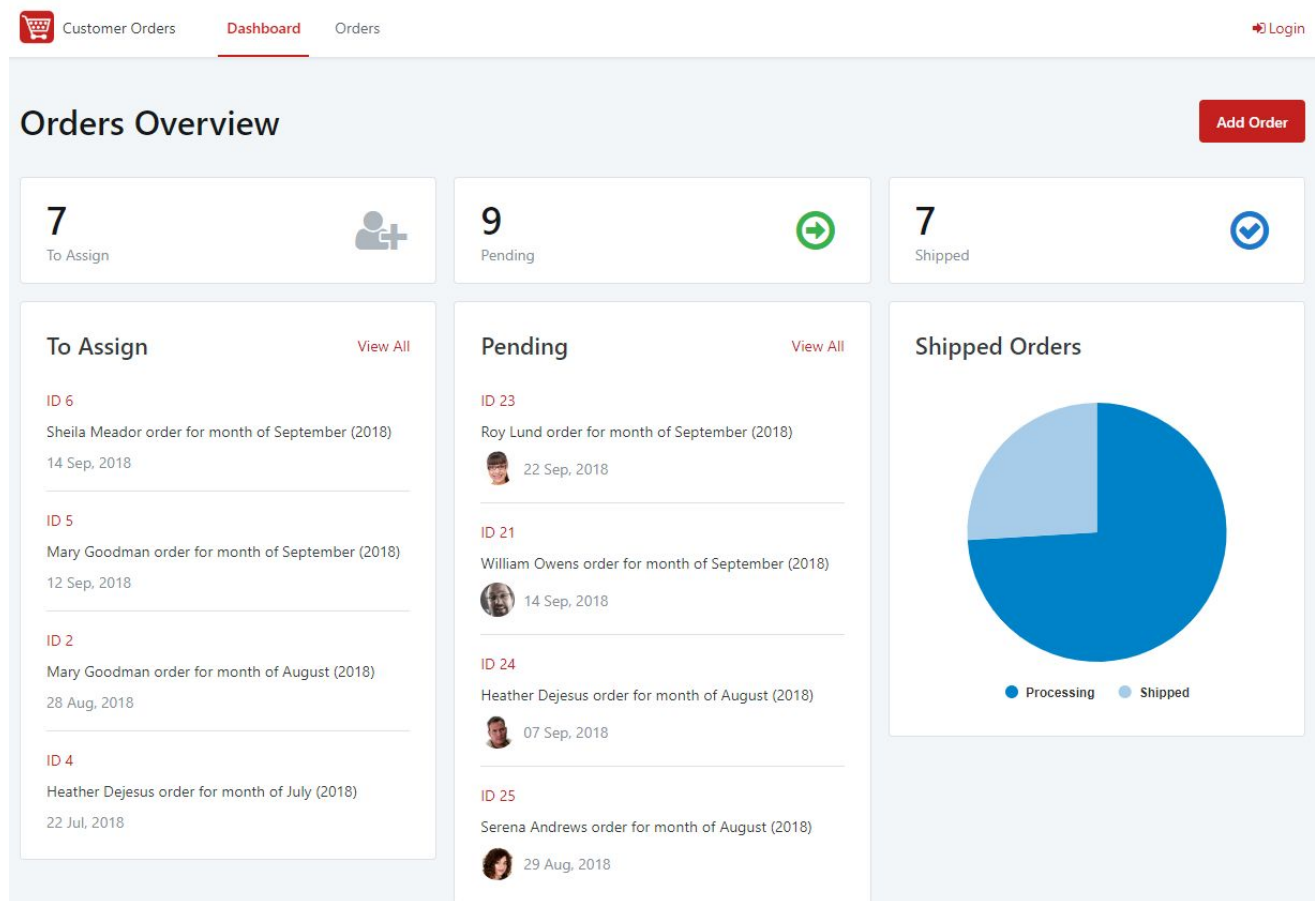


# Accelerating UI Development Lab



# Table of Contents

<b>Table of Contents</b>	<b>2</b>
<b>Introduction</b>	<b>3</b>
<b>Get to know the scenario</b>	<b>4</b>
<b>Setup the applications</b>	<b>5</b>
Upload the Customer Orders Data application	5
Create the Application	5
<b>Create the Dashboard Screen</b>	<b>9</b>
<b>Use real data in the Dashboard Screen</b>	<b>12</b>
Reference the Customer Orders Data Model	12
Replace the data in the Dashboard Screen	13
<b>Create and customize the Orders Screen</b>	<b>26</b>
<b>Create and customize the OrderDetail Screen</b>	<b>32</b>
<b>Link the Screens</b>	<b>41</b>
<b>Delete the sample data and test the application</b>	<b>47</b>
<b>Scaffolding Patterns</b>	<b>50</b>
<b>End of lab</b>	<b>54</b>

## Introduction

In this session, we are going to build a responsive web application that is designed to manage customer orders. While building this application, we will use many features that will speed up the application development process. However, the main focus of this exercise is to utilize the new **Screen Templates**, in the latest OutSystems 11 release.

This application will use data that has already been created in the **Customer Orders Data** application. We will start by building the UI of the application with the new screen templates provided in OutSystems 11. After creating the screens, we will modify them to present the data from the Customers Order Data application.

These new screen templates provide sample data for previewing purposes, so that we are able to immediately preview the new screens working correctly. Since we want to use our data, instead of the sample data used by the templates, we will replace that data using some new accelerators. Also, since everything that is created based on templates is customizable, we will adjust the UI screens according to our needs.

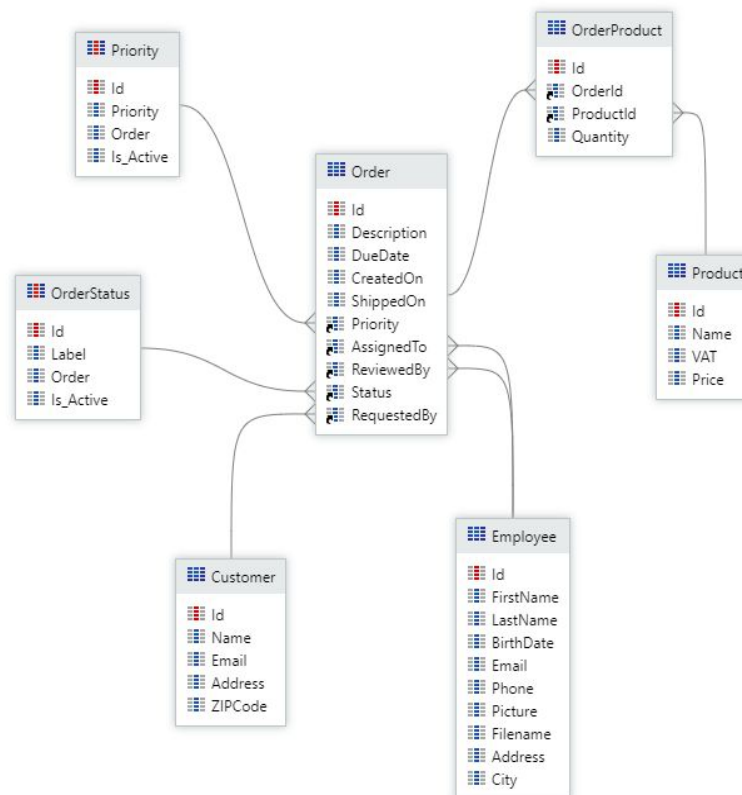
## Get to know the scenario

This exercise lab will use a pre-built application, Customer Orders Data. This application has the data model, that will support the new application we are going to build.

The Customer Orders Data application does not have a UI (user interface). It is an application with a data model defined for Orders, Products and Customers.

This data model consists on an **Order** Entity, which has a many-to-many relationship with **Product**, through the **OrderProduct** Entity, since an order can have many products and a product can be part of different orders. Each Order also has a **Priority** and a **OrderStatus**. The Order can then be assigned to, or reviewed by, **Employees** of the company, and requested by **Customers** that will use our app.

In this lab, we will create the UI based on Screen Templates, which bring their own sample data. Since we will start from an existing data set, it is important to understand well its data model, to help the process of replacing the sample data by this one. The data model can be visible in the following screenshot.



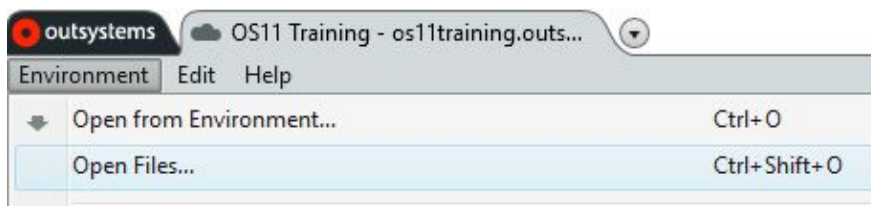
## Setup the applications

We will start the lab by setting up the applications that we are going to work on.

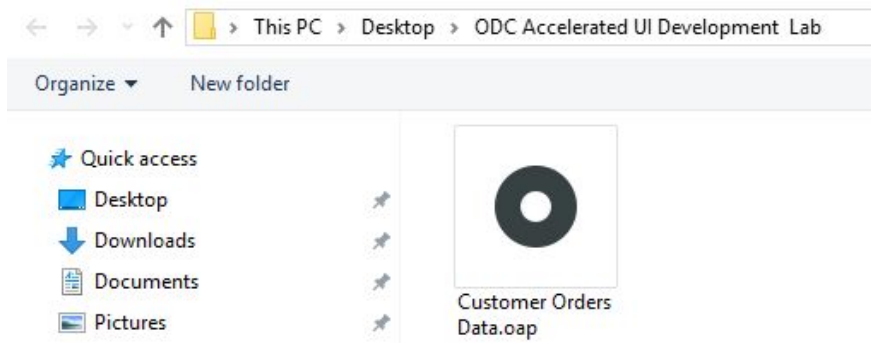
### Upload the Customer Orders Data application

Before we create the Customer Orders application, we will need to upload the Customer Orders Data application that provides the data we need. **This part is not needed in the classroom training.**

- 1) Select the **Open Files ...** option from the Environment menu option.



- 2) Browse to and select the *Customer Orders Data* application file that was downloaded with the student package and click the **PROCEED** button to install the application.

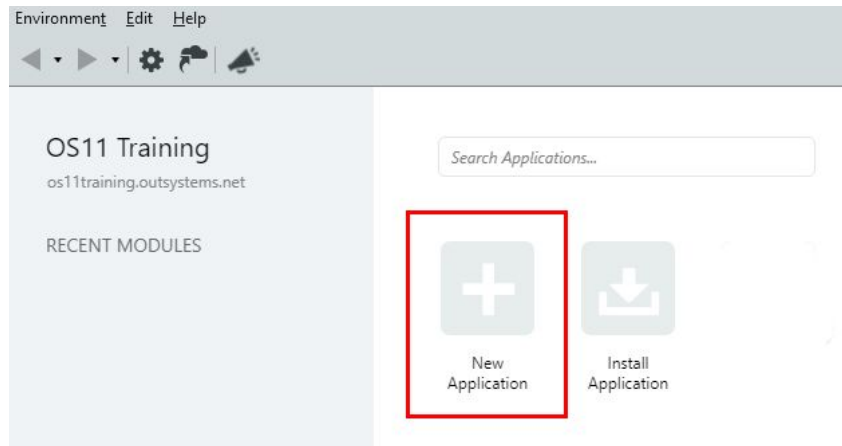


### Create the Application

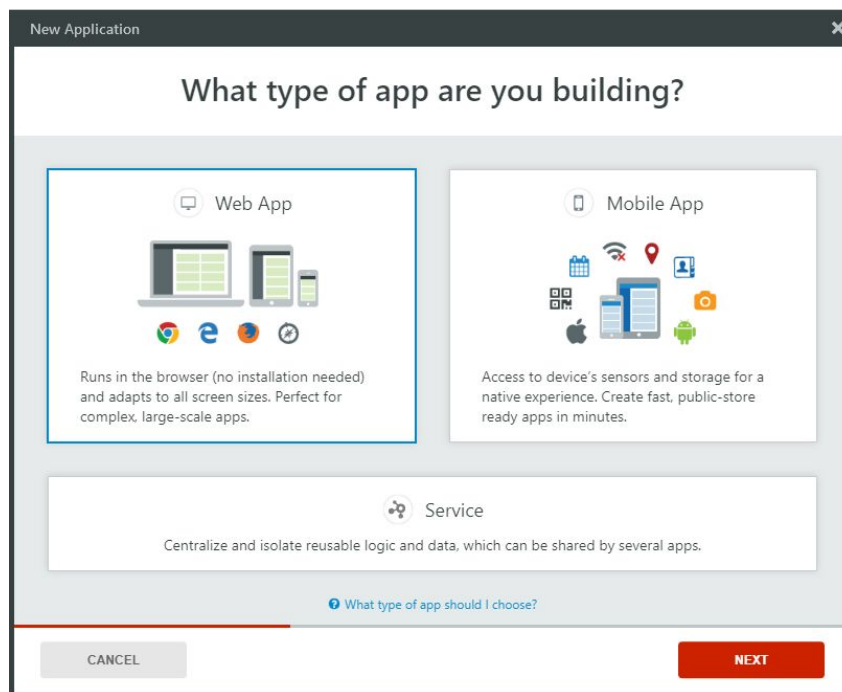
In this exercise lab, we will create the Customer Orders web application, to manage orders, and its products, submitted by Customers. Create the new **CustomerOrders** web responsive module inside the existing Customer Orders application.

- 1) Create the new application in Service Studio called *Customer Orders*, with the TopMenu template. Use the *AppIcon.png* from the **Resources** folder as the application icon.

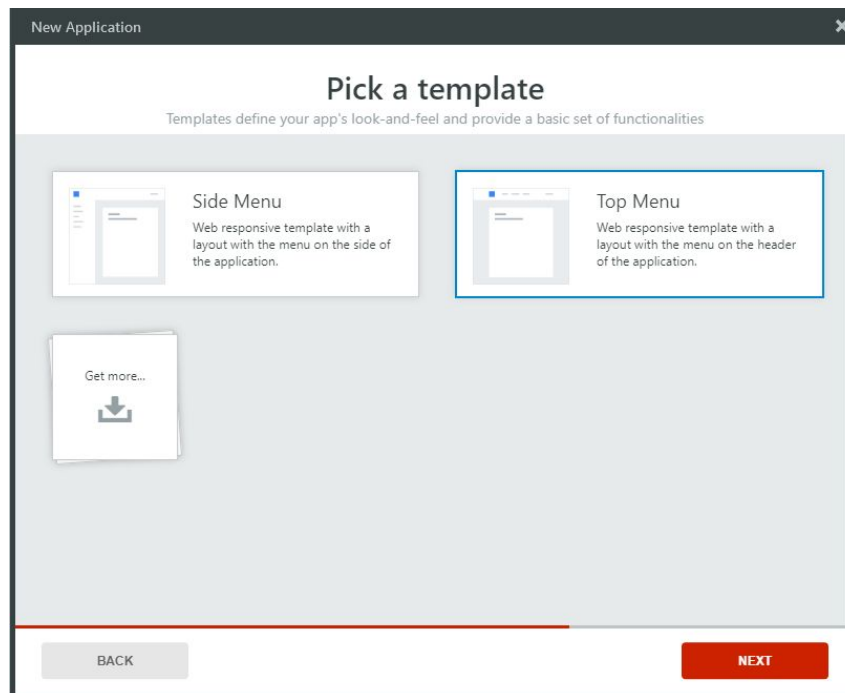
- a) Click the **New Application** button in Service Studio.



- b) Select the **Web App** option in the and click the **NEXT** button.



