Using a Wizard - Popup

Table of Contents

knowledge@outsystems.com

www.outsystems.com

Outline

In this tutorial, you will continue to extend the Order Management application and finish the wizard with the workflow to approve or reject an order. At this point, you're missing the UI and logic to add a new item to an order. So, you will create the functionality that will allow you to add a new order item and also edit existing items associated with an order.

To achieve that, you will:

- Add a new Popup to the Screen. The Popup will have a Form with the Order Item information such as the item, the price, the quantity and the amount.
- Make sure the Popup allows the user to add a new item to the order and to also edit an existing order item.
- Create the logic necessary to save a new order item in the database.
- Create the logic to control when the Popup opens to add an item to the order, or edit an existing one, and when it closes.
- Create the logic to calculate the total amount of the order depending on the items the order will have.

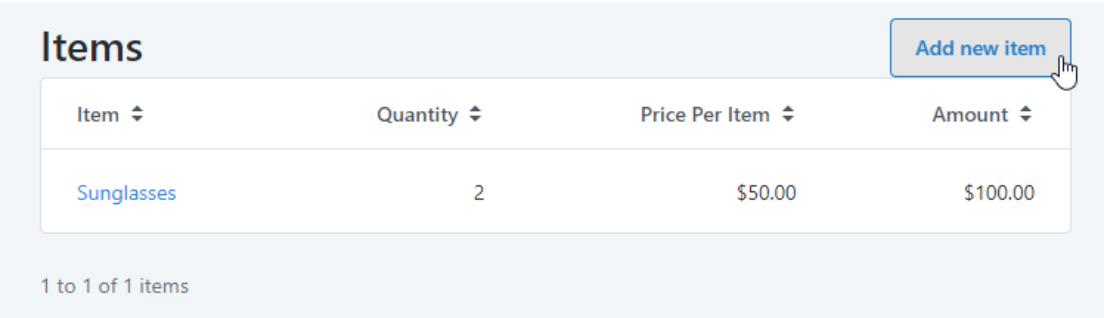
You can implement both of these functionalities with a Popup that allows users to add Items to the Order. In the end, you will be able to open your app in the browser and test order workflow from beginning to end, with items already added to your orders.

Scenario

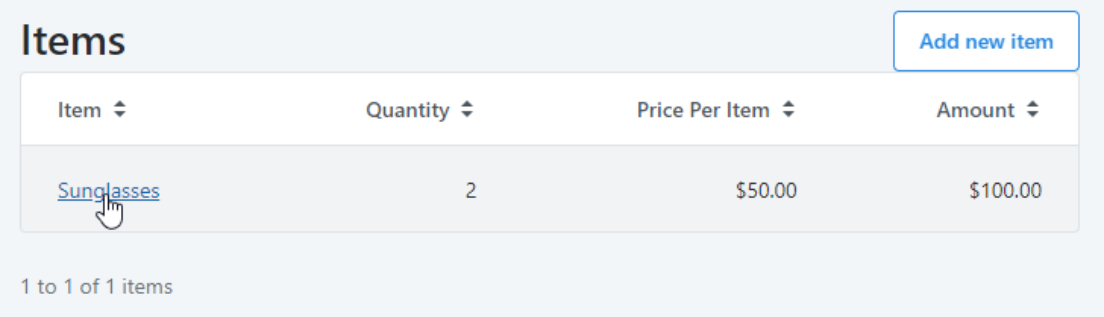
At this point, your Order Management application has the workflow for orders ready and it is already possible for the user to move an order from Draft to Approved or Rejected.

Now, you will close the OrderDetail functionality by allowing the user to add items to orders, and to edit them. The OrderDetail Screen will have two scenarios then:

- Add new item: the user clicks on the Button that allows adding Items to the Order being edited.

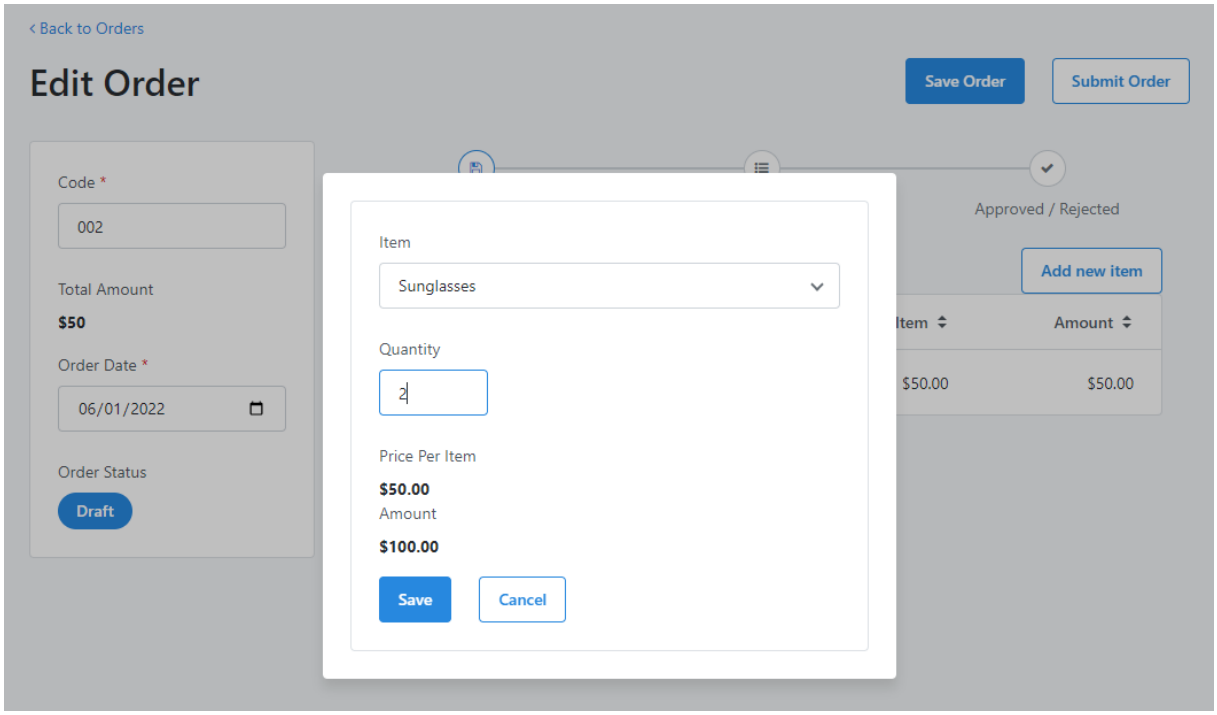


- Edit an existing Order Item: the user clicks on the name of the product and can edit the quantity of the Item that was already added to the Order.



These two scenarios will open a Popup. A Popup is a visual element that allows the creation of sophisticated interactions without having to open a new Screen. It is a part

of the Screen where it is implemented and can be opened and closed depending on circumstances defined by you.



So, at the end of this tutorial, you will have the OrderDetail Screen looking like the picture above, with the popup opening when the user tries to add a new order item or editing an existing order item. You will use some math and logic to calculate the Amount of an Item added to the Order, as well as the Total Amount of the Order.

How-to

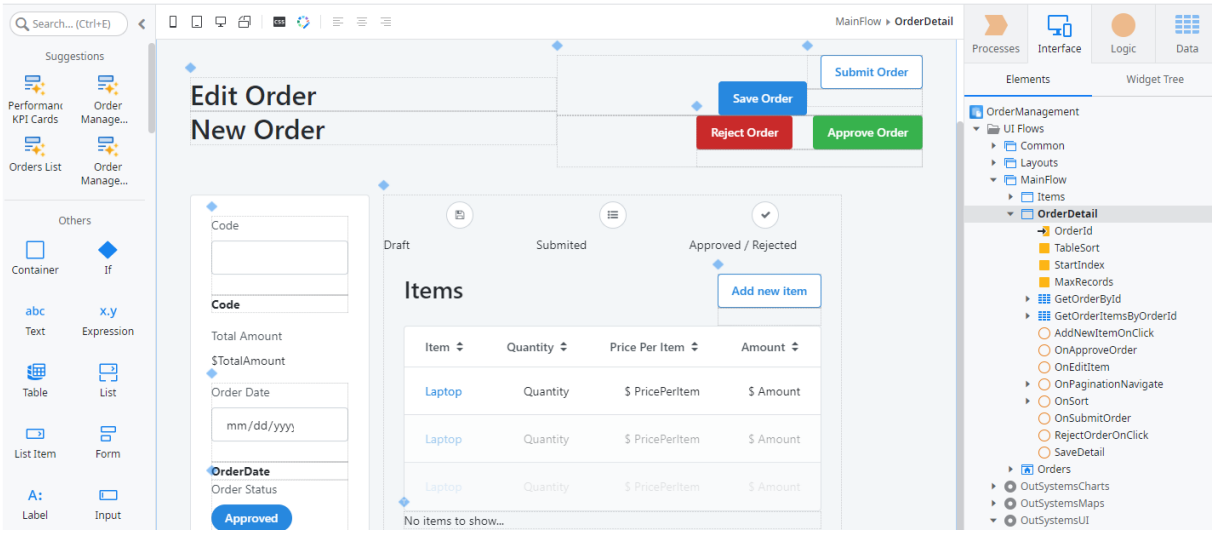
In this section, we'll show a thorough step-by-step description of how to implement the scenario described in the previous section.

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 all of 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.

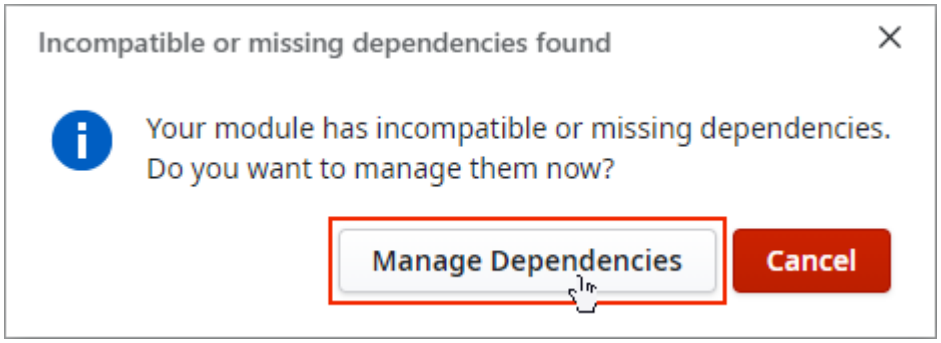


If you did not create the app, you can use the quickstart available in the resources and continue from that point.

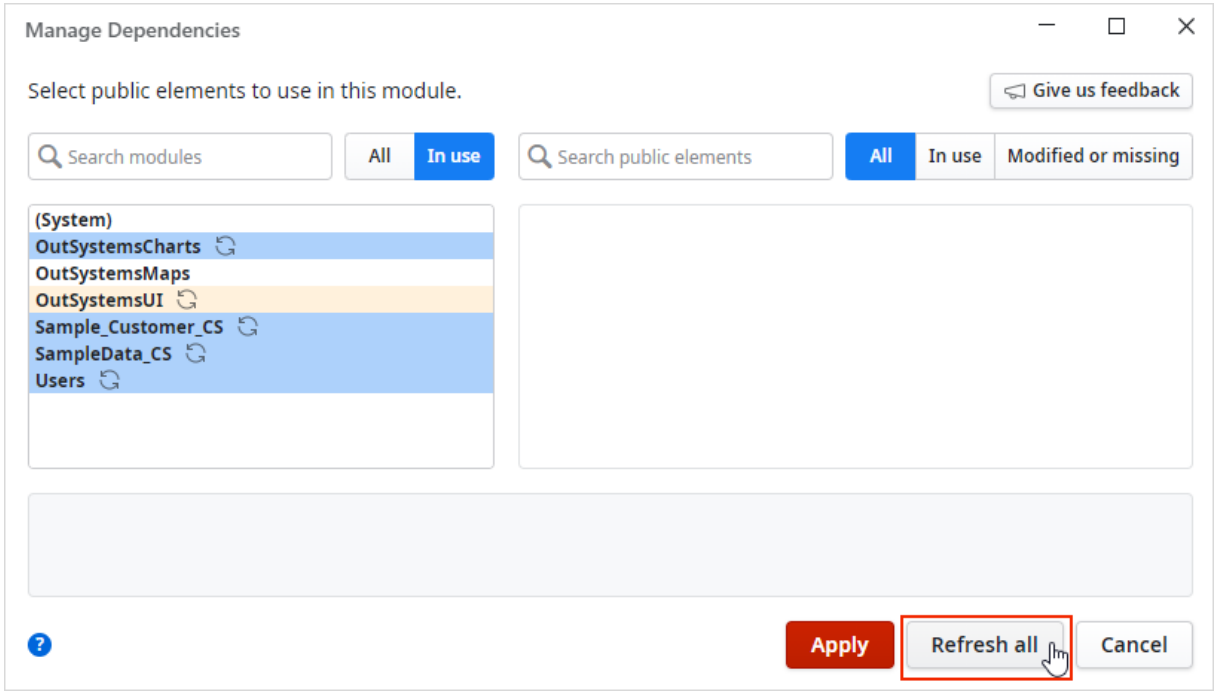
Outdated Dependencies

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.



