

# Filtering Orders

## Table of Contents

<b>Outline.....</b>	<b>2</b>
Scenario	2
Orders Screen	2
<b>How-To.....</b>	<b>5</b>
Resources	5
Getting Started	5
Outdated Dependencies	5
Adding a new Search Section	6
Search by Status	12
Create the Status Dropdown	12
Filter the Search by Status	18
Refreshing the Data	20
Filter by Price Range	21
Create the Range Slider Interval	21
Create the Logic for the Range Slider	27
Filter the Orders by the Amount	33
Final Touches: Adding a Label	34
Testing	36
Improve Visuals	36
Hiding the Table	37
UI for "No Orders Available"	39
Creating the Blank Slate	41
Search Criteria Resulted in No Orders	48
<b>Wrapping up.....</b>	<b>51</b>
References	51

# Outline

In this tutorial, you will continue to extend the Order Management application. This time with the creation of new search features, which will allow a user to search for orders by code, status or price range. In this tutorial you will focus on two major steps.

## Add three search inputs to the Orders Screen:

- An input field where users can type an order code;
- A dropdown with possible statuses that the user can select;
- A range slider that can be adjusted to filter orders by price range.

## Design some UI for two new scenarios in the orders Screen:

- When the app has no orders yet;
- When the search filters return no orders.

You will implement those requirements from scratch and understand some important aspects of OutSystems along the way.

## Scenario

At this point, the app has two Screens created in the previous exercise:

- Orders: lists all the orders and their most important information.
- Order Detail: has a Form that allows viewing and editing relevant information of an order, such as date, status, and order code.

In this tutorial you will work exclusively on the Orders Screen.

## Orders Screen

The Orders Screen displays a list of all the orders in a tabular layout, with the code, total amount, date of the order, and order status.

The Screen will have a search area that allows users to filter orders in three ways:

- A text input field to filter the orders by code;
- A dropdown that allows filtering orders by status;

- A range slider that allows filtering orders by their total amount.

OrderManagement Orders Andrea McKenzie

## Order List

[Add Order +](#)

Amount

Code	Total Amount	Order Date	Order Status
123	\$400.00	11 May 2022	<span>Draft</span>
13	\$1,200.00	10 Feb 2022	<span>Approved</span>
23	\$123.00	11 May 2022	<span>Rejected</span>
1234	\$567.00	1 May 2022	<span>Submitted</span>

Additionally, you will also add some UI to display some information and options to the user when no orders appear on the Screen.

When the search criteria result in no orders, the Orders Screen should look like this:

OrderManagement Orders Andrea McKenzie

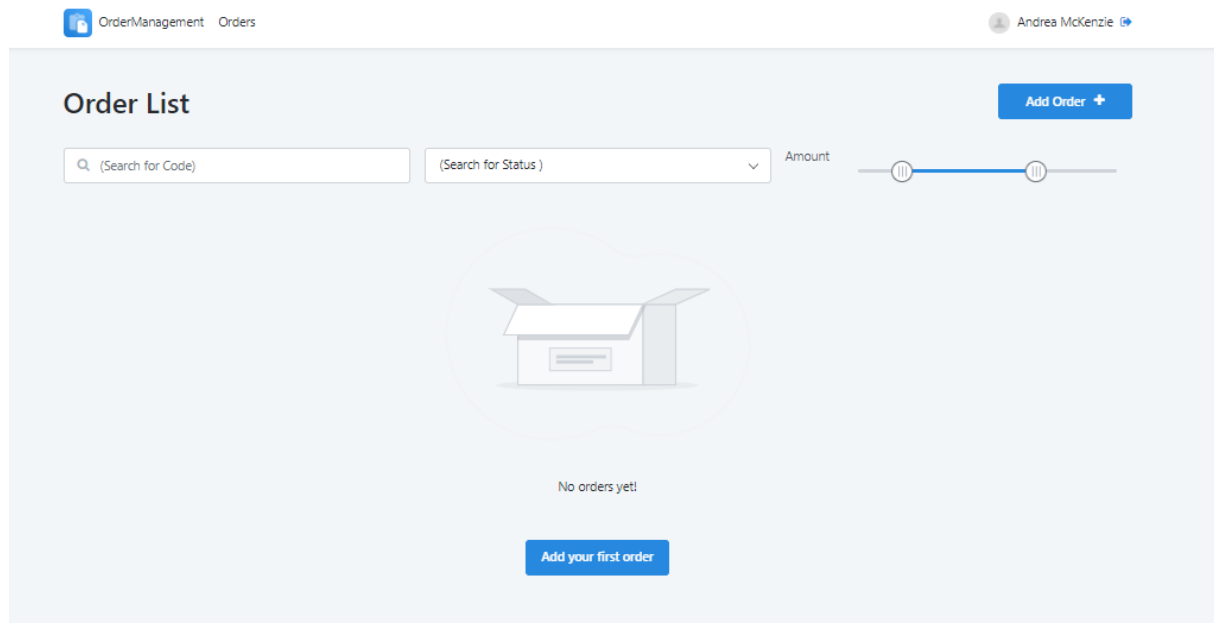
## Order List

[Add Order +](#)

Amount

No results found

Also, when there are no orders yet, the Screen should look like this:



# How-To

Now that you know the scenario, let's continue to expand the Order Management app by following this how-to guide! Are you ready? Let's do it together!

## Resources

This how-to guide has two images that you will need: BlankSlate\_NothingToShow and BlankSlate\_NoResultsFound.

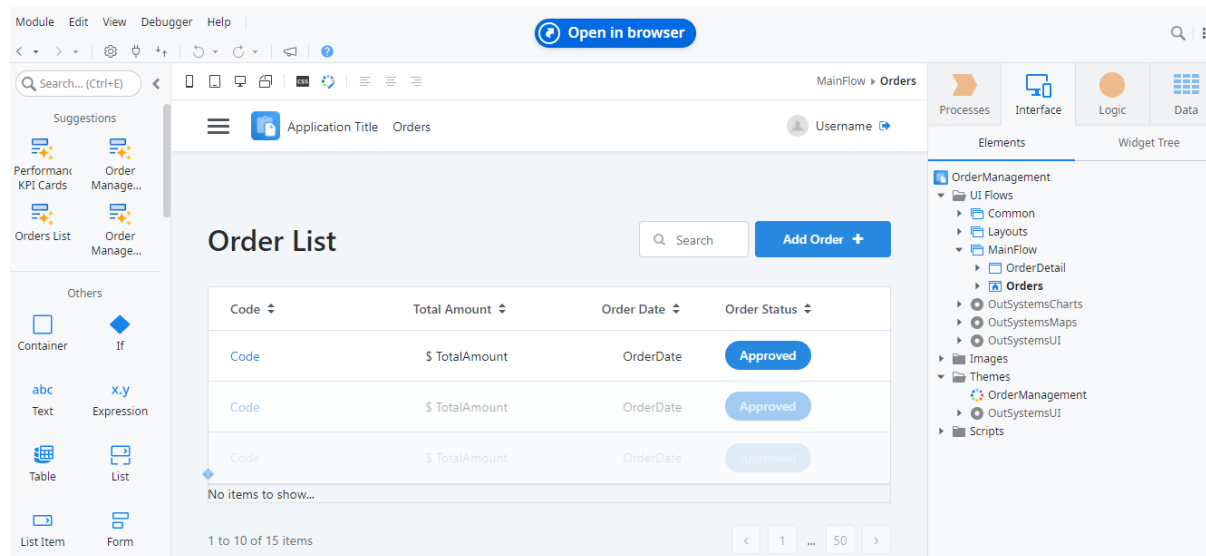
These images will be used to create the UI for when there are no orders visible on the Orders Screen. Don't forget to download them from the **Lesson Materials**.

## Getting Started

In this tutorial, we assume that you have already followed the previous tutorial, and have the two Screens and the data model ready.

If you haven't created it yet, it is important to go back to the previous tutorials and create the application.

To start this exercise, you need the Service Studio with the module OrderManagement opened. You should see the Screen below with the source of our application.

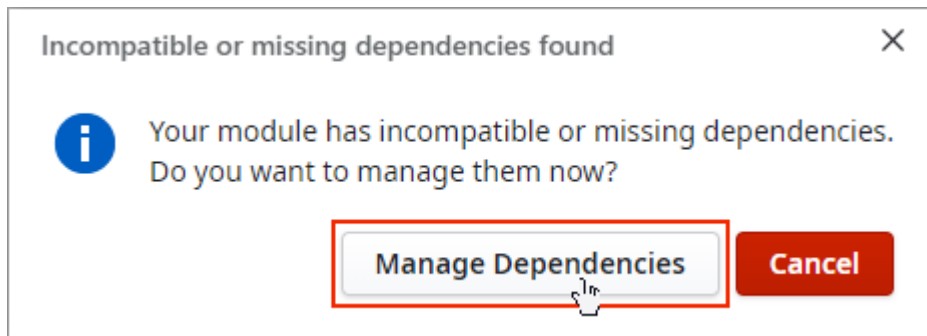


If you don't see it, make sure to open the application and module in Service Studio!

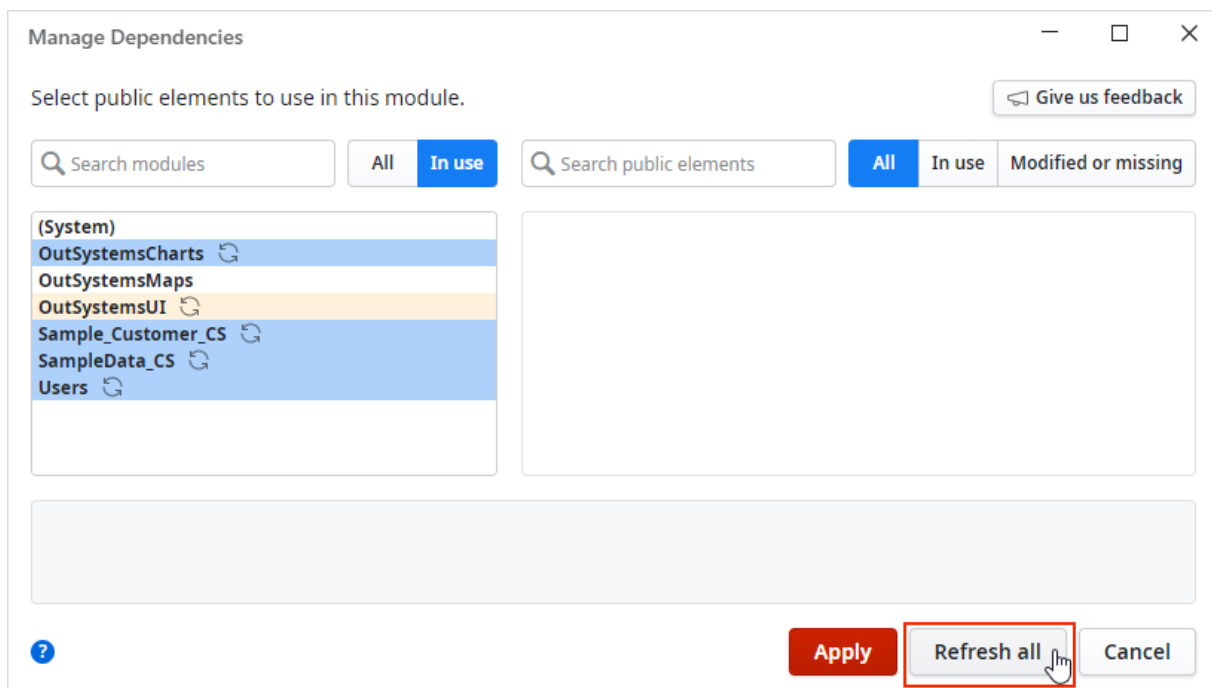
## Outdated Dependencies

You might get a popup message informing that you have outdated dependencies. This is completely normal, since we are always trying to bring a new and updated version of our components!

If that happens, simply click on the button that says *"Manage Dependencies"* to see the outdated components.



Then, click on *"Refresh all"* to update everything at once and *"Apply"* when you are done.



Now publish the module to update the project

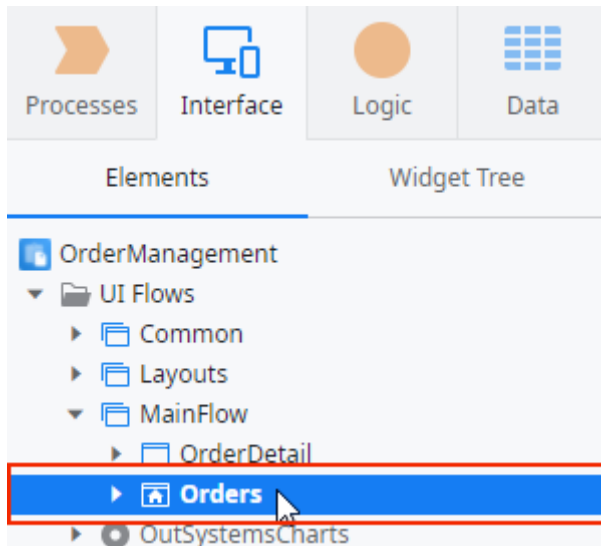


## Adding a new Search Section

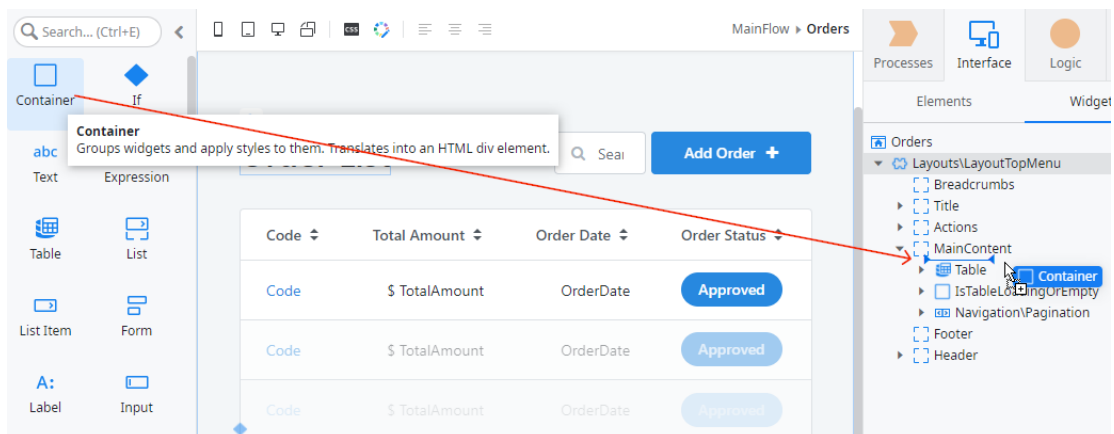
In the previous tutorial, you created the Orders Screen using accelerators. This Screen was already created with a search functionality that allows searching orders by code. Now you will add other search functionalities, so end users can search orders by status and price range as well.

Let's start by creating a search area for our filters and place the one that exists on that new area.

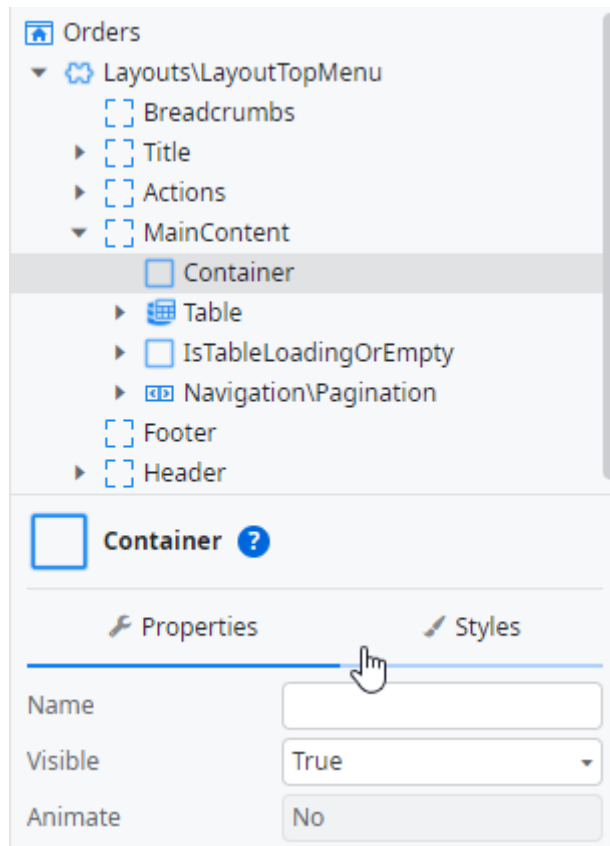
- 1) Under the Interface tab, open the **Orders** Screen by double-clicking on it (if it's not already open).