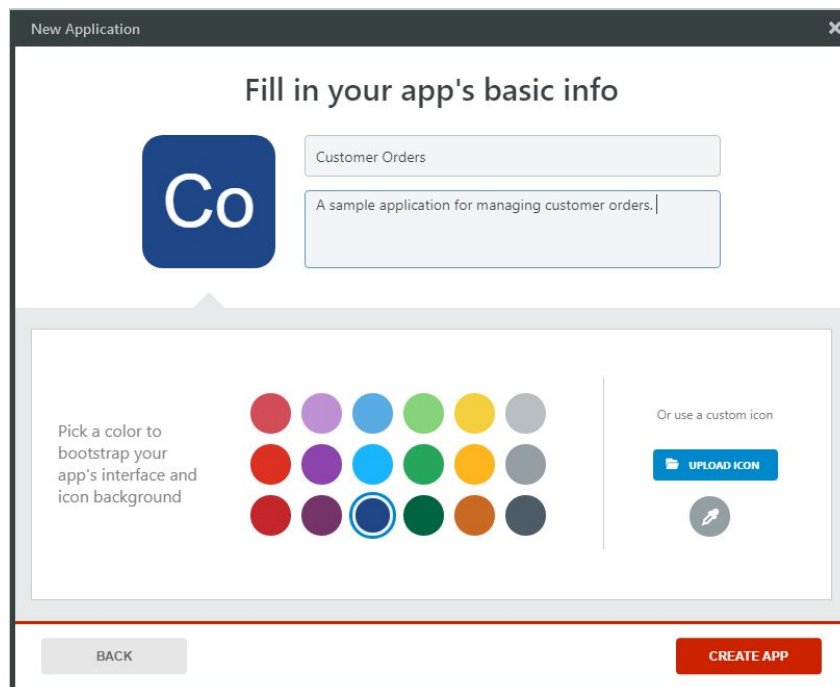
- c) Select the **Top Menu** option and click the **NEXT** button.



- d) Type the following **Name** and **Description** in the New Application window.

**Name:** *Customer Orders*

**Description:** *A sample application for managing customer orders.*



- e) Upload the Applcon.png from the Resources folder as the application icon and click on the Create App button.

- 2) Create a Web Responsive module, called *CustomerOrders*.

- 3) Click the 1-Click Publish to publish and save the module to the server. At this point we don't have Screens that we can use to test our application. The Screens will be added in the following sections.

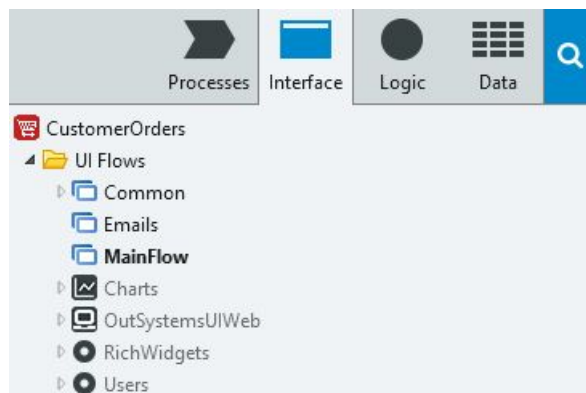


## Create the Dashboard Screen

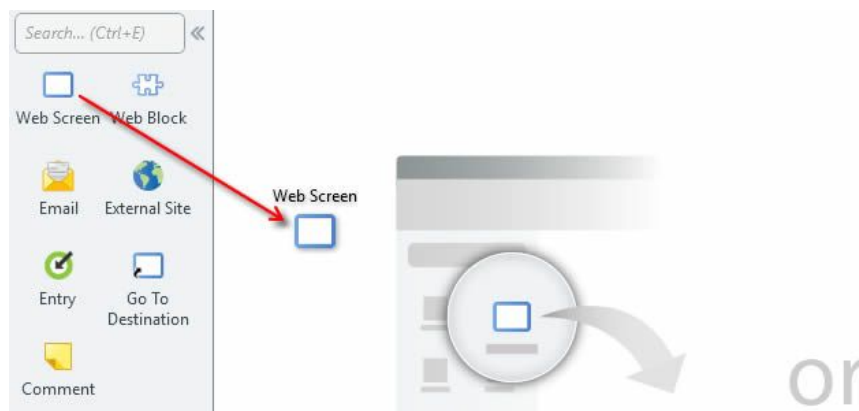
Now that we have the module, let's create the first Screen. We will start by creating the **Dashboard** Screen for high level order management. This Screen will consist of summary data, including categorized rows of information, and a pie chart for understanding the status of our orders at first glance.

This Screen will be created with a Screen template, which is a new feature of OutSystems 11. We have a list of templates that we can choose from, that answer to several business requirements. From that, we have a fully functional Screen, supported by sample data, which we can later customize to our needs.

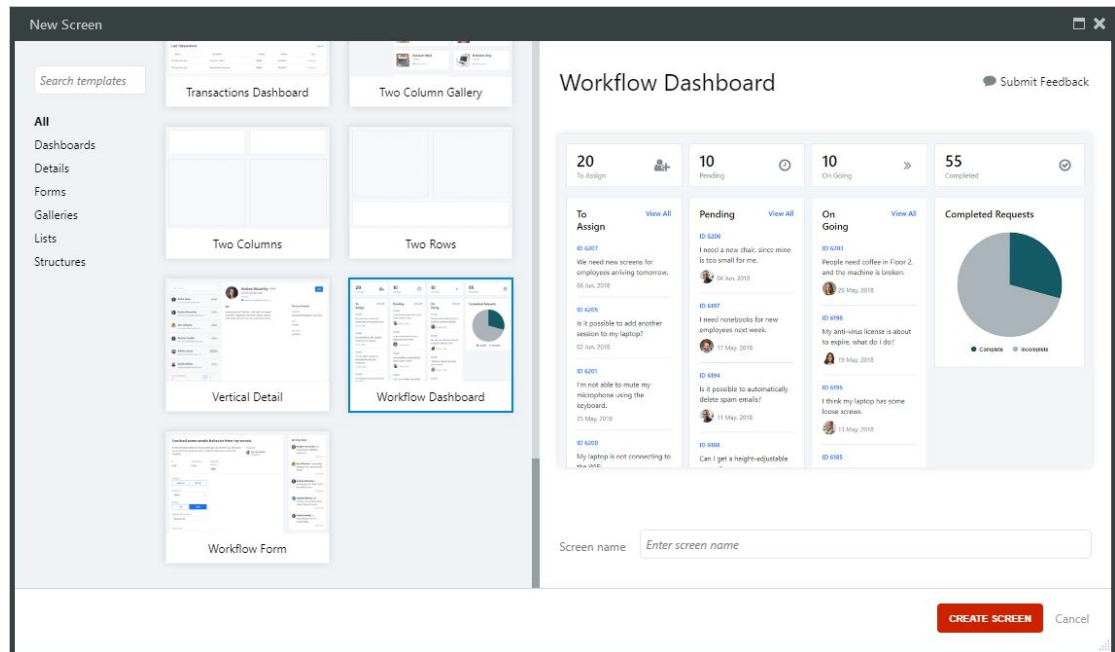
- 1) Create the **Dashboard** Screen using the **Workflow Dashboard** template in your Customer Orders module. Set the Screen access as **Anonymous**.
  - a) In the Interface tab, open the **MainFlow**. This is an UI Flow where multiple Screens can be created.



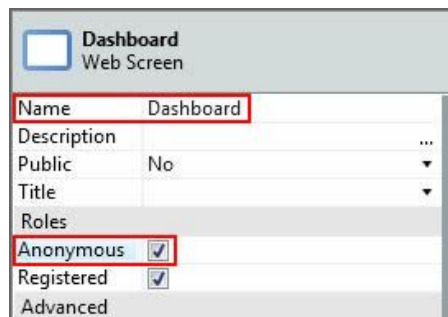
- b) Drag and drop a **Web Screen** from the toolbox to the MainFlow.




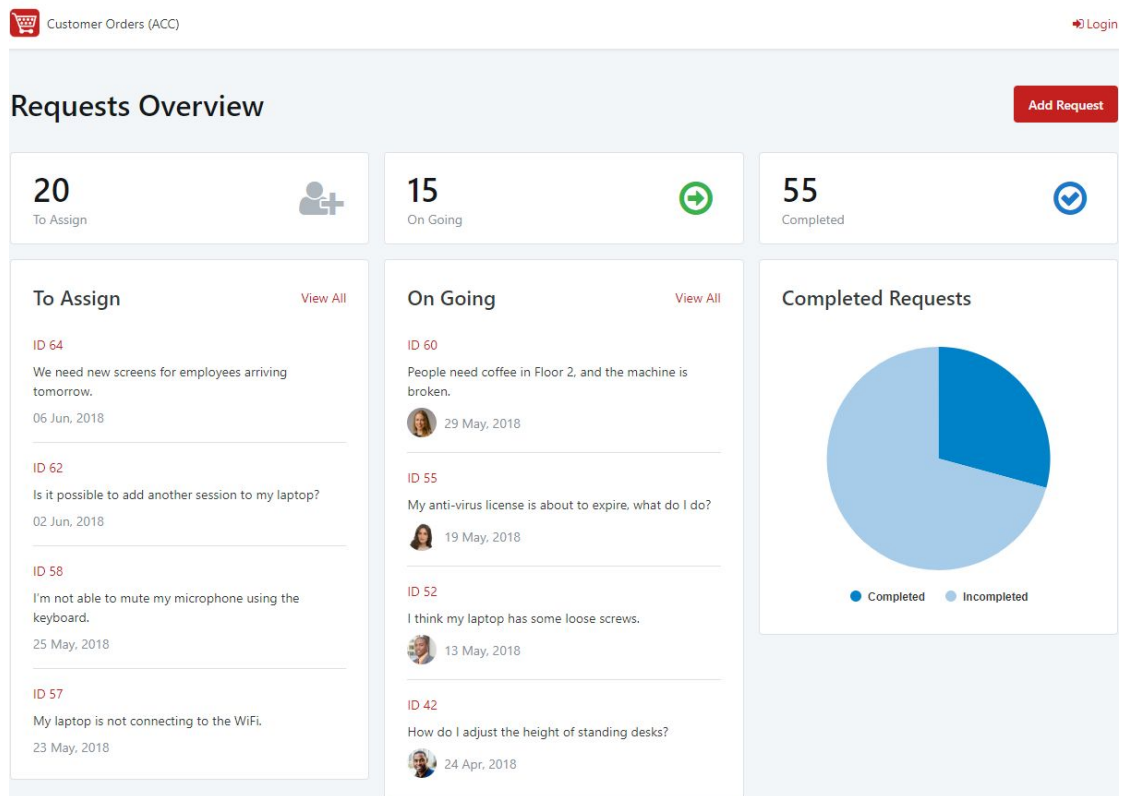
- c) Scroll down to the bottom left of the New Screen window and select the **Workflow Dashboard** template and then click the **Create Screen** button.



- d) In the properties editor of the Screen, change its name to *Dashboard* and tick the **Anonymous Role** checkbox, so that anyone can access the Screen without login.



- 2) Publish the application using 1-Click Publish  button and open the application in the browser. We can see that a fully functional Screen is already created.



## Use real data in the Dashboard Screen

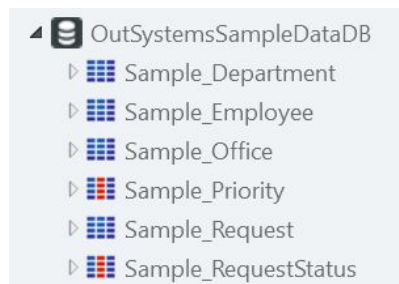
The Screen Templates create fully-functional Screens, using Sample Data, that is created when the Screen is created as well. In this section of the lab, we will replace the sample data by the real Customer Orders data.

### Reference the Customer Orders Data Model

In the next few steps, we will examine the sample data model, connect to our data model, and then customize the application to present our customer orders data.

The Screen was created based on a sample data model with Requests, Employees and Departments. The sample data will depend on the template chosen.

- 1) Examine the sample Entities that are used in the Workflow Dashboard Screen template.
  - a) Expand the **OutSystemsSampleDataDB** section under the Database section, in Service Studio's Data tab.

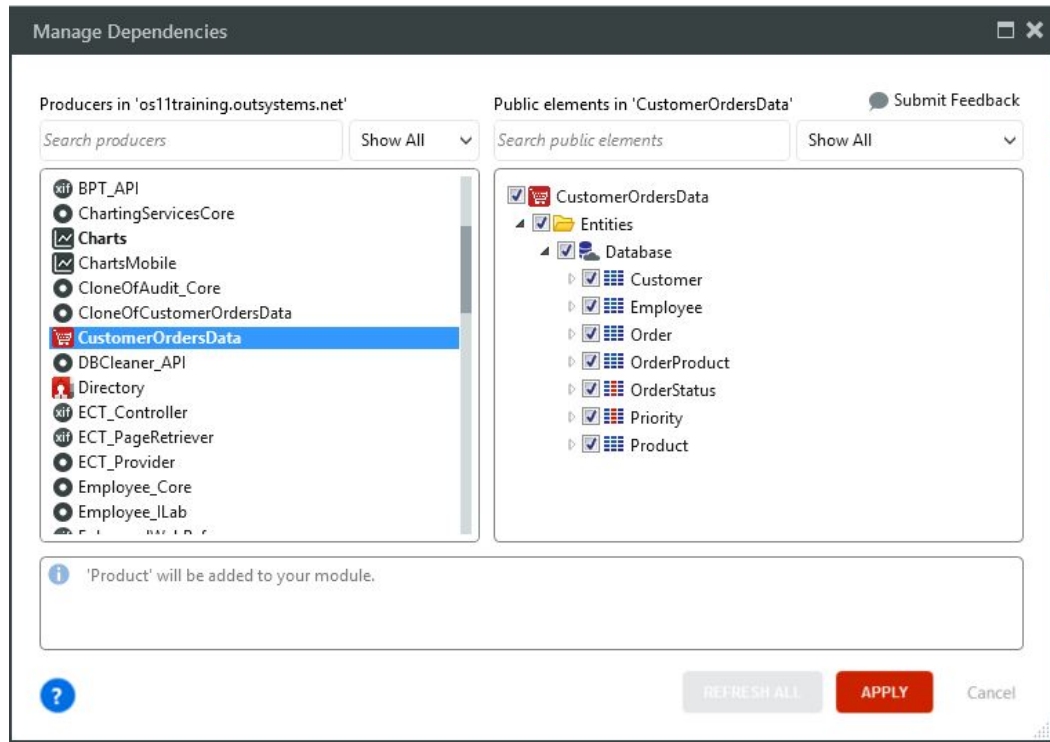


- b) Here we can find the Entities and Entity Attributes that are used from the sample data model in the Screen template.
  - c) Expand each of the Entities to examine its attributes. Notice that these Entities have some differences when compared to the ones in the Customer Orders Data.
- 2) In the **CustomerOrders** module, reference the Entities defined in the Customer Orders Data module.
  - a) In the CustomerOrders module, click the **Manage Dependencies** icon in the Menu Bar at the top left.

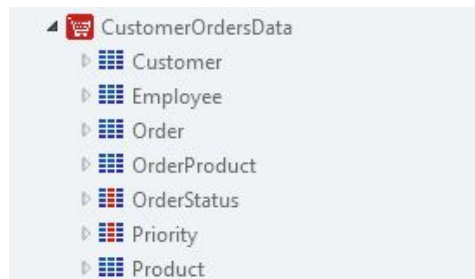


- b) In the Manage Dependencies window, scroll in the producers window (left side) to find the **CustomerOrdersData** module.

- c) Select the module on the left, and then tick the checkbox at the top of the Public Elements window (right side) to include all of the public Entities.



- d) Click the **OK** button to proceed.
- e) We should now be able to find the referenced Entities and Static Entities under the Database section.