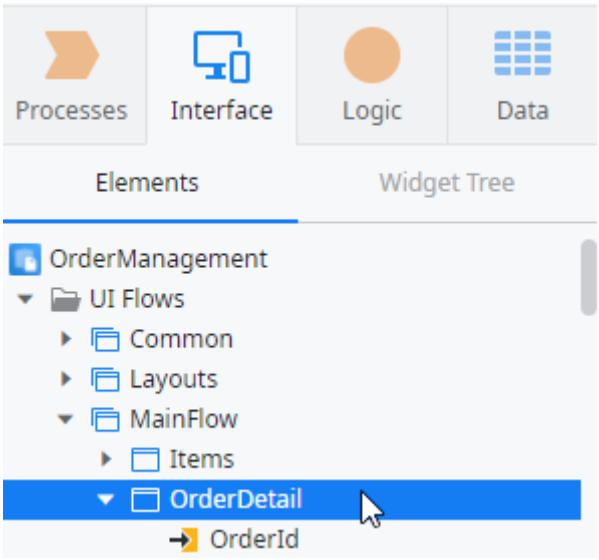
Now publish the module to update the project

Creating a Popup

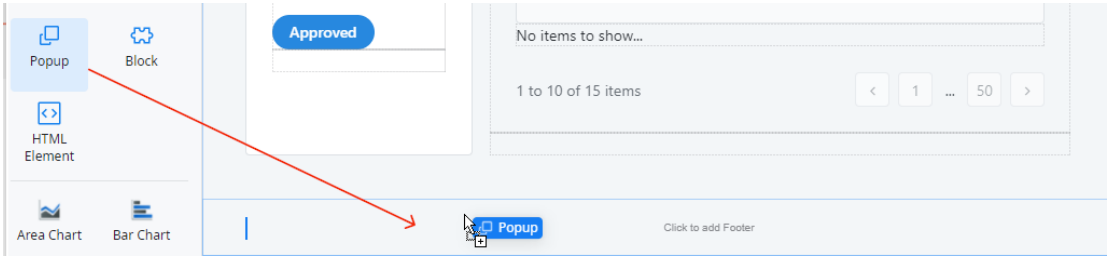
OutSystems already provides an Interface element that helps to create the popup and use it in your Screens. You just need to define its UI, and how and when the popup opens or closes. Since the popup will be used to add new items to orders, let's start by creating a Form.

Adding a Form to Add Items

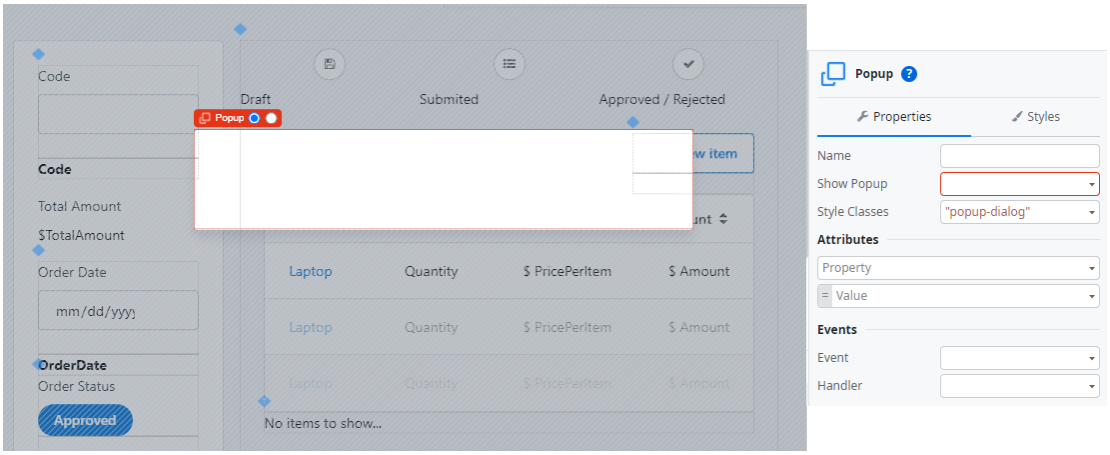
1) If it's not open, double-click the **OrderDetail** Screen to open it.



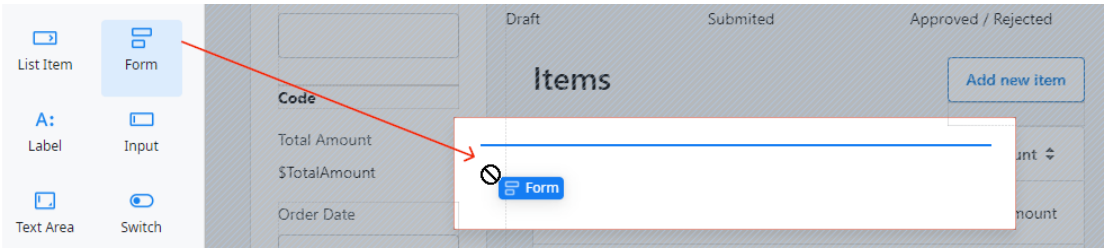
2) Search for a **Popup** in the Toolbox (left sidebar). Drag and drop it into the Footer placeholder, at the bottom of the Screen.



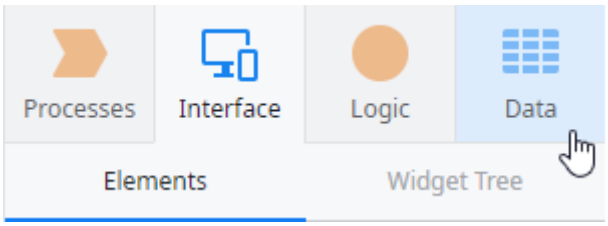
An error appears because of the mandatory property **Show Popup**. This is the property that controls when the popup opens. You will handle this later.



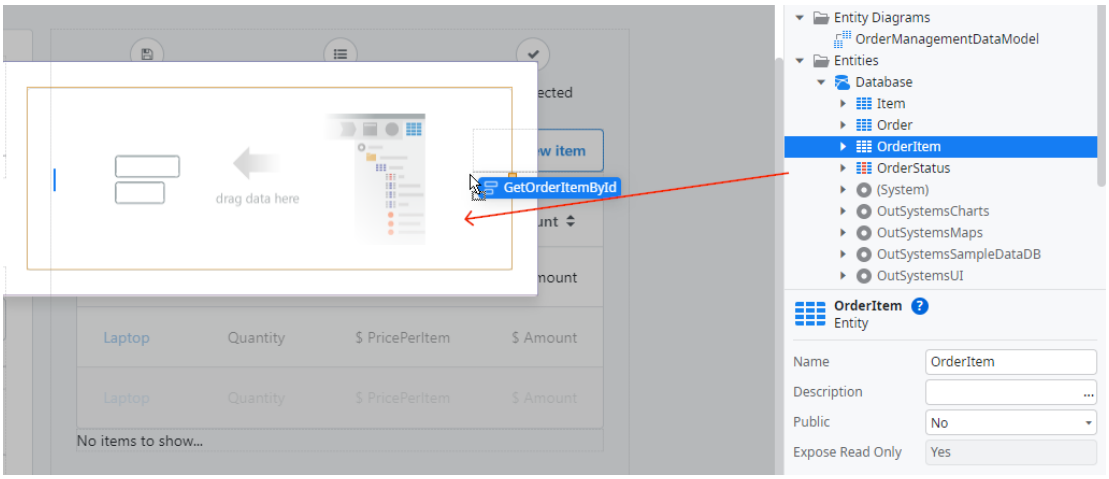
3) Drag a **Form** from the Toolbox and drop it inside the Popup.



4) Switch to the **Data** tab on the top right corner of Service Studio.

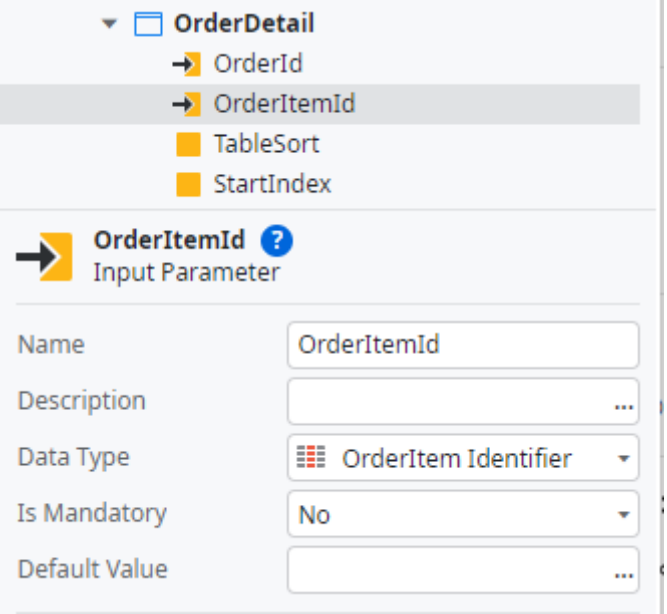


5) Drag and drop the **OrderItem** Entity to the Form.

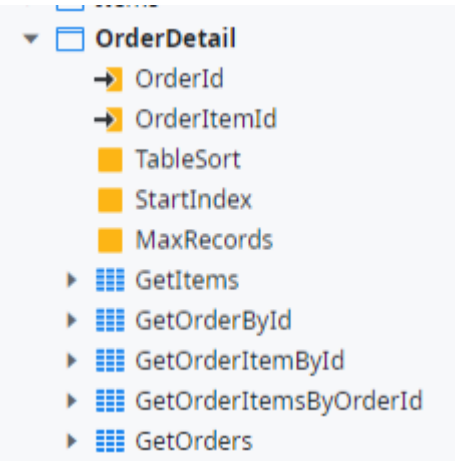


A Form with the fields of an OrderItem is automatically created in the Popup. An Input Parameter named OrderItemId is also created on the Screen, because

since we have a Form, OutSystems understands that at some point we need to identify the OrderItem being edited.



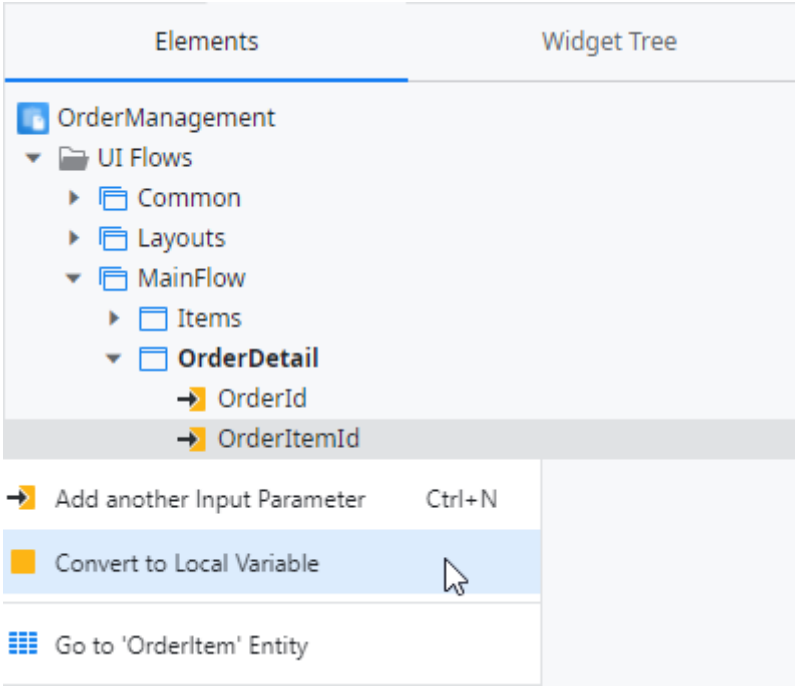
OutSystems also created the **GetOrderItemById** Aggregate, which is what fetches the information that will support the Form. The GetItems and GetOrders Aggregates were also created to populate the dropdowns in the Form.