- 2) Drag a new **Container** from the Toolbox and drop it right above the Orders table. You can use the Widget Tree to guide you.

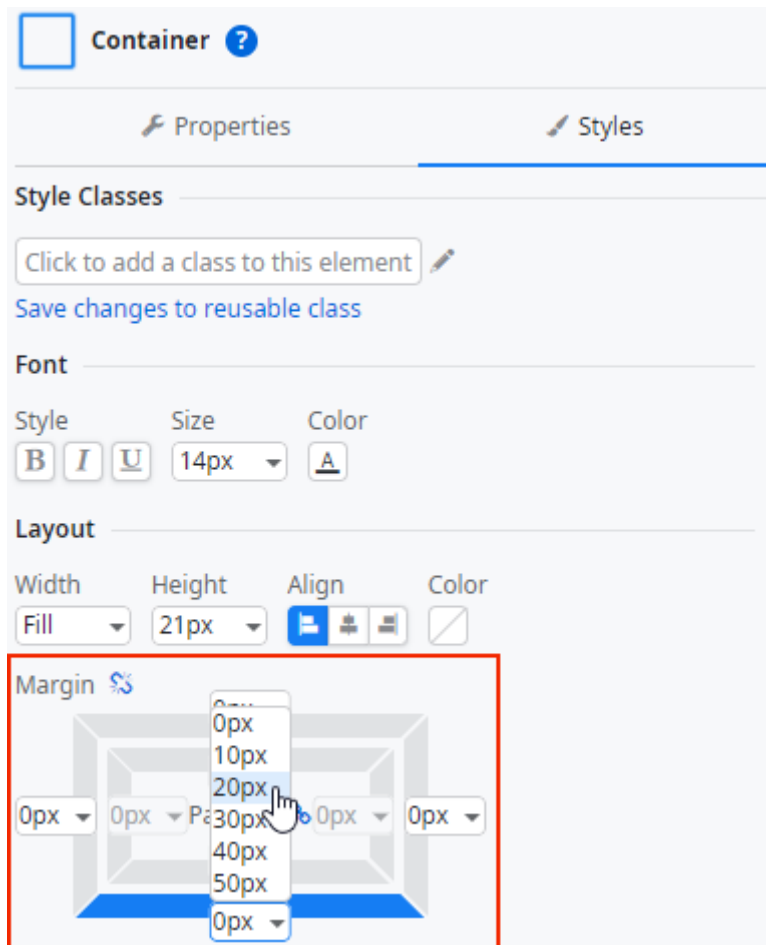


- 3) On the right sidebar, you will be able to see the Container's properties. Switch to the Styles tab by clicking on it.

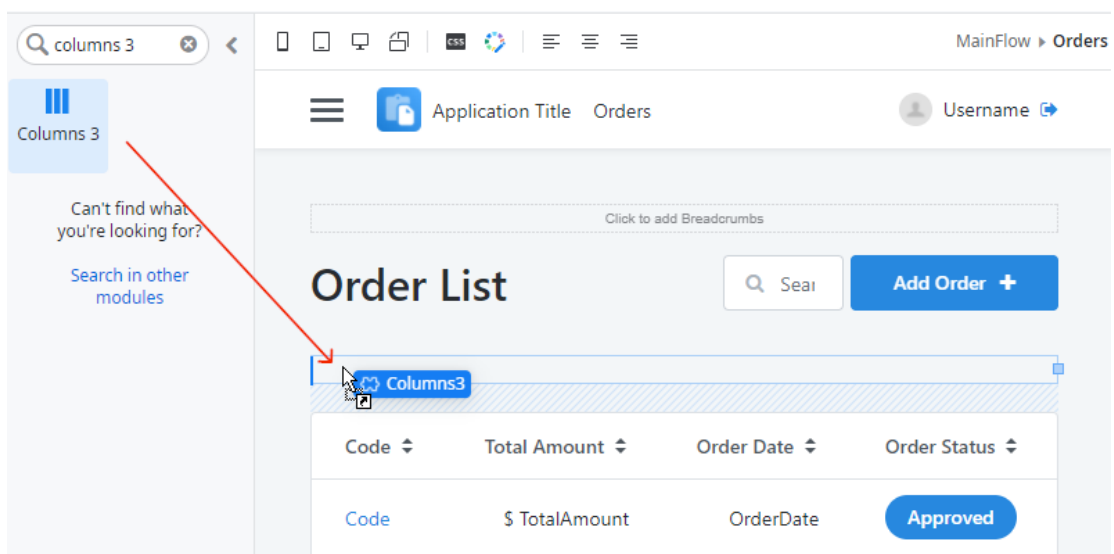




- 4) In the Layout area, add a *20px* margin to the bottom. This will create some space between this Container and the Table underneath it.

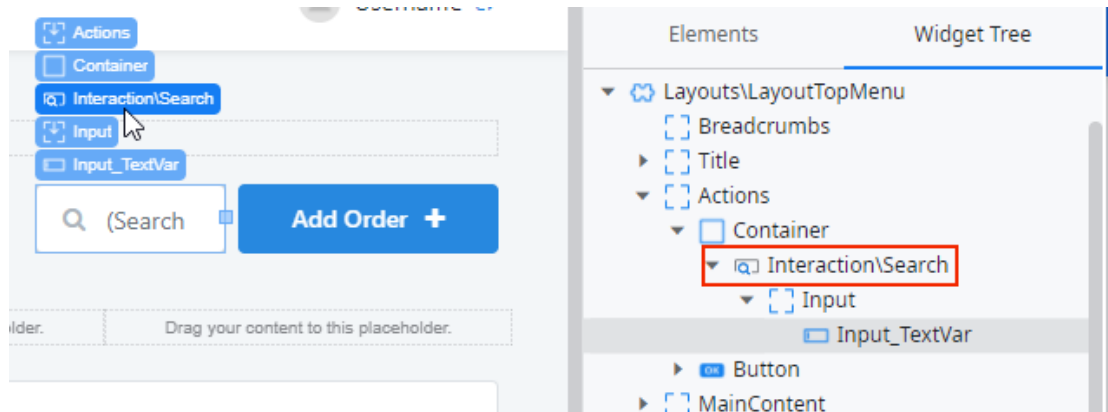


- 5) Going back to the Orders Screen, drag a **Columns 3** Widget from the Toolbox and drop it inside the Container you just created.

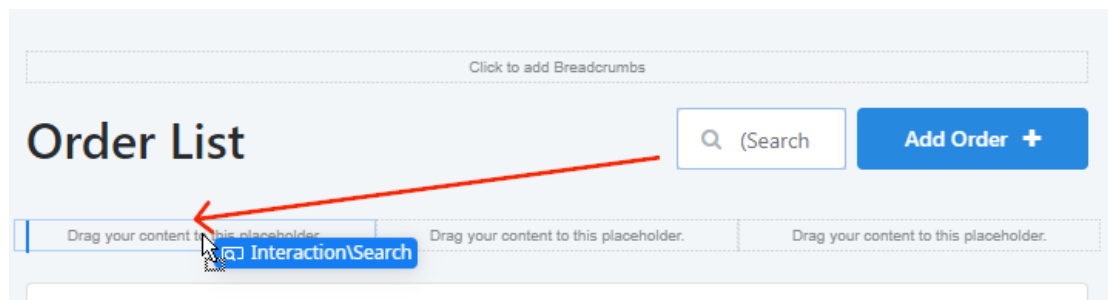


Now you have a section on your Screen that is already split in three columns.

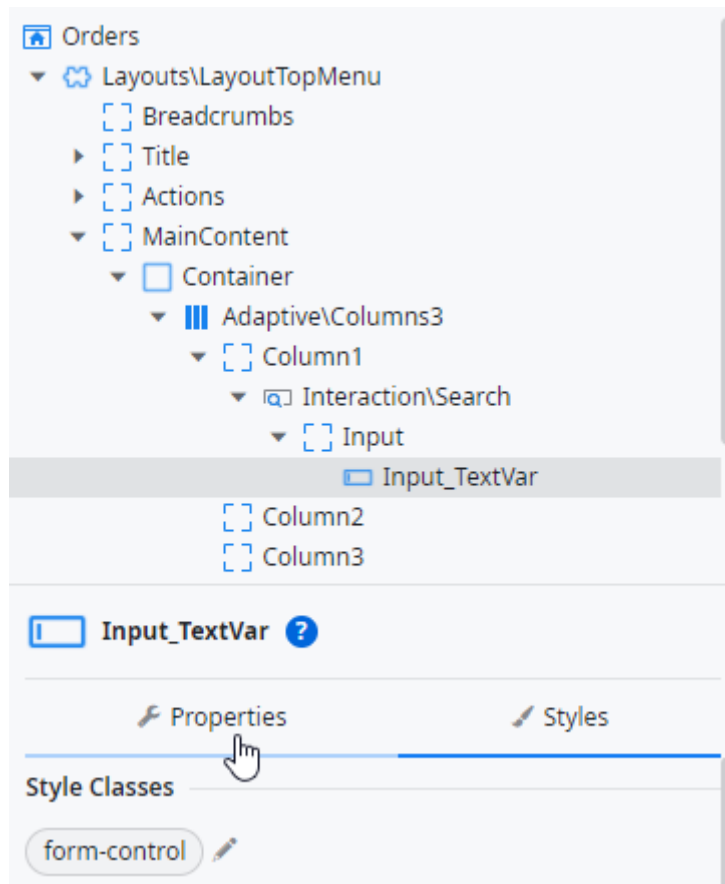
- 6) Select the **Search** input that exists on the Screen. Make sure you're selecting the Interaction\Search element.



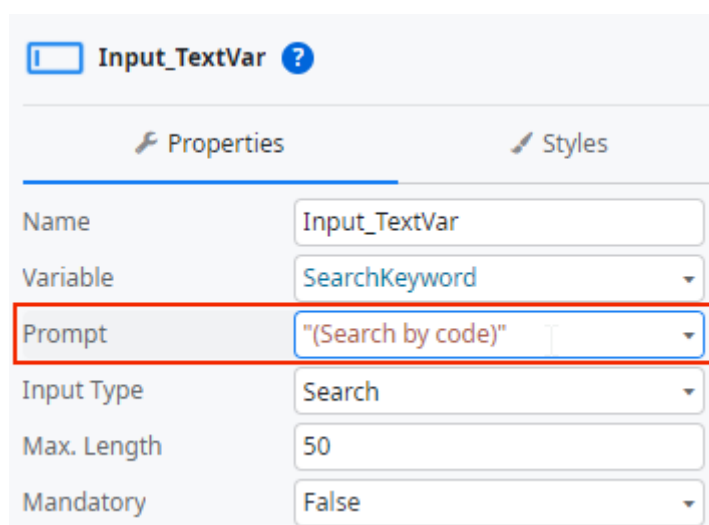
- 7) Drag the element and drop it on the first column.



- 8) Click on the Search input you just moved and look to the right sidebar. Switch to the Properties tab to see the properties of the input.



- 9) Type "(Search by code)" in the Change the **Prompt** property. This is what users will see when they have not typed anything in the search input.



10) Publish the application and open it in the browser.



11) Try to search your orders by code and confirm all is working fine.

Order List

Add Order +

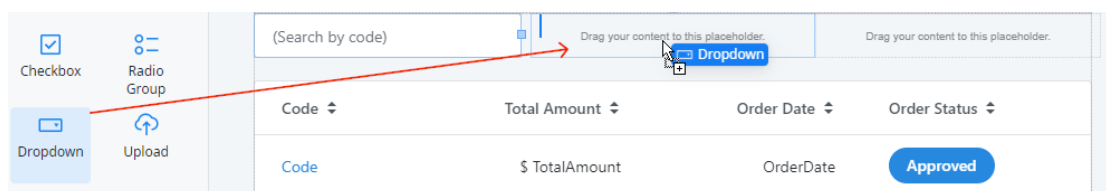
Code ↕	Total Amount ↕	Order Date ↕	Order Status ↕
23	\$123.00	11 May 2022	Rejected
123	\$23.00	11 May 2022	Draft

## Search by Status

Now you will add the search by status using a dropdown that displays all possible status. If a user selects a particular status, then only orders with that status should appear on the table. Let's start by implementing the dropdown!

## Create the Status Dropdown

- 1) Drag a **Dropdown** from the Toolbox (left sidebar) and drop it to the second column.



The Dropdown's properties will contain a few errors, but don't worry! We will fix them in the next steps.

Dropdown1 ?	
Properties	Styles
Name	Dropdown1
Variable	
List	
Options Content	Text Only
Options Text	
Options Value	
Mandatory	False
Enabled	True
Empty Text	
Style Classes	"dropdown"

- 2) The first error is in the **Variable** property. This property indicates where the option selected by the user will be saved. Click on the dropdown in the **Variable** property and select *New Local Variable*.

Dropdown1 ?	
Properties	Styles
Name	Dropdown1
Variable	<div> <div>Select Variable...</div> <div> <b>Suggestions</b> <ul style="list-style-type: none"> <li>SearchKeyword</li> <li>StartIndex</li> <li>MaxRecords</li> <li>TableSort</li> <li><b>New Local Variable</b></li> </ul> </div> </div>
List	
Options Content	
Options Text	
Options Value	
Mandatory	
Enabled	

- 3) Type *SearchedStatus* in the **Name** property of the Variable.

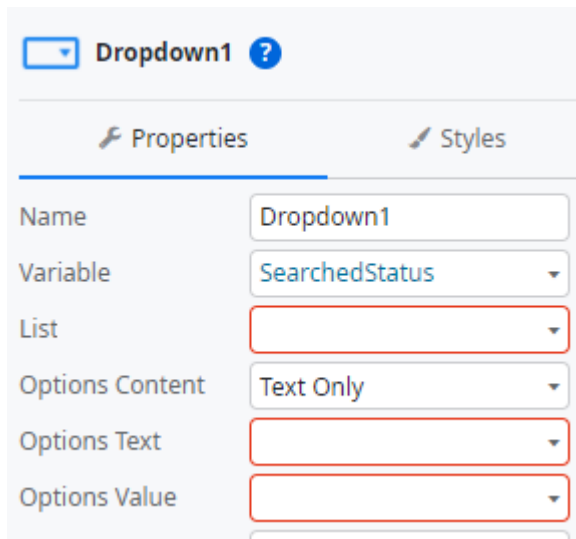
- 4) Change the **Data Type** to **OrderStatus Identifier**.

The screenshot shows the 'SearchedStatus' Local Variable editor. The 'Name' field is 'SearchedStatus'. The 'Data Type' dropdown is currently set to 'Text'. A list of available data types is shown below, with 'OrderStatus Identifier' highlighted by a mouse cursor. The list includes: MenuAction Identifier, MessageStatus Identifier, Order Identifier, OrderItem Identifier, OrderStatus Identifier, and Orientation Identifier. Metadata on the left indicates it was created by angelita.t and last modified by ang.