## Replace the data in the Dashboard Screen

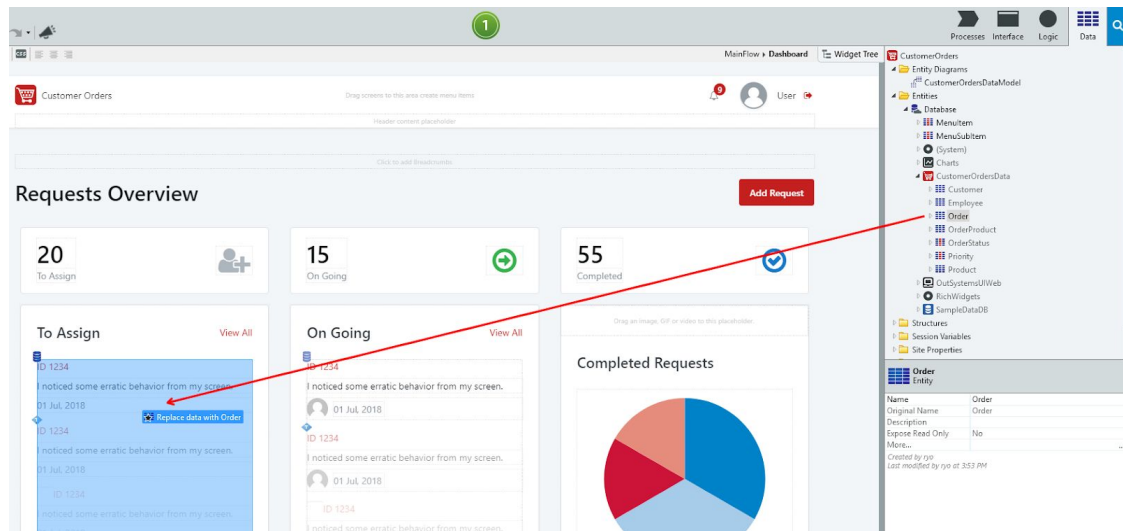
Now that we've referenced the data model we need, it's time to customize the Dashboard Screen to use this data, instead of using the sample data for customer requests.

We will use the new Replace Data feature from OutSystems 11 to help us replacing the existing sample data by our Customer Orders Data. This can be easily done by dragging and dropping

Entities from our data model to the Screen. However, since our data model is not an exact match to the sample data, we will have some extra work to do in this process.

- 1) In the Dashboard Screen, update the **To Assign** column, by dragging the **Order** Entity to the column. Fix the errors that may appear, using the Order Description.

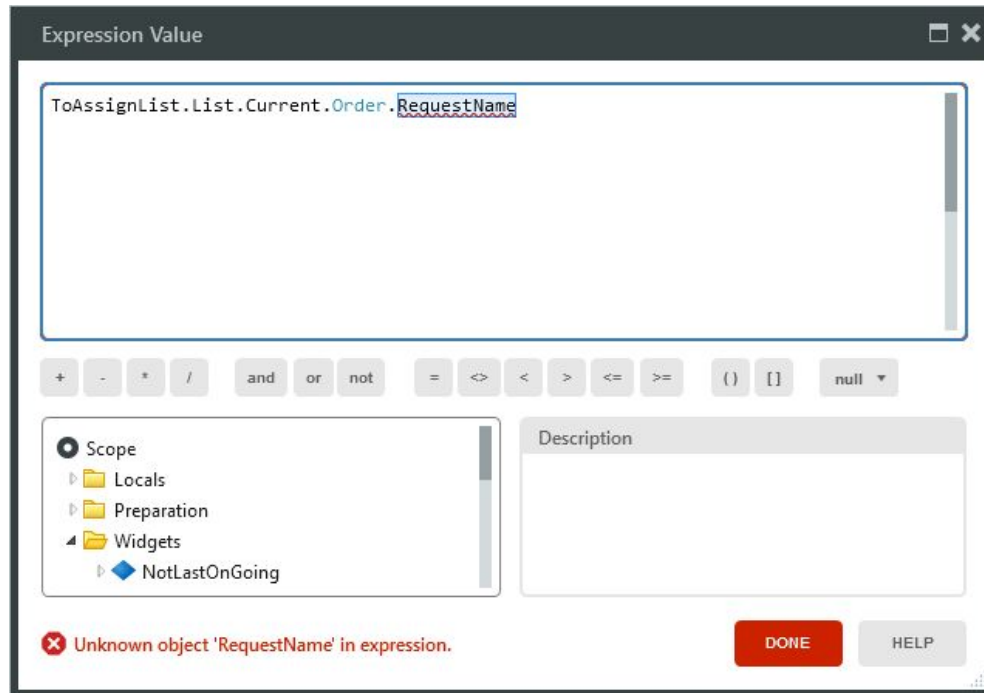
  - a) Drag the Order Entity to the **To Assign** column on the left side of the Screen. Release the mouse when the **Replace data with Orders** tooltip appears.



- b) The **True Change** area in the lower left of Service Studio shows the errors that need to be fixed. Since the Sample\_Request and the Order Entities do not match perfectly, the automatic Replace Data algorithm is not be able to replace and fix all usages of the Sample\_Request for the Order Entity. In this case, we need to replace the *RequestName* field with the *Description* field of the Order Entity.



- c) Double click on the **Invalid Expression** error in the True Change area. This will open the Expression Editor window.

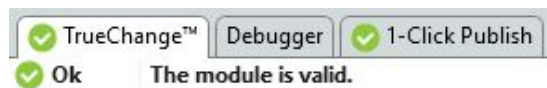


- d) Update the expression in the window to the following:

*ToAssignList.List.Current.Order.Description*

You may also drag the **Description** attribute of the Order Entity to the Expression with the error. This will replace the existing Expression by a new one using the dragged attribute.

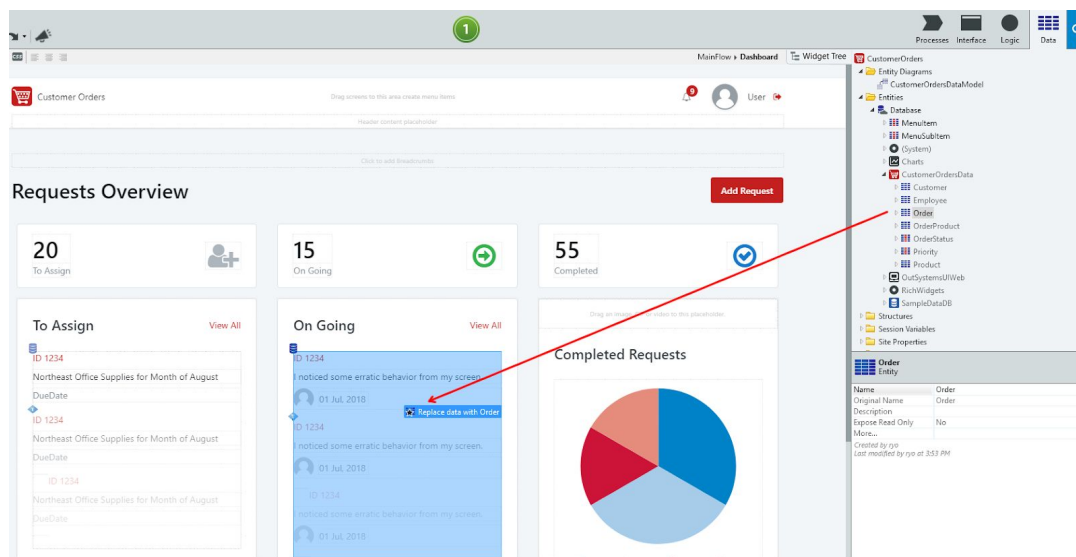
- e) The error message should be replaced with a green validation message.



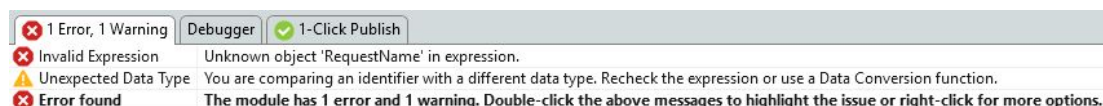
- 2) Following the same strategy, update the **On Going** column using the Order Entity, to list the Pending Orders, and fix all the errors and warnings that may occur. Use again the Order **Description** to replace the attribute mismatch, and then replace the filter in the GetOnGoingOrders Aggregate to check for the **Pending** Order Status.



- a) Drag and drop the Order Entity to the **On Going** column on the center of the Dashboard Screen.

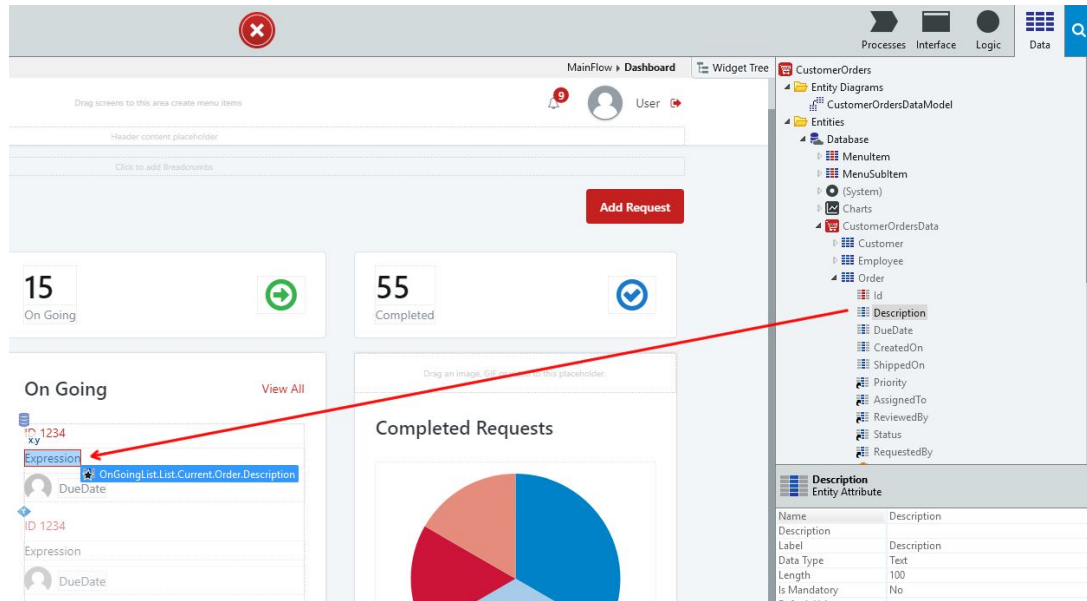


- b) Notice the error message in the **TrueChange** area in the lower left of Service Studio. Similarly to the previous Replace Data operation, we will need to replace the *RequestName* attribute with the *Description* attribute in our Order Entity. There is also a warning message that references the image on the Screen, as well as a data type comparison warning.



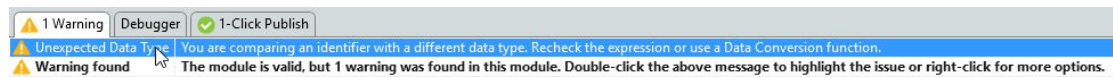


- c) Drag the **Description** attribute of the Order Entity to the expression highlighted in red.



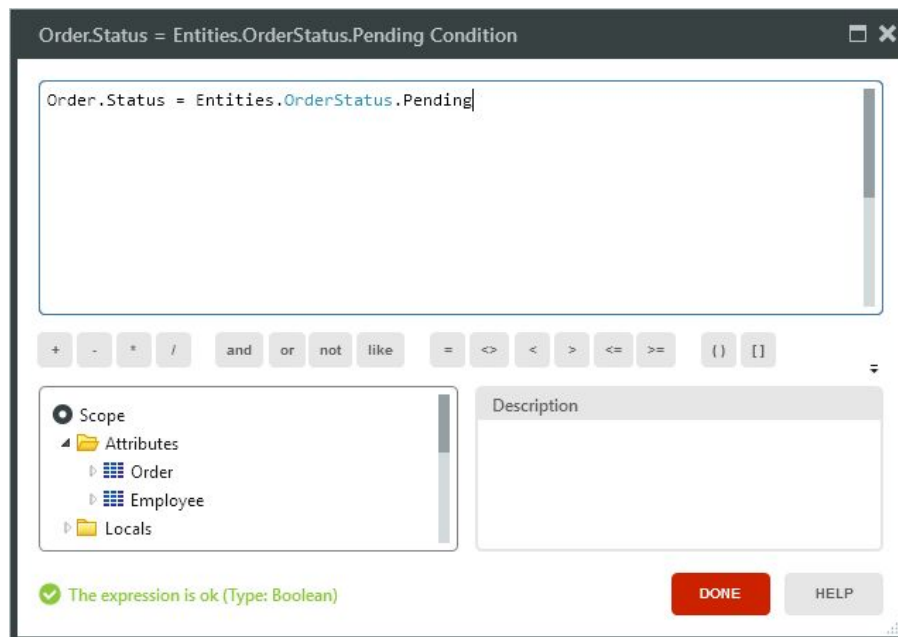
This drag and drop operation is similar to what was done with the Expression on the To Assign column before.

- d) Double-click on the final warning message in the TrueChange area.



The Expression Editor was opened. This will allow you to edit the **GetOnGoingOrders** Aggregate Filter. This Aggregate is defined inside the Preparation of the Dashboard Screen, and it is used to fetch Pending Orders from the database to populate the Pending column in the Screen.

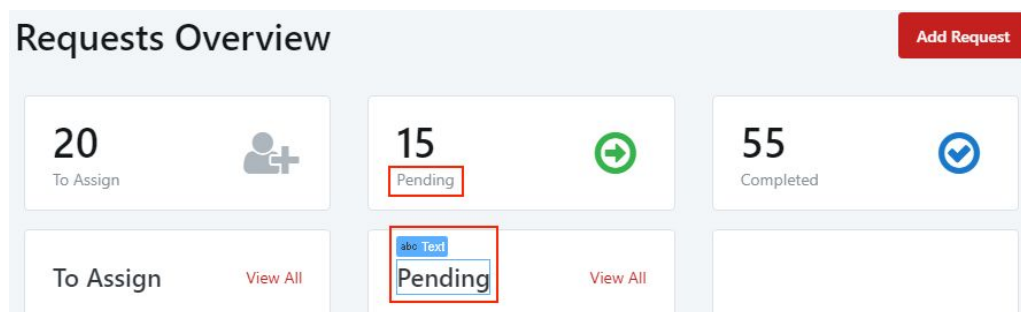
- e) Fix the Expression of the filter, to search for Pending Orders, by changing it to  
*Order.Status = Entities.OrderStatus.Pending*



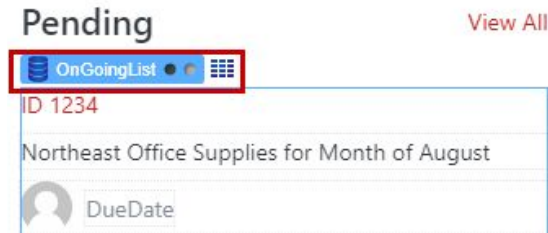
- f) Close the Expression Editor and then change the Aggregate name to **GetPendingOrders**.



- g) At this point, all error and warning messages should be gone.  
 h) Go back to the **Dashboard** Screen and change the *On Going* static text to *Pending*.

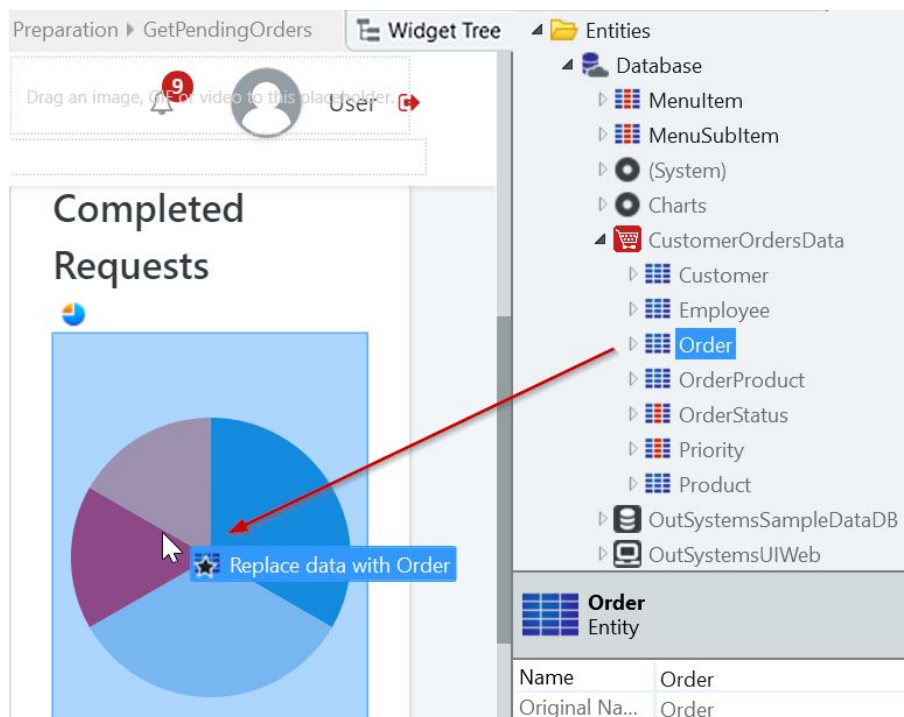


- i) Select the **OnGoingList** and change its name to *OnPendingList*.



