

# Using a Wizard – Preparing the Order Detail Screen

## Table of Contents

|   |           |
|---|-----------|
| <b>Outline.....</b>                               | <b>2</b>  |
| Scenario  | 3         |
| <b>How-to.....</b>                                | <b>5</b>  |
| Getting Started                                   | 5         |
| Adjusting the Layout                              | 6         |
| Adjusting the Order Status                        | 8         |
| Adding the Tag                                    | 8         |
| Implementing the Color Code and Hiding the Status | 13        |
| Adjusting the Total Amount                        | 16        |
| Save Button and Back to Orders                    | 19        |
| Changing the Save Action                          | 24        |
| <b>Wrapping up.....</b>                           | <b>29</b> |
| References  | 29        |

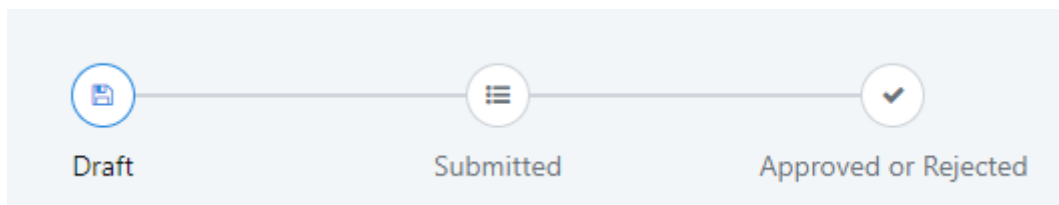
# Outline

In this set of tutorials, you will continue to extend the Order Management application. This time with the creation of a Wizard. This isn't magic, but it will definitely feel like it when you're done!

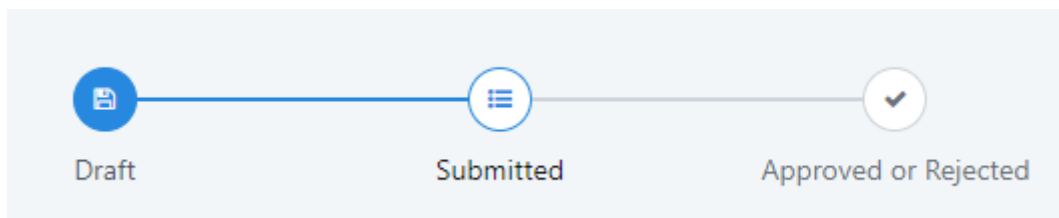
In this application, the Wizard will be used to present the steps that the user needs to complete to move through the several stages of an order, from the creation until the approval/rejection.

The Wizard will be added to the Order Detail Screen and will show three possible statuses of an order:

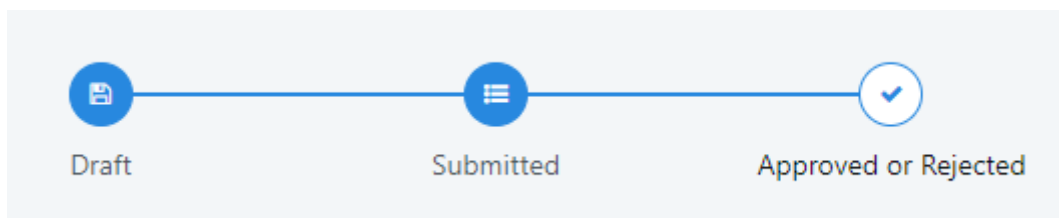
- Draft: An order that was created but not submitted yet. The order can still be modified at this stage.



- Submitted: After the user submits an order. The only possible action here is to approve or reject an order.



- Approved / Rejected : This is the final stage. After this, there are no actions possible.



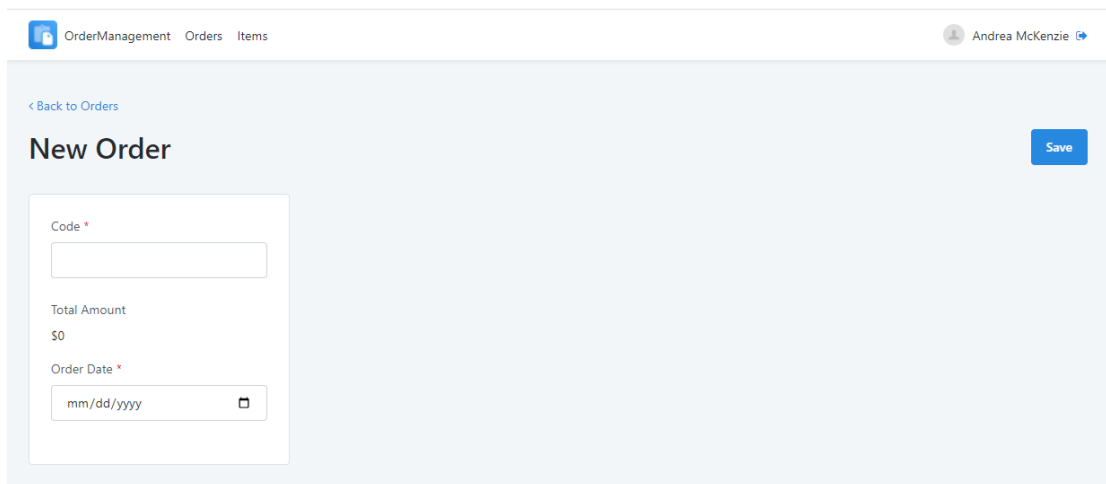
This is the first part of the Wizard implementation. In this specific tutorial, you will adjust the Form that contains the Order details and the Screen layout, to prepare the Screen for the order workflow.

## Scenario

We will continue to build the Order Management web application to manage customer orders, which includes controlling the process of receiving, tracking, and fulfilling them.

At this point, when a user clicks on **Add Order**, they are redirected to the OrderDetail Screen. However, you have a simple Form, where the Total Amount and the Order Status can be chosen by the user. If you think about it, it doesn't make a lot of sense. The total amount will depend on the products the order will have and the status should depend on the workflow of the order itself, which will be controlled by the Wizard. So, the OrderDetail Screen will have two scenarios:


- **New Order:** the user fills out the form to create a new order, then saves it. The Total Amount is an Expression and the Order Status does not appear yet until the order is created.



The screenshot shows a web application interface for 'OrderManagement'. The top navigation bar includes 'Orders' and 'Items'. A user profile for 'Andrea McKenzie' is visible in the top right. The main content area is titled 'New Order' and features a 'Save' button in the top right corner. A form is displayed with the following fields: 'Code \*' (a text input field), 'Total Amount' (displaying '\$0'), and 'Order Date \*' (a date picker showing 'mm/dd/yyyy'). A '< Back to Orders' link is located in the top left of the form area.

- **Existing Order:** if an order already exists, a user can click on the order code in the Orders Screen, and open the Order Detail Screen, this time with the details of the order that already exists. So the order's details must be fetched and

displayed accordingly to the total amount, and the order status should not be editable.

 OrderManagement Orders

[< Back to Orders](#)


## Edit Order

Code \*

Total Amount

\$400

Order Date \*

Order Status

Approved

In this tutorial, you will edit the Order Detail Screen to prepare it for the two scenarios above.

# How-to

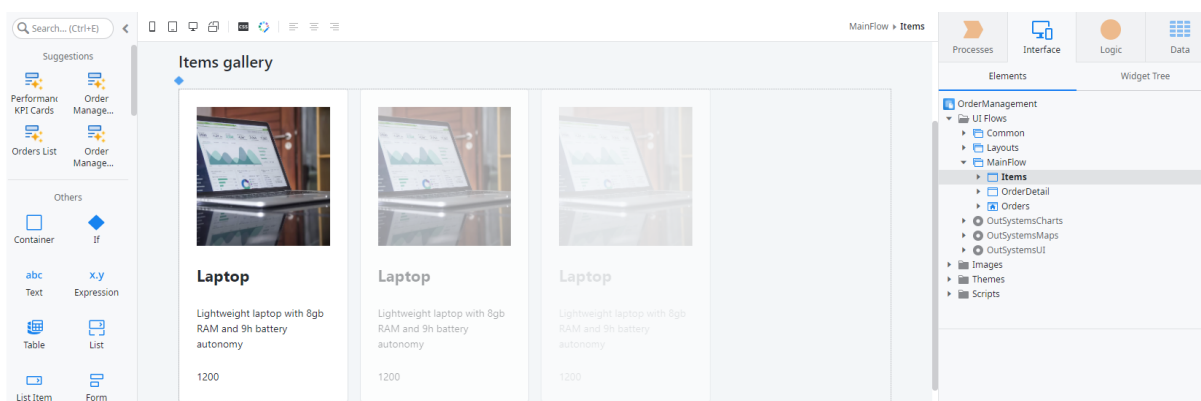
In this section, we'll show a thorough step-by-step description of how to implement the scenario described in the previous section.

## Getting Started

In this tutorial we are assuming that you have already followed the previous tutorials, and have the Orders, Order Detail and Items Screens ready.

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

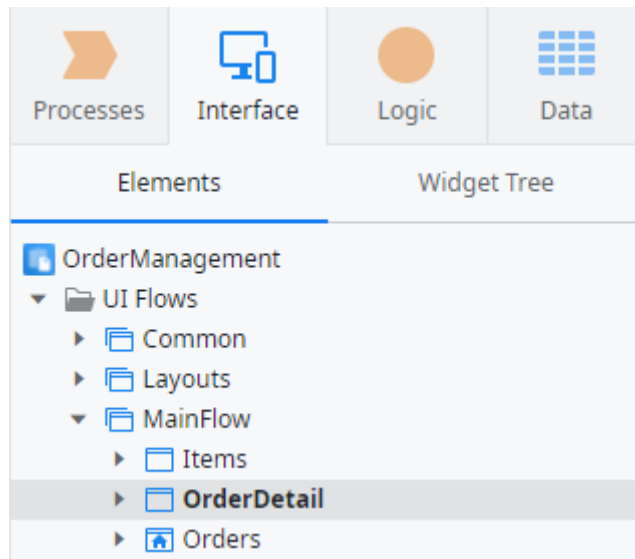
To start this exercise, you need the Service Studio with the module OrderManagement opened. You should see the Screen below with the source of our application.