The Form was created with all the fields of an OrderItem. However, not all of them should be visible or editable. For instance, the user should not choose the Order. Also,

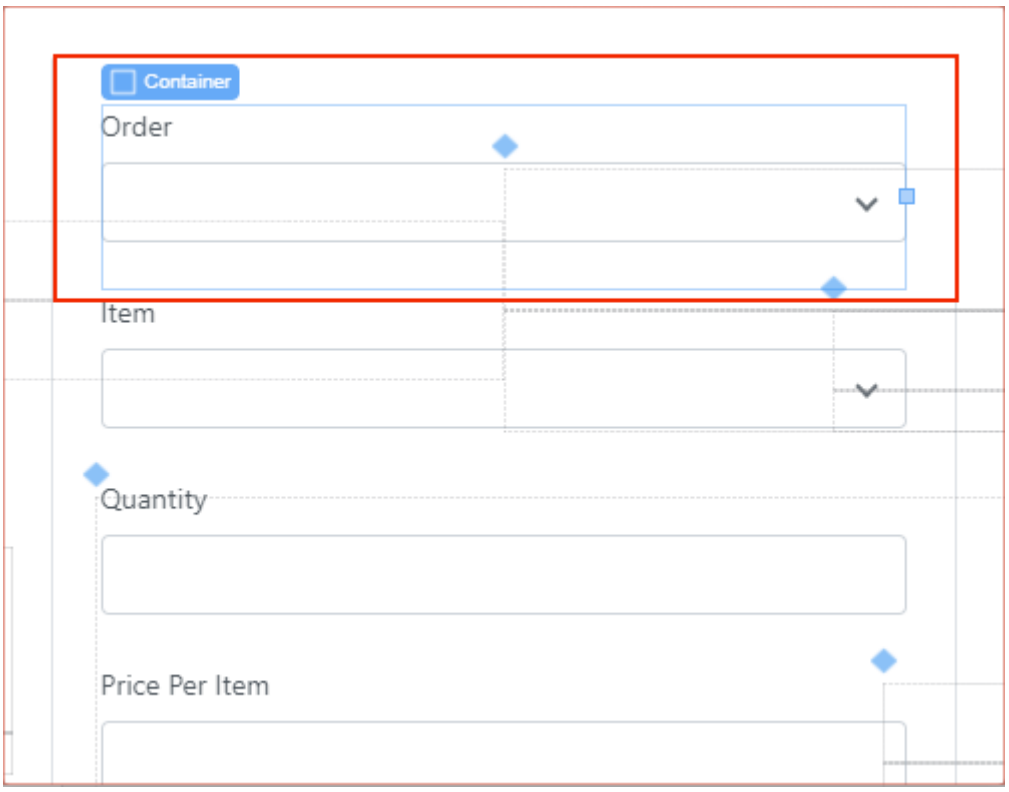
you will adjust a bit what was created on the Screen to adjust the needs of the this scenario.

1) Back in the Interface tab, right-click the OrderItemId Input Parameter and select **Convert to Local Variable**.

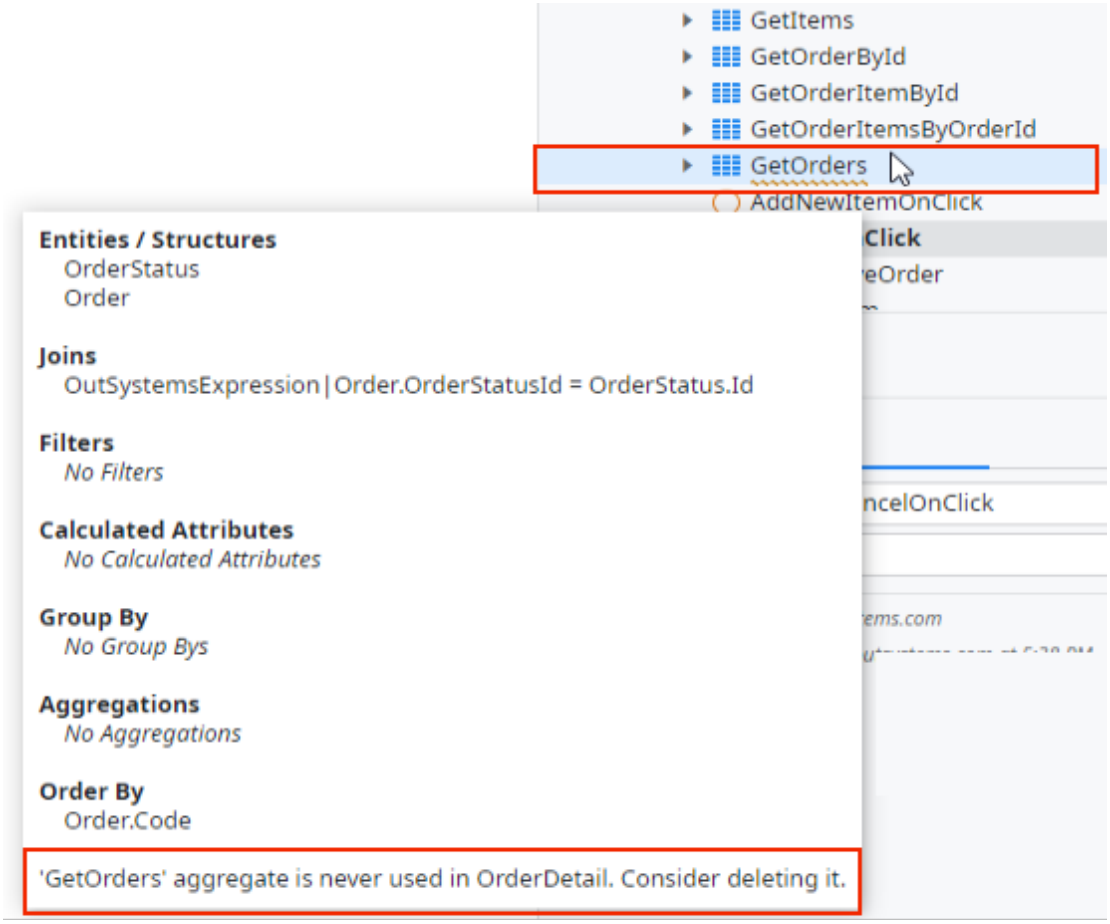


What you just did was convert the Input Parameter into a Local Variable. You will need this variable later to save the OrderItem being edited. But it does **not** need to be an input parameter of the Screen.

2) In the Popup, delete the Container that contains the Order field.

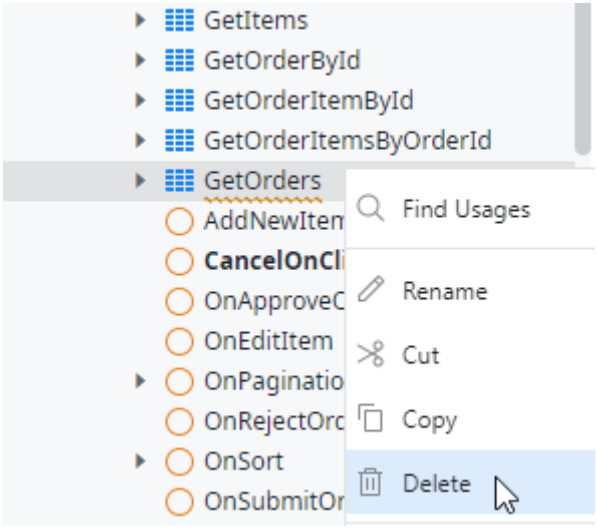


3) Hover the mouse on the Aggregate **GetOrders**.

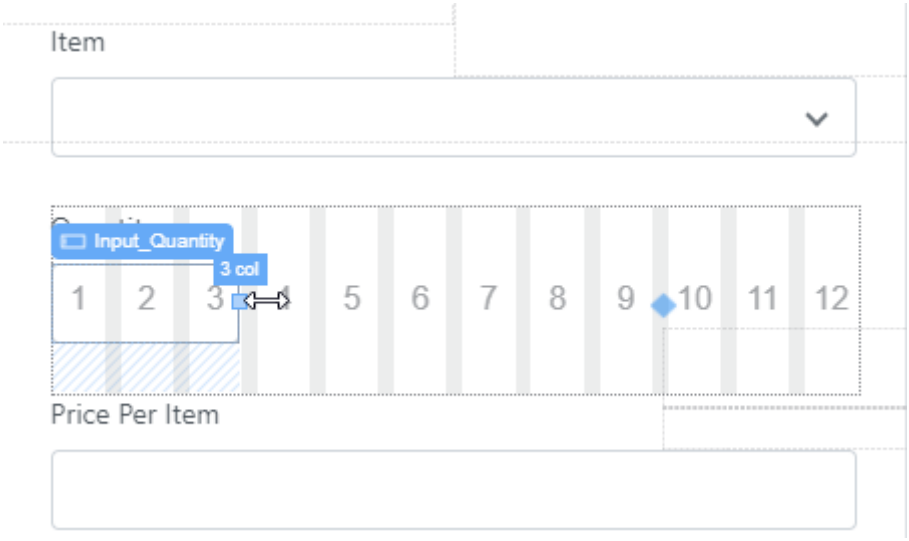


In OutSystems, a warning is a potential problem but does not prevent the deployment of a module to the server. However, it's advisable to check the warnings and solve them. In this case, it is warning you that you have an unused Aggregate, since it was only used in the Form field we just deleted.

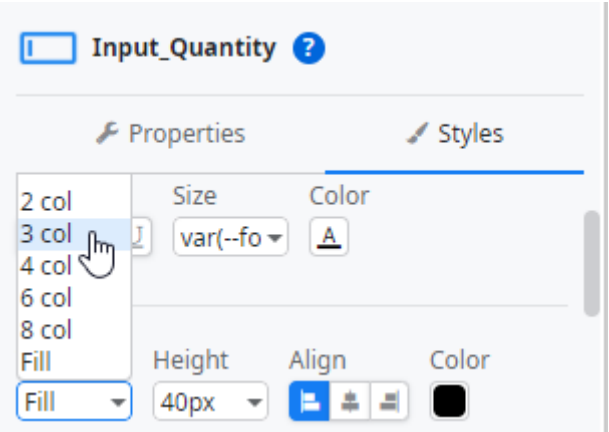
4) Delete the **GetOrders** Aggregate.



5) Back on the Popup, click on the **Quantity** Input Field and change its **Width** to 3 columns. If you click on the right border of the field, you will see an option where you can just drag the field left or right to decrease or increase its width.



You can also modify the number of columns in the style properties, under the Styles tab.



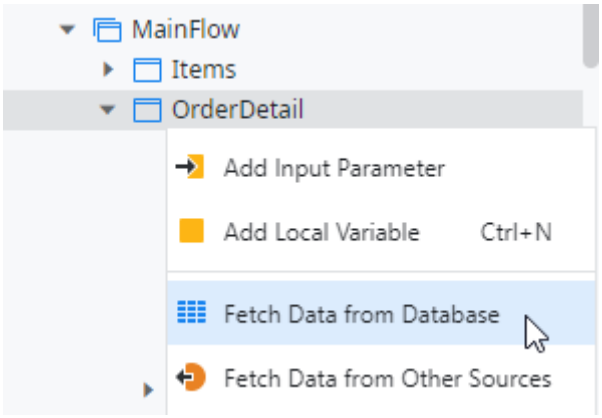
There are more things to adjust. Notice that on our Form, the Item is a dropdown. So, we can choose one item, but nothing stops us to change it for a new one later. So, the choice the user does in that dropdown will influence the price per item and the amount of the OrderItem. Let's work on that!