- 3) Update the **Completed Requests** column and the Pie Chart, by dragging the Order Entity on top of it. The Pie Chart will show the number of Shipped Orders and the number of Processing Orders. Adjust the remaining logic to make sure the applications works properly with the Customer Orders Data Model.

- a) Drag the **Order** Entity from the Data tab and drop it on top of the **Pie Chart**.

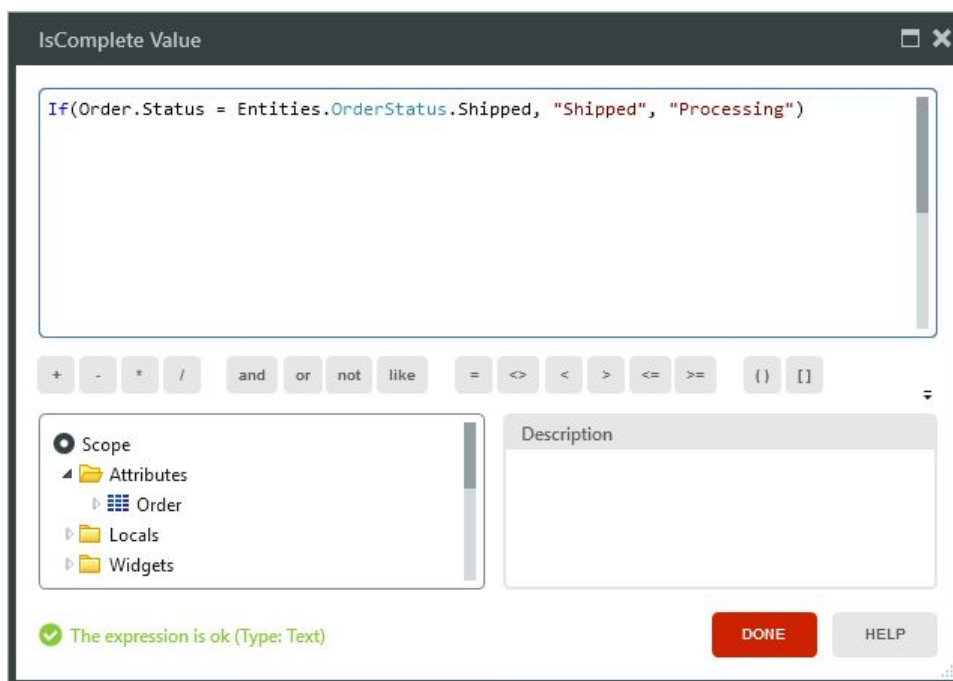


Notice that the title of the Screen has changed from *Requests Overview* to *Orders Overview*, and the *Add Request* button to *Add Order*. This extra automatic step occurs because the Screen no longer uses the Sample Data Model.

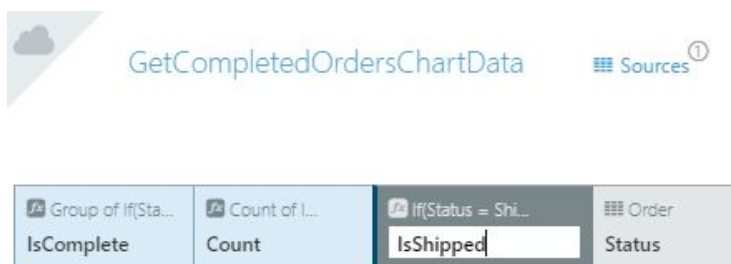
- b) Double-click on the **Unexpected Data Type** warning message in the TrueChange area. This will open the Expression Editor for the calculated attribute **IsComplete**. This attribute displays "Complete" in case the Sample Request was completed, or "Incomplete" otherwise. We now want to change it to show Shipped Orders, or Orders being processed.

- c) Update the expression to the following and then click the **Done** button.

*If(Order.Status = Entities.OrderStatus.Shipped, "Shipped", "Processing")*

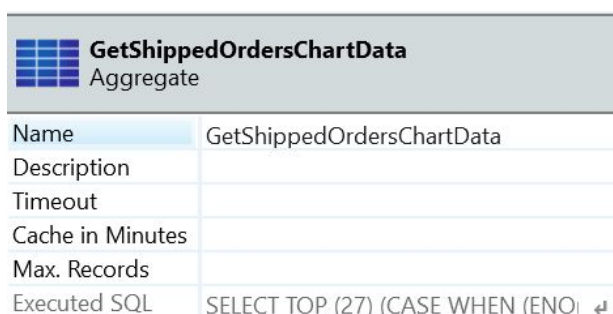


- d) Change the name of the **IsComplete** gray column to *IsShipped*.

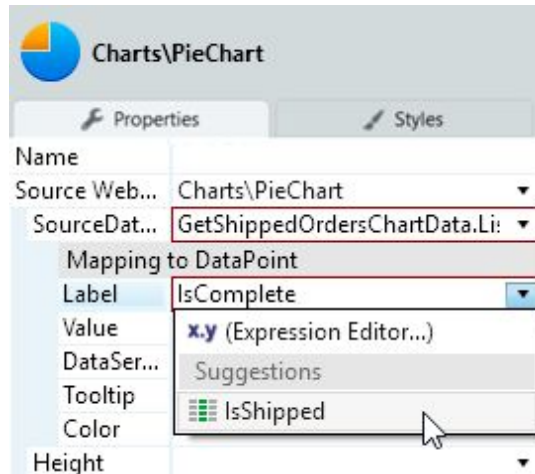


An error will show up in the TrueChange tab. We will fix that issue later on.

- e) Before closing the Aggregate, change the name of the Aggregate to *GetShippedOrdersChartData*.

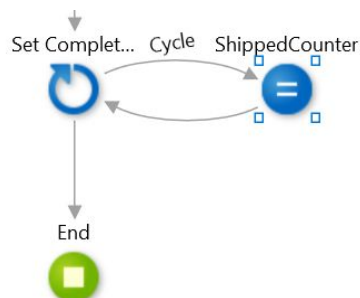


- f) Navigate back to the Dashboard Screen and select the **Pie Chart** Block. It should have a red border around it.
- g) Change its **Label** attribute, under **Mapping to DataPoint**, to *IsShipped*. This is the column in the Aggregate that we changed in the previous steps, to adjust the **Label** of the Chart to Shipped Orders.

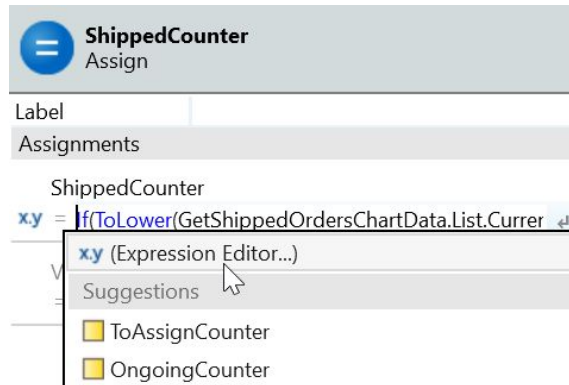


At this point, the TrueChange tab should be clear of all errors and warnings.

- h) Now we need to also update the number at the top of the Completed Orders column, to include the Shipped Orders. Let's start by changing the **Name** of the **CompletedCounter** Local Variable to *ShippedCounter*.
- i) Open the **Preparation** of the **Dashboard** Screen. Locate and select the **ShippedCounter** Assign near the End.

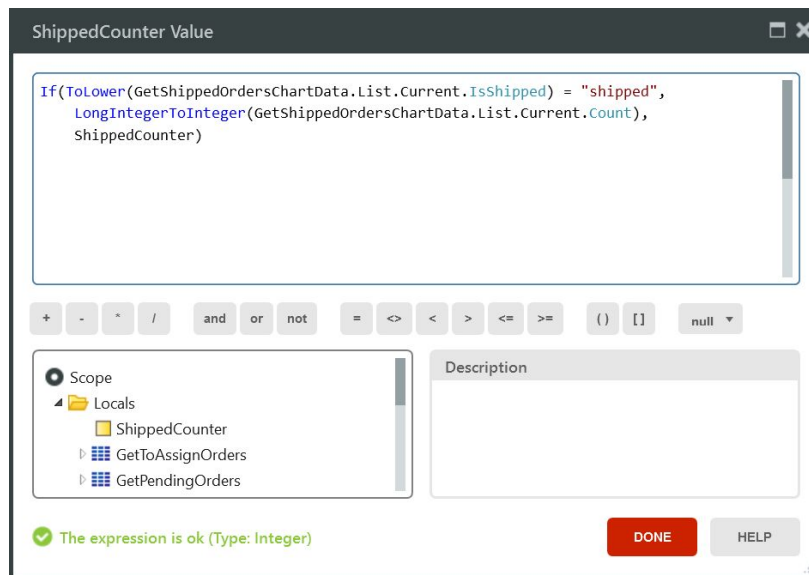


- j) Change the name of the AssignOpen the assignment expression of the **ShippedCounter** Variable by clicking the *(Expression Editor...)*.



- k) Change the expression to:

*If(ToLower(GetShippedOrdersChartData.List.Current.IsShipped) = "shipped",  
 LongIntegerToInteger(GetShippedOrdersChartData.List.Current.Count),  
 ShippedCounter)*

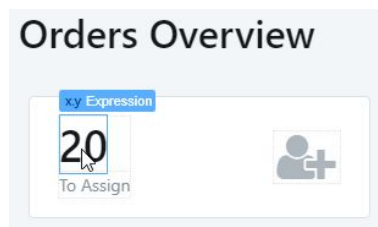


This adjusts the ShippedCounter variable for the current scenario of Shipped Orders, instead of the sample Completed Requests.

- l) Replace the *Completed* status in the two Text widgets, to use *Shipped*.



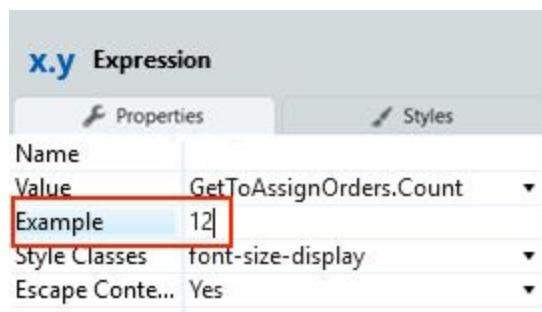
- 4) Fix the To Assign and Pending counters to display the total number of Orders for each Status, using the Counts of the respective Aggregates.
  - a) Double-click the Expression displaying the number of orders **To Assign**.



- b) In the Expression Editor, set the expression to:

*GetToAssignOrders.Count*

- c) Close the Expression Editor, then set the **Example** property to 12.

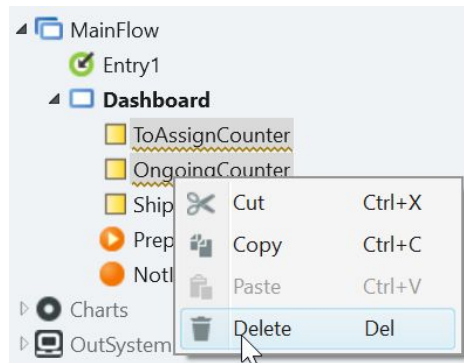


- d) Select the **Pending** Expression number, then change the **Value** property using the Expression Editor to

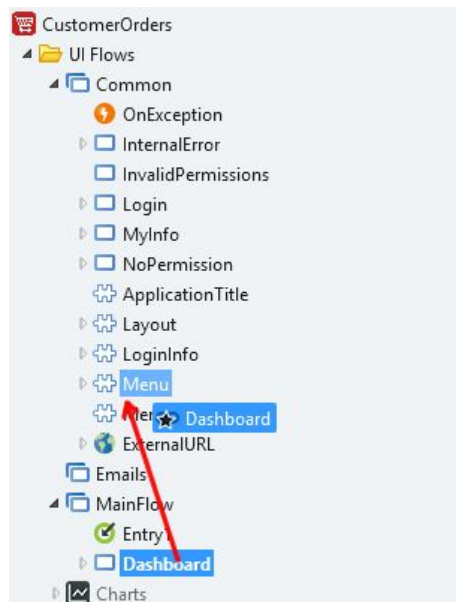
*GetPendingOrders.Count*



- e) Set the **Example** property to 27.
- f) At this point you should have two warnings stating that the **ToAssignCounter** and **OngoingCounter** Local Variables are never used. Delete the two variables under the Dashboard Screen.



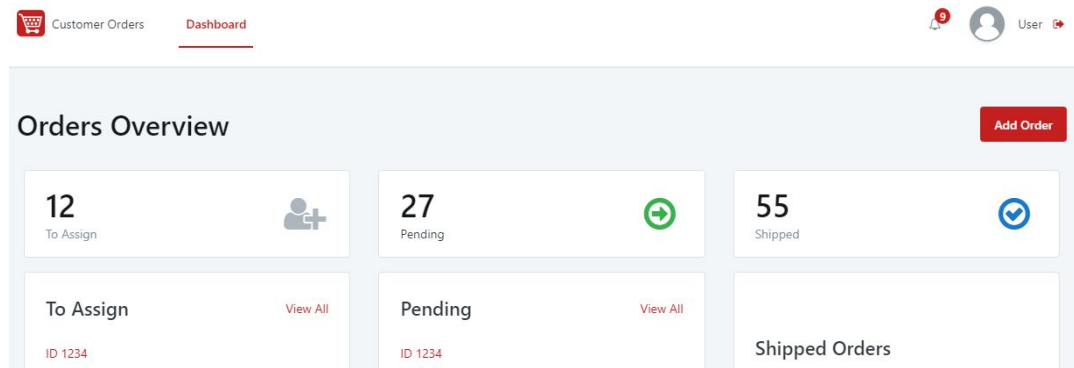
- 5) Add a Link to the Dashboard Screen in the Menu.
- a) In the Interface layer, expand the Common UI Flow.
- b) Drag the **Dashboard** Screen to the **Menu** Block.





This action will create a new Menu item and link it to the Dashboard Screen.

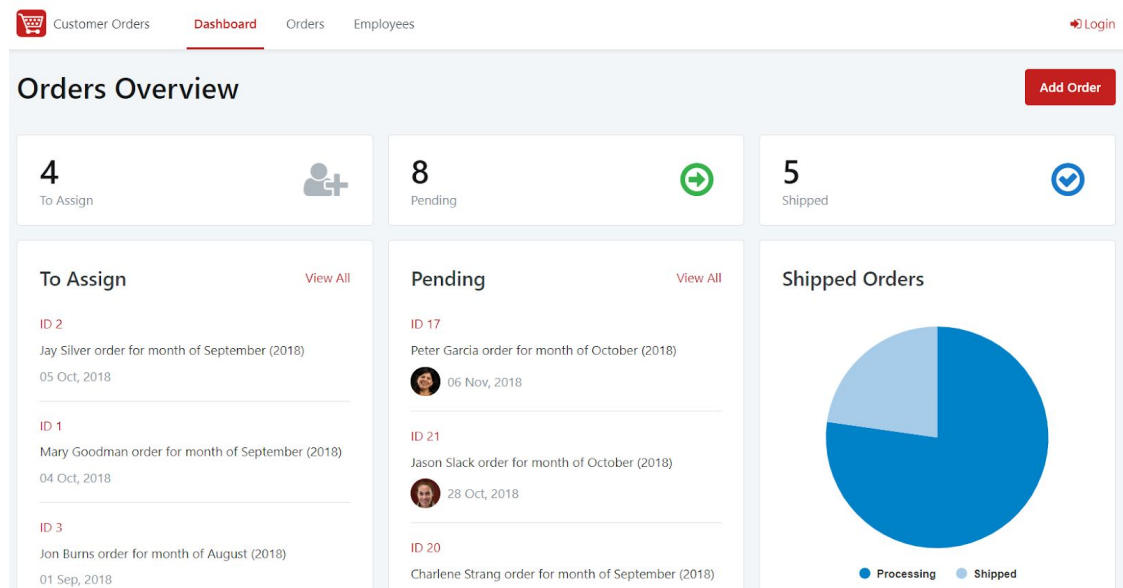


c) In Service Studio, your Screen should look like the image below



6) Publish and test the **Dashboard** Screen in the browser.

