which is displayed on rendered form UI.
  - In case the form is multilingual, you should fill the text resource key in the field.
- **Action**
  - This field is only displayed if the **Action type** is **Submit**.
  - The field is required after it is displayed.
  - You must select an action from the list. List of actions is retrieved from the **Form actions** page.
- **Receipt page**
  - The receipt page is displayed after the rendered form is submitted successfully.
  - This field is optional and only displayed if the **Action type** is **Submit** and the **Enable save as draft** checkbox is unticked.
  - You can select a receipt form from the list. List of receipt forms is retrieved from the **Receipt forms** page.
  - The receipt form is only displayed after is published.
- **Enable save as draft**
  - This field is only displayed if the **Action type** is **Submit**.
  - By default, the **Save as draft** function is disabled. You can enable it by ticking on the checkbox.
  - With this function, you can save the filled data on the rendered form temporarily as a draft before submitting it.

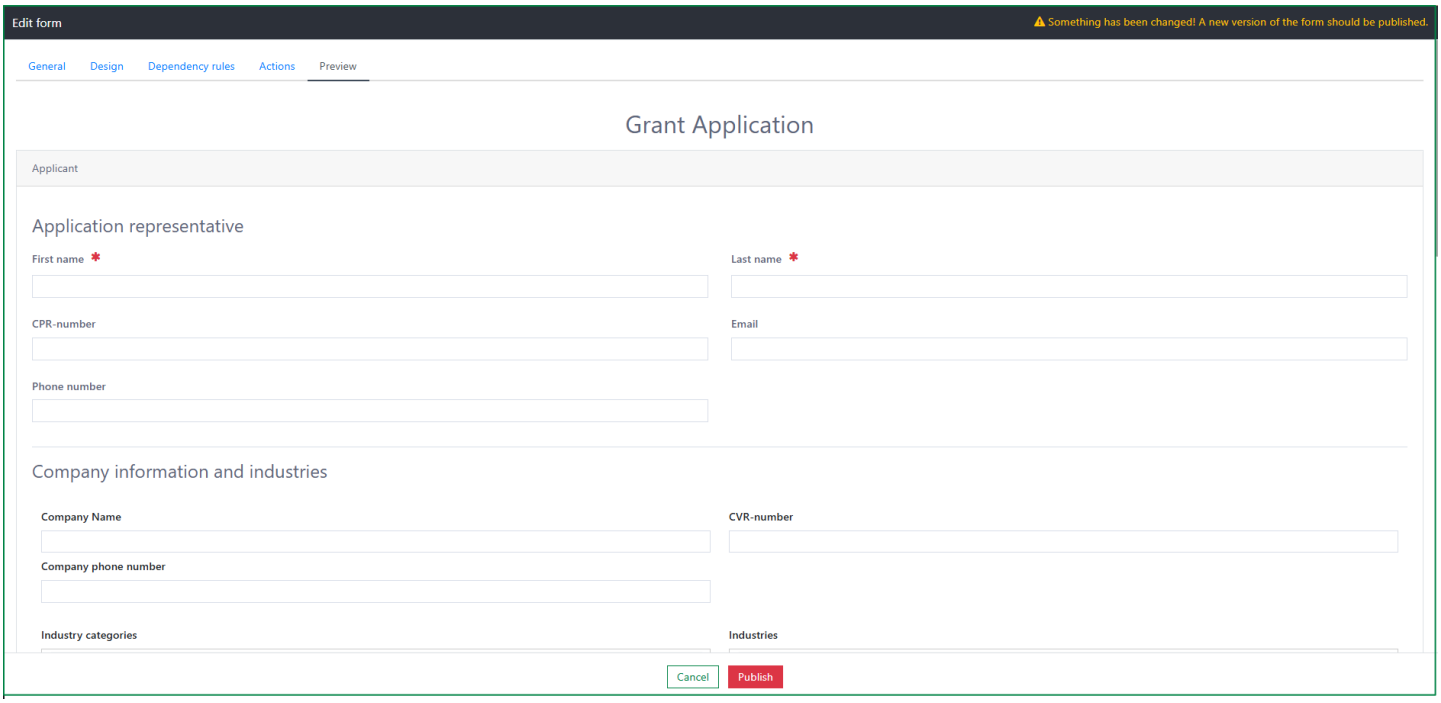
Delete form action



By clicking on the **Delete** button at the right of the action, the form action will be deleted.

The deletion function will be disabled if it is only 1 form action existing in the list. The form must have at least 1 action.

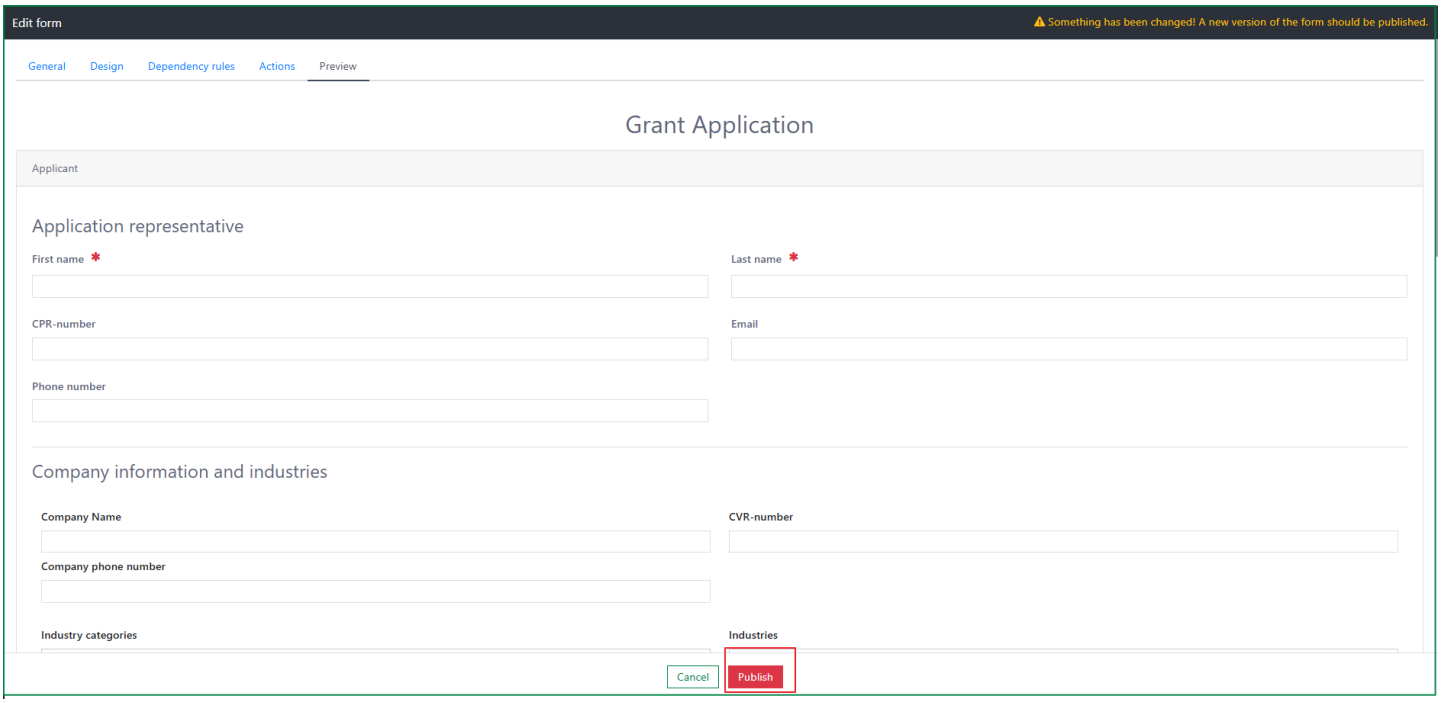
Preview



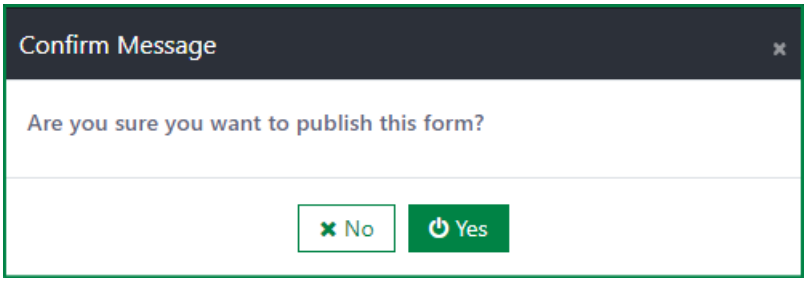
After completing configuring the form action, you must save the changes before going to the **Preview** tab to preview the rendered form and publish it.

In the preview mode, you can view how the designed form displays after being published. You can also try filling the form and check the form behavior.

At this preview step, changes on the form have not been available on the public site yet. It is only available after the form is published successfully. You can publish the form with the **Public** button in the **Preview** tab.

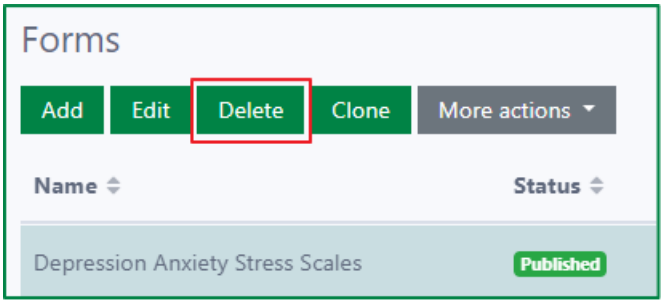


By clicking on the **Public** button, a confirmation dialog will be opened.



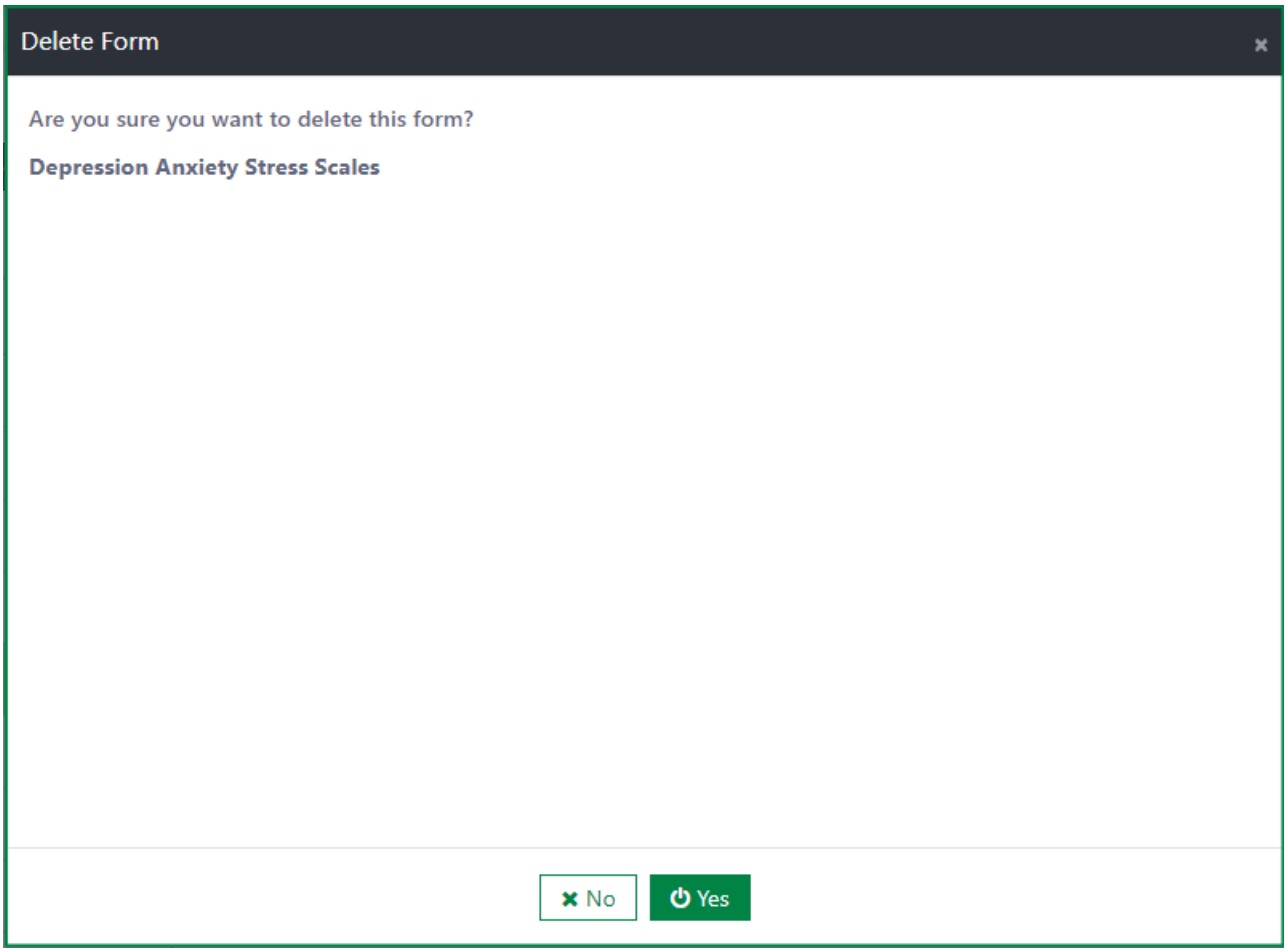
If you select the **Yes** button, the form will be published immediately. If you select the **No** button, the publication will be cancelled.

Delete a form



The **Delete** button is only enabled after a form is selected.

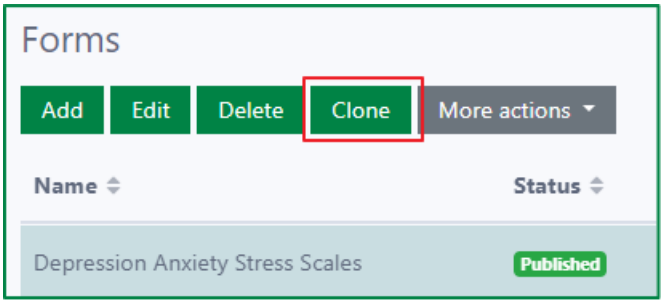
By on the **Delete** button, the deletion confirmation dialog will be opened.



If you select the **Yes** button, the form will be deleted immediately. If you select the **No** button, the deletion will be cancelled. After being deleted, the deleted form will no longer be accessible in the public site.

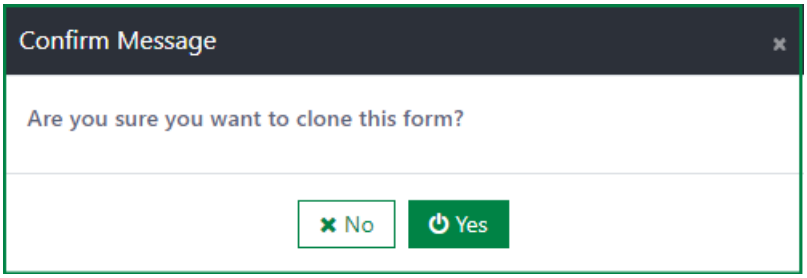
Other actions

Clone new form



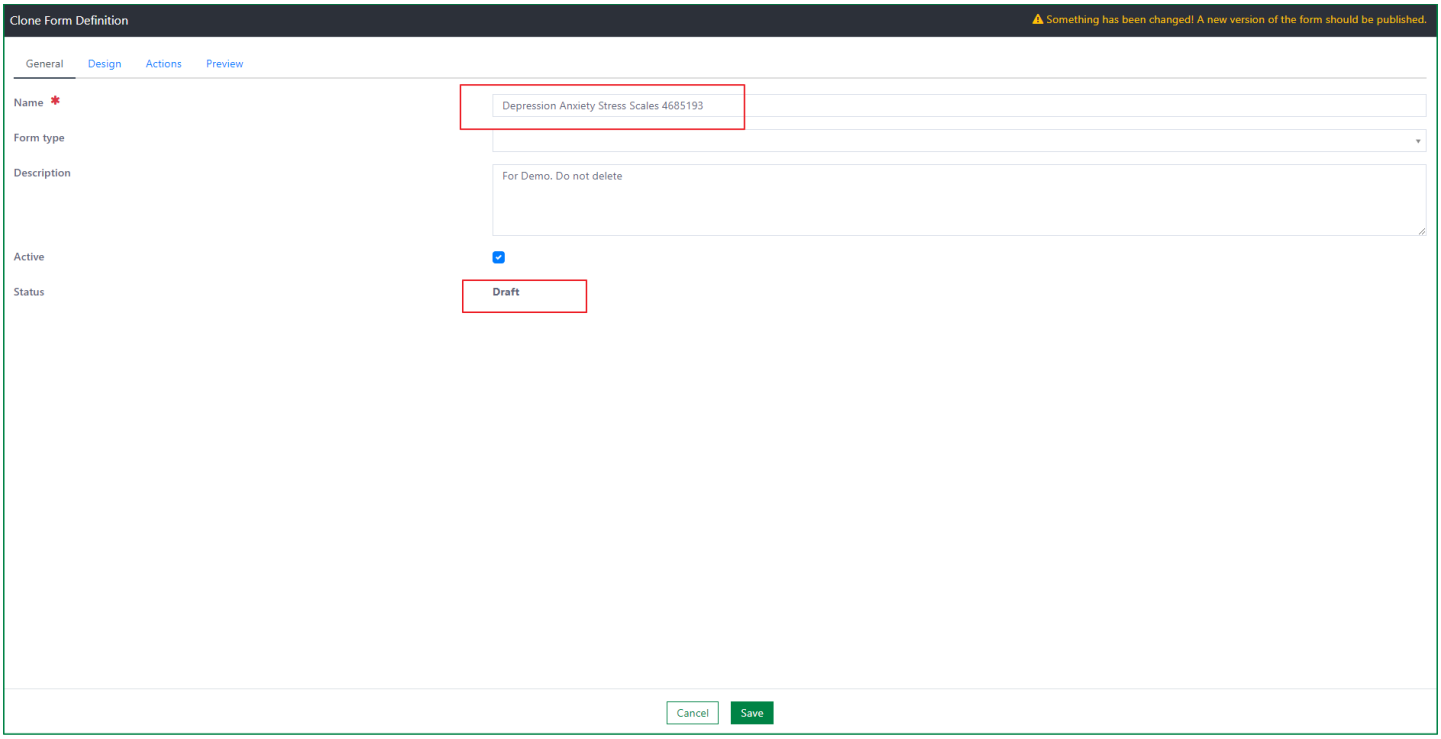
The **Clone** button is only enabled after an item on the list of forms is selected.

By clicking on the **Clone** button, a confirmation dialog for cloning will be opened.



If you select the No button, the cloning process will be cancelled. If you select the Yes button, the cloning process will be started and it will take a while to complete.

After the cloning process is completed, the new form dialog will be opened. Name of the new form includes name of the original form and the suffix. Status of the new form is **Draft** by default.

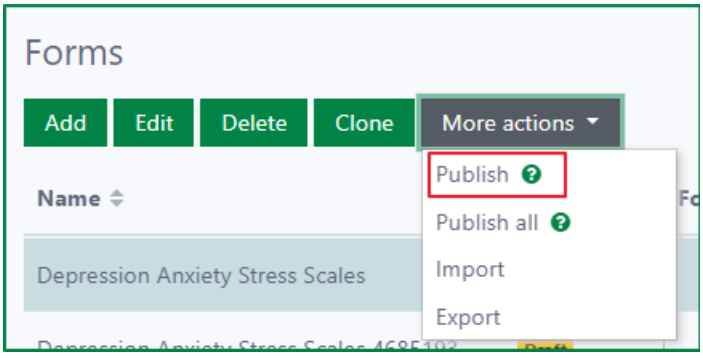


The new form inherits the form design and all configurations of the original form.

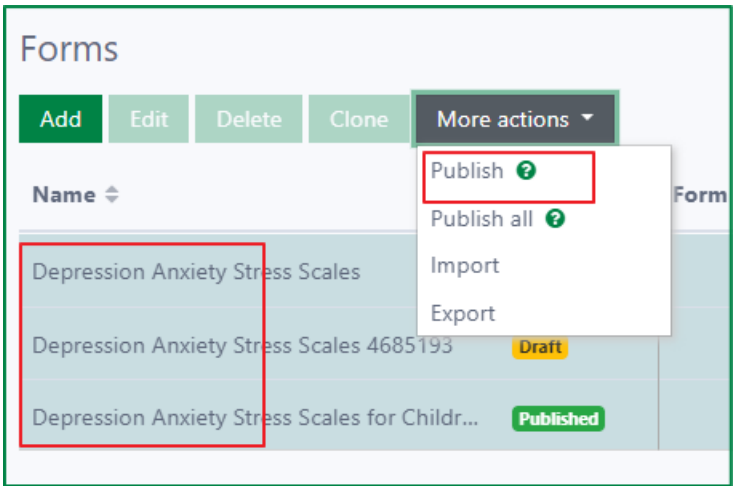
Publish form

Besides publishing form in the **Edit form** dialog. You can also publish a form, multiple forms or all forms but does not have to open each form to publish it.

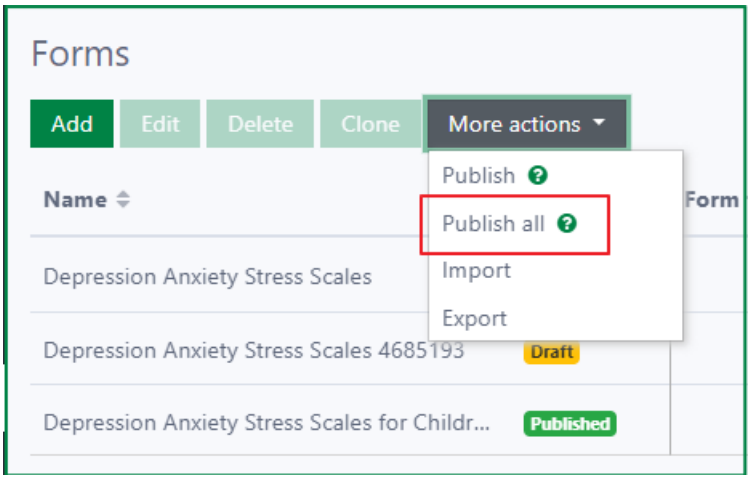
- **Publish a form**
  - To publish a form, select a form then click on the **Publish** button, the selected form will be published immediately.



- **Publish multiple forms**
  - To publish multiple forms at one time, select multiple forms then click on the **Publish** button, the selected forms will be published immediately.

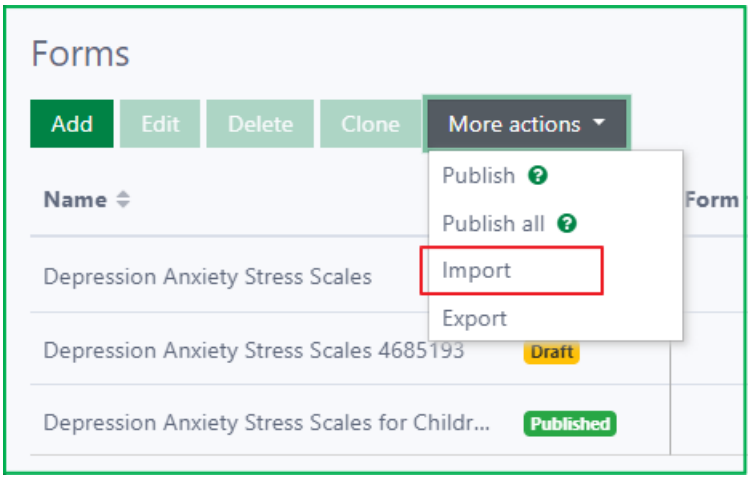


- **Publish all forms**
  - To publish all forms, click on the **Publish all** button, then, a confirmation dialog of publication will be opened. If you select the **Yes** button, all forms of which status is **Draft** will be published immediately. If you select the **No** button, the publication will be cancelled.

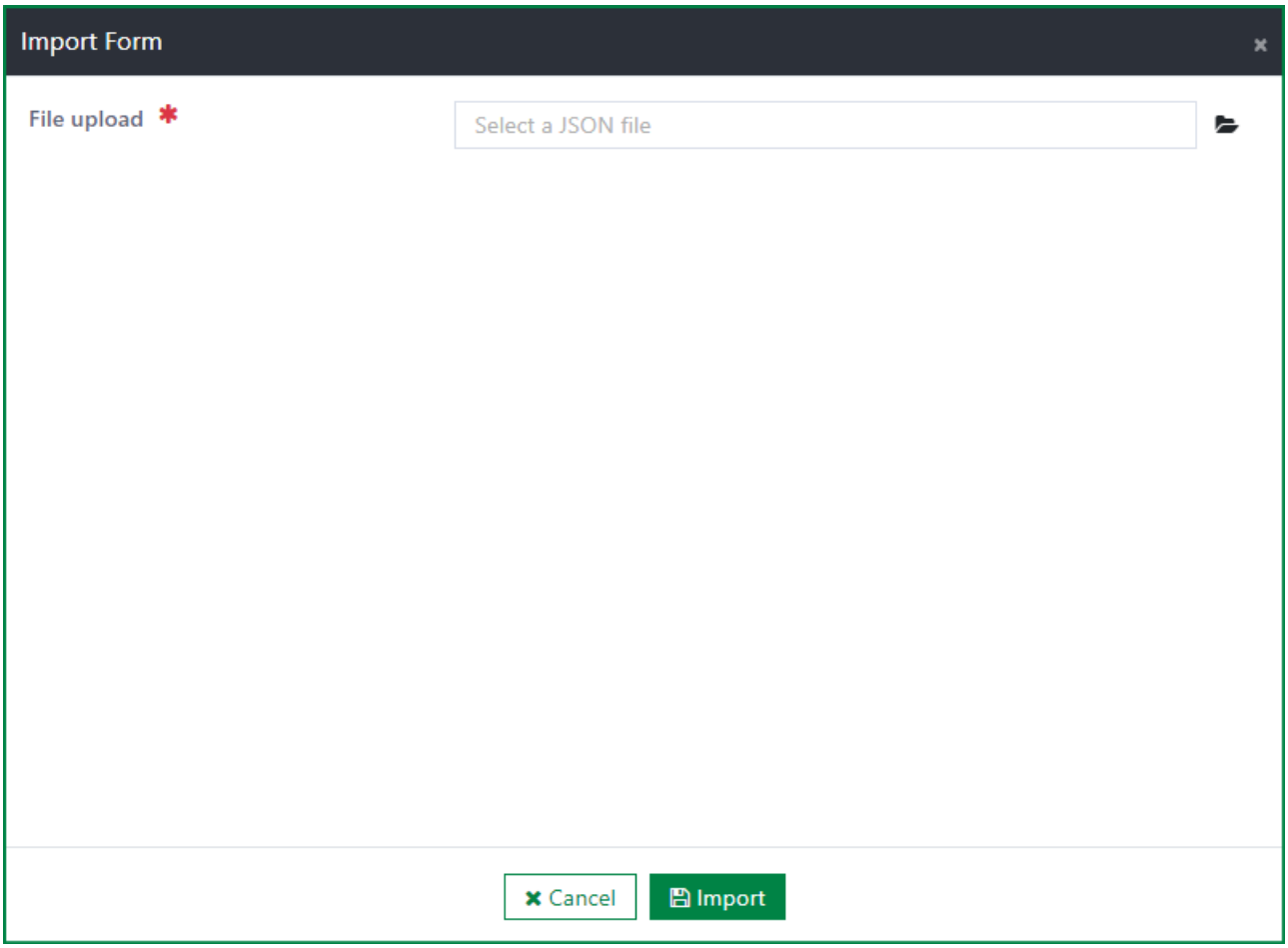


Import form

Besides creating forms manually, you can import a list of forms from a JSON file.

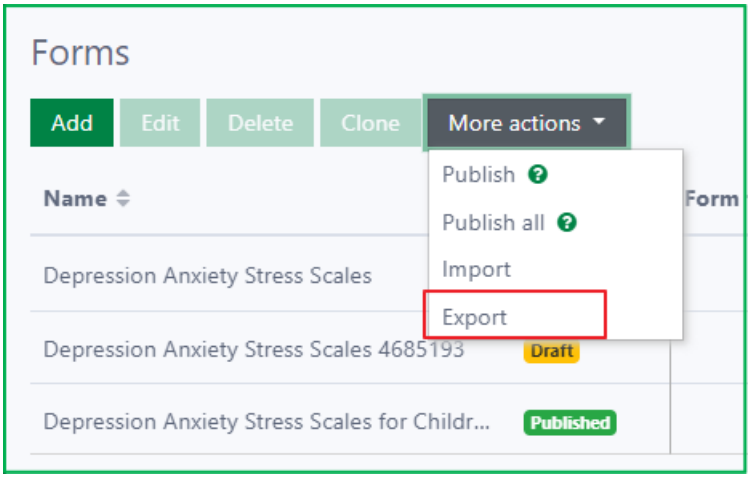


By clicking on the Import button, the Import form actions dialog will be opened. Then, you can select a JSON file and upload it via the **File upload** field.



After completing the import process, the forms will be created and displayed in the **Forms** list properly.

Export form



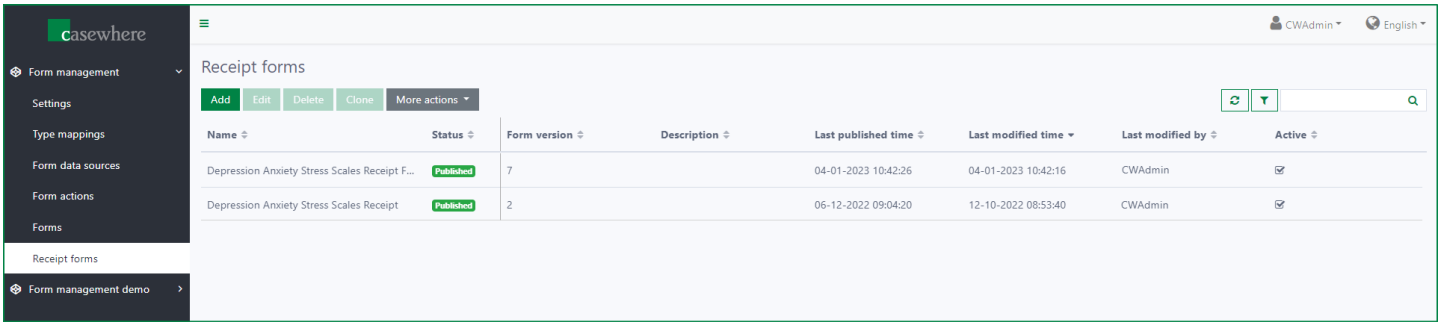
By clicking on the **Export** button, the list of forms will be exported immediately as JSON file. You can also export one or multiple forms instead of whole list by selecting the forms, then clicking on the **Export** button.

Component: Form management - Receipt form

Introduction

The receipt form is used to displayed after the end user submits the form. A receipt form is a digital paper of message proving that information has been submitted successfully.

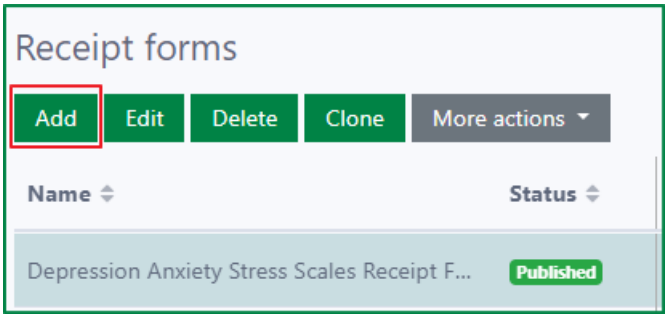
The receipt forms are displayed on the list with information like: **Name, Status, Description, Last published time, Last modified time, Last modified by**, and **Active** status.



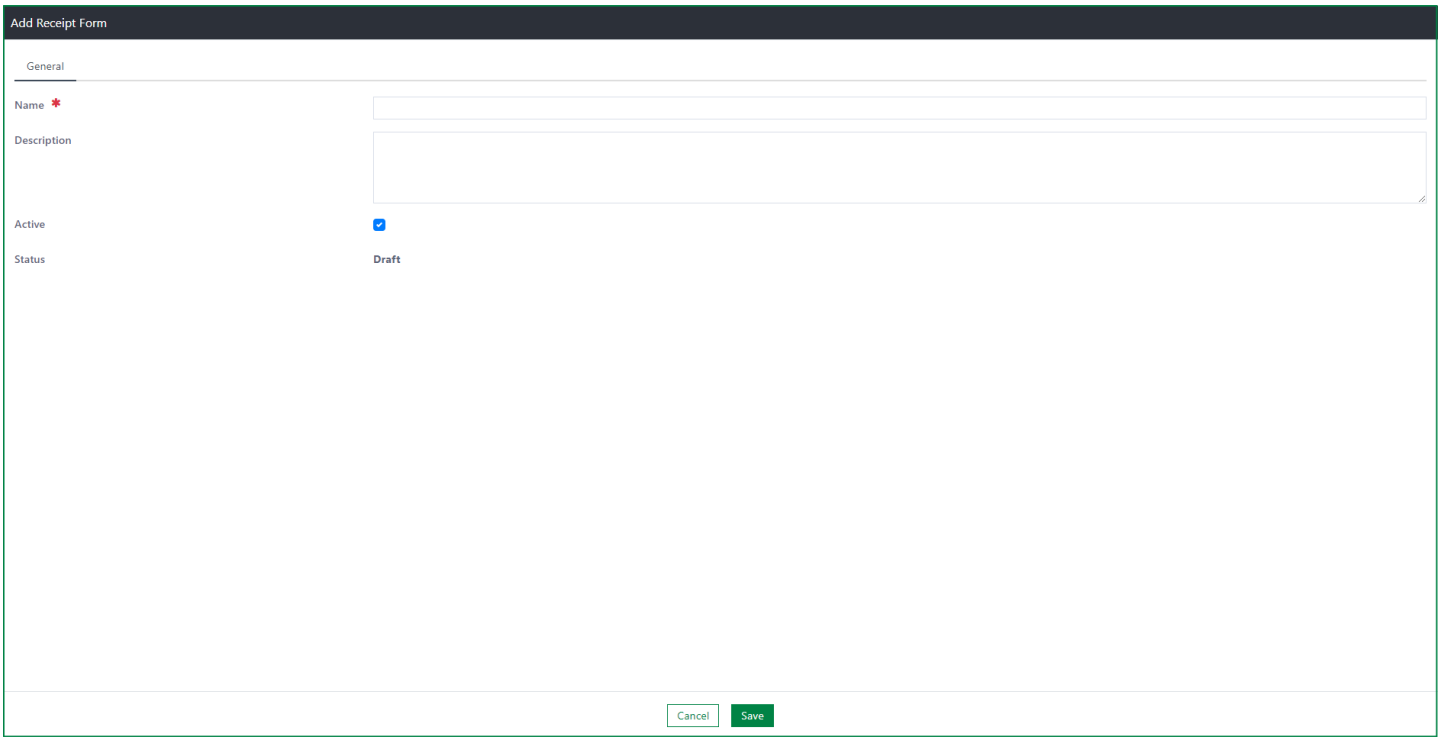
You can add, edit, delete the receipt forms. You can also clone new receipt form from an existing one, publish forms, import forms from JSON file, and export it to a JSON file.

Guideline

Add receipt form



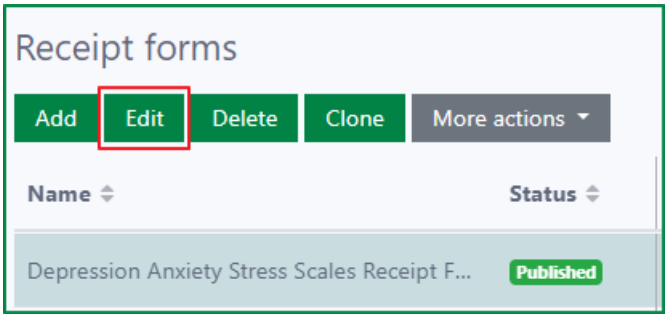
By clicking on the **Add** button, the **Add new receipt form** dialog will be opened.



The dialog includes **General** tab with the following fields:

- **Name**
  - This field is required.
  - Name of the receipt form must be unique.
- **Description**
  - This field is optional.
  - You can describe more information of the receipt form.
- **Active**
  - By default, the receipt form is active. You can deactivate it by unticking the checkbox.
  - An inactive receipt form will not be available in the **Receipt page** configuration in the Action configuration of Form.
- **Status**
  - The default status of a new receipt form is **Draft**.
  - A form can has one of these status: **Draft**, and **Published**.
    - **Draft:** A receipt form has **Draft** status when it is newly created, or it is edited after being published.
    - **Published:** A receipt form has **Published** status after it is published successfully.

Edit receipt form

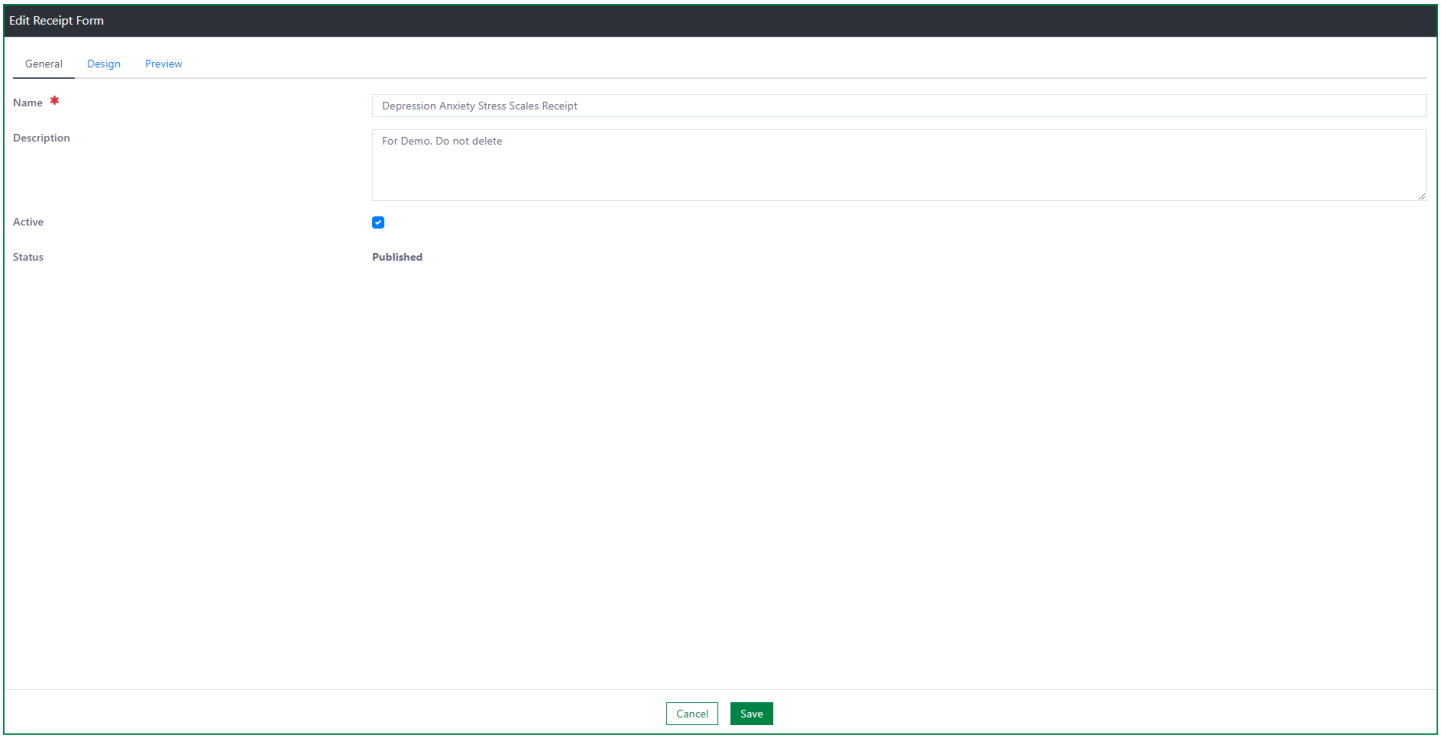


The **Edit receipt form** dialog will be opened after a new form is created successfully or via the **Edit** button,

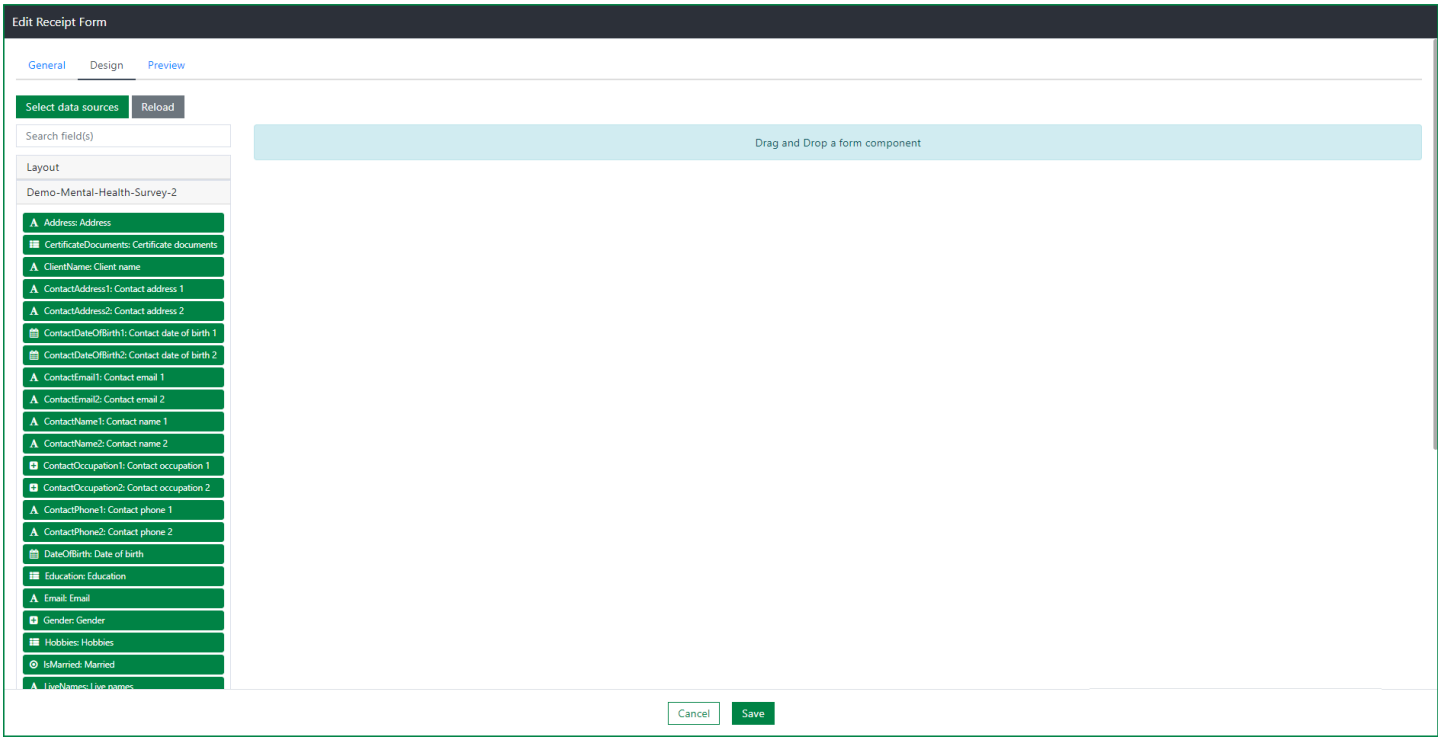
The **Edit receipt form** dialog includes 3 tabs: **General, Design,** and **Preview**.

General

This tab displays all saved general information of the receipt form, like: **Name, Description, Active,** and **Status.** These fields are available to update.



Design

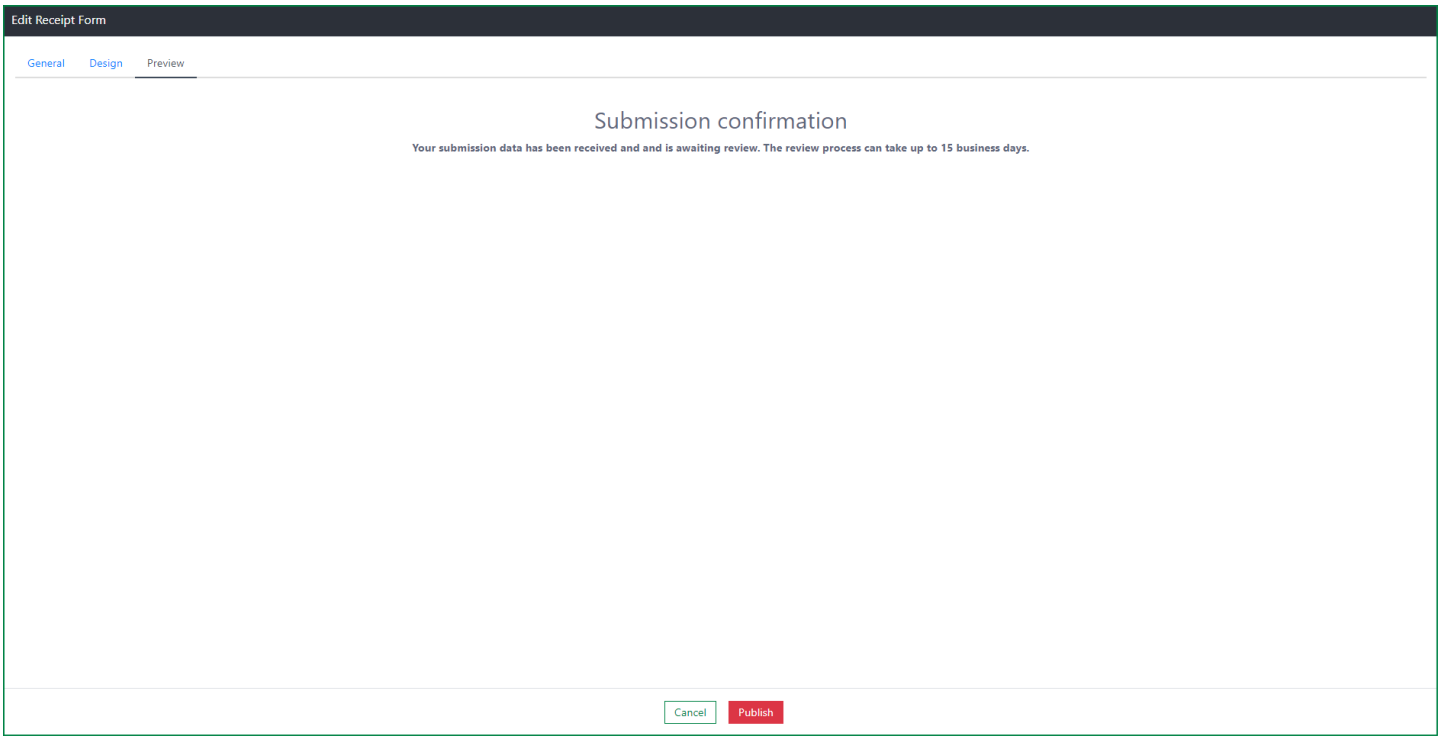


The **Design** tab includes 2 sections: fields list, and form design area.

The fields list includes 2 types of field: layout fields from **Form.io**, and fields from form data source.

Similar to the Form design, you must select the form data sources, then drag and drop the fields to design the receipt form. You can explore more in this article Component: Form Management - Forms

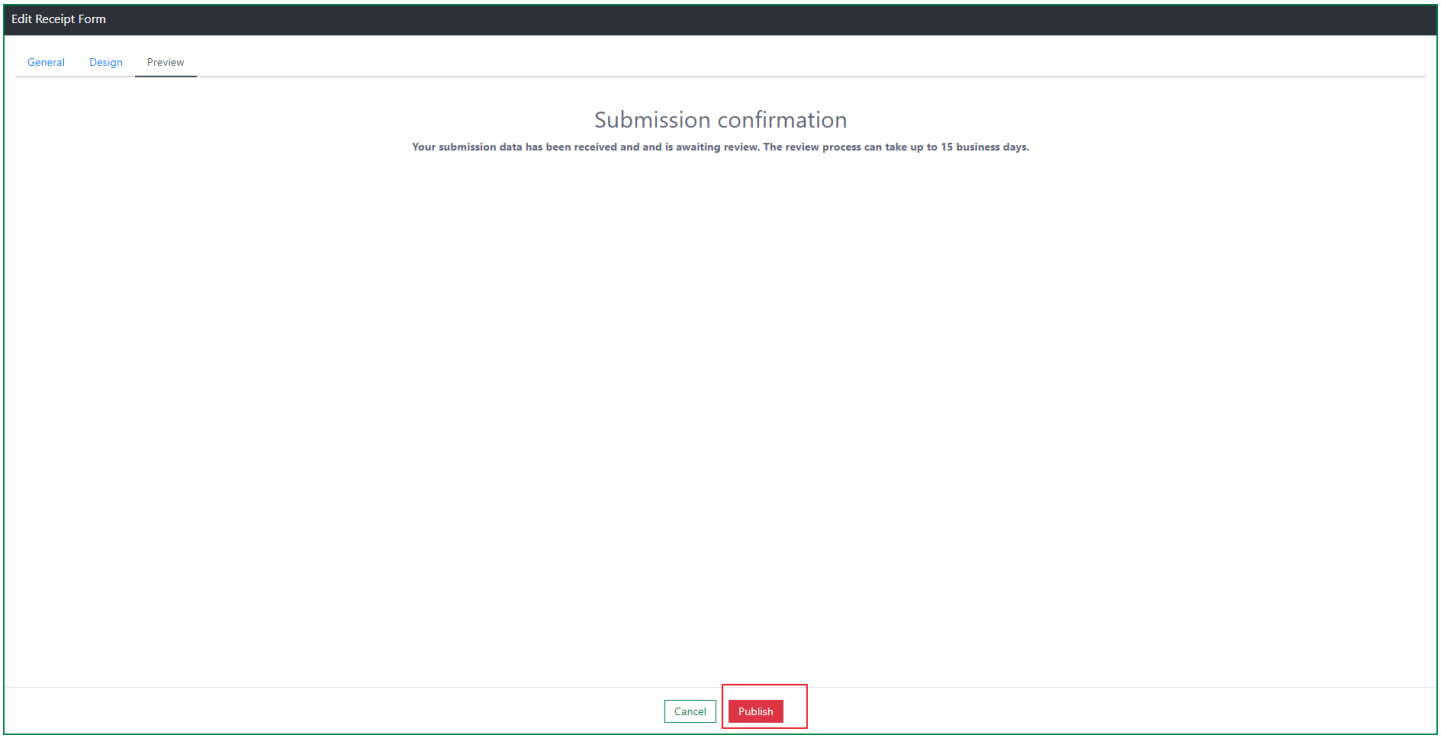
Preview



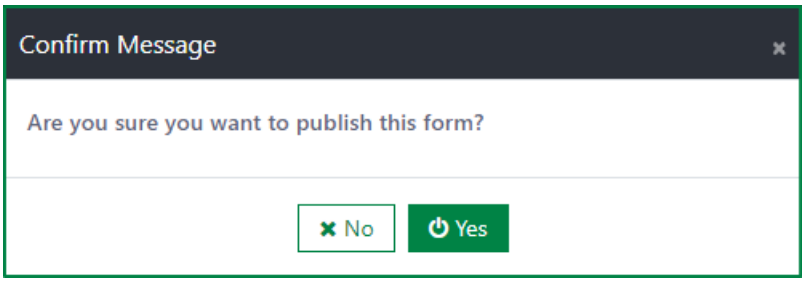
After completing designing, you must save the changes before going to the **Preview** tab to preview the rendered form and publish it.

In preview mode, you can view how the designed form displays after being published. The receipt form is a read-only form, so all fields in preview mode are also in read-only mode.

Besides previewing the designed receipt form, you can publish the form with the **Public** button.

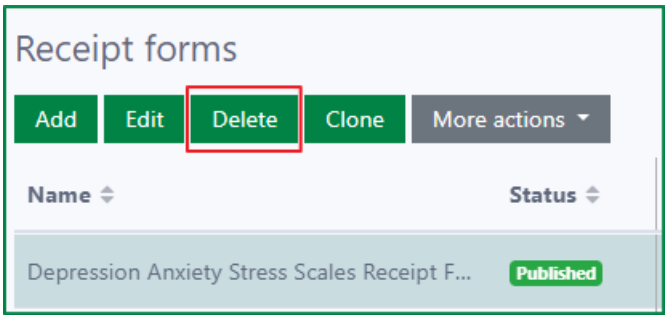


By clicking on the **Public** button, a confirmation dialog will be opened.



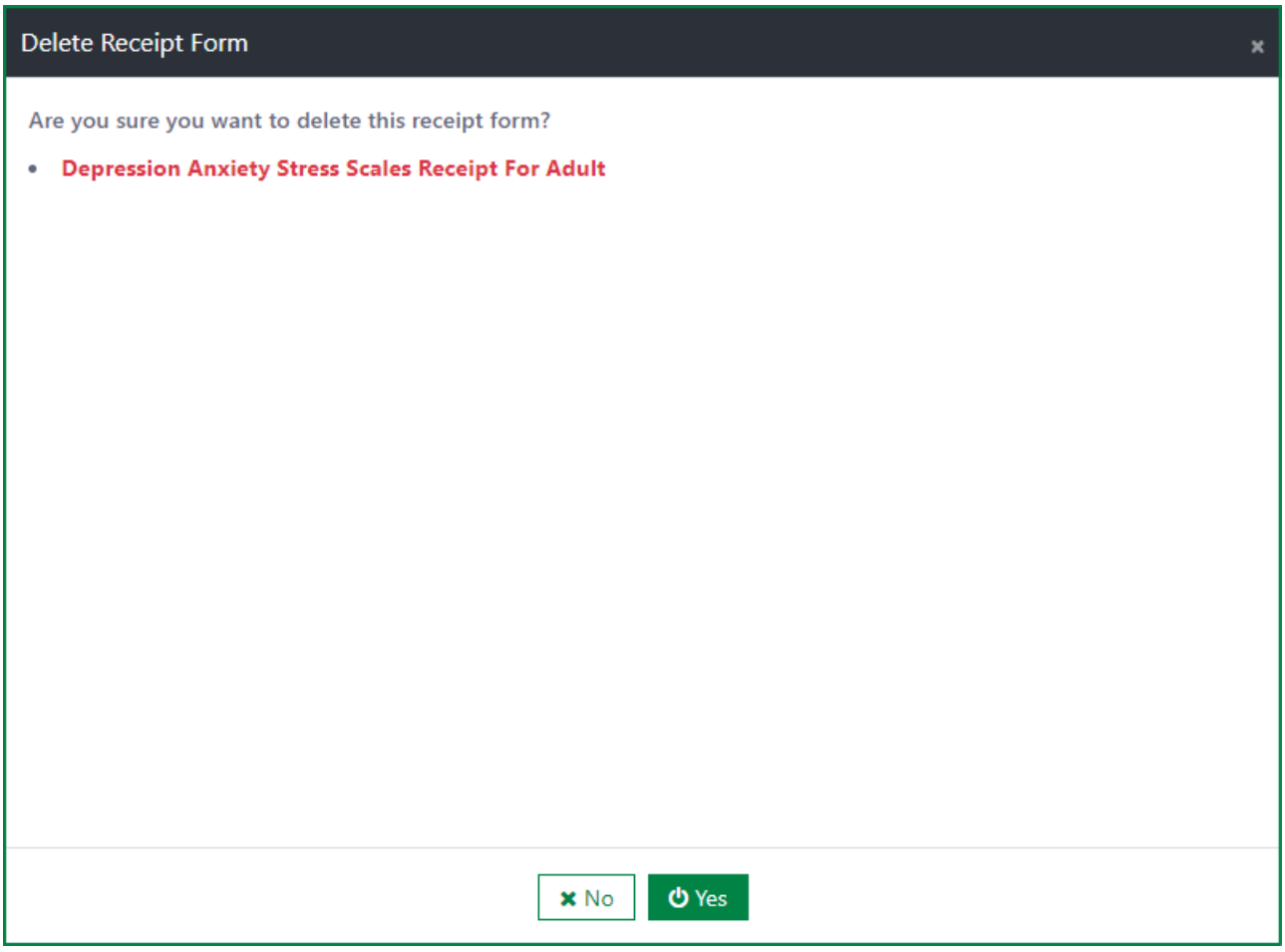
If you selects the **Yes** button, the receipt form will be published immediately. If you selects the **No** button, the publication will be cancelled.

Delete receipt form



The **Delete** button is only enabled after a receipt form is selected.

By clicking on the **Delete** button, the deletion confirmation dialog will be opened.



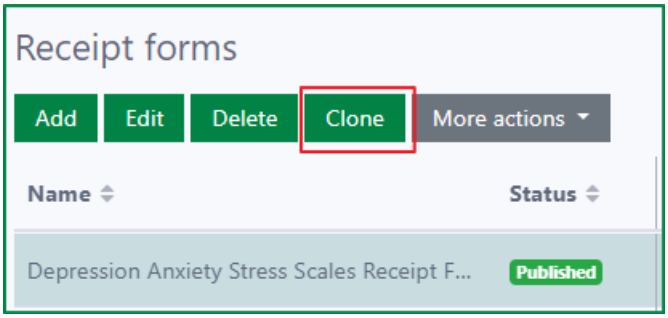
If you selects the **No** button, the deletion will be cancelled.

If you selects the **Yes** button, the receipt form will be deleted immediately if it is not used for any form. If the receipt form is already in use, the deletion is disallowed. After the deletion process has completed, the deleted receipt form will no longer be

accessible in public site.

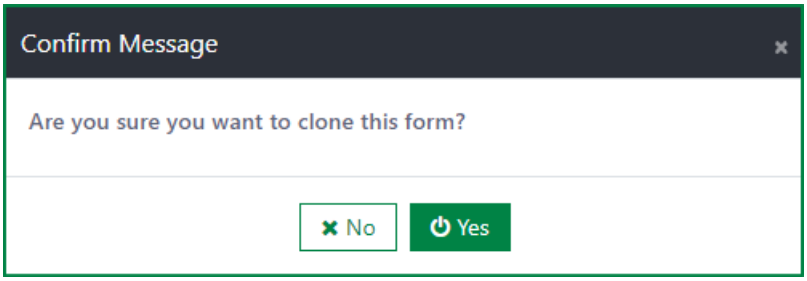
Other actions

Clone new receipt form



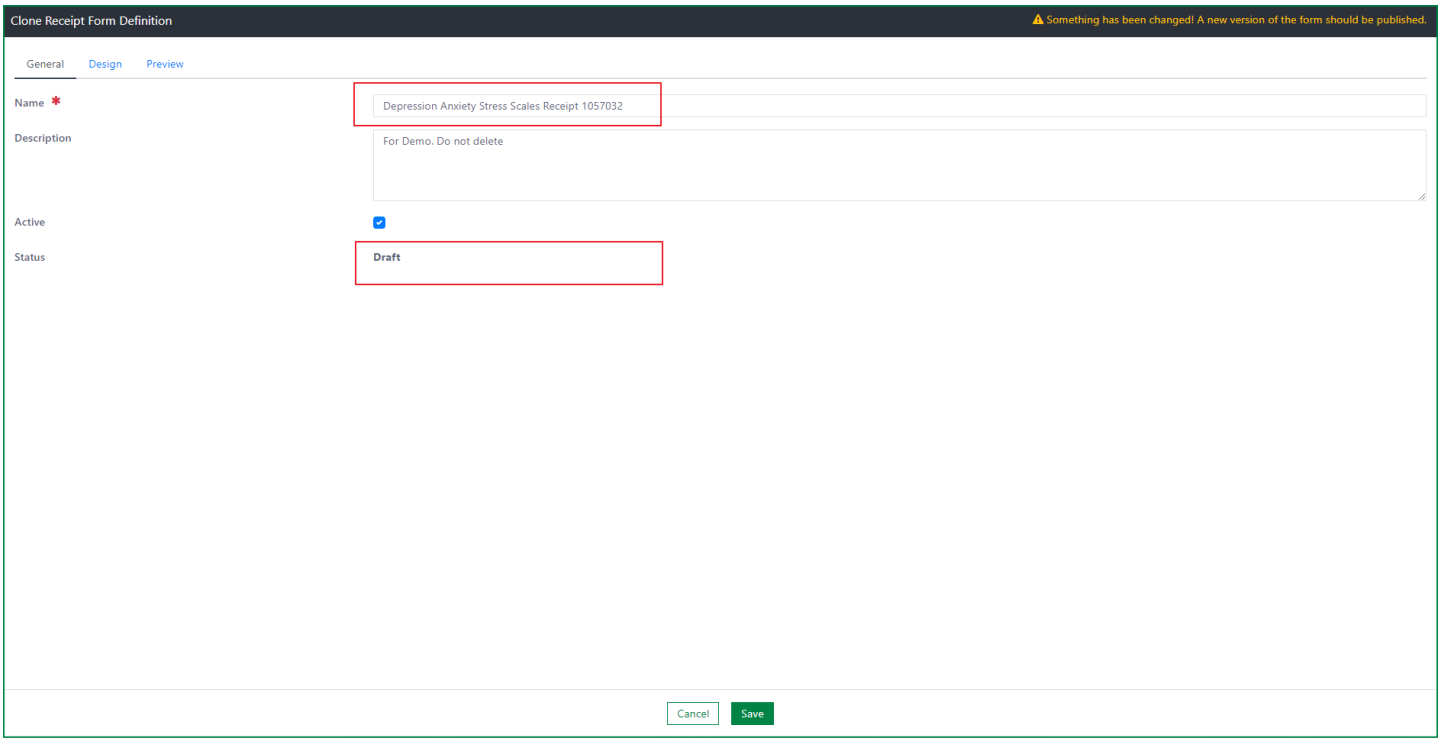
The **Clone** button is only enabled after an item on the list of receipt forms is selected.

By clicking on the **Clone** button, a confirmation dialog for cloning will be opened.



If you select the **No** button, the cloning process will be cancelled. If you select the **Yes** button, the cloning process will be started, and it will take a while to complete.

After the cloning process has completed, the new receipt form dialog will be opened. Name of the new form includes name of the original form and the suffix. Status of the new receipt form is **Draft** by default.

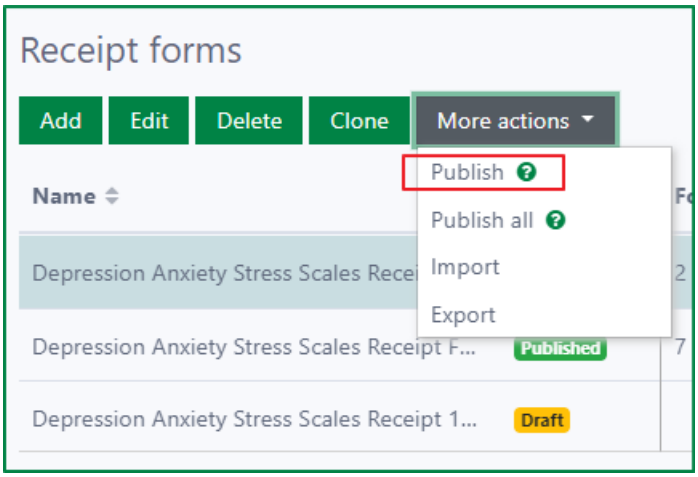


The new receipt form inherits the form design and all configurations of the original receipt form.

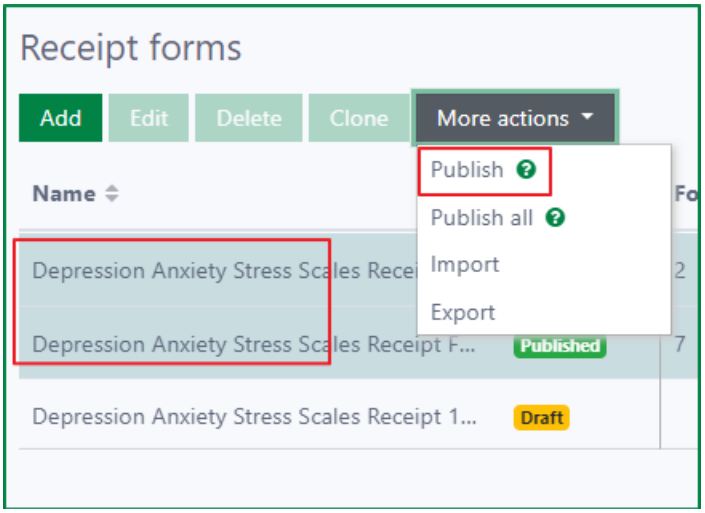
Publish receipt form

Besides publishing form in the **Edit receipt form** dialog, you can also publish a form, multiple forms or all forms but does not have to open each form to publish it.

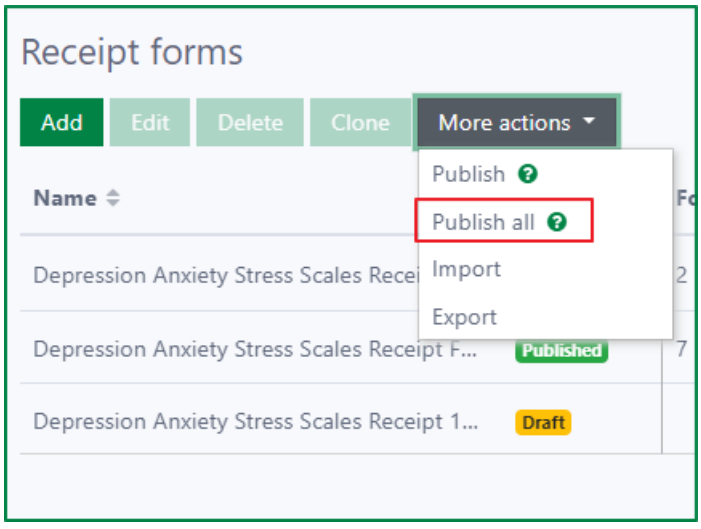
- **Publish a receipt form**
  - You can select a form then clicks on the **Publish** button, the selected form will be published immediately.



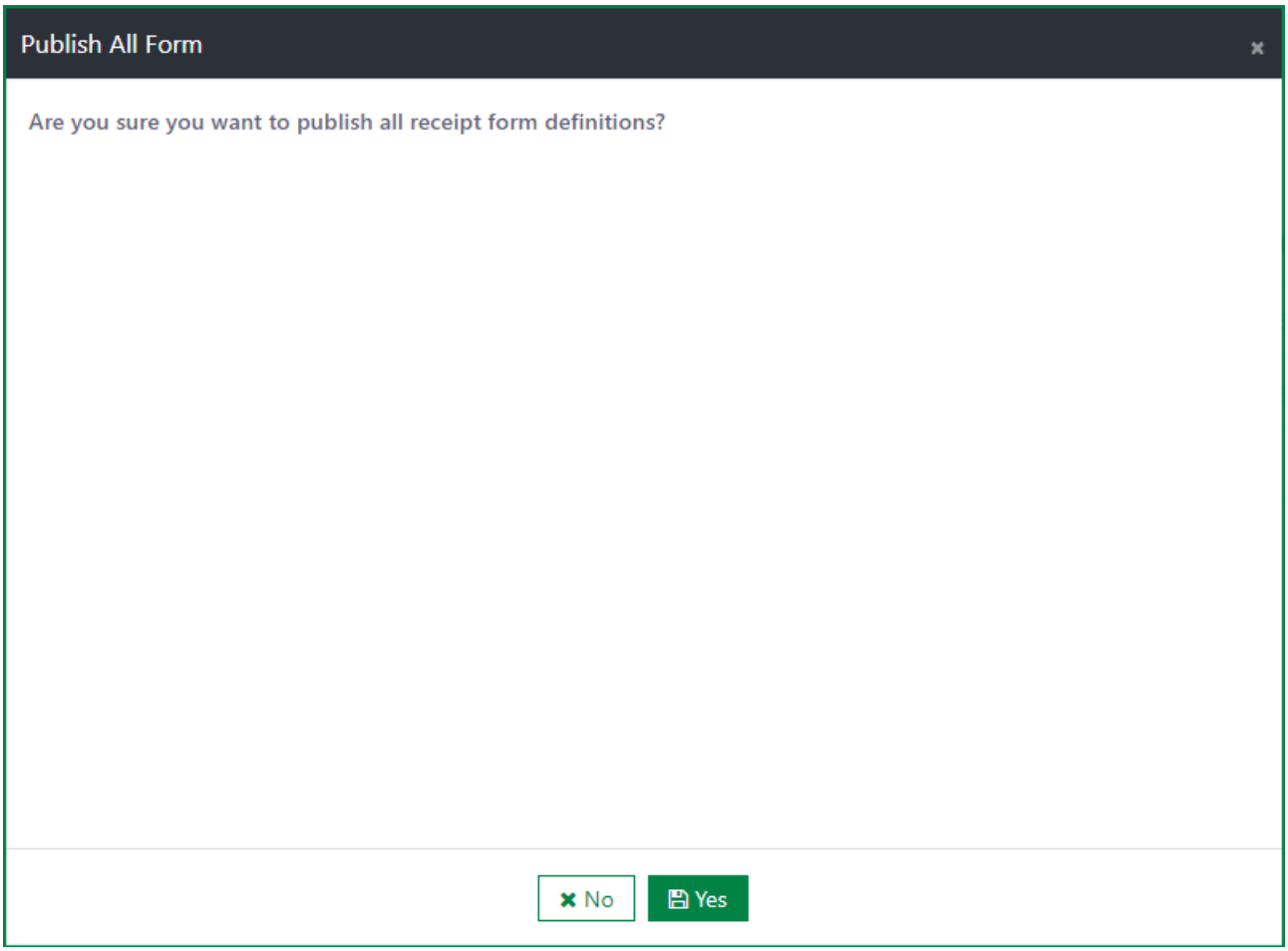
- **Publish multiple receipt forms**
  - You can select multiple forms then clicks on the **Publish** button, the selected forms will be published immediately.



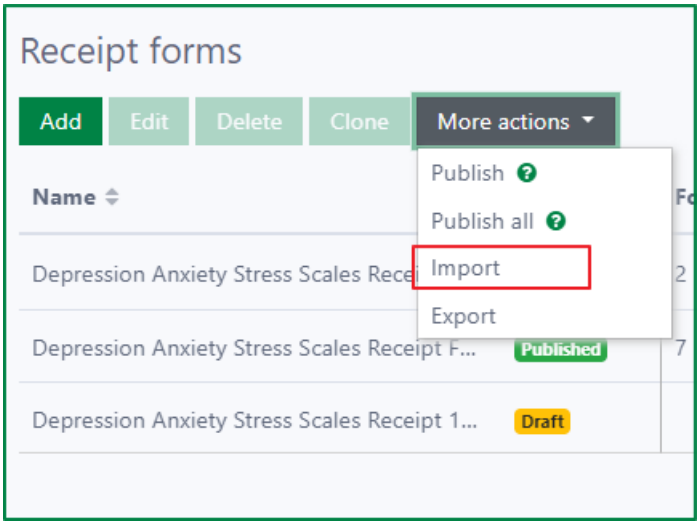
- **Publish all receipt forms**



  - By clicking on the **Publish all** button, a confirmation dialog of publication will be opened. If you select the **Yes** button, all receipt forms of which status is **Draft** will be published immediately. If you select the **No** button, the publication will be cancelled.

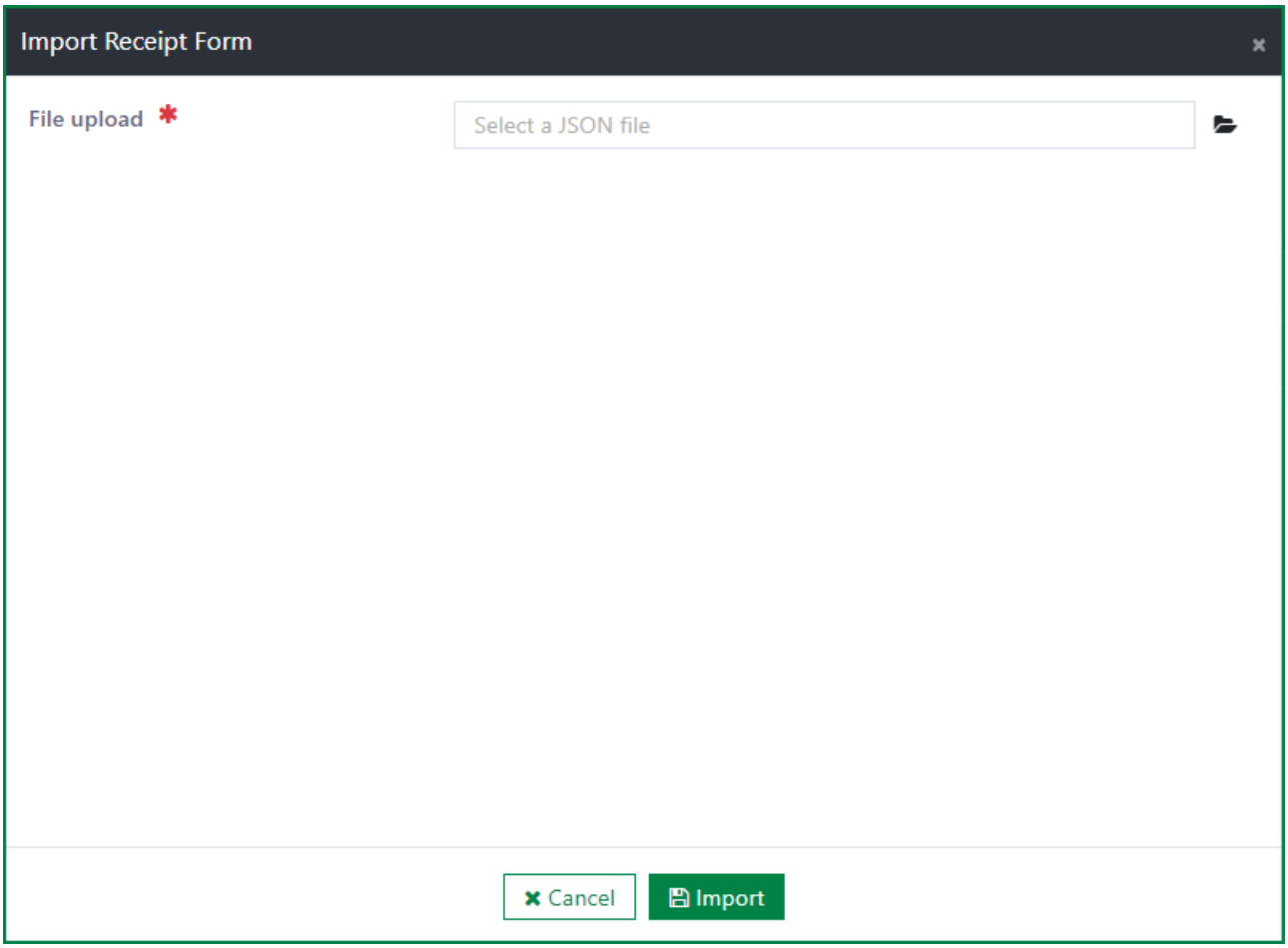


Import receipt forms



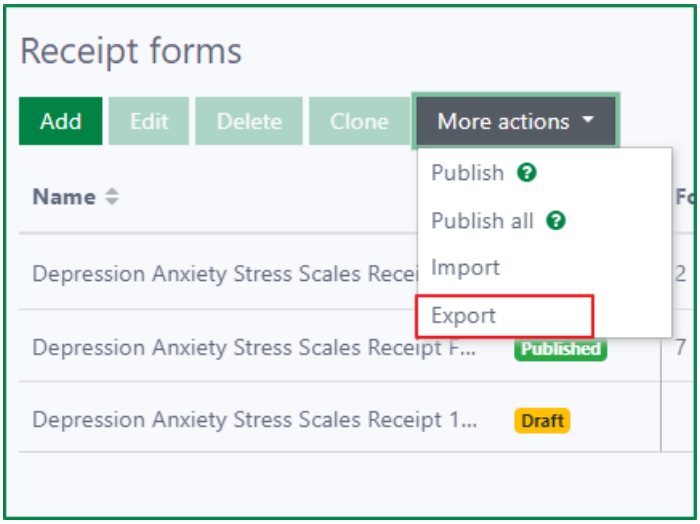
Besides creating receipt forms manually, you can import a list of receipt forms from a JSON file.

By clicking on the Import button, the Import form actions dialog will be opened. You can select a JSON file and upload it via the **File upload** field.



The imported receipt forms will be displayed on the Receipt forms list after the import process finishes.

Export receipt forms



By clicking on the **Export** button, the list of receipt forms will be exported immediately as JSON file. You can also export one or multiple forms instead of whole list by selecting the receipt forms, then clicking on the **Export** button.

Next section

Component: Form Management - Form Actions

General settings of form component

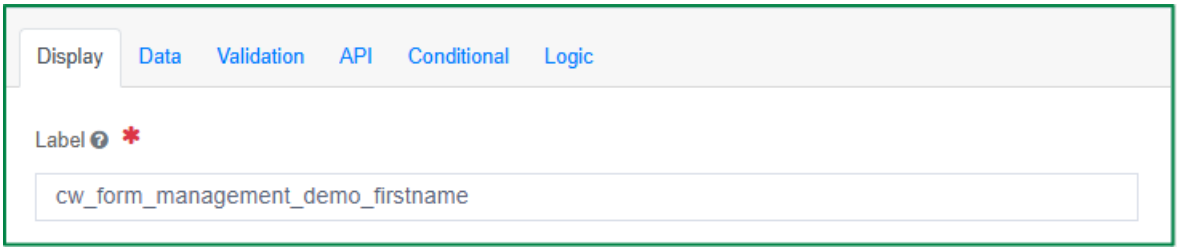
Display settings

The settings in the **Display** tab is for modifying the front-end UI of a component.

Following is the general settings in the **Display** tab of form components.

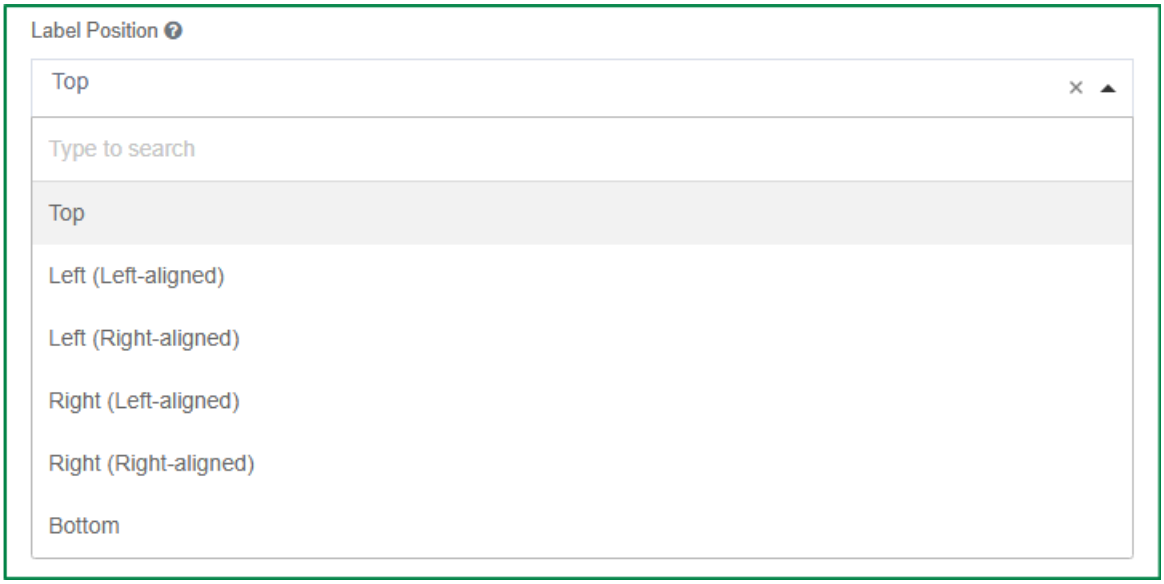
Label

This option is used to define the name or title of the field. You can also input the text resource key for the label if you want to field label to be displayed in multiple languages.



Label Position

This option is used to define the position for the label of the field. There are 6 options in the list of positions, you can choose an item from the provided list.



- **Top**: the label will be displayed on the top of the component and left-aligned



- **Left (Left-aligned)**: the label will be displayed at the left of the component and aligned to the left of the component width span.



- **Left (Right-aligned)**: the label will be displayed at the left of the component and aligned to the right of the component

width span.



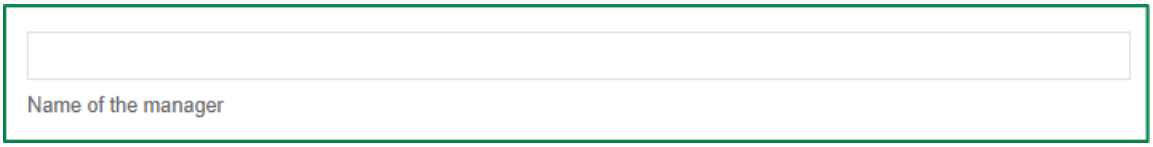
- **Right (Left-aligned)**: the label will be displayed at the right of the component and aligned to the left of the component width span.



- **Right (Right-aligned)**: the label will be displayed at the right of the component and aligned to the right of the component width span.

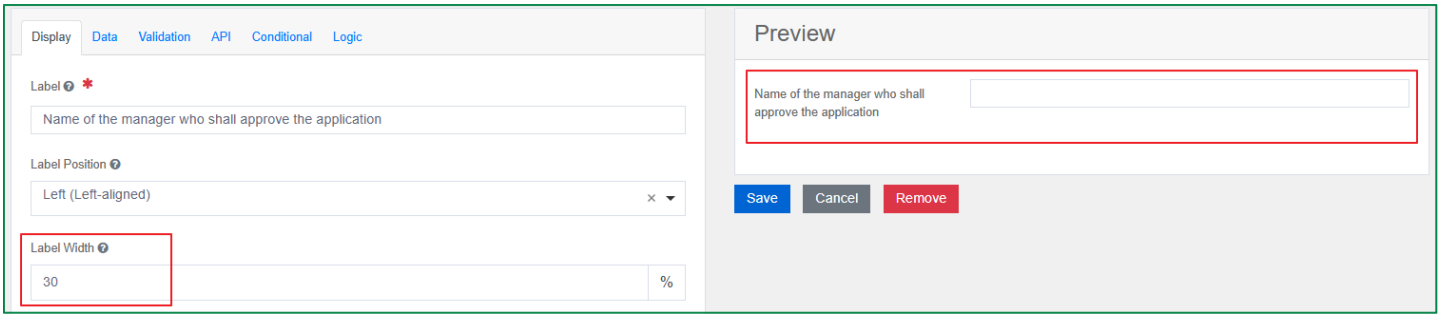


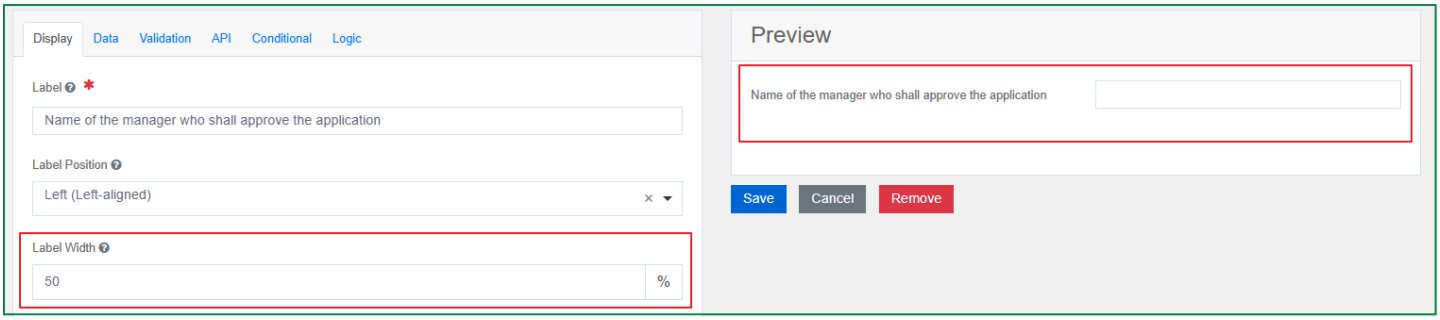
- **Bottom**: the label will be displayed below the component and left-aligned.



Label Width

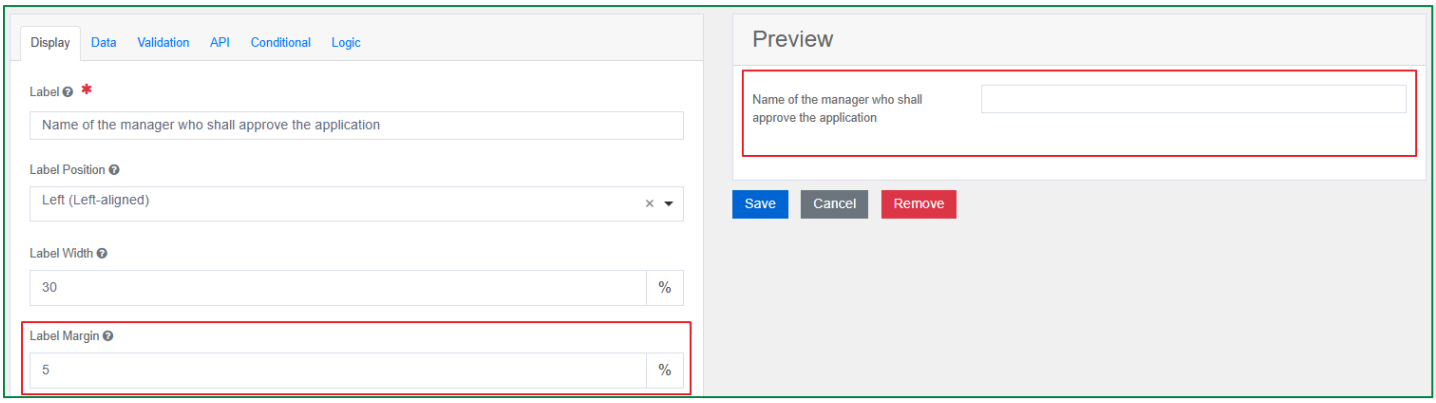
This option is used to define the width of label on line in percentages. By default, the label width is 30%. This option is only enabled if the Label position is Left/Right.

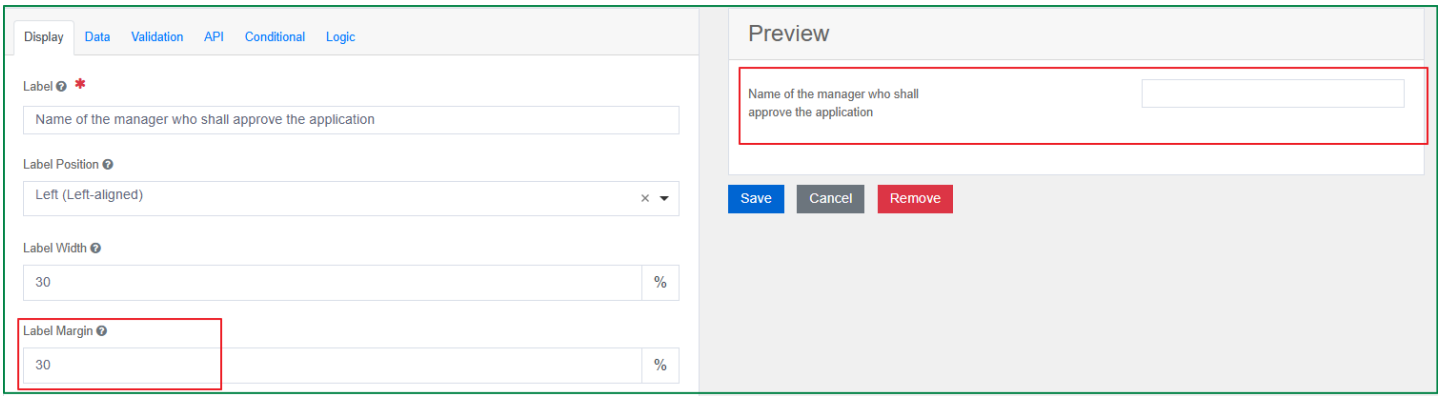




Label Margin

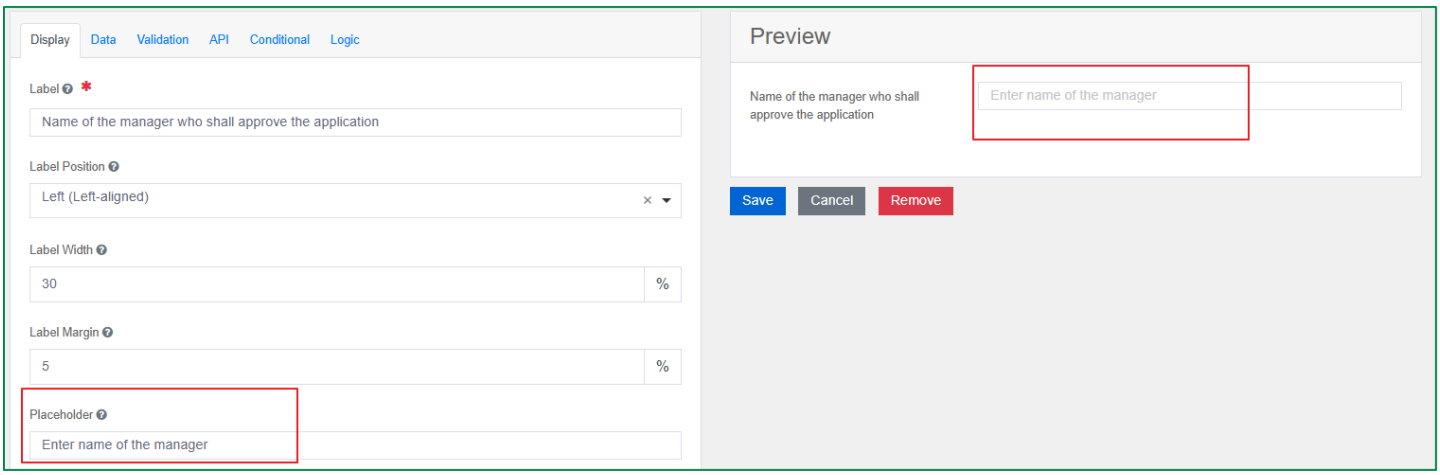
This option is used to define the width of label margin on line in percentages. By default, the label margin width is 3%. This option is only enabled if the Label position is Left/Right.





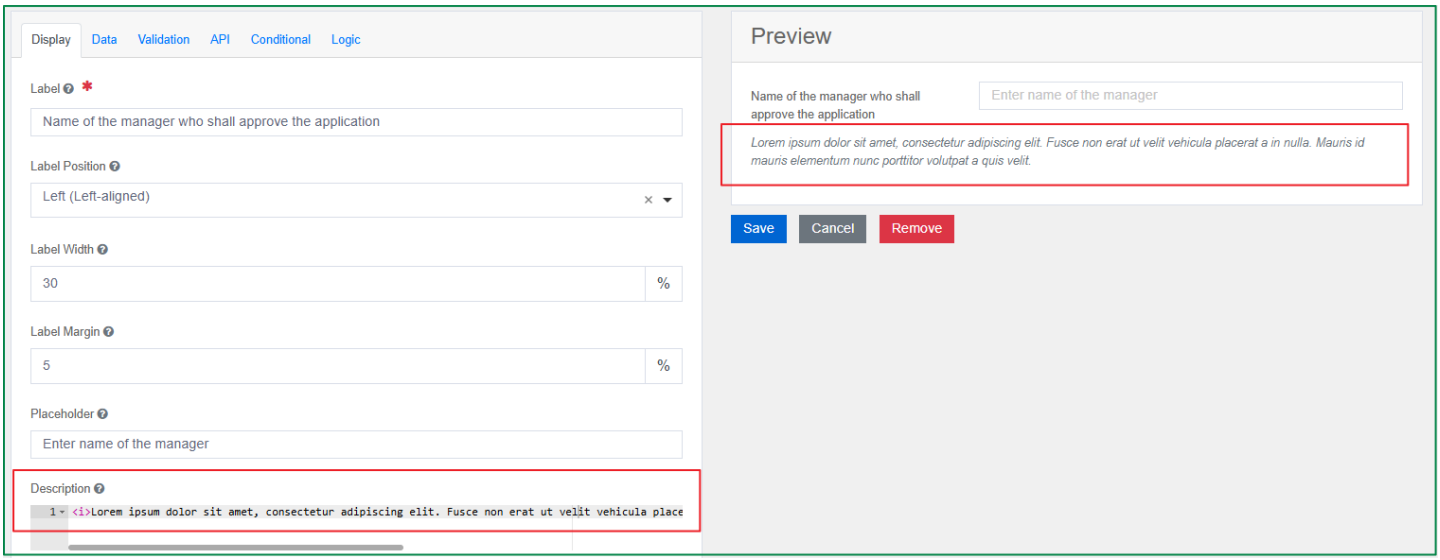
Placeholder

This option is used to define the placeholder text that will appear when this field is empty.



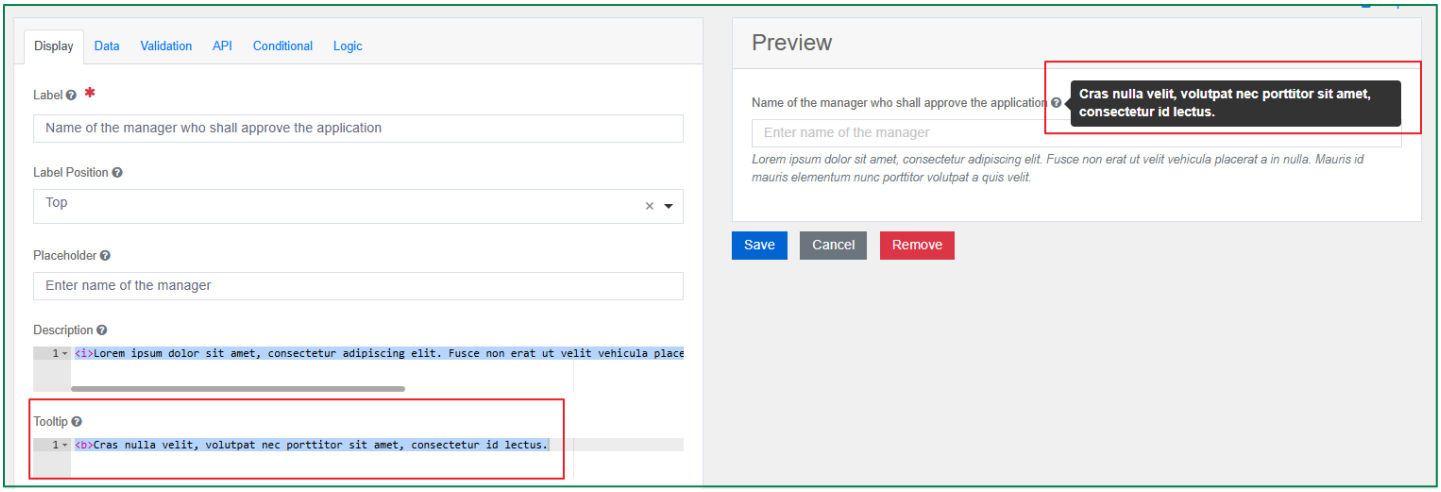
Description

This option is used to define the description of the field which is displayed below the field. You can style the description content by using HTML.



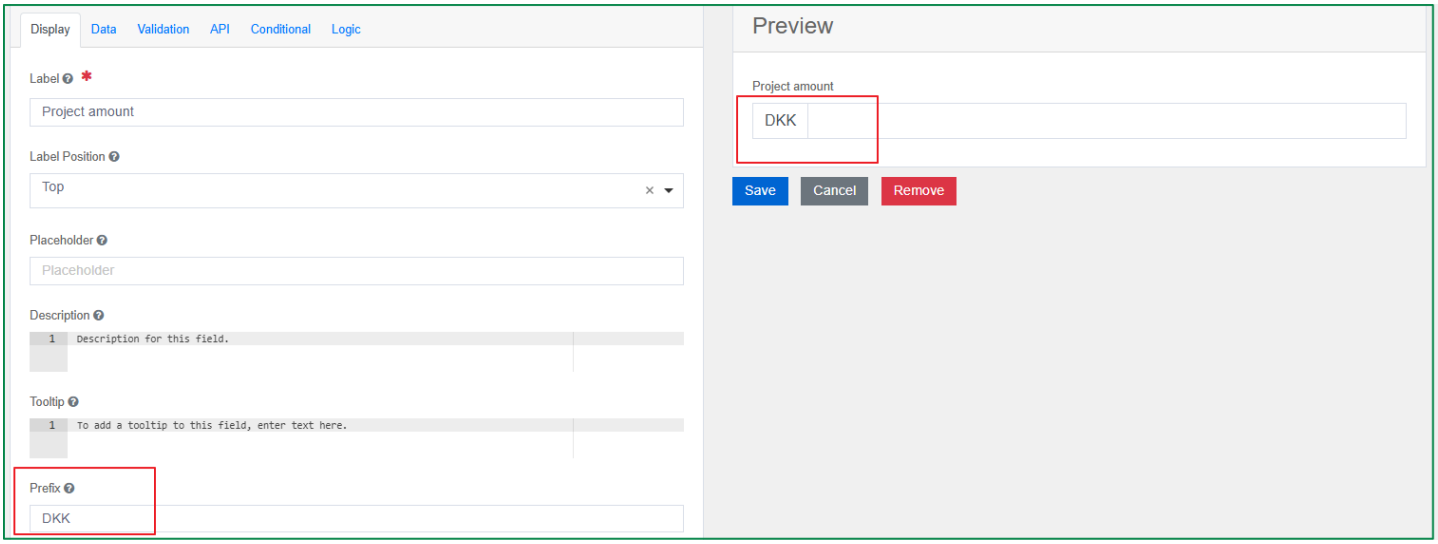
Tooltip

This option is used to define the tooltip of the field. If the tooltip is defined, a question mark icon will be displayed at the right of the field label in front-end view. When the end-user hovers over the icon, the tooltip content will be displayed. You can style the tooltip content by using HTML.



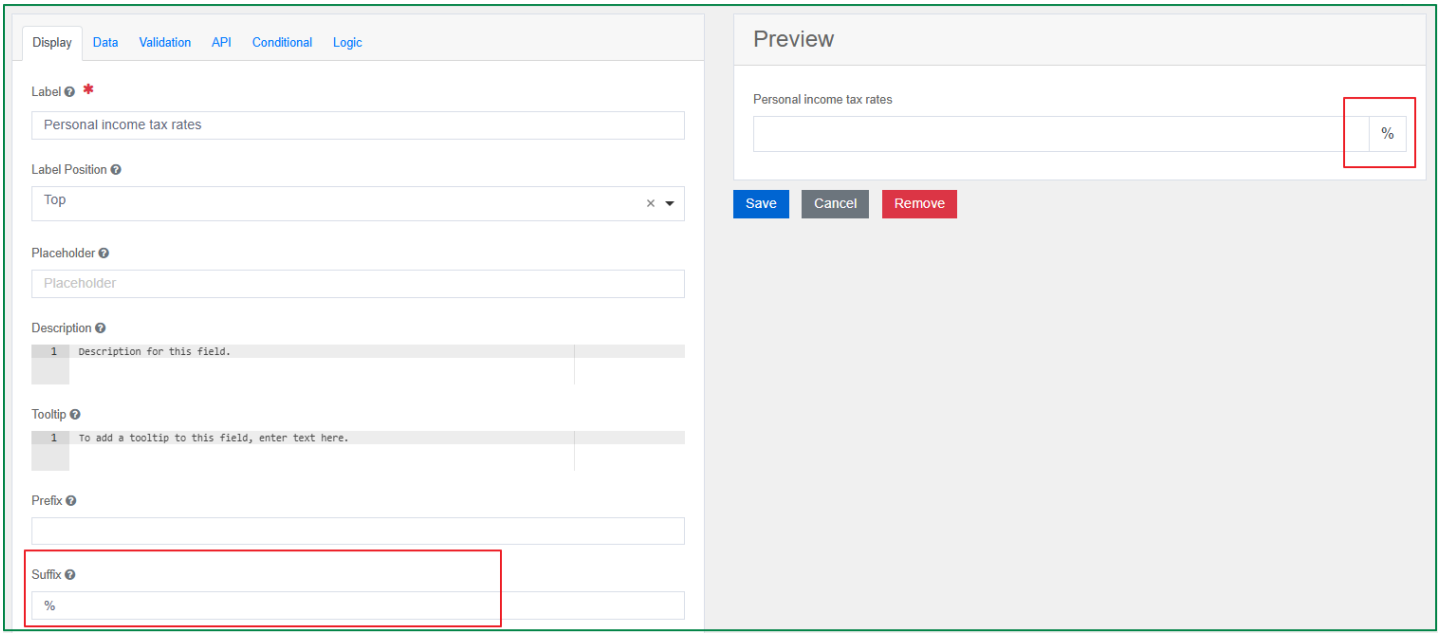
Prefix

This option is used to define the text which is shown before the input field. For example, using 'DKK' for currency.



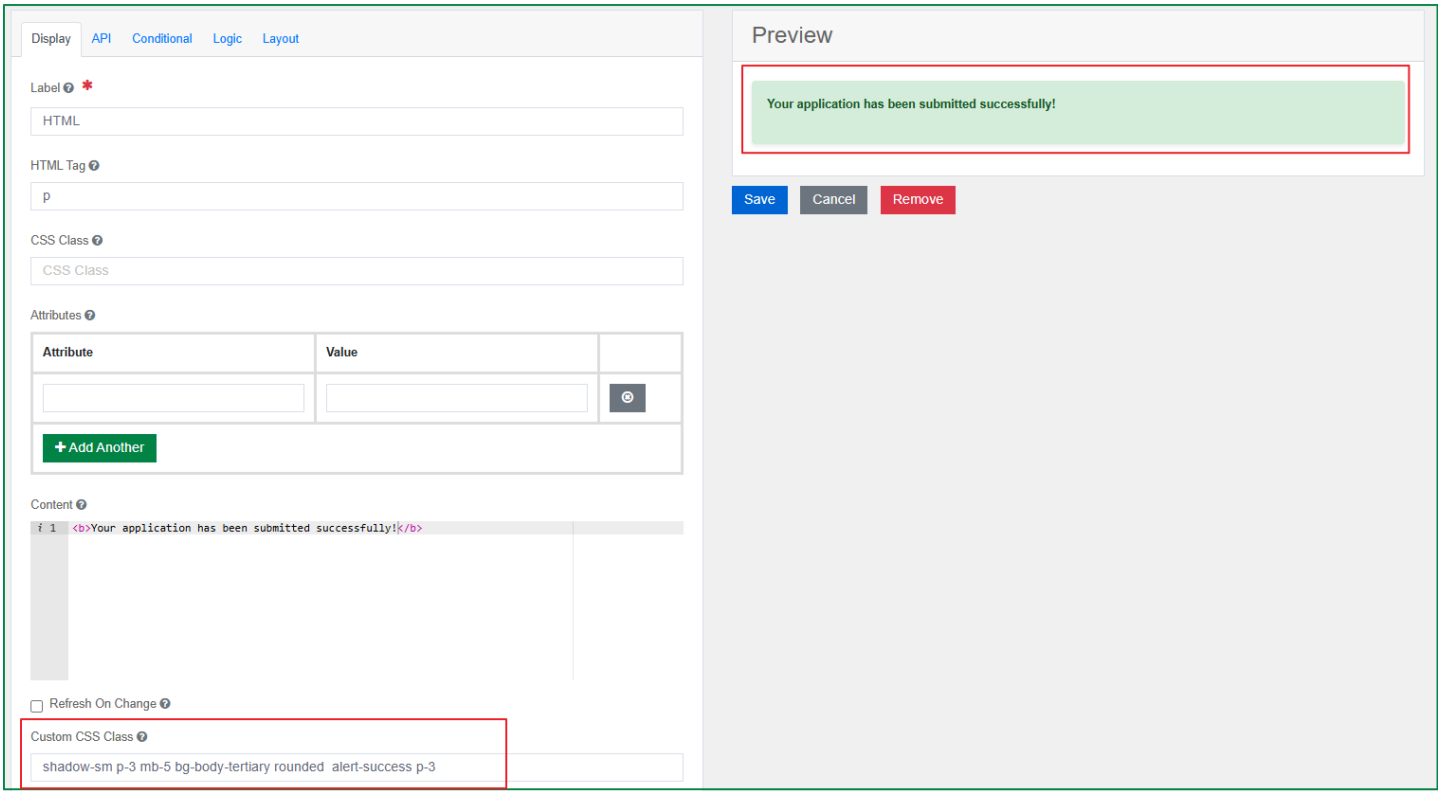
Suffix

This option is used to define the text which is shown after the input field. For example, using '%' for percentage.



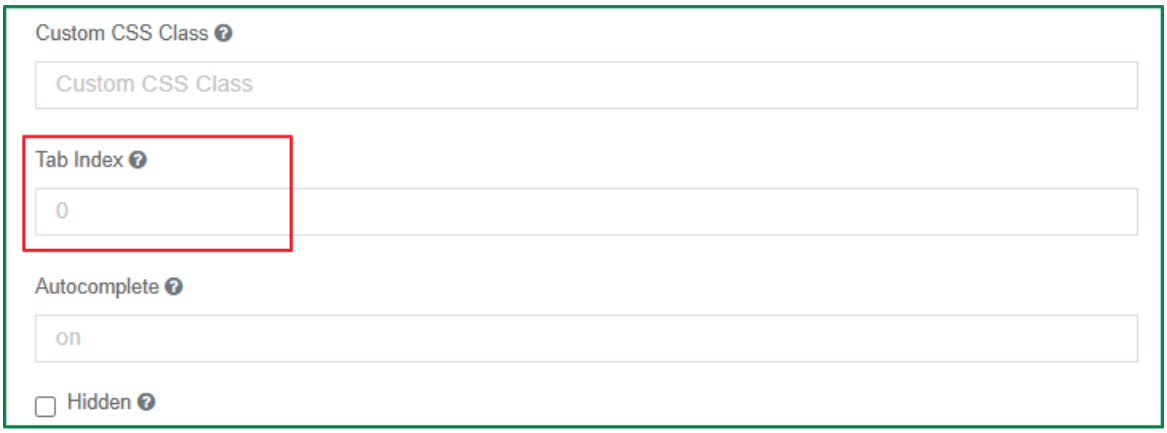
Custom CSS Class

You can add custom CSS class to apply the styling and formatting to the component. You may add multiple class names separated by a space.



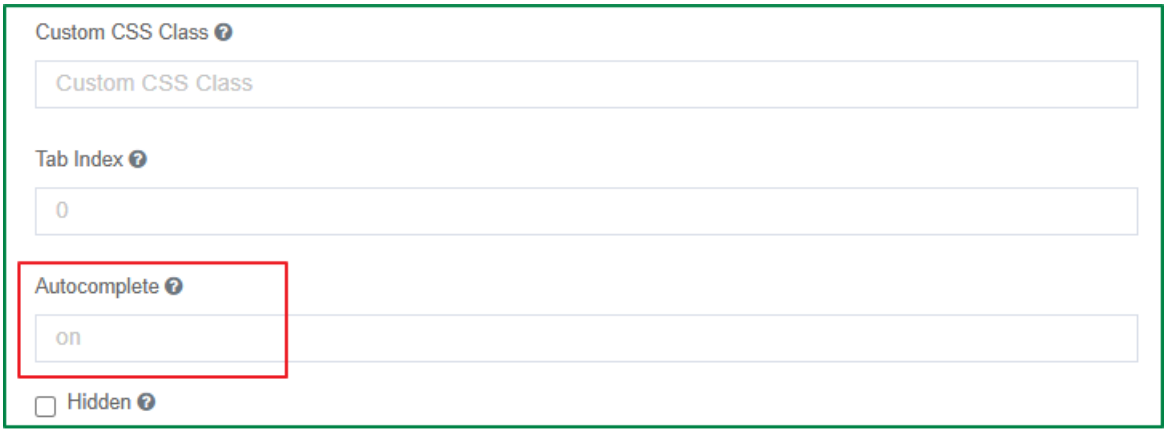
Tab Index

This option is used to define the order of the focused fields when the end-user presses the Tab keyboard. If the tab index is filled, the default tab order of the form will be overridden.



Autocomplete

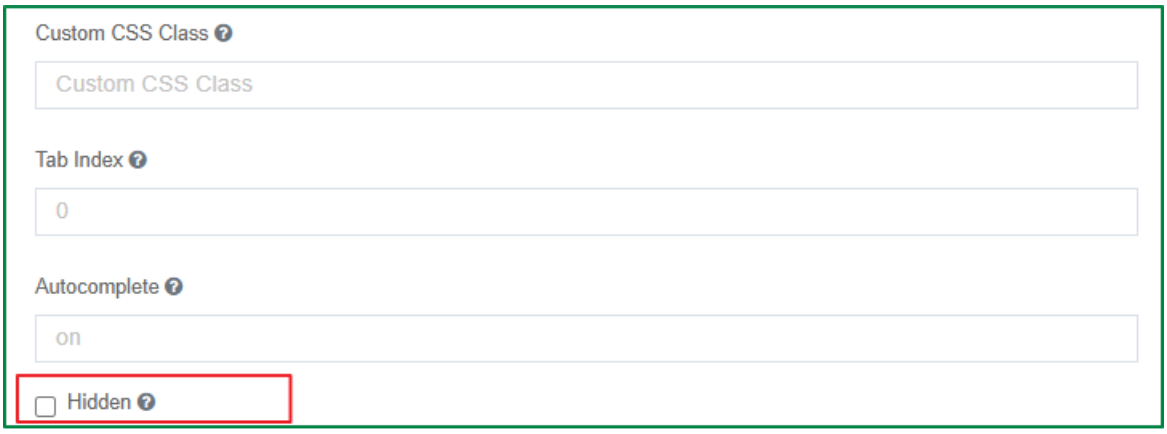
This option is enabled by default. This option is used to indicate whether the input fields will automatically populate data from values previously entered by the user dictated by the cache of the user's browser. Input the *off* value inside the setting field to disable.



Hidden

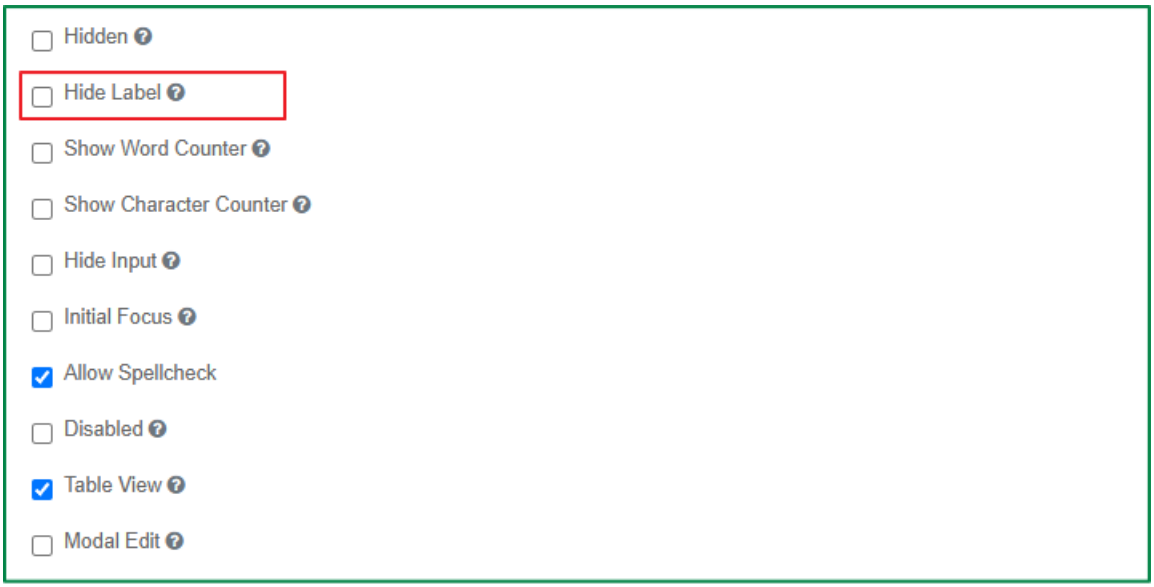
This option is used to hide the field from the rendered form at the view of end-users. However, a hidden field is still a part of the form JSON. You can use it in combination with the dependency rule of visibility and accessibility.

Given the example that *the **Auxiliary documents** field is hidden by default, and is only displayed, enabled, and required if the value of **Project duration** field is longer than 6 months. In this case, the **Hidden** checkbox in the configuration of the **Auxiliary documents** field should be ticked.*



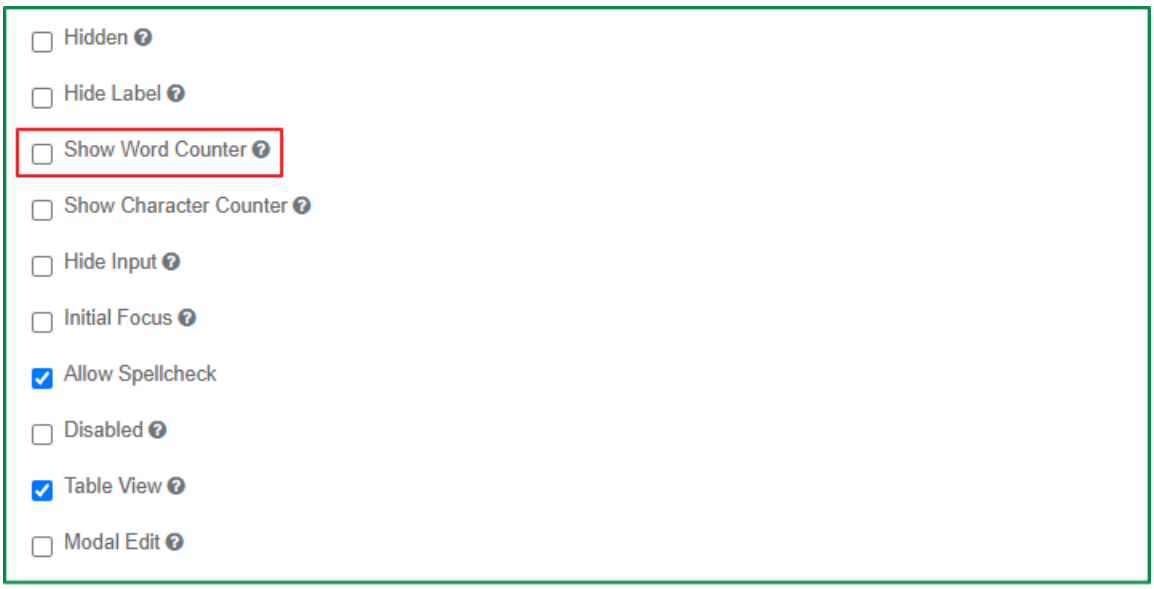
Hide Label

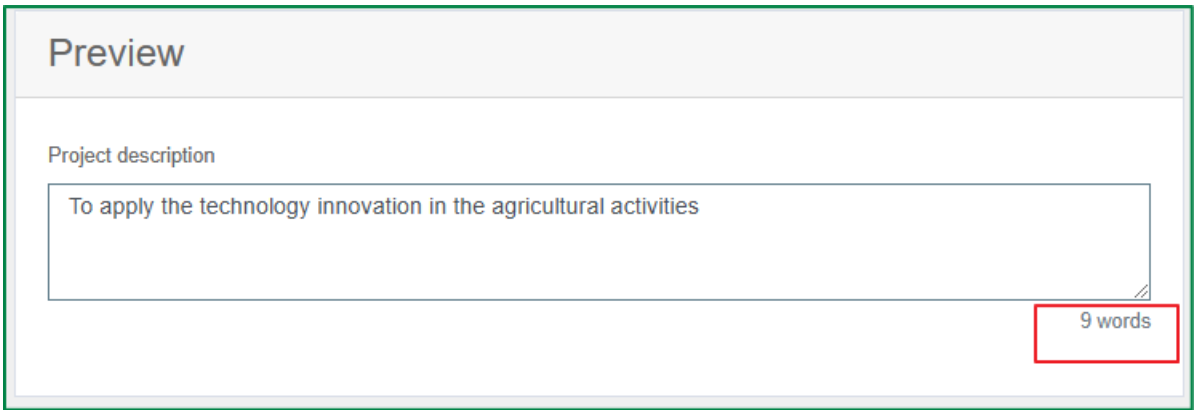
Tick this checkbox if you want to hide the label of this component. After the checkbox is ticked, the field label is still displayed in the design mode, and will be hidden on the rendered form at the view of end-users.



Show Word counter

Tick this checkbox if you want to show the count of the filled words while the end-user is typing.

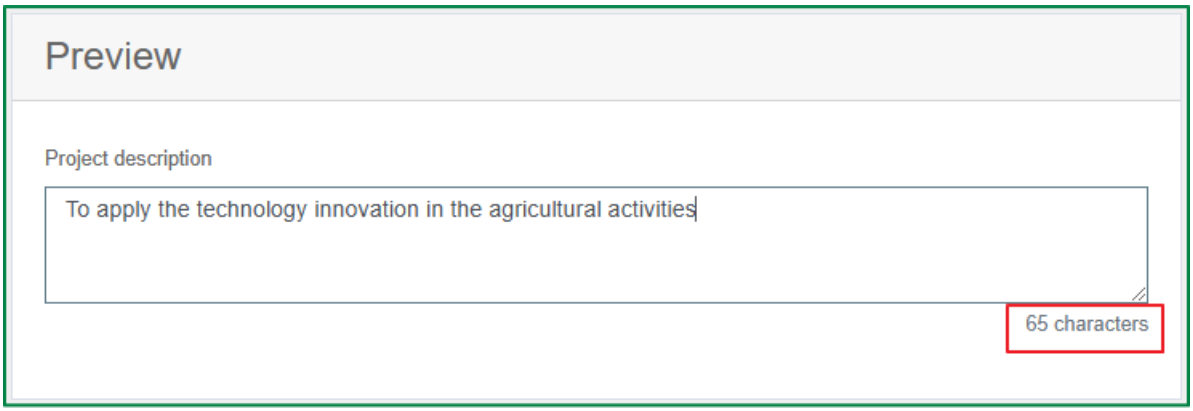




Show characters counter

Tick this checkbox if you want to show the count of the filled characters while the end-user is typing.



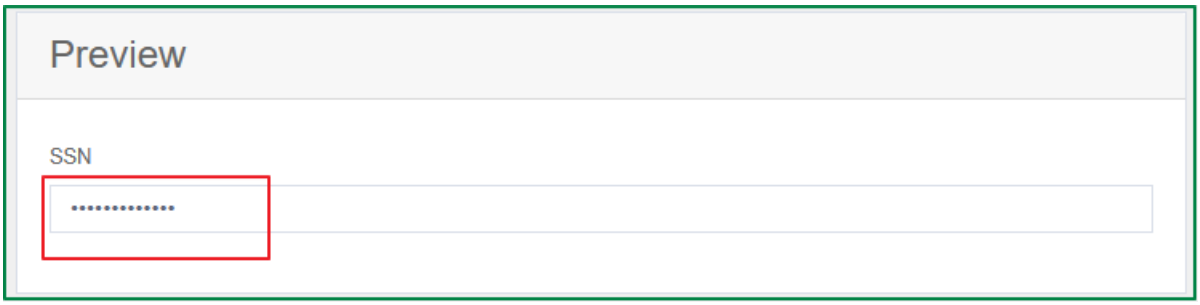


Hide Input

Tick this checkbox if you want the input data on the field is displayed with the circle characters like how it is displayed on the password fields. This settings can be used to handle the case of sensitive data, such as Tax ID or SSN.

This option does not encrypt the data on the server. Do not use this setting for Passwords.





Initial Focus

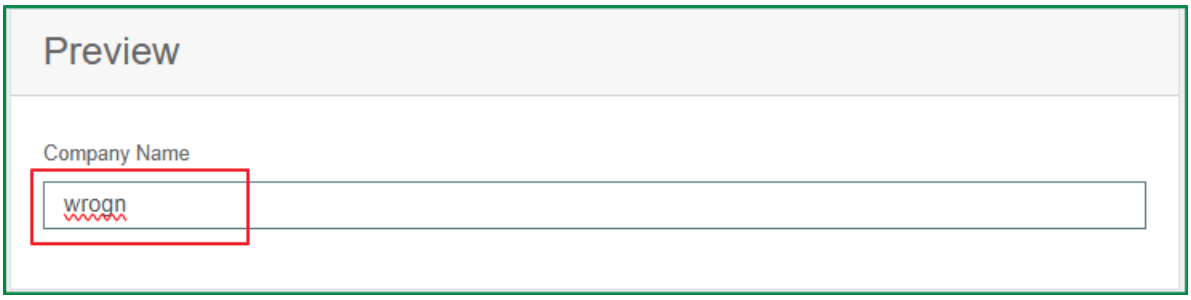
Tick this checkbox if you want the field is focused initially on the rendered form when the form is loaded. Only one component should have the Initial Focus setting enabled. If more than one component on the form has the Initial Focus setting enabled, only the last component in the JSON structure with the setting enabled will be focused on the form.



Allow Spellcheck

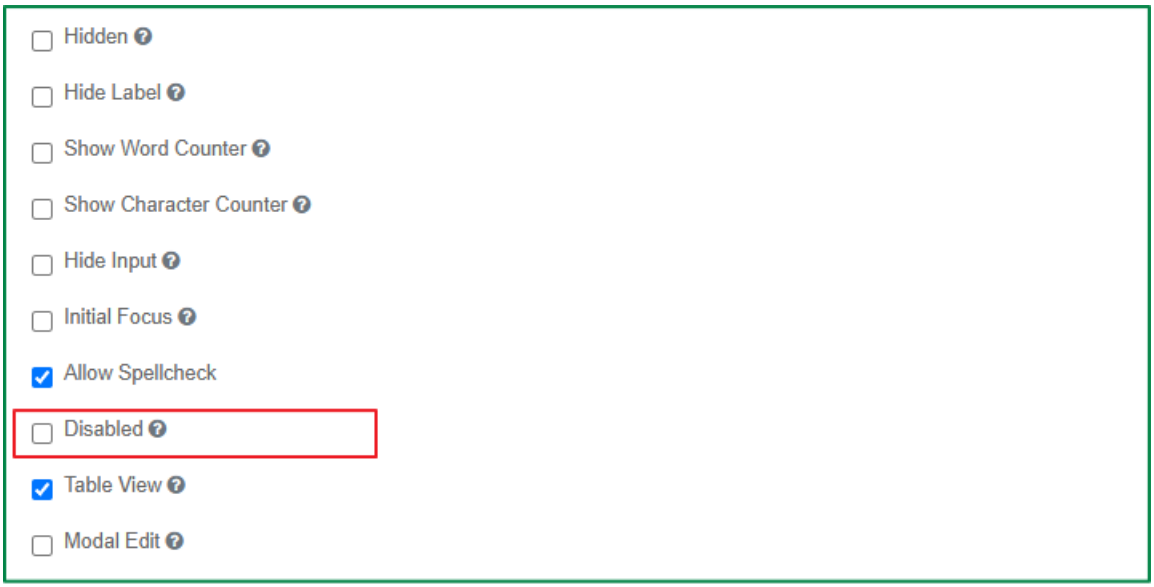
Tick this checkbox if you want to enable the real-time spellchecking for input fields on the rendered form. This settings is turned on by default.





Disabled

Tick this checkbox if you want the field is displayed in disabled state on the rendered form. The end-user cannot input data to the disabled field.



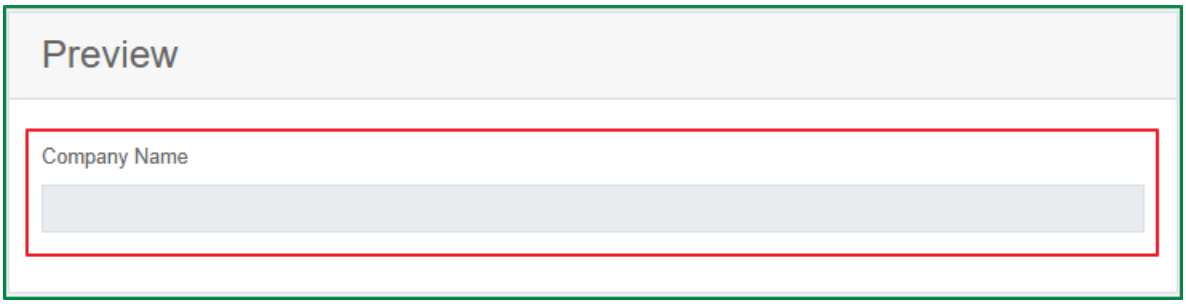
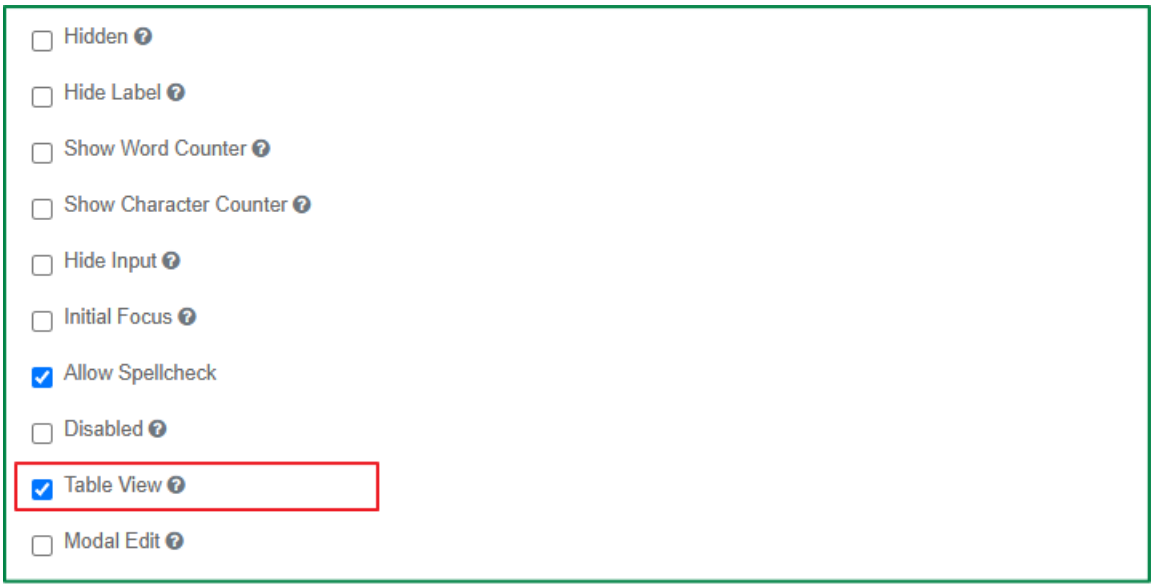


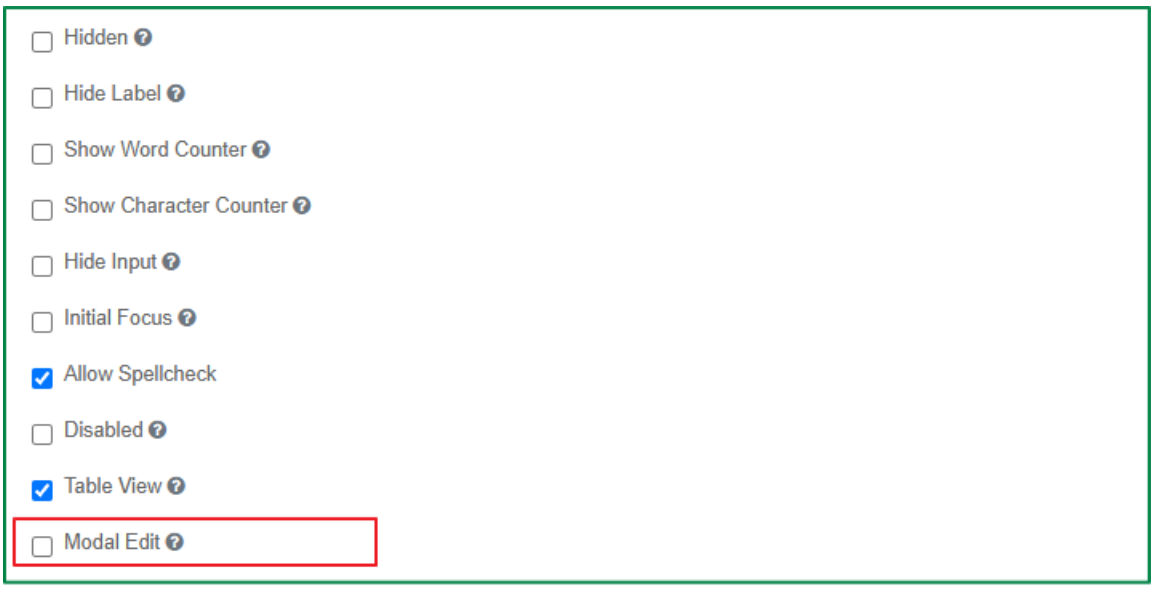
Table View

If checked, the input data of the field will be shown in the table view of the submissions list.



Modal Edit

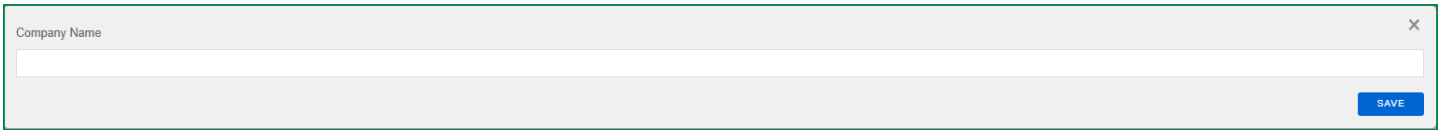
Tick this checkbox if you want the end-user to input data for the field in a modal window.



If this option is enabled, on the rendered form, the input field will be replaced with a button.



Then, when the end-user clicks the button, a modal window will be opened to input, or edit the field data.



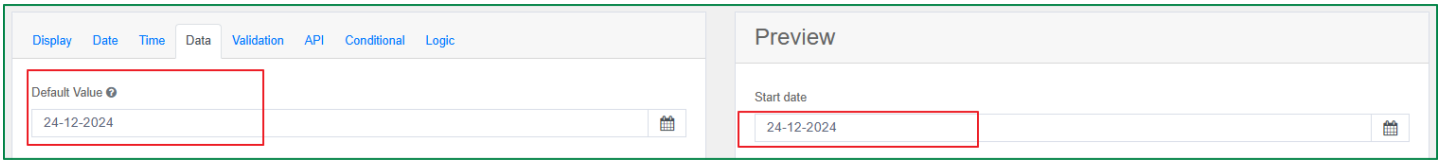
Data settings

The settings in the **Data** tab are utilized for configuring end-user input data, determining how it interacts with other field data within the form, and how it is stored on the server, such as setting a default value or configuring data calculations.

Following is the general settings in the **Data** tab of form components.

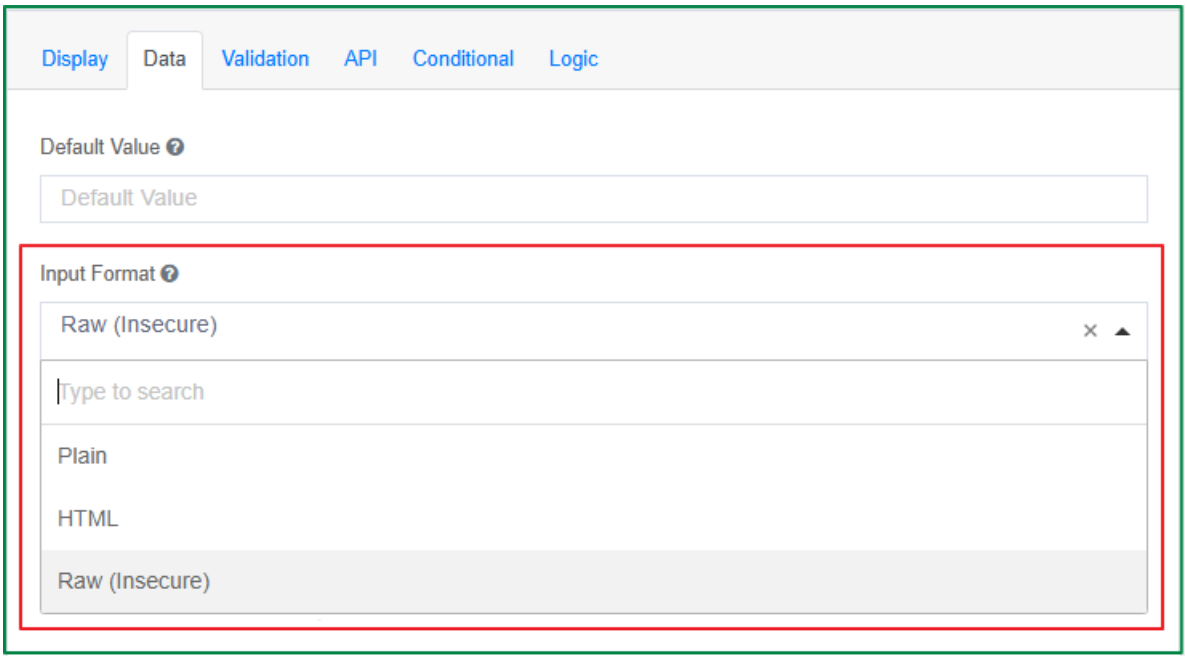
Default Value

This option is used to define the default value which will be displayed in the field when the form is rendered. If the end-user does not make any change in the field, the default value data will be saved with the submission.



Input Format

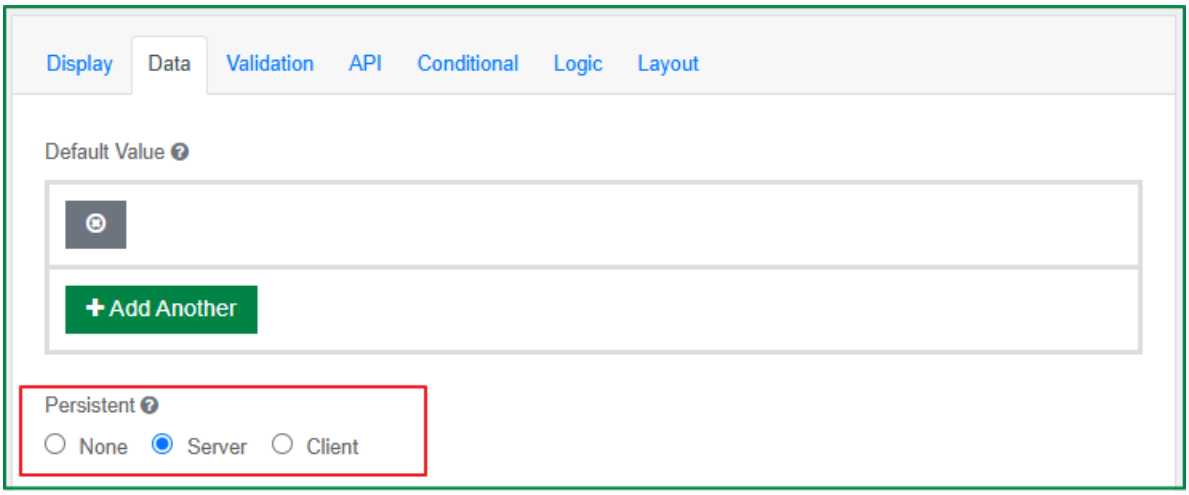
This option is used to force the output of the field to be sanitized in a specific format. There are 3 format options: Plain, HTML, Raw



Persistent

This option is used to determine how the field data will be stored when the form is submitted. There are 3 persistent options: **None, Server**, **Client**. The default selected option is **Server**.

- **None**: The front-End will submit the data to the backend, but the backend will not store the value. This option should be selected in case the client-side logic is needed on the form without the requirement of sending or saving the data to the server.
- **Server**: The Front-End will submit the data to the backend, and the backend will store the value.
- **Client**: The Front-End will not submit the data to the backend, and the backend will not store the value. This option should be selected in case the server needs to intercept data for workflows such as logic or calculations, but there is no need or desire for the data to be stored.



Protected

This option is used to prevent the field data from being queried from an API call. Tick this checkbox if an extra layer of protection is needed for sensitive data.



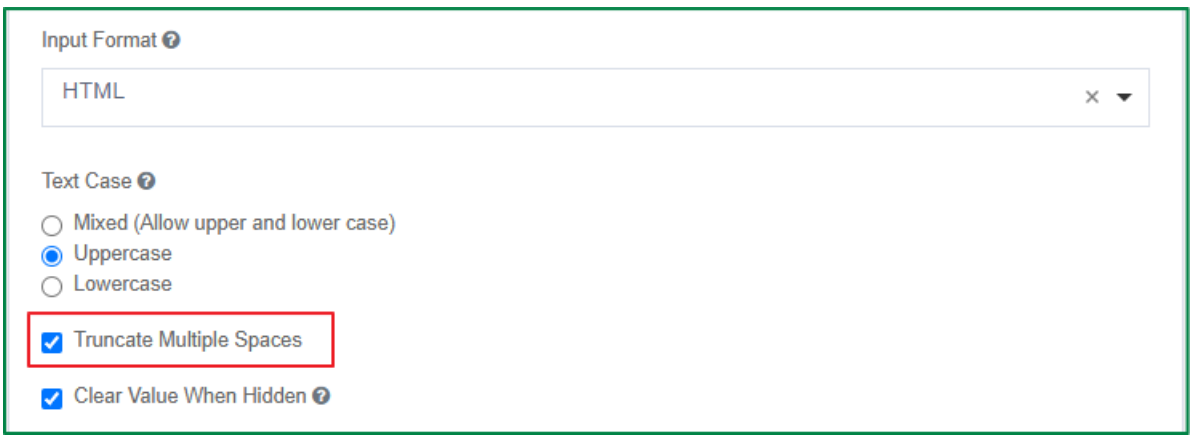
Database Index

Tick this checkbox if you want to set this field as an index within database. It is used to increase the performance for submission queries.



Truncate Multiple Spaces

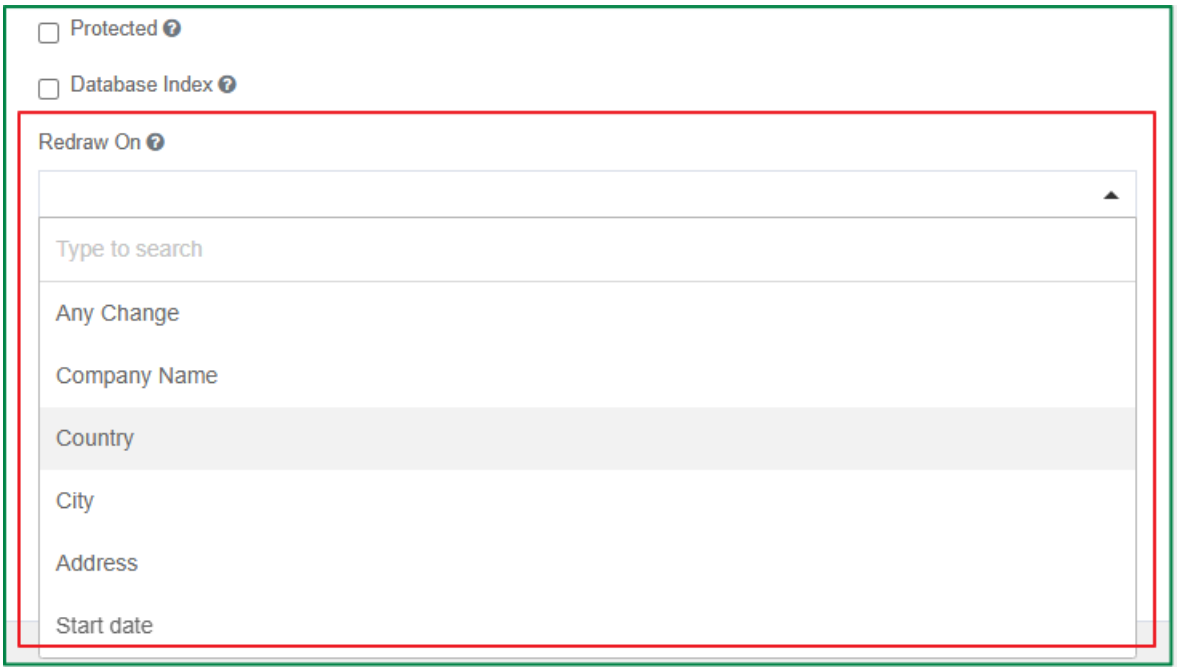
Tick this checkbox if you want to remove the unnecessary spaces between words or characters within the field.



Redraw On

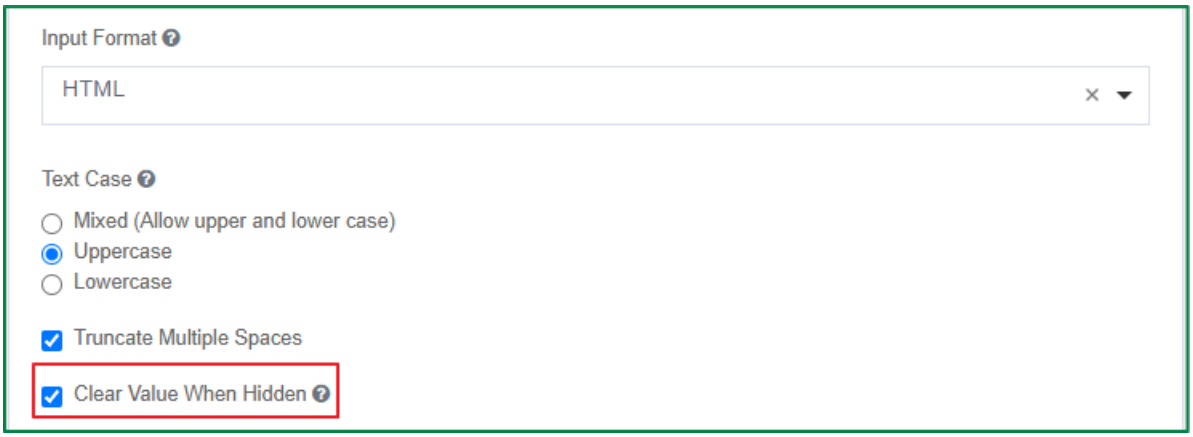
This option is used to refresh the component when a specific changes occurs on the rendered form. This option includes a

dropdown list of fields added in the form, you can select a field from the dropdown list; then, when it is a change on the selected field, the component will be refreshed. If you select the option Any change in the list, the component will be refreshed when it is any change occurs on the rendered form.



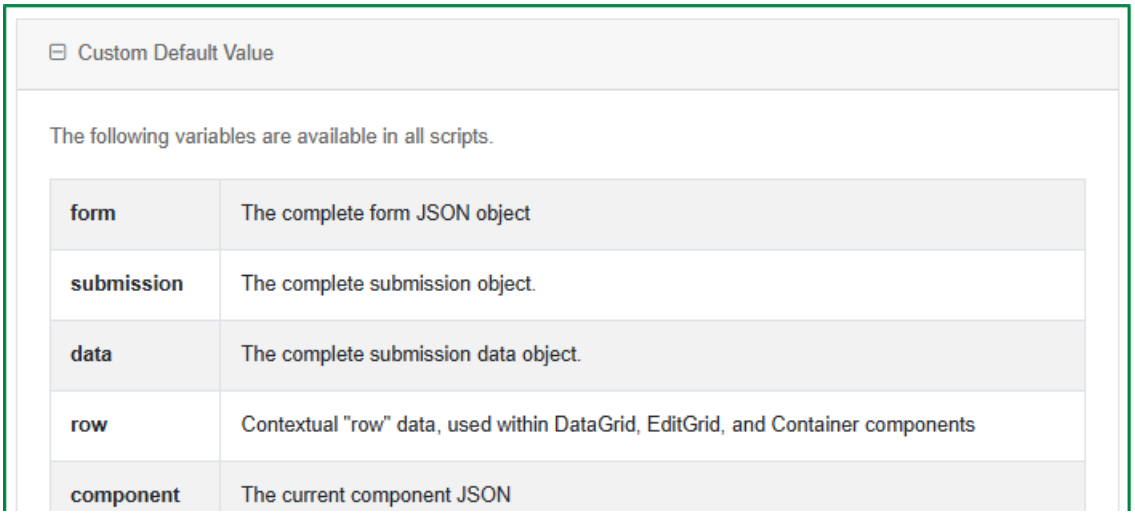
Clear Value When Hidden

This option is used to clear the entered field value when the field is hidden. By default, this settings is on. Untick the checkbox if you want the field value to be retained and subsequently re-displayed when using conditional logic.



Custom Default Value

With this option, you cab write custom Javascript or JSON Logic to set the default value of the field.



instance	The current component instance.
value	The current value of the component.
moment	The moment.js library for date manipulation.
_	An instance of Lodash.
utils	An instance of the FormioUtils object.
util	An alias for "utils".

JavaScript

```
1
```

Enter custom javascript code.

Example:

```
value = data.firstName + " " + data.lastName;
```

JSONLogic

Execute custom logic using JSONLogic.

Full Lodash support is provided using an "_" before each operation, such as `{"_sum": {var: "data.a"}}`

Example:

```
{"cat": [{"var": "data.firstName"}, " ", {"var": "data.lastName"}]}
```

```
1 ""
```

Calculated Value

With this option, you can write custom Javascript or JSON logic to set up calculation logic on the field.

Calculated Value

The following variables are available in all scripts.

token	The decoded JWT token for the authenticated user.
form	The complete form JSON object
submission	The complete submission object.
data	The complete submission data object.
row	Contextual "row" data, used within DataGrid, EditGrid, and Container components
component	The current component JSON
instance	The current component instance.
value	The current value of the component.
moment	The moment.js library for date manipulation.
_	An instance of Lodash.
utils	An instance of the FormioUtils object.
util	An alias for "utils".

JavaScript

```
1
```

Enter custom javascript code.

Example:

```
value = data.a + data.b + data.c;
```

JSONLogic

Execute custom logic using JSONLogic.

Full Lodash support is provided using an "_" before each operation, such as `{"_sum": {var: "data.a"}}`

Example:

```
{"+": [{"var": "data.a"}, {"var": "data.b"}, {"var": "data.c"}]}
```

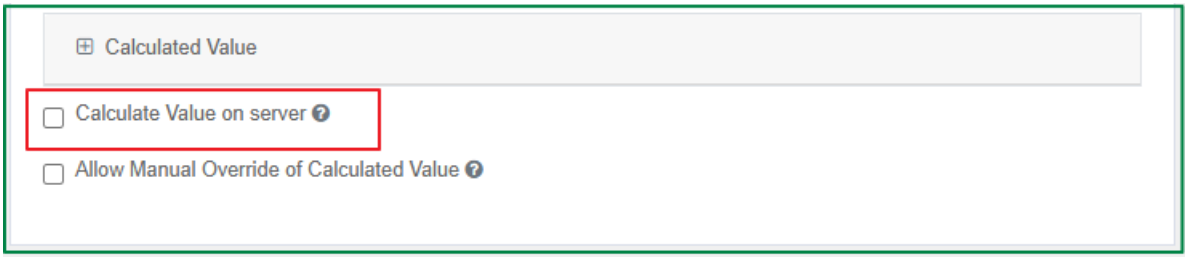
Click here for an example

```
1 ""
```



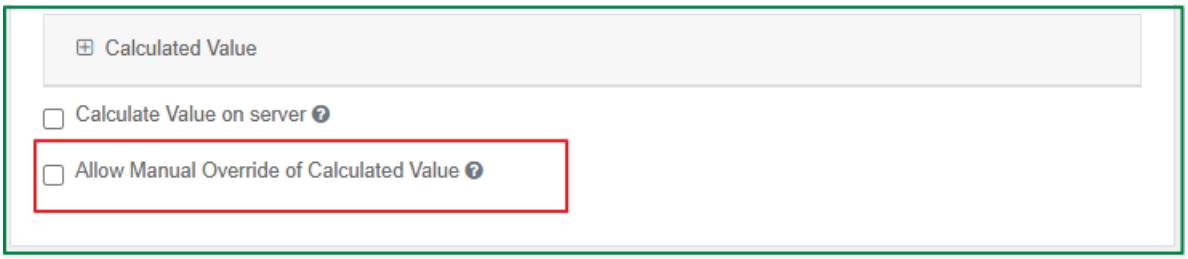
Calculate Value On Server

This option is used to run the calculations on the server side. If you want to override the values submitted on the back-end with the calculations performed on the server, this setting is beneficial.



Allow Manual Override of Calculated Value

By default, the end-user cannot input any value to the field having the set calculations. Tick this option if you want the end-user to be able to input data to override the calculated value in the field.



Validation settings

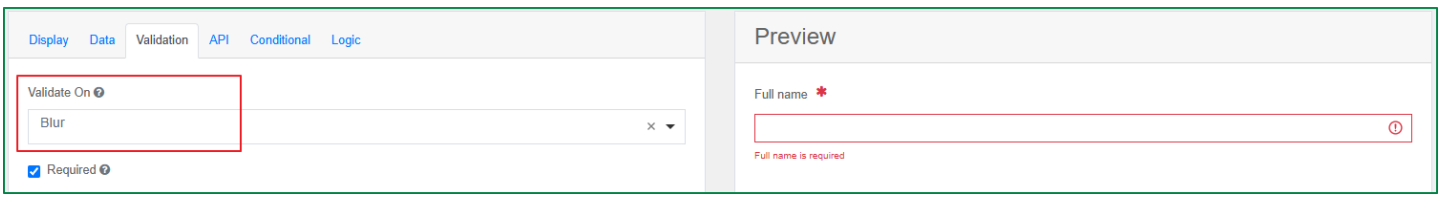
The settings in the **Validation** tab relate directly to the configurable Front-End and Back-End validations for the field, such as required fields, unique data, min/max requirements, custom validations, and custom error messages.

Following is the general settings in the **Validation** tab of form components.

Validate On

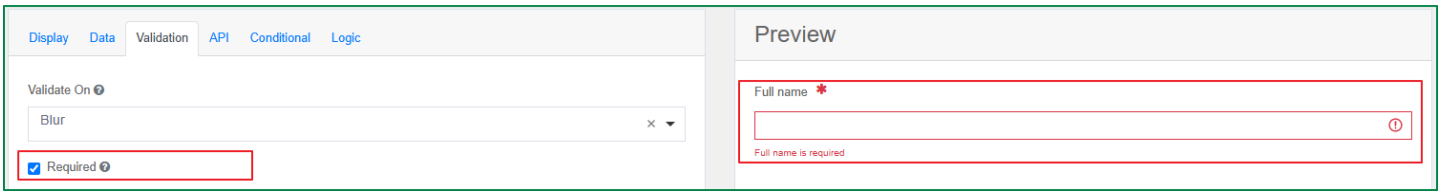
This option is used to determine when the front-end validation of this component should be triggered. There are 2 options: Blur, Change.

- **Blur**: The validation will be triggered when the field is no longer focused. This option is useful for the case that the field requires a certain format, so that the validation will only be triggered when the end-user has completed the data input, for example: email, phone number.
- **Change**: The validation will be triggered whenever it is a change in the field.



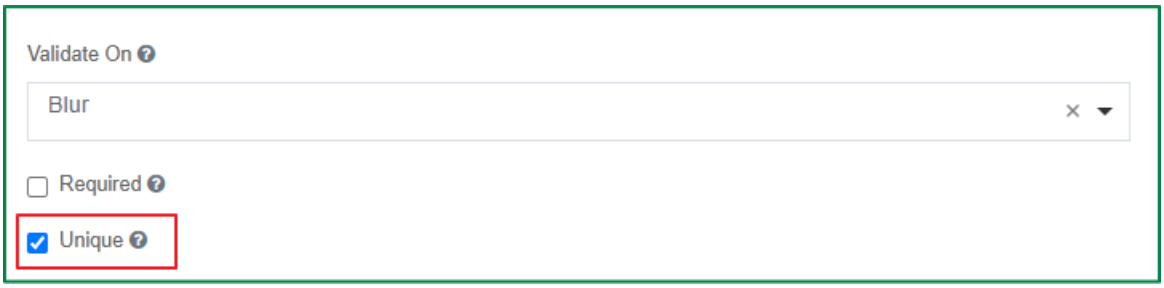
Required

Tick this checkbox if you want to show a red asterisk at the right of the field label, and force the end-user to input data for the field before submitting the form. If the end-user submits the form without filling data for the required field, an error message will be displayed and the form cannot be submitted.



Unique

Tick this checkbox if you want to disallow submitting the same value for the field multiple times. For example: Apply this validation to the Email field of a registration form, so that an email address cannot be submitted twice.

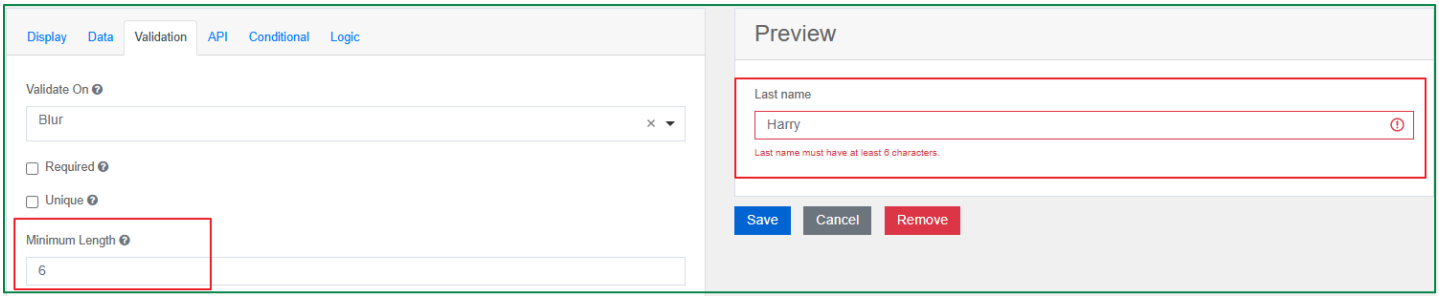


When this setting is on, on the rendered form, if the end-user submits the form with the duplicated value of the field, the submission will be prevented and the error message of duplicated value will be displayed.



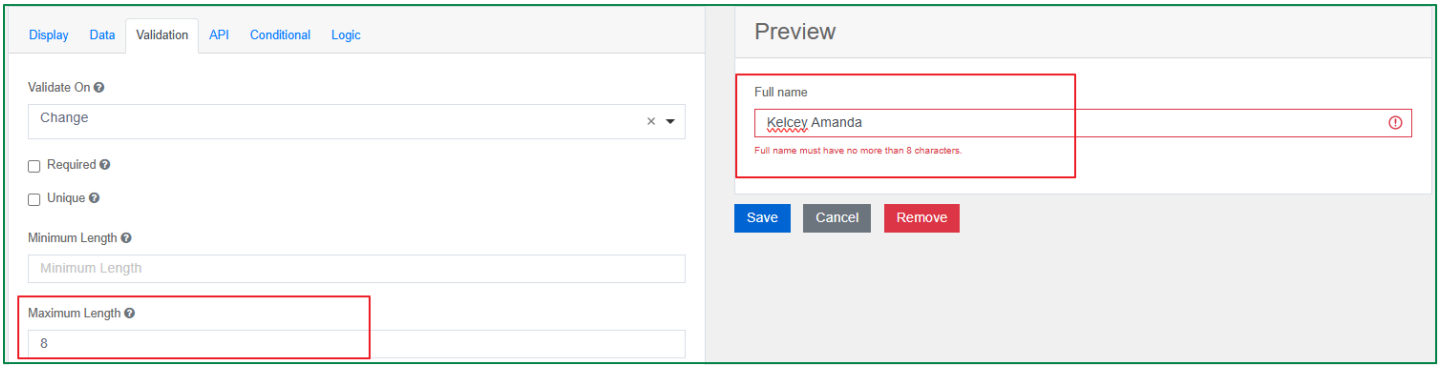
Minimum Length

This option is used to define the minimum number of characters of the input data. If the end-user input the data which is less than the minimum length, the error message of minimum number of characters will be displayed.



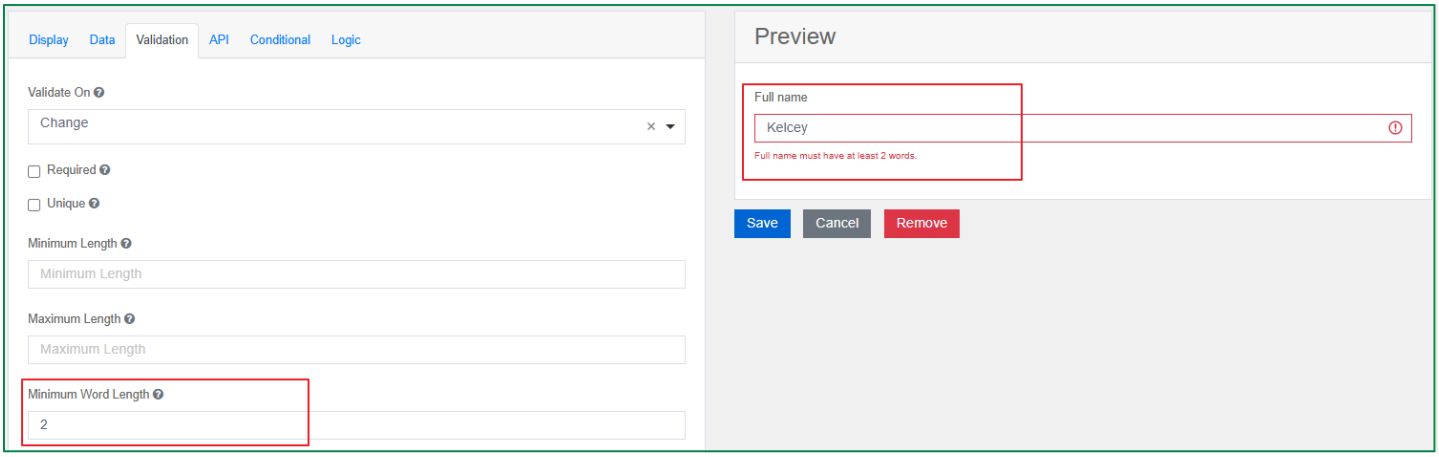
Maximum Length

This option is used to define the maximum number of characters of the input data. If the end-user input the data which is greater than the maximum length, the error message of maximum number of characters will be displayed.



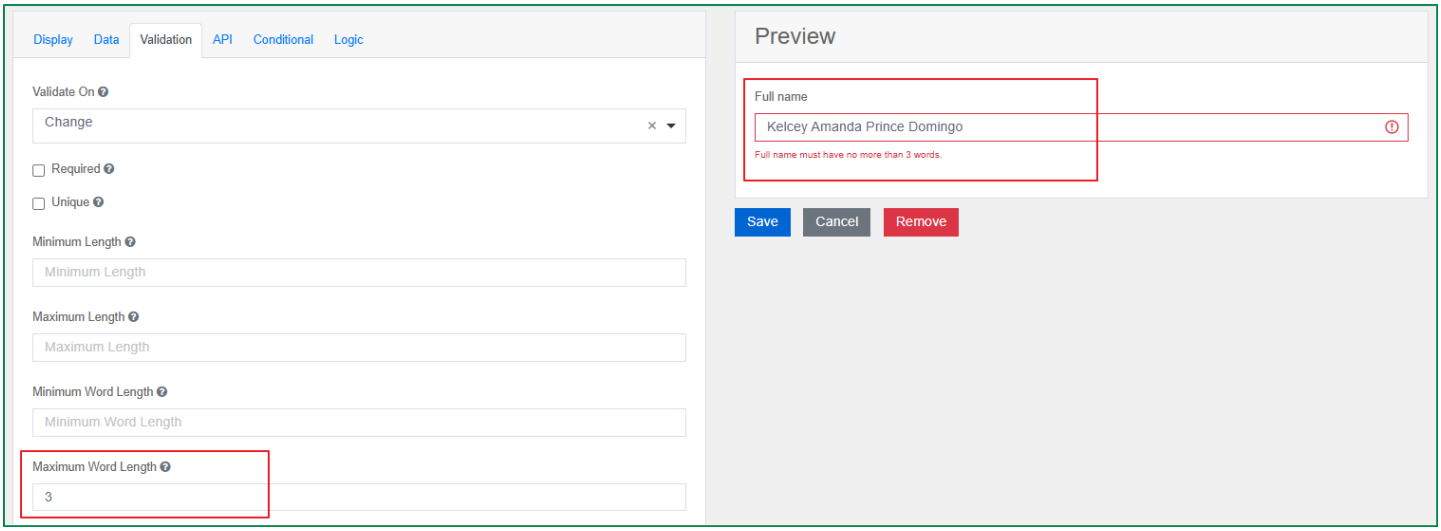
Minimum Word Length

This option is used to define the minimum number of words of the input data. If the end-user input the data which is less than the minimum word length, the error message of minimum number of words will be displayed.



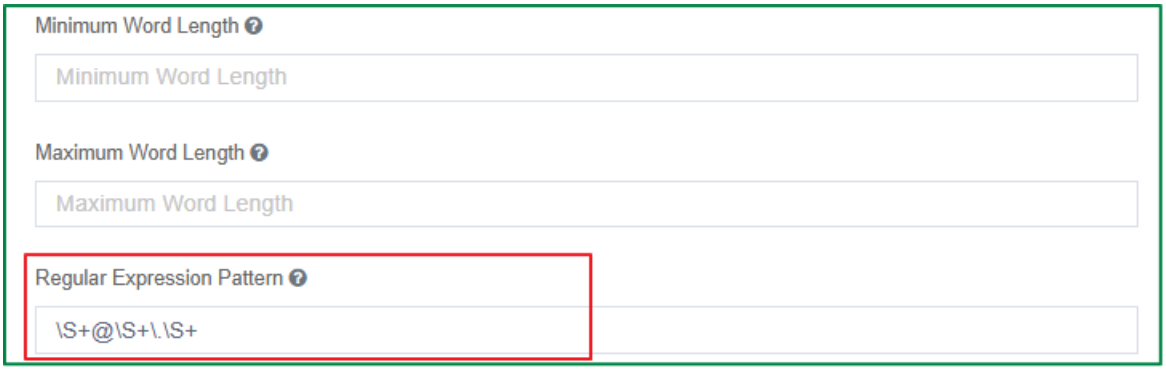
Maximum Word Length

This option is used to define the maximum number of words of the input data. If the end-user input the data which is greater than the maximum word length, the error message of maximum number of words will be displayed.

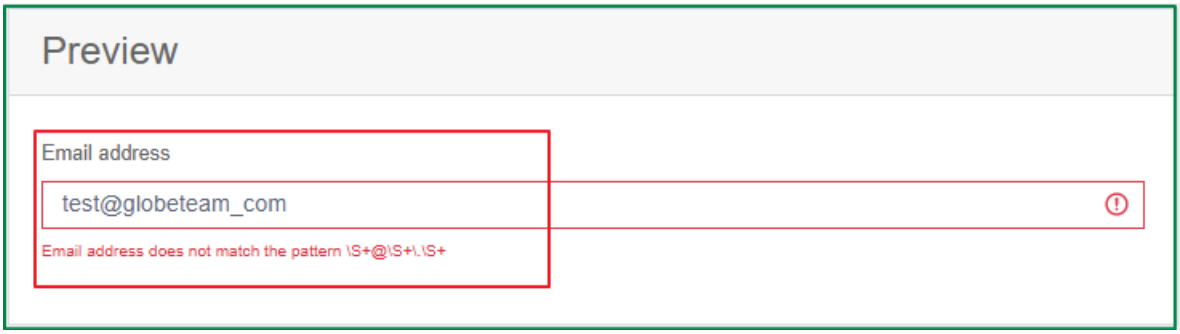


Regular Expression Pattern

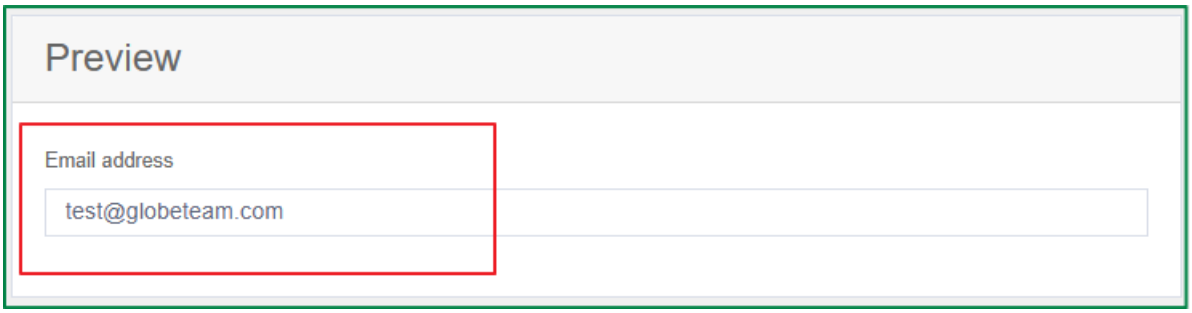
With this option, you can define a Regex pattern (or filter) that details a set of strings that must match the pattern. Following is the example of the email address pattern



On the rendered form, if the end-user inputs the data which does not match the defined pattern, error message of not matched pattern will be displayed as following.

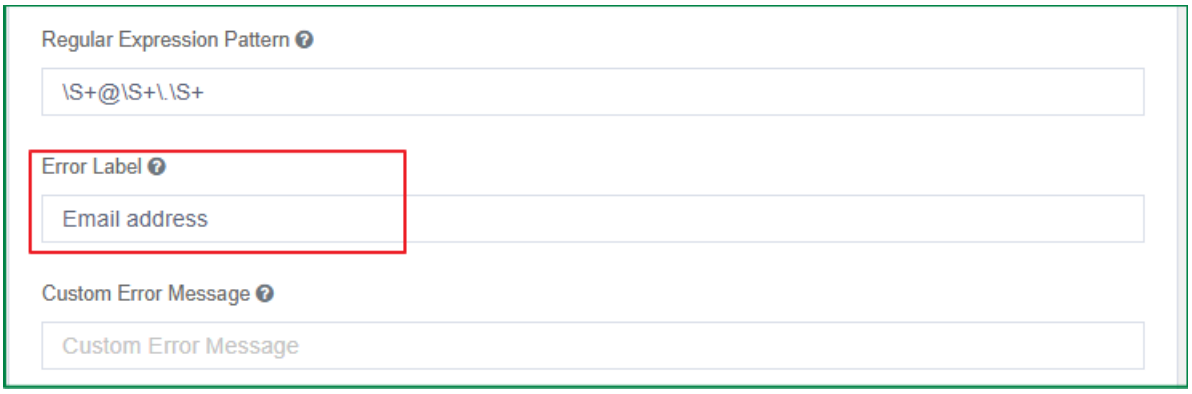


And the validation is passed if the entered data matches the defined pattern

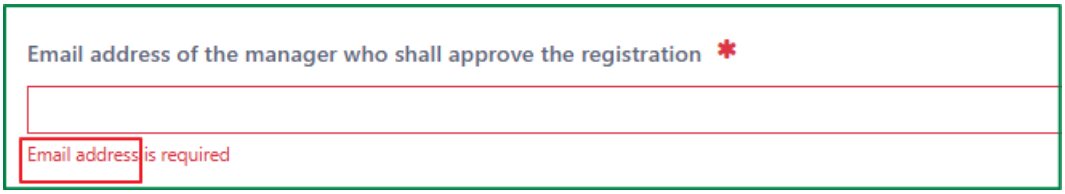


Error Label

With this option, you can define the field label which is shown in the error message when the validation is triggered. This option is useful for the case that the field label is too long and you want to shorten the error message.



Following is the display of the defined error label in error message on the rendered form:



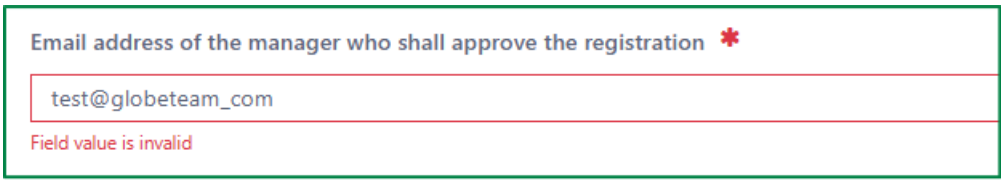
Custom Error Message

With this option, you can define a common error message which is used for all validations.

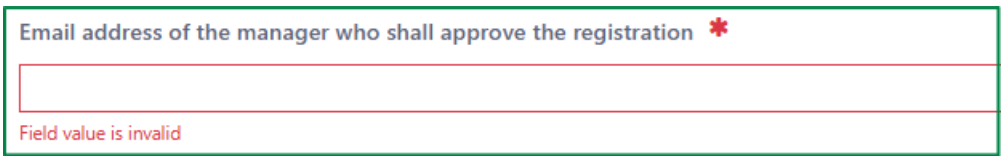


On the rendered form, when any validation is triggered, the common error message is displayed instead of the error message related to the validation.