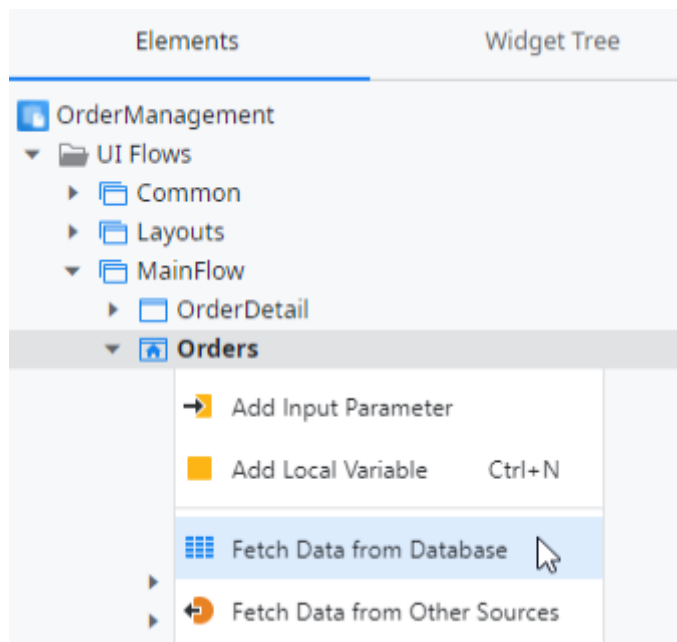
<b>SearchedStatus</b> ? Local Variable	
Name	SearchedStatus
Description	
Data Type	Text
Default Value	
Created by angelita.t	
Last modified by ang	
	MenuAction Identifier
	MessageStatus Identifier
	Order Identifier
	OrderItem Identifier
	OrderStatus Identifier
	Orientation Identifier

We are creating a Local Variable that will exist exclusively on the Orders Screen. The data type is set to OrderStatus Identifier, since the Variable will hold the status selected by the user: draft, submitted, approved, or rejected.

One down, three more to go! Select the Dropdown and see that the next error we need to fix is the **List** property. This property expects the list of elements that will appear in the dropdown and that the user will be able to select.

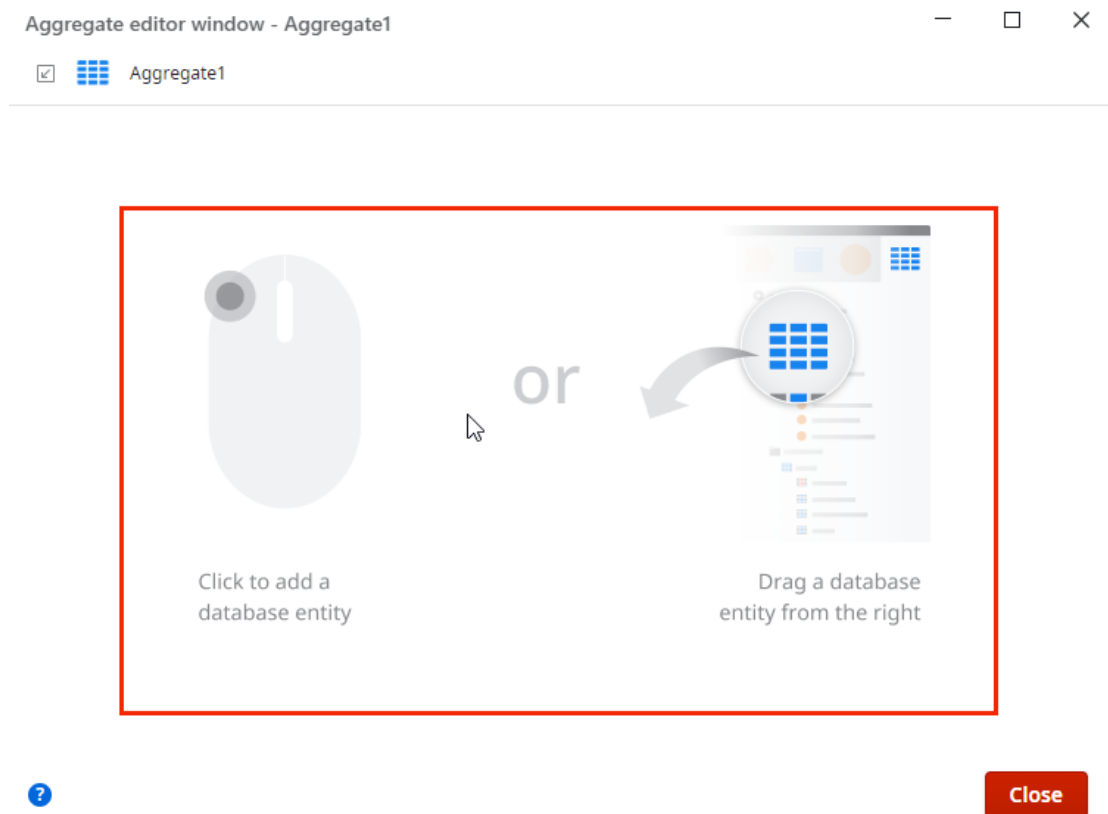


- 1) Right-click the Orders Screen and select **Fetch Data from Database**.

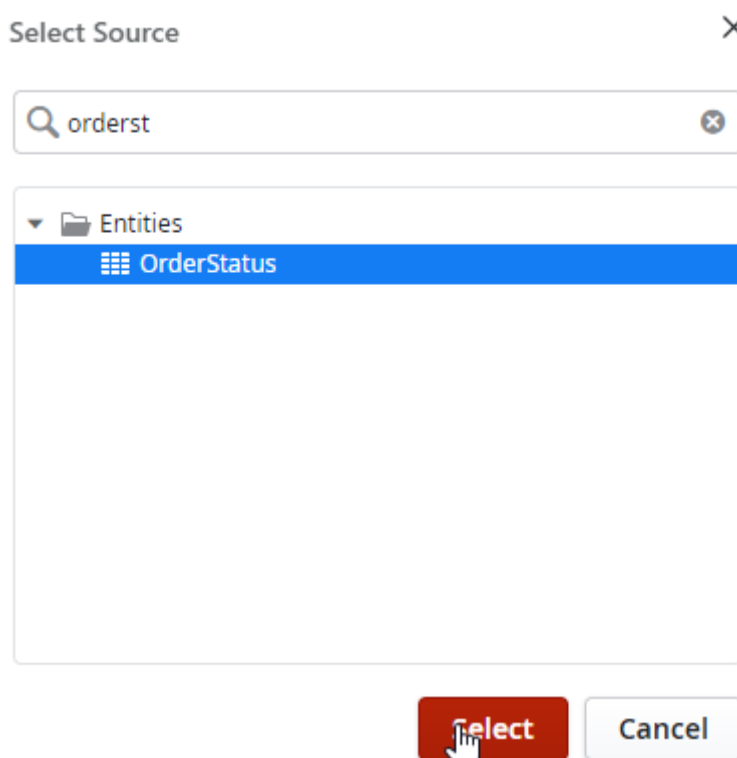


You are creating an Aggregate, which is what OutSystems uses to allow you to fetch data from the database. To read more about Aggregates don't forget to check our [documentation](#) and [courses](#).

2) In the new dialog that appears, click anywhere on the highlighted area...



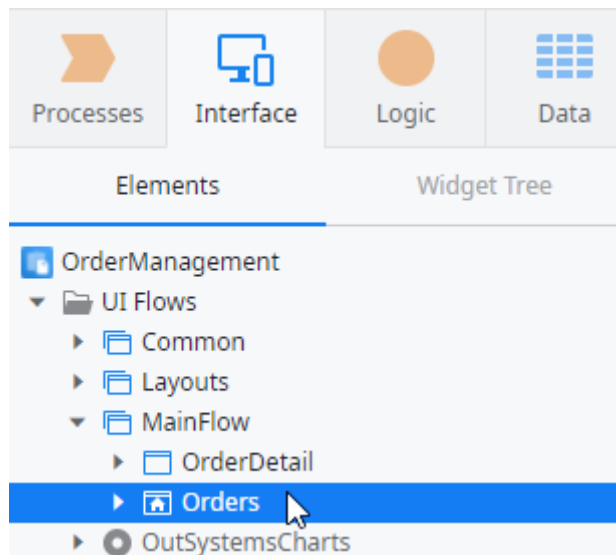
3) ... and select the **OrderStatus** Entity.



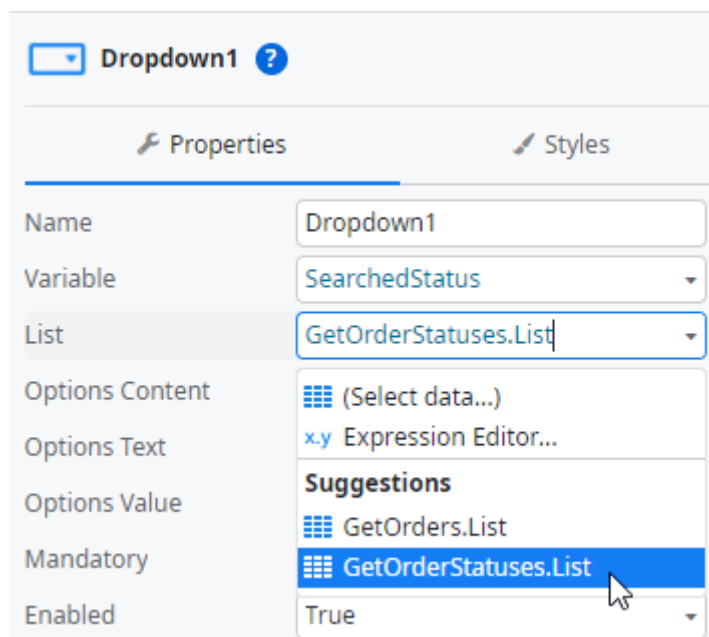


Notice that the Aggregate's name automatically changes to *GetOrderStatuses*. OutSystems tries to name elements in the most intuitive way. This Aggregate will fetch all the statuses of an order available in the database.

- 4) Go back to the Orders Screen by double clicking on its name in the Interface Tab.

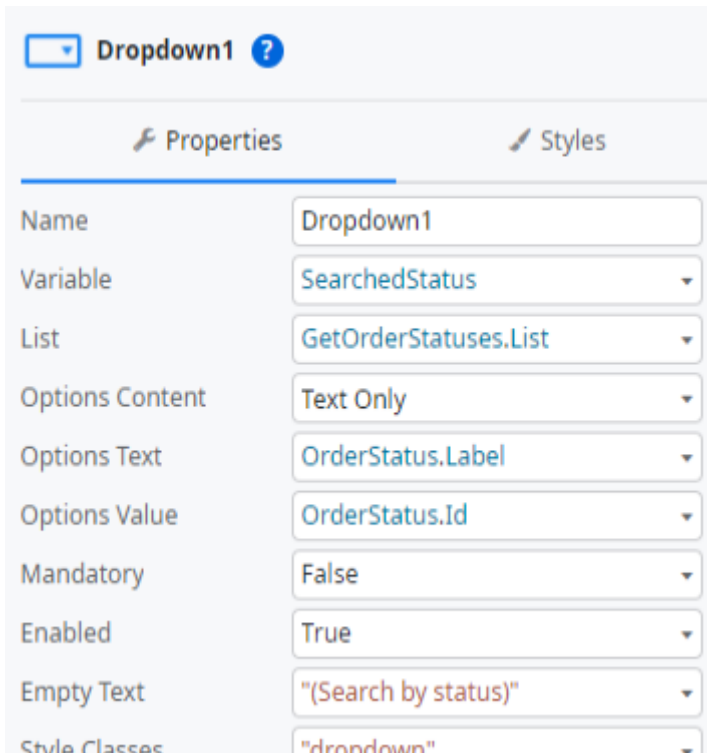


- 5) Open the Dropdown Widget's properties, expand the **List** dropdown, and select **GetOrderStatuses.List** from the suggestions, to tie the property to the output of the new Aggregate we just created.



The remaining properties were filled automatically and we have no more errors. Awesome!

- 6) Let's just change the **Empty Text** property by typing "(Search by Status)".



The screenshot shows the 'Properties' tab of a widget named 'Dropdown1'. The properties are as follows:

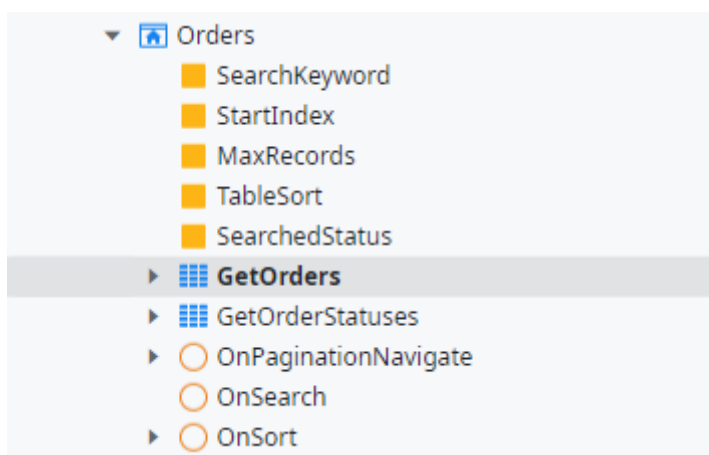
Property	Value
Name	Dropdown1
Variable	SearchedStatus
List	GetOrderStatuses.List
Options Content	Text Only
Options Text	OrderStatus.Label
Options Value	OrderStatus.Id
Mandatory	False
Enabled	True
Empty Text	"(Search by status)"
Style Classes	"dropdown"

This property is similar to the prompt we edited for the "Search by code" input field. This is the text the user will see before selecting an option.

## Filter the Search by Status

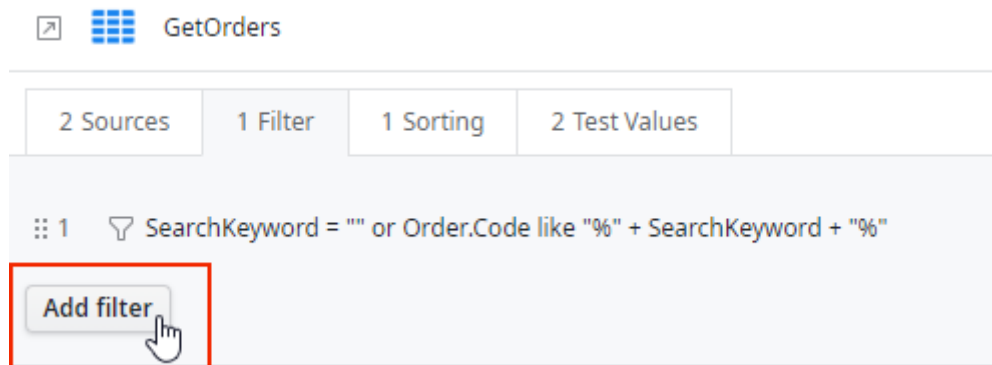
The Dropdown is fully implemented now, but how will it influence the list of orders? We will need to make sure the option selected by the user will have an impact on the orders that appear on the table.

- 1) In the Interface tab, expand the Orders Screen and double-click on the Aggregate **GetOrders** to open it.