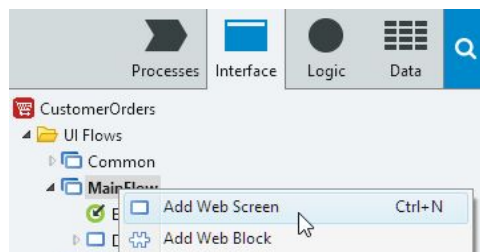
- Click the  **1-Click Publish** button to publish your application to the server.
- Click the  **Preview in Browser** button to preview your application.
- The Dashboard Screen should display your changes just like the following image.



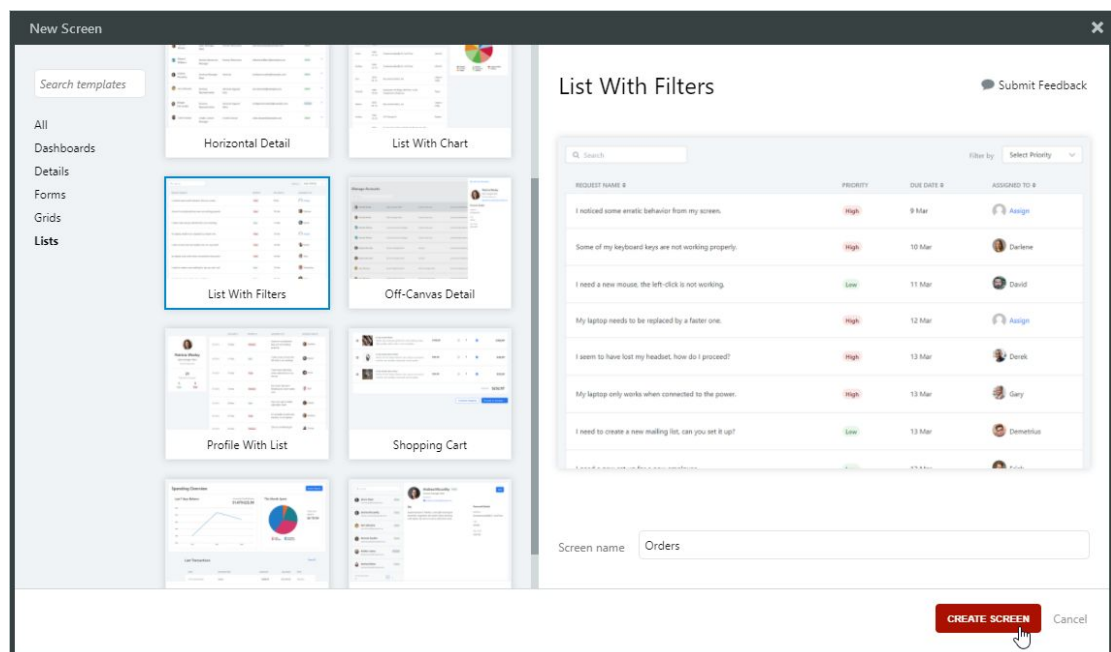
## Create and customize the Orders Screen

Our Dashboard Screen is looking good, but we need to create another Screen to display a list of all orders, sorted by their status. For this Screen, we will also use another Screen template to speed up the development process.

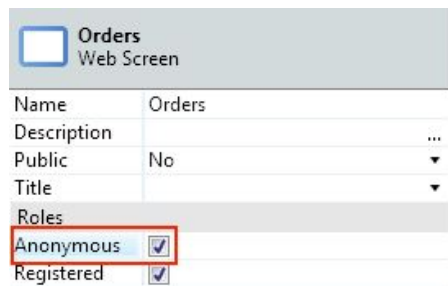
- 1) Create the *Orders* Screen based on the **List With Filters** Screen template and make it accessible anonymously.
- a) Right-click on the MainFlow and choose the **Add Web Screen** option.



- b) In the New Screen window, choose the **List With Filters** template from the Lists Category. Name the Screen *Orders* in the Screen name section, in the lower right corner, and click the **CREATE SCREEN** button.



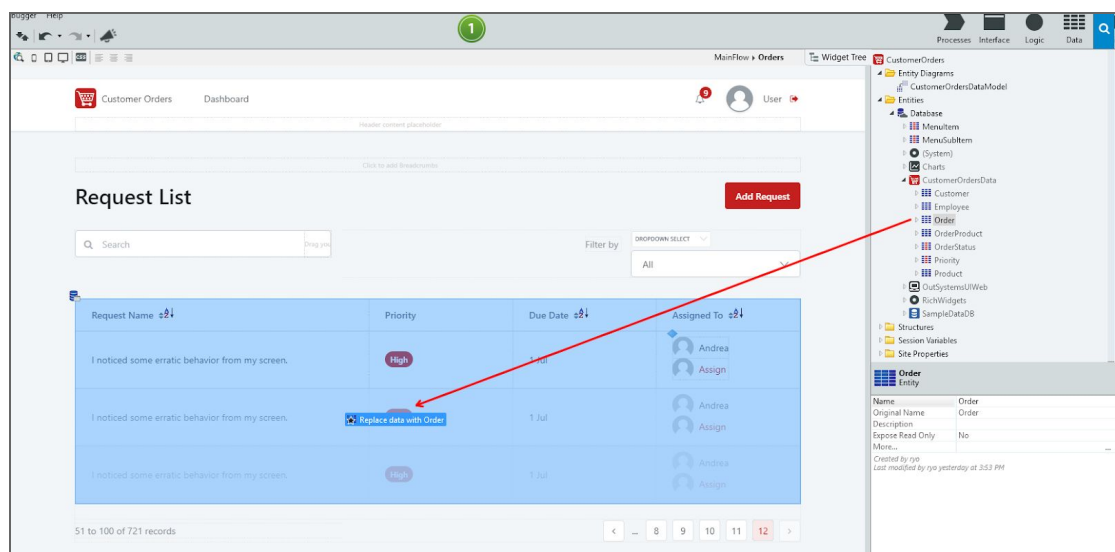
- c) Tick the **Anonymous** checkbox in the properties area.



<b>Orders</b> Web Screen	
Name	Orders
Description	...
Public	No
Title	...
<b>Roles</b>	
Anonymous	<input checked="" type="checkbox"/>
Registered	<input checked="" type="checkbox"/>

- 2) In the Orders Screen, customize the Table Records to list the Orders, and the Search field to search for Orders, using the Replace Data operation. Add the Status of the order as a new column of the Table Records.

- a) Drag and drop the **Order** Entity to the Table Records widget.



Notice that there are two errors. We will tackle these in the next steps.

- b) Double-click on the first error message to open the Filter defined in the **GetOrders** Aggregate. Change the expression to

*SearchKeyword = "" or Order.Description like "%" + SearchKeyword + "%"*

This Aggregate gets all the orders and is the source for the Table Records in the Orders Screen. This particular expression implements the first **Filter** of the Aggregate, which considers the Search of the Orders Screen. With the Sample Requests, the Aggregate was filtered by the *RequestName*. With Orders, we just want to filter by their *Description*.

- c) Double-click on the remaining error message to open the Expression Editor window, update the code to the following, and click the **Done** button.

*Table.List.Current.Order.Description*

This replaces the *RequestName* by the *Description* of the Orders in the Table.

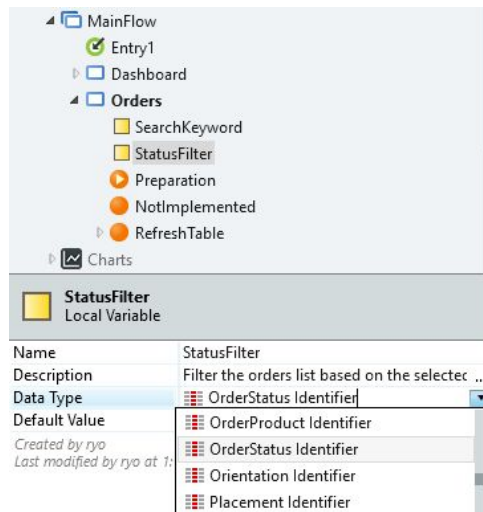
- d) In the **Table Records**, change the title of the first column to *Order*, and remove the existing Icon widget, right next to the column name. The Table Records should look like the image below

Order	Priority	Due Date	Assigned To
Northeast Office Supplies for Month of August	Low	12 Sep(5 days ago)	Employee Assign
Northeast Office Supplies for Month of August	Low	12 Sep(5 days ago)	Employee Assign
Northeast Office Supplies for Month of August	Low	12 Sep(5 days ago)	Employee Assign

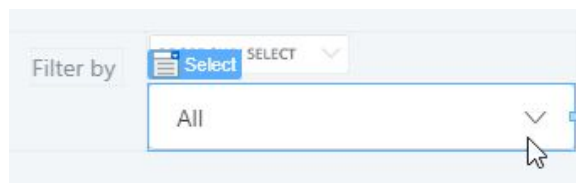
- e) Expand the **Order** Entity and drag the **Status** Attribute to the left side of the **Priority** column in the Table. This will add a new column to the Table Records with the status of each order.

- 3) Customize the **Orders** Screen filter to use the Orders instead of the Sample Requests. This will require changing some Filters, Widgets and Local Variables to work over **OrderStatus** instead of Priorities.

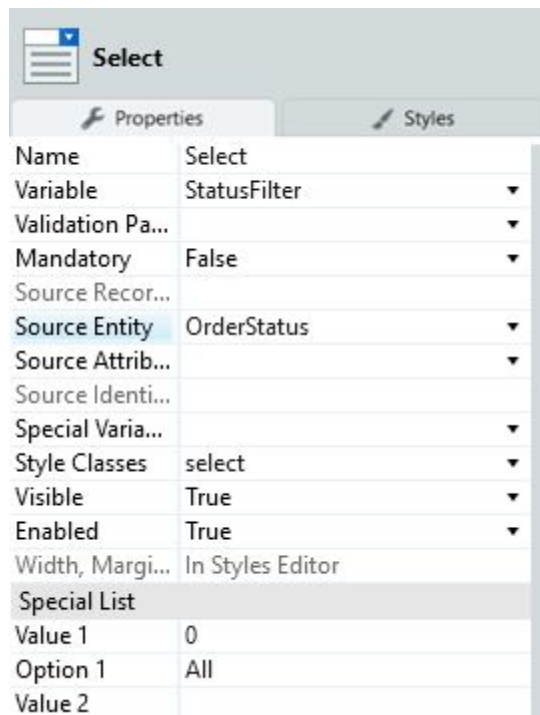
- a) In the Interface tab, expand the Orders Screen and select the **PriorityFilter** Local Variable. Change its name to *StatusFilter* and the **Data Type** to *OrderStatus Identifier*.



- b) Select the **Filter by** Combo Box on the Screen to view its properties.



- c) Change the **Source Entity** to *OrderStatus*.



It is also possible to drag and drop the *OrderStatus* Entity to the existing Combo Box and achieve the same result.

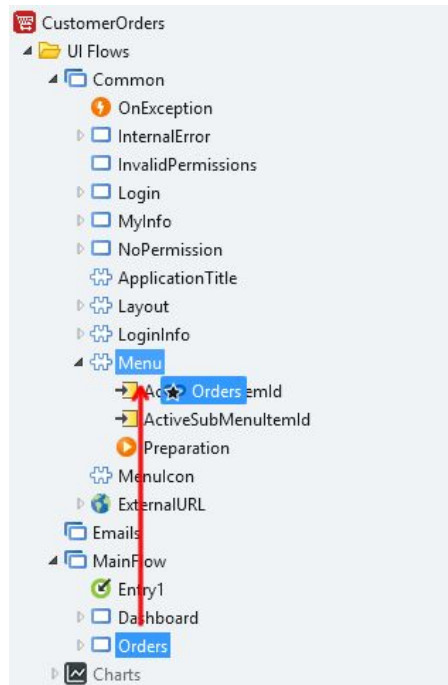
- d) Double-click on the **Unexpected Data Type** warning message in the TrueChange area to open the Expression Editor. This refers to the second **GetOrders** Aggregate filter. Change it to the following expression.

*StatusFilter = NullIdentifier() or Order.Status = StatusFilter*

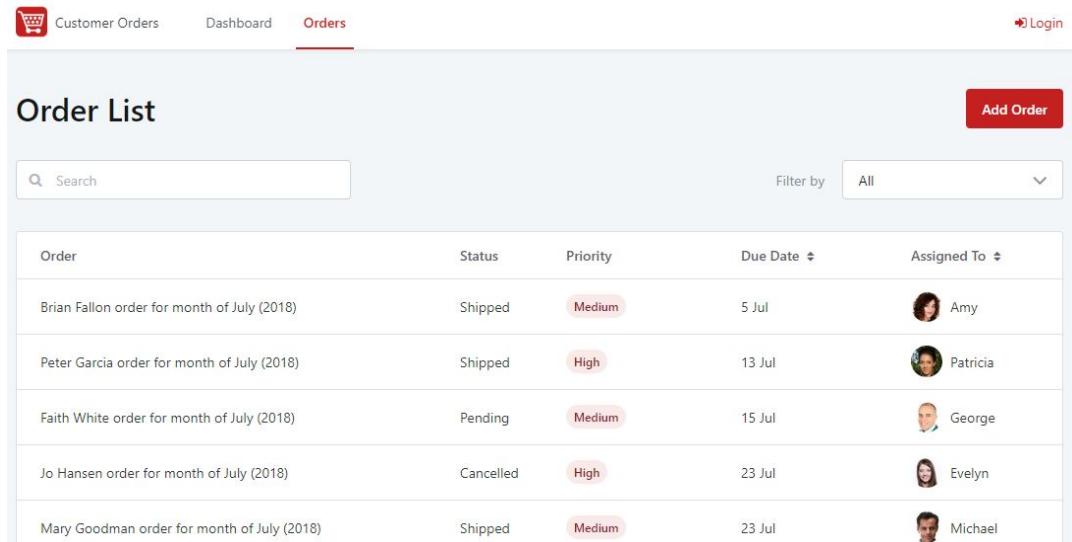
Now, this filter considers the option selected in the Search Combo Box.

- 4) Create the main menu Link to the new **Orders** Screen.
- Expand the **Common** UI Flow in the Interface layer of the application so the Menu Block is visible.

- b) Drag the **Orders** Screen from the MainFlow and drop it onto the Menu Block to create a Link in the main menu of the application.



- c) Publish and test the **Orders** Screen in the browser. Use the new **Orders** link at the top of the page to navigate to the Orders page. Don't forget to test the filter by Order Status.