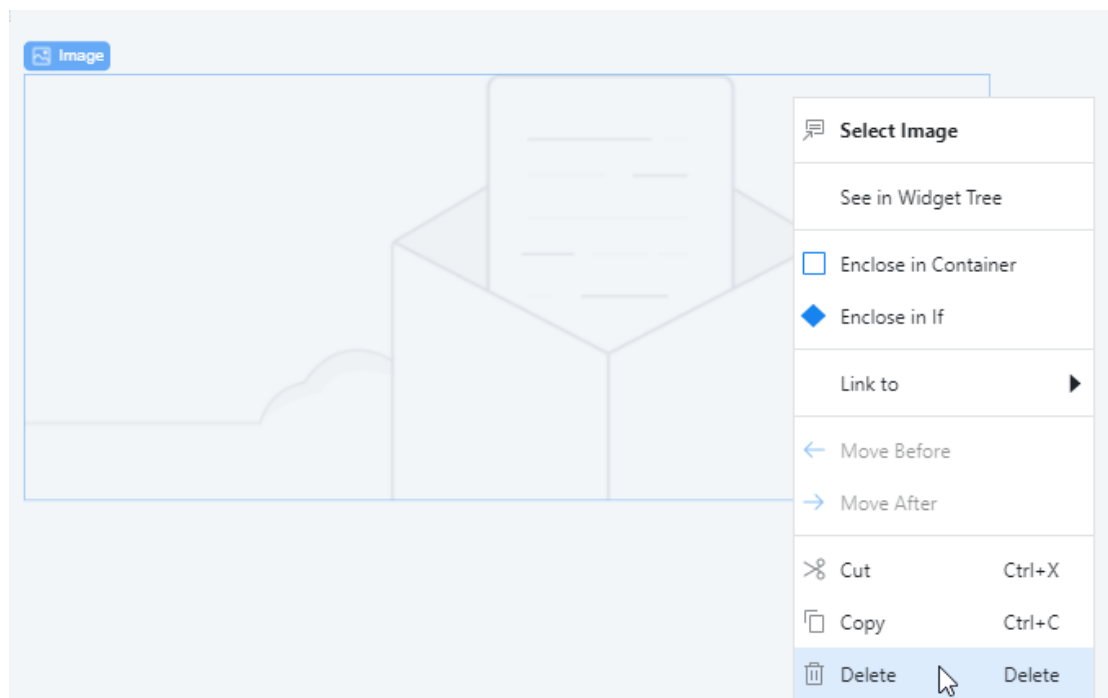
## Adjusting the Layout

Before we actually change any functionality on the OrderDetail Screen, let's adjust the layout of the Screen a bit, to make room for the Wizard.

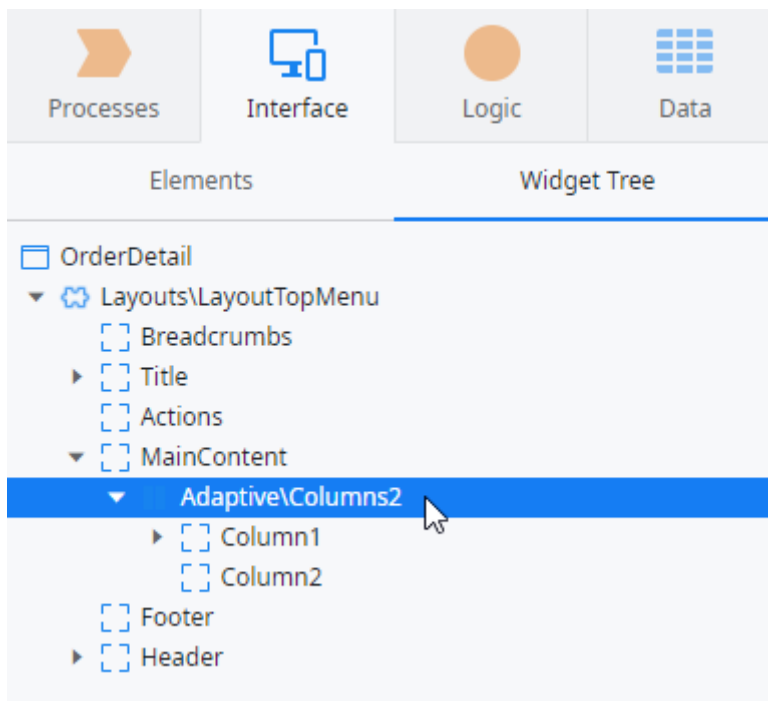
- 1) Open the **OrderDetail** Screen by double-clicking on it in the Interface tab.



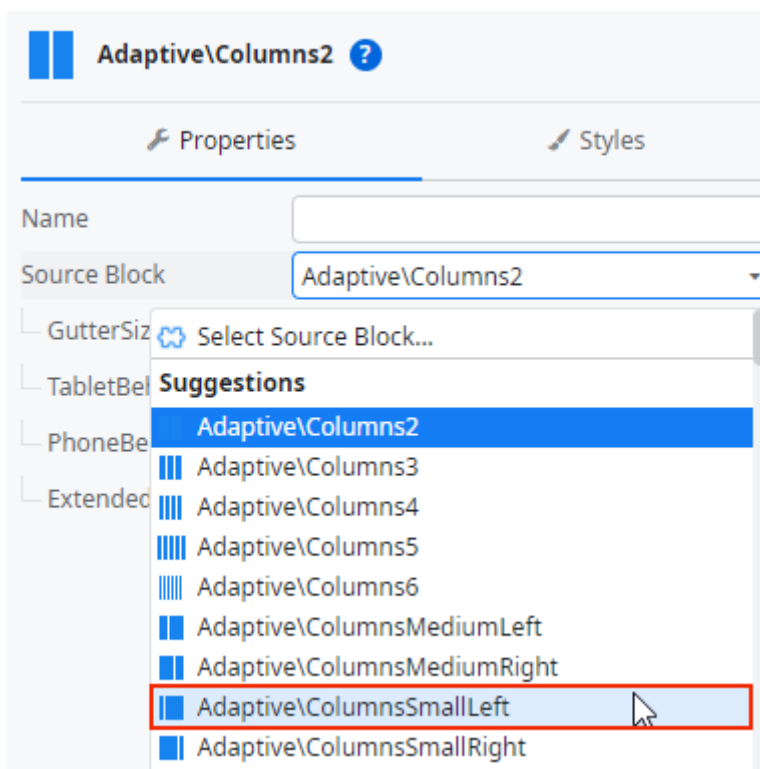
- 2) Click on the image on the right column, then delete it.



- 3) Open the Widget Tree, then click on the **Adaptive\Columns2** Widget to see its properties.



- 4) On the Source Block property of this Widget, expand the dropdown and select the **Adaptive\ColumnsSmallLeft**.



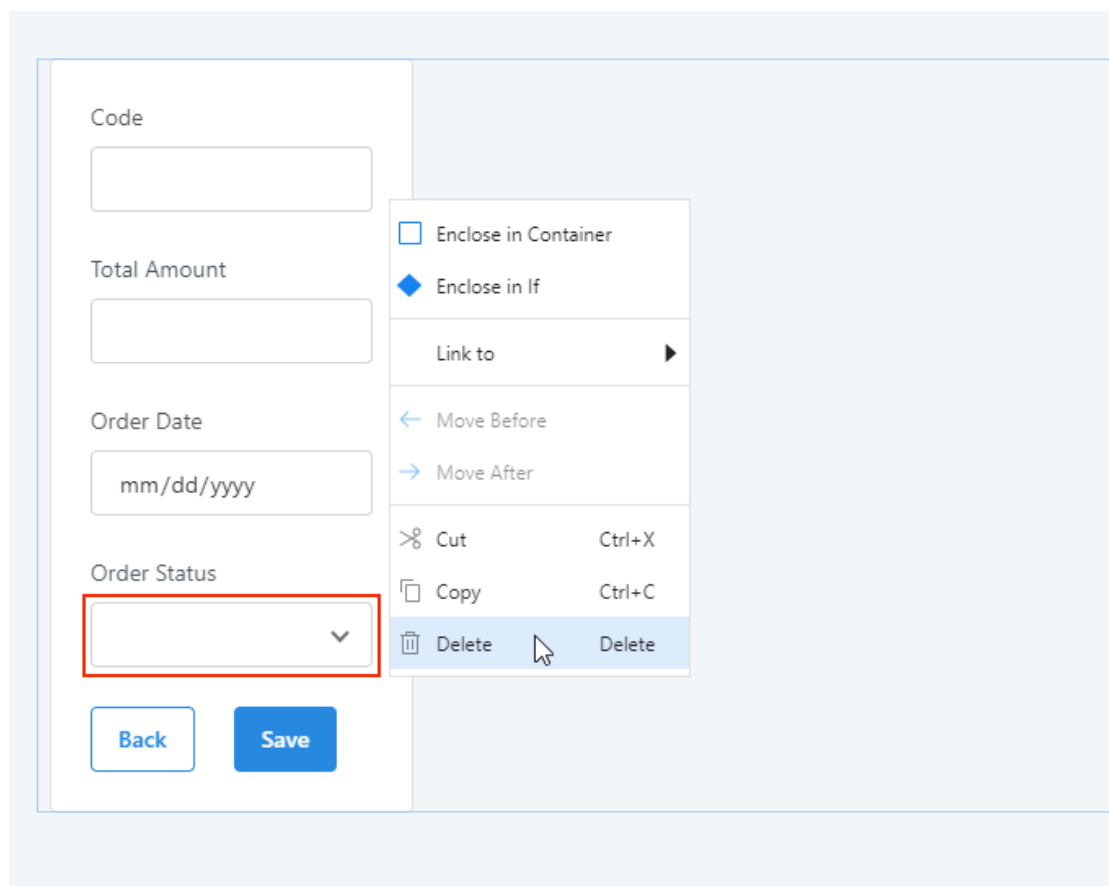
This will still keep the Screen split in two columns, but the Form is now on a much smaller column.

## Adjusting the Order Status

Now it is time to work on the Form fields, starting with the Order Status. You will change it using the Tag Widget, to make it look like what we have in the Orders Screen. You will also make sure the Tag only appears if the Order already exists.