Following are the examples for the display of common error message for the validation of not matching Regex pattern,



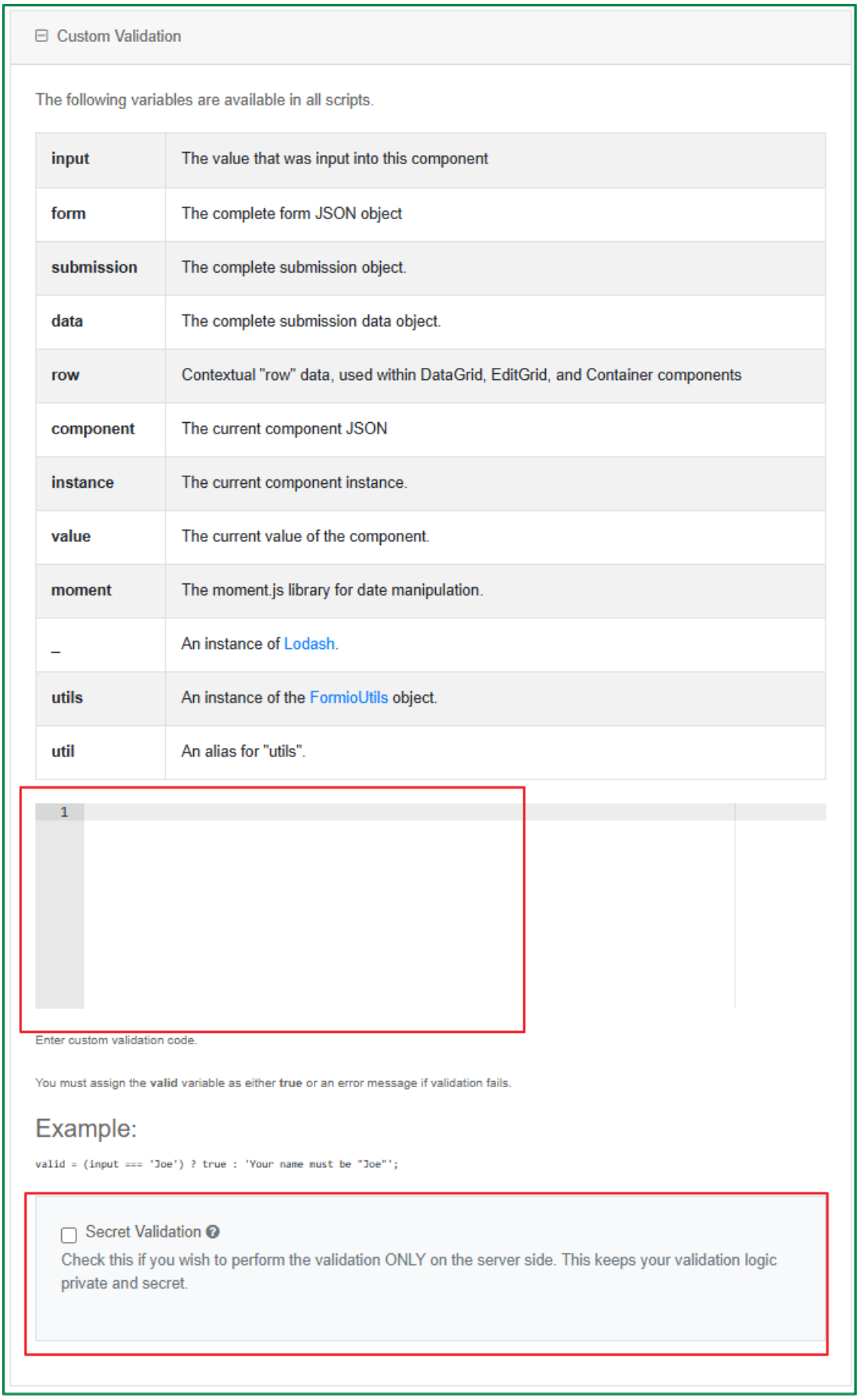
and the validation of required field:



Custom Validation

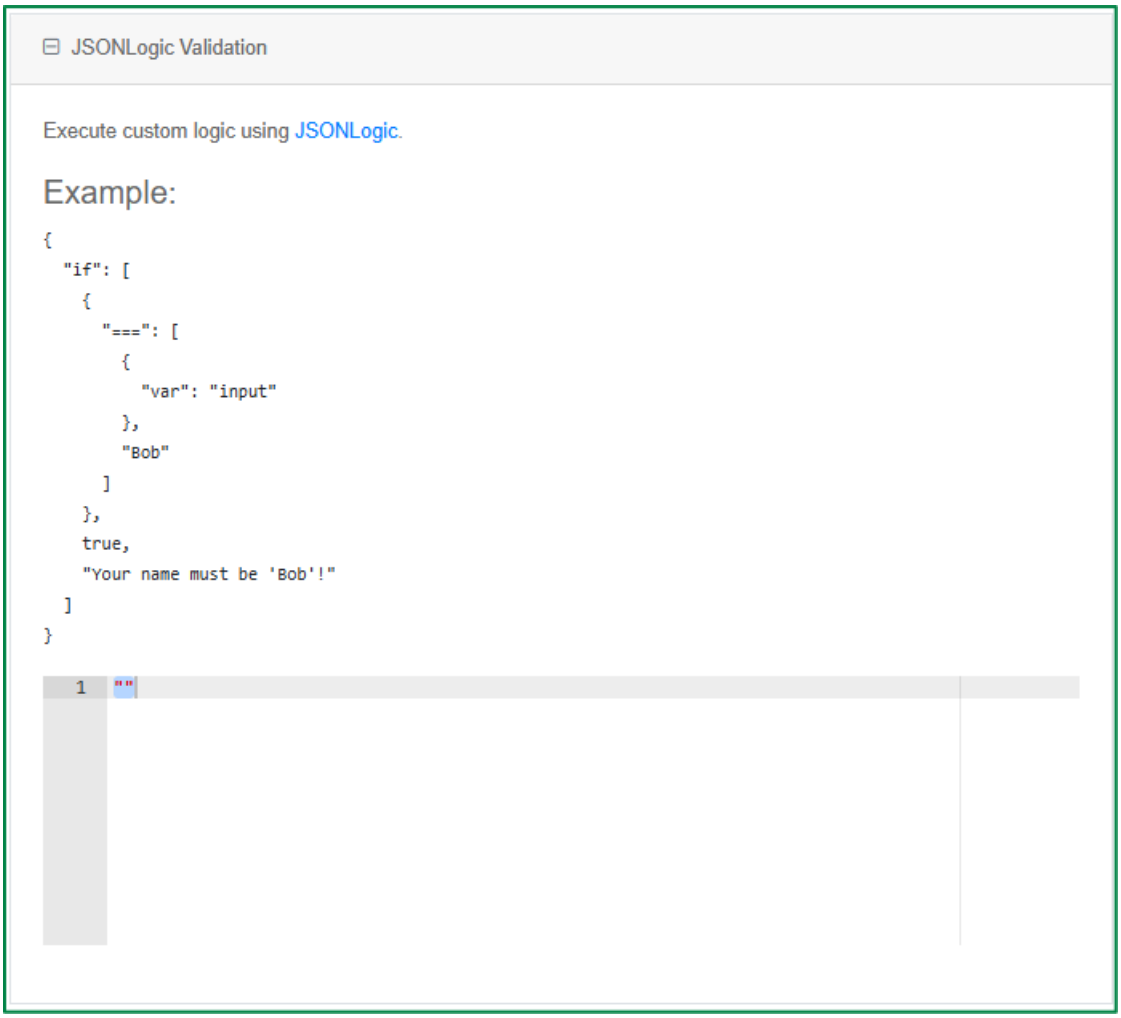
With this option, you can write Javascript to set up your own validation rules for the field.

- **Secret Validation**: Tick this checkbox if you want the custom validation logic is performed only on the server side. This option keeps your validation logic private and secret.



JSONLogic Validation

With this option, you can write JSON logic to set up your own validation rules for the field.



Custom Errors

With this option, you can define different custom error messages for multiple errors, meanwhile the **Custom Error Message** setting only allows you to set one error message for all errors.



API settings

The settings in the API Tab contain Back-End settings such as the Property Name of the field as well as custom properties that can be set such as HTML and Tags.

Read more.

Conditional and Logic settings

The settings in the **Conditional** tab, and **Logic** tab is for the creation of intelligent workflows using Conditions and Logic system of Form.io, such as show or hide the fields, merge component JSON schema, conditionally set field settings, and more, all based on user interaction on the form.

Read more.

Form components and specific settings

Beside general settings, each form component has its specific configuration options, which are different between the components. The configurations allow form admin to turn on authorization, and add on new feature for the component fields. And with these configurations, the form admin have the power to customize the form based on the needs of businesses.

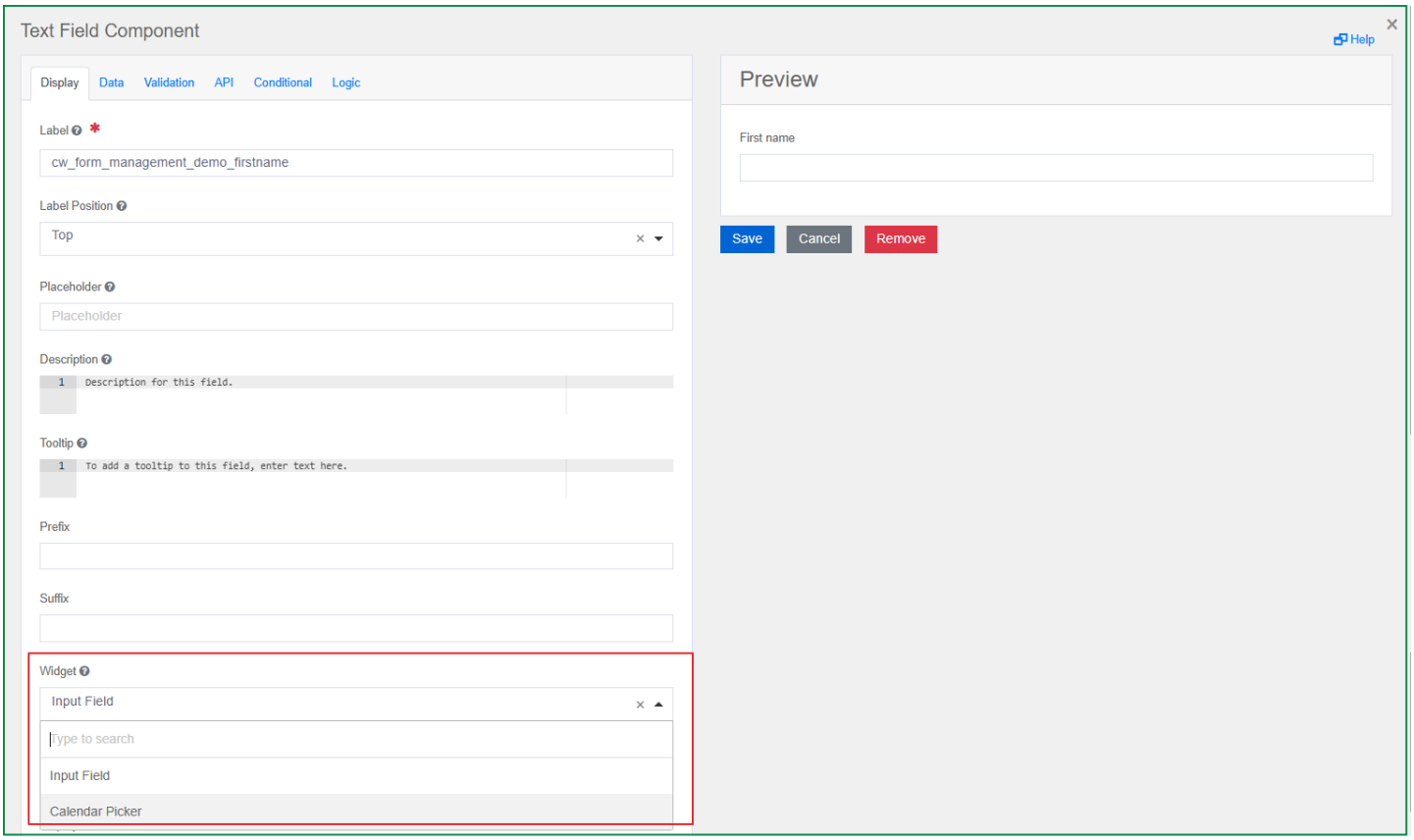
Basic form components

Text field

A Text field can be used for short and general text input. There are options to change the widget, define input masks and validations which allow users to mold information into desired formats.

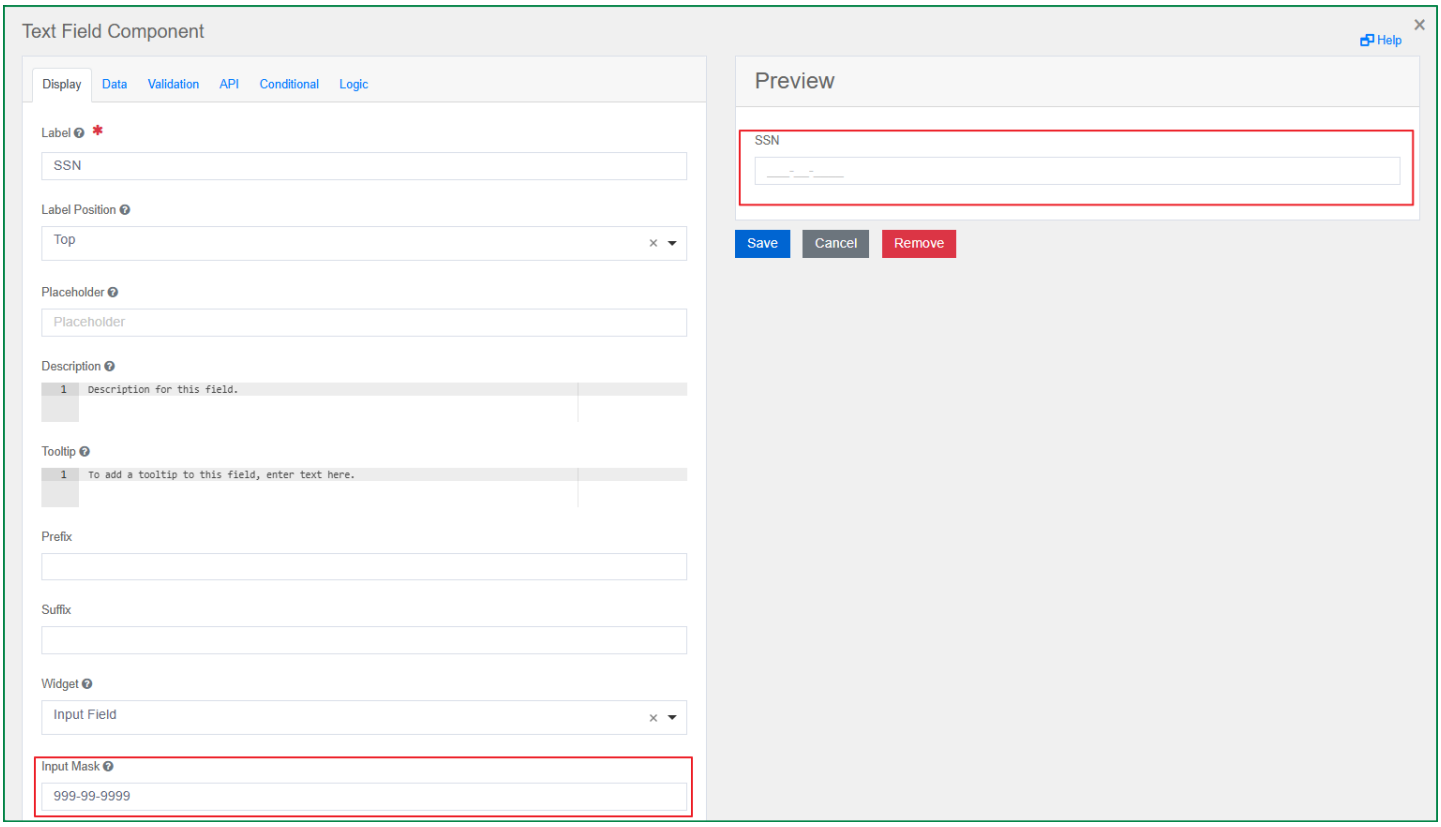
Widget

This option is displayed in the **Display** tab. The widget is the display UI used to input the value of the field. You can change the widget to **Input Field** or **Calendar Picker**



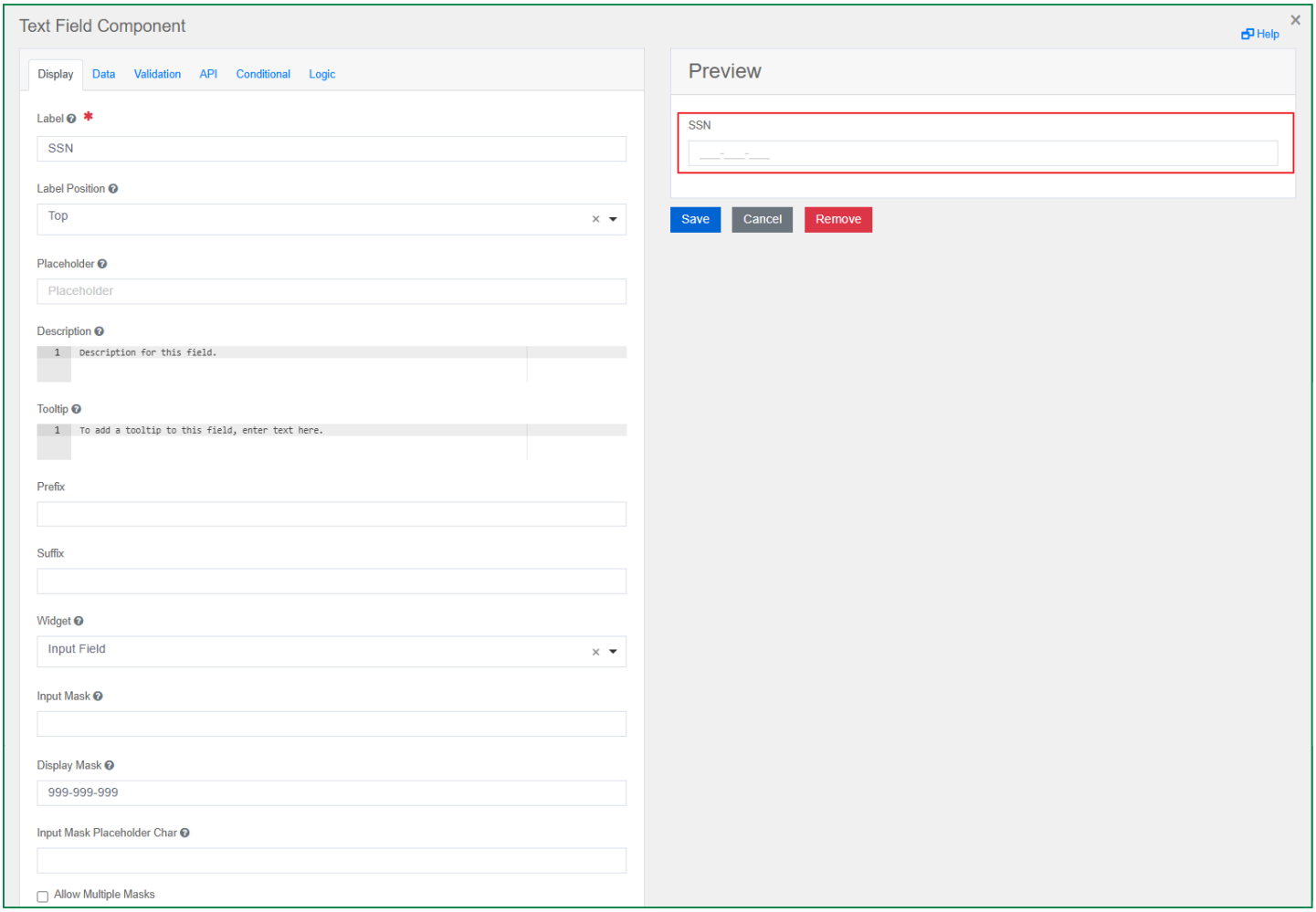
Input Mask

This option is displayed in the **Display** tab. Define the input mask when you want the end-user to input data following a specific format, such as: Postal codes, SSN, etc. The input mask will affect the value which will be saved. If the **Display Mask** is not defined, the field format will follow the input mask.



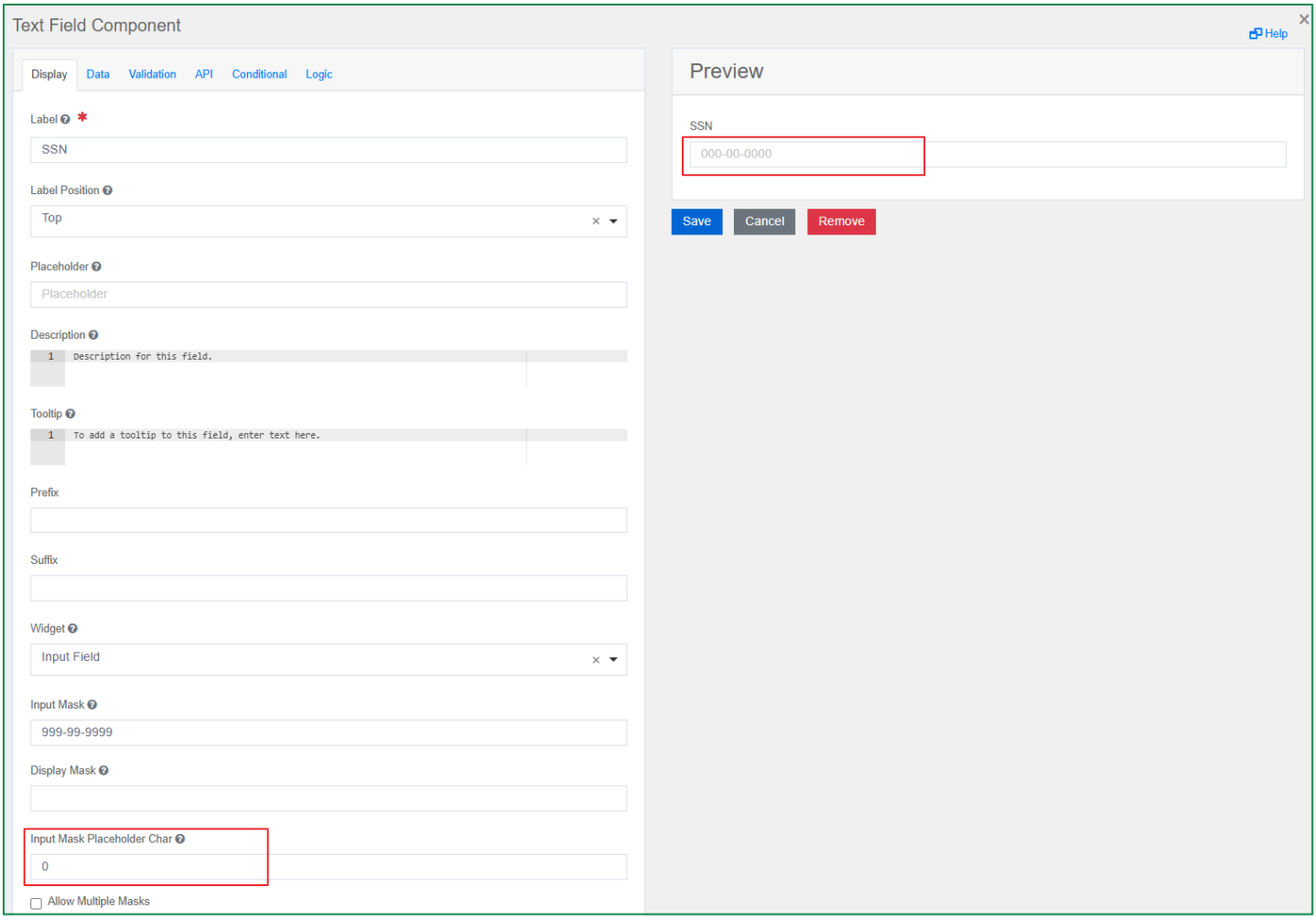
Display Mask

This option is displayed in the **Display** tab. A display mask helps to display the input in a readable way and will not affect the value which will be saved. If you want to affect both view and saved value, you should only use the **Input Mask**.



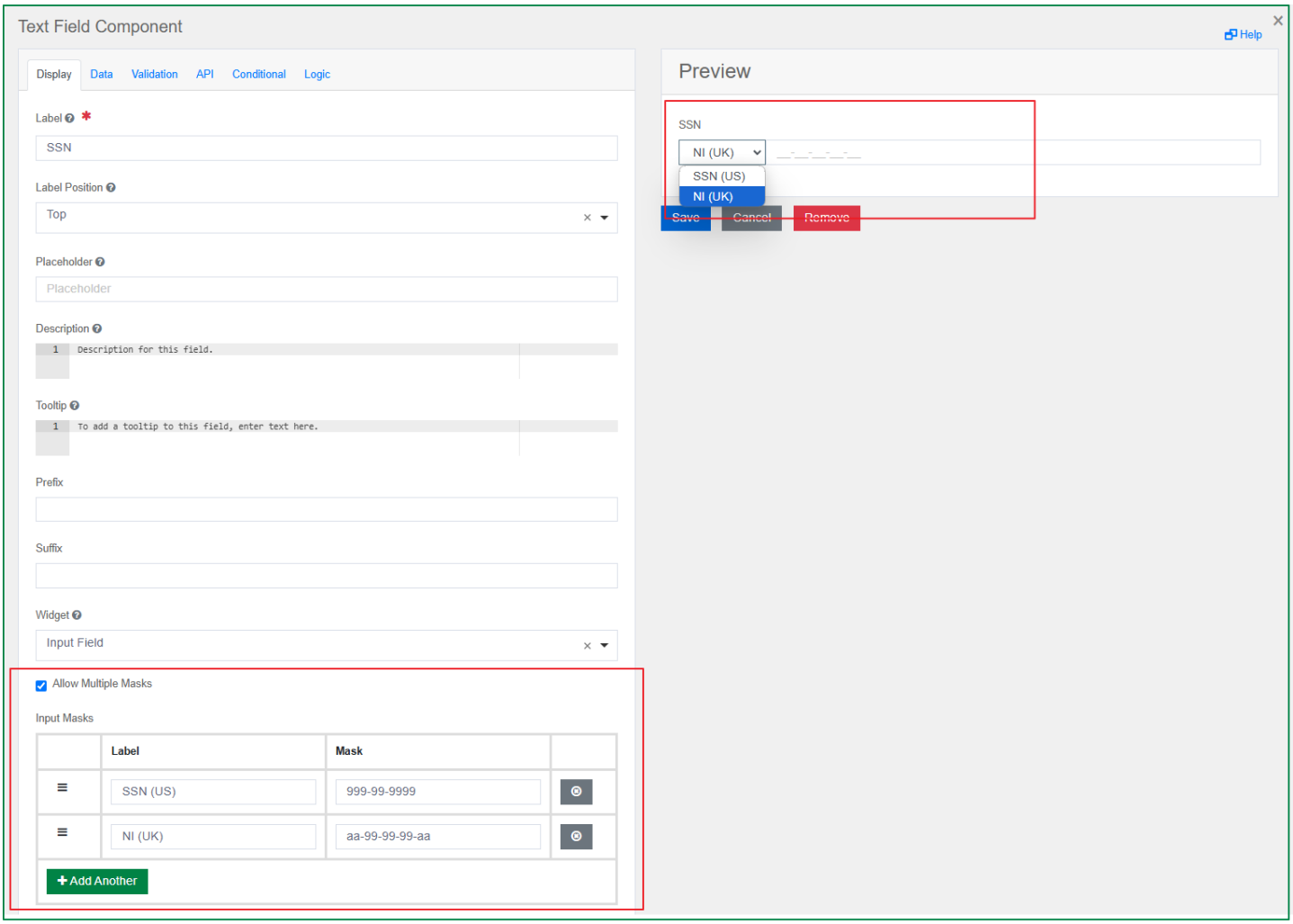
Input Mask Placeholder Char

This option is displayed in the **Display** tab. This option is only enabled if the **Input Mask** or the **Display Mask** is defined. With this option, you can define the character which is used as the placeholder of the field. The character will be displayed following the defined mask.



Allow Multiple Masks

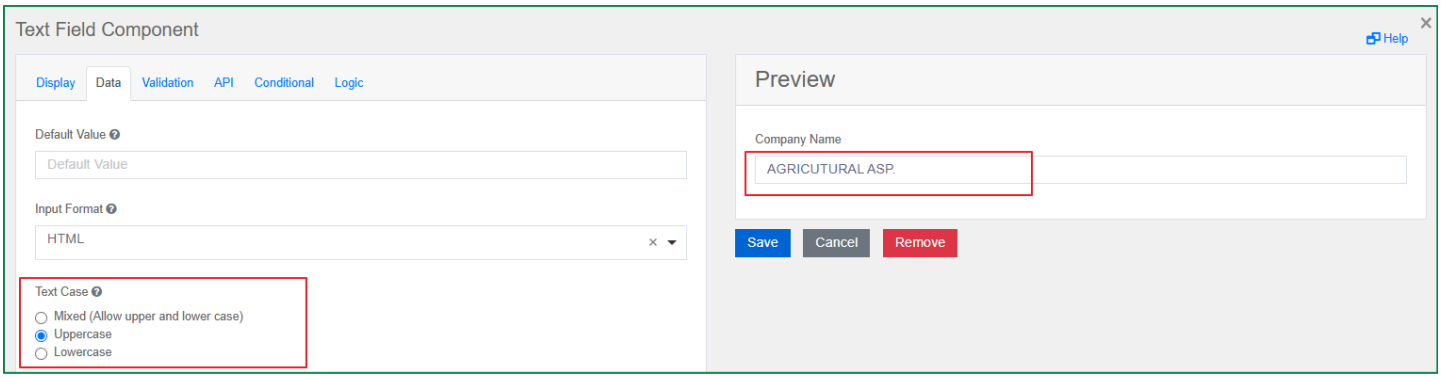
This option is displayed in the **Display** tab. If you want the end-user to be able to select the input format, you can use this settings. When the **Allow Multiple Masks** checkbox is ticked, the **Input Mask** and **Display Mask** fields are hidden. Meanwhile, the grid field **Input Masks** for adding the input masks will be displayed and you can add multiple input masks. If the field has this settings, at the view of end-users, it will be a list of the defined masks for them to select the format they want.



Text Case

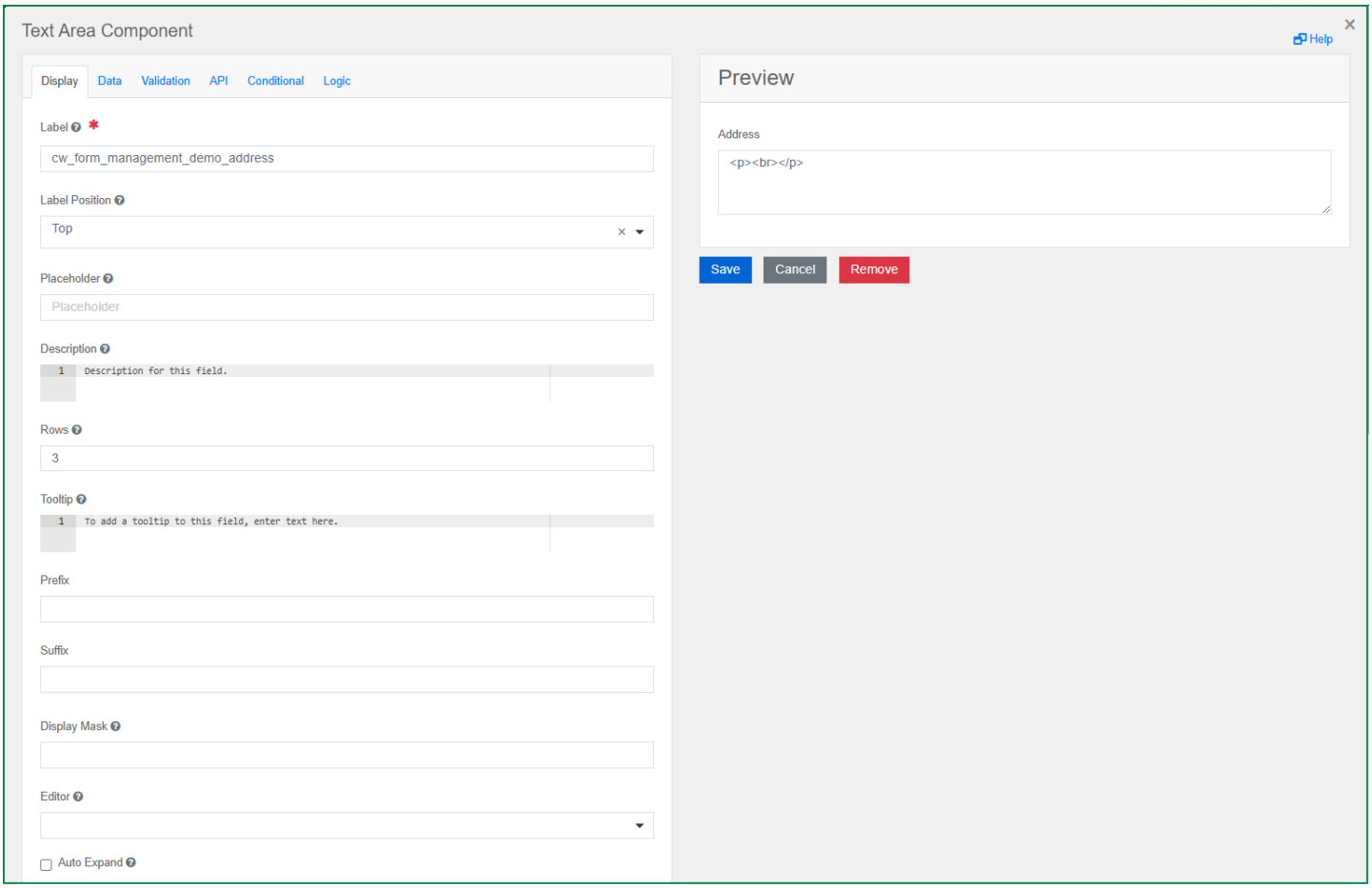
This option is displayed in the Data tab. It is used to defined the case of the value when the end-user input the data. There are 3 options for text case: **Mixed**, **Uppercase**, **Lowercase**.

- **Mixed**: If this option is selected, the entered values are kept as it is, it can be either lowercase or uppercase.
- **Uppercase**: If this option is selected, the entered values are forced to be changed to uppercase.
- **Lowercase**: If this option is selected, the entered values are forced to be changed to lowercase



Text area

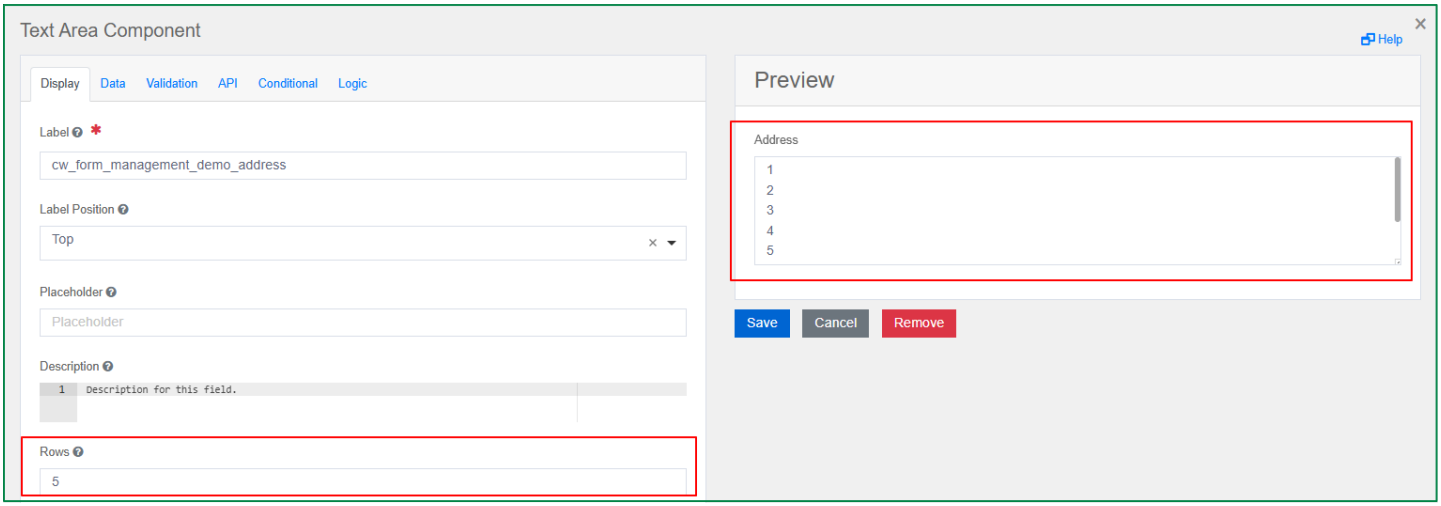
Text area is an input field allowing multiple lines for longer text. Similar to the Text Field component, the Text Area also has the **Display Mask** option. Besides that, you can also convert the text area to a **WYSIWYG editors** which is used by the end-user, and configure the image uploads based on the editor.



Following is the specific settings for the component **Text area**.

Rows

This option is displayed in the **Display** tab. You can define how many default rows are set in the Text Area with this option.

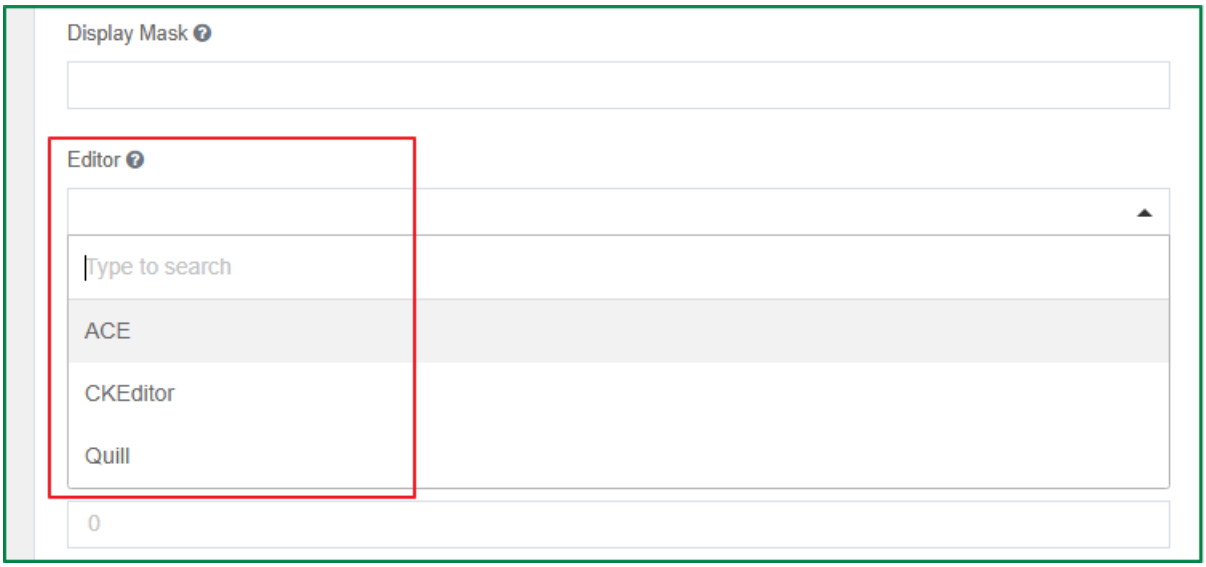


Display Mask

This option is displayed in the **Display** tab. Similar to the Text field component above, the **Display Mask** helps to display the input in a readable way and will not affect the value which will be saved.

Editor

This option is displayed in the **Display** tab. This option will convert the text area into the selected **WYSIWYG Editor** user interface. There are 3 provided editors: **ACE**, **CKEditor**, and **Quill**.



Enable image upload

This option is displayed in the **Display** tab. This option is only enabled and displayed if the selected editor is **CKEditor**, or **Quill**. Tick this checkbox if you want to allow the image uploads to the **WYSIWYG Editor**.

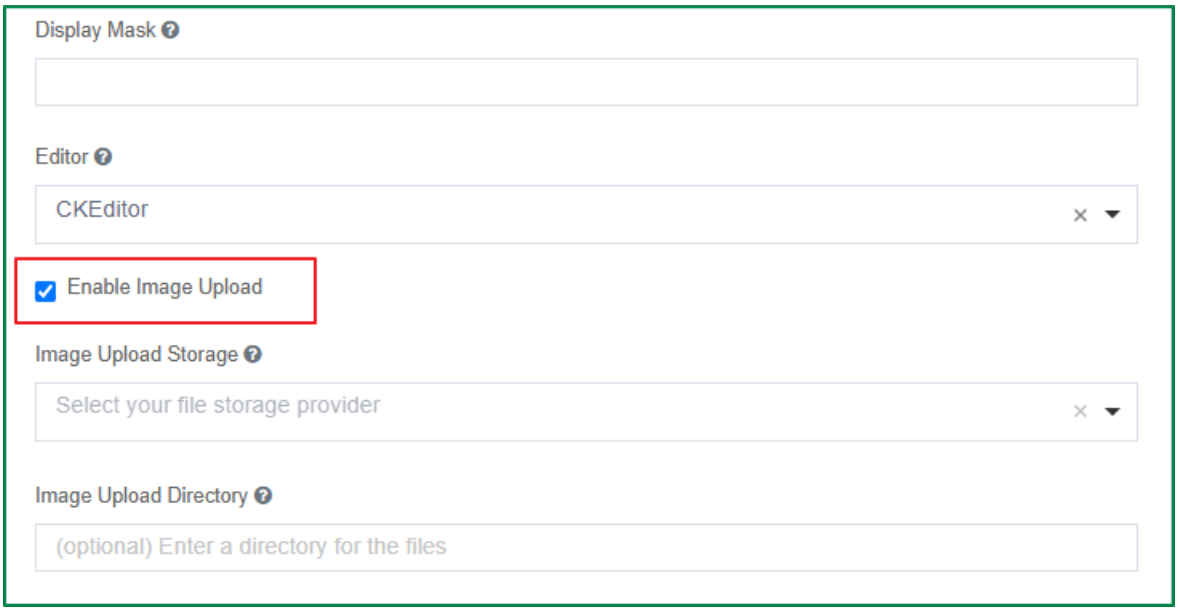


Image Upload Storage

This option is displayed in the **Display** tab. This option is only enabled and displayed if the checkbox **Enable image upload** is ticked. With it, you can select one of the provided storages to save the files in.

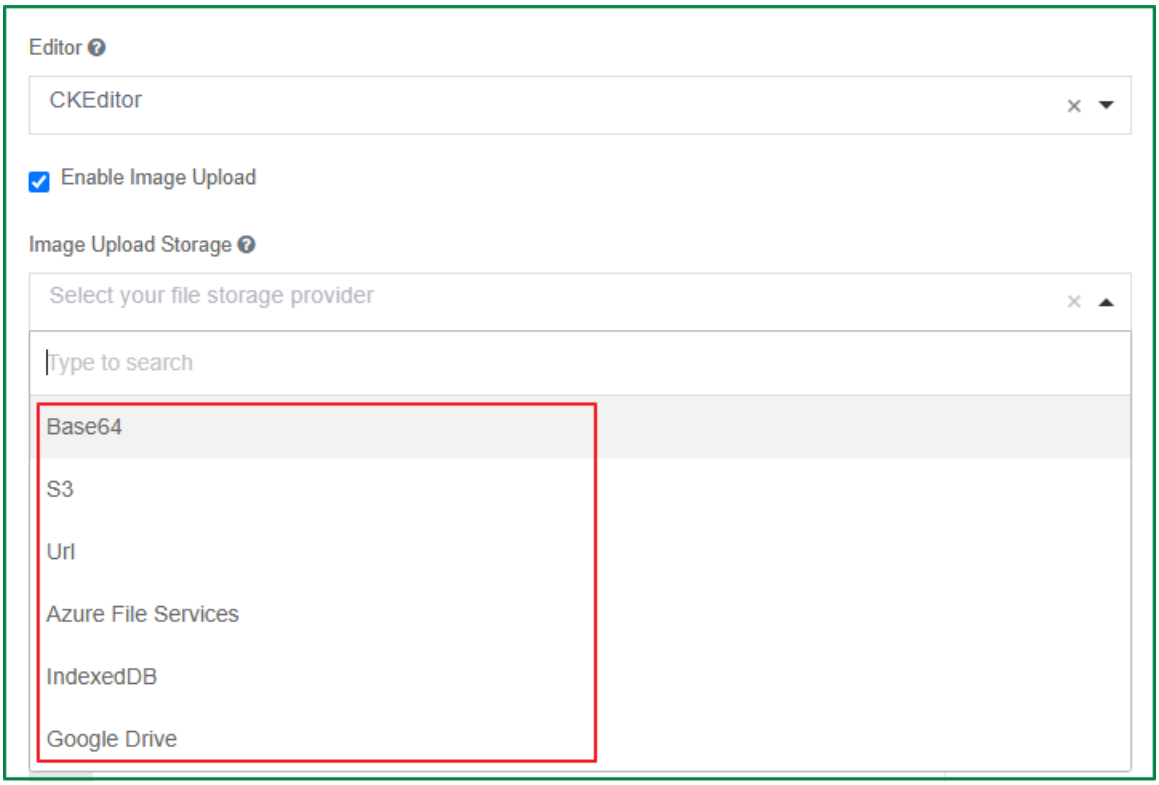


Image Upload Directory

This option is displayed in the **Display** tab. This option is only enabled and displayed if the checkbox **Enable image upload** is ticked. You can enter a directory to place all the uploaded files.

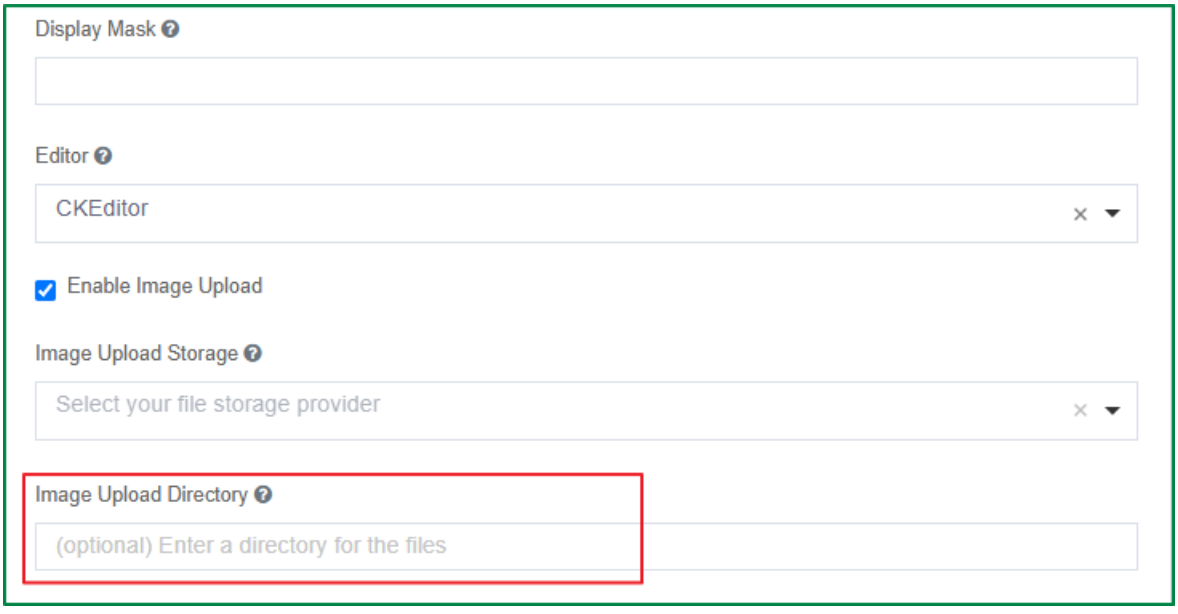
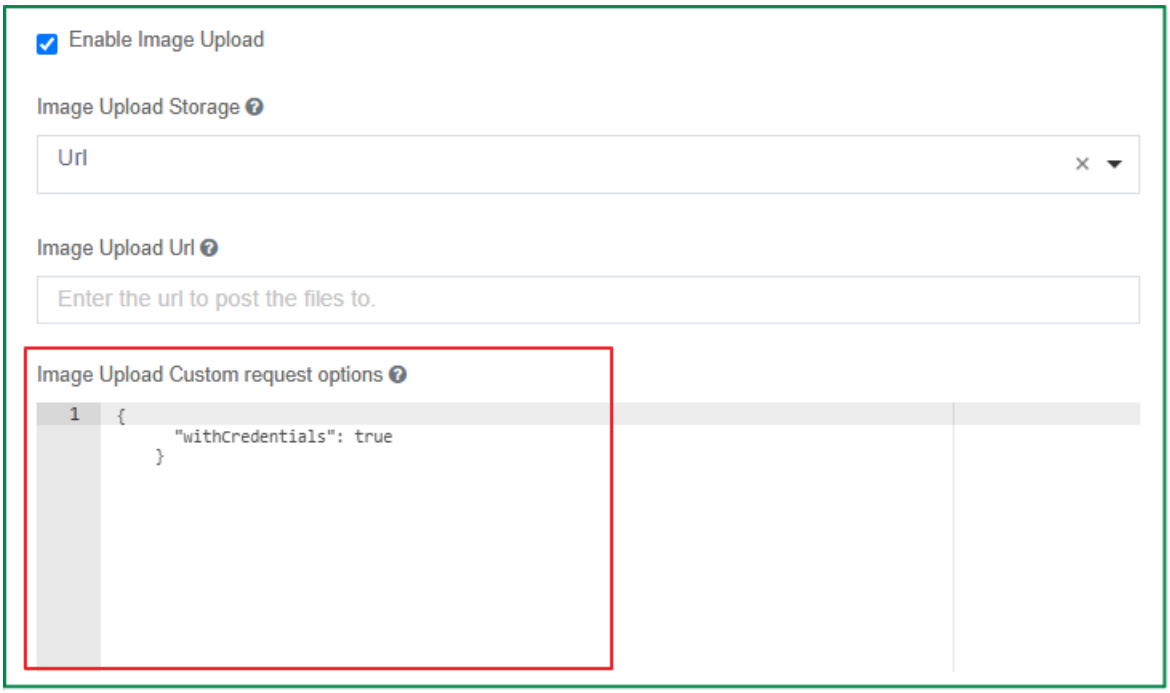


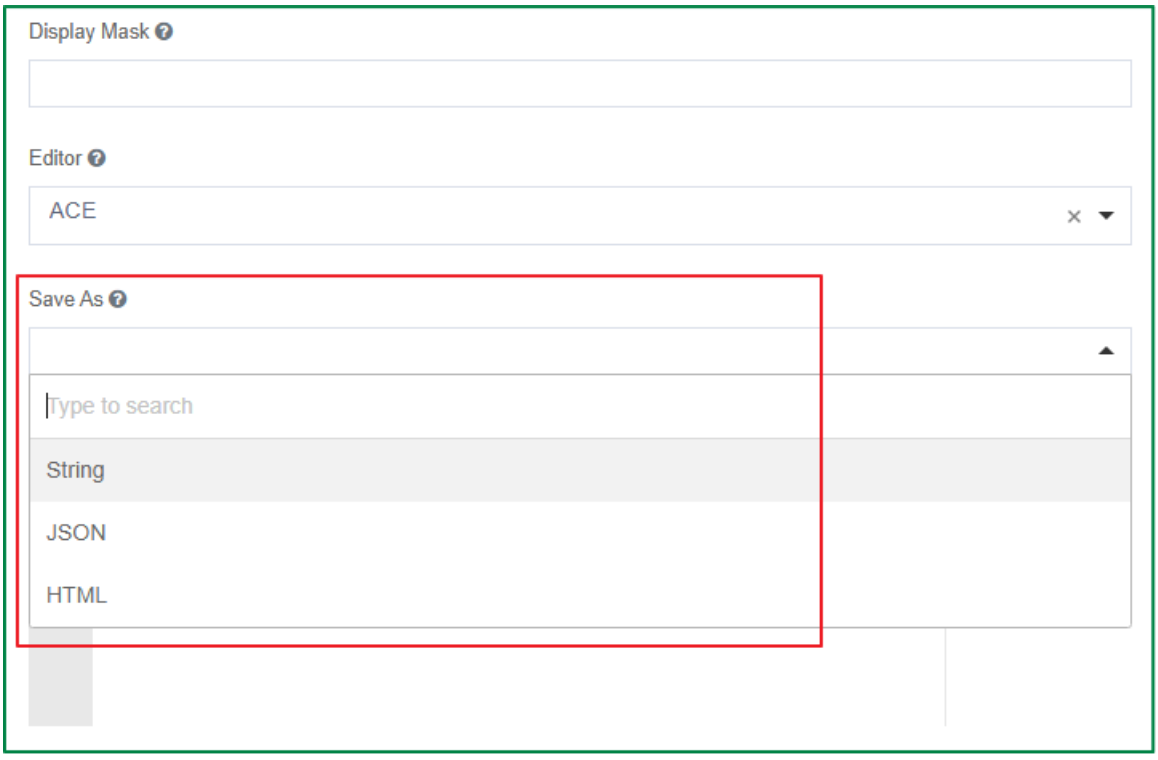
Image Upload Custom request options

This option is displayed in the **Display** tab. This option is only enabled and displayed if the selected image upload storages is **Url**. With this option, you can pass your custom xhr options.



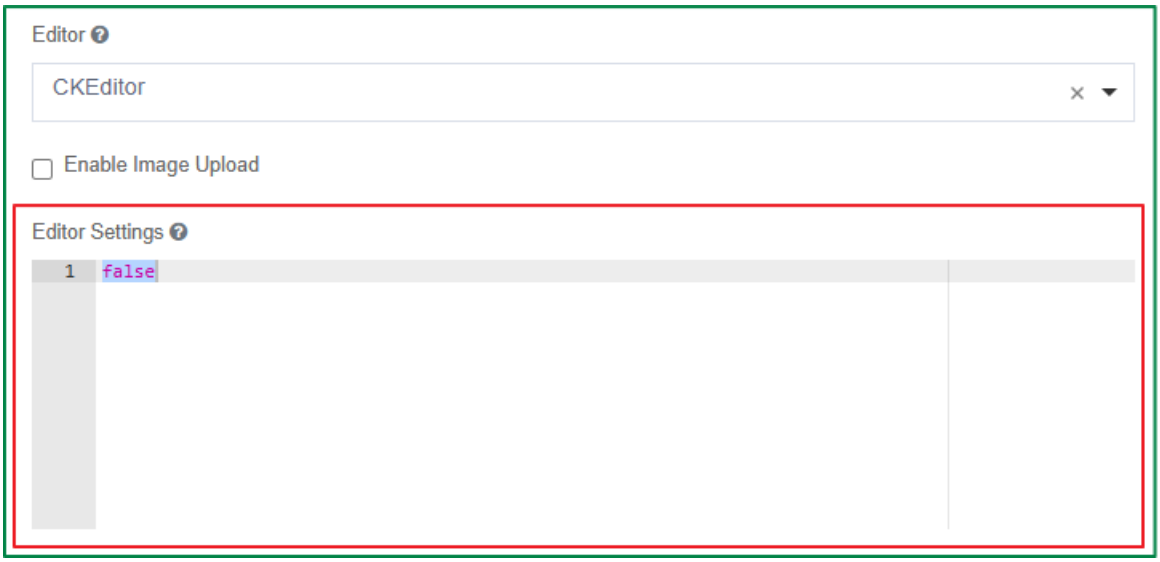
Save as

This option is displayed in the **Display** tab. This option is only enabled and displayed if the selected editor is **ACE**, or **Quill**. You can select whether the value should be entered and stored in database as **String**, **JSON**, or **HTML**.



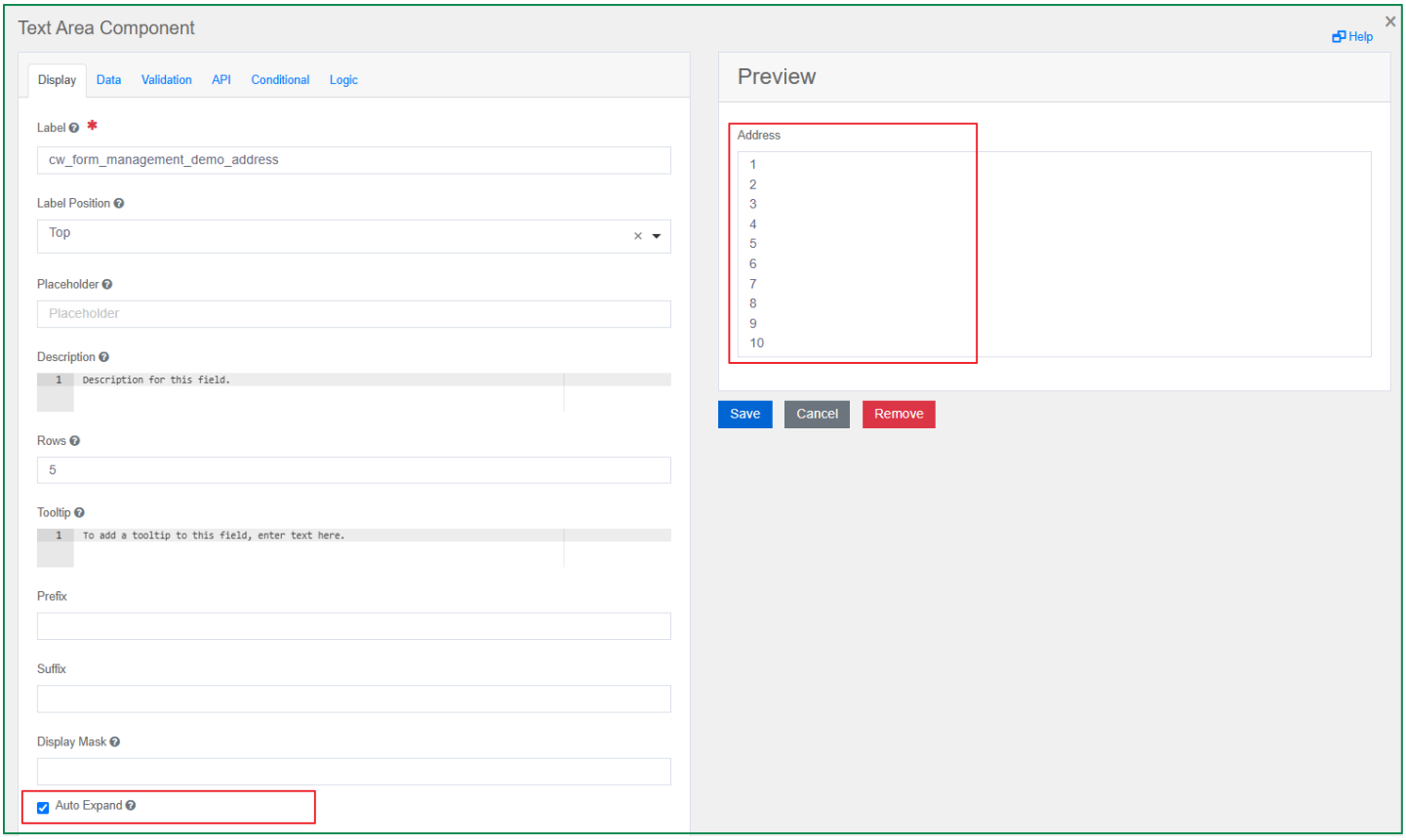
Editor Settings

This option is displayed in the **Display** tab. Here you can modify the code base of the **WYSIWYG Editor** to customize the UI specific to your needs.



Auto expand

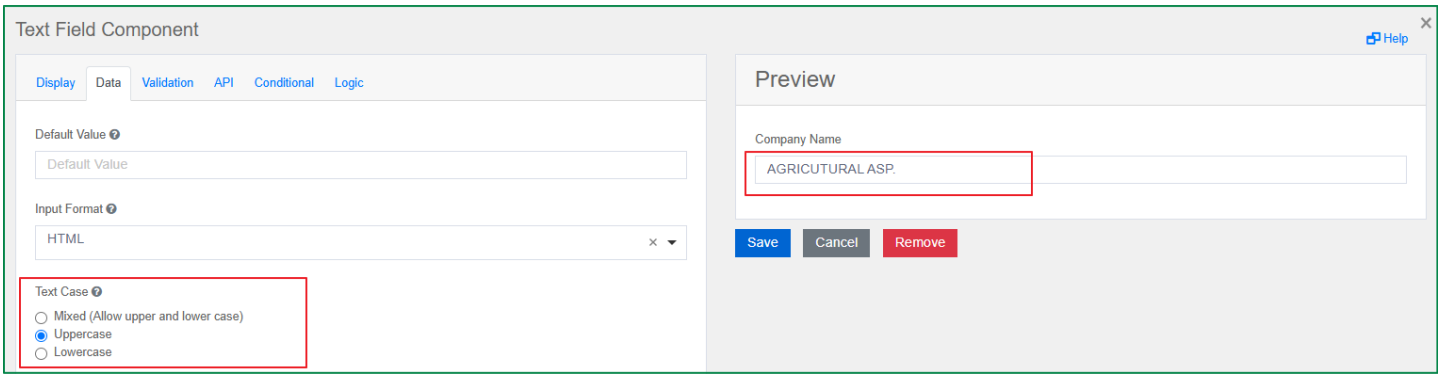
This option is displayed in the **Display** tab. This option is only enabled if no **WYSIWYG editor** is selected. When this option is turned on, the height of the text area field will be expanded automatically according to the number of lines of the entered data. The expansion happens immediately as the user is typing into the field.



Text Case

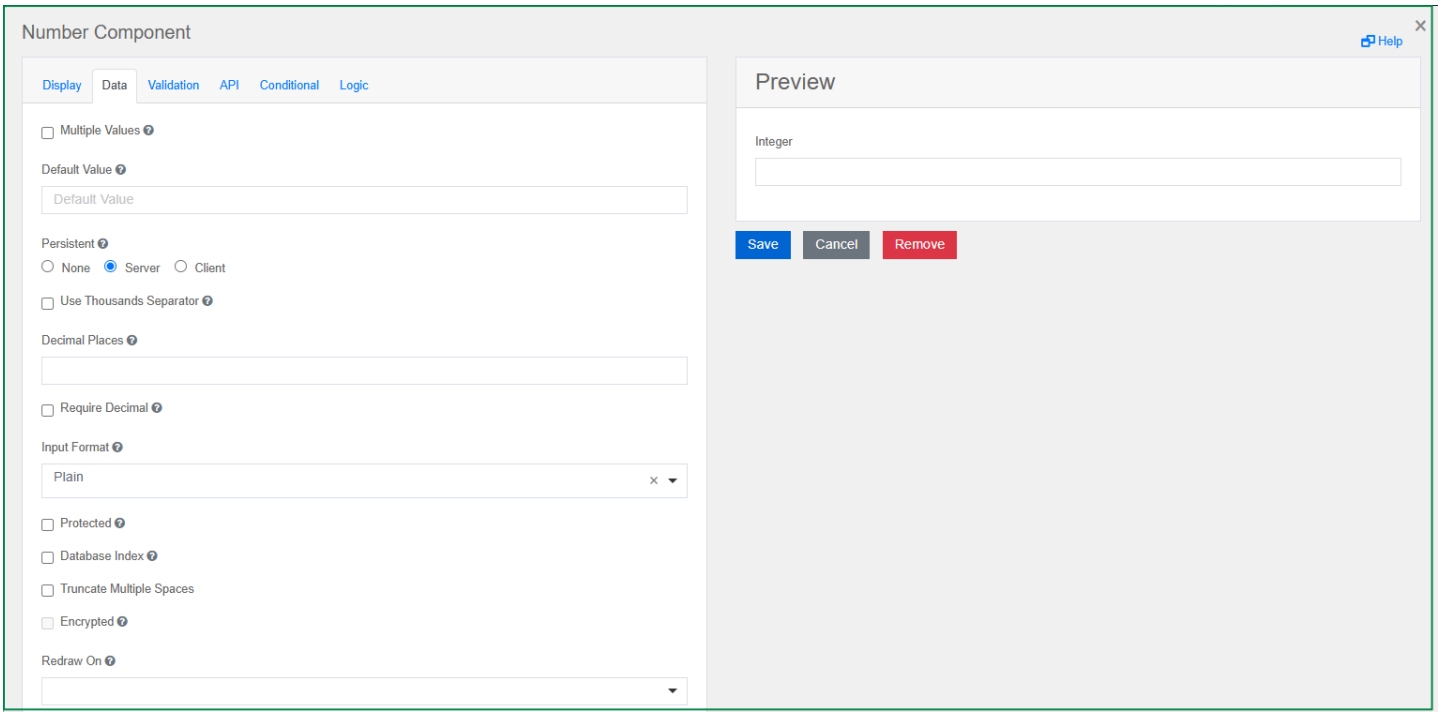
This option is displayed in the Data tab. It is used to defined the case of the value when the end-user input the data. There are 3 options for text case: **Mixed**, **Uppercase**, **Lowercase**.

- **Mixed**: If this option is selected, the entered values are kept as it is, it can be either lowercase or uppercase.
- **Uppercase**: If this option is selected, the entered values are forced to be changed to uppercase.
- **Lowercase**: If this option is selected, the entered values are forced to be changed to lowercase



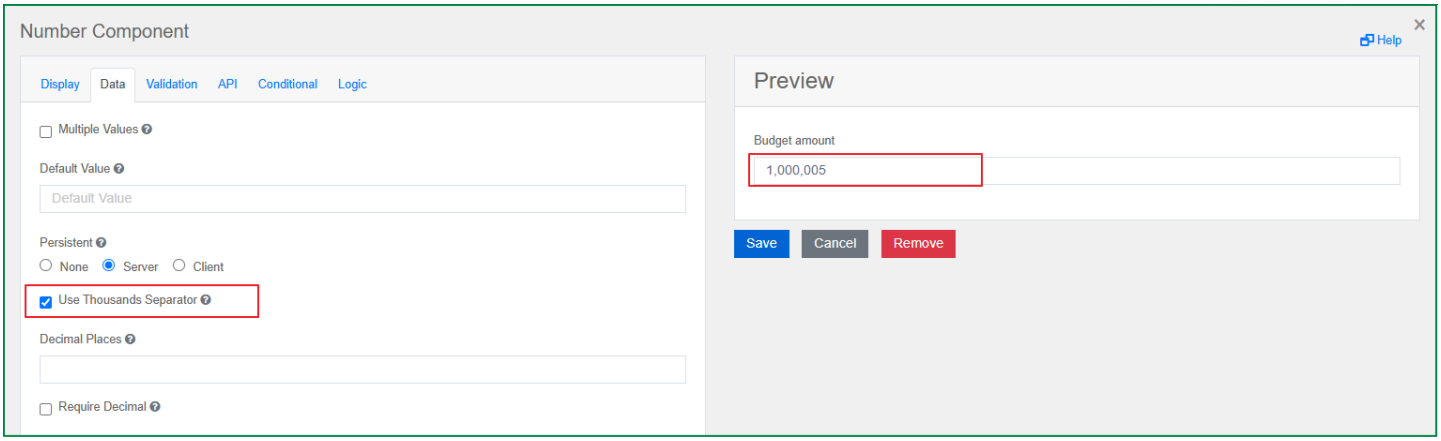
Number

Number field can be used whenever a field should be limited to a type of number value. The specific settings of the Number component includes: **Use Thousands Separator**, **Decimal Places**, and **Require Decimals**.



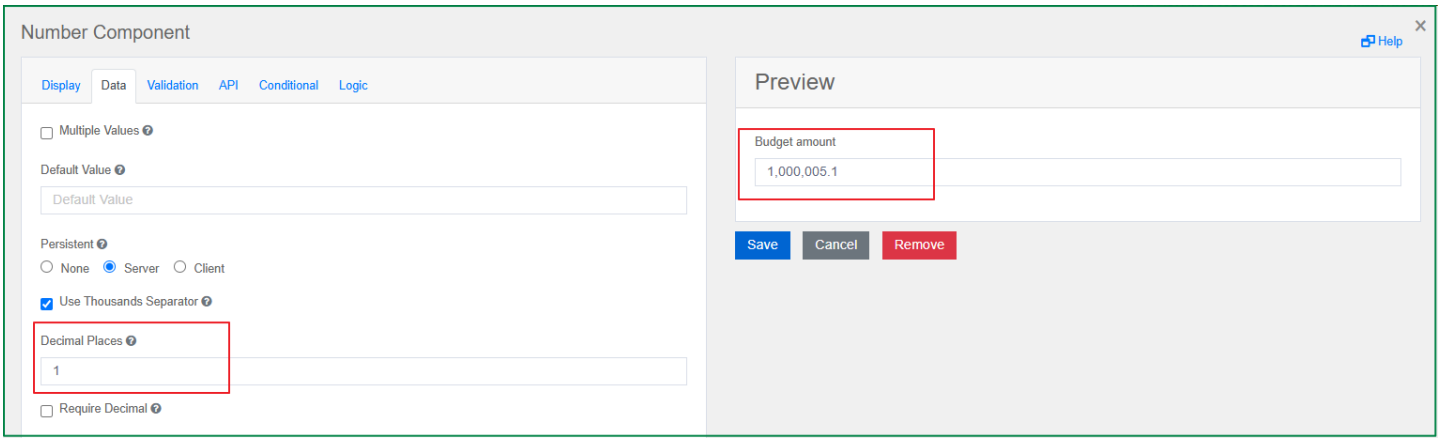
Use Thousands Separator

This option is displayed in the **Data** tab. Tick this option if you want the value in this component to separate thousands by a local delimiter.



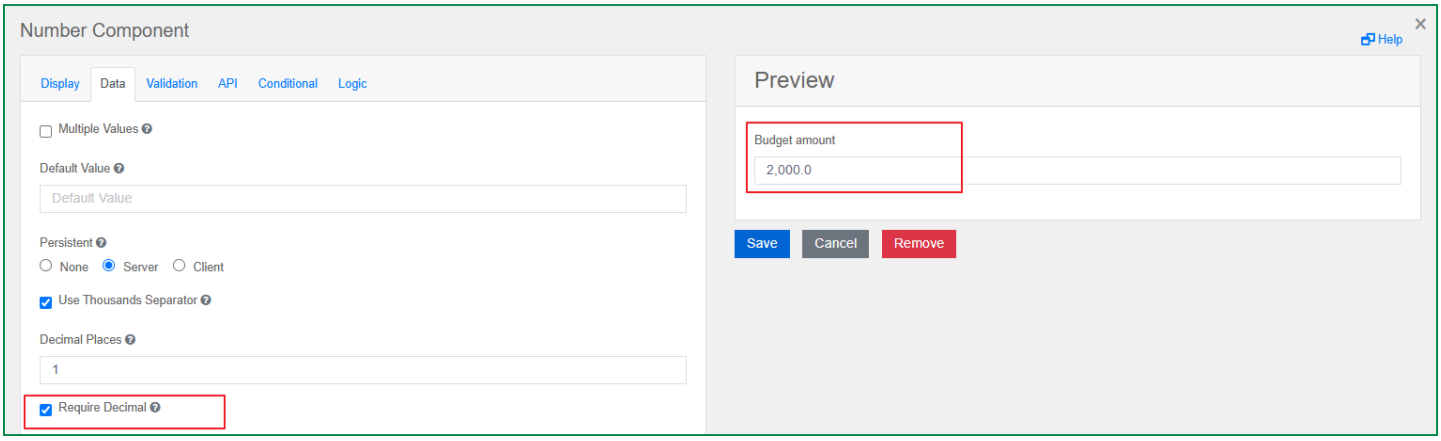
Decimal Places

This option is displayed in the **Data** tab. This option is to specify maximum number of decimal places for the values in this field.



Require Decimal

This option is displayed in the **Data** tab. If you tick this option, the decimals are always shown for this field, even if trailing zeros.



Password

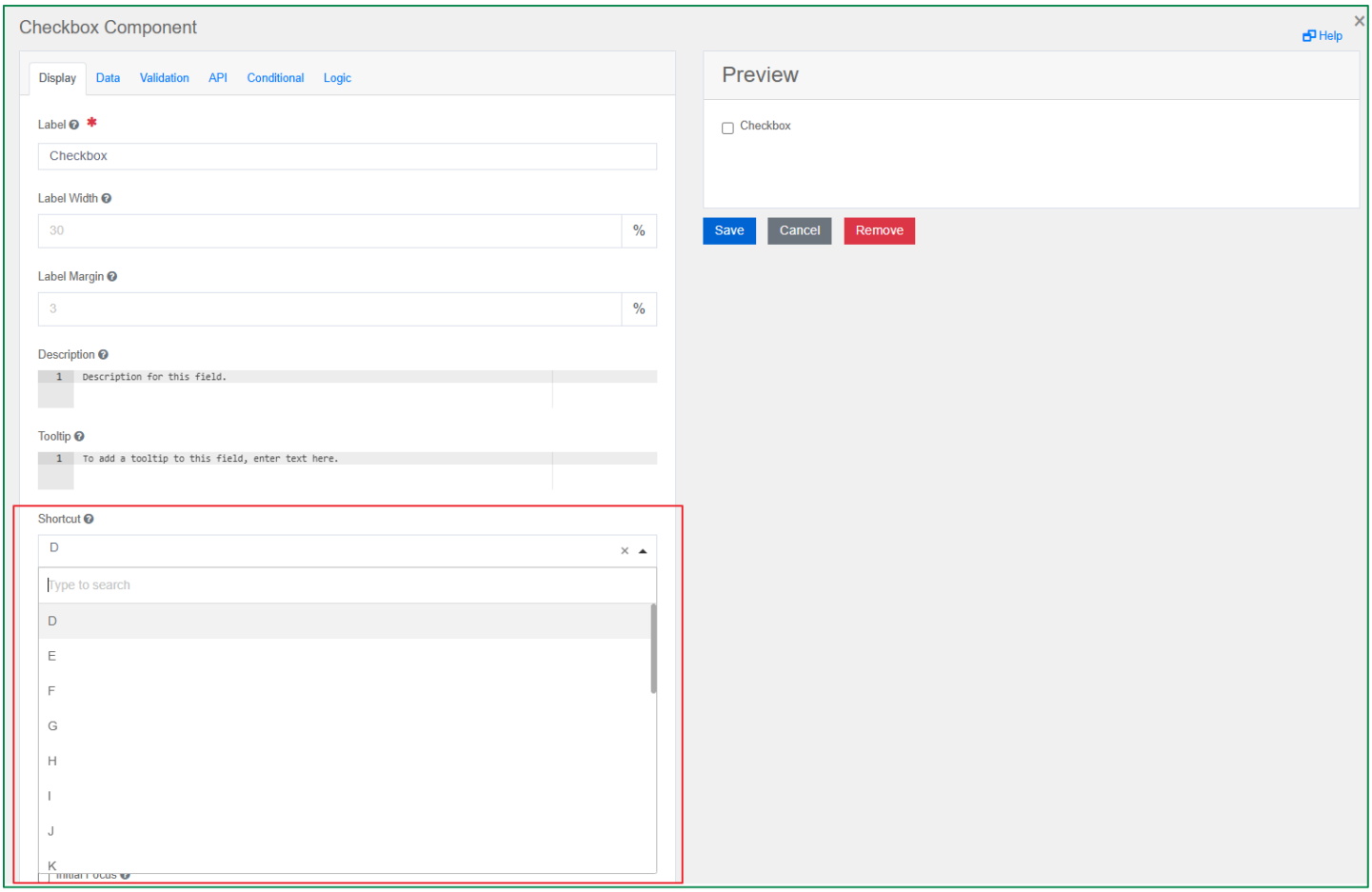
Password component is an input field in which the entered value is displayed as hidden input symbols. Similar to the **Text Field** component, the **Password** component also has the **Display Mask** option. Not like **Text field** component, HTML `<input>` type of the **Password** component is password instead of text. The **Password** component does not have other specific settings.

Check Box

Check Box component can be used for the input field with boolean value. It can either be ticked (true) or unticked (false). You can configure input type for this component.

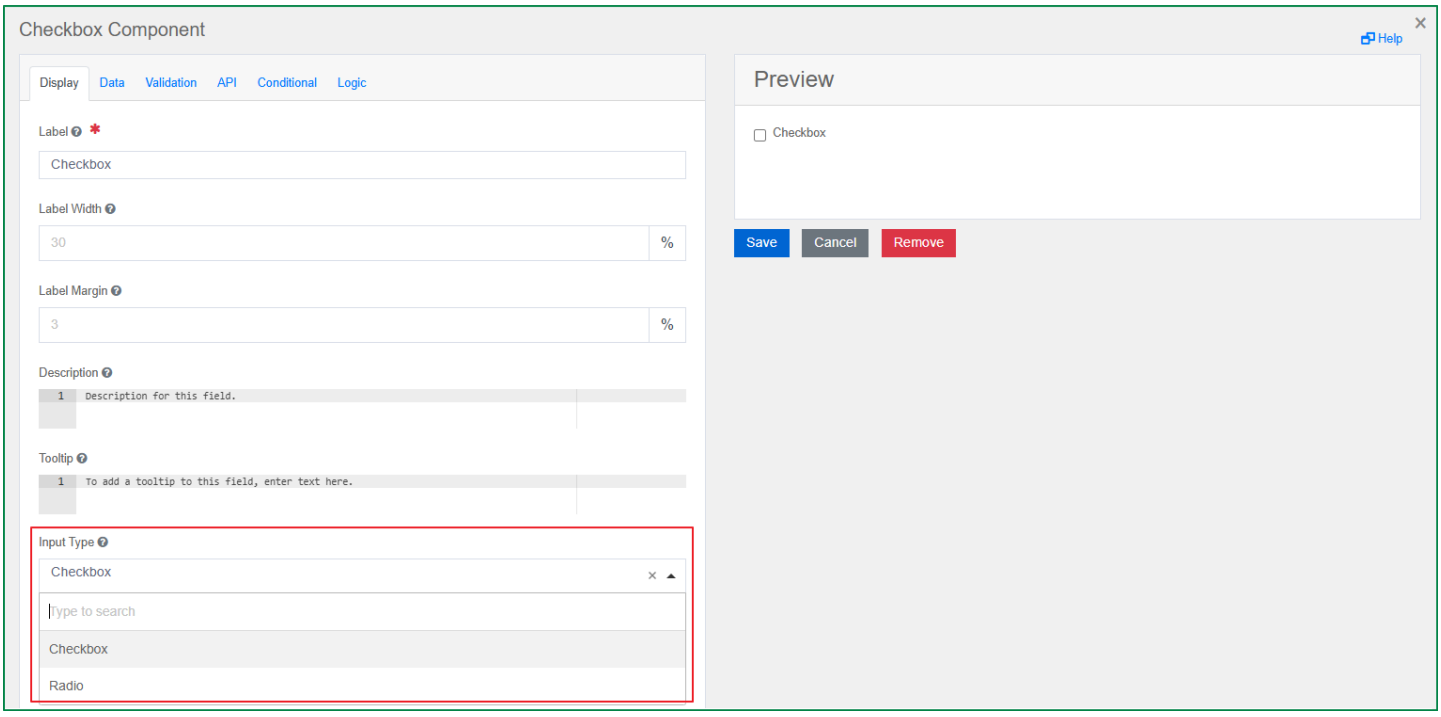
Shortcut

This option is displayed in the **Display** tab. You can select a keyboard from the list which is used as the shortcut to tick or untick the checkbox on the rendered form. On the rendered form, to use the shortcut, press the combination of keys: Ctrl + the selected keyboard.



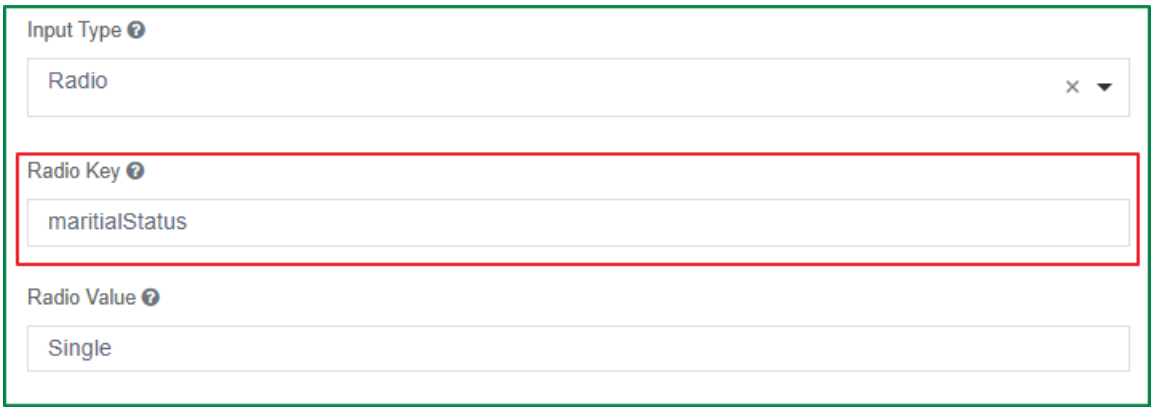
Input Type

This option is displayed in the **Display** tab. You can select the input type for the field, when you drag and drop the field to the form, the default Input Type is Checkbox, you can change it to Radio.



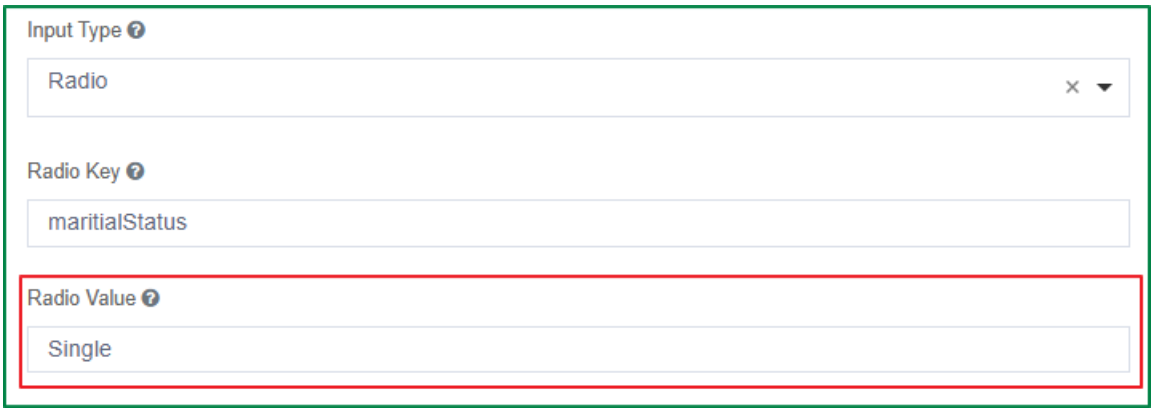
Radio Key

This option is displayed in the **Display** tab. The **Radio Key** setting is only displayed and enabled if the selected **Input Type** is **Radio**. You can input the key of the radio that will store data, e.g: maritialStatus.



Radio Value

This option is displayed in the **Display** tab. The **Radio Value** setting is only displayed and enabled if the selected **Input Type** is **Radio**. You can input the value of the radio key when the checkbox is ticked, e.g: Single.

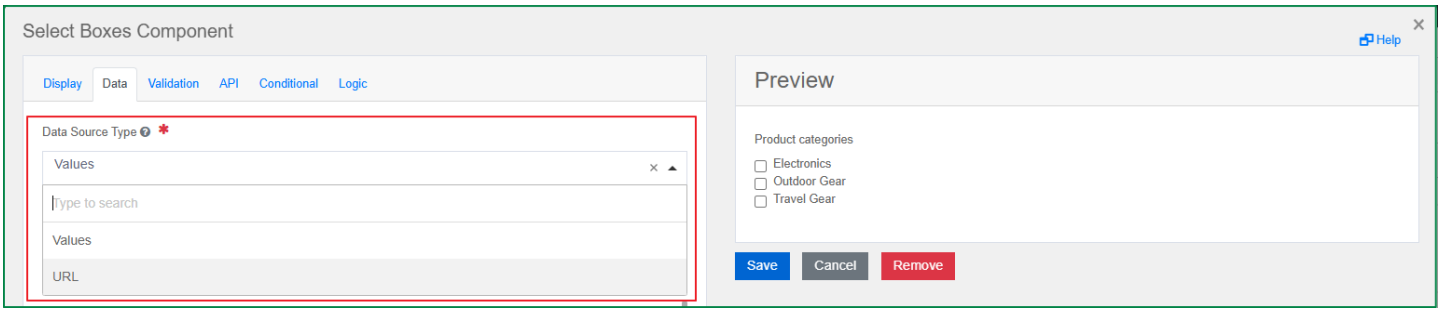


Select boxes

This component allows you to select one or multiple options in checkbox style format. The specific settings of this component include setting values and value shortcut, min/max validation of checked number and checked error messages.

Data Source Type

This option is displayed in the **Data** tab. This option is to define whether the type of data source to use for the component is **Values** or **URL**. The additional setting options will be displayed according to the selected data source type.



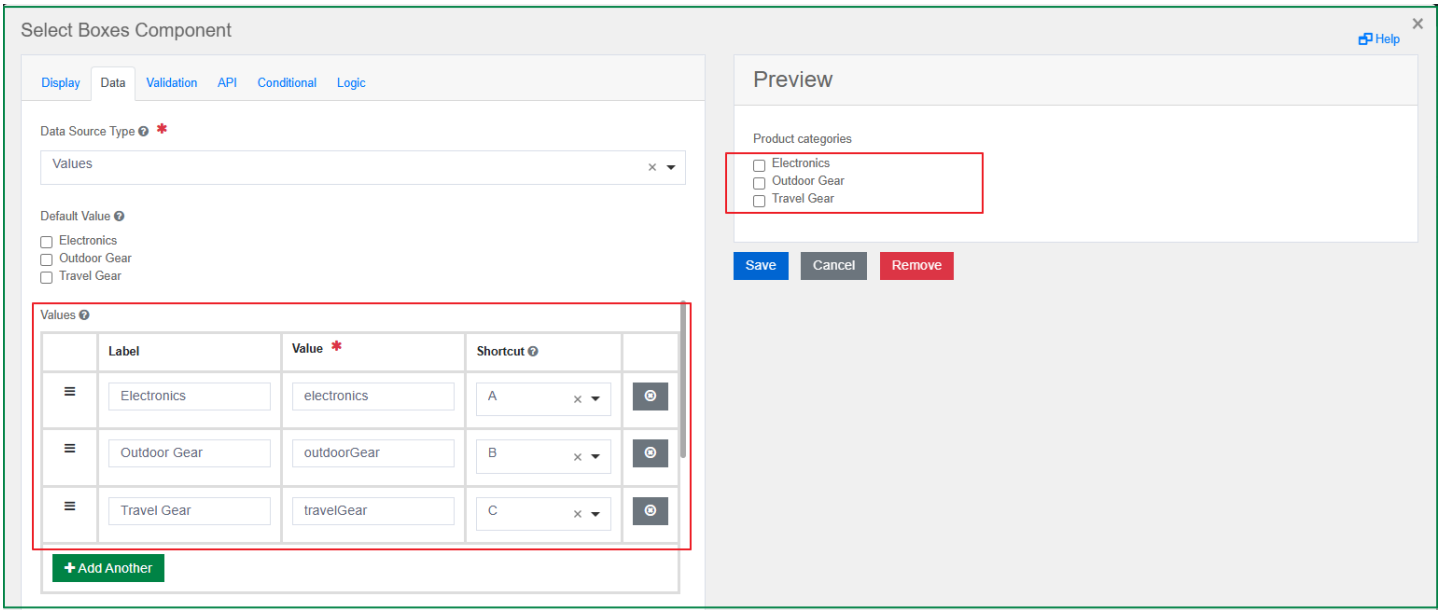
Values

This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **Values**. You can define the values list of the field, each value will be displayed as a checkbox in the checkboxes list. There are 3 related settings for the value, including: **Label**, **Value**, **Shortcut**.

Label is the front-end identifier for the end-user.

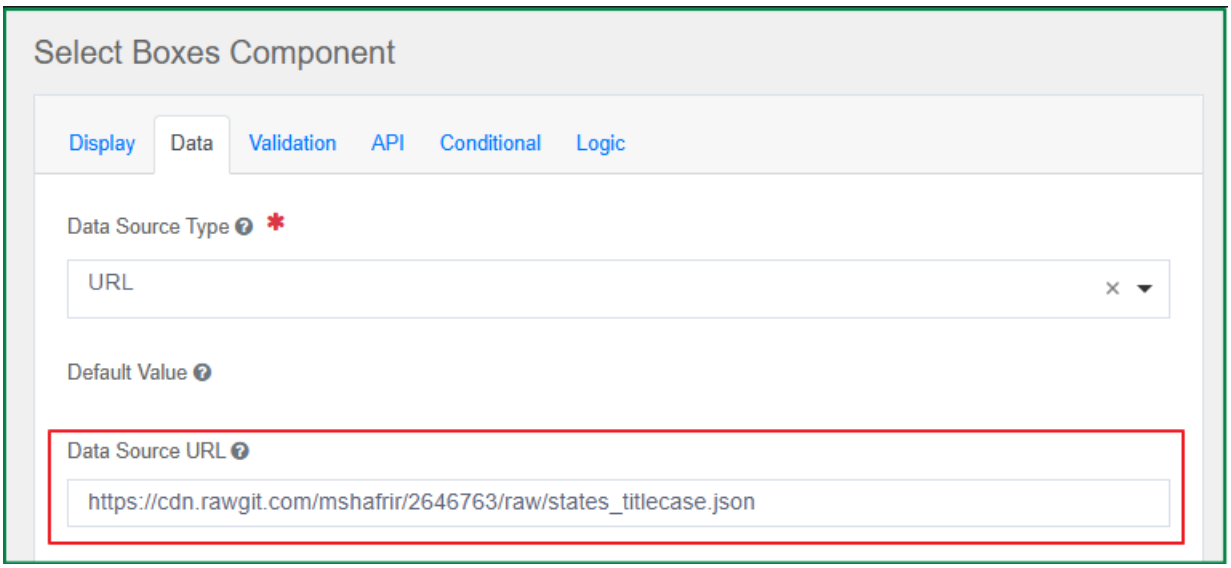
Value is back-end identifier for the option and is what is saved to the database. Only Strings or Numeric values should be set for the **Value**.

Shortcut is the keyboard to tick or untick the corresponding checkbox of the value on the rendered form. To use the shortcut, press the combination of keys: Ctrl + the selected keyboard.



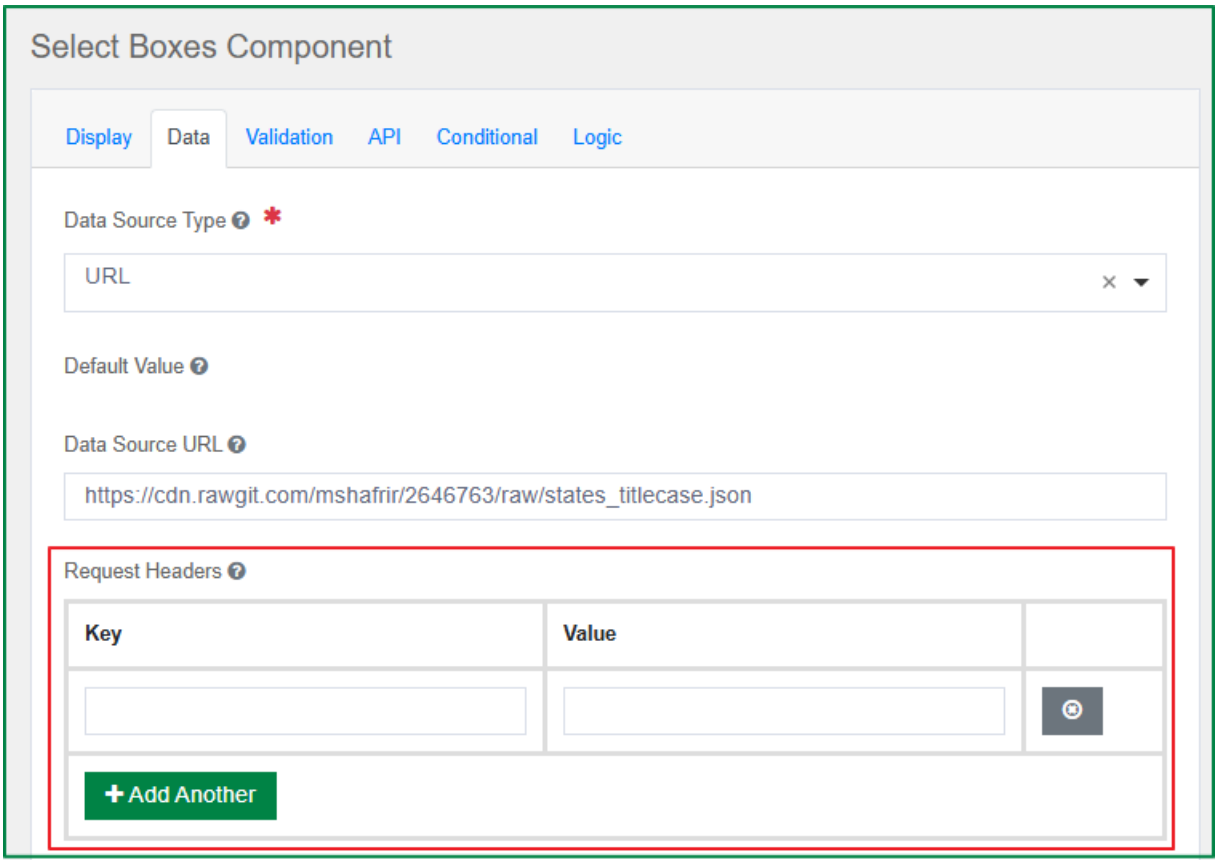
Data Source URL

This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **URL**. You can input the URL that returns a JSON array to use as data source.



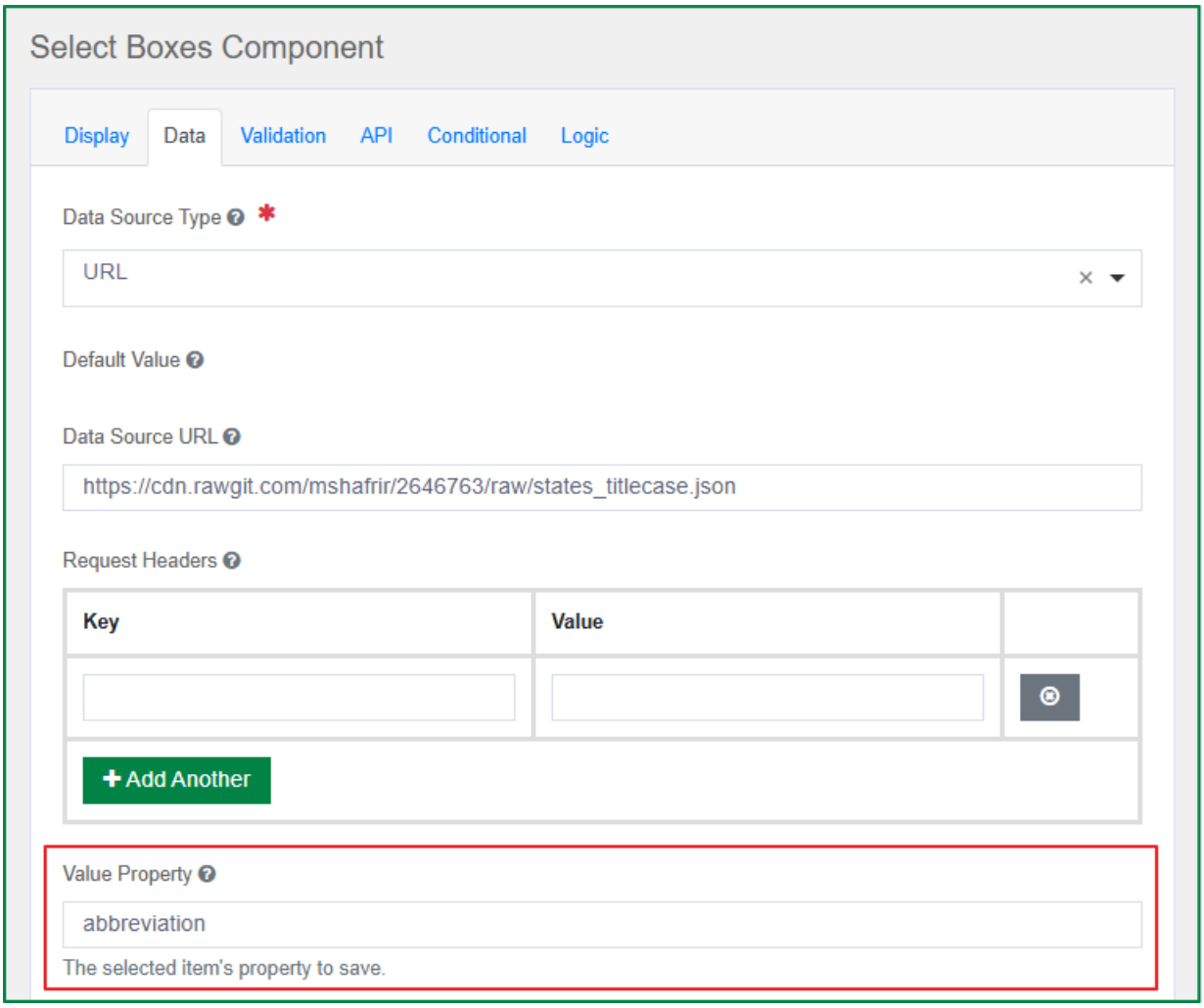
Request Headers

This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **URL**. You can set the headers that should be sent along with the request to the url. This setting is useful for authentication.



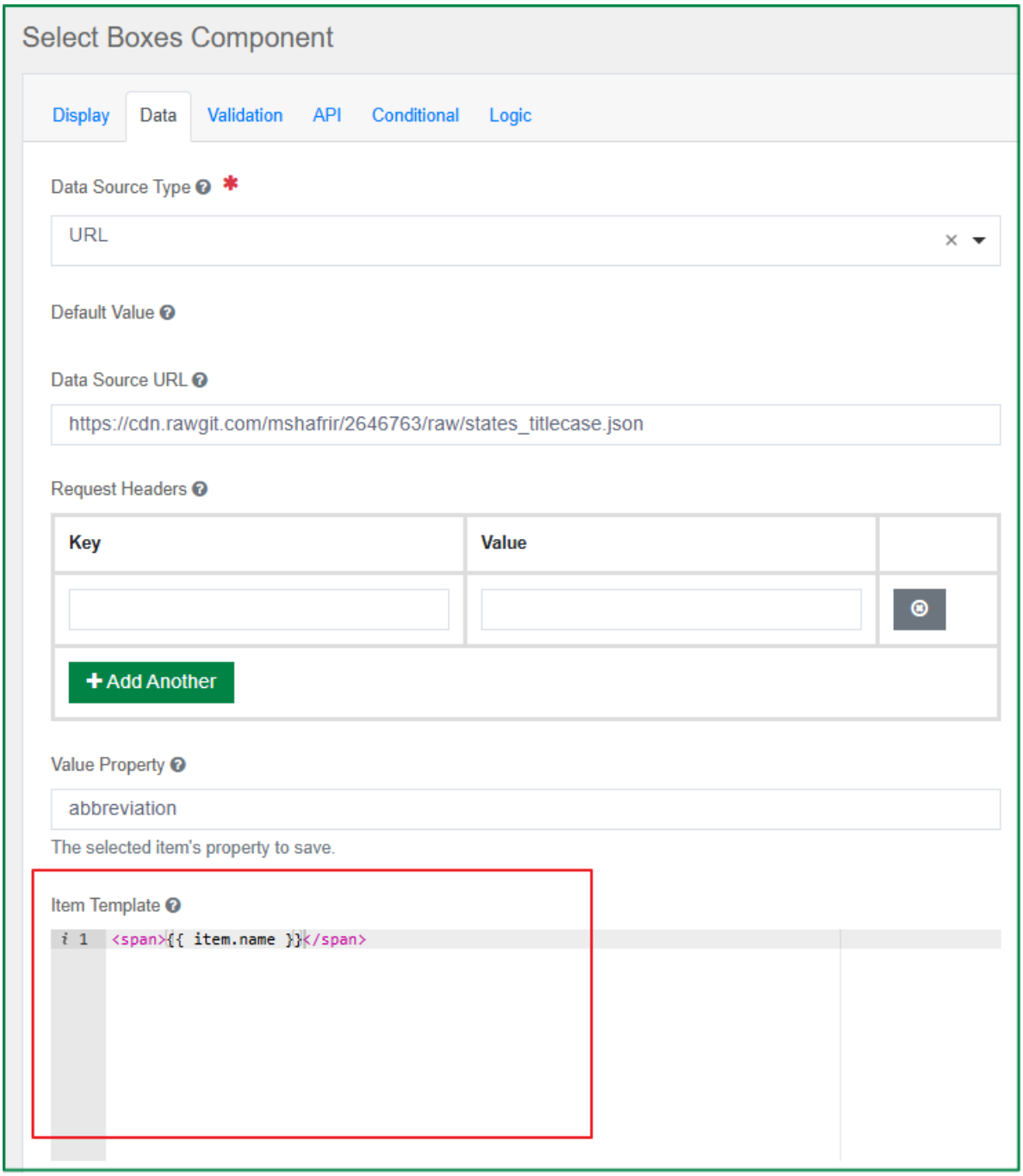
Value Property

This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **URL**. You can specify the value that corresponds to the desired property within each object which is set in the Item Template. If the value is not specified, the item itself will be used.



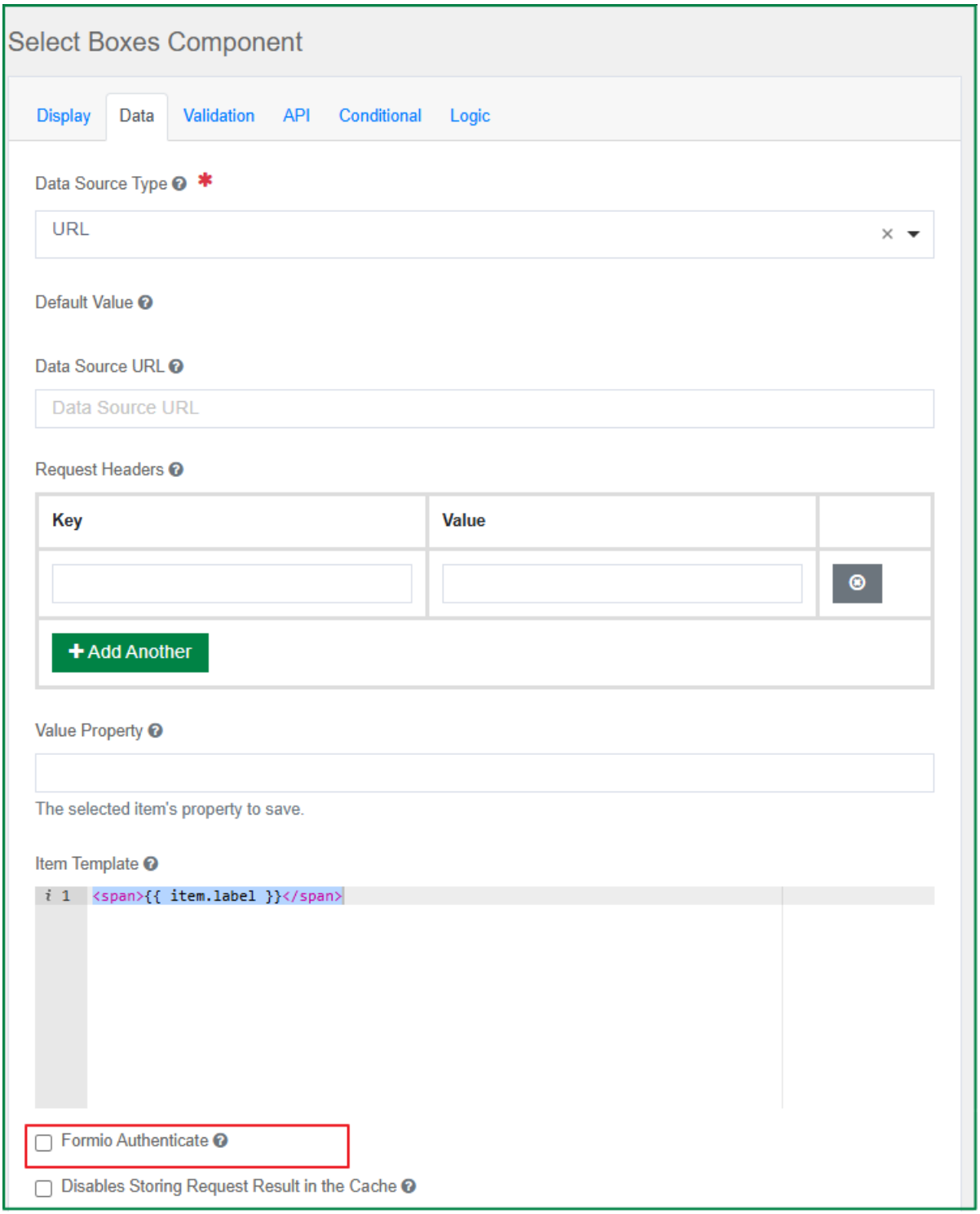
Item Template

This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **URL**. You can input the HTML template for the result data items.



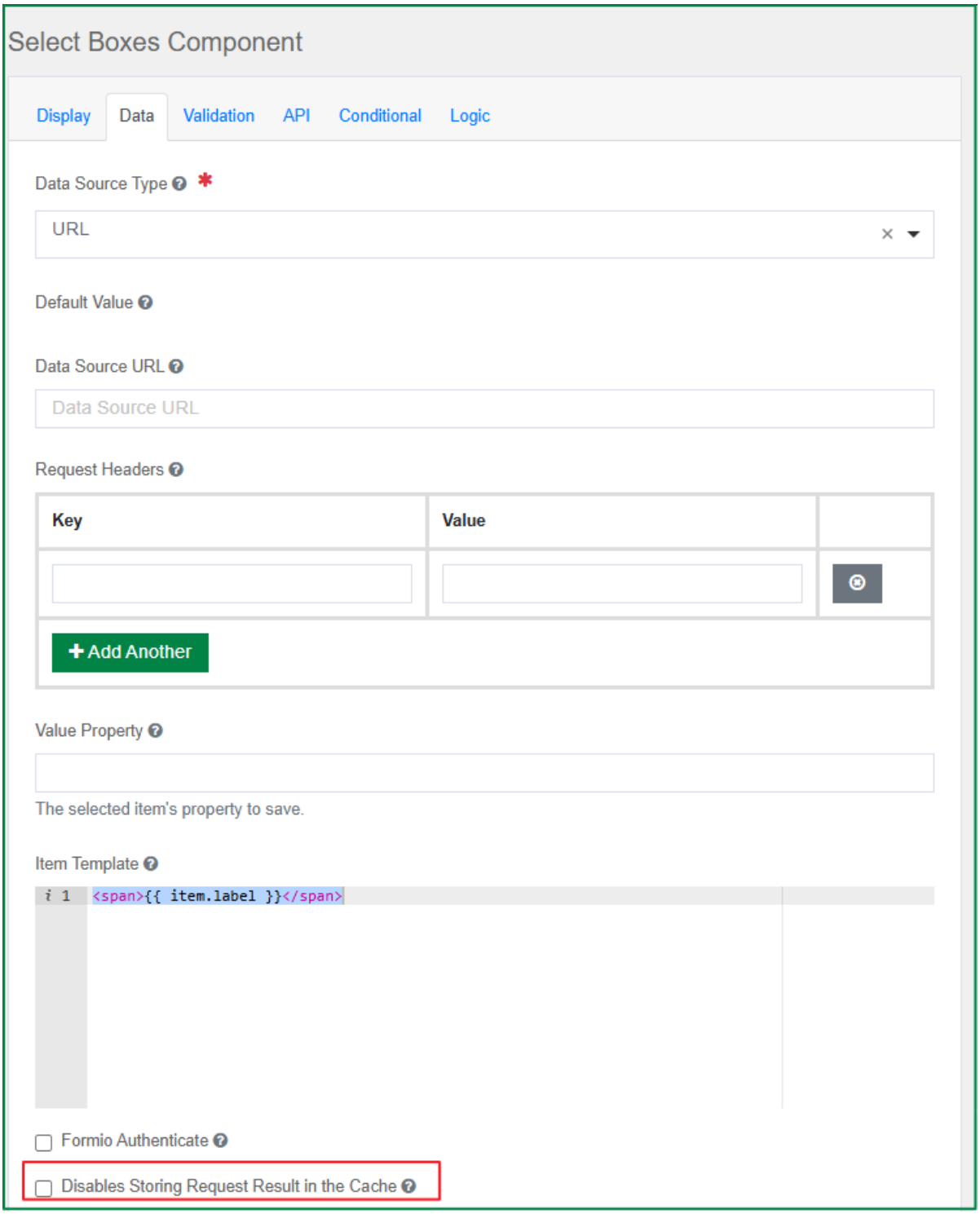
Formio Authenticate

This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **URL**. Tick this checkbox if you want to use Formio Authentication with the request.



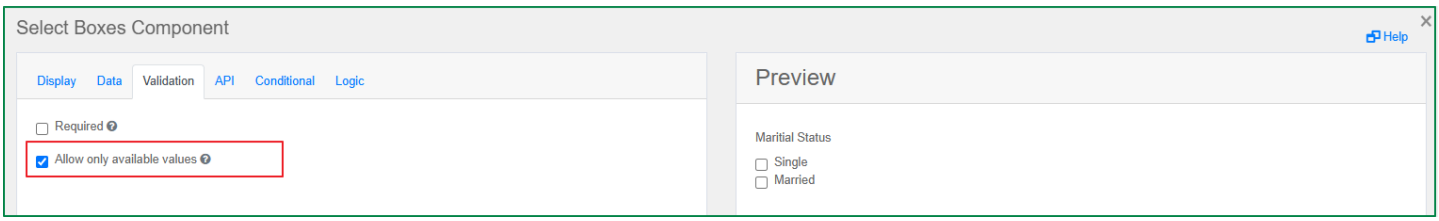
Disables Storing Request Result in the Cache

This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **URL**. Tick this checkbox if you don't want the requests and its results to be stored in the cache.

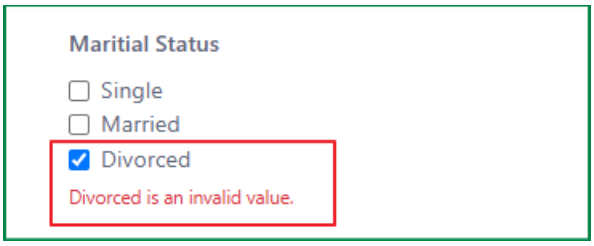


Allow only available values

This option is displayed in the **Validation** tab. Tick this checkbox if you want the selected value to be validated whether it is available before the submission.

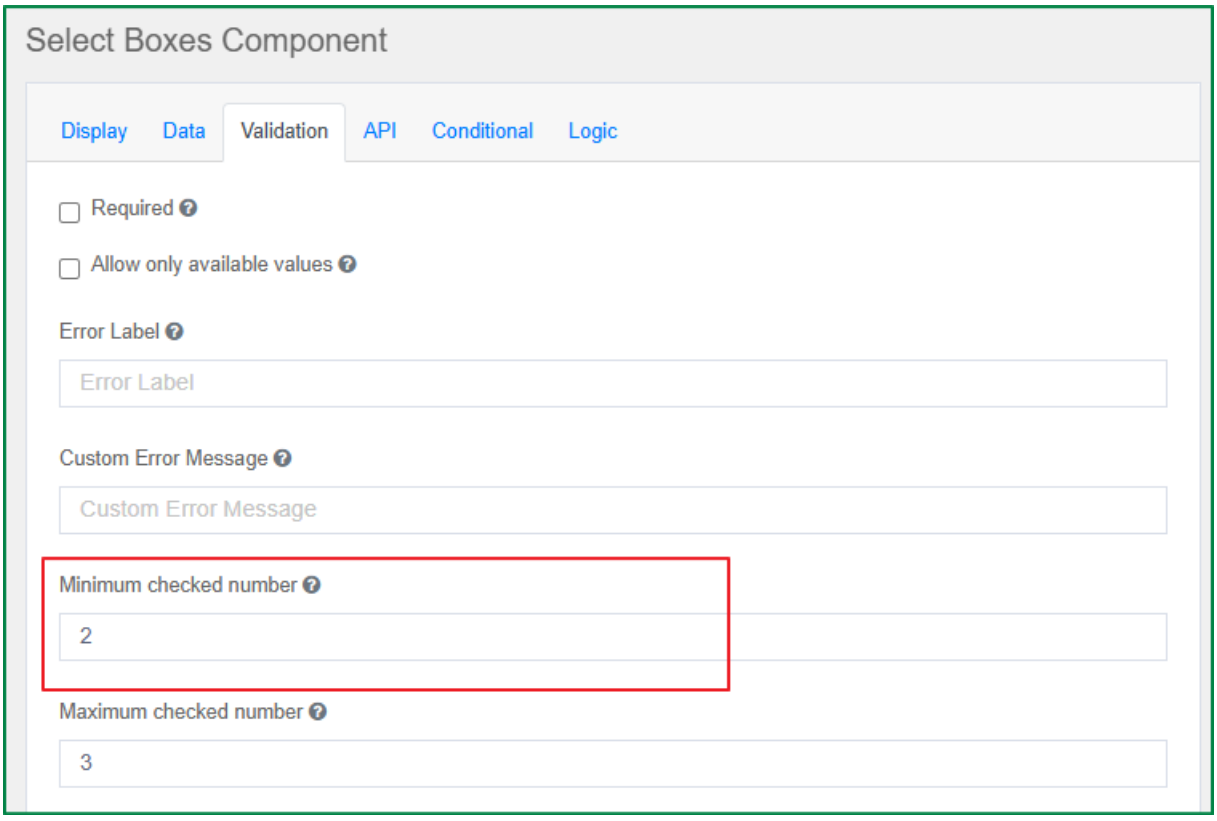


For example, the end-user opens the previous version of the registration form which includes a value of the select box component. This value is no more available in the latest version of the form. When the user selects the unavailable value and submits the form, the validation will be triggered, the submission will be prevented, and the error message of unavailable value will be displayed.

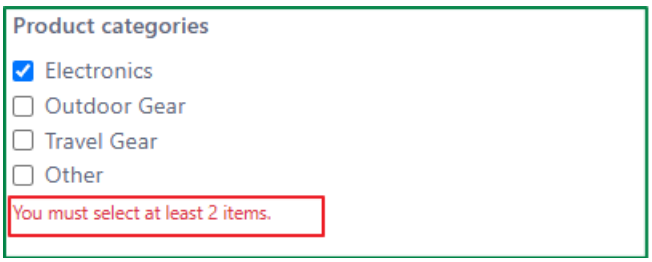


Minimum checked number

This option is displayed in the **Validation** tab. With this option, you can define the minimum number of must-be-ticked checkbox.

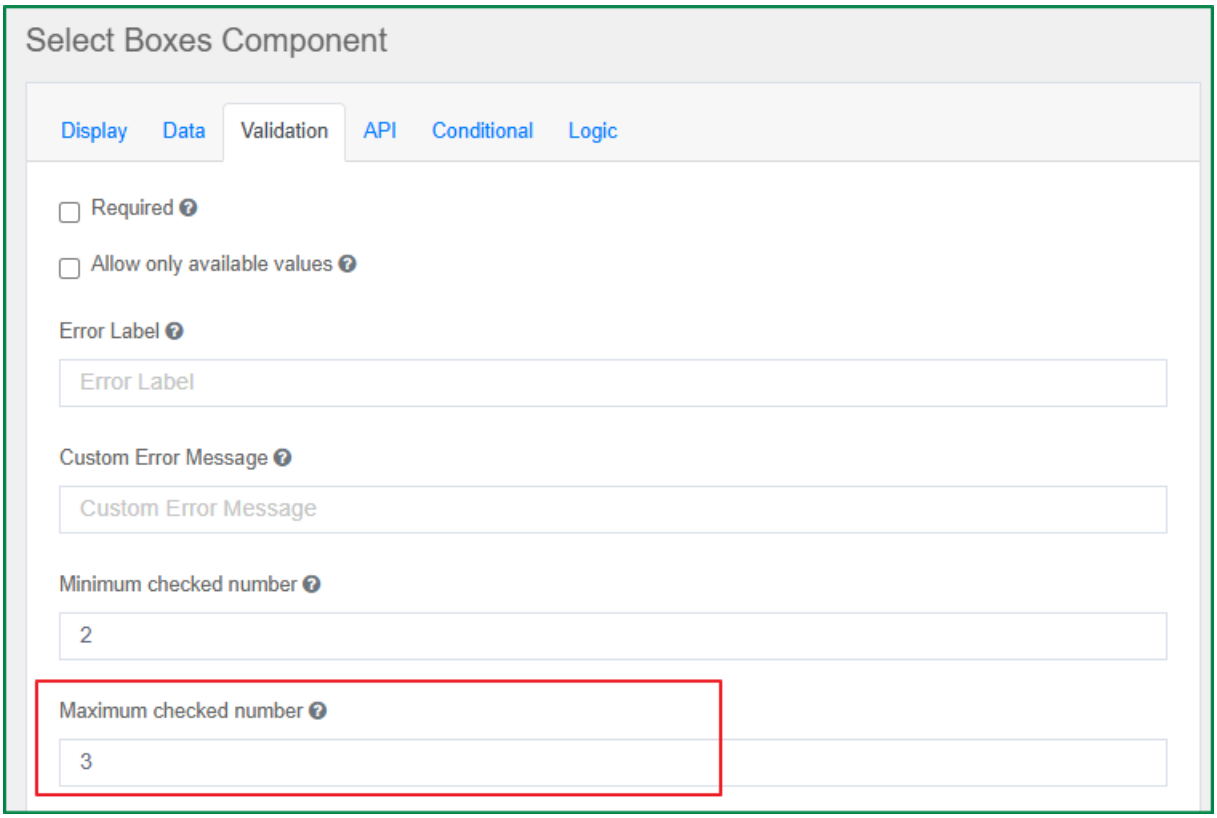


On the rendered form, if the amount of ticked checkbox is smaller than the defined minimum number, the validation will be triggered, and the message for the amount of must-be-ticked checkbox will be displayed, and the end-user cannot submit the form.



Maximum checked number

This option is displayed in the **Validation** tab. With this option, you can define the maximum number of tick-able checkbox.

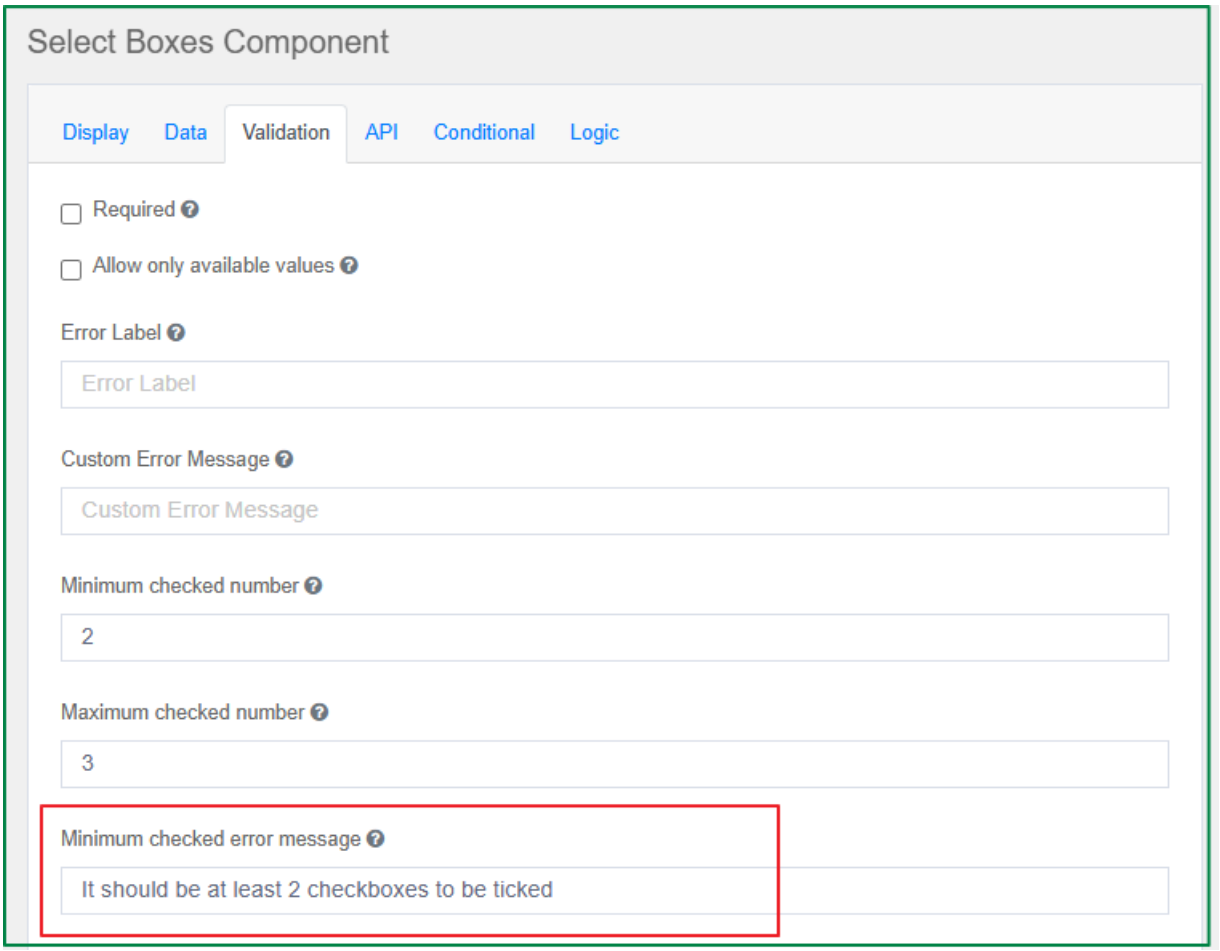


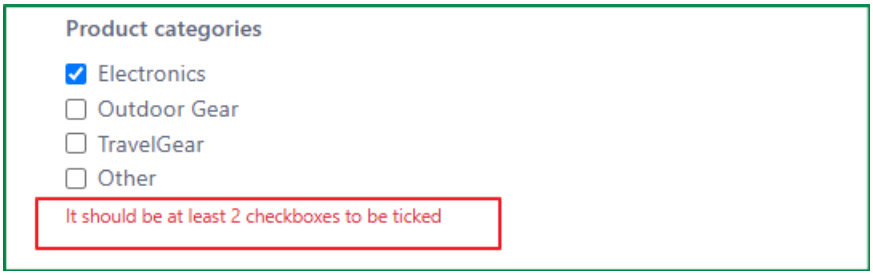
On the rendered form, if the amount of ticked checkbox is equal to the defined maximum number, the validation will be triggered, and the remaining unticked checkboxes will be disabled.



Minimum checked error message

This option is displayed in the **Validation** tab. With this option, you can define the validation message relating the above **Minimum checked number** setting. If this message is not defined, the default message will be shown.



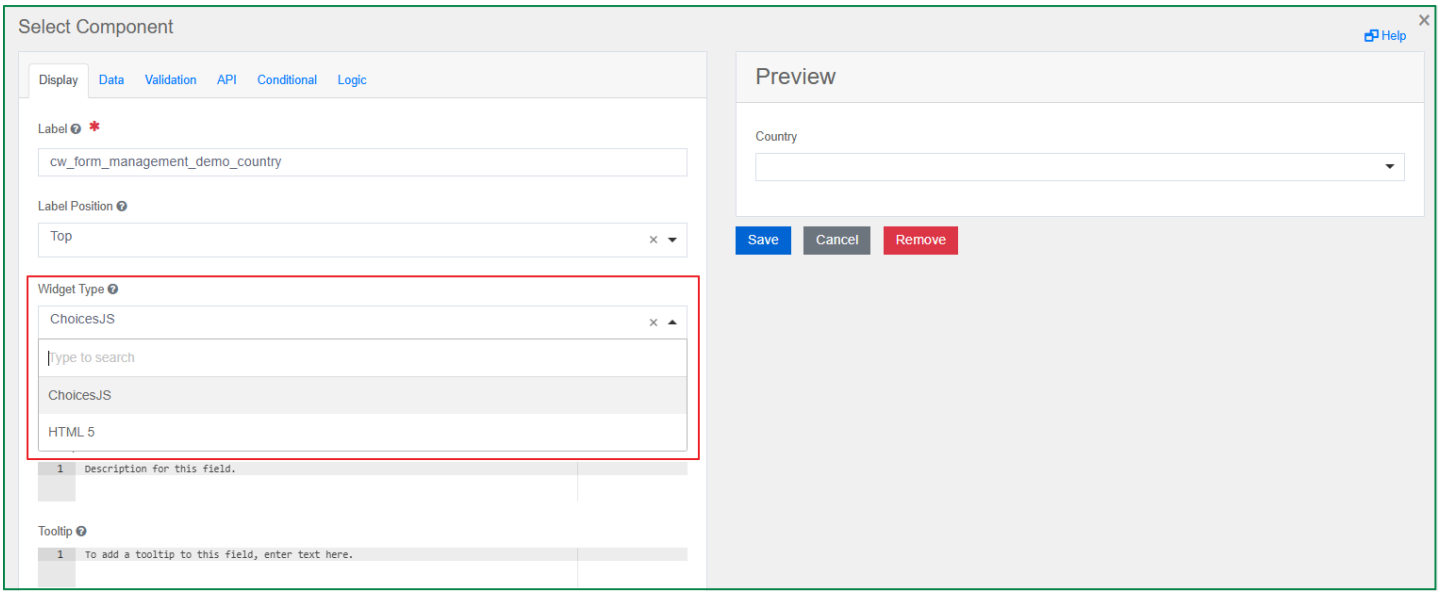


Select

The Select component displays a list of values in a dropdown list where users can select one or multiple the values. There is also a large offering of settings providing different ways of filtering, querying, and loading data values.

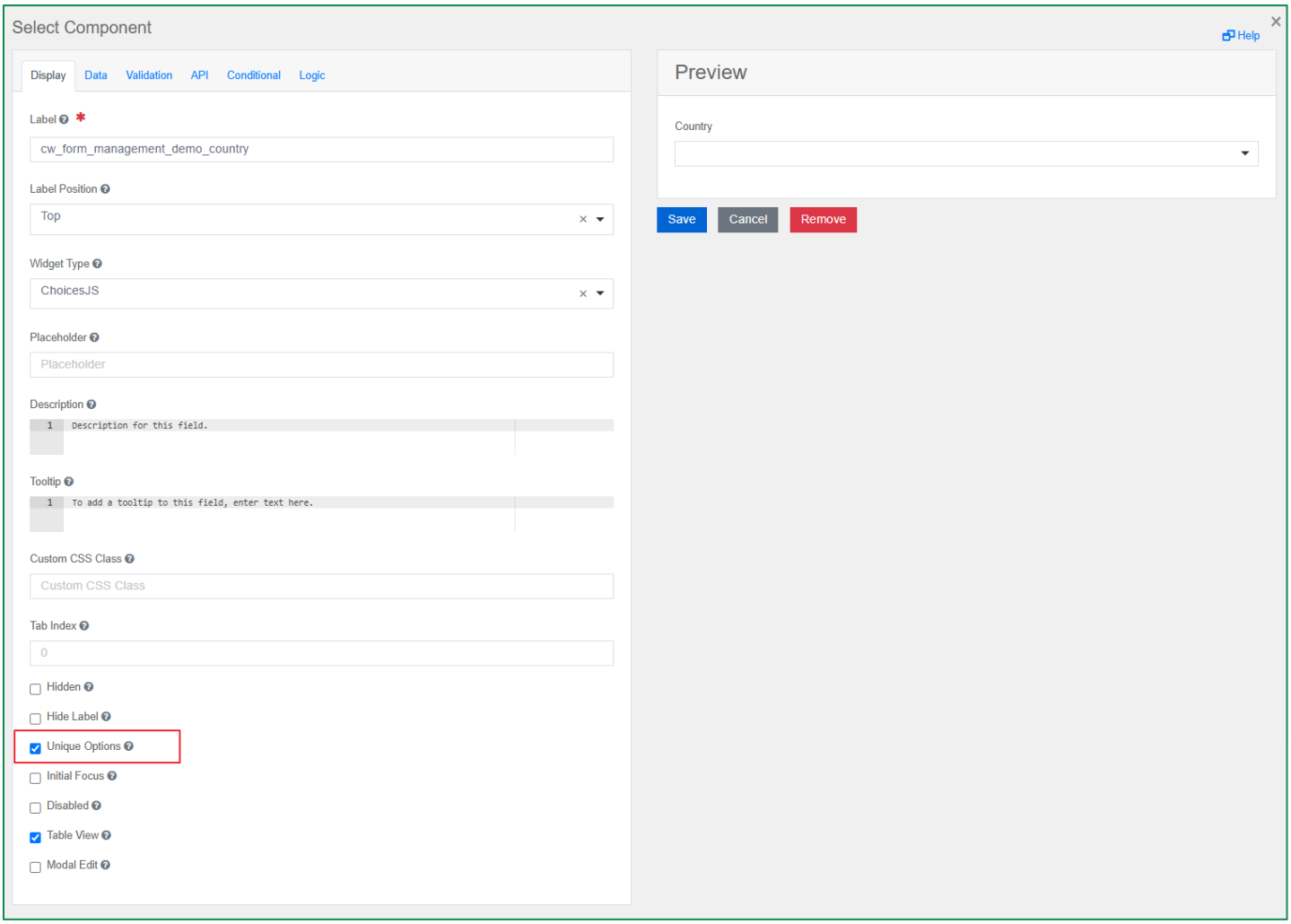
Widget Type

This option is displayed in the **Display** tab. You can select whether the widget type is **ChoiceJS** or **HTML 5**.



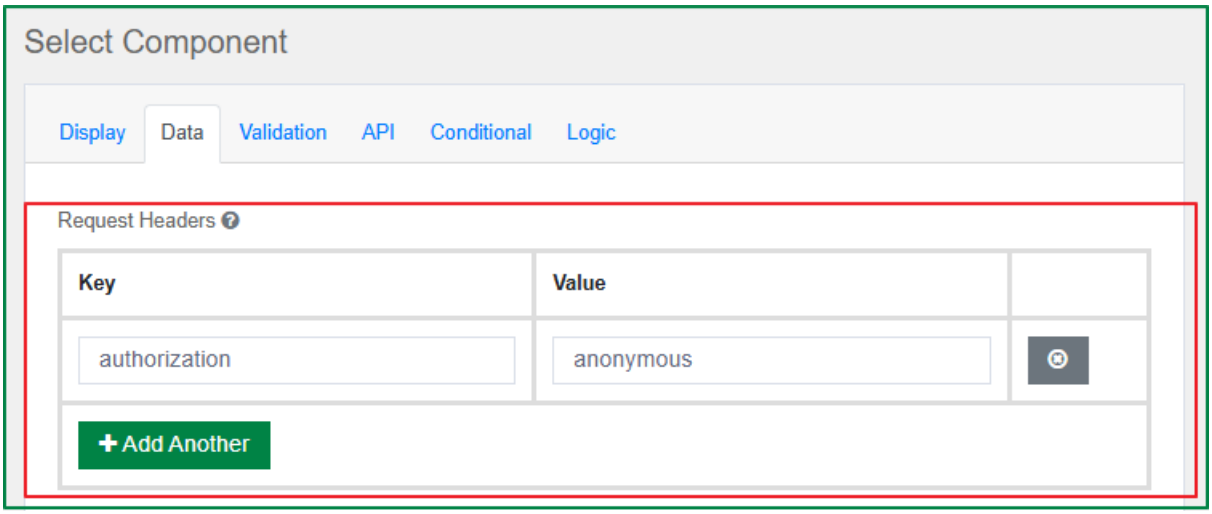
Unique Options

This option is displayed in the **Display** tab. Tick this checkbox if you want the dropdown list only show the unique options and do not display the duplicated ones.



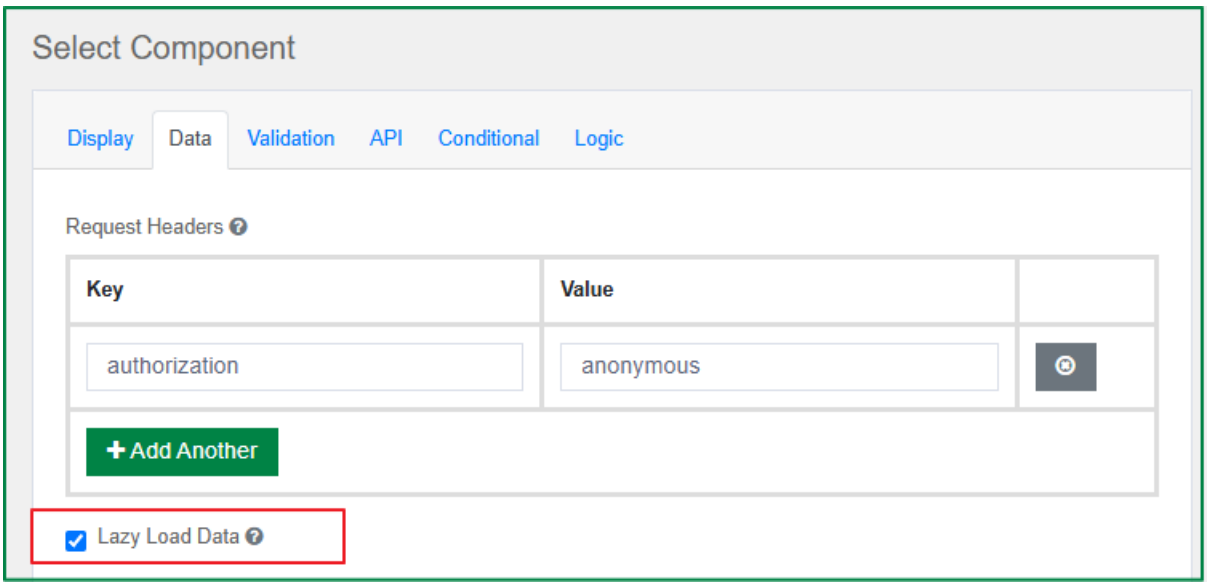
Request Headers

This option is displayed in the **Data** tab. You can set the headers that should be sent along with the request to the url. This setting is useful for authentication.



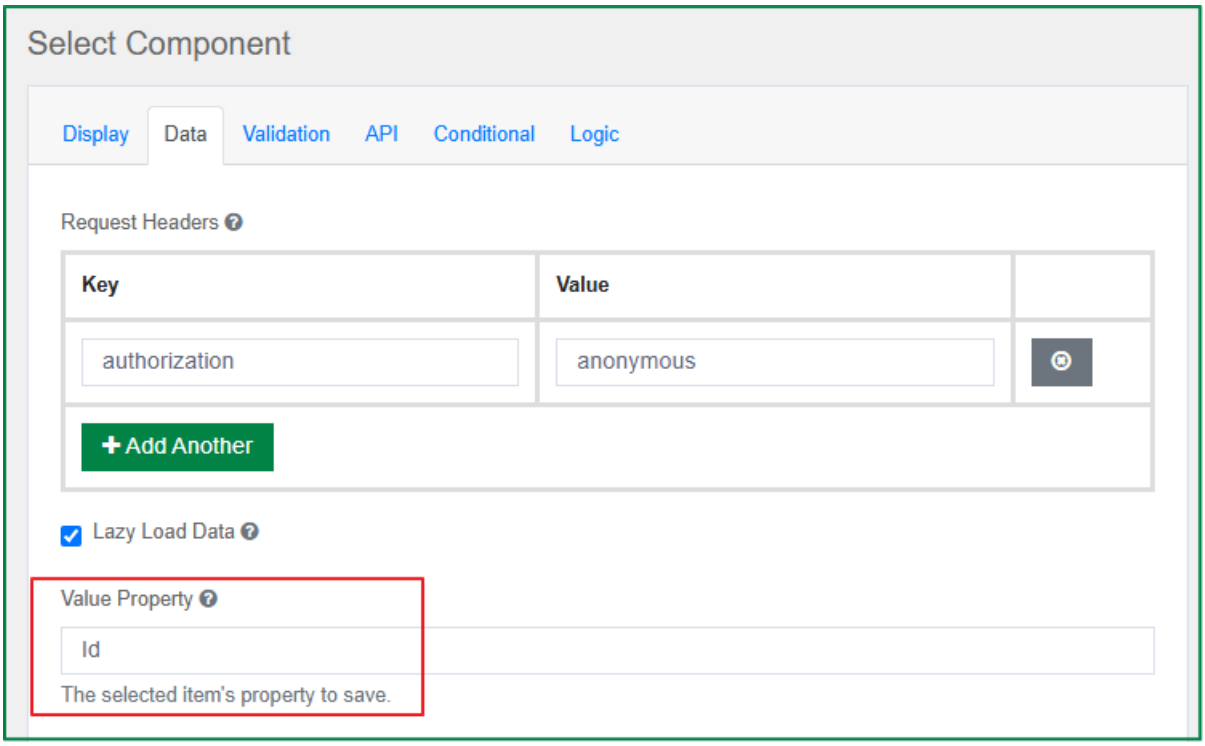
Lazy Load Data

This option is displayed in the **Data** tab. Tick this option to improve the performance if you have many Select dropdowns on your form where the APIs will only fire when the field is activated.



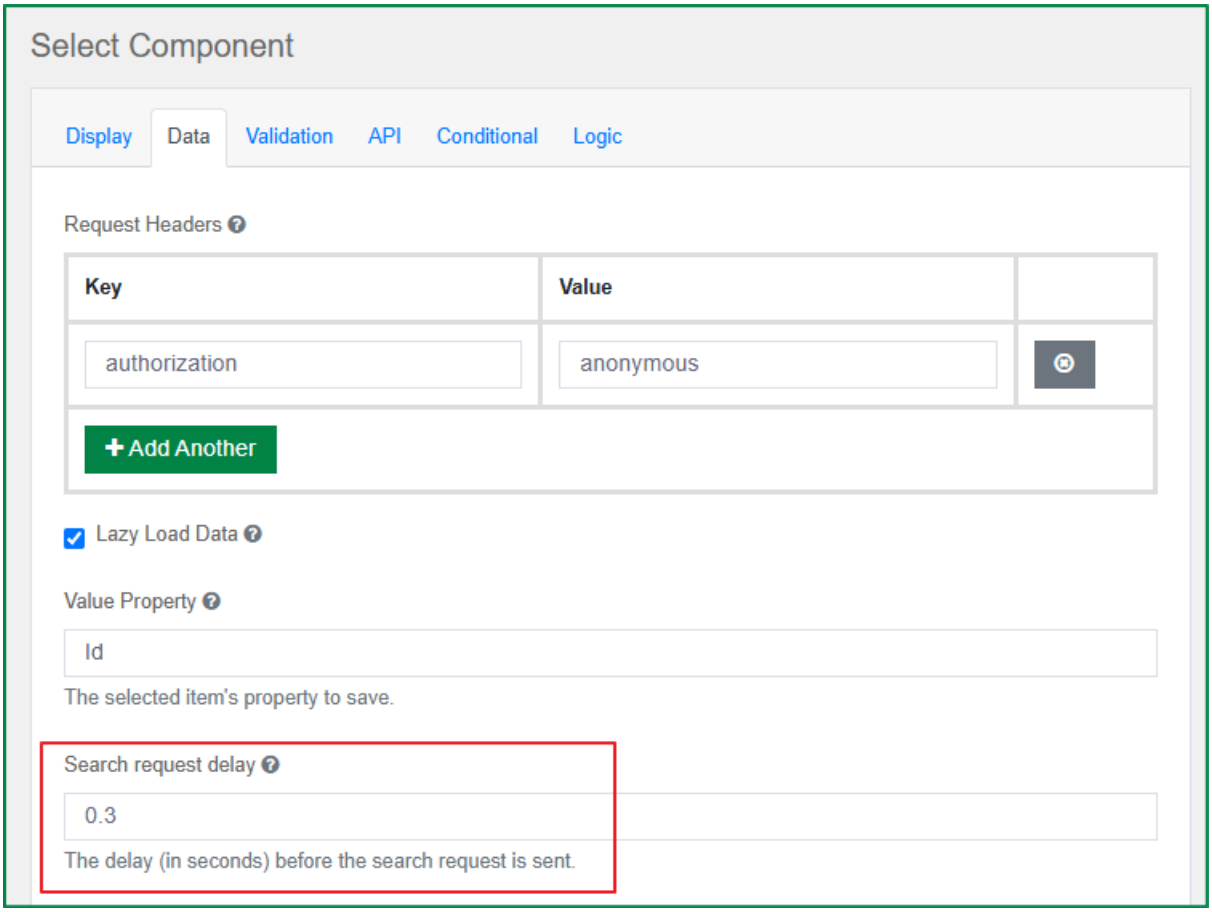
Value Property

This option is displayed in the **Data** tab. You can specify the property of each item in the data source to use as the select value. If the value is not specified, the item itself will be used.



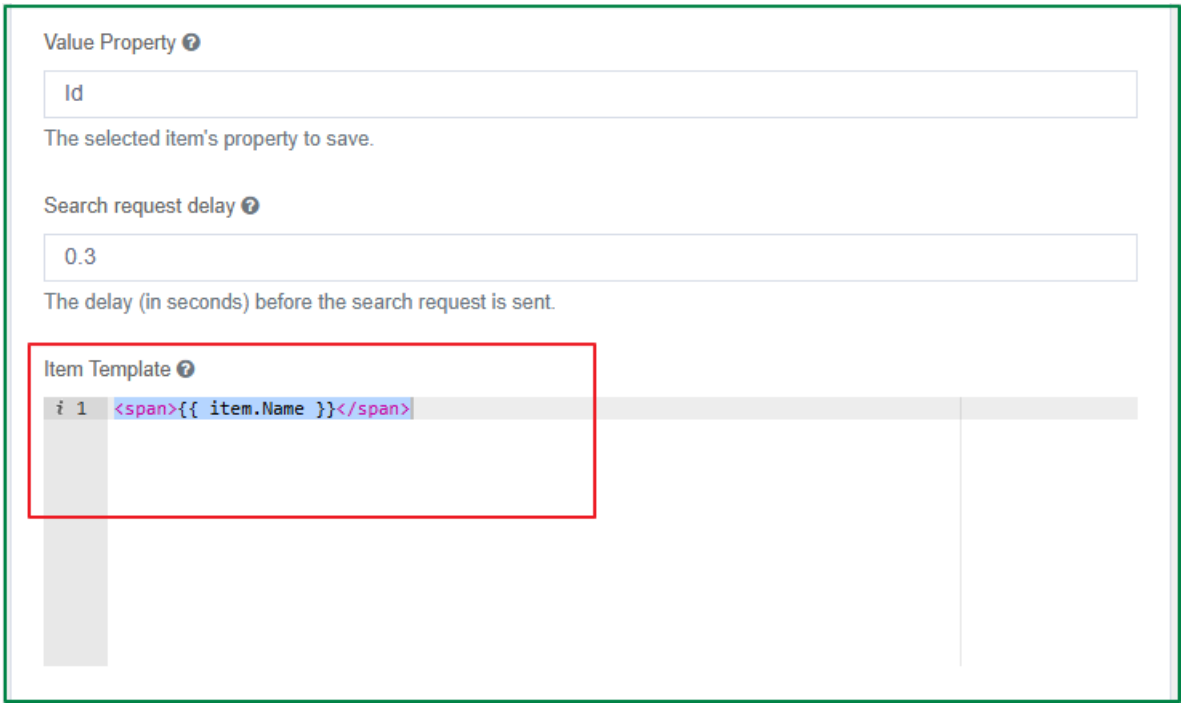
Search request delay

This option is displayed in the **Data** tab. You can input the delay in seconds before the search request is sent, measured from the last character input in the search field. The default is set to 0.3.



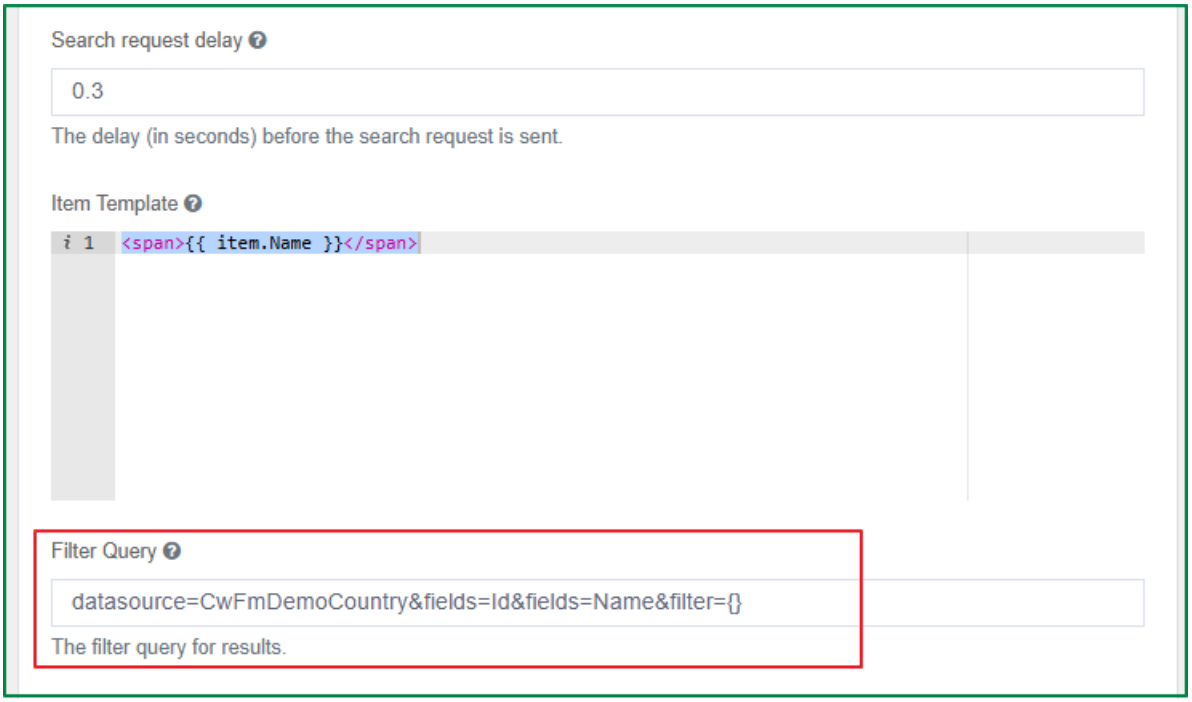
Item Template

This option is displayed in the **Data** tab. You can input HTML template that determines how the select options are displayed within the dropdown. You can use the item variable to access the current object in the array.



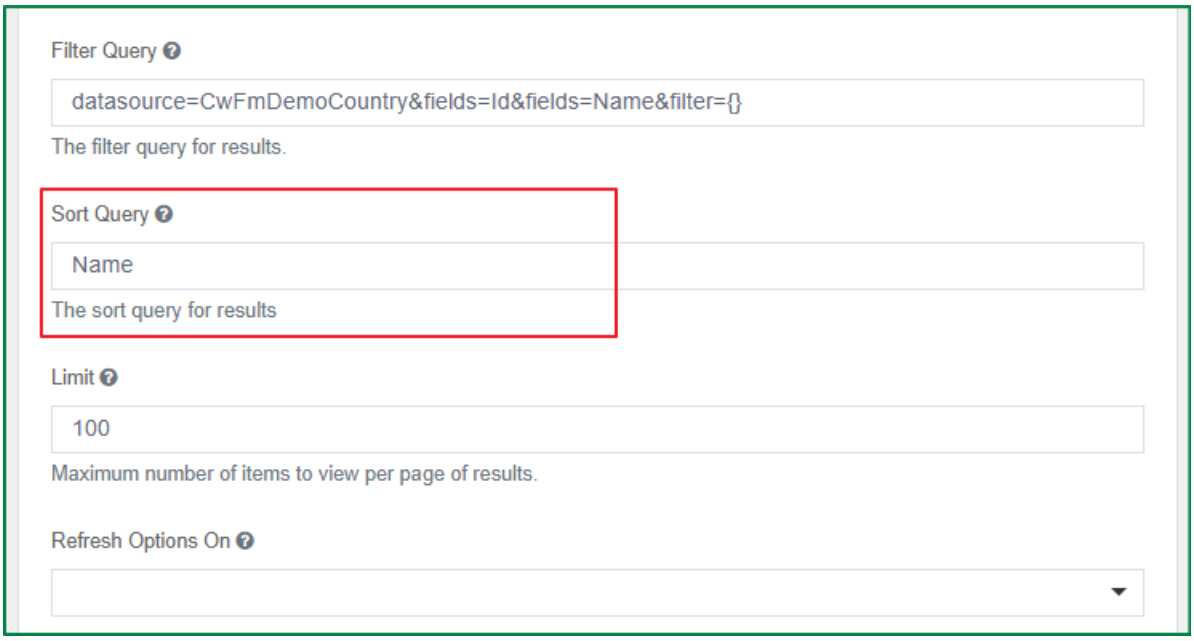
Filter Query

This option is displayed in the **Data** tab. You can use this to provide the additional filtering within the dropdown using query parameters.



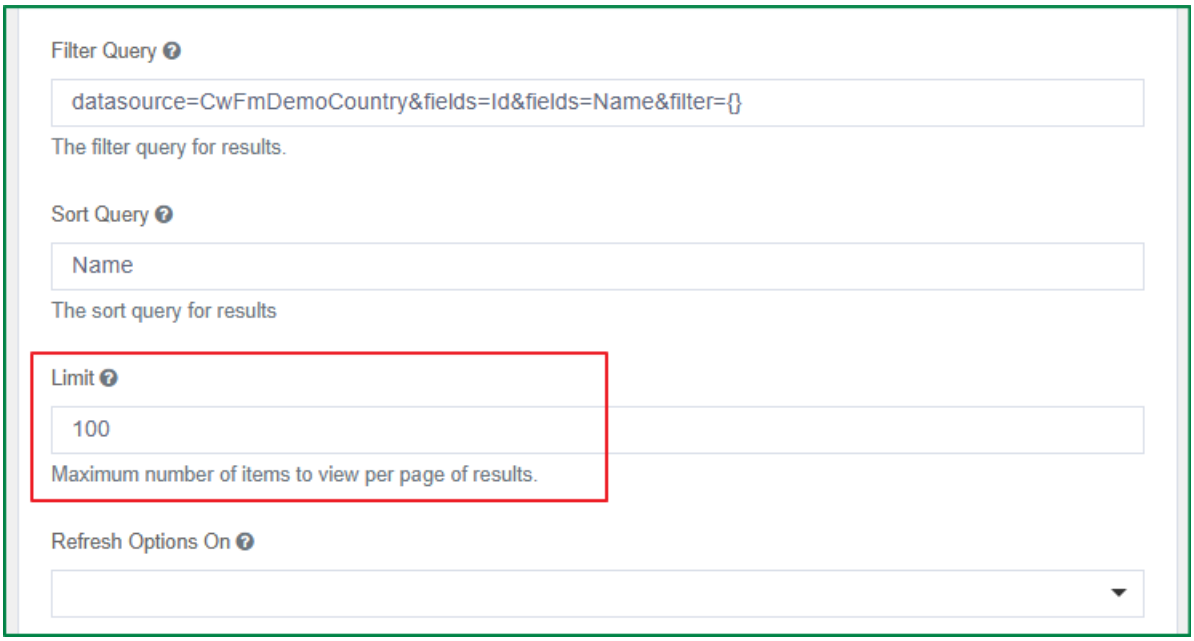
Sort Query

This option is displayed in the **Data** tab. You can use this to provide the additional sorting using query parameters.



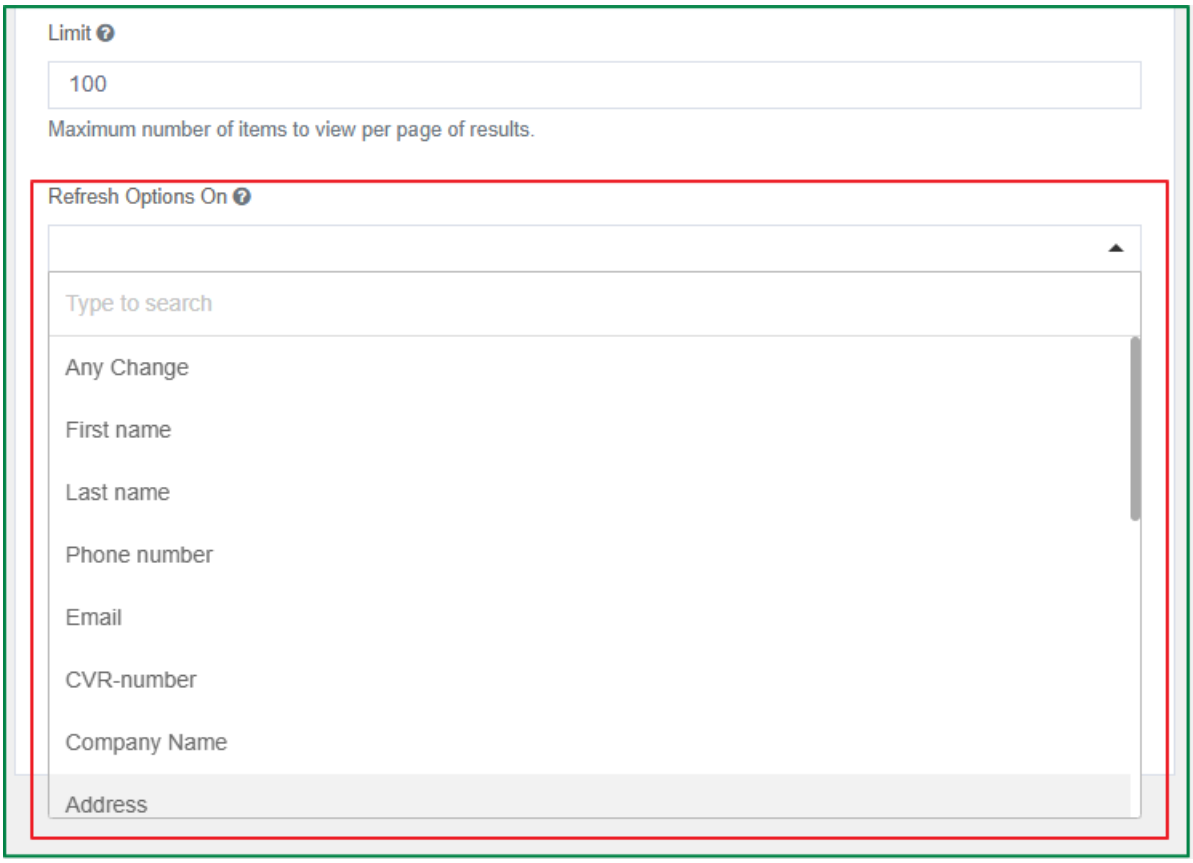
Limit

This option is displayed in the **Data** tab. You can input the limit of the number of items to request or view within the select field dropdown.



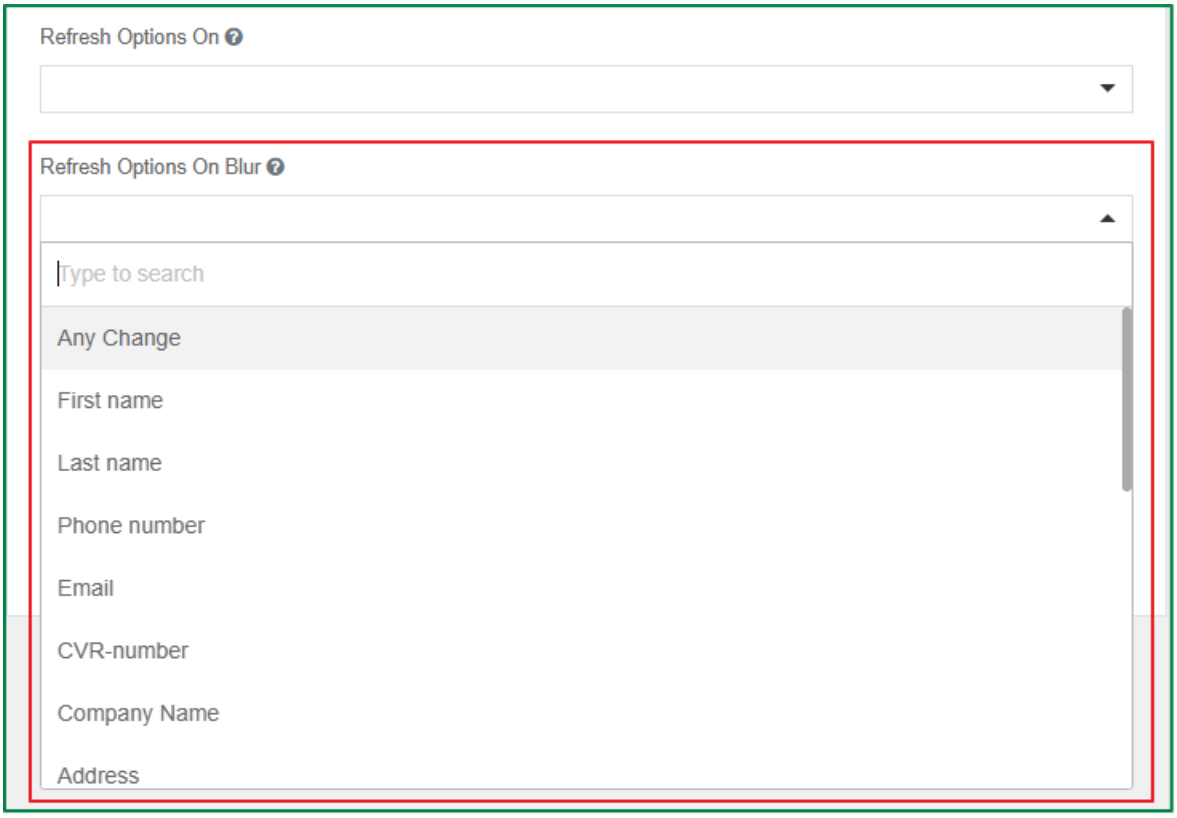
Refresh Options On

This option is displayed in the **Data** tab. You can select the field that if its value is changed, the options of the **Select** field will be refreshed. You can also select **Any Change**, so that, if it is any change on the form, the options of the **Select** field will be refreshed. This option is helpful when dealing with dynamic values.



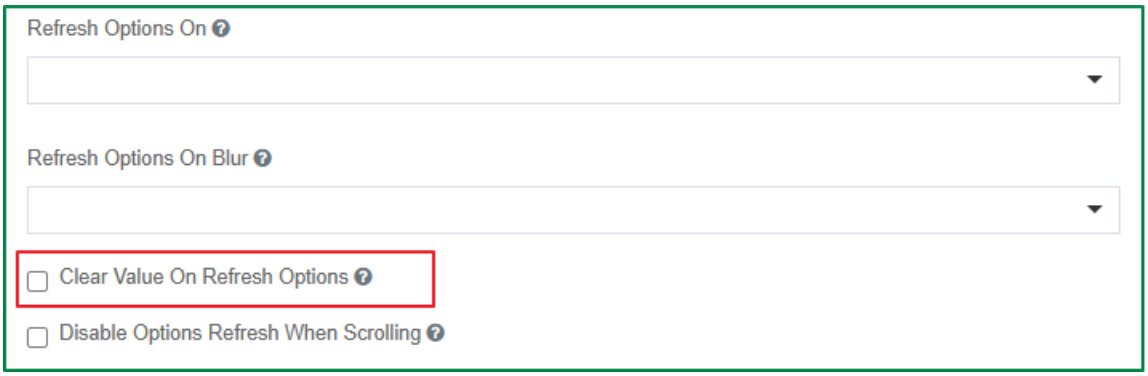
Refresh Options On Blur

This option is displayed in the **Data** tab. You can select the field that if it is blurred, the options of the **Select** field will be refreshed. You can also select **Any Change**, so that, if it is any field on the form is blurred, the options of the **Select** field will be refreshed.



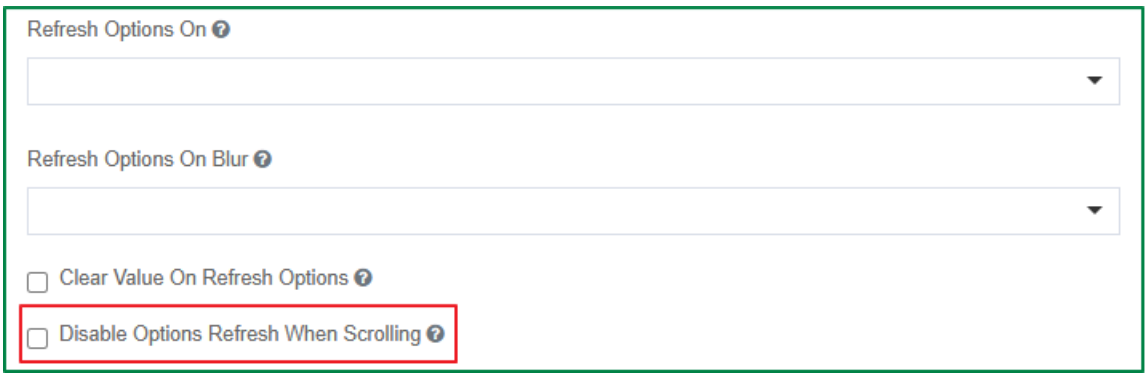
Clear Value On Refresh Options

This option is displayed in the **Data** tab. Tick this checkbox if you want to clear any selected value of the **Select** field when the field is reset by the **Refresh Options On** setting.



Disable Options Refresh When Scrolling

This option is displayed in the **Data** tab. Tick this checkbox if you want the options of the Select field will not be refreshed when end-user scrolls through the list of options.

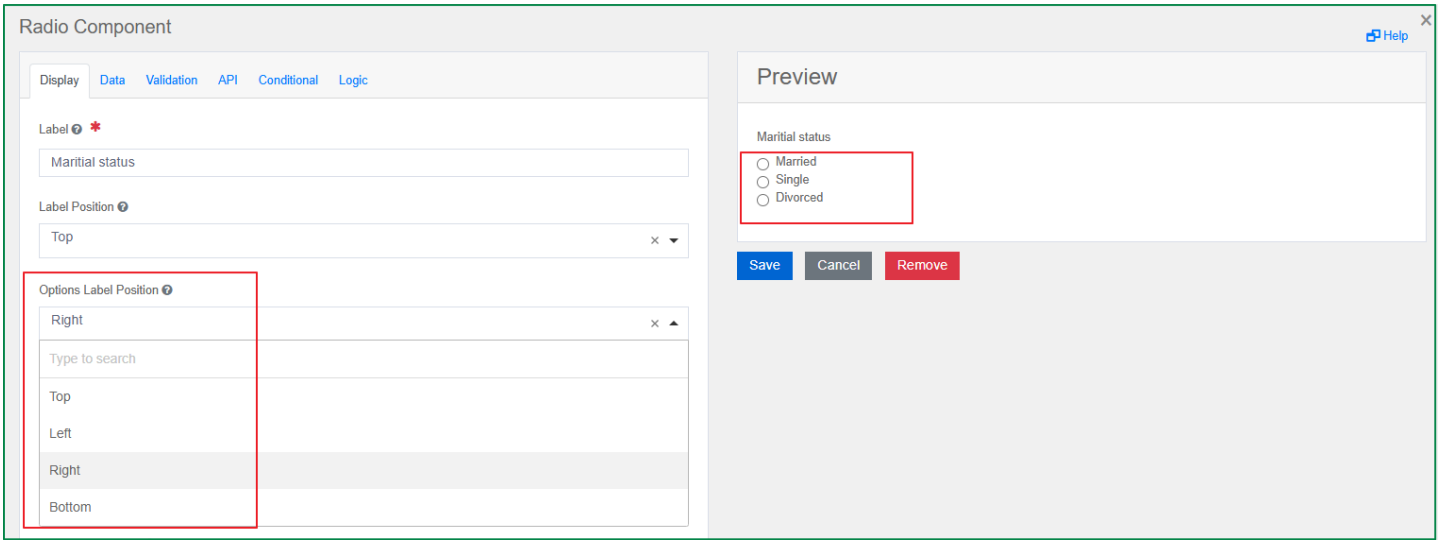


Radio

The radio component allows users to select a single option from a list of options displayed in radio-style format. Users can click a selected option to deselect it, returning the field back to its original state. Following is the specific settings of the radio component.

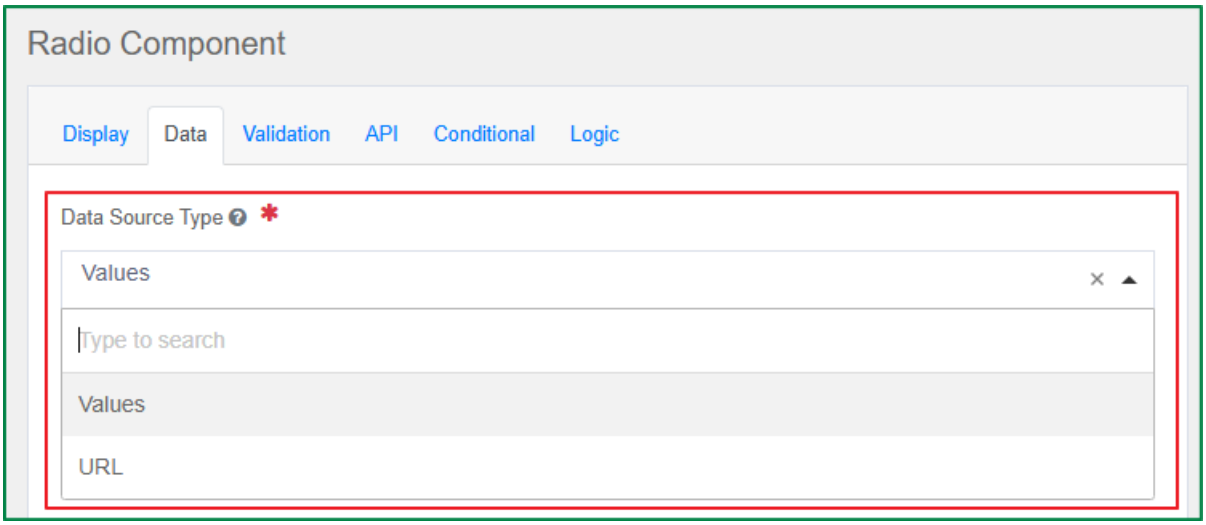
Options Label Position

This option is displayed in the **Display** tab. You can select the position of the option label relative to the option bubble.



Data Source Type

This option is displayed in the **Data** tab. You can select the source to use for the select data. There are 2 data source types, including **Values** and **URL**. The additional setting options will be displayed according to the selected data source type.



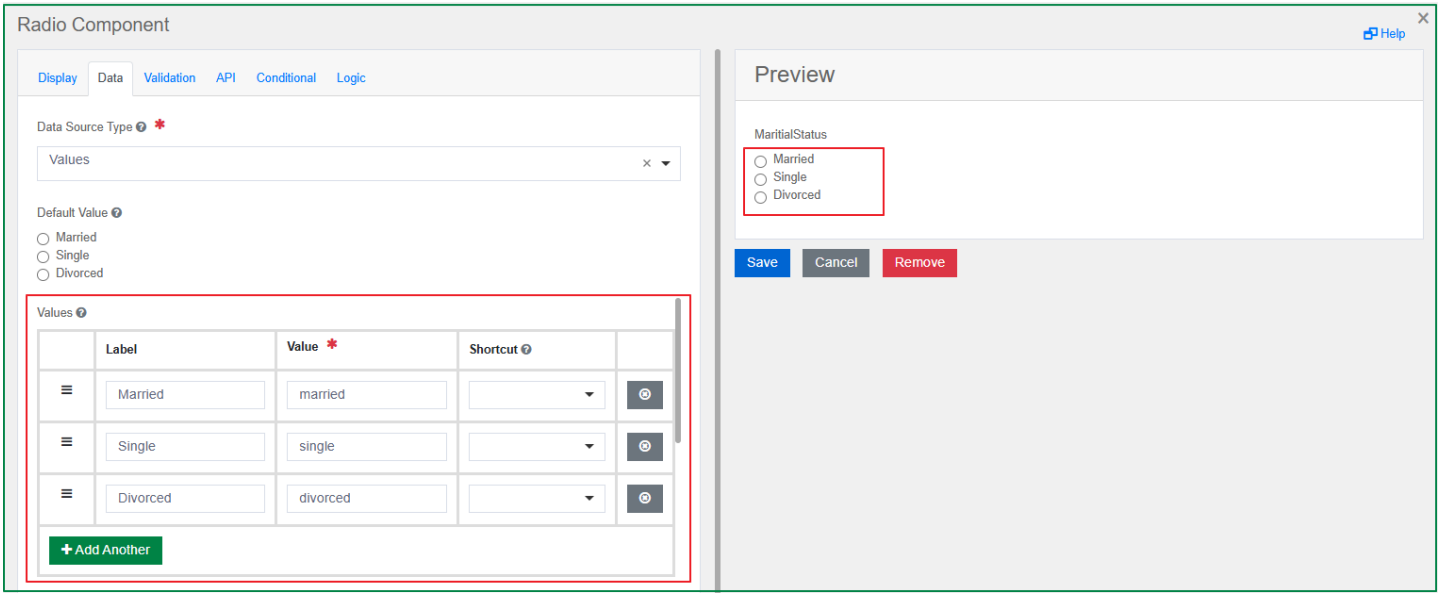
Values

This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **Values**. You can define the list of radio options, each value will be displayed as a radio option. There are 3 related settings for the value, including: **Label**, **Value**, **Shortcut**.

Label is the front-end identifier for the end-user.

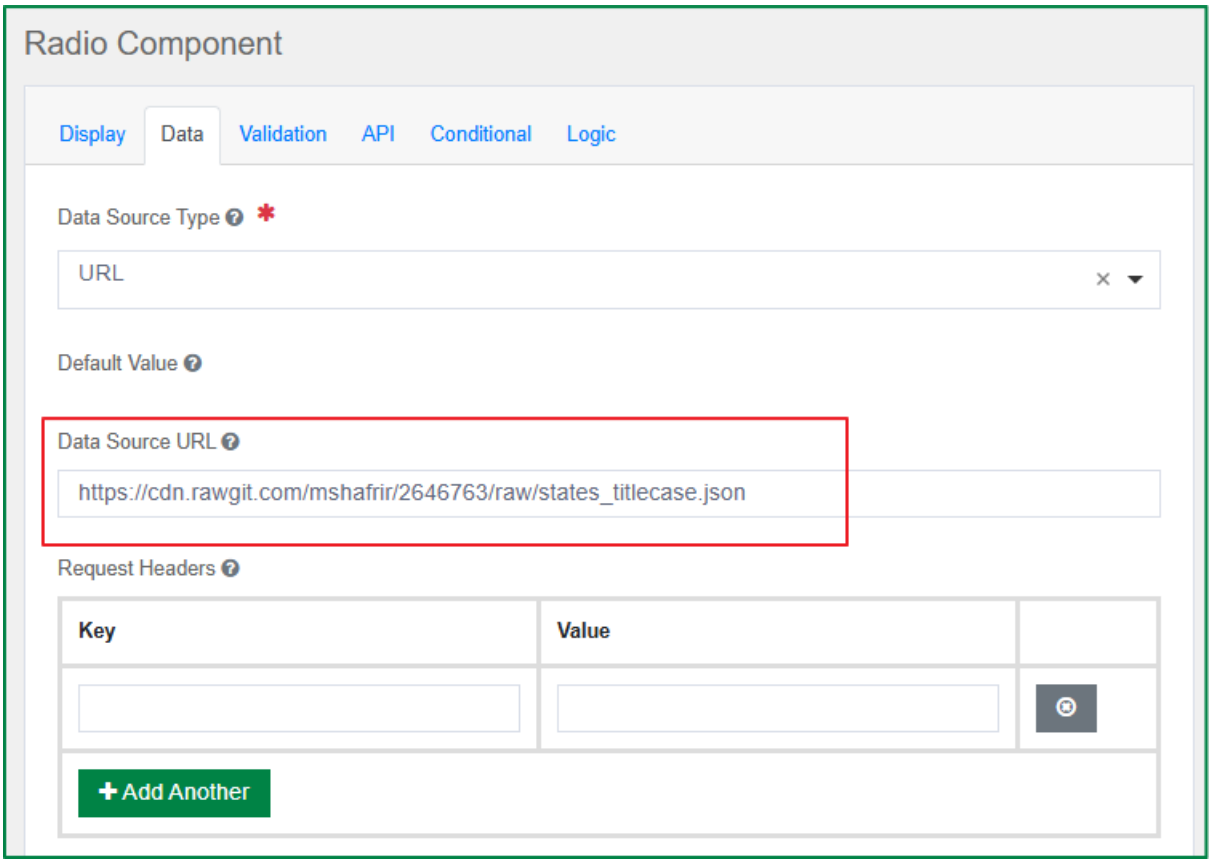
Value is back-end identifier for the option and is what is saved to the database. Only Strings or Numeric values should be set for the **Value**.

Shortcut is the keyboard to tick or untick the corresponding checkbox of the value on the rendered form. To use the shortcut, press the combination of keys: Ctrl + the selected keyboard.