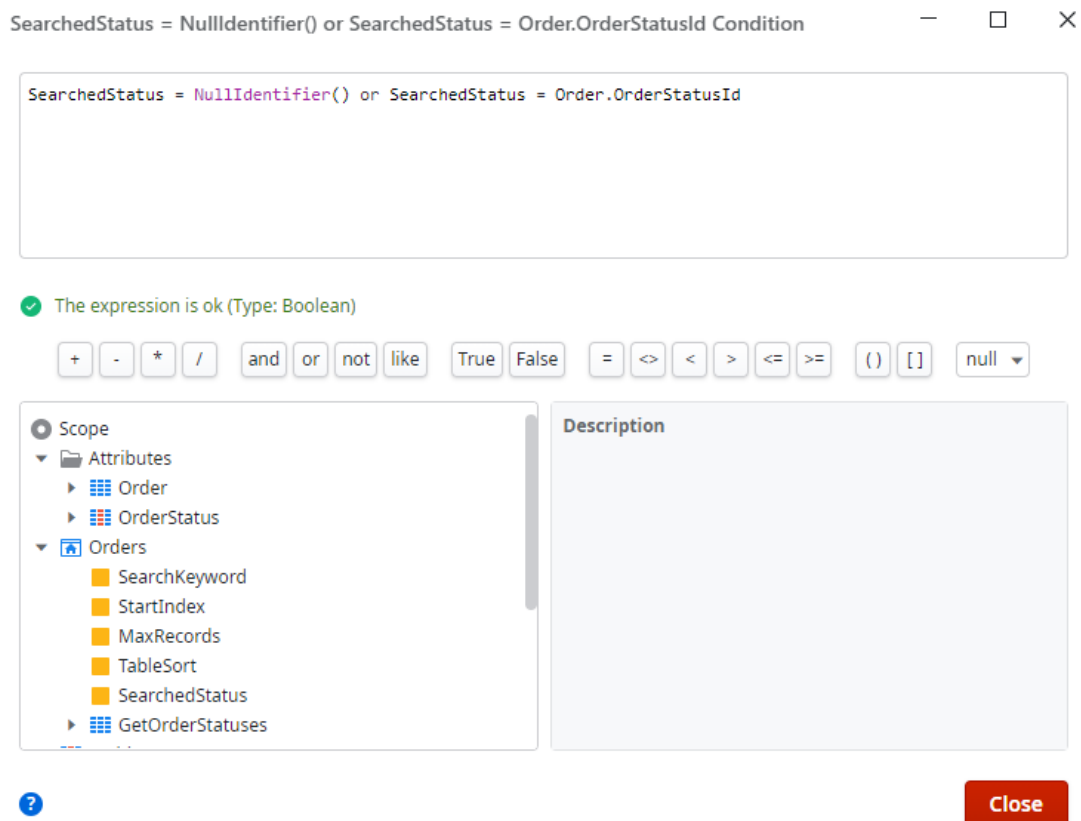
This is the Aggregate that fetches all the orders that are displayed in Screen.

- 2) Switch to the **Filter** tab. You can see that there is a filter already created that is related to the search by code. Now we need to add a new filter for our using the dropdown. Click on the **Add filter** button.



- 3) Add the following expression in the new dialog and click **Close** when you're done.

```
SearchedStatus = NullIdentifier() or SearchedStatus = Order.OrderStatusId
```

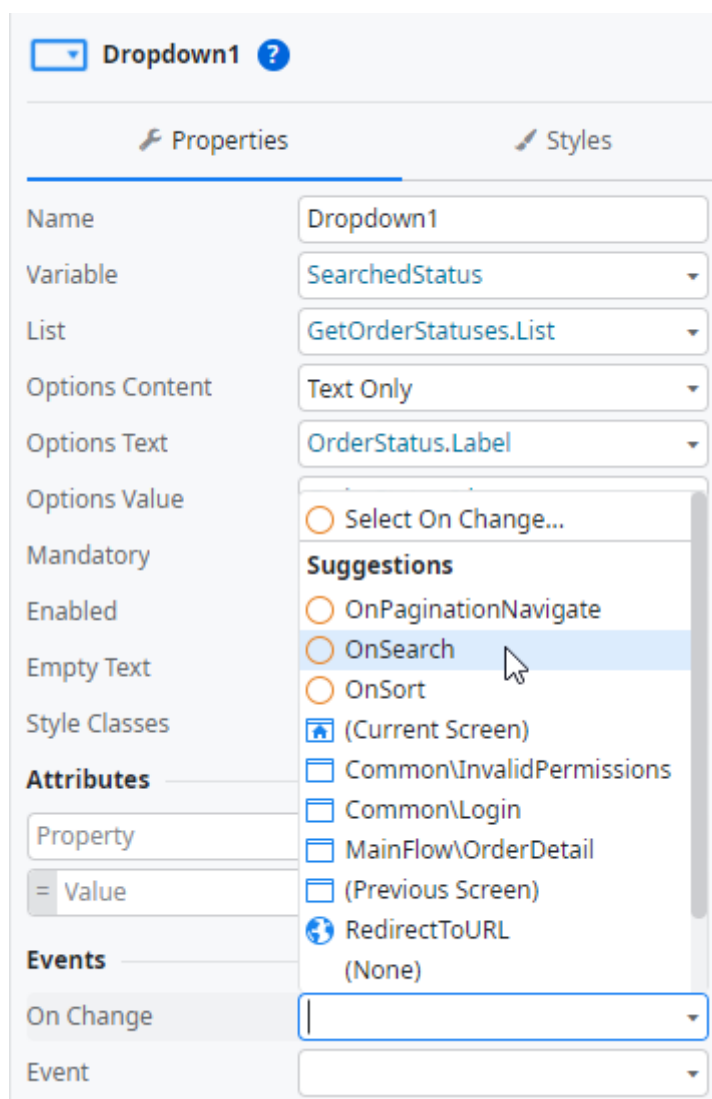


As you can see, we're using the SearchedStatus variable to filter the orders, which is where the option chosen by the user is saved.

## Refreshing the Data

There's one last thing that we need to take care at this stage. When the user selects the status in the Dropdown, the list of orders need to be automatically updated. But, remember that you already tried a search by code at this point? So, there's some good news! The logic is already here, we just need to use it!

- 1) Let's go back to the Orders Screen and click on the Dropdown again.
- 2) Scroll down the properties area until you see the **On Change** property. Expand its dropdown and select the **OnSearch** Action.

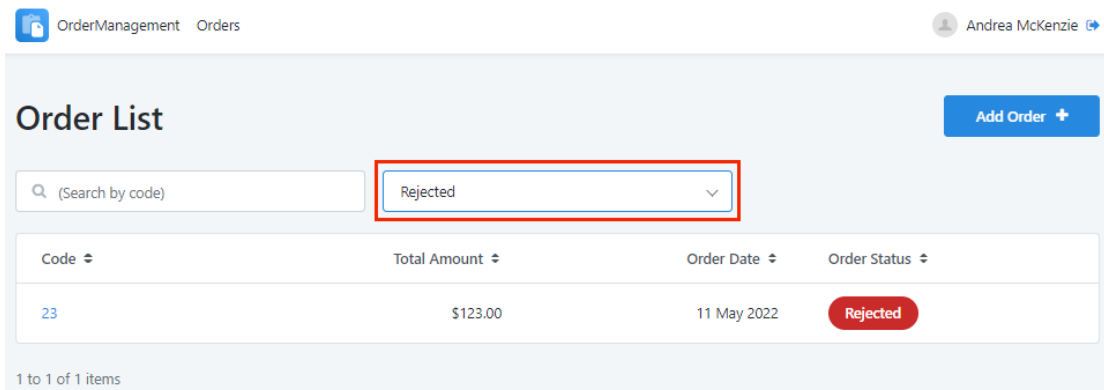


The On Change property will be automatically executed when the user selects an option in the Dropdown. The OnSearch Action is where the logic that we need lives! Let's try this out!

- 3) Publish the application and open it in the browser.



- 4) Try to search your orders by status using the dropdown and see if it works!

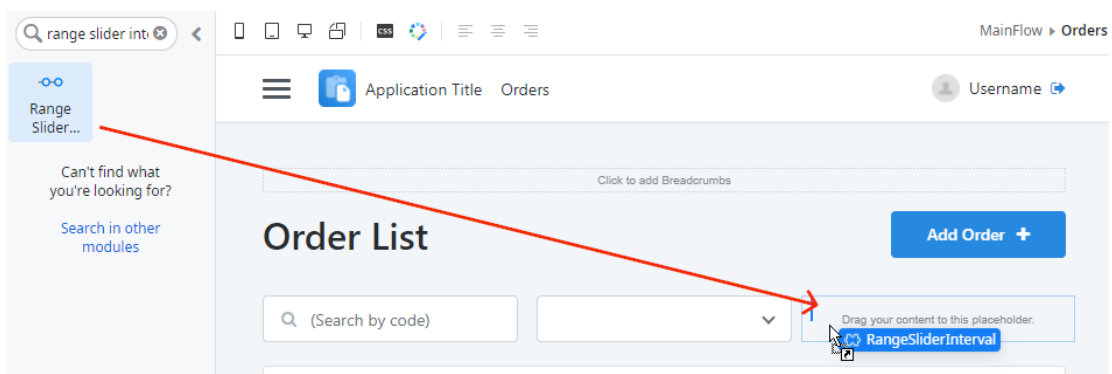


## Filter by Price Range

Now there's the price range left. You will use a Widget called Range Slider Interval.

## Create the Range Slider Interval

- 1) Drag a **Range Slider Interval** Widget from the Toolbox and drop it to the third column.



At this point you will see some errors, but you know the drill! We will fix them!

**Interaction\RangeSliderInterval** ?

**Properties** **Styles**

Name

Source Block

MinValue

MaxValue

StartingValueFrom

StartingValueTo

Orientation

Size

OptionalConfigs

ExtendedClass

**Events**

Event

Handler

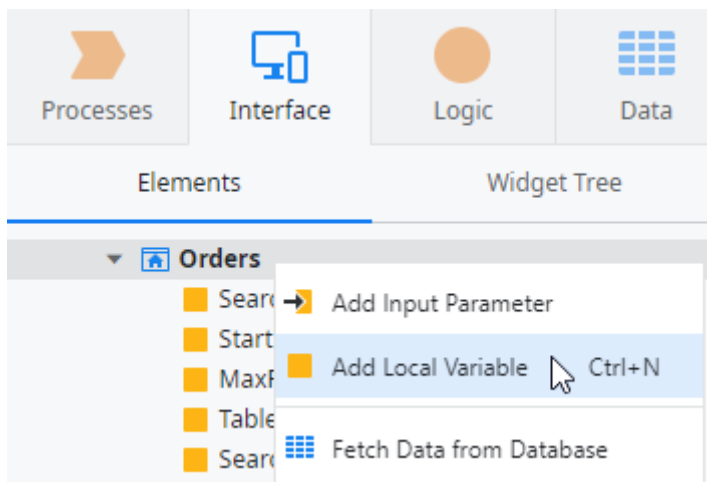
Event

Handler

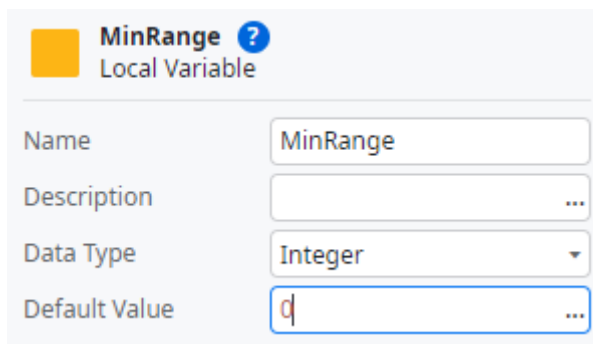
These properties represent the minimum and maximum allowed values for the slider (MinValue and MaxValue) and the min and max boundaries of the interval selected by the user in the slider (StartingValueFrom and StartingValueTo).

Remember what you did for the Dropdown where we needed a Local Variable to help you out? Well, now we need two for the same reason!

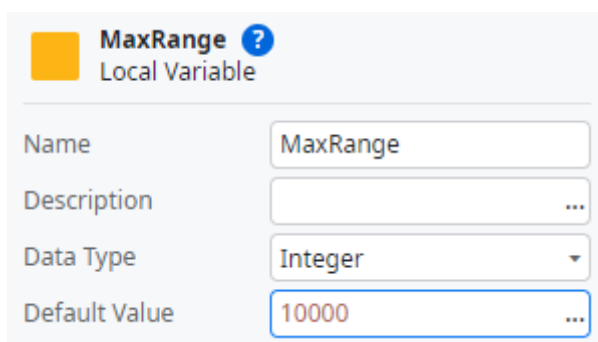
- 2) Switch from the Widget Tree to the Elements section, right-click the Orders Screen and select **Add Local Variable**.



- 3) Name the variable *MinRange* and set the **Default Value** to 0. Make sure the **Data Type** is set to **Integer**.



- 4) Repeat the previous 2 steps and create another Local Variable called *MaxRange* with **Default Value** set to 10000.



We will use fixed intervals for the slider. We could extend the logic as well to make it dynamic.

- 5) Click on the Range Slider Interval, to see the properties editor. Click on the **MinValue** property and select the Local Variable *MinRange* you just created. Do the same for the **StartingValueFrom** property.
- 6) Then, select the *MaxRange* Local Variable in the **MaxValue** and **StartingValueTo** properties.

The screenshot shows the 'Interaction\RangeSliderInterval' properties editor. It has two tabs: 'Properties' (selected) and 'Styles'. The 'Properties' tab contains a list of properties with corresponding dropdown menus. The 'Name' property is empty. The 'Source Block' property is set to 'Interaction\RangeSliderInterval'. The 'MinValue' property is set to 'MinRange'. The 'MaxValue' property is set to 'MaxRange'. The 'StartingValueFrom' property is set to 'MinRange'. The 'StartingValueTo' property is set to 'MaxRange'. The 'Orientation' property is empty. The 'Size' property is empty. The 'OptionalConfigs' property is empty. The 'ExtendedClass' property is empty.

Interaction\RangeSliderInterval ?	
Properties	Styles
Name	<input type="text"/>
Source Block	Interaction\RangeSliderInterval
MinValue	MinRange
MaxValue	MaxRange
StartingValueFrom	MinRange
StartingValueTo	MaxRange
Orientation	
Size	
OptionalConfigs	
ExtendedClass	

This step initializes the minimum amount and maximum amount the user can select in the slider, and sets the interval selected by the user (the actual sliders) to the minimum and maximum range allowed.



- 7) We still have an error. We need to set the **Handler** property. This property will allow triggering some logic, when the slider is used and the interval is changed by the user. Click on the **Handler** property and select **New Client Action**.

The screenshot shows the OutSystems Properties Panel for the **Interaction\RangeSliderInterval** widget. The panel is divided into **Properties** and **Styles** tabs. The **Properties** tab is active, showing various configuration options. The **Events** section is expanded, displaying a list of suggestions for the **Handler** property. The suggestions include **OnPaginationNavigate**, **OnSearch**, **OnSort**, **RangeSliderIntervalOnValueChange**, and **New Client Action**. The **New Client Action** option is highlighted by the mouse cursor. Below the suggestions, there is a red-bordered input field for the **Handler** property.

Property	Value
Name	
Source Block	Interaction\RangeSliderInterval
MinValue	MinRange
MaxValue	MaxRange
StartingValueFrom	MinRange
StartingValueTo	MaxRange
Orientation	
Size	
OptionalConfigs	Select Handler...
ExtendedClass	
Events	
Event	
Handler	
Event	
Handler	

This Action will define the behavior we want when the interval changes. For now, it is empty, but let's adjust it.

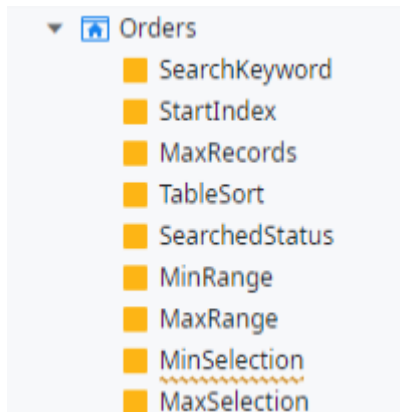
MainFlow ▶ Orders ▶ RangeSliderIntervalOnValueChange



## Create the Logic for the Range Slider

To support the logic we need two extra Local Variables to save the interval selected by the user. Think about the SearchedStatus we created before. The idea is pretty much the same.

- 1) Create two new Local Variables on the Orders Screen, called *MinSelection* and *MaxSelection*, both **Integers**.