## Adding the Tag

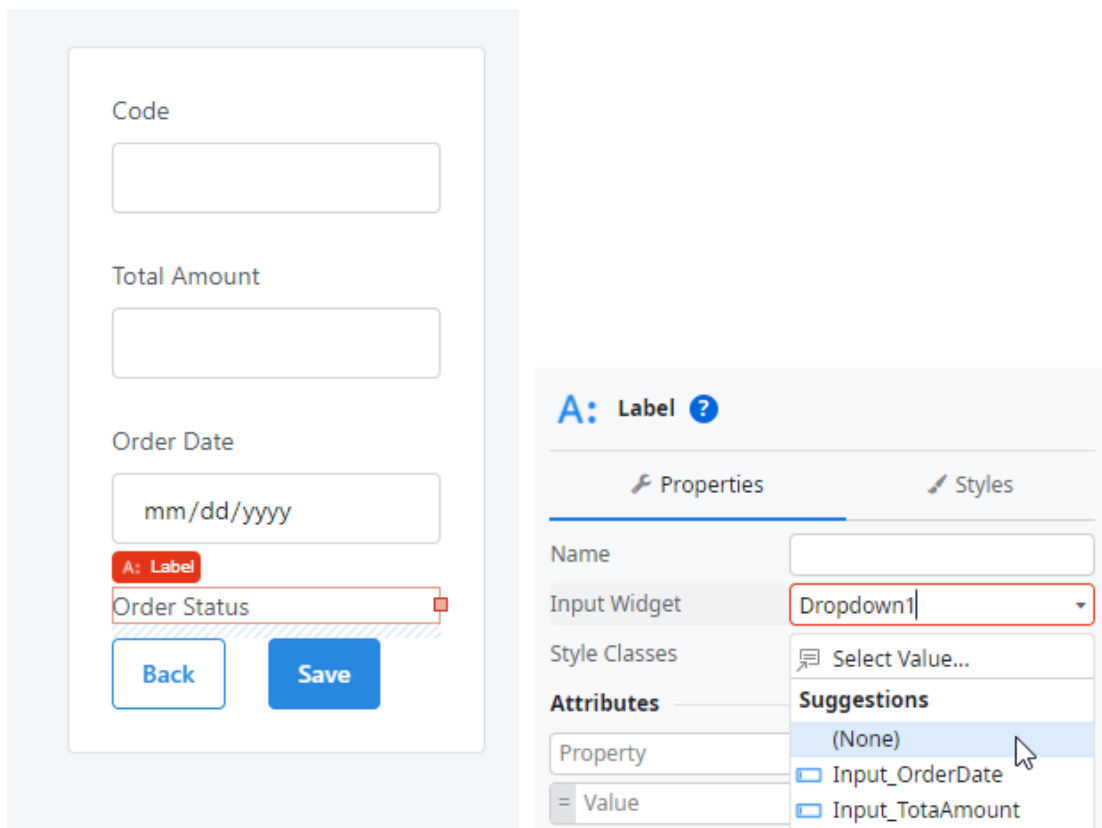
- 1) Click on the **Order Status Dropdown** inside the Form and delete it. (keep the Container around it!)



You will get an error because the Label references the dropdown you have just deleted. Let's fix it!



- Click on the **Label** with the error and change its **Input Widget** property to *(None)*.



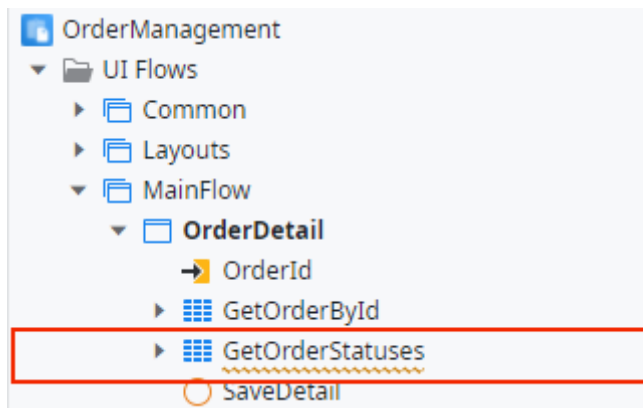
- Drag a **Tag** Widget from the Toolbox (left sidebar) and drop it inside the Container that has the Label of the Order Status.



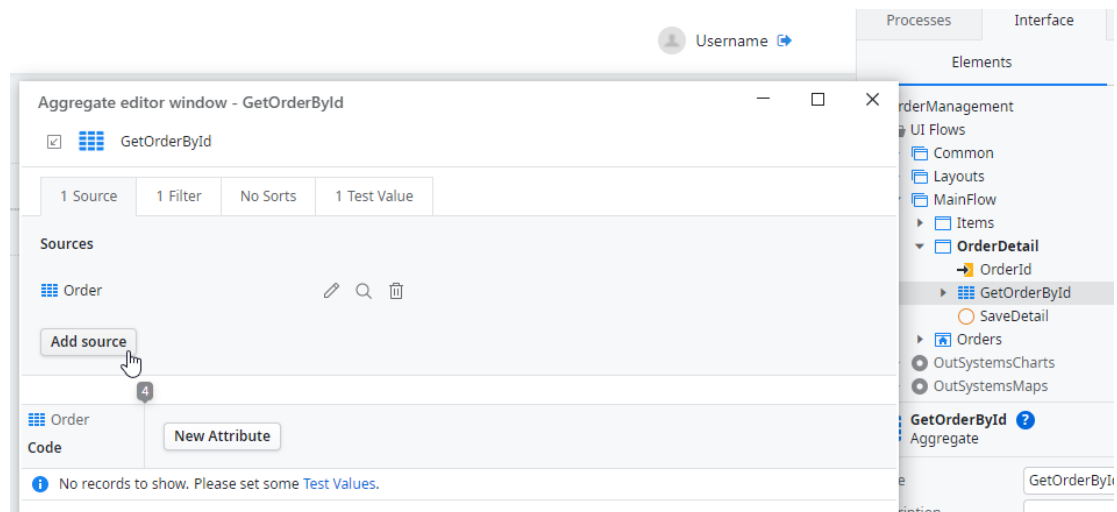
**Note:** Use the widget tree to help you out. Basically, the idea is to replace the deleted Dropdown with the Tag.

- Now you need to make sure the status appears inside the Tag. So you will need an Aggregate to fetch the status information. Go back to the OrderDetail

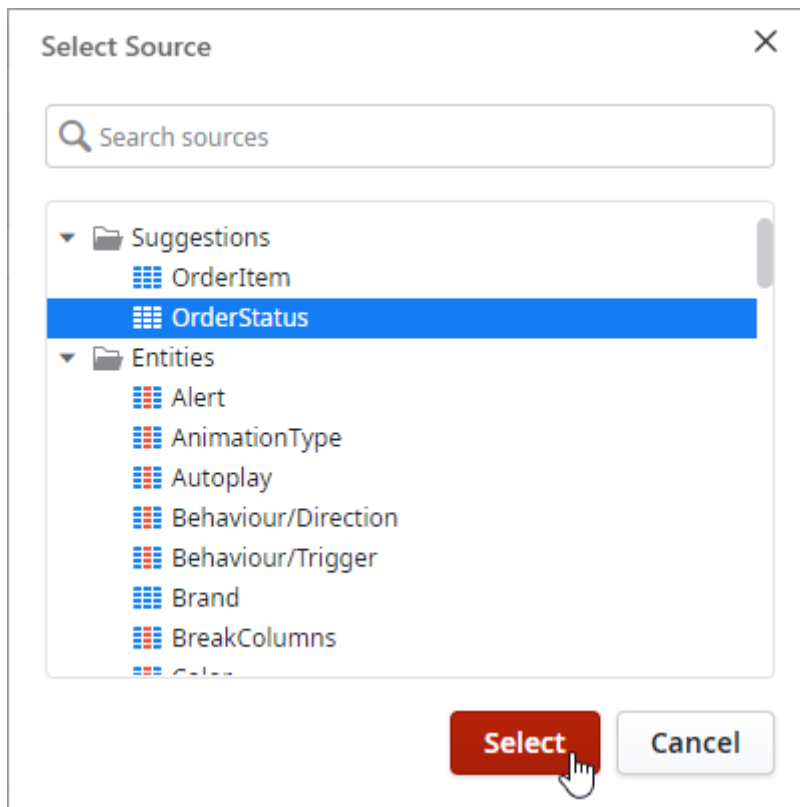
Screen, and delete the **GetOrderStatuses** Aggregate, since it is not being used anymore.



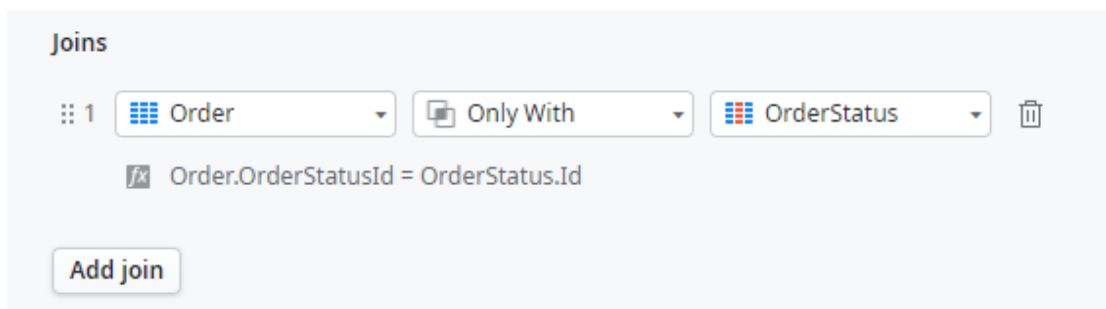
- 5) Open the **GetOrderById** Aggregate by double-clicking on it, and then click on the **Add source** button.



- 6) Select the **OrderStatus** Entity and click on **Select**.

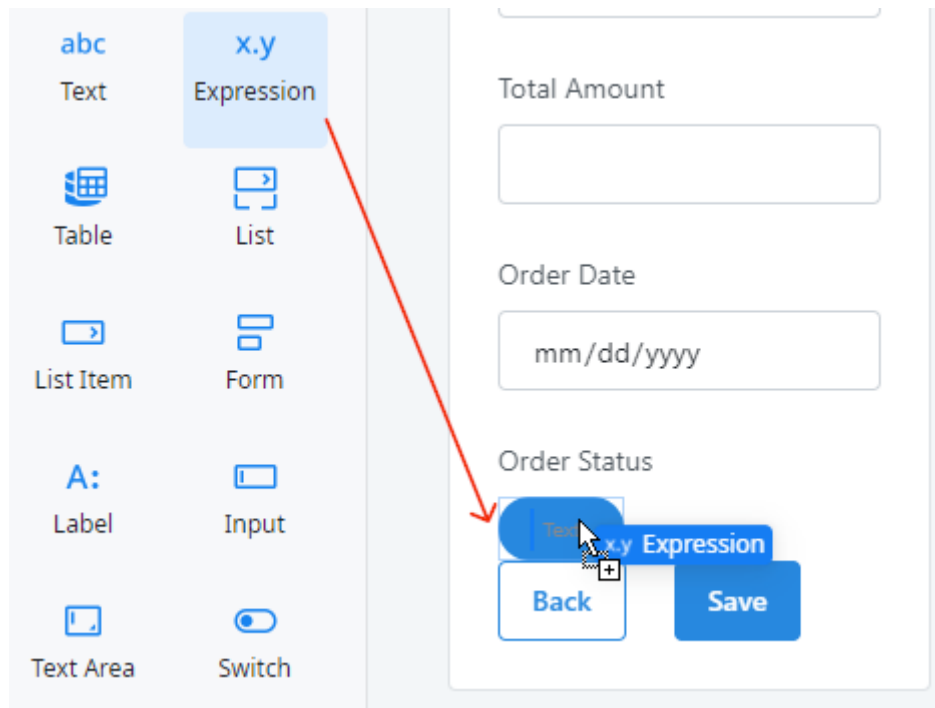


**Note:** A Join will appear on the Aggregate. This happens when you need to combine the records stored in multiple Entities. This Join combines the table Order and the Status Label so it can be used in the Tag component.