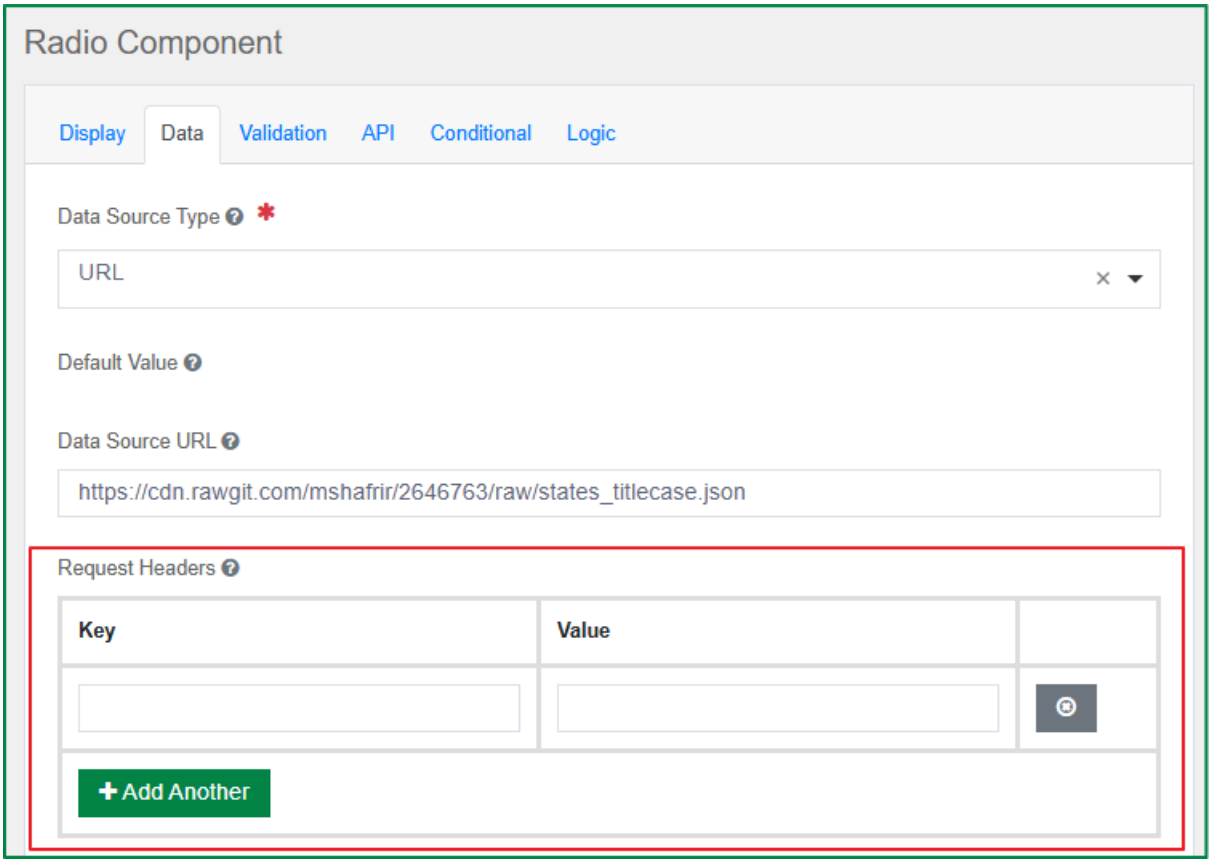
Data Source URL

This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **URL**. You can input the URL that returns a JSON array to use as data source.



Request Headers

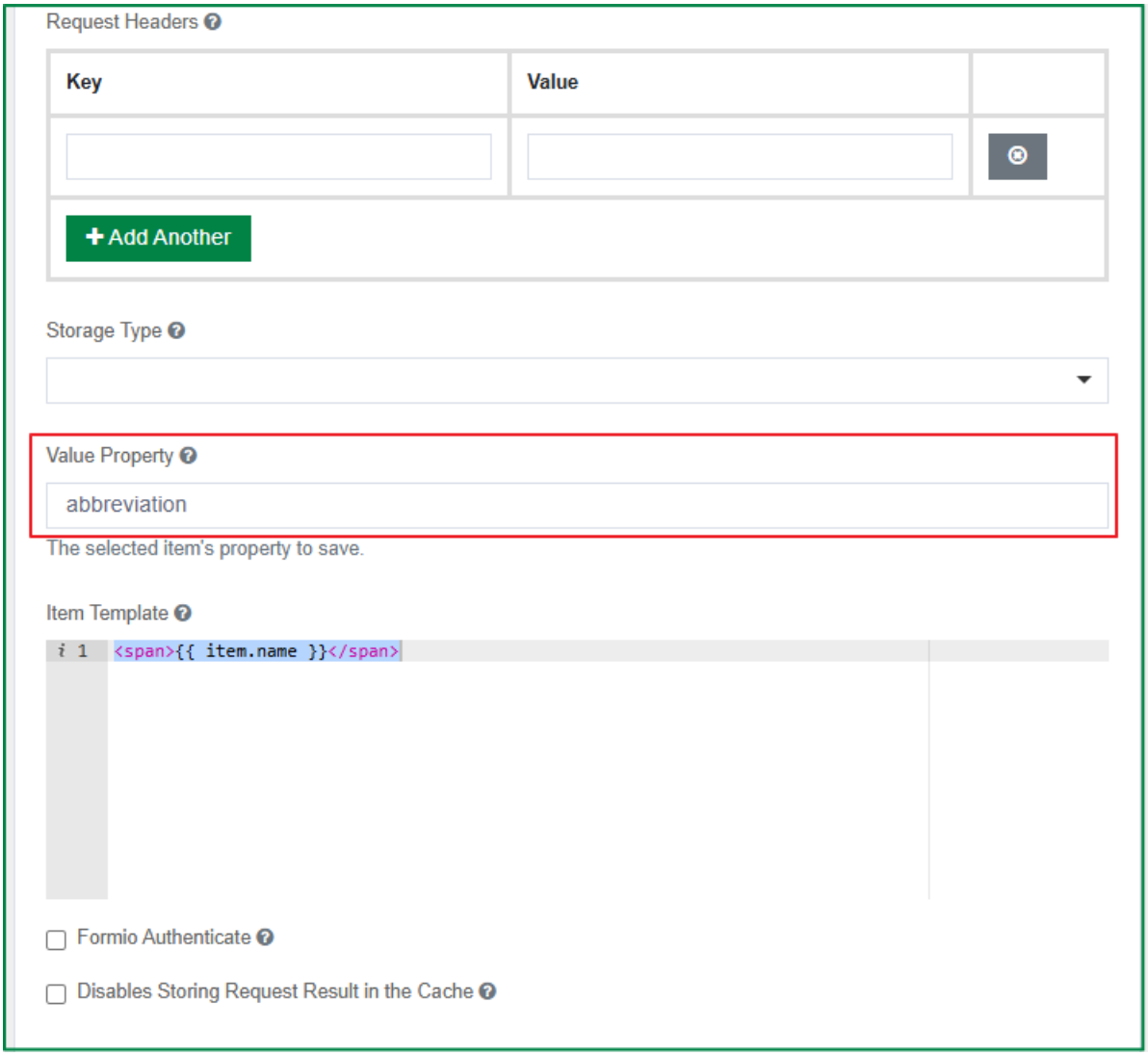
This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **URL**. You can set the headers that should be sent along with the request to the url. This setting is useful for authentication.



Value Property

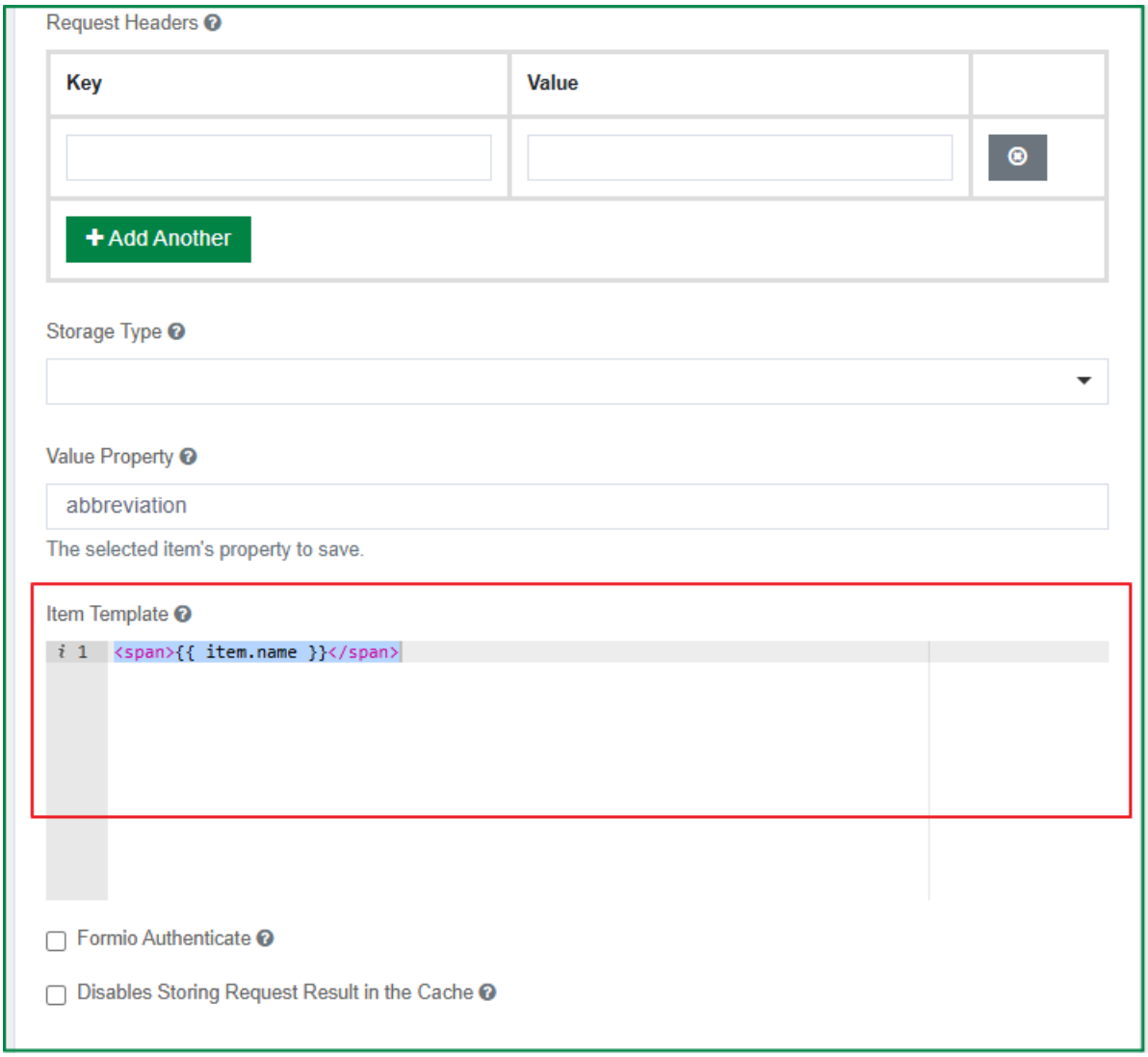
This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **URL**. You can specify the value that corresponds to the desired property within each object which is set in the Item Template. If the value is not specified, the item

itself will be used.



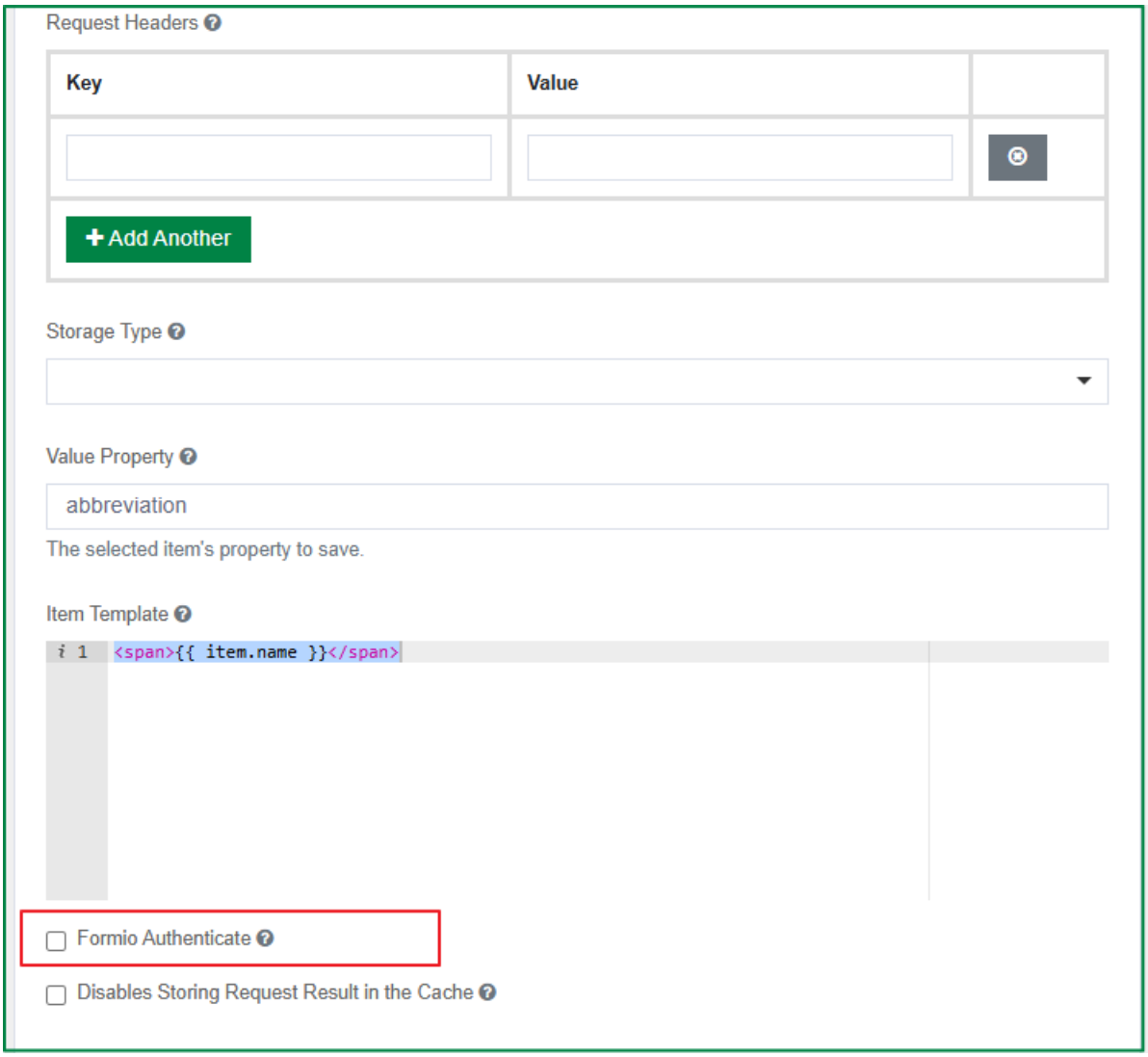
Item Template

This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **URL**. You can input the HTML template for the result data items.



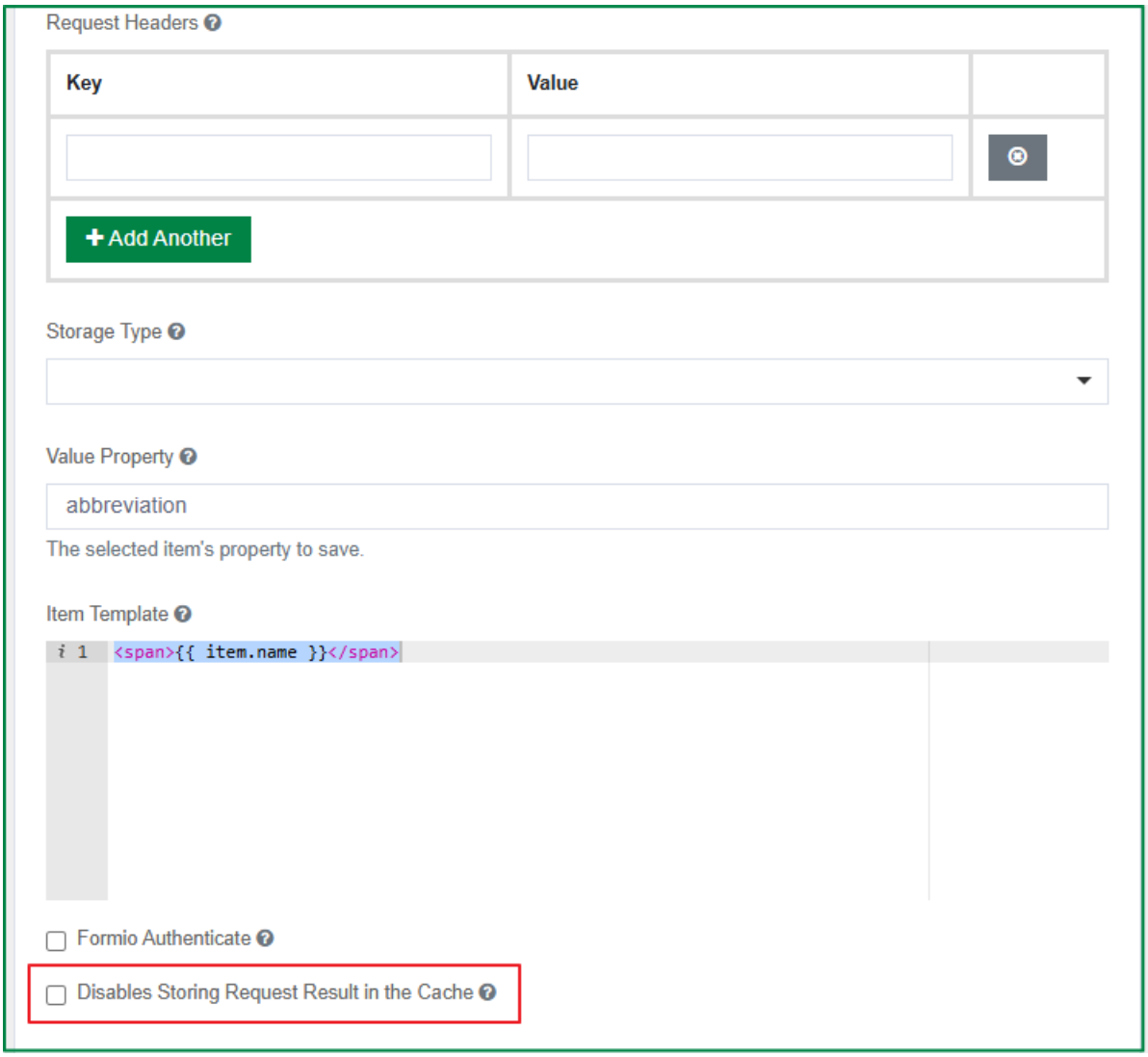
Formio Authenticate

This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **URL**. Tick this checkbox if you want to use Formio Authentication with the request.



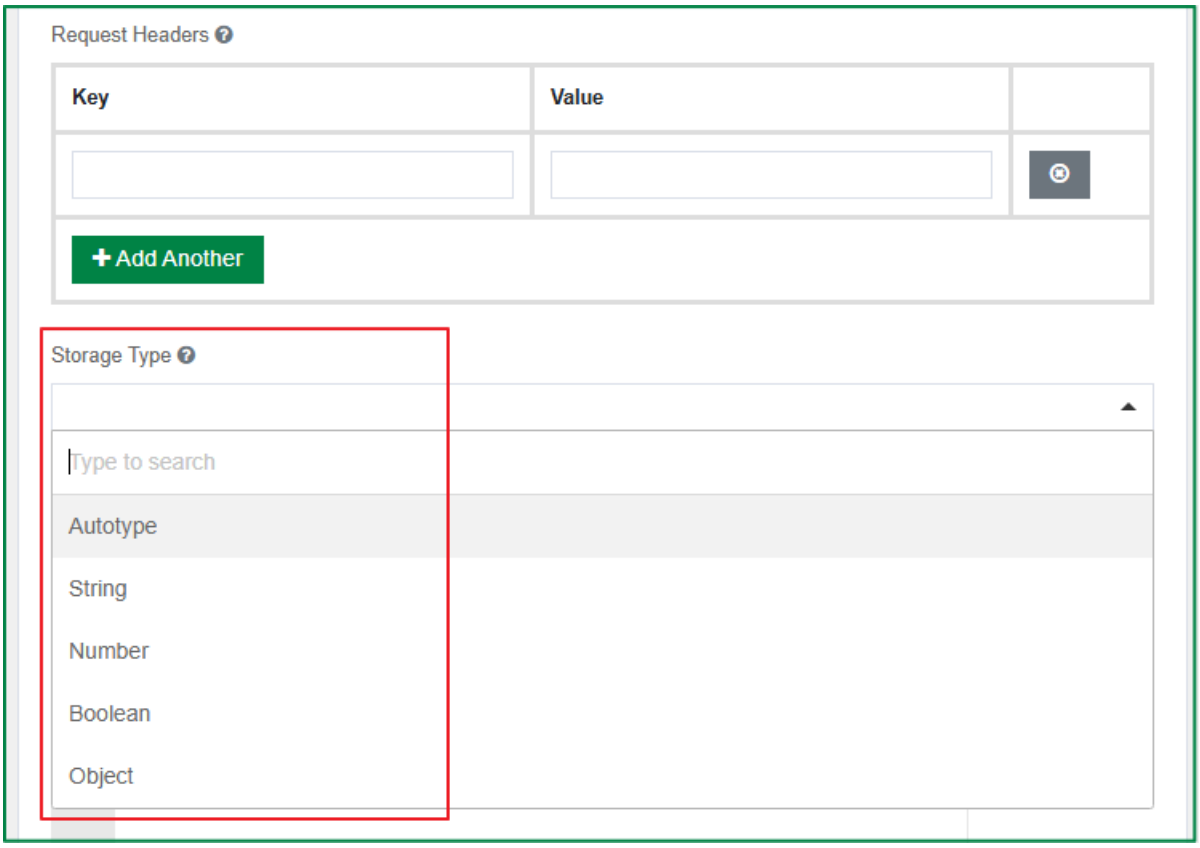
Disables Storing Request Result in the Cache

This option is displayed in the **Data** tab and only enabled if the **Data Source Type** is **URL**. Tick this checkbox if you don't want the requests and its results to be stored in the cache.



Storage Type

This option is displayed in the **Data** tab. If you select a type to store data. If you select a type which is not **Autotype**, the data will be forced to the selected type.



Advanced form components

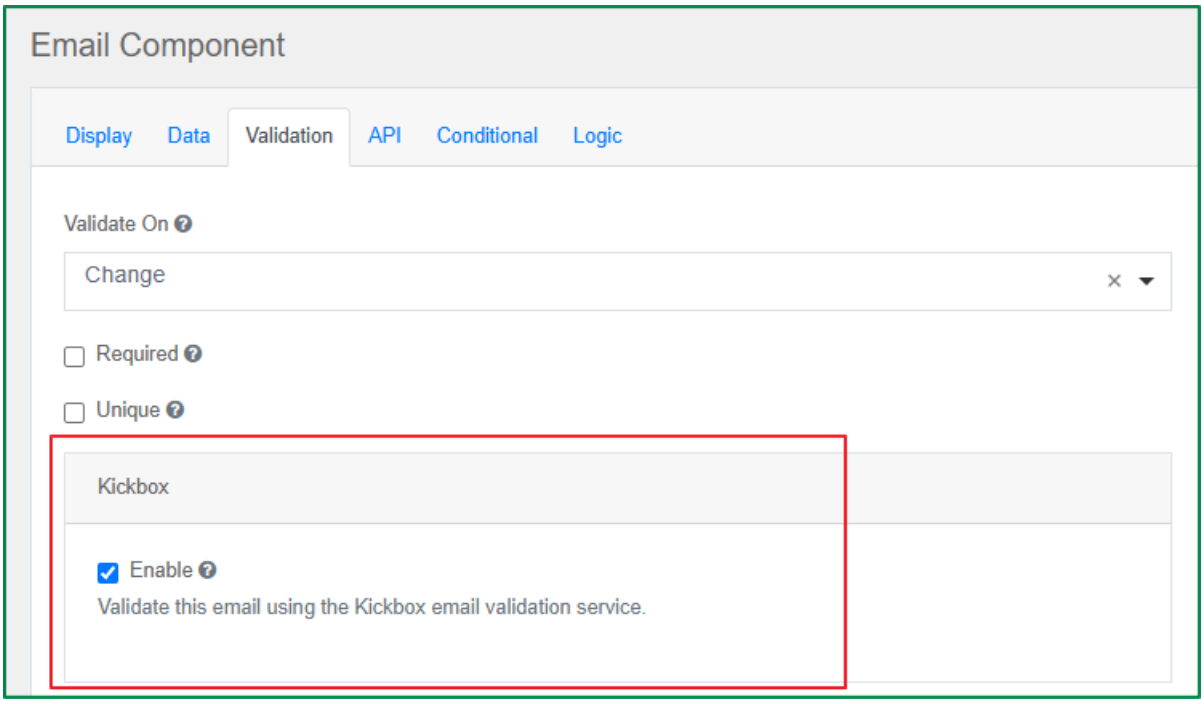
The **Advanced form components** are the extended **Basic form components** so that it can meet the more complex requirements.

Email

The Email component is a string field. This component has the special validation which ensures the entered data is in a valid email format. A valid email address consists of an email prefix and an email domain, both in acceptable formats. Following is the specific settings of the email component.

Kickbox

This option is displayed in the **Validation** tab. You can tick the checkbox to enable the Kickbox email validation service.



URL

The URL component is a string field. This component has the special validation which ensures the entered data is in a valid URL format. With this validation, the entered input data will be checked whether it is at least in the correct format to potentially be a legitimate URL. There are no specific settings for this component.

Phone number

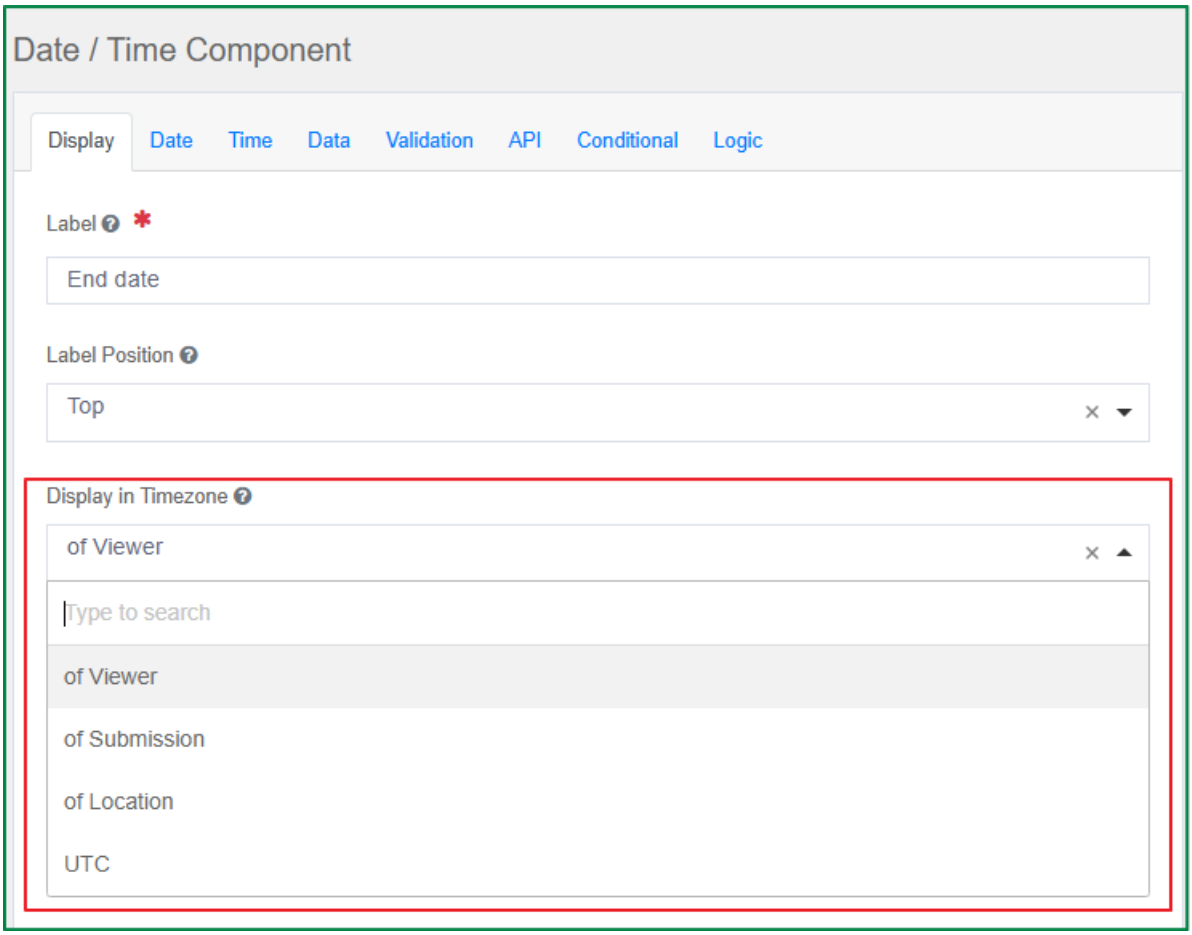
The Phone Number component is a string field. The component has a default input mask to force the user to enter the field data in Phone Number format. There are no specific settings for this component.

Date/Time

The Date/Time componen offers many options for validation, date ranges, and calculations using **Moment.js**. Following is the specific settings of the Date/Time component.

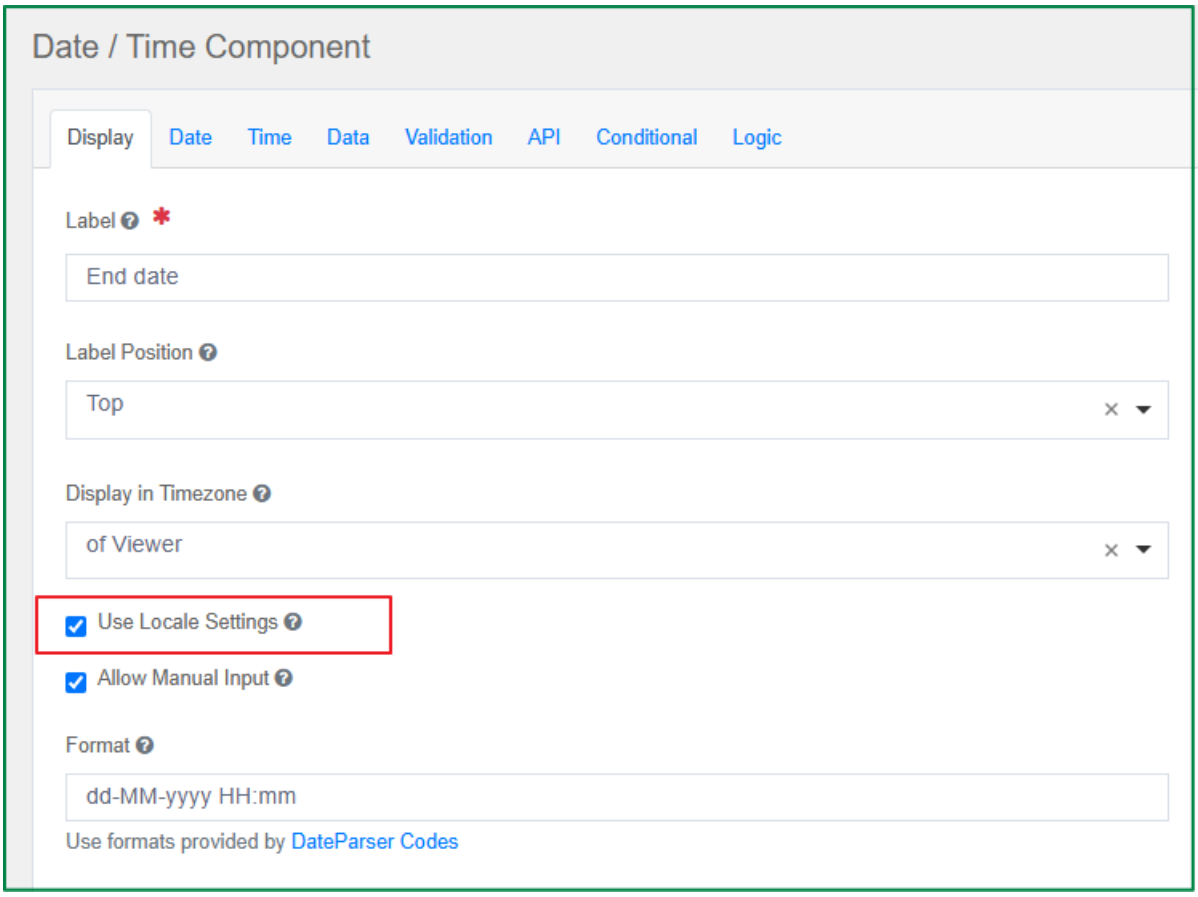
Display in Timezone

This option is displayed in the **Display** tab. You can select whether the display of captured date time should follow the timezone of viewer, submission, location, or UTC.



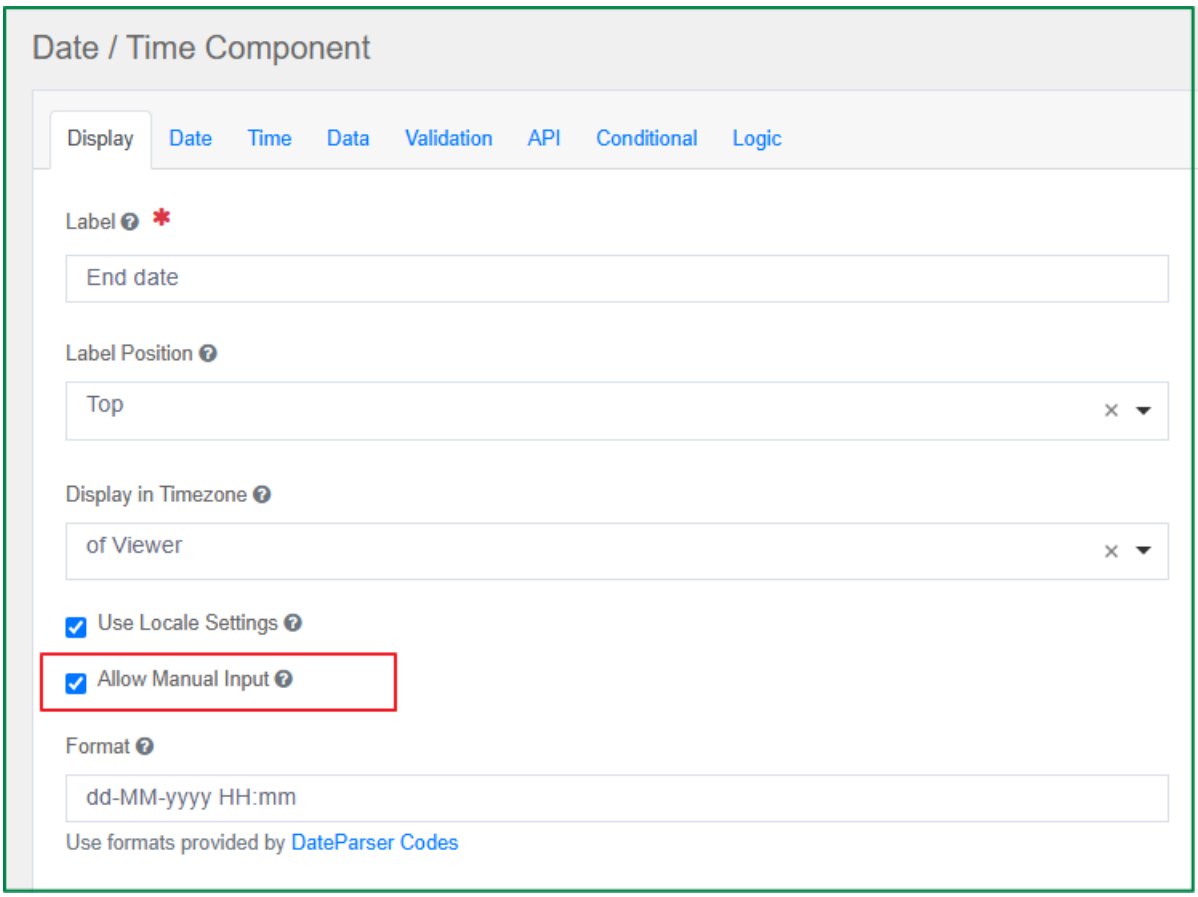
Use Locale Settings

This option is displayed in the **Display** tab. Tick this checkbox if you want to use the Date/Time of your local machine.



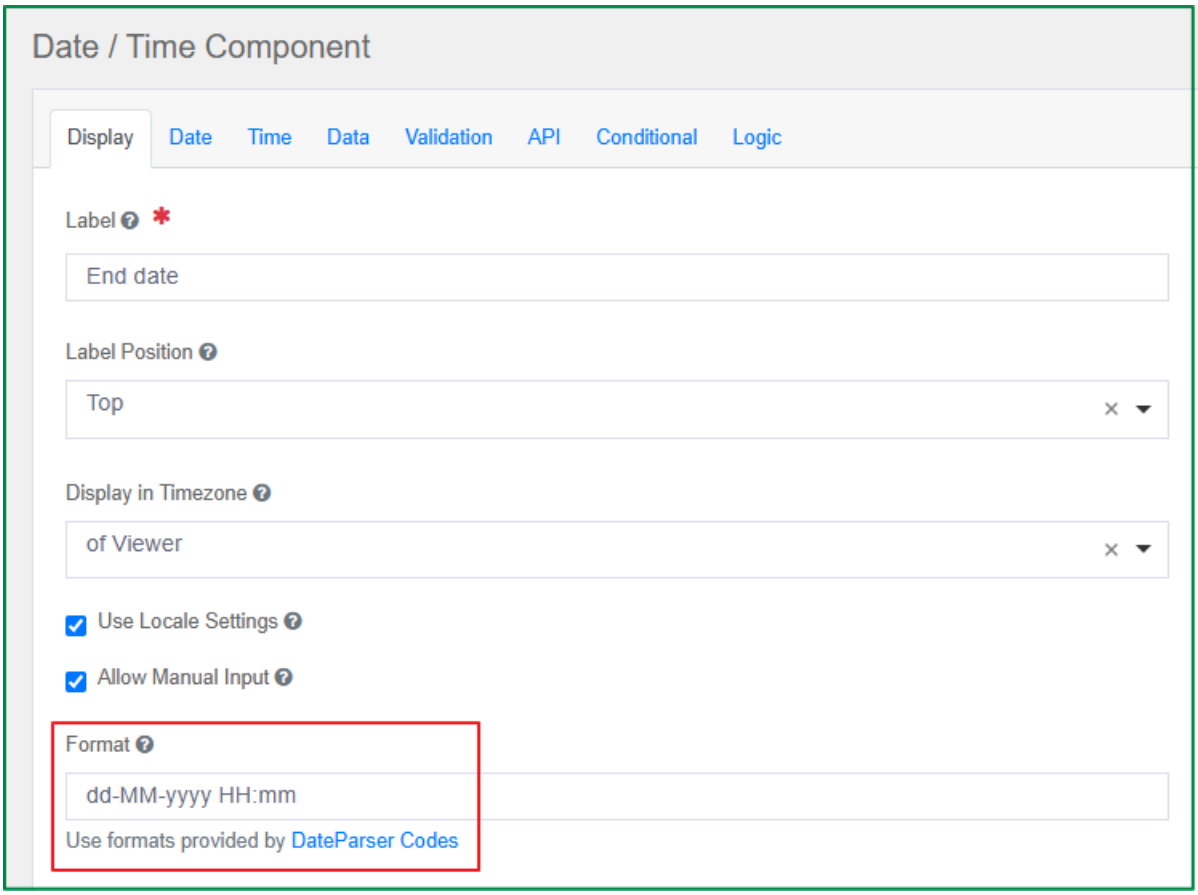
Allow Manual Input

This option is displayed in the **Display** tab. Tick this checkbox if you want to allow the end-user to input the value from the keyboard manually besides selecting date and time from the calendar widget. If the checkbox is unticked, the end-user can only select date and time from the calendar widget.



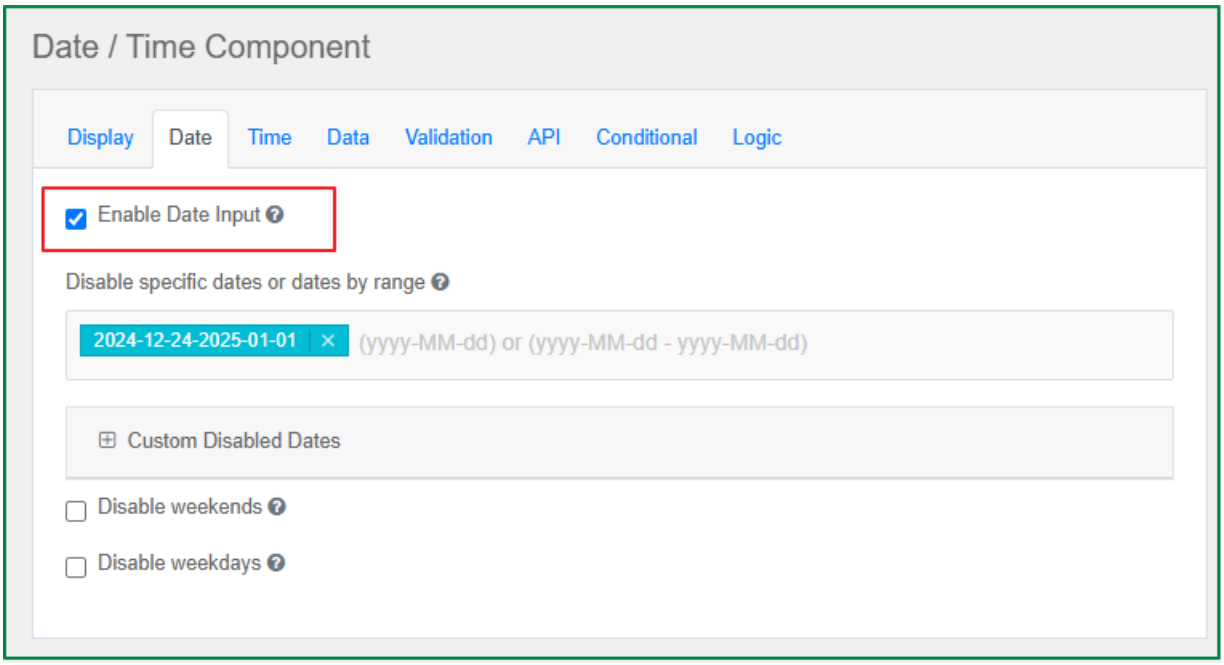
Format

This option is displayed in the **Display** tab. You can configure the format that the Date/Time value will be displayed in. If the time input is enabled, the format value must include *hh:mm a* (for 12-hour time format with AM/PM) or *HH:mm* (for 24-hour time format).



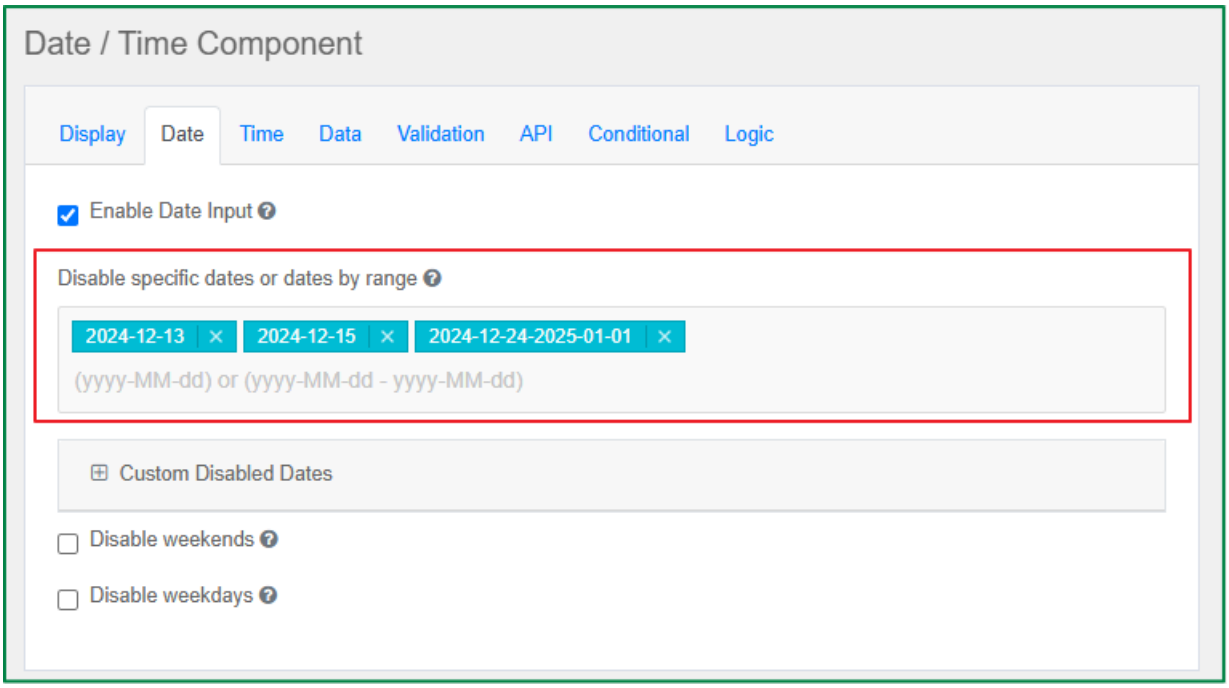
Enable Date Input

This option is displayed in the **Date** tab. This checkbox is ticked by default. If this option is enabled, the end-user can select the date from the calendar widget and can input the date from the keyboard (if the option **Allow Manual Input** is enabled).



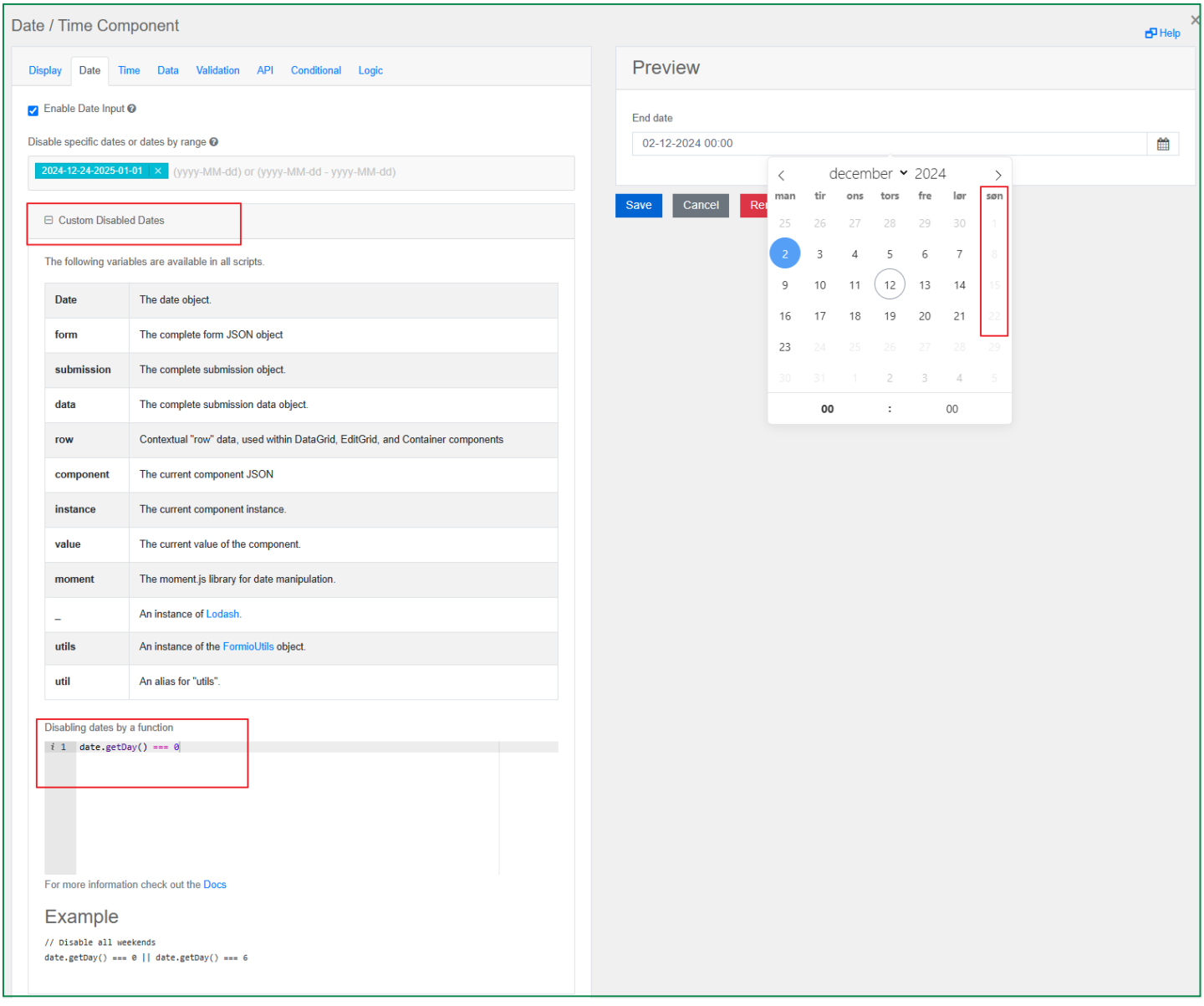
Disable specific dates or dates by range

This option is displayed in the **Date** tab. You can add one or multiple blacklisted dates or date ranges. The end-user cannot input or select the blacklisted date/date range from the calendar widget. You must provide the blacklisted date/date range in the format *yyyy-MM-dd* or *yyyy-MM-dd - yyyy-MM-dd*.



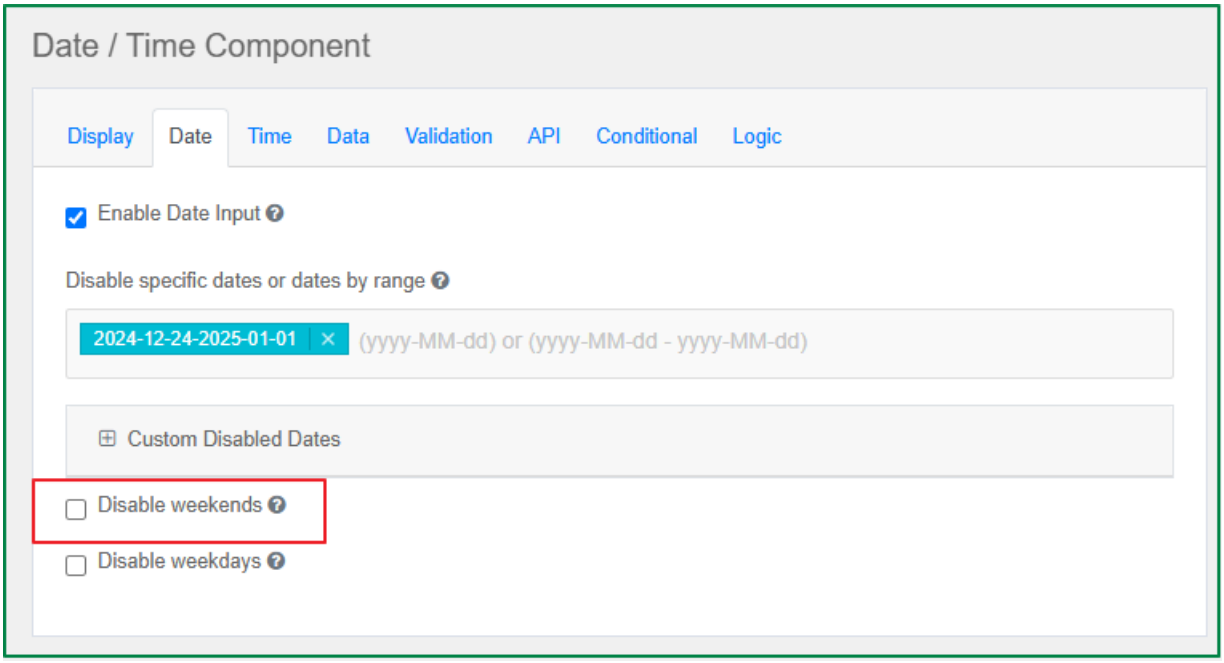
Custom Disabled Dates

This option is displayed in the **Date** tab. When you expand the option,you can write a function with Java scripts to customize the disabled dates in the field**Disabling dates by a function**. The following example is the function of disabling all Sunday.



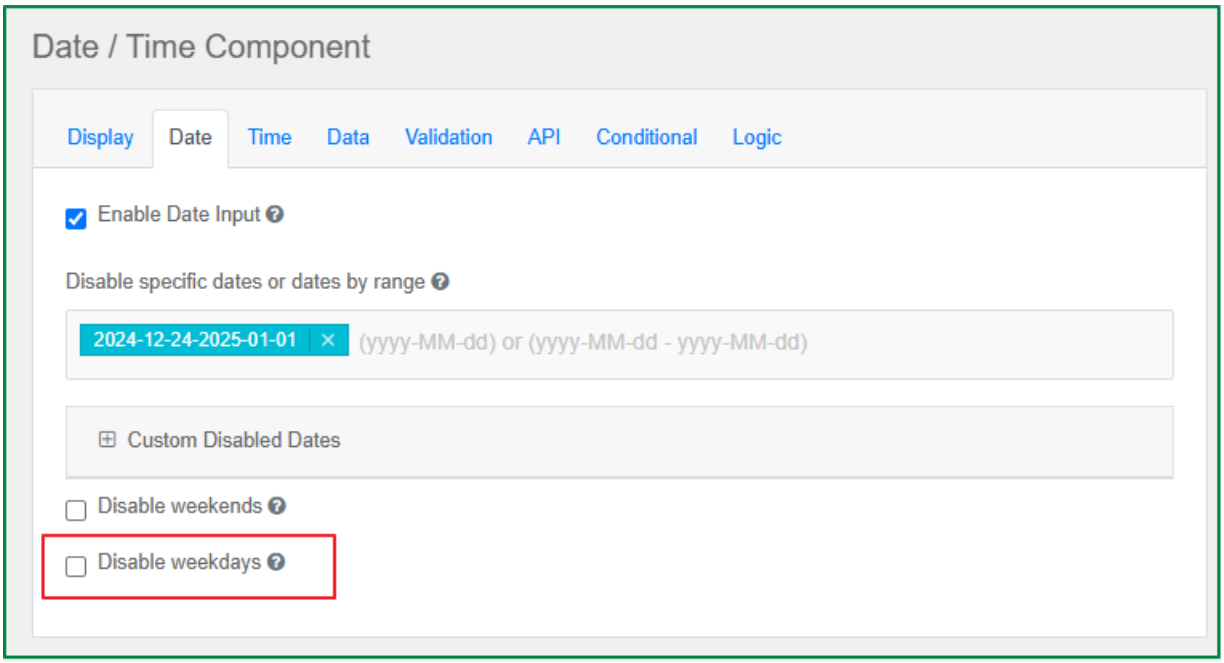
Disable weekends

This option is displayed in the **Date** tab. Tick this checkbox if you want the weekends are disabled from inputting and selecting from calendar widget.



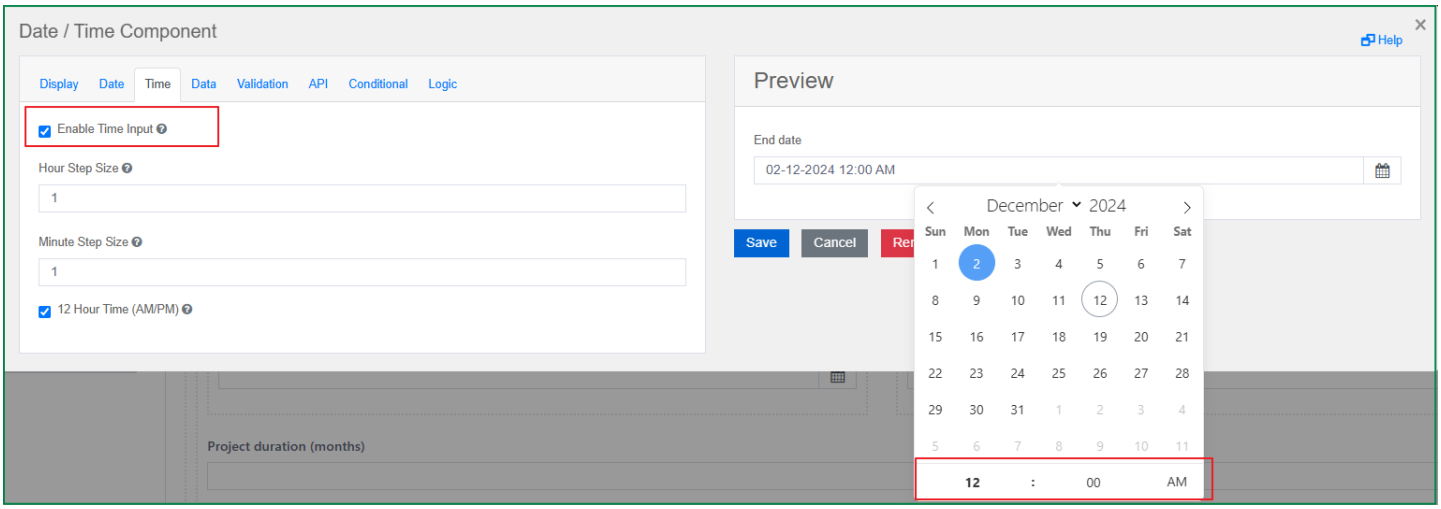
Disable weekdays

This option is displayed in the **Date** tab. Tick this checkbox if you want the weekdays are disabled from inputting and selecting from calendar widget.



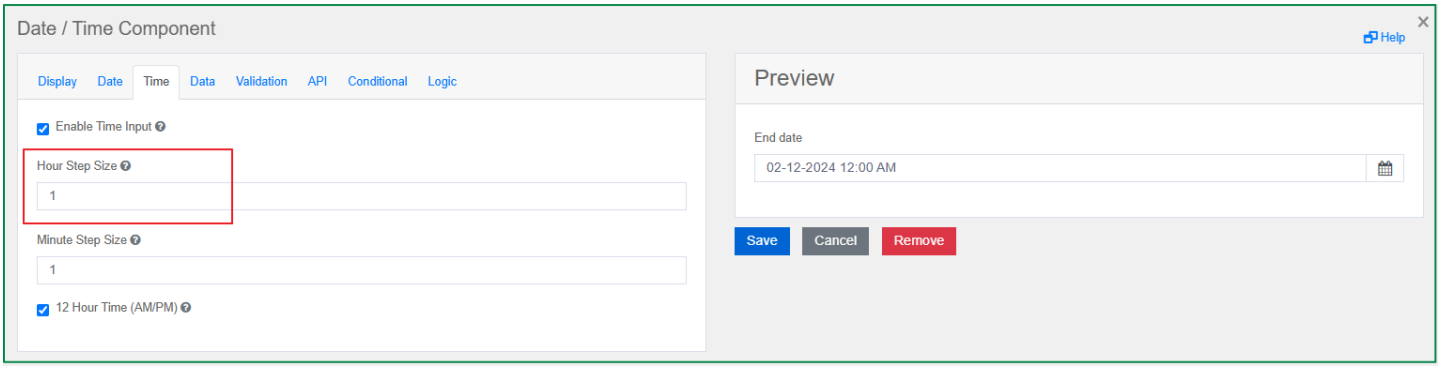
Enable Time Input

This option is displayed in the **Time** tab. This checkbox is ticked by default. If this option is enabled, the end-user can input the time in the calendar widget, or in the input field if the option **Allow Manual Input** is enabled.



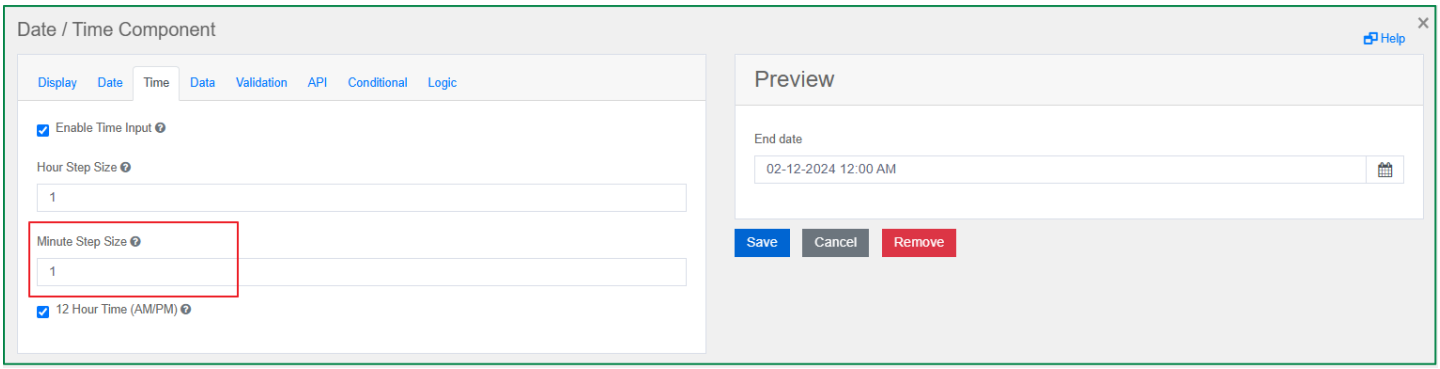
Hour Step Size

This option is displayed in the **Time** tab. You can provide the number of hours to be incremented/decremented in the time picker. The default hour step size is 1 hour.



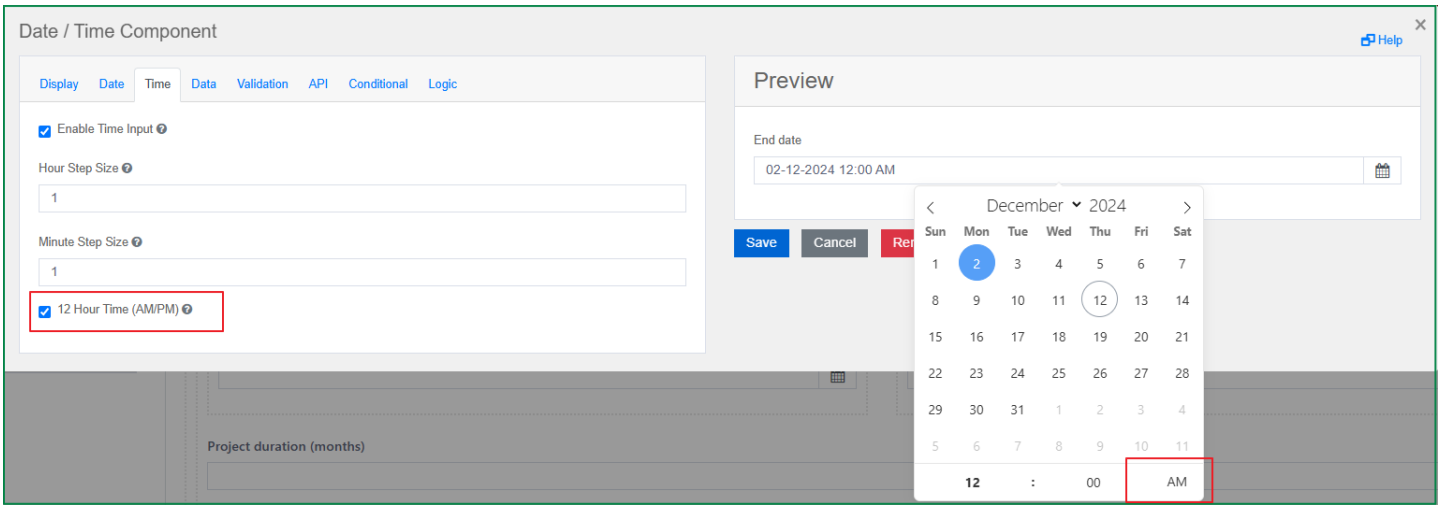
Minute Step Size

This option is displayed in the **Time** tab. You can provide the number of minutes to be incremented/decremented in the time picker. The default minute step size is 1 minute.

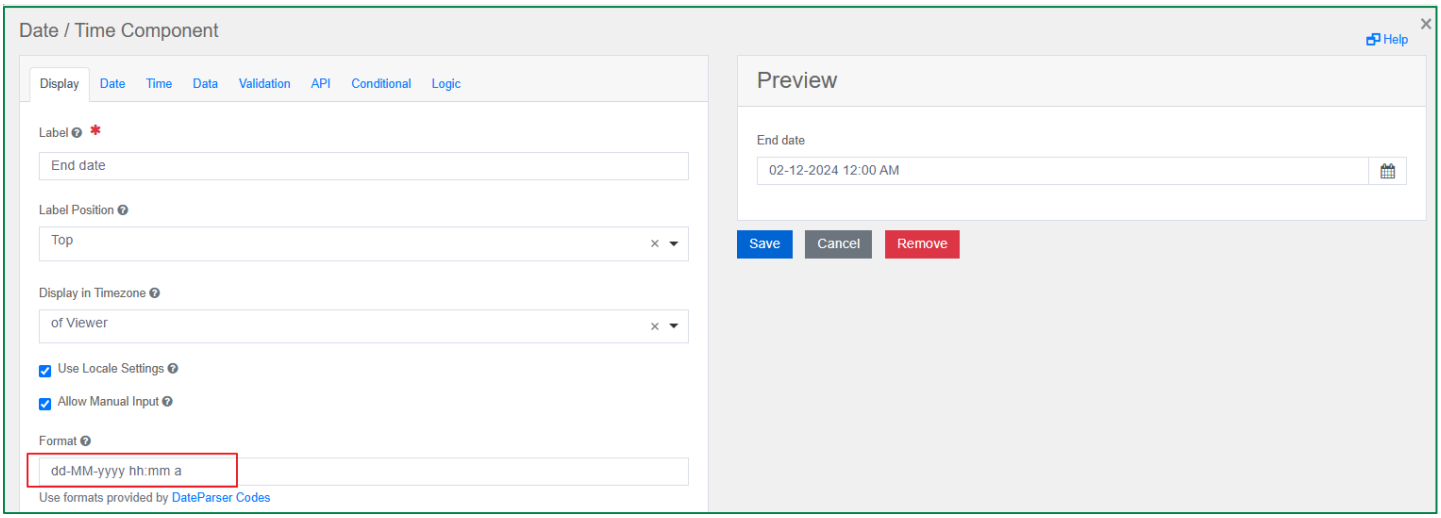


12 Hour Time (AM/PM)

This option is displayed in the **Time** tab. Tick this checkbox if you want the time is displayed in 12-hour time format with AM/PM in the time picker . If the checkbox is unticked, the time will be displayed in 24-hour time format.

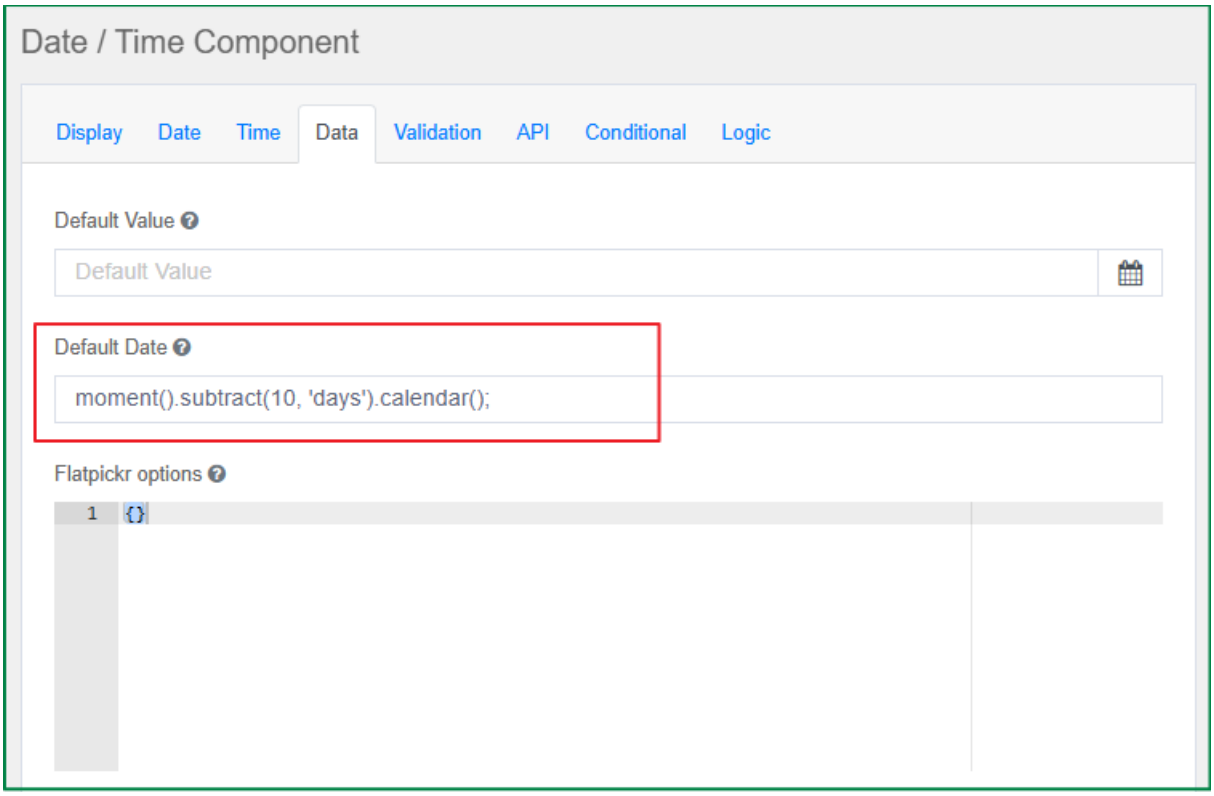


When you tick/untick the checkbox but the defined format in the **Display** tab is inconsistent, the defined format in the **Display** tab will be updated automatically. *For example,* if the defined format is *dd-mm-yyyy hh:mm a* , then you tick this option; after you save the change, the defined format will be updated to *dd-mm-yyyy HH:mm* automatically.



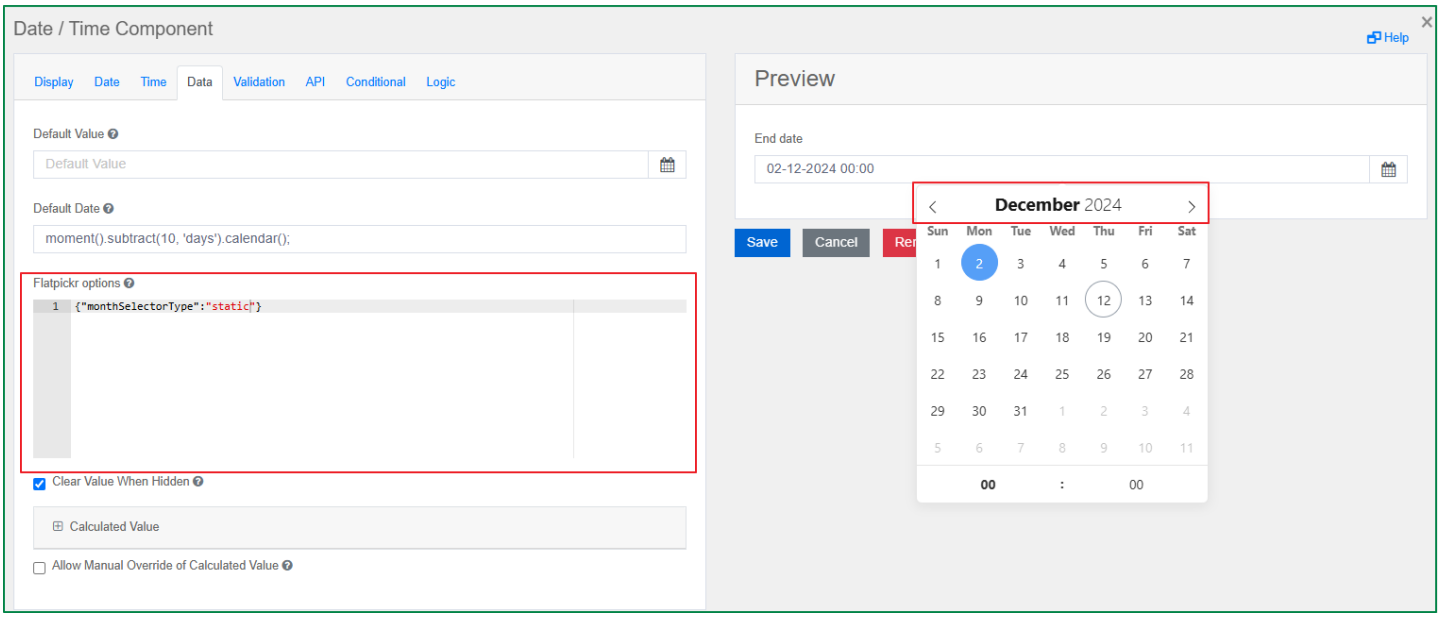
Default Date

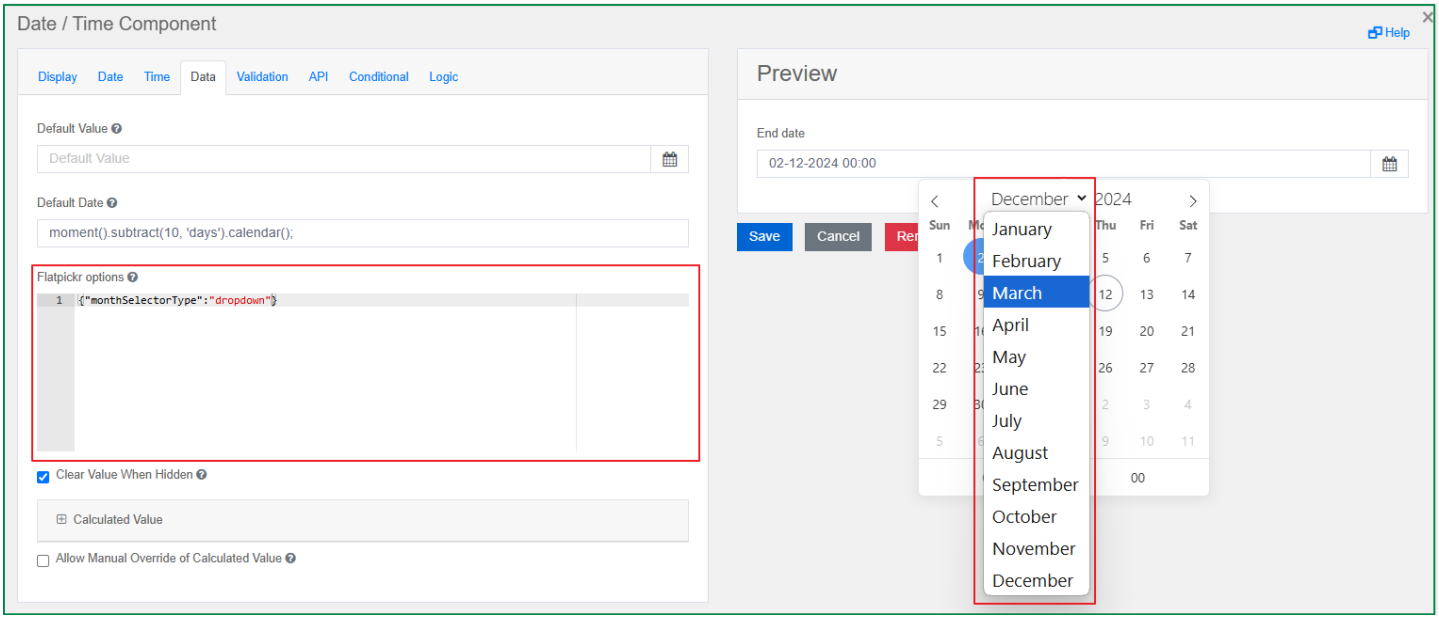
This option is displayed in the **Data** tab. You can set a default value to a specific date by using the Moment.js function.



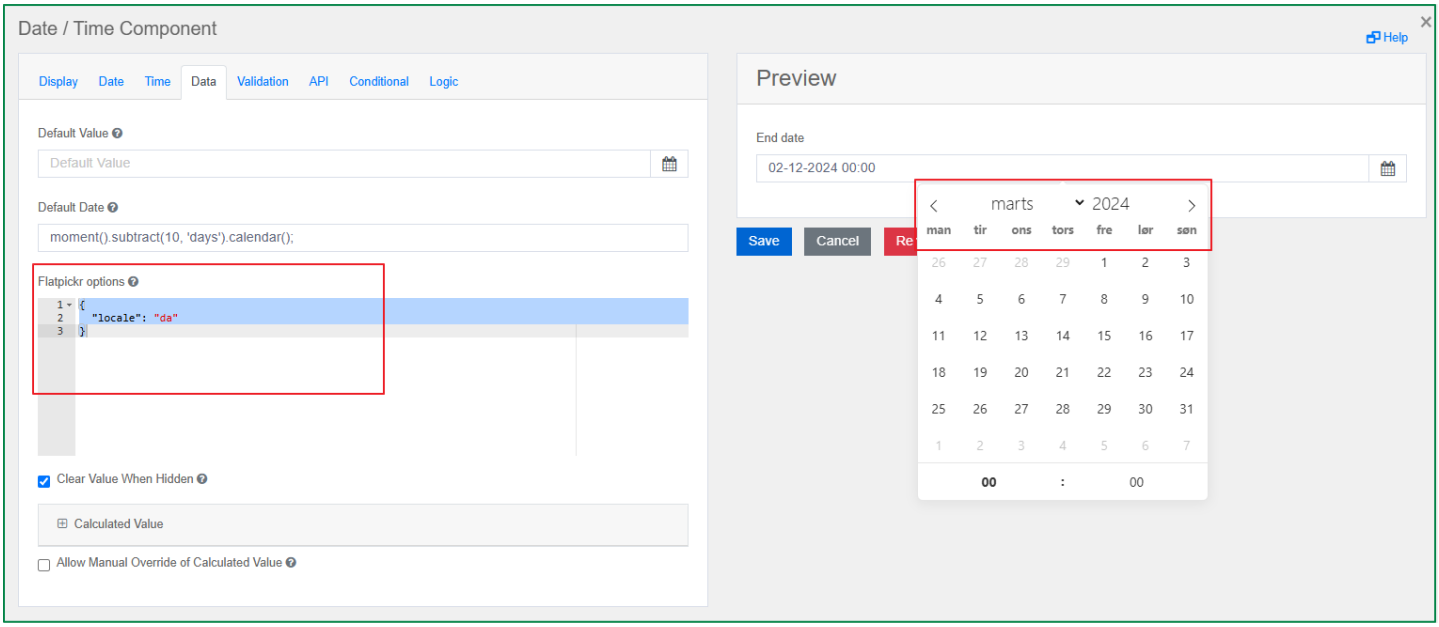
Flatpickr options

This option is displayed in the **Data** tab. You can input the raw JSON to set the options of the Flatpickr library, which is used to create the datepicker widget. *For example*, you can set the selector type of the month to static or dropdown as the following.





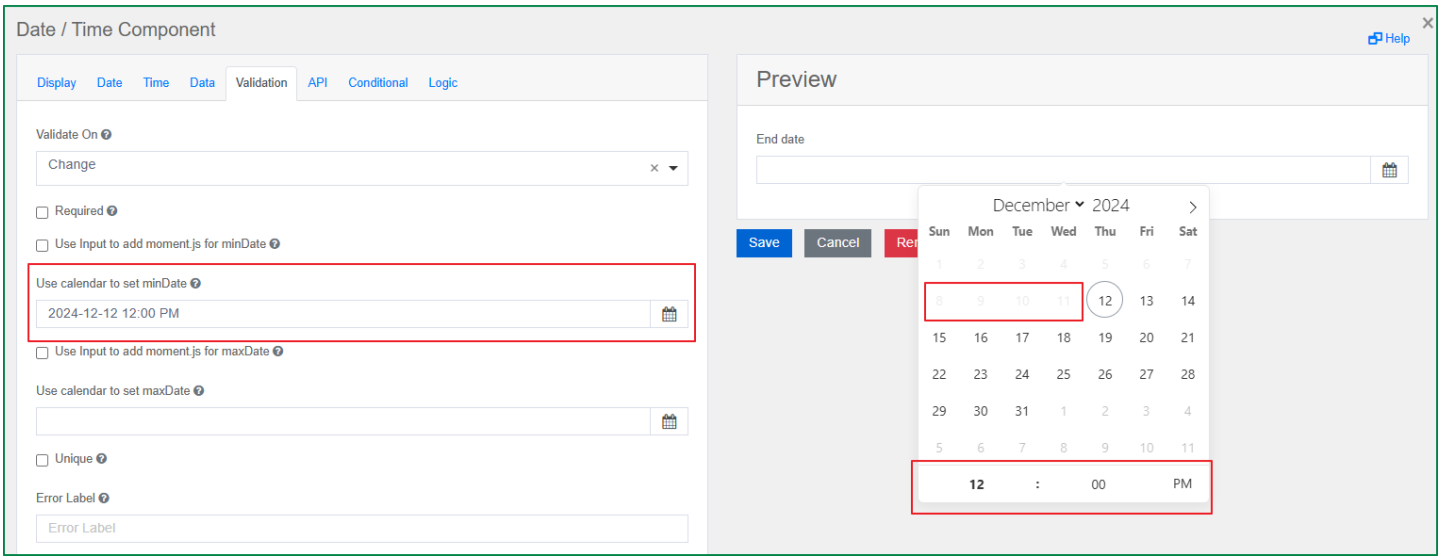
Besides that, you can also use this option to set the translation for the date picker by providing a value to the locale variable. The following example is translating the date picker to Danish.



Use calendar to set minDate

This option is displayed in the **Validation** tab. This option is a calendar widget, you can set the minimum date (time) by selecting a date from the calendar widget. Once the minimum date (time) was set, the end-user cannot select a date (time) which is earlier than the defined minimum date (time).

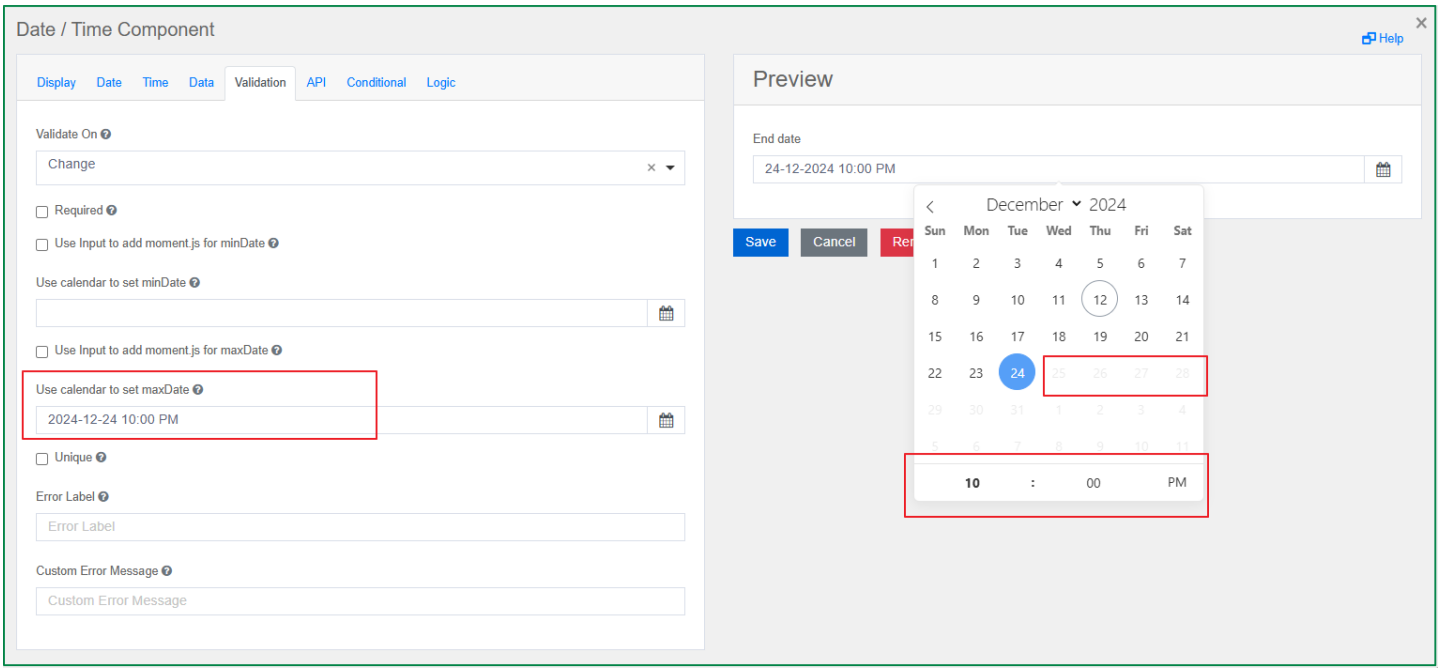
This option will be hidden if the **Use Input to add moment.js for minDate** checkbox is ticked.



Use calendar to set maxDate

This option is displayed in the **Validation** tab. This option is a calendar widget, you can set the maximum date (time) by selecting a date from the calendar widget. Once the maximum date (time) was set, the end-user cannot select a date (time) which is later than the defined maximum date (time).

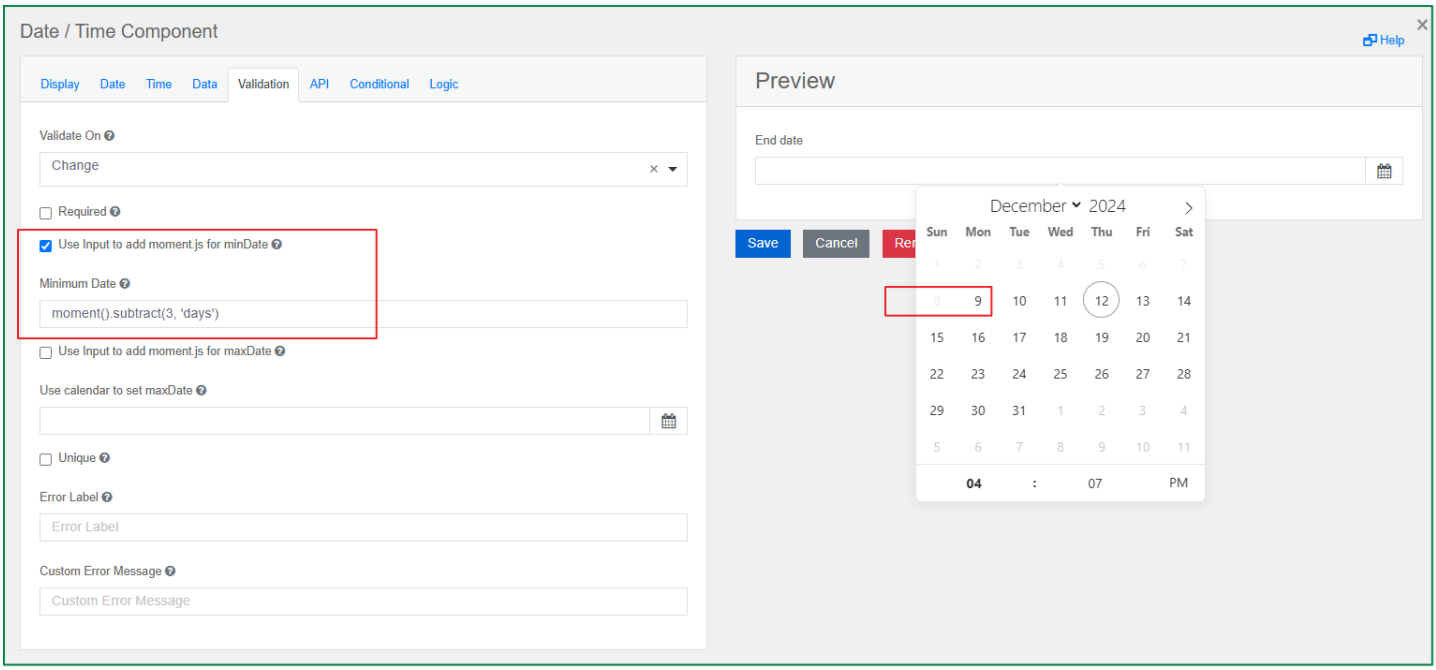
This option will be hidden if the **Use Input to add moment.js for maxDate** checkbox is ticked.



Use Input to add moment.js for minDate

This option is displayed in the **Validation** tab. Similar to the option **Use calendar to set minDate**, this option is used to set the minimum date that the end-user can select. The different is that with this option, you can use the Moment.js instead of picking a specific date.

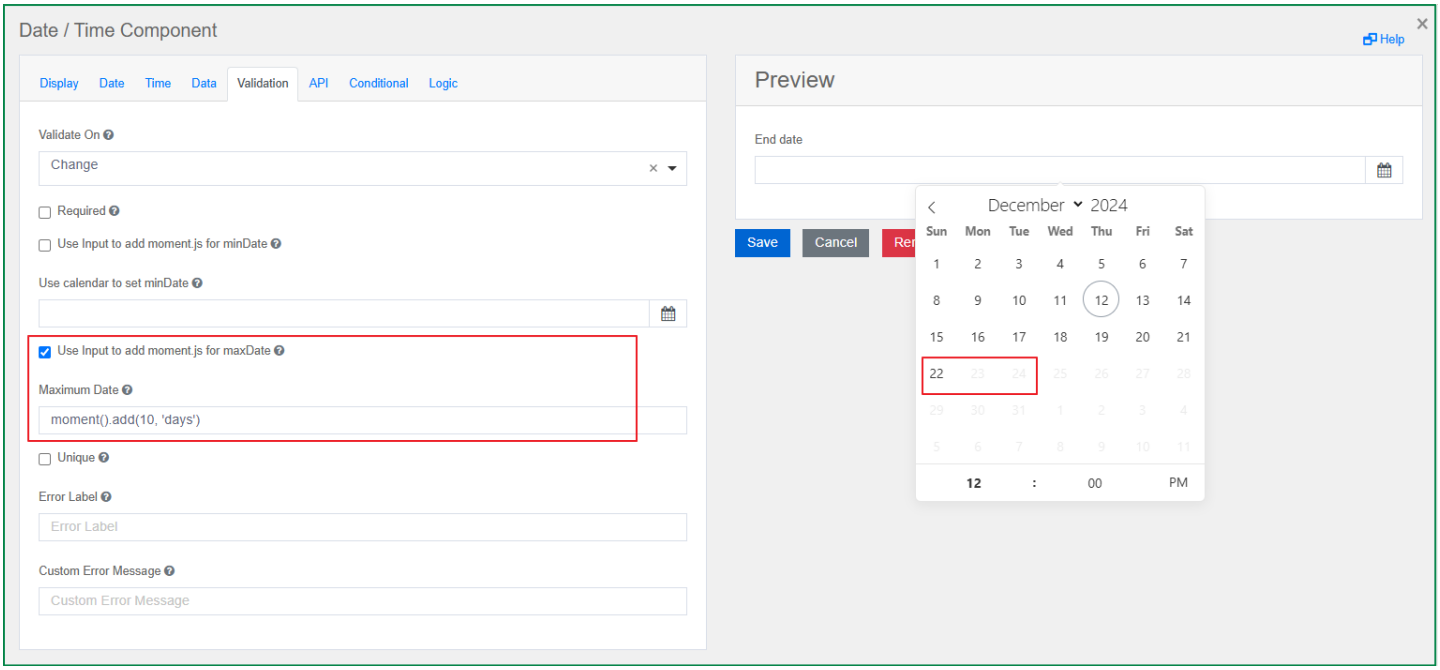
When you tick the checkbox, the input field **Minimum Date** will be displayed and you can fill the Moment.js function to the field to set the minimum date.



Use Input to add moment.js for maxDate

This option is displayed in the **Validation** tab. Similar to the option **Use calendar to set maxDate**, this option is used to set the maximum date that the end-user can select. With this option, instead of picking a specific date, you can use the Moment.js.

When you tick the checkbox, the input field **Maximum Date** will be displayed and you can fill the Moment.js function to the field to set the maximum date.

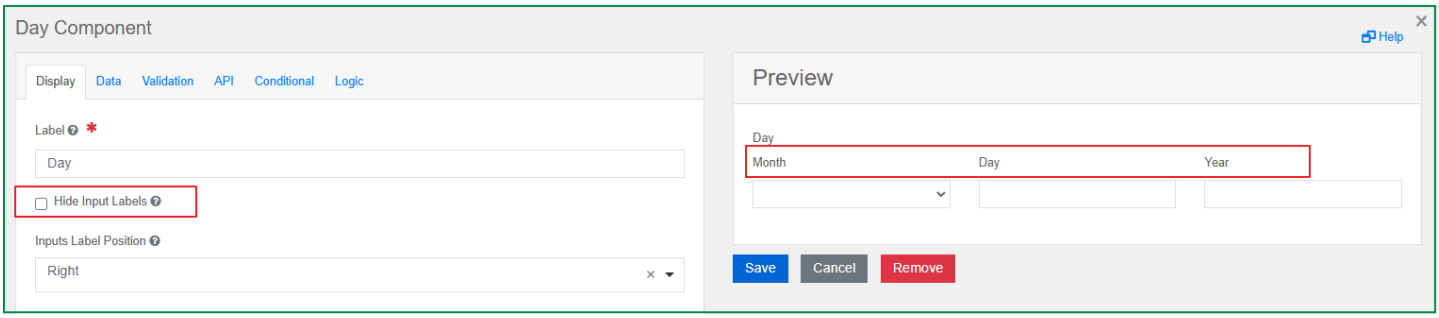


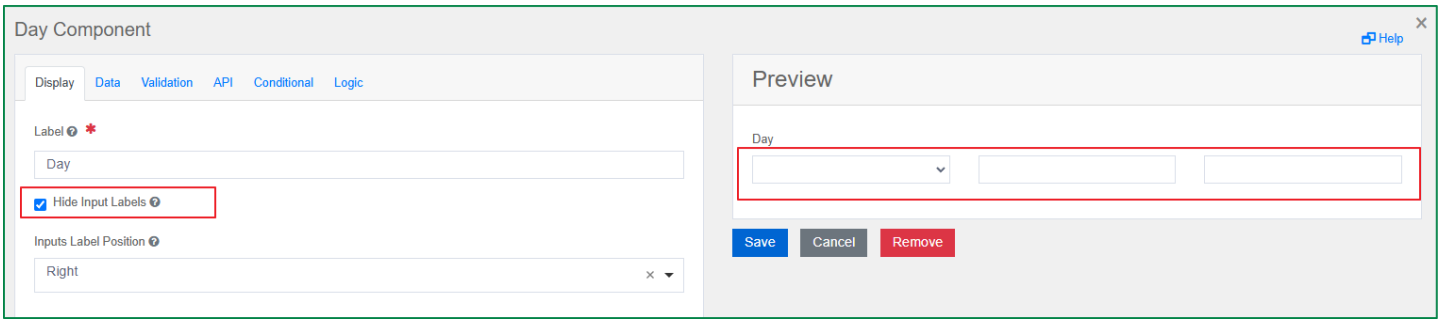
Day

The Day component is used to enter values for the Day, Month, and Year using a number or select type of field. Following is the specific settings of the component.

Hide Input Labels

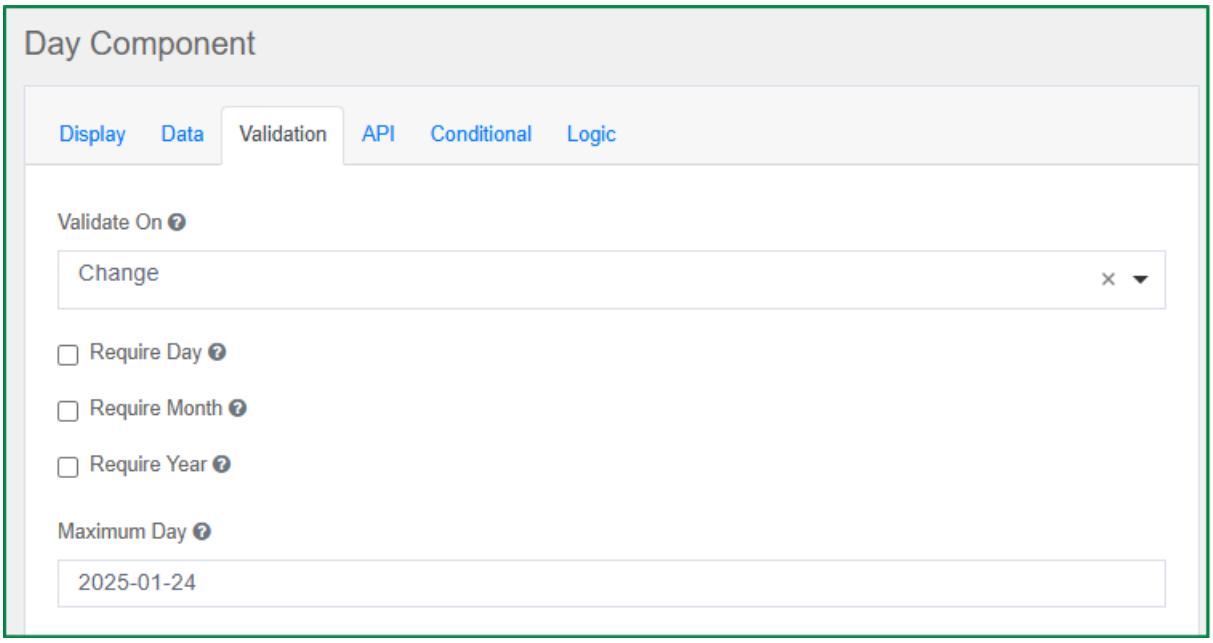
This option is displayed in the **Display** tab. This option is used to show/hide the input labels like Month, Day, Year. Tick the checkbox if you want the input labels to be hidden.

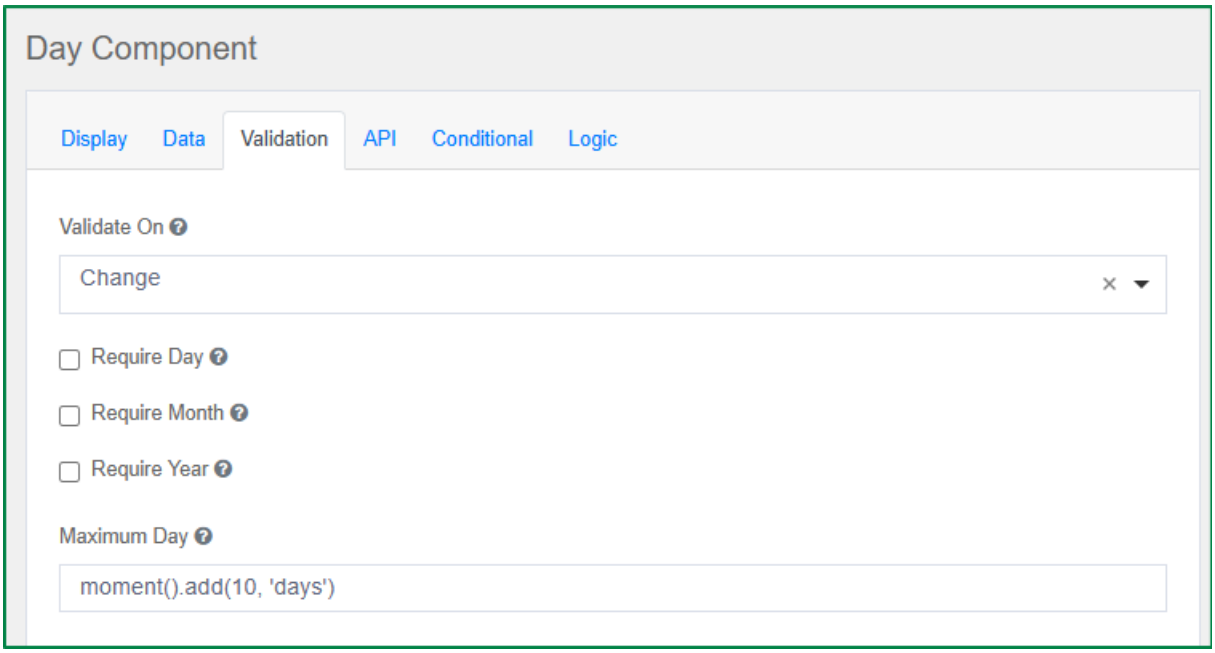




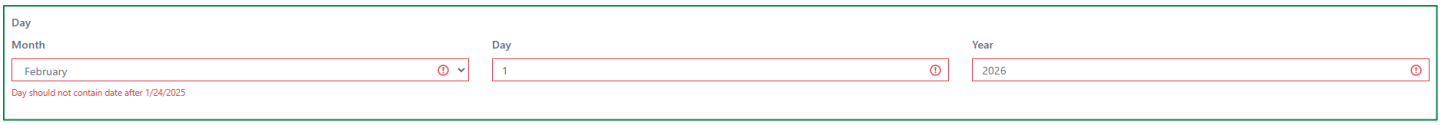
Maximum Day

This option is displayed in the **Validation** tab. With this option, you can define the maximum date that the end-user can set. You can input a specific date or use Moment.js.



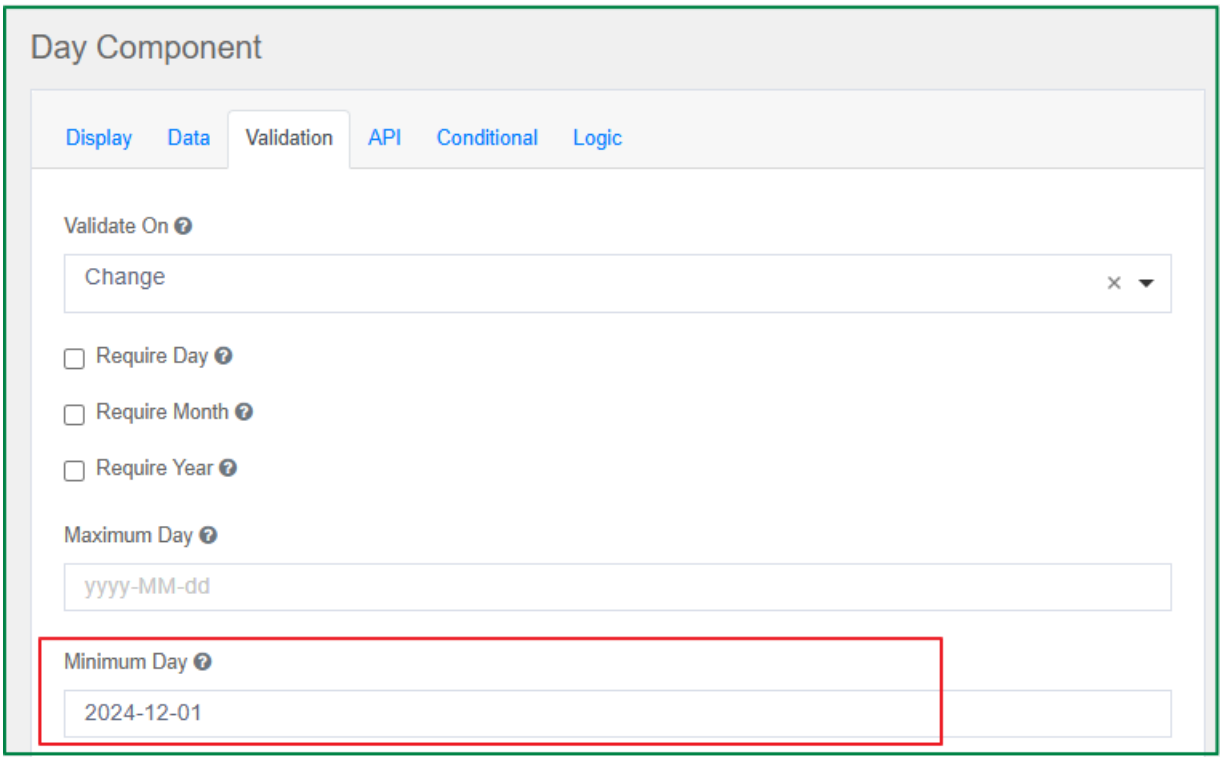


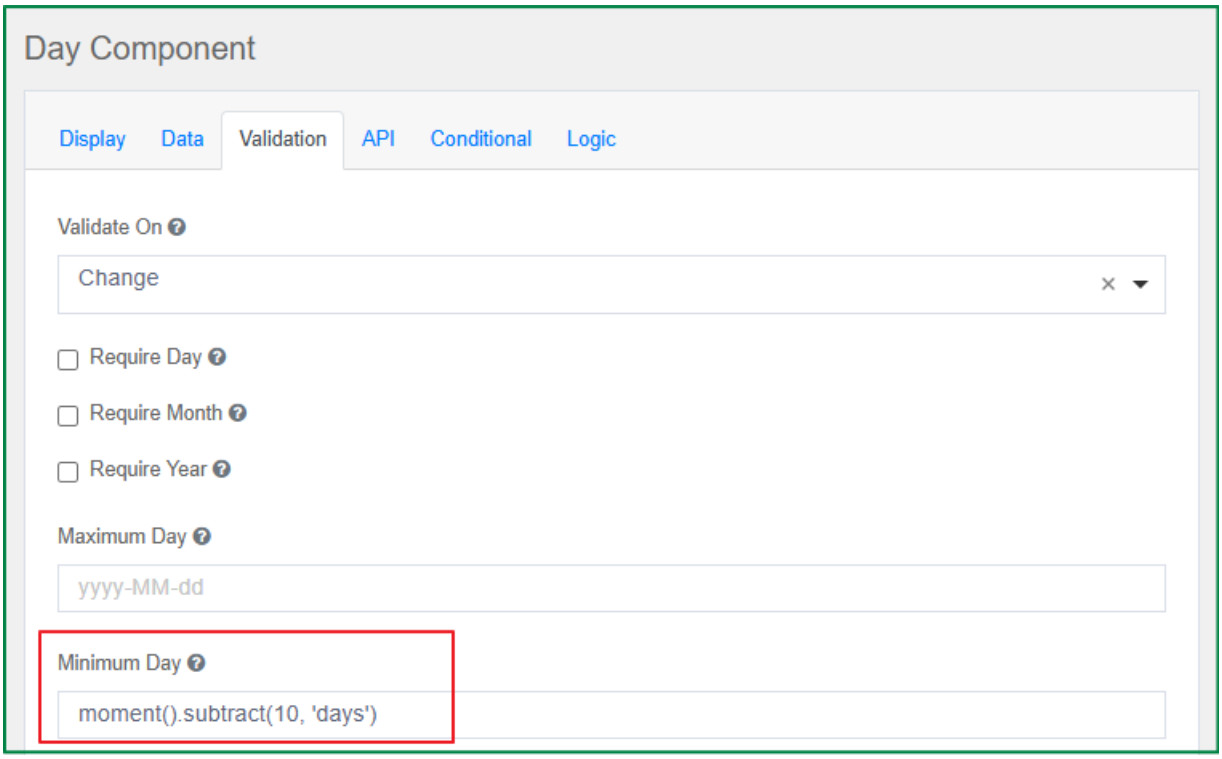
When the maximum day is set, if the end-user selects a date which is later than the set maximum day, error message will be displayed as following and the end-user cannot submit the form.



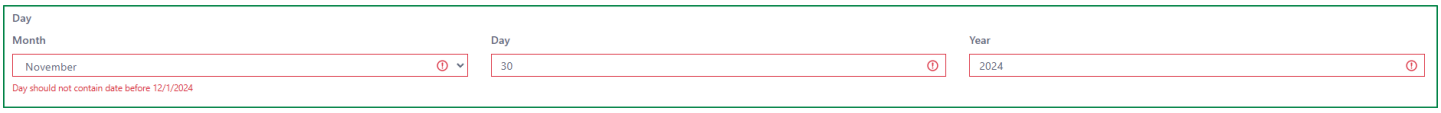
Minimum Day

This option is displayed in the **Validation** tab. With this option, you can define the minimum date that the end-user can set. You can input a specific date or use Moment.js.





When the minimum day is set, if the end-user selects a date which is earlier than the set minimum day, error message will be displayed as following and the end-user cannot submit the form.

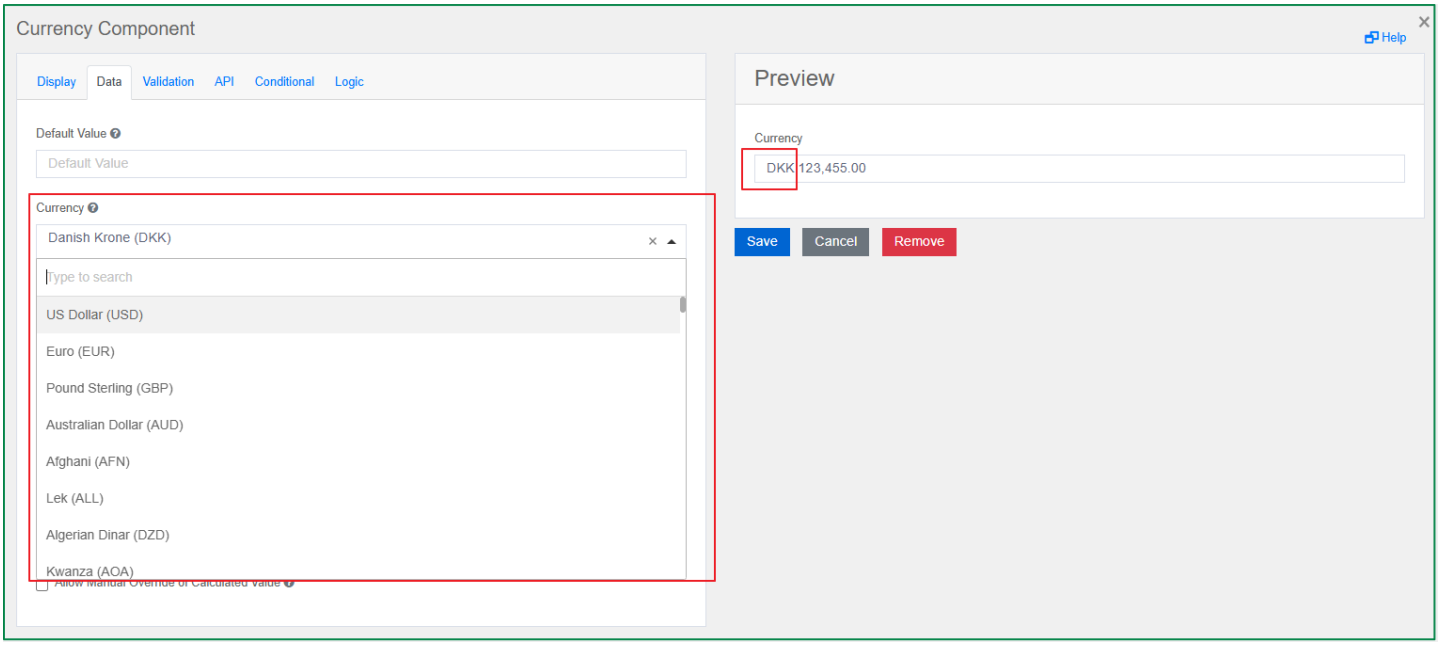


Currency

The Currency component should be used when a field needs to display monetary amounts. This component holds a numeric input mask that allows two decimal values and automatically adds commas as a user inputs a currency amount. The specific setting of this component is the selection of currency type which changes the prefix currency symbol as following.

Currency

This option is displayed in the **Data** tab. It includes a list of currency types and you can select one of them. The prefix of the value will be changed according to the selected currency type.



Survey

The Survey component should be used when you want to display a list of radio button questions. Survey is a great component to utilize when asking multiple questions with the same context of answers or values. With this component, you can configure the questions and the answers within the component settings as following.

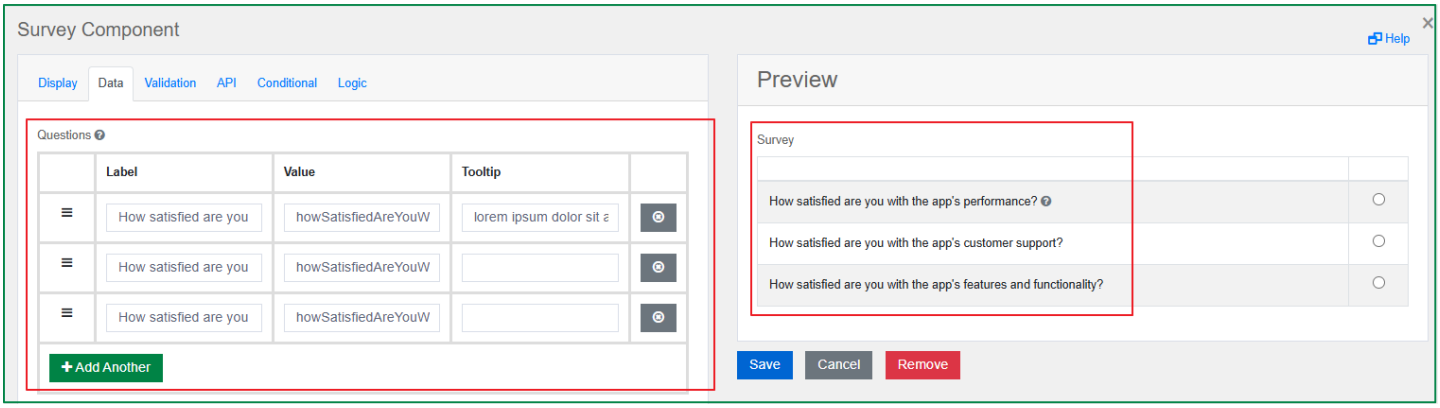
Questions

This option is displayed in the **Data** tab. You can input the questions that you would like to ask in the survey. The question will be displayed vertically in the left-hand column. A question includes 3 parts: **Label**, **Value**, **Tooltip**.

Label is the front-end identifier for the end-user.

Value is back-end identifier for the option and is what is saved to the database.

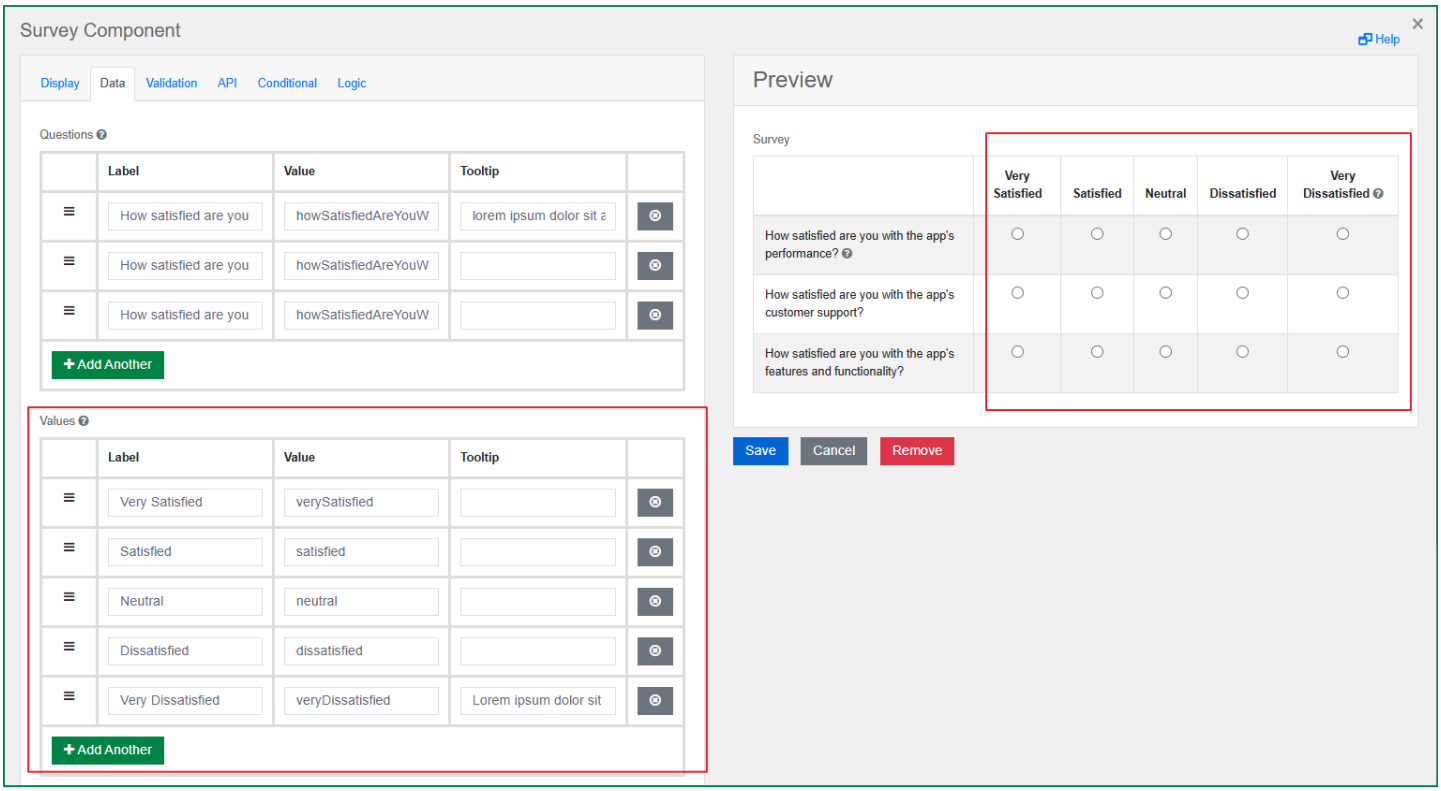
Tooltip is used as the explanation for the end-user. At the view of the end-user, a question mark icon will be displayed at the right of the question label if the tooltip was filled. And when the end-user hover over the icon, the tooltip value will be displayed.



Values

This option is displayed in the **Data** tab. You can input the answers for all the added questions above. The answer options will be the same for all questions. Each answer will be displayed as a column with the label at the top, and the radio buttons under the label. All answer columns are displayed horizontally at the right of the question column. The order of the answer columns from left to right are based on the order of the answers from top to bottom in the component setting.

Similar to the question, an answer includes 3 parts: **Label, Value**, **Tooltip**.



File

The File component allows form users to upload and download files to a form. Following is the specific settings of this component.

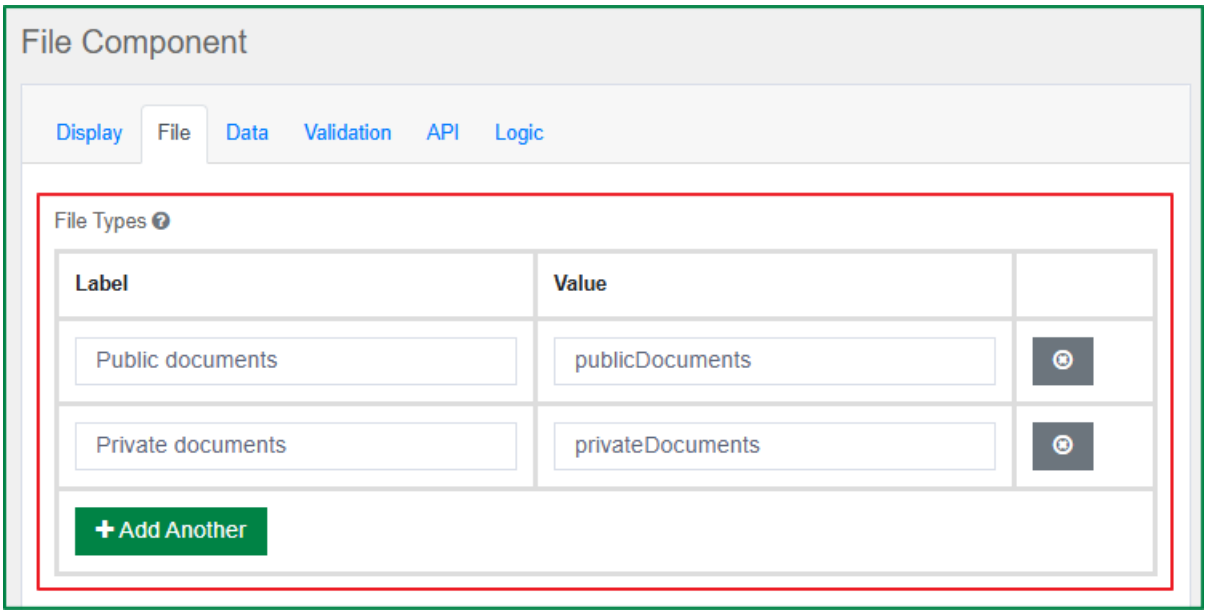
File Types

This option is displayed in the **File** tab. With this option, you can define a list of file types which is for the end-user to use to classify the uploaded files. For example, the uploaded file can be classified into **Public documents**, or **Private documents**.

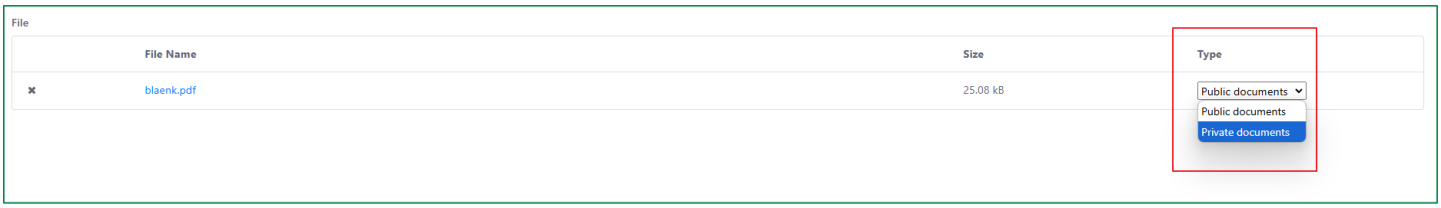
This option includes 2 parts: **Label**, and **Value**.

Label is the front-end identifier for the end-user.

Value is back-end identifier for the option and is what is saved to the database.

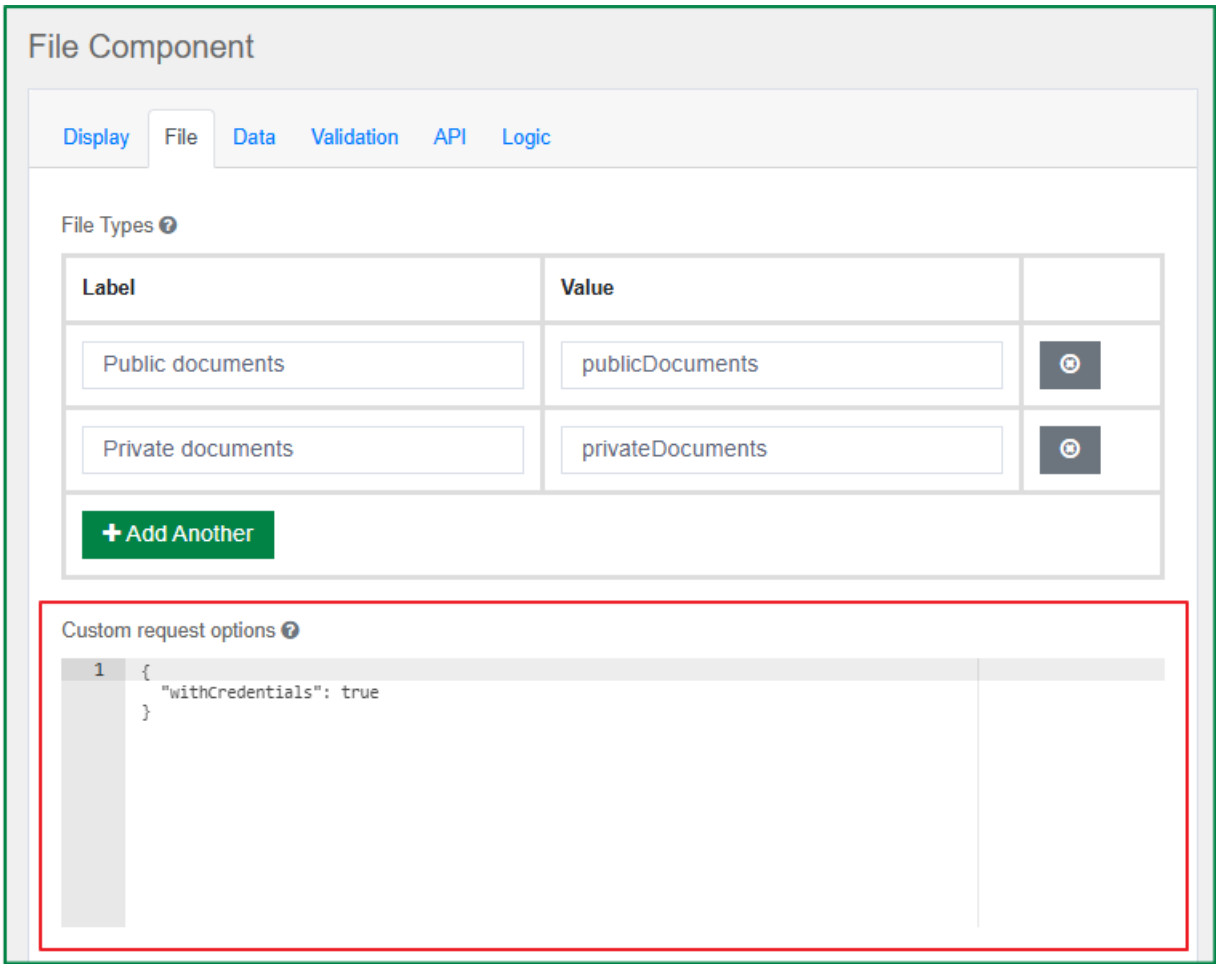


At the view of end-user, after uploading a file, the end-user can select a file type from the list of file types in the **Type** field.



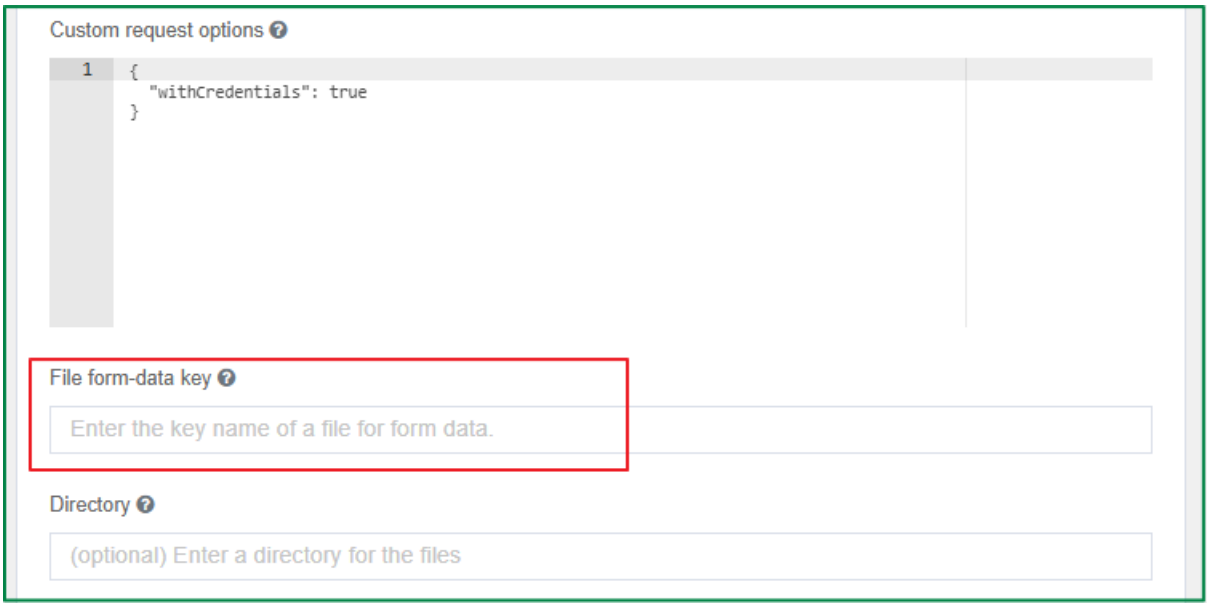
Custom request options

This option is displayed in the **File** tab. This option allows passing the custom xhr options.



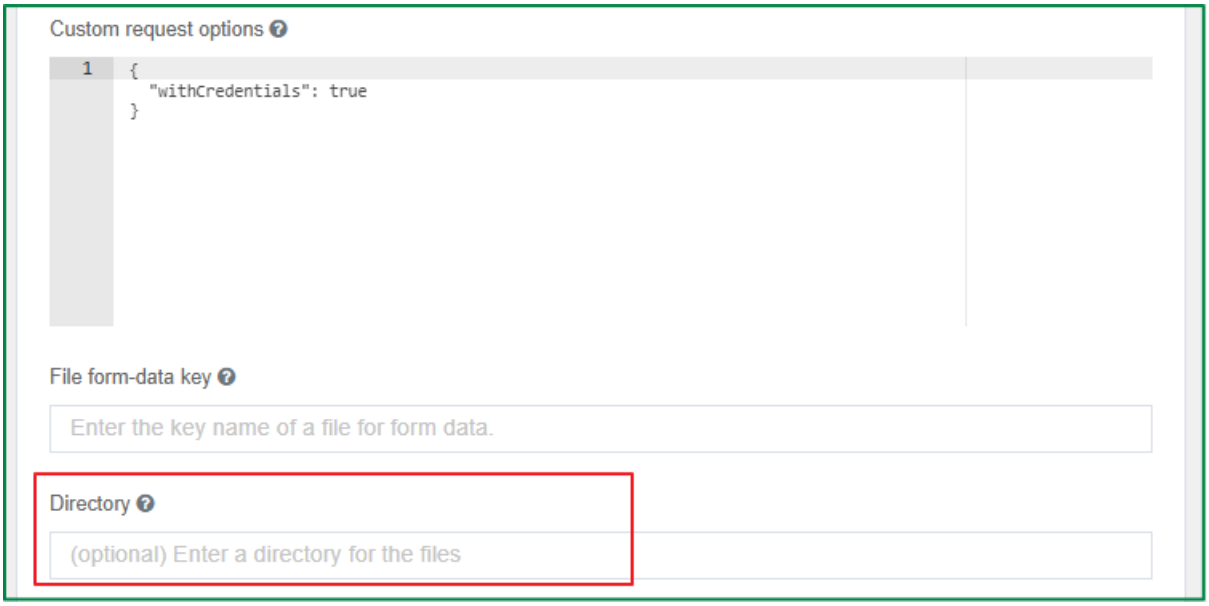
File form-data key

This option is displayed in the **File** tab. You can input a key name which is used to modify for the file while calling API request.



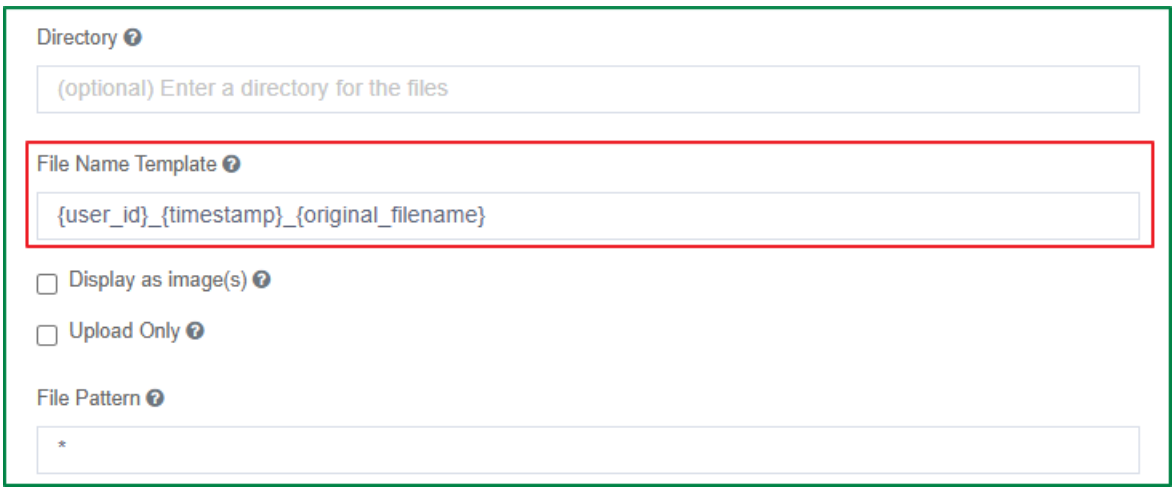
Directory

This option is displayed in the **File** tab. You can input a directory to place all the files which are uploaded in the File field.



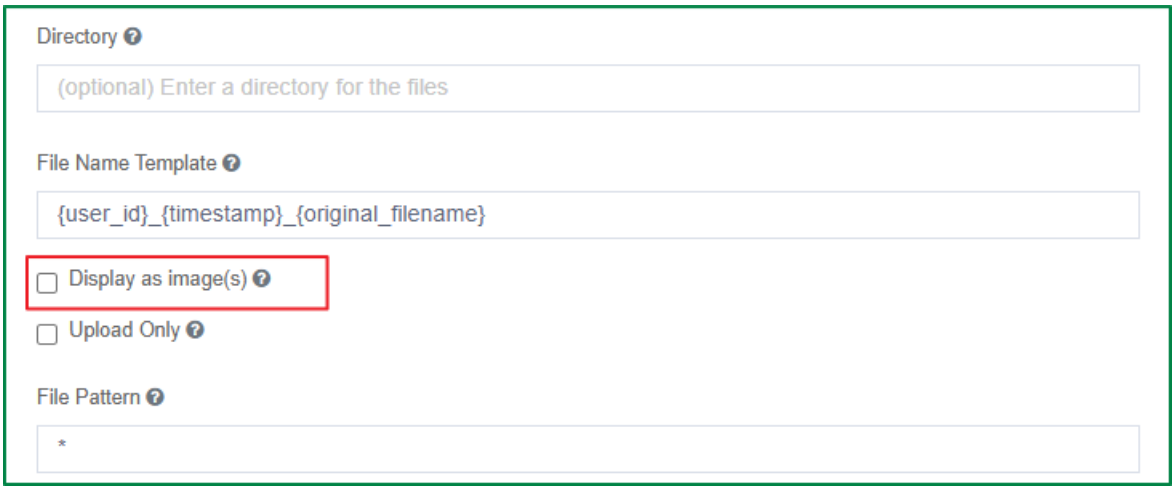
File Name Template

This option is displayed in the **File** tab. You can define a template for the name of the uploaded files. Regular template variables are available (`data`, `component`, `user`, `value`, `moment`, etc.). The variables `fileName and` `guid` are also available. The `guid` part must be present and will be added automatically if not found in the template.

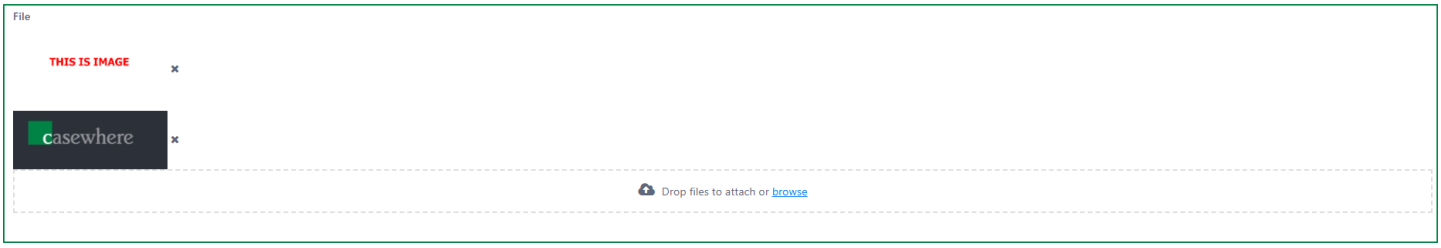


Display as image(s)

This option is displayed in the **File** tab. Tick on this checkbox if you want the uploaded files are displayed as images in view of the end-user instead of being displayed as a link. When this option is ticked, the end-user can only upload the image files.

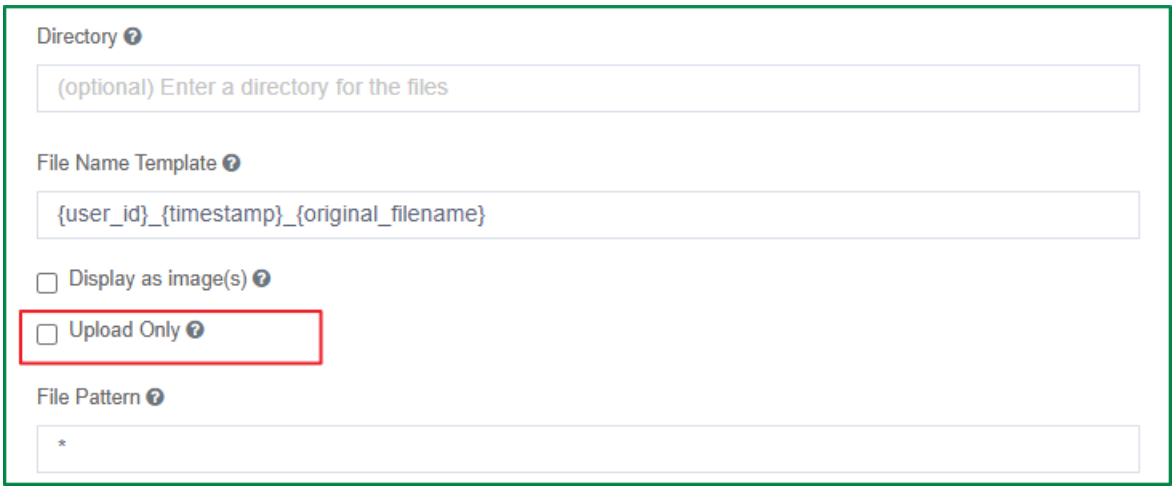


Following is the display of the uploaded images in view of the end-user.

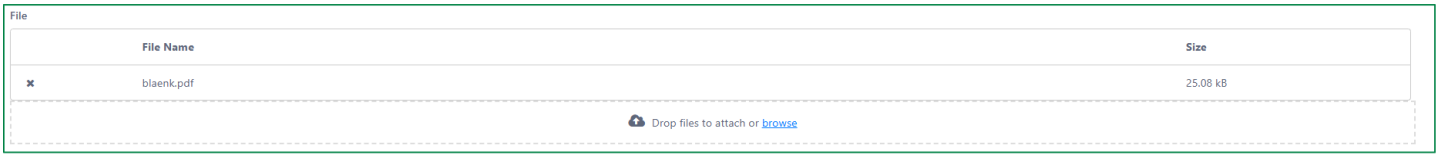


Upload Only

This option is displayed in the **File** tab. Tick this option if you want the end-user to be able to upload the files only. If this option is ticked, the file name of the uploaded file will be displayed as a plain text instead of a link, and the end-user cannot download the uploaded files.

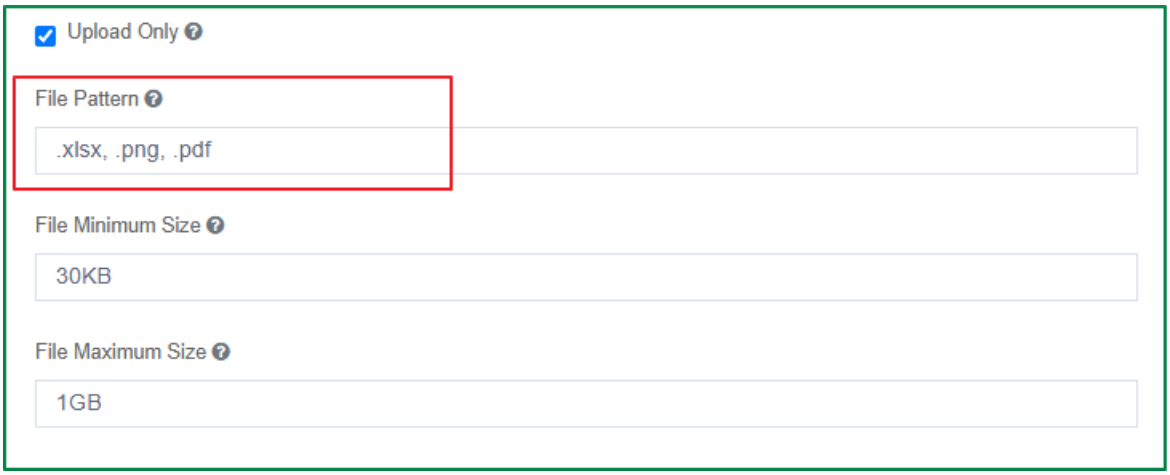


Following is the example of the File field which disallow downloading



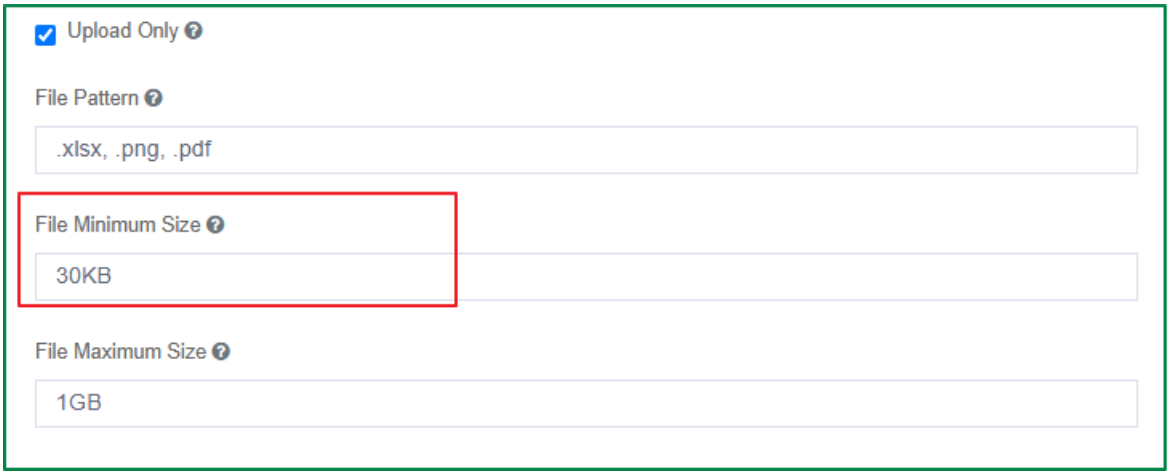
File Pattern

This option is displayed in the **File** tab. You can define the whitelisted file extensions. Only the file having the extension includes in the list can be uploaded. The file having the extension which is not defined in this field will be unable to be uploaded. Input ***** if you do not want file with any file extension can be uploaded.

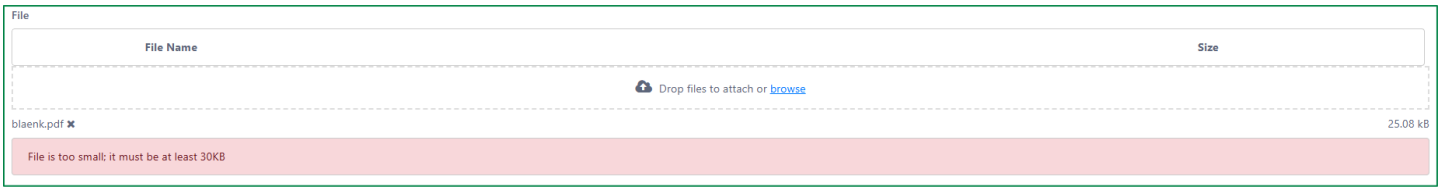


File Minimum Size

This option is displayed in the **File** tab. You can define the minimum size of the uploaded file.

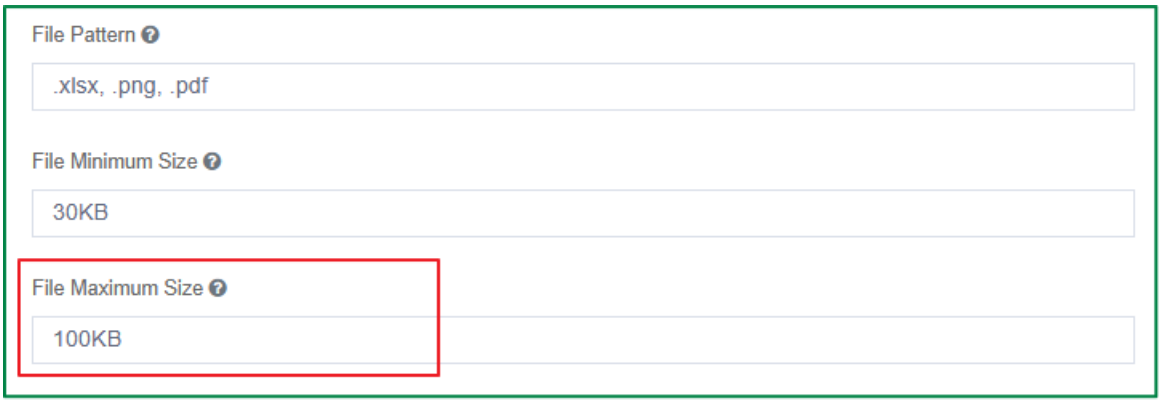


At the view of the end-user, if the uploaded file size is smaller than the minimum size, the file will not be uploaded and error message of too small file size will be displayed.

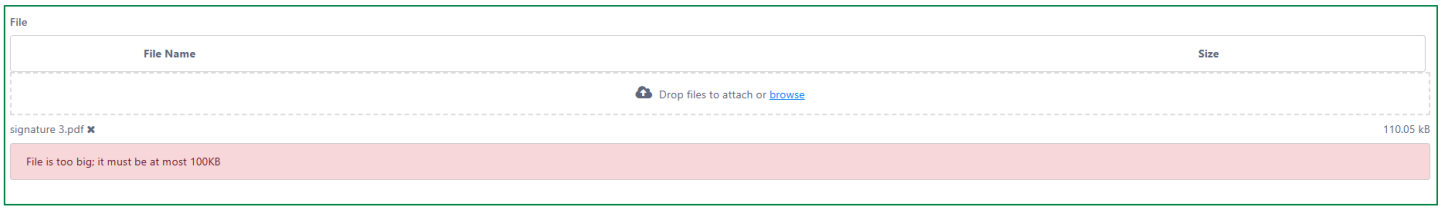


File Maximum Size

This option is displayed in the **File** tab. You can define the maximum size of the uploaded file.



At the view of the end-user, if the uploaded file size is greater than the maximum size, the file will not be uploaded and error message of too big file size will be displayed.



Custom component

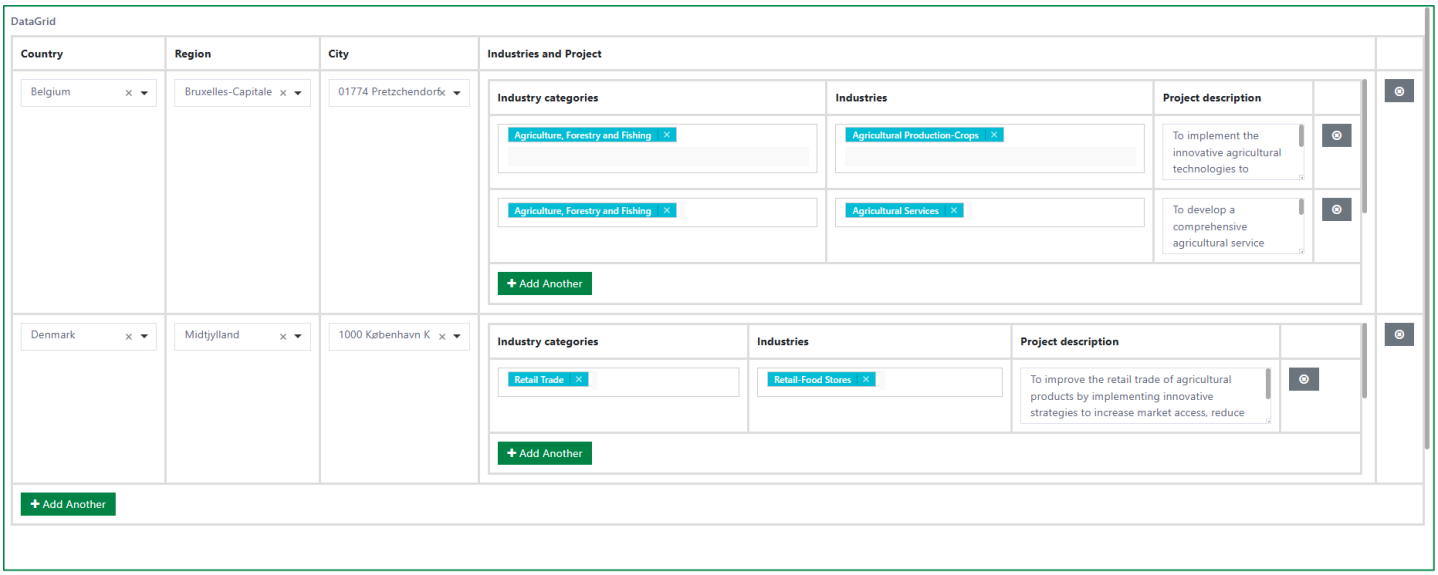
Read here

Data components

Data Grid

Data Grid component supports to add a group of components on a line item grid. After adding the **Data Grid** component to the form, you can drag and drop other form components into the grid. Each component will be displayed as a column in the grid. You can also add other data grid components into the grid. It is especially useful in case of needing the ability to add or duplicate multiple field sets.

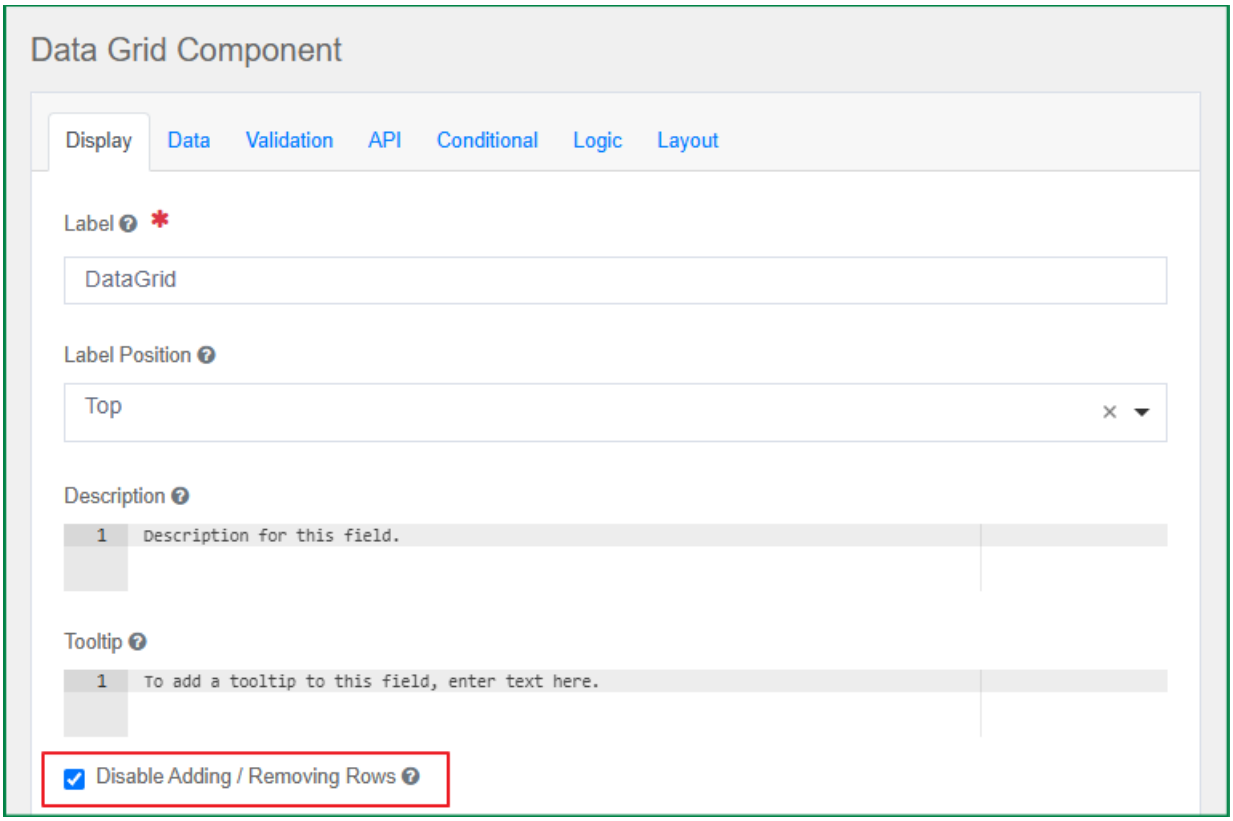
Here is the example of a **Data Grid** component which includes other components, and another **Data Grid** component at the view of end-user:



Following is the specific settings of the Data grid component.

Disable Adding / Removing Rows

This option is displayed in the **Display** tab. Tick this checkbox if you want to hide the **Add Another** button, and the **Remove Row** button.

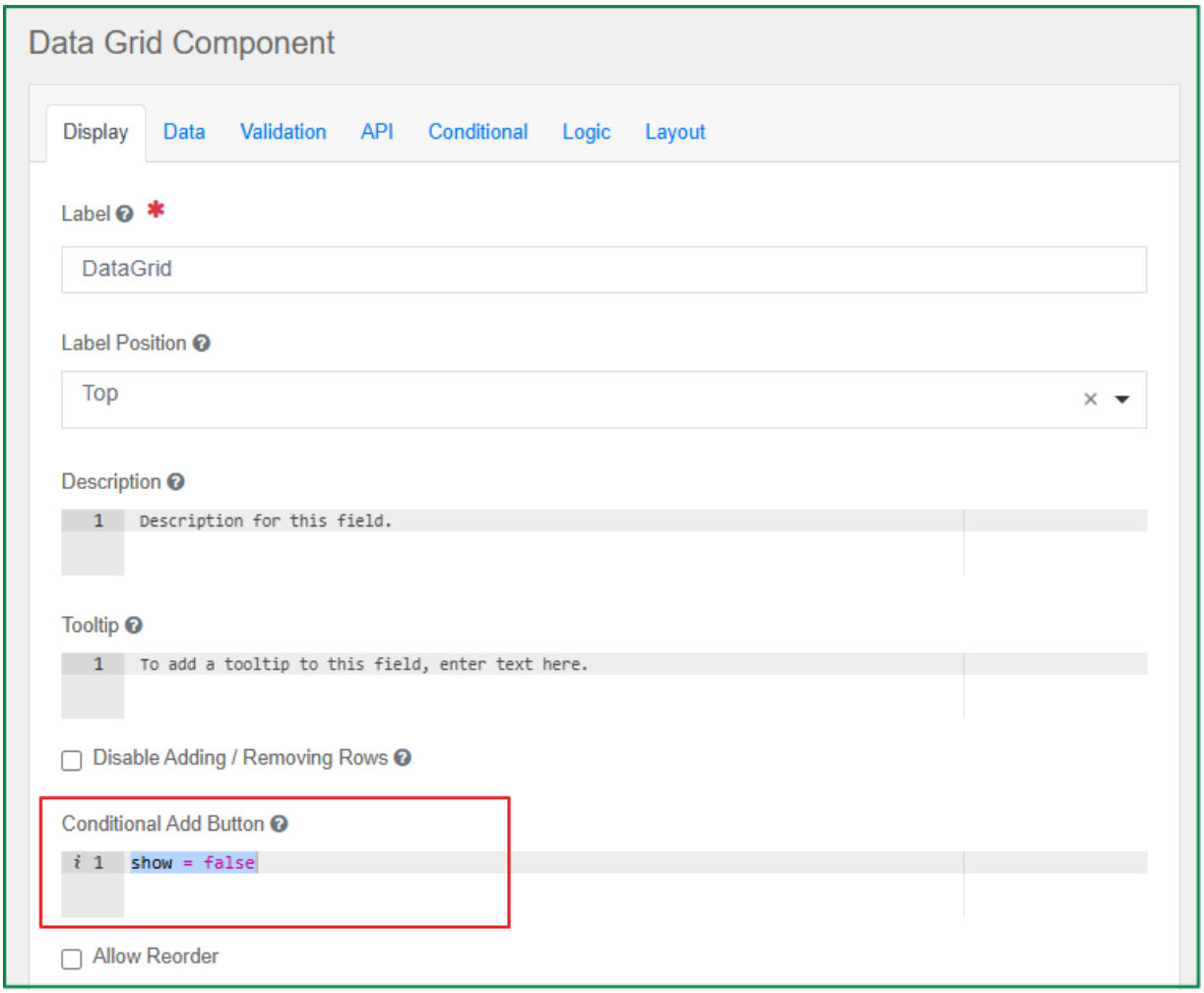


When this option is turned on, the data grid will include only 1 row, the end-user cannot add new row or remove any row.



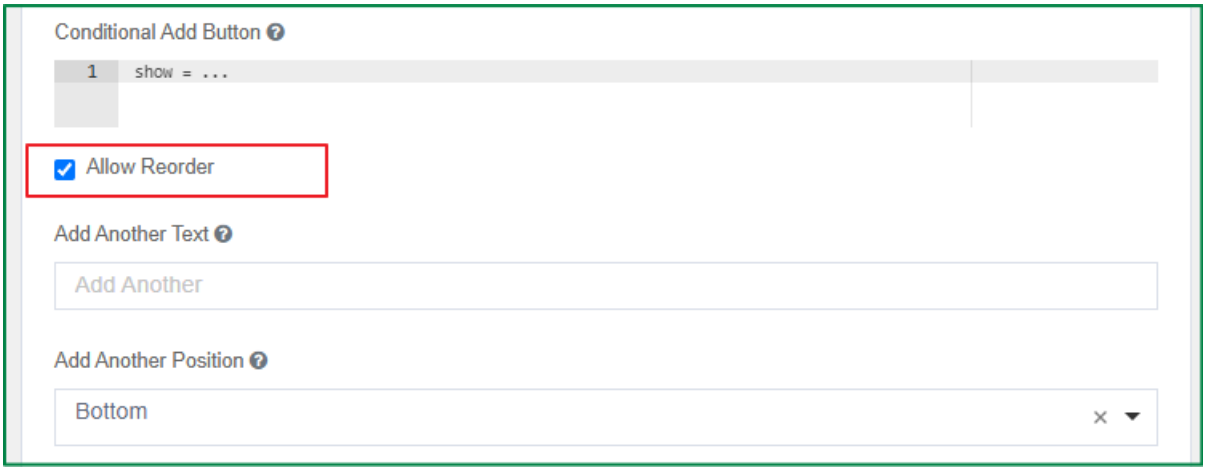
Conditional Add Button

This option is displayed in the **Display** tab. You can define the condition when the Add Another button should be displayed.



Allow Reorder

This option is displayed in the **Display** tab. Tick this checkbox if you want to allow the end-user to change the order of the rows in the grid.

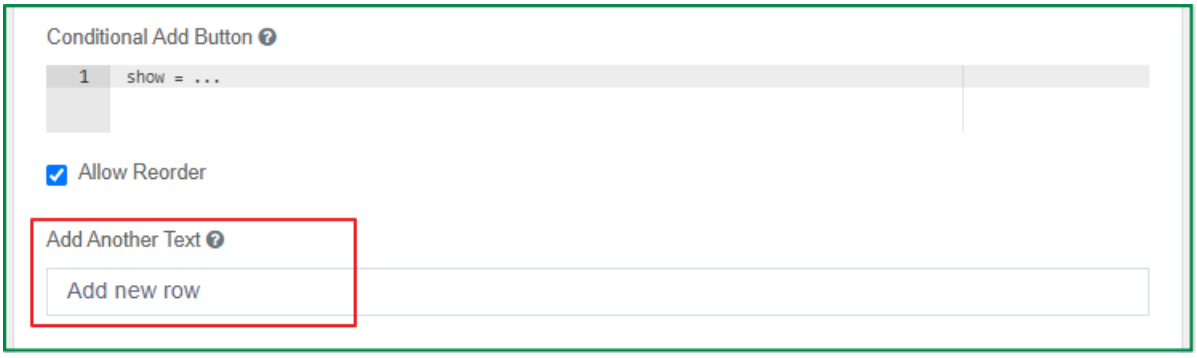


At the view of the end-user, the **Reorder** button will be displayed if this option is turned on.



Add Another Text

This option is displayed in the **Display** tab. This option is used to define the label for the button of adding row in the grid. By default, the button label is **Add Another**, you can define another label for this button.





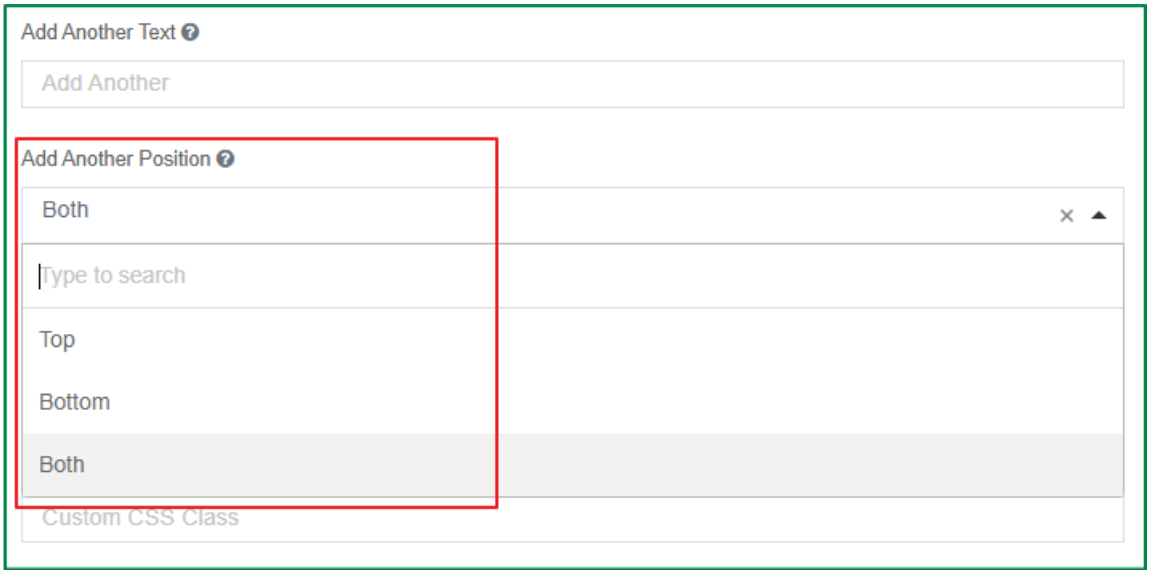
Add Another Position

This option is displayed in the **Display** tab. You can select the position of the button Add Another on the grid. It includes 3 positions: **Top**, **Bottom**, **Both**.

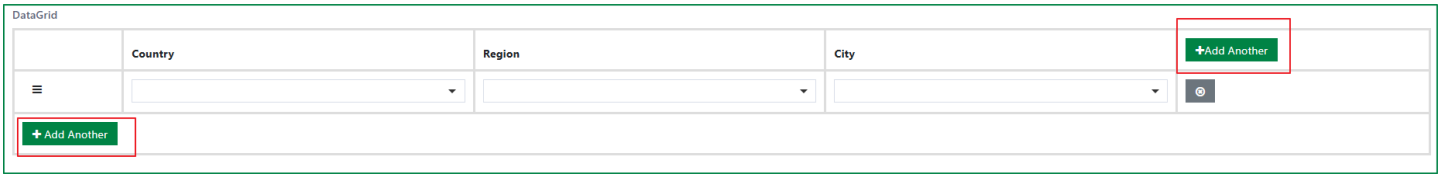
Top: the button will be displayed at the top-right corner of the grid.

Bottom: the button will be displayed at the bottom-left corner of the grid.

Both: 2 buttons will be displayed, one at the top-right corner, and another at the bottom-left corner of the grid.

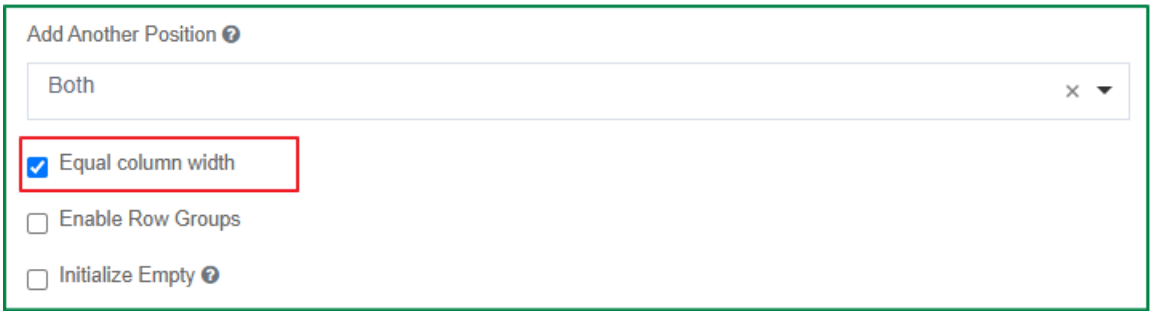


Following is the example of the display of the **Add Another** button in the grid in case the selected position is **Both**



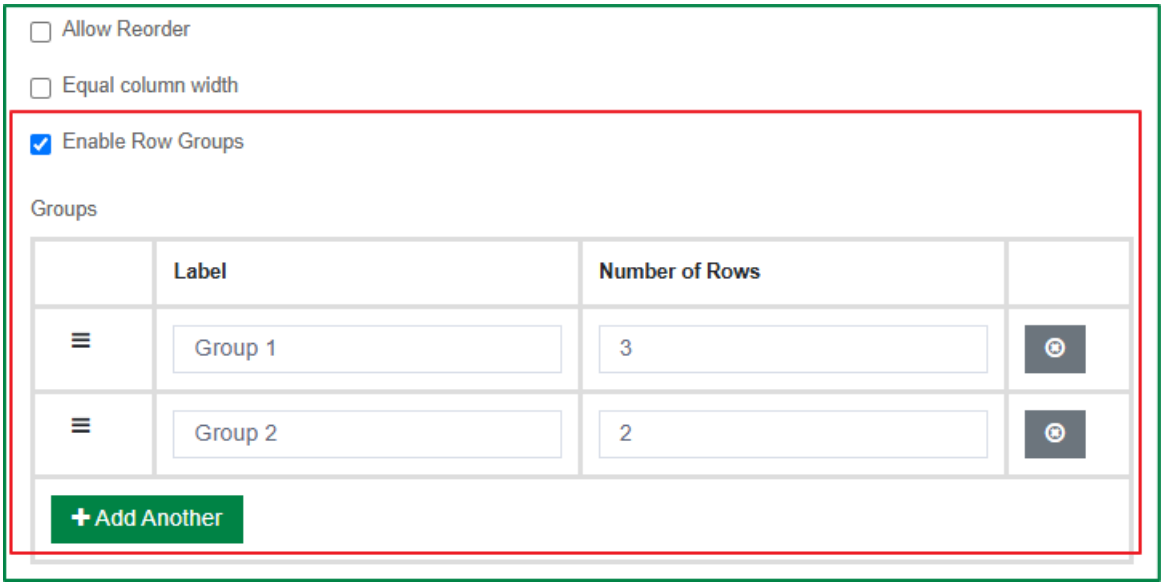
Equal column width

This option is displayed in the **Display** tab. Tick this checkbox if you want all columns in the grid have the equal width.

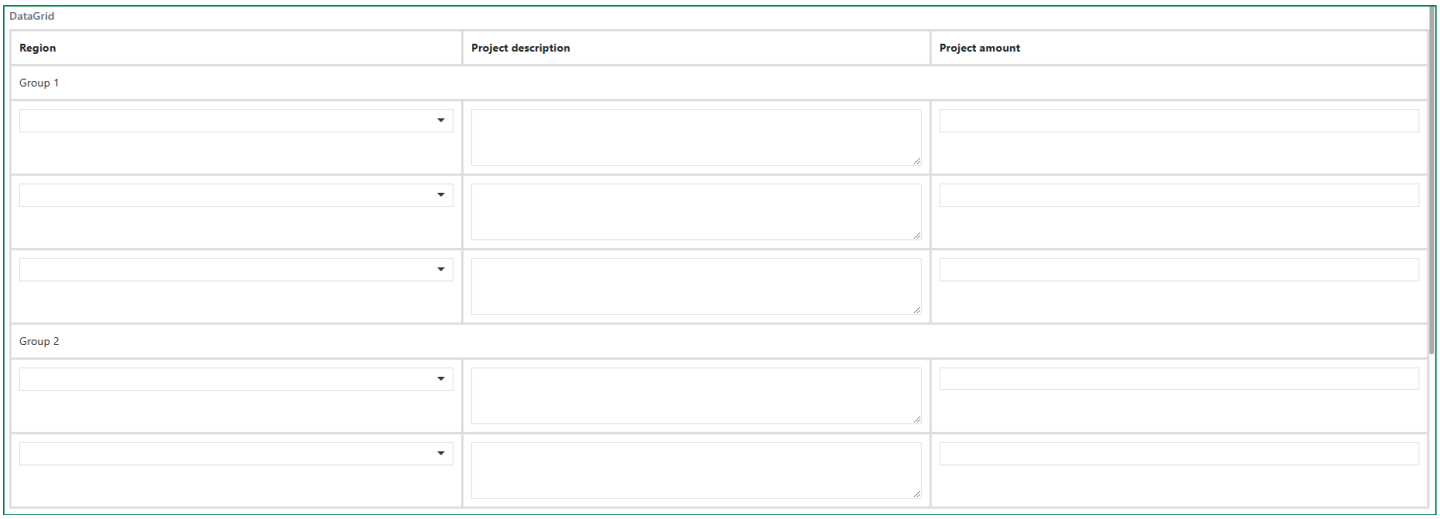


Enable Row Groups

This option is displayed in the **Display** tab. Tick this checkbox if you want to separate the grid into groups of row. When the checkbox is ticked, the field **Group** will be displayed and you can define the label of the groups, and number of the rows of each group.

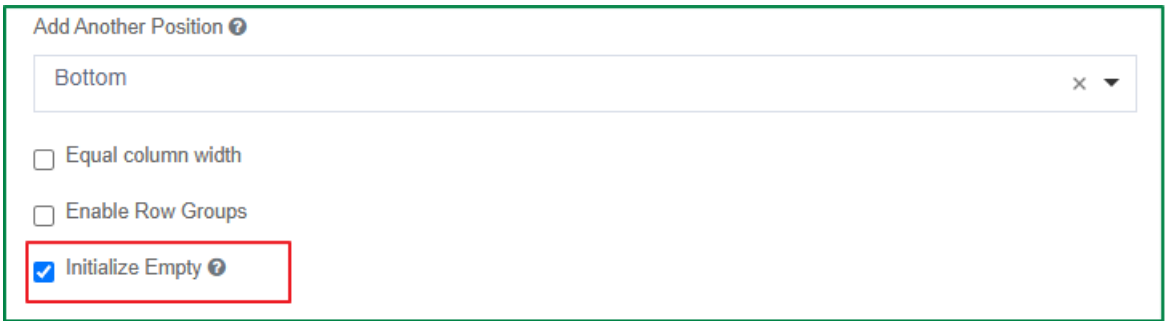


If this option is turned on, the buttons **Add Another** and **Removing Row** will be disabled automatically, it means the number of rows is fixed, and the end-user cannot add or remove any row.



Initialize Empty

This option is displayed in the **Display** tab. Tick this checkbox if you want the Data grid is displayed without any rows initially.