Fetching Item Information

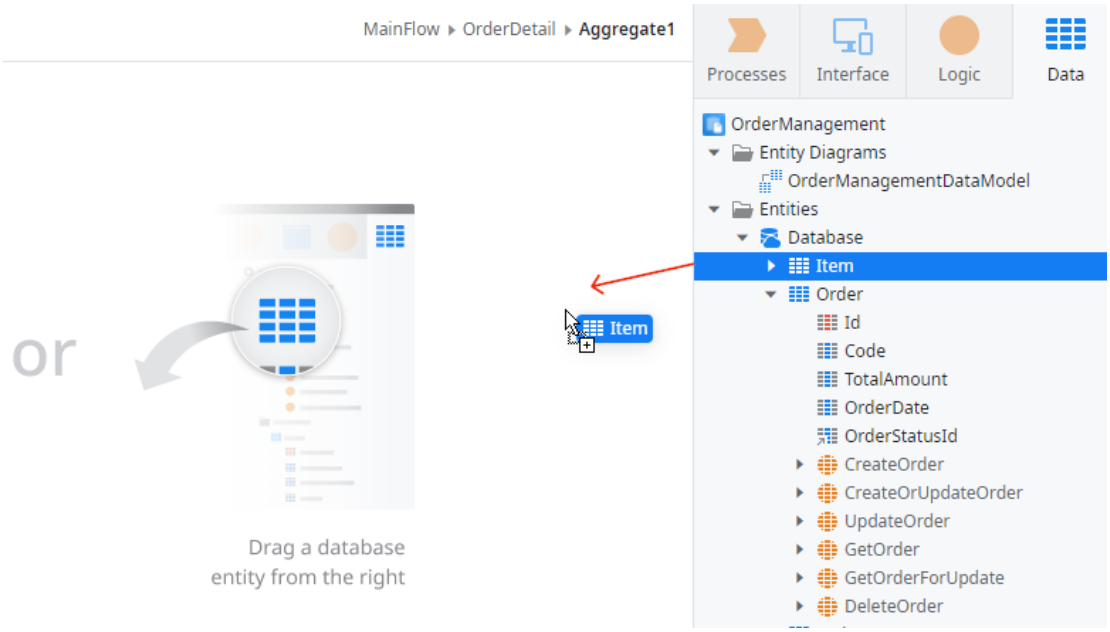
The Popup will use an Aggregate to fetch items and display them in the Form's first dropdown. However, we need to fetch the information of an item, depending on what the user chooses in the dropdown, like its price for instance.

So, you will create a new Aggregate.

1) Go back to the Interface tab, right-click the **OrderDetail** Screen and select **Fetch Data from Database**

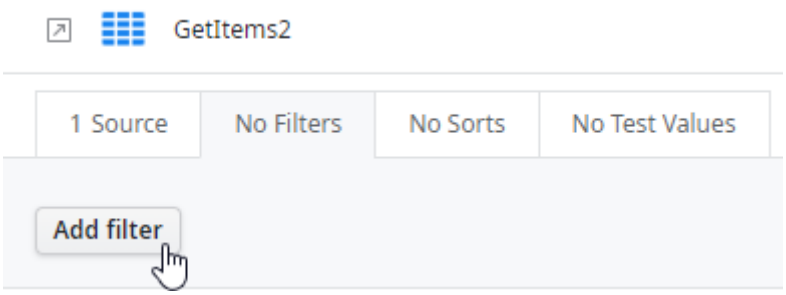


2) Switch to the Data tab. Drag and drop the **Item** Entity to the center of the screen that appears after you created the Aggregate.



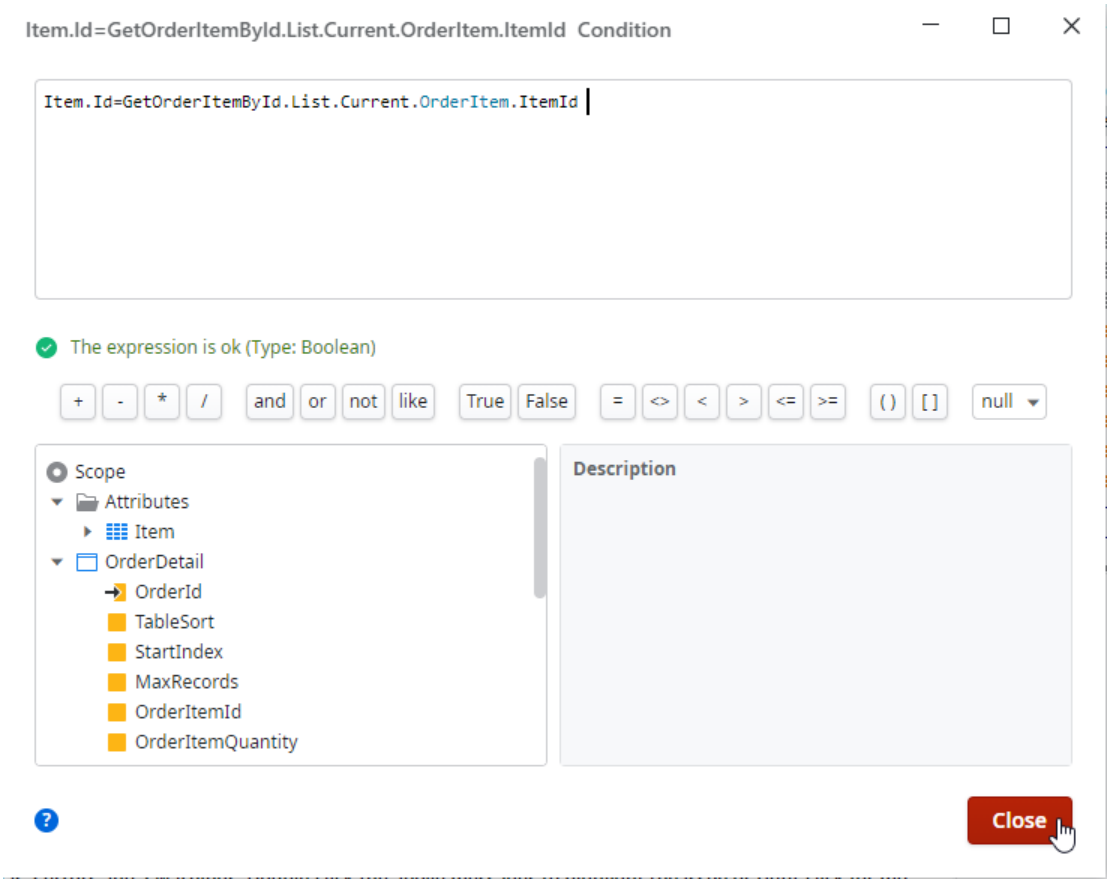
Let's now filter the Aggregate by the Item selected by the user in the dropdown.

3) Click on the **No Filters** tab and then on the **Add filter** option.



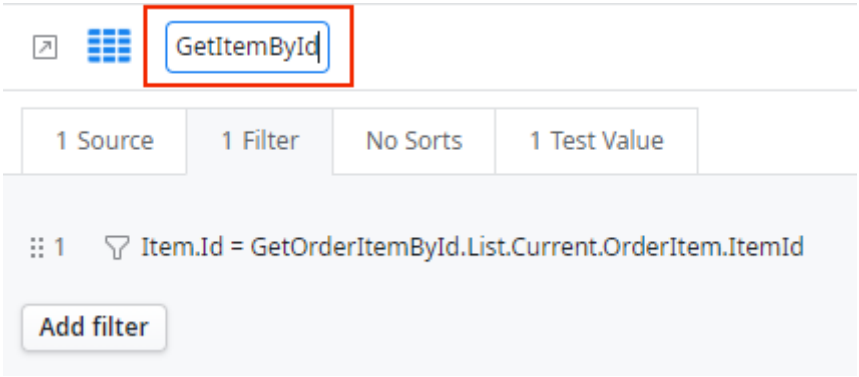
4) Add the following value to the Filter Condition:

```
Item.Id = GetOrderItemById.List.Current.OrderItem.ItemId
```

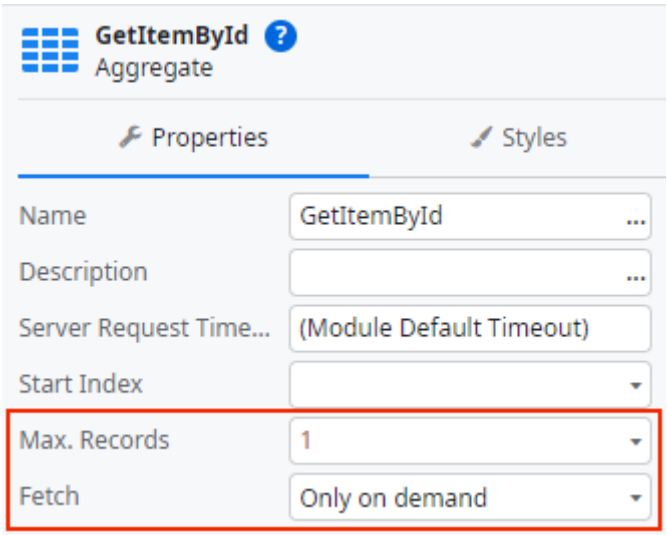


This is filtering the Aggregate to find the Item with the Id that matches the value that is selected in the dropdown (which is saved in the *GetOrderItemById.List.Current.OrderItem.ItemId* field).

5) Click on the Aggregate's name and rename it to *GetItemById*. You can Close the Aggregate after that.



6) Now, on the Aggregate's properties on the right side of the screen, set the **Max Records** to *1*, and change the **Fetch** property to **Only on Demand**.



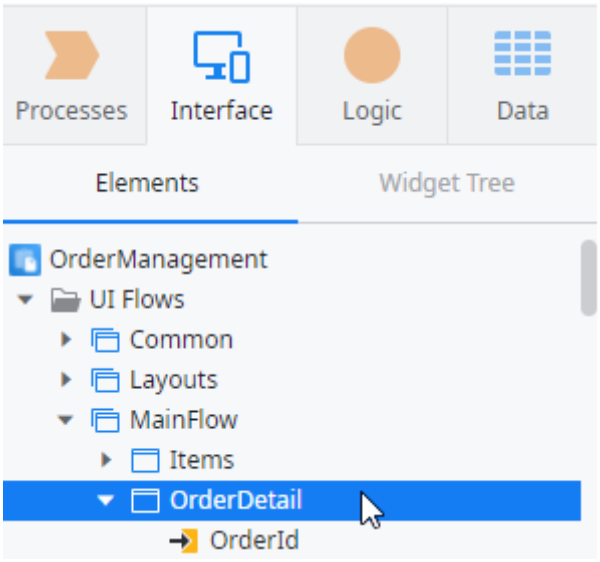
Here, you're setting that the Aggregate only fetches one item, and that it will be executed only when you want to.