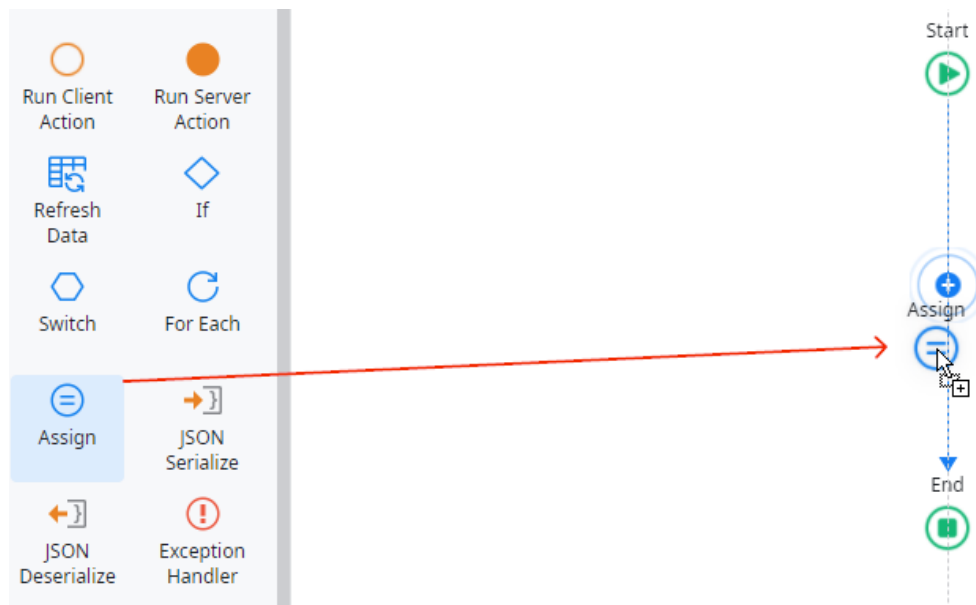


- 2) Set the Default Values of the Local Variables as 0 and 10000.

MinSelection ? Local Variable		MaxSelection ? Local Variable	
Name	MinSelection	Name	MaxSelection
Description	...	Description	...
Data Type	Integer	Data Type	Integer
Default Value	0	Default Value	10000

Now you have two variables to control the selection (the new ones) and two variables to set the minimum and maximum values that can be selected (the variables created in the previous steps).

- 3) Let's go back to the Action. Drag an **Assign** element from the Toolbox and drop it on the Action flow.



The Assign element is used to assign values to variables. In this case, we want the Min Selection value to change according to the MinRange, and the MaxSelection according to the MaxRange.

- 4) Now we need to create the assignments, where we update the MinSelection and MaxSelection variables with the values selected by the user. If you expand

the Action that we're working on, on the right sidebar, you notice two input parameters.

Orders ▶ RangeSliderIntervalOnValueChanged

Start



Assign



End



Processes

Interface

Logic

Data

Elements

Widget Tree

TableSort

SearchedStatus

MinRange

MaxRange

MinSelection

MaxSelection

GetOrders

GetOrderStatuses

OnPaginationNavigate

OnSearch

OnSort

**RangeSliderIntervalOnValueChanged**

IntervalStart

IntervalEnd

OutSystemsCharts

OutSystemsMaps

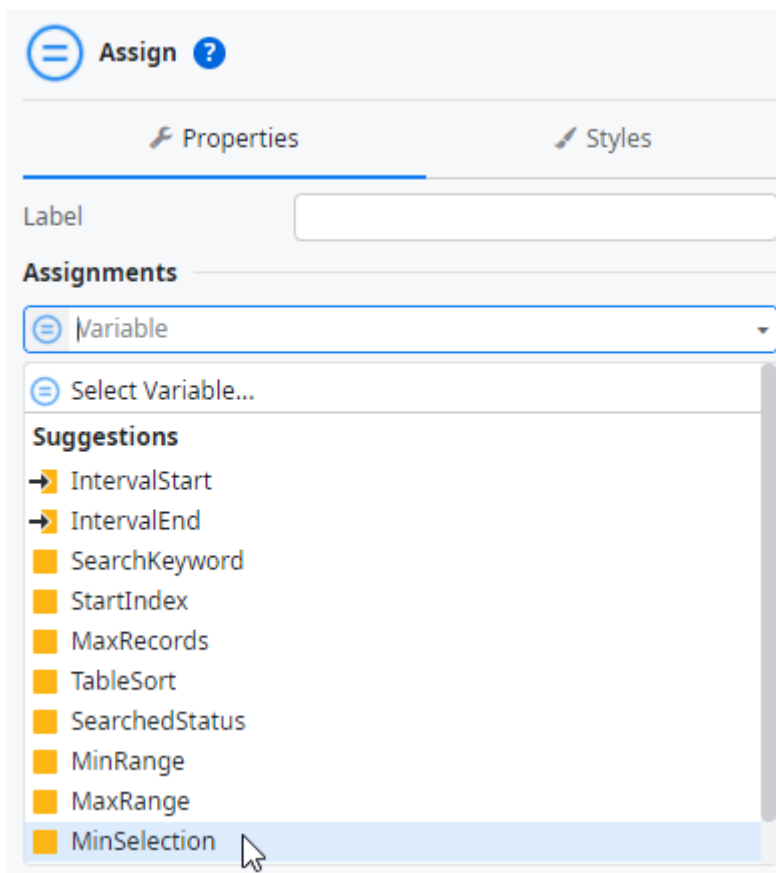
OutSystemsUI

RangeSliderIntervalOnValueChanged ?

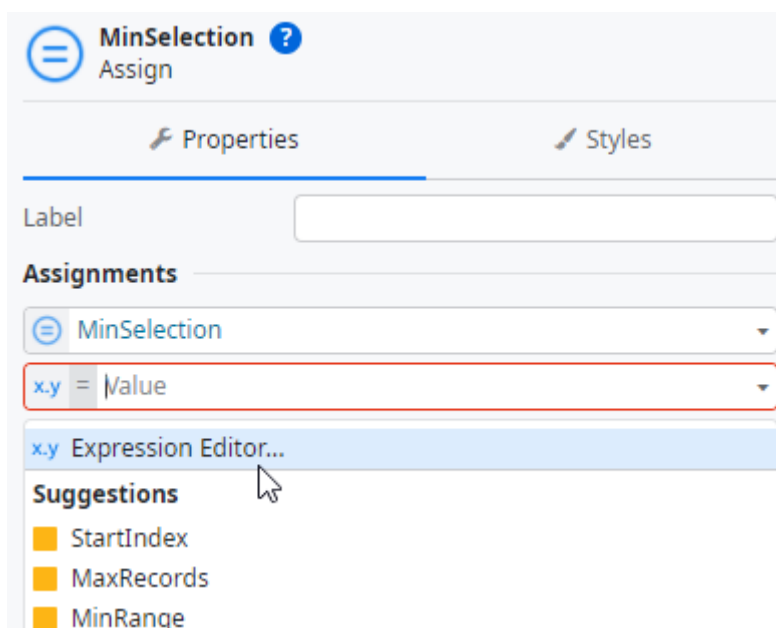
Client Action

So, this Action runs immediately after the user selects the interval, but... how does the Action know what are the values selected by the user? These input parameters will handle that automatically. We just need to assign these values to our new Local Variables, so they don't get lost.

- 5) Click on the Assign element and select the **MinSelection** Local Variable in the Variable dropdown.



- 6) Now, on the **Value** dropdown, select the **Expression Editor...** option.



- 7) In the new dialog, double-click on the **IntervalStart** input parameter of the Action. Click the **Close** button when you're done.

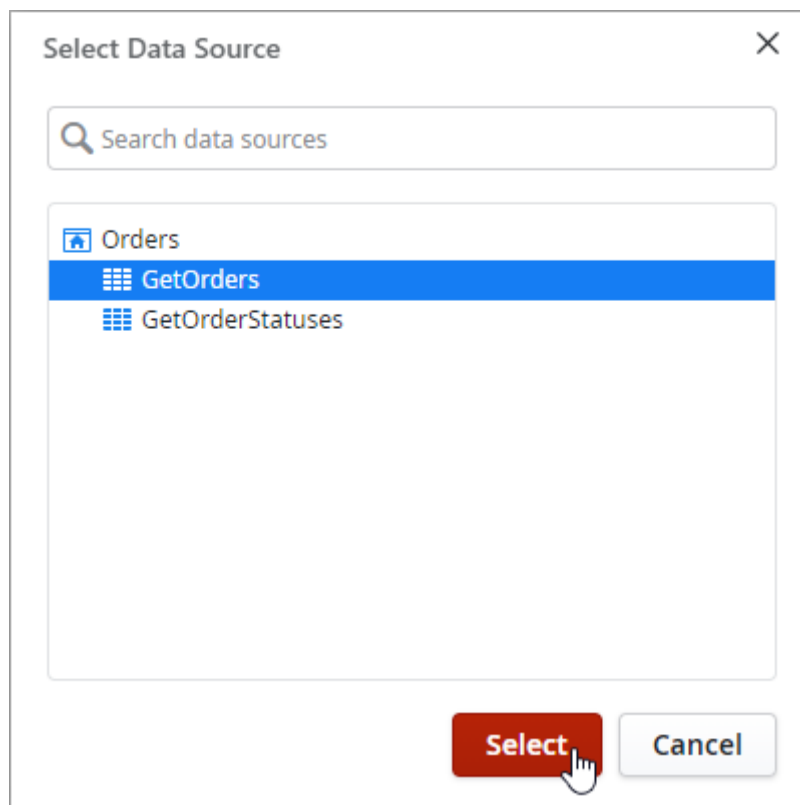
- 8) Go back to the Assign and follow the same strategy to create the following assignment

`MaxSelection = IntervalEnd`

- 9) Now that we have the interval selected by the user using the slider, we need to execute the Aggregate that fetches the orders again, to make sure the table of orders reflects this new sorting criterion. Drag a **Refresh Data** element from the Toolbox and drop it on the flow, after the Assign.



- 10) Select the **GetOrders** Aggregate as the Data Source.

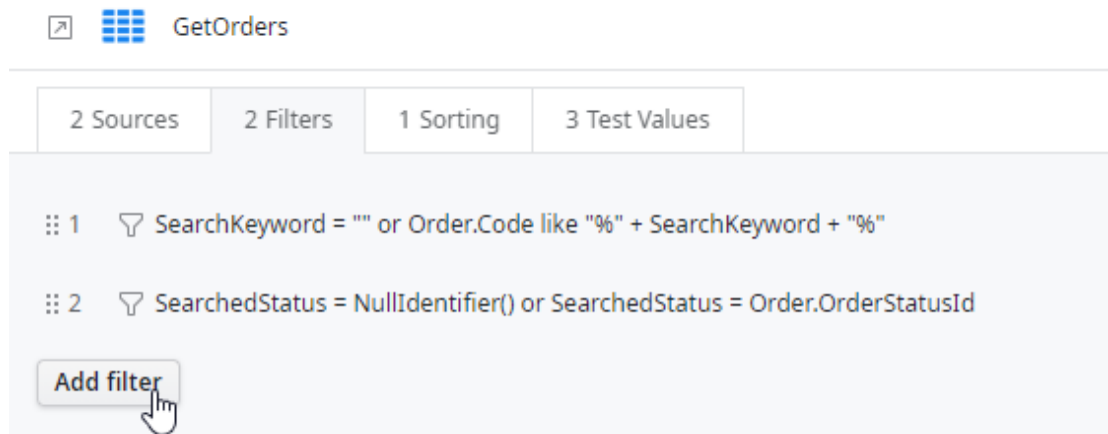




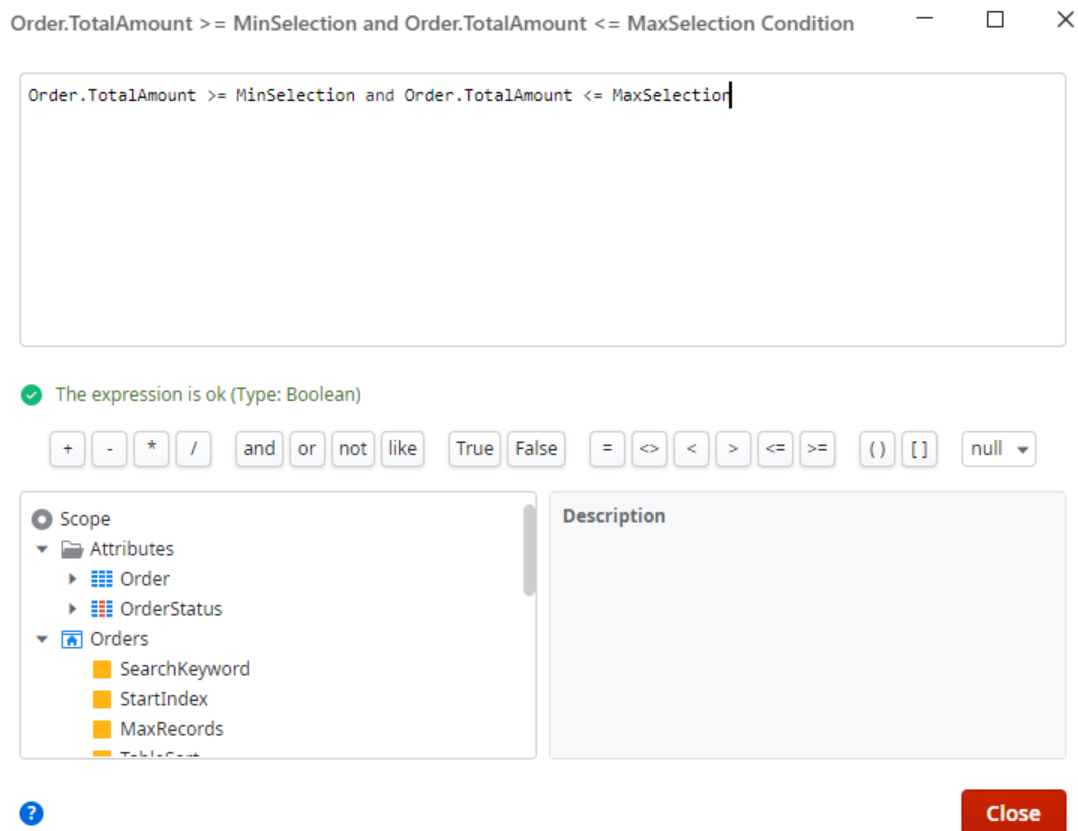
## Filter the Orders by the Amount

Speaking of the Aggregate... Remember the filters? That's right, we have to go back to it and add a third filter, so that the search for orders reflects the interval selected by the user in the range slider.

- 1) Open the **GetOrders** Aggregate, click on the Filter tab and on the **Add filter** button.



- 2) Add the following filter: `Order.TotalAmount >= MinSelection` and `Order.TotalAmount <= MaxSelection`

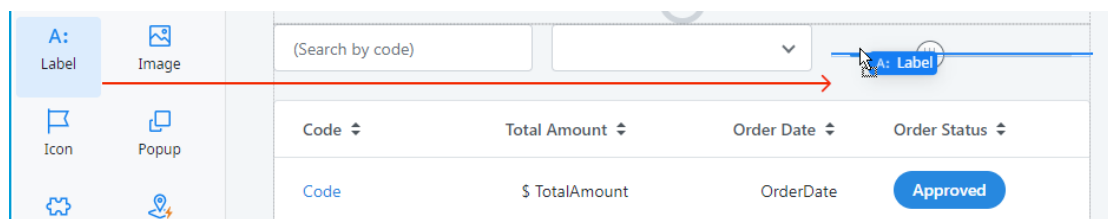


We just want orders with total amount that's within the range selected by the user.

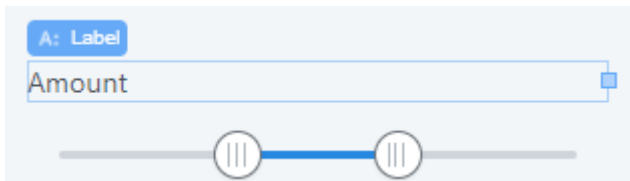
## Final Touches: Adding a Label

We have the logic ready! But, think about the user using your app. They will open the Screen and see a range slider without any info. So, let's help the user out by adding a Label.

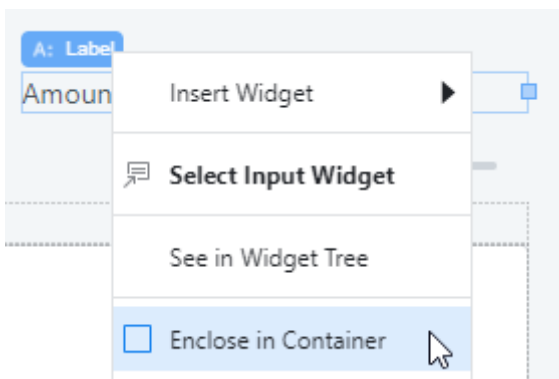
- 1) Back on the Orders Screen, drag a **Label** from the Toolbox and drop it right before the Range Slider.