Add Another Position
Bottom
Equal column width
Enable Row Groups
Initialize Empty


DataGrid
Region
Project description
Project amount
+ Add Another

Application: Form management - Custom component

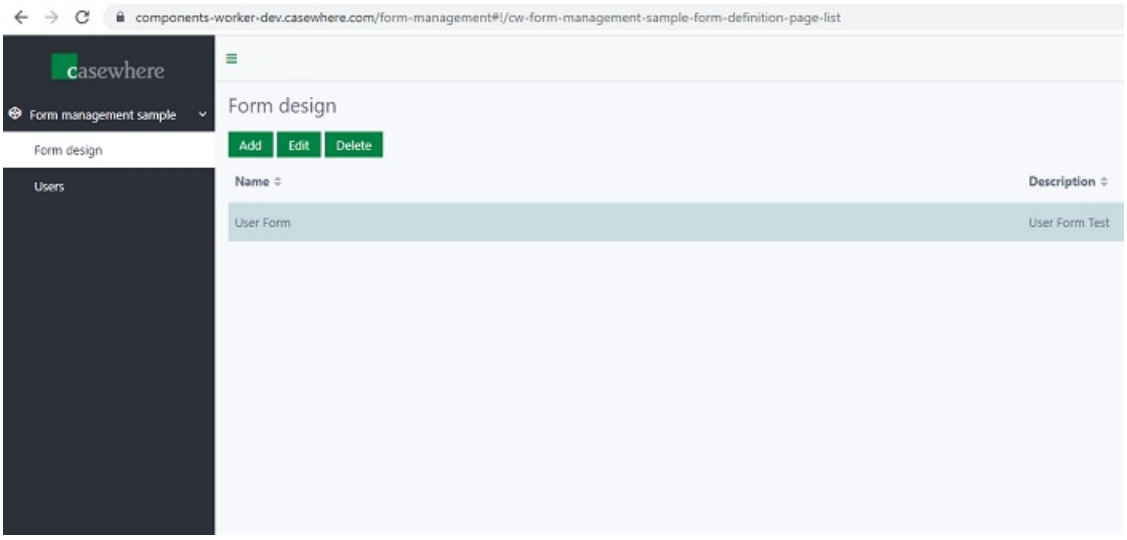
Introduction

One of the more powerful features of the FormIO platform is the ability to create your own custom form components. The process of creating a custom component involves extending a Base class of the component that is "closest" to the implementation you desire, and then override methods or introduce new methods that will implement your custom logic. On the other hand, for many features in the real project, we need some resources from Casewhere system. Thus, the custom component which can be connected between FormIO component and Casewhere resources (pages, widgets...) is the best solution for us. In conclusion, this proof of concept (POC) will help you learn to integrate the custom component of FormIO library into the Casewhere system.

Features

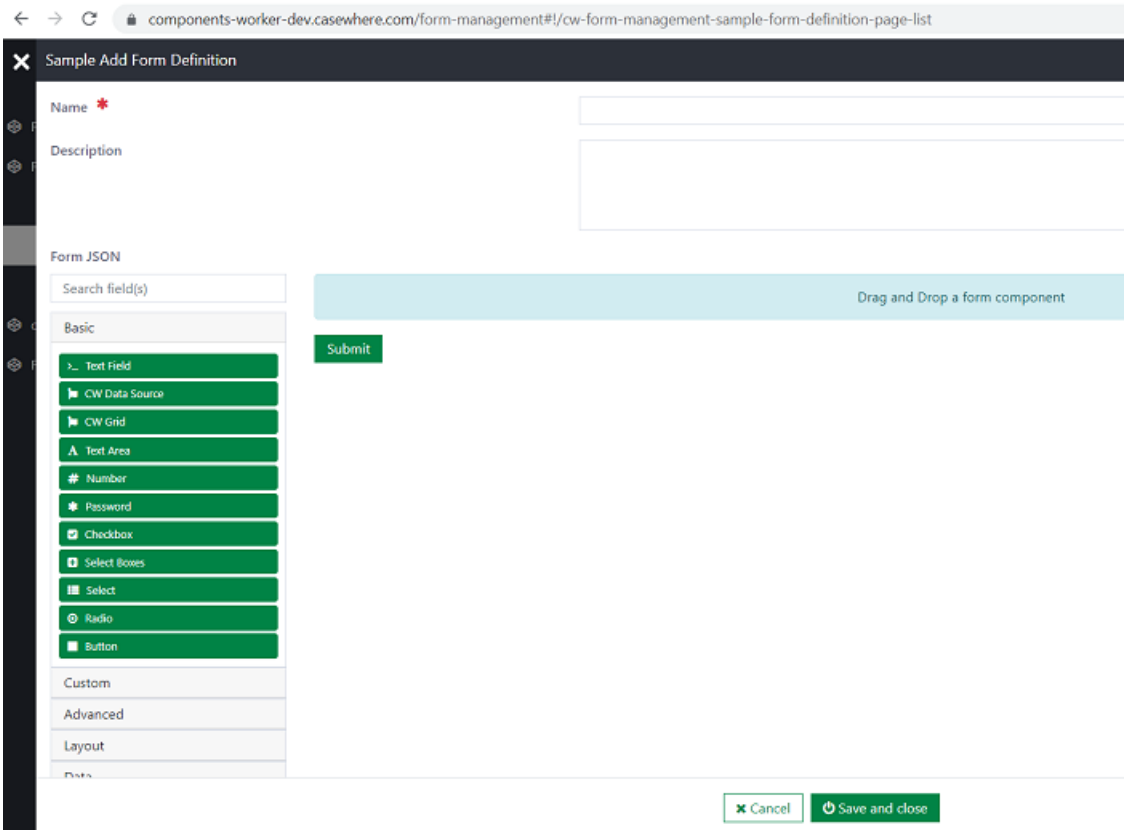
Manage form design

Form design page is used to manage the FormIO design. In the **Form design list** page, you can view all the created forms, also add, edit, delete data.



Form design:

- Name : Required
- Description: Optional
- Form JSON: Optional



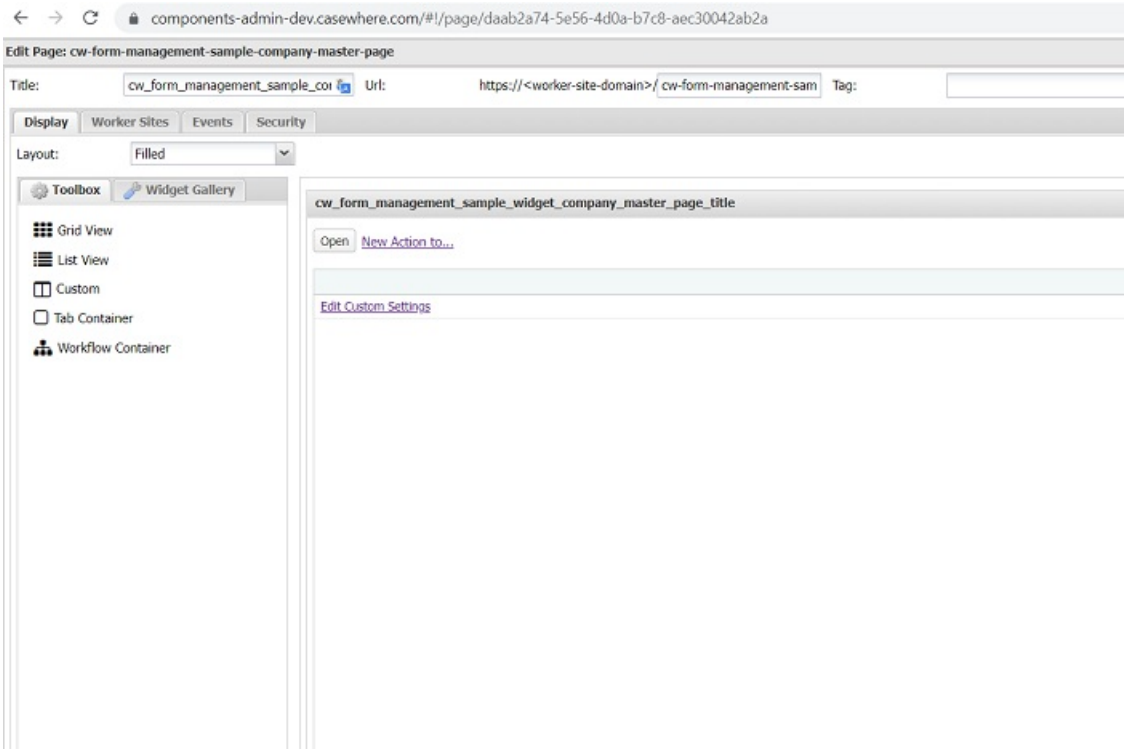
Example

Company custom component

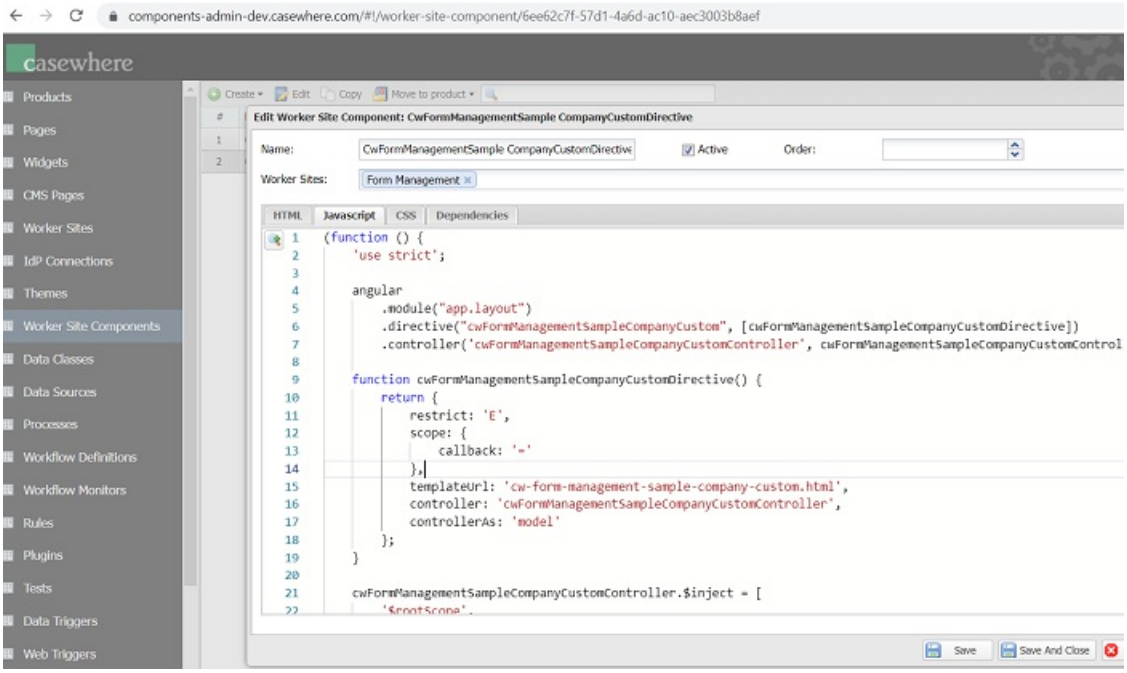
The following is an example custom component. This is a company dropdown component that allows you to configure a component to have a cw_Widget in the cw_Page popup which can be selected company in the dropdown to save the data as a string to database.

There are three steps to creating a FormIO custom component in Casewhere system:

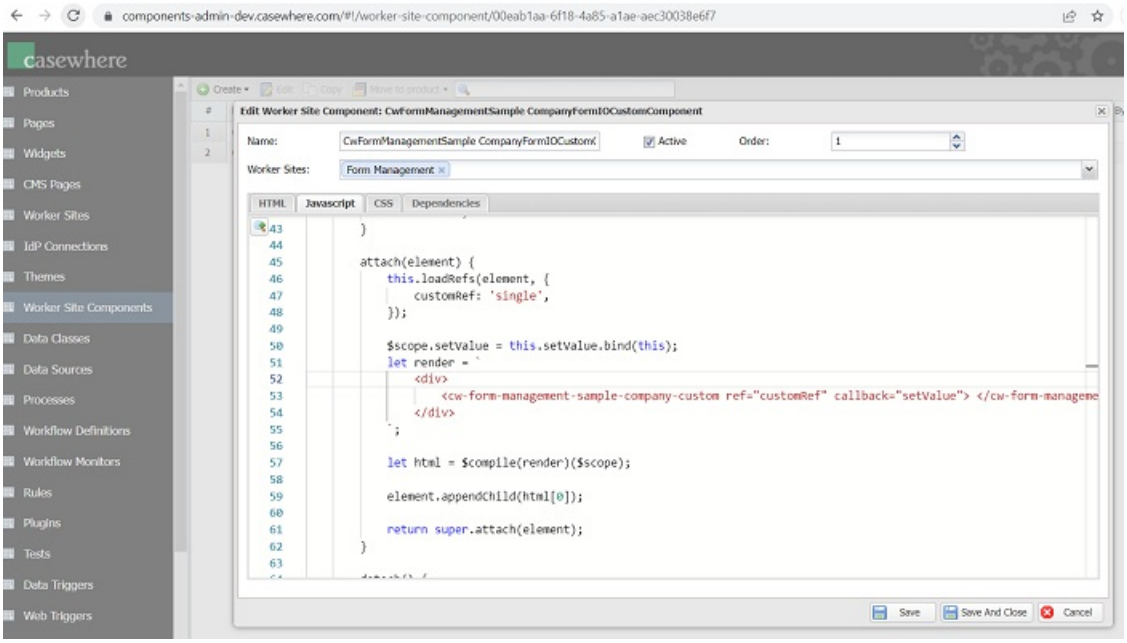
Step 1: Create a Casewhere page, this page is the content of custom component.



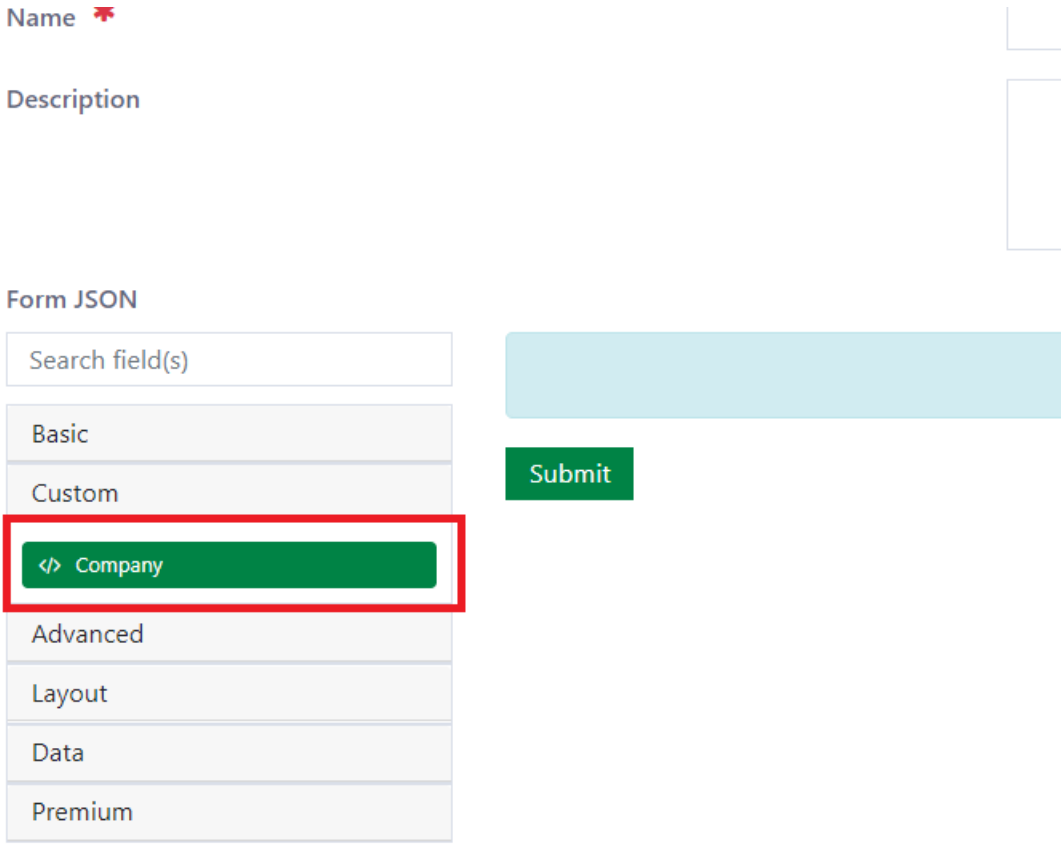
Step 2: You need to create a directive worker site component and include the created page above in this component.



Step 3: Create a FormIO custom component and attach the directive in step 2 to the element.



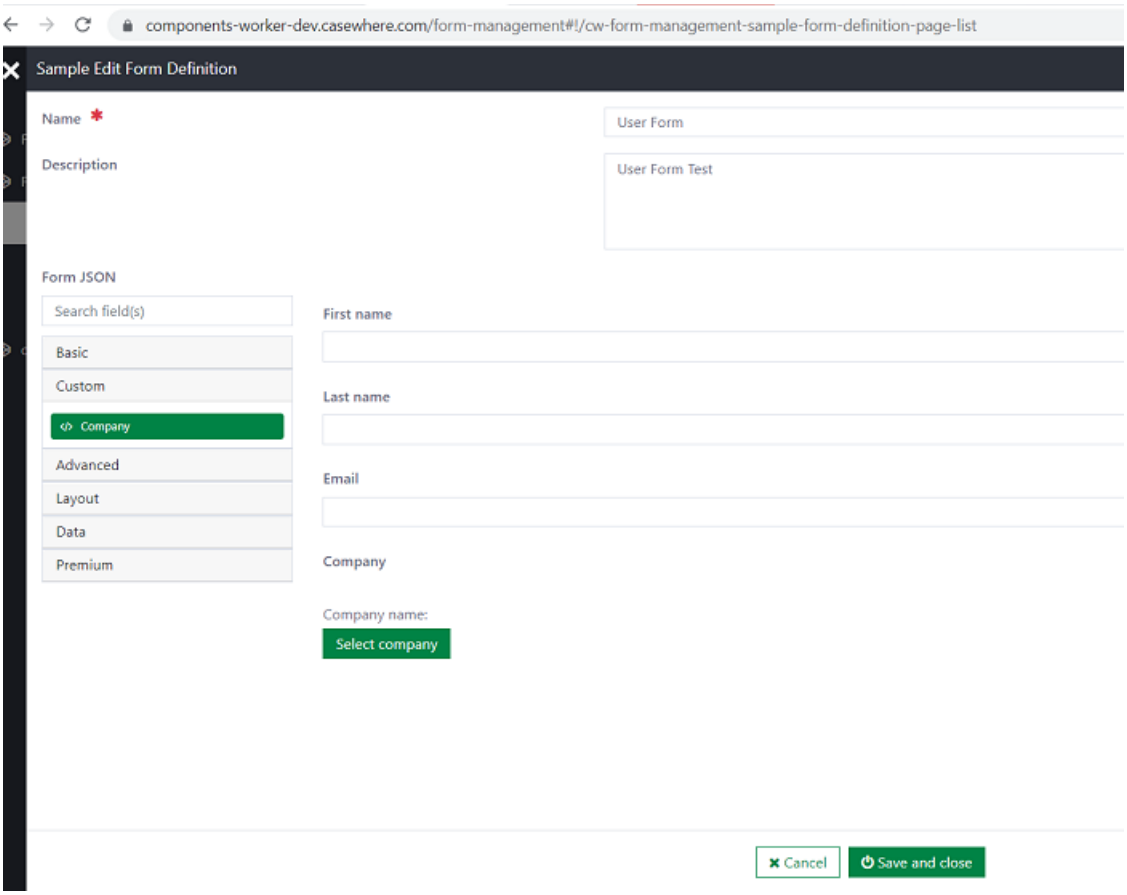
Then, the new custom component will appear in the FormIO builder.



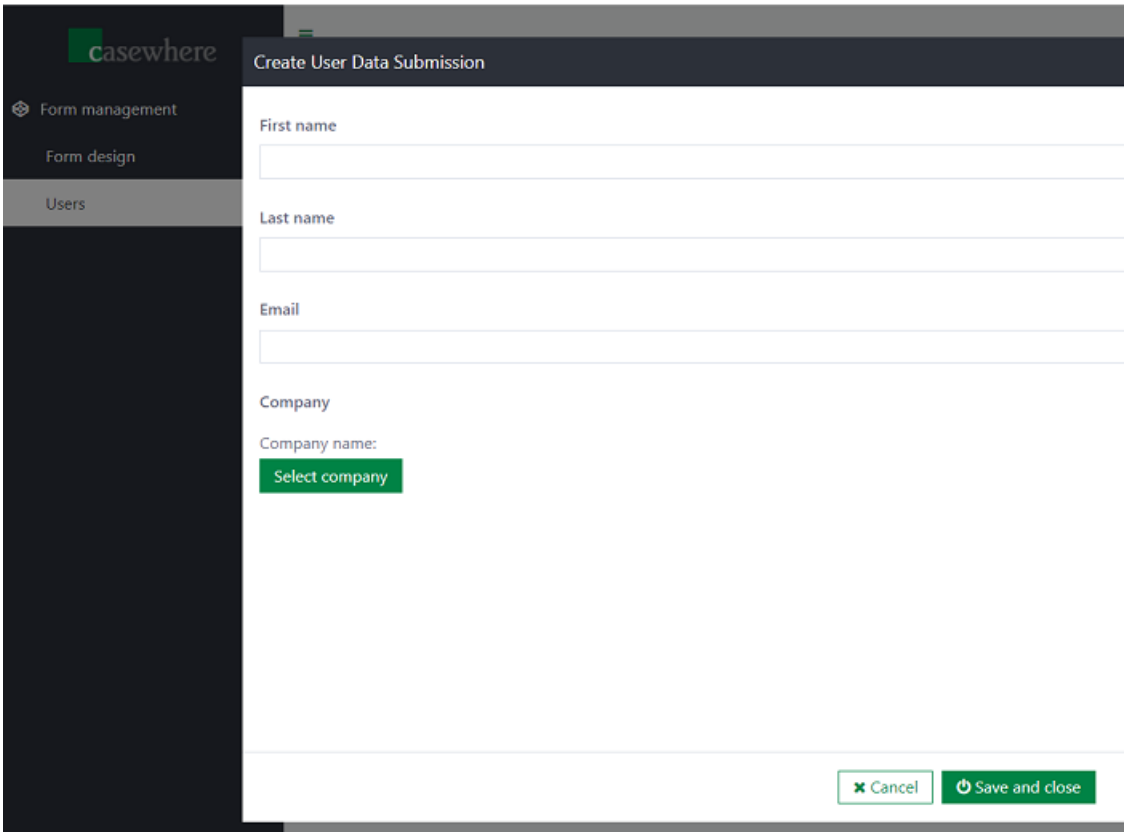
Make a simple User Form with the company custom component

First, create **User Form** in the form design page

- Name : User Form
- Form JSON:
  - First name (Text Field)
    - Label: First name
    - Property Name: FirstName
  - Last name (Text Field)
    - Label: Last name
    - Property Name: LastName
  - Email (Email)
    - Label: Email
    - Property Name: Email
  - Company (Custom)
    - Label: Company
    - Property Name: Company



Then, open **Users** page, now you can add a user with your design



Installation

Requirements

- Casewhere 2.7.1 or later.

Configuration

1. Import the product
2. Link the page to your worker sites and configure access control
3. Create the User Form design

Dependencies

- Application: Form Management

Releases

1.0.0 - 01/07/2022

Changelog

- Custom component FormIO.

Download (login required) Casewhere Form Management Sample

Component: Application Changelog

Introduction

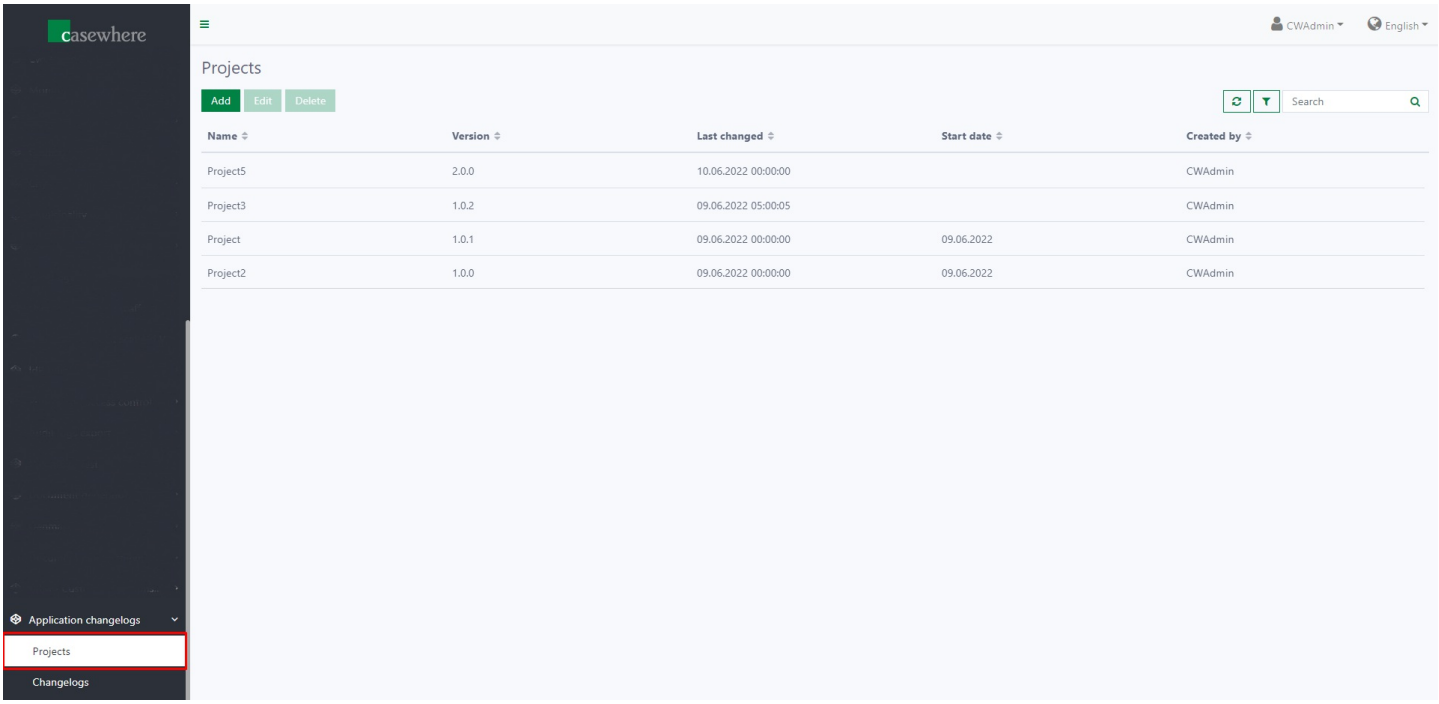
The Application changelog component is a component that allows the administrator to manage changelogs of projects of an application. It helps to reduce the difficulty of the version management. In addition, the changes of the projects are easily traced back.

Features

Projects

This section describes how to manage projects.

The administrator can handle the changes and versions of multiple projects. To view and manipulate projects, go the **Projects** page, below is a screenshot of the page.



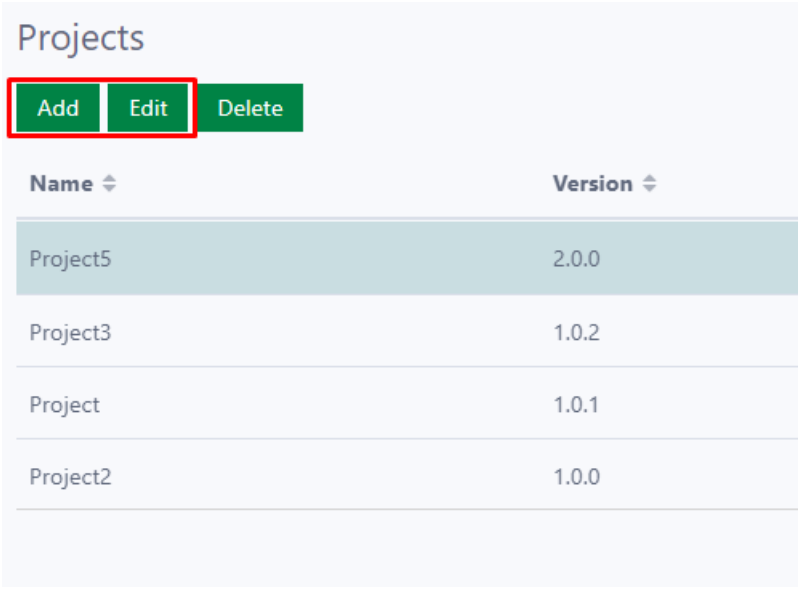
Project List

On the page, you can see a list of all created projects with almost full of information:

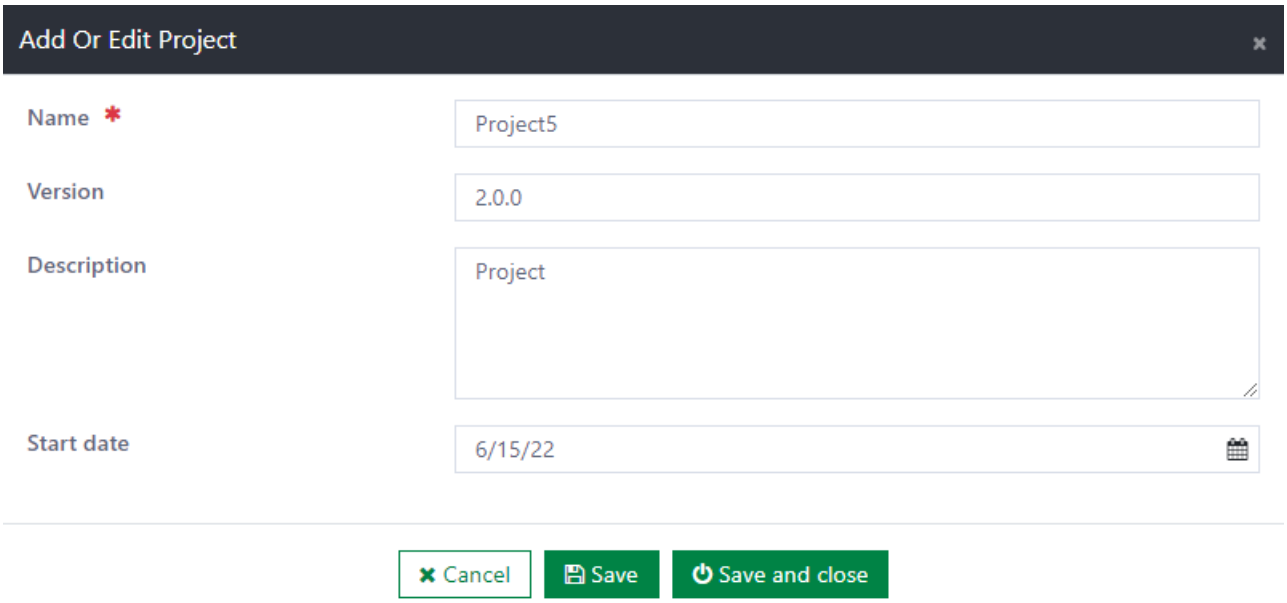
- **Name**: The name of the project
- **Version**: Latest version of the project
- **Last changed**: The timestamp that the latest change was made
- **Start date**: The date that the project starts
- **Created by**: The name of the user that creates the project

Add or Edit a Project

Above the list, you can find the **Add** and **Edit** buttons.



Clicking on either button will trigger the **Add Or Edit Project** dialog to show. Here is a screenshot of the dialog:

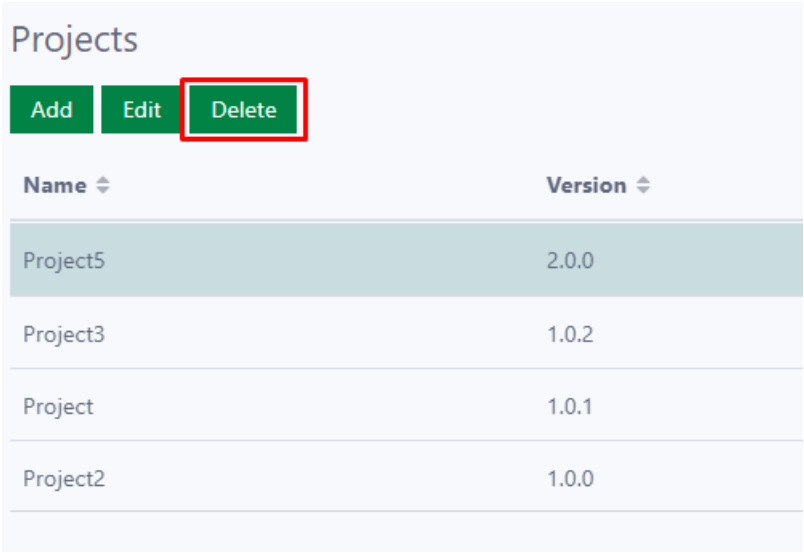


On this dialog, you can input the information of the project that you are working on:

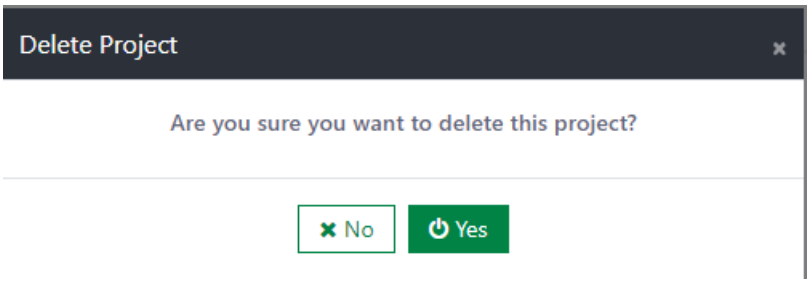
- **Name** *[required]*: You must input a representative name of the project. Project's name must be unique.
- **Version** *[optional]*: You can enter a version for the project. Otherwise, the component will fill in an initial version of 1.0.0.
- **Description** *[optional]*: You can write anything you want in this field to describe the project.
- **Start date** *[optional]*: You can input the date that the project starts.

Delete a Project

To delete a project, select it on the list and click on the **Delete** button.



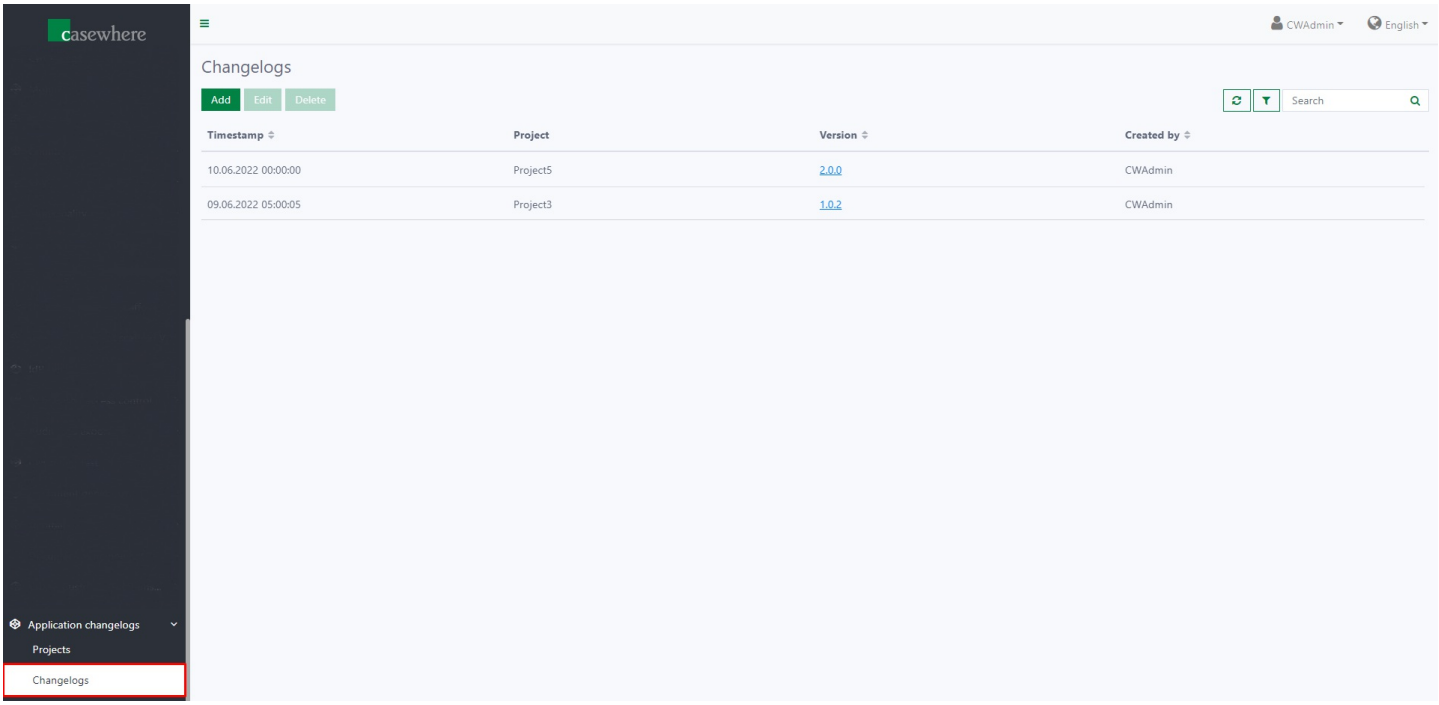
A dialog is shown to ask for confirmation. If you want to proceed, select **Yes**, else, click on **No** button.



Changelogs

This section shows you how to manage changelogs.

To work with changelogs, go the **Changelogs** page, below is a screenshot of the page.



Changelog List

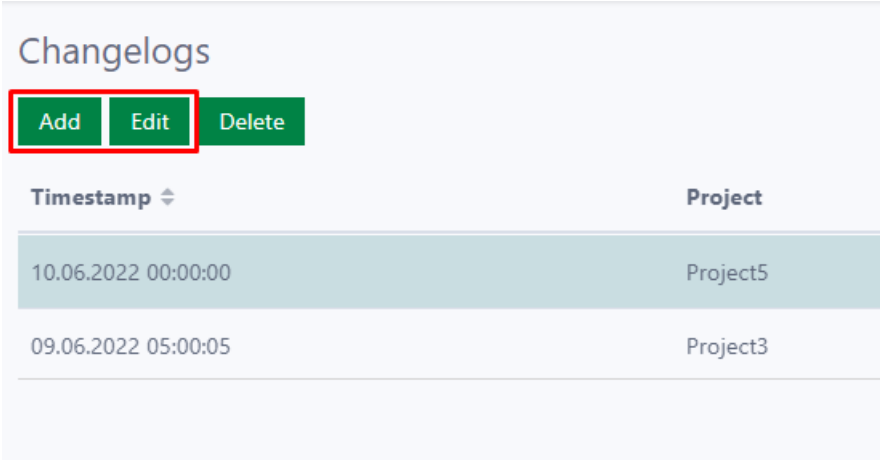
All changelogs are enumerated on the list with these information:

- **Timestamp**: The timestamp that the change takes effect on the project
- **Project**: Name of of the project to which the change belongs

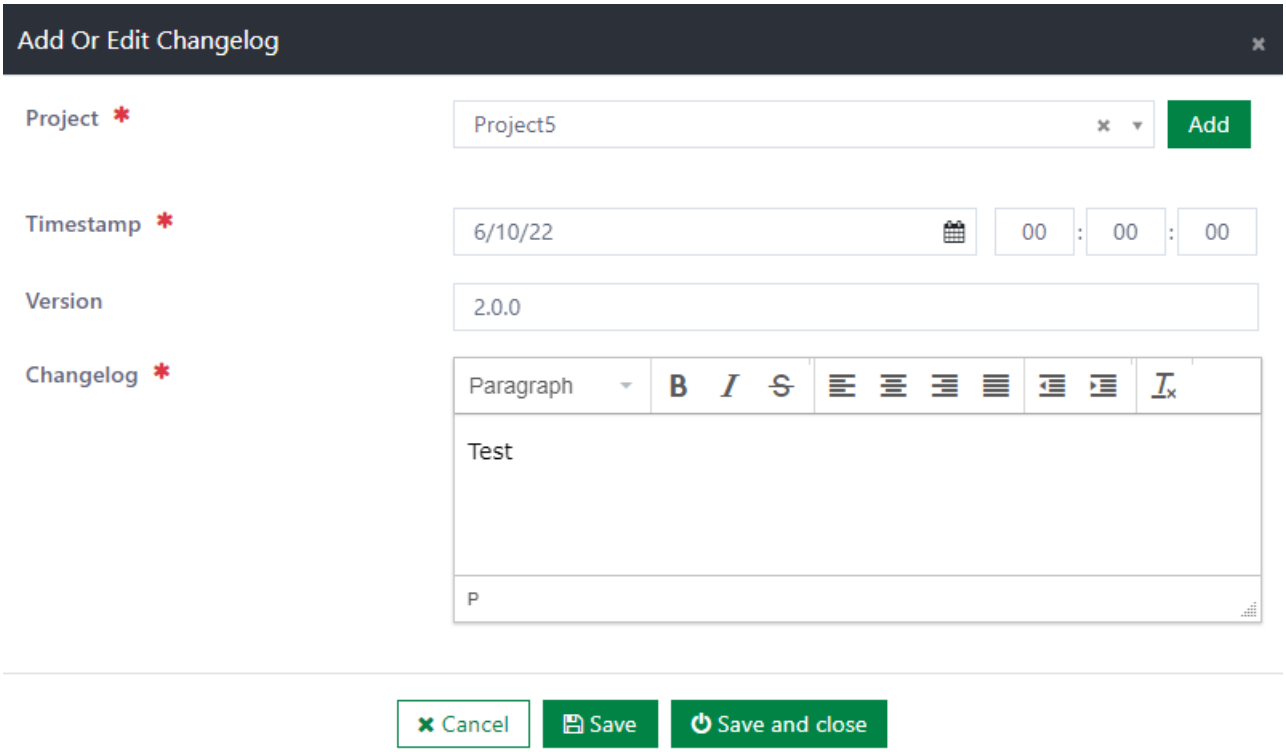
- **Version**: The version which is shipped with the changelog
- **Created by**: The name of the user that creates the changelog

Add or Edit a Changelog

In the same manner as on the **Projects** page, the **Add** and **Edit** buttons are also located above the list.



The **Add Or Edit Changelog** dialog will appear as either button is clicked. Here is a screenshot of the dialog:



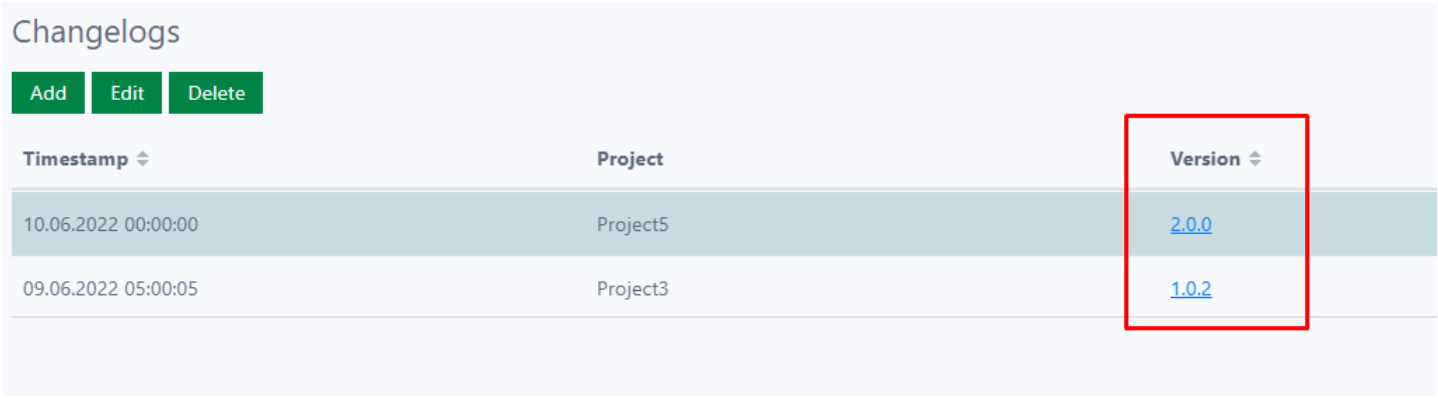
Here is the information that you can input on the dialog:

- **Project** *[required]*: You must select the project to which the changelog belongs. The projects on the list are created projects. If you cannot find the desired project, you can click on the **Add** button in order to add a new project.
- **Timestamp** *[required]*: This input is initialized with the value of the current time.
- **Version** *[optional]*: You can enter a version or the component will fill in an initial version of 1.0.0.
- **Changelog** *[required]*: The content of the changelog. The component shows a toolbox to format the content.

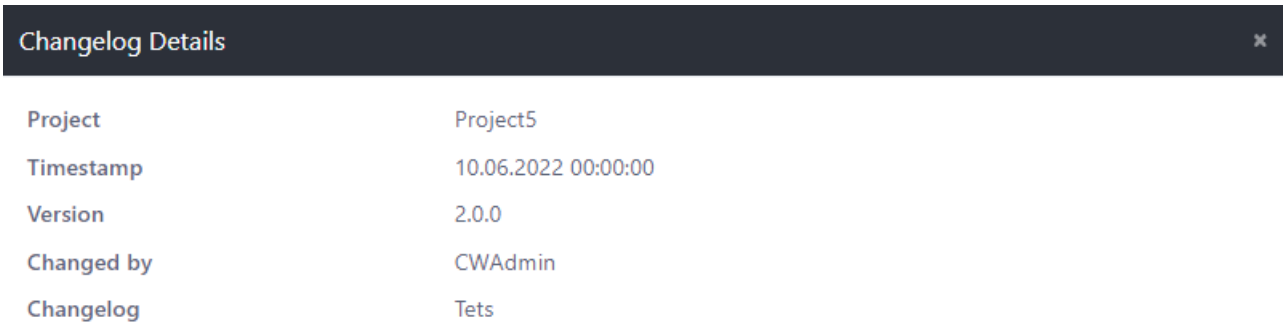
When a changelog is saved, the component automatically updates the version and the last-changed timestamp of the related project.

View Changelog Details

If you just want to view the details of a specific changelog, you can click on the version text of a changelog.

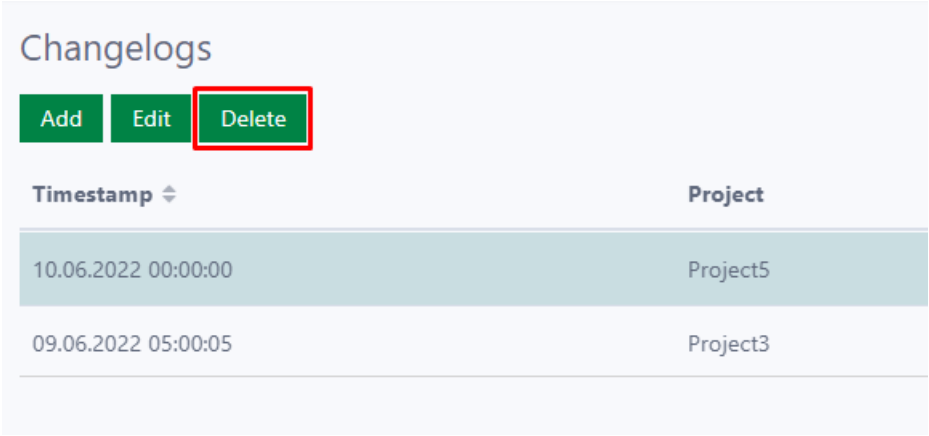


Here is the dialog that displays the details of a changelog:

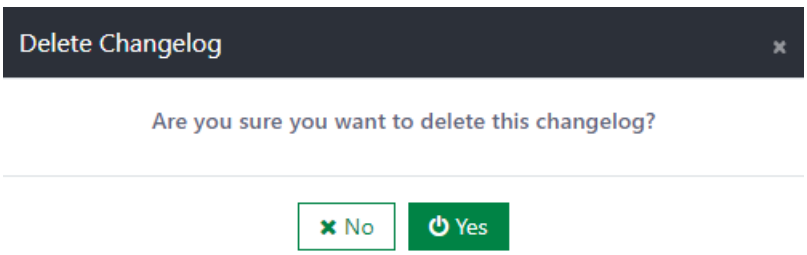


Delete a Changelog

Select a changelog on the list and click on the **Delete** button.



A dialog is shown to ask for confirmation. If you want to proceed, select **Yes**, else, click on **No** button.



Integration - Use The Information On An Application

The component exposes a web trigger that allows retrieving the information of a specific project. Here are the details of the web trigger:

- **URL**: *[Casewhere API URL]*/external/v0.1/trigger/cw-chlog-get-project-information
- **Method**: GET

- **Parameters**:
  - project_name: The name of the project you want to get. It is case-sensitive
  - fields: Fields that you expect the API to return. Fields are separated by a comma (should be encoded if needed). For example: name,version

Example:

```
curl "https://components-api-dev.casewhere.com/external/v0.1/trigger/cw-chlog-get-project-information?project_name=Project&fields=start_date,name"
Response: {"StartDate":"2022-06-08T22:00:00Z","Name":"Project"}
```

Moreover, the component provides two AngularJs components which allow developers to easily show the version on worker sites.

The first one only displays the version of a specific project:

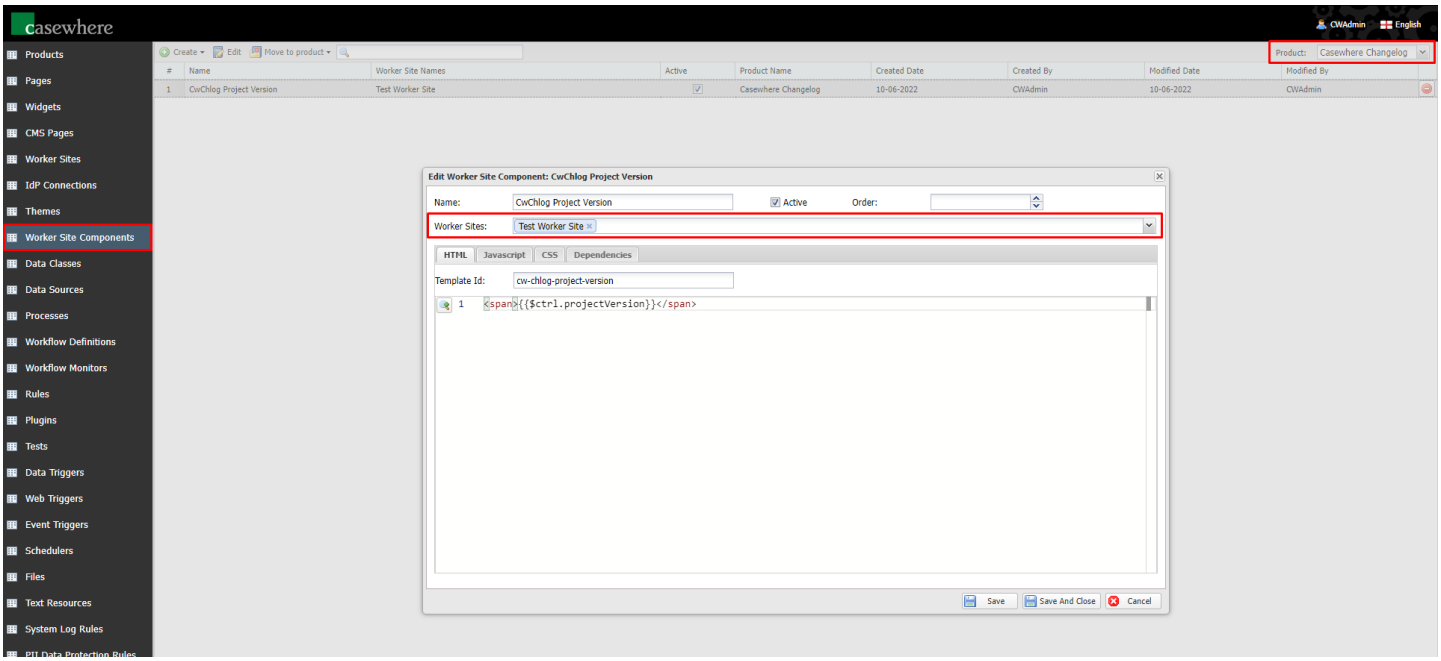
```
<cw-chlog-project-version project-name="Project" />
```

The next one displays the version of Casewhere platform and the version of a specific project:

```
<cw-chlog-product-version project-name="Project" />
```

The text from this AngularJs component follows "<Casewhere version> - <Project version>" format. For example, 2.7.2 - 1.34.0

In order to use these AngularJs components, you must add your worker site to them by going to the Worker Site Components and edit the components like the image below:



Please notice that both AngularJs components are inline HTML elements.

Installation

Requirements

- Casewhere 2.6.0 or later.

Configuration

1. Import **Casewhere Changelog** product
2. Link the *cw-chlog-projects*, *cw-chlog-changelogs* and *cw-chlog-changelog-details* pages to your worker sites and configure access control

3. Configure projects
4. Configure changelogs

Dependencies

No dependencies required

Releases

1.0.0 - 10/06/2022

Changelog

- Projects manager
- Changelogs manager
- Project - changelogs synchronization
- A web trigger to get information of a specific project
- An AngularJs component to display the project's version

Download (login required): Casewhere Changelog v1.0.0

Component: Audit Log Export

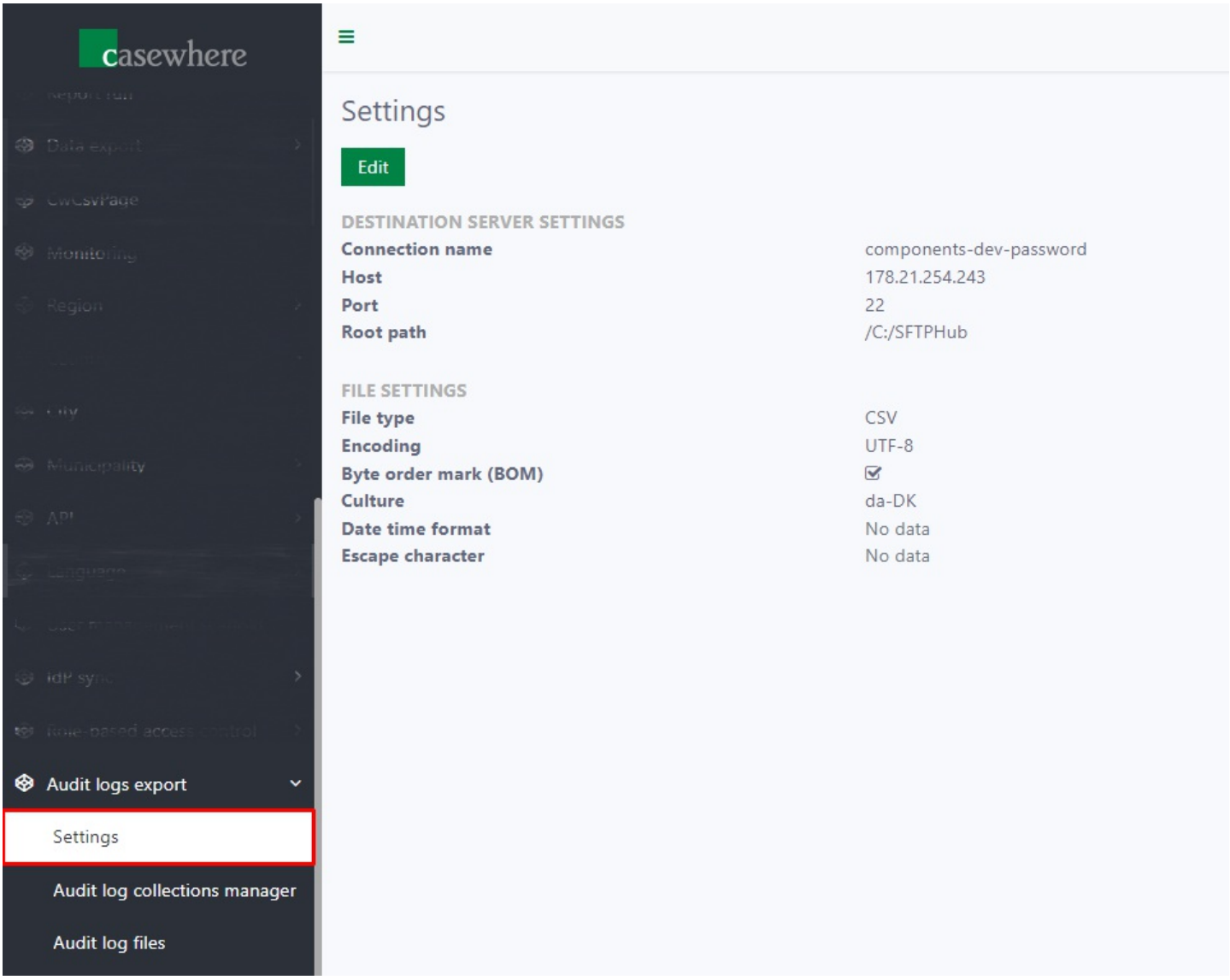
Introduction

Audit Log Export component is a component that exports audit log data periodically. Which data can be exported and how it is represented in the output file can be configured flexibly. The version 1.0.0 only supports CSV output file but it is designed to be extendable.

Features

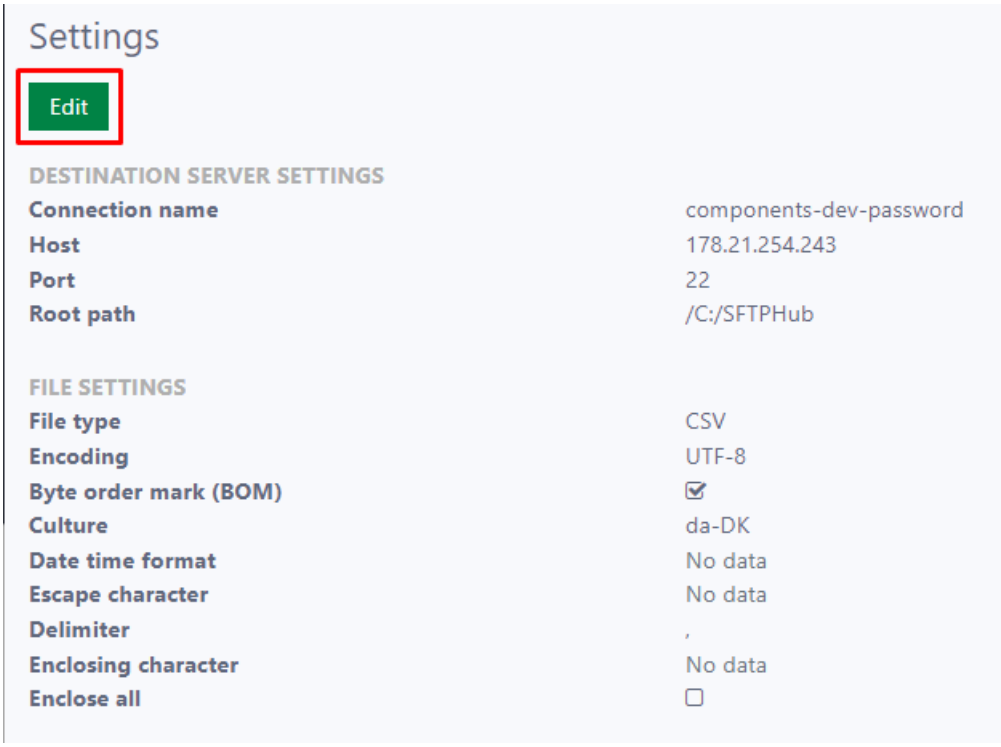
Settings

Settings page is used to manage general settings of the components, such as connection and output file. The settings will be automatically created if it does not exist.

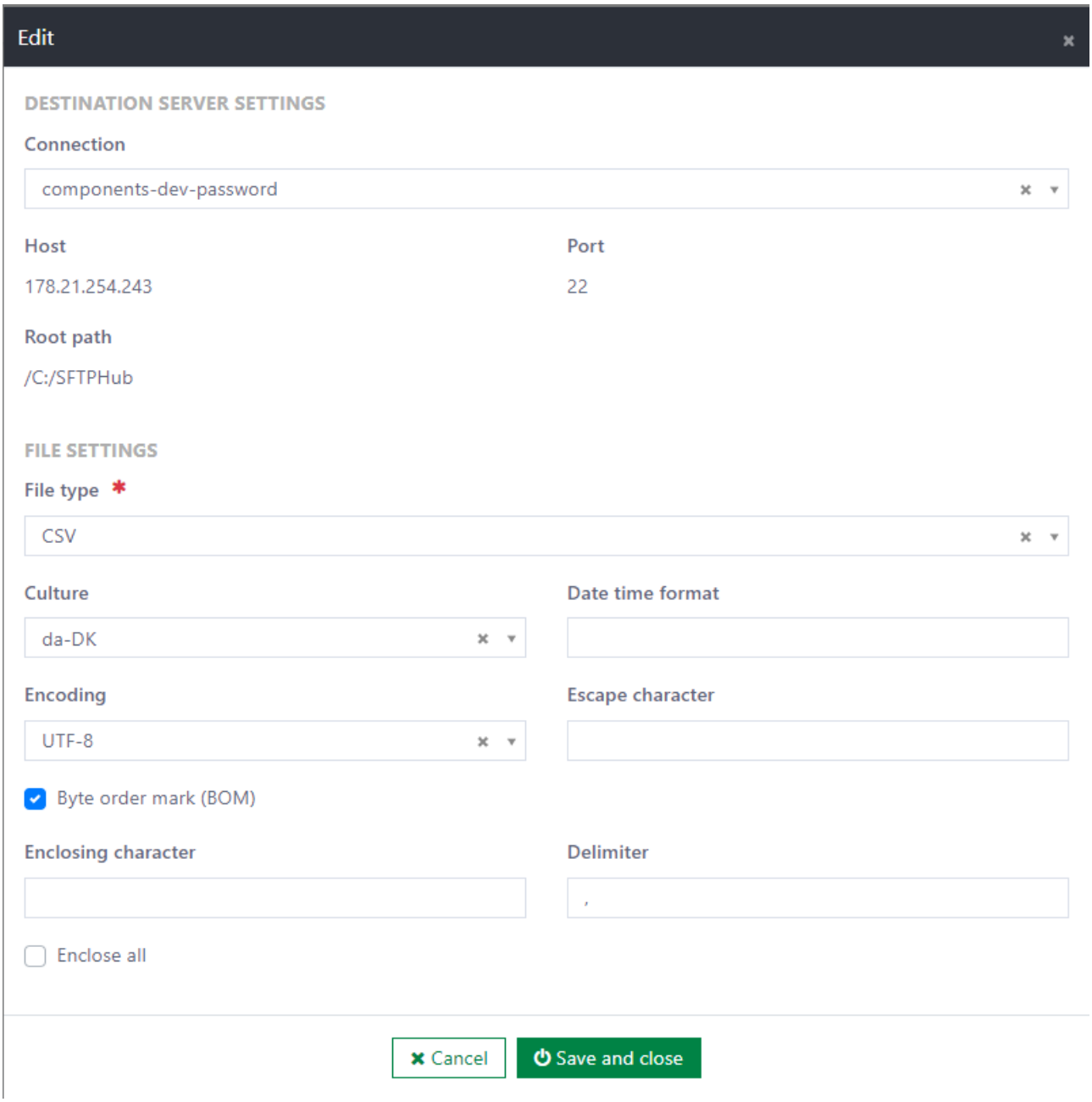


Edit Settings

To edit the settings, click on the **Edit** button.



The dialog of Edit Settings will be opened. On the dialog, you can see two sections:



General settings

SFTP Connection: This setting is required. You must select one SFTP connection from the dropdown list on the dialog.

When an **SFTP connection** is selected, the information of the corresponding **Host**, **Port**, and **Root path** will be shown.

Note: Read SFTP hub to learn how to set up SFTP connection

File settings Below are settings for the output file. These settings, when are set, will override the settings of the plugin which handles the output file.

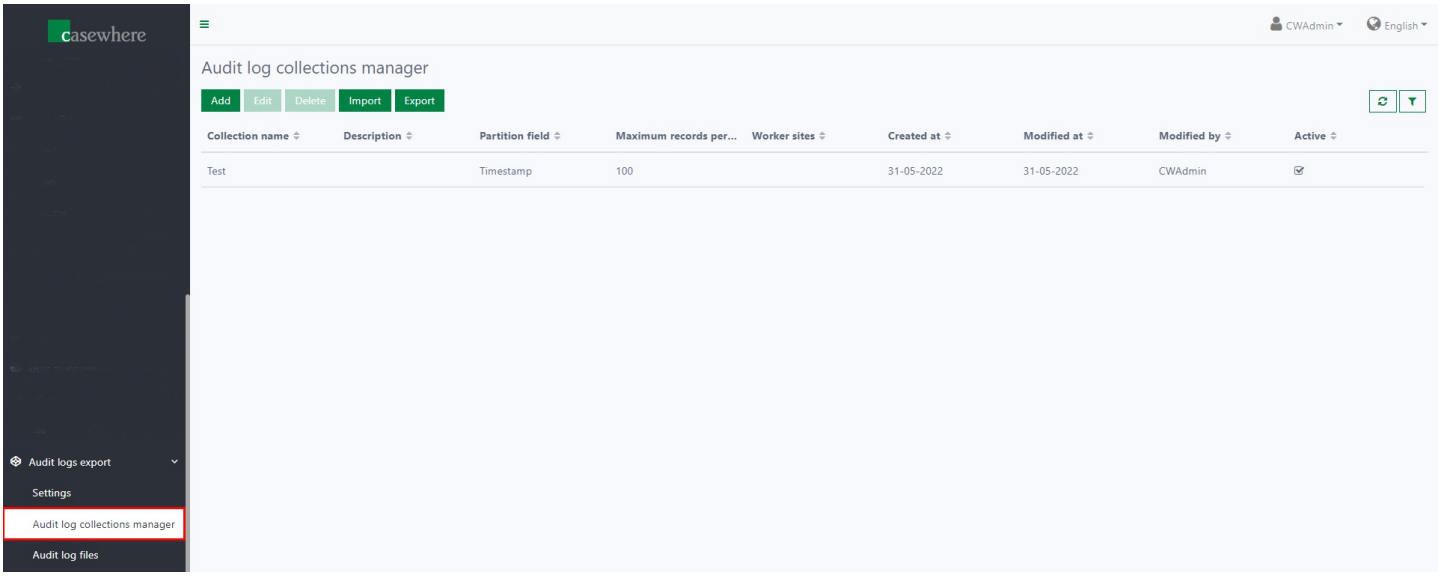
- **File type** *[required]*: You can choose the type of the output file via this setting. CSV is the only option in the current version of the component.
- **Culture** *[optional]*: It is the culture information of the output file which affects formats of some values in the file. The default value is the current culture of the web server.
- **Date time format** *[optional]*: Values which are date time will follow this format setting. If the setting is unset, the short date format of the set or current culture will be used.
- **Encoding** *[optional]*: This setting specifies how the content of the file is encoded. The default value is UTF-8.
- **Byte order mark (BOM)**: The checkbox indicates whether the BOM should be inserted to the file or not. It also depends on the encoding setting. If the used encoding does not support BOM then this setting is ignored. By default, the BOM is disabled.

- **Escape character** *[optional]*: If it is defined, the defined character will be escaped in the content of the file.
- **Enclosing character** *[optional] [For CSV only]*: If any value in the file needs to be enclosed, the component will use this defined character to enclose the value.
- **Enclose all** *[For CSV only]*: With this setting enabled, the component will enclose all values in the content of the file.
- **Delimiter** *[optional] [For CSV only]*: The string that separates fields' values in the content of the output file. If it is not defined, the component will use the delimiter from the the set or current culture.

Audit Log Collection Manager

Audit Log Export component provides flexible configurations for what and how audit logs are exported.

On this page, you can view, add, edit or delete audit log collection configurations. In addition, this page also provides import and export features that help to move the configurations around different environments. These two features can also be used to do the restore and backup tasks.

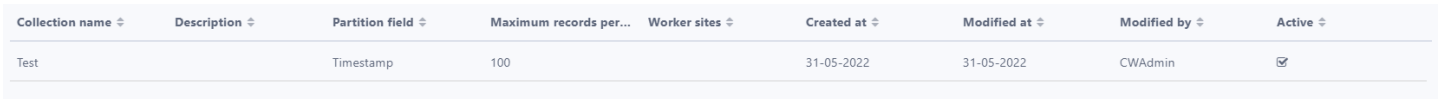


This page has a basic layout which consists of a list and some actions to manipulate items on the list.

Audit Log Collection List

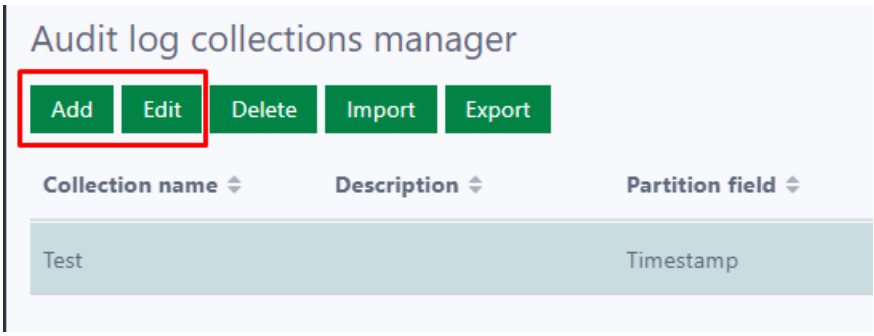
The list includes 9 columns:

- **Collection name:**: The name of the collection
- **Description:** Some details or notes about the collection
- **Partition field**: The field which is used to split the data into partitions for each run
- **Maximum records per file**: The number of maximum records can be included in one file
- **Worker sites**: The worker sites' names that are used to filter audit logs
- **Created at**: The date time that the collection is created
- **Modified at**: The latest date time that the collection is modified
- **Modified by**: The name of the user that modified the collection
- **Active**: Indicates if the collection is included in a run

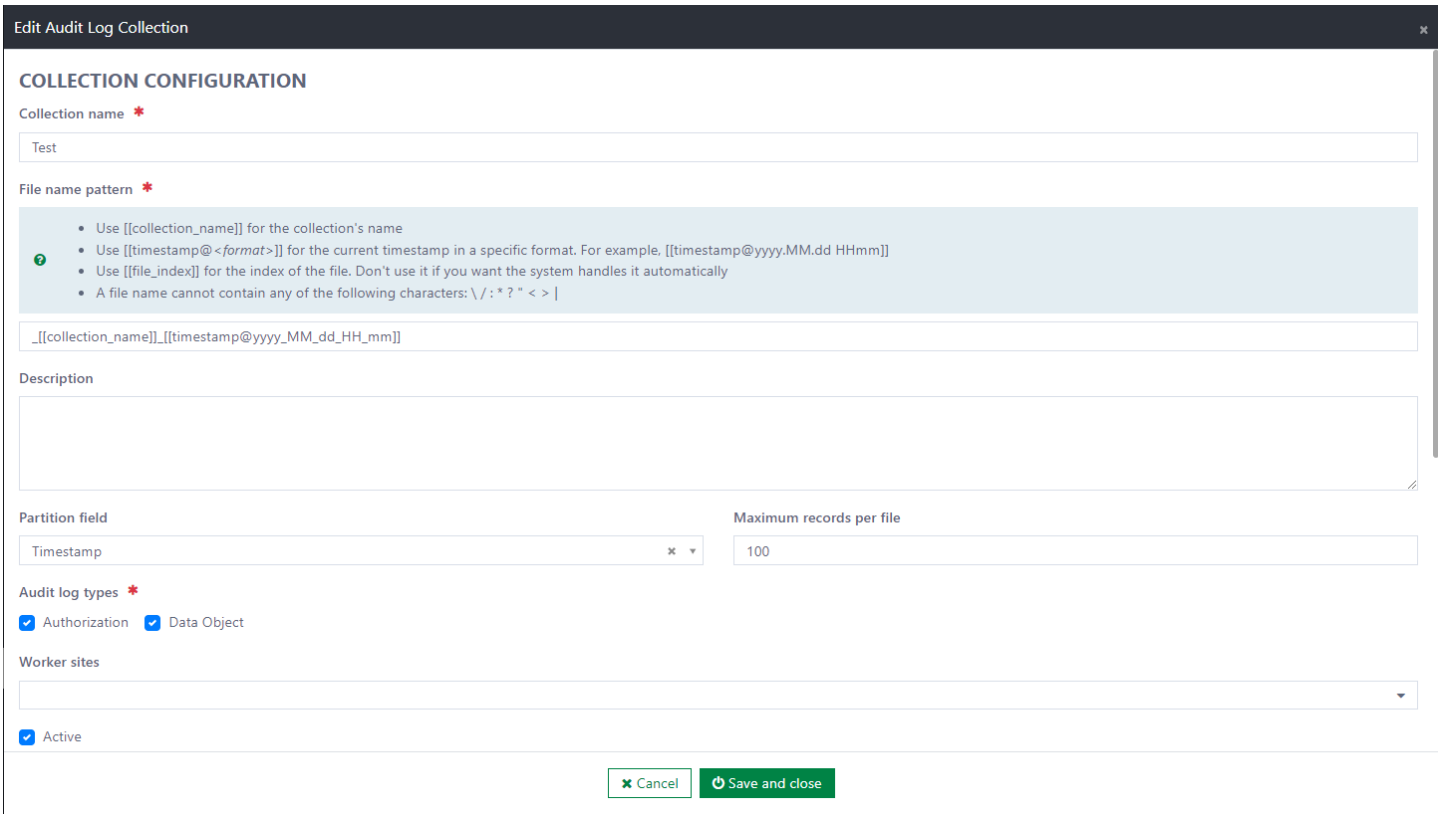


Add or Edit Audit Log Collection

To add a new **audit log collection**, click on the **Add** button. To edit an existing **audit log collection**, click on the **Edit** button.

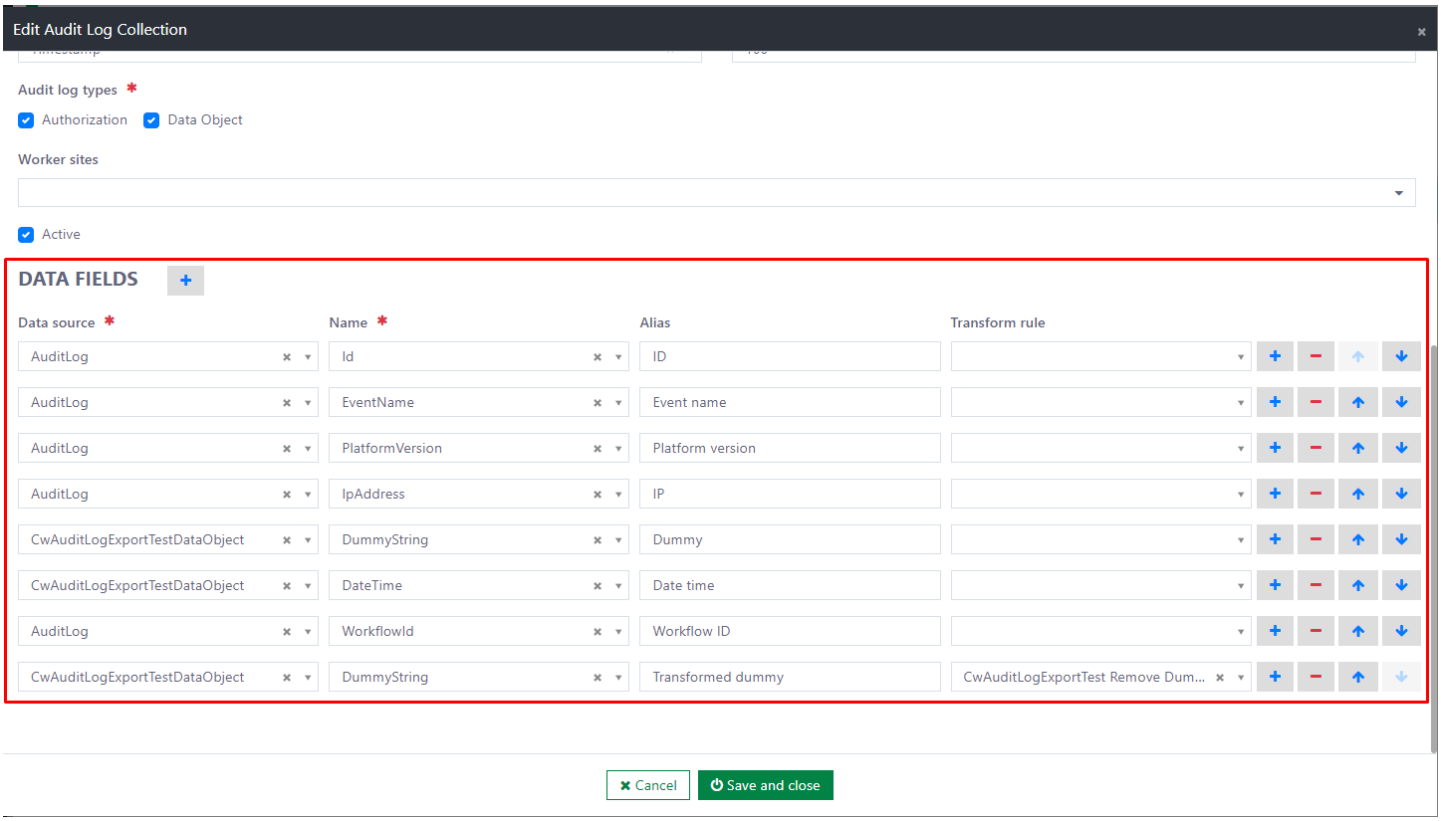


On the popup dialog, you can input the configuration for the audit log collection:



- **Collection name** *[required]*: You must input a representative name of the collection.
- **File name pattern** *[required]*: The pattern of the file name. The name of the output file will follow this pattern. Because of that, the pattern cannot contain any of the following characters: \ / : * ? " < > |. However, the component supports some dynamic fields for the pattern:
    - Use [[collection_name]] for the collection's name
    - Use [[timestamp@]] for the current timestamp in a specific format. For example, [[timestamp@yyyy.MM.dd HHmm]]
    - Use [[file_index]] for the index of the file. Don't use it if you want the system handles it automatically
- **Description** *[optional]*: You can write anything you want in this field to describe the collection.
- **Partition field** *[optional]*: Select a field that separate the audit logs in each run.
- **Maximum records per file** *[optional]*: Enter a maximum number of records in an output file.
- **Audit log types** *[required]*: Currently, there are two types of audit logs, one is Authorization and the other is Data Object. You can select one or both of them. The component will only export the audit logs which have the type that you select.
- **Worker sites**: Audit logs can come from different worker sites. If you wish to get the audit logs on some specific worker sites, you can select as many as you want here. If you want all audit logs regardless of worker sites, you can just leave it empty.
- **Active**: Mark it if you want the component to export audit logs for this collection. Otherwise, the component will ignore the collection.

Beside above configurations, you can also tell the component what data fields to be exported. The next section on the dialog is the list of the data fields.



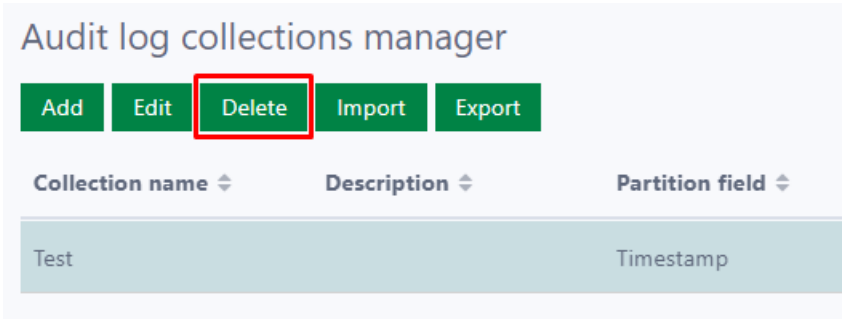
Here are attributes that can be configured for a data field:

- **Data source** *[required]*: Select a data source from the dropdown list. For data fields that belong to the audit log, select AuditLog.
- **Name** *[required]*: When the data source is changed, the selectable list of fields' names is changed correspondingly.
- **Alias** *[optional]*: If it is defined, the component will use it as the header in the output file. If it is empty, the component will use the original field's name instead.
- **Transform rule** *[optional]*: If you want to transform the raw data, select a proper rule. The data will be transformed before the file is generated.

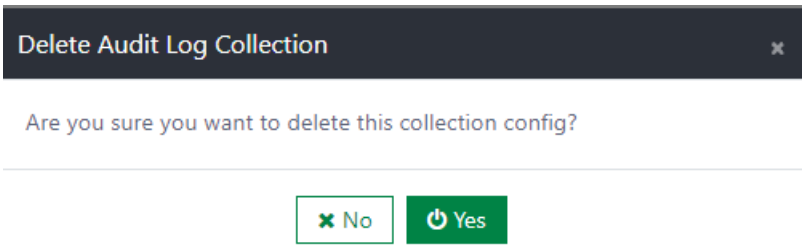
After filling in the information, click on **Save and close** button to complete adding or editing the collection configuration.

Delete an Audit Log Collection

Select an audit log collection and click on the **Delete** button to delete an **Audit log collection configuration**.

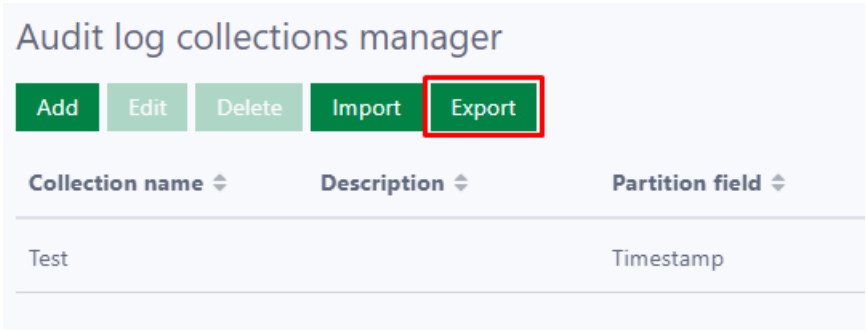


After clicking on the button, a confirmation dialog will be displayed. If you click on the **Yes** button, the configuration will be deleted immediately. If you select the **No** button, the deletion will be cancelled.



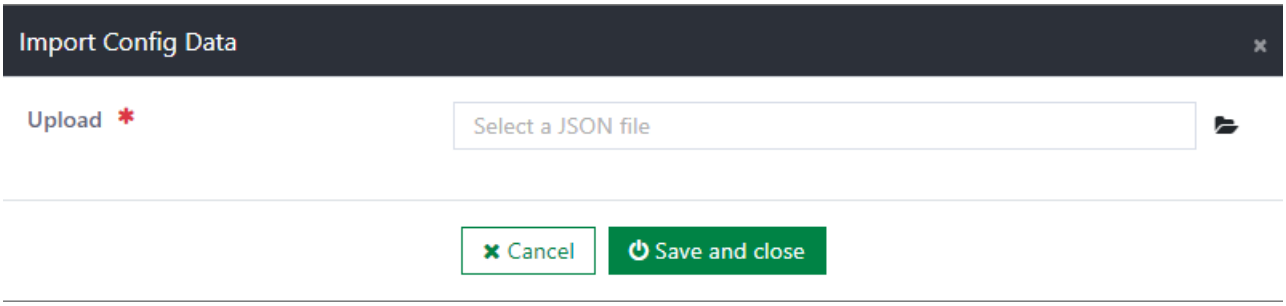
Export Audit Log Collection Configurations

To export one or several audit log collection configurations, select them, then click on the **Export** button. The download will be started shortly. If there is no selection, the component will export all configurations.



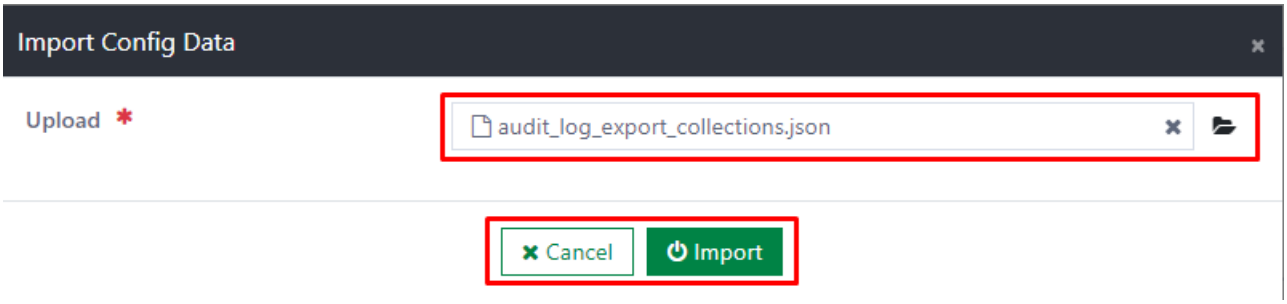
Import Audit Log Collection Configurations

Before getting started, please noted that all the data collection configs that exist in the destination environment will be overridden (Identified by collection name). To import audit log collection configurations, click on the **Import** button to open the import dialog.



Then, click on the placeholder of the **File upload** field to open file selection dialog.

Next, select a JSON file to import. After selecting the file, click on the **Import** button to start the import process. Click on **Cancel** button if you want to call off.



Scheduled Job

The export process is started automatically from a scheduled job. This scheduled job is configured to run daily at a specific time which varies for different applications.

When the time is up, the export starts. It will only process for active audit-log collections, one by one. If the collection is defined

with the Partition field, the component will only get the audit logs, of which the partition field's value is between the latest successful run to the current time. Otherwise, the time is not taken in to the account.

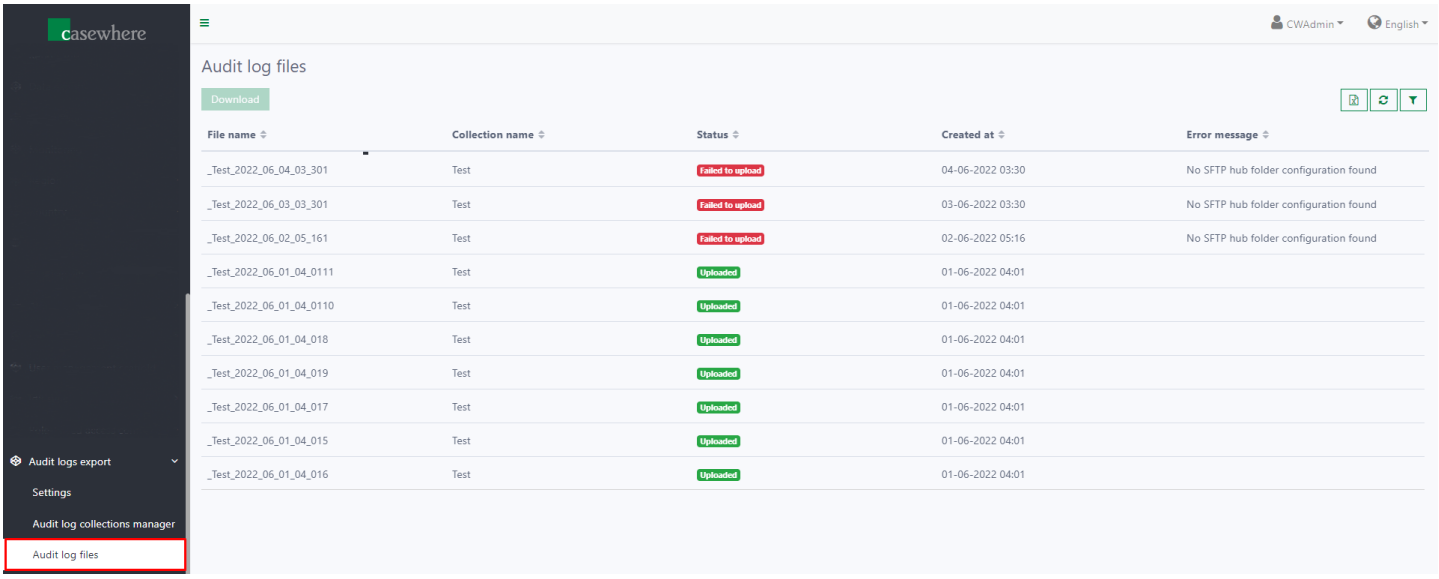
After the audit logs are retrieved and all of the output files are generated successfully, the component will upload those prepared files to the SFTP server which is configured in the **General settings**. If there is any file which cannot be generated, the process will be cancelled.

All files are listed on the **Audit log files** page.

Audit Log File List

On the **Audit log files** page, you can see a list of audit log files which are involved in previous runs. Besides, you can also download generated files.

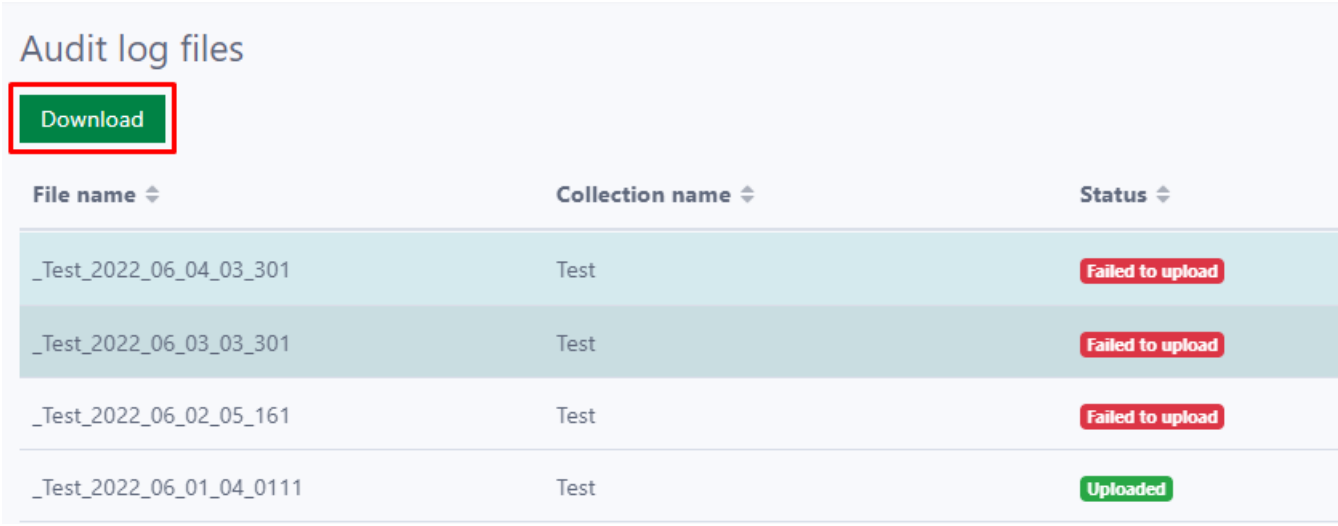
Here is the image of the page:



As you can see in the screenshot, there are 5 columns on the file list, each shows different information:

- **File name**: Obviously is the name of the file
- **Collection name**: The name of the collection that the file belongs to
- **Status**: The status of the file. Each file can have one out of seven different statuses:
    - **New**: The component is ready to create the file
    - **Preparing**: The content of the file is being prepared. The component is creating the file
    - **Failed to prepare**: The component fails to prepare the file. The file is not created
    - **Prepared**: The preparation of the file is complete. The file is saved to the application's storage
    - **Uploading**: The file is being uploaded to the remote host
    - **Failed to upload**: Something goes wrong while the file is being uploaded to the remote host
    - **Uploaded**: The file is successfully uploaded
- **Created at**: The timestamp that the file is prepared and saved to the storage at
- **Error message**: Error that happens in the process

To download a file, select it on the list and click on the **Download** button. The file is available to be downloaded only if its status is **Prepared**, **Uploading**, **Failed to upload** or **Uploaded**.



Installation

Requirements

- Casewhere 2.6.0 or later.

Configuration

1. Import products: **Casewhere SFTP Hub**, **Product-Cw CSV** and **Casewhere Audit Log Export**
2. Link the administration page to your worker sites and configure access control
3. Configure the SFTP hub folder and SFTP connection
4. Configure the Audit Log Export settings

Dependencies

- Component: SFTP hub
- Plugin: CSV

Releases

1.0.0 - 06/06/2022

Changelog

- Audit log export settings
- Collection configuration
- Output file list page

Download (login required): Audit Log Export v1.0.0

Roadmap

- More format
- Support different data storage: Sharepoint, Azure storage rather than SFTP

Component: Automatic logout

Introduction

Security is one of the most important factors of an application. Having the application idle for a certain amount of time could lead to leaking of sensitive data or misuse of the application. The automatic logout component is developed to ensure the application will not be idle for a long time without surveillance. The automatic logout component shows a session timeout popup to ask the user to stay signed in; otherwise, it will automatic logout.

Features

Configuration

The component has a page to manage a list of auto-logout configurations. Each configuration contains the following settings:

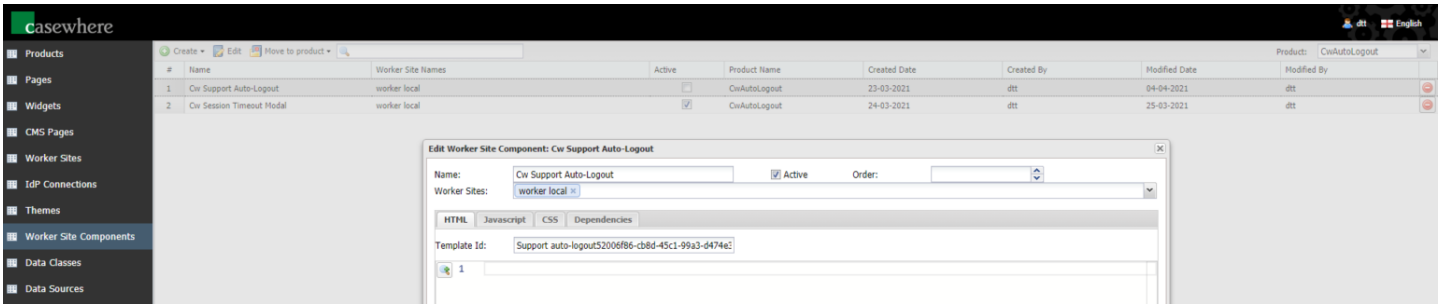
- Worker site: The worker site uses the auto-logout feature.
- Idle timeout: The amount of time in minute, after the application is idle, to show the session timeout popup.
- Stay signed-in timeout: The amount of time in second to wait for the user to decide "Stay signed-in" or "Logout".



Enable the automatic-logout feature on the worker site

Go to Casewhere Admin site and do the following steps:

- Go to the **Worker Site Components** menu.
- Edit the **Cw Support Auto-Logout** component.
- Select the worker site to enable the automatic-logout feature and click the **Save**.
- Edit the **Cw Session Timeout Modal** component.
- Select the worker site to enable the automatic-logout feature and click the **Save**.



Extend the external session

When the status is "Stay signed-in", the event **cwAutoLogout_StaySignedIn** is triggered so that other external session can subscribe to validate or handle the event accordingly.

Installation

Requirements

Casewhere 2.6.10 or later

Known issues

The component only works with the worker sites with the different domains.

GDPR compliance

The component will create a new item, called **IdleTimeout**, in the `localstorage` to keep track of the time to show the session timeout popup pop-up when the application is idle.

Releases

1.0.0 - 13/04/2021

Changelog

- Configuration
- Enable the automatic-logout feature on the worker site
- Extend the external session

Download (login required): Automatic logout v1.0.0.

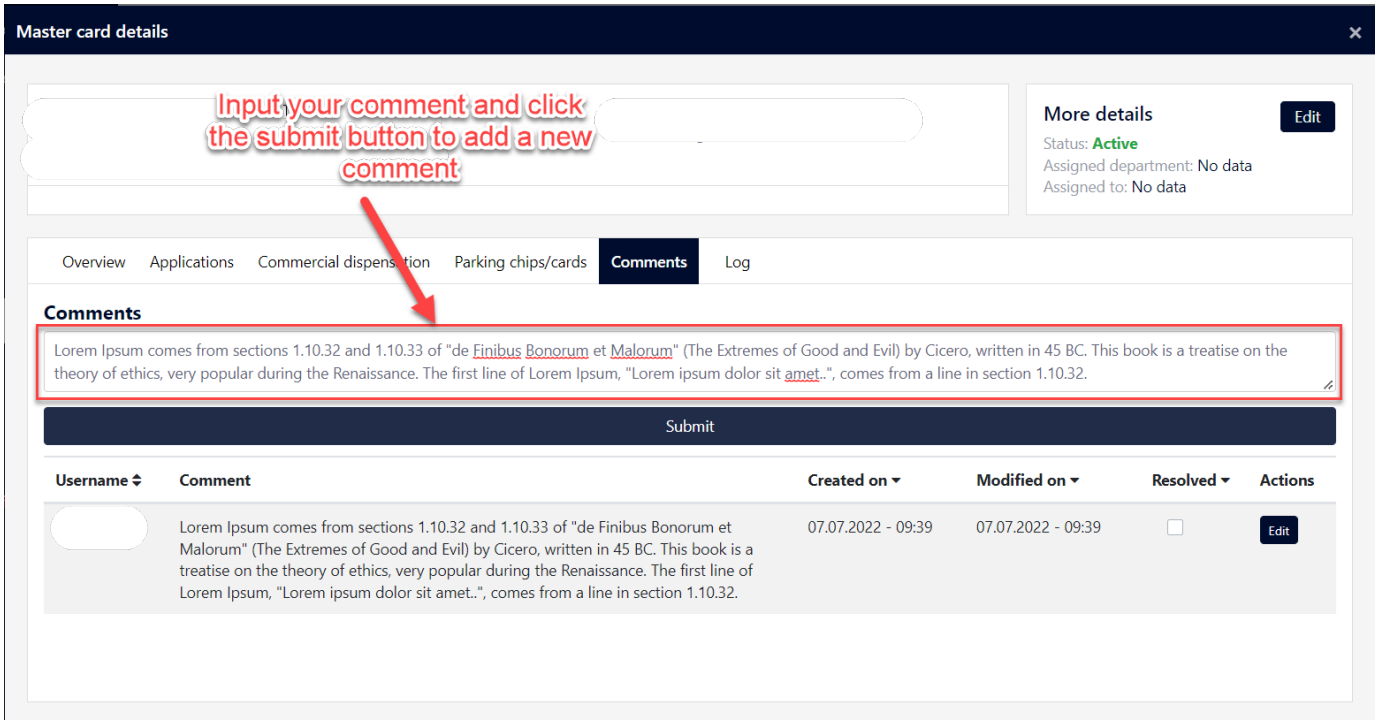
Component: Comment Box

Introduction

Providing comments and taking notes are must-have features in most case administration systems. Implementing a good comment system however is not simple. This component is designed to work out of the box for any Casewhere application. You can also customize the looks and feels in order to provide a good experience for your application users.

Features

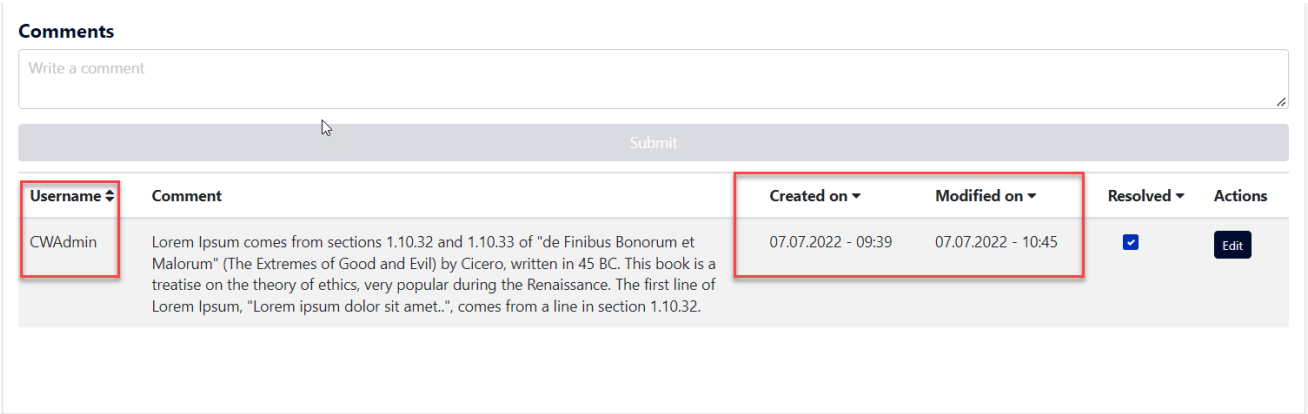
1. Add a new comment



2. Track comments

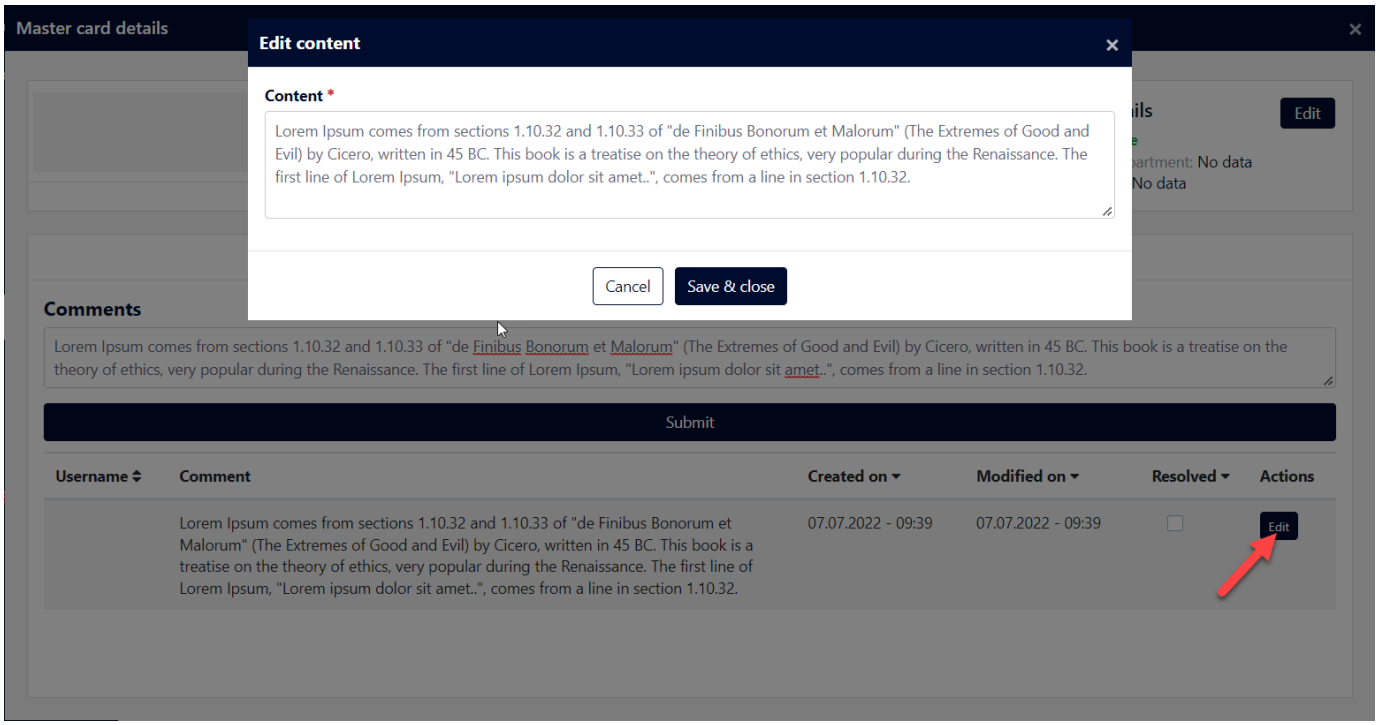
   The comment can be tracked by

   - Username
   - Created on date
   - Modified on date



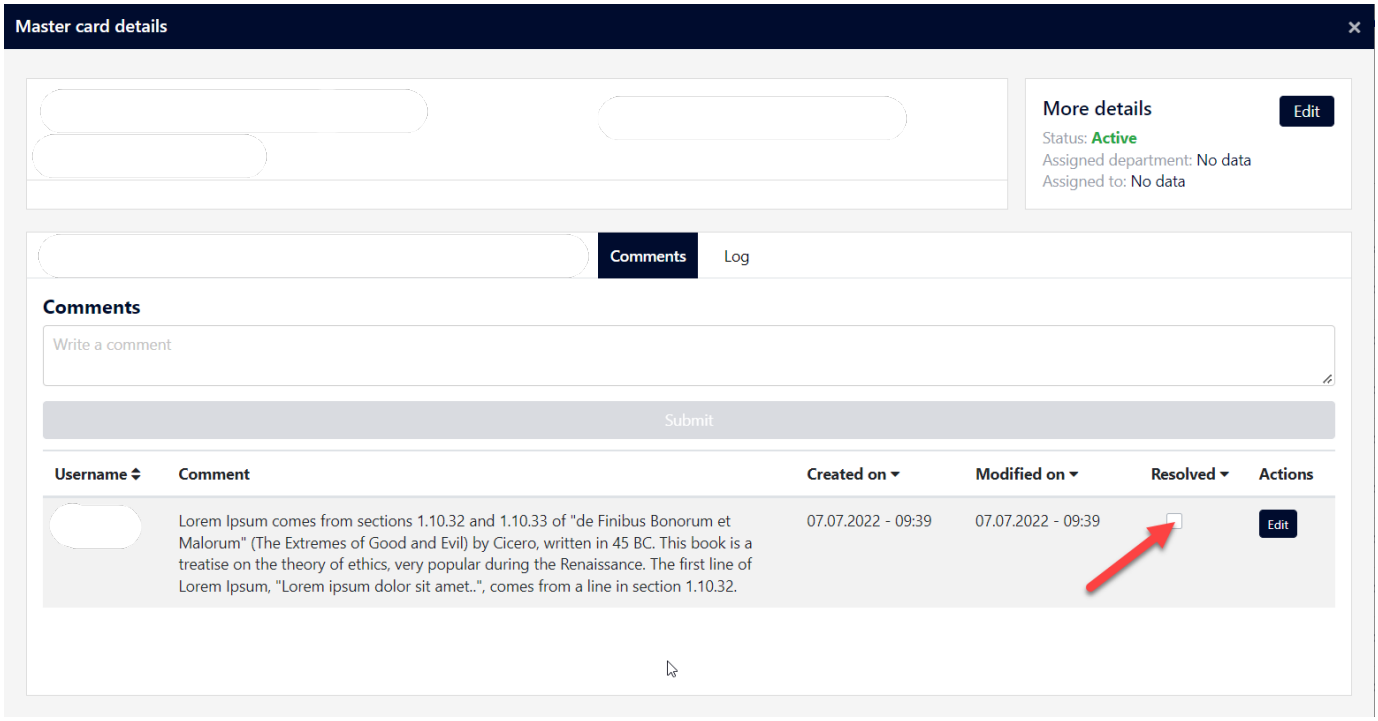
3. Edit a comment

   The comment owner is able to edit their comment any time although the message is closed.



4. Mark a comment as resolved

   Each comment can be a task and the user is able to resolve the task by .checking on the resolve checkbox.



5. Easy and fast configuration

   It's simple like adding a comment. To enable the feature for your application, drag and drop the widget into a page or a workflow. Finally, configure the context so that Casewhere can know which case the comments belong to. See configuration section for more details.

Installation

Requirements

Casewhere 2.6.10 or later

Configuration

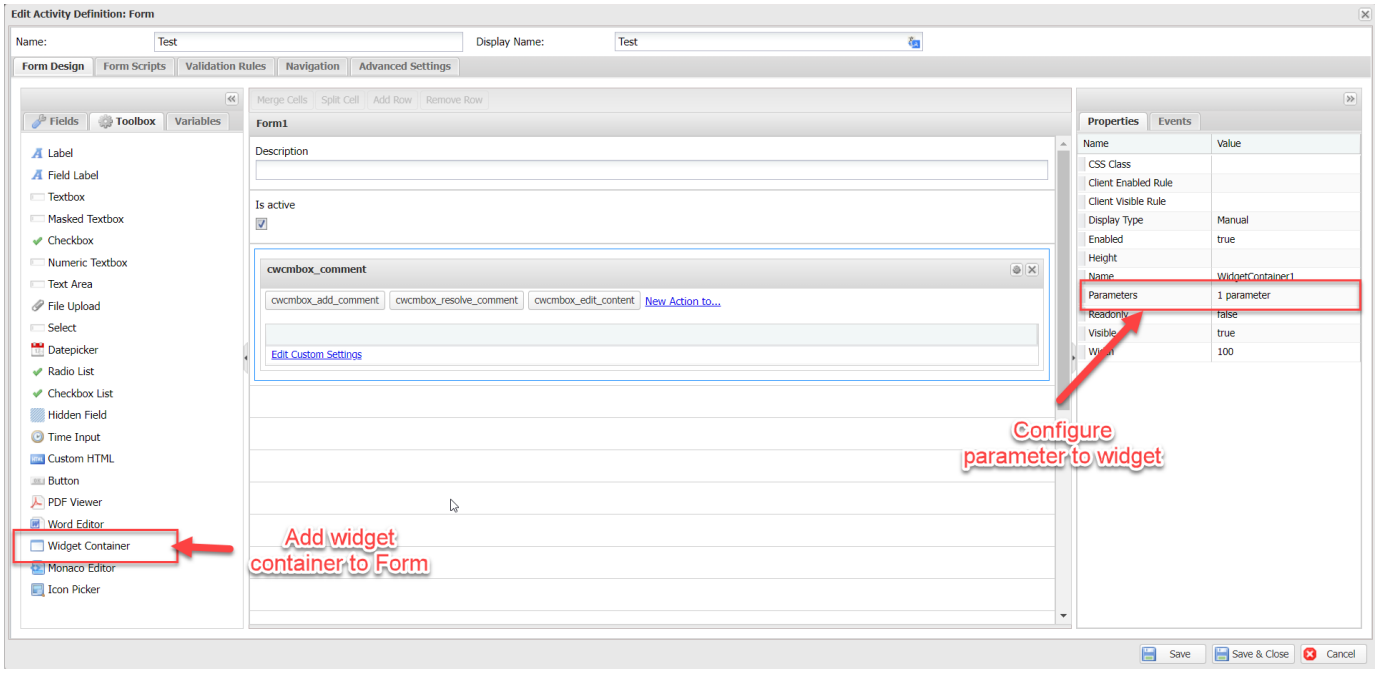
1. Import the product.
2. Enable the feature by adding the comment widget into your pages/workflows.

Example

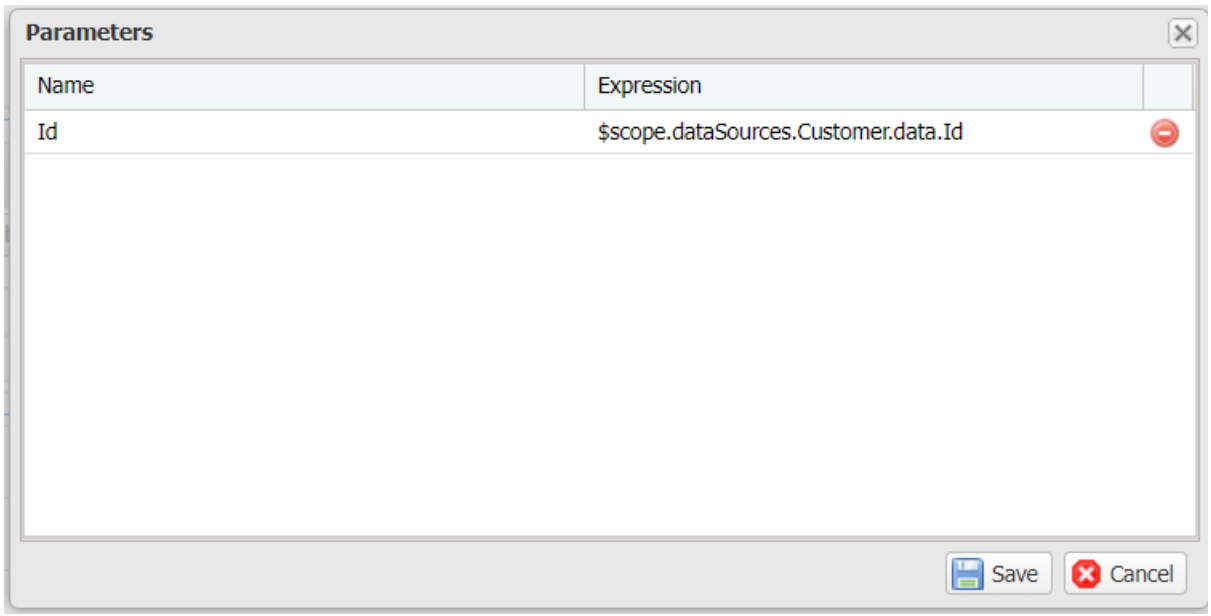
Can use this widget on the workflow by following these steps

1. Create workflow and Form Activity.
    - Create a workflow
    - Create a Form Activity

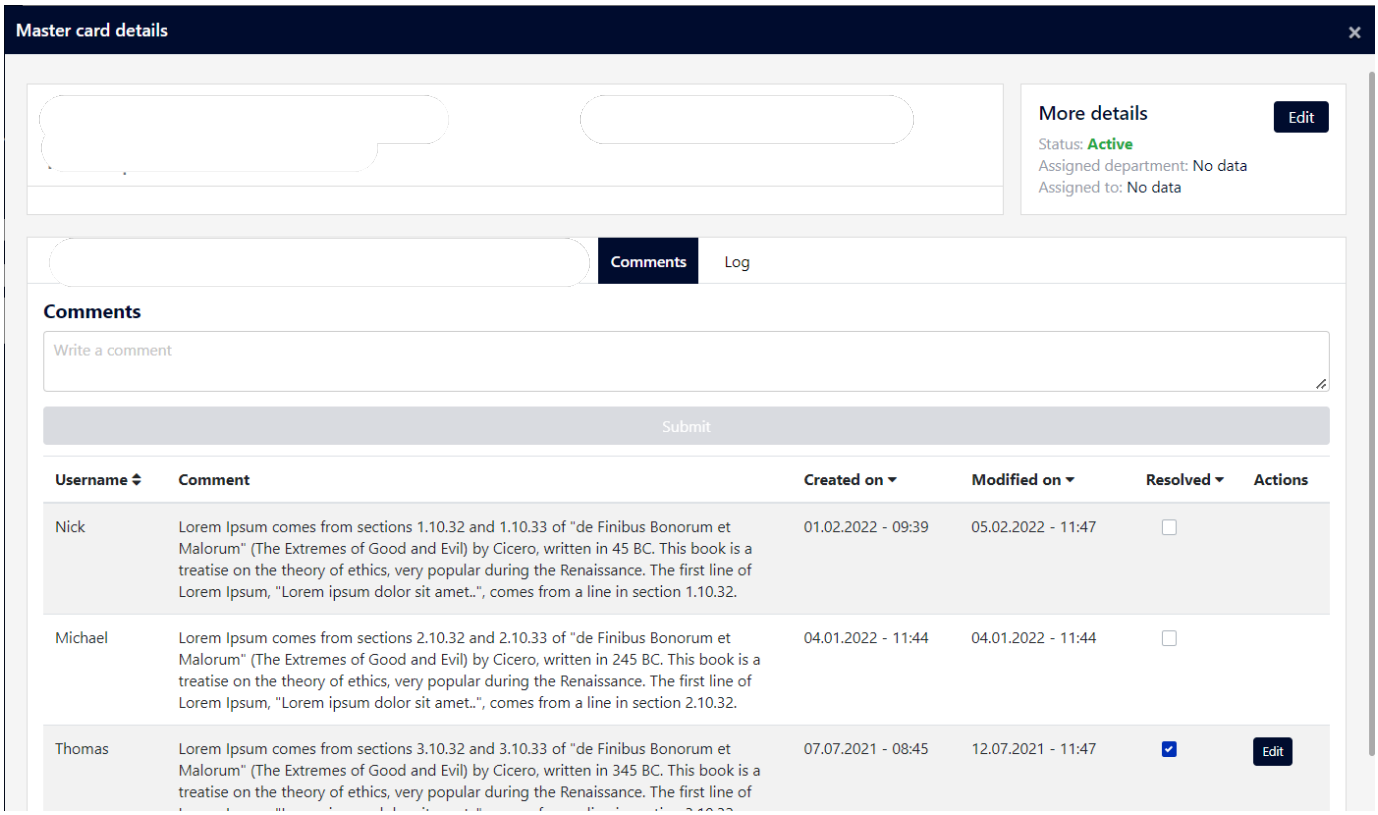
2. Create a widget container, select **cwcmbox_comment**



3. Input a parameter called Id so the widget can know which case the comments belong to



4. The widget will know and render all comments for the relation context (Ex: Application, Customer, User,...).



Releases

1.0.0

Changelog

- Comment widget

Download (login required): Comment Box v1.0.0

Roadmap

- Export comments
- Add tags

Component: Data migration

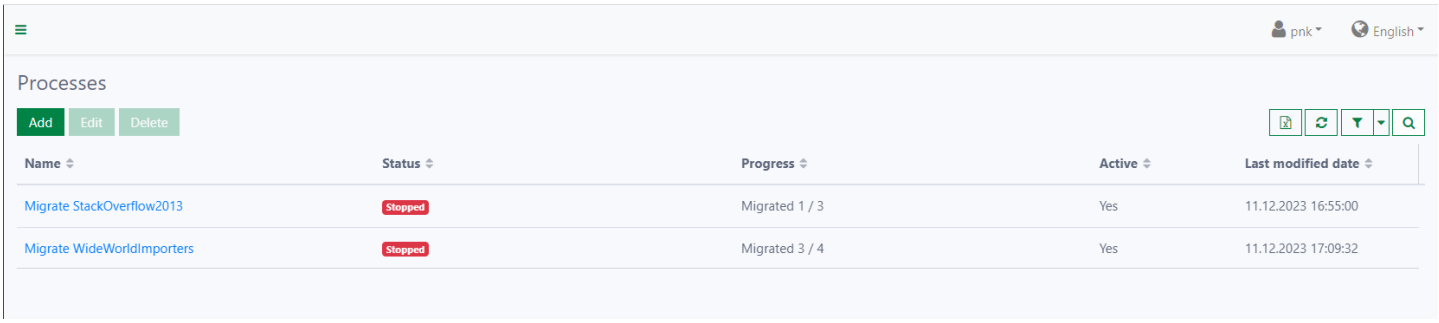
Most of the projects developed on top of Casewhere need data migration. By understanding the demand as well as the complexity of the data migration, Casewhere provides a standard component to facilitate the job:

- A framework for setting up data migration rules
- Automation and batch processing
- Data monitoring
- Standard plugin interface to work with different data sources
- Data verification

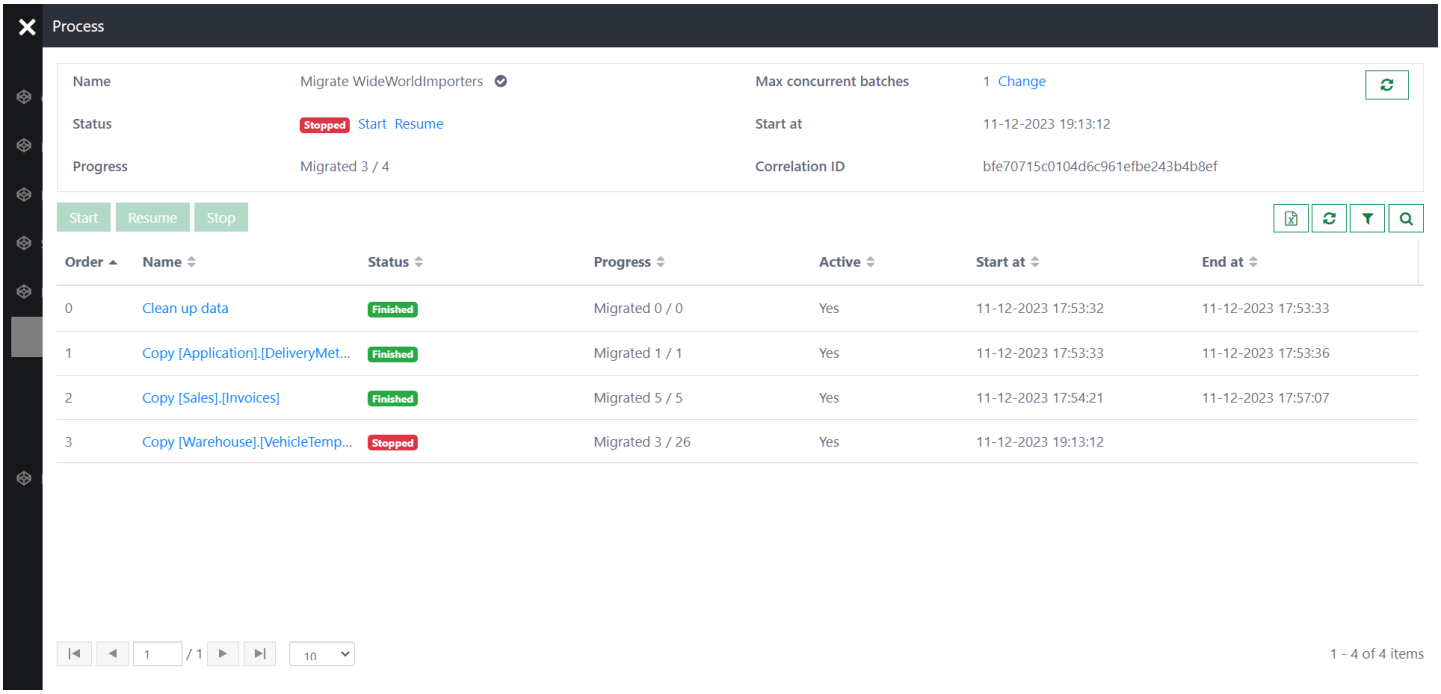
Features

Automated process and batch processing

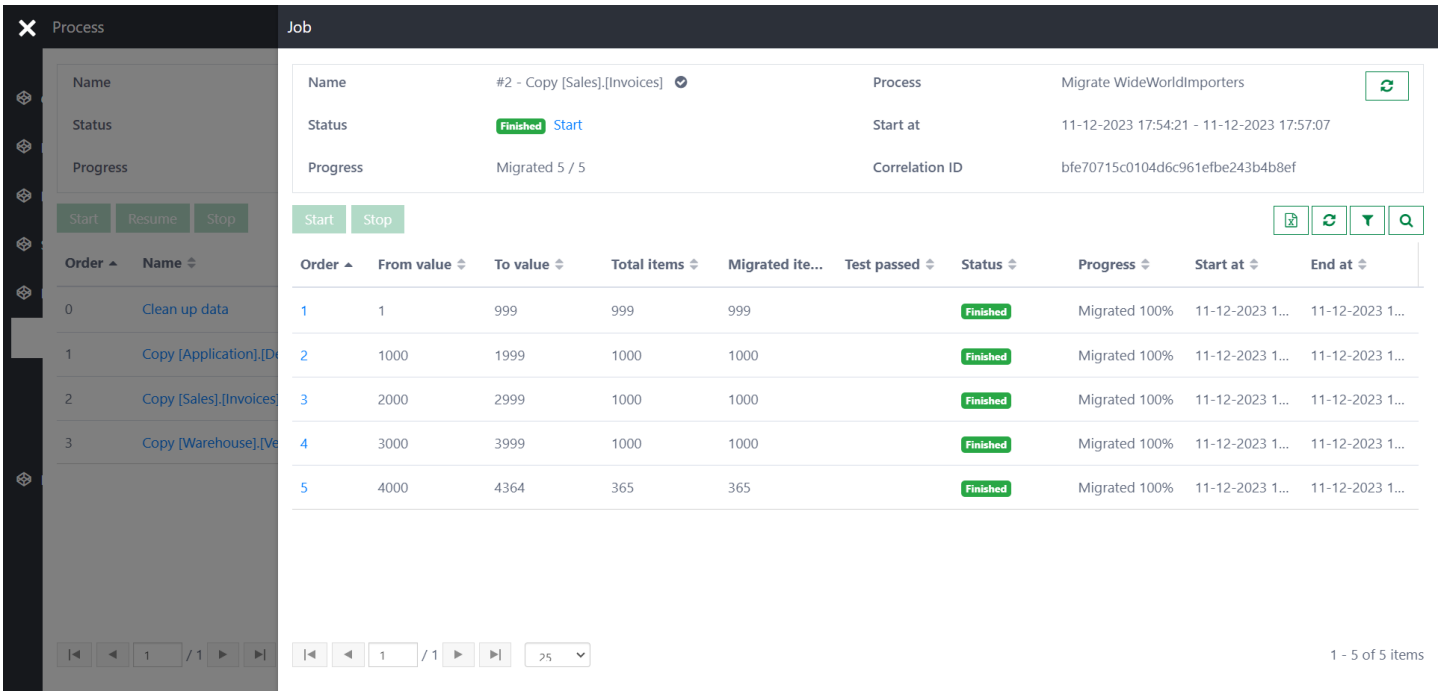
You can organize data migration work into processes. For example, one is for migrating master data, one is for major business entities, another is for moving files, etc.



A process can have as many jobs as needed.

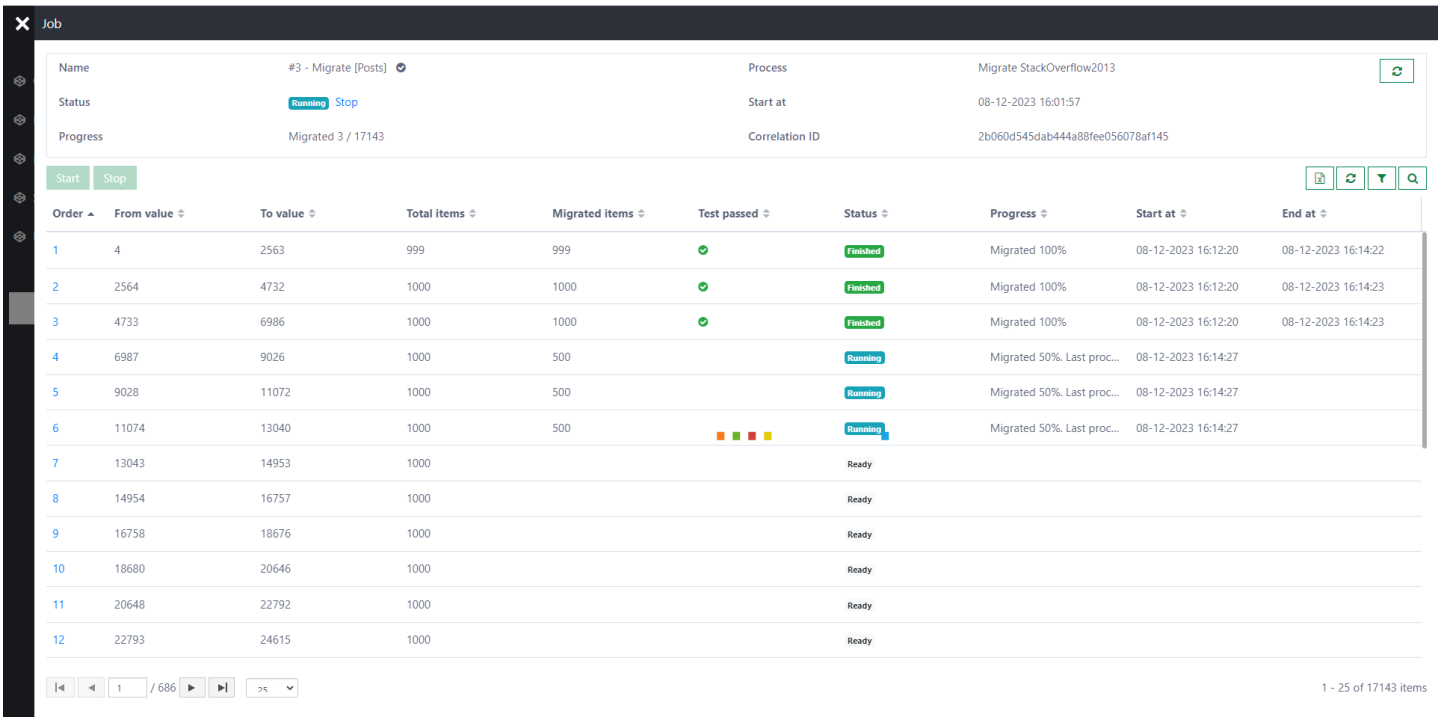


It currently supports two jobs: **Copy data** and **Custom**. For custom jobs, you can trigger a specific workflow to do all kinds of work, from cleaning up data to enriching migrated data. Copying data will be done in batches. It's possible to control how many batches can run simultaneously for a specific process.

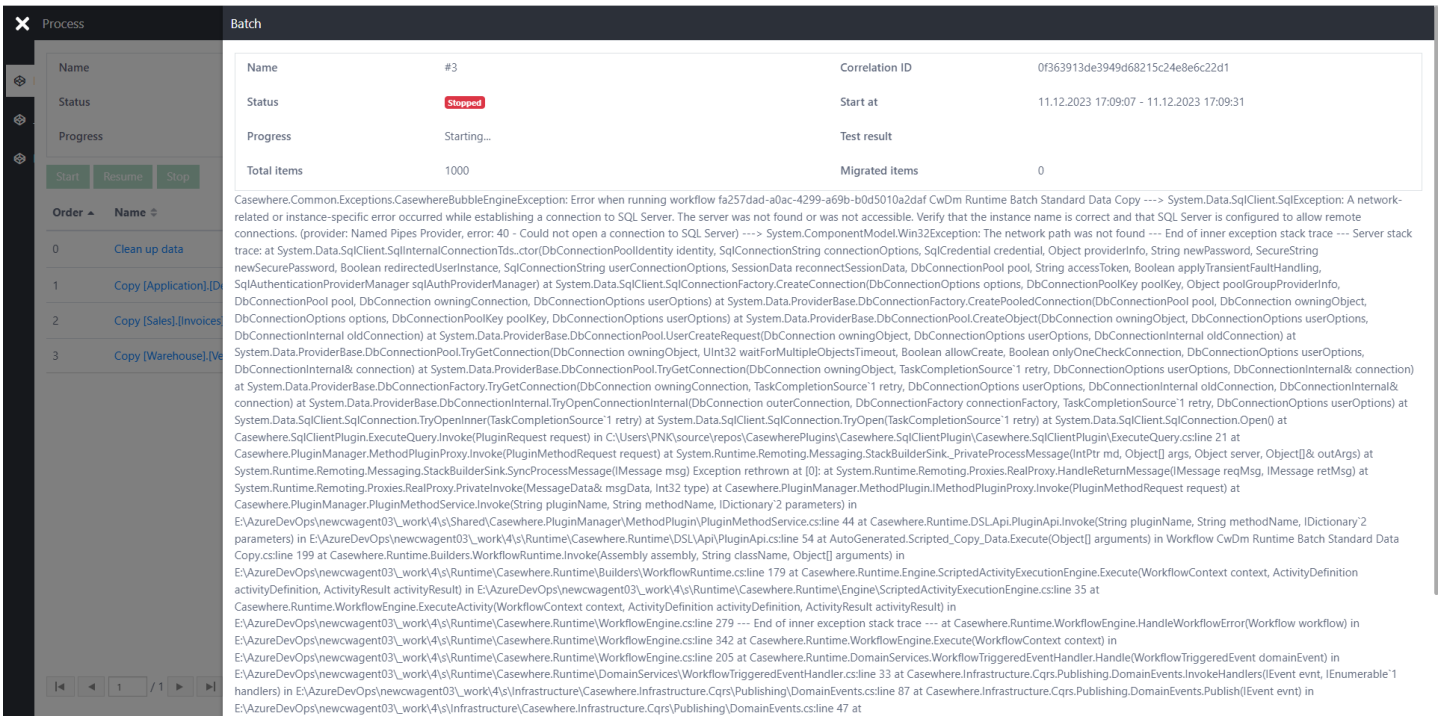


Monitor migration progress

The component provides visual tools for tracking the migration progress. It's always possible to know the current status and progress of processes, jobs, and batches.



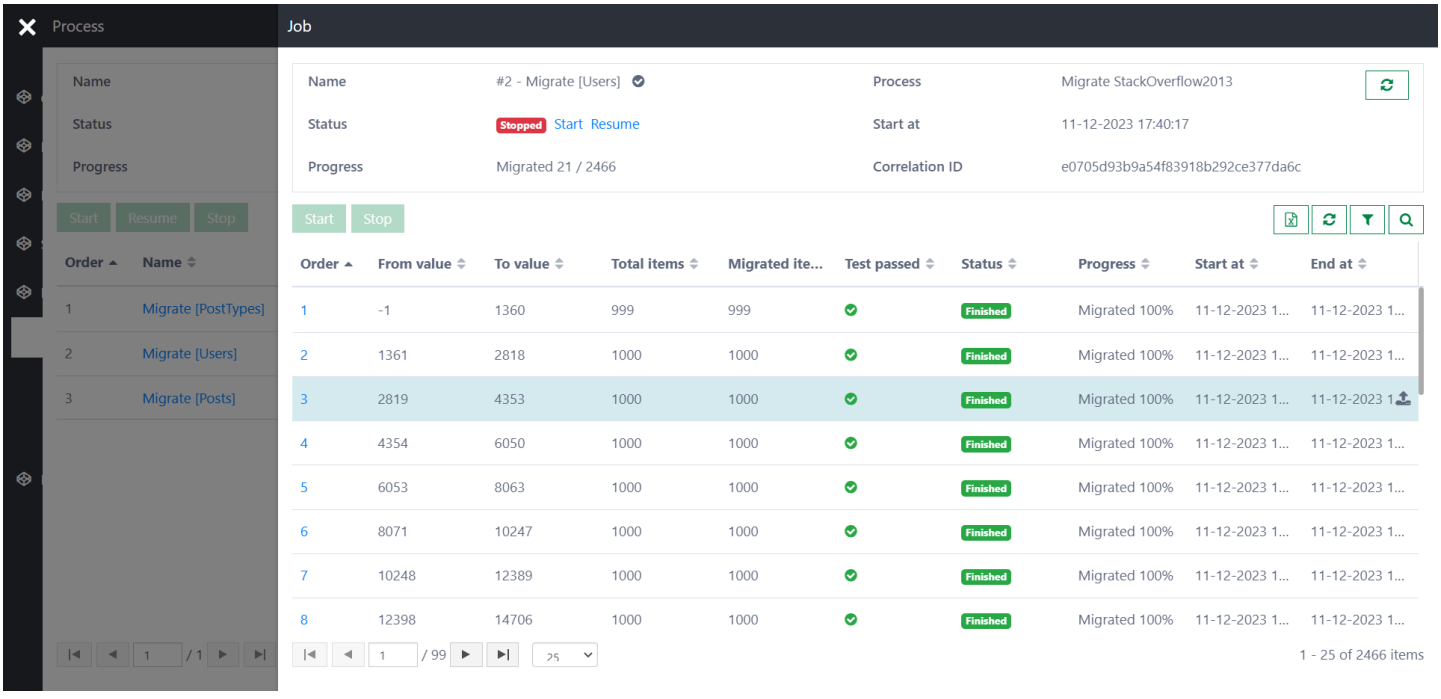
The component will stop whenever it encounters an error. You can view the error details or use the **Correlation ID** to obtain more information.



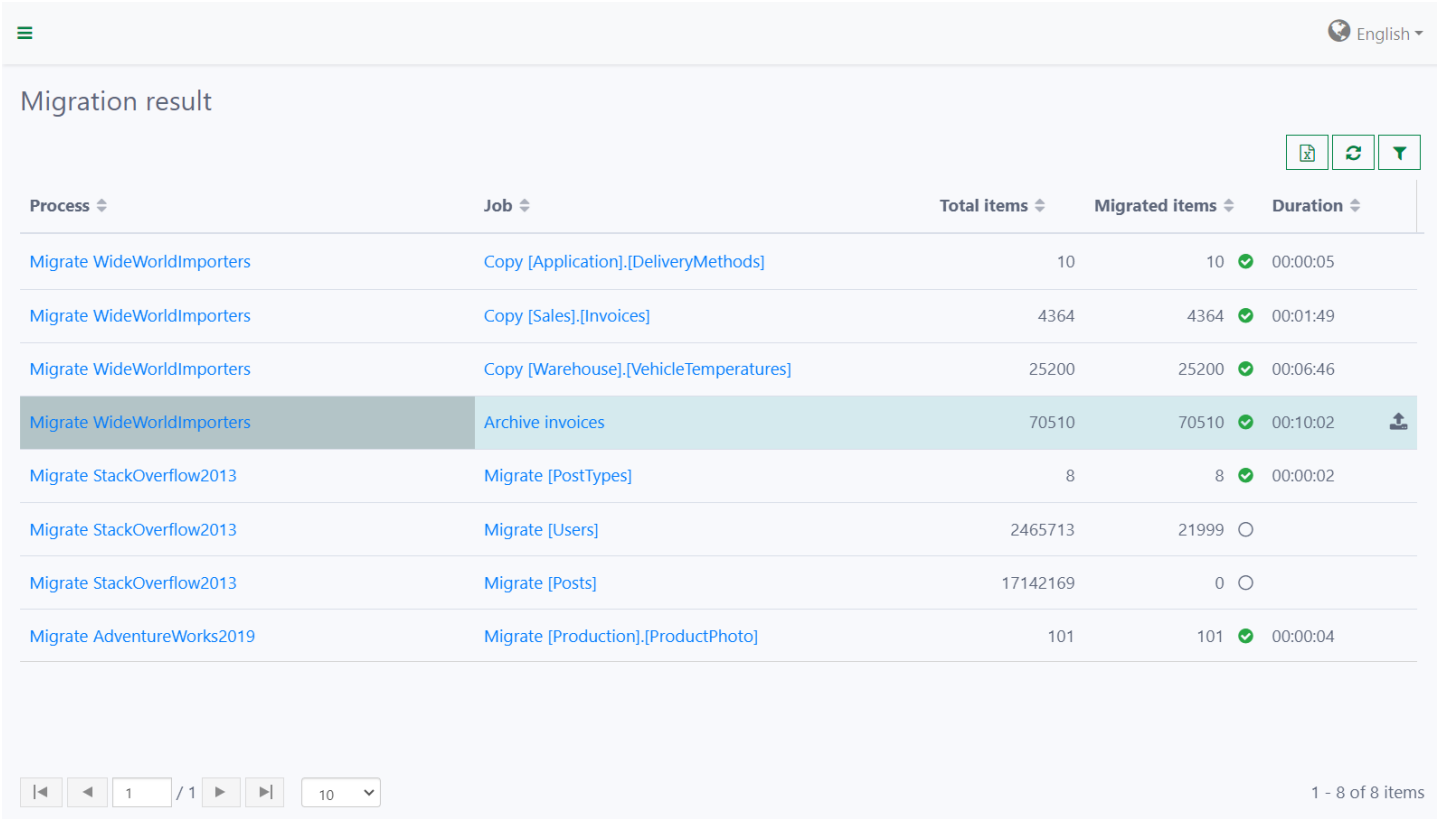
After fixing the error, you can resume the process or the job. It's also possible to rerun a single batch specifically.

Data verification

The component allows defining rules to verify data after migration. The component will run the configured rules right after a batch finishes to ensure everything is migrated as expected. You can decide if the job should continue or stop if a test is failed.



Additionally, the component provides a report comparing the amount of data migrated. You can export the data to Excel.



Installation

Requirements: Casewhere 2.8.0 or later.

Installation guide: Please follow the following steps to install the component:

1. Import the **Casewhere Data Migration** product.
2. Link the pages to your worker sites and configure the necessary access control.
    1. It's recommended to create a temporary worker site for data migration and shut it down after finishing the job.
3. Configure the SQL plugin and update the connection string.
4. Define the data migration process and implement the necessary rules for transforming and verifying data. For more information, please read the configuration guide.

You can learn more here for planning a successful data migration.

Releases

1.0.0 - 09/07/2021

Casewhere 2.6.0 or later.

Changelog

- Process design
- Data migration rules
- Automation and batch processing
- Standard plugin interface to work with different data sources

Download (login required): Data migration v1.0.0

1.1.0 - 12/12/2023

Casewhere 2.8.0 or later.

Changelog

- Process design: More features
- Data verification rules
- Automation and batch processing: More features
- Data migration report

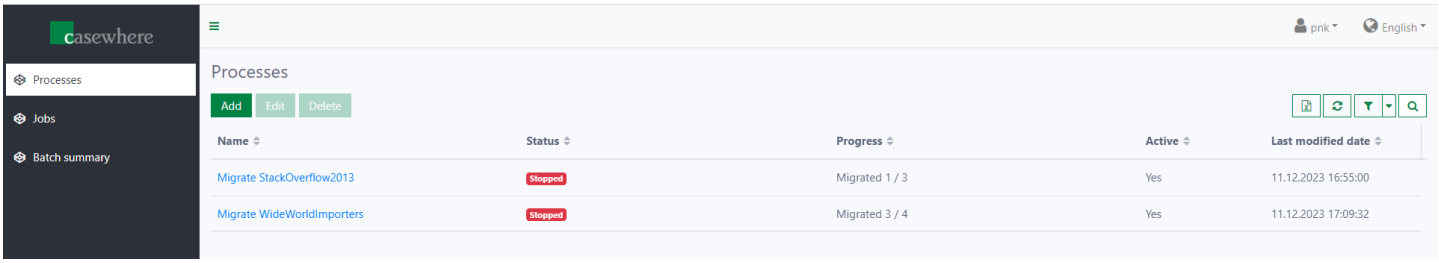
Download (login required): Data migration v1.1.0

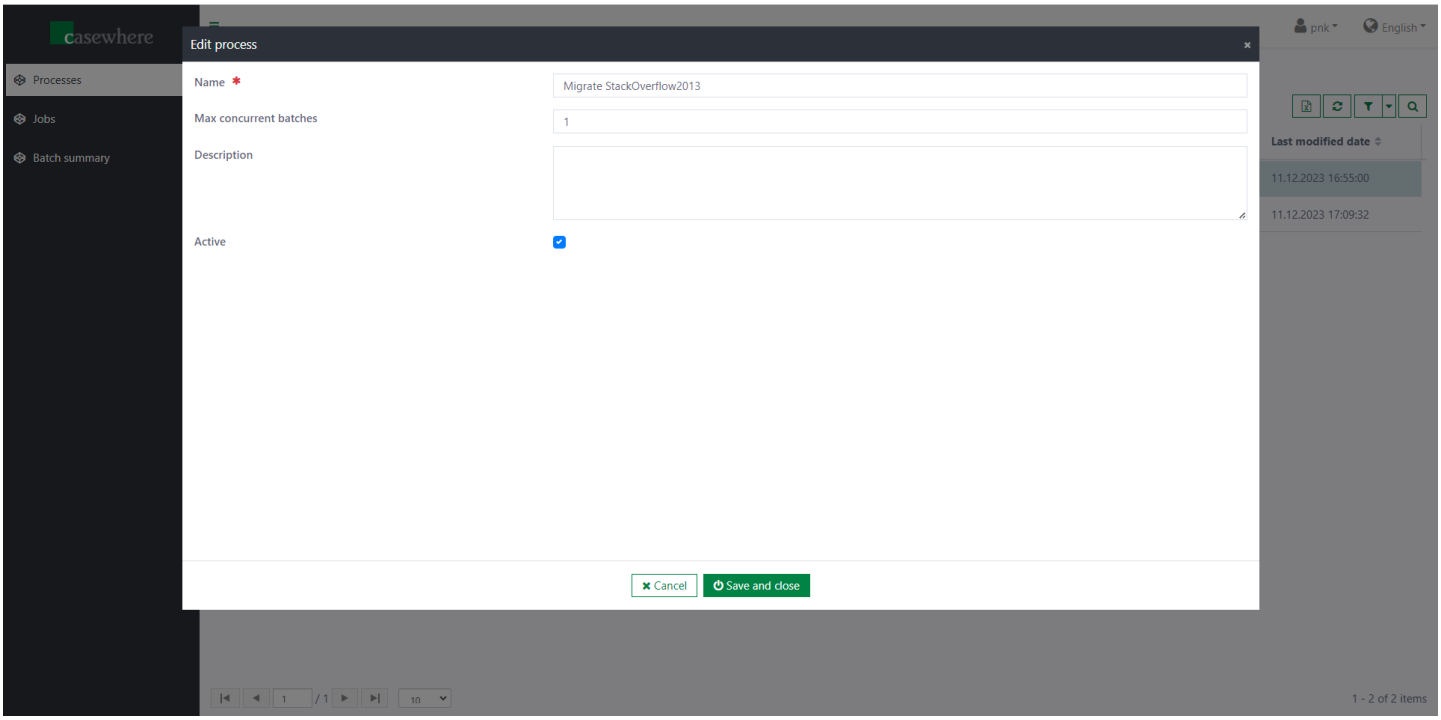
Component: Data migration - Configuration guide

This article will show you the steps to configure a data migration process and some guidelines for monitoring, debugging, and verifying data.

Data migration processes

The component offers a page for managing data migration processes. Here, you can create as many processes as needed.

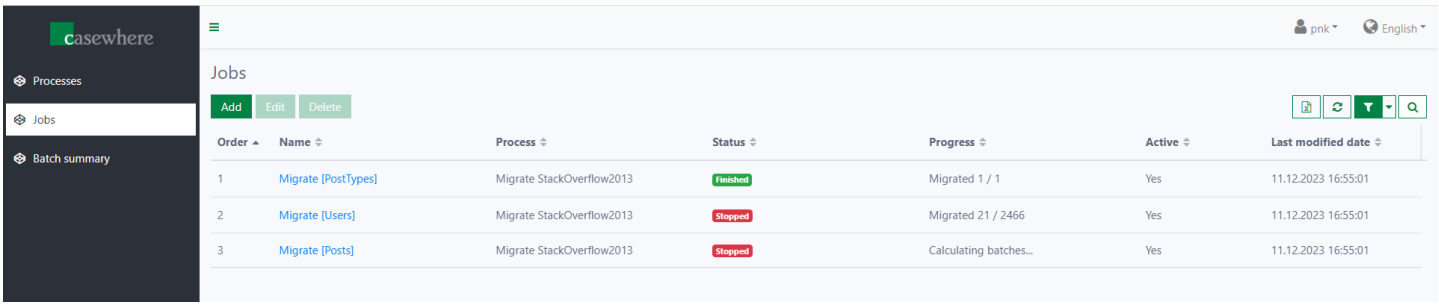




> Please note that you cannot edit or delete a running process except to change the maximum concurrent batches.

Data migration jobs

You can add as many jobs as needed for your data migration process. The component will run jobs sequentially in the order you configure.

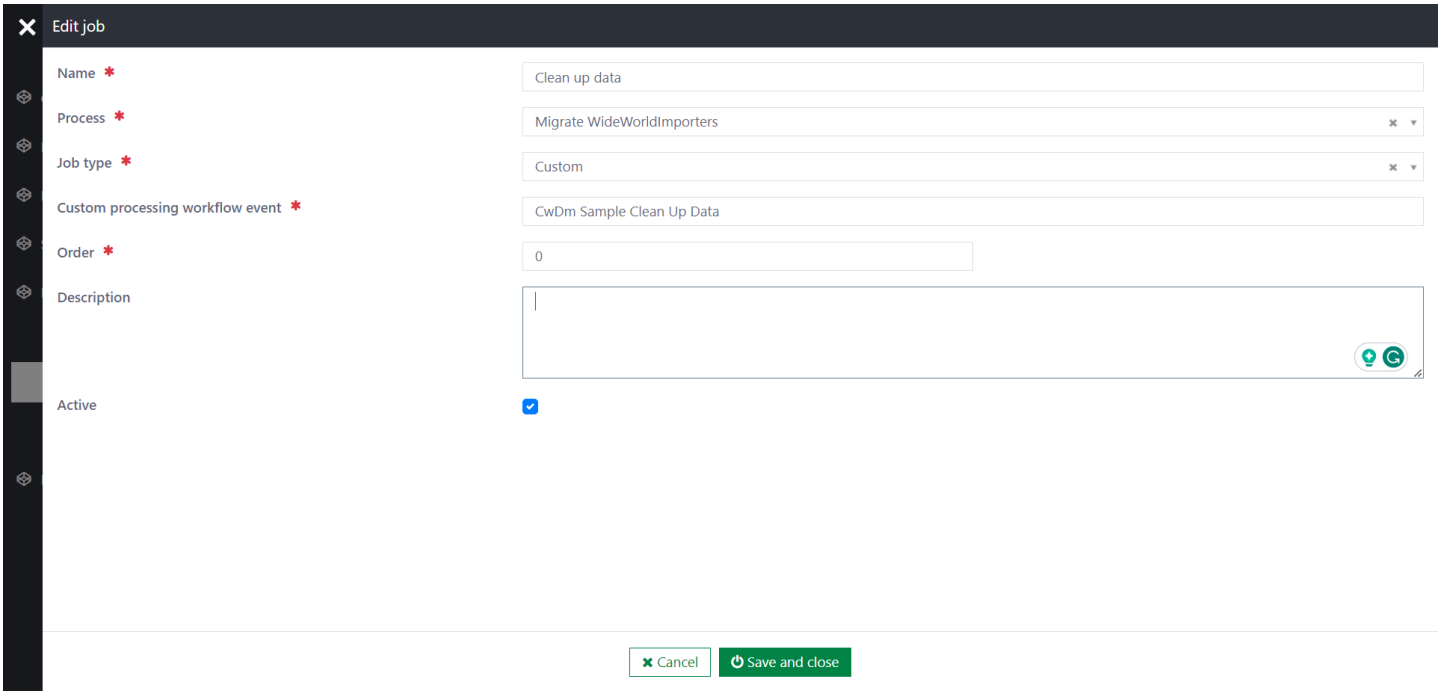


Currently, you can use two types of jobs: **Custom** and **Copy data**.

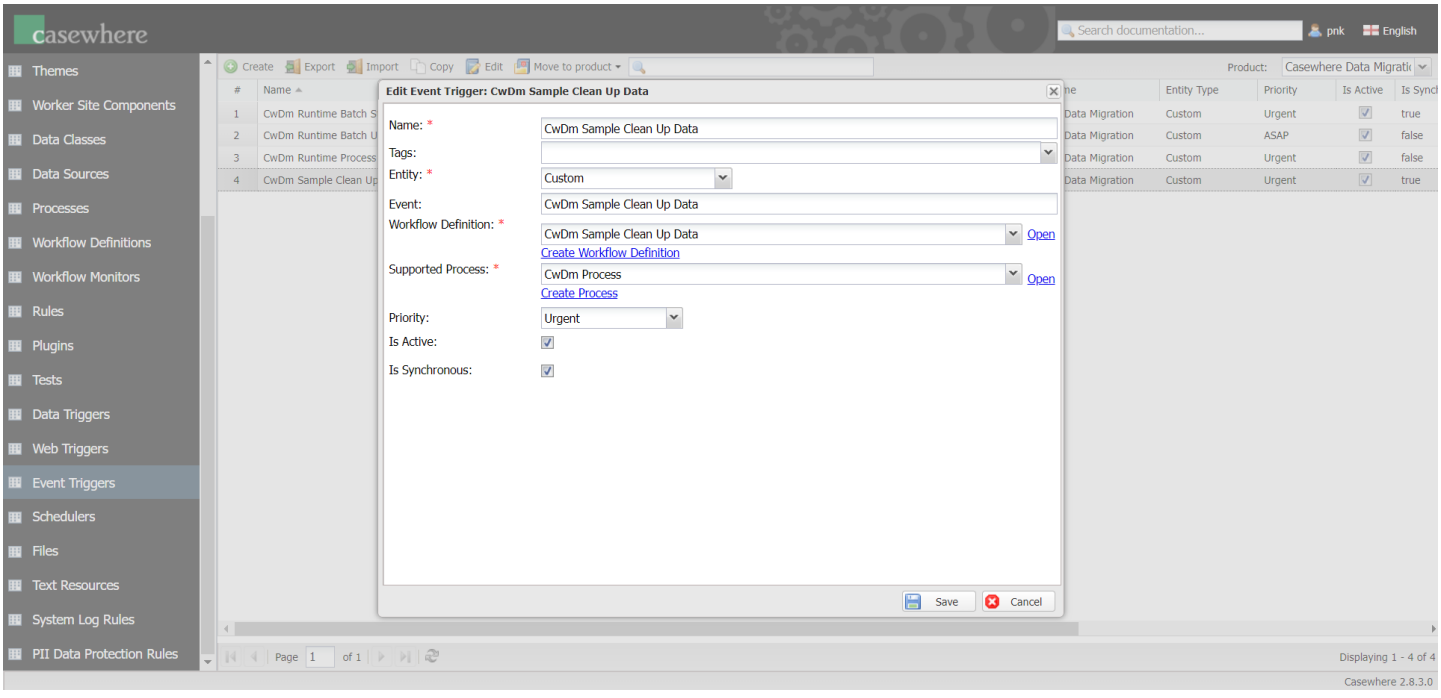
Custom job

The following information is required when configuring a custom job:

- **Custom processing workflow event**: The custom event triggers the workflow.



Please note that you must configure the corresponding event trigger in **Casewhere Admin**. The event data structure should be like this `{JobId: <string>}`.



It's also possible to enable batch processing for custom jobs. Click here to learn more.

Copy data

The following information is required when configuring a copy-data job:

- **Source table name:** To identify the table from which you want to copy data. For example: `[Sales].[Invoices]` or `[StackOverflow2013].[dbo].[Posts]`
- **Custom select (Optional):** The component will default select all fields from the source table. You can customize the select

statement here to select some particular fields or join with other tables. For example:

  - `sql SELECT [InvoiceID], [LastEditedWhen] FROM [Sales].[Invoices]`

- **Custom filter (Optional):** The component will default retrieve all records from the source table. You can customize the filter condition as described below:

  - `sql [LastEditedWhen] < '2013-03-31 07:00:00.0000000'`

- **Partition field name:** The component will use this field to partition the source table into batches of data. For example, if we use `[InvoiceID]` as the partition field, each batch represents a range of data where `InvoiceID >= from_value and InvoiceID <= to_value`.

- **Batch size:** The default maximum number of records per batch is 1000. You can customize it to optimize for your particular scenario. Please note that larger batches might require more server resources (CPU, RAM) to process.

- **Use custom workflow:** Use this if you have custom logic that the standard workflow cannot handle. You can copy the standard workflow and customize it to adapt to your needs.

  - **Batch processing workflow event:** The custom event triggers the workflow. Please note that you must configure the corresponding event trigger in **Casewhere Admin**. The event data structure should be like this `{BatchId: <string>}`.

- **Transform rule:** To transform the source data to the destination data object. The rule is required if you don't use a custom workflow.

  - Parameters:

    - SourceItem: A `JObject` object represents the data fetched from the source table.

  - Output: A `DynamicDataObject` object using the following structure. You can also return an array of `DynamicDataObject` if needed.

    - ```
      c# #predicate var item = @parameters["SourceItem"]; var data = new { Name =
      item["DeliveryMethodName"], CwDmLegacyId = item["DeliveryMethodID"] // Must have }; // You can
      also return an array of DynamicDataObject if needed. return new DynamicDataObject(new {
      DataClassName = "CwDmSampleDeliveryMethod", Data = data, LegacyId = item["DeliveryMethodID"] });
      ```

- **Migration action:** The following options are applicable:

  - **Insert only**: Aways insert new records. Please note that the system will create duplicated records if you run the same batch more than once.
  - **Insert or update**: The component will insert a new record if it cannot find the legacy ID in the database. However, the component will update the existing record if the legacy ID is already present. This behavior ensures no data duplication, even when you run the same batch twice. Obviously, this mode will be slower than **Insert only**.
  - **Ignore**: The component will not do anything if it finds the legacy ID in the database.

- **Test rule:** To verify the data after migration. Please note that to ensure the overall performance, the component only runs the rule against the last data item of the batch. Below is an example:
      ```

```
#predicate
var item = @parameters["SourceItem"];

var dataApi = @apiFactory.Get<IDataApi>();

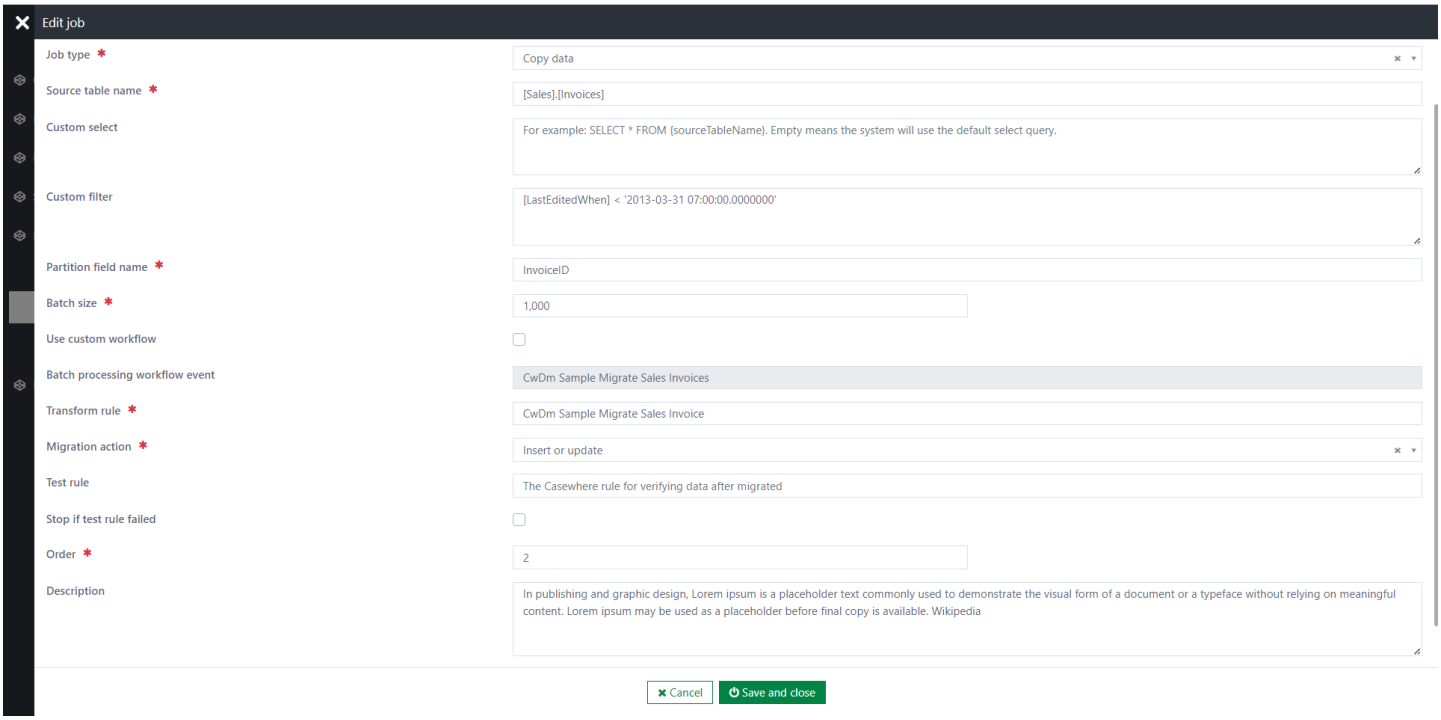
var filter = FilterBuilder.Create().Eq("CwDmLegacyId", item["Id"]).Build();
var migratedItem = (dataApi.Search(DataObjectApiQuery
    .For("CwDmSoPostType")
    .Paging(0, 1)
    .FilterBy(filter)).Data as IList<DynamicDataObject>).FirstOrDefault();

if(migratedItem == null) {
    Log.Error("CwDmSoPostType not found for {Id}", item["Id"]);
    return false;
}

if(migratedItem.Get<string?>("Name") != item["Type"])
{
    Log.Error("Name is not matched. Expected={Expected}. Actual={Actual}", item["Type"],
migratedItem.Get<string?>("Name"));
    return false;
}

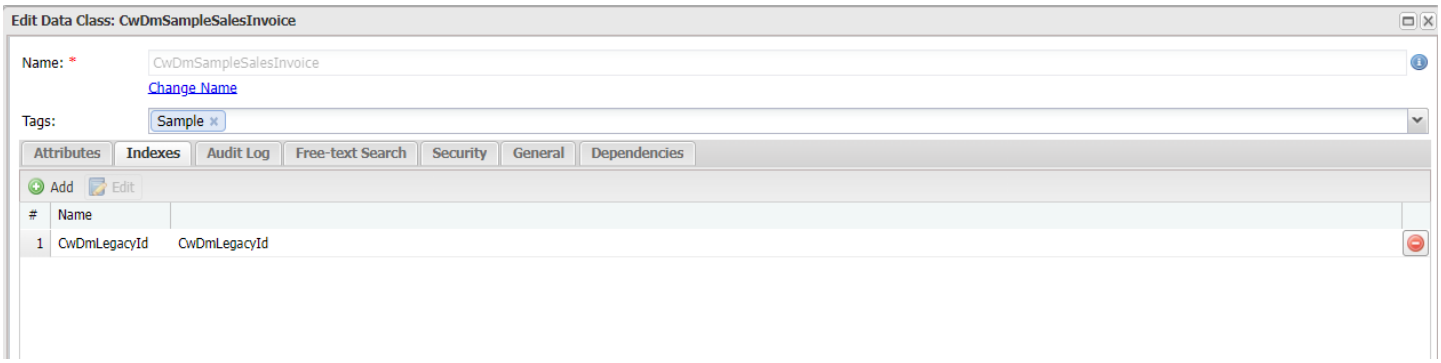
return true;
```

- **Stop if test rule failed:** If checked, the job will stop when a test fails.



For all destination data classes, you must add a new data attribute `CwDmLegacyId`.

- The attribute type depends on the source field
- Remember to add an index for `CwDmLegacyId` to speed up the migration performance


Edit Data Class: CwDmSampleSalesInvoice
Name: *
CwDmSampleSalesInvoice
Change Name
Tags:
Sample
Attributes
Indexes
Audit Log
Free-text Search
Security
General
Dependencies
Add
Edit
#
Name
1
CwDmLegacyId
CwDmLegacyId

Component: Data migration - Preparation guide

Data migration can be a daunting task that requires careful planning and execution. This article outlines the necessary steps to ensure a successful data migration.

Step 1: Set up the production environment

A typical data migration environment for Casewhere requires the following servers:

- Source database: To restore the legacy database.
- Casewhere database: To host the Casewhere MongoDB databases.
- Casewhere web: To deploy the Casewhere instance. This server must be able to access the other servers.

The size of these servers depends on your database size and the available timing window. To ensure Casewhere has sufficient power to migrate data, you can consider temporarily scaling up the environment and downsizing it when the job is done. A typical setup for PROD should be:

- Source database: CPU 4 cores, RAM 16 Gbs
- Casewhere database: CPU 4-8 cores, RAM 16-32 Gbs
- Casewhere web: CPU 4-8 cores, RAM 16-32 Gbs

You also need to adjust the number of background workers in `Casewhere.Worker.Api/web/config` accordingly. For example:

- `dataTrigger:Workers`: `2`
- `eventTrigger:Workers`: `2`
- `cwjobs:Workers`: Start with `5` and then adjust later if needed.

Some other settings in `Casewhere.Worker.Api/web/config` should also be adjusted, turned off, or disabled in order to ensure Casewhere runs smoothly during the migration:

- `events:EnableHotAndCold`: `false`
    - Remember to transfer all events from Hot to Cold before turning it off.
- `perfLog:enabled`: `false`
- `serilog:minimum-level`: `Debug`

Checklist:

- [ ] All 3 servers are up and running.
- [ ] You reviewed and considered the need for a temporary scaleup.
- [ ] You adjusted the background workers in `Casewhere.Worker.Api/web/config`.
- [ ] You disabled the features not optimized for data migration, e.g., Hot & Cold.
- [ ] You documented all the changes so that you can later revert the system configuration to the original state

Step 2: Deploy the business solution and back up databases

How to deploy the solution is outside the scope of this document. When you finish the final smoke tests and everything is up and running, remember to back up the Casewhere databases and ensure the backup is stored as long as needed. This backup will be required when restoring the system to a working state if something is horribly wrong in one of the following steps.

Checklist:

- [ ] The solution is up and running.
- [ ] The system state (data & configuration) is in good condition for a backup.
- [ ] You took a backup and stored it in a safe place.

Step 3: Deploy data migration processes

At this step, you should already have the data migration processes configured and tested. The safest way to deploy the processes to PROD is to package them from one of your TEST environments. After deployed, you can double check:

- Create a temporary worker site and link all the data migration pages to it
    - Use the default theme to keep things simple.
    - Localization: Show seconds in DateTime formats.
    - Security: Ensure only relevant team members are allowed to access. Keep it simple because the site is temporary.
- Check if all the data migration processes and jobs are active and in the correct order.
- Check if the plugin can connect to the Source Database.
- Decide if you should enable the job setting "Stop if the test failed".
- Ensure the "Process watcher" scheduler is running. This is to ensure the data migration can auto-resume if the server goes down for some reason.
- If you need to modify the Source Database (adding indexes, creating views, etc.), make sure those modifications are done before running data migration.

You must also review and list the factors or elements that might interrupt your data migration process. Some of the items worth considering:

- Any workflow trigger that might trigger unwanted actions, e.g., sending emails, creating payment transactions, schedulers, etc.
- Put the worker sites under maintenance to prevent users from making unwanted changes during the migration.

Checklist:

- [ ] You created the temporary worker site for data migration.
- [ ] You deployed the data migration processes.
- [ ] You double-checked the data migration processes, jobs, and other settings.
- [ ] You verified the connection to the Source Database.
- [ ] You verified that the Source Database is updated with items needed for the data migration, such as indexes and views.
- [ ] You disabled all elements that might interrupt the data migration.

Step 4: Backup databases before running data migration

Backing up the databases at this step is critical to ensure that if something goes wrong when running migration, we can quickly reset the environment and run again. Make sure you keep two backup revisions (this one and the one at step 2) separately.

Checklist:

- [ ] The system state (data & configuration) is in good condition for a backup.
- [ ] You took a backup and stored it in a safe place separated from the other backup.

Step 5: Run migration

Run the processes in the expected order.

Monitor the progress constantly.

Monitor the resource consumption on 3 servers. You can consider adjusting the "Max concurrent batches" to best utilize the available resources.

If there is an error that you can't just simply fix and resume the process, you can restore the whole database from the backups taken in the previous steps.

Step 6: Verify data

You can do the following:

- Review the "Migration result" report.
- Browse the data lists to review the data. Pay more attention to the lists with the largest amount of data.
- Run smoke tests to ensure the system still works as expected.

Checklist:

- [ ] You reviewed and compared the data in source and destination databases.
- [ ] You reviewed the data in major data collections.
- [ ] You prepared a set of test cases to smoke-test the system after data migration.
- [ ] You smoke-tested the system after data migration.

Step 7: Restore the original settings

After the data migration finishes, you should restore the configurations to the previous state, which is documented in step 1.

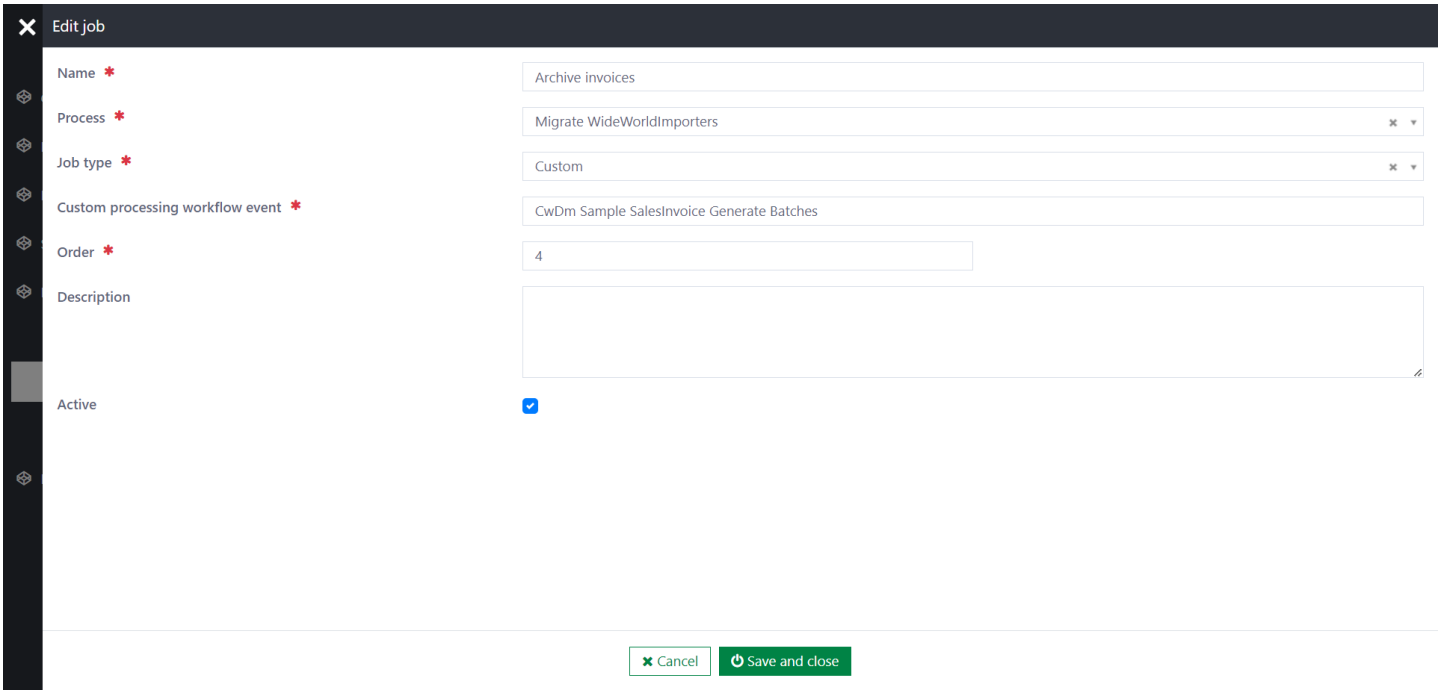
Component: Data migration - Custom batch processing

As described in the configuration guide, the component always processes copy-data jobs in batches. This enables parallel processing as well as makes it easier to monitor and troubleshoot. This article will guide you on configuring batch processing for custom jobs.

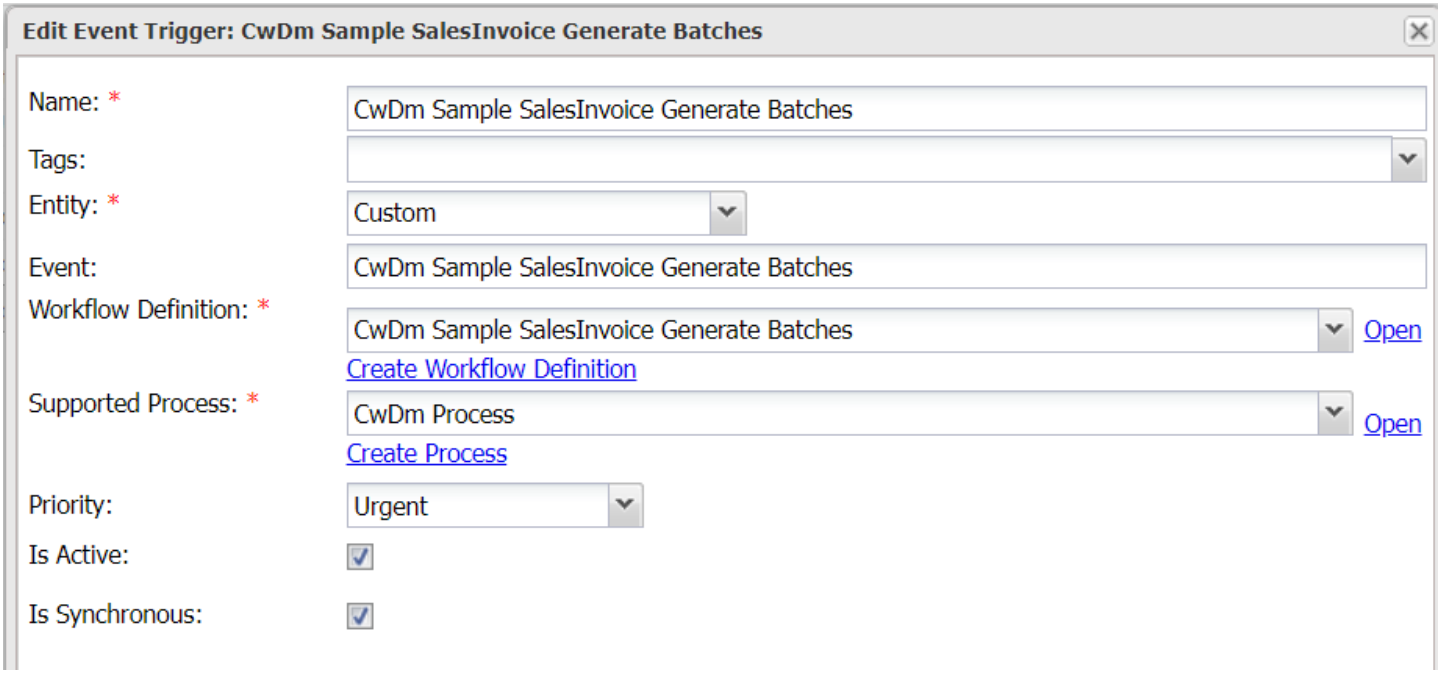
Step 1: Create the custom job

Assuming you are migrating data from the well-known database WideWorldImporters, and after migrating all `[Sales].[Invoices]`, you need to run a custom job to update their statuses to `Archived`.