Customer Orders   Dashboard   **Orders**   Login

## Order List

Search  Filter by All

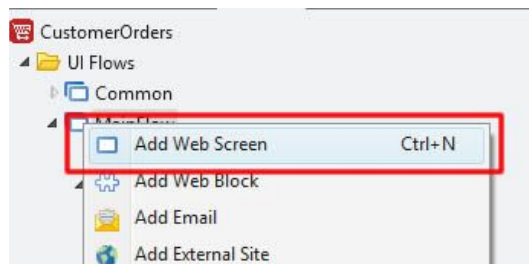
Order	Status	Priority	Due Date	Assigned To
Brian Fallon order for month of July (2018)	Shipped	Medium	5 Jul	Amy
Peter Garcia order for month of July (2018)	Shipped	High	13 Jul	Patricia
Faith White order for month of July (2018)	Pending	Medium	15 Jul	George
Jo Hansen order for month of July (2018)	Cancelled	High	23 Jul	Evelyn
Mary Goodman order for month of July (2018)	Shipped	Medium	23 Jul	Michael

[Add Order](#)

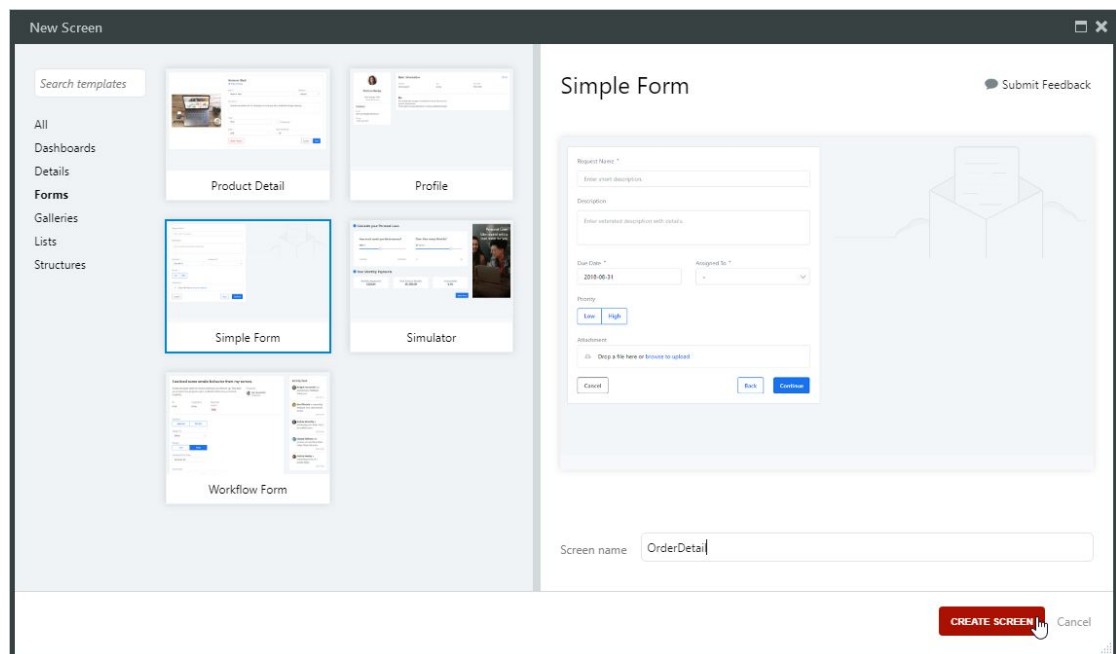
## Create and customize the OrderDetail Screen

The final Screen of this application will be the **OrderDetail** Screen. This Screen will allow editing existing orders, or creating new ones.

- 1) Create the OrderDetail Screen using the **Simple Form** Screen template and make it with Anonymous access.
  - a) In the Interface tab, right-click on the **MainFlow** and choose **Add Web Screen**.



- b) In the New Screen dialog, choose the **Simple Form** template from the Forms Category. Name the Screen *OrderDetail* and click the **Create Screen** button.

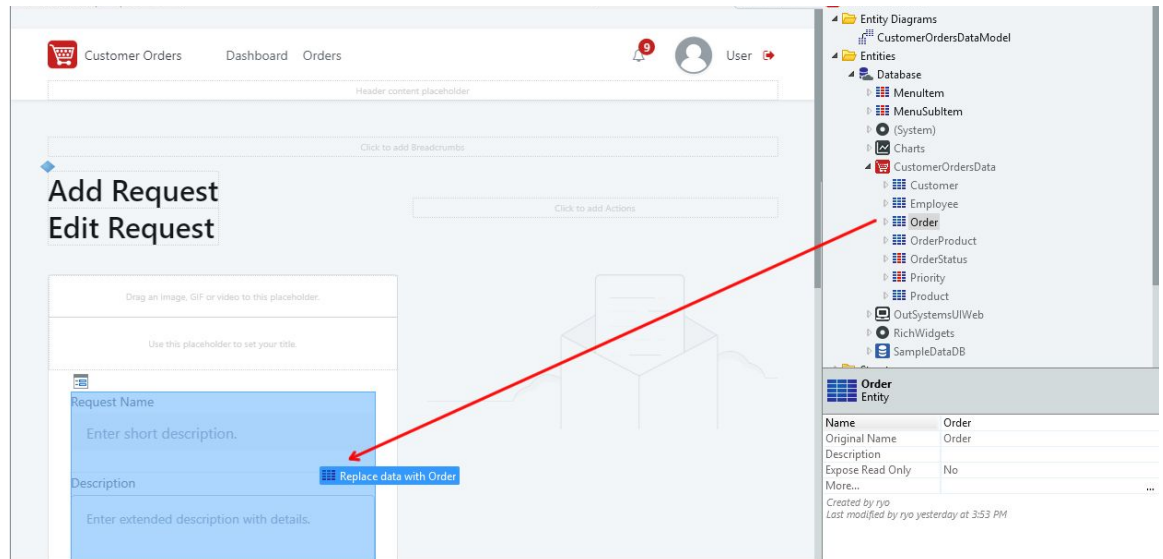


- c) Tick the **Anonymous** checkbox in the properties area.
- 2) Replace the sample data used in the **Form** of the OrderDetail Screen, by the Customer Orders Data Entities, namely the Sample\_Request by the Order and the Sample\_Employee by Employee. Also, make sure the Form is only Enabled when the

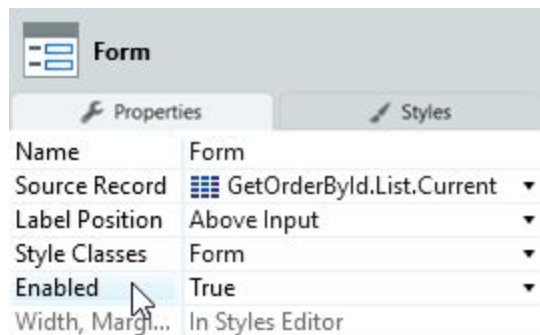


order was not already cancelled or shipped. Delete the fields in the Form that do not make sense for our Orders scenario.

- a) Drag the **Order** Entity to the **Form** on the OrderDetail Screen.



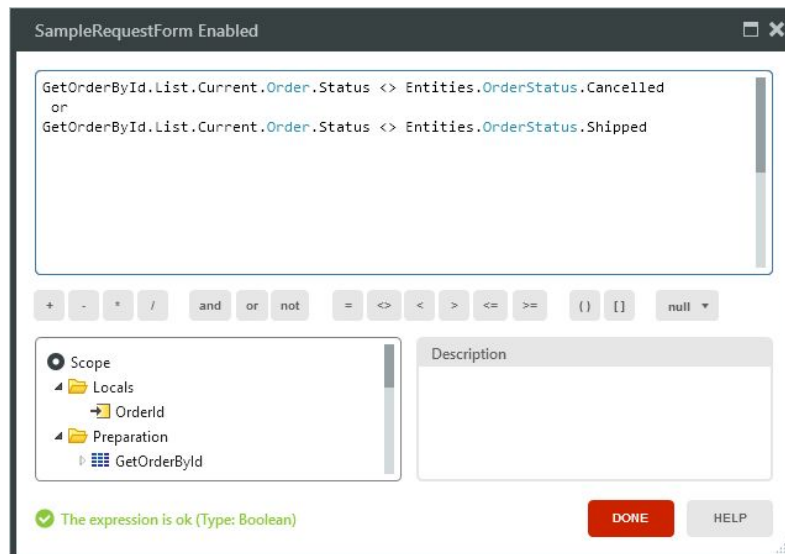
- b) Select the Form on the Screen and in its properties area, double-click the **Enabled** property to open the Expression editor.



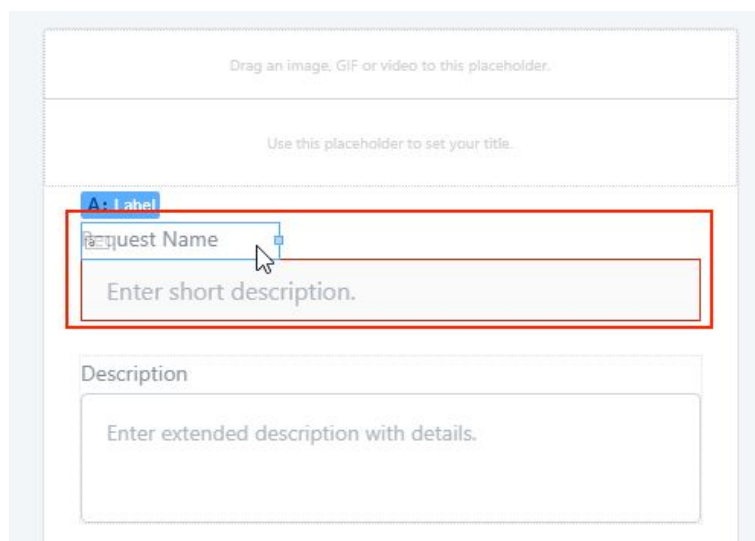
- c) Change the expression to:

*GetOrderById.List.Current.Order.Status <> Entities.OrderStatus.Cancelled  
or GetOrderById.List.Current.Order.Status <> Entities.OrderStatus.Shipped*

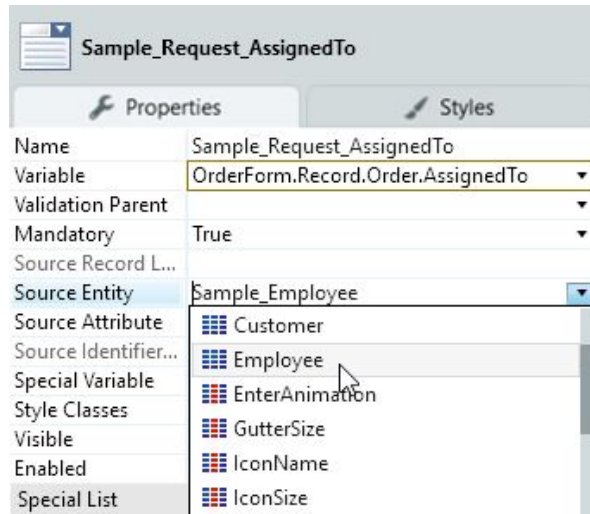
This expression disables the Form when the Order was Cancelled or Shipped, preventing further changes in these cases.



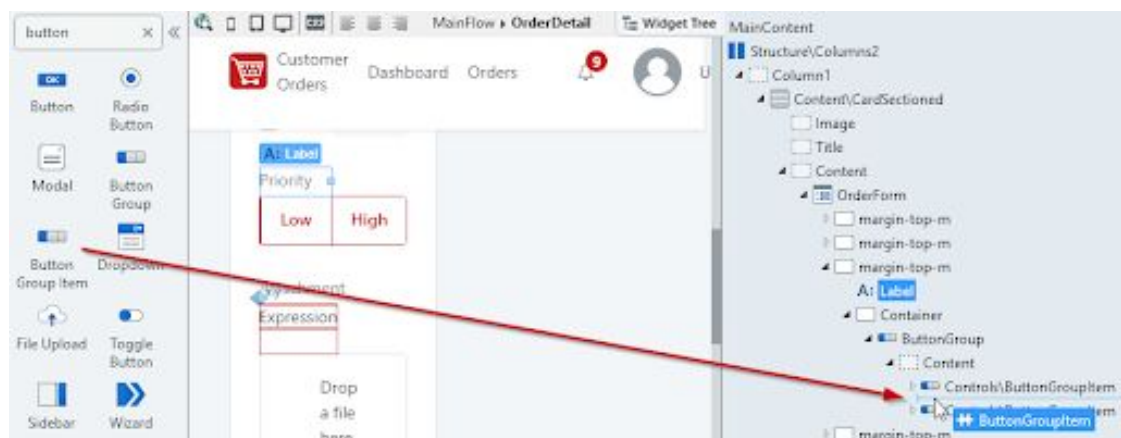
- d) Change the **Name** property of the Form to *OrderForm*.
- e) Select the **Request Name Label** and then delete it. Delete also the **Request Name Input** that was next to the Label.



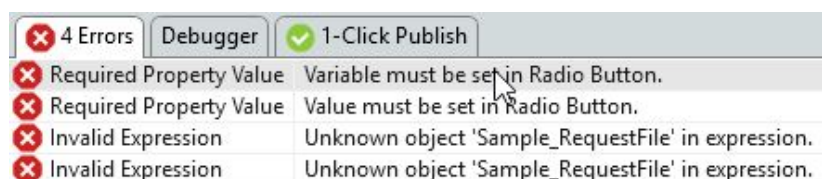
- f) Select the Input Widget with a warning, the **Assigned To** Combo Box, then change the **Source Entity** property from *Sample\_Employee* to *Employee*.



- g) Drag a new **Button Group Item** Widget and drop it **between the Low and High** Button Group Items. Use the Widget Tree to help.

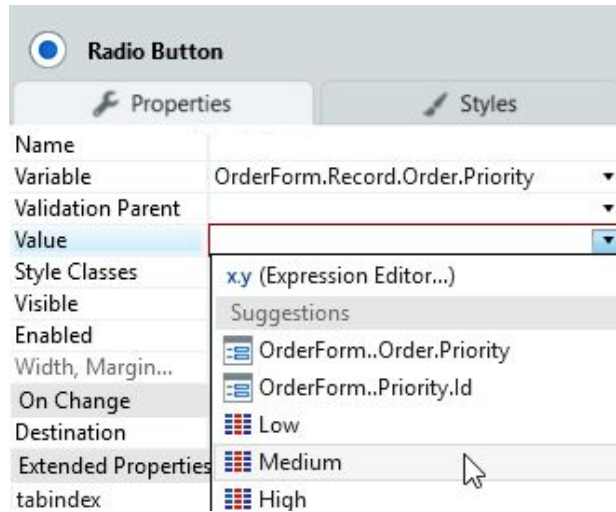


- h) Change the **Label** text to *Medium*.
- i) Click on the first error in the TrueChange, to select the Radio Button that is associated with the Button Group Item that we just dragged.



- j) Set the **Variable** Property of the Radio Button to *OrderForm.Record.Order.Priority*

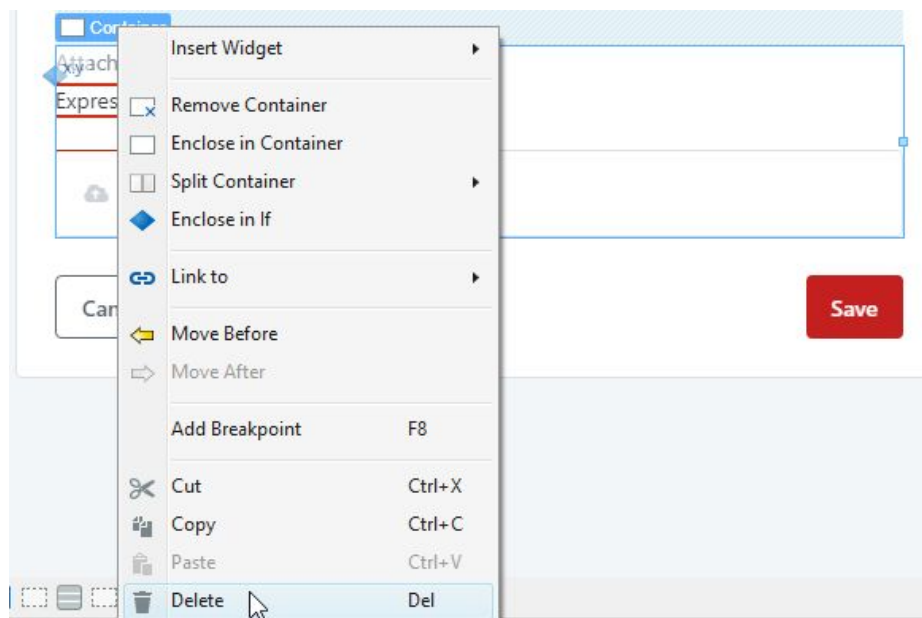
- k) Set the **Value** property to *Medium*. This Button will refer to the **Medium** Priority, which did not exist in the Sample Data.