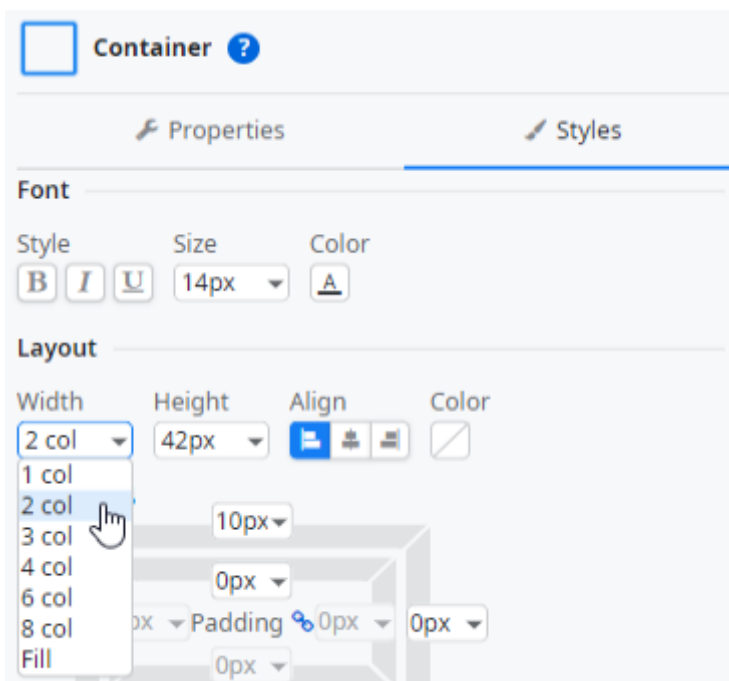
- 2) Write the text *Amount*.



- 3) Right-click the Label, and select **Enclose in Container**, so you can have more control of the Label's appearance.



- 4) In the Container properties, click on the **Styles** tab and select 2 columns in the **Width** property.



- 5) Enclose the Range Slider in a Container as well. Type *10 col* in the Width dropdown. In the end, it should look like this:



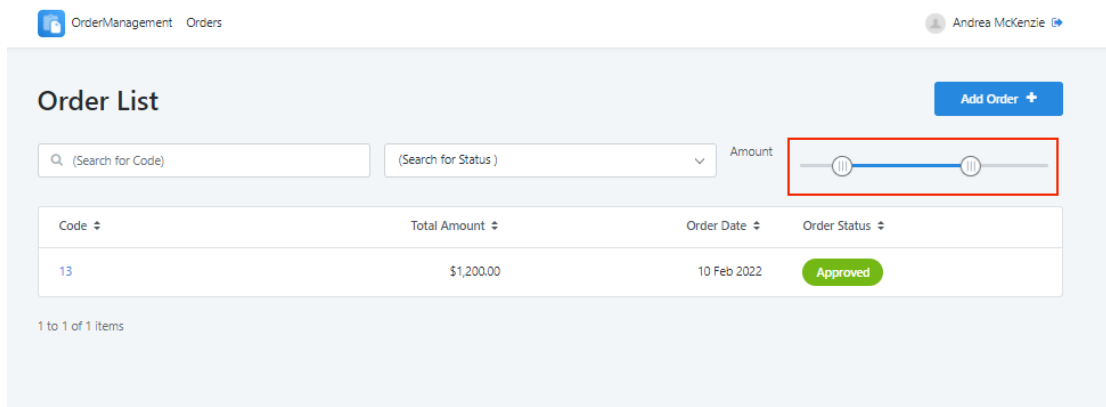
## Testing

Let's test the application!

- 1) Publish the app, then open it in the browser.



- 2) Test the three types of search you created by first typing an Order code, then selecting a status in the dropdown, and then changing the price range in the slider.



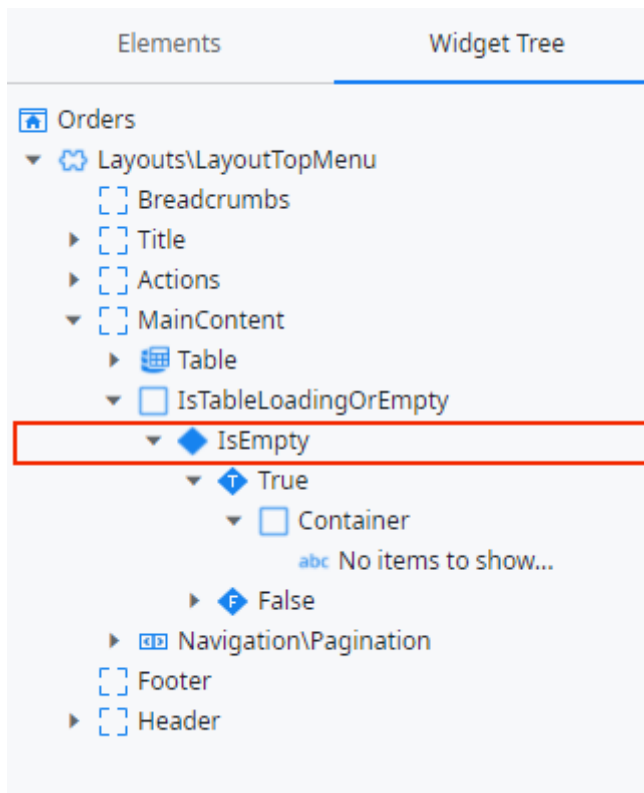
## Improve Visuals

In terms of the actual functionality we want to implement, you're done! Now it is time to improve the visual aspects of the Screen and make sure the user sees something if there are no orders, or if the search did not return any result.

When you open the application at this point and there are no orders, the table header still appears. So, let's start by arranging a bit the Screen to improve that.

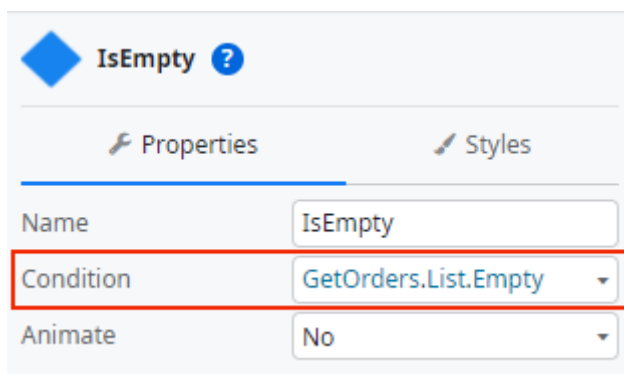
## Hiding the Table

- 1) In the Widget Tree, under **IsTableLoadingOrEmpty**, select the **IsEmpty** element by clicking on it.



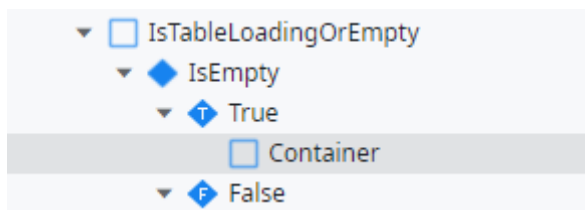
This is an If element that helps define some UI that is showed or not, based on a condition.

- 2) On the right sidebar, switch back to the Properties tab. Change the **Condition** to: `GetOrders.List.Empty`

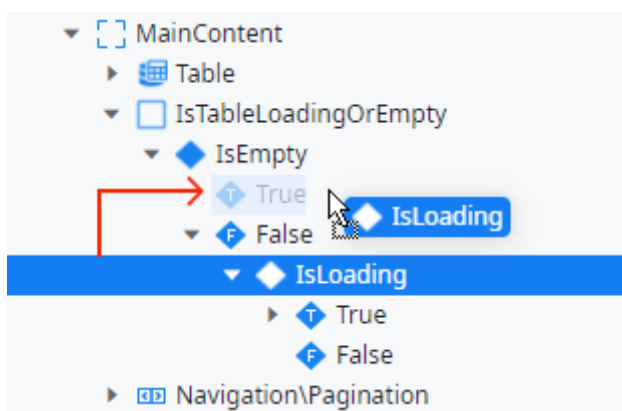


This checks if the GetOrders Aggregate returned any data.

- 3) Expand the If and delete the Container that's inside the True branch. We will come back later and improve this visual experience.

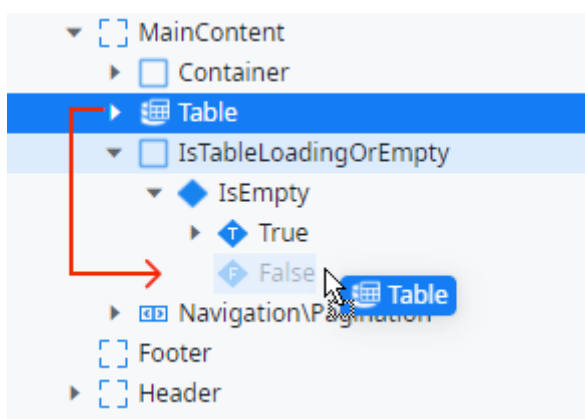


- 4) Expand the **False** branch of the If. You can find more elements in there that define the UI for when the Screen is loading the orders. Select the **IsLoading** If and drag it to the **True** branch of the IsEmpty If.



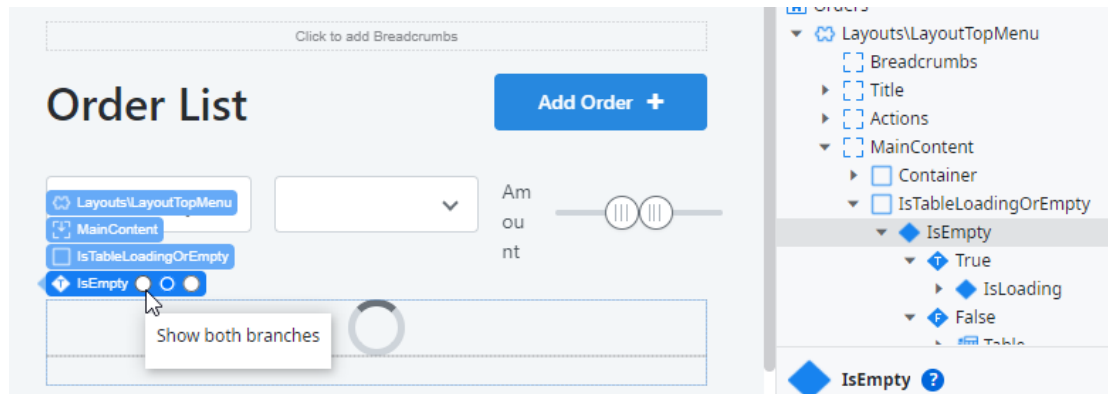
So, if the GetOrders Aggregate has no data yet, it means that the data may be loading. That's why we moved the "Loading part" to the True branch of the IsEmpty If.

- 5) Drag the **Table** to the False branch of the IsEmpty If.



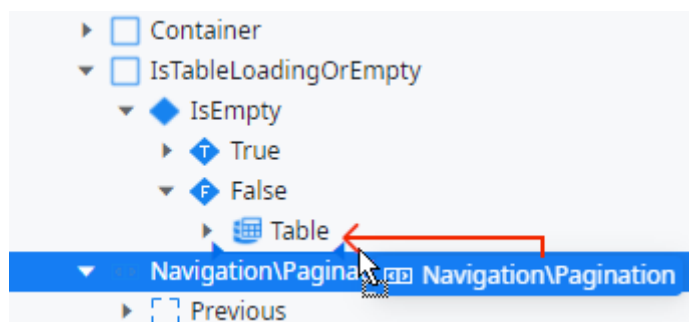
The Table will only appear if the GetOrders Aggregate is not empty.

- 6) The Table "disappeared" from the Screen? Don't worry! It's for preview purposes. Select the **IsEmpty** If, and on the Screen select the leftmost circle to make sure it appears again.



These small circular controls help you decide the branches of the If you want to see in the preview of Service Studio: both, only the true branch or only the false branch.

- 7) Back on the Widget Tree, drag the **Navigation** element and drop it right after the Table, inside the False branch.



The house is cleaned! Let's move on to the new UI.

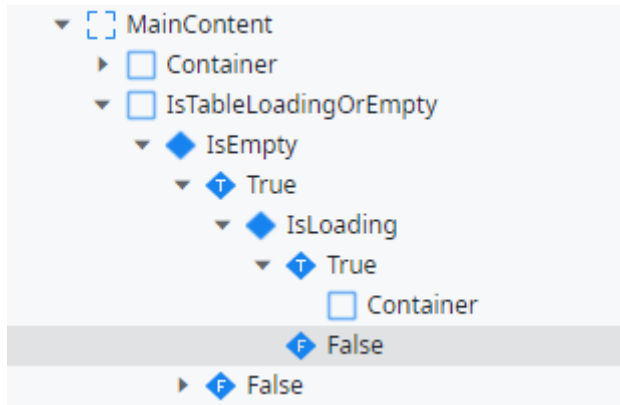
## UI for "No Orders Available"

Let's start by defining the UI for when there are no orders in the database yet. Remember, we are working inside the True branch of the Is Empty If. This means that we know already that the Aggregate resulted in no orders. But that can happen for three reasons:

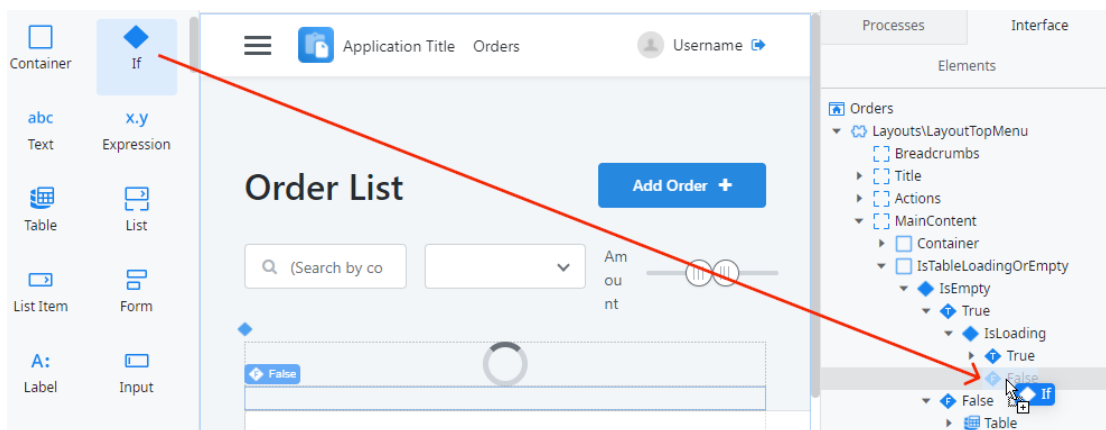
- Data is loading, which is already implemented!
- The search filter returned no results, which we will do in the next section.
- There are actually no orders in the database!

So, if the Aggregate result is empty, and the data is not being loaded... let's proceed from there!

- 1) Expand the **IsLoading** If.