Follow the configuration guide to create a custom job.

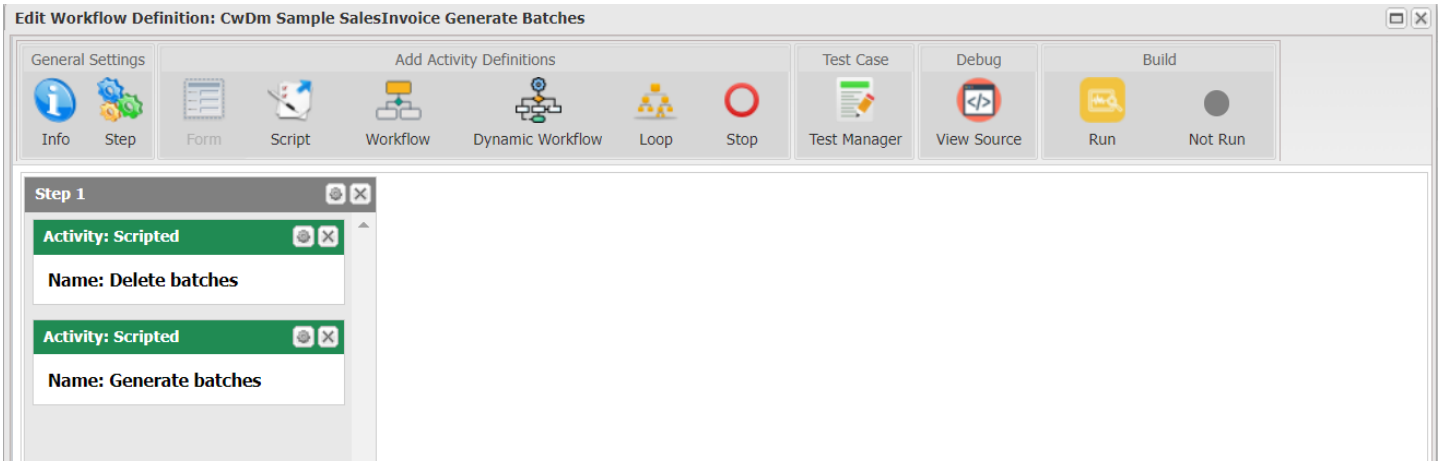


Ensure you add a synchronous event trigger named **CwDm Sample SalesInvoice Generate Batches**.



Step 2: Configure the workflow for job processing

The workflow definition for CwDm Sample SalesInvoice Generate Batches should look like below:



Scripted activity "Delete batches": A job can be rerun. To avoid duplicating batches, the workflow should delete existing batches, if any.

```
var dataApi = ctx.Use<IDataApi>();

try
{
    // Update progress
    var job = dataApi.Load(ctx.Input.Data["JobId"].ToString());
    dataApi.Update(job.Id, new
    {
        Progress = "Cleaning up existing batches...",
    });
    ctx.CommitChanges();

    // Loop through existing batches using DataApi.Enumerate()
    var filter = FilterBuilder.Create().Eq("JobId", job.Id).Build();
    var query = DataEnumerationQuery.For("CwDmBatch").FilterBy(filter).ProjectOn("Id").SetBatchSize(1000);

    foreach (var batch in dataApi.Enumerate(query))
    {
        dataApi.Delete(batch["Id"].ToString());
    }

    ctx.CommitChanges();
}
catch(Exception ex)
{
    Log.Error(ex, "Error in deleting existing batches");
    ctx.Set("Error", ex.ToString());
}
```

Scripted activity "Generate batches": Split the data collection into batches.

```
var dataApi = ctx.Use<IDataApi>();

var job = dataApi.Load(ctx.Input.Data["JobId"].ToString());

// Update progress
dataApi.Update(job.Id, new
{
    Progress = "Generating batches...",
});
ctx.CommitChanges();

var BATCH_SIZE = 1000;
var CUSTOM_WORKFLOW_EVENT = "CwDm Sample SalesInvoice Archive";
```

```
var batchOrder = 1;
var index = 0;
var itemIds = new List<string>();
object fromValue = null;
object toValue = null;

// Loop through all records using DataApi.Enumerate
var filter = FilterBuilder.Create().Build();
var query = DataEnumerationQuery.For("CwDmSampleSalesInvoice")
    .FilterBy(filter)
    .ProjectOn("Id", "CwDmLegacyId")
    .OrderBy("CwDmLegacyId");

foreach (var item in dataApi.Enumerate(query))
{
    if (fromValue == null) fromValue = item.Get<int>("CwDmLegacyId");
    toValue = item.Get<int>("CwDmLegacyId");
    itemIds.Add(item.Get<string>("Id"));

    index++;
    if (index == BATCH_SIZE)
    {
        CreateBatch(fromValue, toValue, itemIds);

        // Reset for new batch
        index = 0;
        fromValue = null;
        toValue = null;
        itemIds = new List<string>();
    }

    if (batchOrder % 100 == 0)
    {
        // Write to DB every 100 batches created
        ctx.CommitChanges();
    }
}

// Create final batch
if (itemIds.Any())
{
    CreateBatch(fromValue, toValue, itemIds);
    ctx.CommitChanges(); // Final write
}

// Mark the job that all batches are generated
dataApi.Update(job.Id, new
{
    IsBatchReady = true
});

string CreateBatch(object fromValue, object toValue, List<string> items)
{
    return dataApi.Add("CwDmBatch", new
    {
        // Standard
        JobId = job.Get<string>("Id"),
        ProcessId = job.Get<string>("ProcessId"),
        FromValue = JsonConvert.SerializeObject(fromValue),
        ToValue = JsonConvert.SerializeObject(toValue),
        TotalItems = items.Count,
        MigratedItems = 0,
        Status = "Ready",
        Progress = "",
```

```
        Order = batchOrder++,

        // Custom
        CustomWorkflowEvent = CUSTOM_WORKFLOW_EVENT,
        CustomItems = items
    });
}
```

The following is required when creating custom batches:

- `CustomWorkflowEvent`: The workflow event to process the batch. It is `CwDm Sample SalesInvoice Archive` in this example.
- `CustomItems`: The identifier of the data item needs to migrate.

Setting `FromValue` and `ToValue` is recommended so users can quickly identify the batch using the visual monitoring tool.

Step 3: Configure the workflow for batch processing

For this example to work, ensure you add a synchronous event trigger named **CwDm Sample SalesInvoice Archive**.

Below is an example of what a batch-processing workflow looks like. It simply loops through the items stored in the batches and updates data accordingly. The example also shows how you can run a test rule to verify data after migration.

```
var pluginApi = ctx.Use<IPluginApi>();
var dataApi = ctx.Use<IDataApi>();
var ruleApi = ctx.Use<IRuleApi>();

var batchId = ctx.Input["Data"]["BatchId"].ToString();
var PROGRESS_THRESHOLD = 100;
var TEST_RULE = "CwDm SO Test SalesInvoice Archive";
var progress = 0;
var total = 0;

try
{
    // Step 0: STANDARD - Extract event data
    var batch = dataApi.Load(batchId);
    var job = dataApi.Load(batch.Get<string>("JobId"));

    // Step 1: STANDARD - Query data
    IList<string> items = batch.GetList<string>("CustomItems");
    total = items.Count;

    // Step 2: STANDARD - Transform data and insert to DB
    foreach(dynamic item in items)
    {
        try
        {
            // STANDARD - Migrate data
            Migrate(batch, job, item);

            // STANDARD - Update progress
            UpdateProgress(batch, job, item);
        }
        catch(Exception ex)
        {
            Log.Error(ex, "Error in processing {@item}", item);
            throw;
        }
    }

    // Step 3: STANDARD - Commit the last progress
    dataApi.Update(batchId, new
    {
```

```
        IsTestPassed = (bool?)null,
        MigratedItems = progress,
        Progress = $"Migrated {Math.Round(1.0 * progress / total, 2) * 100}%"
    });
    ctx.CommitChanges();

    // Step 4: STANDARD - Execute test rule on the last item if test rule is configured
    if (items.Any() && !string.IsNullOrEmpty(TEST_RULE))
    {
        var testItem = items.Last();
        RunTest(batch, job, testItem);
    }
}
catch(Exception ex)
{
    // STANDARD - Throw the error
    Log.Error(ex, "Error in copying data");
    throw;
}

///// STANDARD FUNCTIONS
void UpdateProgress(dynamic batch, dynamic job, dynamic item)
{
    var partitionFieldName = job.Get<string>("CopyDataPartitionFieldName");

    progress += 1;
    if (progress % PROGRESS_THRESHOLD == 0)
    {
        Log.Verbose("Periodically update status: job {Job} batch {Batch}", job["Name"], batch["Order"]);
        dataApi.Update(batchId, new
        {
            MigratedItems = progress,
            Progress = $"Migrated {Math.Round(1.0 * progress / total, 2) * 100}%"
        });
        ctx.CommitChanges(); // Commit partially

        // Try to break the loop if the batch was stopped by user/system
        var stopRuleParameters = new Dictionary<string, object>();
        stopRuleParameters["BatchId"] = batchId;
        var shouldStop = ruleApi.Eval<bool>("CwDm Should Batch Stop", stopRuleParameters);
        if (shouldStop)
        {
            Log.Info("Stopping job {Job} batch {Batch}...", job["Name"], batch["Order"]);
            throw new Exception($"The batch {batchId} was stopped. The system will try to cancel all
change...");
        }
    }
}

void RunTest(dynamic batch, dynamic job, dynamic testItem)
{
    var testRule = TEST_RULE;
    var stopIfTestFailed = job.Has("StopIfTestFailed") && job.Get<bool>("StopIfTestFailed");

    var testRuleParameters = new Dictionary<string, object>();
    testRuleParameters["SourceItem"] = testItem;
    var testResult = ruleApi.Eval<bool>(testRule, testRuleParameters);
    dataApi.Update(batchId, new
    {
        MigratedItems = progress,
        Progress = $"Migrated {Math.Round(1.0 * progress / total, 2) * 100}%",
        IsTestPassed = testResult
    });
    ctx.CommitChanges(); // Commit partially

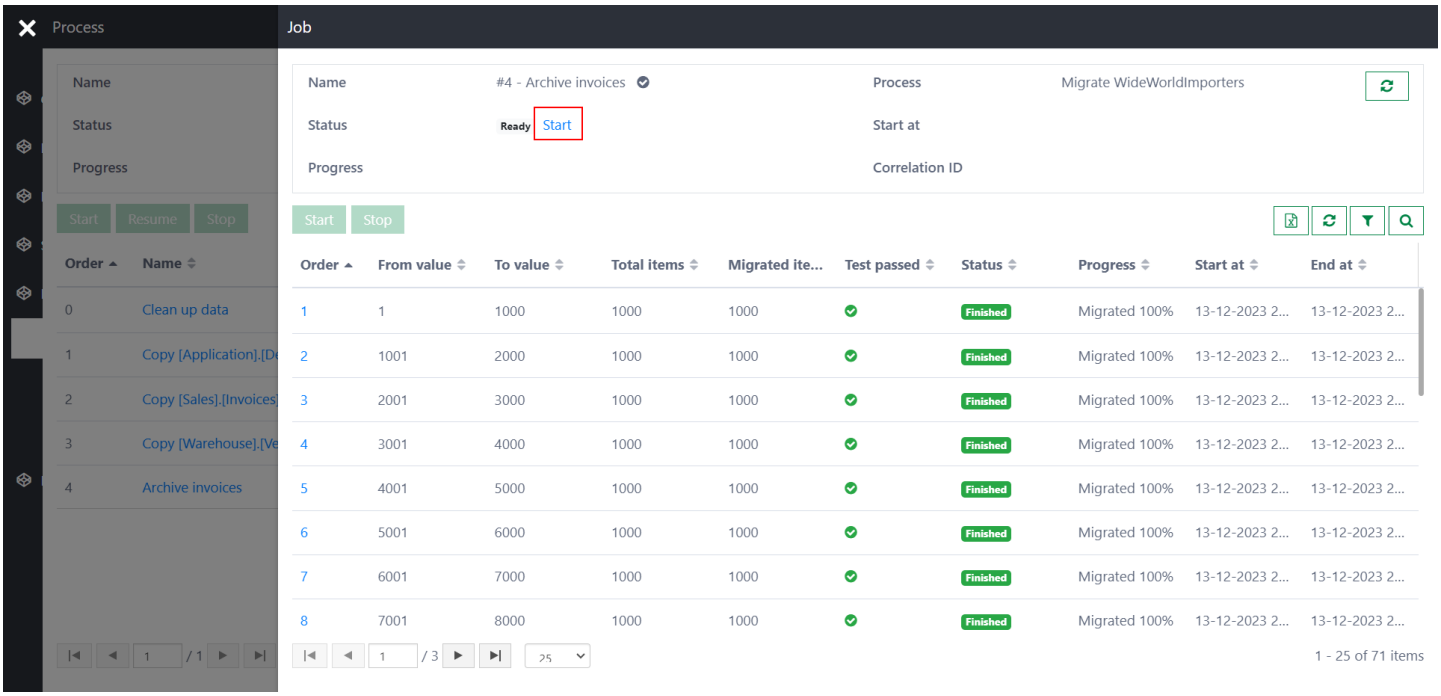
    Log.Debug("Test result for batch {Job} {Batch}: {TestResult}", job.Get<string>("Name"), batch.Get<int>
```

```
    Log.Debug("Test result for batch {Job} {Batch}: {testResult}", job.Get<string>("Name"), batch.Get<int>
("Order"), testResult);
    if (!testResult && stopIfTestFailed)
    {
        Log.Error("Test failed for batch {Job} {Batch}", job.Get<string>("Name"), batch.Get<int>("Order"));
        throw new Exception($"Test failed for batch {job.Get<string>("Name")} {batch.Get<int>("Order")}");
    }
}

void Migrate(dynamic batch, dynamic job, string item)
{
    // You custom migration logic
    dataApi.Update(item, new { IsArchived = true });
}
```

Step 4: Run and verify

Open the job **Archive invoices** and click **Start**. If everything is configured correctly, you will see batches are generated and run sequentially.

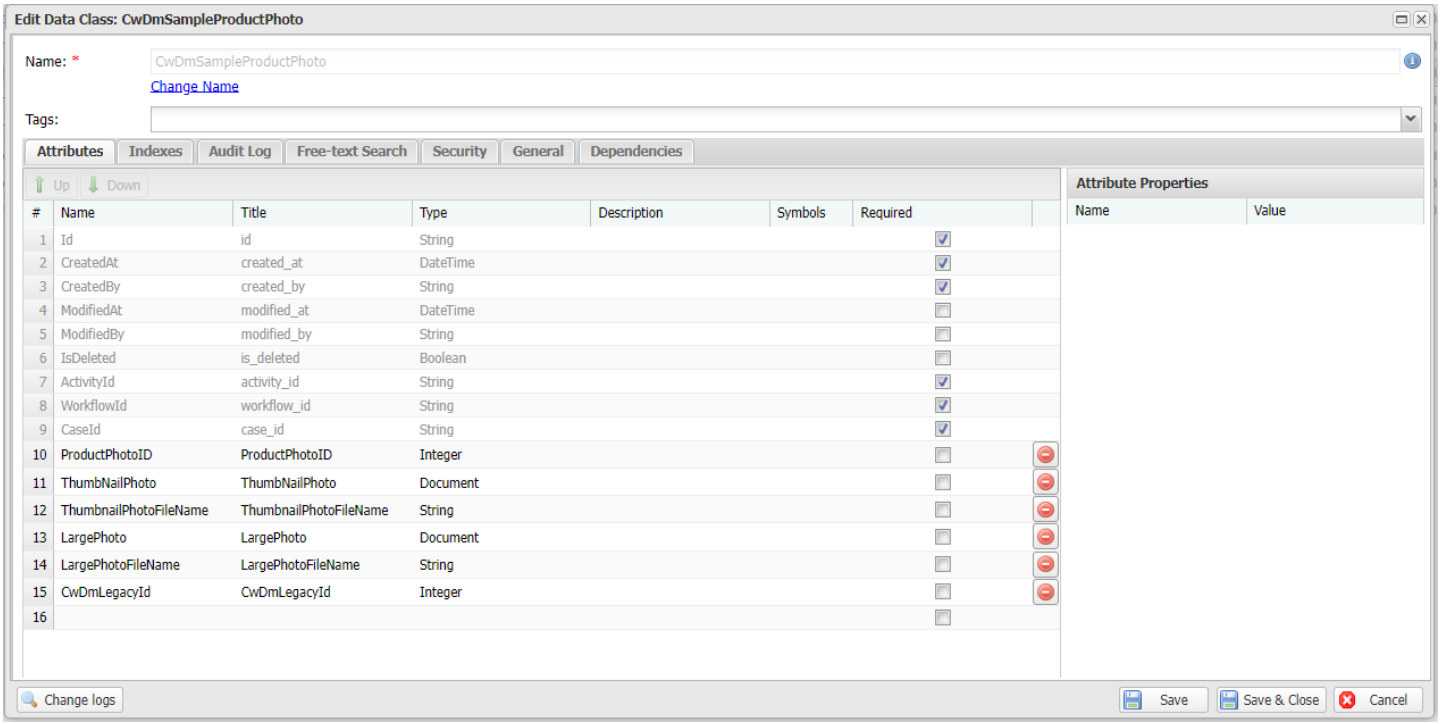


Component: Data migration - Working with files

Assuming you are migrating product photos from the well-known database **AdventureWorks2019**. The article will show how the component handles files when migrating data.

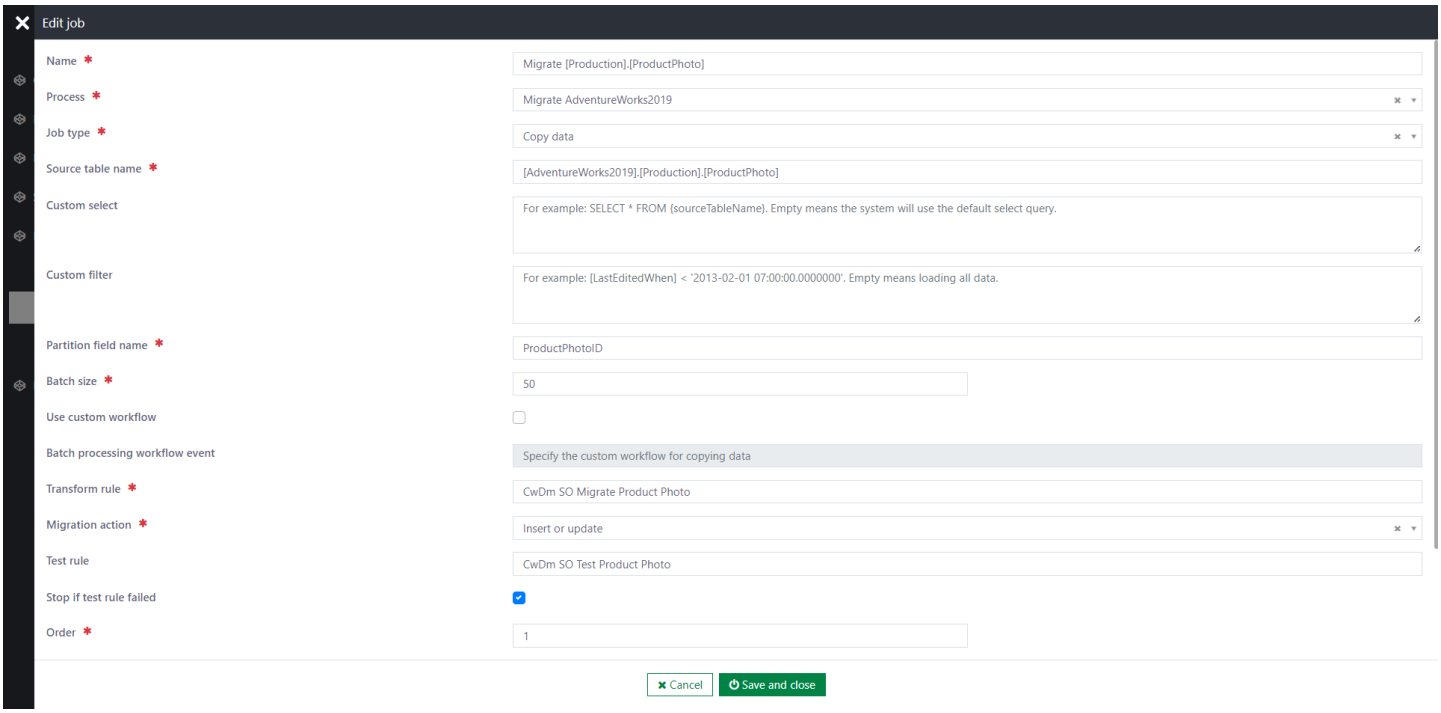
Step 1: Setting up the data class for storing files

Configure a data class **CwDmSampleProductPhoto** for storing data migrated from the table **[AdventureWorks2019].[Production].[ProductPhoto].** The files will be stored in **ThumbNailPhoto** and **LargePhoto**, which have the type **Document**. Don't forget to add the field **CwDmLegacyId** required for tracking data.



Step 2: Configure a copy-data job

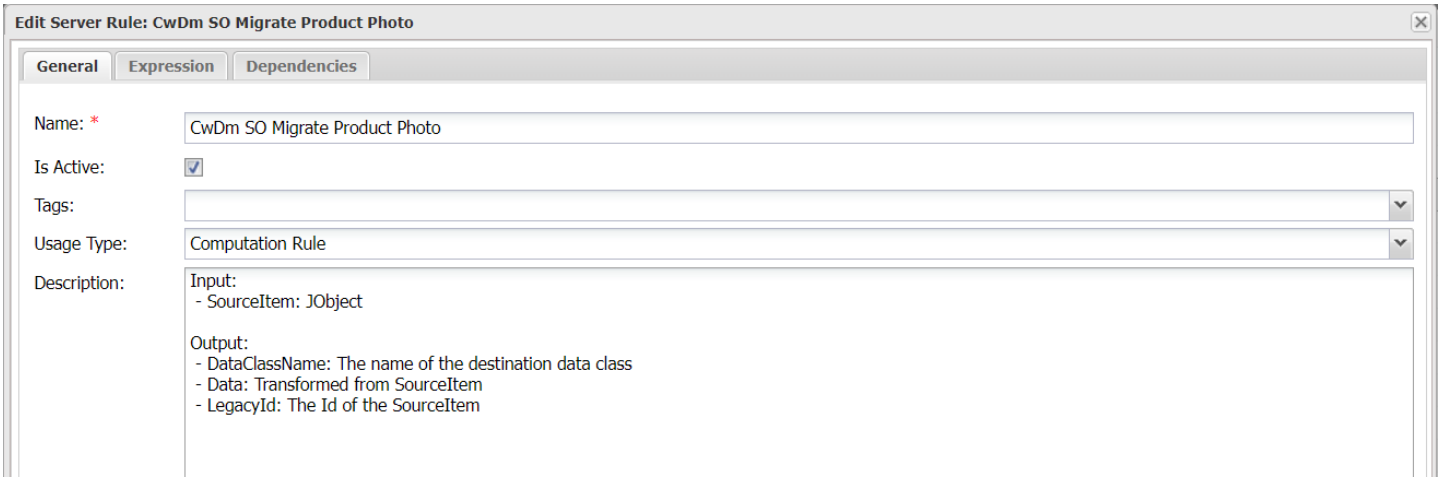
Configure a copy-data job as described below:



We will implement the **Transform rule** and the **Test rule** in the following steps.

Step 3: Implement the data-transform rule

Add a new Casewhere rule **CwDm SO Migrate Product Photo**:



We will use `IDocumentApi` to store the files. Remember to check null before proceeding.

```
#predicate
var documentApi = @apiFactory.Get<IDocumentApi>();
var item = @parameters["SourceItem"];

DocumentInfo thumbnailDocInfo = null;
if (item["ThumbNailPhoto"] != null)
{
    thumbnailDocInfo = documentApi.Create(item["ThumbnailPhotoFileName"]);
    documentApi.Store(thumbnailDocInfo, Convert.FromBase64String(item["ThumbNailPhoto"].ToString()));
}

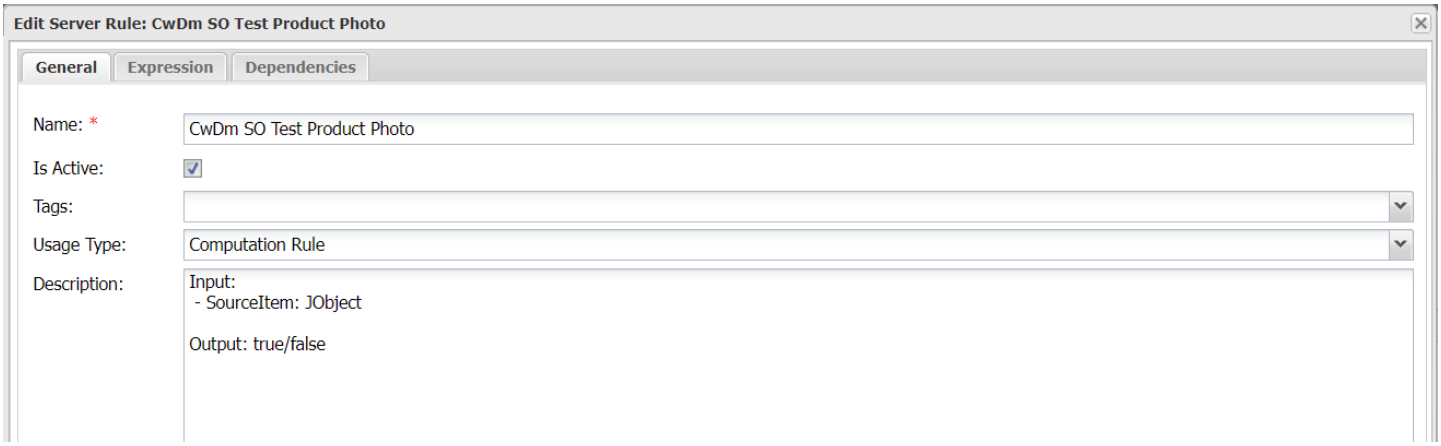
DocumentInfo largeDocInfo = null;
if (item["LargePhoto"] != null)
{
    largeDocInfo = documentApi.Create(item["LargePhotoFileName"]);
    documentApi.Store(largeDocInfo, Convert.FromBase64String(item["LargePhoto"].ToString()));
}

var data = new
{
    ProductPhotoID = item["ProductPhotoID"],
    ThumbnailPhotoFileName = item["ThumbnailPhotoFileName"],
    ThumbNailPhoto = thumbnailDocInfo,
    LargePhotoFileName = item["LargePhotoFileName"],
    LargePhoto = largeDocInfo,
    CwDmLegacyId = item["ProductPhotoID"]
};

// You can either return a DynamicDataObject or a list of DynamicDataObjects
var migratedItem = new DynamicDataObject(new {
    DataClassName = "CwDmSampleProductPhoto",
    Data = data,
    LegacyId = item["ProductPhotoID"]
});
return new List<DynamicDataObject>(){ migratedItem };
```

Step 4: Implement the data-verification rule

Add a new Casewhere rule **CwDm SO Test Product Photo**:



Remember to log errors that fail the test.

```
#predicate
var item = @parameters["SourceItem"];

var dataApi = @apiFactory.Get<IDataApi>();

var filter = FilterBuilder.Create().Eq("CwDmLegacyId", item["ProductPhotoID"]).Build();
var migratedItem = (dataApi.Search(DataObjectApiQuery
    .For("CwDmSampleProductPhoto")
    .Paging(0, 1)
    .FilterBy(filter)).Data as IList<DynamicDataObject>).FirstOrDefault();

if(migratedItem == null) {
    Log.Error("CwDmSampleProductPhoto not found for {ProductPhotoID}", item["ProductPhotoID"]);
    return false;
}

if(migratedItem.Get<DocumentInfo>("ThumbNailPhoto") == null && item["ThumbNailPhoto"] != null)
{
    Log.Error("ThumbnailPhoto is missing");
    return false;
}

if(migratedItem.Get<string?>("ThumbnailPhotoFileName") != item["ThumbnailPhotoFileName"])
{
    Log.Error("ThumbnailPhotoFileName is not matched. Expected={Expected}. Actual={Actual}",
item["ThumbnailPhotoFileName"], migratedItem.Get<string?>("ThumbnailPhotoFileName"));
    return false;
}

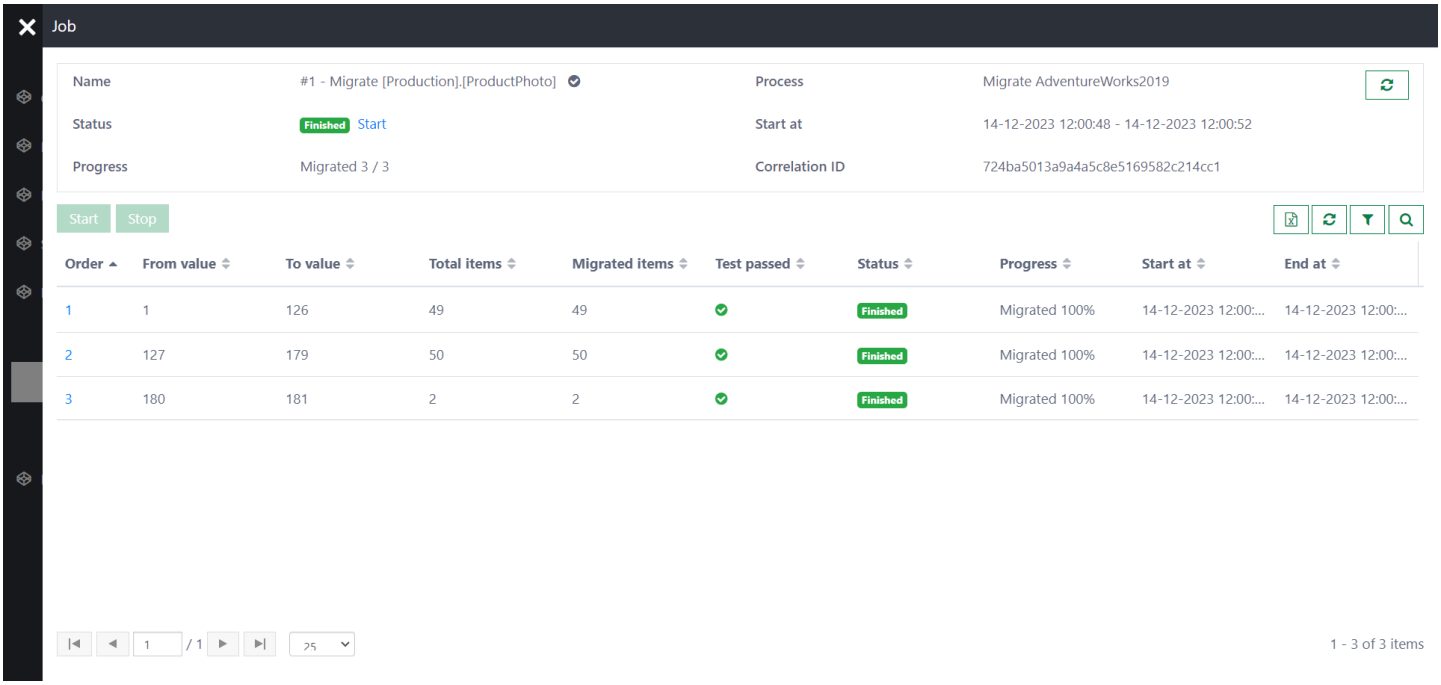
if(migratedItem.Get<string?>("LargePhotoFileName") != item["LargePhotoFileName"])
{
    Log.Error("LargePhotoFileName is not matched. Expected={Expected}. Actual={Actual}",
item["LargePhotoFileName"], migratedItem.Get<string?>("LargePhotoFileName"));
    return false;
}

if(migratedItem.Get<DocumentInfo>("LargePhoto") == null && item["LargePhoto"] != null)
{
    Log.Error("LargePhoto is missing");
    return false;
}

return true;
```

Step 5: Run and verify

Open the job **Migrate [Production].[ProductPhoto]** and click **Start**. If everything is configured correctly, you will see batches are generated and run sequentially.



Component: Data Export

Introduction

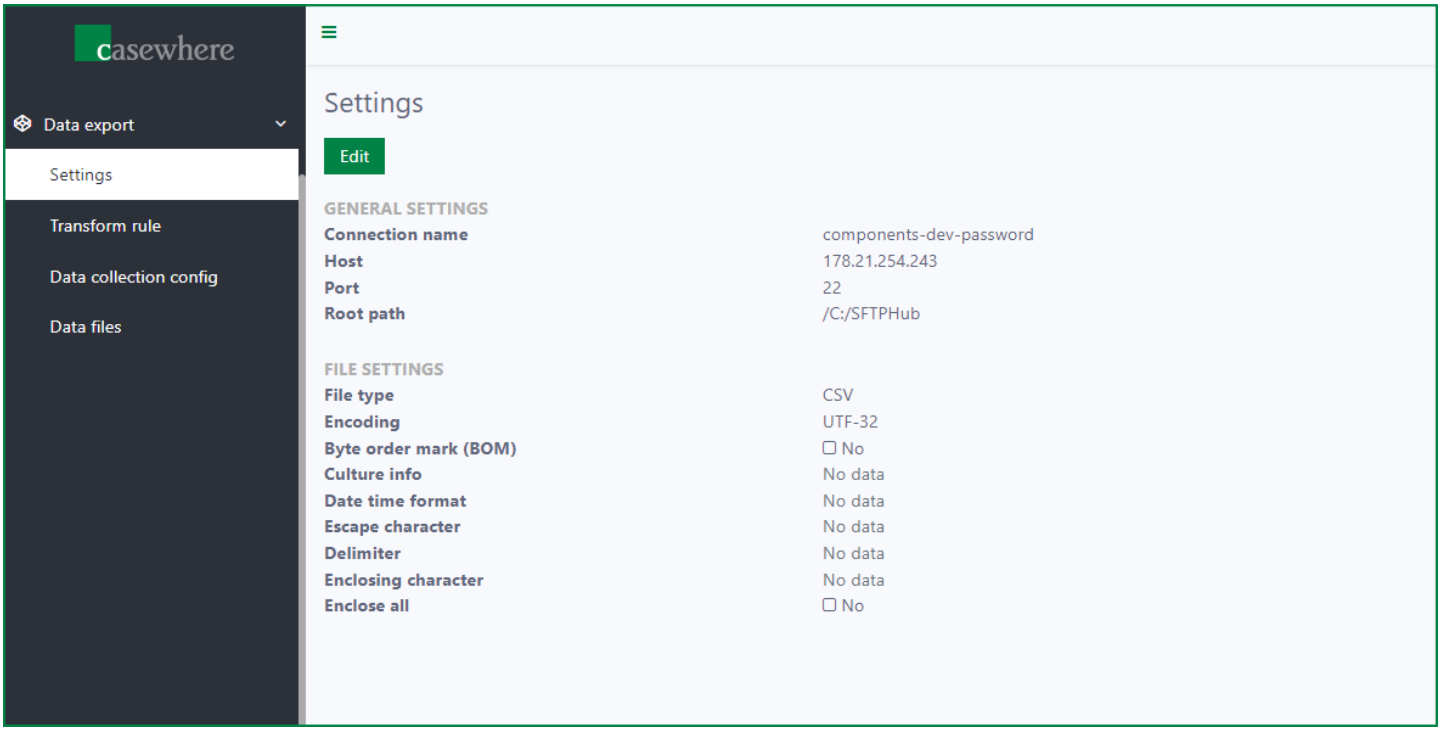
Data export component is the general component that supports the user to set up a job to export the data into a specific format (CSV for now) by setup a data collections, transform rule. After exporting the data, the user can monitor the exporting process using the data file management.

Features

Settings

Settings page is used to manage the SFTP connection where all the data export files will be uploaded after the data export process is successfully completed, and the settings for exported files.

The page displays the saved settings in read-only state. You can edit the settings. However, creating a new setting or deleting the setting is not supported.

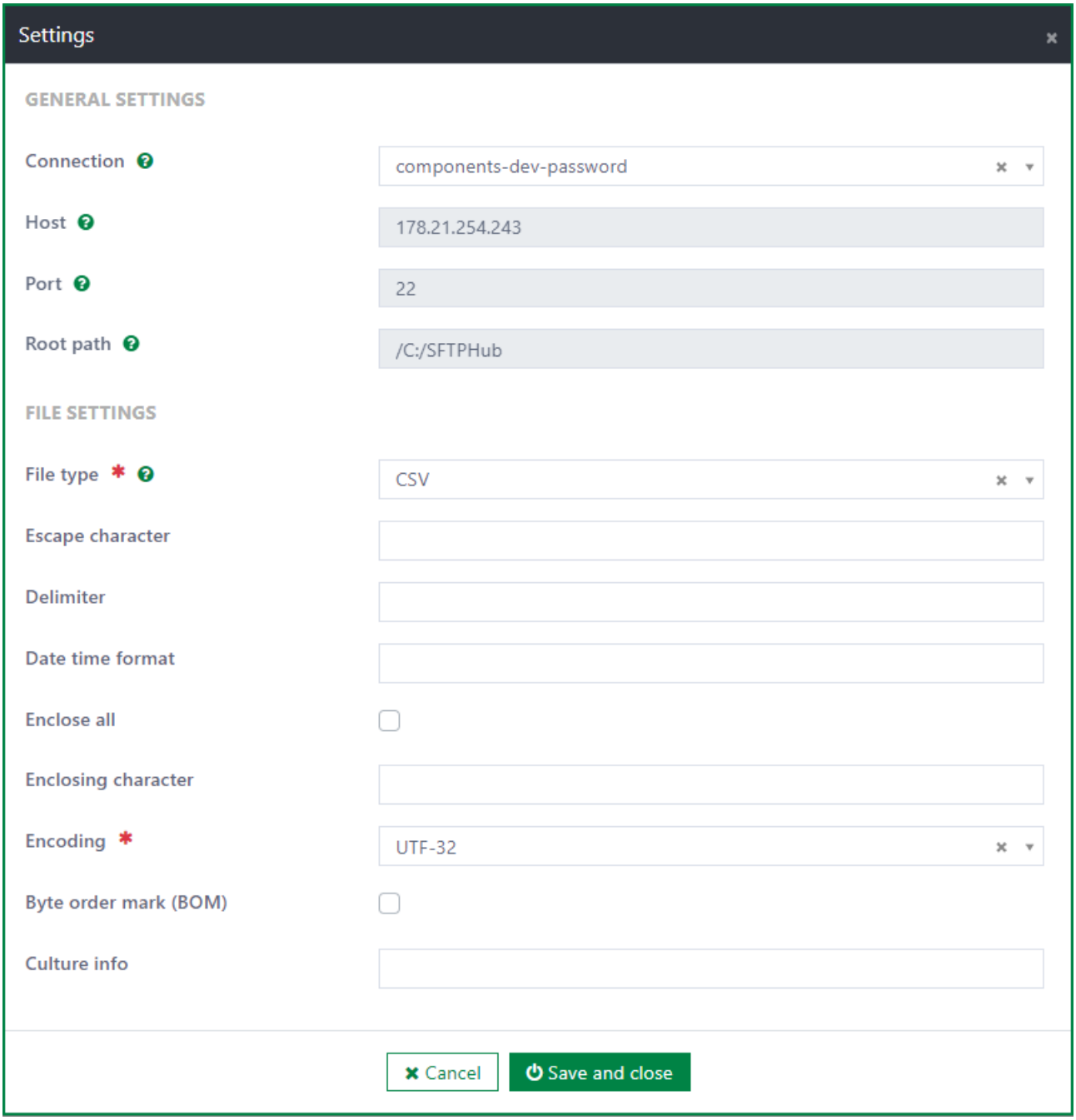


Edit setting

To edit the setting, click on **Edit** button



After that, the **Edit Settings** dialog will be opened with two sections: **General settings**, and **File settings**



General settings

The General settings section is used to configure the SFTP connection. It includes 4 fields:

- **SFTP Connection**: This field is required. It is a dropdown list with all SFTP connections in system. You must select 1 connection.
- **Host**: This field is always disabled and auto filled according to the selected SFTP connection.
- **Port**: This field is always disabled and auto filled according to the selected SFTP connection.
- **Root path**: This field is always disabled and auto filled according to the selected SFTP connection.

Whenever the SFTP connection is changed, the **Host**, **Port**, **Root path** field s will be also changed.

To learn more about how to setup SFTP connection, read SFTP hub.

File settings

The **File settings** section is used to configure the settings relating to the exported data files. If those settings are not configured, the CSV settings will be applied. If you set up those settings, the CSV settings will be overridden.

Explore more about CSV plugin

- **File type**
    - This field is required.
    - It is a dropdown list with all file types, for now, only CSV is supported.
    - After the file type CSV is selected, the additional fields are displayed.
- **Escape character**
    - This field is optional and only displayed if file type is CSV.
    - The field is used to define the escape character in CSV data file. The default escape character in CSV is double quote if the field is not specified. If it is specified, the specified escape character will be used instead of the default one.
    - If a field value contains a character which is the same to the enclose character, then, the escape character will be added in front of the character to identify it with the enclose character.
    - E.g: if the Address field value is **Block B ; Tower C ; 123.st**, the escape character is double quote, and the enclose character is semicolon, then, in the data file, the double quote will be added in front of the semicolons within the field value, and whole value will be enclosed by 2 semicolons. And, the Address field value will be displayed in data file as **;Block B "; Tower C "; 123.st;**
- **Delimiter**
    - This field is optional and only displayed if file type is CSV.
    - The delimiter character is used to separate fields in CSV file.
    - The delimiter character is specified as *colon* by default. If the delimiter character is not specified, the delimiter will be defined following the **Culture info** field value.
        - E.g: If Culture info is **da-DK**, the delimiter character is *semicolon* . If Culture info is **en-DK**, the delimiter character is *colon*.
- **Date time format**
    - This field is optional and only displayed if file type is CSV.
    - The date time format is specified as **dd.MM.yyyy HH:mm**. If the **Date time format** field is not specified, the date time format will be defined following the **Culture info** field value.
        - E.g: If culture info is **en-GB**, the date time format is **dd MMMM yyyy HH:mm:ss**.
- **Enclose all**
    - This field is optional and only displayed if file type is CSV.

- By default, the checkbox is unticked. It means not all fields will be enclosed.
- If the checkbox is ticked, all fields and header in the file will be enclosed.

- **Enclosing character**
  - This field is optional and only displayed if file type is CSV.
  - The default enclose character is *double quote* if it is not specified. If it is specified, the specified character will be used instead of the default one.
  - Only the field with some special characters will be enclosed. Those are:
    - The field value contains delimiter character.
    - The field value contains enclosing character.
    - The field value contains leading space.
    - The field value contains trailing space.
    - The field value contains line breaks.

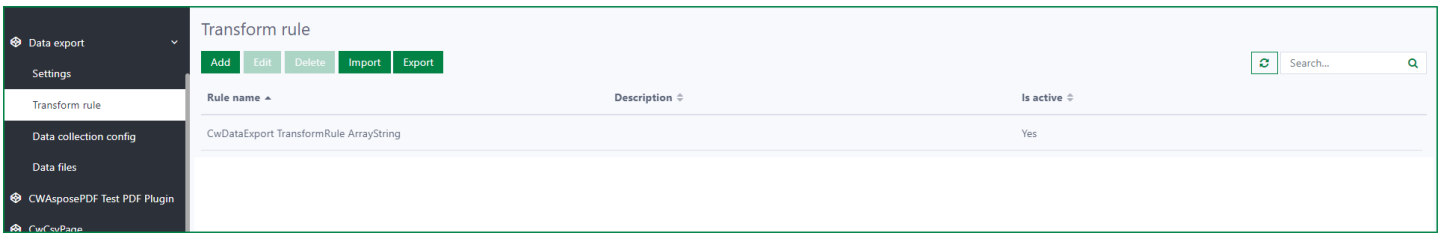
- **Encoding**
  - This field is required and only displayed if file type is CSV.
  - The field stipulates the encoding of the output file.
  - User must select one value from the list of encoding types. The list includes values:
    - UTF-8
    - UTF-32
    - Unicode
    - Big Endian Unicode
    - ASCII

- **Byte order mark (BOM)**
  - This field is optional and only displayed if file type is CSV.
  - The checkbox is unticked by default. When the checkbox is ticked, the data file will include BOM.

- **Culture info**
  - This field is optional and only displayed if file type is CSV.
  - It is the culture information of output file. The default value is current culture information of the web server. The field value will override the plugin configuration if it is specified.
  - The **Culture info** value stipulates the delimiter character and date time format in the file if these fields are not specified. The decimal separator also depends on the **Culture info** value.

Transform rules

Transform rule is used for data transformation. **Transform rules** page is used to manage all the transform rules. You can add, edit, delete, import, export the transform rules.



Transform rules list

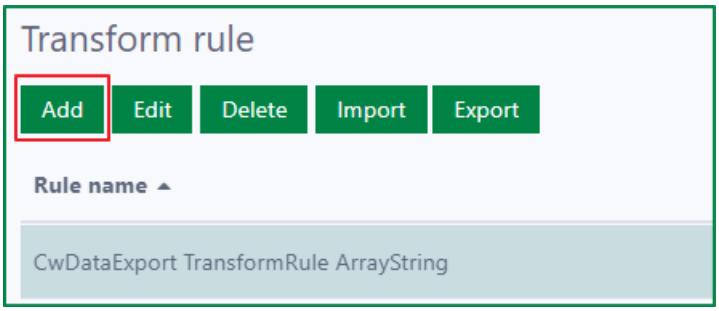
In the Transform rules list, you can view the information of:

- **Rule name:** is the configured name of the rule. The rule name is unique.

- **Description:** is the description of the rule. It is usually used to describe the information of the rule.
- **Is active:** is status of the rule. If a rule is active, it can be used in the configuration of **Data collection configs**.

Add a new transform rule

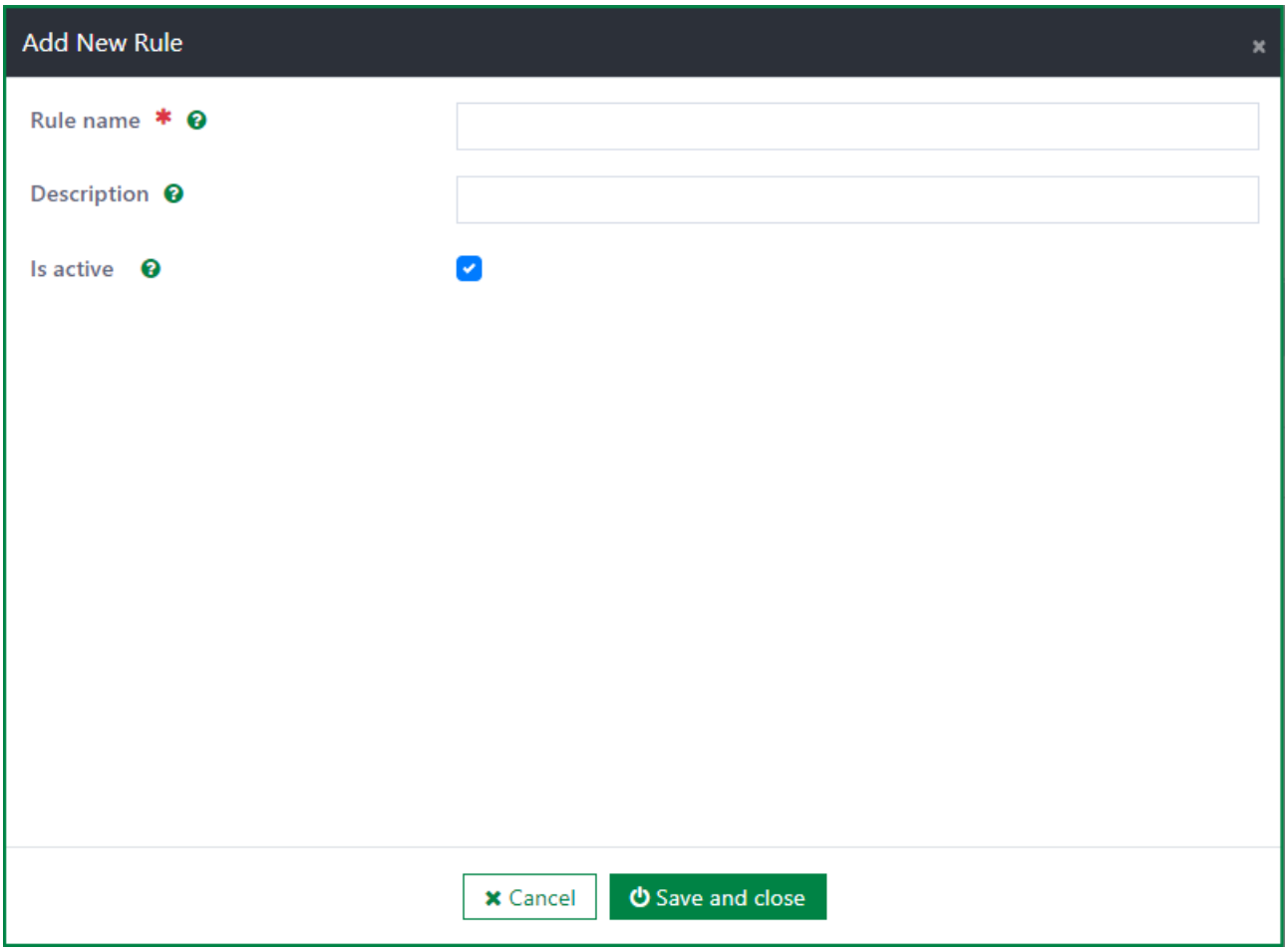
To add a new transform rule, first, click on the the **Add** button to open the **Add new rule** dialog



Then, input value for the following fields:

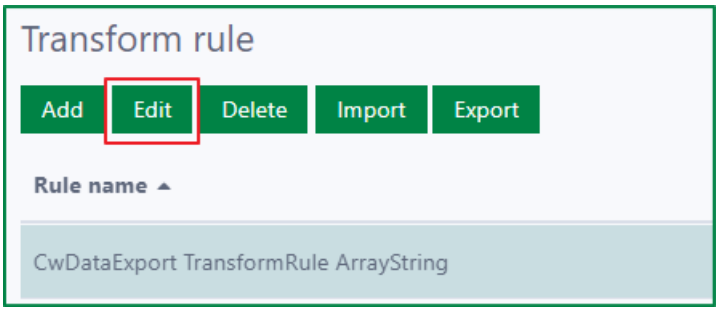
- **Rule name**
  - This field is required.
  - The rule name should be the same to the relevant rule name which was added in backend.
- **Description**
  - This field is optional.
  - This field is used to describe the information of the rule.
- **Is active**
  - If this checkbox is ticked, the rule will be active and can be used in **Data collection configs**.
  - Else, the rule will be hidden in the configuration dialog of **Data collection configs**.

After filling out the fields, click on the **Save and close** button to add the rule.



Edit a transform rule

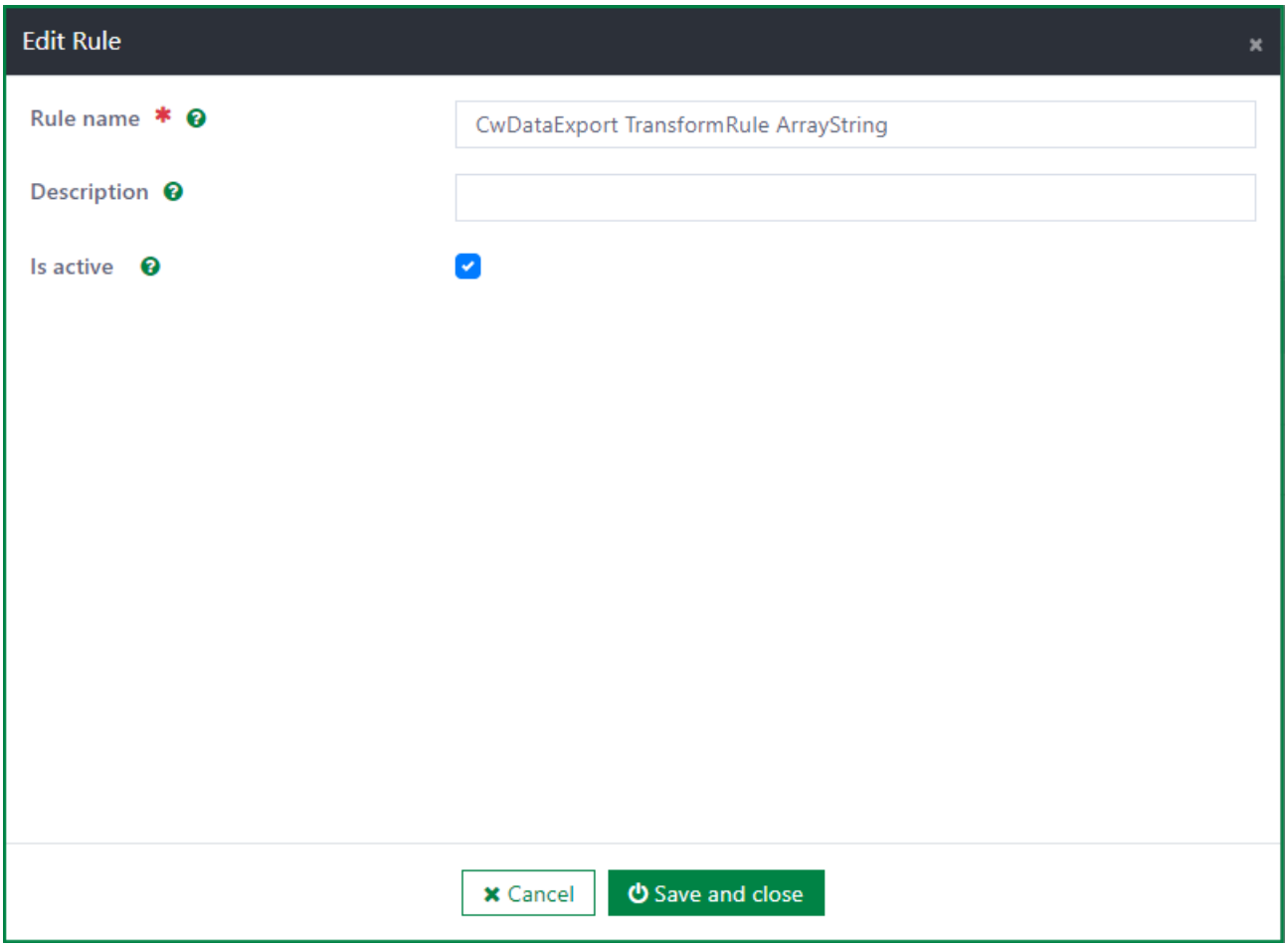
To edit a transform rule, first, select a rule, then click on the **Edit** button to open the **Edit Rule** dialog



In the **Edit Rule** dialog, you can update the fields:

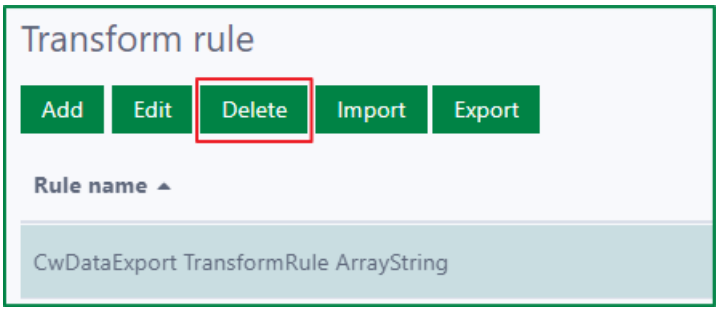
- **Rule name**
- **Description**
- **Is active**

After updating the fields, click on the **Save and close** button to update the rule. If you do not want to save the changes, click on the **Cancel** button.

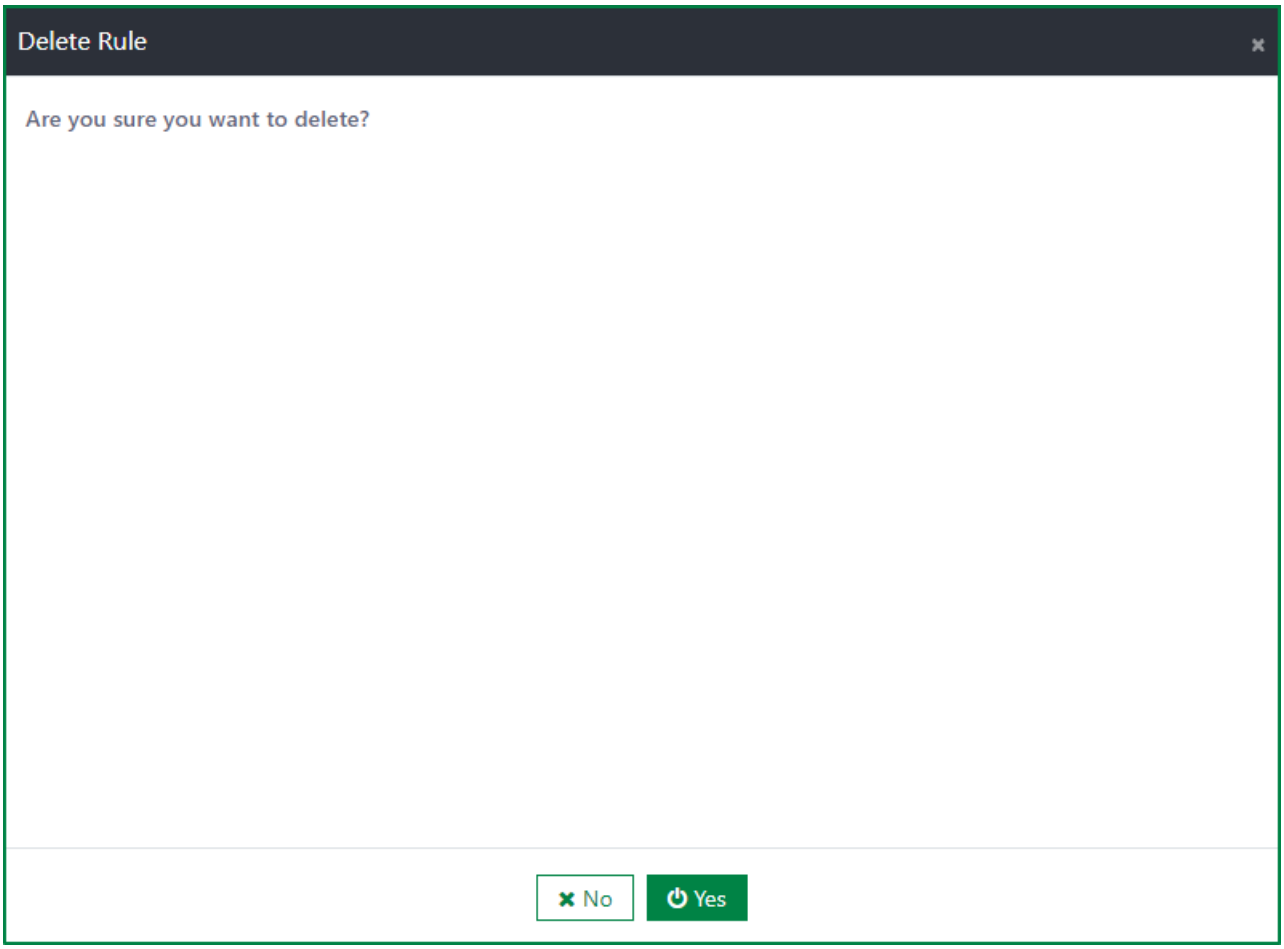


Delete a transform rule

To delete a transform rule, first, select a rule. Then, click on the **Delete** button



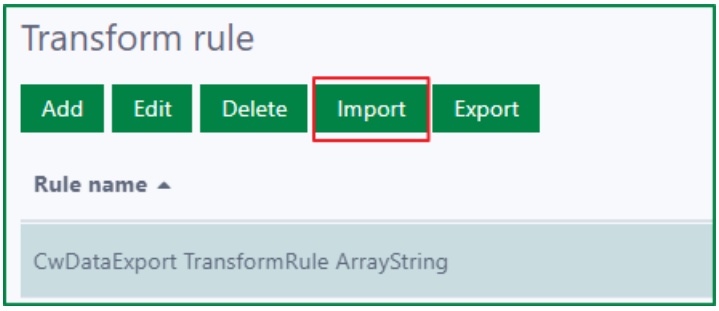
If the rule is not used for any **Data collection configs**, a confirmation dialog will be shown. If you select the **Yes** button, the rule will be deleted immediately. If you select the **No** button, the deletion will be cancelled.



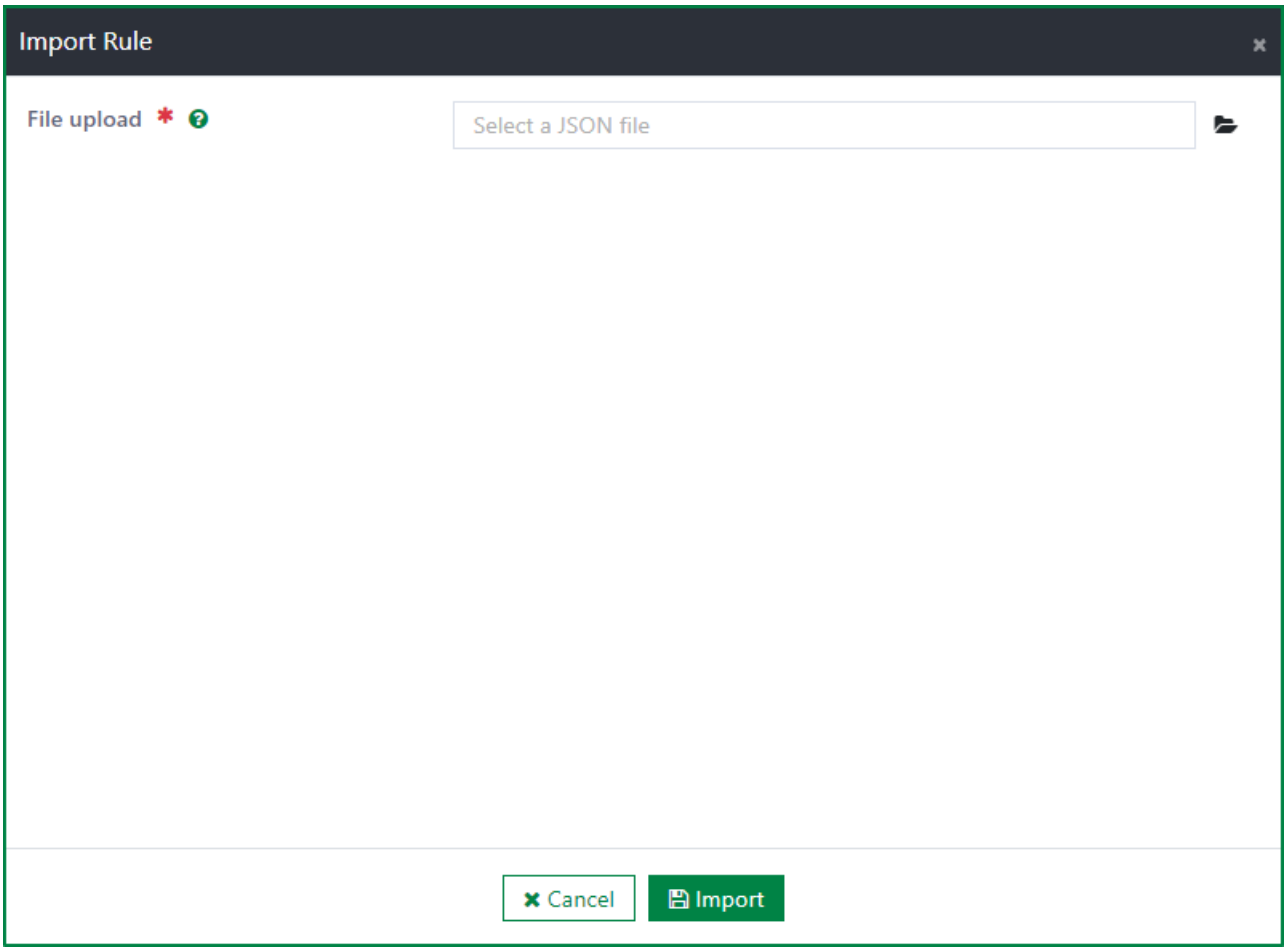
If the rule was already used in a **Data collection configs**, the warning message dialog will be displayed to inform the disallowed deletion

Import the transform rules

To import the transform rules, first, click on the **Import** button to open the **Import Rule** dialog



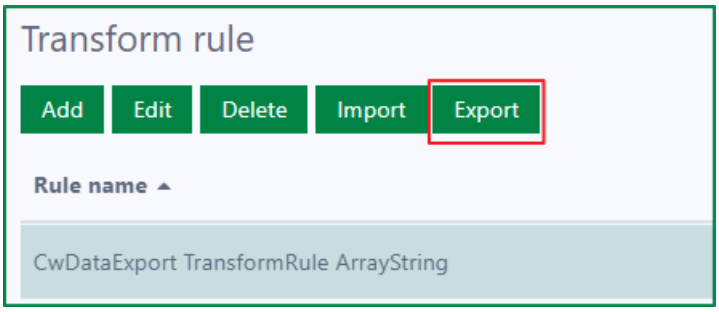
Then, click on the placeholder of the **File upload** field to open file selection dialog



Next, select a JSON file to import (multiple selection is not supported). After selecting the file, click on the **Import** button to import the rules from the JSON file. If you do not want to execute the import process, click on **Cancel** button.

If the imported transform rule has the same name to an existing one in the Transform rules page, the existing one will be overridden by the new one.

Export the transform rules



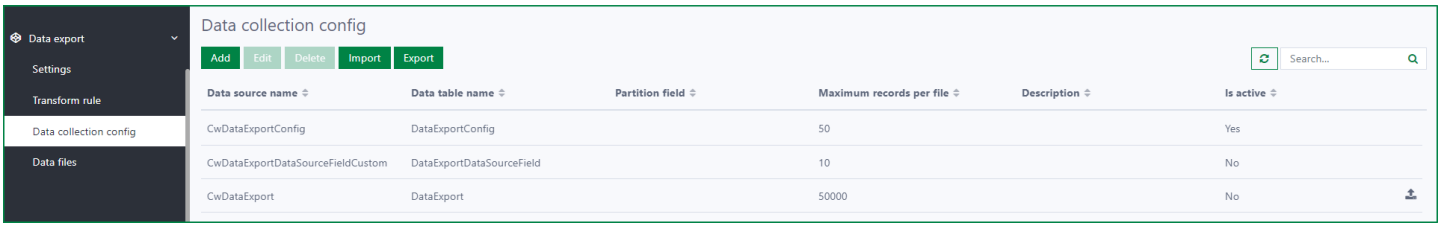
To export 1 or several rules in the list, select the rules, then click on the **Export** button. Then, the JSON file of the selected rules will be downloaded immediately.

To export all rules in the list, click on **Export** button. Then, the JSON file of all rules will be downloaded immediately.

Data collection configs

Data collection config is the configuration for the data included in the exported data file, such as: data source, maximum records...

In **Data collection configs** page, you can view all the created configs, also add, edit, delete, import, export the configs.



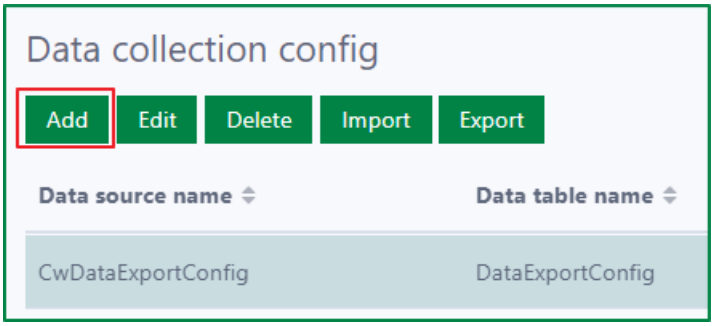
Data collection configs list

In **Data collection configs** list, you can view all configs with the information of:

- **Data source name:** is name of the data source selected in the config
- **Data table name:** is the input data table name of the data source
- **Partition field**: is the date time field which is used as condition to query data
- **Maximum records per file**: is the number of maximum records can be included in 1 file
- **Description**: is the description of the config. It is usually used to describe the information of the config.
- **Is active**: is status of the config. If the config is active, the relevant data files will be sent whenever the scheduler job is run.

Add a new Data collection config

To add a new **Data collection config**, first, click on the **Add** button



Then, in **Add Data collection config** dialog, fill out the fields:

- **Data source name**
  - This field is required.
  - You must input name of the data source.
  - List of **Data source fields** and **Partition fields** will be generated according to the input data source name.
- **Data table name**:
  - This field is optional.
  - **Data table name** can be used as an element in generating exported file name. If **Data table name** is empty, **Data source name** will be used in exported file name instead.
  - This field must not include the following characters **|*:"<>/?**
- **File name pattern**
  - This field is required.
  - The field is used to configure the name of the exported file.
  - The default file name pattern is **[[data*table_name]]*[[timestamp@yyyy.MM.dd]]**
  - The pattern must not contain any of the following characters **\ / : * ? " < > |**. These characters are disallowed characters in a file's path.

- You can add a fixed file name, or configure a dynamic file name by using the following elements in file name pattern:
  - **[[data_table_name]]**: Use it if you want to insert the data table name into file name. If data table name is empty, the data source name will be inserted instead.
  - **[[timestamp@]]**: Use it if you want to insert the timestamp into file name. is used to define the specific format for the timestamp, e.g: [[timestamp@yyyy.MM.dd HH:mm]]
  - **[[file_index]]**: Use it if you want to insert the index of the file into file name. If you do not specify it, the system will add the index at the end of the file name automatically in case the data are separated to multiple files.

- **Partition field**

  - This field is optional.
  - List of Partition fields includes all the date time fields belonging to the input data source name.
  - Partition field is used as a date time filter to query data. If the partition field is specified, system will query data having partition field value greater than the partition field value of last record in the previous exported file for the new exported file.
    - E.g: **Partition field** is **SubmittedDate**. The **SubmittedDate** of last record in the previous exported file is 31.05.2023 23:50. Then, in new exported file, only records having **SubmittedDate** from 31.05.2023 23:51 are included.

- **Maximum records per file**

  - This field is optional.
  - The field defines the maximum records can be included in 1 file. If the record amount exceeds the maximum number, the data will be separated to multiple files.

- **Description**

  - This field is optional. This field is used to describe the information of the config.

- **Is active**

  - If this checkbox is ticked, the relevant data files will be sent whenever the scheduler job is run.

- **Generate upload folder everyday**

  - If the checkbox is ticked, new folder will be created everyday to store the relevant exported file.

- **Folder name pattern**

  - This field is optional if the **Generate upload folder everyday** checkbox is unticked.
  - It will be mandatory if the **Generate upload folder everyday** checkbox is ticked.
  - The field is used to configure the folder name that contains the exported files.
  - Folder name patter must not contain the following characters **\ / : * ? " < > |**
  - You can add a fixed folder name, or configure a dynamic folder name with the following elements:
    - **[[data_table_name]]**: Use it if you want to insert the data table name into folder name. If data table name is empty, the data source name will be inserted instead.
    - **[[timestamp@]]**: Use it if you want to insert the timestamp into folder name. is used to define the specific format for the timestamp, e.g: [[timestamp@yyyy.MM.dd HH:mm]]. If it is not specified, system will insert the timestamp to folder name automatically with format **yyyyMMddHHmm**.

- **Data source fields**:

  - By clicking on **+** button, the additional fields to fill the information of the data source field will be displayed. You can add multiple data source fields. After adding the data source field, you must select the attribute field, and configure additional information for it with the following fields:

    - **Name**:

- This field is required.
- You must select 1 attribute from the list
- List of attribute fields is retrieved from the selected Data source.

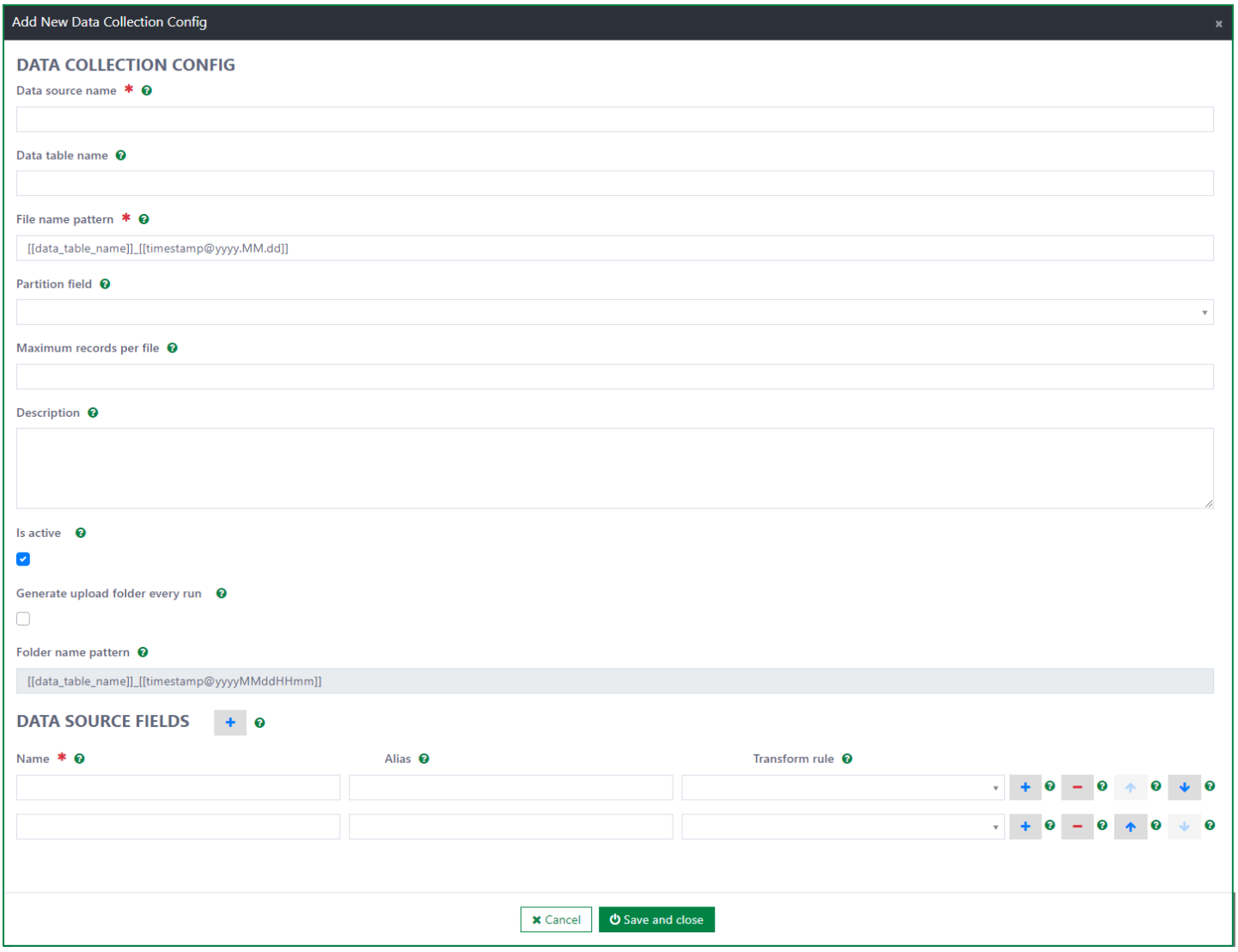
- **Alias**

  - This field is optional.
  - This field is used to input the alias for the selected attribute field.
  - In the exported file, the Alias is used as header of the attribute field. If Alias is not specified, the attribute field name will be used instead.

- **Transform rule**

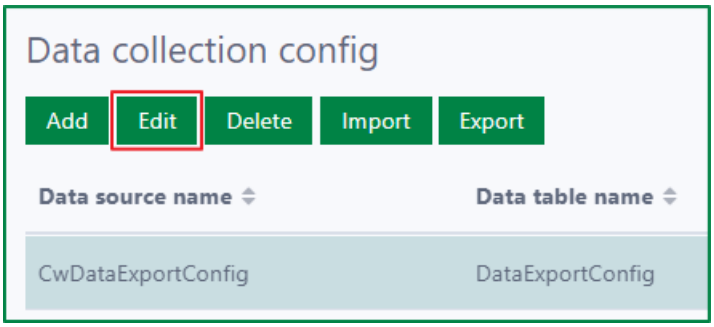
  - This field is optional.
  - This field is used to define the transformation rule for the data of the attribute in the exported file.
  - List of transformation rules is retrieved from the **Transform rules** page.

After filling the information, click on **Save and close** button to complete adding the config.



Edit a Data collection config

To edit a Data collection config, first, click on **Edit** button to open **Edit Data collection Config** dialog



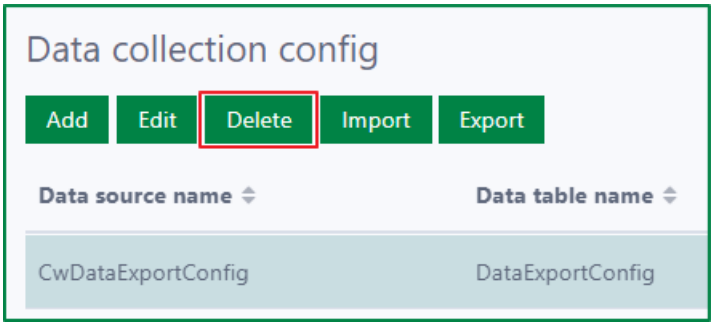
In the **Edit Data collection Config** dialog, you can edit:

- **Data source name**
- **Data table name**
- **File name pattern**
- **Partition field**
- **Maximum records per file**
- **Description**
- **Is active**
- **Generate upload folder everyday**
- **Folder name pattern**
- **Data source fields**:
    - **Name**
    - **Alias**
    - **Transform rule**

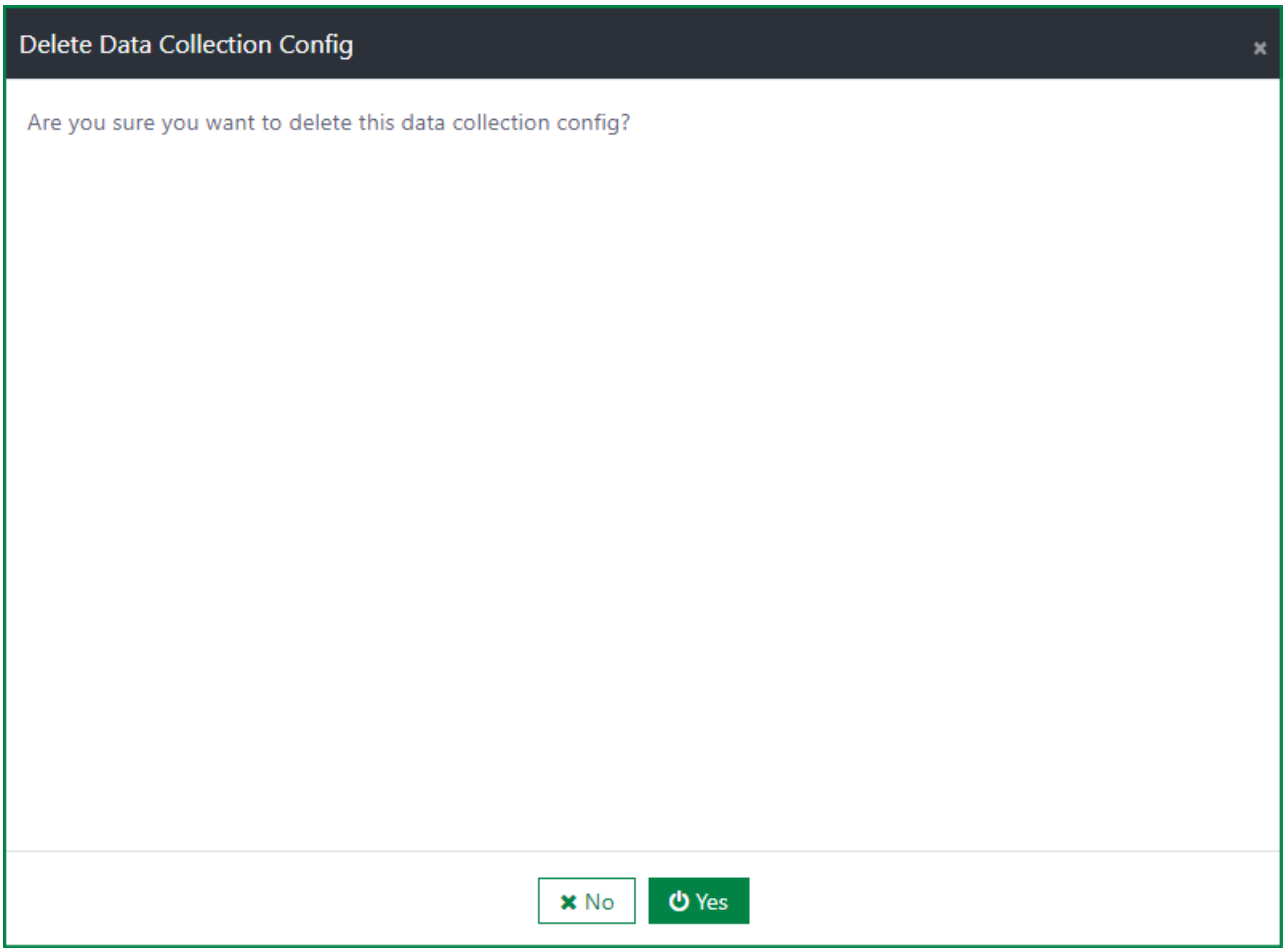
After updating the information, click on **Save and close** button to save the changes. Or you can click on **Cancel** button to discard the changes.

Delete a Data collection config

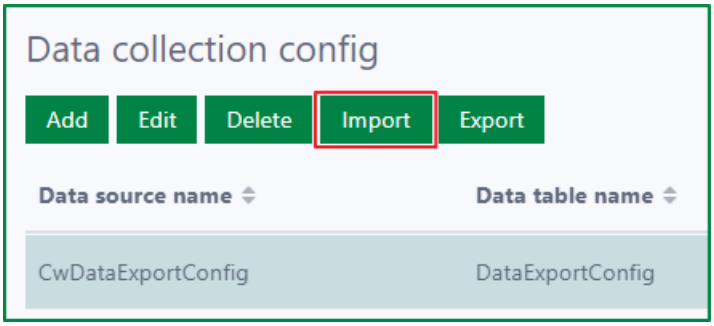
To delete a **Data collection config**, first, click on the **Delete** button



Then, a confirmation dialog will be displayed. If you select the **Yes** button, the config will be deleted immediately. If you select the **No** button, the deletion will be cancelled.



Import the Data collection configs



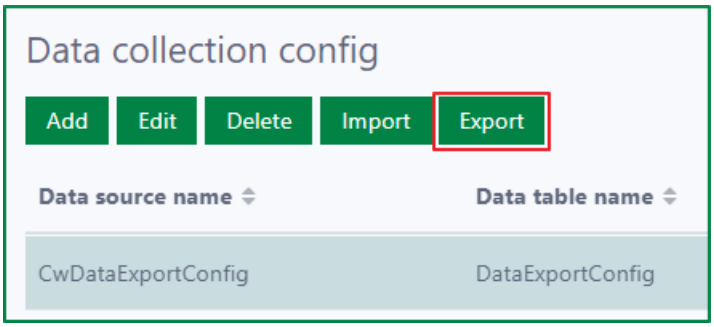
To import the Data collection configs, first, click on the **Import** button to open the **Import Data collection Configs** dialog

Then, click on the placeholder of the **File upload** field to open file selection dialog

Next, select a JSON file to import (multiple selection is not supported). After selecting the file, click on the **Import** button to import the rules from the JSON file. If you do not want to execute the import process, click on **Cancel** button.

If the imported data collection config has the same name to an existing one in the Data collection configs page, the existing one will be overridden by the new one.

Export the Data collection configs



To export 1 or several configs in the list, select the configs, then click on the **Export** button. Then, the JSON file of the selected configs will be downloaded immediately.

To export all configs in the list, click on **Export** button. Then, the JSON file of all configs will be downloaded immediately.

Scheduler job overview

The Scheduler job will get the data based on the active **Data collection configs** and only get the data that has a Partition field within the range from the last successful run until the current time. The data is sorted ascending by the Partition field. If no Partition field is defined, then the system will get all the data.

In the background, the job will calculate and divide the data into multiple files based on the maximum record per files value.

After finishing the preparing data process and all of the CSV files are generated successfully, the system will trigger the uploading process and uploaded to the SFTP server which is configured in **Settings** after the **Scheduler job** is run successfully. If there are one or more files is generated fail, the job will stop working.

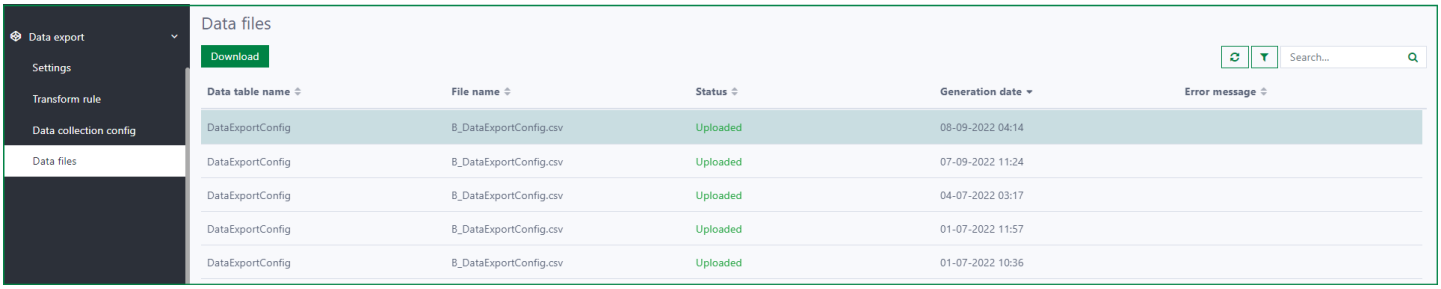
During uploading, if there is one more file is uploaded failed, the job will revert the uploading process by removing the successfully uploaded file to the SFTP server.

All data files and status are managed in the **Data files** page.

Data files management

Data files page is used to manage the data files which are uploaded to the SFTP server. In this page, you can view all the data files with the status and other information (file name, generated date, etc.), and download the data files.

Data files list



In **Data files** list, you can view the information of the data files, they are:

- **Data table name:** is the configured **Data table name** in the relevant **Data collection configs.** If **Data table name** is not configured, **Data source name** will be displayed instead.
- **File name:** is the name of the uploaded data files. If there are multiple data files generated from a **Data collection config** at one job running time. The data files will have the numeral suffix, for example: *yyyy.MM.dd_data table name(s)_1.csv*, : *yyyy.MM.dd_data table name(s)_2.csv*
- **Status:** is the status of the data files. There are 7 statuses:
    - **New:** the data file has just been created.

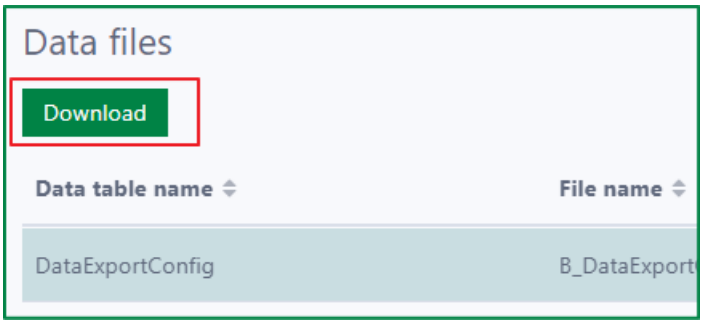
- **Preparing:** the data file is in preparation.
- **Preparing failed:** the preparation for the data file has failed.
- **Prepared:** the preparation for the data file has completed.
- **Uploading:** the uploading of the data file to SFTP server is in progress.
- **Uploading failed:** the uploading of the data file to SFTP server has failed.
- **Uploaded:** the data file has been uploaded to SFTP server successfully.

- **Generation date:** is the date time when the data file preparation has started.
- **Error message**: is the error message when the data file preparation or data file uploading processes has failed.

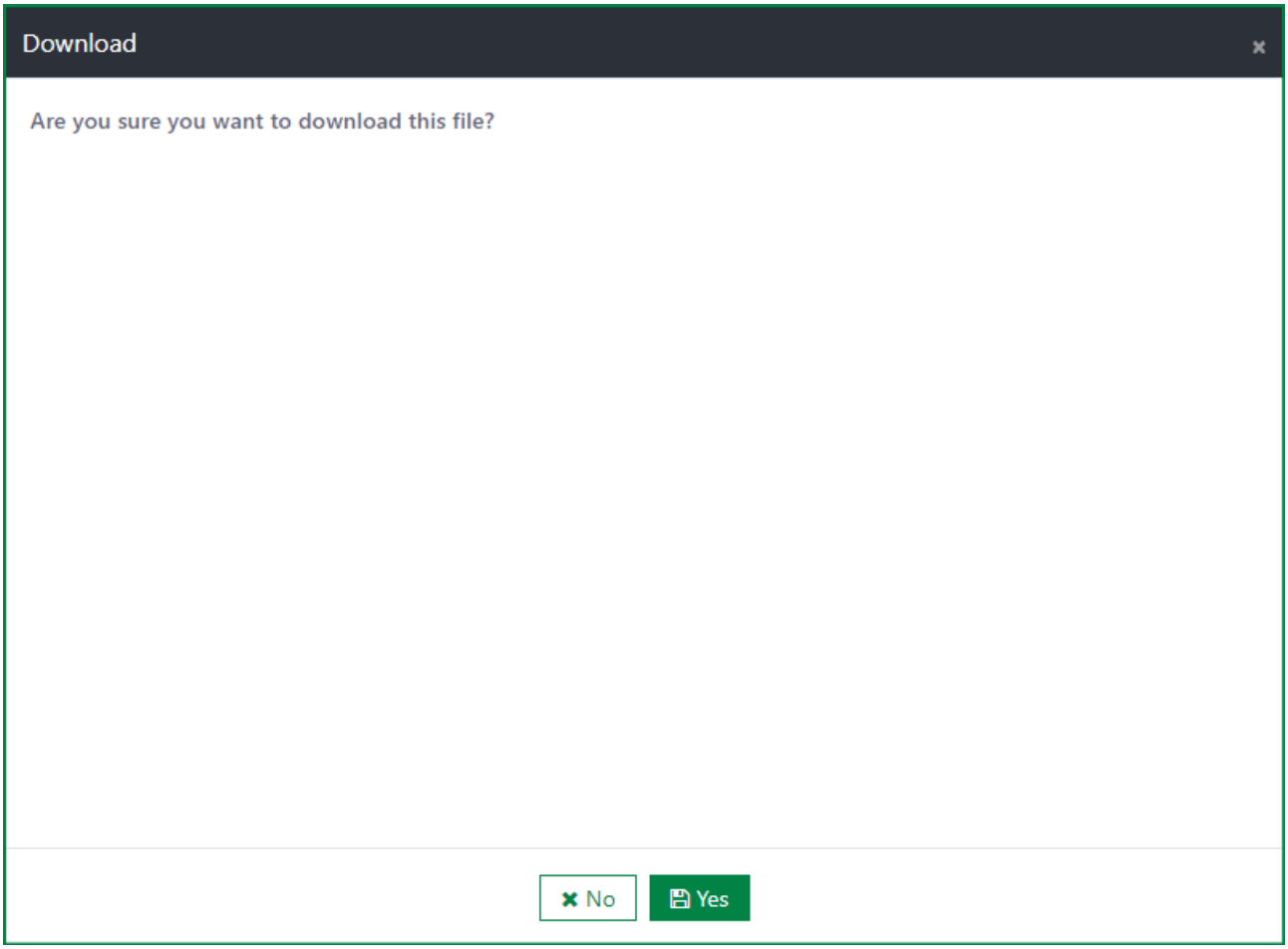
Download data files

You can only download the data files after it is prepared successfully. The download-able files has one of these status: **Prepared, Uploading, Uploading failed, Uploaded.** Only 1 data file can be downloaded at a time.

To download the data file, first, select the data file and click on the **Download** button



After that, a confirmation dialog will be opened. To continue downloading the file, click on the **Yes** button. If you want to cancel the downloading, click on the **Cancel** button.



- # Installation

  Requirements

  - Casewhere 2.6.0 or later.

  Configuration

  1. Import products: **Casewhere SFTP Hub**, **Product-Cw CSV** and **Casewhere Data Export**
  2. Link the administration page to your worker sites and configure access control
  3. Configure the SFTP hub folder and SFTP connection
  4. Configure the Data Export settings

  Dependencies

  - Component: SFTP hub
  - Plugin: CSV

Releases

1.1.0 - 12/04/2022

Changelog

- Support file settings
- Support reordering data source fields
- Support showing data source fields in the dropdown
- Support showing header for empty CSV file
- More friendly UI for data file management

Download (login required): Data Export v1.1.0

1.0.0 - 24/02/2022

Changelog

- Data export settings
- Data collection configuration
- Transform rules management
- Data files management

Download (login required): Data Export v1.0.0

Roadmap

- More format
- Support different data storage: Sharepoint, Azure storage rather than SFTP

Component: IdP Sync

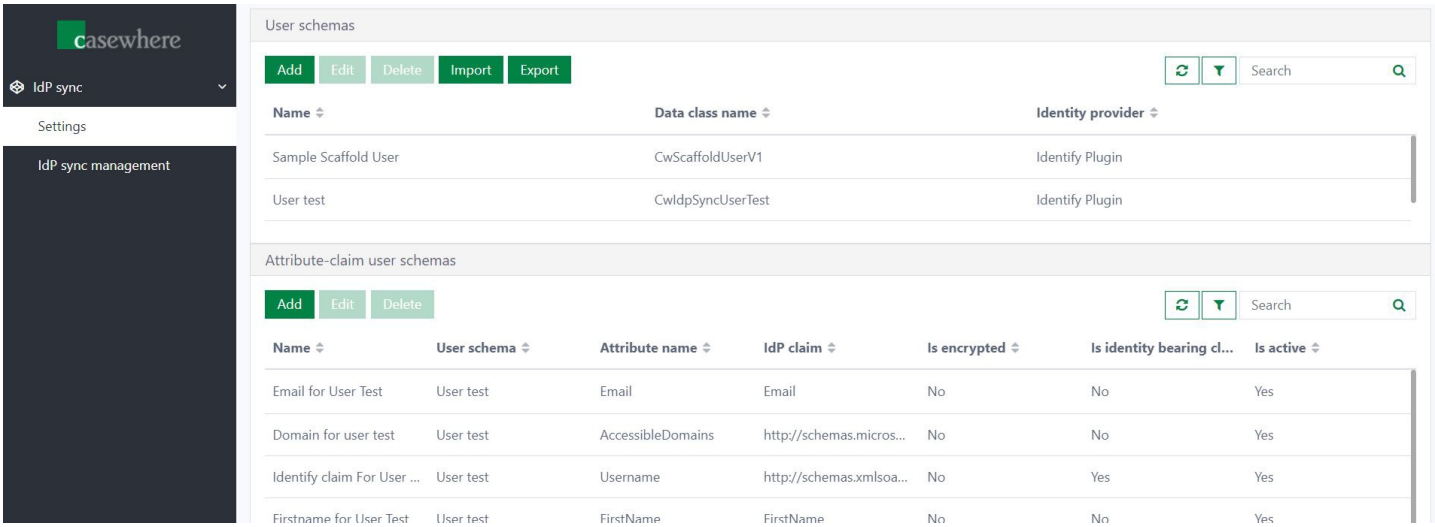
Introduction

User management is a feature that often appears in most of applications. It allows to create, edit or delete users in the given identity provider (IdP) such as Safewhere Identity Provider, Azure Active Directory (AAD), OneLogin. The way for creating users in IdP is the same but the user information in each application is different. To close this matter, the **IdP sync** component provides the mechanism to synchronize users from Casewhere to IdP for any change.

Features

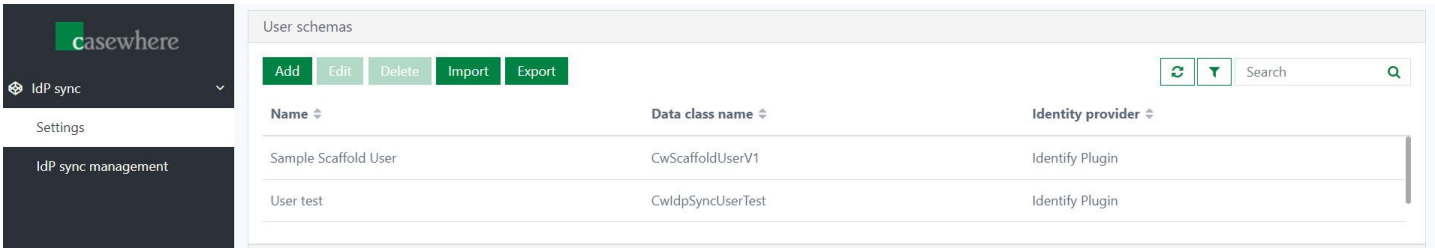
Settings

Settings page provides two data grids. The first one is used to manage the user schema. You can add, edit, delete, import, export user schemas. The second grid is attribute claims of user schema, you can add, edit, and delete attribute claims. If you select one user schema, then all attribute claims of selected user schema will reveal in the attribute claims grid.



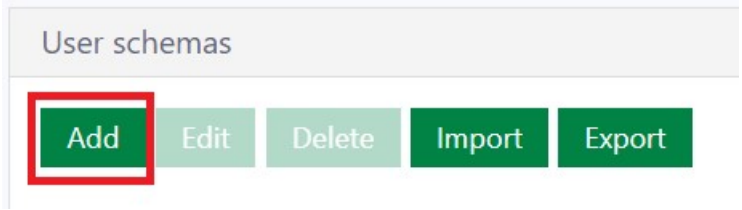
Manage user schemas

The user schema is the way to determine which data class in Casewhere is used to store the users and which identity provider we want to use to create the corresponding users.



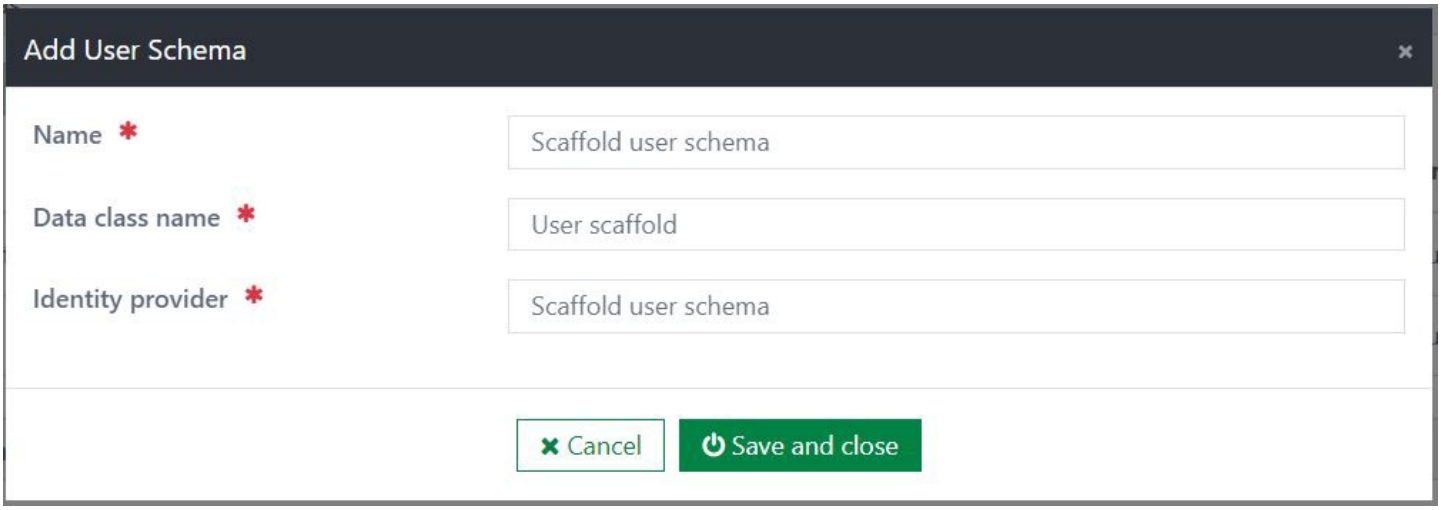
Add new user schema

To add a new user schema, click on the the **Add** button to open the **Add User Schema** dialog



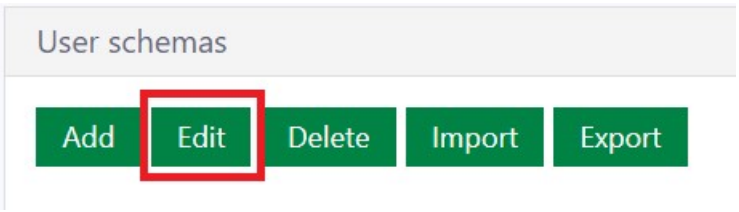
Then, input value for the fields:

- **Name:** The name of user schema. This field is required and unique.
- **Data class name:** The data class name in Casewhere where is the place to store users. This field is required and unique.
- **Identity provider:** The plugin name in Casewhere. This field is required.
    - Example: *Identify Plugin*



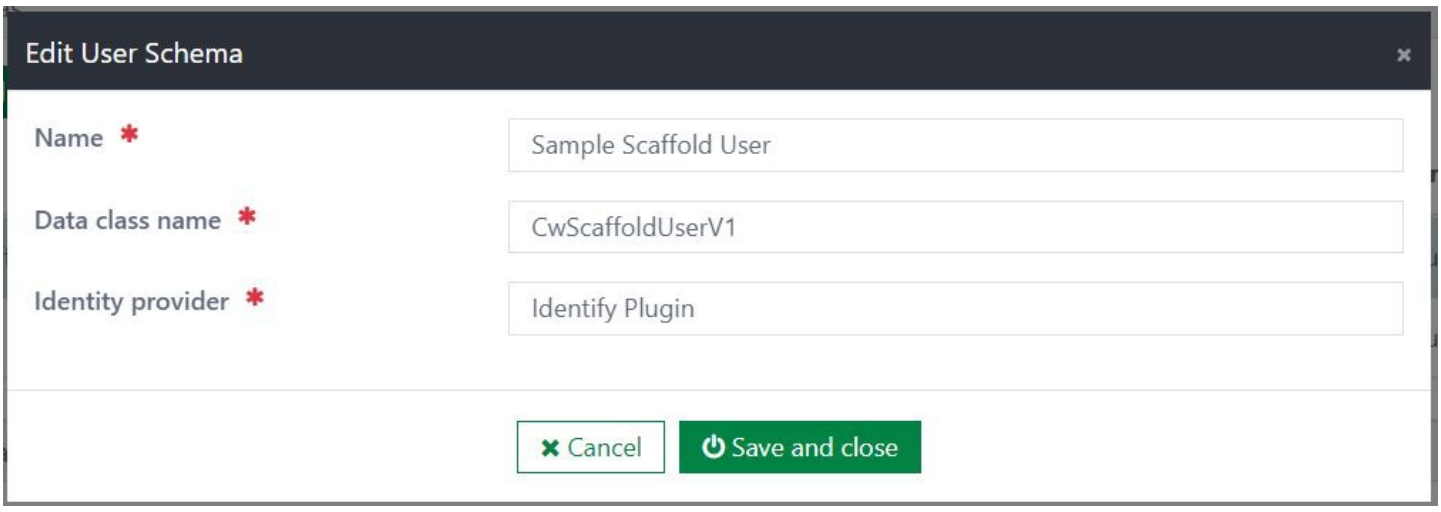
Edit user schema

To edit user schema, you must select a user schema. Then click on the **Edit** button to open **Edit User Schema** dialog



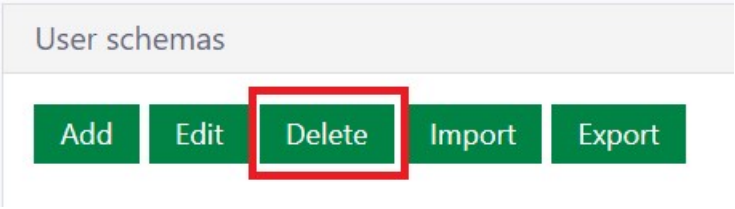
In **Edit User Schema** dialog, you can update the fields:

- **Name**
- **Data class name**
- **Identity provider**



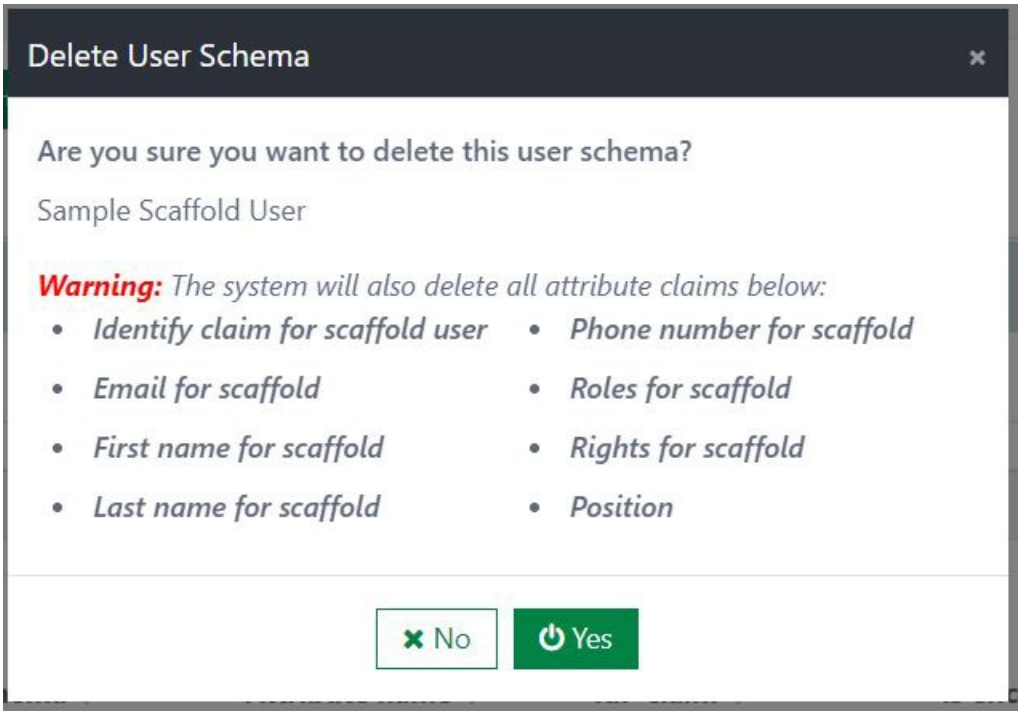
Delete user schema

To delete user schema, select a user schema. Then, click on the **Delete** button



The confirmation dialog will be shown. If you select **Yes** button, the user schema will be deleted immediately. If you select **No** button, the deletion will be cancelled.

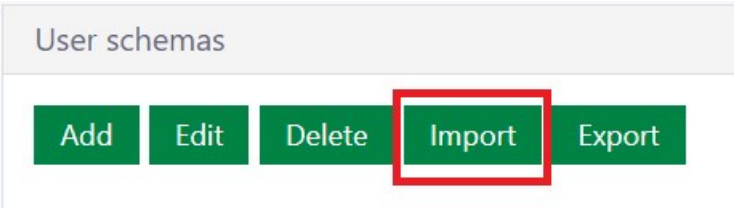
- **Note:** The system will also delete all attribute claims which are linked to user schema.



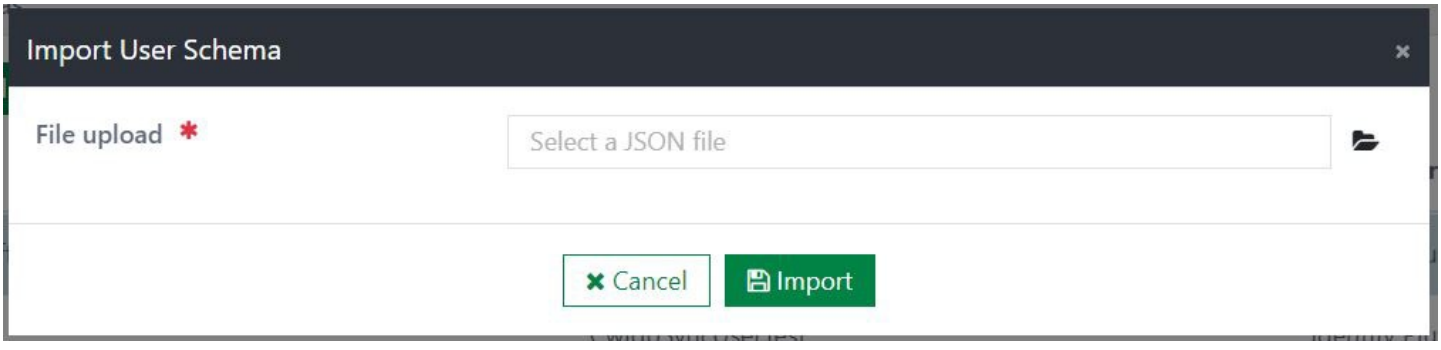
Import user schemas

To import user schemas, first, click on **Import** button to open the **Import User Schema** dialog

- **Note:** All user schemas that exists in the destination environment will be overridden. (Detected by user schema name)

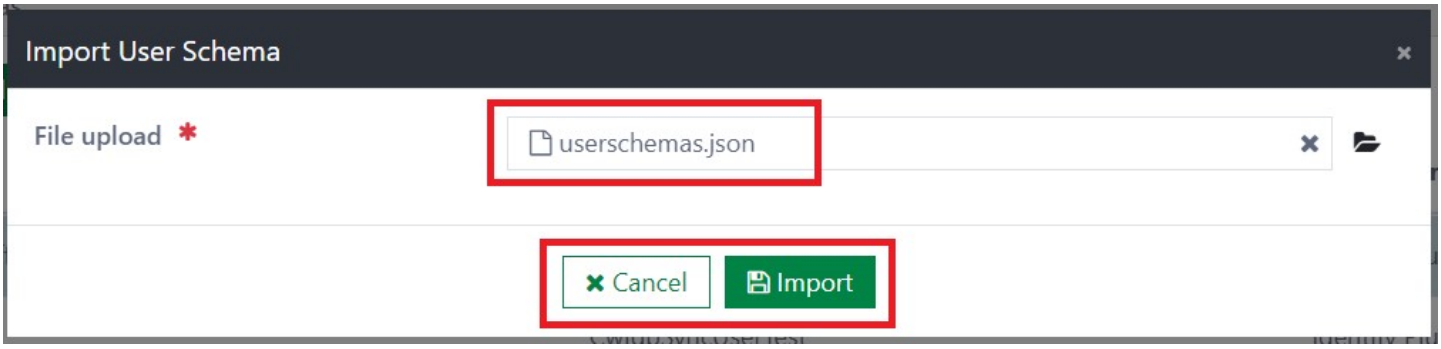


Then, click on the placeholder of the **File upload** field to open file selection dialog



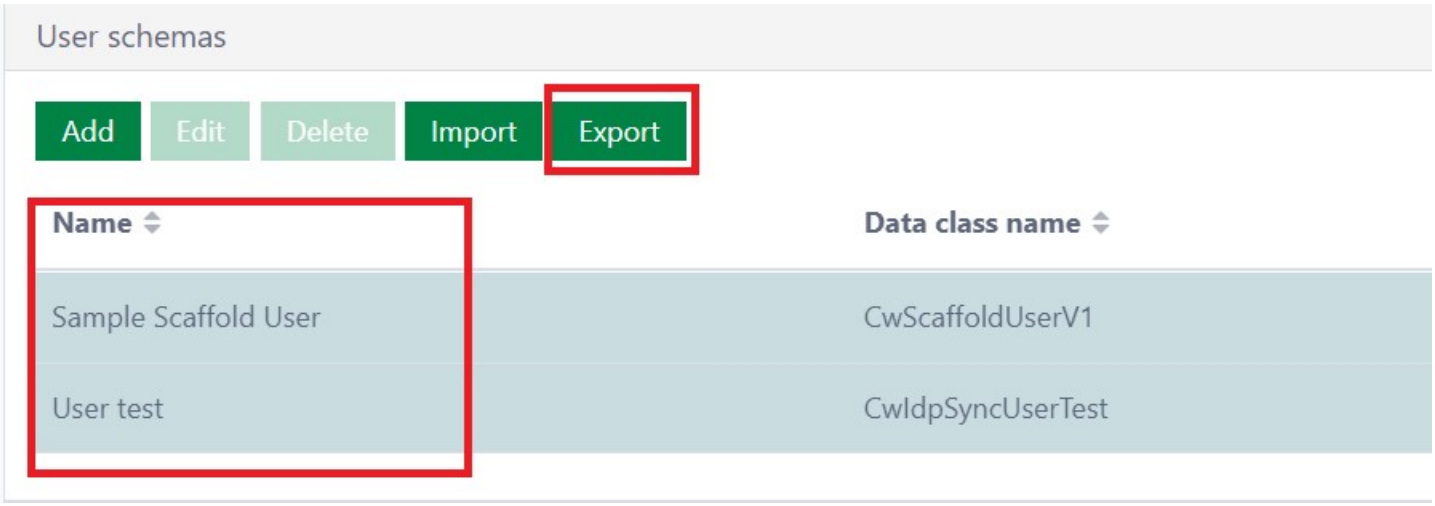
Next, select a JSON file to import (multiple selection is not supported). After selecting the file, click on **Import** button to import

schemas from the JSON file. If you do not want to execute the import process, click on **Cancel** button.

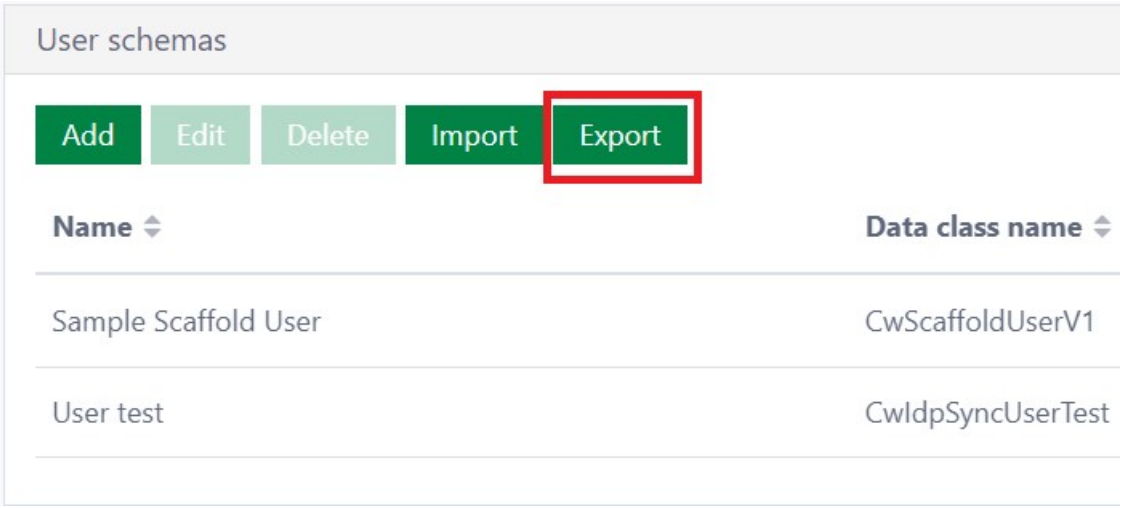


Export user schemas

To export one or several schemas in the list, select one or multiple schemas. Next, click on the **Export** button. Then, the JSON file of the selected schemas will be downloaded immediately.



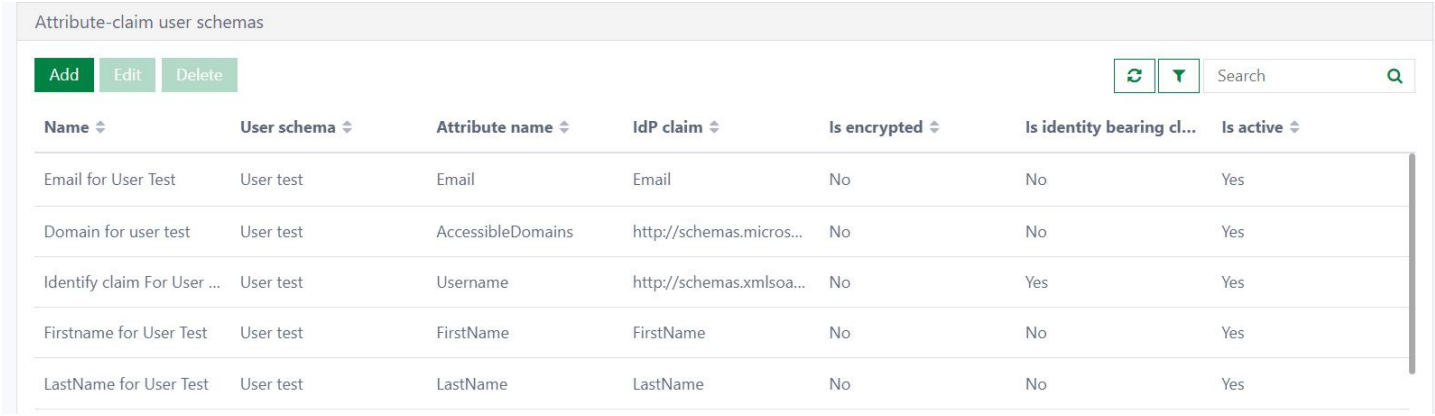
To export all schemas in the list, click on **Export** button. Then, the JSON file of all schemas will be downloaded immediately.



Manage attribute claims

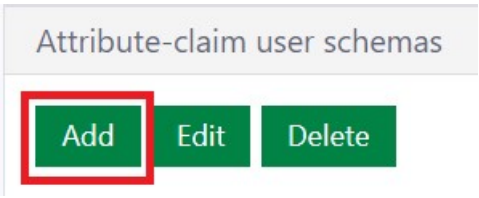
The attribute claim user schema is the way to determine the mapping between a data class attribute in Casewhere and a corresponding claim in IdP.

- **Note:** The claim definition must exist in the IdP before we set up claim mapping.



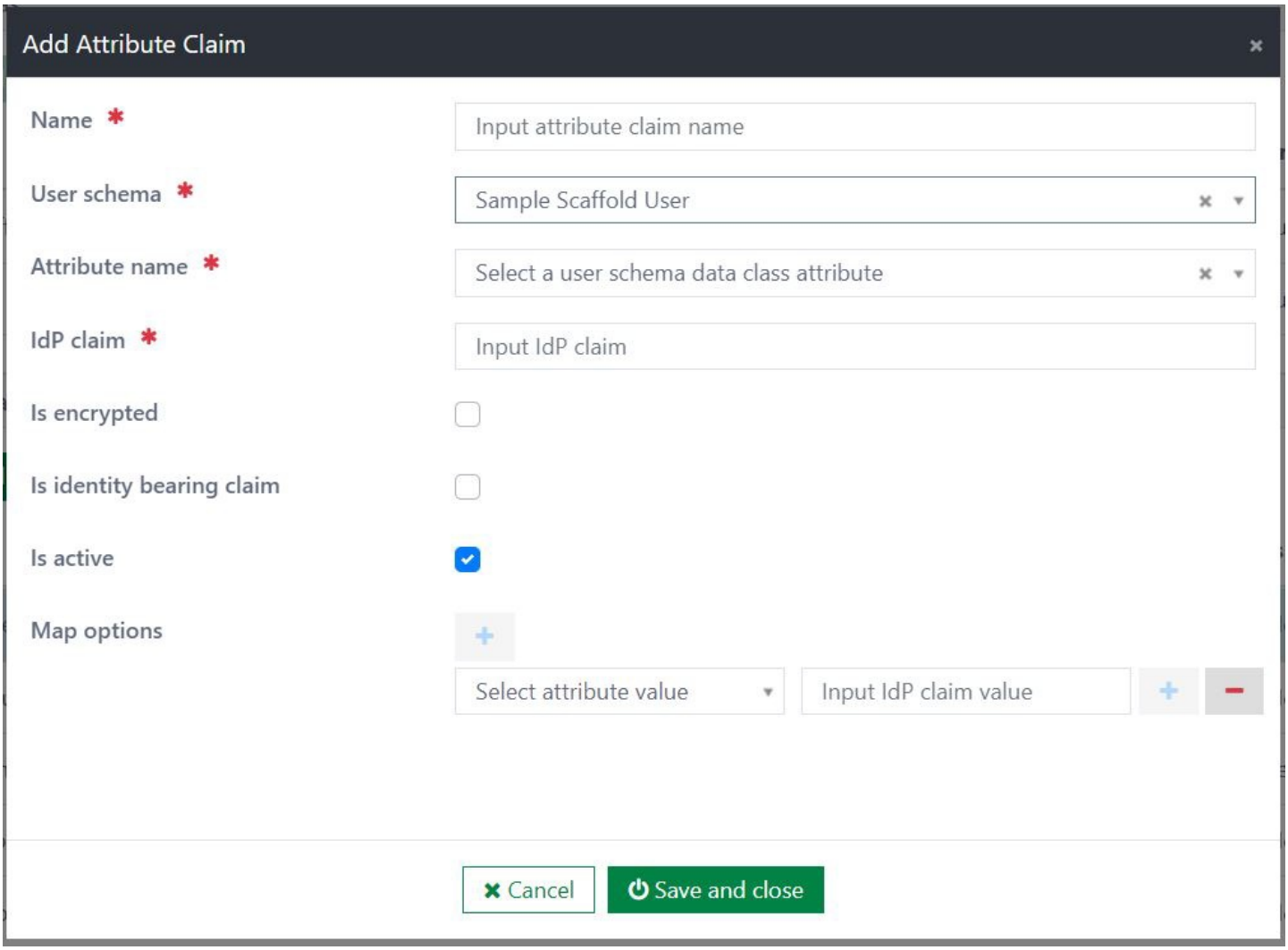
Add new attribute claim

To add new attribute claim, click on the **Add** button



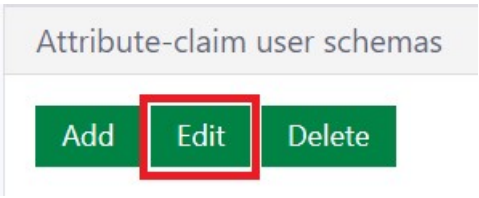
Then, in **Add Attribute Claim** dialog, fill out the fields:

- **Name:** You must input name of the attribute claim. This field is required and unique in a user schema.
- **User schema:** It is a dropdown list with all user schemas. This field is required.
- **Attribute name:** The list of **Attribute name** is generated according to the selected user schema's data class. This field is required.
- **IdP claim:** The claim name in external IdP system. This field is required and unique in a user schema.
- **Is encrypted:** If this checkbox is ticked, the value will be encrypted before sending it to the Identity provider.
- **Is identity bearing claim:** There is only one **Is identity bearing claim** in a user schema. The bearing claim is used to identify the user. Normally, we use email as a bearing claim.
    - The first attribute claim of a user schema must be ticked **Is identity bearing claim**.
    - If the user add or edit a claim which is ticked **Is identity bearing claim**, the system will auto override the last bearing claim to a normal claim and the new claim will become identity bearing claim of the selected user schema.
- **Is active**: This field is the status of attribute claim. The default value is **Active**.
- **Map options:** This field only visible when the selected **Attribute name** in data class is array of enum type. By clicking on **+** button, the additional fields to fill the information of **Map options** will be displayed. You can add multiple fields:
    - **Attribute value**: This field is required. It is a list of the attribute enum options.
    - **IdP claim value**: This field is required. This field is used to input the IdP claim option in IdP system.



Edit attribute claim

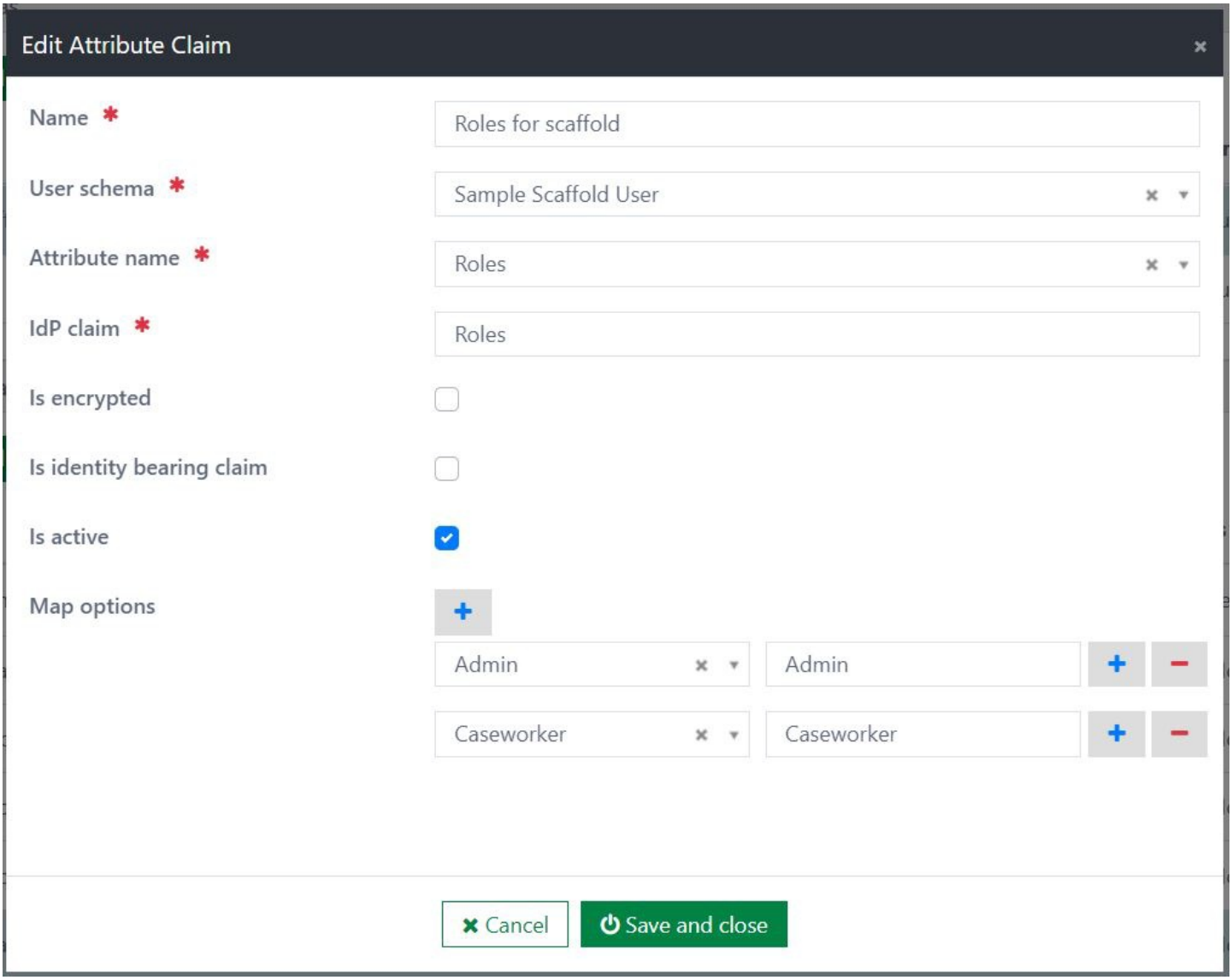
To edit attribute claim, first, click on **Edit** button to open **Edit Attribute Claim** dialog



In the **Edit Attribute Claim** dialog, you can edit:

- **Name**
- **User schema**
- **Attribute name**
- **IdP claim**
- **Is encrypted**
- **Is identity bearing claim**
- **Is active**
- **Map options**:
    - **Attribute value**

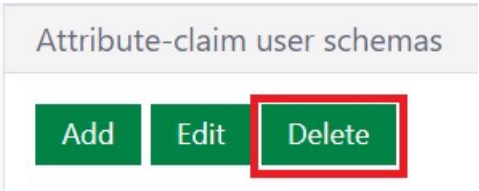
- **IdP claim value**



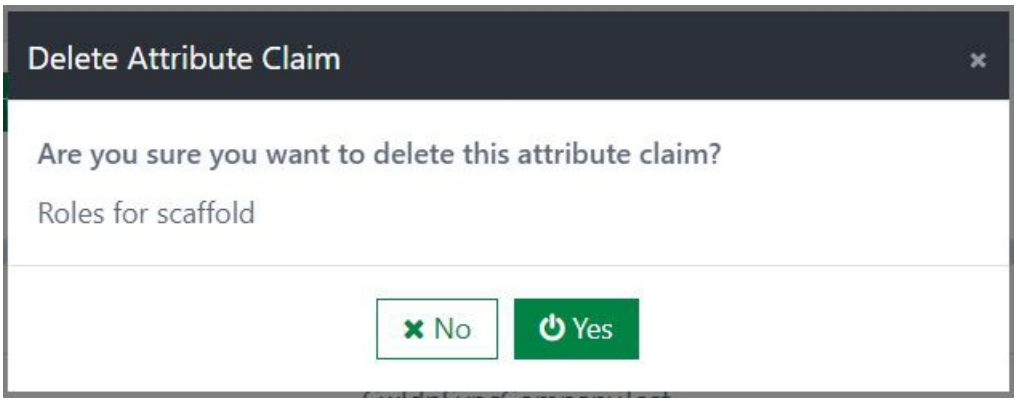
After updating the information, click on **Save and close** button to save the changes. Or you can click on **Cancel** button to discard the changes.

Delete attribute claim

To delete attribute claim, first, click on the **Delete** button



Then, a confirmation dialog will be displayed. If you select the **Yes** button, the config will be deleted immediately. If you select the **No** button, the deletion will be cancelled.



Synchronize users to IdP

It is the core in the component to synchronize the users in Casewhere to IdP automatically. It will be reused a lot in many applications and avoid duplicating the code to create the users in many applications. It operates based on broadcasting some custom events that are defined in the component in advance. Every time we collect the user information to create or update the user in Casewhere, then you need to raise a corresponding custom event with its data.

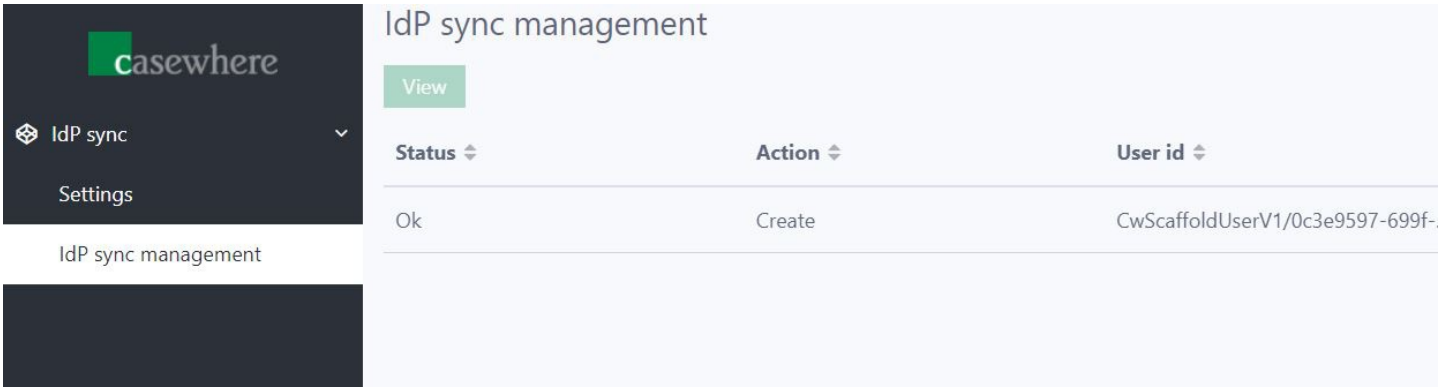
Custom events

This component includes 3 custom events:

- CW_CreateUser: This event is raised when the user is created. Its data include:
    - UserId: Be a data object containing the user information in the application.
    - Username: Be a value to identify that user. It's used to support troubleshooting.
- CW_EditUser: This event is raised when the user is updated. Its data include:
    - UserId: Be a data object that contains the user information in the application.
    - Username: Be a value to identify that user. It's used to support troubleshooting.
    - IdentifyUserId: Userid which is returned from Identity Provider.
- CW_DeleteUser: This event is raised when the user is deleted. Its data include:
    - UserId: Be a data object that contains the user information in the application.
    - Username: Be a value to identify that user. It's used to support troubleshooting.
    - IdentifyUserId: Userid which is returned from Identity Provider.

The workflow that handles this event will find the user schema that corresponds to the data class of that object. From that, we know where to create/update/delete the users in IdP.

Monitor IdP sync status



In IdP sync management page, you can view the information of the IdP sync data, they are:

- **Status:** is the status of event data. There are 3 statuses:

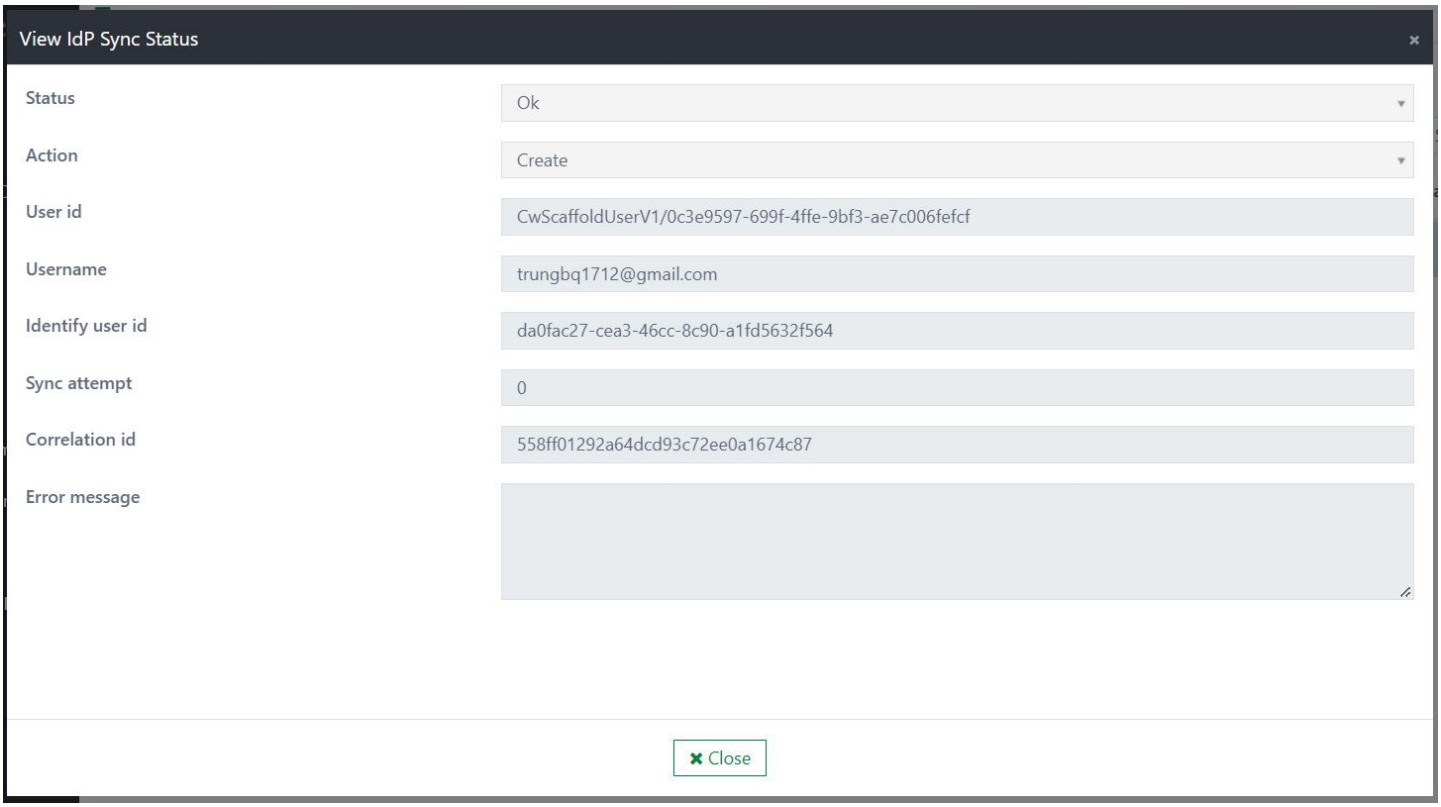
    - **Pending:** the user data has just been created.
    - **OK:** the user data has been synced to IdP successfully.
    - **Failed:** the user data has been synced to IdP failed or the data has errors in the preparation.

- **Action:** is the type of action events. There are 3 actions:

    - **Create:** the action raised when the user is created.
    - **Edit:** the action raised when the user is updated.
    - **Delete :** the action raised when the user is deleted.

- **User id:** is the user id of event data.
- **Username:** is the user name of event data.
- **Identify user id:** is the user id of IdP system.
- **Sync attempt:** is the number of resend attempts has been triggered.
- **Correlation id:** is the id used to identify related log entries.
- **Error message**: is the error message when the preparation or syncing of the data has failed.

    - Note: The system will auto retry three times if the sync has failed.



Installation

Requirements

- Casewhere 2.7.1 or later.

Configuration

1. Import the product
2. Link the page to your worker sites and configure access control
3. Configure the IdP sync settings

Dependencies

- Plugin: Identify Plugin

Releases

1.0.0 - 24/04/2022

Changelog

- Mapping the user from Casewhere to IdP
- Synchronize users to IdP

Download (login required): IdP Sync v1.0.0

Roadmap

- Activate/deactivate the user
- Sync the possible values of the claims to IdP
- Configure the connection string to IdP

Component: Location data

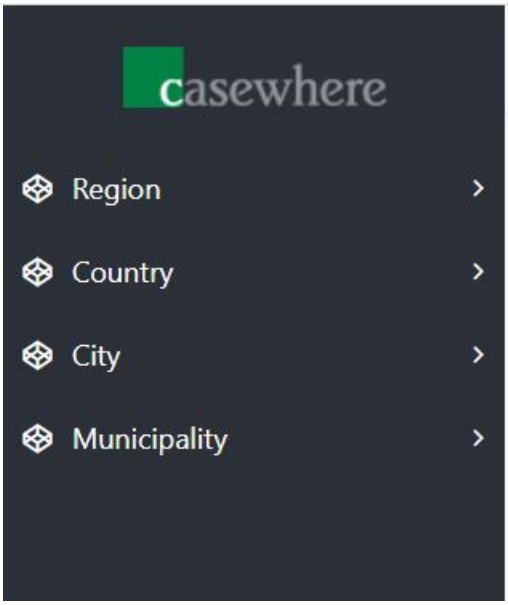
Introduction

The **Location data** is present in all projects from small to large projects. Because it is master data for the project. Obviously, the Casewhere system is no exception. We have also been using **Location data** such as country, city... Developing the **Location data** component for each new project in Casewhere is very time-consuming. And, it's inefficient to implement the **Location data** component every time. It takes a lot of project resources. Therefore, we have provided the **Location data** component package which is easy to import to new projects to save time and cost.

Features

Administration interfaces for managing the following:

- Country
- Region
- Municipality
- City



Import/export: To export the data from an environment, e.g., DEV, and import it to another, e.g., PROD.

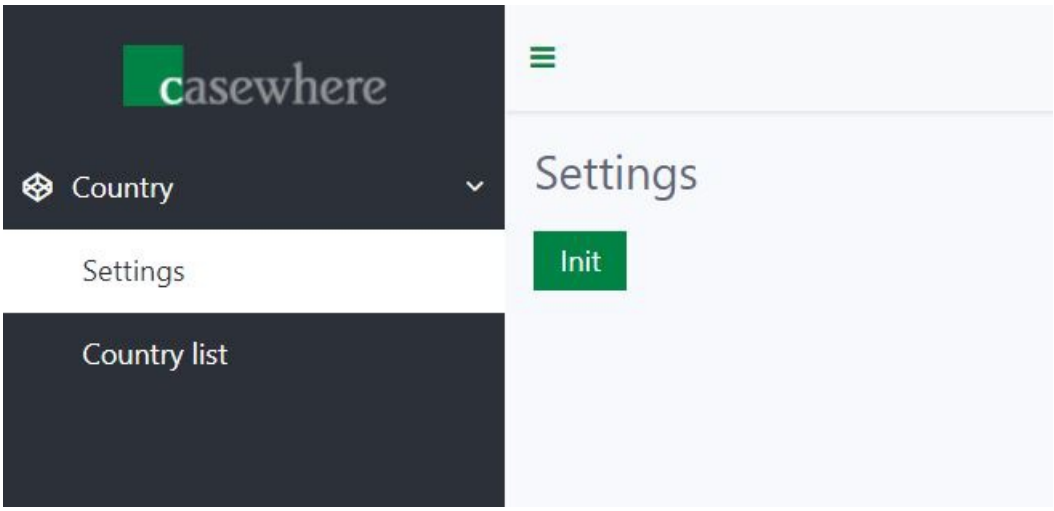
Friendly configuration page with many applicable settings. There you can also generate default data for your application.

Manage country

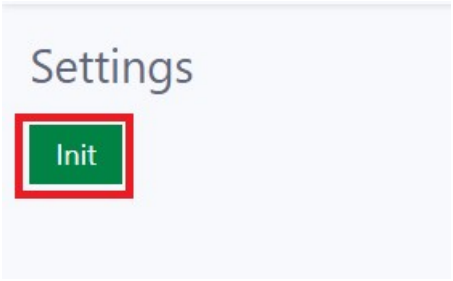
The component provides two pages to manage and set up the country.

Settings

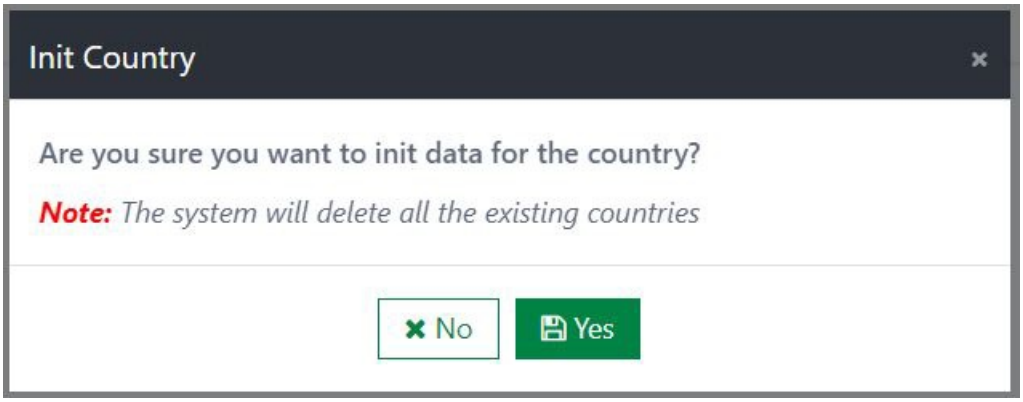
The settings page is used to settings for the country where is the place the user can generate the default country data by clicking the **Init** button. All the default countries are in a JSON file which is stored on the back end site. If you click the **Init** button twice the system will clear all the existing countries and re-generate the new data. Hence, you must consider when using the

initialization data feature. 

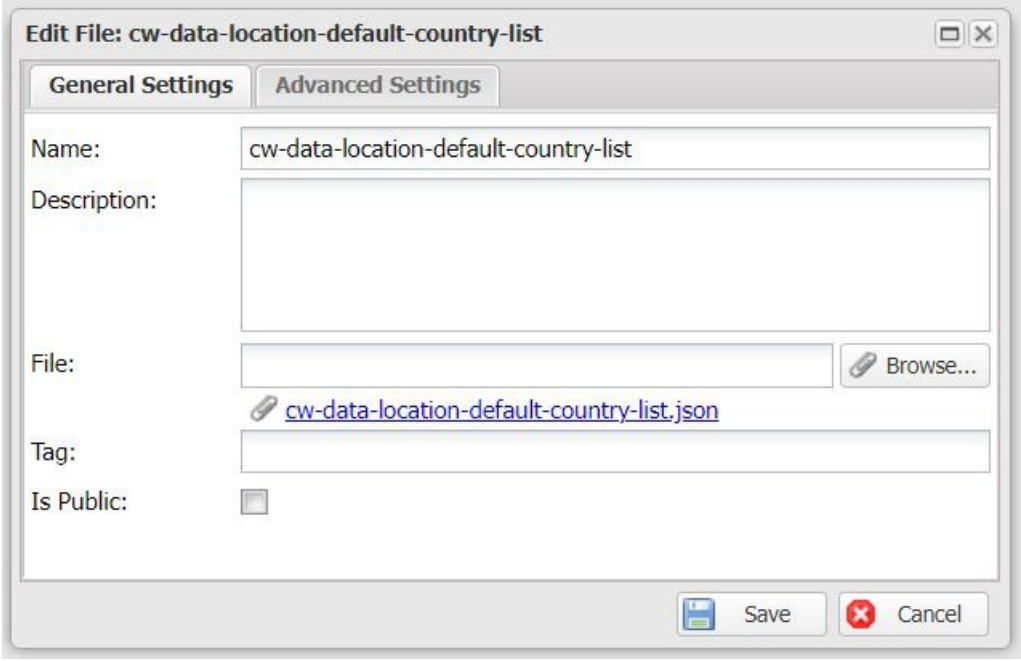
To initialize data in the settings page, click on the **Init** button



After that, the confirm dialog of the Settings will be opened with two options Yes/No.

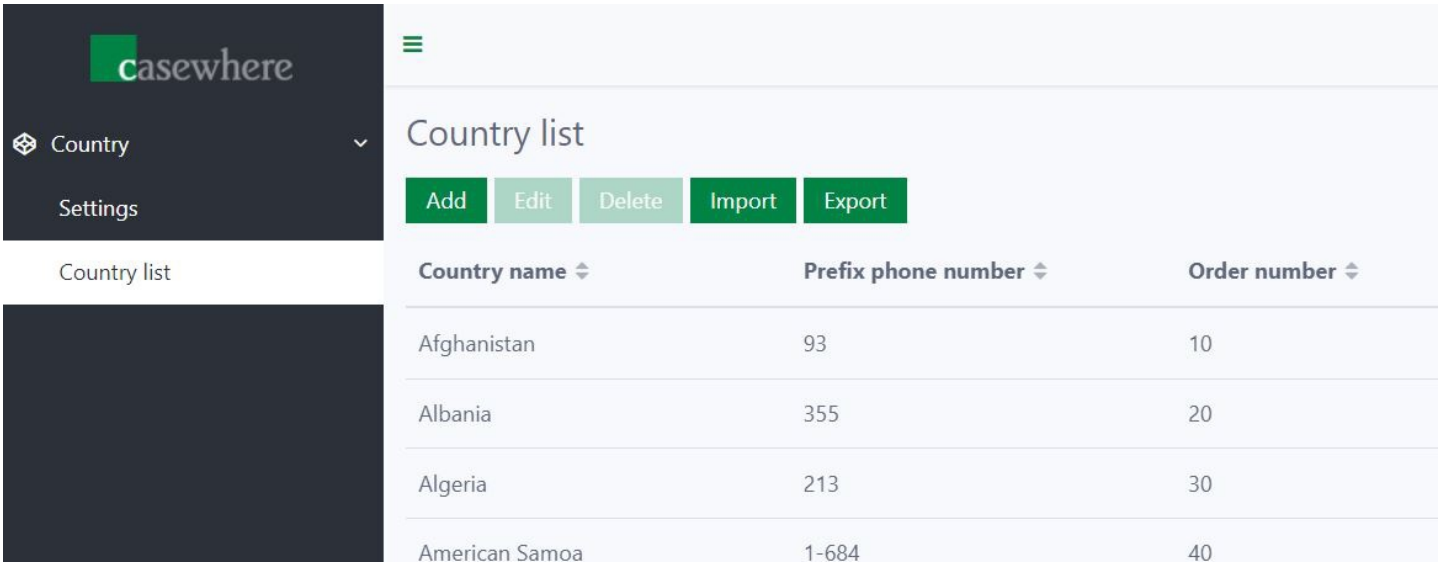


All default countries will be listed in the JSON file. You can modify their values based on your needs.



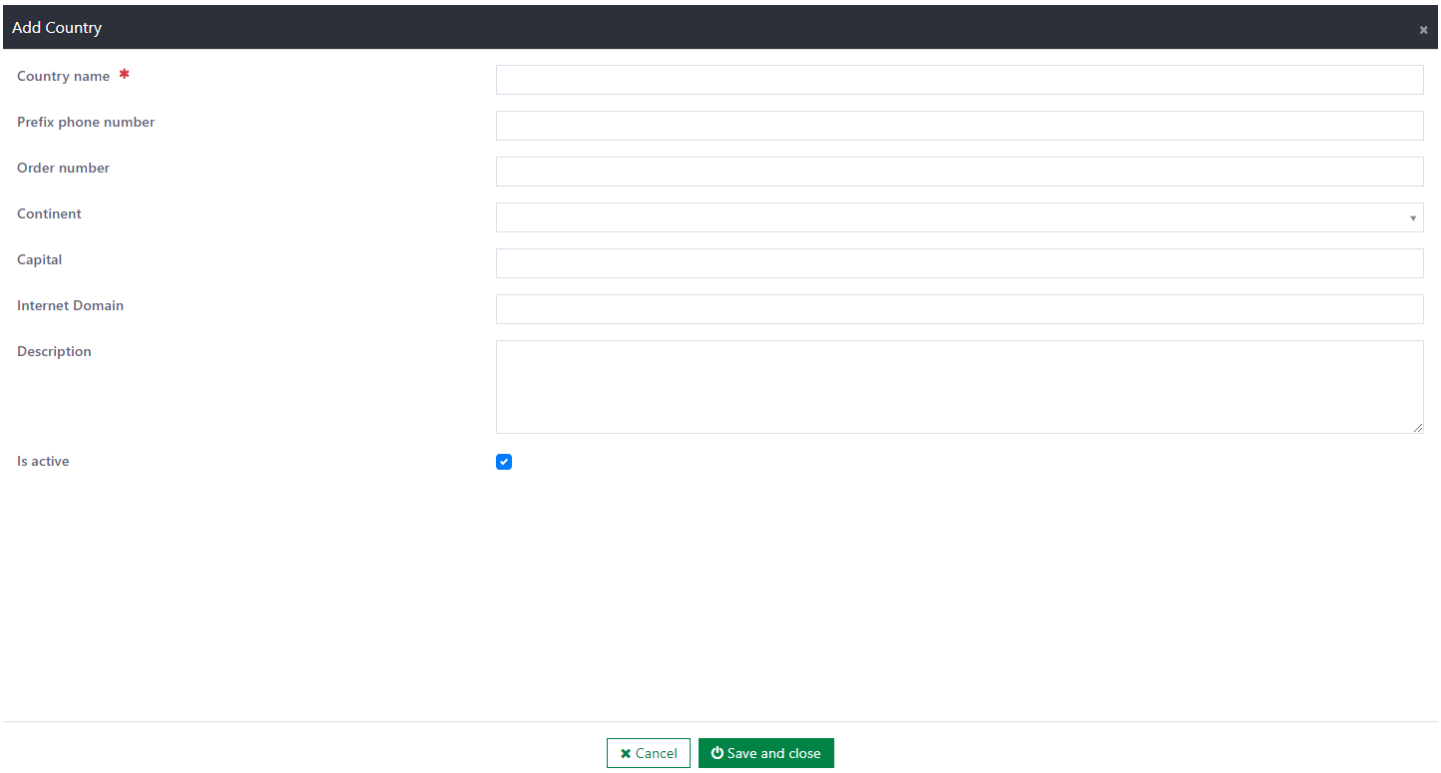
Country list

Country list page is used to manage the country data. On the **Country list** page, you can view all the created countries, also add, edit, delete, import, export the country data.



The following information must be provided when creating/editing a country:

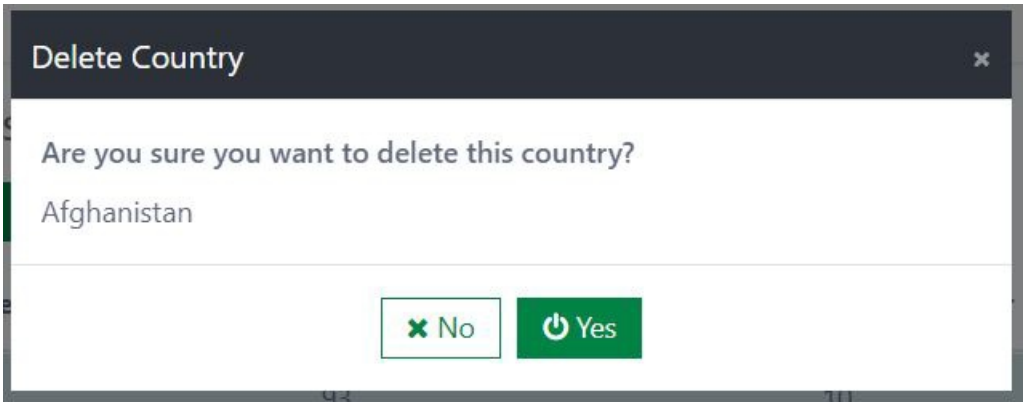
- **Country name**: The name of the country. This field is required and unique.
- **Prefix phone number**: This field is used to input the pre-fix phone number of the country. This field is optional.
- **Order number**: This field defines the priority of the country you want to show. This field is optional.
- **Continent**: This field is used to input the continent of the country. This field is optional.
- **Capital**: This field is used to input the capital of the country. This field is optional.
- **Internet domain**: This field is used to input the internet domain of the country. This field is optional.
- **Description**: This field is used to describe the information of the country. This field is optional.
- **Is active**: This field is the status of the country. The default value is **Active**.



Delete a country:

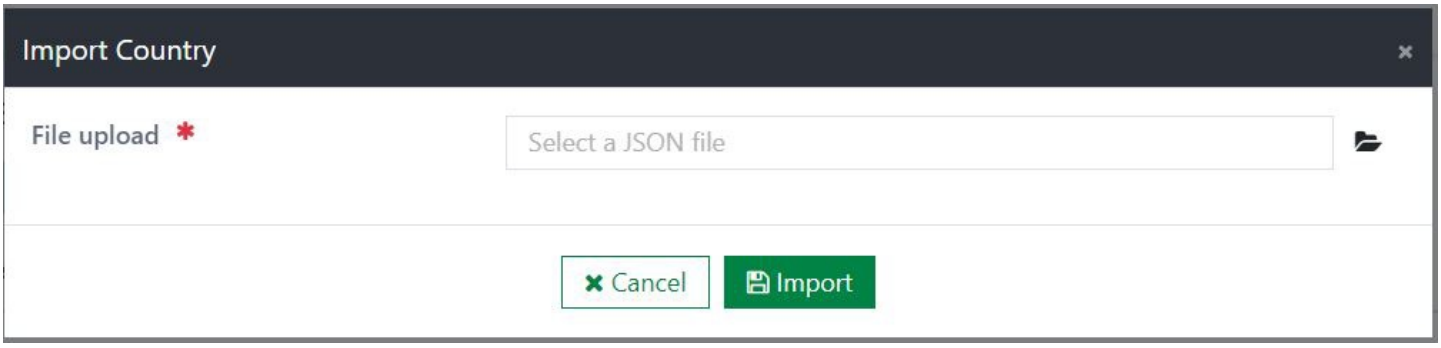
To delete a country, click on the **Delete** button. Then, a confirmation dialog will be displayed. If you select the **Yes** button, the country will be deleted immediately. If you select the **No** button, the deletion will be canceled.

- **Note:** If the country is in use in the system. It may cause an error in the UI.



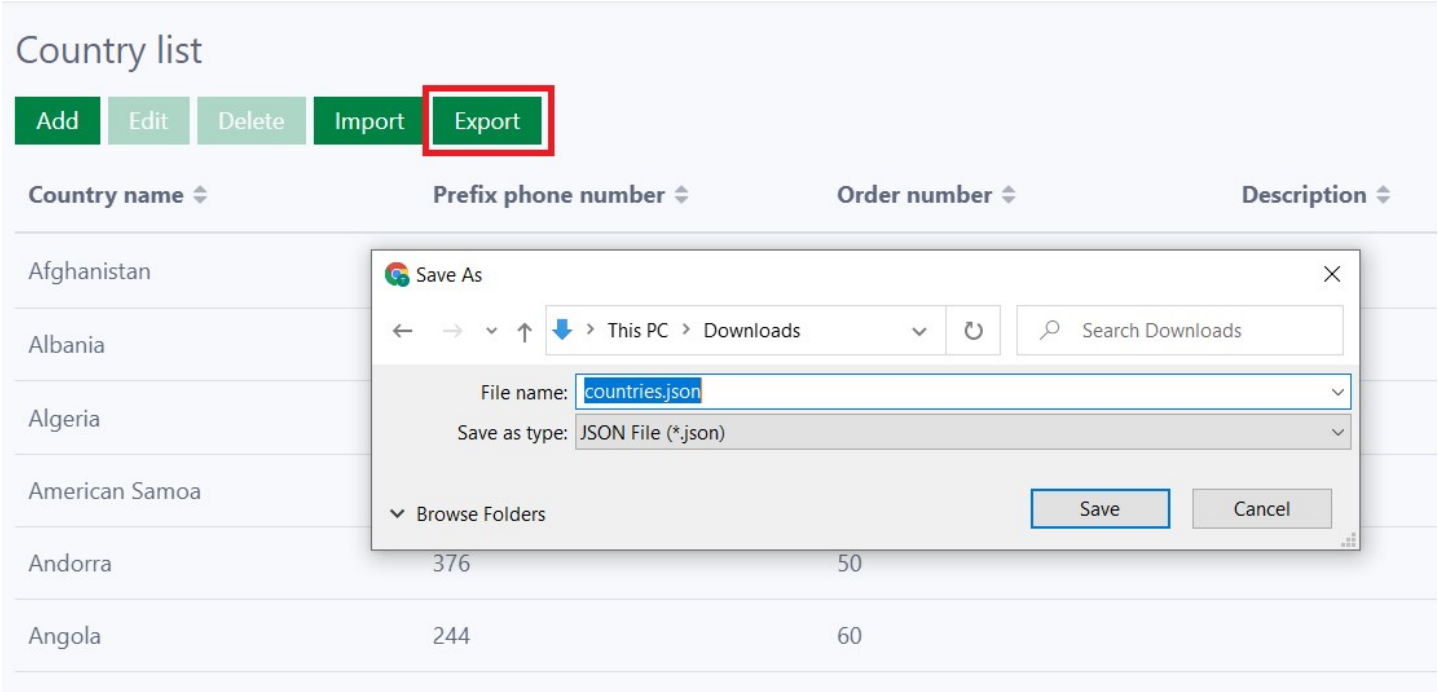
Import the country data:

To import the country data, click on the **Import** button to open the **Import Country** dialog. All the country data that exists in the destination environment will be overridden. (Detected by country name)



Export the country data:

To export 1 or several countries in the list, select the country, then click on the **Export** button. Then, the JSON file of the selected countries will be downloaded immediately.

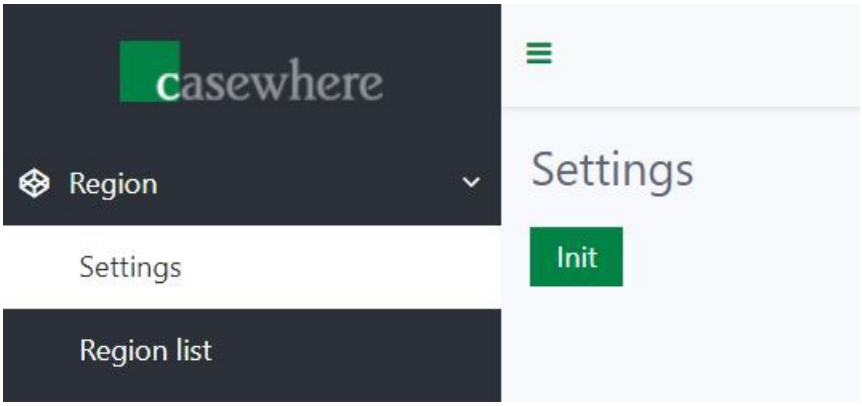


Manage region

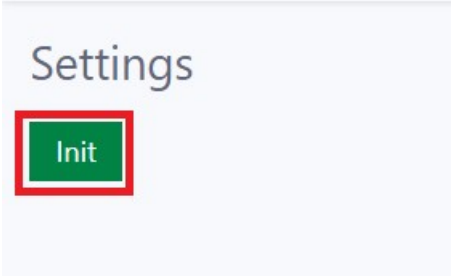
The component provides two pages to manage and set up the region.

Settings

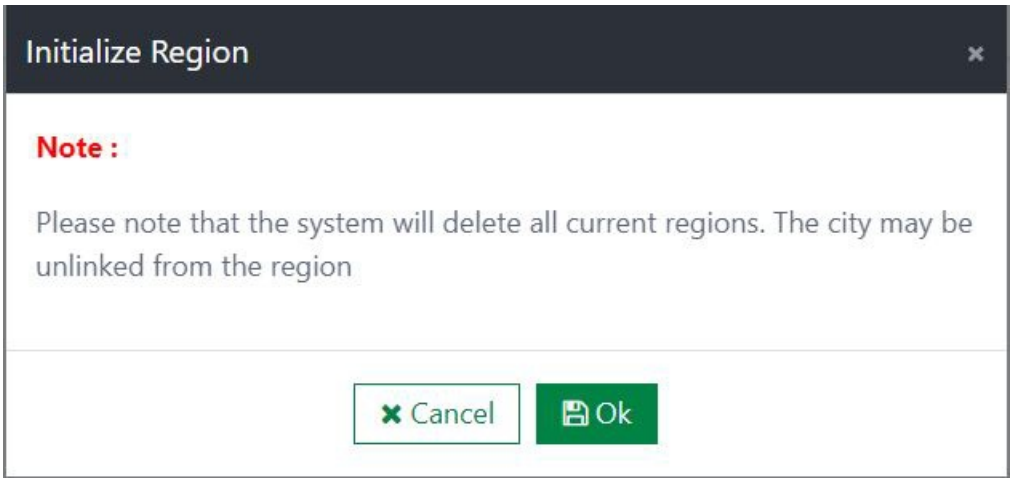
The settings page is used to settings for the region where is the place the user can generate the default region data by clicking the **Init** button. All the default regions are in a JSON file which is stored on the back end site. If you click the **Init** button twice the system will clear all the existing regions and re-generate the new data. Leads to it will break the relationship between region and city, municipality. Therefore, you must consider when using the initialization data feature.



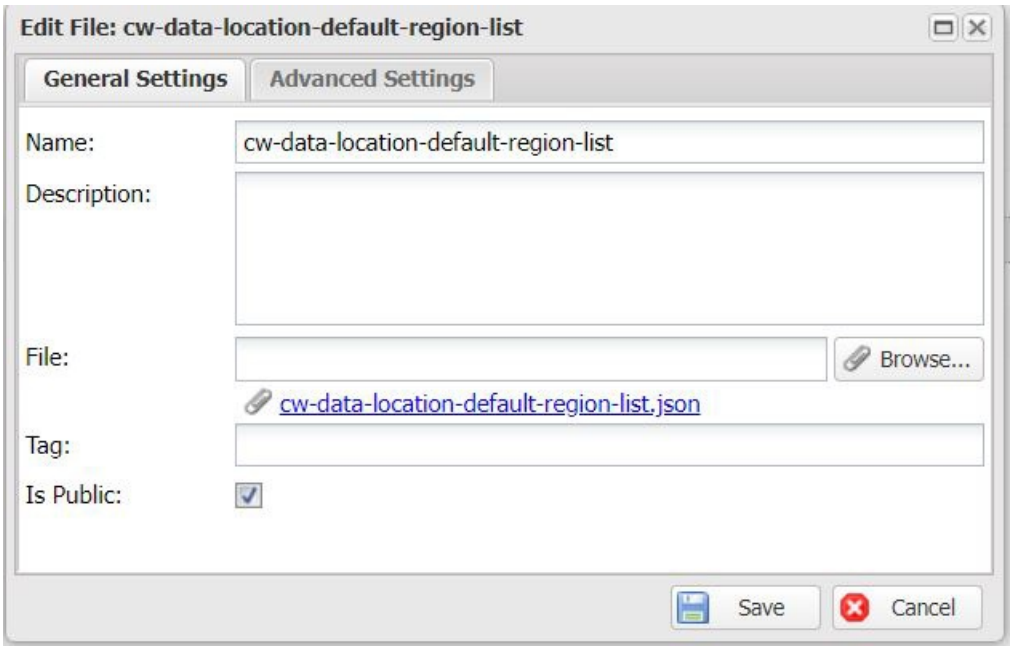
To initialize data in the settings page, click on the **Init** button



After that, the confirm dialog of the Settings will be opened with two options Cancel/Ok.

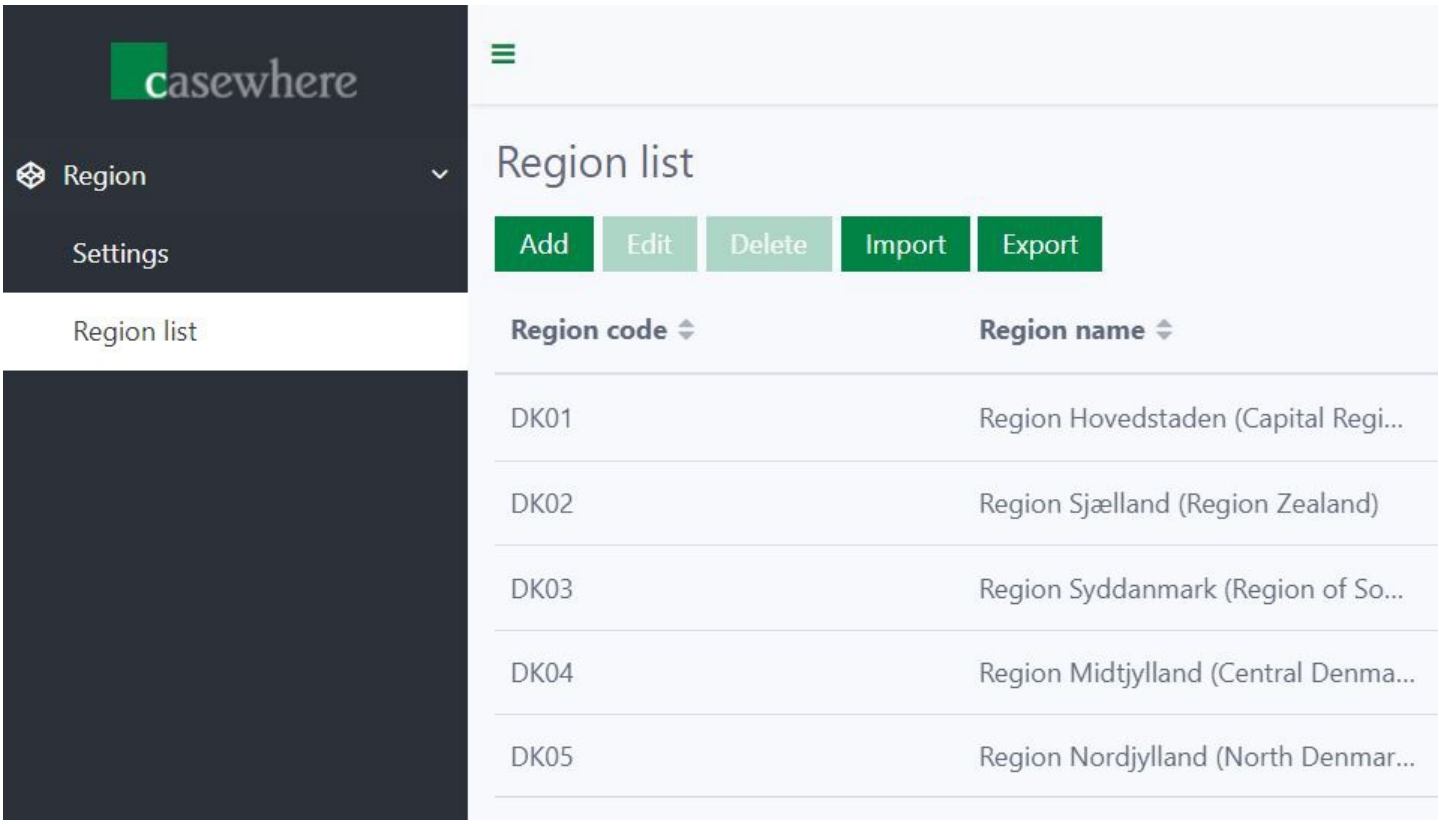


All default regions will be listed in the JSON file. You can modify their values based on your needs.



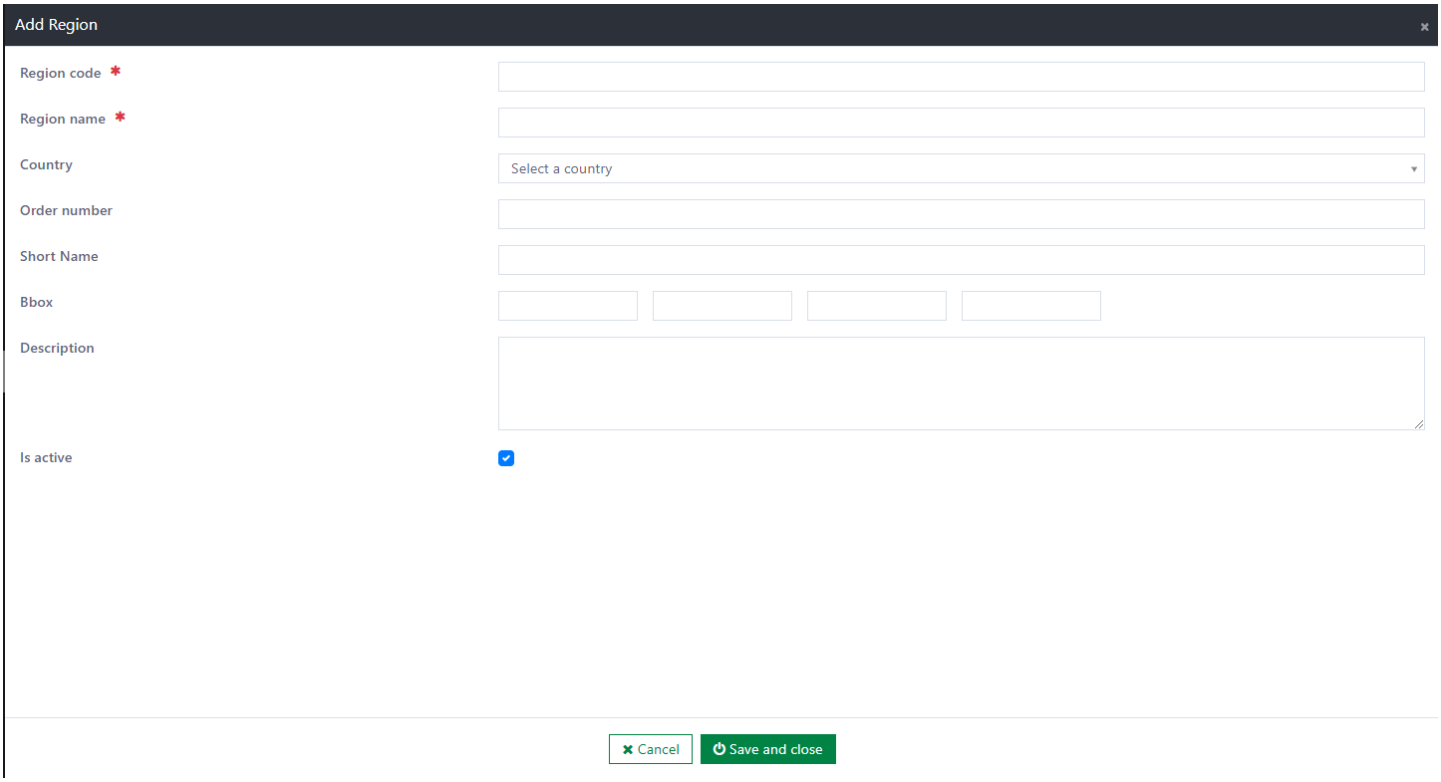
Region list

Region list page is used to manage the region data. On the **Region list** page, you can view all the created regions, also add, edit, delete, import, export the region data.