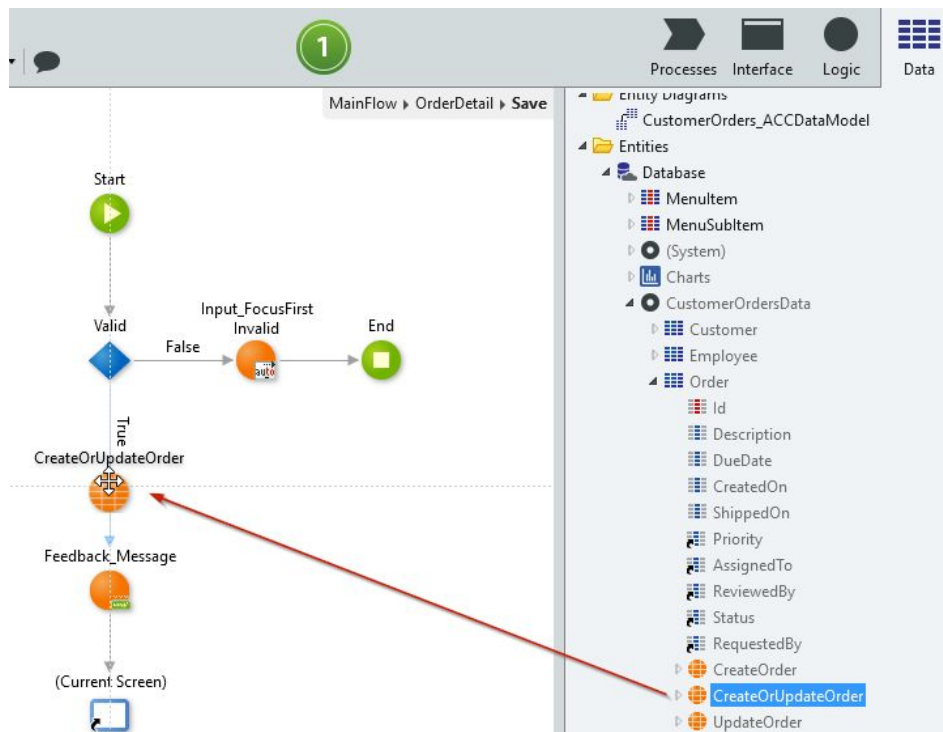
- l) At this point, the Priority Buttons should look like this



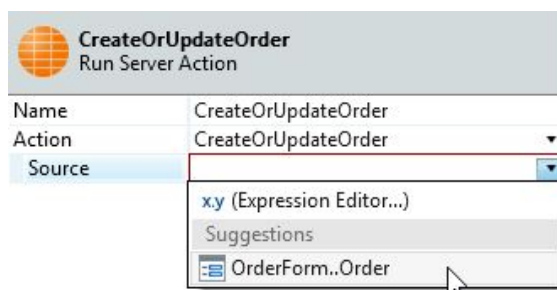
- m) The last Input on the Form is the **Attachment**, which we don't need. On the Widget Tree, select the last Container under the OrderForm and delete it.



- 3) Fix the logic used in the **OrderDetail** Screen (Save Action) to allow creating or updating an order in the database.
  - a) Double-click the **Save** Button to open the **Save** Screen Action.
  - b) From the Data tab, under the Order Entity, drag the **CreateOrUpdateOrder** Entity Action and drop it between the If statement and the Feedback Message. This will Save a new Order in the database, when a user clicks on the Save Button. So far, this Action didn't do anything besides showing a Feedback message to the user.



- c) Set the Source property of the **CreateOrUpdateOrder** to *OrderForm.Record.Order*



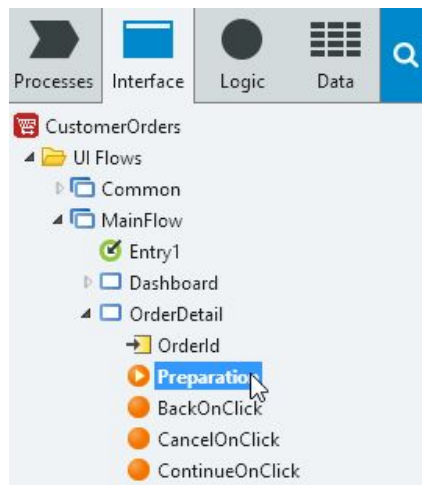
The record of the Form will be saved in the database.

- d) Select the **Feedback\_Message** statement then change the **MessageText** parameter to *"Order saved."*

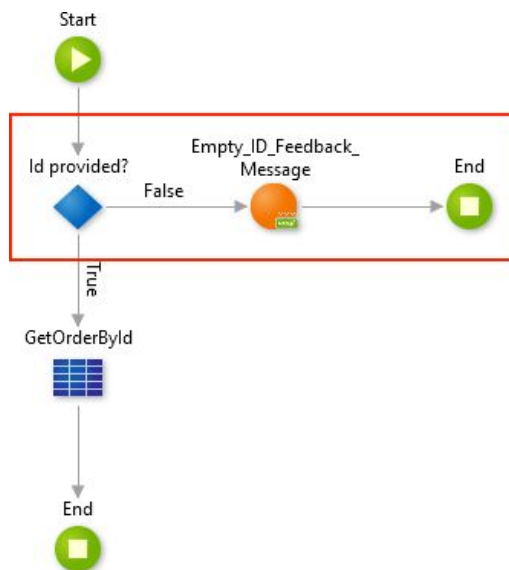
Feedback_Message Run Server Action	
Name	Feedback_Message
Action	Feedback_Message
MessageText	"Order saved."
MessageType	Entities.MessageType.Info

- 4) Remove the **Feedback Message** from the **OrderDetail** Screen Preparation.

- a) From the Interface layer, open the **OrderDetail** Screen Preparation.



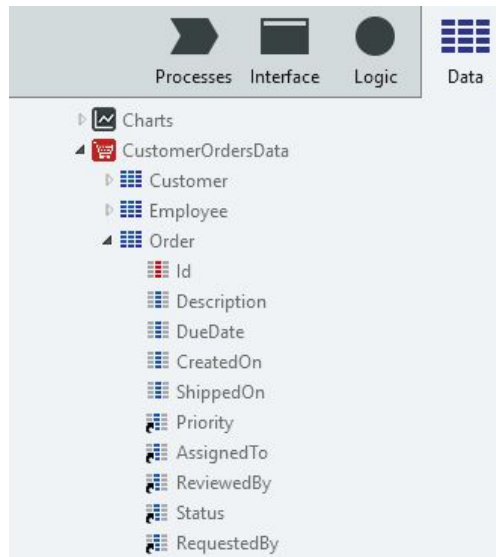
- b) Delete the **If** statement and all the elements in the False branch.



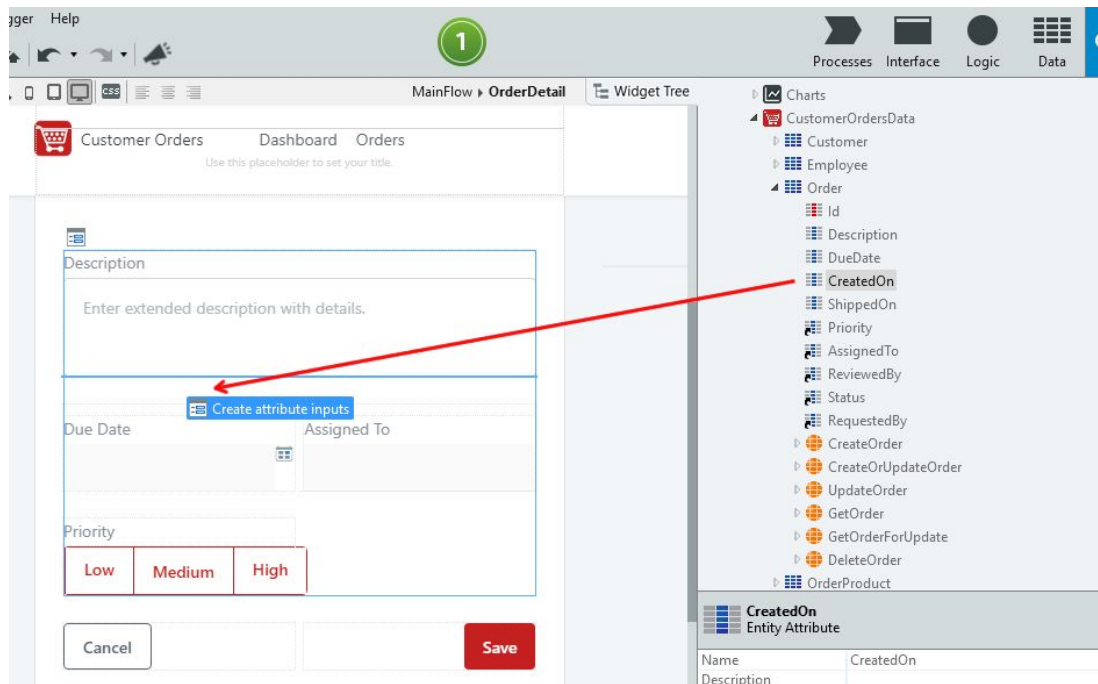
If needed, add the connector from the **Start** to the **GetOrderById** Aggregate.



- 5) Add the Inputs for the **CreatedOn**, **ReviewedBy** and **Status** attributes to the Form on the OrderDetail Screen.
- a) Open the **OrderDetail** Screen, then switch to the Data tab and expand the **Order** Entity.



- b) Drag and drop the following Entity Attributes to the Form, below the **Description**, in the following order: CreatedOn; ReviewedBy; Status.



c) The Form in the Screen should look similar to the following screenshot.

The screenshot shows a form with the following fields and controls:

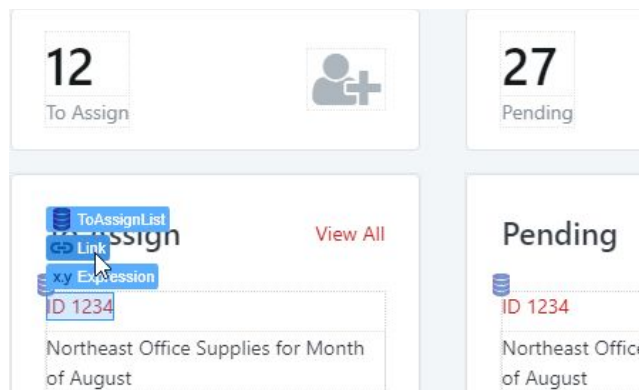
- Description:** A text area with the placeholder text "Enter extended description with details."
- Created On:** A date picker field showing the date "14".
- Reviewed By:** A text field with a hyphen "-" as the value.
- Status:** A text field with a hyphen "-" as the value.
- Due Date:** A date picker field.
- Assigned To:** A text field.
- Priority:** A set of three buttons labeled "Low", "Medium", and "High". The "Low" button is currently selected.
- Buttons:** A "Cancel" button and a "Save" button.



## Link the Screens

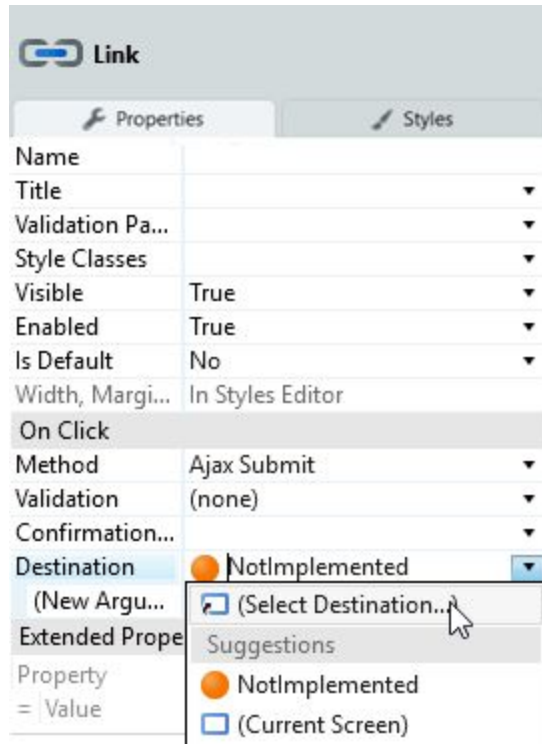
We have created three working Screens very quickly and they are all using the data from our Customers Orders Data application. In the next few steps, we will link these Screens together. By the end, it will be possible to navigate from the **Dashboard** to the **OrderDetail** Screen, and also from the **Orders** to the **OrderDetail**.

- 1) Update the Links on the **Dashboard** Screen to navigate to the **OrderDetail** Screen. Add a **Link in each Order** of the **ToAssign** and **Pending** columns to the OrderDetail Screen.
  - a) Open the Dashboard Screen.
  - b) Select the **ID 1234** Expression in the **To Assign** column and select the Link surrounding it.



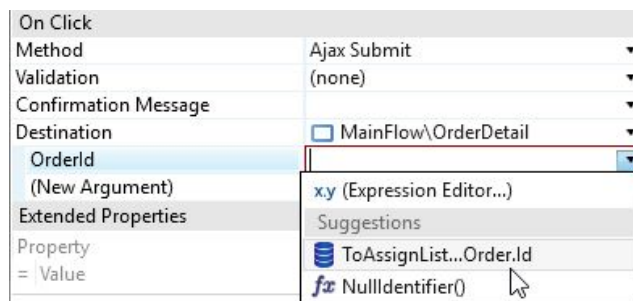
When a widget is selected, hovering the tooltip displays the widgets hierarchy.

- c) In the properties of the Link, open the suggestions dropdown of **Destination** property, select *(Select Destination...)* and choose the **OrderDetail** Screen.



- d) Set the **OrderId** parameter to:

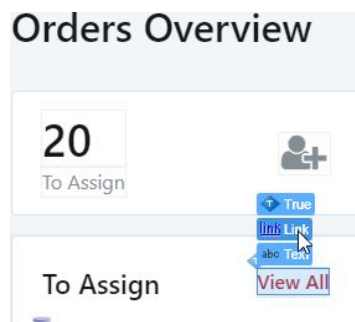
*ToAssignList.List.Current.Order.Id*



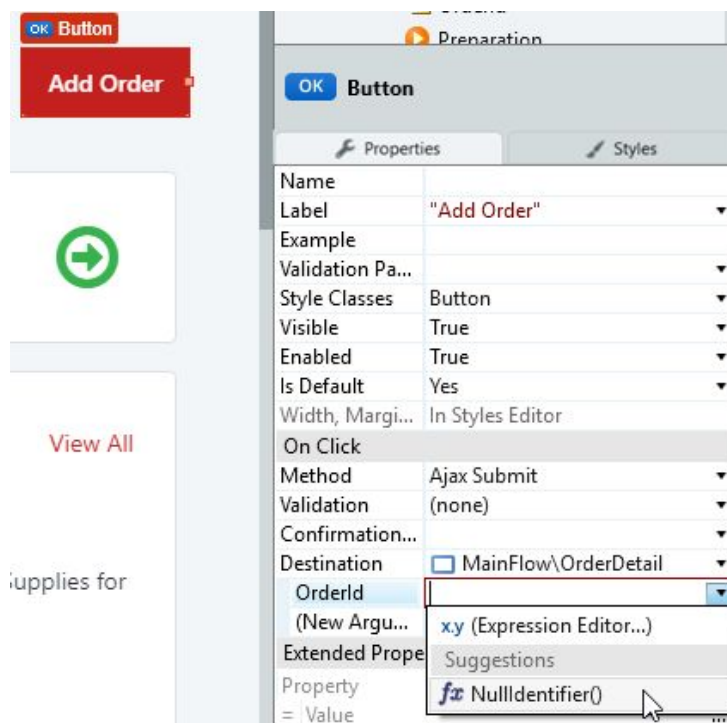
This defines the Input Parameter of the OrderDetail Screen to the **Id** of the Order.