Adjusting the Price Per Item Information

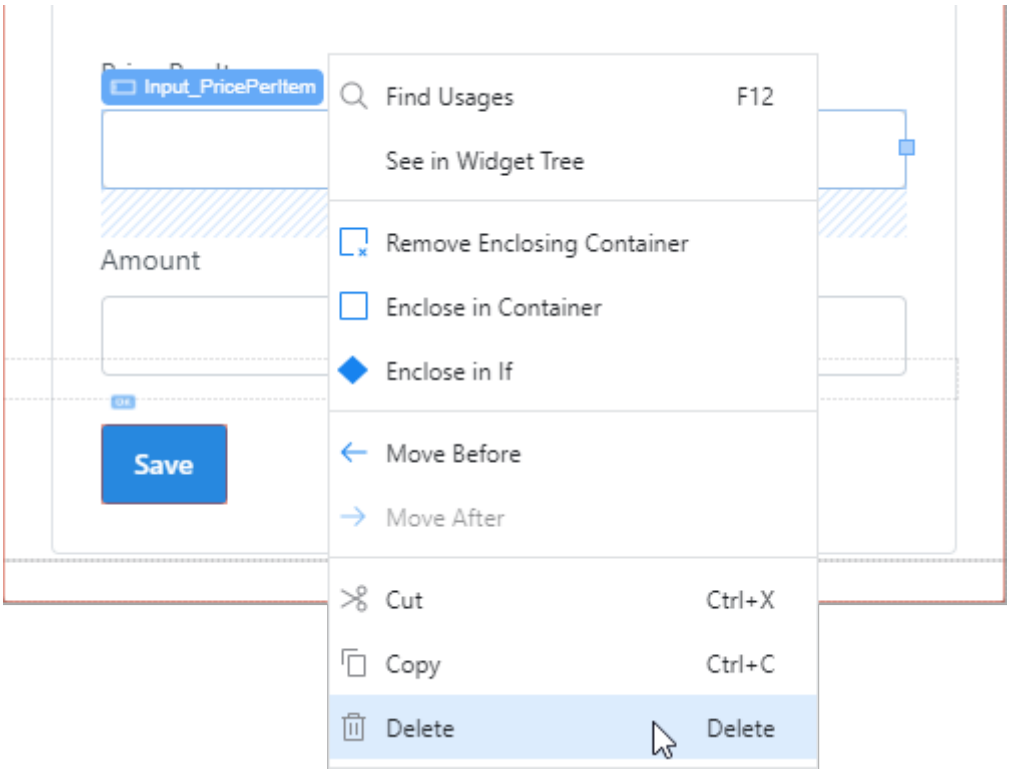
Now that you have the Item information available, based on the choice the user makes, you can adjust the Form to make sure the Price Per Item is not an input field that the

user can interact with, but an expression that just displays the price. Sounds like a lot of work, but it's simple once you get used to it!

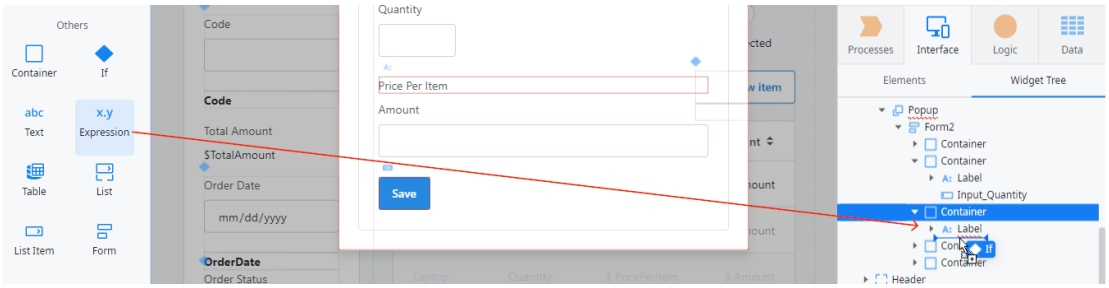
1) Go back to the **OrderDetail** Screen.



2) Delete the **PricePerItem** Input field (but keep the Label!).

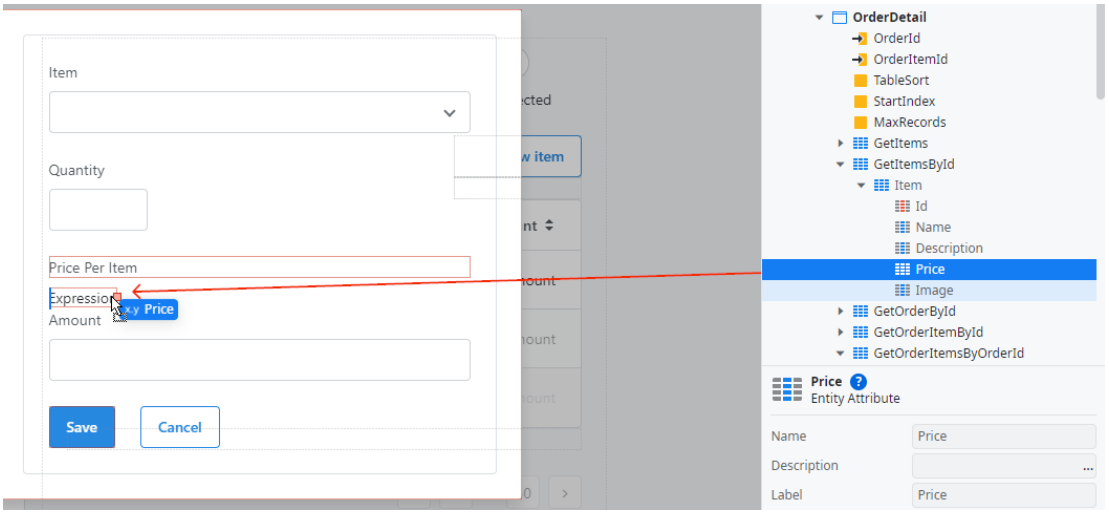


3) Drag an **Expression** from the Toolbox and drop it to the Container where the PricePerItem Input was. Use the Widget Tree to make sure you are dragging the Expression to the right place, like in the picture below.

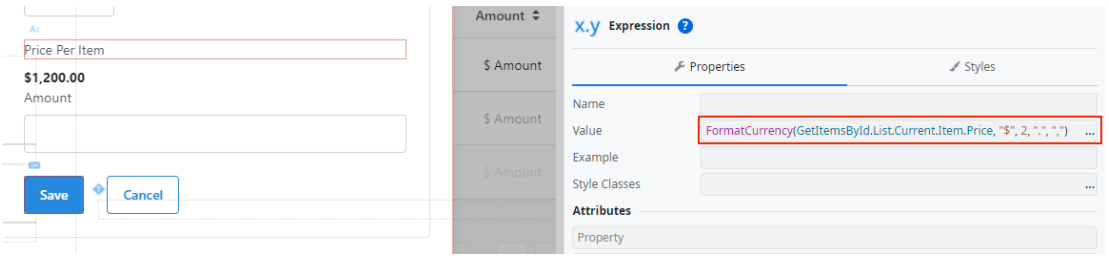


Close the dialog that appears.

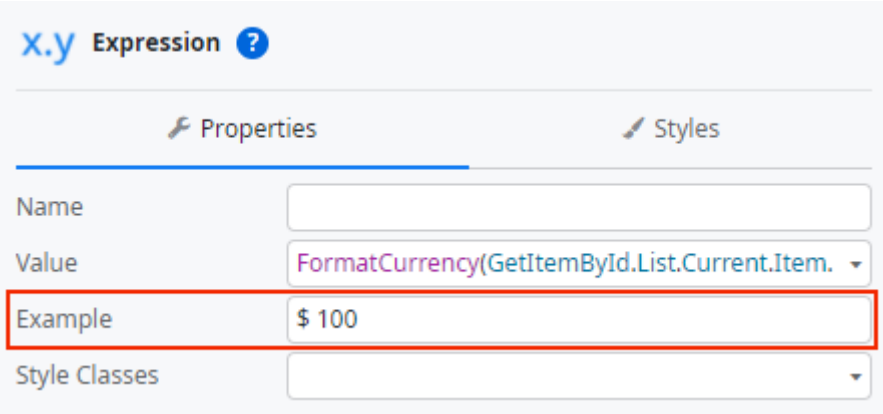
4) Drag the **Price** attribute from the **GetItemById** Aggregate to the Expression.



5) Click on the Expression to see its properties on the right sidebar. Notice that the Expression's **Value** is already adapted to the attribute you dragged.

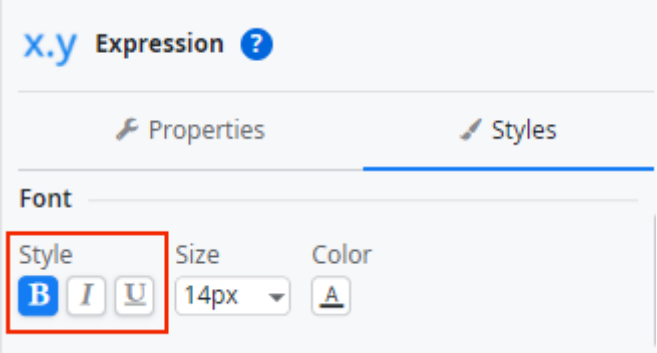


6) Type *$ 100* in the **Example** property of the Expression.

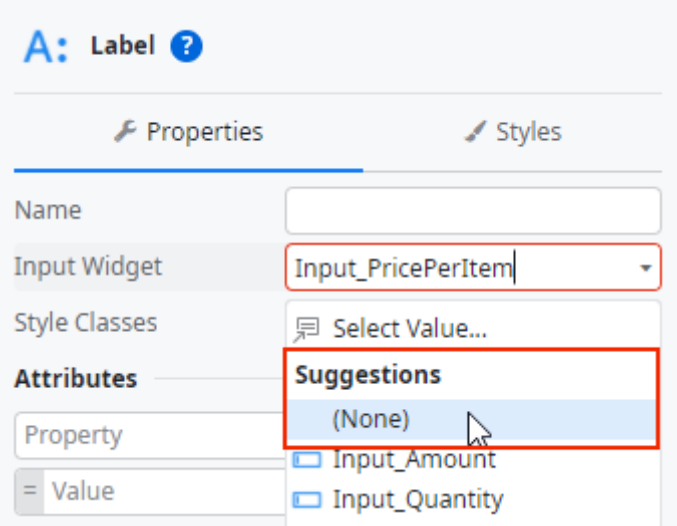


This is just for preview purposes, so the developer knows how it will look like in the browser.

7) Switch to the **Styles** tab and then select the **Bold** style.



8) Now, the Label has an error because you deleted its Input Widget. Select the Label, switch back again to its Properties, and set the **Input Widget** to *(None)*.

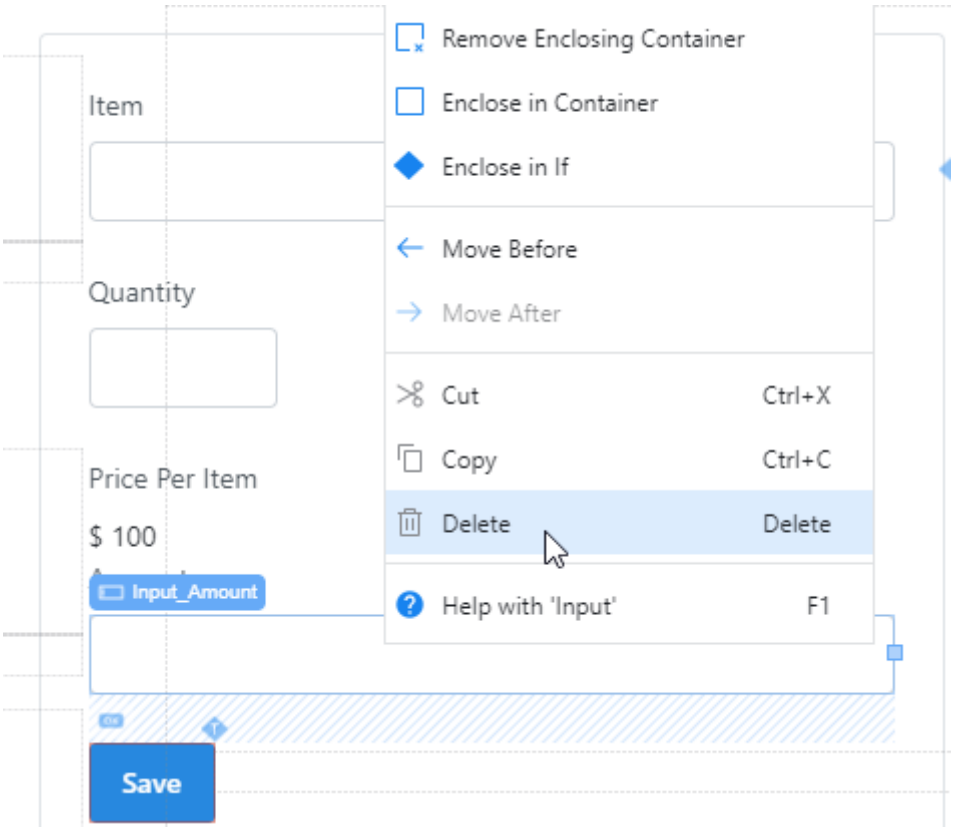


Well, there's another field now that you need to work on. It's the Amount, which will basically depend on the quantity and the price.