The following information must be provided when creating/editing a region:

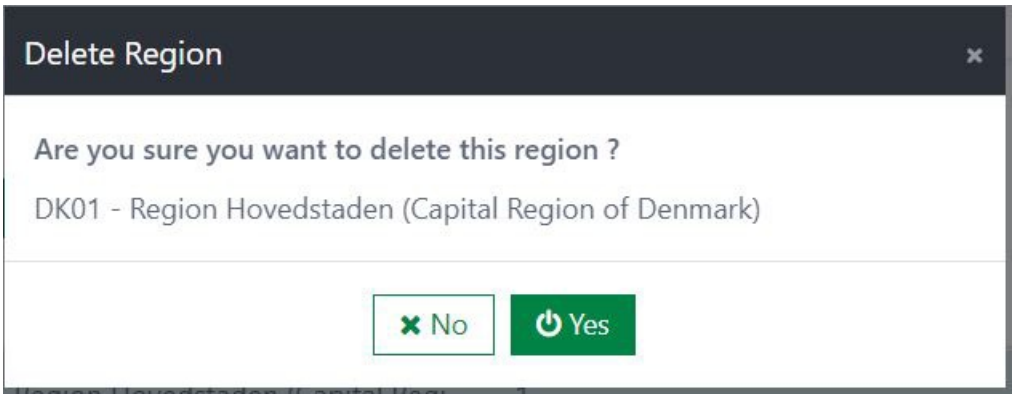
- **Region code**: The code of the region. This field is required and unique.
- **Region name**: The name of the region. This field is required and unique.
- **Country**: The dropdown of the country. It loads data from the country list in the system. This field is optional.
- **Order number**: This field defines the priority of the region you want to show. This field is optional.
- **Short name**: The short name of the region. This field is optional.
- **Bbox**: This field returns the bounding rectangle of the geometry of the region. This field is optional.
- **Description**: This field is used to describe the information of the region. This field is optional.
- **Is active**: This field is the status of the region. The default value is **Active**.



Delete a region:

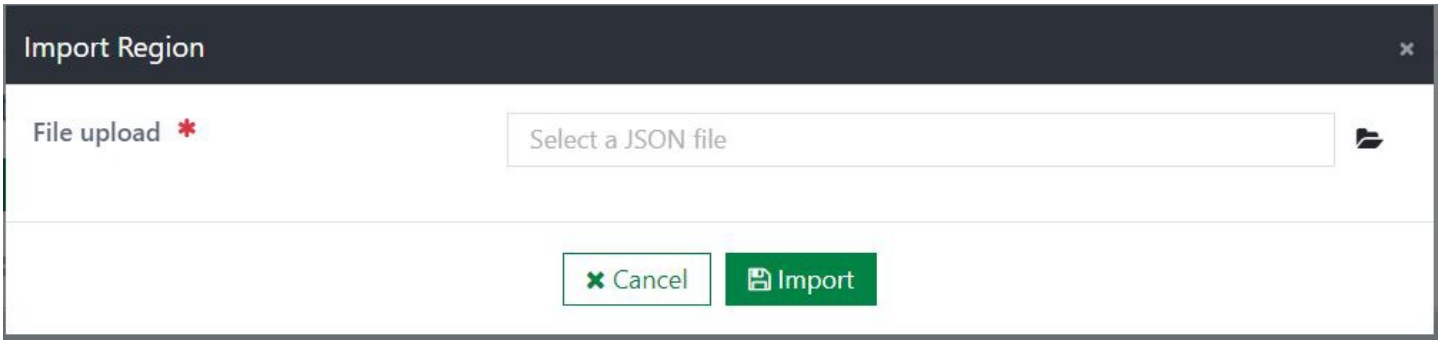
To delete a region, click on the **Delete** button. Then, a confirmation dialog will be displayed. If you select the **Yes** button, the region will be deleted immediately. If you select the **No** button, the deletion will be canceled.

- **Note:** If the region is in use in the system. It may cause an error in the UI.



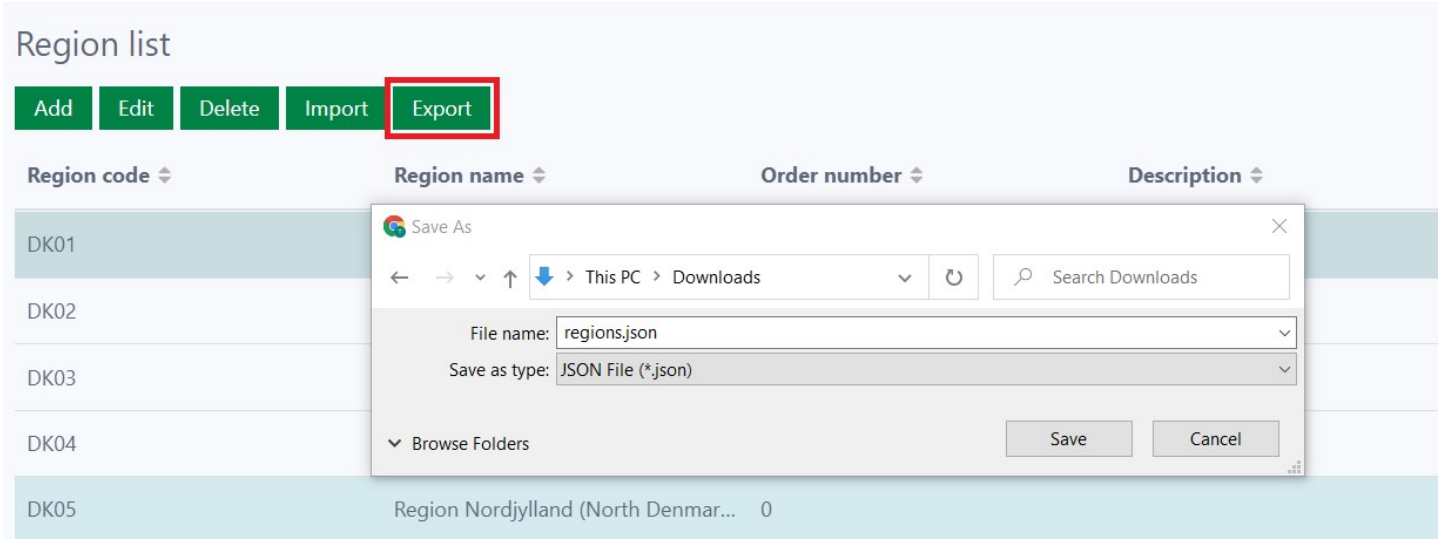
Import the region data:

To import the region data, click on the **Import** button to open the **Import Region** dialog. All the region data that exists in the destination environment will be overridden. (Detected by region code)



Export the region data:

To export 1 or several regions in the list, select the region, then click on the **Export** button. Then, the JSON file of the selected regions will be downloaded immediately.

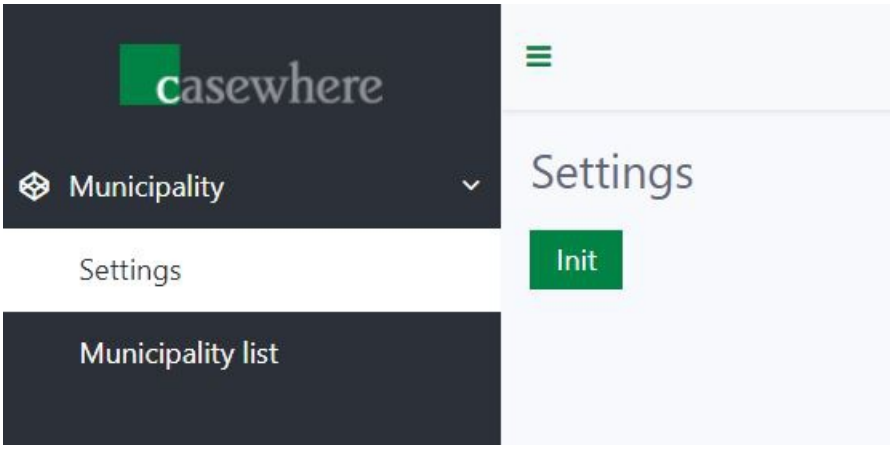


Manage municipality

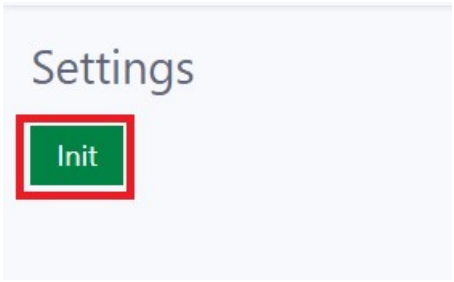
The component provides two pages to manage and set up the municipality.

Settings

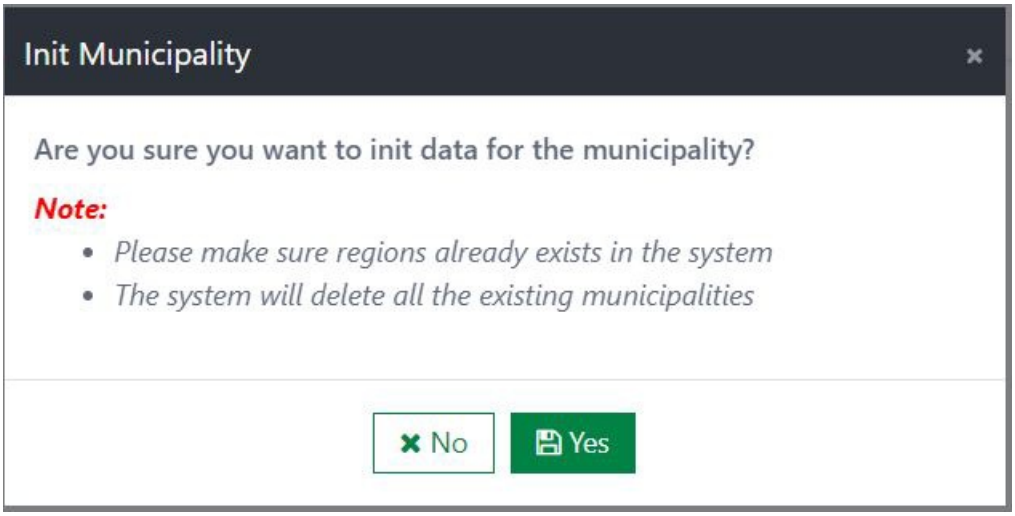
The settings page is used to settings for the municipality where is the place the user can generate the default municipality data by clicking the **Init** button. All the default municipalities are in a JSON file which is stored on the back end site. If you click the **Init** button twice the system will clear all the existing municipalities and re-generate the new data. Therefore, you must consider when using the initialization data feature.



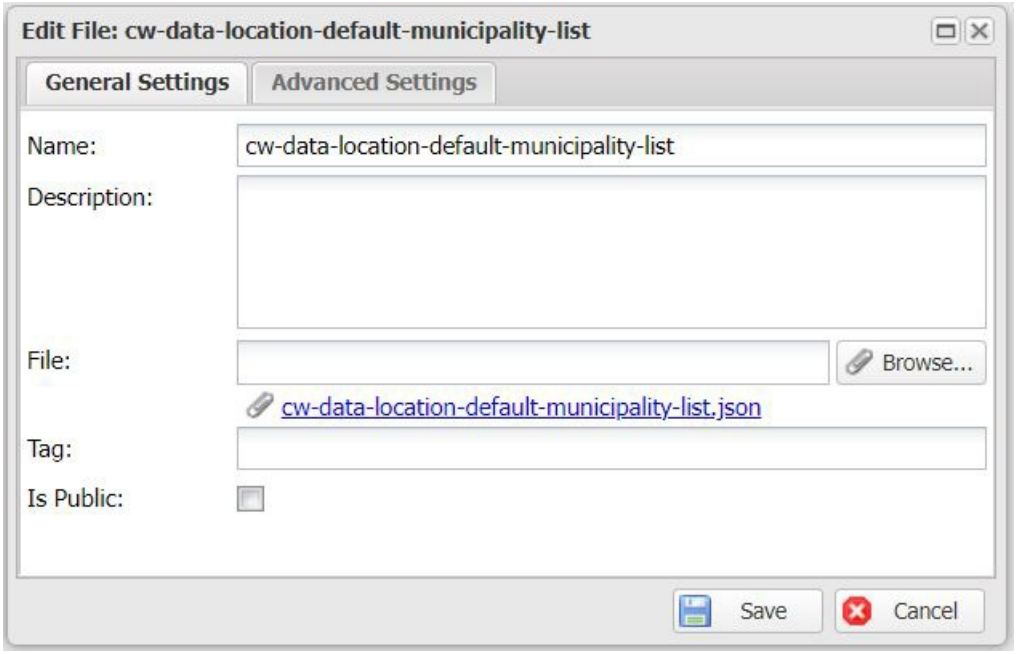
To initialize data in the settings page, click on **Init** button



After that, the confirm dialog of the Settings will be opened with two options Yes/No.

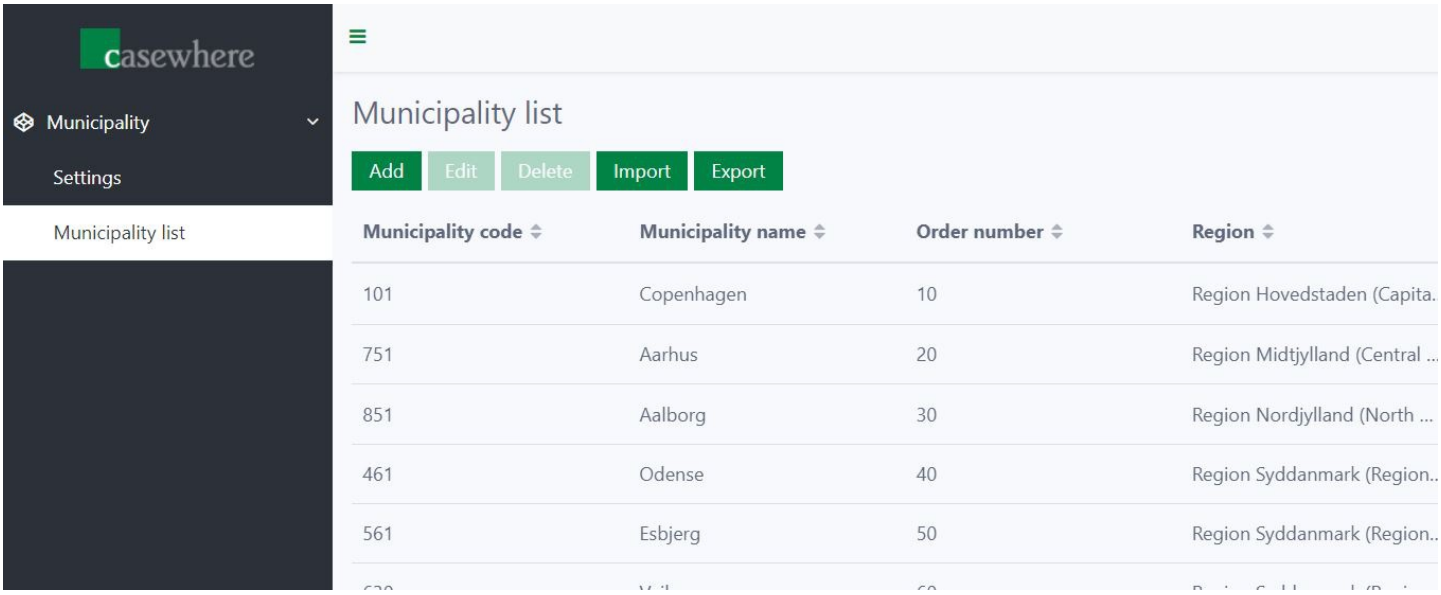


All default municipalities will be listed in the JSON file. You can modify their values based on your needs.



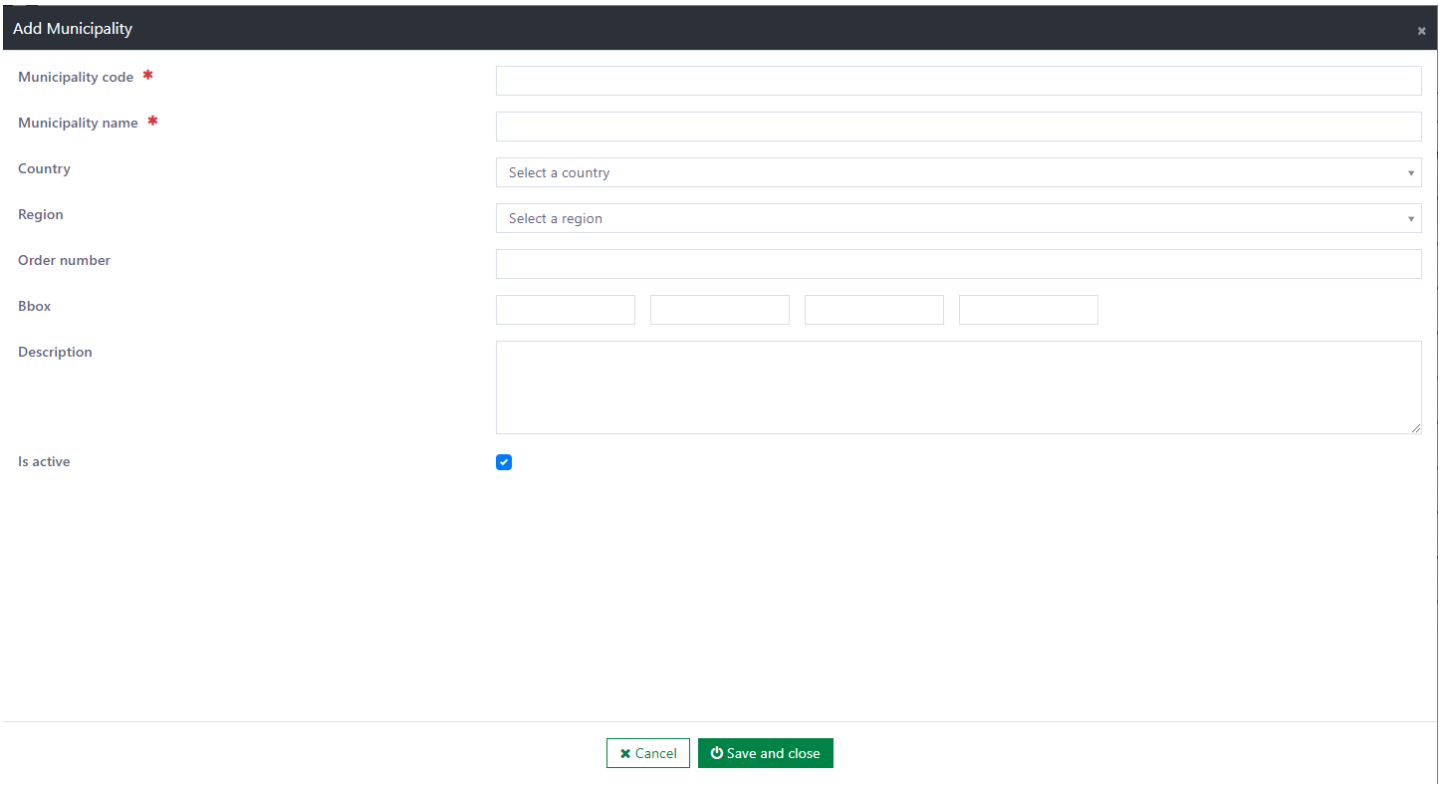
Municipality list

Municipality list page is used to manage the municipality data. On the **Municipality list** page, you can view all the created municipalities, also add, edit, delete, import, export the municipality data.



The following information must be provided when creating/editing a municipality:

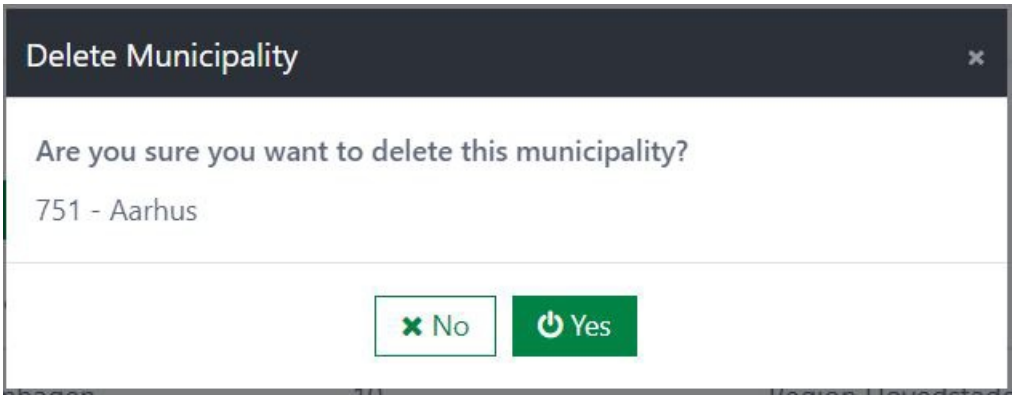
- **Municipality code**: The code of the municipality. This field is required and unique.
- **Municipality name**: The name of the municipality. This field is required and unique.
- **Country**: The dropdown of the country. It loads data from the country list in the system. This field is optional.
- **Region**: The dropdown of the region. It loads data from the region list in the system. It is an area to which the municipality belongs. This field is optional.
- **Order number**: This field defines the priority of the municipality you want to show. This field is optional.
- **Bbox**: This field returns the bounding rectangle of the geometry of the region. This field is optional.
- **Description**: This field is used to describe the information of the municipality. This field is optional.
- **Is active**: This field is the status of the municipality. The default value is **Active**.



Delete a municipality:

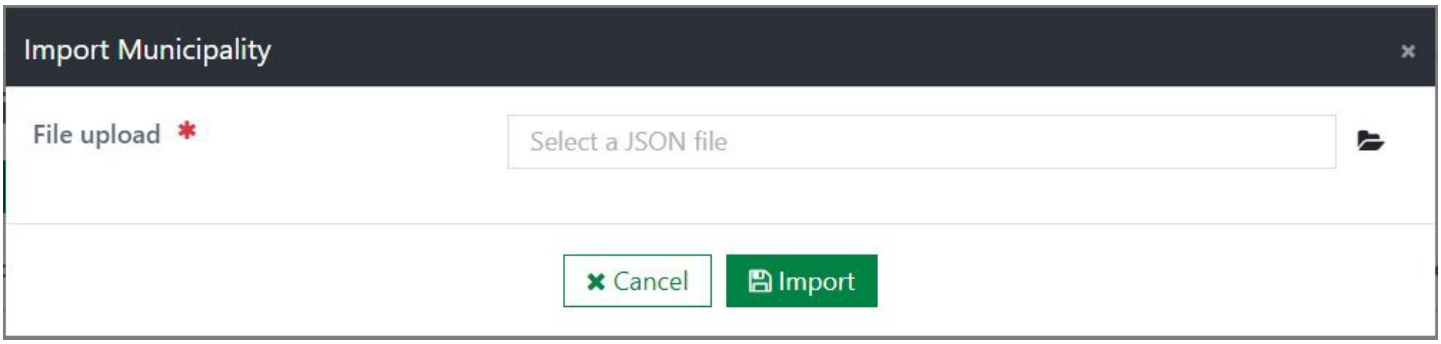
To delete a municipality, click on the **Delete** button. Then, a confirmation dialog will be displayed. If you select the **Yes** button, the municipality will be deleted immediately. If you select the **No** button, the deletion will be canceled.

- **Note:** If the municipality is in use in the system. It may cause an error in the UI.



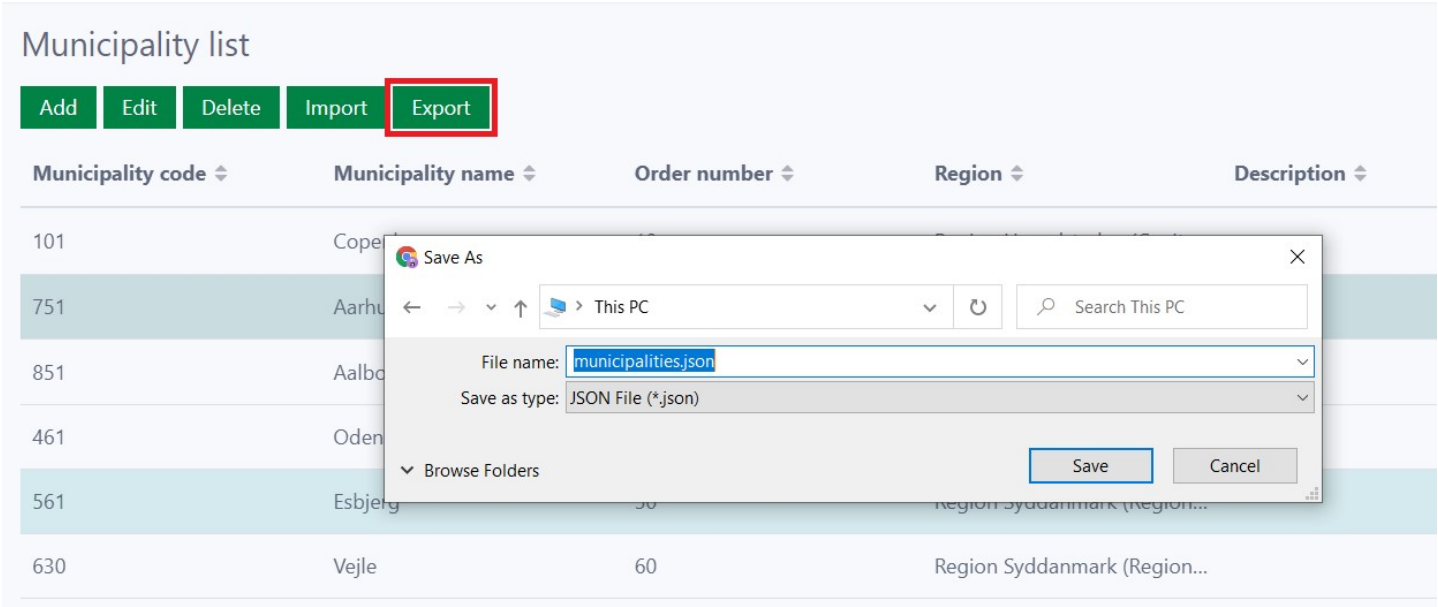
Import the municipality data:

To import the municipality data, click on the **Import** button to open the **Import Municipality** dialog. All the municipality data that exists in the destination environment will be overridden. (Detected by municipality code)



Export the municipality data:

To export 1 or several municipalities in the list, select the municipality, then click on the **Export** button. Then, the JSON file of the selected municipalities will be downloaded immediately.

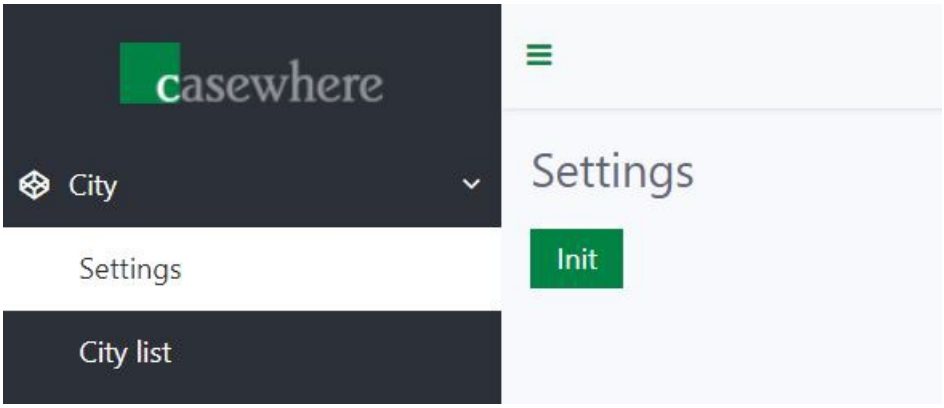


Manage city

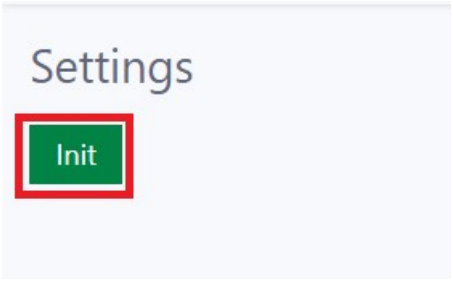
The component provides two pages to manage and set up the city.

Settings

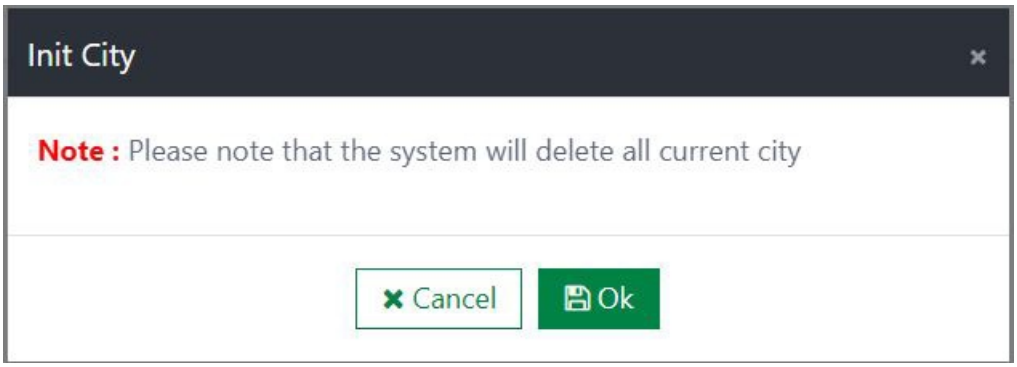
The settings page is used to settings for the city where is the place the user can generate the default city data by clicking the **Init** button. All the default cities are in a JSON file which is stored on the back end site. If you click the **Init** button twice the system will clear all the existing cities and re-generate the new data. Therefore, you must consider when using the initialization data feature.



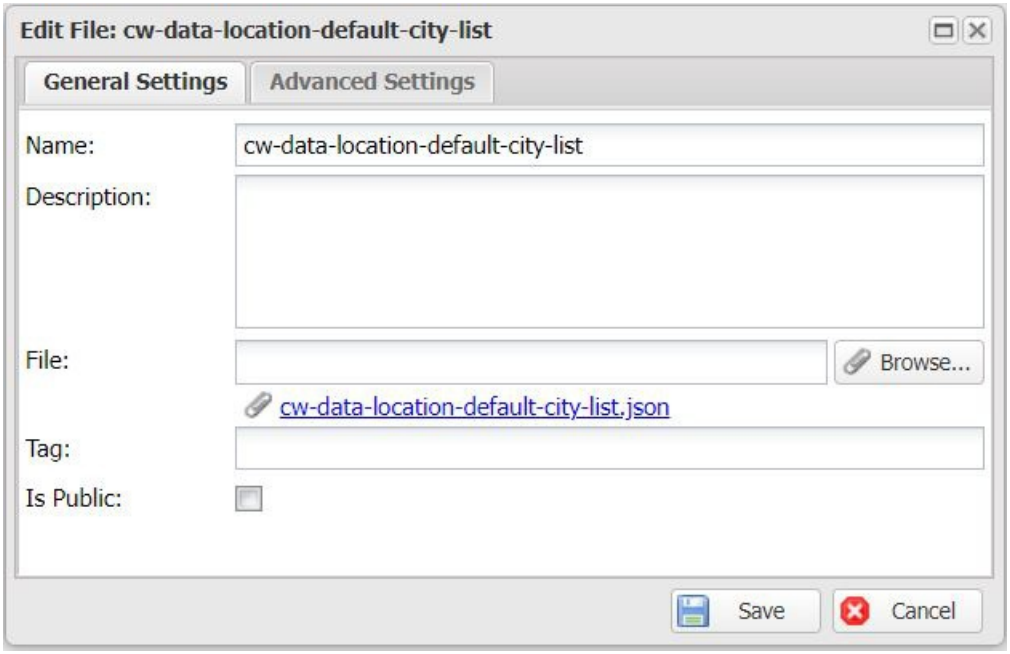
To initialize data in the settings page, click on **Init** button



After that, the confirm dialog of the Settings will be opened with two options Cancel/Ok.

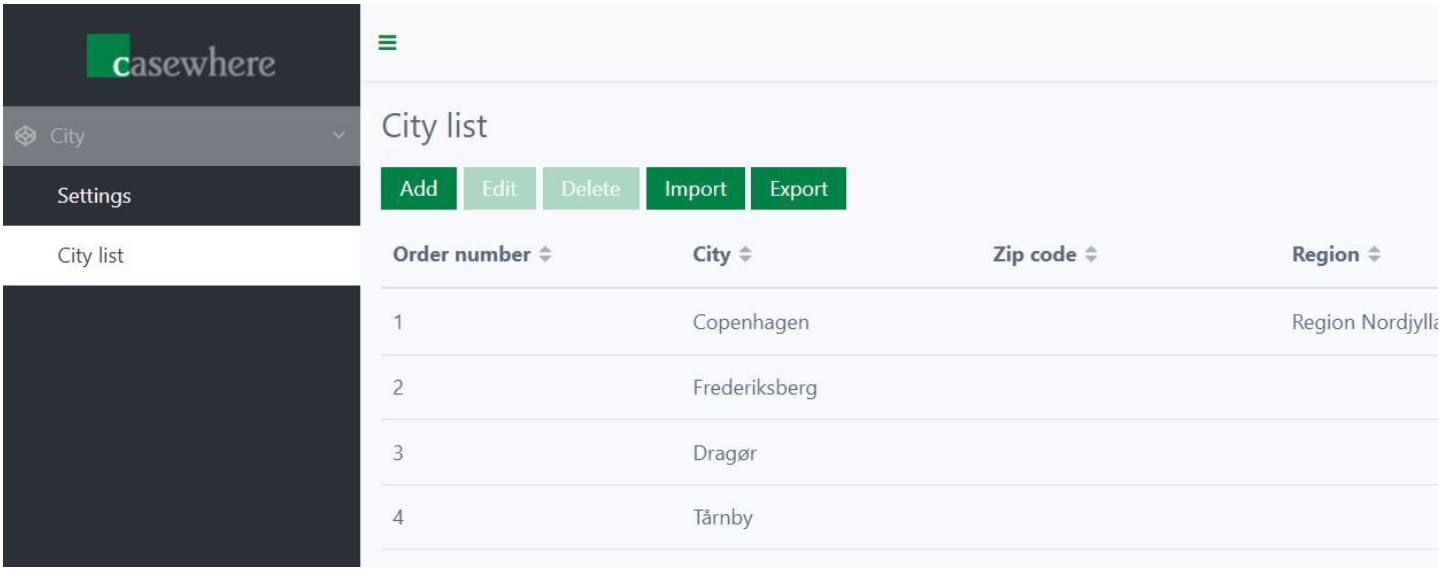


All default cities will be listed in the JSON file. You can modify their values based on your needs.



City list

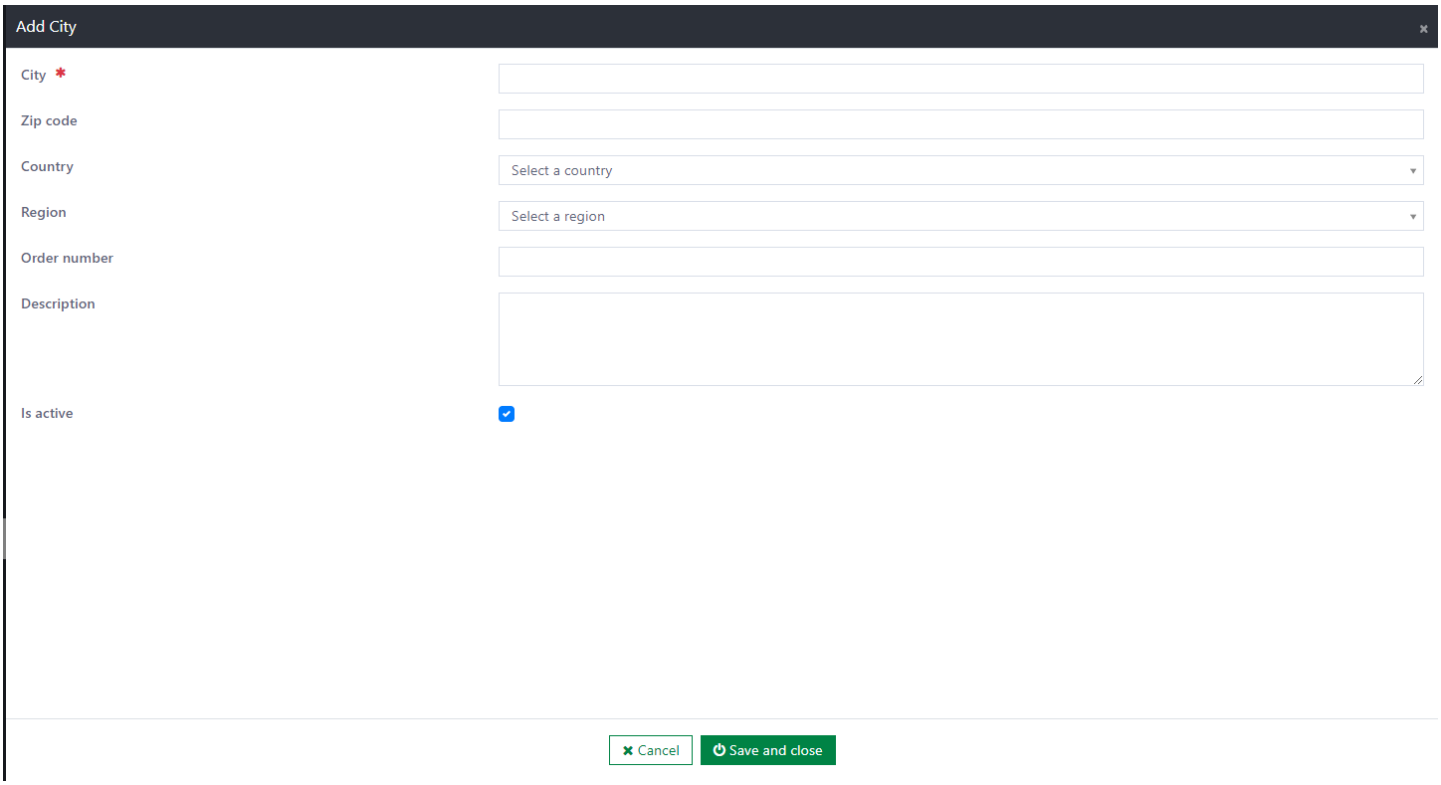
City list page is used to manage the city data. On the **City list** page, you can view all the created cities, also add, edit, delete, import, export the city data.



The following information must be provided when creating/editing a city:

- **City**: The name of the city. This field is required and unique.
- **Zip code**: The postal code of the city. It is a series of numbers that communicates information about people within different geographic groupings. This field is optional.

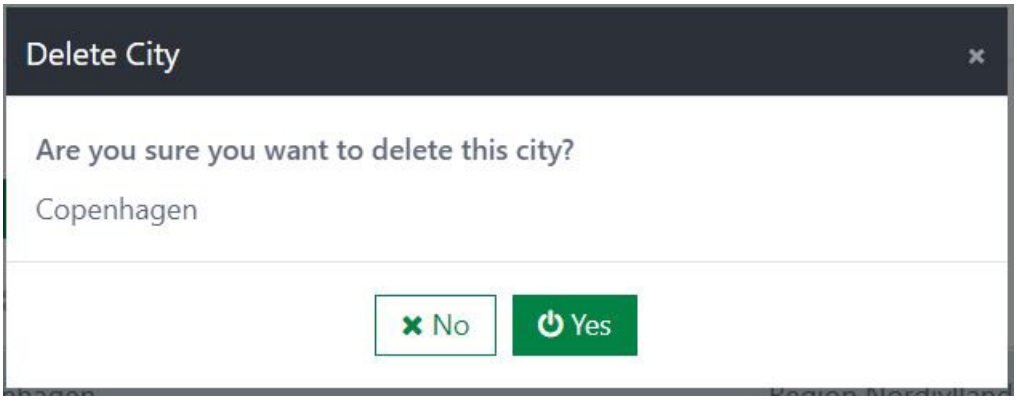
- **Country**: The dropdown of the country. It loads data from the country list in the system. This field is optional.
- **Region**: The dropdown of the region. It loads data from the region list in the system. It is an area to which the city belongs. This field is optional.
- **Order number**: This field defines the priority of the city you want to show. This field is optional.
- **Description**: This field is used to describe the information of the city. This field is optional.
- **Is active**: This field is the status of the city. The default value is **Active**.



Delete a city:

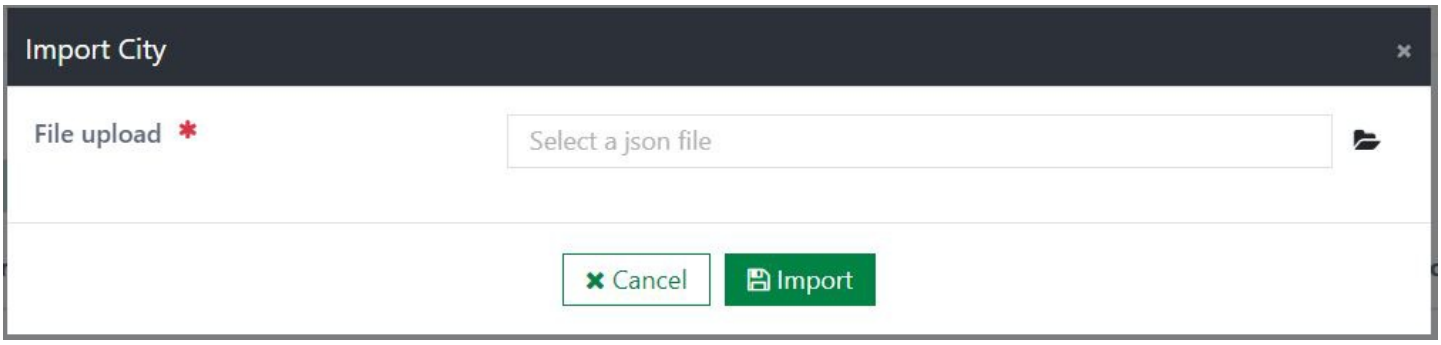
To delete a city, click on the **Delete** button. Then, a confirmation dialog will be displayed. If you select the **Yes** button, the city will be deleted immediately. If you select the **No** button, the deletion will be canceled.

- **Note:** If the city is in use in the system. It may cause an error in the UI.



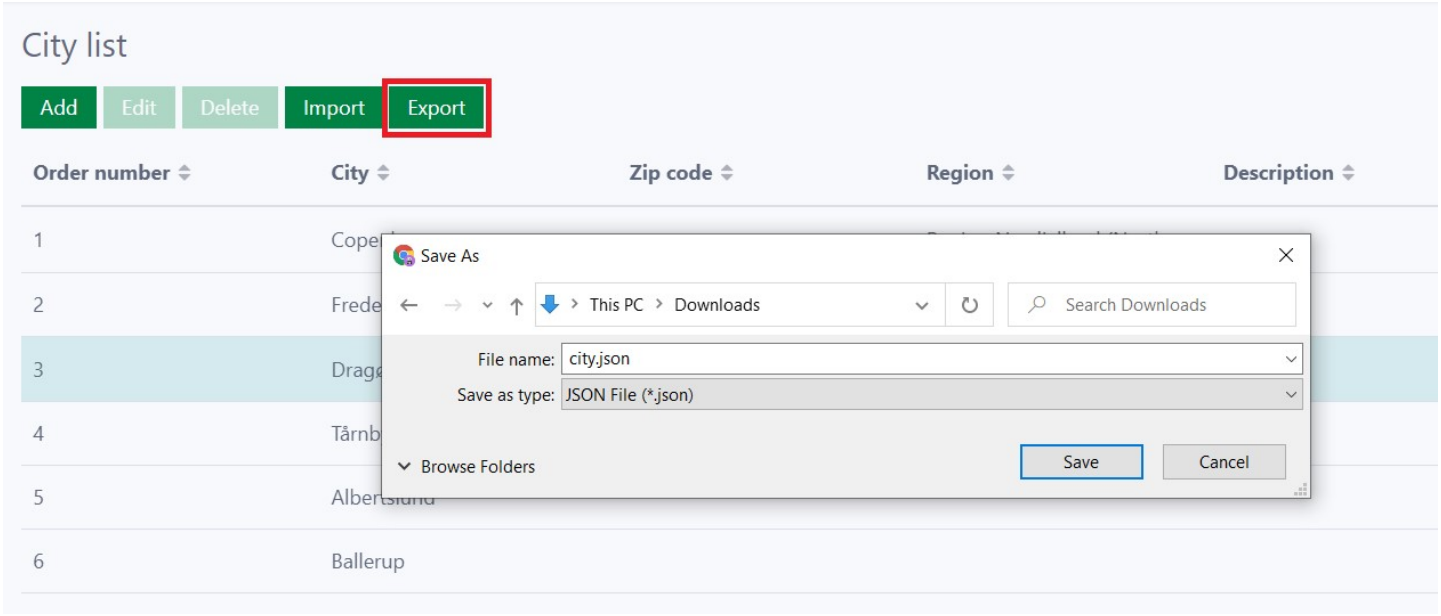
Import the city data:

To import the city data, click on the **Import** button to open the **Import City** dialog. All the city data that exists in the destination environment will be overridden. (Detected by city name)



Export the city data:

To export 1 or several cities in the list, select the city, then click on the **Export** button. Then, the JSON file of the selected cities will be downloaded immediately.



Installation

Requirements

- Casewhere 2.6.10 or later

Configuration

1. Import the product
2. Link the page to your worker sites and configure access control
3. You can generate default data in the component's configuration page

Dependencies

- None

Releases

1.1.0 - 11/07/2022

Changelog

- Administration interfaces for managing regions, municipalities, cities.
- Import/export
- Initialization data

Download (login required): Location Data v1.1.0

Roadmap

- Integration ISO code

Component: OData URI Parser

Introduction

OData URI Parser component is used to parse URI which follows the OData format into smaller components and then compose the DataSourceApiQuery object.

Features

"CwODataUriParser Build DataSourceApiQuery From OData Query String" server rule to parse

This component provides a server rule named "CwODataUriParser Build DataSourceApiQuery From OData Query String". The rule receives a few parameters and outputs a corresponding `DataSourceApiQuery` object. Here is the list of required parameters:

- **DataSourceName** [`string`]: The name of the data source that the rule builds the query for
- **QueryString** [`IDictionary<string, IEnumerable<string>>`]: Parameters from the query URI
- **WhitelistSelectFields** [`IEnumerable<string>`]: A list of fields that can be selected. If it contains a "*" element, the rule allows selecting all fields

The QueryString parameter can contain:

- **$select**: List of field to get. Example: Name,Type
- **$filter**: Filter string to query. Example: `Name eq 'John' and Type eq 'type1'`. Do not use double quote for values
- **$orderby**: String that represents single or multiple order-by statements. If the clause has "desc" keyword, the data is sorted by the defined field in descending order. Example: Name,Type desc. With this example, the data is sorted by Name in ascending order and then by Type in descending order
- **$top**: The maximum number of records that can be retrieved. Default value is 100
- **$skip**: The number of records to skip. Default value is 0

Currently, the component only supports a small set of operators in the filter parameter:

OPERATOR	DESCRIPTION	EXAMPLE
eq	Equal	City eq 'Redmond'
ne	Not equal	City ne 'London'
gt	Greater than	Price gt 20
ge	Greater than or equal	Price ge 10
lt	Less than	Price lt 20
le	Less than or equal	Price le 100
and	Logical and	Price le 200 and Price gt 3.5
or	Logical or	Price le 100 or Price gt 150

Installation

Requirements

- Casewhere 2.6.0 or later.

Setup

Import product: **Casewhere OData URI Parser**

Dependencies

- Plugin: Query Filter String Parser

Releases

1.0.0 - 14/07/2022

Changelog

- "CwODataUriParser Build DataSourceApiQuery From OData Query String" server rule
- Test workflows

Download (login required): OData URI Parser v1.0.0

Component: REST data sources

Introduction

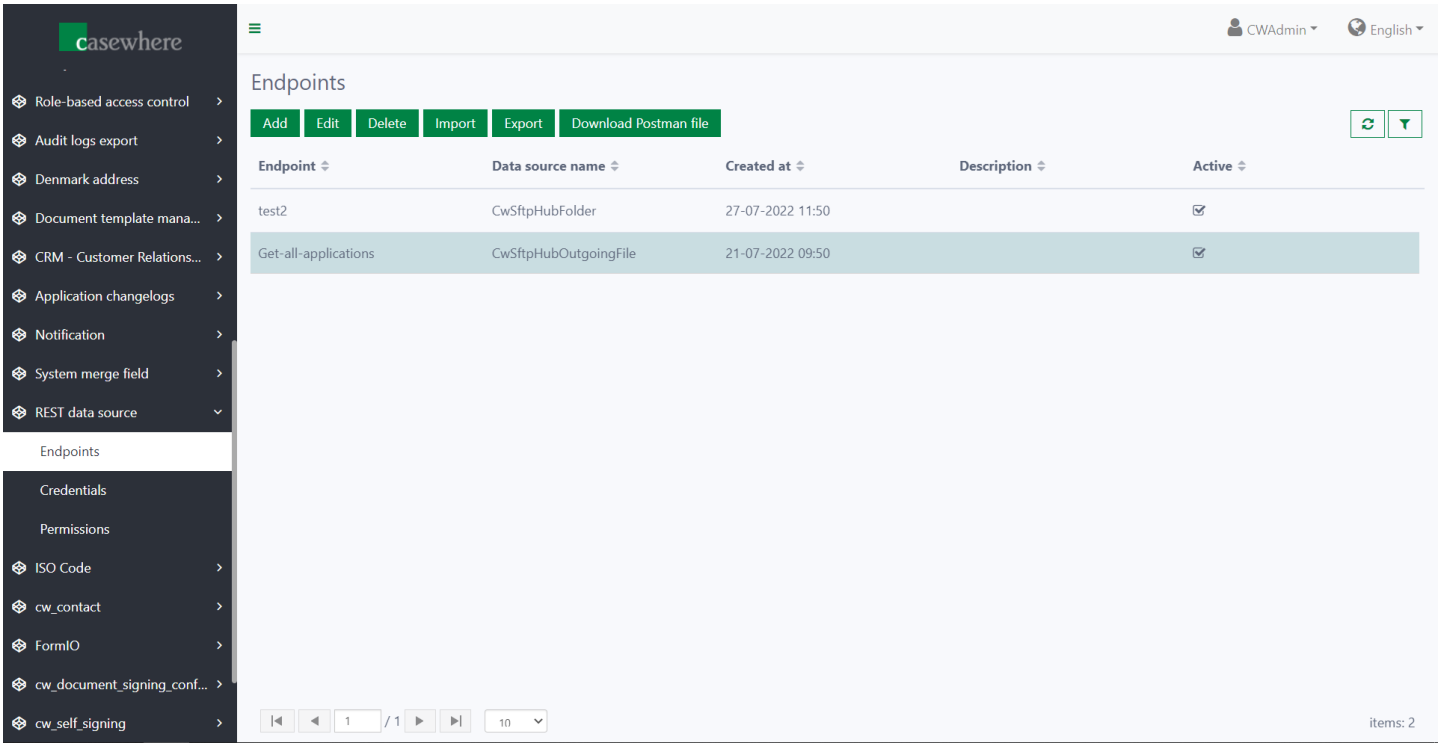
REST API is one of the most popular data exchange protocols these days. However, implementing either a good data solution or a REST API is never easy. This component is designed to provide a robust, secure, and convenient way to quickly enable access to your Casewhere data sources.

Features

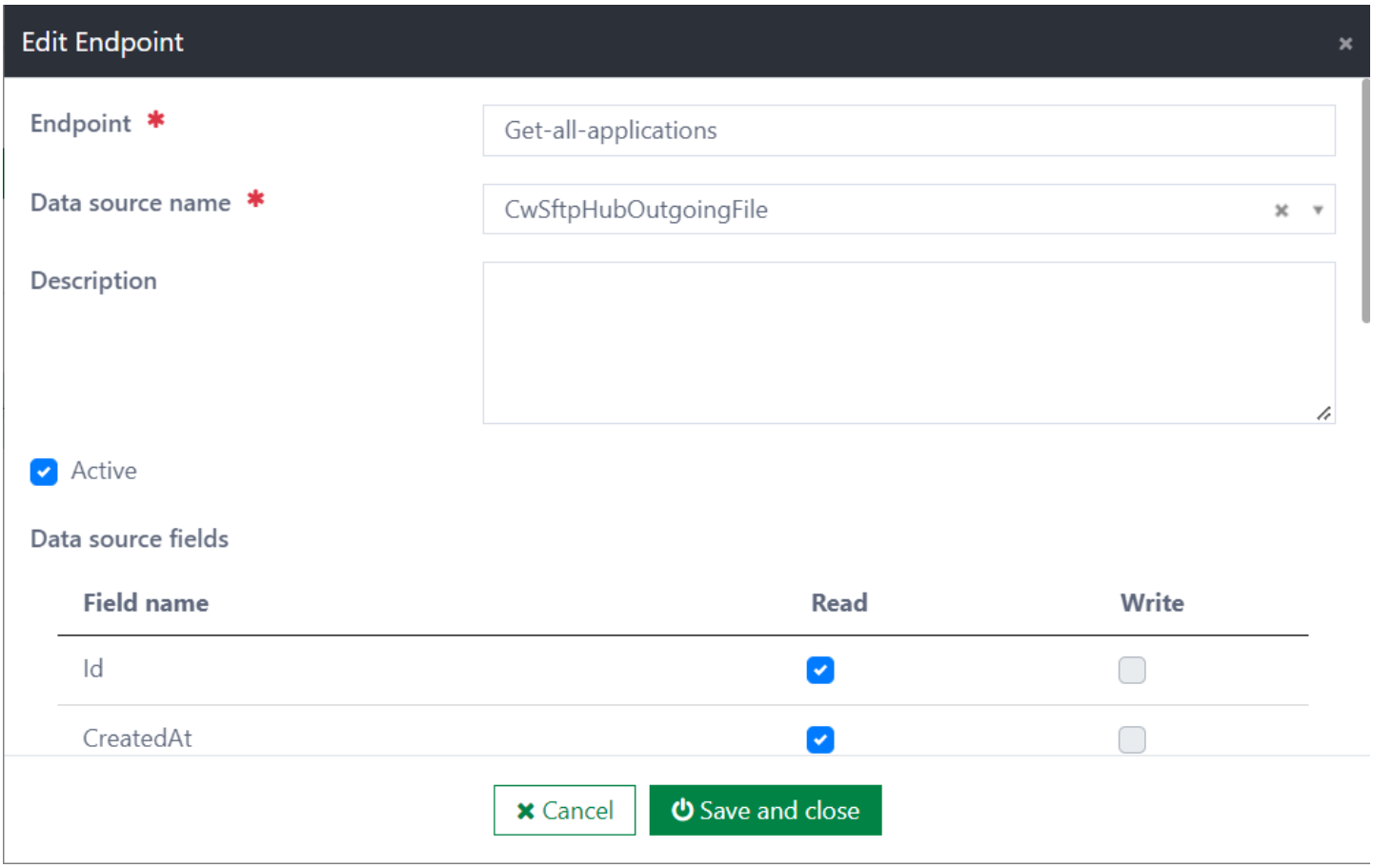
Configure Endpoints

The component provides an interface for administrating endpoints. Here the end-users can select and decide which data sources are accessible through REST API. It's also possible to configure to only return a subset of your data source by explicitly selecting the fields to respond.

On this configuration page, there are some other functions such as delete, import, export and Postman file download.



When adding or editing an endpoint, the component shows a dialog that allows inputting some information for the endpoint.

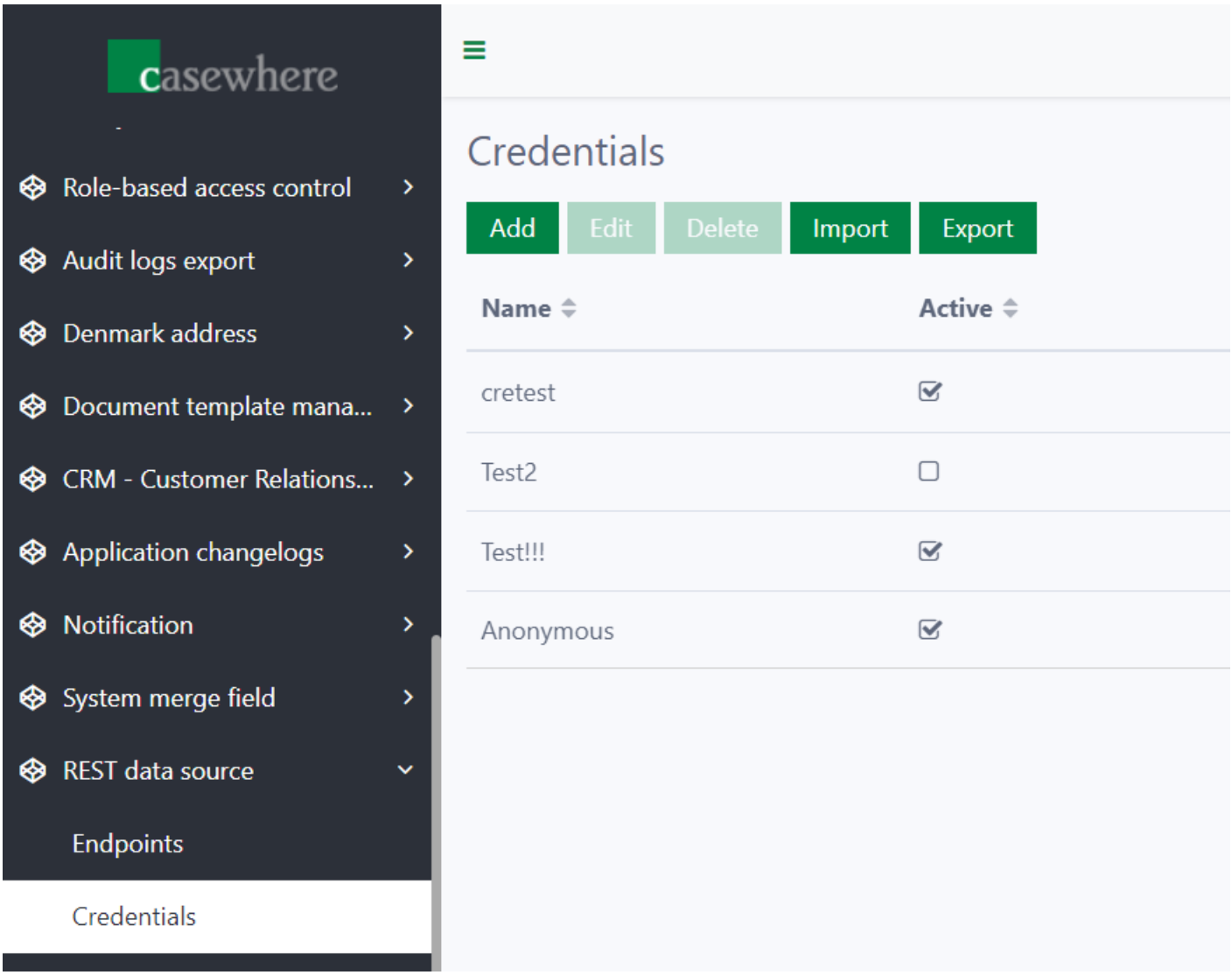


Here is a list of input fields:

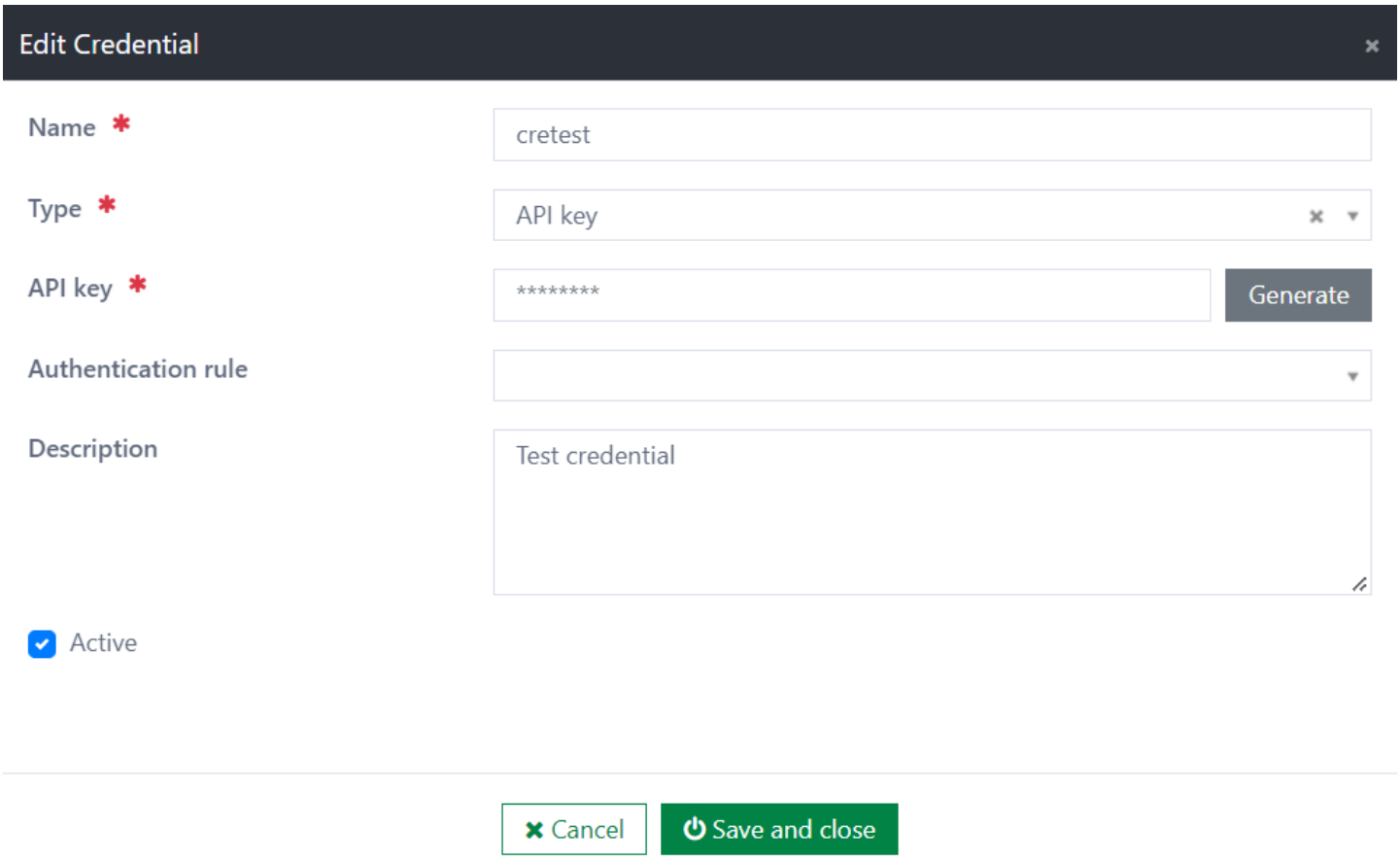
- **Endpoint** *[required]*: A string that is used by the consumer to allocate the endpoint to access. It only can contain alphanumeric characters, hyphens and underscores. Letters in the string can be in uppercase or lowercase.
- **Data source name** *[required]*: Data source that provides data for incoming requests to the endpoint
- **Description** *[optional]*: You can write anything you want in this field to describe the endpoint
- **Active**: The endpoint is available or not
- **Data source fields**: This section shows a list of data fields which belong to the selected data source. For each data field, you are able to configure that it can be read or written

Configure Credentials

To setup authorization for requests, you can define credentials. The component has a Credentials page on the worker site which allows managing credentials.



The data for adding/editing is input on the credential edit dialog.



On the dialog, you can fill in:

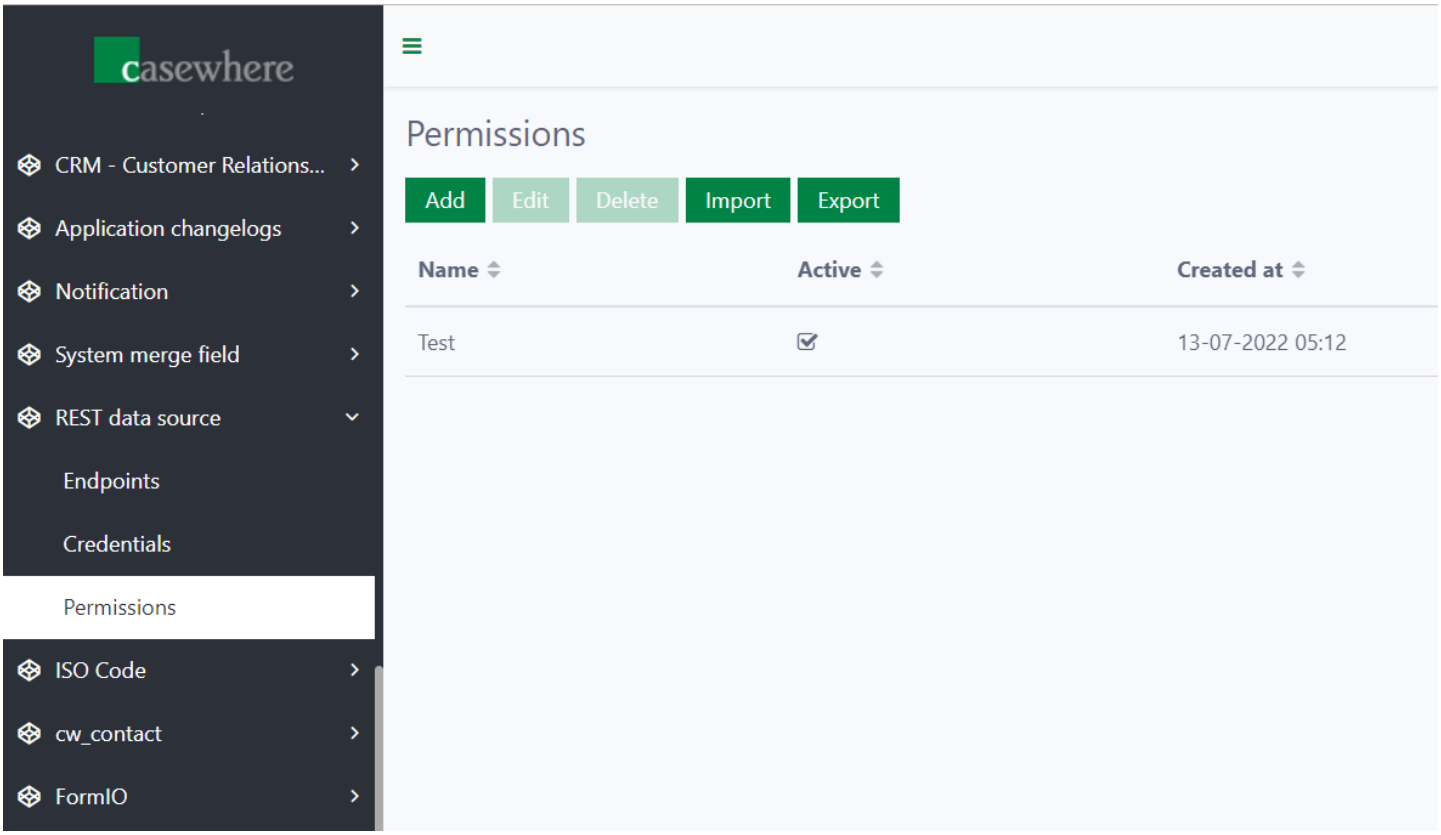
- **Name** *[required]*: A unique name for a credential to identify different credentials
- **Type** *[required]*: Type of the credential. Currently, it only supports two types which are API Key and anonymous. Other types will be implemented later
- **Authentication rule** *[optional]*: Optional custom rule to validate the request. On the dialog, this field is selectable with a drop-down. All server rules are listed. Refer to the next section to learn how to implement a rule
- **Description** *[optional]*: Give more detail about the credential

Beside those information, each credential's type has its own properties.

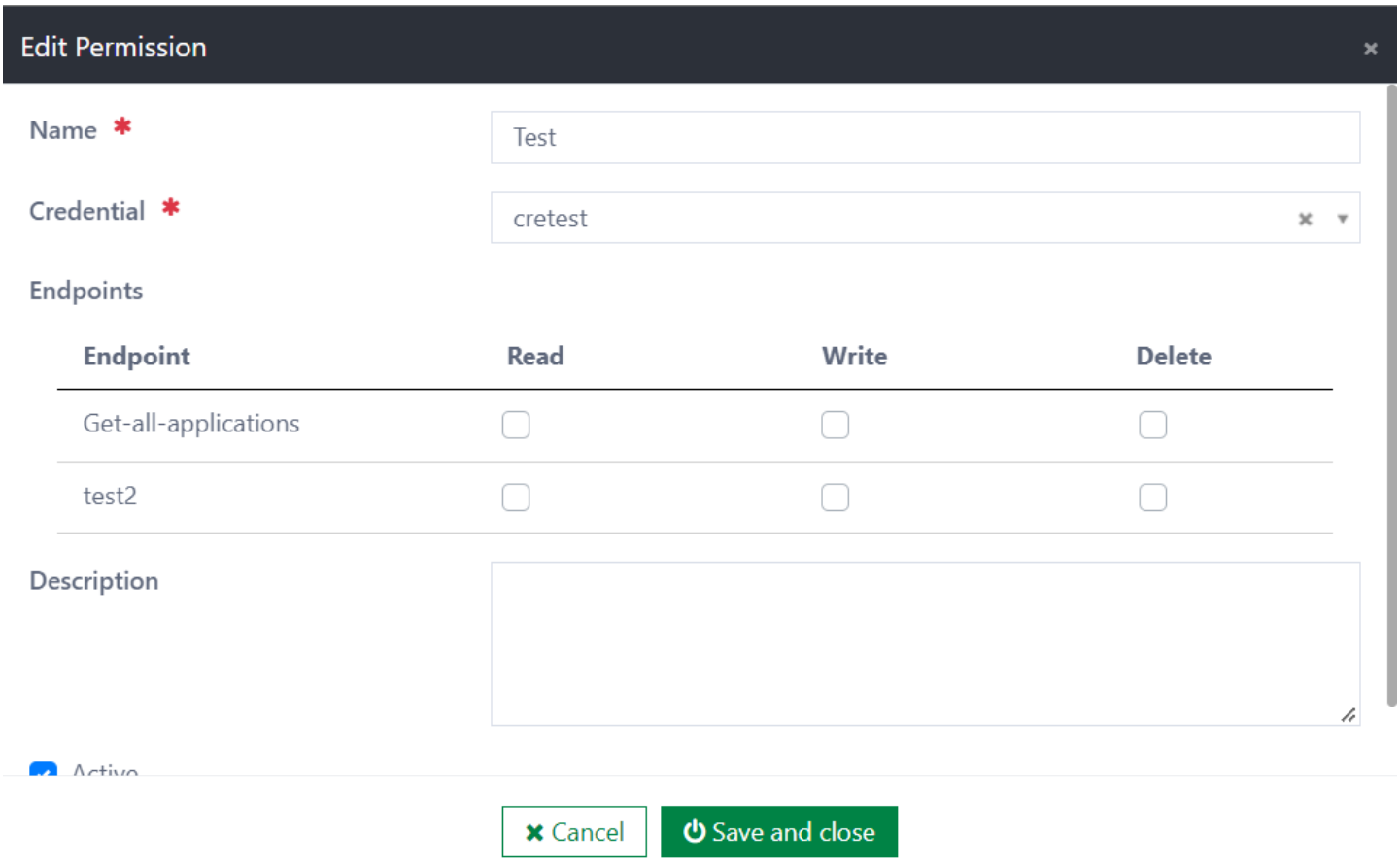
For API-key credential, the dialog shows one more property that let the operator to enter or generate an API key.

Configure Permissions

You can setup permissions for endpoints on the Permissions page.



When adding or editing a permission, you have to enter data on the permission edit dialog.



Here is input fields:

- **Name** *[required]*: A name to be an alias for the permission. It is not crucial to be unique
- **Credential** *[required]*: Select a created credential from the drop-down list
- **Endpoints**: Permission for each endpoint. Permission for each endpoint can be Read, Write and/or Delete
- **Description** *[optional]*: Give more detail about the permission

- **Active**: The permission is available or not

GET Request

Consumers can make GET requests to retrieve data flexibly. To support flexibility, the component provides a sufficient syntax. Below is a list of parameters that can/must be included in requests.

- **$endpoint** *[required]*: Name of an endpoint that the request is made for. Besides, the consumer can define this parameter with an ID of the needed data. Examples:
  - /dynamic/datasources?$endpoint='Customer'
  - /dynamic/datasources?$endpoint='Customer(Customer/5a2fev1br94rehfefef2)'
- **$top** *[optional]*: Maximum number of data records to get
- **$skip** *[optional]*: Number of data records that the consumer wants to skip
- **$search** *[optional]*: Keyword for full text search. Must be put between a pair of quotes
- **$filter** *[optional]*: Condition to query. The filter string follows OData syntax. Refer to OData URI Parser component for supported operators
- **$select** *[optional]*: Fields' names separated by commas. Only fields that are configured to be able to read are returned. If this parameter is undefined, the component will respond with all allowed fields
- **$orderby** *[optional]*: Define data's order. Can order by multiple fields, separated by commas. Here are two cases:
  - Fields that are used to sort in descending order: [Field name] desc
  - Otherwise, [Field name]

Example:

```
[API URL]/dynamic/datasources?$endpoint='Customer'&$top=5&$skip=2&$search='blue'&$filter=Description eq 'lorem ...' and ActiveDate eq 2022-06-04T15:31:36.000Z&$select=Description,ActiveDate&$orderby=CreatedDate desc,ModifiedAt
```

Response

The component will respond differently for different cases. Here is the table that describes all status codes that are used explicitly by the component.

CODE	DESCRIPTION	PAYLOAD
200	Request is valid and data is found	`{ Data: {...}, TotalCount: 123 }`
400	Request is invalid	Error message if any
401	Request is unauthorized	
403	Request to inaccessible endpoint or request is unauthenticated	
404	Endpoint is not found	
500	An error occurs when processing the request	Correlation ID and error message if any

Installation

Requirements

Casewhere 2.6.10 or later

Configuration

1. Import the product
2. Link configuration pages to your worker sites and configure access control
3. Configure endpoints, credentials and permissions

Releases

1.0.0 - 10/08/2022

Changelog

- Configuration pages
- GET endpoint

Download (login required): REST data sources v1.0.0

Component: SFTP hub

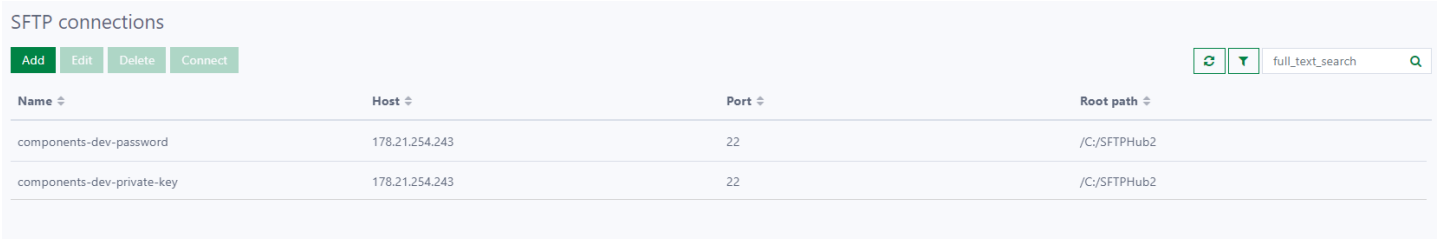
Introduction

Exchanging files through SFTP hub is an essential function in almost Casewhere applications. SFTP hub component is developed to simplified and standardized the process to integrate with the hub. The integrated application can easily send and receive files without interacting directly with the SSH Plugin.

Features

Manage connection info

The component provides a page to manage the server connection. The system can use a connection that was created when invoking an SSH method to interact with the SSH Plugin. If connection info does not pass as parameters, it will use plugin settings by default.

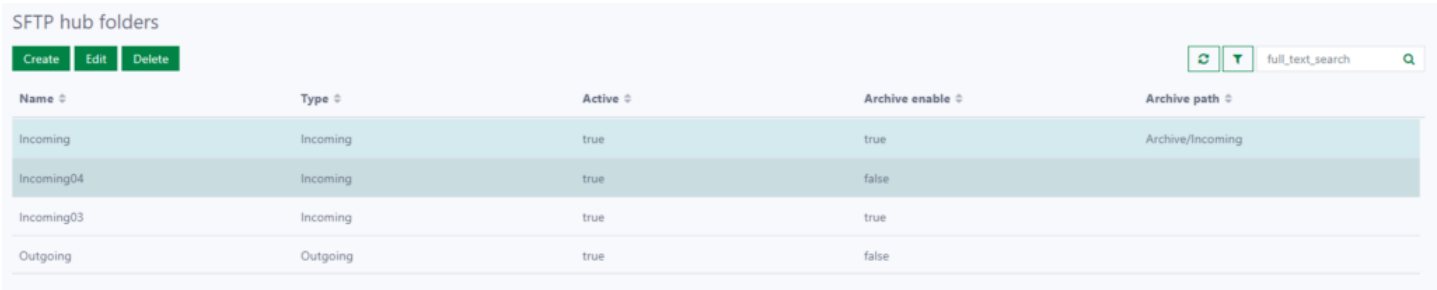


The following information must be provided when creating a connection:

- **Name**: the name of connection.
- **Host**: the host to connect, can be an IP address or URL.
- **Port**: the port to connect, must be a valid port number (0 - 65535).
- **AuthenticationType**: the type to authenticate to a server.
    - **Password**: for using password.
    - **Certificate**: for using public / private key pair.
- **Username**: the username to connect.
- **Password**: the password to connect, only used in case AuthenticationType is Password.
- **PrivateKey**: the private key to connect, only used in case AuthenticationType is Certificate. Please refer to this section to know how to generate a private key that is used for authentication.
- **Passphrase**: the passphrase to protect the private key, only used in case AuthenticationType is Certificate.
- **RootPath**: the destination path to connect.

Manage hub folder structure

The component provides a page to manage the hub folder structure.



The following information must be provided when creating a folder:

- **Name**: the name of folder, it should be identical to the folder name under the hub's root path.
- **Type**: the type of folder

    - **Incoming**: for receiving files from the hub.
    - **Outgoing**: for sending files to the hub.

- **ArchiveEnable**: is the integration responsible for archiving files. For now, it only applies to incoming folders.
- **ArchivePath**: the path to the archive folder.
- **ConnectionInfoId**: the information to connect to the SFTP server. Note that a folder belongs to only one server.

Receive files

The component provides the workflow **CwSftpHub Receive File** for receiving files that can be used to integrate with the application's workflow triggers. One typical scenario is using a scheduler to scan the incoming folders and download files, if any, periodically. A **CwSftpHubIncomingFile** object will be created if a file is downloaded successfully. If `ArchiveEnable=true`, the component will automatically archive the file by moving it to the specified folder in the hub.

Below is the data structure of the **CwSftpHubIncomingFile**:

- **FileName**: the file name.
- **FolderId**: the configured **Incoming** folder.
- **Content**: the file content.
- **Status**:
    - **Received**: when the file is downloaded successfully.
    - **Archived**: when the file is moved to the archive folder.

- **RelatedEntityId**: optional, refer to an application's Entity Id.
- **SyncDate**: the time when the file is downloaded (in UTC+0 format).
- **HubCreatedDate**: the time the file was created in the hub (in UTC+0 format).

Send files

The application can send a file to the hub by creating a **CwSftpHubOutgoingFile** object. The application can choose to send either synchronously or asynchronously by turning on or off the **IsAsync** attribute. Then, the component will automatically send the file to the hub through a built-in data trigger. In addition, the component will try to resend the file for a specific time if the file is failed to send for some reasons.

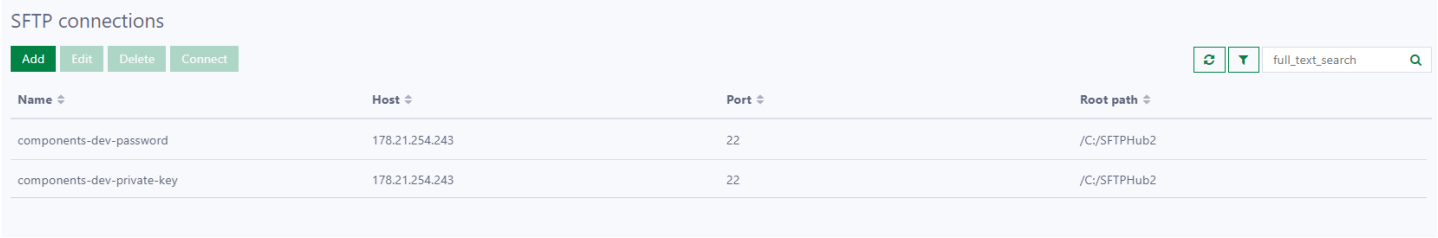
Below is the data structure of the **CwSftpHubIncomingFile**:

- **FileName**: the file name.
- **FolderId**: the configured **Outgoing** folder.
- **Content**: the file content.
- **Status**:
    - **Created**: the data object is created.
    - **Ready**: ready to send.
    - **SentSuccess**: the file is sent successfully.
    - **SentFailed**: the file is failed to send.

- **RelatedEntityId**: optional, refer to an application's Entity Id.
- **SyncAttempts**: the number of upload attempts has been triggered.
- **MaxSyncAttempts**: the maximum retry times if the file sending is failed.
- **IsAsync**: upload immediately or wait in the queue.

Manage connection

The application allows user to manage the connections.



The following information must be provided when creating a connection:

- **Name**: the name of connection.
- **Host**: the host to connect, it can be an IP address or URL.
- **Port**: the port to connect, valid port number should be in the range (0 - 65535).
- **AuthenticationType**: the type to authenticate the connection.
    - **Password**: using password.
    - **Certificate**: using public/private key pair.

- **Username**: the username to connect.
- **Password**: the password to connect, used in case AuthenticationType is Password.
- **PrivateKey**: the private key to connect, used in case AuthenticationType is Certificate.
- **Passphrase**: the passphrase to protect the private key, used in case AuthenticationType is Certificate.
- **RootPath**: the destination path to connect.

Installation

Requirements

- Casewhere 2.6.0 or later.

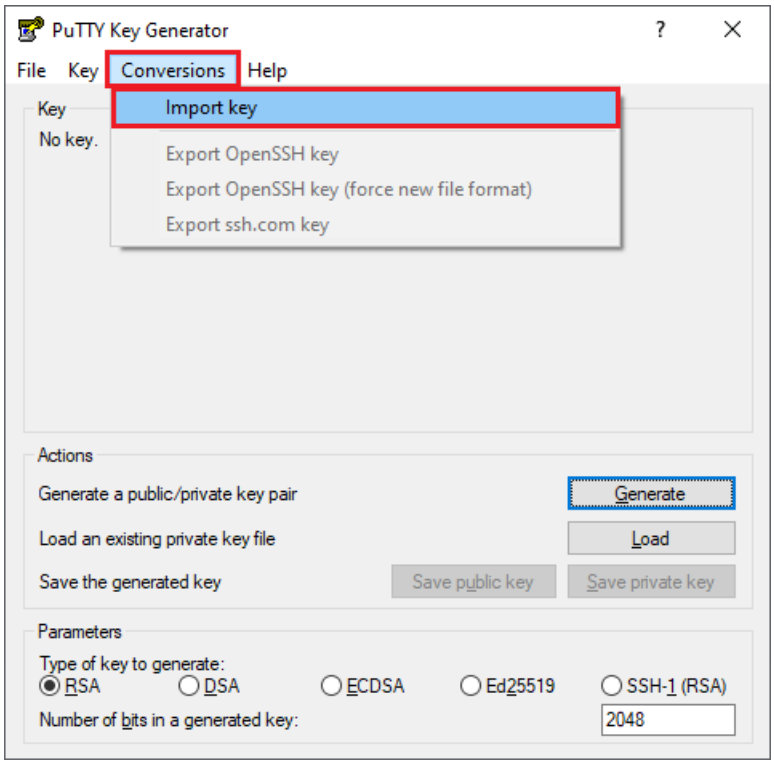
Configure SSH Plugin

Get private key from a .ppk file.

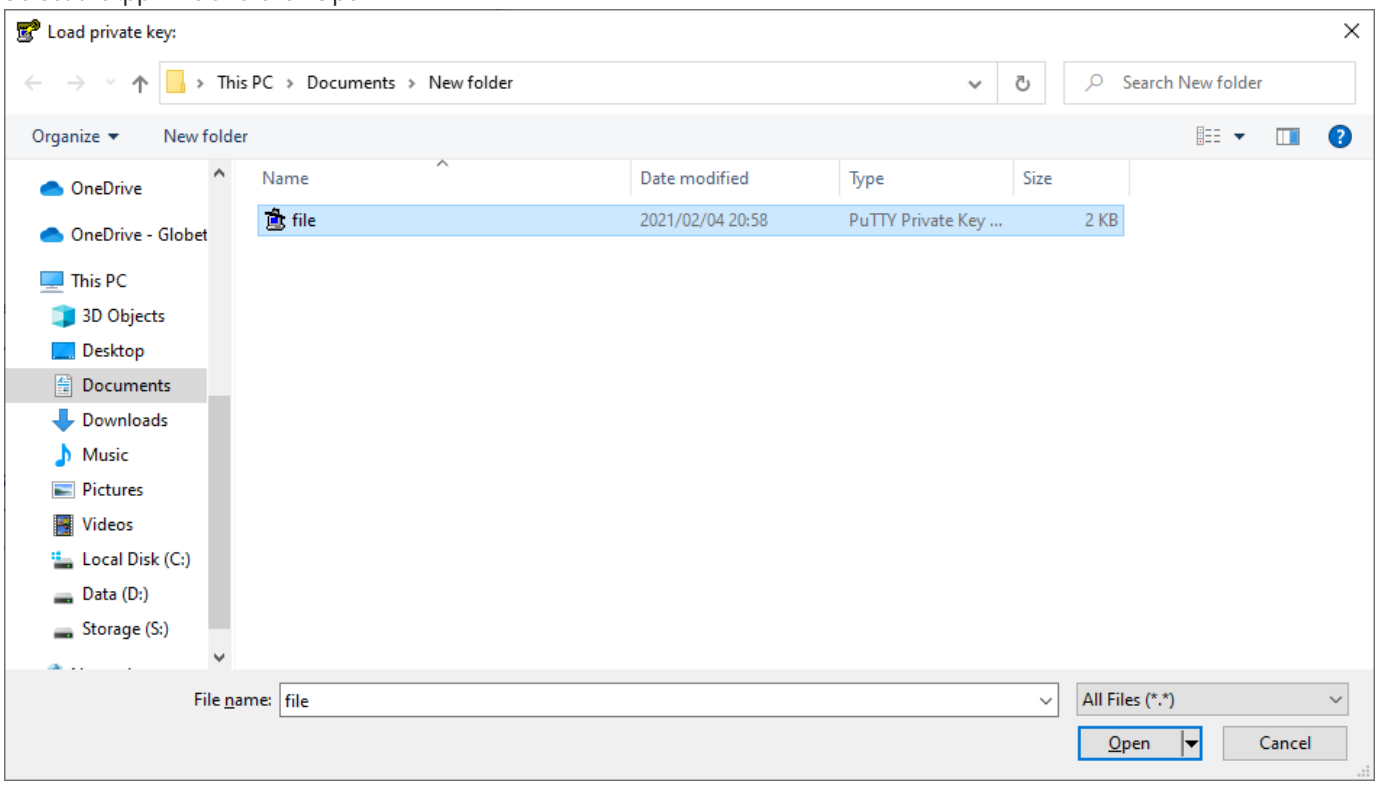
- Download and install PuTTY.
  Follow the instructions on this page: https://www.ssh.com/ssh/putty/windows/install to dowload and install PuTTY.

- Convert the .ppk file to the private key file. After installing PuTTY, run the PuTTYgen application and follow the steps below to get and install a private key:

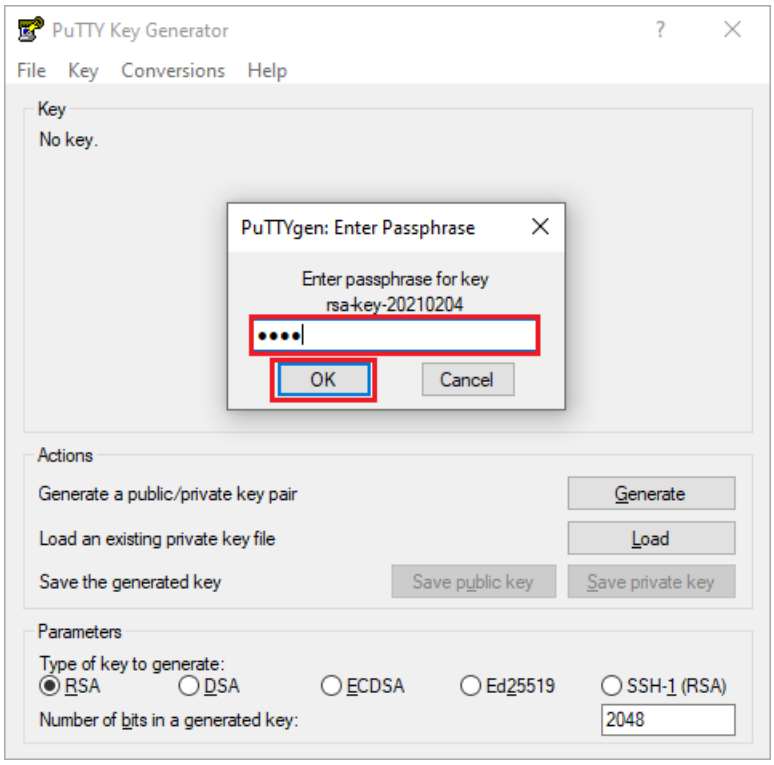
1. Go to Conversions menu and select Import key



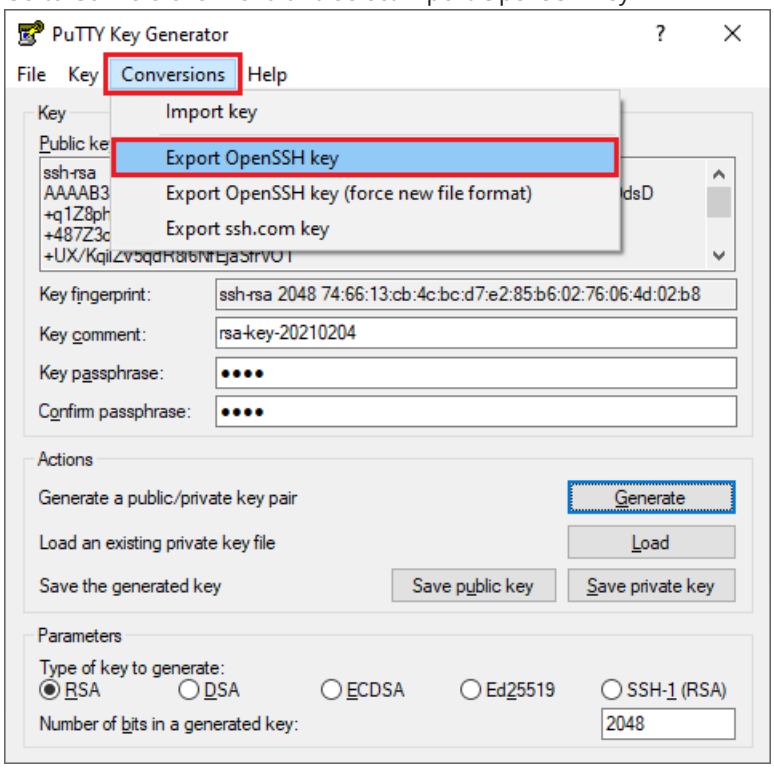
2. Select the .ppk file and click Open



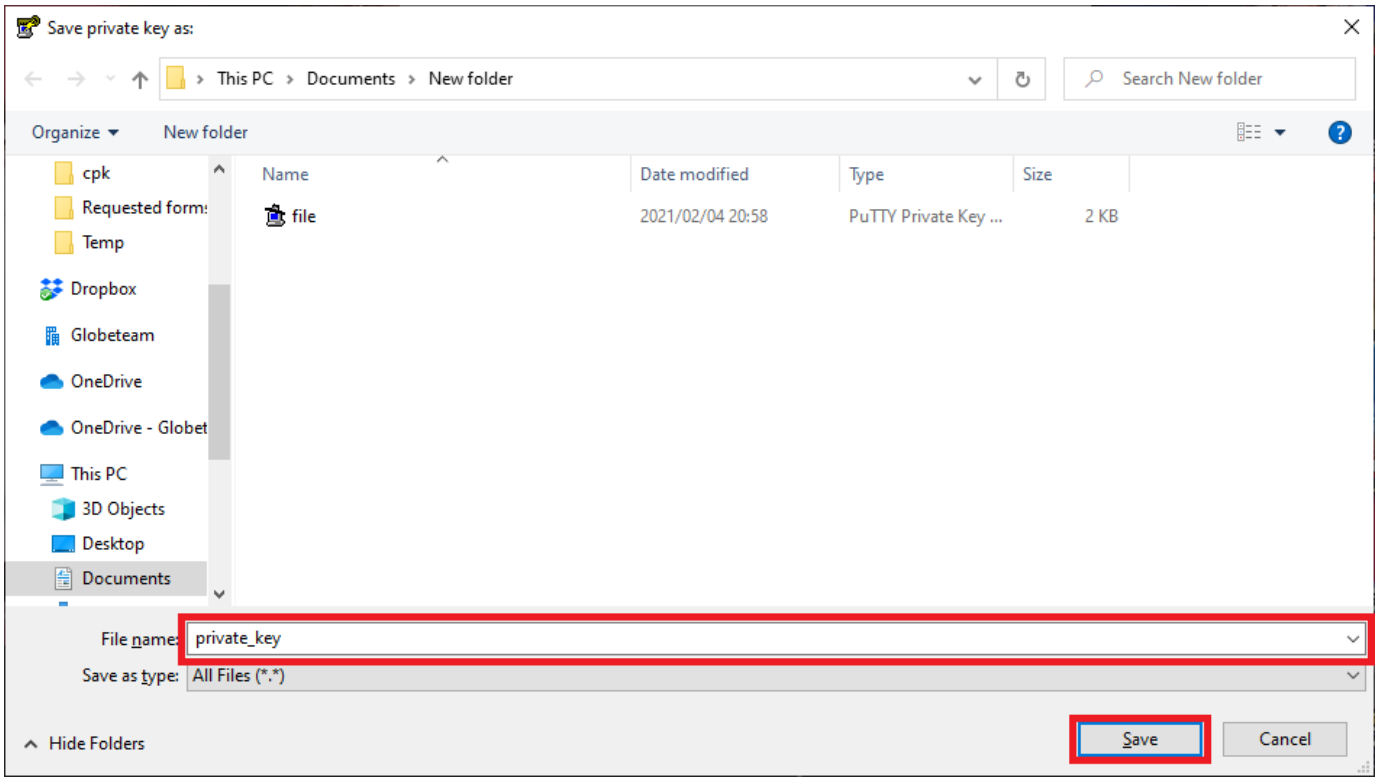
3. Enter the passphrase that protects the key in the .ppk file



4. Go to Conversions menu and select Export OpenSSH key



5. Select a place to export the private key file and enter desired name for the file



Configure the apllication

After installing, users must configure the application as below:

```
{
  "Host": // Host IP,
  "Port": //Port,
  "Username": //Username to access the host,
  "Password": //Password,
  "RootPath": //Path to the root folder
}
```

Releases

1.1.0 - 24/01/2022

Changelog

- Manage SFTP connection.
- Support connection parameters.
- Support connect by private / public key pairs.

Download (login required): SFTP hub v1.1.0

1.0.0 - 19/07/2021

Changelog

- Manage hub folder structure
- Receive files
- Send files

Download (login required): SFTP hub v1.0.0

Component: Performance Report

Introduction

Every product has different requirements about creating a Service Level Agreement (SLA) report. It helps to monitor the system during the operation process. There are many type of SLA reports based on their purpose. This component focuses on creating a SLA report based on the response time. So, it's built

- To create the SLA report periodically
- To deliver it automatically

A SLA report of the response time provides the following information given.

- A given list of transaction types need to measure and each transaction type can contain one more endpoints
- A response time threshold for each transaction type
- Operation time duration in a day
- A percentile (like 95%) to determine a measurement is passed or failed

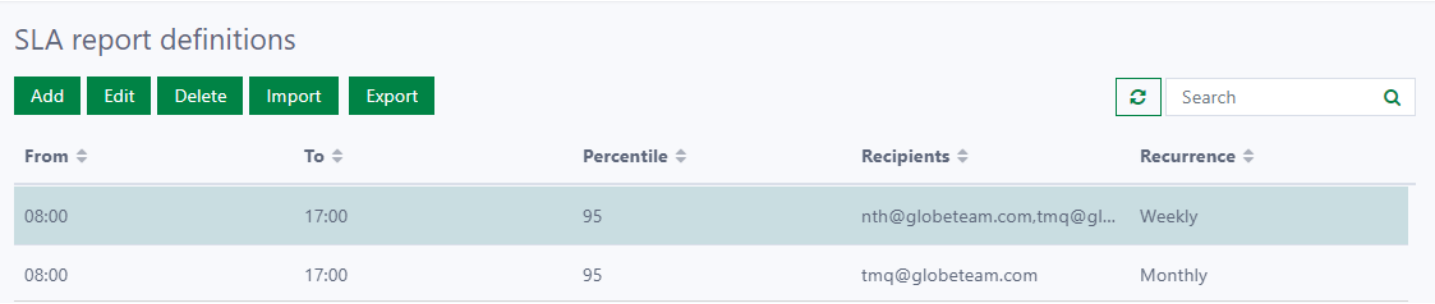
Features

The component includes two main parts. The one is a plugin to be used to connect the database and calculate the measurements for the transaction types. The other is a page to manage SLA report definitions. Each definition include two sections: report settings and delivery settings

Manage SLA report definitions

Provide a list of the SLA report definition with the basic operations such as Create, Edit, Delete, Import and Export as below. You can view the basic information in each report definition like

- The operation time duration is required for transaction types,
- The percentile of measurements
- Who will receive the SLA report
- Which recurrence the report is run and delivered automatically.

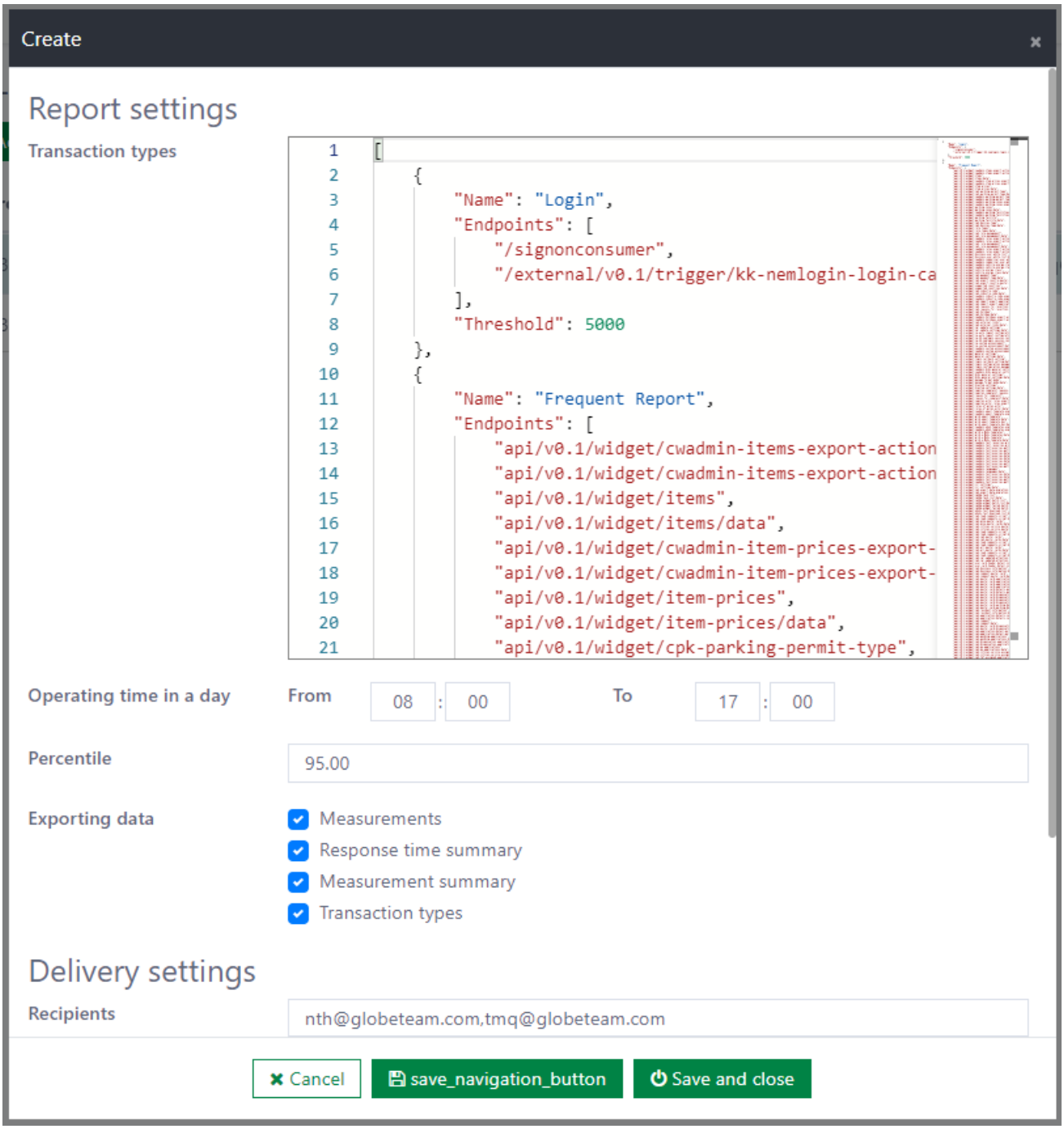


To create a new report definition, the following information need to be provided

Report settings

To define what need to be measured to evaluate the operational efficiency. It includes

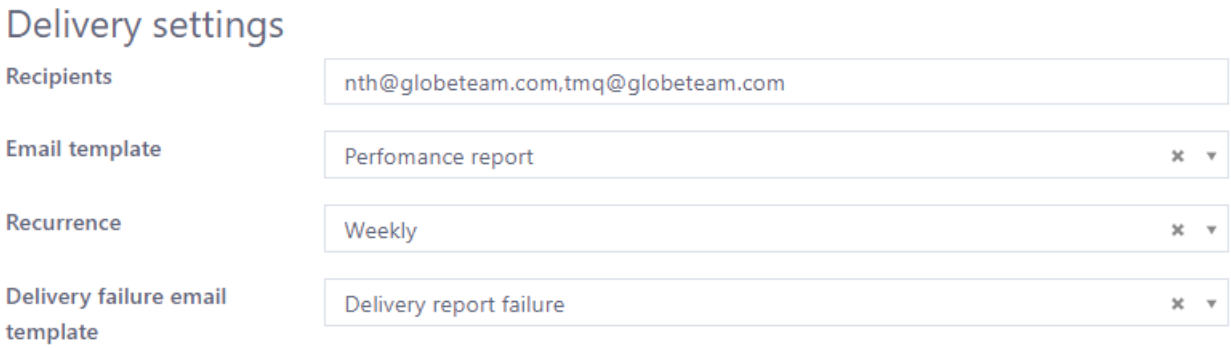
- Transaction types (Required): Determine a list of transaction types interested in the SLA report. Each transaction type has a name, a list of the endpoints, and the response time threshold.
- Operating time in a day (Required): Determine a duration in a day so only the endpoints happened in the duration are collected during calculating measurements.
- Percentile (Required): Determine the percentile value is used
- Exporting data (Required): Determine which data will be encompassed in the SLA report



Delivery settings

To define who will receive the SLA report when the report is created periodically. It includes

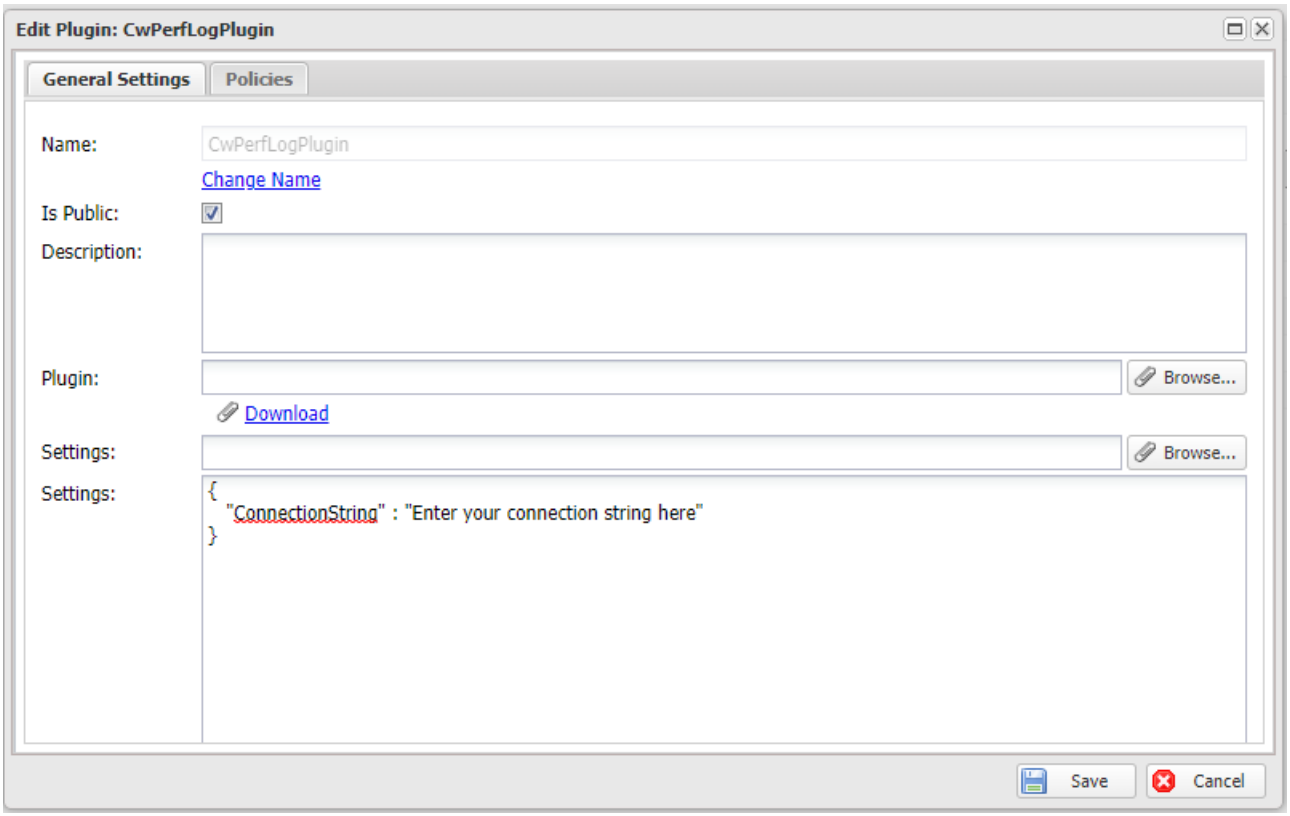
- Recipients (Required): Determine who will receive the report, even when the process of generating it is failed
- Email template (Required): Determine which email template is used when the report is generated successfully
- Recurrence (Required): Determine which interval (weekly/monthly) the report is run as planned.
- Delivery failure email template (Required): Determine which email template is used when the report is generated failed.



You can change the settings by editing an existing definition. Furthermore, you can export a list of SLA report definitions under a JSON file and share it with someone. Otherwise, you can import it from a given JSON file.

Connect to a database

The SLA report is calculated based on the data from the database. You need to change the settings of the **CwPerfLogPlugin** plugin, so it can connect to the proper database.



Limitations

- For each Casewhere instance, the component only can create many SLA reports against an database at the same time. It means you cannot create SLA reports with multiple databases.
- Generating SLA report is retried 3 times before an email of failure is sent to the recipient. The admin can manually retry it via the workflow monitor.

Dependencies

It depends on the following components

- Casewhere Mail

Installation

Requirements

- Casewhere 2.6.10 or later

Configuration

- Import two products: **Casewhere PerfLog Plugin** and **Casewhere Performance Report**
- Create the SLA report definition that follows the customer's requirement
- Change the plugin setting to connect the database

Releases

1.0.0 - 03/17/2022

Changelog

Download (login required): [TBD]

Component: Document management

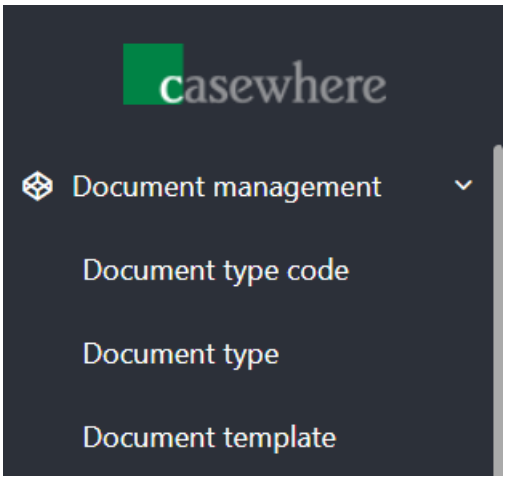
Introduction

The **Document management** component is used to manage the Document code, Document type and Document template. A document template must belong to a document type. The Document template is used in generating the document.

Features

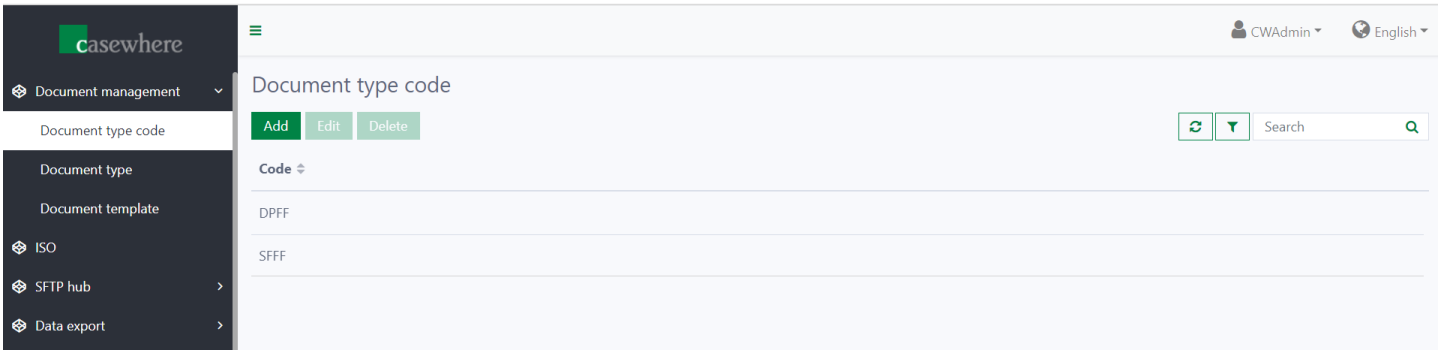
Administration interfaces for managing the following:

- Document code
- Document type
- Document template



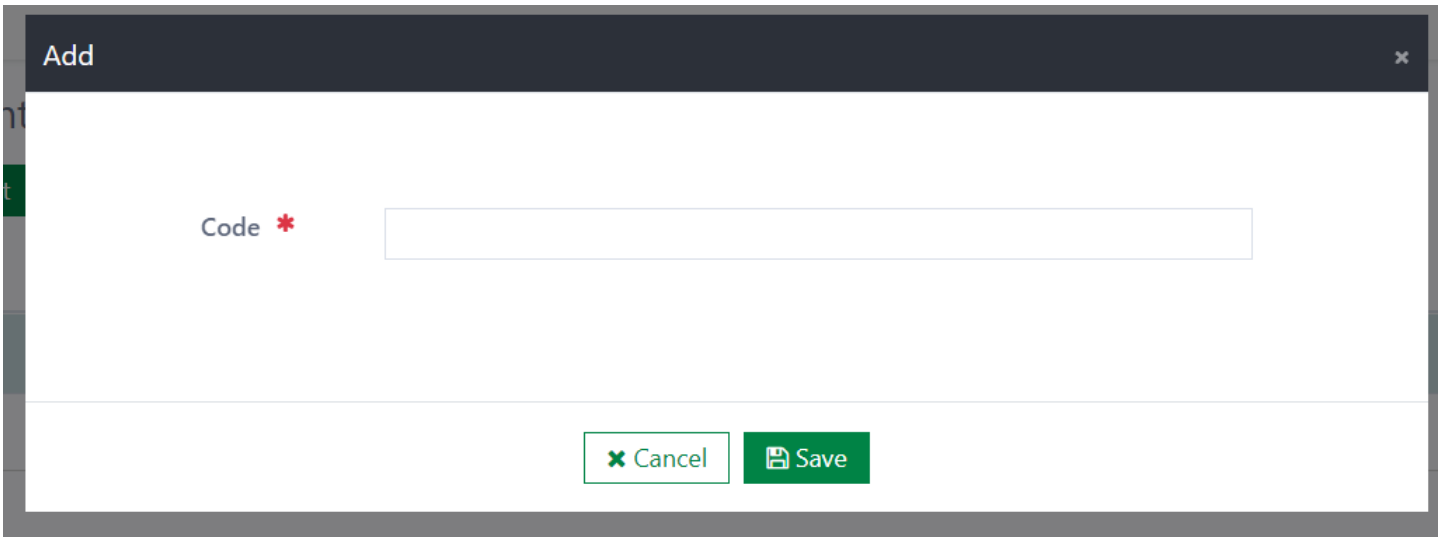
Manage document code

The component provides a simple page to create/edit/delete code for document type, the codes are used for the case a document type has a specific business rule that other document types don't have according to the project's requirement. In this page, you can view all the codes, also add, edit, delete the code data.



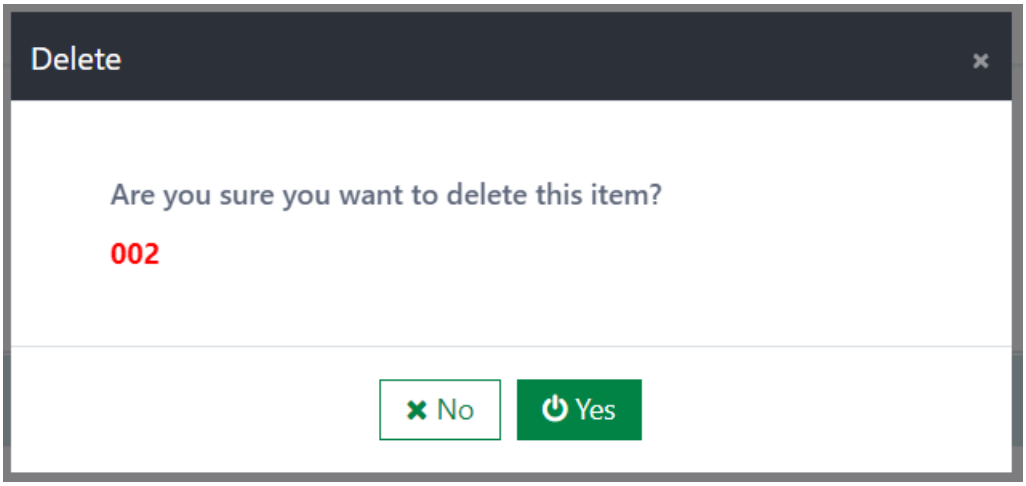
The following information must be provided when creating/editing code:

- **Code:** This field is required.



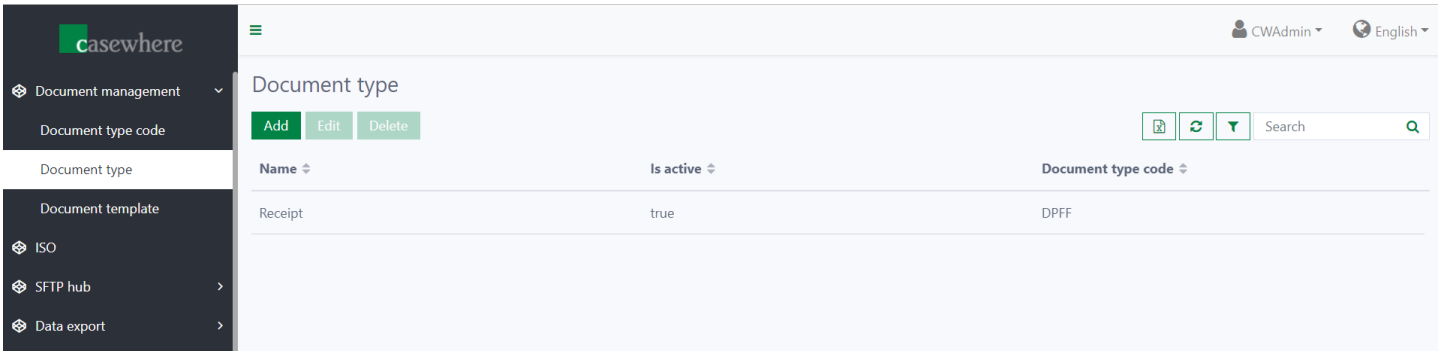
Delete code :

To delete a code, click on the Delete button. Then, a confirmation dialog will be displayed. If you select the Yes button, the code will be deleted immediately. If you select the No button, the deletion will be cancelled.



Manage document type

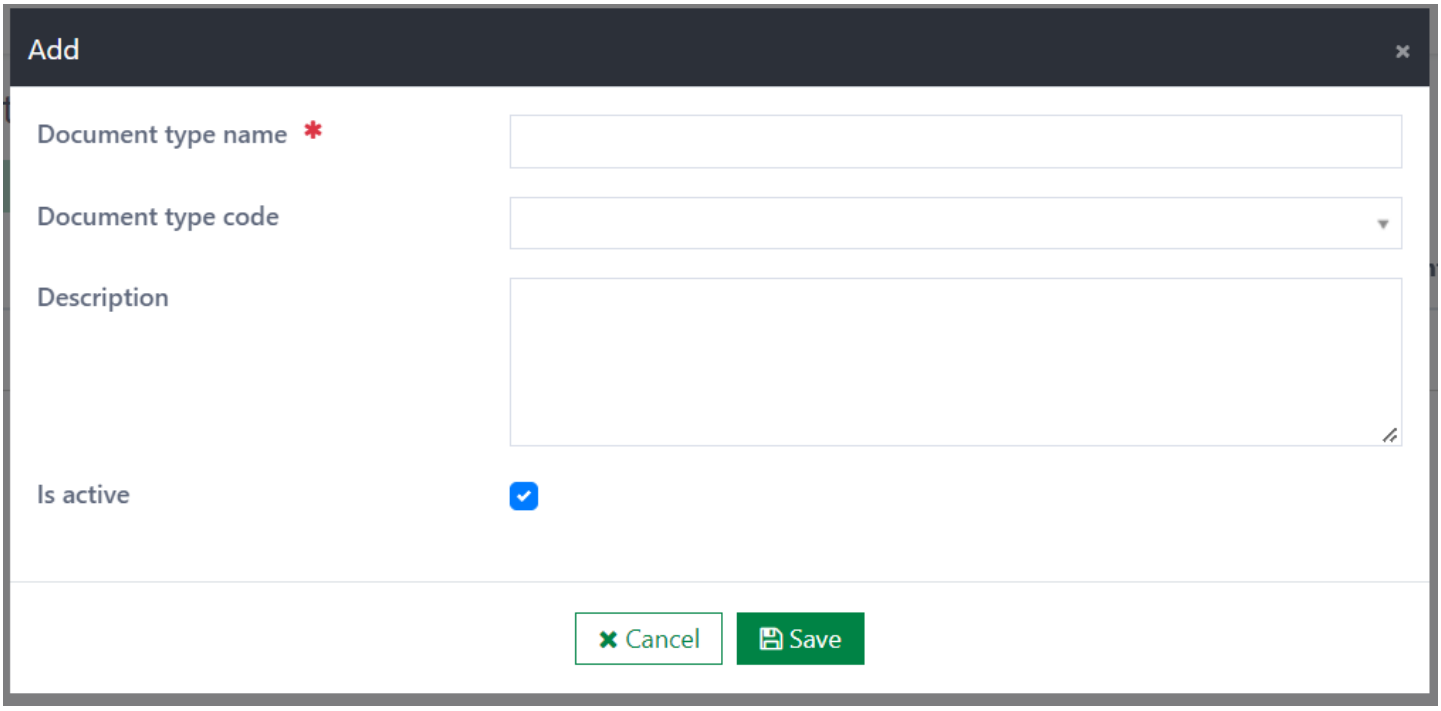
The component provides a page to manage all document types that are used in document template. In this page, you can create/edit/delete the document type



The following information must be provided when creating/editing document type:

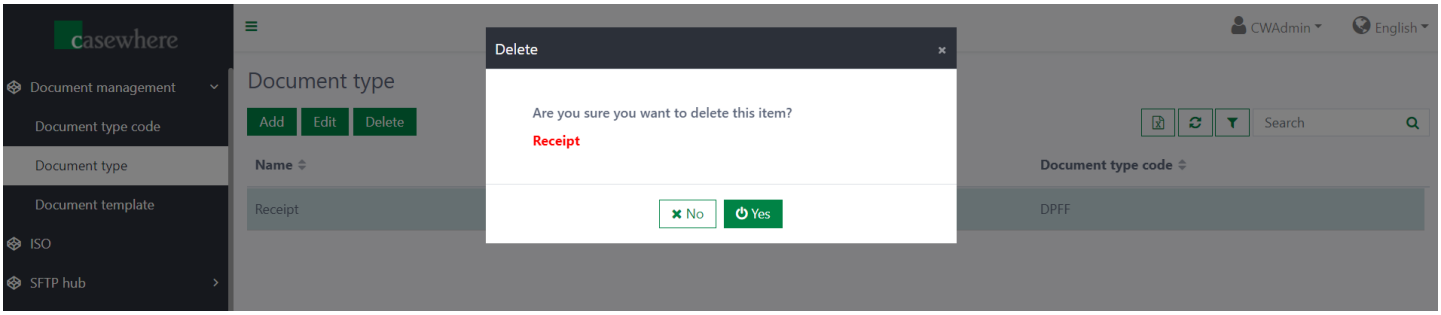
- **Document type name**: The name of the document type. This field is required and unique.
- **Document type code**: This field is a dropdown list including all document codes that are configured in the Document code page. This field is optional. In case of editing document type, the document type code cannot be edited.
- **Description**: This field is used to describe the other information of the document type. This field is optional.
- **Is active**: This field is the status of the document type. The default value is Active. When the document type is active, it will

be available in the Document types list of the Document template configuration. In editing, the document type can only be changed to inactive if it is not used for any document template.



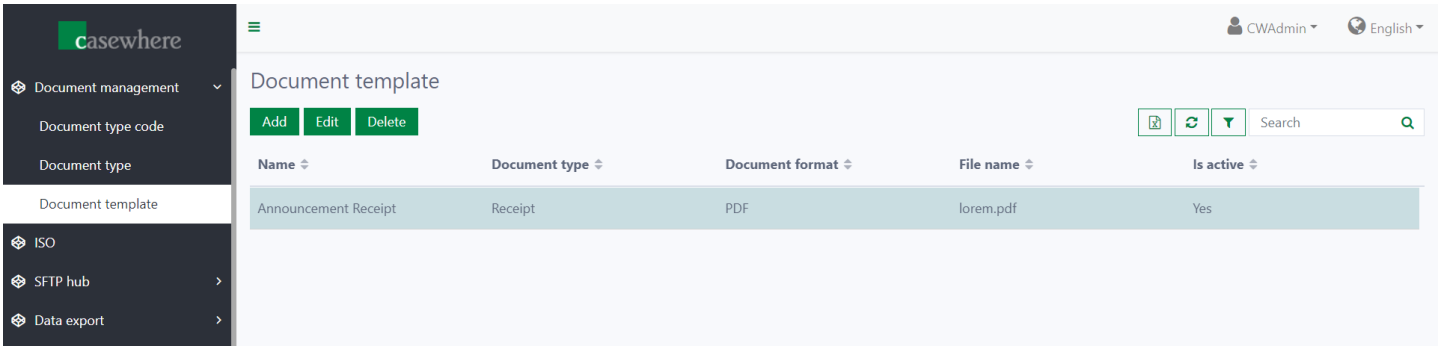
Delete a document type:

To delete a document type, click on the **Delete** button. Then, a confirmation dialog will be displayed. If you select the **Yes** button, the document type will be deleted immediately. If you select the **No** button, the deletion will be cancelled. The document type which is in use cannot be deleted. An error message will appear when user tries to delete the in-use document type.



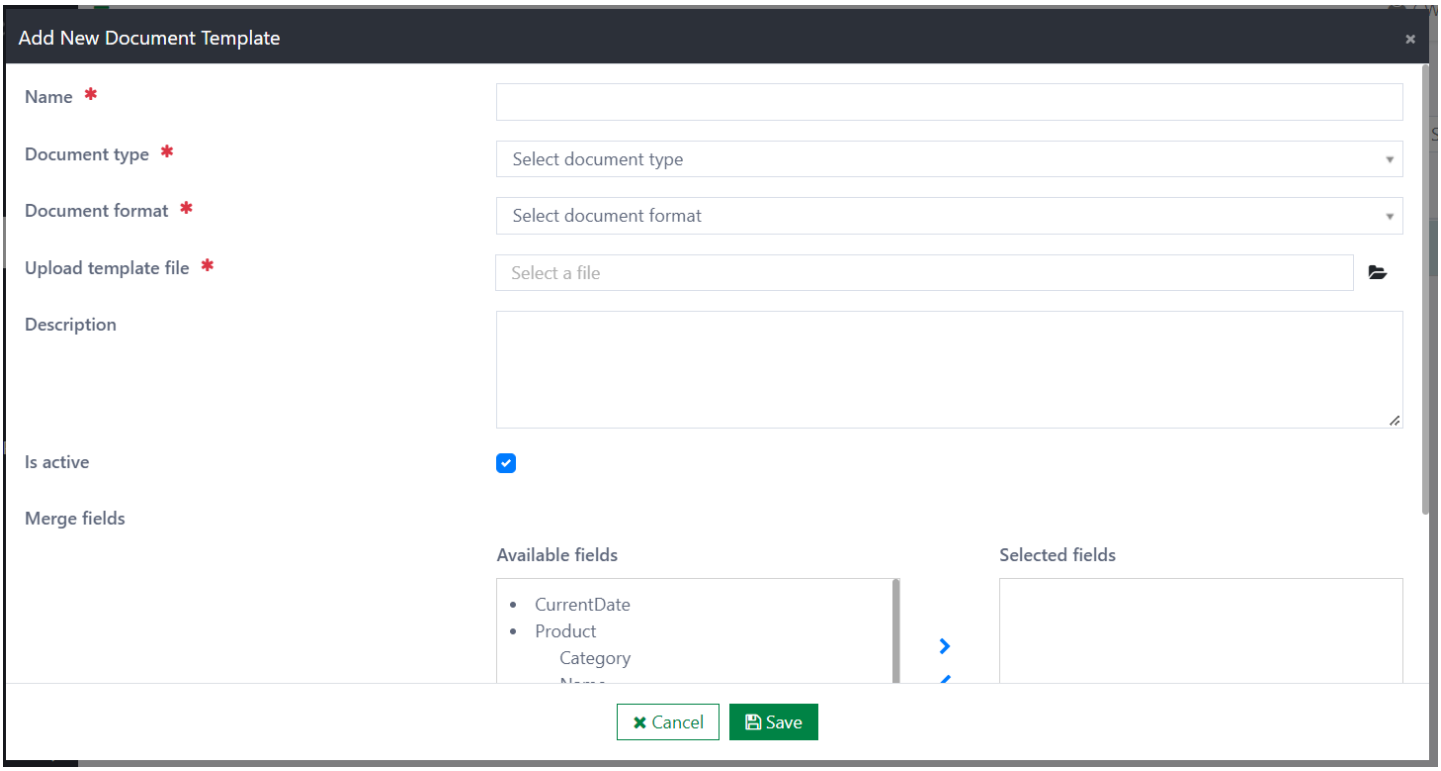
Manage document template

The component provides a page to manage all document templates that are used to generate document files. In this page, you can create/edit/delete the document template



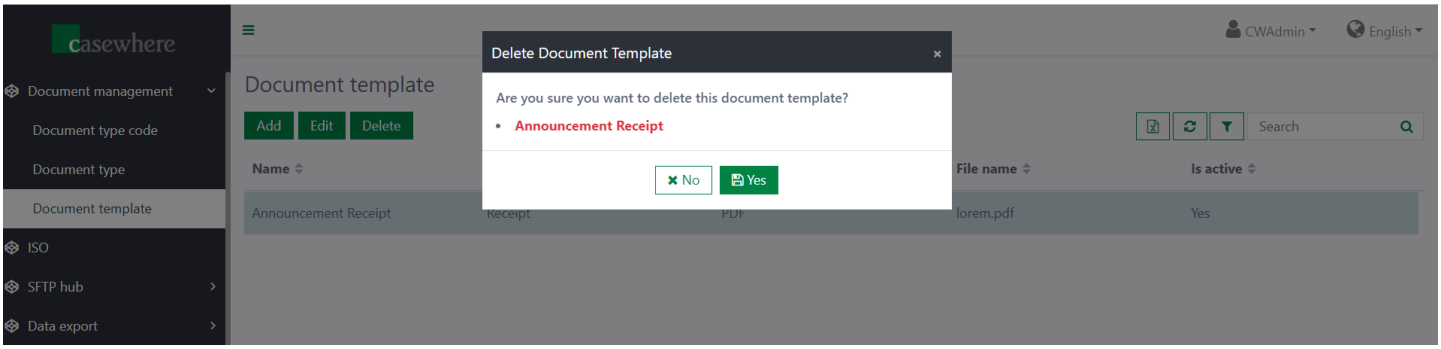
- The following information must be provided when creating/editing document template:
  - **Name**: The name of the document template. This field is required and unique.
  - **Document type**: This field is a dropdown list including all active document types that are getting from the Document types page. This field is required. In editing, the document type cannot be changed.

- **Document format**: This field is used to configure the extension of generated file. This field is required. The supported file extension : DOC, PDF, XML, TXT, HTML. In case of editing document template, the document format cannot be changed.
- **Upload template file**: This field is used to upload the template file that is used to generate the document files. This field is required. Only these file types can be uploaded: DOC, PDF, XML, TXT, HTML
- **Is active**: This field is the status of the document type. The default value is Active. When the document type is active, it will be available in the Document types list of the Document template configuration. In editing, the document type can only be changed to inactive if it is not used for any document template.
- **Merge fields**:
  - Available fields: includes all the supported merge fields. These merge fields are configured in the Merge field setup page. Each document template will have different supported fields.
  - Selected fields: includes all merge fields which are selected to use to populate the data of the upload template file. The template file must contain the corresponding merge fields.



- Delete a document template:

  To delete a document template, click on the **Delete** button. Then, a confirmation dialog will be displayed. If you select the **Yes** button, the document type will be deleted immediately. If you select the **No** button, the deletion will be cancelled. The document template which is in use cannot be deleted. If user tries to delete an in-use document template, error message will appear.



Installation

Requirements

- Casewhere 2.6.10 or later

Configuration

1. Import the product
2. Link the page to your worker sites and configure access control

Dependencies

- None

Releases

1.0.0 - 13/10/2021

Changelog

Component: API key management

Introduction

Most projects often need to expose the API(s) to allow other services to interact with. We need an approach for secure service-to-service communication. The common approach is using the API Keys. Based on this demand, Casewhere provides the standard component to:

- Support generating an API key
- Support managing the API key and its access rights

Feature

To make the API work, we will need to prodide the API key and grant access right(s) to each external service.

In the section **Api rights**, the developer can manage the list of available access rights. The developer can use it to restrict API key access and set up the specific actions that an API key can carry out.

In the section **API keys**, the developer can manage the list API keys. Each API key here is used to identify a spectific external service and authorize the request.

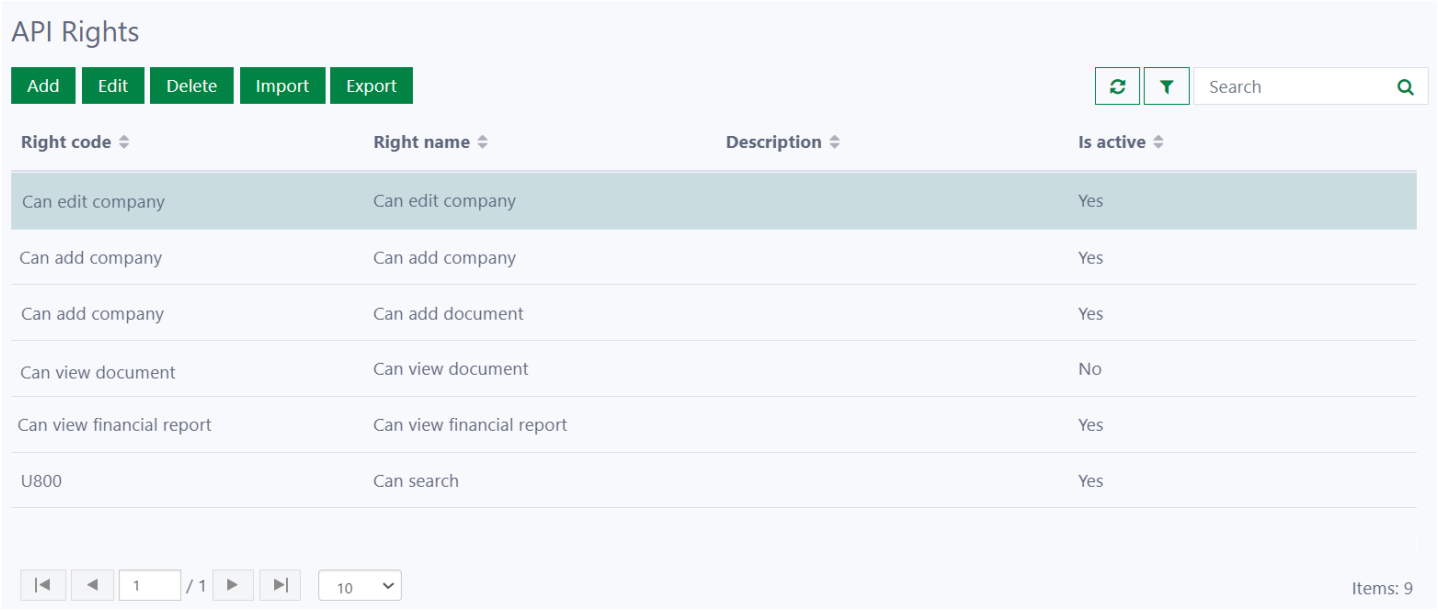
Api rights

Api rights list overview

The API rights are the list of available permissions that the developer can grant to a specific API key. An API key can have one or more rights.

In **Api right** list, you can view all API rights with the information of:

- **API right code**: is the code of the API right
- **API right name**: is the name of the API right
- **Description**: is the description of the API right. It is usually used to describe the information of the API right.
- **Is active**: is the status of the API right. If the API right is not active, then this right is disabled and the external system will no longer have this access right

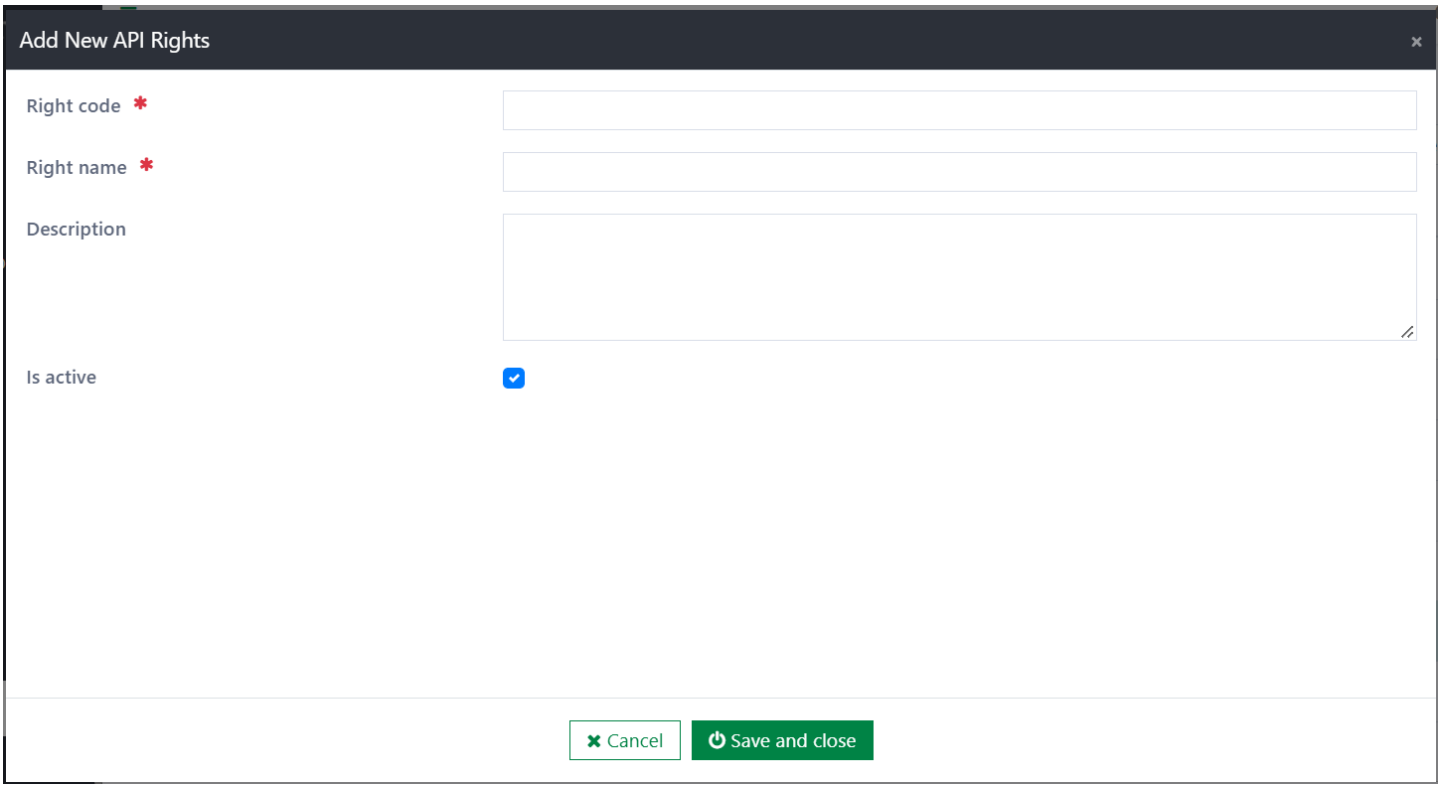


Add a new API right

To add a new **API right**, first, click on the **Add** button Then, in **Add API right** dialog, fill out the fields:

- **Right code:** this field is unique, required. You must input code of the API right
- **Right name**: this field is required.
- **Description**: this field is optional. This field is used to describe the information of the API right.
- **Is active**: If this checkbox is ticked, the API will be active.

After filling the information, click on **Save and close** button to complete adding the API.

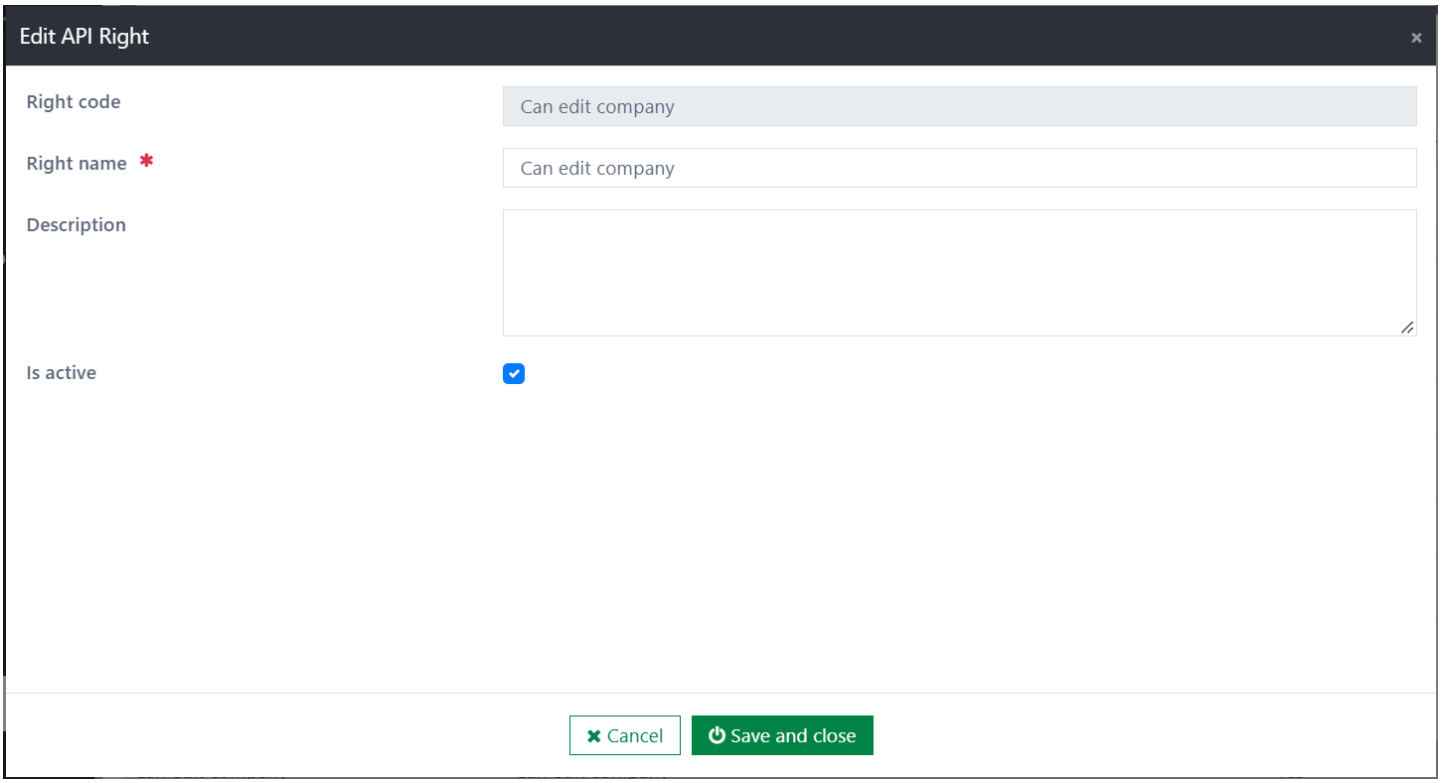


Edit a API right

To edit an **API right**, first, click on the **Edit** button Then, in **Edit API right** dialog, fill out the fields:

- **Right code**: this field is readonly. We only can edit it once at the creation time
- **Right name**: This field is required.
- **Description**: This field is optional. This field is used to describe the information of the API right.
- **Is active**: If this checkbox is ticked, the API right will be active.

After filling the information, click on **Save and close** button to complete adding the API.

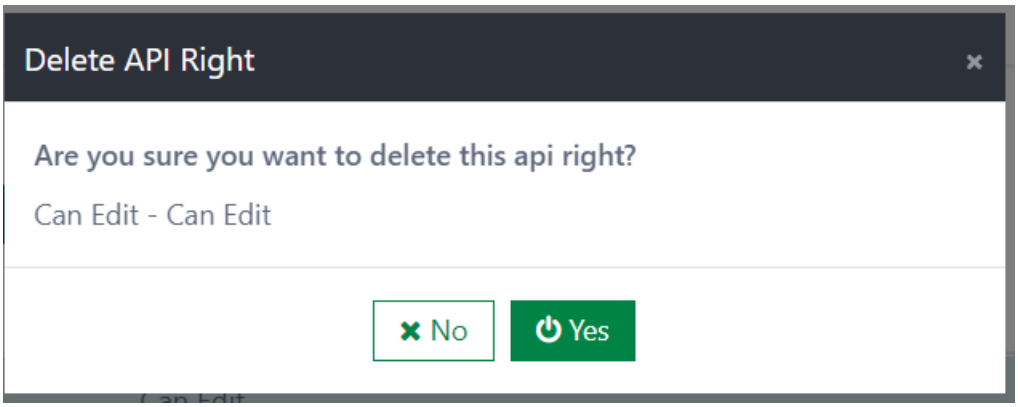


Delete a API right

To delete a API right, first, select a right. Then, click on the **Delete** button

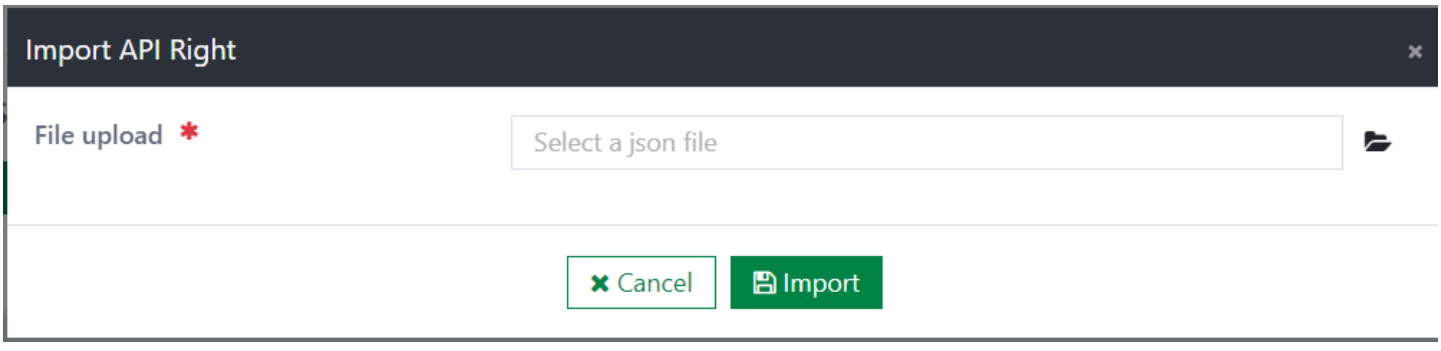
If the right is not used for any **Api key**, a confirmation dialog will be shown. If you select the **Yes** button, the API right will be deleted immediately. If you select the **No** button, the deletion will be cancelled.

If the right was already used in a **Api key**, deletion will be disallowed, and a warning message will be shown.



Import the API right

To import the API right, first, click on the **Import** button to open the **Import API right** dialog



- **Note:** All the API right that exists in the destination environment will be overridden. (Detected by transform API right code)

Then, click on the placeholder of the **File upload** field to open file selection dialog

Next, select a JSON file to import (multiple selection is not supported). After selecting the file, click on the **Import** button to import the api rights from the JSON file. If you do not want to execute the import process, click on **Cancel** button.

Export the API right

To export 1 or several API rights in the list, select the rights, then click on the **Export** button. Then, the JSON file of the selected api rights will be downloaded immediately.

- **Note:** All the API rights that exists in the destination environment will be overridden. (Detected by API right code)

To export all the rights in the list, click on **Export** button. Then, the JSON file of all API rights will be downloaded immediately.

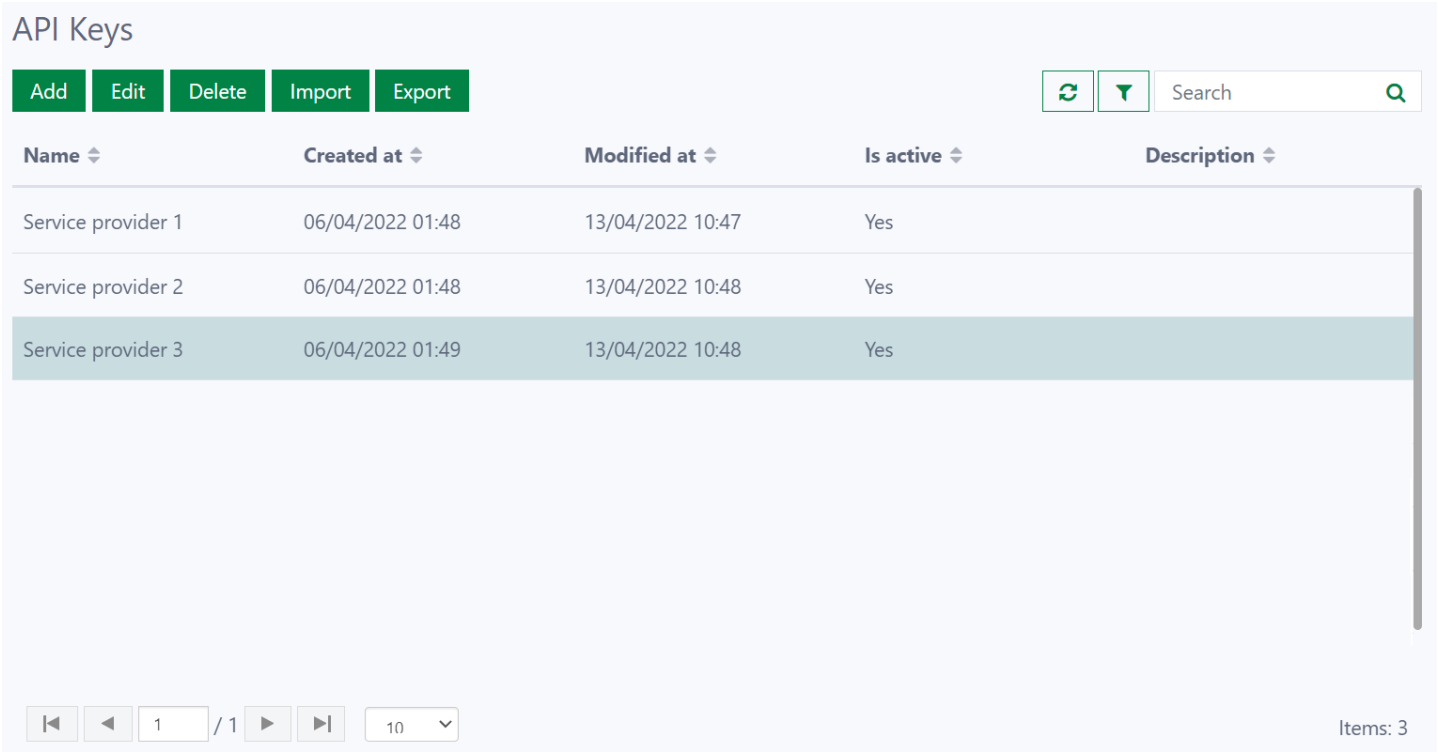
Api keys

Api key list overview

API Keys is a page which manage all of the access keys and their access right(s) for all of the external systems.

In **Api key** list, you can view all API keys with the information of:

- **API name**: is the name of the API key
- **Created at**: is the creation date of this API key
- **Modified at**: is the last modified date of this API key
- **Is active**: is status of the API.
- **Description**: is the description of the API key. It is usually used to describe the information of the API key.

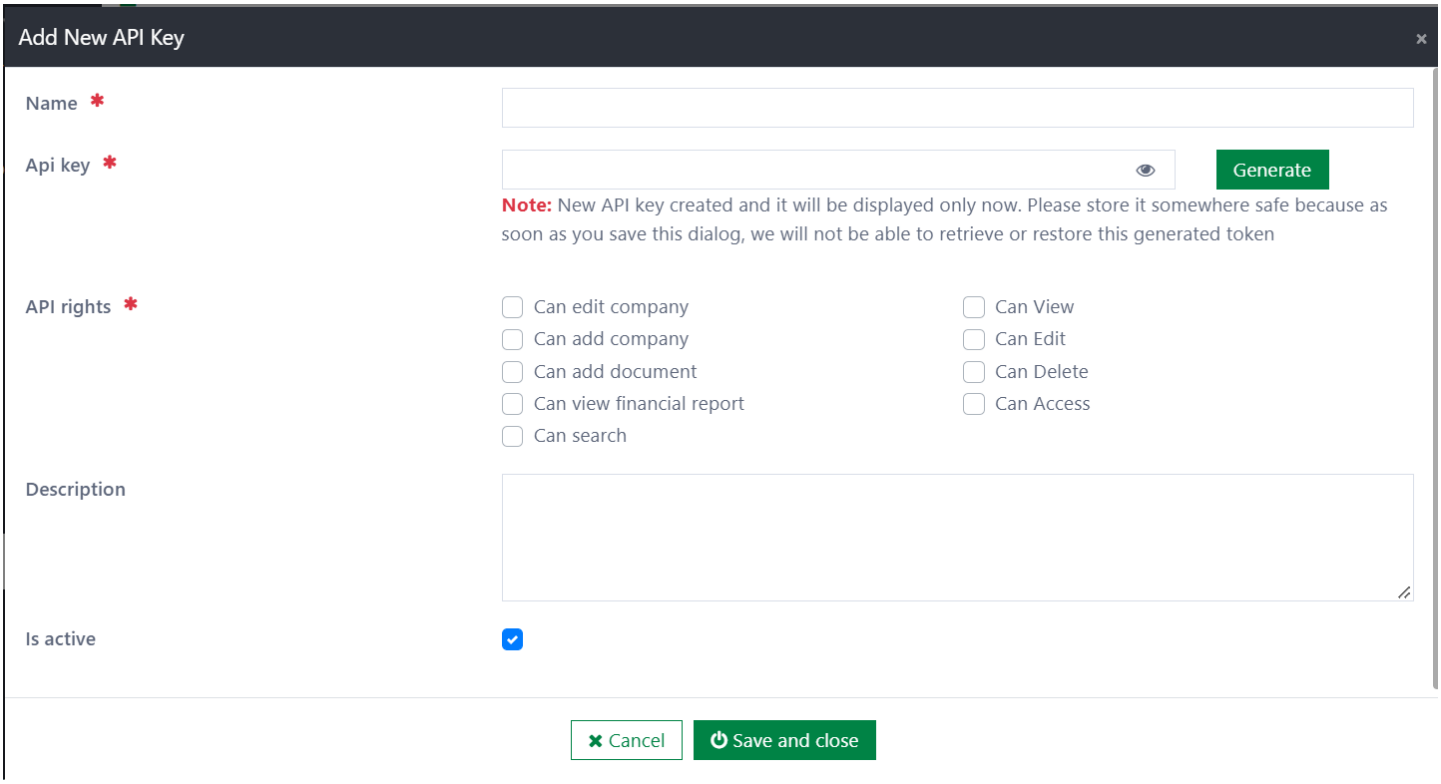


Add a API key

To add a new **API key**, first, click on the **Add** button Then, in **Add API key** dialog, fill out the fields:

- **API name**: this field is unique and required. You must input name of the API
- **API key**: this field is required. We can input or choose to generate a random API key, since the API key itself is an identity by which to identify the external system or the user, it needs to be unique, random and non-guessable. API keys that are generated will also use alphanumeric and special characters. The user can enter a desirable key or choose to **Generate** the random value.
- **API rights**: this field is required. it is used to restrict API Key access and choose specific actions that an API key can carry out.

- **Description**: this field is optional. This field is used to describe the information of the API right.
- **Is active**: if this checkbox is ticked, the API will be active. Otherwise, it becomes being inactive and the system will reject accessing through this API key

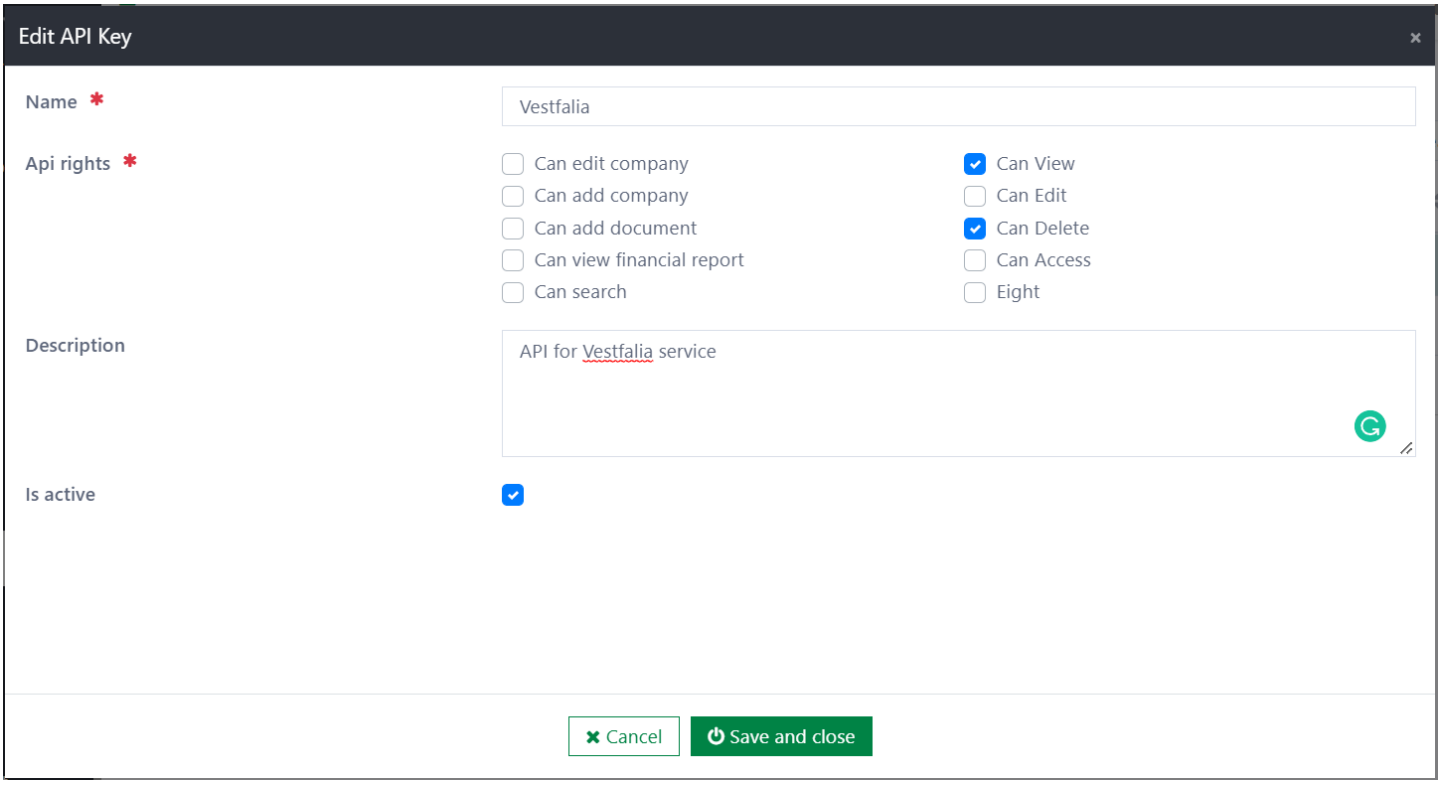


After filling the information, click on **Save and close** button to complete adding the API.

Edit an API

To edit a API, first, click on **Edit** button to open **Edit API** dialog

In the **Edit API** dialog, you can edit **API name**, **API key**, **API rights**,**Description**,**Is active**



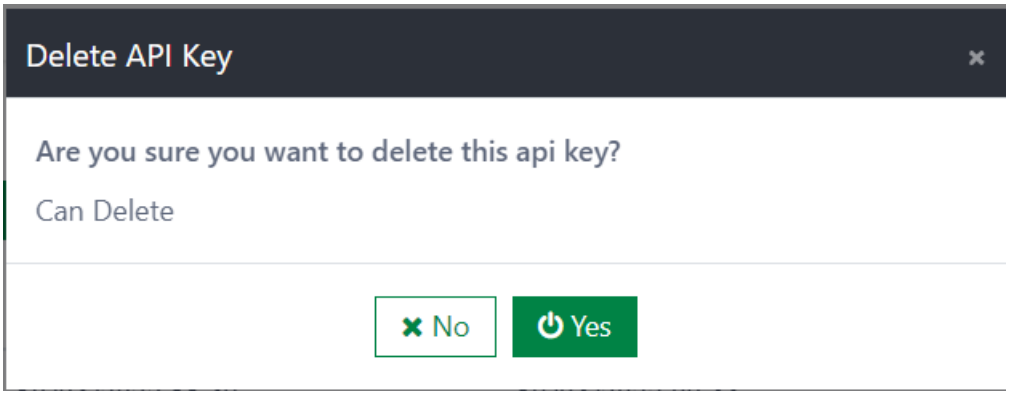
After updating the information, click on **Save and close** button to save the changes. Or you can click on **Cancel** button to discard

the changes.

Delete a API key

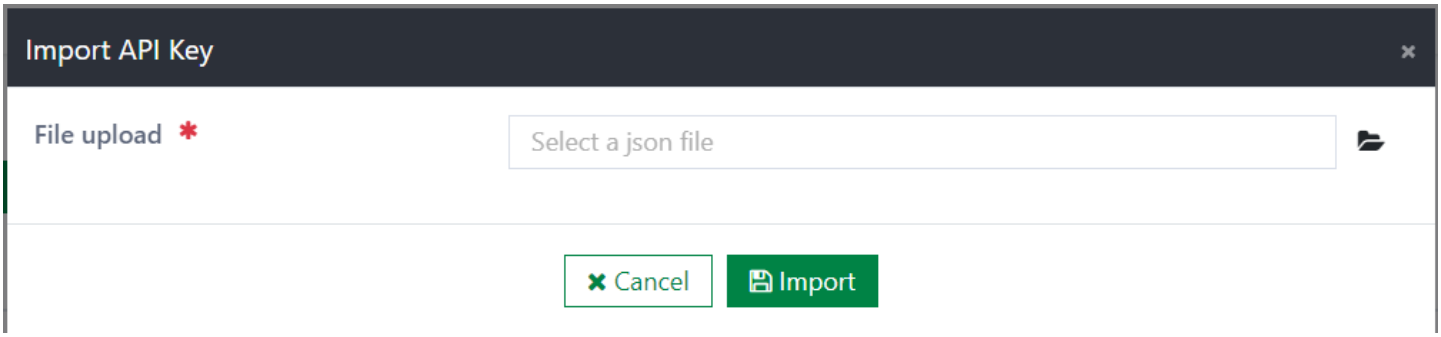
To delete a **API key**, first, click on the **Delete** button

Then, a confirmation dialog will be displayed. If you select the **Yes** button, the API key will be deleted immediately. If you select the **No** button, the deletion will be cancelled.



Import the API key

To import the API keys, first, click on the **Import** button to open the **Import Data collection Configs** dialog



- **Note:** All the API keys that exist in the destination environment will be overridden but the generated key will be remained (Detected by API name)

Then, click on the placeholder of the **File upload** field to open file selection dialog

Next, select a JSON file to import (multiple selection is not supported). After selecting the file, click on the **Import** button to import the API rights from the JSON file. If you do not want to execute the import process, click on **Cancel** button.

Export the API key

To export one or several API keys in the list, select the API key(s), then click on the **Export** button. Then, the JSON file of the selected API keys will be downloaded immediately.

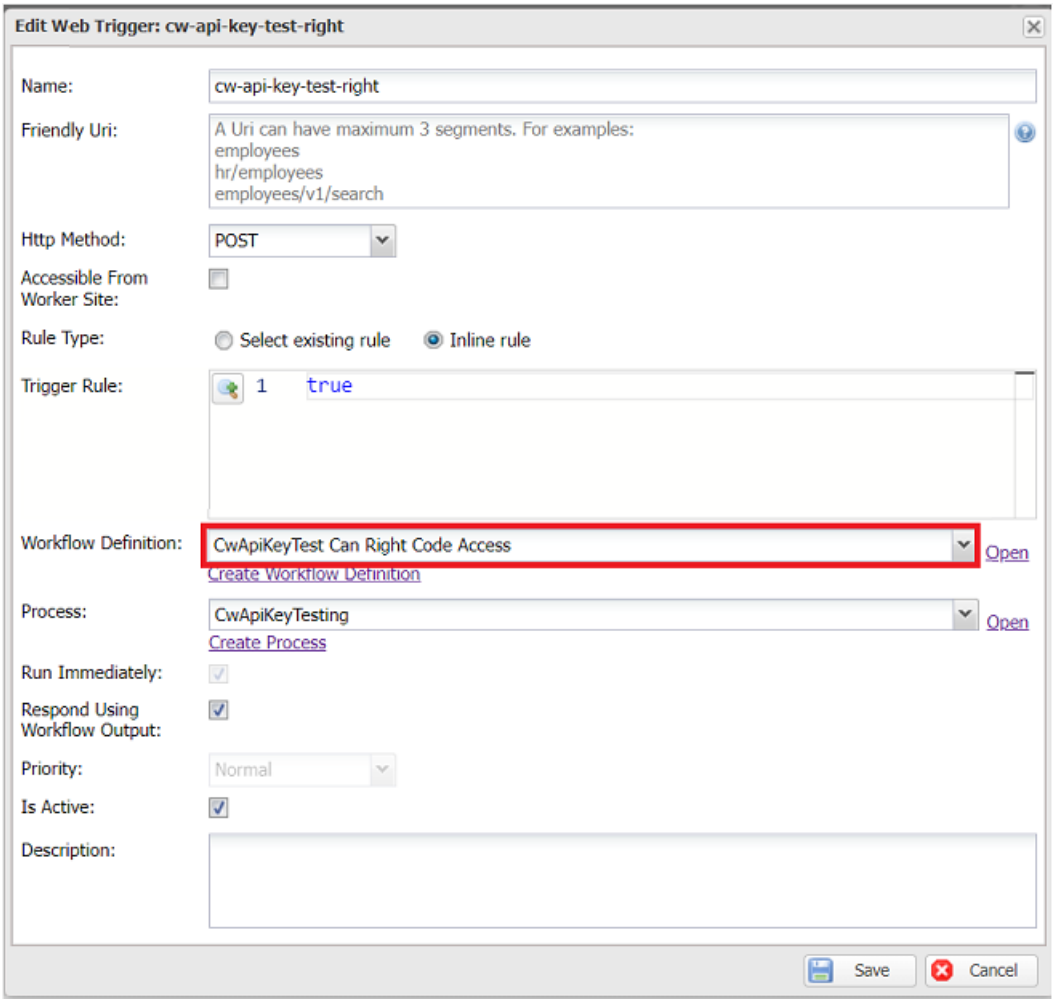
To export all API keys in the list, click on **Export** button. Then, the JSON file of all API keys will be downloaded immediately.

Extension rules

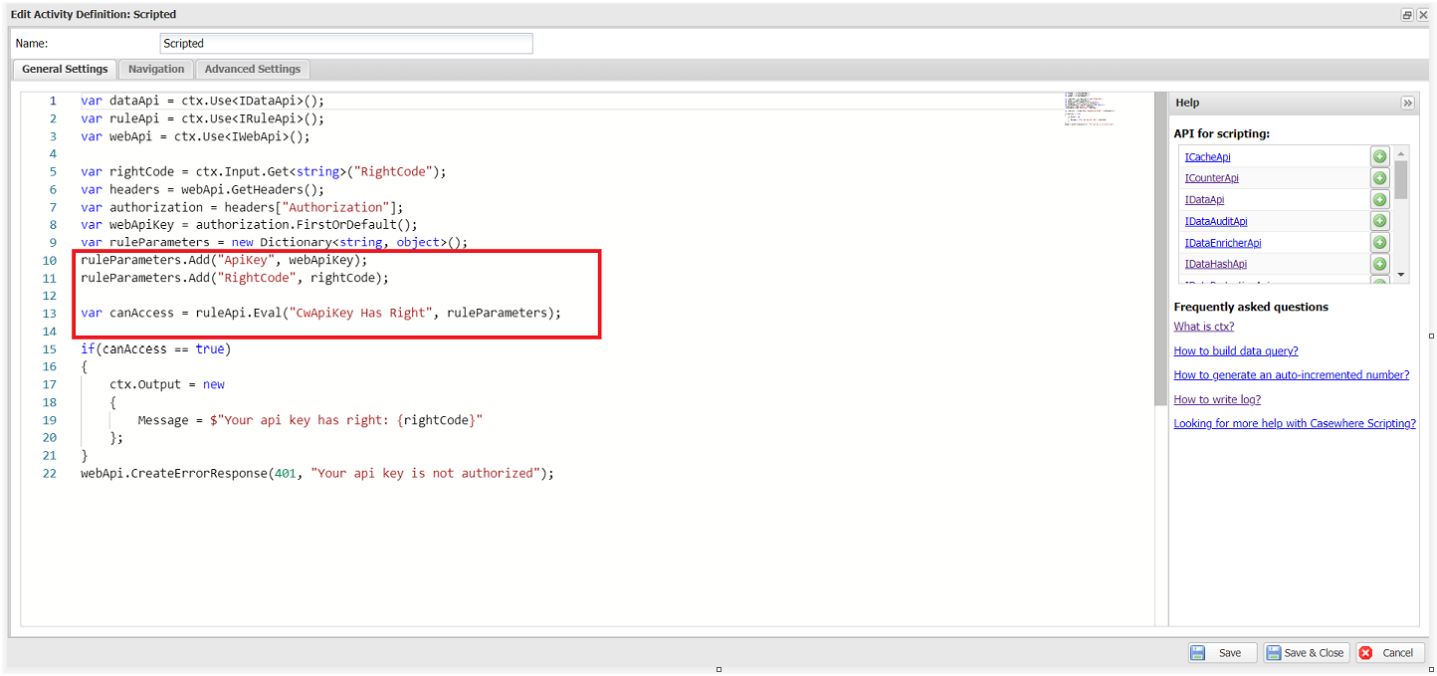
Casewhere API management component provide the rule below to support the authentication and authorization process:

- **Has Right**: Checking if the **API key** has a specific **API right**
  - Input: API key, API right
  - Output: true/false

This is a sample. A web trigger invokes a workflow that calls the rule `Has Right`



Scripted activity inside workflow calls rule `Has Right`



Installation

Requirements

- Casewhere 2.6.10 or later

Configuration

- Import the product

- Configure the API right and API keys
- When implementing the web trigger, the developer needs to implement the code to check if the API key is allowed to access to the web trigger or not by querying the information of the API key in the database. Casewhere also provides a common rule mentioned in the section **Extension rules** to support this checking.