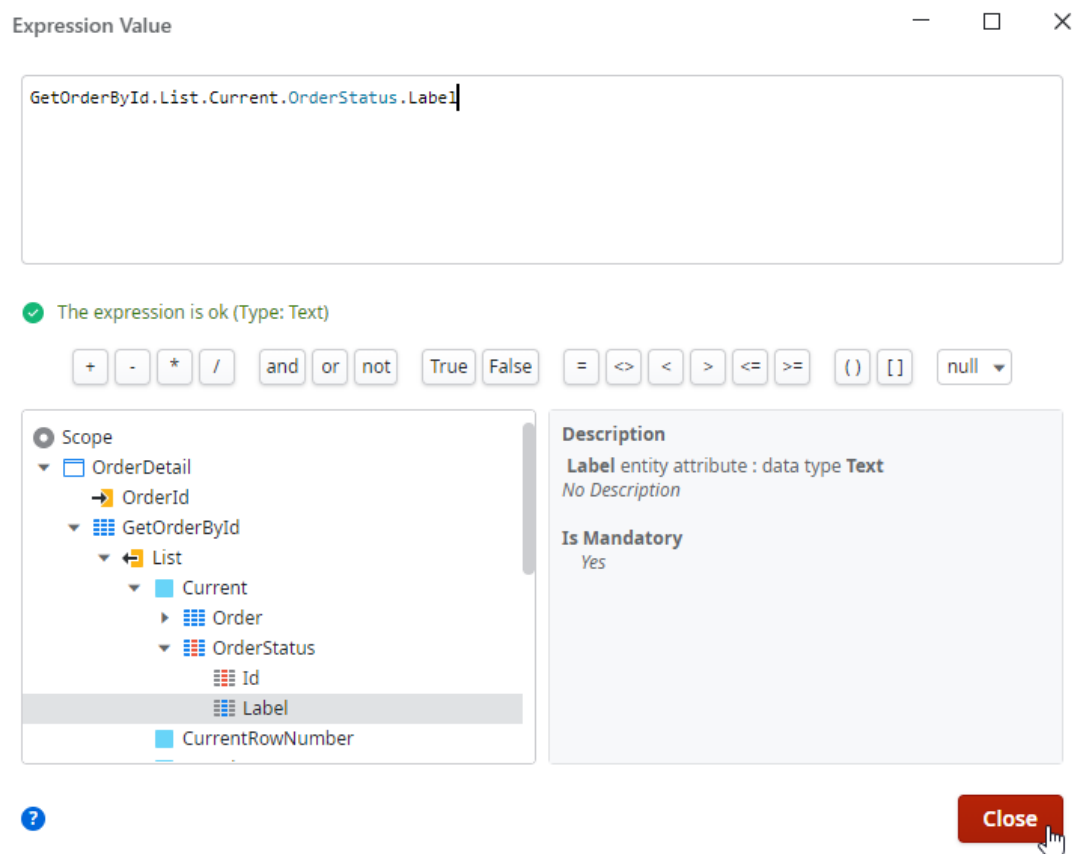


- 7) You're done in the Aggregate, so you can close it and open again the OrderDetail Screen (if it's not already).

- 8) Drag an **Expression** from the Toolbox and drop it **inside the Tag**.



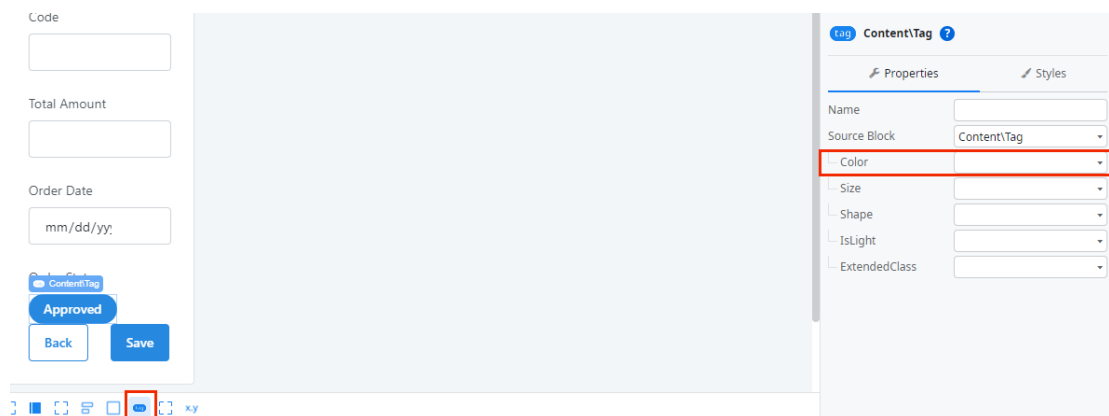
- 9) Select the OrderStatus Label inside the expression using `GetOrderById.List.Current.OrderStatus.Label`.



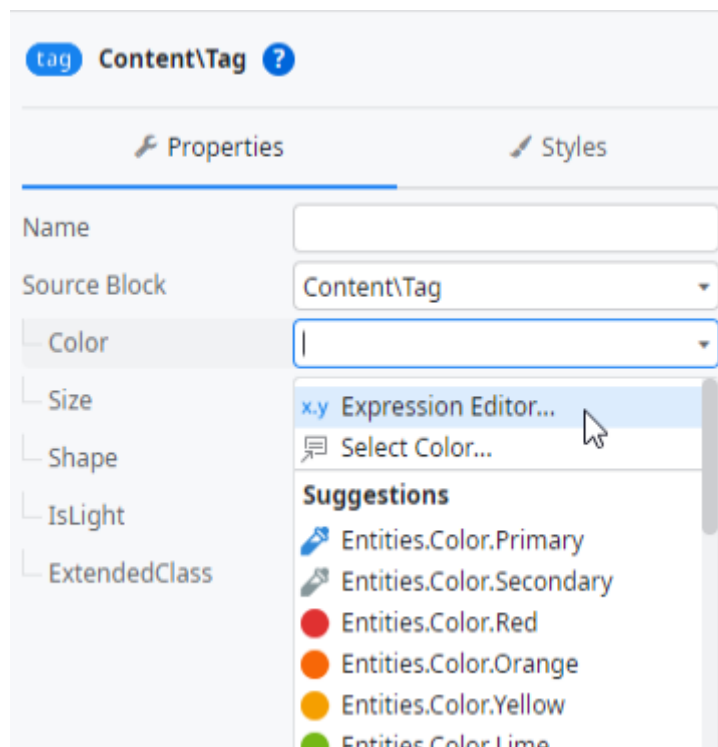
And now you have the status of the order being displayed inside the Label. You just need to now add the color code, just like you have on the Orders Screen.

## Implementing the Color Code and Hiding the Status

- 1) Click on the **Tag** Widget, and make sure to select the actual **Tag** Widget, like the picture highlights. Locate on the right sidebar the **Color** property.



- 2) Expand the dropdown of the **Color** property and select *Expression Editor....*



- 3) Add the following condition to the dialog open:

```
If(GetOrderById.List.Current.Order.OrderStatusId =
Entities.OrderStatus.Draft, Entities.Color.Primary,
If(GetOrderById.List.Current.Order.OrderStatusId =
Entities.OrderStatus.Submitted,Entities.Color.Yellow,
If(GetOrderById.List.Current.Order.OrderStatusId =
Entities.OrderStatus.Approved,Entities.Color.Lime,Entities.Color.Red)))
```

Content\Tag Color Value

— □ ×

```
If(GetOrderById.List.Current.Order.OrderStatusId = Entities.OrderStatus.Draft, Entities.Color.
Primary,
If(GetOrderById.List.Current.Order.OrderStatusId = Entities.OrderStatus.Submitted,Entities.Color.
Yellow,
If(GetOrderById.List.Current.Order.OrderStatusId = Entities.OrderStatus.Approved,Entities.Color.Lime,
Entities.Color.Red)))
```

✓ The expression is ok (Type: Color Identifier)

+ - \* / and or not True False = <> < > <= >= ( ) [ ] null ▾

Scope

- OrderDetail
  - OrderId
  - GetOrderById
  - Widgets
- Entities
- Client
- User Functions
- Built-in Functions
- Resources
- Scripts

Description

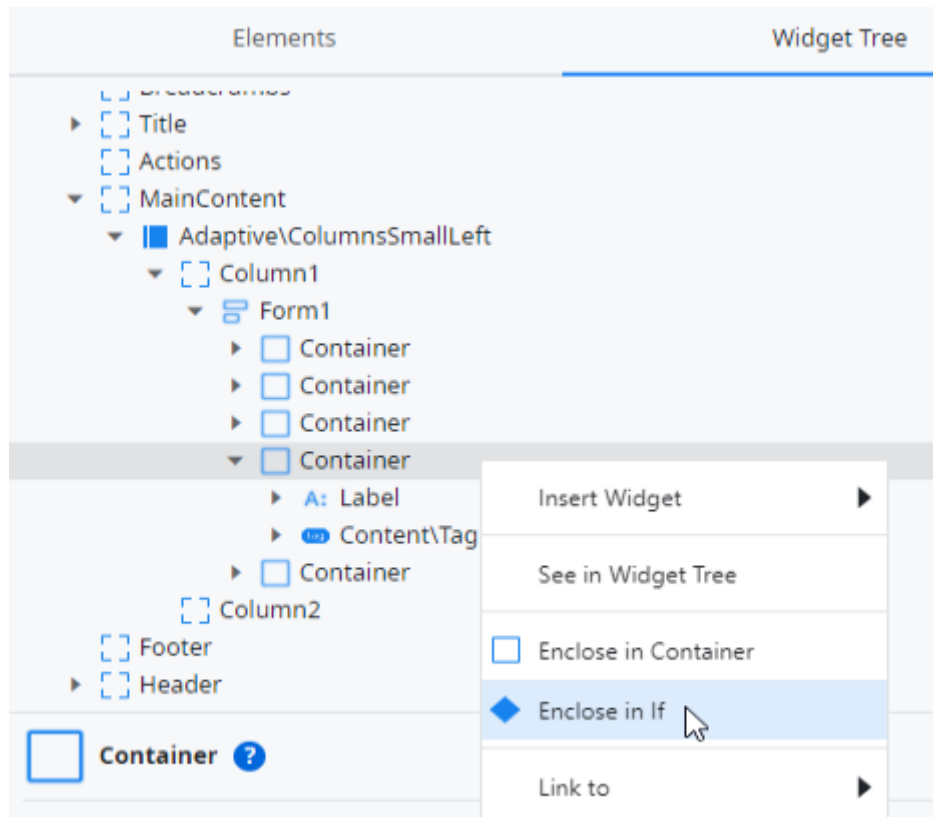


Close

Remember this expression from the first tutorial? It's exactly the same and it's also going to be used to add some color code on this tag.