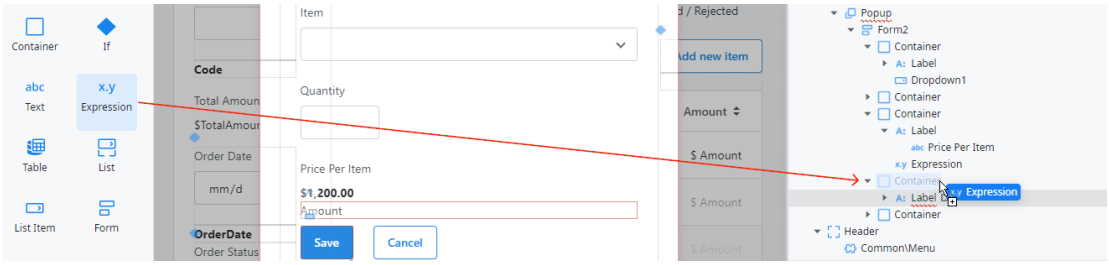
Adjusting the Amount Information

The Amount field is a calculated value, and not editable by the user. So you will apply the same logic used in the Price Per Item field.

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

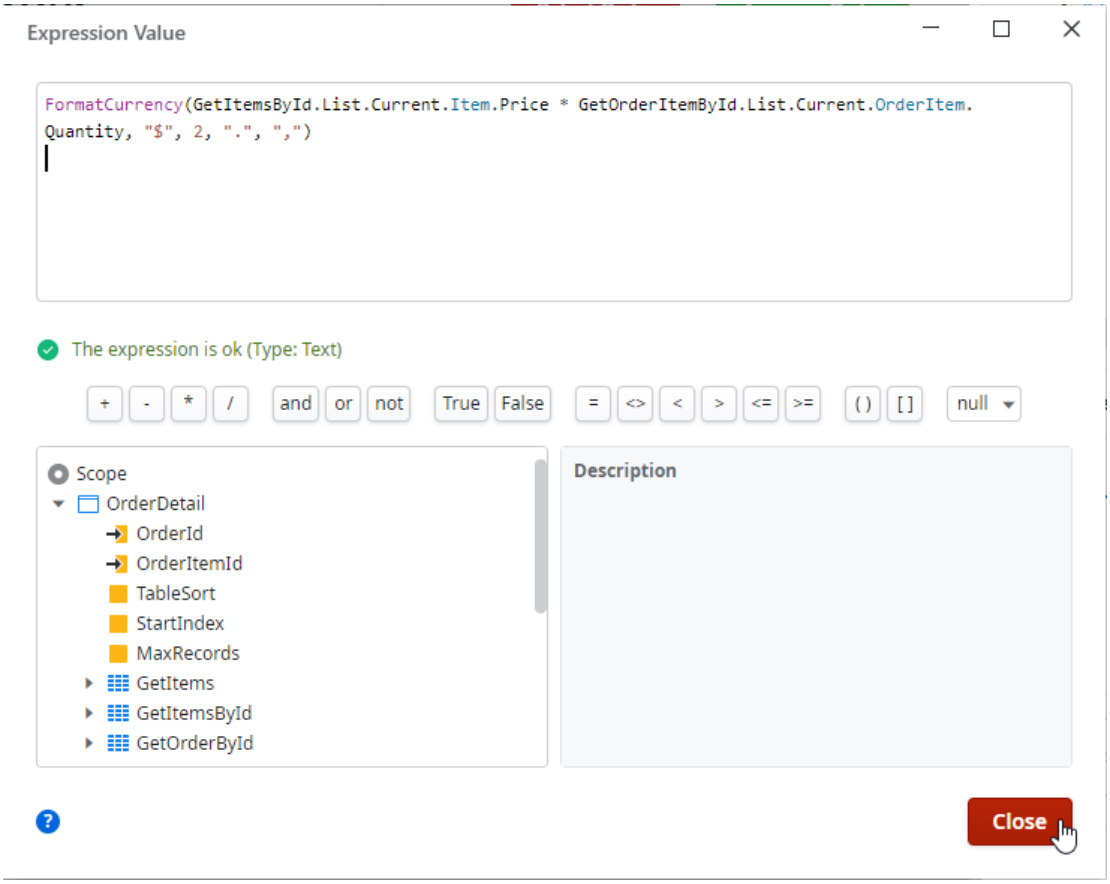


2) Drag an **Expression** from the Toolbox and drop it on the Container where the Amount Input was. Use the Widget Tree to make sure you are dragging the Expression to the right place, like in the picture below.



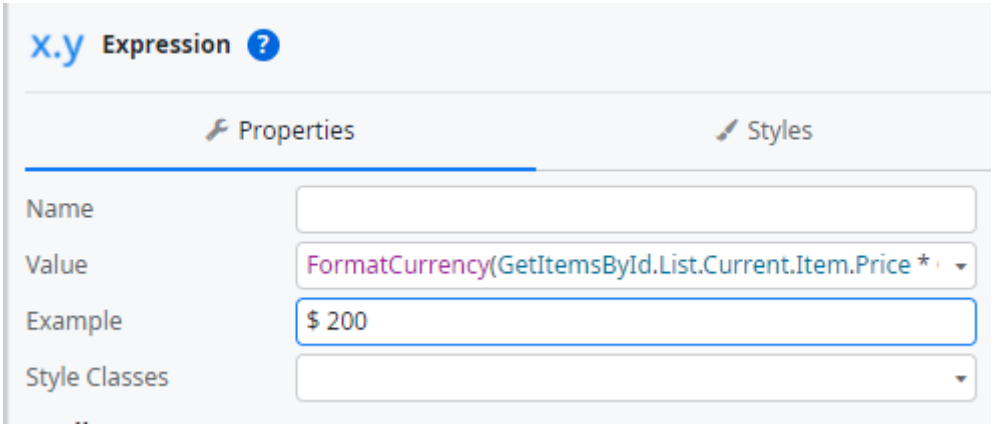
3) The Amount is calculated by multiplying the Price of the Item by the Quantity. Add the following value to the Expression Value:

```
FormatCurrency(GetItemById.List.Current.Item.Price *
GetOrderItemById.List.Current.OrderItem.Quantity, "$", 2, ".",
",")
```

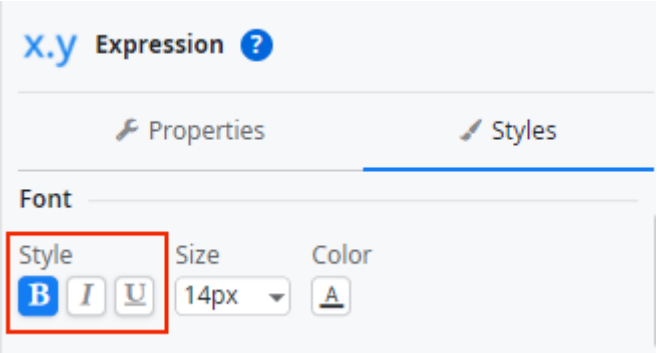


Notice the expression is using the Price you are fetching from the new Aggregate GetItemById and the Quantity. The **FormatCurrency** Action helps you define the look and feel for the currency. In this case we are using US dollars, with 2 decimal places.

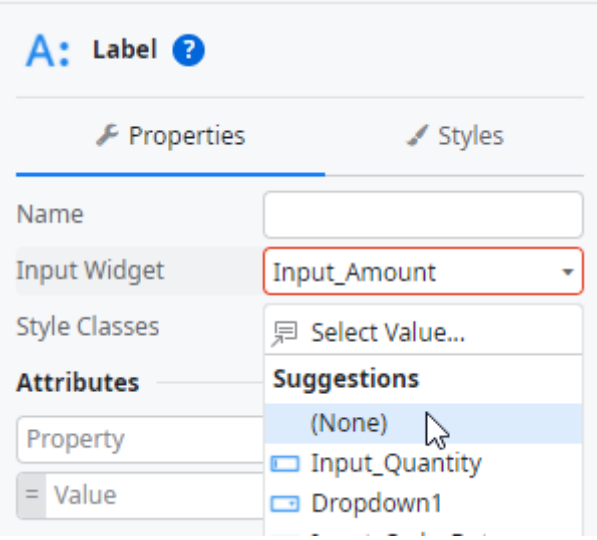
4) Click on the Expression to see its properties. Type *$ 200* in the **Example** property of the Expression.



5) Switch to the **Styles** tab and then select the **Bold** style.



6) The Label has an error because you deleted its Input Widget. Let's fix it by changing the **Input Widget** property to *(None)*.

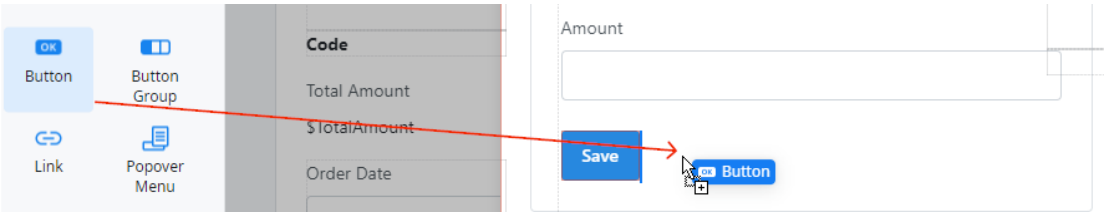


Awesome! The fields of the Form are right exactly where we want them!

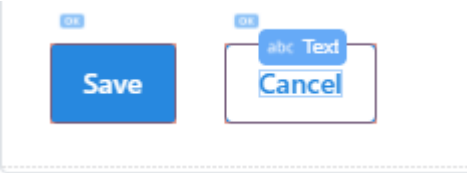
Adding a Cancel Button

To close the UI of the new Form, it is important to also give the option to cancel the choices made so far, and eventually close the Popup. So, let's add a Cancel Button.

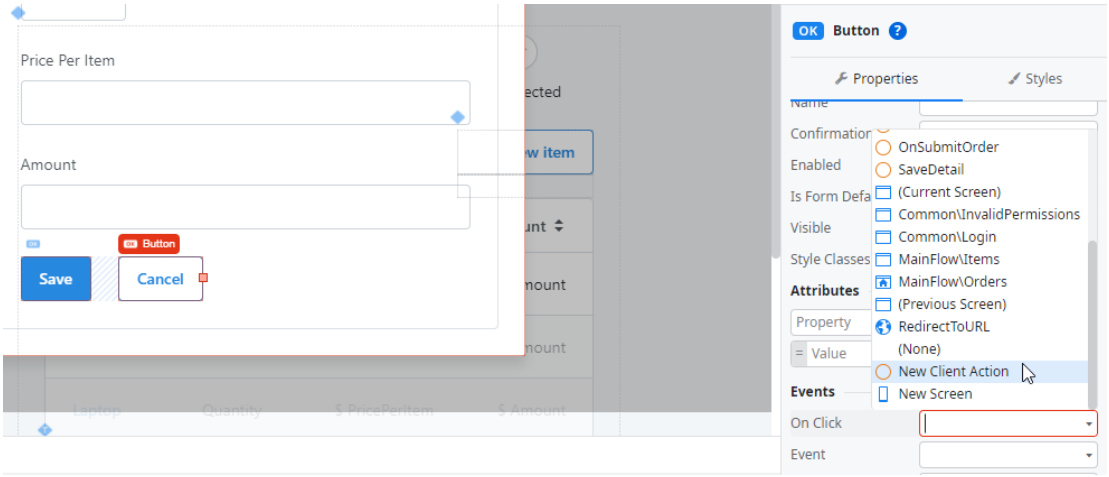
1) Drag a **Button** from the Toolbox and drop it to the Container that includes the Save Button in the Popup Form.