Releases

1.0.0 - 24/02/2022

Changelog

- API key management

Download (login required): API Key Management v1.0.0

Component: Global settings

Introduction

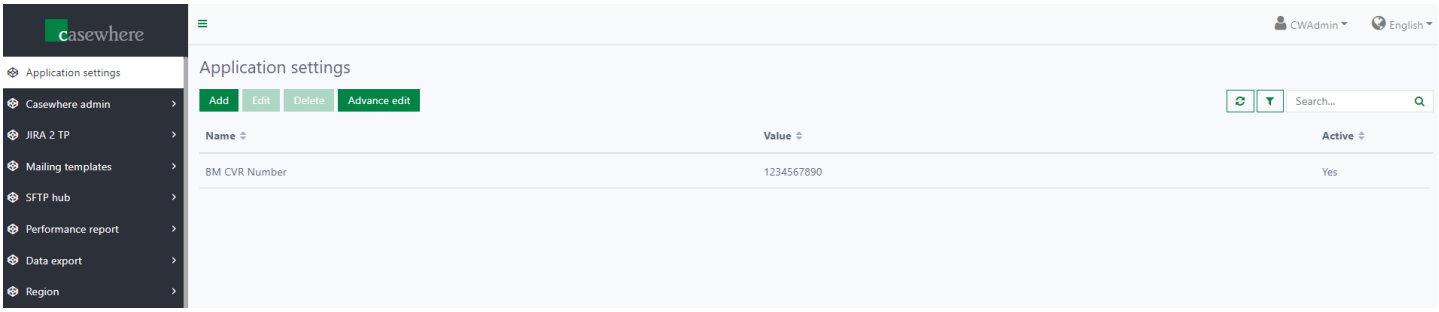
Most solutions need to manage global variables that can be used as parameters in server rules, workflows, etc.
These parameters are deployment specific and should be managed by end users.

Features

- List of settings, the list settings must have the following essential functions:
  - Add
  - Edit
  - Delete
  - Advanced edit
- The component should offer some rules as functions to retrieve a setting by name
- The value will be always encrypted in the data store

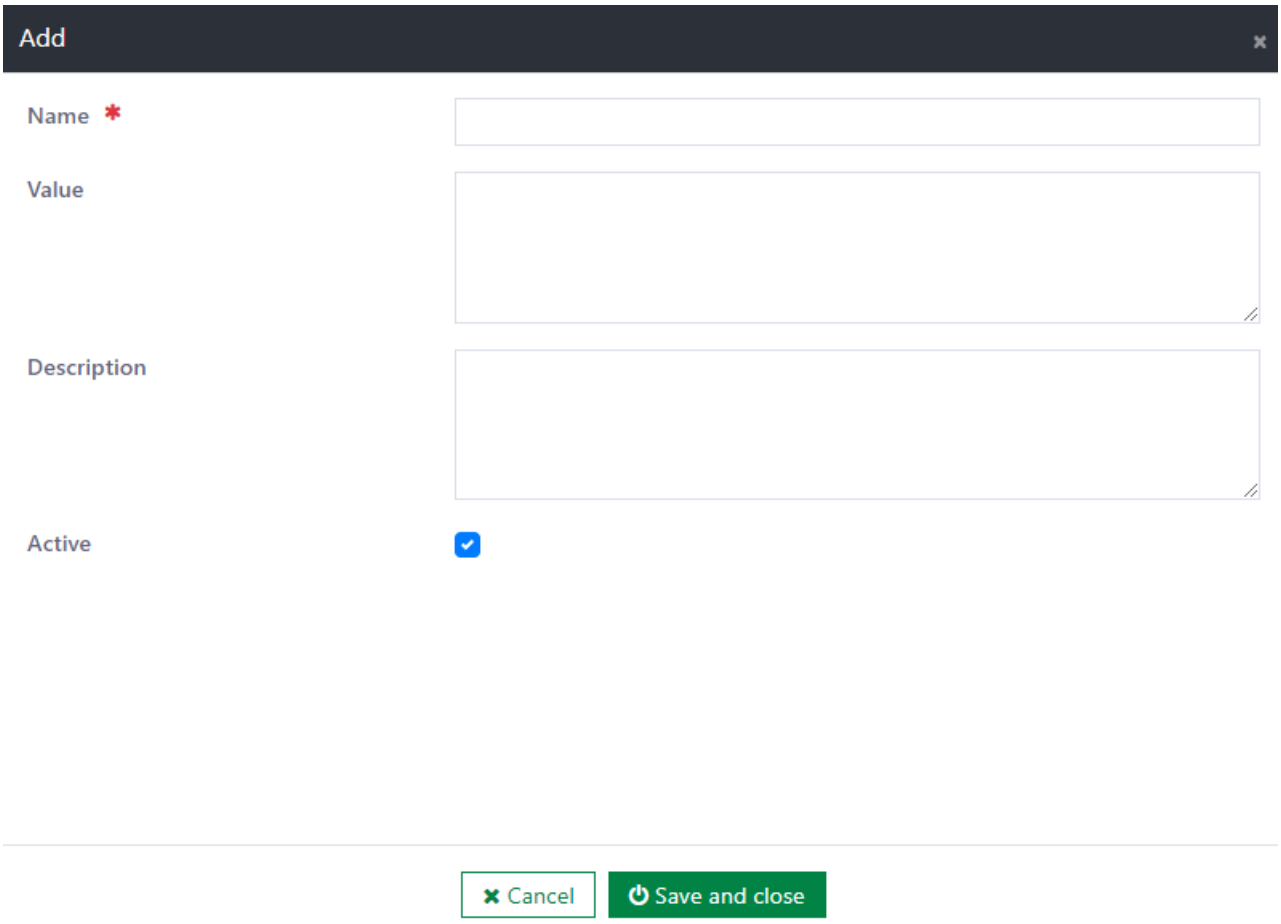
Manage settings

The component provides a page to manage all global settings that are used in the application.
In this page, you can add/edit/delete the setting

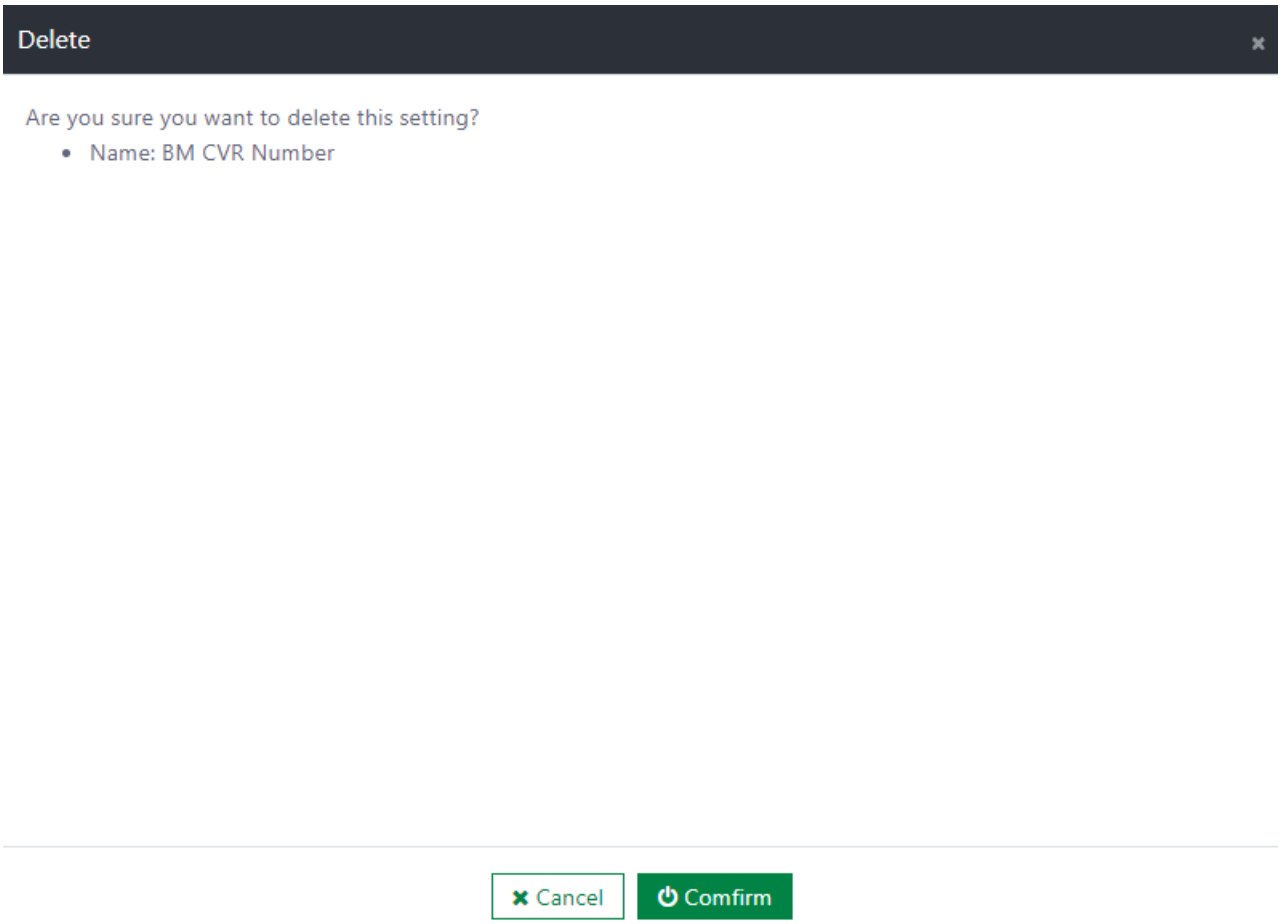


The following information can be provided when creating/editing setting:

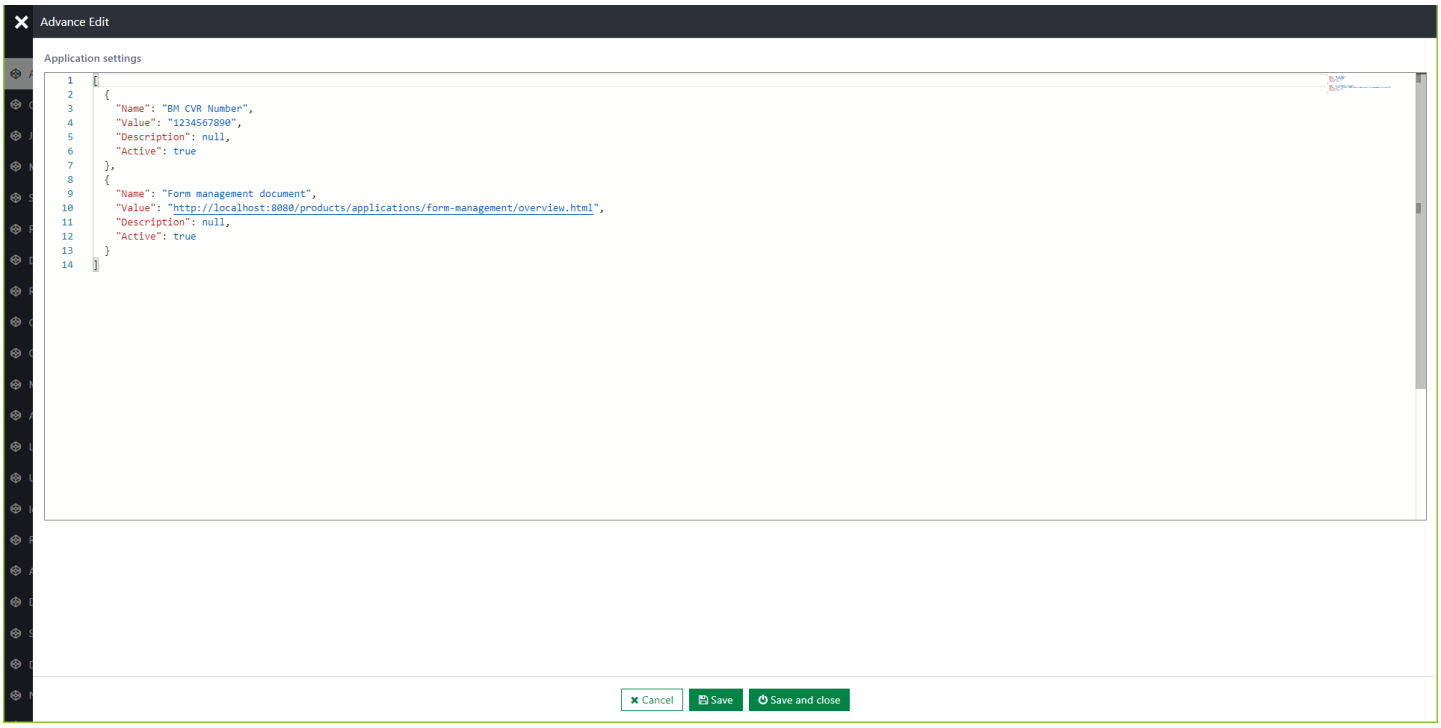
- Name: The name of the setting
  - This field is required and unique
- Value: The value of the setting
  - This field is optional
- Description: This field is used to describe the other information of the setting
  - This field is optional
- Active: This field is the status of the setting
  - The default value is Active



To delete a setting, click on the Delete button.
Then, a confirmation dialog will be displayed.
If you select the Confirm button, the setting will be deleted immediately.
If you select the Cancel button, the deletion will be cancelled.



Advance edit used to add/edit/delete multiple settings at the same time.
You can add new json object to create a new one, edit value to update the existing one.
Delete a object or delete a whole json array to remove all settings.



Using settings

The component provides a set of server rules to retrieve an application setting value by its own name.
All rules required an input object that contains name of the setting.

- CwGs Get By Name

    - Output: a string value

```
var parameters = new Dictionary<string, object>();
parameters.Add("Name", "BM CVR Number");
var ruleApi = ctx.Use<IRuleApi>();
var result = ruleApi.Eval<string>("CwGs Get By Name", parameters);
```

- CwGs Get Integer By Name
    - Output: an int value

```
var parameters = new Dictionary<string, object>();
parameters.Add("Name", "Example setting");
var ruleApi = ctx.Use<IRuleApi>();
var result = ruleApi.Eval<int>("CwGs Get Integer By Name", parameters);
```

- CwGs Get DateTime By Name
    - Output: a DateTime value

```
var parameters = new Dictionary<string, object>();
parameters.Add("Name", "Example setting");
var ruleApi = ctx.Use<IRuleApi>();
var result = ruleApi.Eval<DateTime>("CwGs Get DateTime By Name", parameters);
```

- CwGs Get Decimal Numer By Name
    - Output: a decimal value

```
var parameters = new Dictionary<string, object>();
parameters.Add("Name", "Example setting");
var ruleApi = ctx.Use<IRuleApi>();
var result = ruleApi.Eval<decimal>("CwGs Get Decimal By Name", parameters);
```

- CwGs Get Boolean By Name
    - Output: a bool value

```
var parameters = new Dictionary<string, object>();
parameters.Add("Name", "Example setting");
var ruleApi = ctx.Use<IRuleApi>();
var result = ruleApi.Eval("CwGs Get Boolean By Name", parameters);
```

- CwGs Get Long By Name
    - Output: a long value

```
var parameters = new Dictionary<string, object>();
parameters.Add("Name", "Example setting");
var ruleApi = ctx.Use<IRuleApi>();
var result = ruleApi.Eval<long>("CwGs Get Long By Name", parameters);
```

Installation

Requirements

- Casewhere 2.6.10 or later

Configuration

1. Import the product
2. Link the page to your worker sites and configure access control

Dependencies

- None

Releases

1.0.0

Changelog

- List of settings
  - Add
  - Edit
  - Delete
  - Advance edit
- Server rules to retrieve setting value by name

Road map

- Client rule to get the application setting from the frontend
- Import/export as a standard component
- Show/hide setting values
- Apply caching to avoid making too many requests to the database on retrieving application settings

Component: Self-service theme

Introduction

To provide a standard theme which can be used for every self-service worker sites in the future, the self-service theme has been developed. It is well referred to the https://designsystem.dk/design/.

In this design system, it defines various things, includes:

- Colors
- Typography
- Corners, lines, shadows
- Icons
- Logos
- Data visualization and infographics
- Layout
- Design resources
- Design check
- Availability

Components

The theme has a set of components with predefined styles, which you can use directly in your HTML.

Below is the set of supported components:

- Accordions
- Badges
- Alerts
- Breadcrumbs
- Cards
- Cookie notification
- Date indication
- Date picker
- Details
- Dropdown menu
- Tabs
- Error messages
- Error summary
- Fields
- Footers
- Form
- Function link
- Headers
- Buttons
- Modal window
- Notification (Toast)
- Overflow menu
- Pagination
- Radio button
- Page navigation
- Skip link
- Spinner

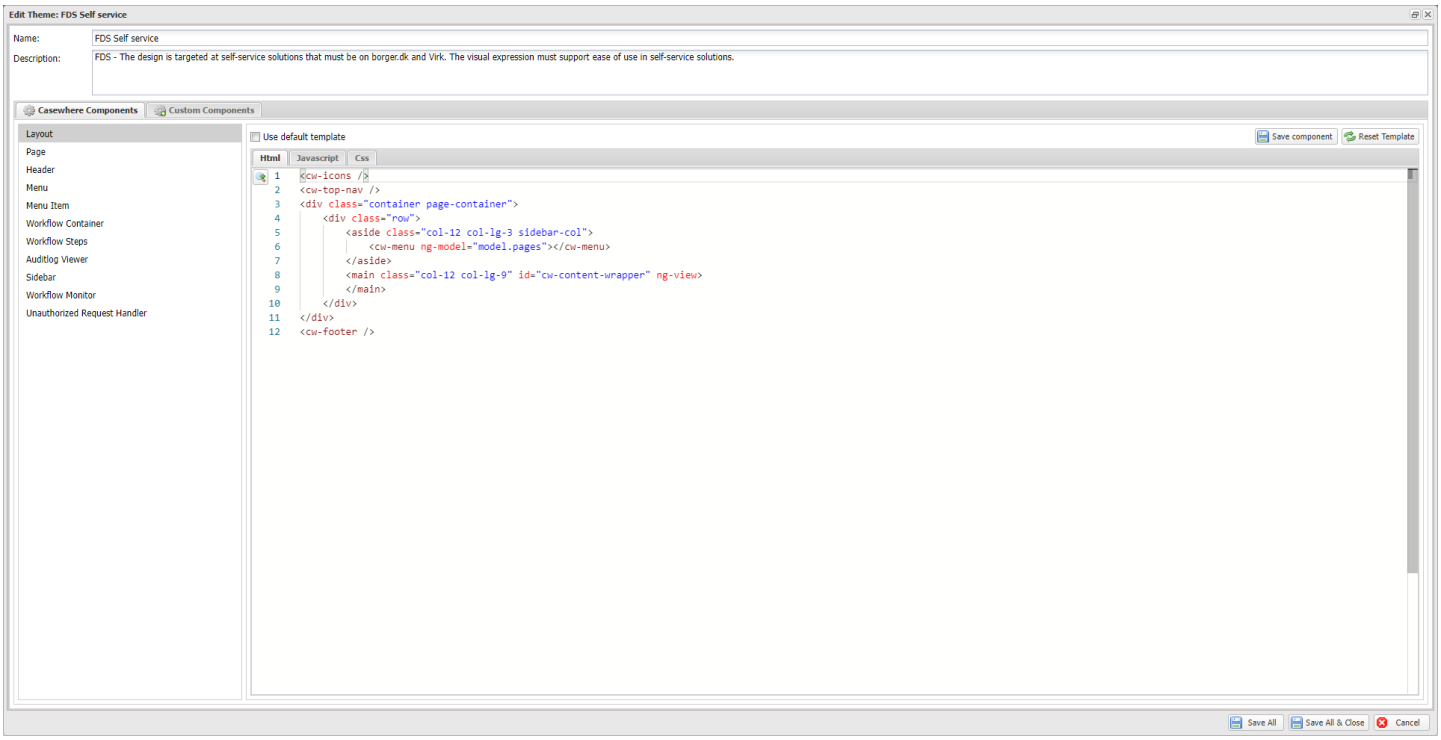
- Language selector
- Structured lists
- Search field
- Back link
- Tables
- Tags
- Text area
- Check box
- Toggle switch
- Tool tip
- Step indicators
- Attach file (Upload)

Note: for some components, it requires Javascript to function.

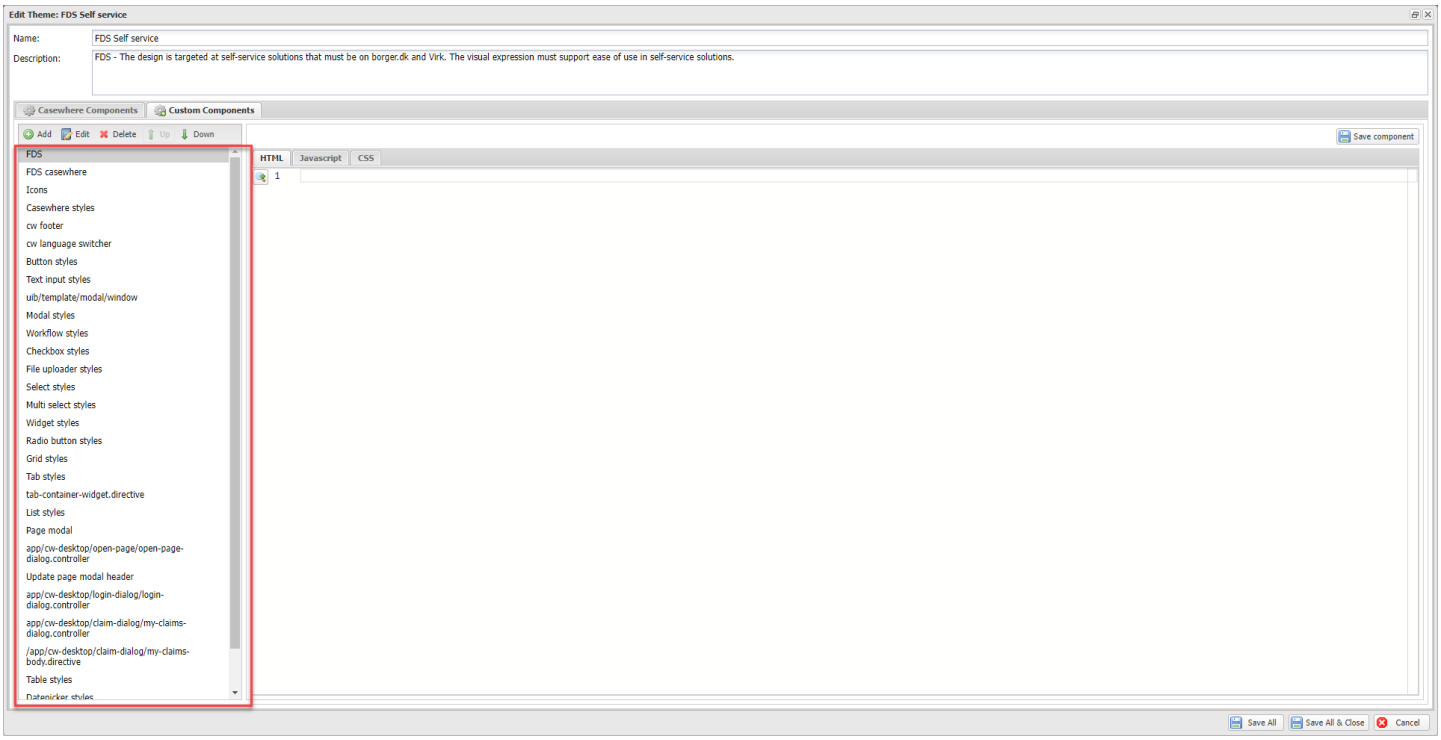
To get more information about how to use these components, you can visit to the link https://designsystem.dk/kode/komponenter/.

Self-service theme in CW

In this theme, we styled all the built-in components to comply with the common design system.



Apart from that, we also apply the look and feel to all the supported components in CW.



Installation

Requirements

- Casewhere 2.6.10 or later

Configuration

- Import product: **DK Common Design System Theme**
- Create a worker site, select **FDS Self service** as its theme.

Releases

1.0.0 - 03/17/2022

Download (login required): DK Common Design System Theme v1.0.0

Roadmap

- Apply L&F for CMS widgets
- Handle more UI cases

Integration: Service Platform CPR Update

Subscribe to CPR change events to trigger your custom event handler or enrich your local data.

Features

Configure the event handler

Configure the event handler - a workflow to receive CPR events.

Configure subscription

There are two methods: pull and push.

Event monitoring

The component provides an administration tool for your end-users to view all events received, their statuses.

Installation

Requirements

- Casewhere 2.6.0 or later

Configuration

- Import the product
- Configure the plugin settings
- Grant permissions for admin users

Releases

1.0.0 - 01/09/2020

Changelog

- Configure the event handler
- Configure subscription
- Event monitoring

Download (login required): CPR Update v1.0.0

Integration: Service Platform Print Service

Introduction

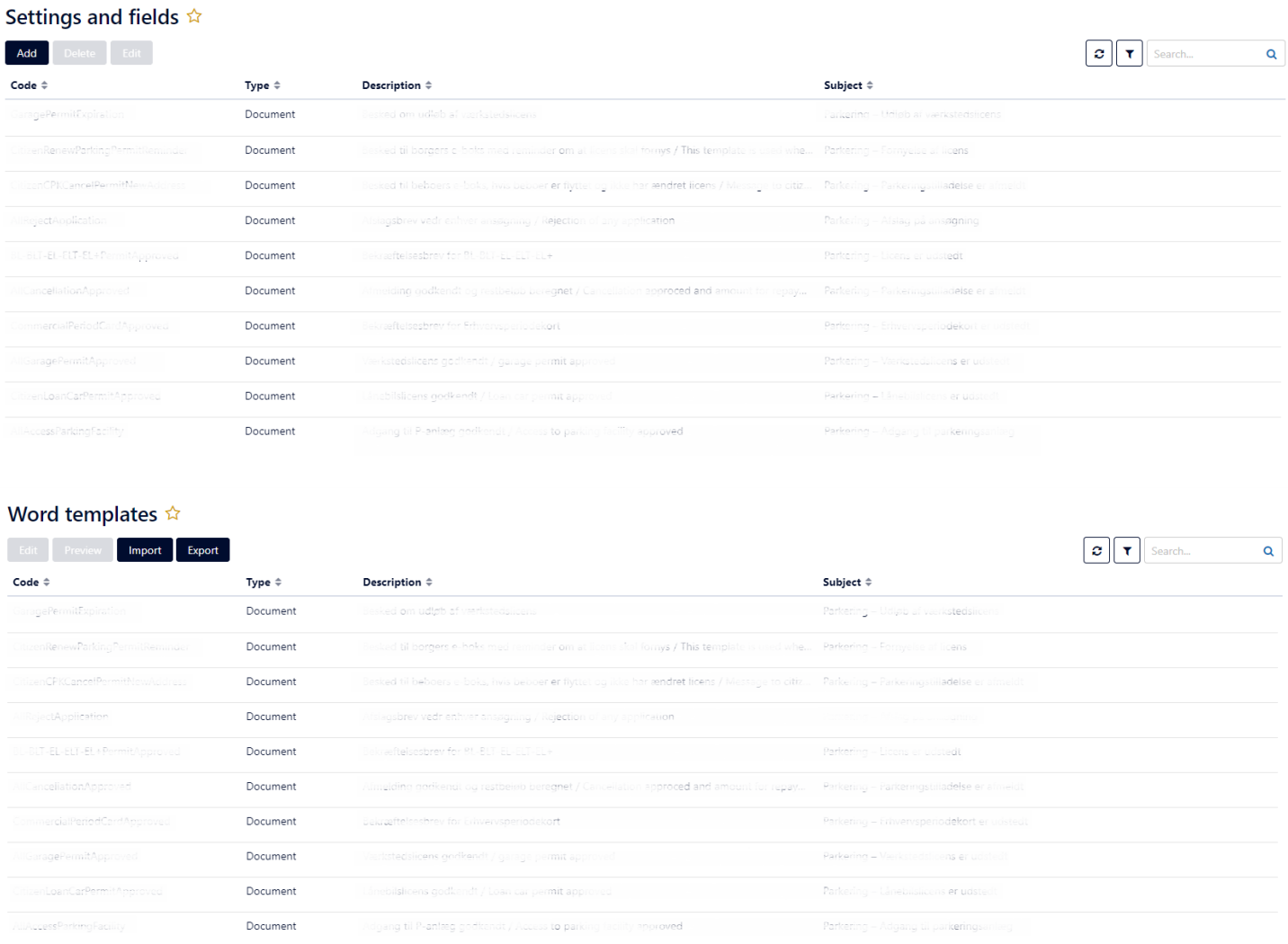
The integration makes it possible for municipal professional solutions to send letters to citizens and companies, either via the Danish Digitization Agency's Digital Post solution (Digitaliseringsstyrelsens Digital Post-l�sning) or as a physical letter via the municipality's Remote Print Solution (Fjernprintl�sning). The integration also provides the opportunity to send letters between authorities.

This integration provides a method to post a document to the Service Platform via the service SF1600. More information and the integration document can be found here: https://digitaliseringskataloget.dk/integration/sf1600

Features

Manage letter templates

You can define letter templates with merge fields. Letters are then created based on those defined templates. The merge fields in the template will be replaced with context-specific data at runtime to create final letters before posting. The data merger is done by the plugin Aspose Word. The letter templates are Microsoft Word's documents. After templates are composed, they can be uploaded to the Casewhere via the configuration page.



Post a document

Post a document to the Service Platform.

Method Name: PostDocument

Input

- Channel (required): The channel to post onto.
  - "D" for DigitalPost,
  - "F" for physical mail
  - ~~"A" for automatic on SP~~ (no template implemented)
  - ~~"P" for automatic on Remove print~~ (no template implemented)
  - ~~"S" for NemSMS~~ (no template implemented)
- Priority (required): The priority of the posting
  - "D" for Direct
  - "M" for Mass Shipping
- Recipient (required): Information of the recipient
- Uuid (required): Document's UUID
- Title (required): Title of the document
- Content (required): Content of the document represented as base64 string
- LetterDate (required): The date that the letter is created
- MessageIdentifierTransaction (required): Message ID transaction

Output

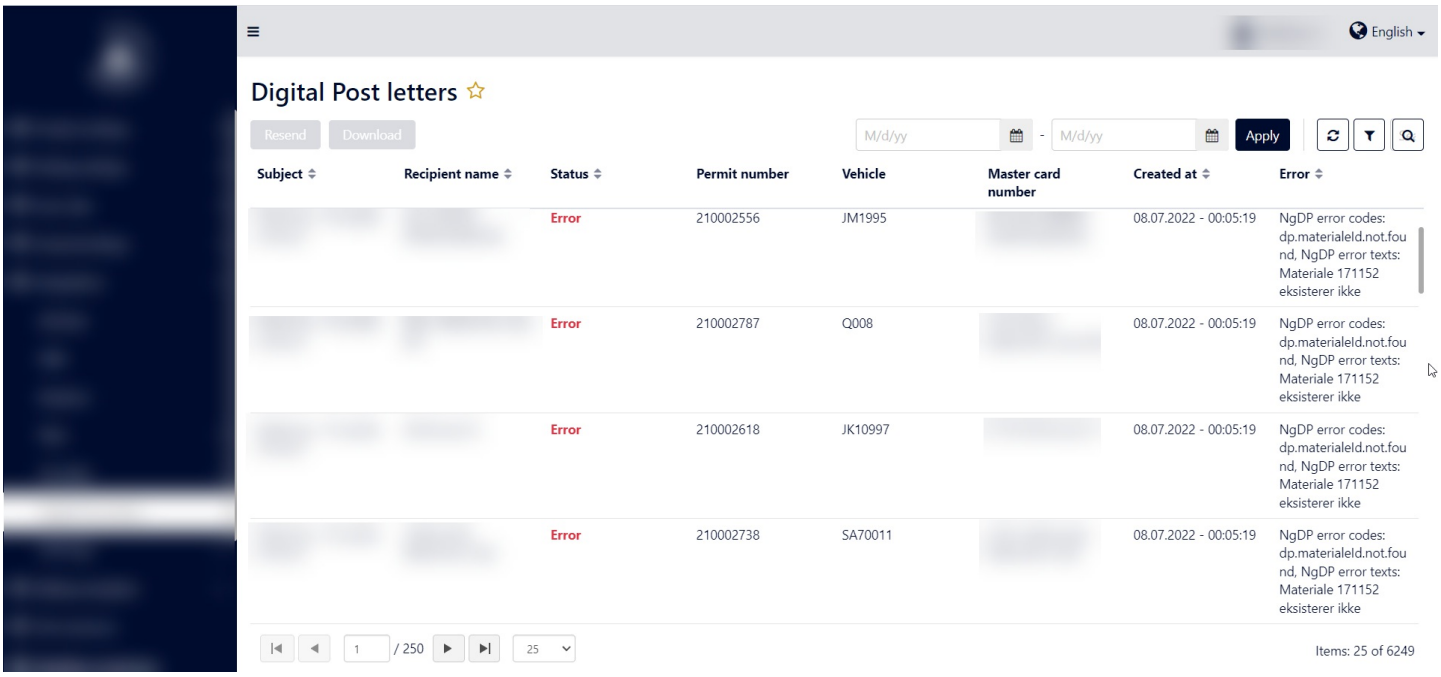
- Data: A boolean value indicates the posting is succeeded or not
- Error: Object that stores error information if at least one error occurs

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var data = new
{
  Channel = "D",
  Priority = "D",
  Recipient = new
  {
    Identifier = "5401060221",
    IdentifierType = "Cpr", // Cpr or Cvr
    Name = "Sherlock Holmes",
    FloorNumber = "2",
    HouseNumber = "221B",
    StreetName = "Baker Street",
    ZipCode = "NW1 6XE",
  },
  Uuid = "b7fcb343-bfeb-4c8a-9394-88533728a563",
  Title = "The Hound of the Baskervilles",
  Content =
"TXIuIFNoZXJsb2NrIEhvbG1lcywgd2hvIHdhcyB1c3VhbGx5IHZlcnkgbGF0ZSBpbiB0aGUgbW9ybmluZ3MsIHNhdmUgdXBvbiB0aG9zZSBub3QgaW5mcmVxdWVudCBvY2Nhc2lvbnMgd2hlbiBoZSB3YXMgdXAgYWxsIG5pZ2h0LCB3YXMgc2VhdGVkIGF0IHRoZSBicmVha2Zhc3QgdGFibGUu",
  LetterDate = DateTime.Parse("1901-08-15T17:25:27.159Z"),
  MessageIdentifierTransaction = "00000000000000000000020220404144320"
};
var parameters = new Dictionary<string, object>
{
  { "Data", JsonConvert.SerializeObject(data) }
};
var pluginResult = pluginApi.Invoke("CwDigitalPost", "PostDocument", parameters);
var output = pluginResult.GetObject("Data");
```

Monitor letter sending

The component provides an administration tool for your end-users to view all letters created, their delivery statuses.



Installation

Requirements

Casewhere v2.6 or later

Configuration

To make the integration work, the configuration as following must be done:

Configure plugin

The plugin **CwDigitalPost** must be configured properly. There are two modes: Stub and Real. Stub mode is used to test internally when the Service Platform is down, Real mode is used to test with the Service Platform.

Here is the configuration for Stub mode:

```
{
    "Mode" : "Stub",
    "StubDataApiEndPoint" : "[Stub endpoint]", // Endpoint to stub data provider.
    "StubDataApiKey": "123456", // API key for authentication.
}
```

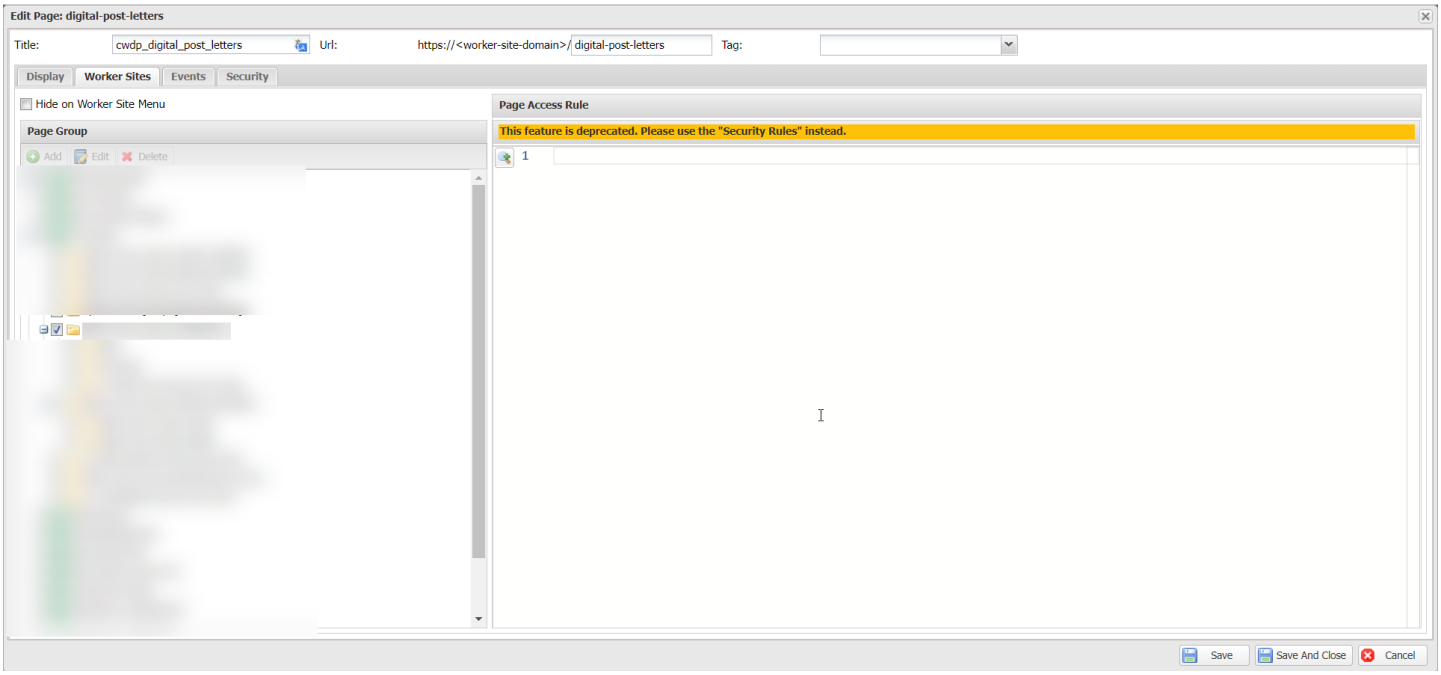
Here is the configuration for Real mode:

```
{
    "Mode": "Real",
    "IsStrictSecurity": false,
    "MinimumFreshness": "0:15:0",
    "ClientCertificateStoreName": "My",
    "ClientCertificateStoreLocation": "CurrentUser",
    "ClientCertificateThumbprint": "[Certificate thumbprint]",
    "SigningClientCertificateStoreName": "TrustedPeople",
    "SigningClientCertificateStoreLocation": "CurrentUser",
    "SigningClientCertificateThumbprint": "[Certificate thumbprint]",
    "SigningClientCertificateAlias": "[Certificate alias]",
    "DigitalPostServiceEntityId": "http://print.serviceplatformen.dk/service/forsendelse/2",
    "DigitalPostEndpointUrl": "https://exttest.serviceplatformen.dk/service/Print/Print/2",
    "StsIssuer" : "[STS issuer URL]",
    "StsEndpoint" : "[STS endpoint]",
    "StsCertificateStoreName": "TrustedPeople",
    "StsCertificateStoreLocation": "CurrentUser",
    "StsCertificateThumbprint": "[Certificate thumbprint]",
    "StsCertificateAlias": "[Certificate alias]",
    "Cvr": "[CVR of the company that is registered with the Service Platform]",
    "SendingIdentifier": "123456",
    "MaterialId": "123456",
    "AccountingInfo": "Sender",
    "OnBehalfOfUser": "Sender"
}
```

Configure pages

Add the configuration pages to the worker site, it is usually the config site.

- Digital Post letters: **digital-post-letters**
- Digital Post setting: **eboks_settings**
- Digital Post Templates: **e-Boks-templates**



Dependencies

It depends on the plugin Aspose Word

Releases

1.0.0 - 22/09/2021

Changelog

- Manage letter templates
- PostDocument
- Monitor letter sending

Download (login required): Service Platform Print Service plugin v1.0.0

Roadmap

- Send multiple documents
- Integrate with Document Template component
- Friendly UI to configure Service Platform Print service

Integration: Navision

Many Casewhere solutions need to sync economic transactions to a Navision system for accounting. Understanding the need, Casewhere offers a standard solution to automate the process. Instead of being slow down by the mundane technical challenges, developers can now just focus on the business side of things.

Features

Configure transaction templates

You can decide what and how your data will be sent to Navision by designing the transaction templates.

Configure SFTP connection

The communication between Casewhere and Navision is exchanging files through SFTP.

Configure the sync schedule

The process is automated, and you can configure when it will be triggered. For example, Sunday midnight, weekly.

Monitor transactions

Transactions will be sent to Navision in batches. The component provides an administration tool for your end-users to manage the Navision batches, their statuses.

Installation

Requirements

- Casewhere 2.6.0 or later.

Configuration

1. Import the product
2. Configure the SFTP plugin settings
3. Link the administration page to your worker sites and configure access control

Releases

1.0.0 - 01/09/2020

Changelog

- Configure transaction templates
- Configure the sync schedule
- Configure SFTP connection
- Monitor transactions

Download (login required): Navision v1.0.0

Plugin: Aspose Word

Introduction

This plugin provides the most complete and efficient set of Word document processing solution without any Microsoft Word dependencies.

Features

Create PDF from Word template

Merge data to a Word document template and return a PDF as result.

Method Name: CreatePDFFromWordTemplate

Input

- WordTemplate (required)
- MergedData (required)

Output

- Data as array bytes of pdf file.

Example

```
var documentApi = ctx.Use<IDocumentApi>();

var input = ctx.Input;
var wordDoc = input["WordDocument"];
var wordDocBytes = documentApi.ReadAllBytes(wordDoc);

var mergedData = new Dictionary<string, object>();
mergedData.Add("first_name", "John");
mergedData.Add("last_name", "Doe");
mergedData.Add("address", "Virumgårdsvej 17A, 2830");
mergedData.Add("city", "Virum");
mergedData.Add("phone", "+45 70 26 29 70");
mergedData.Add("email", "contact@globeteam.com");

var sysDocApi = ctx.Use<ISysDocumentApi>();
var result = sysDocApi.ReadAllBytes("avatar");
mergedData.Add("ImageField", result);

var parameters = new Dictionary<string, object>();
parameters.Add("WordTemplate", wordDocBytes);
parameters.Add("MergedData", mergedData);

var pluginApi = ctx.Use<IPluginApi>();
var pluginResult = pluginApi.Invoke("CwAsposeWord", "CreatePDFFromWordTemplate", parameters);

var pdfDocBytes = pluginResult.Get<byte[]>("PDFResult");
var pdfDoc = documentApi.Create(Path.GetFileNameWithoutExtension(wordDoc.FileName)+".pdf");

documentApi.Store(pdfDoc, pdfDocBytes);
var dataApi = ctx.Use<IDataApi>();
dataApi.Update(input["Id"], new
{
    PDFDocument = pdfDoc
});
```

Create file from Word template

Merge data to a Word document template and return the binary of file, user must input the save format.

Method Name: CreateFileFromWordTemplate

Input

- WordTemplate: It's required, this field is the binary of file **Word** template
- MergedData: It's optional, contains merged data which is used to populate to the merge fields which are textual, formatted placeholders in a template.
- SaveFormat: It's required. This field presents the file extension of file output (e.g. doc, docx, pdf, text, html...), refer supported save format

Output

- FileResult: binary of file.
- IsSuccess:
  - `true` if create file successfully.
  - `false` if there are exception while creating file.
- Error: if there are exception while creating file, it has value

Example

```
var documentApi = ctx.Use<IDocumentApi>();
var sysDocumentApi = ctx.Use<ISysDocumentApi>();

var wordDocBytes = sysDocumentApi.ReadAllBytes("template-word");

var mergedData = new Dictionary<string, object>();
mergedData.Add("first_name", "William");
mergedData.Add("last_name", "Hill");
mergedData.Add("address", "Virumgårdsvej 17A, 2830");
mergedData.Add("city", "Virum");
mergedData.Add("phone", "+45 70 26 29 70");

var parameters = new Dictionary<string, object>();
parameters.Add("WordTemplate", wordDocBytes);
parameters.Add("MergedData", mergedData);
parameters.Add("SaveFormat", "Docx");

var pluginApi = ctx.Use<IPluginApi>();
var pluginResult = pluginApi.Invoke("CwAsposeWord", "CreateFileFromWordTemplate", parameters);

var fileBytes = pluginResult.Get<byte[]>("FileResult");
```

Installation

Requirements

- Casewhere v2.6 or later
- A valid Aspose Word license

Configuration

We need to provide the valid purchased license to make sure no water mark appeared on the generated pdf document.

Example

```
{
    "License": //a base64 string
}
```

Releases

1.0.0 - 15/05/2020

Changelog

- Create PDF from Word template

Download (login required): Aspose Word plugin v1.0.0

1.1.0 - 31/05/2022

Changelog

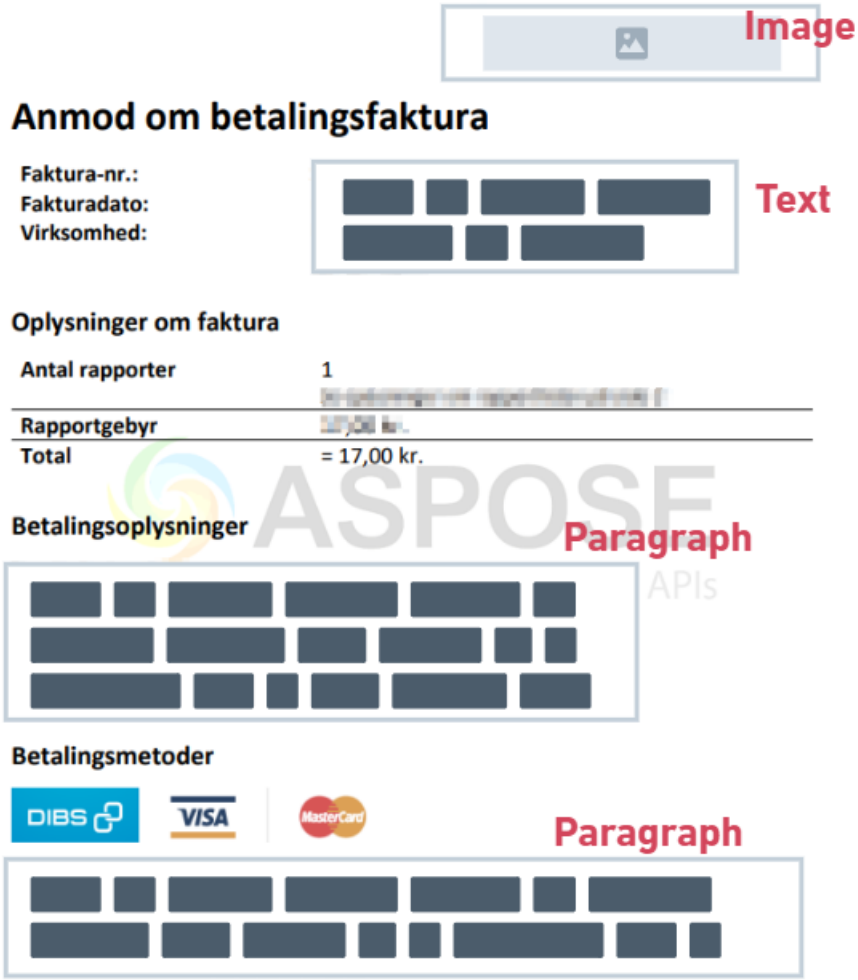
- Create File from Word template

Download (login required): Aspose Word plugin v1.1.0

Plugin: Aspose PDF

Introduction

The plugin provides the ability to add, replace text, paragraph and image to a PDF file by using Aspose.PDF library.



Features

Add a text into a PDF file

Add a text into a PDF file at a specified position and return a PDF file as a result.

Method name: AddTextToPDF

Input:

- PDFDocument(required): The document to add text into.
- Text(required): Text to be added into the document.

Output:

- PDFDocument: The document with text added in.

Example

```
var parameters = new Dictionary<string, object>();
parameters.Add("PDFDocument", pdfDocBytes); // pdfDocBytes is byte array of the PDF file.
parameters.Add("TextContent", ctx.Get<string>("TextContent"));
parameters.Add("PositionX", ctx.Get<int>("PositionX"));
parameters.Add("PositionY", ctx.Get<int>("PositionY"));
var pluginApi = ctx.Use<IPluginApi>();
var pluginResult = pluginApi.Invoke("CwAsposePDF", "AddTextToPDF", parameters);
var pdfDocResultBytes = pluginResult.Get<byte[]>("PDFResult");
```

Add a paragraph into a PDF file

Add a paragraph into a PDF file at a specified position and return a PDF file as a result.

Method name: AddParagraphToPDF

Input:

- PDFDocument(required): The document to add paragraph into.
- Text(required): Paragraph to be added into the document.

Output:

- PDFDocument: The document with paragraph added in.

Example

```
var parameters = new Dictionary<string, object>();
parameters.Add("PDFDocument", pdfDocBytes); // pdfDocBytes is byte array of the PDF file.
parameters.Add("ParagraphContent", textContent);
parameters.Add("UpperRightX", ctx.Get<int>("UpperRightX"));
parameters.Add("UpperRightY", ctx.Get<int>("UpperRightY"));
parameters.Add("LowerLeftX", ctx.Get<int>("LowerLeftX"));
parameters.Add("LowerLeftY", ctx.Get<int>("LowerLeftY"));
parameters.Add("SubsequentLinesIndent", ctx.Get<int>("SubsequentLinesIndent"));
var pluginApi = ctx.Use<IPluginApi>();
var pluginResult = pluginApi.Invoke("CwAsposePDF", "AddParagraphToPDF", parameters);
var pdfDocResultBytes = pluginResult.Get<byte[]>("PDFResult");
```

Replace a text in a PDF file

Replace a text in a PDF file at a specified position and return a PDF file as a result.

Method name: ReplaceTextInPDF

Input:

- PDFDocument(required): The document to replace text.
- OldValue(required): The text to be replaced in a document.
- NewValue(required): The text to replace.

Output:

- PDFDocument: The document with the text replaced.

Example

```
var parameters = new Dictionary<string, object>();
parameters.Add("PDFDocument", pdfDocBytes); // pdfDocBytes is byte array of the PDF file.
parameters.Add("OldValue", oldValue);
parameters.Add("NewValue", newValue);
var pluginApi = ctx.Use<IPluginApi>();
var pluginResult = pluginApi.Invoke("CwAsposePDF", "ReplaceTextInPDF", parameters);
var pdfDocResultBytes = pluginResult.Get<byte[]>("PDFResult");
```

Add an image into a PDF file

Add an image into a PDF file at a specified position and return a PDF file as a result.

Method name: AddImageToPDF

Input:

- PDFDocument(required): The document to add image into.
- Image(required): The image to be added into the document.

Output:

- PDFDocument: The document with the image added in.

Example

```
var parameters = new Dictionary<string, object>();
parameters.Add("PDFDocument", pdfDocBytes); // pdfDocBytes is byte array of the PDF file.
parameters.Add("Image", imageBytes); // imageBytes is byte array of the image file.
parameters.Add("UpperRightX", ctx.Get<int>("UpperRightX"));
parameters.Add("UpperRightY", ctx.Get<int>("UpperRightY"));
parameters.Add("LowerLeftX", ctx.Get<int>("LowerLeftX"));
parameters.Add("LowerLeftY", ctx.Get<int>("LowerLeftY"));
var pluginApi = ctx.Use<IPluginApi>();
var pluginResult = pluginApi.Invoke("CwAsposePDF", "AddImageToPDF", parameters);
var pdfDocResultBytes = pluginResult.Get<byte[]>("PDFResult");
```

Replace an image in a PDF file

Replace an image in a PDF file and return a PDF file as a result.

Method name: ReplaceImageInPDF

Input:

- PDFDocument(required): The document to add image into.
- Image(required): The image to replace the image in the document.
- ImageIndex(required): The index of the image to be replaced in the document.

Output:

- PDFDocument: The document with the image replaced.

Example

```
var parameters = new Dictionary<string, object>();
parameters.Add("PDFDocument", pdfDocBytes); // pdfDocBytes is byte array of the PDF file.
parameters.Add("Image", imageBytes); // imageBytes is byte array of the image file.
parameters.Add("ImageIndex", ctx.Get<int>("ImageIndex"));
parameters.Add("PageNumber", ctx.Get<int>("PageNumber"));
var pluginApi = ctx.Use<IPluginApi>();
var pluginResult = pluginApi.Invoke("CwAsposePDF", "ReplaceImageInPDF", parameters);
var pdfDocResultBytes = pluginResult.Get<byte[]>("PDFResult");
```

Convert Html content to PDF document

Able to convert Html as a string, and return a PDF file as a result.

Method name: HtmlToPdf

Input:

- HtmlContent (required): string - The html content that will be used to convert.

Output:

- PDFResult: byte[] - The stream content as Byte array
- IsSuccess: boolean - true if successful
- Error: string

Example

```
var htmlContent = "<html><head></head><body><h1>My First Heading</h1><p>My first paragraph.</p></body>
</html>";
var parameters = new Dictionary<string, object>();
parameters.Add("HtmlContent", htmlContent);
var pluginApi = ctx.Use<IPluginApi>();
var pluginResult = pluginApi.Invoke("CwAsposePdf", "HtmlToPdf", parameters);
var pdfDocResultBytes = pluginResult.Get<byte[]>("PDFResult");
```

Plugin Result

Check license result

```
var licenseValid = pluginResult.Get<byte[]>("LicenseValid");
var licenseError = pluginResult.Get("LicenseError");
// just have value when LicenseValid is false
```

Error message

```
var outputResult = pluginResult.Get<string>("Output");
```

Installation

Requirements

- Casewhere v2.6 or later
- A valid AsposePDF license

Configuration

A valid purchased license needs to be provided to have no watermark appeared on the generated pdf document.

Example

```
{
    "License": //a base64 string
}
```

Releases

1.0.1 - 23/09/2022

Changelog

- Add method: Convert Html to Pdf

Download (login required): Aspose PDF plugin v1.0.1

1.0.0 - 10/01/2022

Changelog

- Add text to PDF
- Add paragraph to PDF
- Replace text in PDF
- Add image to PDF
- Replace image in PDF

Download (login required): Aspose PDF plugin v1.0.0

Plugin: CVR

Introduction

This plugin provides methods to query company information using the REST API provided by virk.dk.

Features

Search by CVR number

Search companies by CVR number.

Method Name: SearchByCvrNumber

Input

- CvrNumber(required): a string represents a CVR number to search

Output

- Data: The search result
  - CvrNumber
  - Name
  - CityName

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
     { "CvrNumber", "31627885" }
};
var pluginResult = pluginApi.Invoke("CVRPlugin", "SearchByCvrNumber", parameters);
var output = pluginResult.GetObject("Data");
```

Search by CVR number v2

Search companies by CVR number.

Method Name: SearchByCvrNumberV2

Input

- CvrNumber(required): a string represents a CVR number to search

Output

- Data: The search result
  - MunicipalityCode
  - MunicipalityName
  - Address
  - StreetCode
  - StreetName
  - HouseNumber
  - FloorNumber
  - DoorLetterSide
  - ZipCode
  - City

- PhoneNumber
- CompanyName
- CvrNumber
- PNumbers: array of P numbers
- MainIndustry
- BranchCode

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
     { "CvrNumber", "31627885" }
};
var pluginResult = pluginApi.Invoke("CVRPlugin", "SearchByCvrNumberV2", parameters);
var output = pluginResult.GetObject("Data");
```

Search by P number

Search companies by P number.

Method Name: PNumber

Input

- PNumber(required): a string represents a CVR number to search

Output

- Data: The search result
  - PNumber
  - CvrNumber
  - Name
  - COName
  - Floor
  - Sidedoor
  - StreetName
  - HouseNumber
  - ZipCode
  - PhoneNumber
  - Email

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
     { "PNumber", "31627885" }
};
var pluginResult = pluginApi.Invoke("CVRPlugin", "SearchByPNumber", parameters);
var output = pluginResult.GetObject("Data");
```

Search by P number v2

Search companies by P number.

Method Name: PNumber

Input

- PNumber(required): a string represents a CVR number to search

Output

- Data: The search result

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
     { "PNumber", "31627885" }
};
var pluginResult = pluginApi.Invoke("CVRPlugin", "SearchByPNumberV2", parameters);
var output = pluginResult.GetObject("Data");
```

Installation

Requirements

- Casewhere v2.5 or later

Configuration

Configure the plugin as below:

```
{
  "ApiUrl": "http://distribution.virk.dk/",
  "ApiIndex": "cvr-permanent",
  "Username": // username,
  "Password": // password
}
```

Releases

1.1.0 - 15/05/2021

Changelog

- Search by CVR number v2
- Search by P number v2

Download (login required): CVR plugin v1.1.0

1.0.0 - 25/02/2019

Changelog

- Search by CVR number
- Search by P number

Download (login required): CVR plugin v1.0.0

Plugin: File compression

Introduction

This plugin provides methods to ease working with zip files.

Features

Zip an array of bytes

Compress an array of bytes.

Method Name: ZipBytes

Input

- FileBytes (required): a key-value dictionary where
  - Key: Name of the file
  - Value: A byte-array - the content to be compressed

Output

- Output: An object
  - IsSuccess: `true` or `false`
  - Data: a base64 string represents the compressed content
  - Error: The error message in case something goes wrong

Example

```
var pluginApi = ctx.Use<IPluginApi>();

var input = new Dictionary<string, byte[]>();
var fileName = ctx.Get<string>("FileName");
var bytes = ctx.Get<byte[]>("Bytes");
input.Add(fileName, bytes);

var parameters = new Dictionary<string, object>()
{
     { "FileBytes", input }
};
var pluginResult = pluginApi.Invoke("CwCompression", "ZipBytes", parameters);
var output = pluginResult.GetObject("Output");
```

Unzip an array of bytes

Decompress an array of bytes.

Method Name: UnzipBytes

Input

- ZippedBytes (required): a compressed byte array to decompress

Output

- Output: An object
  - IsSuccess: `true` or `false`
  - Data: A key-value dictionary where
    - Key: Name of the file

        - Value: A byte-array - the content to be compressed
    - Error: The error message in case something goes wrong

Example

```
var pluginApi = ctx.Use<IPluginApi>();

var bytes = ctx.Get<byte[]>("Bytes");
var parameters = new Dictionary<string, object>()
{
     { "ZippedBytes", bytes }
};
var pluginResult = pluginApi.Invoke("CwCompression", "UnzipBytes", parameters);
var output = pluginResult.GetObject("Output");
```

Installation

Requirements

- Casewhere v2.6 or later

Configuration

The plugin requires no configuration.

Releases

1.0.0 - 15/05/2020

Changelog

- Zip an array of bytes

Download (login required): File compression plugin v1.0.0

Plugin: JWT

Introduction

JSON Web Tokens (JWT) are an open, industry standard RFC 7519 method for representing claims securely between two parties. The plugin is developed to provide easy-to-use methods for working with JWT.

Features

Decode a token

Decode a JWT token.

Method Name: Decode

Input

- Token(required): The JWT string to decode.
- Audience(optional): When supplied, the plugin will validate the token against the given audience.
- Issuer(optional): When supplied, the plugin will validate the token against the given issuer.

Output

- Payload: the JSON string represents the JWT payload.
- IsValid: `true` or `false`.
- Error: The error message in case token is not valid.

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>();
parameters.Add("Token",
"eyJhbGciOiJSUzI1NiIsImtpZCI6Ijg0ODM2QUE0NTdGQUY2NDJENzY4NTg1NTQ3MURBNURGOEY5NTU2NUUiLCJ4NXQiOiJoSU5xcEZmNjlr
TFhhRmhWUngybDM0LVZWbDQiLCJ0eXAiOiJKV1QifQ.eyJzdWIiOiJhZG1pbnRodWUiLCJuYW1lIjoiYWRtaW50aHVlIiwidG9rZW5fdXNhZ2
UiOiJhY2Nlc3NfdG9rZW4iLCJqdGkiOiI4NTQ5OGZhZS0wNmJjLTQzODAtODgyMC1mMWI3Y2QxMjM4ZDkiLCJzY29wZSI6ImlkZW50aWZ5KmV
tcHR5IiwiYXVkIjoiaHR0cHM6Ly9naWFzLWRlbW8uY2FzZXdoZXJlLmNvbSIsImF6cCI6ImFkbWludGh1ZSIsImlhdCI6MTYxNzgzNTk2MCwi
bmJmIjoxNjE3ODM1OTYwLCJleHAiOjE2MTc4Mzk1NjAsImlzcyI6Imh0dHBzOi8vbG9naW4tZGVtby5jYXNld2hlcmUuY29tL3J1bnRpbWUvb
2F1dGgyIn0.c6ob82U7Rcc8EhUQhKZBCiwpftbr_wGSL_85zPjGt-
DCinZSzLj0fmva1Yp896hwdg5gT6812HU1hxG9rHuNYetkVCzh8wlgrDmxGfmAlvmXayBYvCzCDpnYXU1d0TuXO52o8buY0m4fn4DFcqkufkf
szegvUzoql13N2CxdtJMieo8F1u0IUwVfZ6xIylNIWFX1cpeXcWZsbXzy2KCzchDlo4D9gjF0O9JlBXVfdPYya6Odb9orITPAndR8c1WKjtlk
Y8_ulLTzRS0MYSYuB1FKBJTqZp2Q_Yt3q6CoWCvwwAD9eTC22tp--n20TE9ermIYbE1LRLloszCmlNSXxg");
parameters.Add("Audience", "https://demo.casewhere.com");
parameters.Add("Issuer", "https://login-demo.casewhere.com/runtime/oauth2");
var result = pluginApi.Invoke("CwJWTPlugin", "Decode", parameters);
```

Installation

Requirements

- Casewhere v2.5 or later.

Configuration

You need to specify the default signing method for encoding/decoding the tokens.

No signing

```
{ SigningAlgorithmType: "None" }
```

HMAC

Plain text symmetric key

```
{
     SigningAlgorithmType: "HMAC",
     SigningKeyInformation: {
         SymmetricKey: "12345678912345678901234567890123",
         EnableBase64: false
     }
}
```

Base64 symmetric key

```
{
     SigningAlgorithmType: "HMAC",
     SigningKeyInformation: {
         SymmetricKey: "MTIzNDU2Nzg5MTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM=",
         EnableBase64: true
     }
}
```

RSA

Certification store location

```
{
     SigningAlgorithmType: "RSA",
     SigningKeyInformation: {
         Thumbprint: "84836aa457faf642d7685855471da5df8f95565e",
         StoreName: "TrustedPeople",
         StoreLocation: "LocalMachine"
     }
}
```

Base64 certificate

```
{
     SigningAlgorithmType: "RSA",
     SigningKeyInformation: {
         Base64Cert:
"MIIDRzCCAi+gAwIBAgIQjLNmp7eE04FFKdfFg5UUJjANBgkqhkiG9w0BAQsFADAjMSEwHwYDVQQDExhjYXNld2hlcmUtNjZkMWI2IFNpZ25p
bmcwHhcNMTkwOTE0MjIwMDAwWhcNMjcwOTE0MjE1OTU5WjAjMSEwHwYDVQQDExhjYXNld2hlcmUtNjZkMWI2IFNpZ25pbmcwggEiMA0GCSqGS
Ib3DQEBAQUAA4IBDwAwggEKAoIBAQDmNXXcgVmu1cVD7vfIzl52Qvp68twLiEo92xe0sd4tcQaGMI86rviJMZtsIqqx7fGyyRTQBLuRzecyb9
AZxIrjC7Ei4nHzz4iu/7lOGZvnCDHjrkV1rLDqjebJG1+MR67tQwzP/km+tdTXUVCCLAwTqYMHM/4hZopEA+r4foJ8Pvv1pWdHNxIuELFYmkD
KkyFhQJbFzKEhG73m62W5+WYhMDxPF76iS3Damzv22XJAkZ/66LiLTkvfxijRT581DWqzJyIYBH4LJQnPNAW2R3ufPgvNFa4DecXlVnc8eUDA
AZKIthkloJ3zsxOPyWkzCwwSufFnf7V3NoqpR3meSSkpAgMBAAGjdzB1MB0GA1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjBUBgNVHQEET
TBLgBAyby8xQTwdz08EQv0Oo2EPoSUwIzEhMB8GA1UEAxMYY2FzZXdoZXJlLTY2ZDFiNiBTaWduaW5nghCMs2ant4TTgUUp18WDlRQmMA0GCS
qGSIb3DQEBCwUAA4IBAQA1ldjnil03R4Ib4fswIV+OBjYYc+QmI+HDo5ZkHUUtEvNw7MclVUV2Ls/IsJGSiclIZOLt5vU4LIlplksyjSMBQcO
3iYCteJKgEM3b0g/uNpTqdeV5HtYGusuS47/EbpZwYNPfvd0/Hxp5/mn134raPQ8fMKmFGWU9328wY7s4r6pbo2w+7YPurLhLjx3iJA7fSXhV
nq6sJeytegyHocfbpNlNTpOcZ5mf3UhJ4Vywu5/HuZKjPkTkK7NU6XMLShr5Oq5DsIcDTxKO48z6stmuqI9JJlTDNOvZmKrLKv9NCqvR67GBu
It6FMpNNCCd8x3D0t446cK733AkaaEiu17X"
     }
}
```

Releases

1.0.0 - 23/04/2021

Changelog

- Decode

Download (login required): jwt plugin v1.0.0

Plugin: Query Filter String Parser

Introduction

The Query Filter String Parser plugin can parse a filter part of query string into a structured object. In the version 1.0.0, the plugin only supports to parse a filter string, which follows the OData format, into a BsonDocument object that represents the MongoDB filter.

Features

Parse OData Filter String To MongoDB Filter

This feature receives an OData filter string and transforms it into a MongoDB filter which is a BsonDocument object.

Method Name: ParseODataFilterStringToMongoDBFilter

Input Filter [required]: filter string that follows the OData format.

Following the below sample:

```
    var parameters = new Dictionary<string, object>
    {
        { "Filter" , "Name eq 'Indiana Jones'" }
    };
```

Use the **Invoke** method in IPluginApi to call **ParseODataFilterStringToMongoDBFilter** method:

```
    IPluginApi.Invoke("CwODataUriParser Query Filter String Parser", "ParseODataFilterStringToMongoDBFilter",
parameters);
```

Output

The method will return a BsonDocument object in `Filter` field if it can parse successfully. Otherwise, it will return error details in `Error` field.

Example

```
    var parameters = new Dictionary<string, object>
    {
        { "Filter" , "Name eq 'Indiana Jones'" }
    };

    var pluginResult = IPluginApi.Invoke("CwODataUriParser Query Filter String Parser",
"ParseODataFilterStringToMongoDBFilter", parameters);

    if (pluginResult.Data.ContainsKey("Error"))
    {
        Log.Error("[CwODataUriParser Query Filter String Parser][ParseODataFilterStringToMongoDBFilter]
{@error}", pluginResult.GetObject("Error"));

        return null;
    }

    var filter = pluginResult.Get<BsonDocument>("Filter");
    var dataFilter = DynamicFilter.Parse(filter);
```

Installation

Requirements

- Casewhere v2.6.0 or later.

  Settings

  In 1.0.0 version, the plugin only supports one setting:

- **ODataParserMaxDepth**: The maximum number of the recursive parsing steps that the parser can perform. Default is 100.

Example

```
{
    "ODataParserMaxDepth": 250
}
```

Releases

1.0.0 - 14/07/2022

Changelog

- ParseODataFilterStringToMongoDBFilter method.

Download (login required) QueryFilterStringParser plugin v1.2.0

Plugin: SQL client

Introduction

One of the challenges when moving an old system to a new one is data migration. With the SQL client plugin, data migration made easy. User can use SQL client plugin to query data from a MSSQL database, transform and insert into Casewhere.

Features

Execute a data query

Execute a SQL query and retrieve data. The plugin accepts parameters to protect against SQL injection vulnerabilities.

Method Name: ExecuteQuery

Input

- Query (required): a SQL statement
  - Example: `select top(500) from [DimProduct] where [ProductKey] > @ProductKey`
- Parameters (optional): parameters used in the SQL statement. It should be a JSON string.
- Timeout (optional): command timeout.
- ConnectionString (optional).

Output

- Data: list of objects.

Example

```
var documentApi = ctx.Use<IDocumentApi>();
var pluginApi = ctx.Use<IPluginApi>();
var dataApi = ctx.Use<IDataApi>();

// Construct plugin parameters
var parameters = new Dictionary<string, object>();
parameters.Add("Query", "select top(500) * from [DimProduct] where [ProductKey] > @ProductKey");
parameters.Add("Parameters", JsonConvert.SerializeObject(new { ProductKey = 10 }));

// Invoke plugin
var pluginResult = pluginApi.Invoke("SqlClient", "ExecuteQuery", parameters);
var products = pluginResult.GetArray("Data");

// Insert data to DB
foreach(dynamic product in products)
{
  // binary data is received in base64 format
  var docInfo = documentApi.Create($"photo{product.ProductKey}.jpg");
  documentApi.Store(docInfo, Convert.FromBase64String(product.LargePhoto.ToString()));
  dataApi.Add("CwSqlDimProduct", new
  {
      ProductKey = product.ProductKey,
      Description = product.EnglishDescription,
      ModelName = product.ModelName,
      ProductName = product.EnglishProductName,
      LargePhoto = docInfo,
      ListPrice = product.ListPrice,
      DealerPrice = product.DealerPrice,
      Color = product.Color,
      Style = product.Style
  });
}
```

Execute a scalar query

Execute a SQL query and retrieve a scalar value. The plugin accepts parameters to protect against SQL injection vulnerabilities.

Method Name: ExecuteQueryScalar

Input

- Query (required): a SQL statement.
    - Example:

      ```
      select [LargePhoto] from [DimProduct] where [ProductKey] = @ProductKey
      ```

- Parameters (optional): parameters used in the SQL statement. It should be a JSON string.
- Timeout (optional): command timeout.
- ConnectionString (optional).

Output

- Data: a scalar value.

Example

```
var documentApi = ctx.Use<IDocumentApi>();
var pluginApi = ctx.Use<IPluginApi>();
var dataApi = ctx.Use<IDataApi>();

// Query an existing record in Casewhere
var filter = FilterBuilder.Create().Eq("ProductKey", 13).Build();
var result = dataApi.Search(DataObjectApiQuery
  .For("CwSqlDimProduct")
  .FilterBy(filter));
dynamic product = result.Data[0];

// Construct plugin parameters to query an image (bytes) for a given product
var parameters = new Dictionary<string, object>();
parameters.Add("Query", "select LargePhoto from [DimProduct] where [ProductKey] = @ProductKey");
parameters.Add("Parameters", JsonConvert.SerializeObject(new { ProductKey = 13 }));

// Execute plugin
var pluginResult = pluginApi.Invoke("SqlClient", "ExecuteQueryScalar", parameters);
var bytes = pluginResult.Get<byte[]>("Data");

// Retrieve the image and update back to Casewhere
var docInfo = documentApi.Create($"photo{product.ProductKey}.jpg");
documentApi.Store(docInfo, bytes);
dataApi.Update(product.Id, new
{
  LargePhoto = docInfo
});
```

Get table schema

Get table schema by name.

Method Name: GetTableSchema

Input

- Name(required): table name.
- ConnectionString (optional).

Output

- Schema: an object in following format.

```
{
"Name": "DimProduct",
"Columns": [
 {
    "Name": "ProductKey",
    "DataType": "Int32",
    "IsKey": true,
    "IsNullable": false
 },

 {
    "Name": "ProductAlternateKey",
    "DataType": "String",
    "IsKey": false,
    "IsNullable": true
 }
]
}
```

Example

```
var pluginApi = ctx.Use<IPluginApi>();

// Construct plugin parameters
var parameters = new Dictionary<string, object>();
parameters.Add("Name", "DimProduct");

// Invoke plugin
var pluginResult = pluginApi.Invoke("SqlClient", "GetTableSchema", parameters);
var schema = pluginResult.GetObject("Schema");
Log.Info("{schema}", JsonConvert.SerializeObject(schema, Newtonsoft.Json.Formatting.Indented));
```

Installation

Requirements

- Casewhere v2.6 or later.
- MSSQL active connection. ### Configuration
- ConnectionString: default connection string.

Example

```
{
    "ConnectionString":"Data Source=MSSQLSERVER;Initial
Catalog=AdventureWorksDW;UserId=sa;Password=a_password;MultipleActiveResultSets=true;"
}
```

Releases

1.0.0 - 23/04/2021

Changelog

- Execute a data query
- Execute a scalar query
- Get table schema

Download (login required)

- SQL client plugin v1.0.0
- Sample Database

Plugin: SFTP

Introduction

The plugin provides an easy way to transfer files over the web using SFTP.

Features

Upload file

Upload file using SSH.

Method Name: Upload

Input

- Data: the object contains upload information.
  - FileName(required): a string represents a file name to upload.
  - Data(required): a file content.
  - ReplaceIfExist(optional): default value is false, if true the existing file will be replaced.
- Host(optional): a server host, default value is set from plugin setting.
- Port(optional): a server port, default value is set from plugin setting.
- RootPath(optional): a root path to transfer file, default value is set from plugin setting.
- Username(optional): a username to authenticate, default value is set from plugin setting.
- Password(optional): a password to authenticate, default value is set from plugin setting.
- AuthenticationType(optional): a type to authenticate, it can be password or private key.
- PrivateKey(optional): a private key to authenticate.
- Passphrase(optional): a passphrase to protects the private key.

Output

- Data: the object contains upload information
  - IsSuccess: If the transferring is success or not.

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var uploadData = new
{
    FileName = "file_name.csv",
    Data = new byte[]{}
};
var parameters = new Dictionary<string, object>()
{
    { "Data", JsonConvert.SerializeObject(uploadData) },
    { "Host", "127.0.0.1" },
    { "Port", 22 }
};
var pluginResult = pluginApi.Invoke("SFTPPlugin", "Upload", parameters);
var output = pluginResult.GetObject("Data");
```

Upload file using temporary file on server

Upload file via SSH using temporary file on the server, to avoid memory leak in case the file is large.

Method Name: UploadWithFileName

Input

- Data: the object contains upload information.
    - FileName(required): a string represents a file name to upload.
    - TempFileName(required): a temporary file on the server.
    - ReplaceIfExist(optional): default value is false, if true the existing file will be replaced.
- Host(optional): a server host, default value is set from plugin setting.
- Port(optional): a server port, default value is set from plugin setting.
- RootPath(optional): a root path to transfering file, default value is set from plugin setting.
- Username(optional): a username to authenticate, default value is set from plugin setting.
- Password(optional): a password to authenticate, default value is set from plugin setting.
- AuthenticationType(optional): a type to authenticate, it can be password or private key.
- PrivateKey(optional): a private key to authenticate.
- Passphrase(optional): a passphrase to protects the private key.

Output

- Data: the object contains upload information.
    - IsSuccess: If the transferring is success or not.

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var uploadData = new
{
    FileName = "file_name.csv",
    TempFileName = System.IO.Path.GetTempFileName()
};
var parameters = new Dictionary<string, object>()
{
    { "Data", JsonConvert.SerializeObject(uploadData) },
    { "Host", "127.0.0.1" },
    { "Port", 22 }
};
var pluginResult = pluginApi.Invoke("SFTPPlugin", "UploadWithFileName", parameters);
var output = pluginResult.GetObject("Data");
```

Installation

Requirements

- Casewhere v2.5 or later

Configuration

Configure the plugin as below:

```
{
  "Host": "127.0.0.1",
  "Port": 22,
  "Username": // username,
  "Password": // password,
  "RootPath": "/C:/STFP"
}
```

Releases

1.1.0 - 19/01/2022

Changelog

- Update plugin parameters to include connection.

Download (login required) SFTP plugin v1.1.0 link to be updated

1.0.0 - 15/07/2021

Changelog

- Upload file.
- Upload file using temporary file on server.

Download (login required) SFTP plugin v1.0.0 link to be updated

Plugin: SSH

Introduction

The plugin provides an easy way to transfer files over the web.

Features

Upload file

Upload file via SSH protocol.

Method Name: UploadV2

Input

- FileName(required): a string represents a file name to upload.
- Bytes(required): a file content.
- Host(optional): a server host, default value is set in plugin settings.
- Port(optional): a server port, default value is set in plugin settings.
- RootPath(optional): a root path to transfer file, default value is set in plugin settings.
- Username(optional): a username to authenticate, default value is set in plugin settings.
- Password(optional): a password to authenticate, default value is set in plugin settings.
- AuthenticationType(optional): a type to authenticate, it can be password or private key.
- PrivateKey(optional): a private key to authenticate.
- Passphrase(optional): a passphrase to protects the private key.
- Override(optional): an option to override existing file when upload, if does not have value is set in plugin settings or parameters, default is true.
- CreateDirectory(optional): an option to allow to create a directory when upload if it does not exists, if does not have value is set in plugin settings or parameters, default is true.

Output Data object:

- IsSuccess: True if file transfers successfully, else False.
- Data: The file content. Null if file transferred failed.
- Error: Error message if any.

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
    { "FileName", "file_name.csv" },
    { "Bytes", new byte[]{} },
    { "Host", "127.0.0.1" },
    { "Port", 22 }
};
var pluginResult = pluginApi.Invoke("SSHPlugin", "UploadV2", parameters);
var output = pluginResult.GetObject("Data");
```

Download file

Download file content via SSH protocol.

Method Name: DownloadV2

Input

- FileName(required): a string represents a file name to download.
- Host(optional): a server host, default value is set in plugin settings.
- Port(optional): a server port, default value is set in plugin settings.
- RootPath(optional): a root path to transfer file, default value is set in plugin settings.
- Username(optional): a username to authenticate, default value is set in plugin settings.
- Password(optional): a password to authenticate, default value is set in plugin settings.
- AuthenticationType(optional): a type to authenticate, it can be password or private key.
- PrivateKey(optional): a private key to authenticate.
- Passphrase(optional): a passphrase to protects the private key.

Output Data object:

- IsSuccess: True if file transfers successfully, else False.
- Data: The file content. Null if file transferred failed.
- Error: Error message if any.

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
    { "FileName", "file_name.csv" },
    { "Host", "127.0.0.1" },
    { "Port", 22 }
};
var pluginResult = pluginApi.Invoke("SSHPlugin", "DownloadV2", parameters);
var output = pluginResult.GetObject("Data");
```

Delete file

Delete file via SSH protocol.

Method Name: DeleteV2

Input

- FileName(required): a string represents a file name to delete
- Host(optional): a server host, default value is set in plugin settings.
- Port(optional): a server port default value is set in plugin settings.
- RootPath(optional): a root path to transfer file, default value is set in plugin settings.
- Username(optional): a username to authenticate, default value is set in plugin settings.
- Password(optional): a password to authenticate, default value is set in plugin settings.
- AuthenticationType(optional): a type to authenticate, it can be password or private key.
- PrivateKey(optional): a private key to authenticate.
- Passphrase(optional): a passphrase to protects the private key.

Output Data object:

- IsSuccess: True if file is deleted successfully, else False.
- Error: Error message if any.

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
    { "FileName", "file_name.csv" },
    { "Host", "127.0.0.1" },
    { "Port", 22 }
};
var pluginResult = pluginApi.Invoke("SSHPlugin", "DeleteV2", parameters);
var output = pluginResult.GetObject("Data");
```

List files

List files in a folder via SSH protocol.

Method Name: ListFilesV2

Input

- FolderName(required): a string represents a folder name to get files.
- LastModified(optional): get files that have LastWriteTimeUtc greater than LastModified value,get all files in folder by default.
- Host(optional): a server host, default value is set in plugin settings.
- Port(optional): a server port, default value is set in plugin settings.
- RootPath(optional): a root path to transfer file default value is set in plugin settings.
- Username(optional): a username to authenticate, default value is set in plugin settings.
- Password(optional): a password to authenticate, default value is set in plugin settings.
- AuthenticationType(optional): a type to authenticate, it can be password or private key.
- PrivateKey(optional): a private key to authenticate.
- Passphrase(optional): a passphrase to protects the private key.

Output Data object:

- IsSuccess: True if get files successfully, else False.
- Data: An array contains the list of files.
- Error: Error message if any.

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
    { "FolderName", "Archive/Incoming" },
    { "LastModified", new DateTime(2021, 12, 01) }
    { "Host", "127.0.0.1" },
    { "Port", 22 }
};
var pluginResult = pluginApi.Invoke("SSHPlugin", "ListFilesV2", parameters);
var output = pluginResult.GetObject("Data");
```

Move file

Move file to other location via SSH protocol.

Method Name: MoveFileV2

Input

- FileName(required): a string represents file name to move.
- ToFolder(required): a string represents a destination folder.
- Host(optional): a server host, default value is set in plugin settings.
- Port(optional): a server port, default value is set in plugin settings.

- RootPath(optional): a root path to transfer file, default value is set in plugin settings.
- Username(optional): a username to authenticate, default value is set in plugin settings.
- Password(optional): a password to authenticate, default value is set in plugin settings.
- AuthenticationType(optional): a type to authenticate, it can be password or private key.
- PrivateKey(optional): a private key to authenticate.
- Passphrase(optional): a passphrase to protects the private key.

Output Data object:

- IsSuccess: True if file is relocated successfully, else False.
- Error: Error message if any.

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
    { "FileName", "file_name.csv" },
    { "FolderName", "Archive/Outgoing" },
    { "Host", "127.0.0.1" },
    { "Port", 22 }
};
var pluginResult = pluginApi.Invoke("SSHPlugin", "MoveFileV2", parameters);
var output = pluginResult.GetObject("Data");
```

Rename directory

Rename directory via SSH protocol.

Method Name: RenameDirectory

Input

- Path(required): a string represents current path.
- NewPath(required): a string represents a new path.
- Host(optional): a server host, default value is set in plugin settings.
- Port(optional): a server port, default value is set in plugin settings.
- RootPath(optional): a root path to transfer file, default value is set in plugin settings.
- Username(optional): a username to authenticate, default value is set in plugin settings.
- Password(optional): a password to authenticate, default value is set in plugin settings.
- AuthenticationType(optional): a type to authenticate, it can be password or private key.
- PrivateKey(optional): a private key to authenticate.
- Passphrase(optional): a passphrase to protects the private key.

Output Data object:

- IsSuccess: True if directory is renamed successfully, else False.
- Error: Error message if any.

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
    { "Path", "Archive/Outgoing" },
    { "NewPath", "Archive/Outgoing_New" },
    { "Host", "127.0.0.1" },
    { "Port", 22 }
};
var pluginResult = pluginApi.Invoke("SSHPlugin", "RenameDirectory", parameters);
var output = pluginResult.GetObject("Data");
```

Installation

Requirements

- Casewhere v2.5 or later

Configuration

Configure the plugin as below:

```
{
  "Host": "127.0.0.1",
  "Port": 22,
  "Username": // username,
  "Password": // password,
  "RootPath": "/C:/STFP",
  "Override": true,
  "CreateDirectory": true
}
```

Releases

1.1.2

Changelog

- Add Override configuration. Default is true.
- Add CreateDirectory configuration. Default is true.
- Add new method RenameDirectory **Download (login required)**: SSH Plugin v1.1.1 update link.

1.1.1 - 21/03/2022

Changelog

- Update the lasted version for Renci.SshNet.
- Add unit tests.
  Download (login required): SSH Plugin v1.1.1

1.1.0 - 20/01/2022

Changelog

- UploadV2
- DownloadV2
- DeleteV2
- ListFilesV2
- MoveFileV2
  Download (login required): SSH plugin v1.1.0 update link.

1.0.0 - 24/07/2021

Changelog

- Upload
- Download
- Delete
- ListFiles
- MoveFile
  Download (login required): SSH plugin v1.0.0 update link.

Plugin: CSV

Introduction

The CSV plugin lets you export the raw data from the Casewhere system to the CSV file using the CsvHelper library.

Features

Export a CSV file

From version 1.2.0, the method Export is replaced by WriteData. There are 2 mandatory paramaters, FilePath and Data. Optional parameters are described in the next section.

Method Name: WriteData

Input

- FilePath: Required
- Data: Required, it's must be `IEnumerable<IEnumerable<object>>`
- Append: Boolean value which indicates the file should be appended or overwritten
- IncludeHeaders: Boolean value which indicates whether the first line in the input data. If it is true, the first line is treated as headers
- Options: `IDictionary<string, object>`. The options are the same with the plugin's settings. If an option is defined here, it will overwrite the corresponding plugin's setting

Following the below sample:

```
    var options = new Dictionary<string, object>
    {
        { "CultureInfo", "da-DK" },
        { "EscapeCharacter", '"' },
        { "Delimiter", "," },
        { "DateTimeFormat", "dd-MMM-yyyy HH:mm:ss" },
        { "EncloseAll", false },
        { "EnclosingCharacter", '"' },
        { "Encoding", "UTF-8" },
        { "ByteOrderMark", false },
        { "BigEndian", false },
        { "EncloseAll", false }
    };
    var parameters = new Dictionary<string, object>
    {
        { "FilePath" , "\Test\output.csv" },
        { "Data" , data },
        { "Append", true },
        { "IncludeHeaders", true },
        { "Options", options }
    };
```

Use the **Invoke** method in IPluginApi to call **WriteData** feature:

```
    IPluginApi.Invoke("CwCsv", "WriteData", IDictionary<string, object> parameters);
```

- **Example:**

```
var path = Path.GetTempPath();
var fileName = $"cw_csv_export_sample_{DateTime.Now.ToString("ddMMyyyyHHmmss")}.csv";
var filePath = Path.Combine(path, fileName);
DateTime rst = TimeZoneInfo.ConvertTimeBySystemTimeZoneId(DateTime.UtcNow, "Romance Standard Time");
DateTime cst = TimeZoneInfo.ConvertTimeBySystemTimeZoneId(DateTime.UtcNow, "China Standard Time");
DateTime utc = DateTime.UtcNow;

var data = new List<List<object>>
{
    new List<object> { "Name", "Age", "BirthDate" }
    new List<object> { "Romance Standard Time", 18, rst }
    new List<object> { "China Standard Time", 19, cst }
    new List<object> { "Current Time", 25, DateTime.Now }
    new List<object> { "UTC én af de påkrævede", 25, DateTime.Now }
}

var parameters = new Dictionary<string, object>
{
    { "FilePath" , filePath },
    { "Data" , data },
    { "Append", false },
    { "IncludeHeaders", true }
};

var pluginResult = pluginApi.Invoke("CwCsv", "WriteData", parameters);

var result = pluginResult.Data;
bool isSuccess = result["IsSuccess"];
string errorMessage = result["Error"];
```

Method Name: [Obsolete] Export

Input

- Headers: Optional, it is the list of header columns in the CSV file. if it's null or empty, the CSV file has no header.
- FileName: Required.
- Data: Required, it's must be a `List<Dictionary<string, object>>`, `List<object>` or a serialized JSON string of array object.
- FileMode: Optional, value is `Create` or `Append` (Append is default).
    - If the FileMode is `Create`, the new file will be created and overridden the existing one on the temp folder.
    - If the FileMode is `Append`, the system opens the file if it exists and append the data from the end of the file, or creates a new file.
- CultureInfo: Optional. It is a CSV format option, which will override the configuration (refer to section Configuration) if it is specified.
- Escape: Optional. It is a CSV format option, which will override the configuration (refer to section Configuration) if it is specified.
- Delimiter: Optional. It is a CSV format option, which will override the configuration (refer to section Configuration) if it is specified.
- DateTimeFormat: Optional. It is a CSV format option, which will override the configuration (refer to section Configuration) if it is specified.
- EncloseAll: Optional. It is a CSV format option, which will override the configuration (refer to section Configuration) if it is specified.
- EncloseCharacter: Optional. It is a CSV format option, which will override the configuration (refer to section Configuration) if it is specified.
- UTF8BOM: Optional. It is a CSV format option, which will override the configuration (refer to section Configuration) if it is specified.

Follow the below sample:

```
    var parameters = new Dictionary<string, object>
    {
        { "Headers" , new List<string>() },
        { "FileName", "fileName" },
        { "Data", JsonConvert.SerializeObject(data) },
        { "FileMode", "Create" },
        { "CultureInfo", "da-DK" },
        { "Escape", "\"" },
        { "Delimiter", "," },
        { "DateTimeFormat", "dd-MMM-yyyy HH:mm:ss" },
        { "EncloseAll", true },
        { "EncloseCharacter", "\"" },
        { "UTF8BOM", false }
    };
```

Use the **Invoke** method in IPluginApi to call **Export** feature:

```
    IPluginApi.Invoke("CwCsv", "Export", IDictionary<string, Object> parameters)
```

Output

The method returns data with two fields `IsSuccess`, `Error`. When exported successfully, the CSV file will be created in the temp folder.

Example

```
    var pluginApi = ctx.Use<IPluginApi>();
    var dataApi = ctx.Use<IDataApi>();
    var fileName = $"cw_csv_export_sample_{DateTime.Now.ToString("ddMMyyyyHHmmss")}.csv";

    // Get data
    var filter = FilterBuilder.Create()
                            .Eq("CaseState", "Created")
                            .Eq("IsDeleted", false)
                            .Build();

    IList<DynamicDataObject> data = dataApi.Search(DataSourceApiQuery
                                        .For("CwUsers")
                                        .FilterBy(filter))
                                        .Data;

    // Construct plugin parameters
    var parameters = new Dictionary<string, object>
    {
        { "Headers" , new List<string>{ "Name", "Age" } },
        { "FileName", fileName },
        { "Data", JsonConvert.SerializeObject(data) },
        { "FileMode", "Create" }
    };

    // Invoke plugin
    var pluginResult = pluginApi.Invoke("CwCsv", "Export", parameters);

    var result = pluginResult.Data;
    bool isSuccess = result["isSuccess"];
    string errorMessage = result["Error"];
```

The following code is used to get the file path of the CSV file:

```
var filePath = Path.Combine(Path.GetTempPath(), fileName);
```

- **Note:** The function will not delete the file after exporting, we need to delete it manually after use.

Installation

Requirements

- Casewhere v2.6.0 or later. ### Configuration The plugin supports some settings while exporting to csv file, such as:
- **CultureInfo**: Name of the culture. Default is the culture of the thread that execute the plugin
- **DateTimeFormat**: Format for date-time value. Default is the full date-time format of the culture
- **Delimiter**: The delimiter to separate fields. Default is the list delimiter of the culture
- **NewLine**: the string that marks the new line. Default is the new line string of the running environment
- **Encoding**: Name of the encoding. Default is `UTF-8`
- **BigEndian**: Value that indicates whether to use big endian for Unicode encoding or not. If the encoding does not support the big endianm it will be ignored. Default value is `false`
- **ByteOrderMark**: Value that indicates whether to insert byte-order-mark in the exported file or not. If the encoding does not support, it will be ignored. Default value is `false`
- **EscapeCharacter**: Special character that is needed to be escaped. Dafault value is `"`
- **EnclosingCharacter**: Character that is used to enclose fields. Default value is `"`
- **EncloseAll**: Value that indicates whether to wrap all fields in the enclosing character. Default value is `false`

Example

```
{
    "CultureInfo": "da-DK",
    "DateTimeFormat": "dd-MM-yyyy HH:mm:ss",
    "Delimiter": ",",
    "NewLine": "\r\n",
    "Encoding": "UTF-8",
    "ByteOrderMark": false,
    "EscapeCharacter": "\"",
    "EnclosingCharacter": "\"",
    "EncloseAll": true
}
```

Releases

1.0.0 - 01/03/2022

Changelog

- Export CSV file.

1.1.0 - 11/03/2022

Changelog

- Support header parameter when exporting file

1.2.0 - 23/05/2022

Changelog

- Reorganize CSV settings
- Add more CSV settings
- New method WriteData that handles headers more naturally

Download (login required) CSV plugin v1.2.0

Plugin: Service Platform CPR

Introduction

This plugin provides methods to query people's information from the Service Platform via the service SF1520. More information and integration documents can be found here: https://digitaliseringskataloget.dk/integration/sf1520

Features

Search by CPR Number

Search a person's information by a CPR number.

Method Name: SearchByCprNumber

Input

- CprNumber (required): a string represents a CPR number to search

Output

- Data: The search result from the Service Platform. The result is in the JSON format.
  - CprNumber
  - Name
  - Address
  - ...
- Error: Error object if one or more errors happen while searching

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
     { "CprNumber", "0105141491" }
};
var pluginResult = pluginApi.Invoke("CwCpr ServicePlatform", "SearchByCprNumber", parameters);
var output = pluginResult.GetObject("Data");
```

Search With Conditions

Search people's information by conditions.

Method Name: SearchBy

Input

- Filter (required): a list of simple objects represents the conditions
- Skip (optional): Skip a number of records, default is 0
- Take (optional): Take a number of records, default is 10

Output

- Data: The search result from the Service Platform. The result is in the JSON format.
  - Total number of results that meet the condition
  - List of people's information
- Error: Error object if one or more errors happen while searching

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
    Skip = 1,
    Take = 2,
    Filter = new object[]
    {
        new
        {
            Field = "StreetCode",
            Value = "116",
            Op = "eq"
        },
        new
        {
            Field = "ZipCode",
            Value = "9400",
            Op = "eq"
        },
        new
        {
            Field = "Status",
            Value = new string[] {"01", "03"},
            Op = "in"
        },
        new
        {
            Field = "Birthday",
            Value = DateTime.Parse("2000-04-01"),
            Op = "lte"
        }
    }
};
var pluginResult = pluginApi.Invoke("CwCpr ServicePlatform", "SearchBy", parameters);
var output = pluginResult.GetObject("Data");
```

Installation

Requirements

- Casewhere v2.6 or later

Configuration

To make the plugin work, it must be configured properly.

There are two modes: Stub and Real. Stub mode is used to test with internal stub data, Real mode is used to test with the data from the Service Platform.

Here is the configuration for Stub mode:

```
{
  "Mode" : "Stub",
  "StubDataApiEndPoint" : "[Stub endpoint]", // Endpoint to stub data provider.
  "StubDataApiKey": "123456", // API key for authentication.
}
```

Here is the configuration for Real mode:

```
{
  "Mode" : "Real",
  "MinimumFreshness": "0:15:0",
  "ClientCertificateStoreName" : "My",
  "ClientCertificateStoreLocation" : "CurrentUser",
  "ClientCertificateThumbprint" : "[Certificate thumbprint]",
  "SigningClientCertificateStoreName" : "TrustedPeople",
  "SigningClientCertificateStoreLocation" : "CurrentUser",
  "SigningClientCertificateThumbprint" : "[Certificate thumbprint]",
  "SigningClientCertificateAlias": "[Certificate alias]",
  "ServicePersonEntityId" : "http://cpr.serviceplatformen.dk/service/personbasedataextended/4",
  "ServiceQueryEntityId" : "http://cpr.serviceplatformen.dk/service/query/2",
  "QueryEndpointUrl" : "https://exttest.serviceplatformen.dk/service/CPR/Query/2",
  "PersonEndpointUrl" : "https://exttest.serviceplatformen.dk/service/CPR/PersonBaseDataExtended/4",
  "StsIssuer" : "[STS issuer URL]",
  "StsEndpoint" : "[STS endpoint]",
  "StsCertificateStoreName" : "TrustedPeople",
  "StsCertificateStoreLocation" : "CurrentUser",
  "StsCertificateThumbprint" : "[Certificate thumbprint]",
  "StsCertificateAlias" : "[Certificate alias]",
  "Cvr" : "[CVR of the company that is registered with the Service Platform]"
}
```

Releases

1.0.0 - 10/09/2021

Changelog

- Search by CPR Number
- Search With Conditions

Download (login required): CPR plugin v1.0.0

Roadmap

- Friendly UI to support building queries for **SearchBy** method

Plugin: Service Platform CVR

Introduction

This plugin provides methods to query companies' information from the Service Platform via the service SF1530. The integration document can be downloaded here: https://digitaliseringskataloget.dk/integration/sf1530

Features

Search By CVR Number

Search a company's information by a CVR number.

Method Name: LookupCompanyInfo

Input

- CvrNumber (required): a string represents a CVR number to search

Output

- Data: The search result from the Service Platform. The result is in the JSON format.
  - Cvr number
  - Company's name
  - Address
  - ...
- Error: Error object if one or more errors happen while searching

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
     { "CvrNumber", "27340725" }
};
var pluginResult = pluginApi.Invoke("CwCvr ServicePlatform", "LookupCompanyInfo", parameters);
var output = pluginResult.GetObject("Data");
```

Search By Production Unit Number

Search a production unit by its identification number.

Method Name: LookupPnumber

Input

- PNumber (required): a string represents a production unit number to search

Output

- Data: The search result from the Service Platform. The result is in the JSON format.
  - Production unit's name
  - Address
  - Activity
- Error: Error object if one or more errors happen while searching

Example

```
var pluginApi = ctx.Use<IPluginApi>();
var parameters = new Dictionary<string, object>()
{
     { "PNumber", "1020691286" }
};
var pluginResult = pluginApi.Invoke("CwCvr ServicePlatform", "LookupPnumber", parameters);
var output = pluginResult.GetObject("Data");
```

Installation

Requirements

- Casewhere v2.6 or later

Configuration

To make the plugin work, it must be configured properly.

```
{
  "Mode" : "Real",
  "MinimumFreshness": "0:15:0",
  "ClientCertificateStoreName" : "My",
  "ClientCertificateStoreLocation" : "CurrentUser",
  "ClientCertificateThumbprint" : "[Certificate thumbprint]",
  "SigningClientCertificateStoreName" : "TrustedPeople",
  "SigningClientCertificateStoreLocation" : "CurrentUser",
  "SigningClientCertificateThumbprint" : "[Certificate thumbprint]",
  "SigningClientCertificateAlias": "[Certificate alias]",
  "CvrServiceEntityId" : "http://cvr.serviceplatformen.dk/service/online/2",
  "CvrEndpointUrl" : "https://exttest.serviceplatformen.dk/service/CVR/Online/2",
  "StsIssuer" : "[STS issuer URL]",
  "StsEndpoint" : "[STS endpoint]",
  "StsCertificateStoreName" : "TrustedPeople",
  "StsCertificateStoreLocation" : "CurrentUser",
  "StsCertificateThumbprint" : "[Certificate thumbprint]",
  "StsCertificateAlias" : "[Certificate alias]",
  "Cvr" : "[CVR of the company that is registered with the Service Platform]"
}
```

Releases

1.0.0 - 10/09/2021

Changelog

- Search By CVR Number
- Search By Production Unit Number

Download (login required): CVR plugin v1.1.0

Roadmap

- Search for CVR number by company name, address, start and end time, advertising code, associated P-numbers and company form, etc.
- Lookup related P-numbers of a P-number. Returns a list of information (name and address) of related P-numbers.
- Search for P-number by a combination of search parameters, a list of matching P-numbers is returned.

Plugin: SMTP

Introduction

SMTP plugin allows you to send email via SMTP, which means you can easily integrate it into an existing SMTP server by minor modifications in the SMTP plugin configuration.

Features

Send an email

There is one method here to allow the user to send an email from Casewhere to the external system.

Method Name: SendEmail

Input

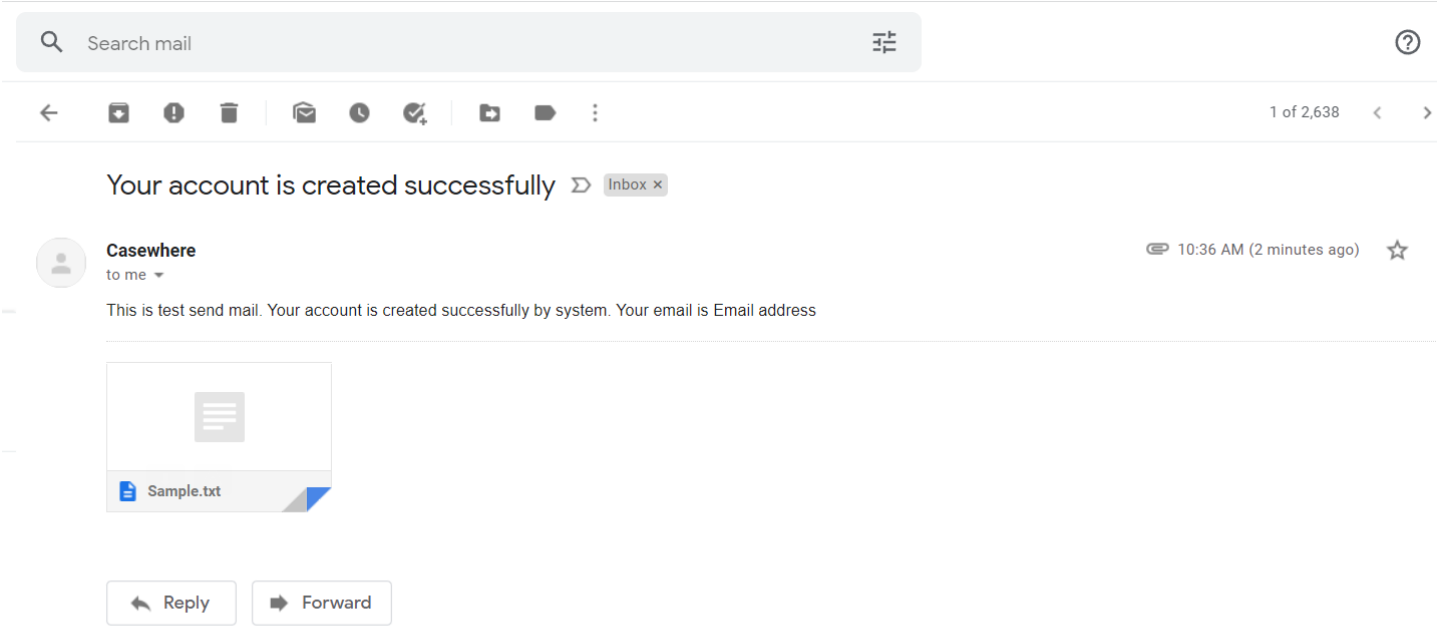
- subject: The email's subject. This field is required.
- body: The email's content, it can be plain text or text with HTML format. This field is required.
- to: The list of recipient email addresses in the serialized JSON format. This field is required.
- cc: The list of CC email addresses in the serialized JSON format. This field is optional.
- bcc: The list of BCC email addresses in the serialized JSON format. This field is optional.
- isHtml: Field value is either true or false (false is default), it supports display HTML in content email. This field is optional.
  - If `true`, the email's content displays with HTML format
  - If `false`, the email's content doesn't display with HTML format although it contains HTML
- attachments: the serialized JSON string of attachments. This field is optional.

Follow the below sample:

```
var bytes = Encoding.ASCII.GetBytes("text");
var attachments = new List<Dictionary<string, string>>();
attachments.Add(new Dictionary<string, string>
{
    { "Name", "text.txt" },
    { "Content", Convert.ToBase64String(bytes, 0, bytes.Length)  }
});

var mailParameters = new Dictionary<string, string>()
{
    { "subject", "Subject email" },
    { "body", "Content email" },
    { "isHtml", true },
    { "to", JsonConvert.SerializeObject(new List<string> { "casewhere@globeteam.com" }) },
    { "cc", JsonConvert.SerializeObject(new List<string> { "casewhere@globeteam.com" }) },
    { "bcc", JsonConvert.SerializeObject(new List<string> { "casewhere@globeteam.com" }) },
    { "attachments", JsonConvert.SerializeObject(attachments) }
};
```

This is sample email:



Use the **Invoke** method in IPluginApi to call **SendEmail** feature:

```
IPluginApi.Invoke(string pluginName, string methodName, IDictionary<string, Object> parameters)
```

Output

The method returns data with two fields `IsSuccess`, `Error`.

Example

```
var pluginApi = ctx.Use<IPluginApi>();

// Prepare parameters
var parameters = new Dictionary<string, string>()
{
    { "subject", "Subject email" },
    { "body", "Content email" },
    { "isHtml", "true" },
    { "to", JsonConvert.SerializeObject(new List<string> { "casewhere@globeteam.com" }) },
    { "cc", JsonConvert.SerializeObject(new List<string> { "casewhere@globeteam.com" }) },
    { "bcc", JsonConvert.SerializeObject(new List<string> { "casewhere@globeteam.com" }) }
};

// Invoke plugin
var pluginResult = pluginApi.Invoke("CWMail", "SendEmail", parameters);

var isSuccess = pluginResult.Value<bool>("IsSuccess");
```

Installation

Requirements

- Casewhere v2.6.0 or later. ### Configuration
- Username: SMTP username.
- Password: SMTP password.
- EmailAddress: The sender's email address.
- DisplayName: The sender's name.

- SMTP: SMTP server name or IP address.
- Port: SMTP transactions's port
- EnableSsl: option to enable Secure Sockets Layer (SSL) in order to encrypt the connection. Value is `true` or `false`.

Example

```
{
    "Username": "Username",
    "Password": "Password",
    "EmailAddress": "no-reply@application.com",
    "DisplayName": "Sample application",
    "SMTP": "application.com",
    "Port": 587,
    "EnableSsl": true
}
```

Releases

1.0.0 - 23/03/2022

Changelog

- Send an email.

Download (login required) SMTP plugin v1.0.1

Roadmap

- Send test email
- Previewing email
- Email mapping

Overview

Introduction

This template is an internal template that can be used to generate hybrid mobile apps for you Casewhere project.

Use it as a starting point and adding more features according to your specific needs.

How to use the template - video guideline (may requires login)

Features

Getting started

- Overview
- Environment setup
- Project structure
- Before you start
- Start a project
- Generate new feature
- Custom URL scheme

Components

- Authentication
- Biometric authentication
- Multiple language support
- REST API
- List layout
- Grid layout
- Notification
- Roles & rights
- form.io integration

Tutorials

- Extracting an APK file

Release

1.0.0 - 30.06.2022

Download (login required): MobileScaffold

Changelog

- Project overview
- Authentication with Safewhere Identify
- Rest API with Casewhere web trigger

1.1.0 - 15.07.2022

Changelog

- Biometric authentication
- Multiple languages
- Better security with native browser

1.2.0 - 30.07.2022

Changelog

- Roles & rights
- Support iOS
- Notification

1.3.0 - 05.08.2022

Changelog

- Theme & layout
- form.io integration

Roadmap

- Extend component list for more use cases

Environment setup

The template is written in Angular, and built on top of Ionic Framework.

To get started with the template, you will need a Node & npm environment.

For code editor, Visual Studio Code is recommended.

Install Node.js

Node.js is required for interacting with the Ionic ecosystem. Download the LTS version here.

You can skip this section if you already have a working Node.js.

To verify if node.js was installed, open a new terminal window and run:

```
$ node --version
$ npm --version
```

Git

Make sure you have `git` installed. Otherwise, follow these instructions.

Ionic VS code extension

Though not required, this extension will help you perform various functions that are common to developing an app.

Install the extension

Read more here

Project structure

Main folders

You will notice the project has a base structure, that includes all the building blocks that you need to continue developing your project.

Android

This folder includes all the files required to build an Android app using the following configuration

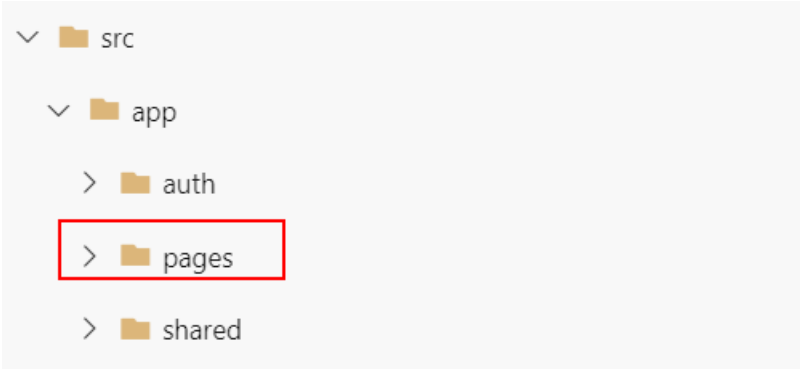
- File `android/app/main/AndroidManifest.xml` has been configured to support a custom URL scheme for the authentication component
- File `android/gradle.properties` include the simulator configuration

src/app/pages

The `src` folder is where you write your code.

The `app` folder contains the files to bootstrap the app and the main structure we will build the app upon. Inside the `app`, there are several important folders:

- `pages`: each project's feature will be stored in an individual folder inside pages



src/app/shared

Folder `shared` stores all shared service/model/ that can be used globally across the project

src/assests

Folder `assests` places all fonts and images required for the app.

src/environments

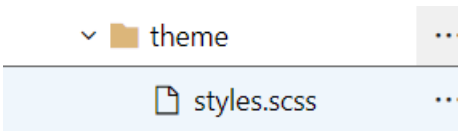
Folder `environments` is where we store all the variables that are similar but have different environments.

Read more here

src/theme

The folder `theme` stores all Sass files (styles, mixins..) for customizing the app's styles.

The file `style.scss` defines the global CSS rules. Each folder will also have its own SCSS files to define specific rules.



src/app/auth

- **auth-guard.service.ts**: a guard deciding if a route can be activated
- **auth-http.service.ts**: HTTP service for calling Restful APIs
- **auth-module.config.ts**: configure HttpInterceptor (Read more here)
- **auth-storage.factory.ts**: custom storage to store access_token, refresh_token and other tokens
- **auth.factory.ts**: contain auth configurations such as issuer, clientId, redirectUri
- **auth.module.ts**: configure the injector, the compiler and help organize related things together
- **auth.service.ts**: main service to oauth2 login flow, logout, validate token

How to contribute

Step 1. Create a branch from the **master** branch (Link to template)

- Branch name format must be "#id_name_in_snake_case", i.e. #12345_feature_name

Step 2. Add new features, coding, and editing as you wish

Step 3. Wrap up code

- Remove unused declared variables
- Remove Console.log that you used for debugging
- Code clean checkup
- Use `$ ng lint` to check es2020 standard

Step 4. Submit a PR and ask for review

Step 5. Resolve reviewers' comments

- Coding styles follow this guide

Before you start

Install Ionic

Navigate to `MobileScaffold` folder, then use this command line to install Ionic

```
$ npm install -g @ionic/cli
```

Install prerequisite packages

```
$ npm install
```

This command will install

- all packages defined in `package.json`
- all dependencies packages in `package-lock.json`

Prerequisite knowledge

An understanding of the following concepts will be required:

- Angular
- Ionic

Start a project

Build project

```
$ ionic build
```

This command will compile web assets and prepare them for deployment.

Read more

Run project

```
$ ionic serve
```

When you run this command, the code inside of `src/` is transpired into the correct Javascript version that the browser understands.

Read more

Custom URL Scheme

Introduction

Custom URL schemes provide a way to reference inside your app. Users can open a custom URL in the browser or tap on any link. Other apps can also trigger your app to launch with specific context data; for example, a photo library app might display a specified image.

Configuration

Android

Assume we have generated the **android** folder, if not, see the link here for setup.

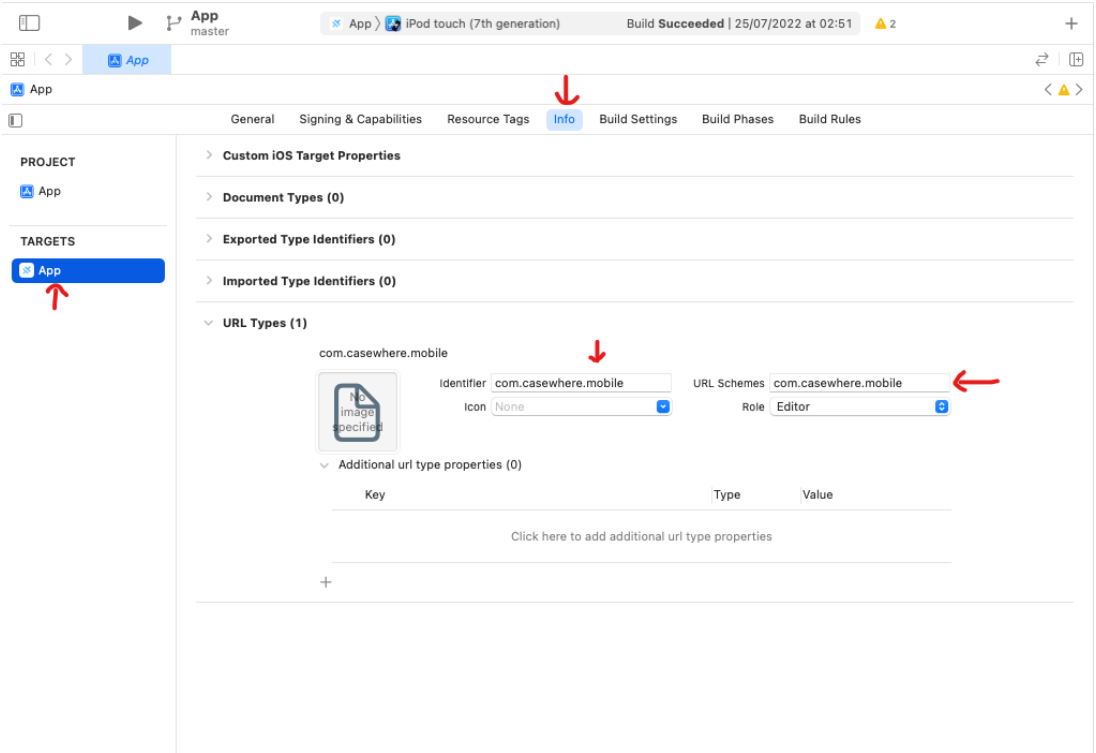
- Step 1: Open the **AndroidManifest.xml** file, in the **android/app/src/main** folder.
- Step 2: Add an intent filter for incoming links. Inside `activity` tag, add the following lines of code:

```
<intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <!-- Accepts URIs that begin with "com.casewhere.mobile://” -->
    <data android:scheme="com.casewhere.mobile"/>
</intent-filter>
```

IOS

Assume we have generated the **ios** folder, if not, see the link in ionic or capacitor for setting and running.

- Step 1: Open the project by Xcode then select It.
- Step 2: Select tab Info.
- Step 3: Go to the URL Types and click on ‘+’.
- Step 4: Enter `com.casewhere.mobile` to Identifier and URL Schemes.



Reference

- Capacitor IOS Documentation
- Android Deep Links
- IOS Custom URL Scheme

Overview

Introduction

The template uses OAuth/OIDC for authentication.

You can select an identity provider of your choice.

In this guideline we will use Safewhere Identify to demonstrate template's features

Prerequisite

- Safewhere Identify version 5.12 and above
- Abstract knowledge of OAuth/OIDC, authentication with PKCE flow

Config

Introduction

This article shows how to config login and logout functions.

Guide

- Step 1. Go to `auth.factory.ts`

Update `Issuer` and `ClientID`

Example:

```
issuer: 'https://login3-demo.casewhere.com/runtime/oauth2',

clientId: 'xj2oKyIdW4JUVT9ic6a5', // The "Auth Code + PKCE" client
```

- Step 2. Go to `authFactory` function

Check callback URLs in the `redirectUri`.

Example setup for mobile and browser

For mobile:

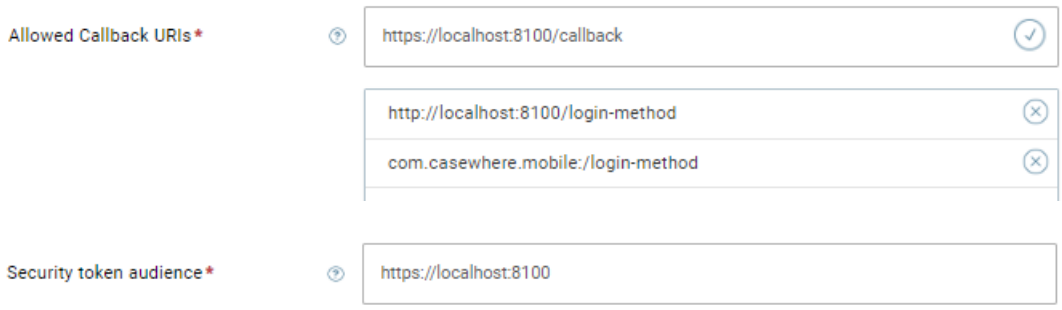
```
let redirectUri = 'com.casewhere.mobile:/login-method';
let silentRefreshRedirectUri = 'com.casewhere.mobile:/';
let logoutUrl = 'com.casewhere.mobile:/login-method';
```

For browser:

```
if (!platform.is('cordova')) {
    redirectUri = window.location.origin + '/login-method';
    silentRefreshRedirectUri = window.location.origin + '/silent-refresh';
    logoutUrl = window.location.origin + '/login-method';
}
```

- Step 3. Making sure the `redirectUri` and the `logoutUrl` match with your identify provider settings.

For example, below is the settings we made in Safewhere Identify.

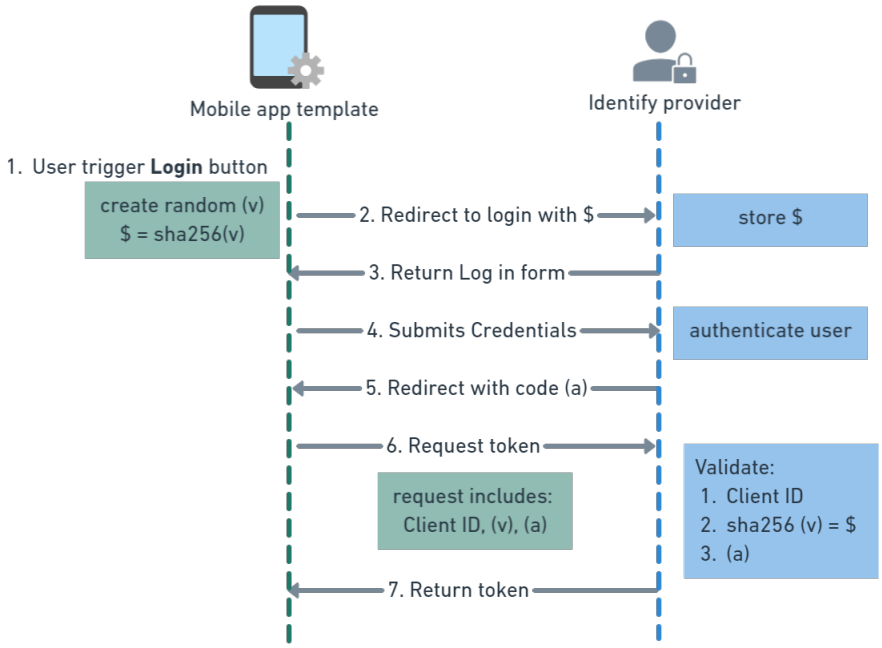


- Step 4. Build and run the project again for testing

PKCE Authentication flow explanation

This template adopted PKCE flow, which is a new, more secure authorization flow (based on the OAuth 2.0 spec) that was designed to improve the security of mobile apps.

Authentication flow is handled by the angular-oauth2-oidc library, and can be briefly summarized as following:



1. User presses login button. The app create a random value `v`. Then `v` will be hashed into a value called `$`
2. The app redirect to the login page with `$`
3. Identify provider returns a login form
4. The app submits the credentials received from users and send to Identify, from here goes on Identify provider will handle authenticate the users.
5. Identify redirect with code `a`
6. App send request, includes requests for Client Id, `v`, and `a`
7. Identify validate with Client ID, check if sha256(`v`) = `$`

Set up identity provider

Safewhere Identify

Safewhere Identify (IdP) is a trusted authentication provider. The service can handle projects with millions of users and varying levels of complexity while maintaining the highest level of security.

We will use IdP as the identify provider in this demo.

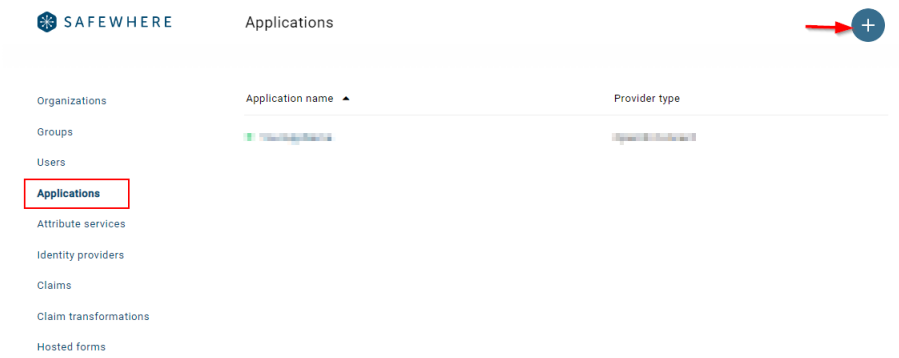
Config IdP

Create new application in IDP

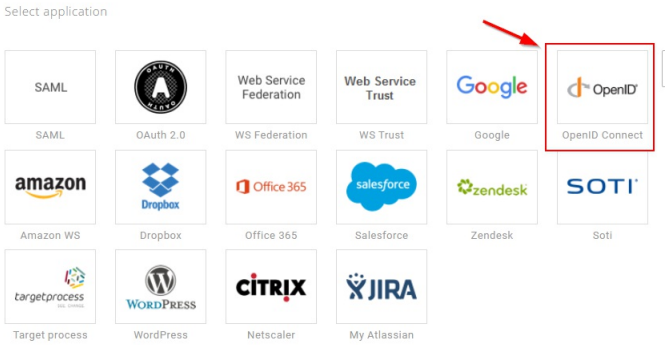
- Step 1. Login to IdP product portal

The template only supports IdP version 5.12 or above

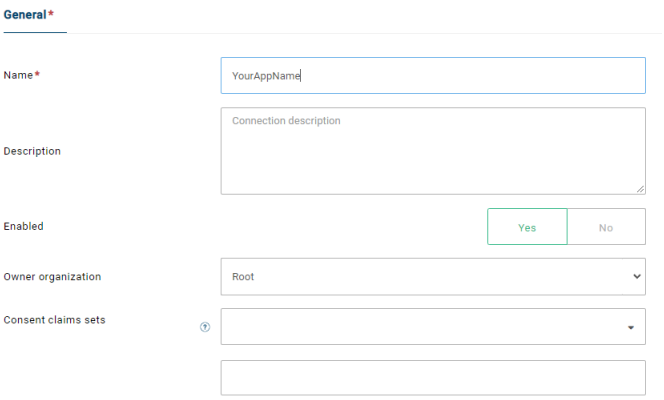
- Step 2. Go to **Applications**, click new application button



Select **OpenID Connect** as application type



- Step 3. Input your application name, click **Save**



Navigate to **Connection** tab

- Step 4. Now you will see a list of necessary settings that will be useful for your development process

Client id: a random, unique sequence

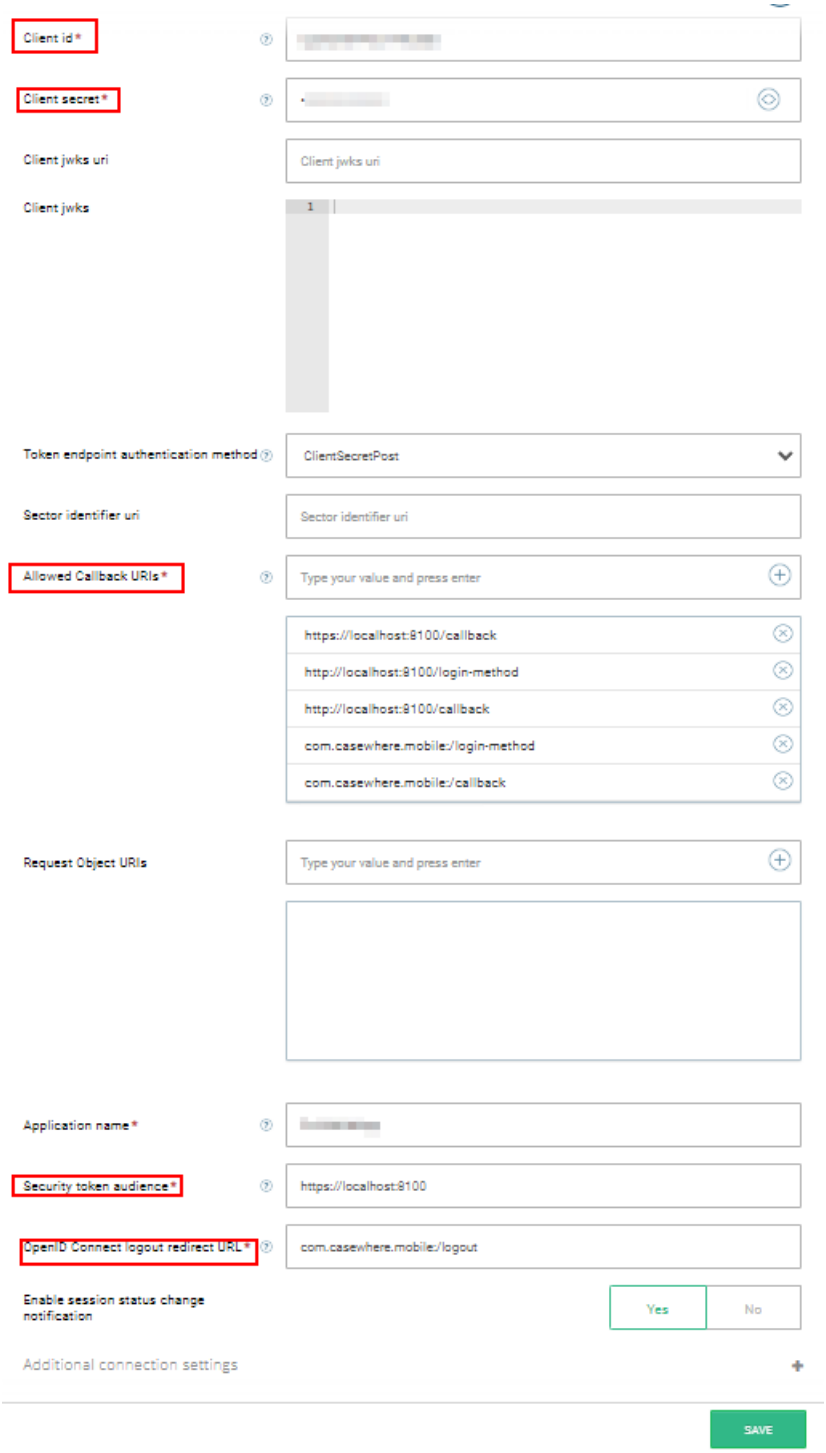
Client secret: a random, unique, secret sequence

Allowed Callback URLs: list of all redirect URL after authentication successfully

Security token audience: set the audience field of tokens which are issued for the application

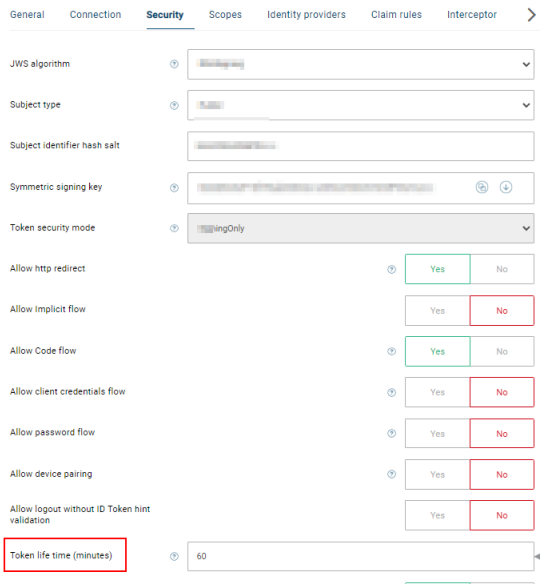
OpenID Connect logout redirect URL: redirect URL after successful SLO

The image below show an example of the demo app settings



Config token life time in IDP

- Step 1. Navigate to **Security** tab
- Step 2. Input the number (as minutes) to **Token life time**



> Tip: if you encounter any issues when working with Identify, check out Safewhere document site or contact Safewhere*Identify team directly for support

Demo

For a feature showcase, watch demo video - Authentication and REST API (may requires login).

Demo with IDP

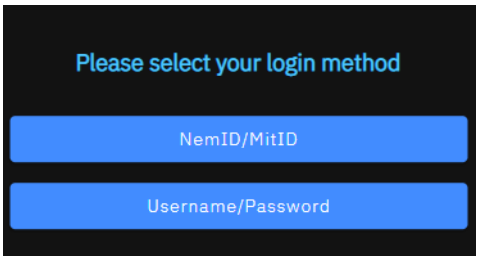
Example demo for Casewhere Mobile app template with Safewhere*Identify

Login

Clicking the logout button should terminate the session and redirect the user to the login page

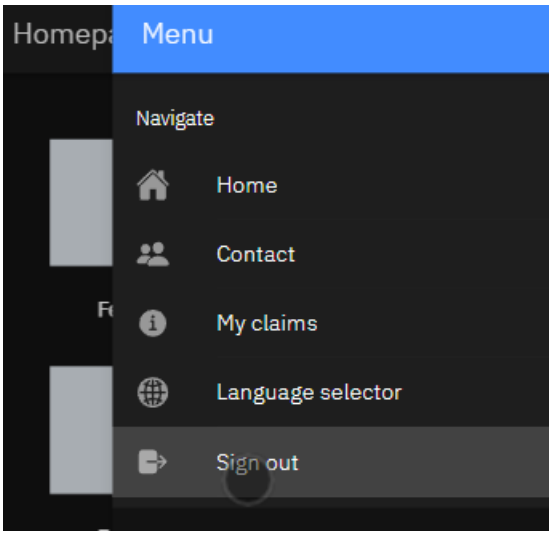
- Username and password
- NemID
- MitID
- Social media

  For adding or removing a method, config in Identify



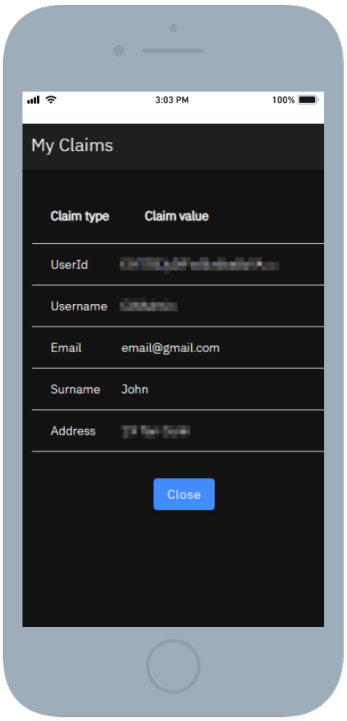
Logout

Log out function will sign out user immediately. User will have to log in again to use the app



My claim page

List of the claims issued by Identify


3:03 PM
100%
My Claims
Claim type
Claim value
UserId
Username
Email
email@gmail.com
Surname
John
Address
Close

Biometric Authentication

Introduction

This post presents a way to set up Safewhere Identify to turn on Biometric method. We will also include a short guideline for end user to setting up and change biometric infos.

Features

- Enable Biometric in Identify
- Log in to the mobile app with biometric infos

Create biometric login method

Configuration in Identity providers

Biometric authentication

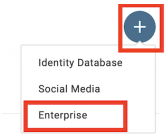
This section will show you how to create a method for users to use their biometric infos to log in to the mobile app

- Step 1. Go to Safewhere tenant application (i.e. https://login-example.casewhere.com/adminv2)
- Step 2. Choose Username & password



- Step 3. Log in with admin account
- Step 4. Add new identity providers for biometric authentication

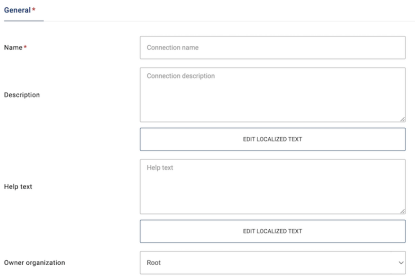
Navigate to Identity providers. Click **(+)** and then choose **Enterprise**



- Step 5. Choose One Time Password



The form will be shown as follows

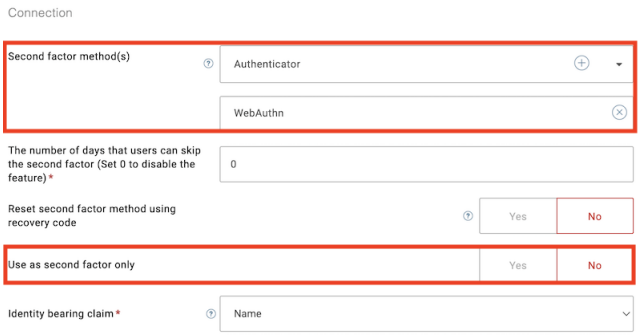


You can choose any name for your biometric login method

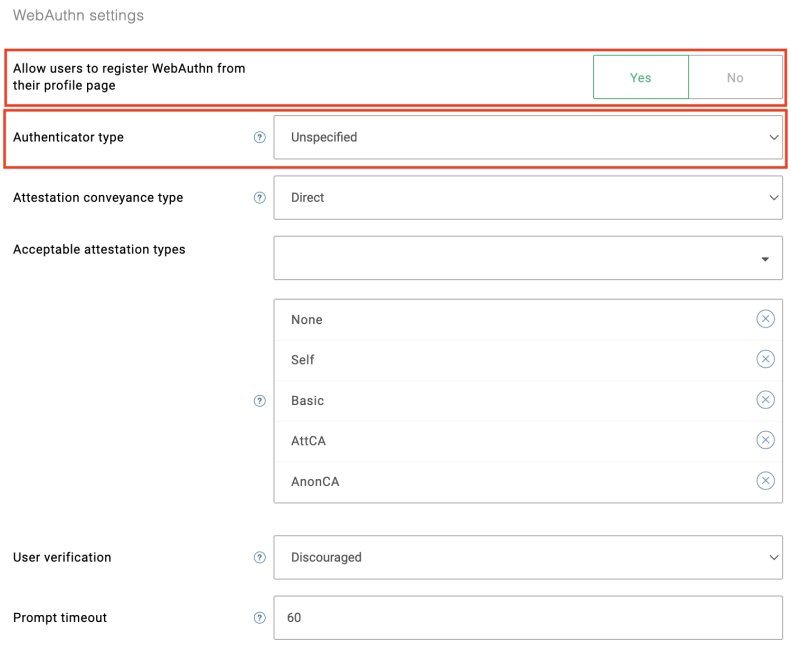
But in this guide, we set name equal to **Biometric Authentication<index>** (index avoid duplicated) and click save

- Step 6. Identity provider configuration

Open Connection tab then configure as below
Connection section



WebAuthn settings



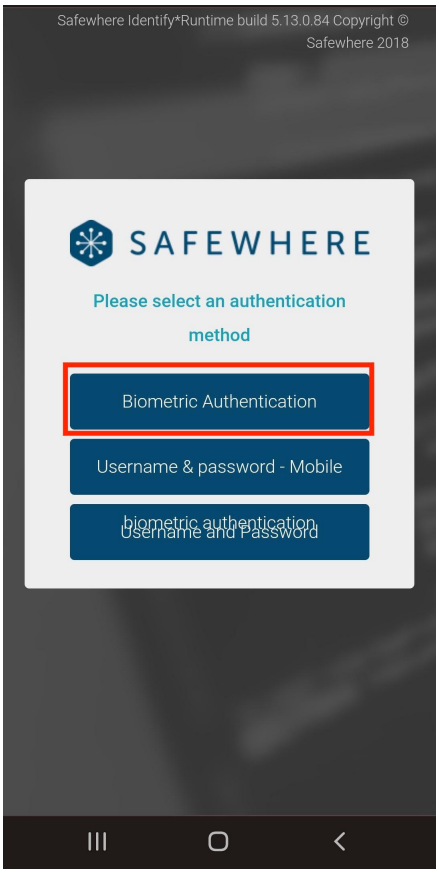
- Step 7. Activate biometric authentication login method for mobile app

Navigate to Application page -> open CwMobileApp application -> Identity providers tab
Find your identity provider and switch it status to active



- Step 8. Check the result

Open app and choose Login
Now you will see your identity configuration will be displayed in the screen

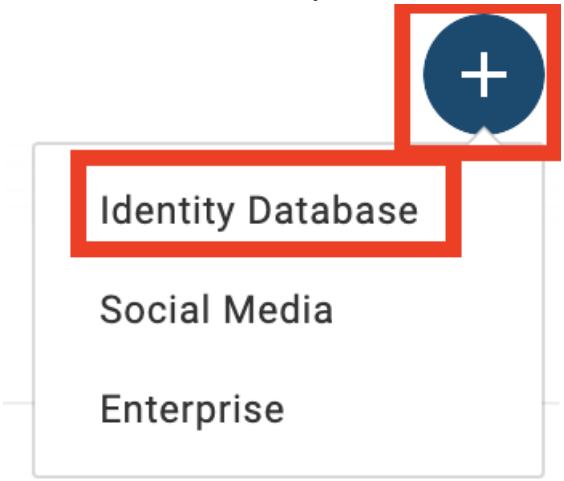


Biometric register on mobile app

This section will show you how to create a biometric register on mobile app

- Step 1. Create an identity provider for mobile app biometric register

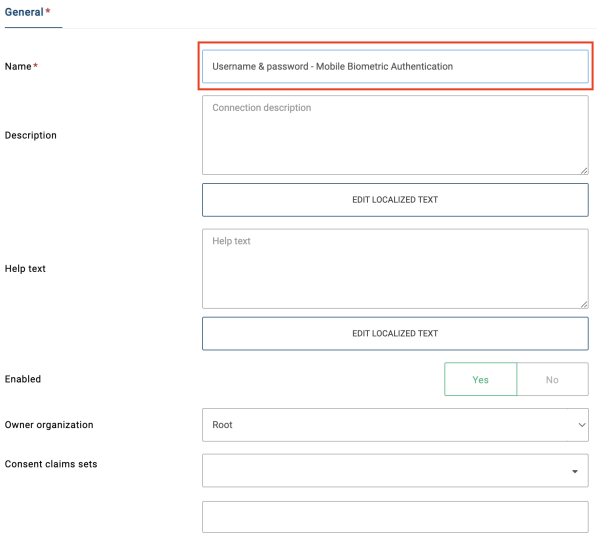
Navigate to Identity providers
Click (+) then choose Identity Database



- Step 2. Select **Username & Password** as a provider



- Step 3. The form will be shown as follows

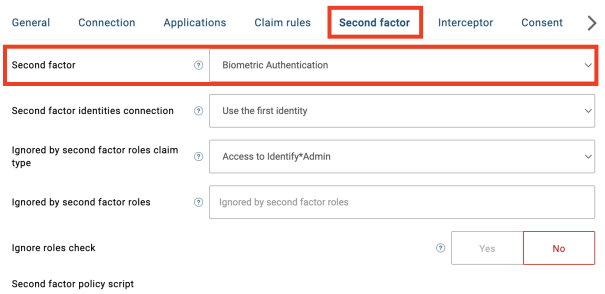


You can choose any name for your biometric login method

But in this guide, we set name equal to **Username & password - Mobile Biometric Authentication<index>** (index avoid duplicated) and click save

- Step 4. Set up second factor

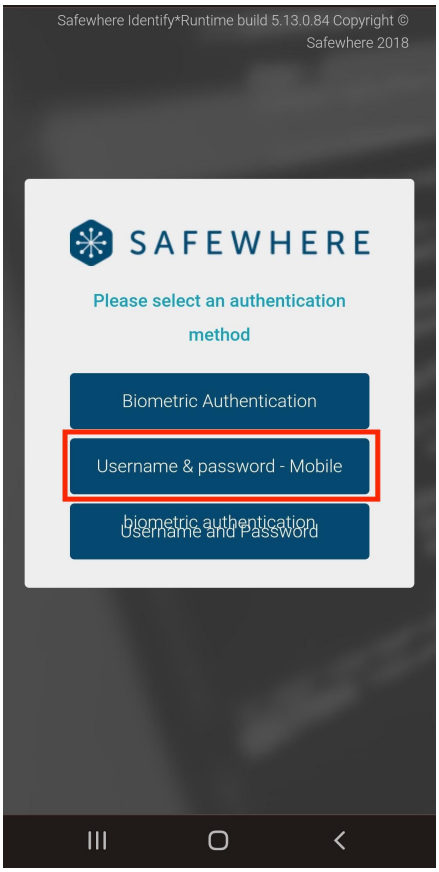
Navigate to **Second factor** tab



Choose **Biometric Authentication** (or your biometruc authentication name) as **Second factor**

- Step 5. Check the result

Open app and choose Login
Now you will see your identity configuration will be displayed in the screen



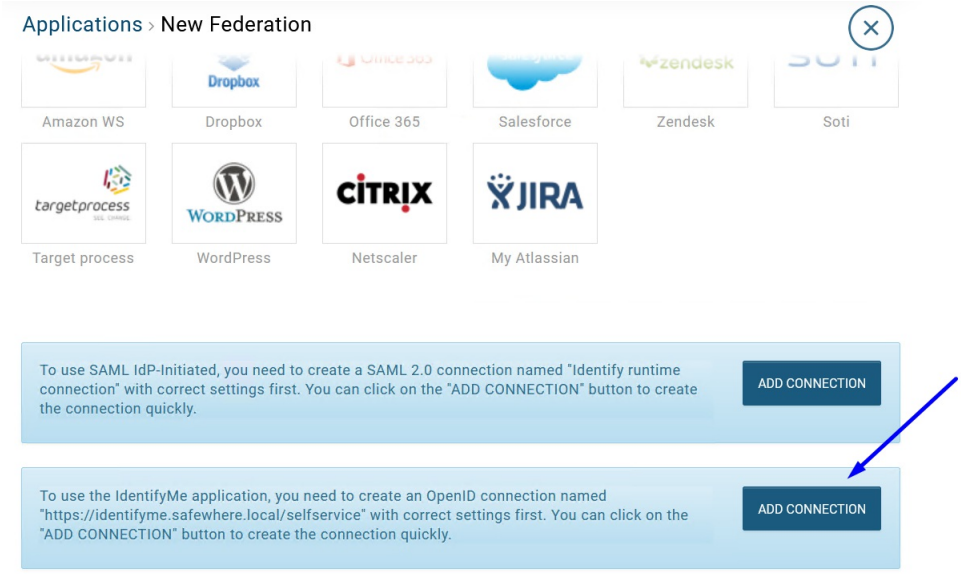
Conclusion

Now we can check here for how to use biometric login method

Setting up Identify Me

Configure in Identify Me

- Step 1. Go to Applications -> add selfservice connection



IdentifyMe settings in Settings page will be activated

- Step 2. Go to Safewhere tenant application (i.e. https://login.example.casewhere.com/adminv2)

Navigate to Settings

SAFEWHERE

Organizations

Groups

Users

Applications

Attribute services

Identity providers

Claims

Claim transformations

Hosted forms

Messaging

Clients

Script library

Integrations

Analytics

Settings

Logging

System infomation

- Step 3. Choose IdentifyMe setting and modify as follows



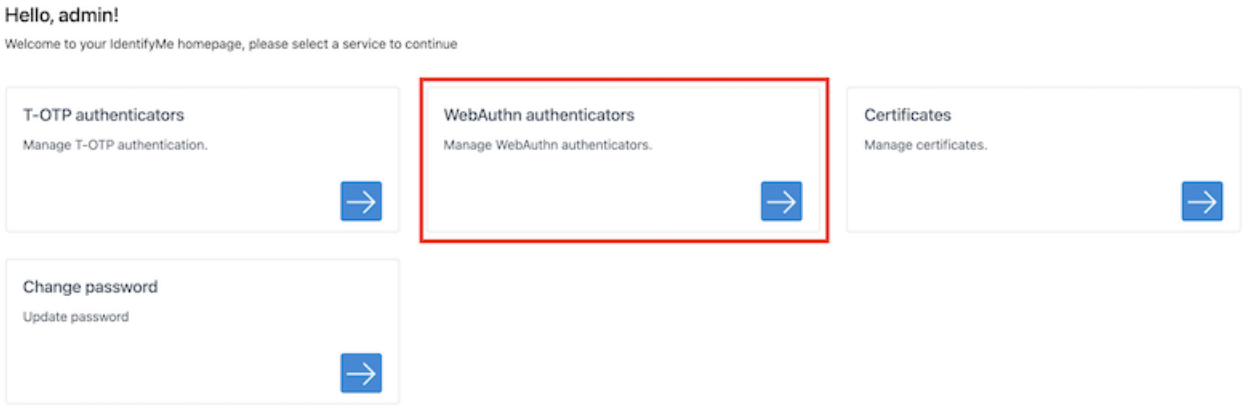
User guideline

Set up biometric info

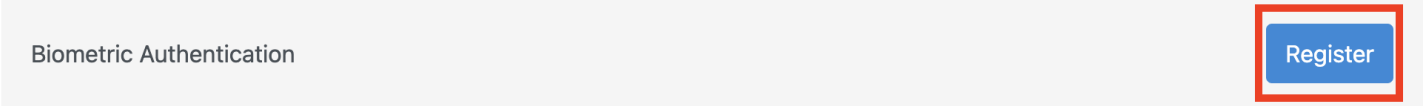
Direct in browser

This section will show you how to set up your biometric infos to log in to the app

- Step 1. Go to Safewhere tenant application (i.e. https://login.example.casewhere.com/selfservice)
- Step 2. Choose WebAuthn authenticators

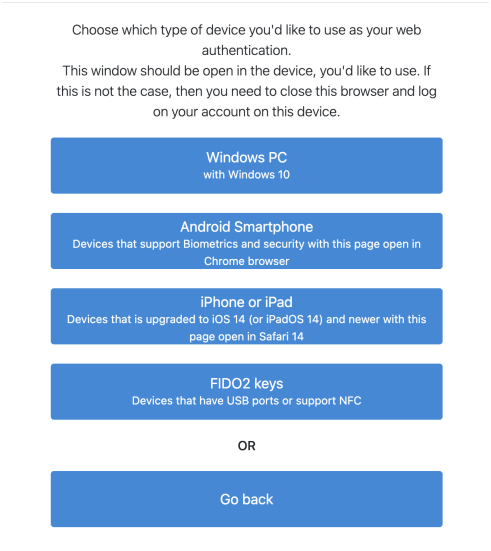


- Step 3. Choose your identity, click Register



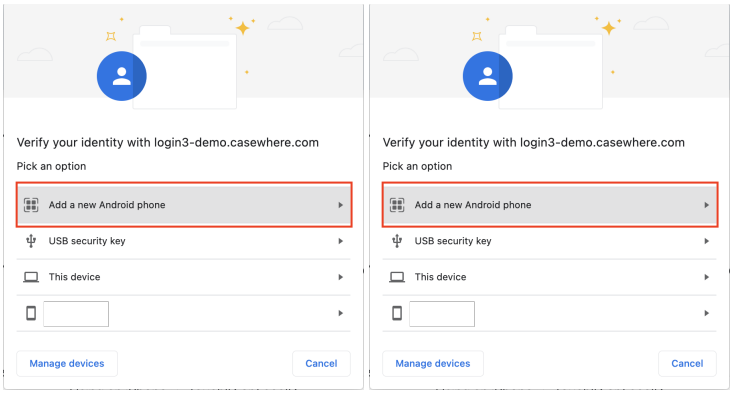
- Step 4. Set up biometric on mobile phone

If you did not set up your biometric on your phone yet, choose **Receive help setting up authentication** and follow the instructions

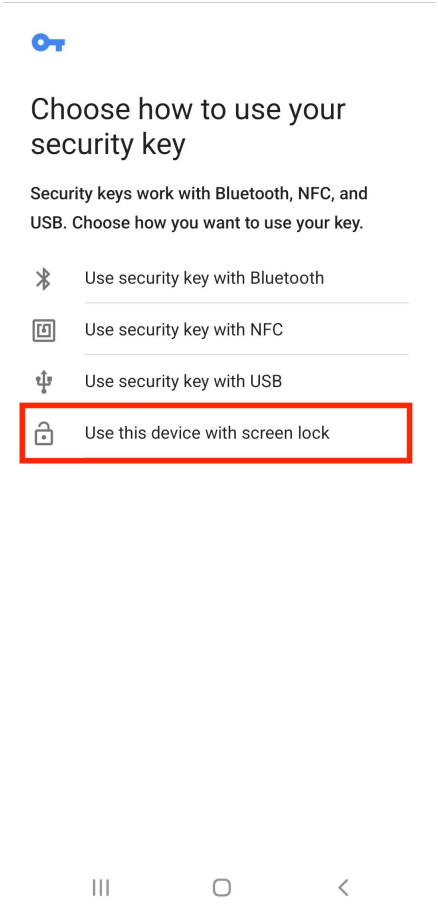


- Step 5. If you already setting up biometric authentication on your phone or complete Step 4, click Register
- Step 6. Choose mobile device for your account

Choose Add a new android phone
It will give you a QR code for you to scan it with your phone



If you use mobile browser



Use your biometric info on your phone to verify

- Step 7. Login

Open app, click Login button
A login form will open

- Step 8. Choose login method

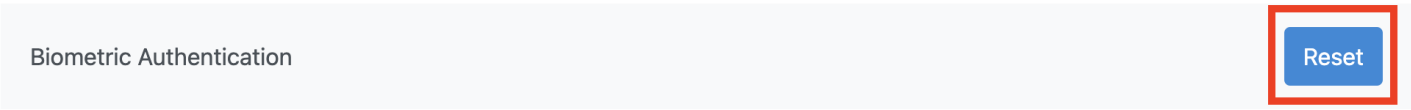
Choose **biometric** (the login method can be different depends on project)

- Step 9. Use your biometric info to log in

Reset Biometric authenticators

- Step 1. Go to Safewhere identify me application (i.e https://login3-demo.casewhere.com/selfservice)
- Step 2. Choose WebAuthn authenticators

Instead of Register, now you will see the Reset button



- Step 3. Click Reset
- Step 4. Register again

Now you have to register again with new biometric authenticator

Follow this section

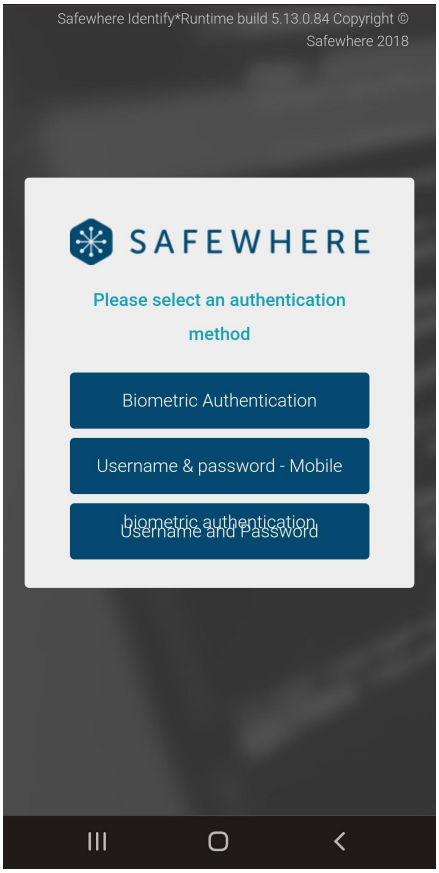
In-direct through mobile app

This section will show you

1. How to use your biometric authenticator to log in to the mobile app
2. Set up your biometric infos for your mobile app (or Attach your biometric infos into your account)

- Step 1. Open the mobile app

Choose login



We will see there are 3 login methods

1. Biometric Authentication
    - You can only use this when you finish setting up biometric in Identify me
2. Username & password - Mobile Biometric Authentication

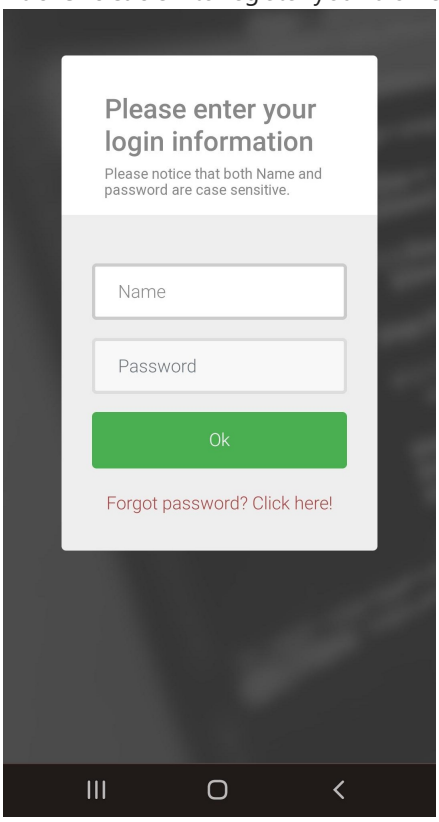
    - You can use this to set up biometric for your account if your account does not have biometric info yet
    - If your account already has biometric info, biometric info will act as 2nd factor
3. Username & password
    - The standard way to log in

- Step 2. Log in with **Biometric Authentication**

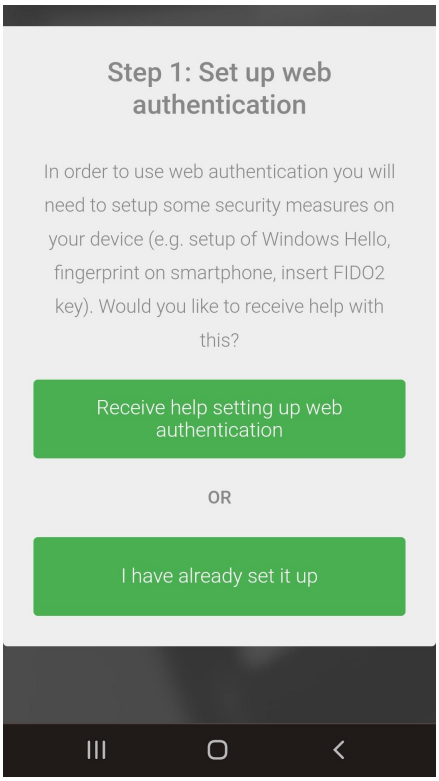
If you want to log in with the first method - **Biometric Authentication**, you need to set up your biometric info. Read here

- Step 3. Log in with **Username & password - Mobile Biometric Authentication** (using biometric as second factor)

If you did not set up biometric info yet, you can use the second method - **Username & password - Mobile Biometric Authentication** to register your biometric info



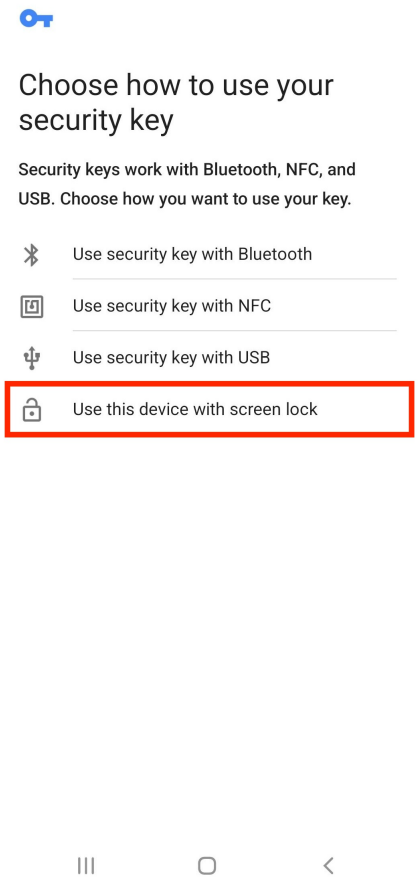
After logging in, you will see



If you did not set up your biometric on your phone yet, choose Receive help setting up authentication and follow the instructions

- Step 4. If you already setting up biometric authentication on your phone or complete Step 3, click Register

The following screen will be shown



Choose **Use this device with screen lock** and then use your biometric info to log in

- Step 5. Register biometric info successfully

The next time you use the app, you can use **Biometric Authentication** method to log in

Limitation

- This is not a native feature
- Does not support android face recognition well
- Does not support iOS yet

Text resources for mobile app

Introduction

This mobile template is developed for users who come from different nations and speak different languages

This tutorial will show you how to create and modify your language resources (enable multiple languages for mobile app)

Prerequisite

Click here to know how to create text resource on Casewhere

How multiple languages work on mobile

- Step 1

The mobile app will load offline text resources files to display

- Step 2

The mobile app will take text resources from Casewhere and then overwrite the data from offline text resources files

- Step 3

If the mobile does not have internet connection, the mobile app will take offline text resources to display

Setting up text resources key

The syntax

First we need to fetch all the text resources keys in our mobile app

The syntax

```
{{'key' | translate}}
```

We can find all of the keys in html files

We will take contact-list page in previous section as example

Navigate to src/app/pages/contact/contact-list/contact-list.page.html

These are text resources key we extract from contact-list page

```
{{'cw_mb_contact_list' | translate}}
{{'cw_mb_contact_add' | translate}}
{{'cw_mb_common_reload' | translate}}
```

Get translation from Casewhere

Click here to know how to set up text resource on Casewhere

Go to Casewhere Admin, Text resources

We need to add 3 new text resources

Add 3 new text resources keys of contact-list page and the set the value for them

KEY	DA	EN	FR
cw_mb_contact_list	[DK] Contact	Contact	[FR] Contact
cw_mb_contact_add	[DK] Add	Add	[FR] Add
cw_mb_common_reload	[DK] Reload	Reload	[FR] Reload

Support offline mode

Update existed languages

- Step 1

Navigate to src/assets/i18n, find your language text resources (i.e da.json for Danish, en.json for English)

- Step 2

Add new text resource key

```
{
    "key_1": "One",
    "key_2": "Two",
    "key_3": "Three"
    ....,
    "key_n": "n"
}
```

If the text resource key already existed, please find the key in the json file and update it

Add new language

- Step 1

We will take French as a new language to add

Navigate to src/assets/i18n, create a new JSON file fr.json (fr is shortcut for French)

This file will contain all keys and its value for French

- Step 2

Add text resource keys to `fr.json`

```
{
    "cw_mb_contact_list" : "[FR] Contact",
    "cw_mb_contact_add"  : "[FR] Add",
    "cw_mb_common_reload": "[FR] Reload"
}
```

Enable multiple languages

This section will answer the question: "How can we apply text resources into the mobile app?"

1. In every page, navigate to **your-page**.module.ts, import TranslateModule

```
import { TranslateModule } from '@ngx-translate/core';
```

In the same **your-page**.module.ts, import `TranslateModule.forChild()`

```
@NgModule({
    imports: [
        ...
        TranslateModule.forChild()
    ],
    ...
})
```

1. Navigate to src -> app -> app.component.ts

- Import necessary packages

```
import { TranslateService } from '@ngx-translate/core';
import { LanguageService } from './pages/language/language.service';
```

- Add these lines to constructor

```
constructor(
    ...
    public translate: TranslateService,
    private langService: LanguageService
    ) {
        ...
        this.setLanguage();
    }`
```

`setLanguage()` responsible for applying text resources to our mobile app

```
setLanguage() {
    var appLang = this.langService.getAppLang();
    this.langService.getTextResource({ language: appLang }).subscribe(
        (data) => {
            for (const property in data) {
                this.translate.set(property, data[property]);
            }
    });

    this.translate.addLangs(['en', 'da', 'vi']);
    this.translate.use('da');
    this.translate.use('en');
    this.translate.use('vi');
    setTimeout(() => {
        this.translate.use(appLang);
    }, 200)
}
```

Change languages on mobile app

We need to create a page for user to change language

Read [here](../../../../products/template/mobile-app/getting-started/generate new feature) to know how to create a page

language.page.html

```
<ion-header>
    <ion-toolbar>
        <ion-title>{{'cw_mb_language' | translate}}</ion-title>
    </ion-toolbar>
</ion-header>

<ion-content class="ion-padding">
    <ion-text>
        <h5>{{'cw_mb_select_default_language' | translate}}</h5>
    </ion-text>

    <ion-list>
        <ion-item (click)="selectLang('en')" button="true" [fill]="lang == 'en' ? 'solid' : ''">
            <ion-avatar slot="start">
                <img src="/assets/images/language-img.png" />
            </ion-avatar>
            <ion-label>
                <h2>{{'cw_mb_english' | translate}}</h2>
            </ion-label>
        </ion-item>
        <ion-item (click)="selectLang('da')" button="true" [fill]="lang == 'da' ? 'solid' : ''">
            <ion-avatar slot="start">
                <img src="/assets/images/language-img.png" />
            </ion-avatar>
            <ion-label>
                <h2>{{ 'cw_mb_danish' | translate}}</h2>
            </ion-label>
        </ion-item>
    </ion-list>

    <ion-row class="ion-justify-content-around ion-padding-vertical">
        <ion-button size="medium" type="button" class="text-transform-unset" (click)="saveLang()">
            {{'cw_mb_language_save_and_colse' | translate}}
        </ion-button>
    </ion-row>
</ion-content>
```

language.page.ts

```
import { Component, OnInit } from '@angular/core';
import { NavController } from '@ionic/angular';
import { TranslateService } from '@ngx-translate/core';
import { LanguageService } from './language.service';

@Component({
    selector: 'app-language',
    templateUrl: './language.page.html',
    styleUrls: ['./language.page.scss'],
})
export class LanguagePage implements OnInit {

    lang: string;
    constructor(public translate: TranslateService, private navController: NavController, private
langService: LanguageService) {
        this.lang = this.langService.getAppLang() || translate.currentLang;
    }

    ngOnInit() {
    }

    selectLang(lang) {
        this.lang = lang;
    }

    saveLang() {
        this.translate.use(this.lang).subscribe(() => {
            this.updateLanguage();
            this.navController.back();
        });
    }

    updateLanguage() {
        this.langService.changeAppLang(this.lang);
        this.langService.getTextResource({ language: this.lang })
            .subscribe(
                (data) => {
                    for (const property in data) {
                        this.translate.set(property, data[property]);
                    }
                });
    }
}
```

Create a service for language page `language.service.ts`

```
import { Injectable } from '@angular/core';
import { AuthHttpService } from '../../auth/auth-http.service';
import { Observable, throwError } from 'rxjs';
import { catchError, map } from 'rxjs/operators';
import { environment } from 'src/environments/environment';
const LANG = 'LANG';

@Injectable({
    providedIn: 'root'
})
export class LanguageService {

    constructor(private autHttp: AuthHttpService) { }

    getTextResource(params?: any): Observable<any> {
        return this.autHttp.get(`${environment.API_URL}/cw-mobile-text-resources-get`, params)
            .pipe(
                map(res => {
                    return res;
                }),
                catchError((err, cautch) => { return throwError('Something bad happened; please try again
later.') })
            );
    }

    getAppLang() {
        return localStorage.getItem(LANG) || 'en';
    }

    changeAppLang(value: string) {
        localStorage.setItem(LANG, value);
    }
}
```

If you wanna add more languages options, avigate to `src/app/pages/language/language.page.html`
Add new language as an item to the existed list

```
<ion-item (click)="selectLang('vi')" button="true" [fill]="lang == 'vi' ? 'solid' : ''">
    <ion-avatar slot="start">
        <img src="/assets/images/language-img.png" />
    </ion-avatar>
    <ion-label>
        <h2>{{ 'cw_mb_vi' | translate}}</h2>
    </ion-label>
</ion-item>
```

Text resources for mobile app

Introduction

This mobile template is developed for users who come from different nations and speak different languages

This tutorial will show you how to create and modify your language resources (enable multiple languages for mobile app)

Prerequisite

Click here to know how to create text resource in Casewhere

Guide

How multiple languages work inmobile

- Step 1

The mobile app will load offline text resources files to display

- Step 2

The mobile app will take text resources from Casewhere and then overwrite the data from offline text resources files

- Step 3

If the mobile does not have internet connection, the mobile app will take offline text resources to display

Set up text resources key

First we need to fetch all the text resources keys in our mobile app

The syntax

```
{{'key' | translate}}
```

We can find all of the keys in html files

We will take the contact-list page in previous section as example

Navigate to src/app/pages/contact/contact-list/contact-list.page.html

These are text resources key we extract from contact-list page

```
{{'cw_mb_contact_list' | translate}}
{{'cw_mb_contact_add' | translate}}
{{'cw_mb_common_reload' | translate}}
```

Get translation from Casewhere

Click here to know how to set up text resource on Casewhere

Go to Casewhere Admin, Text resources

In this tutorial, we use Casewhere Mobile Demo product

Click filter, choose Casewhere Mobile Demo product

Add 3 new text resources key of contact-list page and the set the value for them

KEY	DA	EN	FR
cw_mb_contact_list	[DK] Contact	Contact	[FR] Contact
cw_mb_contact_add	[DK] Add	Add	[FR] Add
cw_mb_common_reload	[DK] Reload	Reload	[FR] Reload

Support offline mode

Update existed languages

- Step 1

Navigate to src/assets/i18n, find your language text resources (i.e da.json for Danish, en.json for English)

- Step 2

Add new text resource key

```
{
    "key_1": "One",
    "key_2": "Two",
    "key_3": "Three"
    ....,
    "key_n": "n"
}
```

If the text resource key already existed, please find the key in the json file and update it

Add new language

- Step 1

We will take French as a new language to add

Navigate to src/assets/i18n, create a new JSON file fr.json (fr is shortcut for French)

This file will contain all keys and its value for French

- Step 2

Add text resource keys to `fr.json`

```
{
    "cw_mb_contact_list" : "[FR] Contact",
    "cw_mb_contact_add"  : "[FR] Add",
    "cw_mb_common_reload": "[FR] Reload"
}
```

Enable multiple languages

This section will answer the question: "How can we apply text resources into the mobile app?"

1. In every page, navigate to **your-page**.module.ts, import TranslateModule

```
import { TranslateModule } from '@ngx-translate/core';
```

In the same **your-page**.module.ts, import `TranslateModule.forChild()`

```
@NgModule({
    imports: [
        ...
        TranslateModule.forChild()
    ],
    ...
})
```

1. Navigate to src -> app -> app.component.ts

- Import necessary packages

```
import { TranslateService } from '@ngx-translate/core';
import { LanguageService } from './pages/language/language.service';
```

- Add these lines to constructor

```
constructor(
    ...
    public translate: TranslateService,
    private langService: LanguageService
    ) {
        ...
        this.setLanguage();
    }`
```

`setLanguage()` responsible for applying text resources to our mobile app

```
setLanguage() {
    var appLang = this.langService.getAppLang();
    this.langService.getTextResource({ language: appLang }).subscribe(
        (data) => {
            for (const property in data) {
                this.translate.set(property, data[property]);
            }
    });

    this.translate.addLangs(['en', 'da', 'vi']);
    this.translate.use('da');
    this.translate.use('en');
    this.translate.use('vi');
    setTimeout(() => {
        this.translate.use(appLang);
    }, 200)
}
```

Change languages on mobile app

We need to create a page for user to change language

Read [here](../../../../products/template/mobile-app/getting-started/generate new feature) to know how to create a page

`language.page.html`

```
<ion-header>
    <ion-toolbar>
        <ion-title>{{'cw_mb_language' | translate}}</ion-title>
    </ion-toolbar>
</ion-header>

<ion-content class="ion-padding">
    <ion-text>
        <h5>{{'cw_mb_select_default_language' | translate}}</h5>
    </ion-text>

    <ion-list>
        <ion-item (click)="selectLang('en')" button="true" [fill]="lang == 'en' ? 'solid' : ''">
            <ion-avatar slot="start">
                <img src="/assets/images/language-img.png" />
            </ion-avatar>
            <ion-label>
                <h2>{{'cw_mb_english' | translate}}</h2>
            </ion-label>
        </ion-item>
        <ion-item (click)="selectLang('da')" button="true" [fill]="lang == 'da' ? 'solid' : ''">
            <ion-avatar slot="start">
                <img src="/assets/images/language-img.png" />
            </ion-avatar>
            <ion-label>
                <h2>{{ 'cw_mb_danish' | translate}}</h2>
            </ion-label>
        </ion-item>
    </ion-list>

    <ion-row class="ion-justify-content-around ion-padding-vertical">
        <ion-button size="medium" type="button" class="text-transform-unset" (click)="saveLang()">
            {{'cw_mb_language_save_and_colse' | translate}}
        </ion-button>
    </ion-row>
</ion-content>
```

language.page.ts

```
import { Component, OnInit } from '@angular/core';
import { NavController } from '@ionic/angular';
import { TranslateService } from '@ngx-translate/core';
import { LanguageService } from './language.service';

@Component({
    selector: 'app-language',
    templateUrl: './language.page.html',
    styleUrls: ['./language.page.scss'],
})
export class LanguagePage implements OnInit {

    lang: string;
    constructor(public translate: TranslateService, private navController: NavController, private
langService: LanguageService) {
        this.lang = this.langService.getAppLang() || translate.currentLang;
    }

    ngOnInit() {
    }

    selectLang(lang) {
        this.lang = lang;
    }

    saveLang() {
        this.translate.use(this.lang).subscribe(() => {
            this.updateLanguage();
            this.navController.back();
        });
    }

    updateLanguage() {
        this.langService.changeAppLang(this.lang);
        this.langService.getTextResource({ language: this.lang })
            .subscribe(
                (data) => {
                    for (const property in data) {
                        this.translate.set(property, data[property]);
                    }
                });
    }
}
```

Create a service for language page

`language.service.ts`

```
import { Injectable } from '@angular/core';
import { AuthHttpService } from '../../auth/auth-http.service';
import { Observable, throwError } from 'rxjs';
import { catchError, map } from 'rxjs/operators';
import { environment } from 'src/environments/environment';
const LANG = 'LANG';