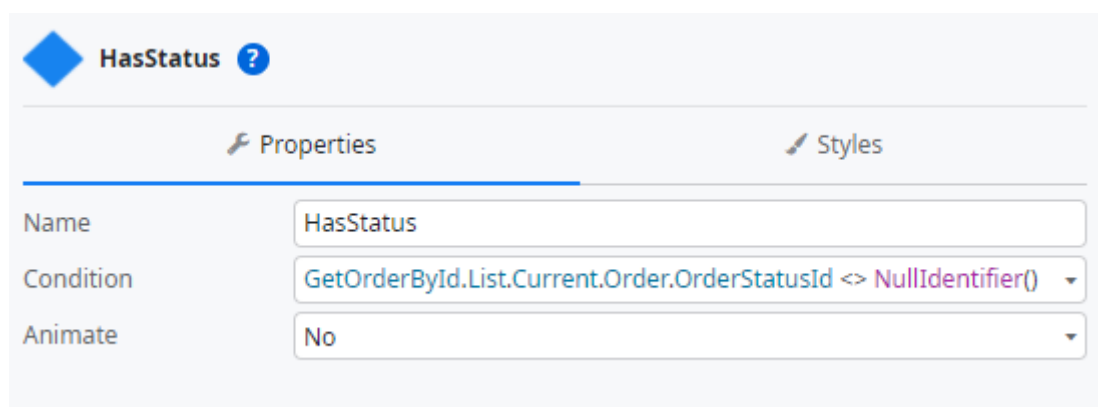
Now, the Order Status should only be visible for Orders that already exist. For a new order, the user should not see the status of the order. You will use an If to create this condition.

- 4) Right-click the **Container** that encloses the Order Status label and Tag in the Form, and select **Enclose in If**.



- 5) Name the If *HasStatus* and set the **Condition** to:

```
GetOrderById.List.Current.Order.OrderStatusId <> NullIdentifier()
```

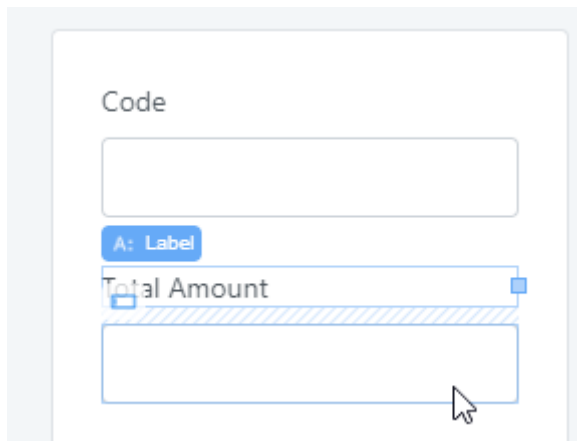


If the order has a status, then it exists and the tag should be displayed. Otherwise, it will stay hidden.

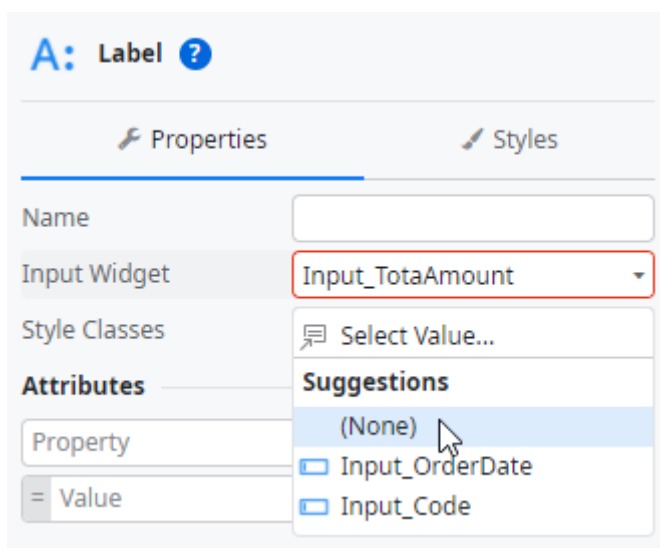
## Adjusting the Total Amount

Now let's do something similar for the Total Amount field. So far, the Total Amount is an input field in the Form. But the total amount of the order should be **calculated** depending on the type and number of products added to the order.

- 1) Delete the Total Amount **Input** field.

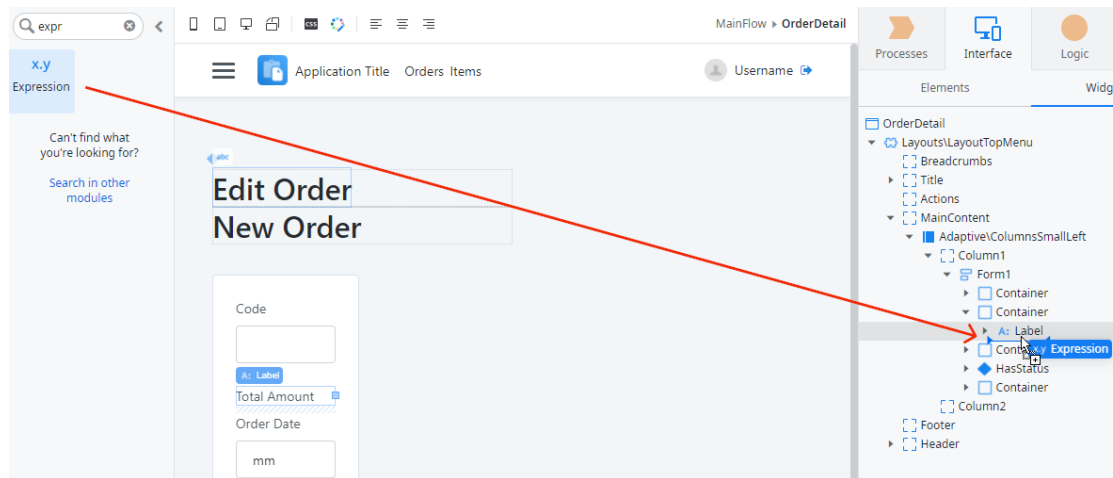


- 2) Change the **Input Widget** property to *(None)*.

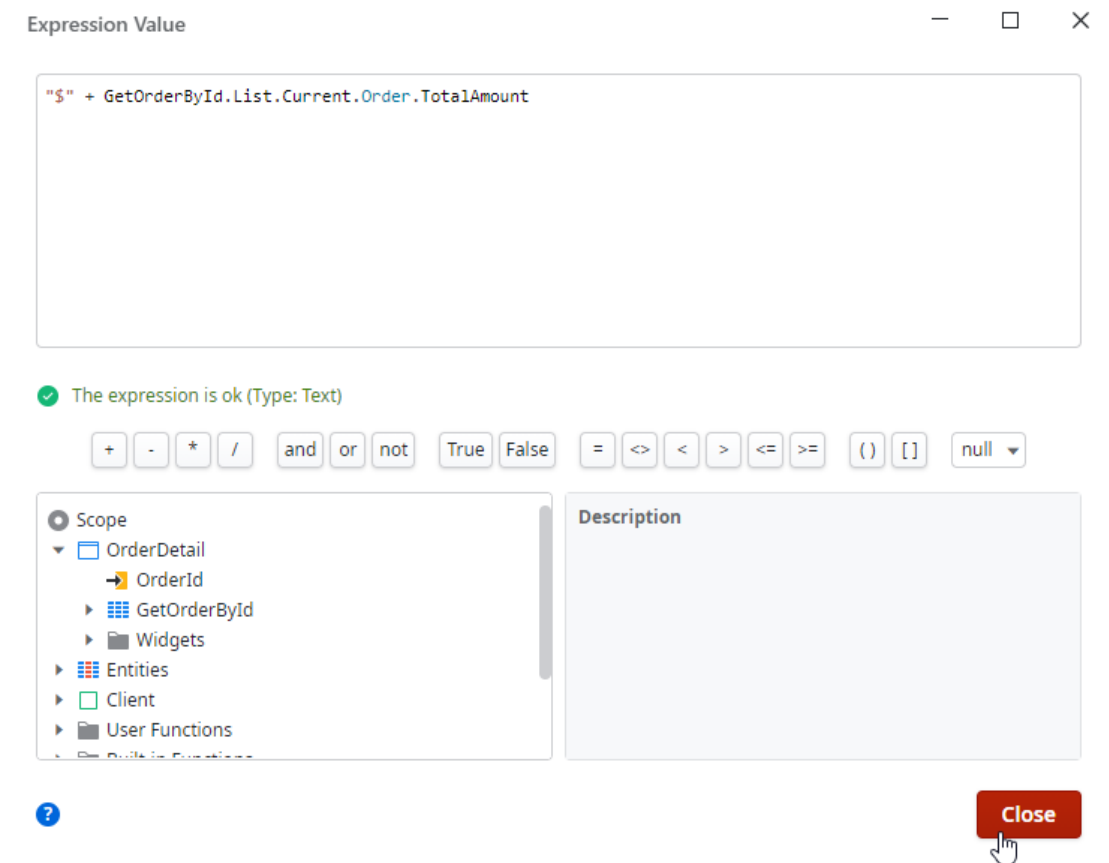




- 3) Drag an **Expression** from the Toolbox and drop it in the place of the Input that you just deleted it.