- e) Repeat the above 3 steps for the **Pending** column. On the Id of the Destination, don't forget that this is the **Pending List** and not the To Assign List

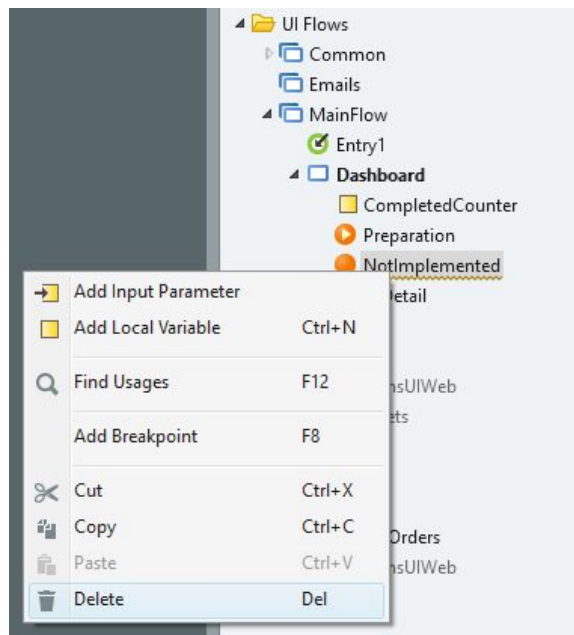
- f) Select the Link that encloses the **View All** text in the **To Assign** column.



- g) In the properties of the Link, set the **Destination** property to the Orders Screen.
- h) Repeat the above 2 steps for the **View All** text in the **Pending** column.
- 2) Update the **Add Order** Button to link to the OrderDetail Screen.
- a) Select the **Add Order** Button in the upper right corner of the Screen.
- b) In the Properties of the Button, set the **Destination** to the OrderDetail Screen and set the **OrderId** parameter to `NullIdentifier()`.



- c) Delete the unused Screen Action **NotImplemented**.

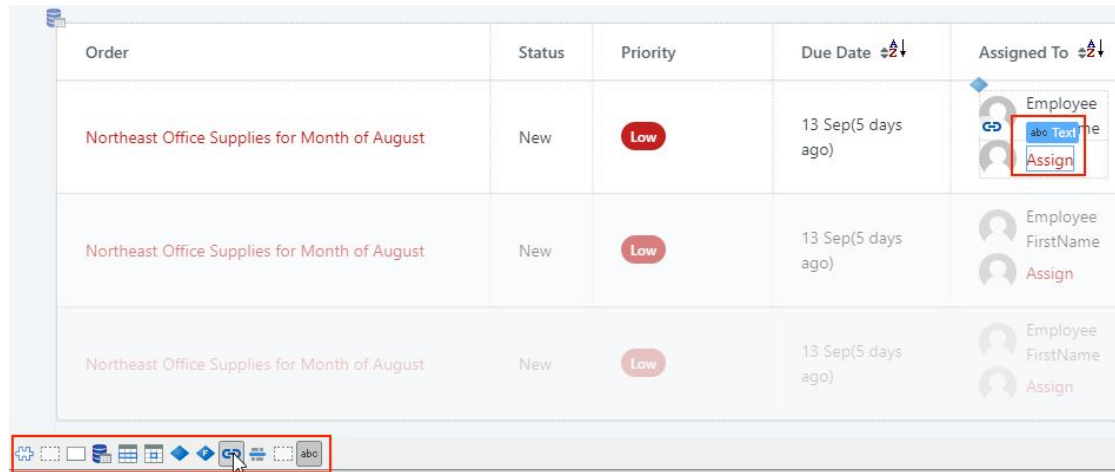


- 3) On the Orders Screen, add a **Link** to the OrderDetail Screen **in every Order** listed on the Table Records.
- a) Open the Orders Screen and right-click on the **Description** in the left column of the Table Records. Choose the **Link to MainFlow\OrderDetail** option to link to the OrderDetail Screen.



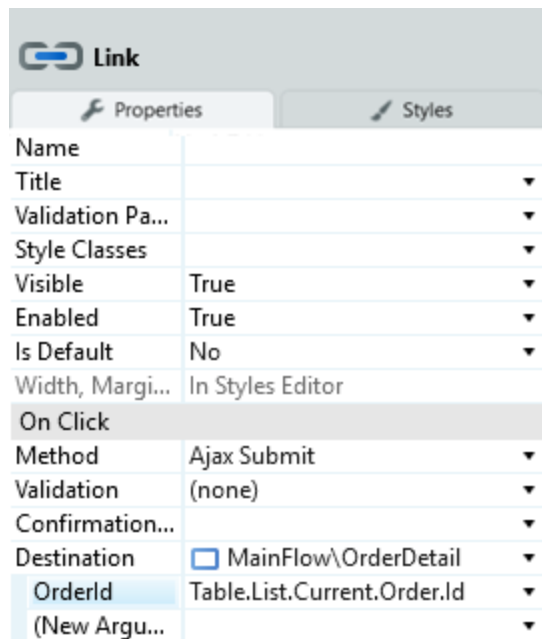
This accelerator encloses the Expression in a Link, sets the **Destination** property to the OrderDetail Screen, and also sets the Screen Input value (*OrderId*) based on the current item of the list.

- b) Go back to the Orders Screen. Select the **Assign** text on the right column of the Table Records, then using the widget breadcrumb on the bottom of the Screen, select the Link enclosing it.



- c) Change the **Destination** property of the Link to the OrderDetail Screen, then set the **OrderId** Input Parameter to:

*Table.List.Current.Order.Id*



- 4) Update the **Add Order** Button to link to the OrderDetail Screen, to add new orders.
- a) Select the **Add Order** Button in the upper right corner of the Orders Screen.

- b) In the Properties area, set the **Destination** to the OrderDetail Screen and the **OrderId** to *NullIdentifier()*.

The screenshot shows the 'Properties' tab of a 'Button' widget in OutSystems. The 'On Click' section is expanded, showing several properties. The 'Destination' and 'OrderId' properties are highlighted with a red box. 'Destination' is set to 'MainFlow\OrderDetail' and 'OrderId' is set to 'NullIdentifier()'.

Button	
Properties	
Name	
Label	"Add Order"
Example	
Validation Pa...	
Style Classes	Button
Visible	True
Enabled	True
Is Default	Yes
Width, Margi...	In Styles Editor
On Click	
Method	Ajax Submit
Validation	Server
Confirmation...	
Destination	<input type="checkbox"/> MainFlow\OrderDetail
OrderId	NullIdentifier()
(New Argu...	

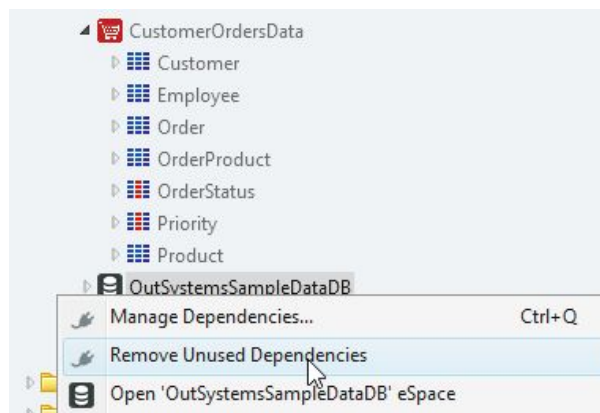
- c) Delete the unused Screen Action **NotImplemented** under the Orders Screen.

## Delete the sample data and test the application

Our application now has three functional Screens, with all of them using the Customer Orders data model. In the next few steps, we will remove the sample data model references and test our application. Removing the references to the Sample Data Model is not mandatory, but recommended since it will ensure that you don't have any reference to the Sample Data model.

Our sample data model has been useful in helping us create Screens very quickly, with the Screen templates. However, we no longer need this sample data, so it is time to delete the references to it.



- 1) Delete the sample data model dependency in the **CustomerOrders** module.
  - a) Switch to the Data layer and locate the **OutSystemsSampleDataDB** Entities.
  - b) Right-click the OutSystemsSampleDataDB and select **Remove Unused Dependencies**.



This Action can also be manually performed using the Manage Dependencies window, although the context menu option can be quicker.

- c) A popup should appear mentioning the 7 removed Entity references.



- 2) The Customer Orders application is now complete. Publish and test the **Customer Orders** application in the browser.
- a) Click the  **1-Click Publish** button to publish your application to the server.
  - b) Click the  **Open in Browser** button to preview your application in the browser.
  - c) The **Customer Orders** application should allow the user to browse to the dashboard, see all customer orders on the Orders Screen, view and update order details, and add new orders to the database.
- 3) At this point, lots of things have changed and we have the first good version of our Customer Orders application. To take it further, complete the following extra challenges.
- a) Fix the Cancel button in the OrderDetail Screen.



- b) Tweak the OrderDetail Screen layout to look similar to the one below

## Edit Order

Description

Roy Lund order for month of September (2018)

Created On

2018-09-15 14:58:14

Status

Pending

Reviewed By

-

Requested By

Roy Lund

Due Date

2018-09-22

Assigned To

Leslie

Priority

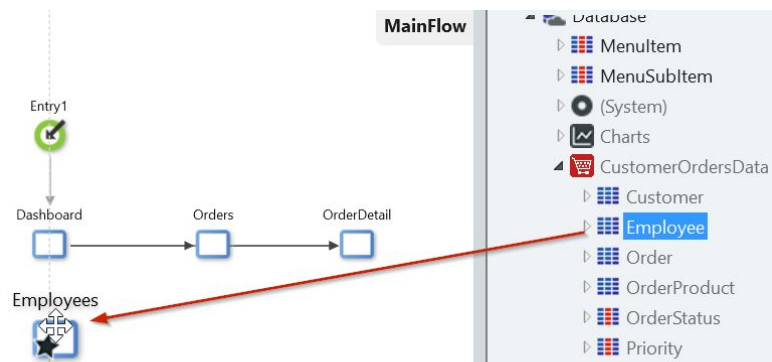
Low Medium High

Cancel Save

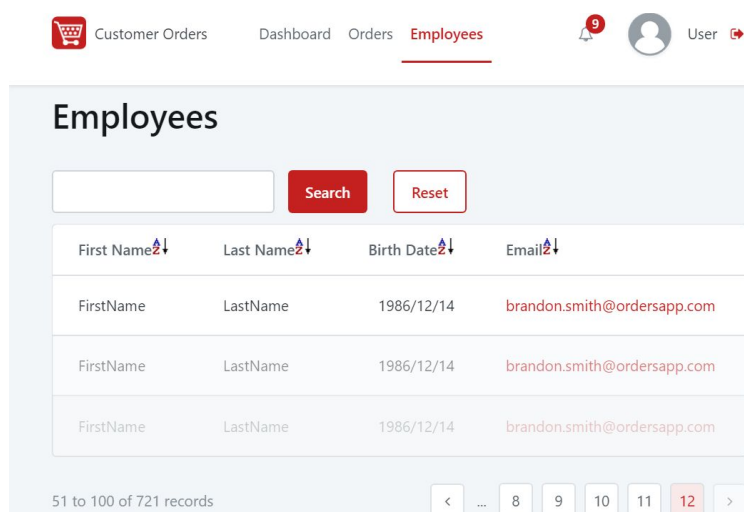
## Scaffolding Patterns

In this part of the exercise, we will create two Screens to manage information about the Employees. However, instead of making use of the new **Screen Templates** released in OutSystems 11, we are going to use some scaffolding patterns. These patterns also enable us to speed-up the development process.

- 1) Create the Employee Screen that list all the employees in the organizations, using a scaffolding pattern. Tick the Anonymous Role on the new Screen.
  - a) In the Interface tab, open the **MainFlow**.
  - b) Switch back to the **Data** tab, to see in the **Entities** folder the **Employee Entity** under the **CustomerOrdersData** module.



- c) Drag and drop the **Employee** Entity to the **MainFlow**. Notice that, some stars appeared, indicating that a Screen was generated automatically.
- d) Open the newly generated **Employee** Screen. You should have a Screen similar to the following screenshot.



---

**NOTE:** Dragging an Entity to the **MainFlow** generates a List Screen for that Entity. This is a Scaffolding pattern.

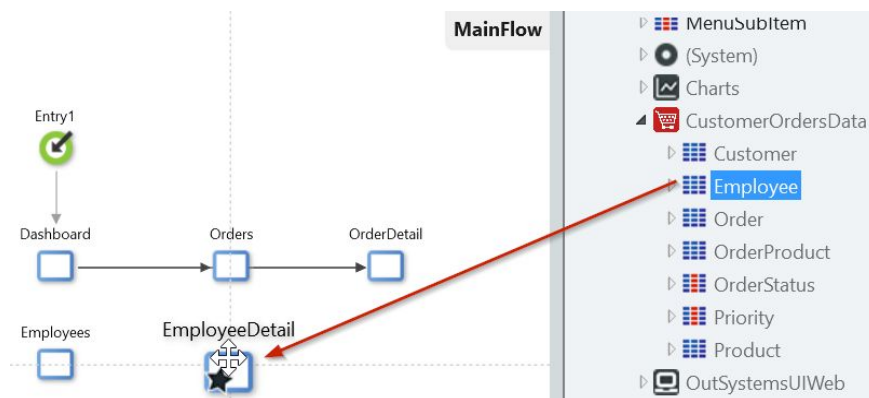
This accelerator generates the UI of the Screen, according to the Theme and CSS Styles used. It creates a Preparation, to get all the records of the dragged Entity, as well as a **Table Records** already set and ready. The logic and UI for search filters are also automatically created.

Some RichWidgets are also automatically added to the Screen, like **List\_SortColumn** and **List\_Navigation**.

Although the interface and logic is automatically generated, it is possible to modify every aspect of the generated elements.

---

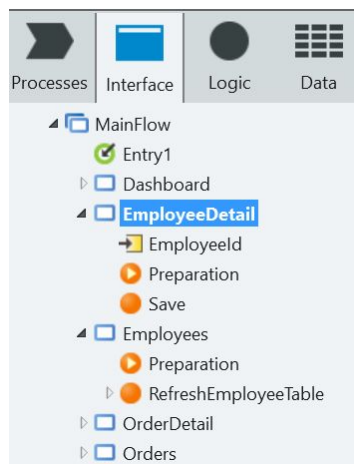
- e) Tick the **Anonymous** Role on the Properties of the Screen.
- 2) Create the **EmployeeDetail** Screen that allows to edit the information of an existing employee or add a new employee, using a scaffolding pattern. Tick the **Anonymous** Role on the new Screen.
  - a) Open the **MainFlow** again and then switch back to the Data tab again.
  - b) Drag and drop again the **Employee** Entity to the **MainFlow**. It will create a **EmployeeDetail** Screen.





- c) Tick the **Anonymous** Role in the properties of the Screen.
- d) Open the **EmployeeDetail** Screen. You should have a Screen similar to the following screenshot.

**NOTE:** By dragging the Entity to the **MainFlow** a second time, it automatically generates a Detail Screen, as the List Screen was already previously created.

- e) Notice that the needed logic and UI was also created for this Screen, as in the **Employees** Screen.



- f) Open the **Employees** Screen. Notice that in the Actions placeholder, there is a new Link to the recently created **EmployeeDetail**, *Create a new Employee*.

- 3) Publish and test the **Customer Orders** application in the browser.
  - a) Click the  **1-Click Publish** button to publish your application to the server.
  - b) Click the  **Open in Browser** button to preview your application in the browser.
  - c) The **Customer Orders** application should now allow list all of the employees in the organization as well as update information of an existing employee or add new employees.

## End of lab

In this exercise lab, we created a new web application to manage orders and its products, as well as the Customers that purchased the orders.

We started with the data model defined in the Customers Orders Data application, so we could focus on developing the UI of the new application. For that, we created 3 Screens using Screen templates, a new functionality in OutSystems 11. These Screen Templates allow creating fully-functional Screens in just a couple of clicks, which can be later adjusted to our needs.

After creating each Screen, we adapted them to our use case of Customer Orders, with focus on the Replace Data operation, where we replaced the sample data from the templates, with the data model in the Customer Orders Data application.

We ended up with an application fully created based on Screen templates and other accelerators.