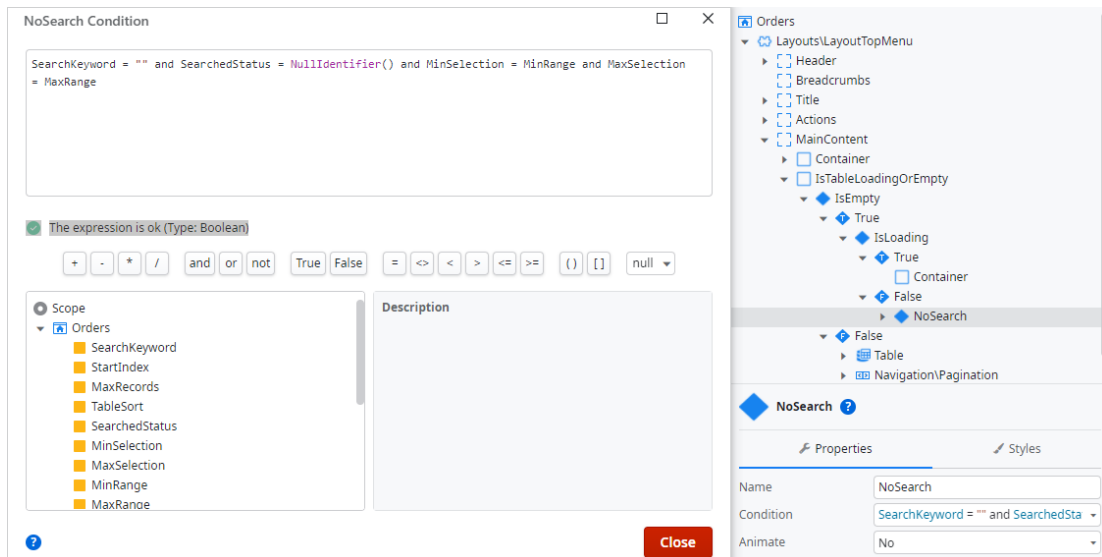


- 2) Drag another If from the Toolbox and drop it on the **False** branch of the IsLoading If.





- 3) Name the If **NoSearch** and set the If Condition to: `SearchKeyword = ""` and `SearchedStatus = NullIdentifier()` and `MinSelection = MinRange` and `MaxSelection = MaxRange`

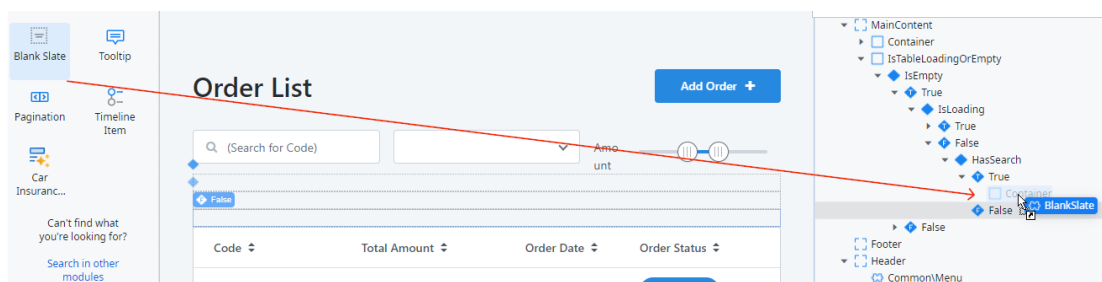


This Condition returns true if there is no search criteria selected: the search for code and status is empty or null, and the range slider stands between the very first and very last value of the interval. So, if there is no search criteria and the data is not loading, this can only mean one thing: there are no orders in the database!

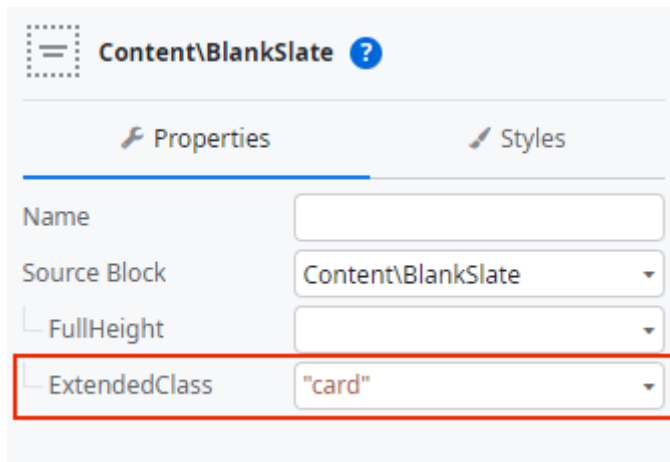
## Creating the Blank Slate

Let's handle the UI part now!

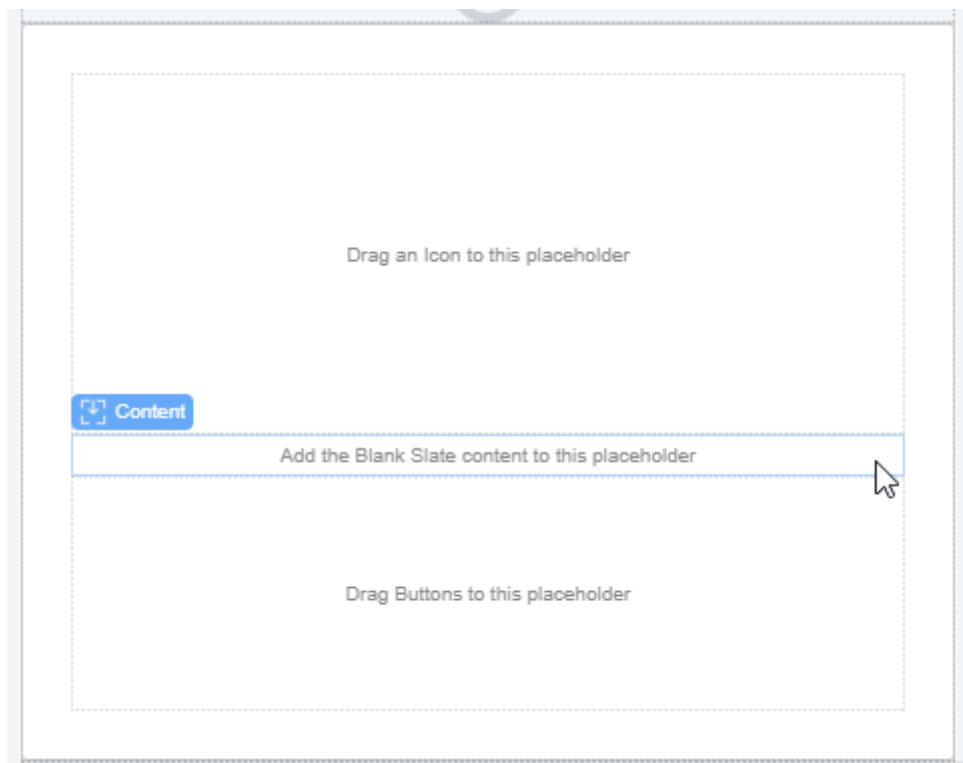
- 1) Search for a **BlankSlate**, then drag and drop it to the True branch of the HasSearch If.



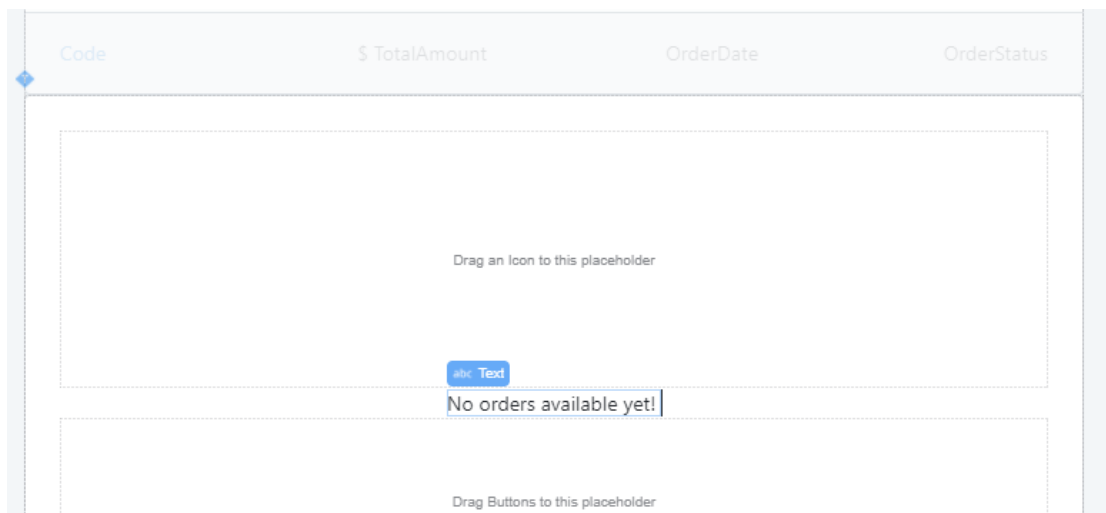
- 2) Enclose the Blank Slate in a Container, then type "card" in the **ExtendedClass** property.



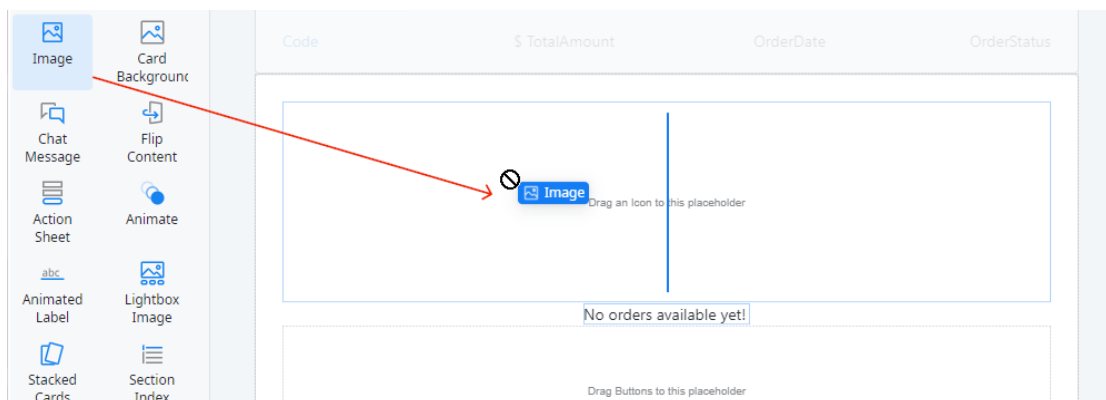
- 3) Expand the Blank Slate to see its structure in the Widget Tree and on the Orders Screen preview. It has three areas or placeholders: Icon, Content and Actions. Select and delete the icon inside the Icon placeholder.



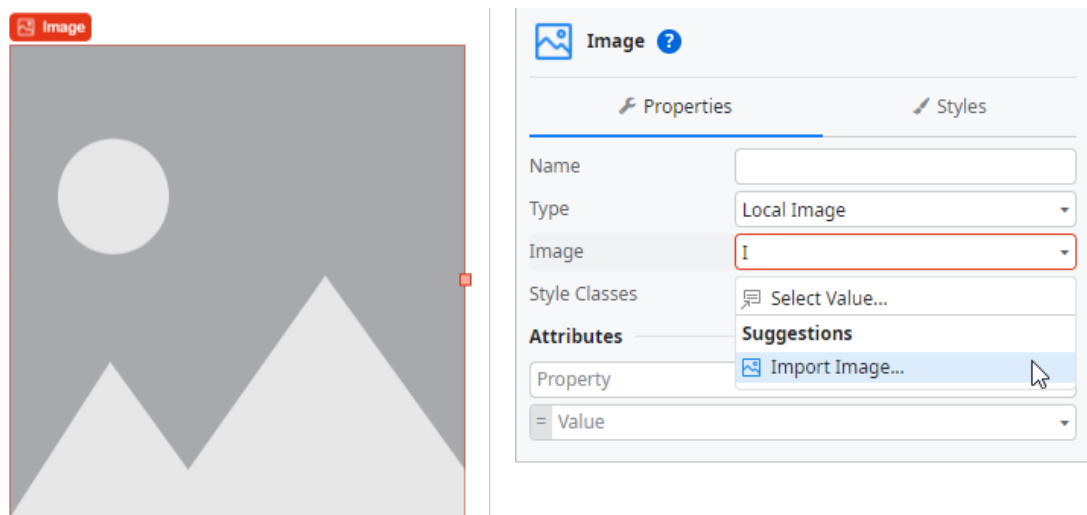
- 4) Select the Content placeholder on the Blank Slate and add the text *No orders available yet!*.



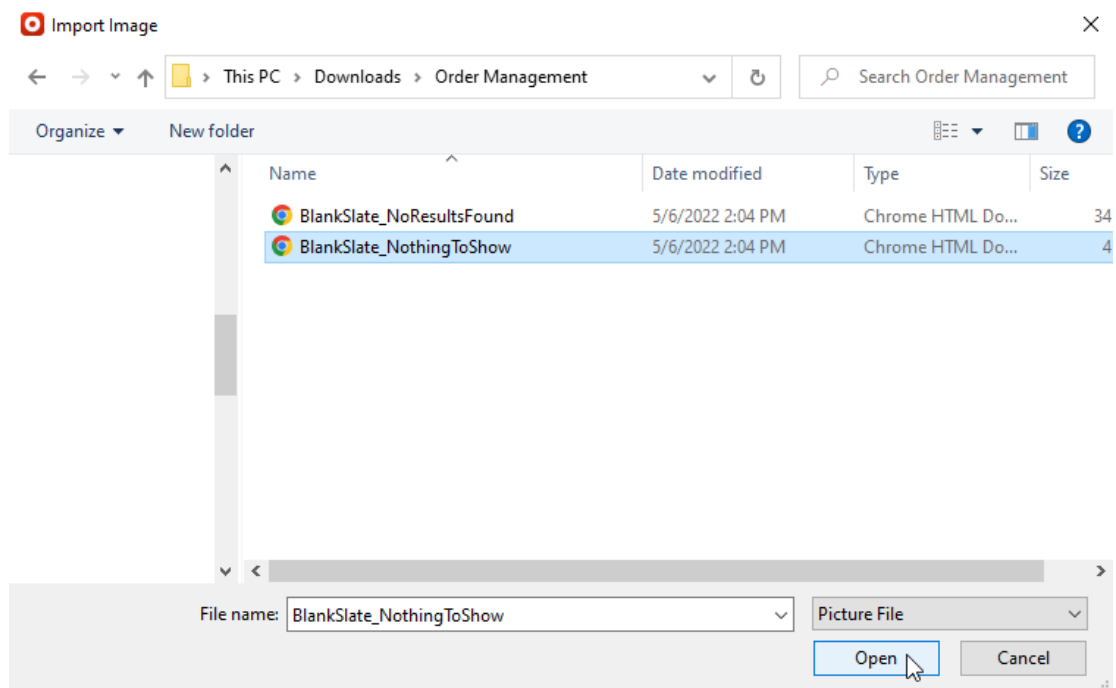
- 5) Drag an **Image** from the Toolbox and drop it in the Icon placeholder.



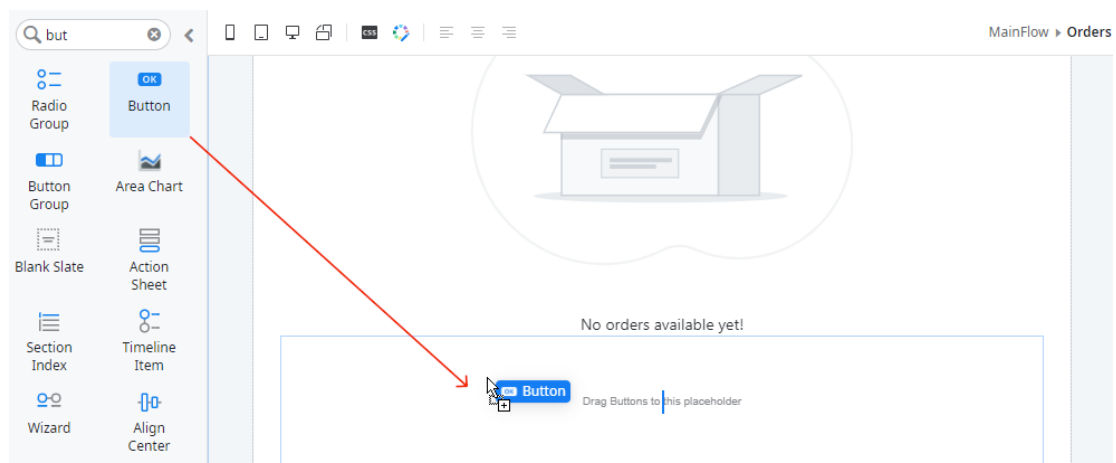
- 6) In the image properties, click on the Image property and select **Import Image...**



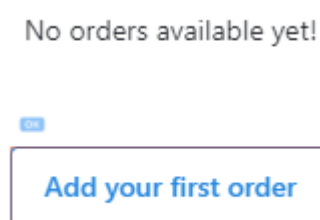
- 7) Select the image **BlankSlate\_NothingToShow** available on the Resources downloaded from the Lesson Materials.