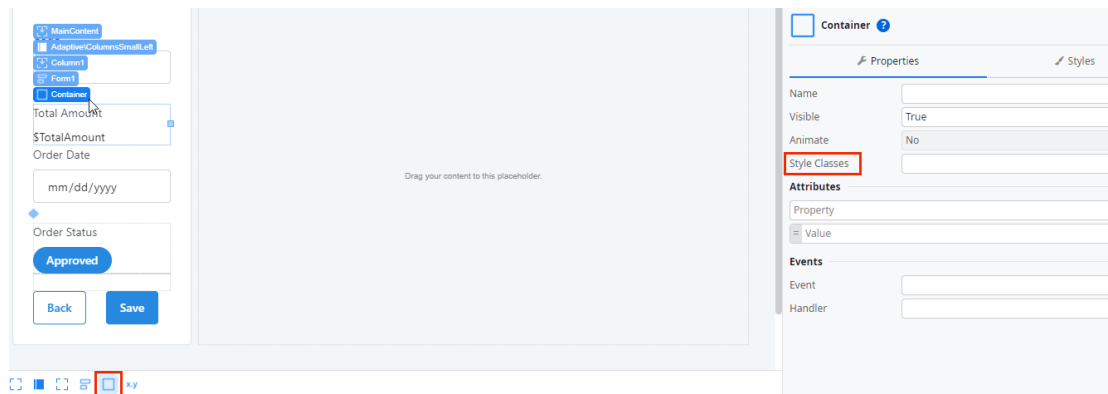
- 4) Type "\$" + GetOrderById.List.Current.Order.TotalAmount in the Expression Value Dialog. Close the dialog when you are done.



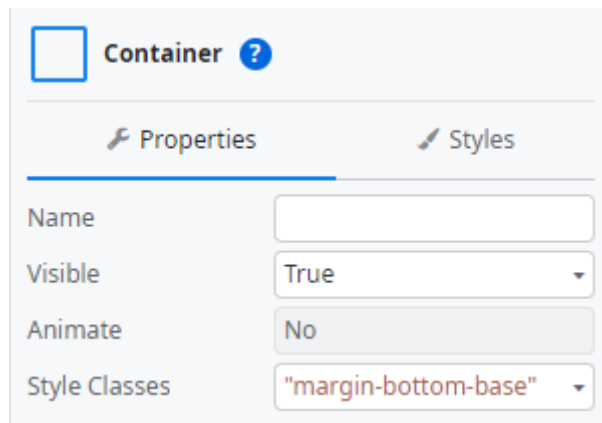
This will display the total amount of the order, with the dollar sign before it.

- 5) Select the Container around the Total Amount Label and the new Expression,

just like it shows in the image. Locate the **Style Classes** property on the right sidebar.



- 6) Add the class *"margin-bottom-base"* to the **Style Classes** property. This will add some space in the bottom of the container.



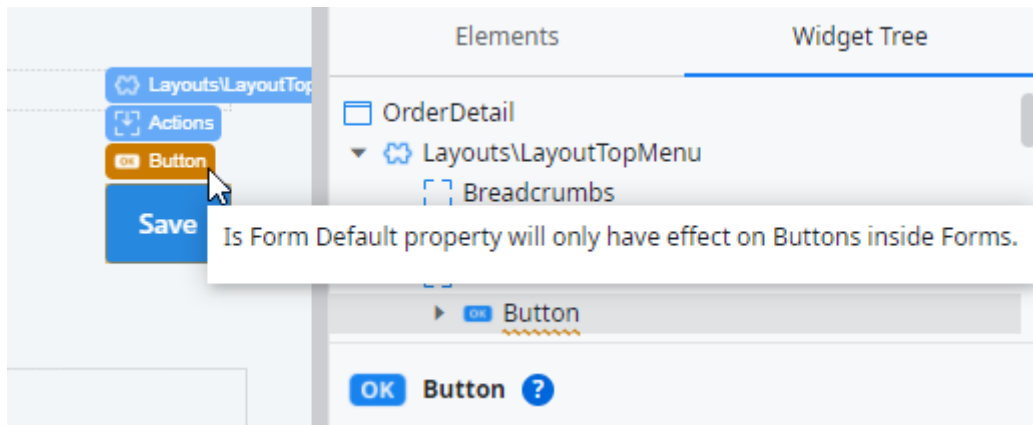
Your Form should look like this:

## Save Button and Back to Orders

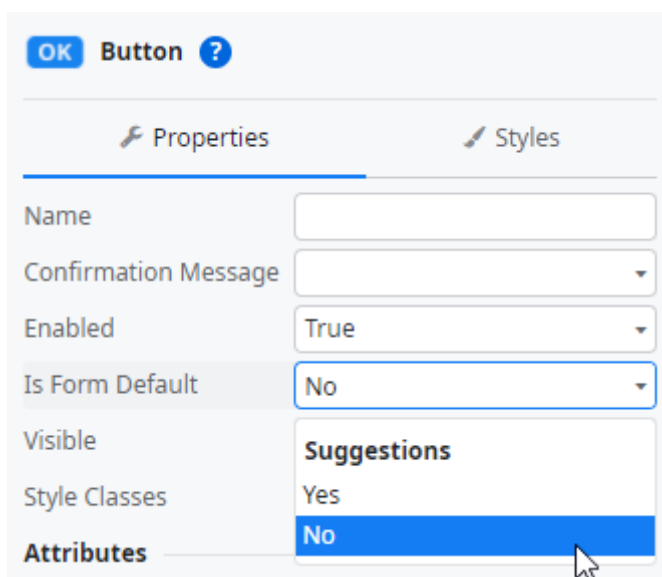
Now it is time to reposition the Save button to the top right corner of the Screen, and also add a link so users can go back to the Orders Screen.

- 1) Drag the Save button that is inside the Form and drop it on the area of the Screen on the top right (called the Actions placeholder).

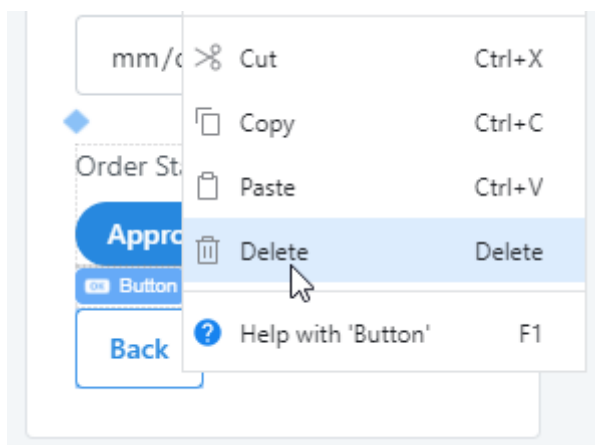
**Note:** A Warning will appear. If you hover the mouse on the element where the warning is, you can see the warning message. Let's fix it.



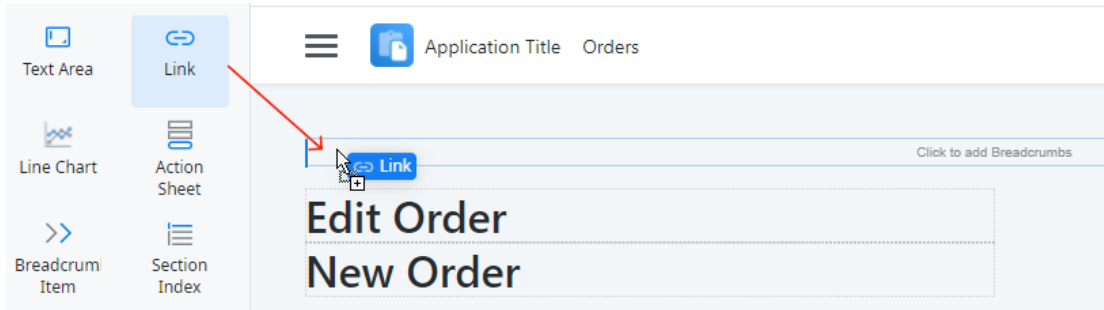
- 2) Click on the **Save** Button and change its **Is Form Default** property to *No*. The warning should disappear.



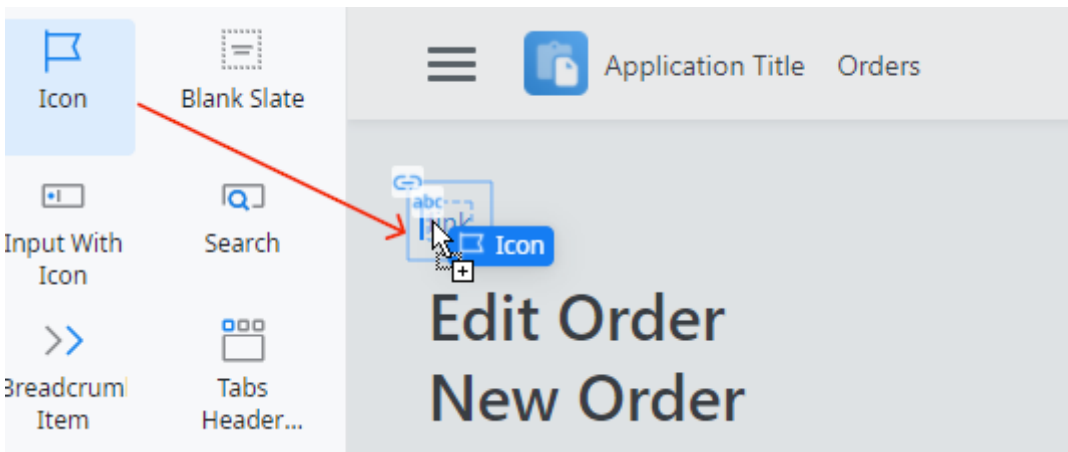
- 3) Click on the **Back** Button inside the Form and delete it.