@Injectable({
    providedIn: 'root'
})
export class LanguageService {

    constructor(private autHttp: AuthHttpService) { }

    getTextResource(params?: any): Observable<any> {
        return this.autHttp.get(`${environment.API_URL}/cw-mobile-text-resources-get`, params)
            .pipe(
                map(res => {
                    return res;
                }),
                catchError((err, cautch) => { return throwError('Something bad happened; please try again
later.') })
            );
    }

    getAppLang() {
        return localStorage.getItem(LANG) || 'en';
    }

    changeAppLang(value: string) {
        localStorage.setItem(LANG, value);
    }
}
```

If you wanna add more languages options

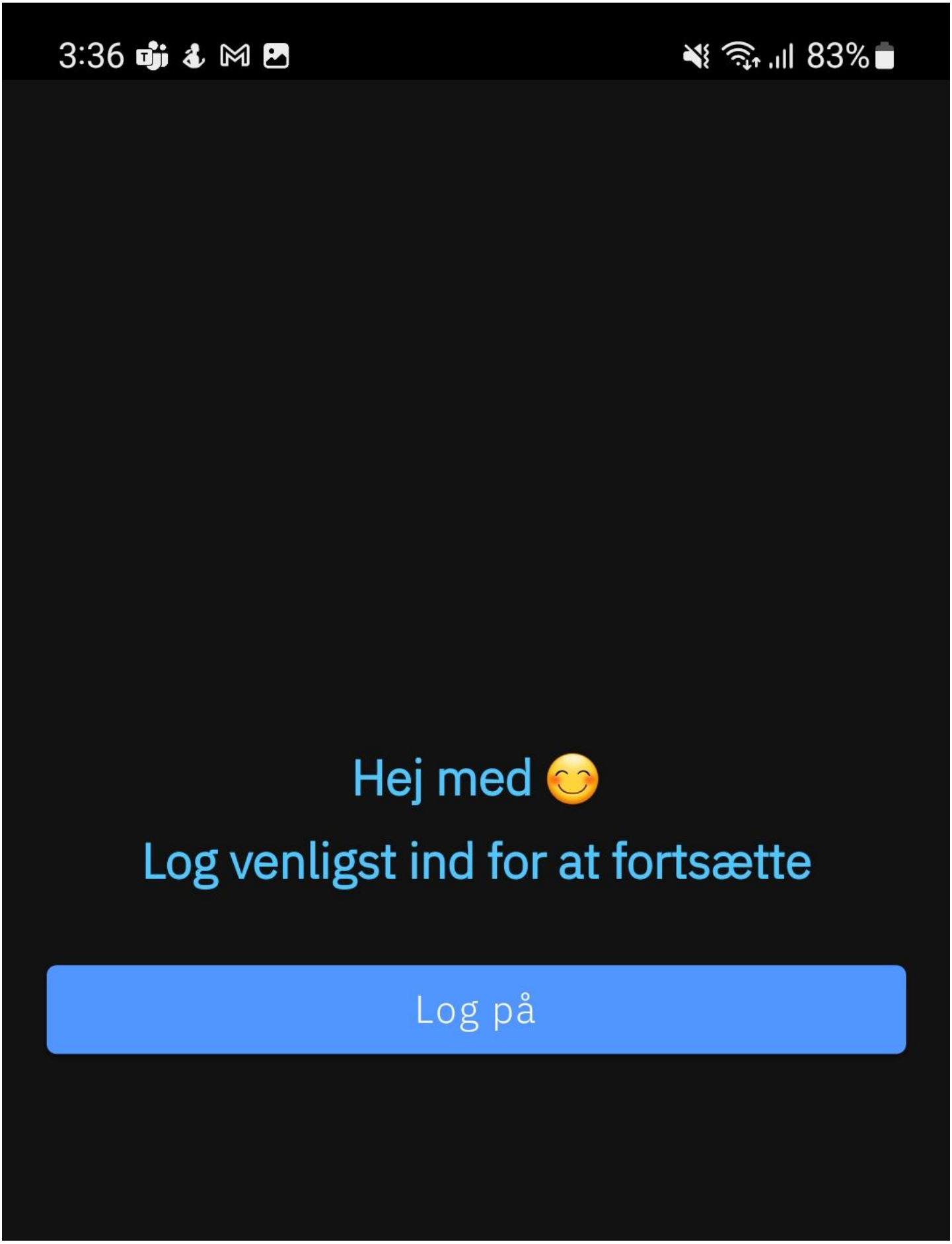
Navigate to src/app/pages/language/language.page.html
Add new language as an item to the existed list

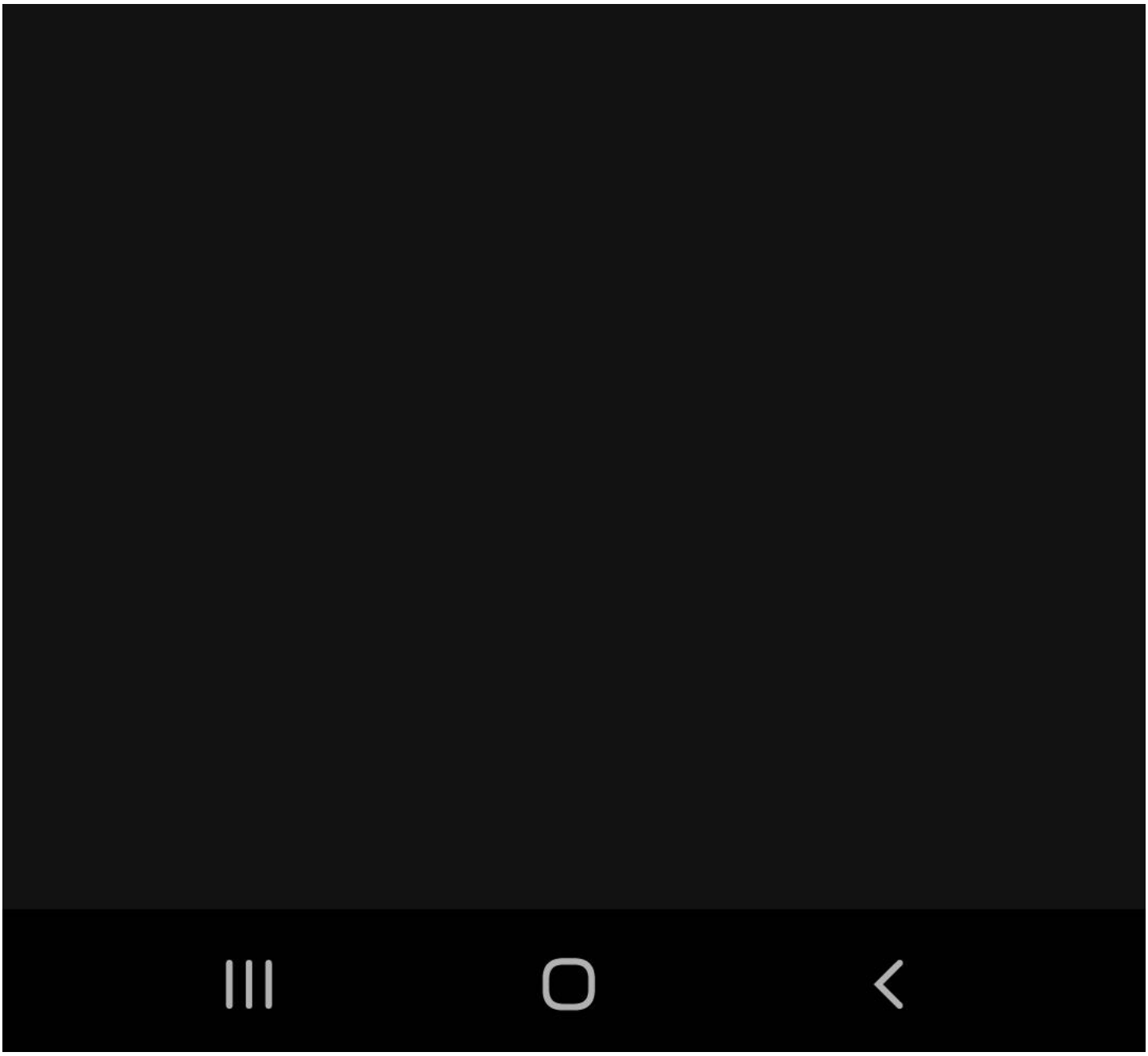
```
<ion-item (click)="selectLang('vi')" button="true" [fill]="lang == 'vi' ? 'solid' : ''">
    <ion-avatar slot="start">
        <img src="/assets/images/language-img.png" />
    </ion-avatar>
    <ion-label>
        <h2>{{ 'cw_mb_vi' | translate}}</h2>
    </ion-label>
</ion-item>
```

Mobile text resources demo

- Step 1

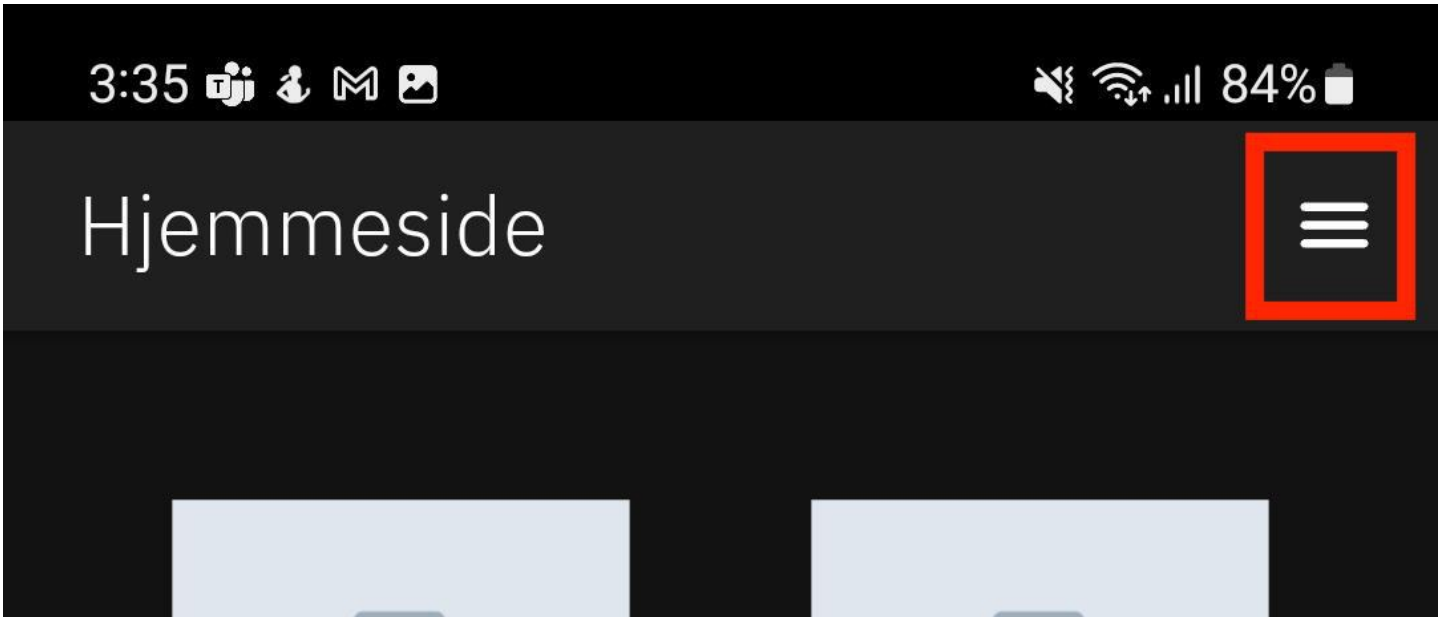
Open the app





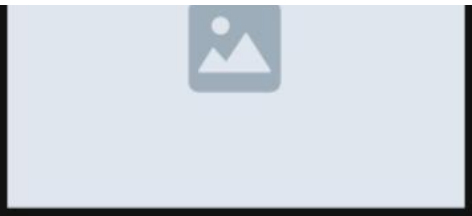
- Step 2

Login





Feature 1



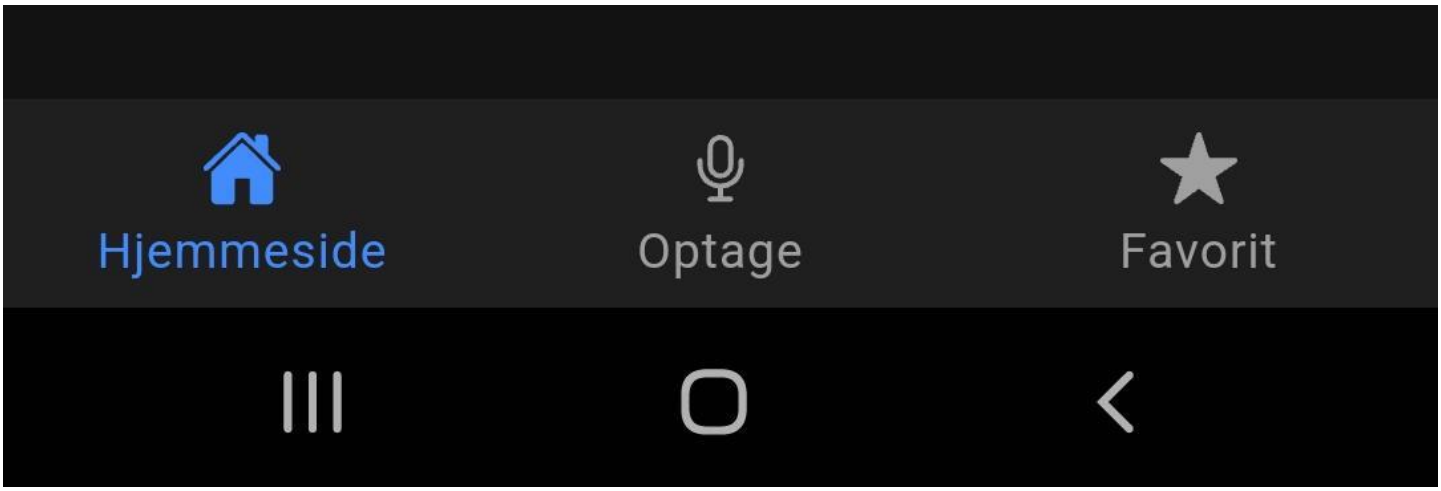
Feature 2



Feature 3

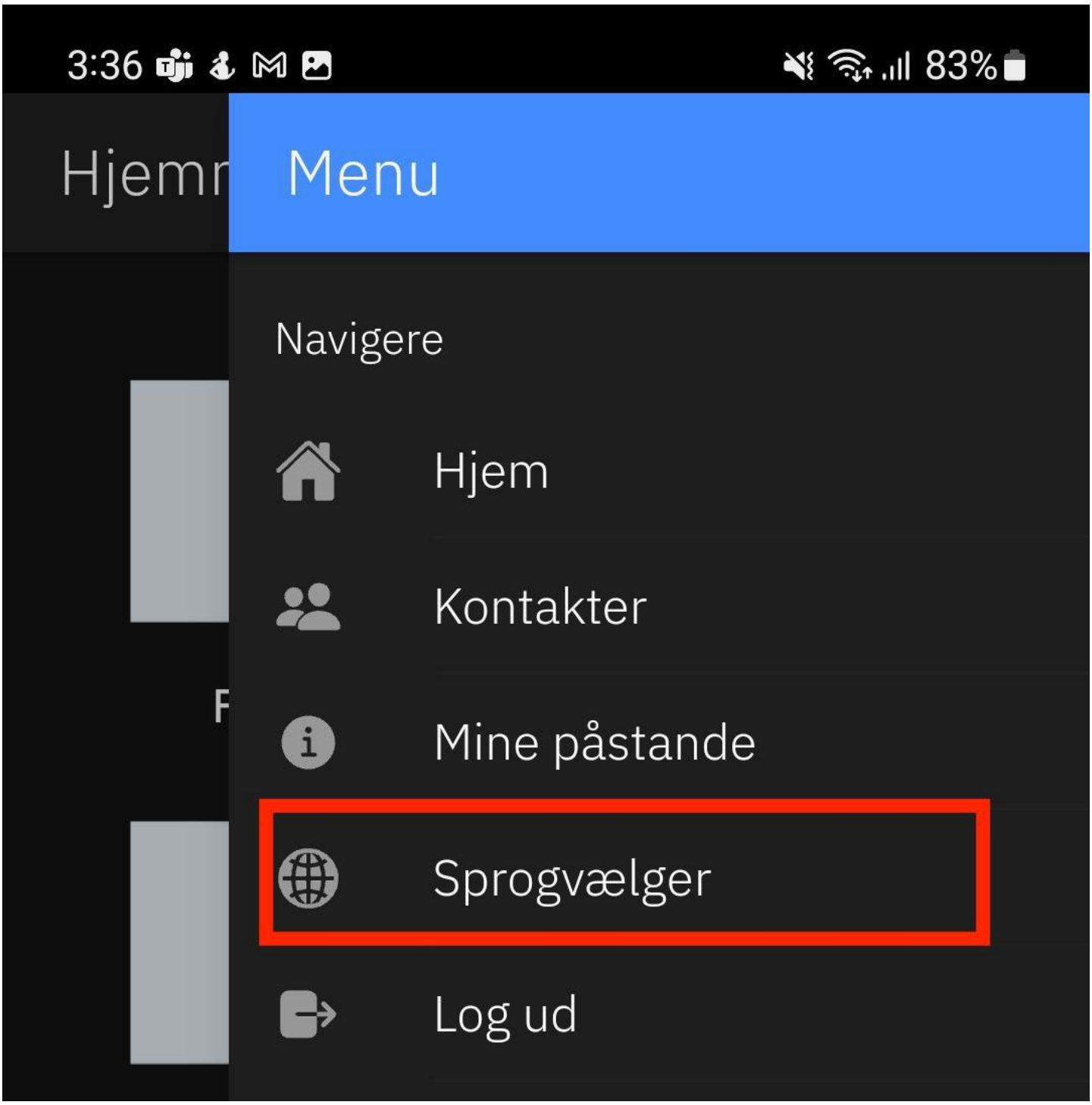


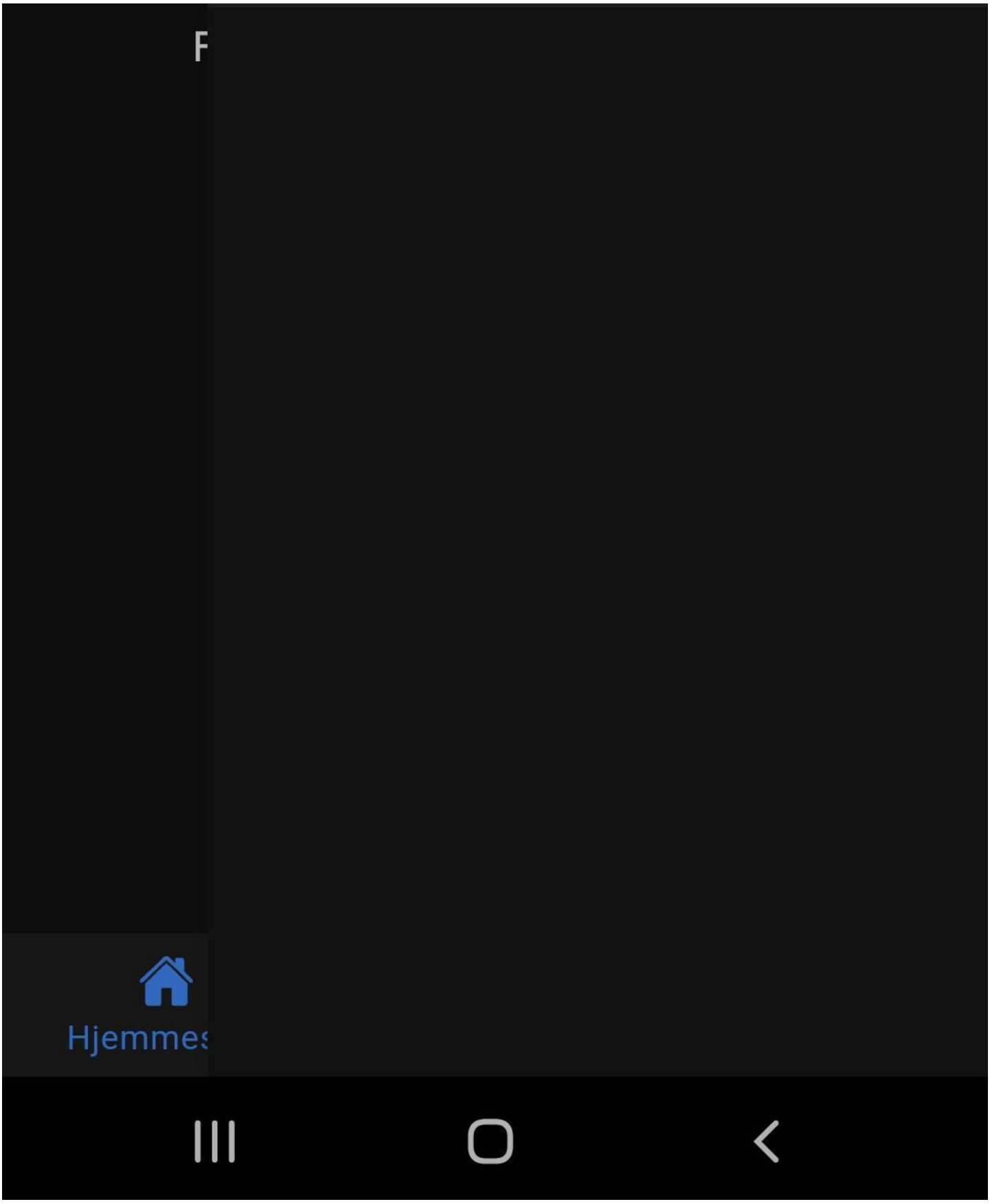
Feature 4



- Step 3

Navigate to Menu -> Language selector





- Step 4

Choose English then **Save and close**



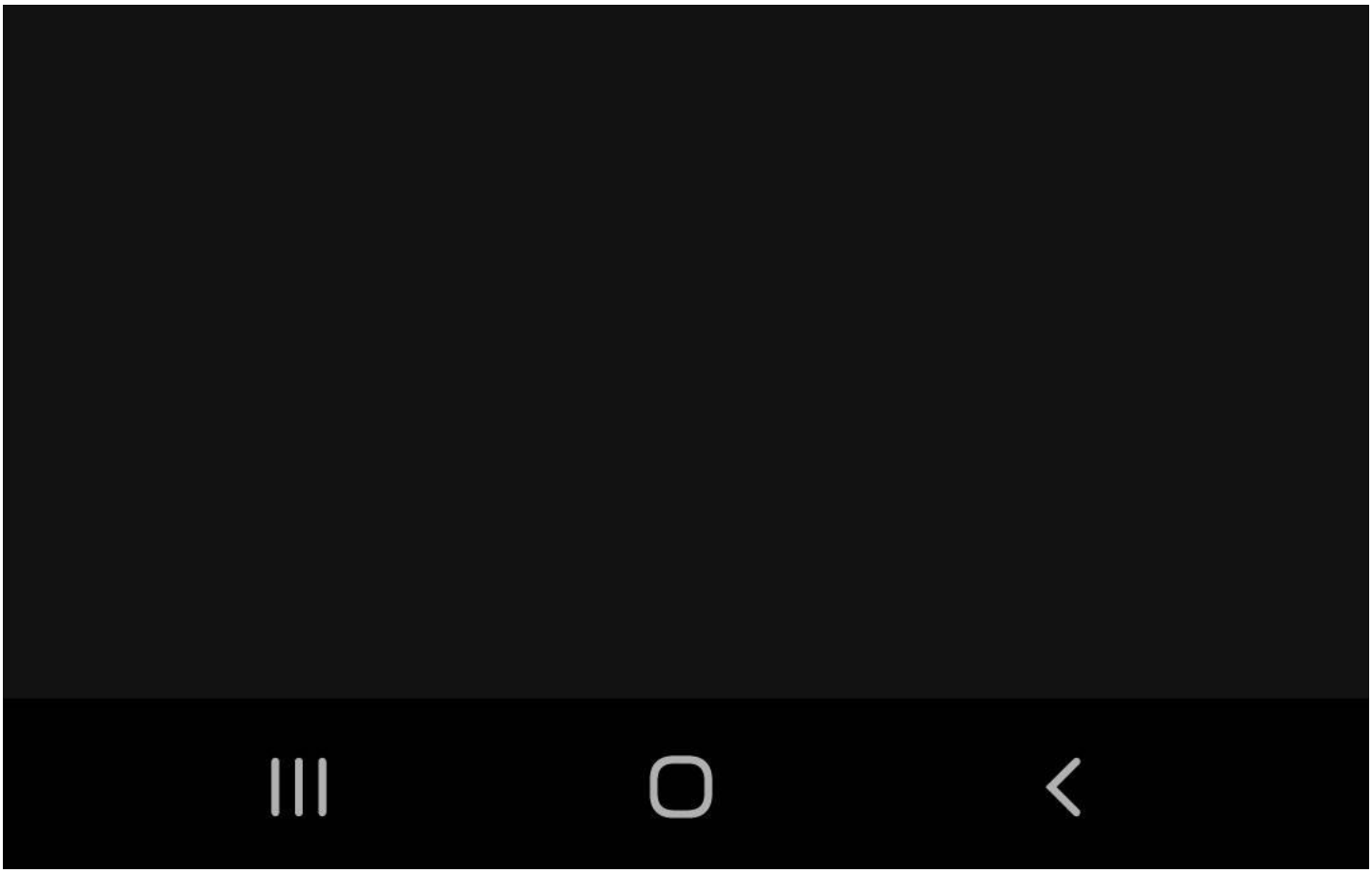
Sprog

Vælg standardsprog


English

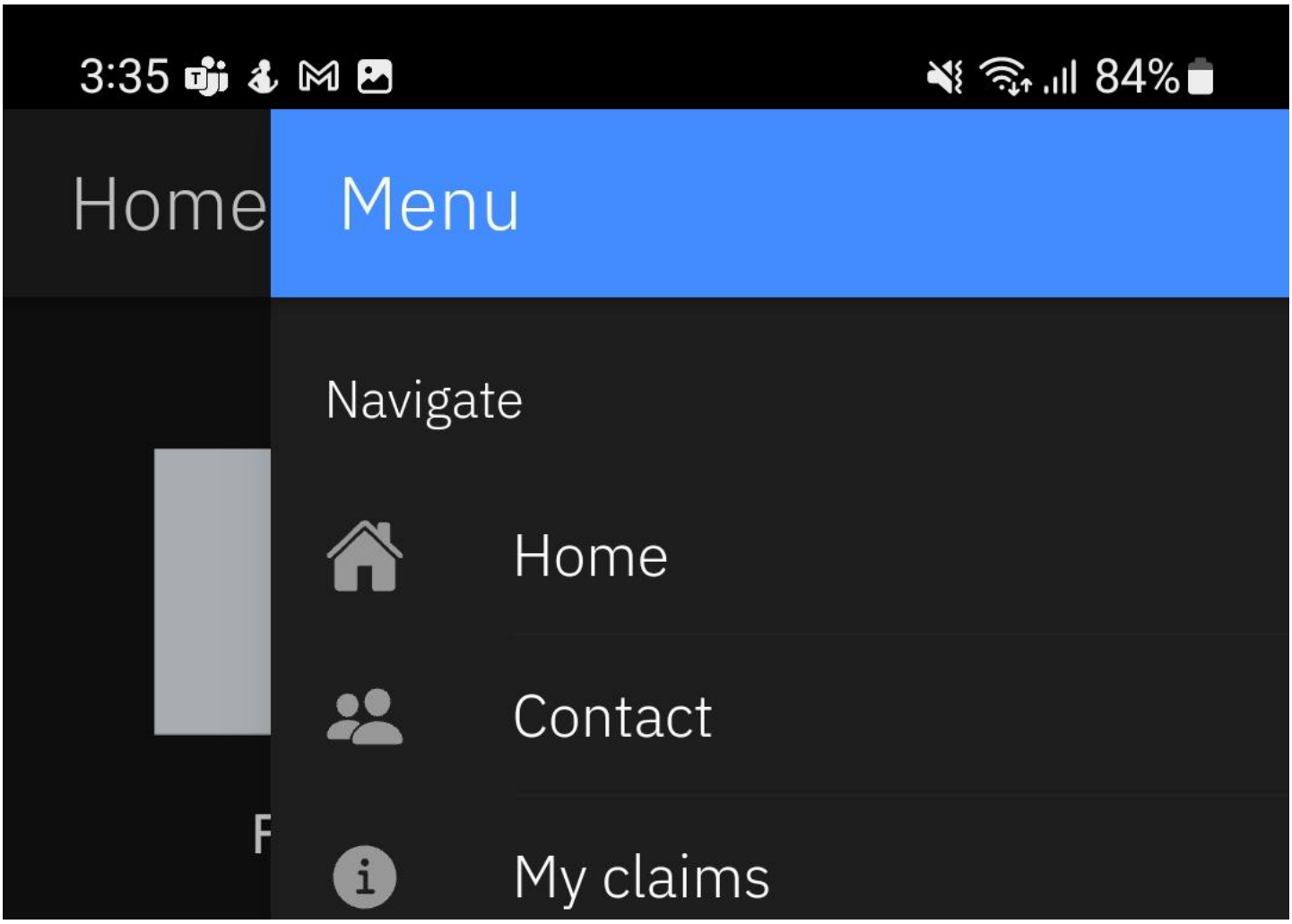

Danish

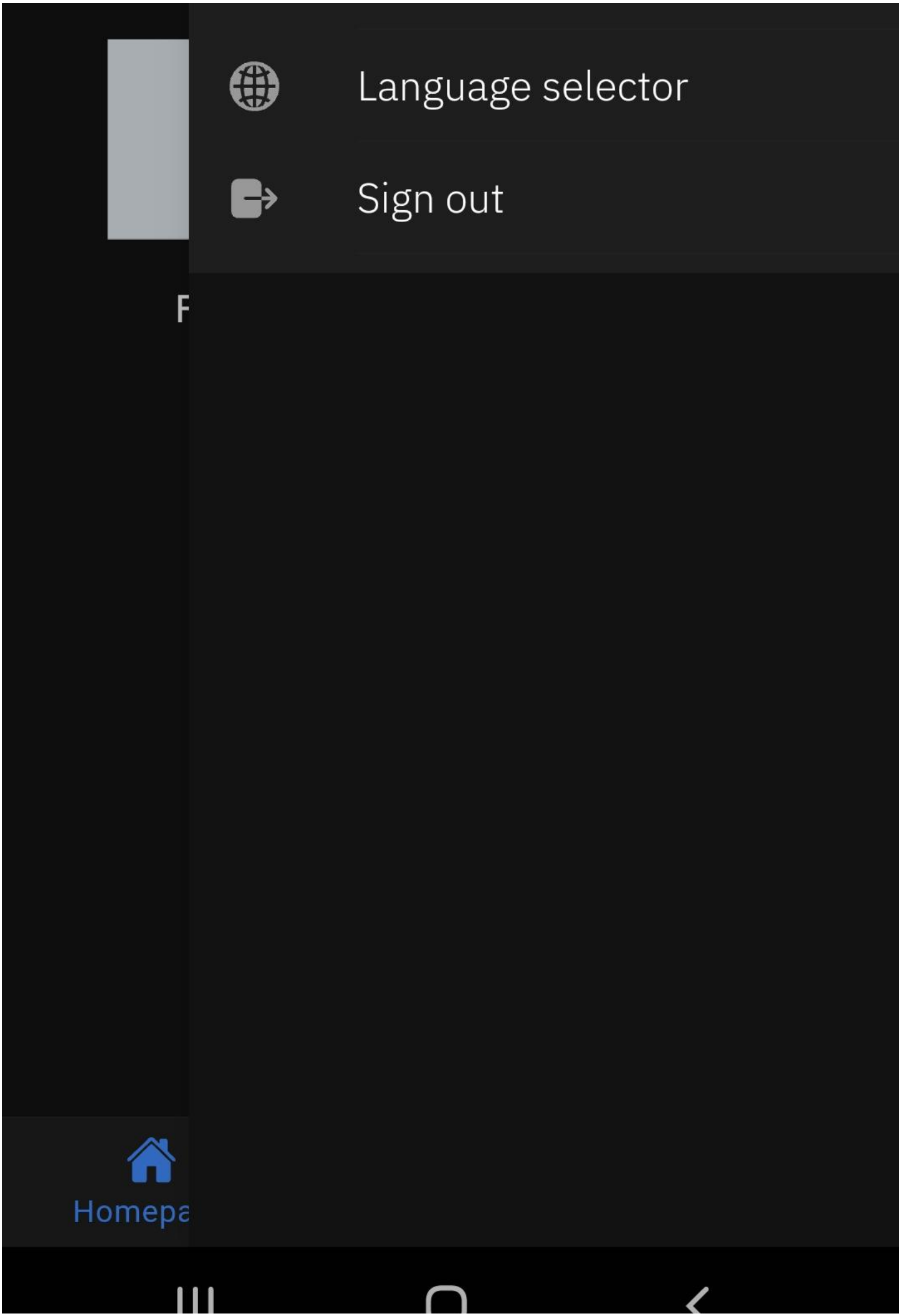
Gem og luk

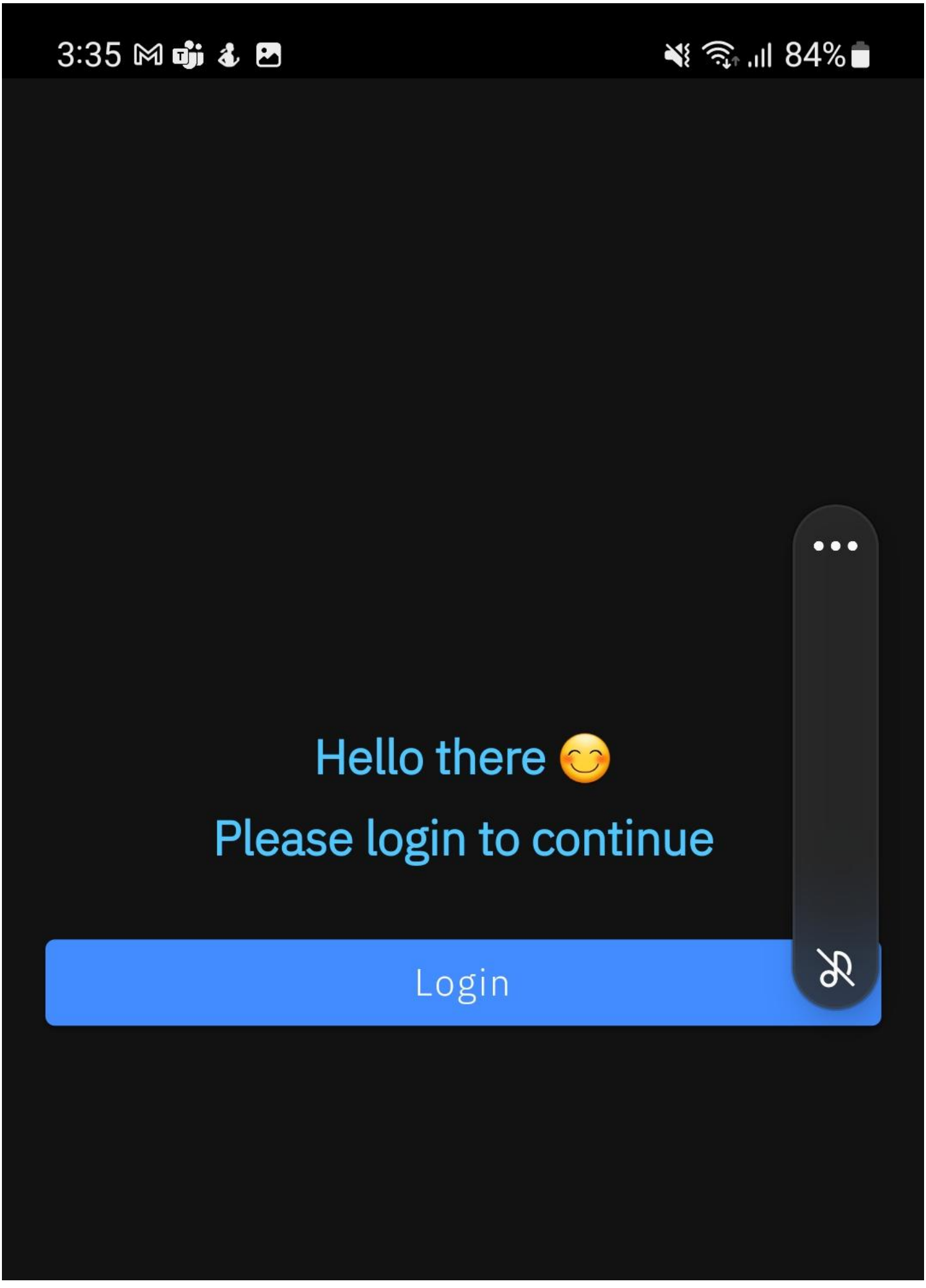


- Step 5

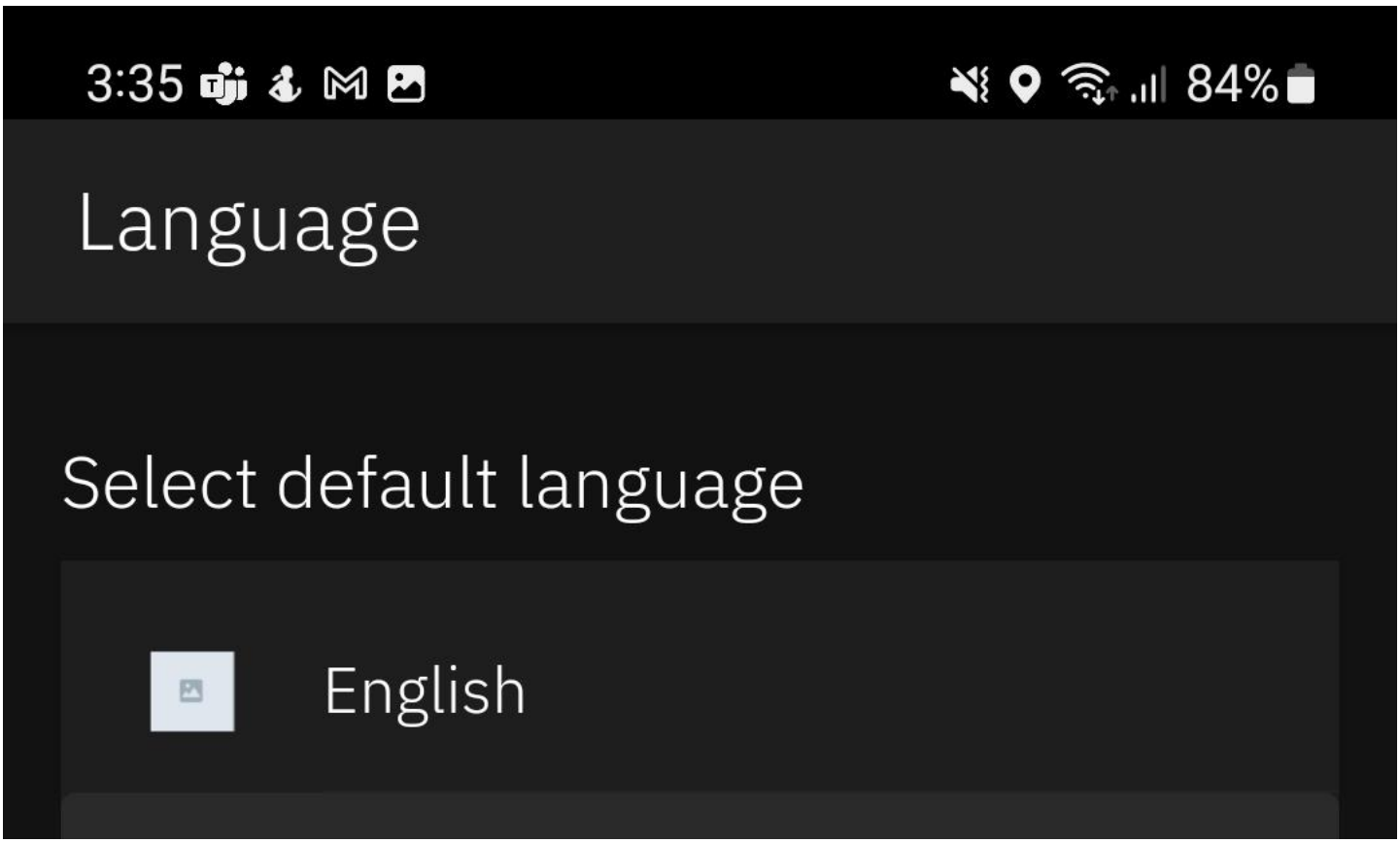
The language now changed to English




Language selector
Sign out
F
Homepa


3:35
84%
Hello there 😊
Please login to continue
Login




3:35
84%
Language
Select default language
English



Danish

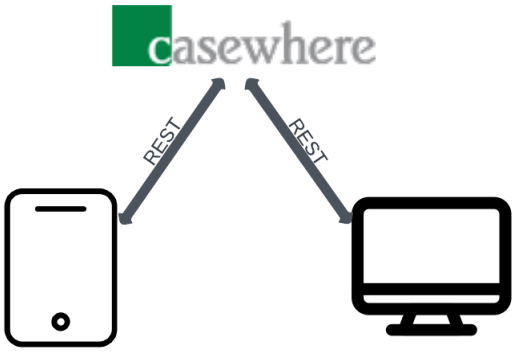
Save & Close



Overview

Introduction

Using the REST API, the mobile app template will query the data source directly from Casewhere via web trigger.



Method

Create HTTP connection

Navigate to `src/app/auth/auth-http.service.ts`.
This file has all the methods you can use to work with REST API.
i.e GET, POST, PUT, PATCH, etc...

GET

```
get(url: string): any {
    return this.http
        .get(url, { headers: this.httpOptionsJson })
        .pipe (catchError(this.handleHttpErrors));
}
```

POST

```
post(url: string, request: any): any {
    return this.http.post(url, request).pipe (catchError(this.handleHttpErrors));
}
```

Change API destination

- Step 1. Create Service, and inject `contact.service.ts`
- Step 2. Config `environment.API_URL` to execute the request

For example, in `getContactById` function

```
getContactById(id: string): Observable<any> {
    if (!id || id === '0') {
        this.contact.next(new Contact());
    }
    else {
        return this.autHttp.get(`${environment.API_URL}/contact?Id=${id}`)
            .pipe(
                map((res: any) => {
                    this.contact.next(res);
                    return res;
                }),
                catchError((err, cautch) => ContactService.handleError(err))
            );
    }
}
```

- Step 3. Modify `environment.API_URL` to change the destination

Casewhere web triggers

Navigate to Casewhere Admin > Web Triggers

Read more about web trigger here

Config

Introduction

This document shows the steps neede to work with REST API.

Guide

- Step 1. Navigate to `MobileScaffold` folder, open integrated terminal
- Step 2. Run `$ ionic generate`

Generate two new pages

`pages/contact/contact-list`

`pages/contact/contact-detail`

- Step 3. There are 6 auto-generated files: (* represents contact-list or contact-details)

1. ***-routing.module**: manage routing
2. ***.module**: Component module contains all the configuration
3. ***.page.html**: UI template for the page
4. ***.page.css**: Component style sheet. Can be used for custom style for your page
5. ***.page.spec.ts**: Unit test go here
6. ***.page.ts**: All the features code / logic code go here

- Step 4. Modify `contact-detail.page.html`

```
<ion-header>
    <ion-toolbar>
        <ion-title>Contact detail</ion-title>

        <ion-buttons slot="end">
            <ion-menu-toggle>
                <ion-button>
                    <ion-icon slot="icon-only" name="menu"></ion-icon>
                </ion-button>
            </ion-menu-toggle>
        </ion-buttons>
    </ion-toolbar>
</ion-header>

<ion-content class="ion-padding page-home-content">
    <section class="ion-text-center ion-margin-bottom">
        <ion-avatar class="avatar-block">
            <img src="/assets/images/avatar-rey.png" />
        </ion-avatar>
    </section>
    <section class="ion-margin-bottom">
        <ion-title>Name</ion-title>
        <ion-input class="" required [placeholder]="'cw_mb_contact_placeholder' | translate"
[(ngModel)]="contact.name">
        </ion-input>
    </section>
    <section class="ion-margin-bottom">
        <ion-title>Type</ion-title>

        <ion-select placeholder="Select type" [(ngModel)]="contact.type">
            <ion-select-option value="reporter">Reporter</ion-select-option>
            <ion-select-option value="relevant">Relevant</ion-select-option>
```

```
            <ion-select-option value="relevant">Relevant</ion-select-option>
            <ion-select-option value="invitee">Invitee</ion-select-option>
        </ion-select>
    </section>
    <section class="ion-margin-bottom">
        <ion-title>Address</ion-title>
        <ion-input required [placeholder]="'cw_mb_contact_placeholder' | translate"
[(ngModel)]="contact.address">
        </ion-input>
    </section>

    <ion-row class="ion-justify-content-around" *ngIf="!isAdding">
        <ion-col size="3">
            <ion-button size="medium" color="danger" type="button" class="text-transform-unset"
                (click)="deleteContact()">
                <ion-title>Delete</ion-title>
            </ion-button>
        </ion-col>
        <ion-col size="3">
            <ion-button size="medium" color="primary" type="button" class="text-transform-unset"
                (click)="updateContact()">
                <ion-title>Update</ion-title>
            </ion-button>
        </ion-col>
    </ion-row>
    <ion-row class="ion-justify-content-between" *ngIf="isAdding">
        <ion-col size="3"></ion-col>
        <ion-col size="3">
            <ion-button size="medium" color="primary" type="button" class="text-transform-unset"
(click)="addContact()">
                <ion-title>Add</ion-title>
            </ion-button>
        </ion-col>
    </ion-row>

</ion-content>
```

- Step 5. Modify `contact-detail.page.scss`

```
.page-home-content {
    .avatar-block {
        display: inline-block;
    }
}
```

- Step 6. Modify `contact-detail.page.ts`

```
import { Component, OnInit } from '@angular/core';
import { ContactService } from '../contact.service';
import { ActivatedRoute } from '@angular/router';
import { Contact } from '../../../shared/model';
import { LoadingController } from '@ionic/angular';

@Component({
    selector: 'app-contact-detail',
    templateUrl: './contact-detail.page.html',
    styleUrls: ['./contact-detail.page.scss'],
})
export class ContactDetailPage implements OnInit {
    id: string;
    contact: Contact = new Contact();
    isAdding = false;
    loading: any;

    constructor(private contactService: ContactService, private route: ActivatedRoute, private loadingCtrl:
```

```
LoadingController) { }

    ngOnInit() {
        this.contactService.contact$.subscribe((val) => {
            this.contact = val;
        });

        this.id = this.route.snapshot.paramMap.get('id');
        if (!this.id || this.id === '0') {
            this.isAdding = true;
            this.contactService.resetContactModel();
        }
        else {
            this.getContactById();
        }
    }
    async getContactById() {
        await this.createAndPresentLoading();
        this.contactService.getContactById(this.id)
            .subscribe(
                (val) => this.loading.dismiss(),
                (err) => this.loading.dismiss()
            );
    }
    deleteContact = async () => {
        await this.createAndPresentLoading();
        this.contactService.deleteContact(this.id)
            .subscribe(
                (val) => this.loading.dismiss(),
                (err) => this.loading.dismiss()
            );
    }
    updateContact = async () => {
        await this.createAndPresentLoading();
        this.contactService.updateContact(this.contact)
            .subscribe(
                (val) => this.loading.dismiss(),
                (err) => this.loading.dismiss()
            );
    }
    addContact = async () => {
        await this.createAndPresentLoading();
        this.contact.date = new Date();
        this.contactService.addContact(this.contact)
            .subscribe(
                (val) => this.loading.dismiss(),
                (err) => this.loading.dismiss()
            );
    }
    async createAndPresentLoading() {
        this.loading = await this.loadingCtrl.create({
            message: 'Loading...',
            spinner: 'circles'
        });
        this.loading.present();
    }
}
```

Step 7. Modify `contact-list.page.html`

```
<ion-header>
    <ion-toolbar>
        <ion-title>Contact list</ion-title>

        <ion-buttons slot="end">
            <ion-menu-toggle>
                <ion-button>
                    <ion-icon slot="icon-only" name="menu"></ion-icon>
                </ion-button>
            </ion-menu-toggle>
        </ion-buttons>
    </ion-toolbar>
</ion-header>

<ion-content>
    <section class="ion-padding-horizontal ion-text-right" style="margin-top: 4px;">
        <ion-button size="medium" color="primary" type="button" class="text-transform-unset"
            [routerLink]="['/menu/main/contacts', 0]">
            <ion-icon name="add-outline" slot="start"></ion-icon>
            <ion-title>Add</ion-title>
        </ion-button>
        <ion-button size="medium" color="primary" type="button" class="text-transform-unset"
(click)="getContacts()">
            <ion-icon name="reload-outline" slot="start"></ion-icon>
            <ion-title>Reload</ion-title>
        </ion-button>
    </section>
    <ion-list>
        <ion-item *ngFor="let contact of contacts" [routerLink]="['/menu/main/contacts', contact.id]">
            <ion-avatar slot="start">
                <img src="/assets/images/avatar-rey.png" />
            </ion-avatar>
            <ion-label>
                <h2>{{contact.name}}</h2>
                <h3>{{contact.type}}</h3>
                <p>{{contact.address}}</p>
            </ion-label>
        </ion-item>
    </ion-list>
</ion-content>
```

Step 8. Modify `contact-list.page.ts`

```
import { Component, OnInit } from '@angular/core';
import { LoadingController } from '@ionic/angular';
import { ContactService } from '../contact.service';

@Component({
    selector: 'app-contact-list',
    templateUrl: './contact-list.page.html',
    styleUrls: ['./contact-list.page.scss'],
})

export class ContactListPage implements OnInit {

    contacts = [];
    constructor(private contactService: ContactService, private loadingCtrl: LoadingController, public
translate: TranslateService) { }

    async ngOnInit() {
        this.getContacts();
        this.contactService.contacts$.subscribe((val) => {
            this.contacts = val;
        });
    }

    async getContacts() {
        const loading = await this.loadingCtrl.create({
            message: 'Loading...',
            spinner: 'circles'
        });
        loading.present();
        this.contactService.getContacts()
            .subscribe(
                (val) => loading.dismiss(),
                (err) => loading.dismiss()
            );
    }
}
```

- Step 9. Generate `contact` service

```
CREATE src/app/pages/contact/contact.service. spec.ts (362 bytes)
CREATE src/app/pages/contact/contact.service.ts (136 bytes)
[OK] Generated service!
```

- Step 10. Modify `contact.service.ts`

```
import { HttpErrorResponse } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { NavController } from '@ionic/angular';
import { BehaviorSubject, Observable, throwError, pipe } from 'rxjs';
import { catchError, filter, map } from 'rxjs/operators';
import { Contact } from 'src/app/shared/model';
import { AuthHttpService } from '../../auth/auth-http.service';
import { environment } from 'src/environments/environment';

@Injectable({
    providedIn: 'root'
})
export class ContactService {

    private contacts = new BehaviorSubject<Contact[] | null>([]);
    public readonly contacts$ = this.contacts.asObservable();

    private contact = new BehaviorSubject<Contact | null>(new Contact());
    public readonly contact$ = this.contact.asObservable();
```

```
    private static handleError(error: HttpErrorResponse): any {
        if (error.error instanceof ErrorEvent) {
            console.error('An error occurred:', error.error.message);
        }
        else {
            console.error(`Backend returned code ${error.status}, ` + `body was: ${error.error}`);
        }
        return throwError('Something bad happened; please try again later.');
    }

    constructor(private autHttp: AuthHttpService, private navController: NavController) { }

    changeData(data: any) {
        this.contacts.next(data);
    }

    /**
     *
     * @param isNeedGoBack
     */
    getContacts(isNeedGoBack?: boolean): Observable<any> {
        return this.autHttp.get(`${environment.API_URL}/contacts`)
            .pipe(
                map(res => {
                    this.changeData(res);
                    if (isNeedGoBack) {
                        this.navController.back();
                    }
                    return res;
                }),
                catchError((err, cautch) => ContactService.handleError(err))
            );
    }
    getContactById(id: string): Observable<any> {
        if (!id || id === '0') {
            this.contact.next(new Contact());
        }
        else {
            return this.autHttp.get(`${environment.API_URL}/contact?Id=${id}`)
                .pipe(
                    map((res: any) => {
                        this.contact.next(res);
                        return res;
                    }),
                    catchError((err, cautch) => ContactService.handleError(err))
                );
        }
    }
    deleteContact(id: string): Observable<any> {
        const model = {
            contactId: id
        }
        return this.autHttp.post(`${environment.API_URL}/contact/delete`, model)
            .pipe(
                map((res: any) => {
                    this.getContacts(true).subscribe();
                    return res;
                }),
                catchError((err, cautch) => ContactService.handleError(err))
            );
    }
    updateContact(model: Contact): Observable<any> {
        return this.autHttp.post(`${environment.API_URL}/contact/edit`, { ...model, contactId: model.id })
            .pipe(
                map((res: any) => {
```

```
                map((res: any) => {
                    this.getContacts(true).subscribe();
                    return res;
                }),
                catchError((err, cautch) => ContactService.handleError(err))
            );
    }
    addContact(model: Contact): Observable<any> {
        return this.autHttp.post(`${environment.API_URL}/contact`, model)
            .pipe(
                map((res: any) => {
                    this.getContacts(true).subscribe();
                    return res;
                }),
                catchError((err, cautch) => ContactService.handleError(err))
            );
    }
    resetContactModel() {
        this.contact.next(new Contact());
    }
}
```

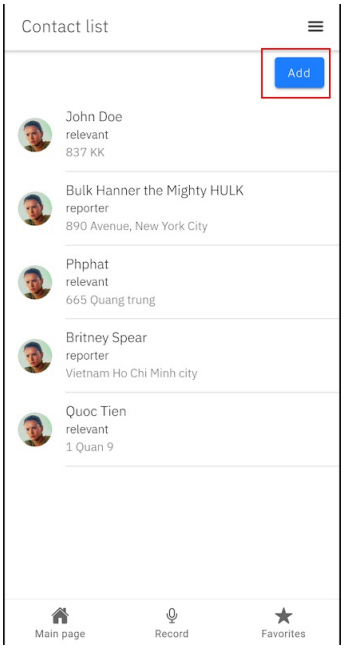
Demo

This post presents a simple REST demo with the basic methods for adding, editing and deleting.

For a feature showcase, watch demo video - Authentication and REST API (may requires login).

Add

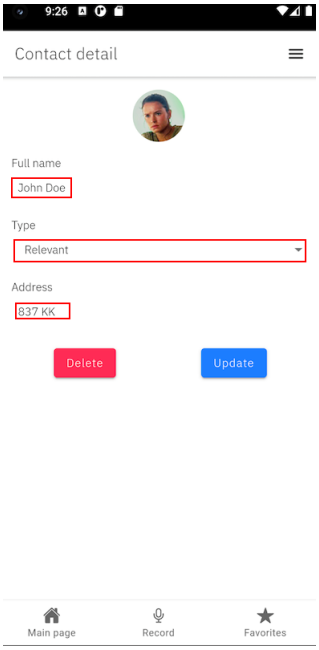
Tap **Add** button to add a new contact



Edit

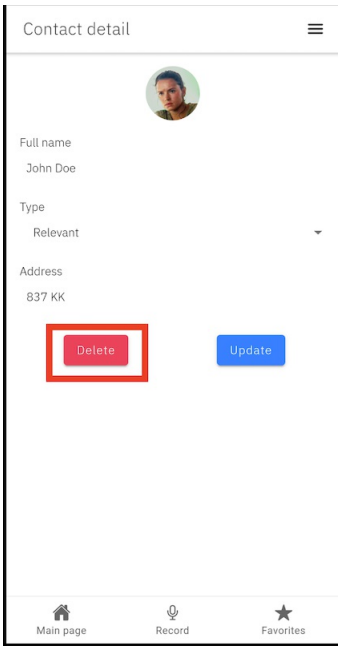
Select any contact to enter edit mode.

Edit any fields, then tap **Update** button to update the information.



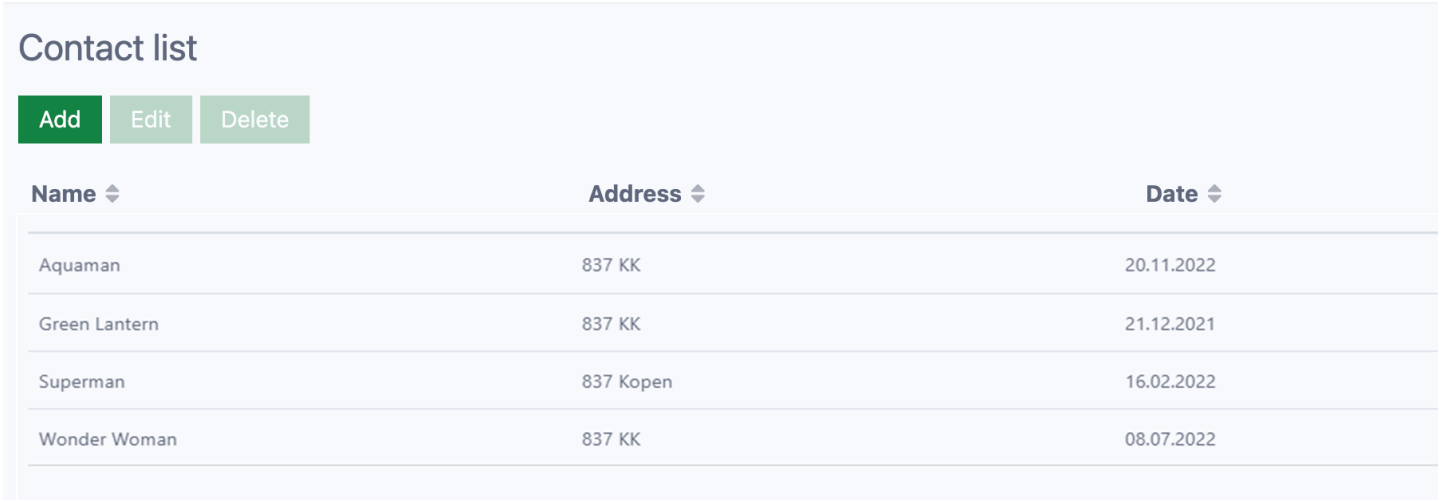
Delete

Tap **Delete** to remove a contact from the database



If you've previously created a basic web app in Casewhere, the database will be updated and synced across the mobile and that web app.

You can add/edit/delete in one device to see the changes on the other.



Notes

You can build the project from:

- The source code
- APK file (with an emulator or a real device)

> Test in both emulator and real devices to check the app behavior

Configuration

Introduction

This article shows you how to use the list on the page, then demo the contact list that the app is using.

Usage

There are examples of the list, you can use them by coping, then put them into your HTML file:

- List of text items

```
<ion-list>
    <ion-item>
        <ion-label>Pokémon Yellow</ion-label>
    </ion-item>
    <ion-item>
        <ion-label>Mega Man X</ion-label>
    </ion-item>
    <ion-item>
        <ion-label>The Legend of Zelda</ion-label>
    </ion-item>
</ion-list>
```

- List of sliding items

```
<ion-list>
    <ion-item-sliding>
        <ion-item>
            <ion-label>Item 1</ion-label>
        </ion-item>
        <ion-item-options side="end">
            <ion-item-option (click)="unread(item)">Unread</ion-item-option>
        </ion-item-options>
    </ion-item-sliding>

    <ion-item-sliding>
        <ion-item>
            <ion-label>Item 2</ion-label>
        </ion-item>
        <ion-item-options side="end">
            <ion-item-option (click)="unread(item)">Unread</ion-item-option>
        </ion-item-options>
    </ion-item-sliding>
</ion-list>
```

  Let's dive into the detail:

  - There are two sliding items defined by `<ion-item-sliding>` tag.
  - `<ion-item>` tag contains the UI of the item.
  - `<ion-item-options>` contains the list of `<ion-item-option>`, show them by swiping the item from the left-to-right or vice versa.

You can click here for more detail.

Demo

The contact feature is now using the sliding list type. It supports swiping actions that you can use for revealing options.

See the lines of code below, you can easily find It in the **contact-list.page.html** file, in **src/app/pages/contact/contact-list**

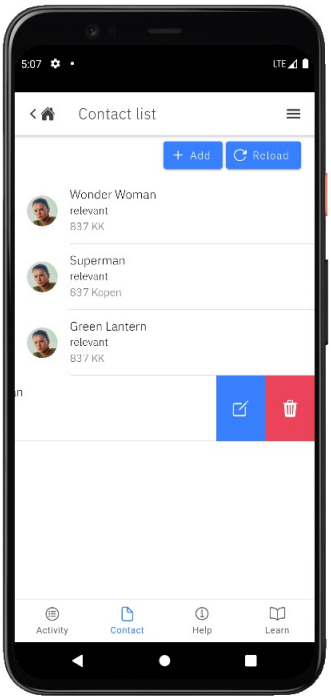
folder.

```
<ion-list>
    <ion-item-sliding *ngFor="let contact of contacts">
        <ion-item>
            <ion-avatar slot="start">
                <img [src]="contact.avatar" />
            </ion-avatar>
            <ion-label>
                <h2>{{contact.name}}</h2>
                <h3>{{contact.type}}</h3>
                <p>{{contact.address}}</p>
            </ion-label>
        </ion-item>
        <ion-item-options side="end" (ionSwipe)="deleteContact(contact.id)">
            <ion-item-option [routerLink]="['/contacts', contact.id]">
                <ion-icon name="create-outline" slot="icon-only"></ion-icon>
            </ion-item-option>
            <ion-item-option color="danger" expandable (click)="deleteContact(contact.id)">
                <ion-icon name="trash" slot="icon-only"></ion-icon>
            </ion-item-option>
        </ion-item-options>
    </ion-item-sliding>
</ion-list>
```

We also have the appropriate actions in the **contact-list.page.ts**.

```
...
export class ContactListPage implements OnInit {
    contacts = [];
    ...

    deleteContact = async (id: string) => {
        ...
    }
}
```



References

- ion-item-options

- ion-item-sliding

Configuration

Introduction

A grid contains row(s) and column(s). Columns will expand to fill the row, and will resize to fit additional columns. It is based on a 12 columns layout with different breakpoints based on the screen size.

Usage

There are examples of the grid, see detail inside the following, you can use them by coping, then put them into your HTML file:

```
<ion-grid>
    <!-- Normal list -->
    <ion-row>
        <ion-col>ion-col</ion-col>
        <ion-col>ion-col</ion-col>
        <ion-col>ion-col</ion-col>
        <ion-col>ion-col</ion-col>
    </ion-row>

    <!-- List with size -->
    <ion-row>
        <ion-col size="6">ion-col [size="6"]</ion-col>
        <ion-col>ion-col</ion-col>
        <ion-col>ion-col</ion-col>
    </ion-row>
</ion-grid>
```

You can click here for more detail.

Demo

The contact detail also implemented a simple grid, It contains **Delete** and **Update** buttons.

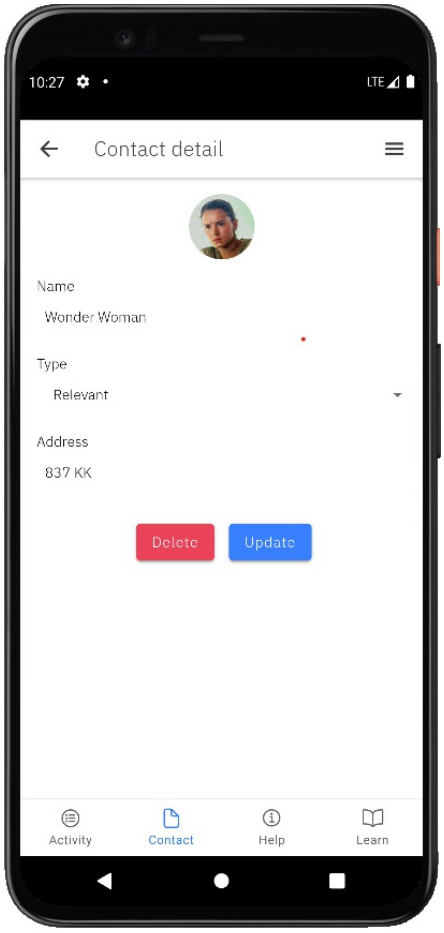
See the lines of code below, you can easily find It in the **contact-detail.page.html** file, in **src/app/pages/contact/contact-detail** folder.

```
<ion-grid>
    <ion-row class="ion-justify-content-around">
        <ion-col class="ion-text-right">
            <ion-button size="medium" color="danger" type="button" class="text-transform-unset"
(click)="deleteContact()">
                Delete
            </ion-button>
        </ion-col>
        <ion-col>
            <ion-button size="medium" color="primary" type="button" class="text-transform-unset"
(click)="updateContact()">
                Update
            </ion-button>
        </ion-col>
    </ion-row>
</ion-grid>
```

We also have the appropriate actions in the **contact-detail.page.ts**.

```
...
export class ContactDetailPage implements OnInit {
    contacts = [];
    ...

    deleteContact = async (id: string) => {
        ...
    }
    updateContact = () => {
        ...
    }
}
```



References

- Responsive Grid

Overview

Introduction

This documentation helps you build a notification feature.

Prerequisite

- Web trigger for creating & updating notifications.

Configuration

Introduction

This article shows how to implement the notification feature.

Guide

This section will help you create a notification page, and service, then show the number of unread notification items on the home page.

Generate notification pages

From the command line, run the following steps to create a new page:

- Step 1: Type `ionic generate page`.
- Step 2: Type the name of the page: `pages/notification`.

After successfully completing step 2, you will see the **notification** folder inside **src/app/pages**.

Notes: You might need to move the **notification** item to the top of the default item - inside **app-routing.module.ts**:

```
{
  path: 'notification',
  loadChildren: () => import('./pages/notification/notification.module').then(m => m.NotificationPageModule)
},
{ path: '**', redirectTo: ... }
```

Update UI

Let's replace **notification.page.html**'s content with the following line of codes:

```
<ion-header>
    <ion-toolbar>
        <ion-buttons slot="start">
            <ion-back-button></ion-back-button>
        </ion-buttons>
        <ion-title>Notification</ion-title>
    </ion-toolbar>
</ion-header>

<ion-content class="page-notification-content">
    <ion-list>
        <ion-item-sliding class="noti-item" *ngFor="let noti of notifications"
[class.highlight]="!noti.isRead">
            <ion-item button (click)="updateNoti(noti)">
                <ion-label>
                    <h2>{{noti.content}}</h2>
                    <i class="date">{{noti.createdAt | date:'d.MM.yy h:mm'}}</i>
                </ion-label>
            </ion-item>
        </ion-item-sliding>
    </ion-list>
</ion-content>
<ion-footer class="ion-no-border">
    <div class="ion-padding ion-text-center">
        <ion-button class="text-transform-unset" type="button" routerLink="/home">Close</ion-button>
    </div>
</ion-footer>
```

Then, we will have:

- The back button is defined by the tag `ion-back-button`.
- The contact list is wrapped inside the tag `ion-list`.
- The close button will redirect to the home page inside the `ion-footer`.

Update logic

Redirect to **notification.page.ts** and replace it by the below lines of code:

```
import { Component, OnInit } from '@angular/core';
import { NotificationService } from './notification.service';
import { finalize } from 'rxjs/operators';
import { Notification } from 'src/app/shared/model';

@Component({
    selector: 'app-notification',
    templateUrl: './notification.page.html',
    styleUrls: ['./notification.page.scss'],
})
export class NotificationPage implements OnInit {

    notifications = [];
    constructor(private notificationService: NotificationService) { }

    ngOnInit() {
        this.getNotifications();
        this.notificationService.notifications$.subscribe((val) => {
            this.notifications = val;
        });
    }

    async getNotifications() {
        this.notificationService.getNotifications().subscribe();
    }

    updateNoti(model: Notification) {
        if (!model.isRead) {
            model.isRead = true;
            this.notificationService.updateNoti(model).subscribe();
        }
    }
}
```

Let's dive into the detail:

- Data will be called and loaded from the service on **ngOnInit()** when the page has initialized.
- The **updateNoti()** function will update the notification item.

Adding styles

Let's decorate a little bit for the page. Open the file **notification.page.scss** and add the following lines of code:

```
.page-notification-content {
    .noti-item {
        .date {
            font-size: 1.2rem;
        }
        .icon {
            color: #51cabd;
        }
    }
    .highlight {
        --ion-background-color: #e9f1ff !important;
    }
}
```

Adding model

We need to define the **Notification** model and Its properties:

- Step 1: Open **model.ts** inside **src/app/shared** folder.
- Step 2: Add with following lines of code:

```
export class Notification {
    id?: string = null;
    content: string = null;
    createdAt: Date;
    isRead: boolean;
}
```

Adding service

The service uses mainly for getting data from API and processing the data.

Let's get started by opening the command line, then:

- Step 1: Type `ionic generate service`.
- Step 2: Type the file name `pages/notification/notification`.

You will see the new file named **notification.service.ts** in **src/app/pages/notification** folder, let's open It.

- Step 3: Update Its content with the following lines of code:

```
import { Injectable } from '@angular/core';
import { AuthHttpService } from '../../auth/auth-http.service';
import { BehaviorSubject, Observable, throwError } from 'rxjs';
import { catchError, map } from 'rxjs/operators';
import { environment } from 'src/environments/environment';
import { Notification } from 'src/app/shared/model';
import { HttpErrorResponse } from '@angular/common/http';

@Injectable({
    providedIn: 'root'
})
export class NotificationService {
    private notifications = new BehaviorSubject<Notification[] | null>([]);
    public readonly notifications$ = this.notifications.asObservable();

    private static handleError(error: HttpErrorResponse): any {
        return throwError('Something bad happened; please try again later.');
    }

    constructor(private autHttp: AuthHttpService) { }

    getNotifications(params?: any): Observable<any> {
        /*
        1. We configurated the variable `environment.API_URL` with value is `https://components-api-
dev.casewhere.com/dynamic`.
        2. authHttp is a wrapper class, that uses Angular HttpClient service inside.
        3. Make sure the web trigger is available and the returned data is the list of Notification.
        */
        return this.autHttp.get(`${environment.API_URL}/cw-mb-notification-get`, params)
            .pipe(
                map((res: Notification[]) => {
                    this.notifications.next(res);
                    return res;
                }),
                catchError((err, cautch) => NotificationService.handleError(err))
            );
    }
    updateNoti(model: Notification): Observable<boolean> {
        //Make sure the web trigger is available
        return this.autHttp.post(`${environment.API_URL}/cw-mb-notification-edit`, { ...model,
notificationId: model.id })
            .pipe(
                map((res: boolean) => {
                    this.getNotifications().subscribe();
                    return res;
                }),
                catchError((err, cautch) => NotificationService.handleError(err))
            );
    }
}
```

The bell on the home page

Let's show the number of unread notification items on the home page.

- Step 1: Open **home.page.html** inside **src/app/pages/home** folder.
- Step 2: Replace the div `notification-block` with the following lines of code:

```
<div class="notification-block ion-margin-top">
    <div class="bell-block">
        <ion-badge class="badge-cus" color="primary">{{notifications.length}}</ion-badge>
        <ion-icon class="icon-notification" name="notifications" routerLink="/notification"></ion-icon>
    </div>
</div>
```

- Step 3: Open **home.page.ts** inside **src/app/pages/home** folder.
- Step 4: Update with the below content:

```
import { NotificationService } from '../notification/notification.service';
...
export class HomePage implements OnInit {

    notifications = [];
    constructor(
        ...
        private notificationService: NotificationService
    ) { }

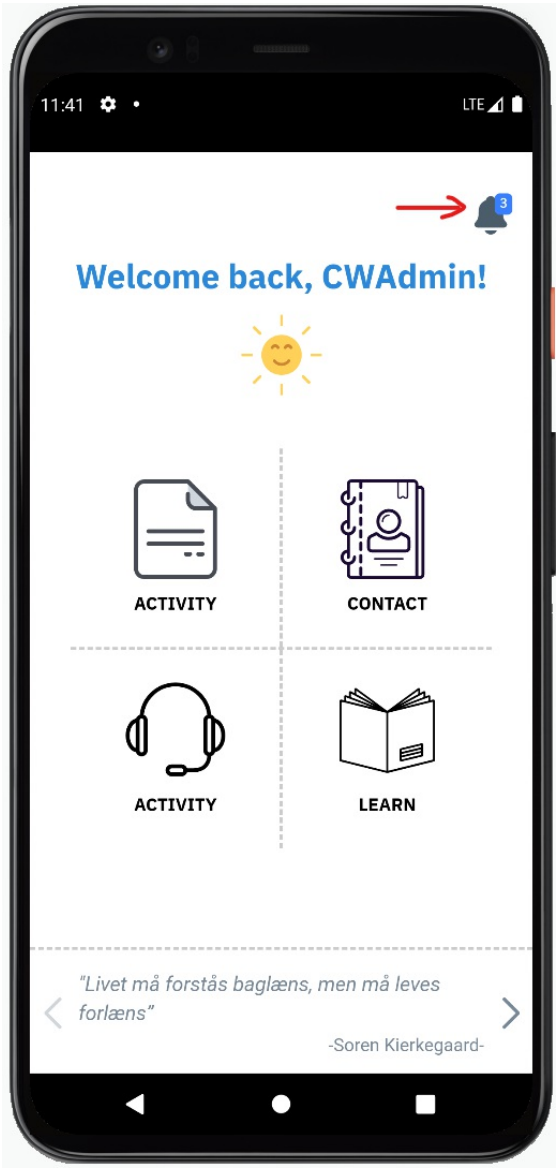
    ngOnInit() {
        ...
        this.notificationService.getNotifications().subscribe();
        this.notificationService.notifications$.subscribe((val) => {
            this.notifications = val.filter(noti => !noti.isRead);
        });
    }
}
```

Congratulations! You have finished the tutorial. Let's start the project and see the result.

Demo

Home page

The home page now has the bell icon with the number of unread notifications on the top-right corner.

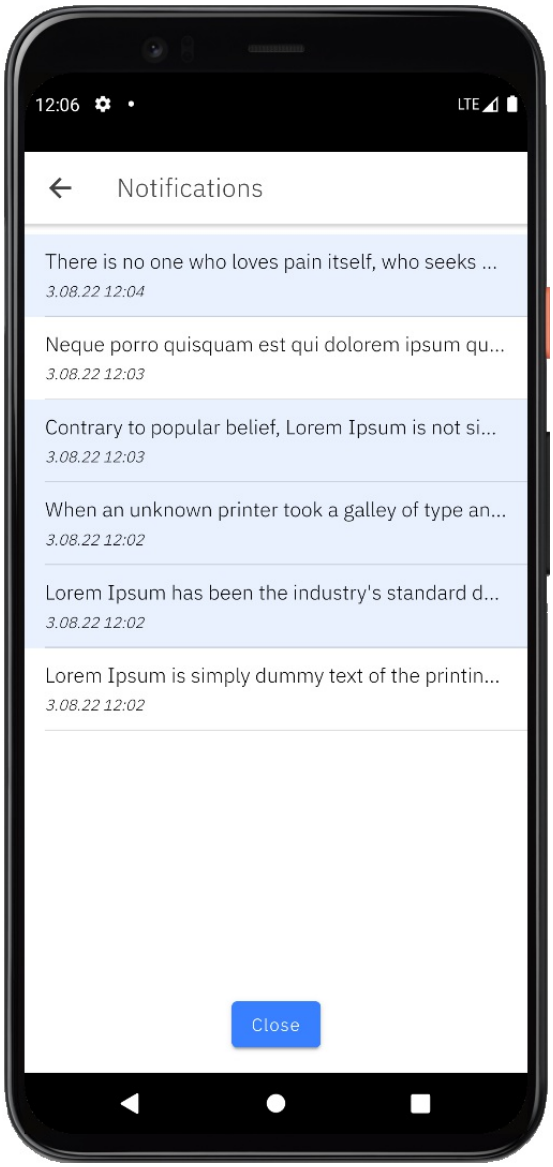


Notification page

After clicking on the bell icon, you will be redirected to the notification page.

This page gets data from the web trigger and then shows the data as the list, if the item is unread, It has a blue background.

If you click on the unread item, It will call a web trigger and change Its status. The background will be blank.



The close button will redirect you back to the home page.

Overview

Introduction

This documentation helps you build the roles & right feature. We focus on creating a directive, then use that for determining which feature the logged-in user can access.

Configuration

Introduction

This article shows how to implement the role & right feature.

Guide

Define roles and permissions

We need to define the types of roles and permissions that we want to grant to users.

Let's open `model.ts` file in `src/app/shared` folder and update add the following:

Build classes for the permissions of permission

- Step 1: Add a new folder inside `src/app` called `permission`.
- Step 2: Add a new file called `base.permission.ts` with the following:
- Step 3: Add a new file called `unknown.permission.ts` that implements the `PermissionBase` class.
- Step 4: Create a new file called `permission.factory.ts`, It will build the instance of the class of permission

Create the service

The service is to determine whether or not a user can perform a specific action. It can be accessed from any component or directive.

In the `permission` folder, create a new file called `permission.service.ts` with the following:

- It will firstly get the access token from `authService`. We have already implemented that service in the template.
- Then decode the token to get the roles, in `eSyncRoles` property, It is similar to:
- isGranted method receives a permission type as a parameter, iterates the user permissions and return true if found.

Create a directive

Let's create a new file called `is-granted.directive.ts` inside `src/app/shared` folder with the following:

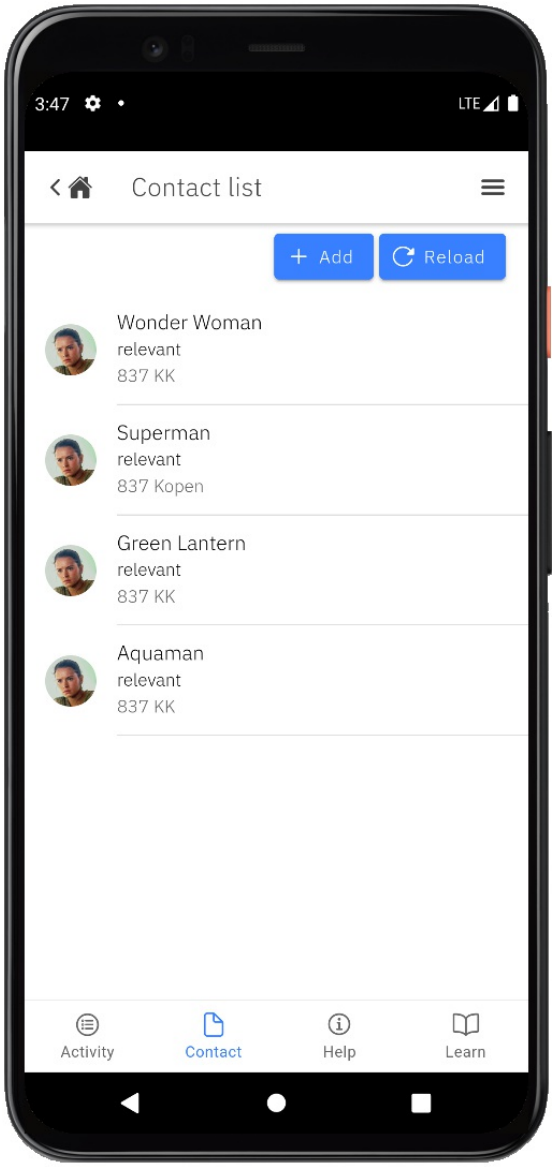
Directive usage:

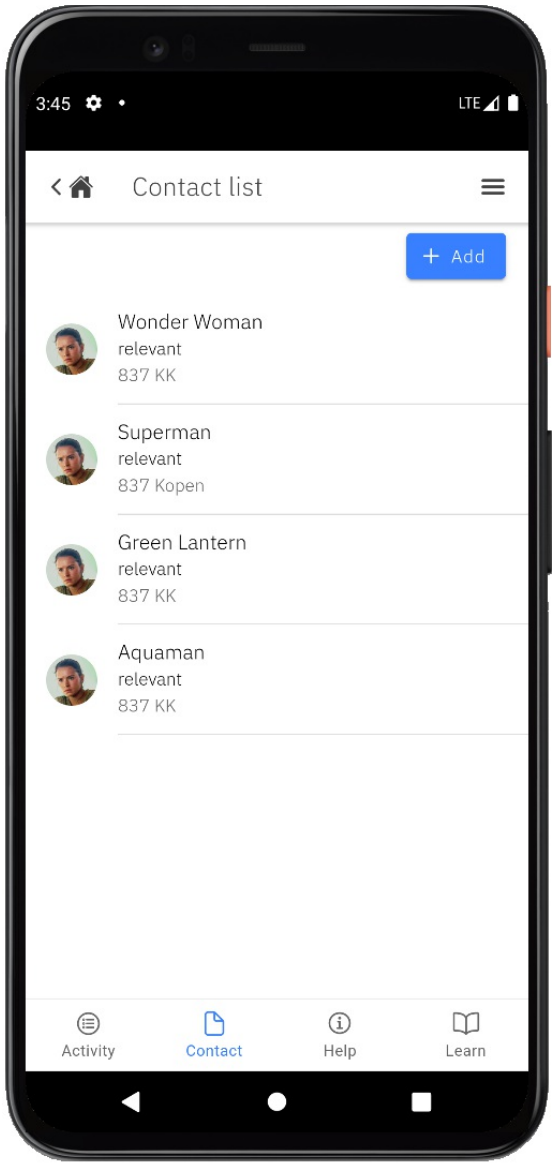
Update the reload button in the `contact-list.page.html` file with the following. It will show when the logged-in user has `CONTACT_CREATE` permission.

```
// This block will only be shown to granted users
<ion-button size="medium" color="primary" type="button" *appIsGranted="'CONTACT_RELOAD'">
    <ion-icon name="reload-outline" slot="start"></ion-icon>Reload
</ion-button>
```

Demo

If the logged-in user has 'CONTACT_RELOAD' permission, the reload button will show, and hide If It's not.





Overview

Introduction

This post shows you how to integrate form.io into the mobile template.

We focus on rendering the form, from the JSON format, and submitting the form.

Configuration

Guide

This section will help you create a simple report list, and the report detail with form.io integration.

Install package

To use from.io in our Angular project, we need to install a library called `angular-formio`.

Let's open the terminal and type the following:

```
npm install --save angular-formio
```

Define data model

We need a Report model with Its properties, then you can imagine the data structure.

- Step 1: Open **model.ts** file, in **src/app/shared** folder.
- Step 2: Add a new class with the following:

```
export class Report {
  id?: string = null;
  name: string = null;
  createdAt: Date;
  formDefinitionJson: any;
  formDefinitionId?: string;
  formDefinitionName?: string;
  actions?: object[];
}
```

A report item will have the example data like below:

Create Report List

Now, we need to generate a page named "report-list", to display the list of reports

- Step 1: Open the terminal and type `ionic generate page`
- Step 2: Type page name `pages/report/report-list`

> **Notes**: You might need to move the **reports** item to the top of the default item - inside **app-routing.module.ts**:

- Step 3: Open **report-list.page.html** in **src/app/pages/report/report-list** folder to update the UI

```
<ion-header>
  <ion-toolbar>
      <ion-title>Reports</ion-title>
      <ion-buttons slot="end">
          <ion-menu-toggle>
              <ion-button>
                  <ion-icon slot="icon-only" name="menu"></ion-icon>
              </ion-button>
          </ion-menu-toggle>
      </ion-buttons>
  </ion-toolbar>
</ion-header>
<ion-content class="page-report-content">
  <ion-refresher slot="fixed" (ionRefresh)="doRefresh($event)">
      <ion-refresher-content pullingIcon="chevron-down-circle-outline" refreshingSpinner="bubbles">
</ion-refresher-content>
  </ion-refresher>
  <ion-list>
      <ion-item-sliding class="report-item" *ngFor="let item of reports">
          <ion-item [routerLink]="['/reports', item.id]">
              <ion-label>
                  <h2>{{item.name}}</h2>
                  <i class="date">{{item.createdAt | date:'d.MM.yy h:mm'}}</i>
              </ion-label>
          </ion-item>
          <ion-item-options side="end">
              <ion-item-option [routerLink]="['/reports', item.id]">
                  <ion-icon name="create-outline" slot="icon-only"></ion-icon>
              </ion-item-option>
          </ion-item-options>
      </ion-item-sliding>
  </ion-list>
</ion-content>
```

- Step 4: Open **report-list.page.ts** to update the logic code
- Step 5: Open **report-list.page.scss** to update the styles

```
.page-report-content {
    .report-item {
        .date {
            font-size: 1.2rem;
        }
        .icon {
            color: #51cabd;
        }
    }
}
```

Create Report Detail

Now, we need to generate a page named "report-detail", to display the report item with formio

- Step 1: Open the terminal and type `ionic generate page`
- Step 2: Type page name `pages/report/report-detail`

> **Notes**: You might need to update and move the navigation item to the top of the default item - inside **app-routing.module.ts**:

- Step 3: Open **report-detail.module.ts** and import **FormioModule**

```
...
import { FormioModule } from 'angular-formio';

@NgModule({
    ...
  imports: [
      ...
      FormioModule
  ]
})
export class ReportDetailPageModule {}
```

- Step 4: Open **report-detail.page.html** in **src/app/pages/report/report-detail** folder to update the UI

```
<ion-header>
  <ion-toolbar>
      <ion-buttons slot="start">
          <ion-back-button></ion-back-button>
      </ion-buttons>
      <ion-title>Report detail</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content class="page-report-detail-content ion-padding">
  <ion-text class="ion-margin-bottom">
      <h3 class="report-name">{{report.name}}</h3>
  </ion-text>
  <formio [form]='report.formDefinitionJson' (submit)="onSubmit($event)" [refresh]="refreshForm"
      [submission]="submission"></formio>
  <ion-button color="primary" *ngFor="let item of report.actions" size="medium" type="button"
      class="text-transform-unset" (click)="clickButton(item)">{{item.ButtonText}}</ion-button>
</ion-content>
```

  - `<formio>` is a directive provided by Formio.
  - Based on the `actions` property, we will generate the appropriate buttons for the UI.

- Step 5: Open **report-detail.page.ts** to update the logic code
- Step 6: Open **report-detail.page.scss** to update the styles

```
.page-report-detail-content {
    .report-name {
        margin-top: 0;
    }
}
```

Create Report service

- Step 1: Open the terminal and type `ionic generate service`
- Step 2: Type the name `pages/report/report`
- Step 3: Update with the following:

```
import { Injectable } from '@angular/core';
import { AuthHttpService } from 'src/app/auth/auth-http.service';
import { Report } from 'src/app/shared/model';
import { environment } from 'src/environments/environment';
import { catchError, map, switchMap } from 'rxjs/operators';
import { BehaviorSubject, Observable, throwError } from 'rxjs';
import { HttpErrorResponse } from '@angular/common/http';

@Injectable({
```

```
@Injectable({
  providedIn: 'root'
})
export class ReportService {

  private reports = new BehaviorSubject<Report[] | null>([]);
  public readonly reports$ = this.reports.asObservable();

  private report = new BehaviorSubject<Report | null>(new Report());
  public readonly report$ = this.report.asObservable();

  private static handleError(error: HttpErrorResponse): any {
      return throwError('Something bad happened; please try again later.');
  }

  constructor(private autHttp: AuthHttpService) { }

  changeData(data: Report[]) {
      this.reports.next(data);
  }

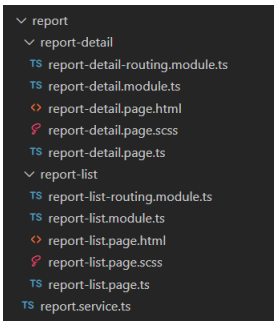
  getReports(): Observable<Report[]> {
      return this.autHttp.get(`${environment.API_URL}/reports`)
          .pipe(
              map((res: Report[]) => {
                  this.changeData(res);
                  return res;
              }),
              catchError((err) => ReportService.handleError(err))
          );
  }
  getReportById(id: string): Observable<Report> {
      return this.autHttp.get(`${environment.API_URL}/report?Id=${id}`)
          .pipe(
              switchMap((res: Report) => {
                  this.report.next(res);
                  return this.getFormDefinitionByName(res.formDefinitionName);
              }),
              catchError((err) => ReportService.handleError(err))
          );
  }
  getFormDefinitionByName(name: string): Observable<Report> {
      return this.autHttp.get(`${environment.API_URL}/get-form-definition?formname=${name}`)
          .pipe(
              map((res: any) => {
                  let report: Report = {
                      ...this.report.value,
                      formDefinitionJson: JSON.parse(res.formDefinitionJson),
                      actions: res.actions
                  };
                  this.report.next(report);
                  return res;
              }),
              catchError((err) => ReportService.handleError(err))
          );
  }
  submitReport(data: any): Observable<boolean> {
      return this.autHttp.post(`${environment.API_URL}/submit-form`, data)
          .pipe(
              map((res: boolean) => res),
              catchError((err) => ReportService.handleError(err))
          );
  }
}
```

- We assume that the **getReportById()** will return the report info and only `formDefinitionName`.
- Then we must call **getFormDefinitionByName()** with the `formDefinitionName` is parameter, to get `formDefinitionJson` and `actions`.

Demo

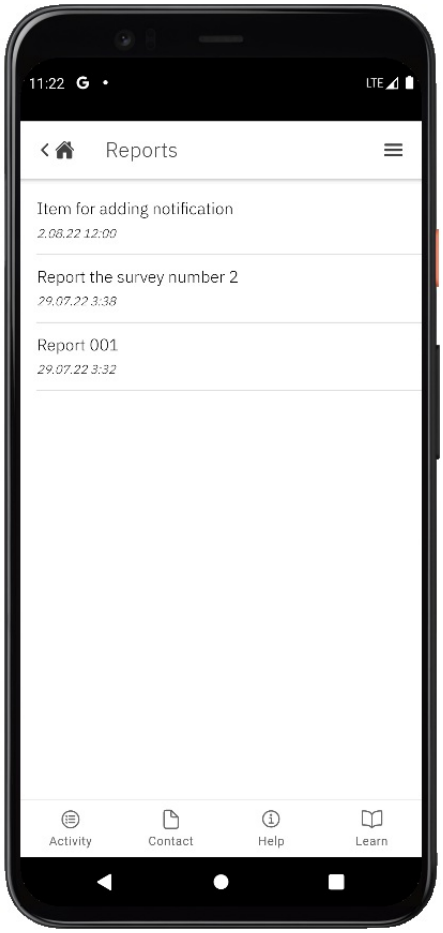
File structure

The structure includes a list, detail, and service.



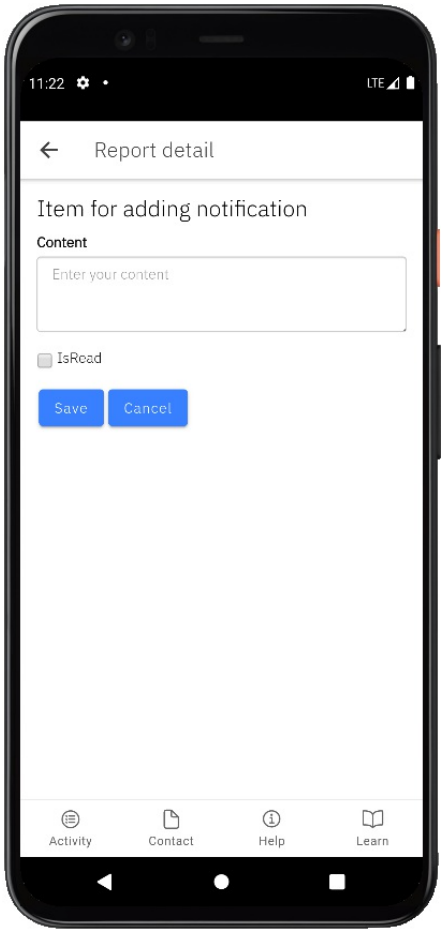
Report list

The list shows the simple information: report name and date. When clicking the item, you will be redirected to the detail page.



Report detail

This page shows the report name, form rendered by formio, and buttons rendered by `actions` properties.


11:22
LTE
Report detail
Item for adding notification
Content
Enter your content
IsRead
Save
Cancel
Activity
Contact
Help
Learn

Extracting an APK file

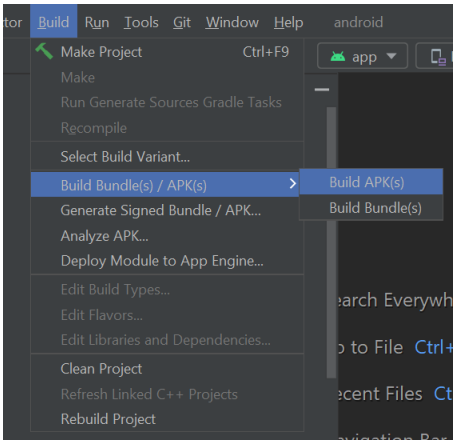
Introduction

Often time, you would use emulator while developing your Android app. In case you want to share the app with other people for their feedback, you can extract your app as an .apk file right in Android Studio.

Guideline

- Step 1. Open Android studio

Navigate to **Build > Build Bundle(s) / APK (s) > Build APK(s)**.



- Step 2. Android studio will take some time to build the APK.

Once done, save the file in your local.

- Step 3. That’s all. Rename the file and share to other devices.

Notes

Enable the "Allow unknown sources" option in your mobile phone's settings to install this .apk file.

Casewhere release overview

Casewhere versioning

Casewhere versioning has the form **MAJOR**.**RAPID**.**MINOR**.**PATCH** where:

- **MAJOR**: Major releases are made available when there are incompatible changes that require manual data migration. For example, version 2.0 is a major release where we migrated from RavenDB to MongoDB.
- **RAPID**: Rapid releases are made available approximately once each quarter and generally mark the introduction of new features and improvements.
- **MINOR**: Minor releases are made available when there is functionality added in a backward-compatible manner. There is no fixed schedule for minor releases, mainly when customer projects demand new features.
- **PATCH**: Patch releases are made available when there are backward-compatible bug fixes.

Casewhere releases

> **Notes**: login is required to download Casewhere builds.

VERSION	UPGRADE NOTES	DOWNLOAD
Casewhere 2.6	N/A	Azure builds | On-premise builds
Casewhere 2.7	Upgrade to 2.7	Azure builds | On-premise builds
Casewhere 2.8	Upgrade to 2.8	Azure builds | On-premise builds
Casewhere 2.9	Upgrade to 2.9	Azure builds | On-premise builds

Casewhere 2.9-series release notes

Beta. **ETA**: Q1 2025.

Highlights

ID	NAME	STATUS
76547	Data Class Inheritance	In Progress

Upgrade to 2.9

Please read this document Upgrade-to-2.9 to upgrade your Casewhere deployment to 2.9.

Download Casewhere 2.9 here (login required).

Casewhere 2.9 is compatible with MongoDB 5.0, and 7.0.

Changelog

2.9.0.0 - 01/08/2025

Features

ID	NAME
49164	In-App Documents: v2.9
53048	Workflow should update DO Ref only when the associated data class link into it
53772	Workflow Container Widget: Auto reload
53803	Datepicker General Validation Issues
53773	Casewhere Dialog: Toggle Big/Medium
108973	Group widget actions
108997	Reuse Enum Definitions Across Data Classes
109000	Custom Css Class For Cell
109001	Custom Css Class For Field Component
109005	Form navigation: Back/Forward and Save draft
109092	Quick Filters for Grid/List Widgets
109708	Boolean Quick Filter for Grid/List Widgets

Bugs

ID	NAME
62707	[workflow definitions] Allow edit special characters in workflow name
64031	[Export DataClass] Hard to deselect filter by Product / Tags
64694	[UI/UX][Tab container] Text resource list is cut when input tab container widget's title
65616	[Tab] PageScope in the Parameter rule works unexpectedly when setting Widget in the Tab Container
65624	Alias InputParameters in the Dynamic Workflow does not work when using it in the Workflow Definition
65625	Dynamic Workflow with Edit case does not work when removing the ProcessName field
65626	[Datepicker] Required message is shown unexpectedly when opening the Form Create
65627	[Date time picker] Error message 'invalid_time' does not show when inputing invalid time
65628	[Datepicker] 'invalid_date' message is not shown when setting default value a day before the min date
65629	[Hidden field] Error message appeared when setting DateTime.Now for Hidden Field
65645	[DevOpsPackages] Cannot remove access rule resource
108484	"Unsave change confirmation" popup appears on when setting default value for datetime fields
109157	[Import excel] Got an error when importing an Excel file with more than 24 records
109900	Remove the prefix "session:" in the Casewhere Admin and Hangfire
110060	Generate Chart error when pasting Code from the "Export to Mermaid" with some special characters

2.9.1.0 - 03/10/2025

Bugs

ID	NAME
110072	[Reported from SGAV][DateTime] Show error message 'invalid_date' when the user clears the value in the DateTime field.
110454	Confirming saving change does not work when hosting the workflow in a page or tab
110585	Workflow dialog buttons are incorrectly swapped between Maximize and Restore
110980	[Reported from SGAV] It's possible to save draft on non-editable activity
111543	[BUG reported from SGAV]Data Class has not updated after importing product package
111712	[BUG reported from SGAV] Cannot view audit log detail when inputting Decimal field = 0
111746	[BUG reported from SGAV] Attributes of Data class extension are shown twice on the primary data class

Casewhere 2.8-series release notes

Beta. **ETA**: Q1 2023.

Highlights

ID	NAME	STATUS
13400	Event Sourcing: Event Archive Tool	In Progress
136371	Event Sourcing: Compress Workflow Events	In Progress
108858	IPIIDataAnonymizationApi Extension: Anonymize specific cases and data objects	Done
109455	Audit Log for Read Activities	Done
109546	Platform General Improvements v2.8: Workflow Engine, Data Validation, Document Api, etc.	In Progress
110576	Web Trigger Extensions v2.8: Parameterized URI, File Streaming, Validation	In Progress
105893	Collaboration: Display Editing Users	Done
107544	Stop Activity	Done
108864	Enable Hard Delete for Specific Data	Done
122542	WCAG Compatibility	Done
76547	Data Class Inheritance	In Progress
123707	Data Class Extension	In Progress
137237	Changelog	Done
127500	Security Rules for Pages and Widgets: Accept Parameters	In Progress
137233	Widget improvements v2.8: Responsive, UI/UX	In Progress
124211	Grid Widget Personalization	Done
137229	Geospatial Query and Indexes	Done
94834	Loop Activity v2	Done

Upgrade to 2.8

Please read this document Upgrade-to-2.8 to upgrade your Casewhere deployment to 2.8.

Download Casewhere 2.8 here (login required).

Casewhere 2.8 is compatible with MongoDB 4.4, and 5.0.

Changelog

2.8.0.0 - 12/04/2023

Features

ID	NAME
112182	Support Run Build In Scripted Activity
110190	Validate non-persistence for the trigger workflow
114541	Upgrade the version of pdfjs library
113361	Support Tags for Worker Site Components
113368	Support Tags for Scheduler
111600	Support Tags for all features in Configuration Site
111695	Support metadata for custom data source via DataSourceService
116536	Support custom CSS class in Widget Actions
116539	Support add object with Id value on IDataApi.Migrate method
101778	Debugging Server Rule
114543	Improve Monaco editor with intellisense API code
103836	Form Activity: Support Multi-Object Data Source
116530	CMS Worker Site: Should Force Login When Anonymous User Browse An Authorized Page
105843	Web Trigger Parameterized URI
110575	Web Trigger URIs Support 4 Levels
108859	API to Anonymize Specific Cases
108860	API to Anonymize Specific Data Objects
109954	Extend IDataAPI to enable hard delete for Specific DataObjects
105895	Display Editing Users in Edit Dialog
110129	[UI/UX] Cannot refresh UI controls after updating workflow definition/process
110385	[Workflow] Update isBackground status of workflow in some case
111601	Enable Audit Log for Data Source
111602	Enrich Audit Log with Searchable Information - User Claims

ID	NAME
111606	Audit Log Viewer: Support Data Source Audit Log
116590	Manage Rule Dependencies
117310	Expose DSL.DataApi.Inc
70099	Expose DSL.Api.ILockApi
110244	Implement Configuration Api read Plugin Information
110239	Implement Configuration Api read Rule Information
117776	Support Create index for dynamic object attribute of Data Class
117806	Audit Log Viewer: Filter and search
118794	Disable constraint CanCreateDataObjectInCase
119484	Enhance WorkflowContainer to support FormIO
119576	Write detail log when evaluating rule failed
120087	Support RuleEngine Api
120158	Enhance Data trigger rule in 2.8
120791	Plugin Configuration File
121280	Add more page size on Grid control

Bugs

ID	NAME
84528	Can't retry the transactional workflow when it's failed
111763	[Rules] Edit Rule popup is opened incorrectly when coping Rule
111083	[Validation rules] Code is not saved after edited and click save
110771	Don't include Casewhere.Storage.GridFS in the Azure package release
112117	[Rules] Resources are copied when clicking on Copy button Rule
116792	[Audit Logs] Automation Tests and schedule job failed due to UserClaims is null
117277	[Form Activity] Getting error when a form contains over 2 components binding to a field
117419	[Worker site component] Cannot select item in the Existing dependencies popup

ID	NAME
115009	Can't retry the transactional workflow when it's failed
118065	[Web Trigger - Rule] Missing Usage Type rule when opening Server Rule popup
118809	[Form activity] Validation Async Trigger rule is not shown error message when clicking on Save button
118830	[Audit Log] Can not view audit log with array (foreign key)
120112	[Audit log] Can not audit log Array Document Attribute of data class
121213	Computed Attribute not work with predicate code script
121196	[Workflow Definition] Reference component data source when cloning Workflow
121346	[Widget] The Widget was cloned that references rules in the Widget action

2.8.1.0 - 19/05/2023

Features

ID	NAME
119913	Audit log viewer page
119912	Performance log viewer page
122055	Support an option for numeric text to disable decimal format
121699	Support Copy CMS Master Page
121644	Support export/import resource feature on Configuration Web
117784	Support Create index for data source via IDataEnricherApi.CreateIndex method
122056	Data Attribute Validator should show attribute name in exception
121956	[Configuration] Support more description height in Edit Product Dialog

Bugs

ID	NAME
122694	Hotfix - Move to product RuleDependencies should not effect to primary resources
122373	[Workflow Definitions] System error appeared when coping resource
121624	[Products] Scale up the description field in the Edit Product popup

2.8.2.0 - 30/09/2023

Features

ID	NAME
116579	Support show icon and badge number on MenuItem of default theme
124490	Support Tab Widget can configure which tab added to menu when less width
123516	Support Search page on the combobox when adding page action
123838	Support Search Data Source in the "Configure Data Source" popup in the Workflow Form activity
124515	Support call back function on custom-widget action
123456	Support an option to disable enforce constraints on IDataApi.Migrate method
123673	Support API to verify lockId is locking or not
123670	Support a new form event FormSubmitting in FormActivity
124494	Improve Import Process in Configuration Admin
123669	Extend the new ICounterApi.Next(string name, int step) with increment value
123930	Export Data Class should include built-in data source
123480	AuditLogViewer should show details value-change of object attribute value
117879	Dynamic workflow definition
121274	Find and improve slow part on embedded workflow
122544	[WCAG] Set <html lang=""> flexible with the language user choose to display
123484	[WCAG] Should read Error Suggestion when inputting invalid value
122552	[WCAG] Set Text of 2 rows are not overlapped when increasing text size (200%)
122546	[WCAG] Set keyboard as Mouse click on the table header (focus => Enter)
122553	[WCAG] Set focus and get out to finish choosing multi select component in toolboxes
122550	[WCAG] Set aria-label to explain the icon button
122549	[WCAG] Set all the links need to be underlined on hover/focus as default style
122551	[WCAG] Set all labels marked with * as aria-label='First name, required'
124180	[WCAG] Fix issues left on the new version
122547	[WCAG] Buttons in the filter popup should be focused when pressing Tab
122545	[WCAG] Need a mechanism to display a tooltip when using the keyboard to navigate on the table cell.

ID	NAME
123882	[WCAG] Need a mechanism to Select record and get out on the cell table
123740	Support Responsive on some widgets
123861	Support a Diagnose and Troubleshooting page in Configuration Web
123287	Support API to get list of index of DataClass via IDataEnricherApi.GetIndexes method
123289	Support API to drop index of DataClass via IDataEnricherApi.DropIndex method
124212	Support User-defined filters
123885	Support resize column in Grid
123940	Support Export and Import single Resource for the rest features
125345	Export Workflow Definition should include Component DataSources
125640	Manage Product of Access Rule Resource
125900	Support Export and Import on AccessRuleResource
126709	The API of Widget filters should validate security of widget
123876	Update the UI and filters on Audit-log Viewer Admin page
127504	Support the new security policy rule compatibility with access rule
127505	Apply security policy rule for Page
94835	Simplify the Filter Condition for Loop Activity v2
123749	Support free-text search for object attribute type
107544	Stop Activity
119908	Support downloading large logs via streaming
119914	Runtime data log viewer page
122487	Refactor code of SystemLog Viewer page
122539	Support download large PerfLog as streaming
123883	Update UI and Filters on Perf log viewer Admin page
116580	Remove text-transform css of modal-title
127487	Web Trigger Rule support custom response error (status code, reason phrase)

ID	NAME
127605	Form activity support Close button
127633	Protect Workflow Container with security policy rule
133565	Support method to terminate user session by ID
127454	Support free text search for the rest attribute types
127506	Apply security policy rule for Widgets - CMS Widget
127618	Apply security policy rule For Widget Actions
130706	Web trigger Api can get browser agent
130723	Remove un-necessary properties on Field Label
131105	Support XML ContentType on Web trigger
131285	Support API to translate text resource key TextApi.Translate

Bugs

ID	NAME
133335	[List widget] The list widget has displayed as empty on the Worker site
117662	[Pages] Missing "Query Type" label in the General settings on the page
133116	Error message appeared when changing setting client rules on the Form activity
122269	Can not render Widget after entering Name with special charts ()
122388	[Web Triggers] Friendly Uri is bypass validation unique after coping Web trigger
123909	Cannot filter with the filed including multi select combo box
125590	Error message "Object reference not set to an instance of an object" appears when Adding Dynamic Workflow step after Form activity step
125956	Column has not displayed data when setting the empty name header in the Custom settings widget
126393	The cw-numeric-input will render the variable as null when setting RepayableAmount = 0
133153	[Page-Widget] Use the method _pageWidgetService.GetWidgets() but it returns an empty list.
118878	[Text resource] Can rename Text resource duplicate
127095	[Form activity] Should add "return true" by default in the Event -> FormSubmitting
110940	[Container widget] User is forward to unauthorized page when click on tab which is container not permission widget

ID	NAME
133455	[Caseworker][All Applications pages] Object reference not set to an instance of an object when opening an application on any Applications pages
130750	Main WF still completed although embedded WF is not completed
132555	Can move to next step although embedded WF is not completed
132012	[CMS] Cannot edit/delete event when update widget name
131452	The tooltip shows the label of the column and the content when hovering on it
130691	[Widget template] Cannot edit name widget template as expectedly
130684	Duplicate key error collection when importing package
121003	Cannot deserialize the current JSON array

2.8.3.0 - 10/01/2024

Features

ID	NAME
103092	Runtime Api for managing schedulers
122679	Refactor code of cloning resources
123544	Support "Cancel All" button on the Tests page
123708	Support create/edit Data Class Extension in Configuration.Web
123799	Support Data Class Extension in Runtime
123808	Support build and view source on expression rule
123812	Support create custom data source with extension data class
123831	Show attributes of extension data classes in Form Data Source
124213	Support User widget preferences
127490	[WebTrigger] Support return a file stream
128118	Create new Classic widget container support Parameters
128210	Support icon and counter on Tab Container Item
128360	Built-in data source for extension data class
129132	Support new method to evaluate security policy rule via RuleAPI

ID	NAME
130707	Support getting page, widget and widget actions on IConfigurationApi
130708	Provide client-id for pages, widgets and widget actions
130710	Customize Script and CSS on Page Configuration
130711	[Widget] Support auto height fit by content or container
130732	Workflow step pane: Horizontal or Vertical
130734	Workflow step: Option to always show regardless of included activities
131953	WCAG - Support config landmark on configuration widgets
132015	Support adding hidden columns on Grid Widget
133571	Monaco editor field support XML, Plain text languages
134056	[Geospatial] Support to create geospatial indexes
134060	[Geospatial] Support a way to search query on geo data
134062	[DataTrigger] Provide different field and value information
134240	Add Troubleshooting page menu into Configuration
134350	Should reload or highlight data on Form Close Button
134353	Configuration Web should support to view source code history
134726	Data source support multiple sort in Grid
134864	Event Sourcing: Background Job to archive Workflow events and read data
135060	Event Sourcing: Create install shell and document - How to use run archive tool
135062	Event Sourcing: Load Test Casewhere while the Event Archive Tool is running
135955	Event Sourcing: Compress Workflow Events
136292	[FormActivity] Customize label template for form components
136305	Event Sourcing: Load Test Performance Casewhere with Compress Workflow events
136431	DataApi support force load data from DB
136455	[FormActivity] Support double-click to open edit custom html content
137761	[IntegrationAI] Support to upload training data in Casewhere Doc Pipeline

ID	NAME
138163	[TP-Tool] Automate update Feature description
138171	[TP-Tool] Support translation in Automate update Feature tool
138334	Support to compare two diff change logs of resource in Configuration Web
138337	Moving all client-libs of CMS from Admin to Workersite
138346	Load test and benchmark before and after optimize Workflow engine
138366	[TP-Tool] Write a document on how to run automated update Feature description.
138432	Support new system attributes CreatedByName, ModifiedByName
138479	[FormActivity] Support unsaved change confirm on Cancel and Close buttons
138998	Upgrade TinyMce library
139033	Support Invalidate workflow definition cache by setting in web.config
139455	Auditlog viewer should export excel with large data
139467	Support updateMany API to update many dataobjects
139468	Support lock and check API method
139473	Support call back and update query parameter on Widget Action
140708	Update datetime filter with time set as 23:59 of end date
140940	Leak memory in ManagedApiFactory
141045	Leak memory in DOContextualDataUpdatedEvent
141355	Support Text-Selection GridView in Workersite Advanced Setting
141374	Grid view support copy cell value
141839	Support confirm save change on Workflow Widget Container in Page

Bugs

ID	NAME
117175	Cannot translate workflow display name when put workflow in workflow container widget
135719	Worker site component not work and overrided by Theme
136475	TypeError: Cannot read properties of null when deleting item in the dropdown list

ID	NAME
136879	Cannot run workflow with multiple forms and MinifyScript = true
138009	[Audit Logs] Auditlog has not encrypted the attribute set protect
138260	[WebTrigger] Can not get form data when using multipart form data webtrigger
138342	[Date Time picker] Error message shown when changing Min/Max of Time in the Data Time picker
138345	[PDF Viewer component] Sometimes cannot view letter on the PDF viewer
138424	[Custom Data Source] The Title of the Foreignkey attribute in the Custom data source is not displayed text resource key when changing the title from text resource key
138429	[Date Time] Date time picker component has not been updated correctly after changing date
139521	[Form Activity] Confirm unsaved changes popup has not shown when pressing the Esc key or Close icon
139583	[List View widget] Can not scroll down the list to load all data from page 2
139591	[List view - Sort] Lost data when clicking on the sort icon
140498	[Expand filter] Text of Collapse button still remains "Expand"

2.8.3.1 - 26/01/2024

Features

ID	NAME
140708	Update datetime filter with time set as 23:59 of end date
141839	Support confirm save change on Workflow Widget Container in Page
142880	Change priority of workflow test to Normal

Bugs

ID	NAME
141509	[Datetime] End time is displayed wrong hour on the Client site when saving date time on the Form activity
142937	[Data Class - Extension] "CreatedByName" and "ModifiedByName" fields are duplicated on the DataClassExtension
143111	[Datetime picker] Minute control is not shown error when inputting invalid time
143357	Missing ViewSource button on toolbar in the new version of TinyMCE
143470	[Data Classes - Enforce Constraint] Enforce Constraint foreign key is updated successfully when DO is deleted
143576	[Page - Grid] Can not filter encrypt attribute in dataquery of grid datasource

ID	NAME
143654	[Datetime - ESC] Error message appeared when pressing ESC key on the WF including datetime picker with enable seconds Time
143711	Workersite cannot duplicate tab after an hour
143790	[Date Classes Extension - Index] Extension data class has not load fields in the Primary Data Class
144166	[Datetime- Max Date] Max date does not work as expected

2.8.3.2 - 29/02/2024

Features

ID	NAME
145031	DataSource show the foreign key name on field property
143913	Data source parameters on query script
144086	[Datasource] Support collation setting on Index
144087	[Datasource] Support to specify a default collation for DataSource
144088	[Datasource] Support to specify a default collation for a data query
144079	[Datasource] Default collation configuration
146094	Very Slow performance when importing data class
143941	Async Workflow Job from User Principal should be log as trigger user not CwRuntime
143924	Support Data source parameters on DataSourceApiQuery
139461	Support Priority on queued workflow in LoopV2
144374	Support confirm save change on close tab or navigate page
142183	Grid column alignment configuration
142187	[FormAcitivty] Support text editor on tool box

Bugs

ID	NAME
142280	[Form - Select component] Unsave changes popup is shown when clicking on the Cancel button with the Form
146070	DataClass ensure index will be cancel if import sequential data class
144063	[Form - Numeric textbox] Unsave changes popup is shown when pressing on the Tab key to navigate to Numeric textbox

ID	NAME
145091	CreatedByName can not be load when using dataApi.Load
144057	[Page-Component datasource] Can not filter encrypt attribute in dataquery of grid custom datasource
144966	PDF viewer is not stable
142932	[WCAG] Cannot tab to page number in paging control of list widget
142509	[Filter popup] Can not Close filter popup when displaying multi-select component
142930	[WCAG] Narrator read non-sense text for paging control on List Widget
145411	[DateTimePicker] Hover on start date auto update the end date
134335	[WCAG] Some WCAG issues on Date picker
144426	Displaying 2 landmark "search" on toolbar
145169	[Datetime picker] TypeError appears when inputting time only

2.8.4.0 - 29/03/2024

Features

ID	NAME
146341	Support hint index on query on Data Source
144724	Support filter on foreign key attribute of lookup data class
145075	Workersite support customize metadata header
141381	Expose client API to set Page and Workflow title
114415	Event Trigger: Apply common component to refresh the Workflow/Process controls properly
114417	Workflow Activity: Apply common component to refresh the Workflow/Process controls properly
114418	Workflow Monitor: Apply common component to refresh the Workflow/Process controls properly
114419	Workflow Container: Apply common component to refresh the Workflow/Process controls properly
114420	Loop Activity: Apply common component to refresh the Workflow/Process controls properly
114422	Widget Action: Apply common component to refresh the Workflow/Process controls properly
114421	Scheduler: Apply common component to refresh the Workflow/Process controls properly
113393	Upgrade Tags in Files to the new design

ID	NAME
136111	Optimize loading Workflow Container in Page
140704	Extend IDataAuditAPI to enable hard delete for specific log data
140707	Extend new event WorkflowDataSourceSetEvent in Workflow
140705	Extend IDataAuditAPI to enumerate auditlog data
146559	Smart sorting with collation
146466	Enhance Rebuild Index Maintenance endpoint
141376	[FormActivity] Support custom template for select component
141190	[Widget Action] Support auto-refresh, icon button on Widget Actions
141377	[Widget Action] Support config widget call-back handler
141380	[Widget Action] Support update widget parameter
142510	Workersite enable un-used scripts in Advanced setting
148453	Enhance query data source with collation of Field

Bugs

ID	NAME
144100	The dropdown opens when the user touches it while scrolling
139896	[DataClass-Extension] Cannot load data of Array(ForeignKey) when using DataApi.Load method
142199	[Form activity] TypeError appears when clicking on the Back button multiple times
142251	[Datetime] ReferenceError appears when inputting/deleting time and closing form/dialog
110540	Don't log access token in the system log
143952	[Data Classes] Support Searchable for CreateByName and ModifiedByName
141041	Can not Import dataclass with renamed attributed
142928	[WCAG] [Datetime] Narrator read the time uncorrectly
142189	[Datetime] letters keyboard on the real device is displayed instead of numeric keyboard when inputting time on the date time picker
146921	Unexpected data object is created when editing form with existing data object mapping

2.8.4.1 - 02/04/2024

Features

ID	NAME
149563	[Text Editor] Support select font and font-size on toolbar

Bugs

ID	NAME
148818	The horizontal scroll bar can not be scrolled by mouse drag
149564	Can not get the filter values with custom script datasource
149528	Request Data is duplicated when loading page
149401	Check box list did not clear checked when assign null or empty array
147150	The query can not get the value of data source parameters

2.8.4.2 - 23/04/2024

Features

ID	NAME
147074	Support validated and validating events on form components
149833	Support DeleteMany and enhance updateMany API
147056	Support paging config on Custom Widget
149832	Improve speed of loading data in Widgets
148951	[FormActivity] Enhance custom template for select component
149831	[FormActivity] Support config Select2Option on Select Component
149571	[Datasource] Support allowDiskUse setting

Bugs

ID	NAME
148861	[Workflow Monitor] Can't trigger alert workflow with the transactional workflow when it's failed
148944	[PLF][WCAG] [2.4.2] Page title is the same for all page
149536	"field-data-source" request is duplicated when filtering values in the Filter popup
149773	[Workflow monitor] Cannot trigger alert Workflow update

ID	NAME
150154	Hint query can not work with full text search when setting index name and custom collation in the data source
150536	Too many read user session message in background job
150704	[DataApi] UpdateMany, DeleteMany should not throw exception when not found any data
150796	IdentityApi.Impersonate got exception when have a lot of requests

2.8.4.3 - 20/05/2024

Features

ID	NAME
151466	Display order steps on Worker site when using "Always Show" setting
150405	ForceEmit event on transactional workflow
150846	Paging for custom widget
150832	Improve loading data with configuration on Widgets
149575	Enhance alert workflow monitor
150403	Support set datasoure parameters on Enumeration Query
150705	Support changelogs for datasource in Configuration site
151649	Support searching by letters in the Multiple select box

Bugs

ID	NAME
150539	Correct write Log.Error with exception
150760	Sync DataTrigger failed with computed field when using dataApi.Push and pull with tracking data change
150814	System Log leak memory
150818	[Edit Grid Settting] Cannot open Data Source
150843	The dropdown is too long making it difficult to select on Worker Configuration
150844	Client API to set Page and Workflow title should use angular binding
150845	The label in configuration should stick on the convention
151125	Don't reset identity after impersonating the user in previous request

ID	NAME
152055	[LogViewer] Fail Security when downloading logs
152202	[Workflow Monitor] Data does not loading as expected when click on the Next page icon

2.8.4.4 - 04/06/2024

Features

ID	NAME
153409	DataAPI: Support Unprotect data for enumerate data source
150374	[Grid widget] Enhance export data on grid widget
151694	[Grid widget] Export Excel for the columns combined from multiple fields
151479	[Grid Widget] Download Excel file for grid widget via the form request
153127	Widget: Separate export data endpoint for client side

Bugs

ID	NAME
153408	[Data Class] Cannot search CaseState in the Data Class Extension when creating Indexes
153507	[Export] Export failed when filtering with queryParameter
153510	[Export Excel - Server] The filename not translate
153222	Throw exception after running by CWRuntime

2.8.4.5 - 14/06/2024

Features

ID	NAME
153600	[Grid widget] Support formatter functions in Export Expression from Server

Bugs

ID	NAME
153651	Cannot Export excel file when setting Header column empty
153659	[Text resource] Cannot translate Text resource for the Enum column when switching languages

2.8.4.6 - 02/07/2024

Features

ID	NAME
154830	Skip data change notification on updating data
155530	Support Hint Index For Counting

Bugs

ID	NAME
154574	Indexes of Data Class Extension are not created in the Primary Data Class in DB
154892	Login flow: Reduce the number of direct updates into database in the login flow
155088	[Perflog] Search result has not ordered when searching with ResponseTime

2.8.5.0 - 15/07/2024

Features

ID	NAME
146834	Support view multiple products
146908	Auto-check the relevant attributes of the workflow definition based on the creation context.
146930	Index manager page
152193	Enhance rebuild full text index
148342	Scheduler: Only support the creation workflow in the scheduler
149050	Support label setting for the Time in DateTime picker component
150847	[FormActivity] Add help link What options can be used of Select2Option
151569	[WildcardIX] Support to create a Wildcard Index on All Fields
151764	[WildcardIX] Compound Wildcard Indexes
151692	[WildcardIX] Create a Wildcard Index on a Single Field
151763	[WildcardIX] Support allow user can config wildcard projection in Options
152149	Support Evaluate Index With Options
152052	Workflow step: Show All Steps combine with Always Show
152245	[WCAG] [2.5.3 & 1.3.1] Missing accessible name for dialog
153795	[Grid widget] Export Expression Editor: Improve UI

ID	NAME
154487	Workflow: Store and load the output workflow
148456	Workflow engine can run without storing events to DB
151049	Configuration Site: Config and update event trigger settings in real-time
152156	Configuration Site: Configure data trigger settings per Casewhere instance
152157	Configuration Site: Configure job worker settings per Casewhere instance
152088	Customize data loading timeout in Widget
147926	Custom filtering values
146830	Support query parameters on custom field of data source
150706	Add Description in Event trigger configuration
153228	Update Changelog of Grid Widget Configuration
156601	Remove tracking document changes in WorkerApi

Bugs

ID	NAME
146433	[Export excel] End Date time is displayed wrong format when exporting data
148770	Missing rebuild some indexes due to incorrect checking index definition
149770	[Workflow monitor] Missing Open link on the Workflow Definition field
150293	[WCAG] [4.1.2] [2.1.4] List view sort button missing role, label
150294	[WCAG] Error on Pagination on List
150297	[WCAG] [4.1.1] The screen reader doesn't read long-running task/error/success
150680	[DateTimePicker] Min max error state of the time picker is not cleared
150681	Enrich related data objects when changed still not work as expected
152909	Date time component is shown wrong date after inputting datetime = "10/27/24 23:00:00"
153743	Improve Search function on the Workflow Definition combo boxes
156600	[Data Class - Index] Error message appeared when loading Index in the Data Class
156604	[Date time picker] Broken layout on the Date time picker when viewing it in the Audit log filter

ID	NAME
156400	[Workflow - Widget Action] Can not edit Widget action in the Form builder

2.8.6.0 - 20/08/2024

Features

ID	NAME
158751	Grid widget: Improve full-text search data
158757	Grid Widget: Improve counting data
159029	Support to return empty result in Data Query
159306	Grid Widget: Separate loading data and count requests
159383	Grid widget: Improve counting of full-text search data
154816	Show confirmation dialog when importing product
156357	MongoDB Driver Thread static buffer is already in use
157661	AuditLog: Enhance streaming download excel file
158455	Should skip validation when exporting data from server with token
158622	Enhance IdentityApi methods with claims
158642	Configuration Site: Hide/Show Hot Update Settings
158652	Configuration Site: Enable and disable Run All Unit test
159279	Support Index Size in Index Manager
159306	Grid Widget: Separate loading data and count requests
159475	Support Add Index for Array Object Attribute
159875	Reduce Logs in Anonymization Task
159960	Update PIIAnonymization to anonymize PII data in TempData, Output of workflow
160005	Non-Persistent Workflow Must Not Persist External DO Input
160269	Apply Caching for Authorization Code

Bugs

ID	NAME
150298	[WCAG] Issue on CW datetime
156421	[Date time picker] Label time still displayed on the Worker site when setting Off "Enable Time"
156448	User info of Audit log is not correct in async event trigger
159225	[Index] Index is invalid when creating index with attribute is encrypted
159543	The import dialog is truncated when the content is too long.
159961	PII Anonymization: Got the error when anonymizing the workflow
160151	[2.7.1.10 to 2.8.6] The CMS Page id is rendered wrong
160163	[2.7.1.10 to 2.8.6] Cannot LoadDataObjects If SourceWorkflowId is empty

2.8.6.1 - 24/09/2024

Bugs

ID	NAME
161942	Can not start API after installed CW 2.8.6
162532	GPDR Repositories do not use correct connection

2.8.6.2 - 10/10/2024

Bugs

ID	NAME
163244	Anonymize failed for specific DO with no change values
163246	Change Anonymization Order
163601	Memory leak when updating multiple times with many large DOs

2.8.6.3 - 22/10/2024

Bugs

ID	NAME
154711	Workflow can not run sometime due to EvaluateVisibleComponentRule
163890	[Memory Leak] LoadWidgetQueryHandler and DocumentSession can not release
163954	AuditLog Enumerate methods did not use correct collation on queries

2.8.6.4 - 14/11/2024

Bugs

ID	NAME
163941	Anonymization Task did not change values of Auditlog, DOSnapshot, Event
164212	Minification failed when loading the Worker site with Js/Css bundles
164644	[DateTime] DateTime picker control does not clear the value when setting date = null
165029	App crash when try to release ExcelWriter after closed outside stream

2.8.7.0 - 20/12/2024

Features

ID	NAME
156642	AuditLog: Data enriching - configuration
156643	Auditlog: Data enriching - runtime
154919	Manage DevOps Packages: CRUD
154925	Add/Remove DevOps Package Resources
155085	Import/Export DevOps Package
155086	Lock/Unlock DevOps Package
161968	Enable Change Logs for DevOps Package
165173	[ConfigurationApi] Support Get Data Class By Name and Id
164396	Support IP Whitelist on Document Sharing API
164449	Disable Workflow Logs Setting
159438	Support Widget Action Rule Usage Type
158908	Security Rules for Workflows
156642	AuditLog: Data enriching - configuration
156643	Auditlog: Data enriching - runtime
156645	AuditLog: Try to translate the custom log entries
156639	AuditLog: Deprecate log-reading rules
156644	Auditlog: Log viewer showing enrich
157661	AuditLog: Enhance streaming download excel file

ID	NAME
155553	Extend DslApi with ctx.Error
155554	Custom Form validation error message
155558	Extend Web trigger rule results
160602	Support Refresh Datasource on Widget Setting
163135	Support Search Log with Local or UTC Time
164026	Enhance @diff parameter in Data Trigger
165418	Speed up saving progress in Workflow Definition
157550	Enhance Save Draft Workflow
158638	Configuration Site: Add more constraints on event trigger settings
158639	Configuration Site: Add more constraints on data trigger settings
158641	Configuration Site: Add more constraints on job worker settings
158827	Workflow should not be accessed after configurable time
158896	IdpConnection: Support to download the metadata file
161085	Enriching Search Data Should Skip DOId Value
161681	Publish workflow should show detail log
163098	Refresh API for Page and Widget
163128	Enhance Index Manager Page
163129	Enhance Runtime LogViewer Page
163131	Enhance System LogViewer Page
163137	Reorganize Workersite Edit Dialog
163177	Reorganize Grid/List Widget Settings
143945	Workersite support inject CSharp script on loading
162635	[FormActivity] Support refresh method on Select control
163714	Option to Skip Posted Data

Bugs

ID	NAME
154764	[Data Class - Index] Fulltext index rebuild not include extension attribute
156086	Form data cannot load unsaved data after validation fail in Add dialog
159979	[Save filter] Saving a filter with a Date field and the Equal operator does not work
161308	[Search] DB query has extra character specials
164025	Workflow Navigation API: Custom Save Move the Workflow Backward
109840	Got the error on auth/signonconsumer
153084	Select box show the required message when opening form
164656	Perflog not persited invoked plugin failed
164796	[Datepicker] Cannot reset required validator time field when setting condition to non-required field
165248	[Save Draft] Missing red borders and some error messages required fields when clicking on the Save Draft button
166038	[Index Manager] Missing confirm popup when clicking on the "Clean up unused indexes" button
166257	Send unauthenticated users to unauthorized is not user friendly

Casewhere 2.7-series release notes

Status: Done. **Date**: Q3 2022.

Highlights

ID	NAME	STATUS
100268	Workflow Debugger	Done
102039	Workflow Transaction	Done
84684	Logging Facilitating Alerts	Done
102287	Data Attribute: Default Value, Computed Attribute, Enforce FK	Done
102829	Deep Link for Casewhere Resource in Admin Site	Done
102491	Hot and Cold	Done
102924	Web Trigger: Restful Extensions (See more here)	Done
102925	Form Activity Component Extensions 2.7: More components, more validation settings	Done
102040	Platform License	Moved to v2.8
102929	Widget Extensions 2.7: Default Format for Datetime, Configure Grid Row Height	Done
78568	Workflow Concurrency Control	Done
105819	Text Resource Namespace	Done

Upgrade to 2.7

Please read this document Upgrade-to-2.7 to upgrade your Casewhere deployment to 2.7.

Download Casewhere 2.7 here (login required).

Casewhere 2.7 is compatible with MongoDB 4.2, 4.4, and 5.0.

Changelog

2.7.0.0 - 25/11/2021

Features

ID	NAME
97161	Description for Data Class
96564	User can customize the display format of Log Viewer in Plain Text mode

ID	NAME
101750	New Logging Level: Verbose
101067	New Logging Level: Fatal
100757	Workflow Monitor: shows UI paging rather than infinitive paging
96717	Maintenance Mode Support IP Range
101211	Log Viewer Display Milliseconds
101090	Ability to configure multiple Serilog Sinks
94790	Update MongoDB Driver to latest version
101460	Log should be the safest method
101091	Define 10 Fatal Errors in Casewhere Platform
101703	Publishing CMS Worker Site should not block visitors
101464	Memory Leak - Job Dispatcher may not release the handler properly
102050	Change "Domain Name" label in Worker Site Editor
100705	Workflow Debugger: View Script
101755	Export Data Classes to Excel: Separate Data Class and Data Attribute

Bugs

ID	NAME
84528	Crash Serilog when write log
99580	Workflow Monitor: Not Run Workflows has Failed status
101252	Got error when load widget data
93702	Cannot display maintenance page content when homepage is not set on worker site
99062	[Set languages] Dropdown list of Culture is cut when clicking on it
101266	[Maintenance Mode] Placeholder of IP Whitelist should change to follow the US
101360	Missing Fatal Method in Api Document
101393	Got 404 when entering an URL with a slash at the end
102005	Got error when add description for Data Class

ID	NAME
93209	[Numeric textbox-Precision] Still not validate precision on server-side
101893	LogViewer failed to load when too many errors created in a short time
102092	Fatal log data trigger notification property format
102083	Inconsistent position of Product Name between Data Classes and Data Attributes sheet
102357	[CMS page list] Cannot scroll when select more than 10 items per page
101702	Cannot scroll to the end by scroll wheel when the number of items is less than 10

2.7.1.0 - 17/01/2022

Features

ID	NAME
71647	Form Activity: Field Label Component
93868	Adjust Grid Columns in Casewhere Admin
98408	DataApi: Cursor for looping through large collections
99651	Non Persistent Variable: Share state between workflow activities
99733	Export audit log to Excel
100706	Workflow Debugger: Build
102041	Changing index field order should be easy
102134	Change Name of Worker Site
102495	Default Format Date for Widget
102491	Event Sourcing: Hot & Cold - Dealing with hundreds of millions of events
102761	The Ping endpoint should verify if DB is available for Write

Bugs

ID	NAME
73261	[Engine] Can add duplicate page into worker site
85807	[Workflow Monitor] Server rule is disappeared unexpectedly after clicking on dropdown list icon of Server rule
85808	Description sort does not work on the Theme page when clicking on sort icon

ID	NAME
90153	[System Log Rules] Sort function does not work on the System Log Rules page
91005	[UI/UX][Web Trigger-EventTrigger] Cannot list Process after set Data Class into workflow from Shortcut
92220	[UI/UX] Worker site settings display enhancement
92856	[Data Trigger] "Open" link displays un-expectedly when Clicking on "Create Workflow Definition" link to creating new Workflow Form Activity
93008	Name columns in some lists are too narrow
93533	The 'Enable Audit Log' option for Workflow should be removed
93657	Error message " Process is required" displays when editing workflow definition and click "Save" button on the Scheduler popup
93660	Missing filter the 'Automatic Workflow' on 'Web Trigger' and 'Scheduler Trigger'
97490	The sample code of new test for workflow definition is not correct
97839	[UI/UX][Data class] Attribute type dropdown is too long
97973	[Data Attribute] [Access Rule] Validation Rule for some system fields does not work
98224	The export format does not work when setting it in the Grid view widget
98411	[Workflow Monitor] Should hide retry if build failed
98417	[Workflow Monitor] Retry All should be hide if all workflow monitored success
98932	An error 500 appears when filtering with empty value
101246	[Log viewer] All dropdown list options are unchecked when clicking Reset button
101739	Cannot render Mermaid Diagram
101754	Updating Data Class is too slow, sometimes crashed for large data collections
102075	[Export Data Class] Missing translation in "Select data classes to export" dialog
102079	[UI/UX][Export DataClass] Select all checkbox still keep checked when uncheck one data class
102158	[CMS] Classic widget is returned search result incorrectly
102302	Missing search function on some pages (Products, Worker Sites, Themes, Processes, Workflow Monitors, Plugins, Schedulers)
102303	Missing fields Create, modify date, by on the Rules page
102359	Wrong header is set for Web Trigger

ID	NAME
102463	[Datetime] Time (HH:MM) has not been saved in the DB when setting timezone = (UTC) Coordinated Universal Time in the Worker Site
102566	Data Trigger does not execute for "User authorizing"
102599	IdP Connection error popup displays when changing Theme in the Edit worker site page
102803	Unexpected error on cw-choices component
102884	[LogViewer] Can't write log to Log Viewer after logging large data
103100	Change some Text "Schemasow" labels as source code
103104	Workflow job can be run more completely than one time because the "runworkflow" request response not return success

2.7.1.1 - 14/02/2022

Bug Hotfix

ID	NAME
105019	Cannot update claim on "User authorizing" event after upgrading to 2.7.1.0

2.7.1.2 - 10/03/2022

Bugs

ID	NAME
106068	Codeless Application Insights failed in 2.7.1
106014	Fix Rule API performance issue

2.7.1.3 - 28/04/2022

Features

ID	NAME
106756	FilterBuilder supports Nin

Bugs

ID	NAME
106743	Error in searching CMS content
106221	Time span config not compatible with hour config
107218	Conflict lock when publish cms and recycle occur

ID	NAME
107132	Duplicate ComponentRules when publishing the worker site

2.7.1.4 - 07/06/2022

Features

ID	NAME
108698	Data Trigger: Delay Persisting Notifications Until Workflow Persisted to DB
108708	Event Trigger: Delay Persisting Events Until Workflow Persisted to DB

Bugs

ID	NAME
108390	Publish CMS: Slow on Azure when there are 1000+ files
108346	Memory leak: Old worker site bundles were not released
108472	Data Source Query Always Count Total

2.7.1.5 - 11/08/2022

Bugs

ID	NAME
110377	Failed to store file to Azure storage
109844	Azure Storage: Got a lot of error with status 409

2.7.1.6 - 13/09/2022

Feature

ID	NAME
113710	DataTrigger should validate rule before verify the case state

Bugs

ID	NAME
113565	Application InSight Auto-Instrumentation has failed with new Azure Storage Libs
114344	DataApi.Enumerate got execution timeout on big collection which has more than 1M rows

2.7.1.7 - 20/10/2022

Bug

ID	NAME
115482	Missing Perflog on HttpApi and PluginApi when got exception

2.7.1.8.patch - 20/10/2023

Feature

ID	NAME
134521	Security patch to fix vulnerabilities on Worker API

2.7.2 - 20/07/2022

Features

ID	NAME
106043	Log Methods are slow than expected
106048	Event Sourcing: Update documentation How to run transfer tool
108012	MemLeak Possibilities: We need to cache the regex
107841	Workflow Execution Lock: Simplify the code & logic
106689	Workflow Execution Lock: Refactor the LockManager
103809	Workflow Execution Lock
103071	Concurrency settings for Workflow Activity
106248	RuleApi: Query Rules in Casewhere
103716	DataApi: Enumerate Data Source
105819	Text Resource: Auto fill prefix key of Product
103897	System Correlation ID for Audit Log (not the existing CorrelationId)
94762	Default Values for Data Class Attribute
103329	Data Class Dependencies
103265	Enforce FK
103263	Computed Attribute
102288	Auto Increment Attribute
102908	Deep Link: The URL reflects the editing resource
102906	Deep Link: The developer can open a product resource from a link

ID	NAME
102617	A generic way to duplicate Casewhere resources
88318	Auto reset "Test Status" when user updates Resource Under Test
109071	DslCompilerContext: Improve SetTemp, GetTemp
76528	Query string parameters are passed to workflows as variables
102580	Web Trigger: Support PUT method
102581	Web Trigger: Support DELETE method
102582	Form Activity: Custom Validation for Form Component
101225	[Wcag] Help-text icon should be focusable and show when enter
83336	Component data source use "Data Source" instead of "Data Class"
102043	Activity Scope: Auto reject all "Data Object" events if an activity has unexpected exception
104997	Workflow Scope: Able to set the transactional scope in Workflow
108354	Async Data Trigger support Transaction
108355	Async Event Trigger support Transaction
108774	Async Workflow Job support Transaction
108734	Workflow Queue: Delay Persisting Messages Until Workflow Persisted to DB
92894	Maintenance Mode for Worker Site v1
90411	Config row height for grid widgets
84622	Test Workflow: Stub plugin calls
110424	A setting to skip Component Dependencies from Being Minified
108478	Update frontend libs for security patches
107789	Move Scripts from CDN to Local
107303	WebApi: Get the url of the current request
103089	Form Validation: Support Async Validation
110312	It should be possible to use Auto-Increment Attribute in Computed Expression
110240	Implement Configuration Api read DataClass Information

ID	NAME
110241	Implement Configuration Api read DataSource Information
110245	Implement Configuration Api read File Information
110246	Implement Configuration Api read General Setting Information
110244	Implement Configuration Api read Plugin Information
110238	Implement Configuration Api read Product Information
110239	Implement Configuration Api read Rule Information
110243	Implement Configuration Api read Worker Site Information
110242	Implement Configuration Api read Workflow Definition Information

Bugs

ID	NAME
78767	Should not execute mongoDB code when generate data source script
106078	TypeError cannot read "id" message appears when selecting item in the dropdown list
106798	[Server Rule] Have issue in server rule when using IWebApi.GetHeaders
106522	Got error message text index when using full-text search
108941	Async data triggers/event triggers can run before the request ends due to the same CorrelationID
111139	ctx.FormData["WorkflowVariables syntax does not works in release 2.7.2

Casewhere 2.6-series release notes

Status: Done. **Date**: Q3 2021.

Highlights

- #69021 - New content management system (CMS): provides a completely new way to manage and design UI, more user-friendly and more flexibility
- #69946 - Reusable access rules: rules are managed centrally and can be reused for different purposes
- #70751 - Package manager v2: support for CMS and handling resource dependencies better
- #74667 - Data attribute rules : control access at data attribute levels
- #75965 - Data filter rules : we no longer have to repeat filter logic everywhere in widgets
- #77109 - Workflow monitor : a standard way to monitor automatic workflows, retry failed workflows
- #80566 - Time input component for form activity
- #69933 - Endpoint performance monitoring
- #78205 - Automation test
- #80492 - Workflow queue using MongoDB: fit well both cloud and on-premise, no longer depends on Azure Queue
- #74874 - Web triggers v2: More out-of-the-box settings to decouple web trigger and workflow definition, create more opportunities for reuse, make workflows are more testable
- #83571 - Event Trigger: Applications and standard components can emit events so consumers can subscribe/intercept and handle their own business logic.
- #74857 - Casewhere File: support caching
- #78800 - Reuse Idp connection for different worker sites
- #86775 - Search Engine: n-gram tokenizer: It is beneficial for querying languages that don't use spaces or that have long compound words, like Danish.
- #90088 - GDPR - Anonymize Personal Data v1.0
- #80231 - Non-persistent Workflow An option not to persist workflow states to DB. In this case, the workflow exists in memory only and will be discarded when the workflow engine finishes. This setting can significantly improve performance for scenarios where storing workflows does not make much sense, e.g., integrating with IoT devices, providing read-only APIs through Web Trigger.
- #96826 - Integrate Antivirus Software
- #74867 - Data diagram

Casewhere 2.6 is tested with MongoDB 5.0.

Changelog

2.6.14.8

- #107789 - Move Scripts from CDN to Local

2.6.14.7

- Bug fixes
    - #101457 - [Time fields] Do not change placeholder after change language

2.6.14.6

- #100750 - Improve Performance Log for Request with Query String
- #92902 - Remove non-used JS libs
- Bug fixes
    - #93690 - Need filter in Maintenance page dropdown list
    - #100544 - Got error in browser console when load System document
    - #100067 - Checkbox is wrong when search
    - #99637 - Tooltip in last column is crop
    - #99104 - Time picker auto scroll value should start from defaul value than current time

- #98624 - Datatime picker Default value should be TT:MM:SS in Danish instead of HH:MM:SS
- #95475 - Filter not consistent between UI and filter result item
- #101110 - All icons on the CMS admin are not displayed when loading CMS admin page

2.6.14.5

- Bug fixes
  - #99292 - User session is cached when using load balancer

2.6.14.4

- Bug fixes
  - #99189 - Prevent Recursive in Self-Lookup

2.6.14.3

- #99093 - Data Source Join Data Class using GraphLookUp

2.6.14.2

- Bug fixes
  - #99029 - Data Api Count/Any does not utilize index due to collation

2.6.14.1

- #98756 - Apply limit for Text Search
- Bug fixes
  - #98932 - An error 500 appears when filtering with empty value

2.6.14.0

- #90088 - GDPR - Anonymize Personal Data v1.0
- #80231 - Non-persistent Workflow
- #96826 - Integrate Antivirus Software
- #74867 - Data diagram
- Bug fixes
  - #92390 - Data Object is failed to load when there is a Data class Attribute Rule configured
  - #94424 - Error message "Command DropIndexes failed" is displayed when configuring the index definition via CaseState and CaseId
  - #96410 - Cannot add Expression in the "Input Data Object" when changing the setting to "Creating new case" and "Accept input" of the Workflow definition
  - #97516 - Cannot sort by text-score on MongoDB 4.4
  - #97270 - Performance hit on Custom Data Sources with multiple collections

2.6.13.0

- #96627 - Change titles of Admin Sites to be more informative
- #69933 - Endpoint performance monitoring
- #95692 - Upload file using Web Trigger
- #89684 - Split large script bundles into smaller ones to improve the loading speed
- #83289 - Data Trigger should auto-expire after a certain dequeue attempts
- #91917 - RequestID for Casewhere Worker Api
- Bug fixes
  - #91191 - CSS Bundle Minification
  - #84623 - Enforce rule validation for file-upload endpoint
  - #95116 - DataApi.Search seems not to utilize index when searching by Id, crash for big data collection
  - #96145 - Various issues on Timepicker
  - #91605 - Your app crashed because of System.InvalidOperationException
  - #96716 - Cannot filter audit log by CorrelationId
  - #95998 - Got exception when filtering Array(String)
  - #93727 - Can’t add multiple custom claims in the UserAuthorized event

2.6.12.0

- #92773 - Support IP Address and Browser Agent for Audit Log
- #92882 - Maintenance Mode for CMS Worker Site
- #92750 - API to Search Audit Log
- #91270 - Support second in the time picker
- #87914 - CorrelationID for Data Trigger and Event Trigger
- #88387 - Access IP Address from Web Trigger Rule
- Bug fixes
    - #88786 - Fix broken help links in Scripted Activity
    - #91584 - CaseState is not consistent among Case and DOs
    - #91568 - Command update failed: WriteConflict
    - #90872 - Array of Objects does not work properly
    - #93158 - Cannot create a worker site without Idp connection
    - #88700 - ctx.HasValue does not work correctly
    - #93366 - Cannot save "IdP Connection" in the Security Tab on the Worker site
    - #94714 - Data Trigger Workflow is triggered multiple times
    - #95217 - Serilog self-log can cause trouble when scaling out Casewhere on Azure App
    - #98611 - Filter Builder Regex Options does not work

2.6.11.0

- #83571 - Event Trigger
- #70079 - Users can logout in Default Unauthorized page
- #89791 - Support SVG as Image
- Bug fixes:
    - #89366 - Cannot load worker site having a page in a different product when importing new product version
    - #89903 - Run old published version when clear up/delete the workflow, then import it to the product again

2.6.10.0

- #74857 - File Caching
- #74819 - Search on CMS static content
- #82968 - Access control for CMS Admin
- #83287 - Show the login popup when the Casewhere is expired
- #88711 - HttpApi: Create GET request to the external service
- #83869 - Export test workflows by the current filter
- #86775 - Free-text search: n-gram tokenizer
- Bug fixes
    - #88679 - Cannot filter on encrypted fields on custom-script data sources
    - #85046 - Uploading a large file
    - #89770 - Manage the resource Dependency between worker sites and worker site components
    - #87586 - Calendar returns selected date with the wrong time
    - #86947 - Date time component has not validated Min, Max date when inputting data
    - #88233 - Image has not displayed in the File popup in the CMS admin site after saving edit data in the configuration site
    - #85986 - Cannot Preview the CMS page

2.6.9.0

- #84392 - Ability to customize unauthorized request handler on worker site
- #85104 - Ability to configure multiple Data Trigger Workers
- #83325 - Invalidate script cache when publishing CMS Worker Site / Update theme
- #76558 - Option to force login Worker Site
- #82899 - Auto audit-log all "write" activities
- Many small improvements for Casewhere Admin regarding Data Triggers, Web Triggers, Workflow Editor, providing more shortcuts

- Bug fixes
  - #84604 - DSL-user.js just support get claim by lower case
  - #83888 - Run ALL test cases in the system when run test in Test Manager dialog
  - #84192 - Duplicated key rule caching [Critical]
  - #83828 - Entity ID overrides after importing product package
  - #84624 - Shortcut link is messy
  - #84561 - Cannot open worker site when using the new theme
  - #84570 - DataSources have not applied server rules when running it on the worker site
  - #85473 - Got exception when enabling user filtering
  - #82961 - Scripted custom data source with array field don't work when setting DataSource for grid view
  - #84578 - UnderTestResourceId with null value blocks some engine functions (eg. export product/edit tests/filter test...)
  - #84559 - Got Exception when assigning the rule to Custom Data Source or Page
  - #82977 - Got error in the browser console when tick/untick "Automatic Workflow" option
  - #68813 - Confirm popup does not work when clicking on the Forward button in the Workflow
  - #83662 - Missing some Open button links on the web trigger popup
  - #83774 - Hidden field does not work
  - #83726 - The "Filtering Fields" dropdown is empty when using custom DataSource has Array field
  - #83979 - Search function in Tests page does not work
  - #85495 - Cannot close Login window after logs in successfully

2.6.7.0

- #74874 - Web triggers v2
  - #80501 - Web trigger responds using workflow output
  - #83329 - Handle unexpected errors in Workflow Engine
  - #81696 - Web trigger auto responds error from workflow
  - #83332 - Use existing trigger rules, reduce logic duplication
  - #83330 - Enable Web Trigger Access from Worker Site
  - #82386 - Customize web trigger URI, support multiple path segments

- #79573 - Automate testing for non-workflow resources, e.g., rules, plugins, data sources
  - #83322 - Create Tests from Plugin
  - #82387 - Ability to associate a test workflow with an existing resource
  - #83327 - Create Workflow Tests from Rule
  - #82381 - Create Test Workflows independently of Workflow Definition

- #83297 - Export all Test Workflows to an Excel File
- #83321 - Upgrade C# Version for Casewhere Scripting
- #83290 - Expose DslWorkerSite in IDslCompilerContext
- #83296 - Option to Hide Activity Navigation Pane by default
- Bug fixes
  - #83032 - Resolving Workflow Engine object took 100-200ms
  - #82388 - Encrypt sensitive data in web.config
  - #82374 - Memory leak possibility in ApiFactory
  - #84394 - The form is stopped unexpectedly after an exception throw in Script Activity

2.6.6.0

- #78800 - Reuse Idp connection for different worker sites
- #80191 - Data Trigger: Filter by Data Class
- Security improvements
  - #82419 - Separate database for audit log
  - #82388 - Encrypt sensitive data in web.config

2.6.5.0

- #77109 - Workflow monitor: a standard way to manage and monitor automatic workflows
- #75965 - Data filter rules: we no longer have to repeat filter logic everywhere in widgets
- Bug fixes:
    - #79975 - Audit log does not work for Document attribute
    - #72721 - Cannot open logs.zip after downloaded from the log viewer
    - #82161 - Long-running workflows (> 30s) are executed twice

2.6.4.0

- #80492 - Workflow queue using MongoDB
- #79957 - Query audit log by users
- #80566 - Time input component for form activity
- #80477 - Manage Workflow Tests as Product Resource
- Bug fixes:
    - Data trigger is executed multiple times if it is queued for longer than 1 hour (mostly due to server busy)
    - [Text resources] Text resource export does not follow product filter

2.6.3.1

- #79713 - Users can add descriptions for workflow definitions
- #71547 - Package manager v2 - Manage data triggers as a product resource
- Bug fixes:
    - #79632 - Offer text resource keys as a language
    - #62887 - Form activity: optional file uploader blocks form submission
    - #79473 - Form activity: data picker showing the wrong position. Some other UI issues
    - #79530 - Data class editor: some UI bugs regarding configuring data attribute rules

2.6.3.0

- #78878 - Data attribute rules: control access at data attribute levels
- #78473 - Theme is not overridden when import Casewhere product
- 22 bug fixes:
    - Change is effective immediately when running automation tests
    - Numeric textbox and decimal values
    - Inclusion logic "Included Until Complete" is not correct
    - Data binding in form activity regarding read-only fields
    - Daylight saving time in the form date picker

2.6.2.1

- #78205 - Automation test: provide methods to intercept workflow execution events.
- Update Chart.js v2.9.3

2.6.2.0

- #76915 - Ability to customize workflow wizard from Casewhere Admin
- #77490 - Data trigger using MongoDB as storage: safer and more scalable
- #77492 - An improvement in Workflow Queue: Prevent the possibility of data loss in Workflow Queue when the server down
- Bug fixes:
    - #75350 - #75382 - iOS Safari users sometimes have trouble with login
    - #76431 - The workflow continues to execute after the embedded workflow has to build error
    - #77003 - The steps order is not correct when moving the last step to the first step

2.6.1.1

- Update Hangfire v1.7.12
- A lot of bug fixes.

- #76859 Script in the main workflow is not executed
- #76714 Numeric Input in the properties of the Form activity has not saved when
- #77077 Cannot drag and drop Numeric textbox into Form activity
- #76085 Form Builder: Numeric Input Precision does not work
- #76489 Error message "Sequence contains more than one element" appears when importing then updating workflow Definition
- #76492 Page next has displayed when clicking on Save button

2.6.1.0

- #74896 - Option to configure publishing date for CMS widgets
- #76213 - Workflow can return value to callers: optimized for reusability, provides users better ways to design their products.
- Workflow editor improvements
  - #74898 - Option to force showing "Save and Next" in Form Activity
  - #73369 - Response correct MIME types when downloading Casewhere files
  - #74940 - APIs to evaluate client rules in worker site
- Security bug fixes
  - #70037 - Missing Worker Site Header Validation
  - #73368 - Remove insecure response headers

2.6.0.0

- #69021 - New content management system: provides a completely new way to manage and design UI, more user-friendly and more flexibility
- #69946 - Reusable access rules: rules are managed centrally and can be reused for different purposes
- #69960 - Automation test: it is now possible to automate testing Casewhere workflows
- #70746 - Synchronous data trigger: Data trigger now can be executed in both ways synchronously and asynchronously
- #70751 - Package manager v2: better support for CMS and resource dependencies
- #70090 - Form editor: a way to configure parameters for Dropdown data source
- Many bug fixes & small improvements

Upgrade to 2.9

Recommendations

To upgrade a Casewhere deployment to 2.9, you must be, at least, running a 2.6-series release.

Before upgrading Casewhere, always test your application in a staging environment before deploying the upgrade to your production environment. It would be best if you also backup your production database before upgrading.

If you are deploying Azure App Service using Kudu Zip Push Deploy, you don't have to modify the `web.config` files manually, only need to update the keys in the **Configuration / Application Settings**.

> **Information!** If you are replacing the old web.config with the new one, then you can ignore all update it steps in the section below.

Upgrade versions

The following changes must be applied in the described order.

2.9.0.0 - 01/08/2025

Upgrade to 2.8

Recommendations

To upgrade a Casewhere deployment to 2.8, you must be, at least, running a 2.6-series release.

Before upgrading Casewhere, always test your application in a staging environment before deploying the upgrade to your production environment. It would be best if you also backup your production database before upgrading.

If you are deploying Azure App Service using Kudu Zip Push Deploy, you don't have to modify the `web.config` files manually, only need to update the keys in the **Configuration / Application Settings**.

> **Information!** If you are replacing the old web.config with the new one, then you can ignore all update it steps in the section below.

Upgrade versions

The following changes must be applied in the described order.

2.8.0.0 - 12/04/2023

Support to run fine on the new version of MongoDB.

- Open the web.config in `Casewhere.Configuration.Web` and update the settings below:

```
<runtime>
  <assemblyBinding>
      <dependentAssembly>
          <assemblyIdentity name="System.Text.Encoding.CodePages" publicKeyToken="b03f5f7f11d50a3a"
culture="neutral" />
          <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
          <assemblyIdentity name="Microsoft.Extensions.Logging.Abstractions"
publicKeyToken="adb9793829ddae60" culture="neutral" />
          <bindingRedirect oldVersion="0.0.0.0-2.2.0.0" newVersion="2.2.0.0" />
      </dependentAssembly>
      <dependentAssembly>
          <assemblyIdentity name="Microsoft.Win32.Registry" publicKeyToken="b03f5f7f11d50a3a"
culture="neutral" />
          <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
          <assemblyIdentity name="System.Security.Principal.Windows" publicKeyToken="b03f5f7f11d50a3a"
culture="neutral" />
          <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
          <assemblyIdentity name="Microsoft.Bcl.AsyncInterfaces" publicKeyToken="cc7b13ffcd2ddd51"
culture="neutral" />
          <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
      </dependentAssembly>
  </assemblyBinding>
</runtime>
```

- Open the web.config in `Casewhere.Worker.Api` and update the settings below:

```
<runtime>
  <assemblyBinding>
      <dependentAssembly>
          <assemblyIdentity name="System.Text.Encoding.CodePages" publicKeyToken="b03f5f7f11d50a3a"
culture="neutral" />
          <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
          <assemblyIdentity name="Microsoft.Bcl.AsyncInterfaces" publicKeyToken="cc7b13ffcd2ddd51"
culture="neutral" />
          <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
          <assemblyIdentity name="Microsoft.Win32.Registry" publicKeyToken="b03f5f7f11d50a3a"
culture="neutral" />
          <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
          <assemblyIdentity name="System.Security.Principal.Windows" publicKeyToken="b03f5f7f11d50a3a"
culture="neutral" />
          <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
      </dependentAssembly>
      <dependentAssembly>
          <assemblyIdentity name="Microsoft.Extensions.Logging.Abstractions"
publicKeyToken="adb9793829ddae60" culture="neutral" />
          <bindingRedirect oldVersion="0.0.0.0-2.2.0.0" newVersion="2.2.0.0" />
      </dependentAssembly>
  </assemblyBinding>
</runtime>
```

Support to collaborate nicely without overriding each other's work

Casewhere Application Developer can open an entity such as a workflow, a rule, etc, and they can know who is editing the same thing as me to collaborate nicely without overriding each other's work. To enable that feature you need to add the following settings to the web.config file in the `Casewhere.Configuration.Web` project.

```
    <add key="signalr:UserEditingDocumentExpiredTime" value="00:03:00" />
    <add key="signalr:RefreshUserEditingDocumentIntervalTime" value="00:01:00" />
```

Enable hard delete for specified Data Objects

A typical scenario is that the business only wants to hard delete some specific data objects in a case (or different cases). Hard deleting is the process of completely removing the information from the database. So Casewhere will support a method via IDataApi to remove all information on Events, Read Database. To use that feature you need to add the following settings to the web.config file in the `Casewhere.Worker.Api` project.

```
    <add key="db:EnableHardDelete" value="false" />
```

It is "false" by default if key missing.

Enrich Audit Log with Searchable Information

To enrich log data with searchable information, i.e., fields that can be used for different data query strategies. To do that add the setting below into web.config of `Casewhere.Worker.Api` project.

```
    <add key="auditlog:DeprecateUserClaims" value="false" />
```

Default is "false". Key missing also means "false". If the setting is "true", UserClaims is null when Casewhere writes Audit log.

2.8.1.0 - 19/05/2023

There is no change in configuration.

2.8.2.0 - 30/09/2023

Time to live for System Log

To configure the time to live for the system log, you can add the following setting to the web.config file in the `Casewhere.Configuration.Web` project.

```
    <add key="systemlogs:ExpiredTime" value="7776000" />
```

Default is 90 days if key missing.

Default is 90 days if key is missing.

2.8.3.0 - 10/01/2024

Enable compress Workflow Events

To reduce the number of workflow events at runtime, you can consider adding the following setting to the web.config file in the `Casewhere.Worker.Api` project. This feature will help reduce storage size and increase performance.

```
    <add key="events:EnableCompressEvents" value="false" />
    <add key="events:CompressionEventThreshold" value="10" />
```

This feature is disabled by default if the keys are missing.

Enable invalidate workflow definition cache

This feature should be disabled in the PROD environment to reduce load and build workflow definition. For the DEV environment, you can enable it to run tests with the newest workflow definition.

Adding the following setting to the web.config file in the `Casewhere.Worker.Api` project.

```
    <add key="test:EnableInvalidateWorkflowDefinitionCache" value="true" />
```

This feature is enabled by default if the key is missing.

2.8.3.1 - 26/01/2024

There is no change in configuration.

2.8.3.2 - 29/02/2024

Default collation configuration

To increase performance of data query and support sorting data on specified language, you can consider adding the following settings to the web.config file in the `Casewhere.Worker.Api` and `Casewhere.Configuration.Web` projects.

```
    <!--Default collation-->
    <add key="db:Collation:Locale" value="simple" />
    <add key="db:Collation:Strength" value="" />
```

The default collation from this version to the previous version is Locale: "en" and Strength: "2" if the keys are missing.

> **Important!** If you are installing the new environment from this version, then you can ignore the maintenance step below.

If you change the default collation in the existing environment, you need to access the maintenance URL below to rebuild indexes of all data classes in the Casewhere system to the new collation.

[Admin domain]/Maintenance/RebuildIndexes

2.8.5.0 - 15/07/2024

Configuration the data trigger, event trigger, and rule worker in real-time

In some cases, we can have multiple Casewhere instance point to the same database but each instance can perform the different business. For example, the instance A serves for the front-end business with a few of background workers and the instance B serves for the backend-end business with a lot of the background workers. As result, the current logic cannot solve this case. So, we need to allow to manage the data trigger settings per instance.

To control that settings on each instance, you can add the following setting to the web.config file in the `Casewhere.Worker.Api`.

```
    <!--Instance Name-->
    <add key="runtime:InstanceName" value="Instance A" />
```

Default is empty if key missing.

Configuration the number of the worker thread in Job Site

To control this number, you can add the following setting to the web.config file in the `Casewhere.Hangfire`.

```
    <!--Number of the worker thread in Job Site-->
    <add key="hangefire:WorkerCount" value="4" />
```

Default is 4 workers if key missing.

Upgrade to 2.7

Recommendations

To upgrade a Casewhere deployment to 2.7, you must be running a 2.6-series release.

Before upgrading Casewhere, always test your application in a staging environment before deploying the upgrade to your production environment. It would be best if you also backup your production database before upgrading.

If you are deploying Azure App Service using Kudu Zip Push Deploy, you don't have to modify the `web.config` files manually, only need to update the keys in the **Configuration / Application Settings**.

Upgrade versions

The following changes must be applied in the described order.

2.7.0.0 - 25/11/2021

Worker Api - `Casewhere.Worker.Api`

> **Information!** If you're replacing the old web.config with the new one e.g. when deploying on Azure App Service, you can ignore this step

Apply the below changes in the `web.config`:

- In the `appSettings` section, add the following keys:

```
<add key="serilog:disableSelfLog" value="false" />
```

- In the `runtime/assemblyBinding` section, add the followings:

```
<dependentAssembly>
  <assemblyIdentity name="System.Numerics.Vectors" publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
  <bindingRedirect oldVersion="0.0.0.0-4.1.4.0" newVersion="4.1.4.0" />
</dependentAssembly>
<dependentAssembly>
  <assemblyIdentity name="System.Diagnostics.DiagnosticSource" publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />
  <bindingRedirect oldVersion="0.0.0.0-4.0.1.0" newVersion="4.0.1.0" />
</dependentAssembly>
```

- In the `runtime/assemblyBinding` section, update the `dependentAssembly` for `System.Runtime.CompilerServices.Unsafe`:

```
<dependentAssembly>
  <assemblyIdentity name="System.Runtime.CompilerServices.Unsafe" publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
  <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
</dependentAssembly>
```

Job Site - `Casewhere.Hangfire`

> **Information!** If you're replacing the old web.config with the new one e.g. when deploying on Azure App Service, you can ignore this step

Apply the below changes in the `web.config`:

- In the `runtime/assemblyBinding` section, add the followings:

```
<dependentAssembly>
  <assemblyIdentity name="System.Numerics.Vectors" publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
  <bindingRedirect oldVersion="0.0.0.0-4.1.4.0" newVersion="4.1.4.0" />
</dependentAssembly>
```

- In the `runtime/assemblyBinding` section, update the `dependentAssembly` for `System.Runtime.CompilerServices.Unsafe`:

```
<dependentAssembly>
  <assemblyIdentity name="System.Runtime.CompilerServices.Unsafe" publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
  <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
</dependentAssembly>
```

Configuration Site - `Casewhere.Configuration.Web`

> **Information!** If you're replacing the old web.config with the new one e.g. when deploying on Azure App Service, you can ignore this step

Apply the below changes in the `web.config`:

In the `appSettings` section, add the following keys:

```
<add key="logviewer:MaxJsonLength" value="10000000" />
```

- In the `runtime/assemblyBinding` section, add the followings:

```
<dependentAssembly>
  <assemblyIdentity name="System.Numerics.Vectors" publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
  <bindingRedirect oldVersion="0.0.0.0-4.1.4.0" newVersion="4.1.4.0" />
</dependentAssembly>
<dependentAssembly>
  <assemblyIdentity name="System.Diagnostics.DiagnosticSource" publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />
  <bindingRedirect oldVersion="0.0.0.0-4.0.1.0" newVersion="4.0.1.0" />
</dependentAssembly>
```

- In the `runtime/assemblyBinding` section, update the `dependentAssembly` for `System.Runtime.CompilerServices.Unsafe`:

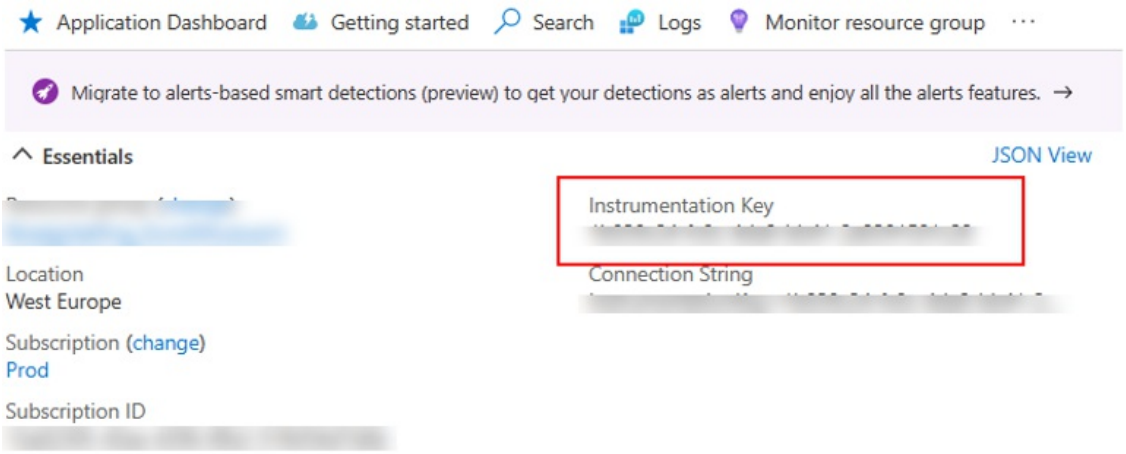
```
<dependentAssembly>
  <assemblyIdentity name="System.Runtime.CompilerServices.Unsafe" publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
  <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
</dependentAssembly>
```

Log fatal events using Azure Application Insights

> **Important!** This step is applied for deploying on Azure App Service only.

In Azure Portal, open the **Worker Api App Service**, navigate to **Configuration / Application Settings** and add the following keys:

KEY	VALUE
serilog:using:ApplicationInsights	Serilog.Sinks.ApplicationInsights
serilog:write-to:ApplicationInsights.telemetryConverter	Serilog.Sinks.ApplicationInsights.Sinks.ApplicationInsights.TelemetryConverters.TraceTelemetryConverter, Serilog.Sinks.ApplicationInsights
serilog:write-to:ApplicationInsights.restrictedToMinimumLev	Fatal
serilog:write-to:ApplicationInsights.instrumentationKey	You can find the **Instrumentation Key** value in Azure portal.



Enable logging for Worker Site

Apply the below changes in the `web.config`:

Add the `configSections` before `appSettings` section:

```
  <configSections>
    <section name="secureAppSettings" type="System.Configuration.NameValueSectionHandler" />
  </configSections>
```

In the `appSettings` section, add the following keys:

```
<add key="serilog:minimum-level" value="Debug" />
<add key="serilog:using:MongoDB" value="Serilog.Sinks.MongoDB" />
<add key="serilog:write-to:MongoDB.collectionName" value="Logs" />
```

Add the `secureAppSettings` section right under the `appSettings` section:

```
<secureAppSettings>
  <add key="serilog:write-to:MongoDB.databaseUrl" value="<Log database connection string. Hint: copy the value from your
Worker.Api>" />
</secureAppSettings>
```

If you are deploying **Azure App Service**, just add all the above keys in **Configuration / Application Settings**.

2.7.1.0 - 17/01/2022

Worker Api - `Casewhere.Worker.Api`

> **Information!** If you're replacing the old web.config with the new one e.g. when deploying on Azure App Service, you can ignore this step

Apply the below changes in the `web.config`:

- In the `appSettings` section, add the following keys:

```
<!-- Log settings -->
<add key="serilog:destructure:ToMaximumStringLength.maximumStringLength" value="100000000" />
<add key="serilog:destructure:ToMaximumCollectionCount.maximumCollectionCount" value="25" />

<!-- Event settings-->
<add key="events:EnableHotAndCold" value="false" />
```

- In the `runtime/assemblyBinding` section, modify the followings:

```
<dependentAssembly>
  <assemblyIdentity name="System.Threading.Tasks.Extensions" publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />
  <bindingRedirect oldVersion="0.0.0.0-4.1.0.0" newVersion="4.1.0.0" />
</dependentAssembly>
```

Worker Site - `Casewhere.Web.CaseWorker`

> **Information!** If you're replacing the old web.config with the new one e.g. when deploying on Azure App Service, you can ignore this step

Apply the below changes in the `web.config`:

- In the `appSettings` section, modify the following key:

```
<add key="owin:appStartup" value="Casewhere.Worker.Web.Startup" />
```

- In the `runtime/assemblyBinding` section, add the followings:

```
<dependentAssembly>
  <assemblyIdentity name="System.Runtime.CompilerServices.Unsafe" publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
  <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
</dependentAssembly>
<dependentAssembly>
  <assemblyIdentity name="System.Numerics.Vectors" publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />
  <bindingRedirect oldVersion="0.0.0.0-4.1.3.0" newVersion="4.1.3.0" />
</dependentAssembly>
```

Configuration Site - `Casewhere.Configuration.Web`

> **Information!** If you're replacing the old web.config with the new one e.g. when deploying on Azure App Service, you can ignore this step

Apply the below changes in the `web.config`:

- In the `appSettings` section, add the following keys:

```
<!-- Event settings-->
<add key="events:EnableHotAndCold" value="false" />
```

Enable Hot and Cold

> **Warning!** Only enable the feature when your project requires doing so

Please follow this guideline to learn about the **Hot and Cold** feature and enable it for your deployment.

2.7.1.1 - 14/02/2022

There is no change in configuration.

2.7.1.2 - 11/03/2022

Worker Api - `Casewhere.Worker.Api`

> **Information!** If you're replacing the old web.config with the new one e.g. when deploying on Azure App Service, you can ignore this step

Apply the below changes in the `web.config`:

- In the `appSettings` section, remove the following keys:
  - serilog:using:ApplicationInsights
  - serilog:write-to:ApplicationInsights.telemetryConverter
  - serilog:write-to:ApplicationInsights.restrictedToMinimumLev
- In the `secureAppSettings` section, remove the key:
  - serilog:write-to:ApplicationInsights.instrumentationKey > > **Important!** This step is applied for deploying on Azure App Service only.

In Azure Portal, open the **Worker Api App Service**, navigate to **Configuration / Application Settings** and remove the following keys:

- serilog:using:ApplicationInsights

- serilog:write-to:ApplicationInsights.telemetryConverter
- serilog:write-to:ApplicationInsights.restrictedToMinimumLev
- serilog:write-to:ApplicationInsights.instrumentationKey

2.7.1.3 - 28/04/2022

> **Important!** In this release we need run migration data for this bug: **Duplicate ComponentRules when publishing the worker site**

To remove the duplication, use the url with the following format:

- **[Admin domain]/Maintenance/RemoveWorkerSiteComponentDuplication**

  This Url to preview the list of duplicated components that will be removed later, but they are not removed

- **[Admin domain]/Maintenance/RemoveWorkerSiteComponentDuplication?acceptRemoval=true**

  This Url to remove the duplicated components and show the list of results.

  2.7.1.4 - 07/06/2022

  > **Information!** If you're replacing the old web.config with the new one e.g. when deploying on Azure App Service, you can ignore this step

- In Azure Portal, open the **Worker site App Service**, navigate to **Configuration / Application Settings** and add the new key:
  - Name: **fcn:StopMonitoringSubFolders**
  - Value: **true**

2.7.1.5 - 11/08/2022

Impact in this version, we had an upgrade of the latest version of Azure Storage SDK (12.13.0). So, these features will be affected:

- The management of the Files on Admin site
- The IDocumentApi in the Scripted Activity
- The Upload file component in the Form Activity
- The download file function from the Worker Site

> **Information!** If you are replacing the old web.config with the new one, then you can ignore this step.

Worker Api - `Casewhere.Worker.Api`

Open the web.config and update the setting key below:

```
<dependentAssembly>
    <assemblyIdentity name="System.Threading.Tasks.Extensions" publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />
    <bindingRedirect oldVersion="0.0.0.0-4.2.0.1" newVersion="4.2.0.1" />
</dependentAssembly>
```

2.7.1.6 - 08/09/2022

> **Important!** From this version, please following instructions below if we are using Azure Application Insight otherwise just skip it.

Enable Azure Application Insight codeless integration

2.7.1.7 - 20/10/2022

There is no change in configuration.

2.7.1.8.patch - 20/10/2023

> **Important!** This version is a patch, please following instructions below to deploy a hotfix path.

- Extract the hotfix patch package to a temp folder.

On Azure

- Go to Kudu site and navigate to the Kudu **wwwroot** directory of each sites:
  - Configuration.Web
  - Casewhere.Web.CaseWorker
  - Casewhere.Worker.Api

- Drag and drop corresponding files to the Kudu **wwwroot** directory of target sites:
  - Configuration.Web.zip
  - Casewhere.Web.CaseWorker.zip
  - Casewhere.Worker.Api.zip

On Premise

- Copy all files and folders from patch to the target folders
  - Configuration.Web
  - Casewhere.Web.CaseWorker
  - Casewhere.Worker.Api

2.7.2 - 20/07/2022

Migrating Parameters for Server rules:

> **Important!** We need to run migration script for query string parameters are used in Web Trigger Server Rules
>
> - Run Api for migrating Parameters syntax for Web Trigger Server Rules:

```
**[Admin domain]/Maintenance/MigrateParametersForServerRule**
```

```
// Old script
@parameters["Code"] == "AP001"

// New script
@parameters["Code"].ToString() == "AP001"
```

Notes: The Query string Name will be case insensitive.

Web Trigger Support PUT/DELETE method:

- Should overwrite the web.config from release package.
- If not overwrite, we need to change the web.config of Worker API and add the setting bellow:

```
<system.webServer>
  <handlers>
    <remove name="WebDAV" />
  </handlers>
</system.webServer>
```

Concurrency settings for Workflow Activity:

- Should overwrite the web.config from release package.
- If not overwrite, we need to change the web.config of Worker API and add the new setting keys bellow:

```
<appSettings>
  <!-- Engine Settings -->
  <add key="engine:ConcurrencyMaxTimeout" value="00:03:00" />
  <add key="engine:ConcurrencyDelayTime" value="2000" />
  <add key="engine:WorkflowExecutionLockMaxTimeout" value="00:01:00" />
</appSettings>
```

Update frontend libs for security patches:

- The client script of worker site will be migrated to:
  - Jquery v3.2.1 -> v3.6.0
  - Bootstrap v4.0.0 -> v4.6.0
  - JqueryUI v1.12.1 -> v1.13.1 > **Important!** Please consider Warning console log in Browser for migrating worker site client script.

DslCompilerContext: Improve SetTemp, GetTemp:

- The **ctx.GetTemp** will throw an exception when key is not existed.
- Support **ctx.HasTempKey** to check the temporary key existed or not.