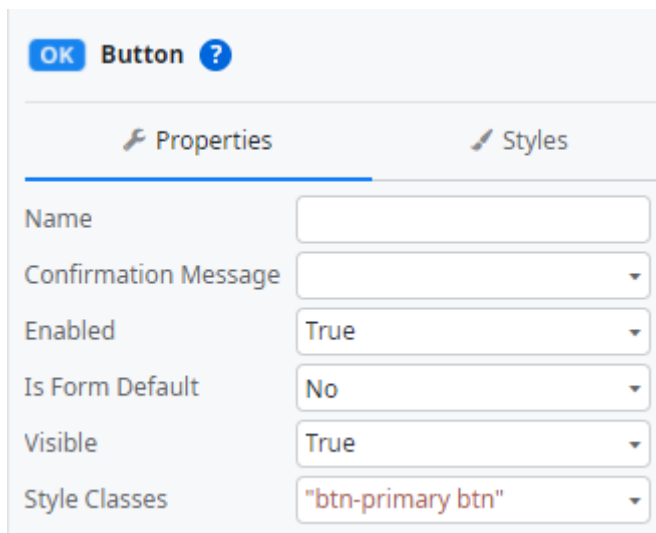
- 8) Add a **Button** to the Actions placeholder.



- 9) Change the Button text to *Add your first order*



- 10) Select the Button, and on its properties in the right sidebar, set the **Style Class** to *"btn-primary btn"*.

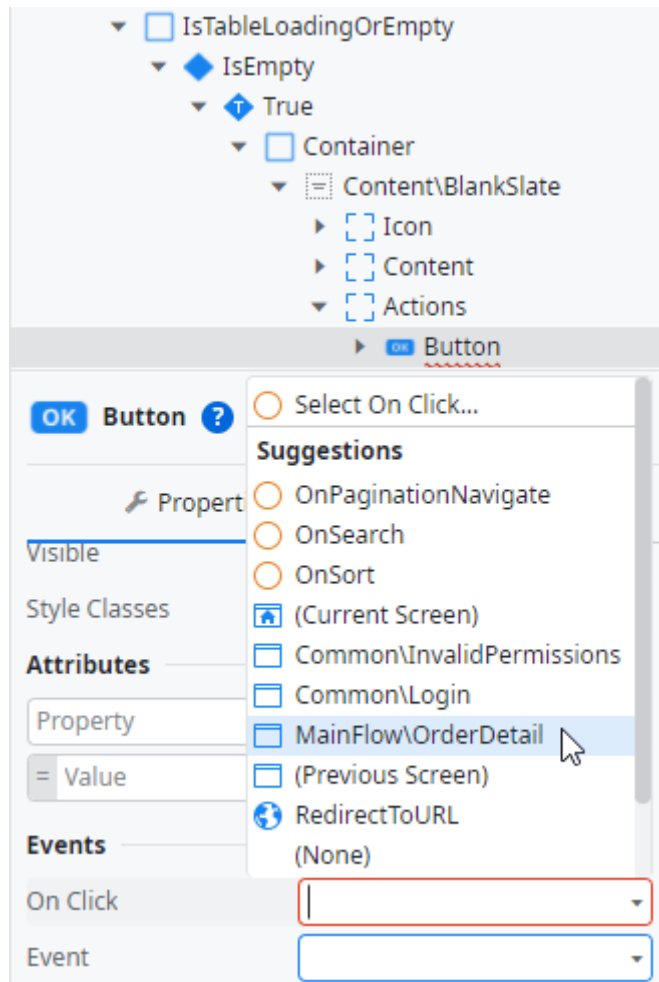


The screenshot shows the 'Properties' tab of the OutSystems Properties Panel for a 'Button' widget. The panel has a header with 'OK', 'Button', and a help icon. Below the header, there are two tabs: 'Properties' (selected) and 'Styles'. The 'Properties' tab contains a list of properties with their corresponding values in dropdown menus:

Property	Value
Name	
Confirmation Message	
Enabled	True
Is Form Default	No
Visible	True
Style Classes	"btn-primary btn"

This will set the color of the Button to use the primary color of the app.

- 11) In the Button's properties, set the **OnClick** Event to the OrderDetail Screen. This means that the user will navigate to the new Screen when clicking on the Button.



- 12) The OrderDetail Screen expects the identifier of the order that we want to edit. Since we don't want to edit and instead create a new one, the identifier does not exist, so we need to set the **OrderId** to `NullIdentifier()`.

The screenshot shows the OutSystems IDE interface for configuring a Button widget. The 'Properties' tab is active, showing various properties like 'Enabled', 'Is Form Default', 'Visible', and 'Style Classes'. Below the properties, the 'Attributes' section is visible, followed by the 'Events' section. In the 'Events' section, the 'On Click' event is configured with 'MainFlow\OrderDetail' as the target. The 'OrderId' argument is set to 'NullIdentifier()'. A suggestions dropdown is visible for the '(New Argument)' field, showing 'NullIdentifier()' as a suggestion.

Property	Value
Enabled	True
Is Form Default	No
Visible	True
Style Classes	"btn-primary btn"

**Attributes**

Property	Value
Property	
= Value	

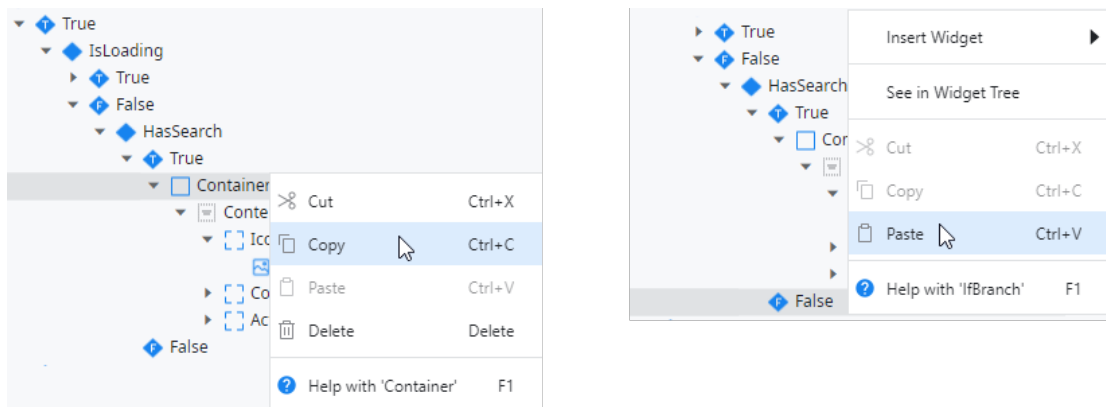
**Events**

Event	Target	Expression
On Click	MainFlow\OrderDetail	NullIdentifier()
(New Argument)		x.y Expression Editor...
Transition		Suggestions
Event		NullIdentifier()

## Search Criteria Resulted in No Orders

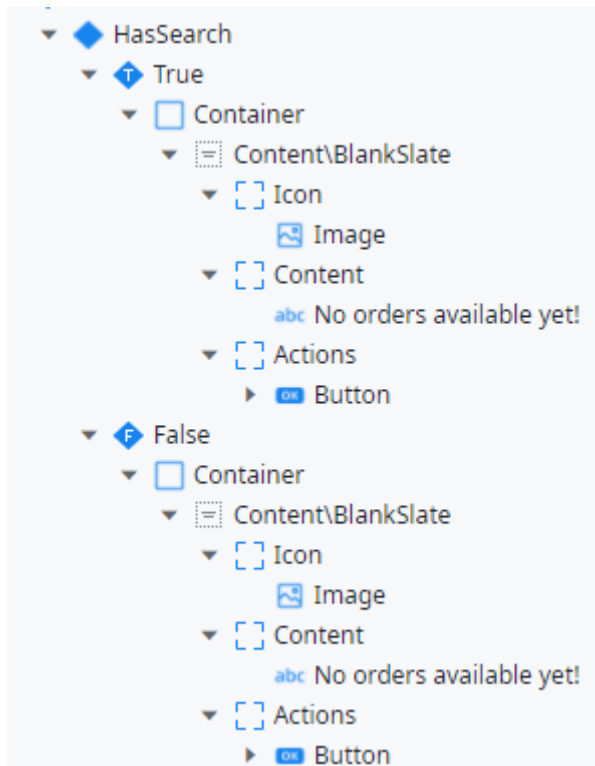
Now we need to create a similar Blank Slate in the False branch of the **HasSearch** if. You're probably a bit tired at this point right? So let's cut some corners!

- 1) Copy the Container enclosing the Blank Slate, and paste it on the False branch of the **HasSearch** If.



**Tip:** You can use the keyboard shortcuts "ctrl + c" and "ctrl + v"

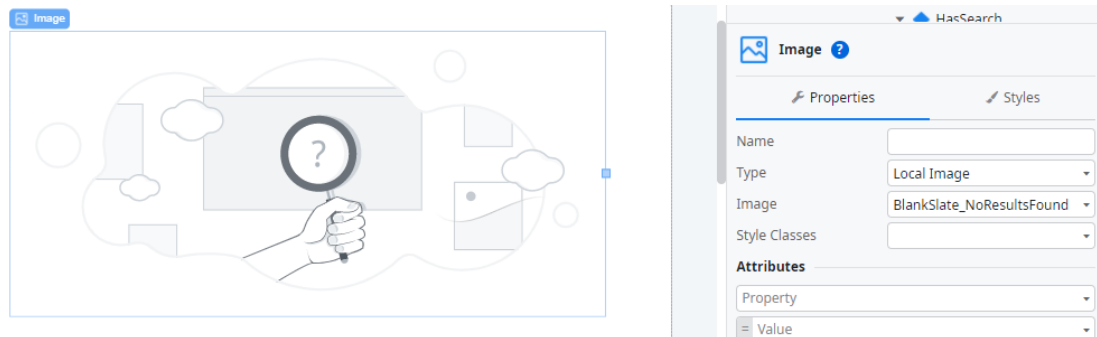
Your Widget Tree should look like this in the end:



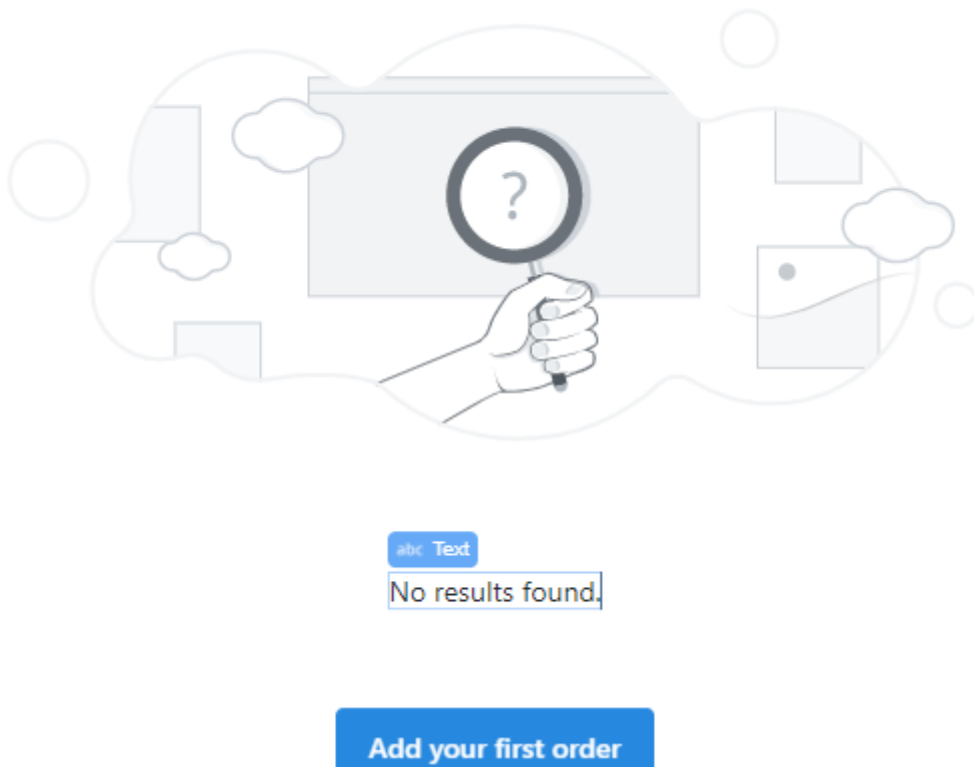
If the HasSearch condition fails, this means we actually have a search criteria that didn't result in any orders!



- 2) Ok! What changes? We have a different image! So, select the image inside the new Blank Slate and change it to the image **BlankSlate\_NoResultsFound**, by importing it from your computer.



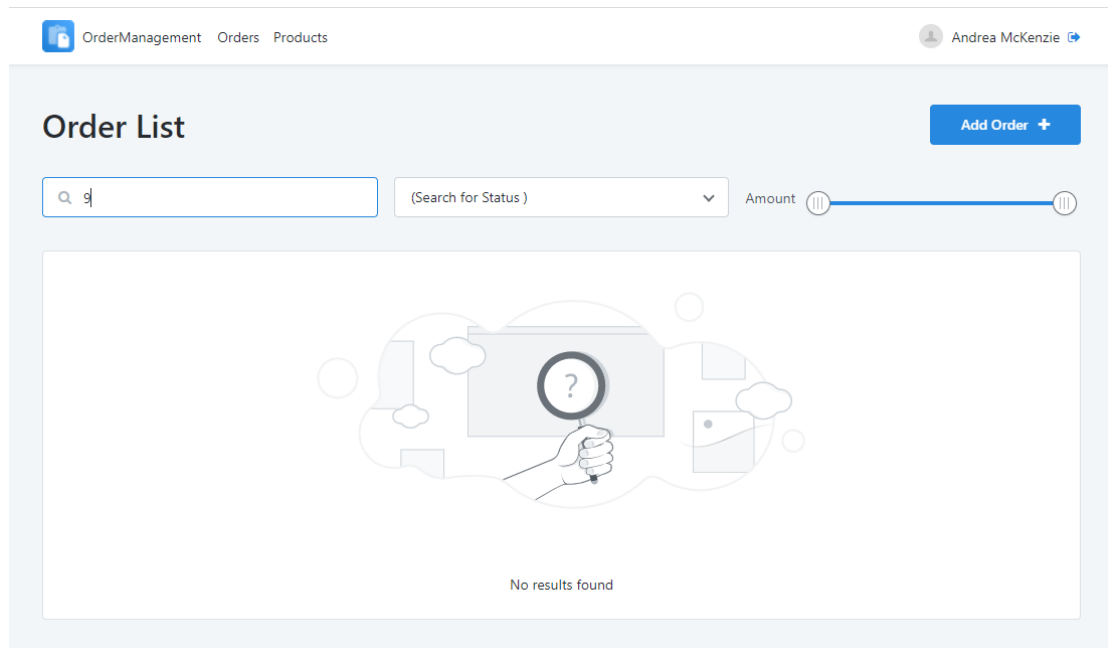
- 3) The text also needs to change! Type the text *No results found*.



- 4) And finally we don't need the Button, so delete it. And that's it! That was faster! Copy and paste is not always bad...
- 5) Time to publish and open the app in the browser.



- 6) Test your app in the browser. For instance, you can test searching for a code that does not exist so you can see the *No results found*.



## Wrapping up

Congratulations on finishing this tutorial! Just take a minute to experiment what you achieved so far! With this exercise, we had the chance to go through some essential aspects of OutSystems and get to know more about the platform.

## References

If you want to deep dive into the subjects that we have seen in this exercise, we have prepared some useful links for you:

- 1) [Aggregates 101](#)
- 2) [Container Widget](#)
- 3) [Dropdown Widget](#)
- 4) [OutSystems UI Patterns](#)

**See you in the next tutorial!**