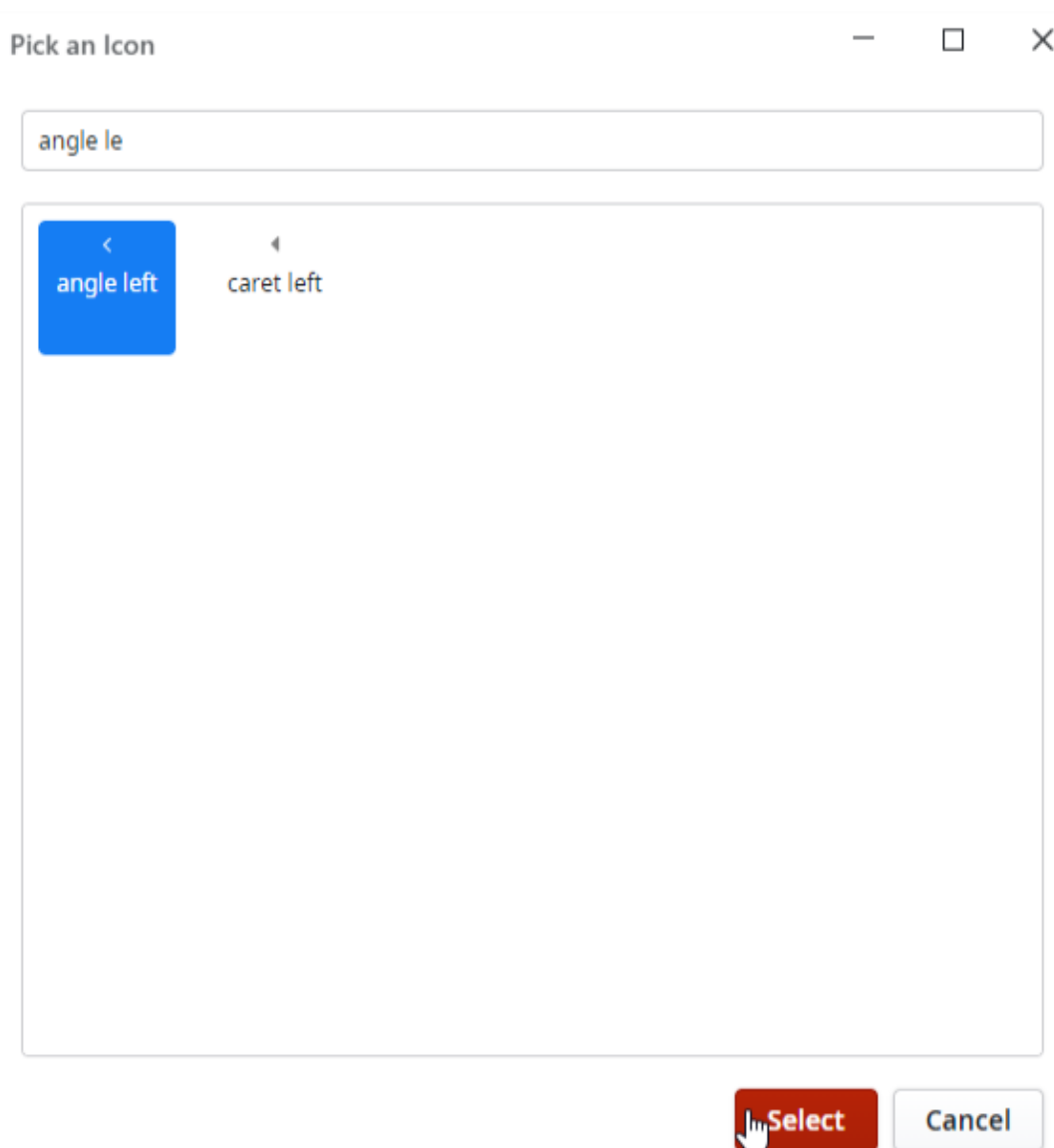
- 4) Delete the **Container** where the Buttons were.
- 5) Drag a **Link** from the Toolbox and drop it on the Breadcrumbs placeholder, right above the title.



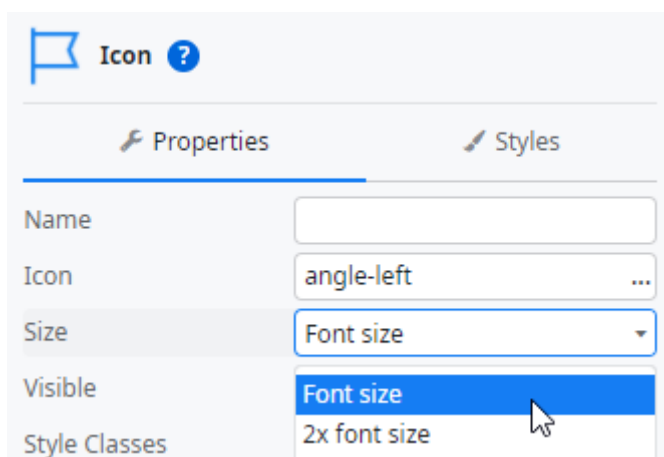
- 6) Drag an **Icon** from the Toolbox and drop it inside the Link, to the left of the text saying *link*.



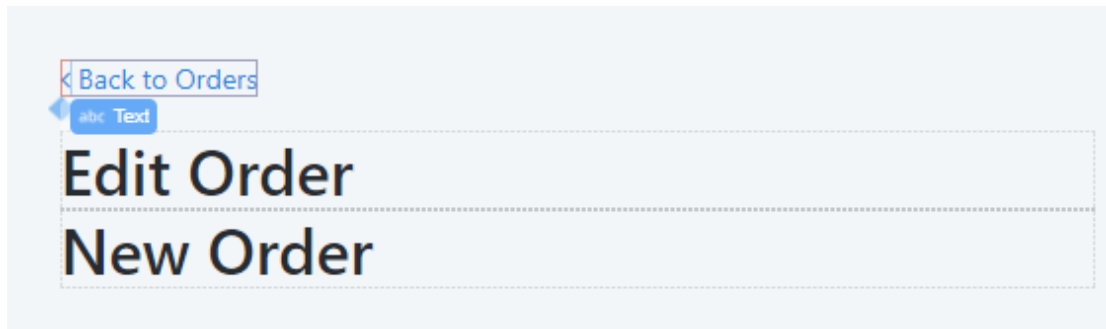
7) Pick the *angle left* icon and click **Select**.



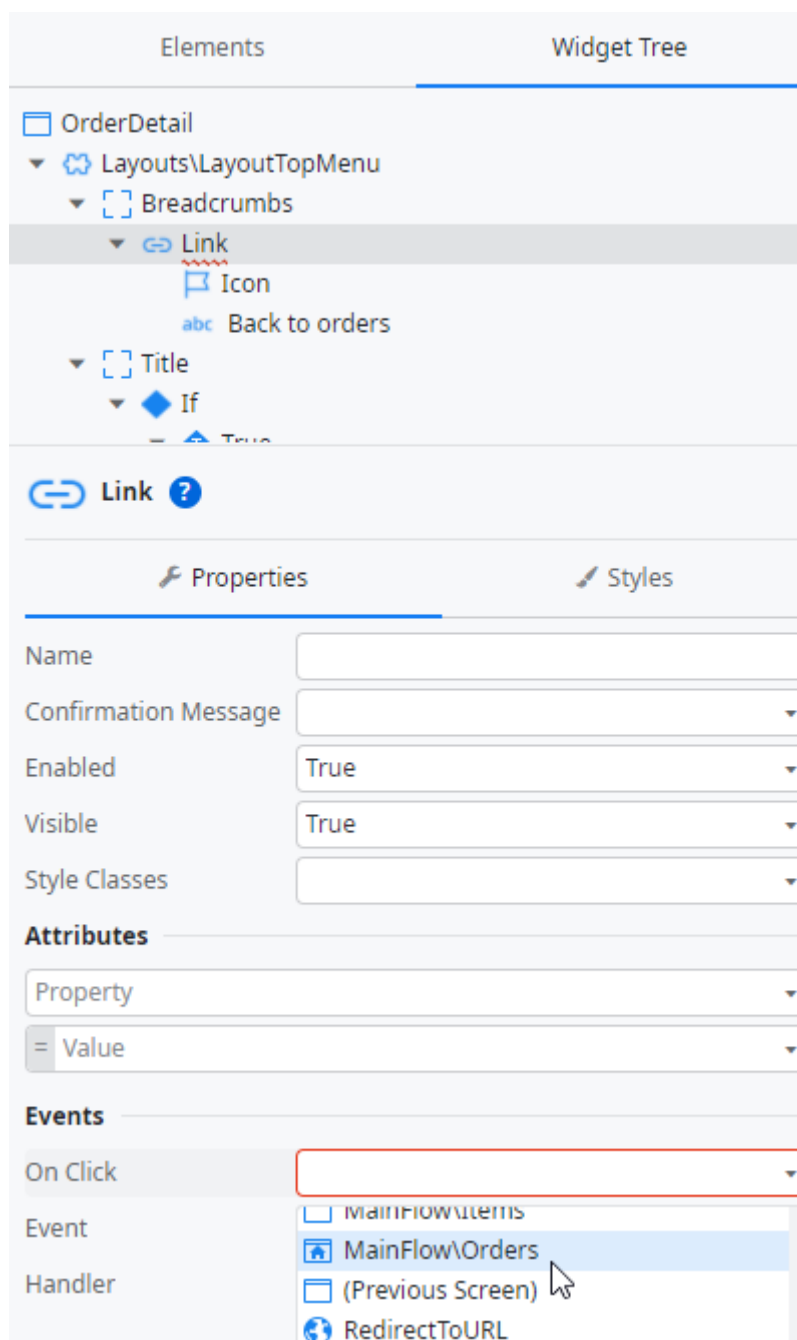
8) Change the **Size** property of the Icon to **Font size**.



- 9) Type *Back to orders* right next to the icon, to replace the *link* text.



- 10) Select the Link (you can use the widget tree to help you), then change the **OnClick** Event to the **Orders** Screen.



## Changing the Save Action

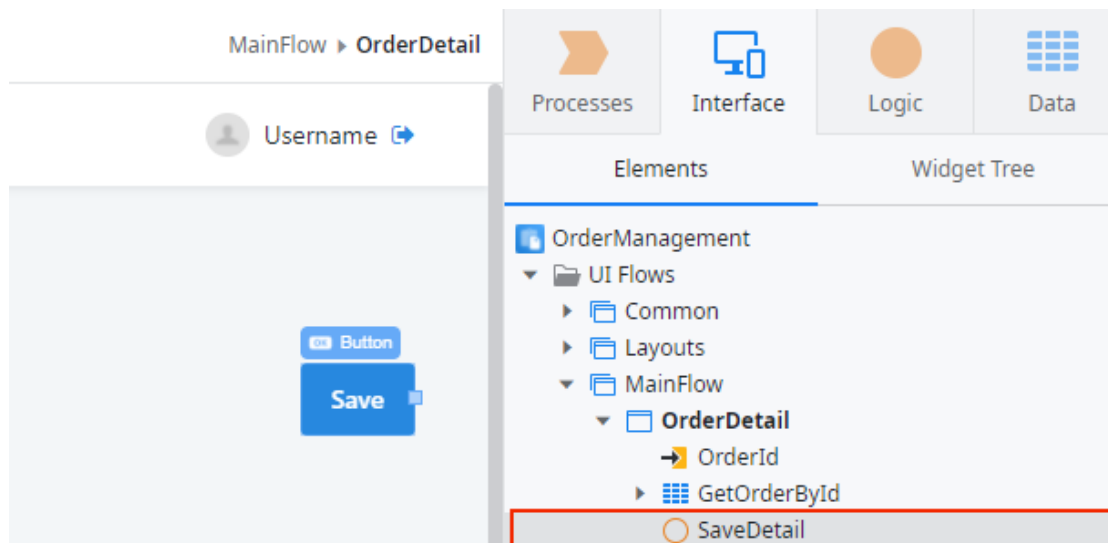
The changes made in the Form will have an impact on the process of saving the Order.