2) Type *Cancel* in the text of the Button.

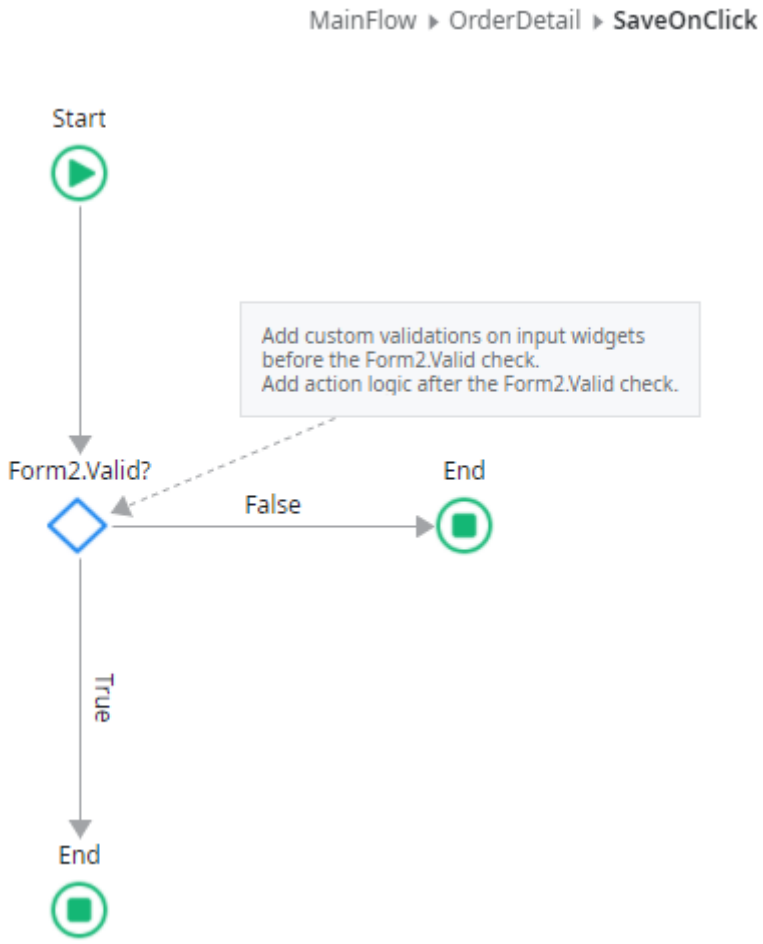


3) Click on the new Button, then go to the Properties area on the right sidebar. Set the **On Click** property to **New Client Action**.

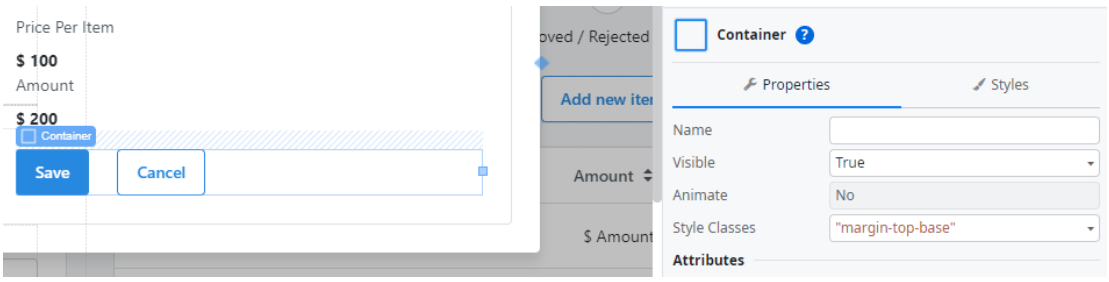


A new Client Action called *CancelOnClick* is created. We will come back here later to define the behavior of the app when a user clicks on the Cancel Button.

4) Open again the **OrderDetail** Screen. Double-click the **Save** Button inside the Popup to also create an Action (called *SaveOnClick*) to execute when the Button is clicked. You will implement it later.



5) Open again the **OrderDetail** Screen. Select the **Container** that is enclosing the Buttons. Set the **Style Class** property to *"margin-top-base"* to add some margin to the top.



The UI of the Popup is done! Now, we need to focus on the logic side. You can't publish at this point, since there are still some errors to fix. Don't worry! You will finish without errors!

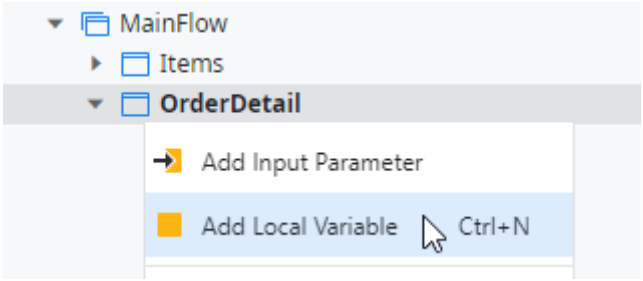
Creating the Logic to Add New Items

On the second part of this tutorial, you will start defining the logic that you need to make everything work. First, we will focus on adding new items to an order. This includes the Add New Item Button that will open the popup and the Save and Cancel Buttons inside the Popup.

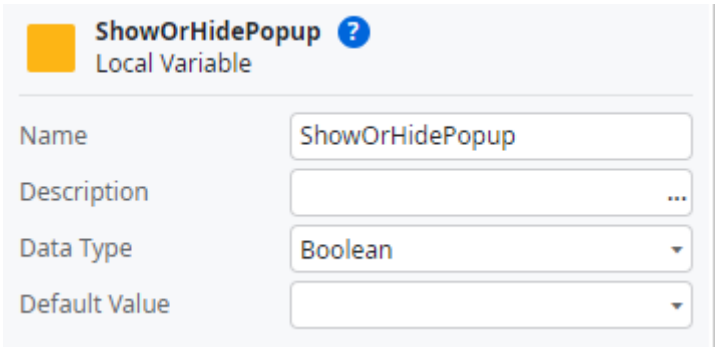
Setting up the Show Popup Property

You cannot publish your module yet, since it has an error. The error is caused by the **Show Popup** property, which needs to be defined. You will use a Local Variable to control when the Popup should appear or not.

1) Right-click on the **OrderDetail** Screen name and select *Add Local Variable*.

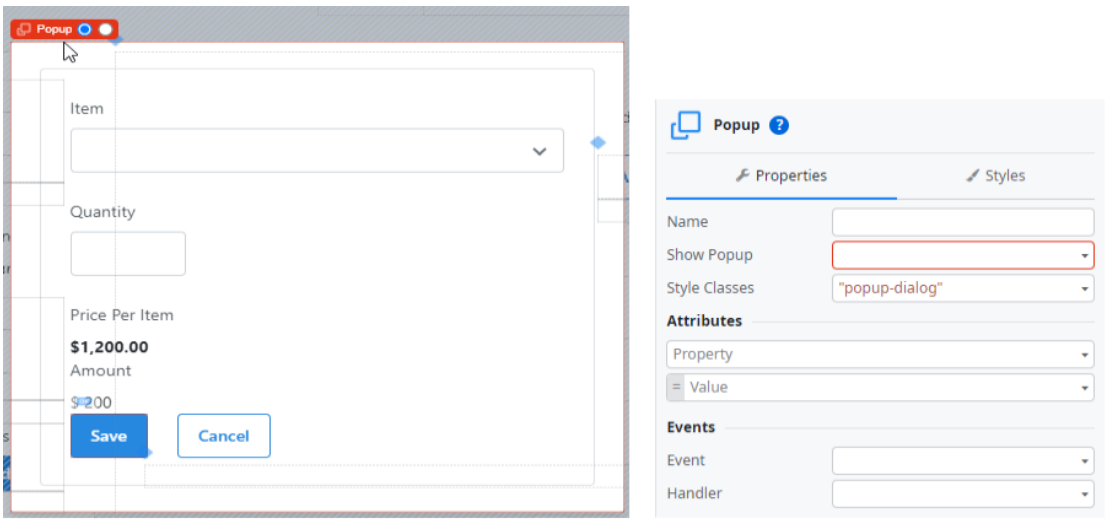


2) Set its **Name** to *ShowOrHidePopup*.



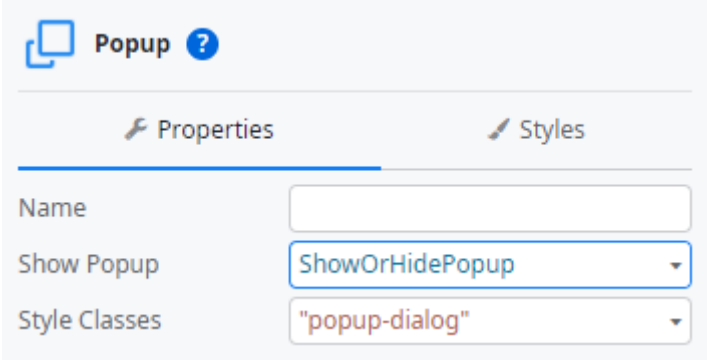
As soon as you rename it, the **Data Type** should change automatically to *Boolean*. Smart, right?

3) Click on the **Popup** on the Screen to see its Properties.



You will see the error highlighted in red.

4) Set the **Show Popup** property to the new Variable that you just created.

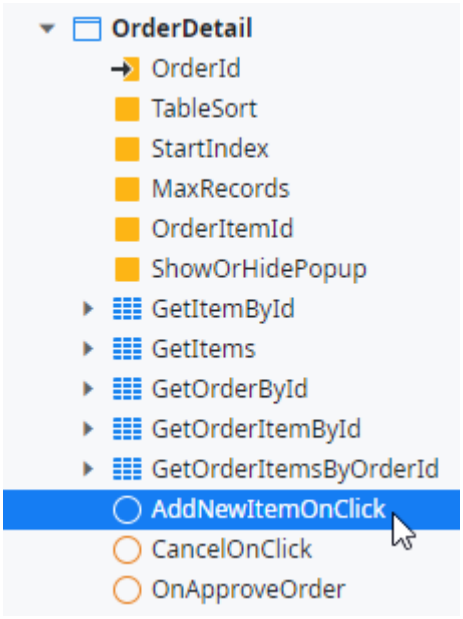


You will from now on change the value of this varible to true or false, whenever you want the Popup to open and close. OutSystems does the rest!

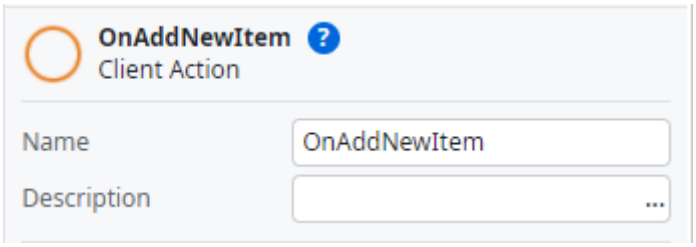
Creating the logic to Add a new Item

Now that we have a Variable that controls the popup, let's implement the logic to add a new item.

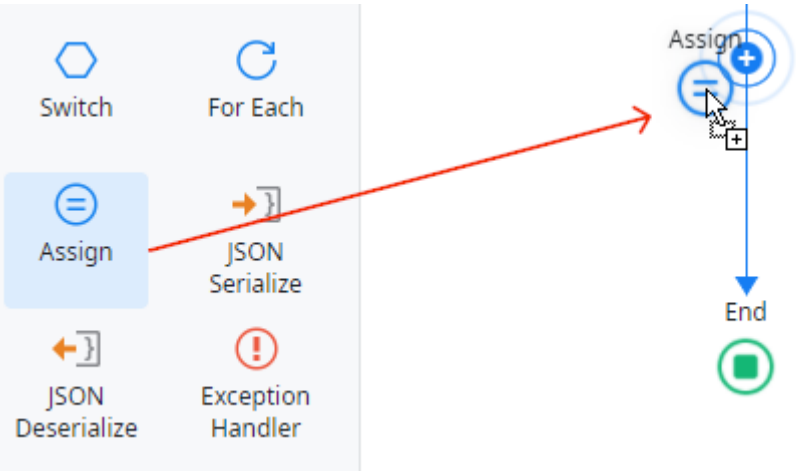
1) Open the **AddNewItemOnClick** Action that was created in previous tutorials.