So far, you were using the Save Action that was automatically created when the Screen was created, but you didn't change anything on it. However, now you have two fields that are not chosen by the user (the amount and the status). So, we have to make sure

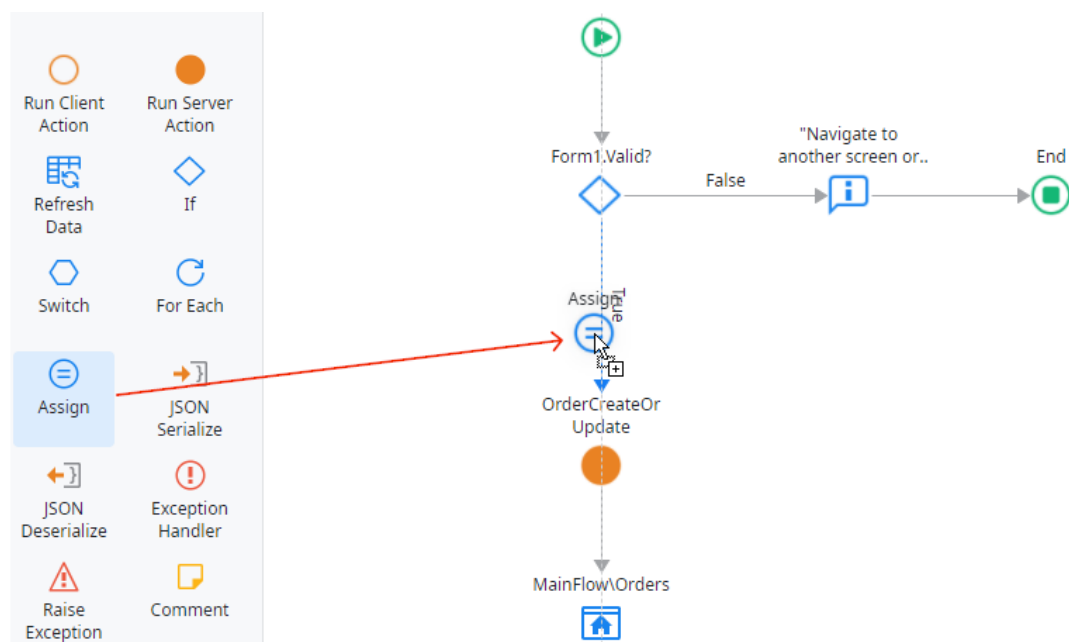


that in the save logic, before adding the order to the database, the total amount and status is properly set, so that the information is properly added to the database.

- 1) Open the **SaveDetail** Action by double-clicking it.

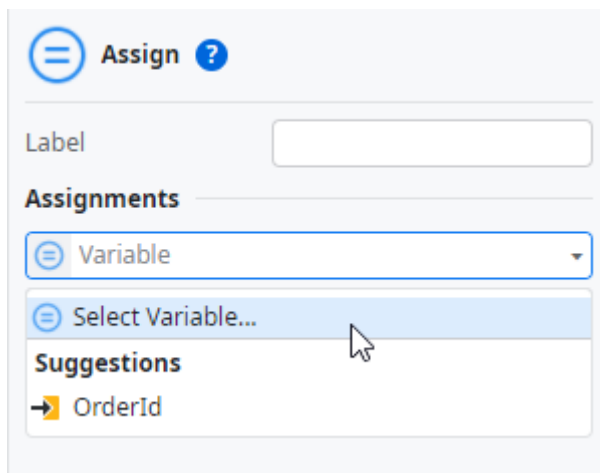


- 2) In the Action flow, drag an Assign element from the Toolbox, and drop it on the flow after the Form validation.

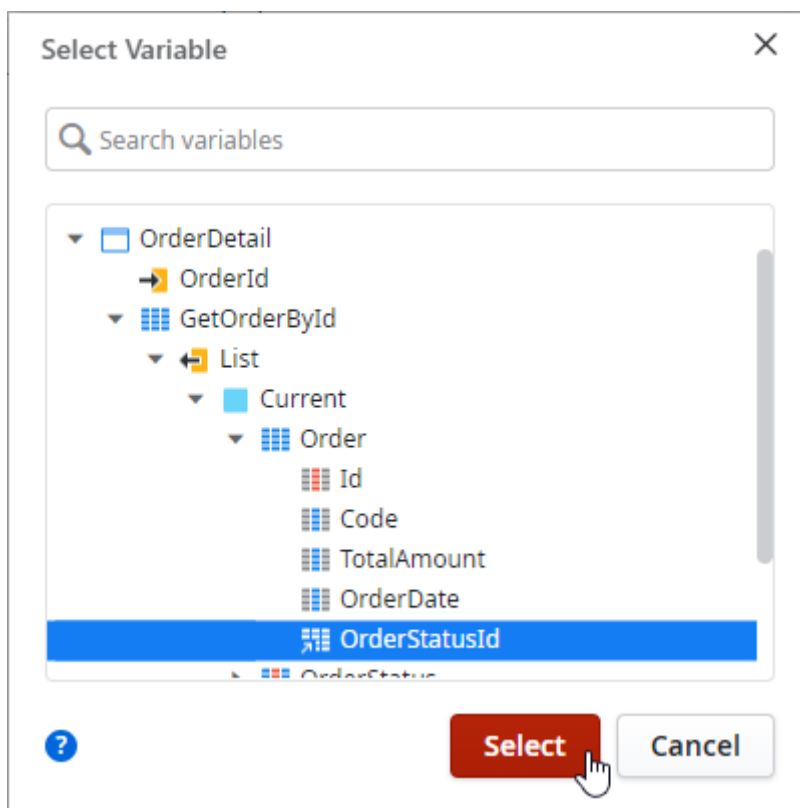


Since we're creating the Order, we want to set its status to Draft.

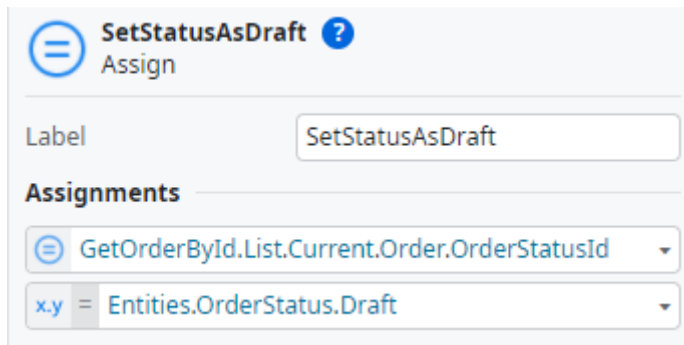
- 3) In the Assign element, expand the Variable dropdown and choose the *Select Variable...* option.



- 4) Select the attribute `GetOrderById.List.Current.Order.OrderStatusId` in the Select Variable dialog. Click on Select at the end.

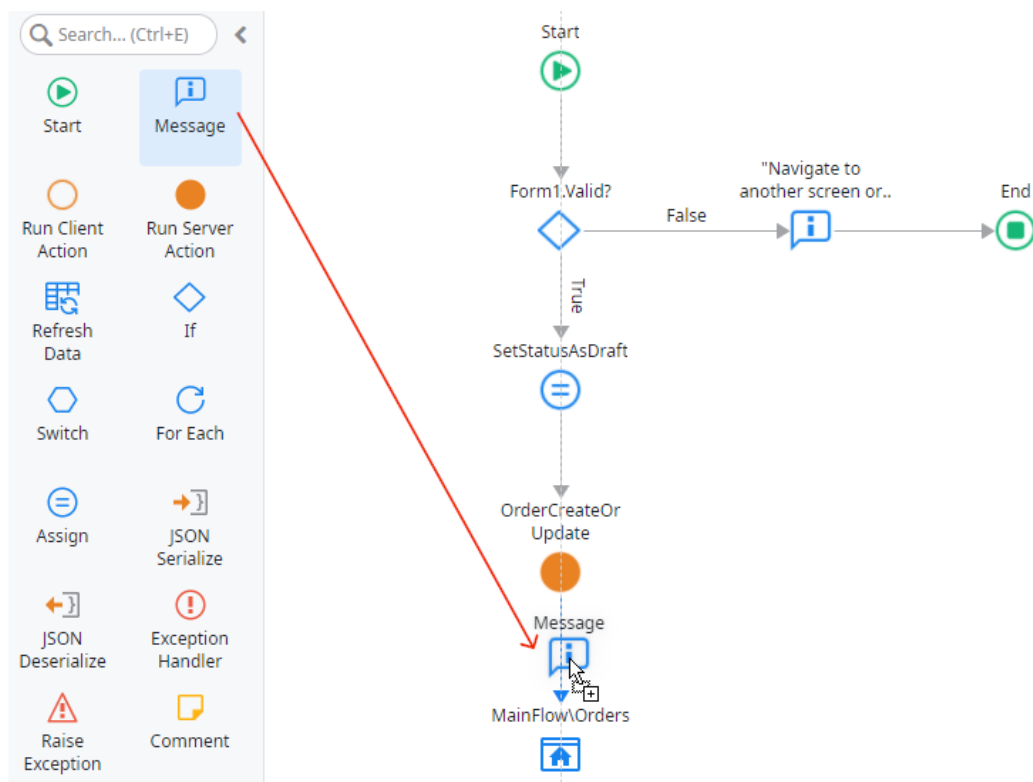


- 5) Assign the `Entities.OrderStatus.Draft` value to the variable selected in the previous step, then change the name of the assign to *SetStatusAsDraft*.



It is also important to display a message so the end user knows if the Order was successfully created. Let's do that.

- 6) Drag a **Message** from the Toolbox and drop it after the OrderCreateOrUpdate Action.



- 7) Set the **Type** property to *Success* and set the **Message** to *"Order successfully created"*.



| "Order successfully created" ? |                                |
|--------------------------------|--------------------------------|
| Message                        |                                |
| Label                          |                                |
| Message                        | "Order successfully created" ▼ |
| Type                           | Success ▼                      |

- 8) Publish the module and open it in the browser.



- 9) Test the changes by clicking on a specific order to open its Detail Screen. You can also create a new order to test it out.

OrderManagement Orders

[< Back to Orders](#)

## Edit Order

Code \*

Total Amount  
\$400

Order Date \*

Order Status

## Wrapping up

Congratulations on finishing this tutorial. 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) [Aggregate](#)
- 3) [Supported Join Types](#)
- 4) [Errors and Warnings](#)
- 5) [Messages](#)

**See you in the next tutorial!**