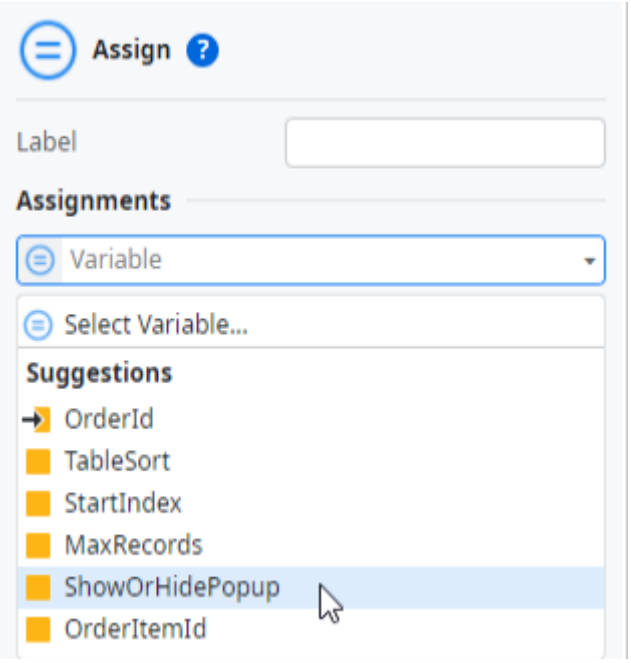
2) Rename it to *OnAddNewItem*.



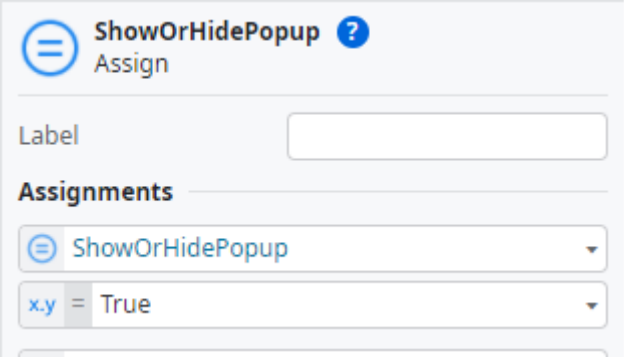
3) Inside the Action, drag an **Assign** from the left sidebar and drop it on the flow.



4) In the **Variable** field, select the **ShowOrHidePopup**.



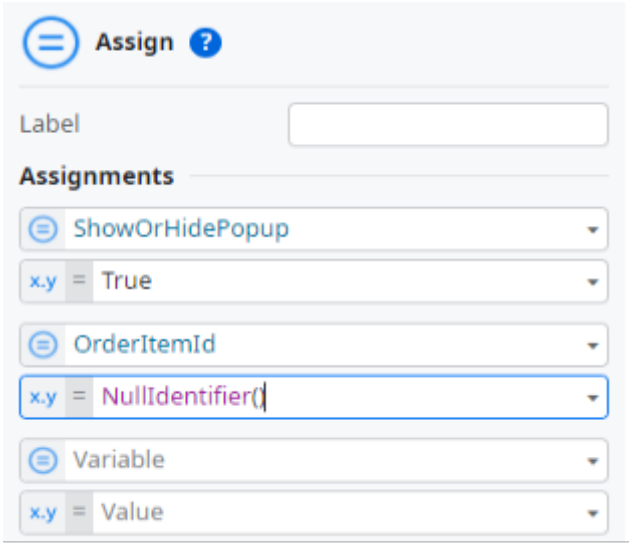
5) We want the Popup to be visible when a user clicks on the Add New Item Button. Set the **Value** to *True*.



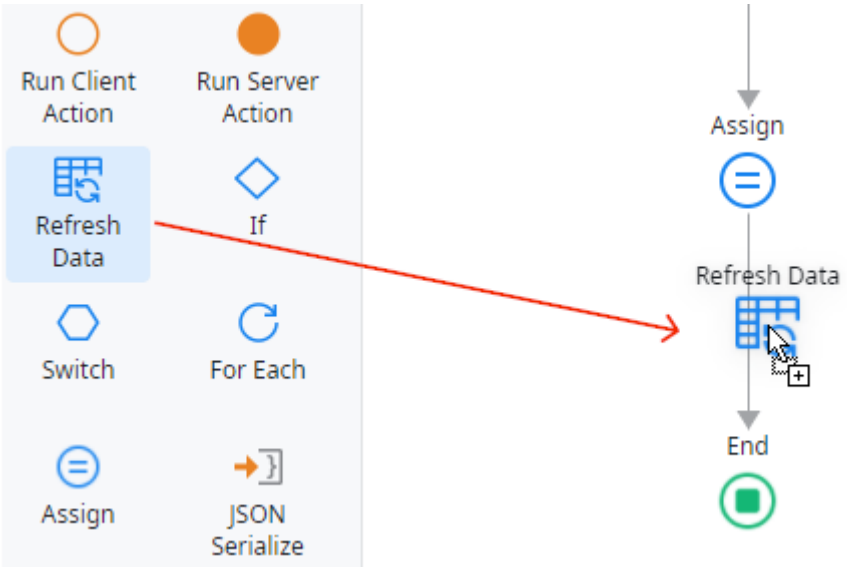
Note: You can assign multiple variables using only one Assign element. You don't need to drag a new Assign!

Remember the **OrderItemId** that was created as Input and we changed to Local Variable? This variable is used as a filter of the GetOrderItemById Aggregate, which is the one that populates the Form inside the Popup. So, we need to set that variable to *NullIdentifier()* here, since we're adding a new order item that does not exist yet.

6) Use the Assign to set the **OrderItemId** to *NullIdentifier()*.

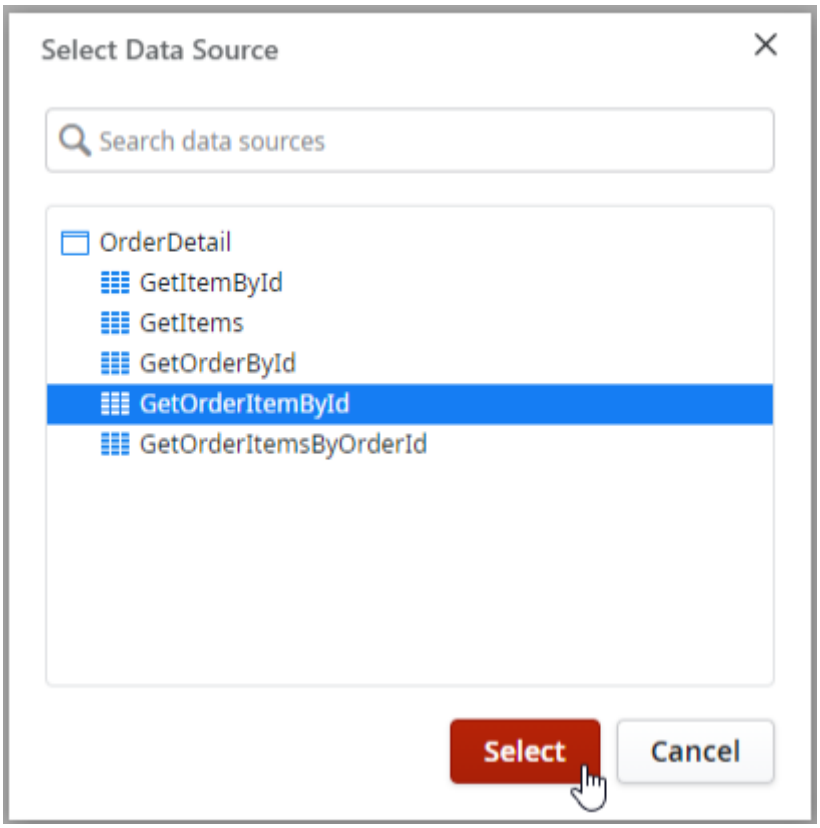


7) Since you changed a value that is used in the filter of the Aggregate, you need to execute the Aggregate again, to make sure the data is up to date. Drag a **Refresh Data** element from the left sidebar and drop it after the Assign.



Let's recap: when clicking the Add New Item Button, the popup opens, the order item identifier is reset and the Aggregate runs again to refresh the data.

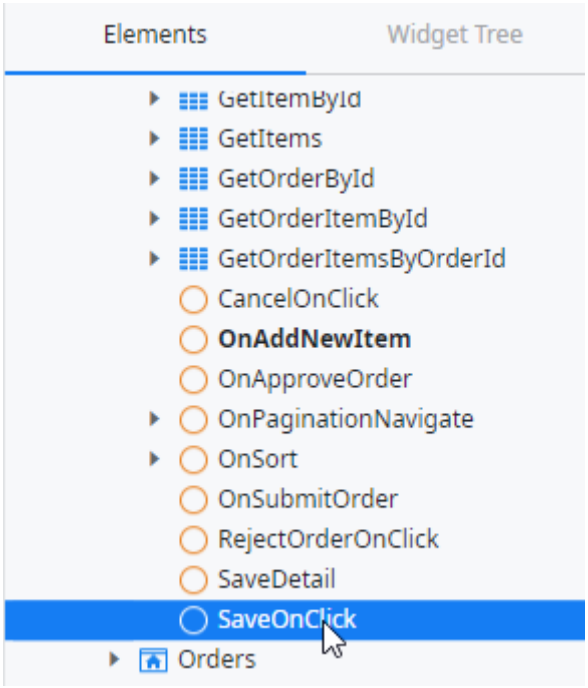
8) Select the **GetOrderItemById** Aggregate in the **Select Data Source** dialog.



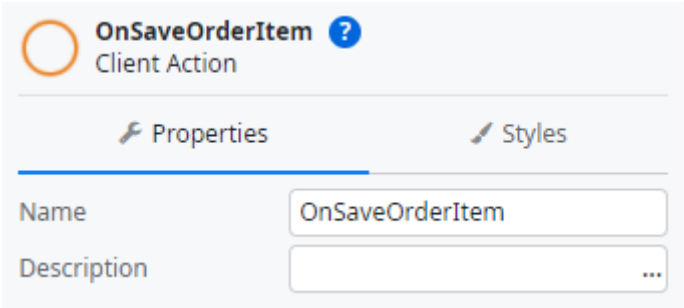
Creating the Logic to Save an OrderItem

Now it is time to work on the behavior of the Save Button of the Popup. Notice that this is not the same as saving an Order! This Button adds a new item to the Order, creating an OrderItem. So we need to create the logic for that.

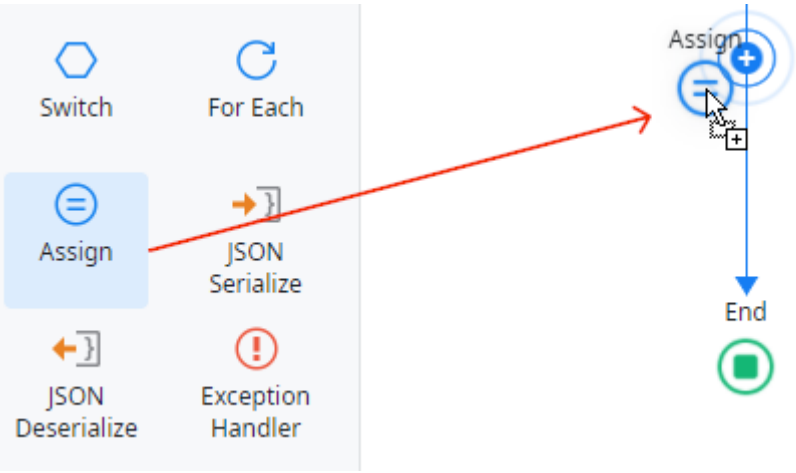
1) Double-click the **SaveOnClick** Action on the right sidebar to open it. This is the Action that runs when the Save Button is clicked.



2) Set the **Name** of the Action to *OnSaveOrderItem*.



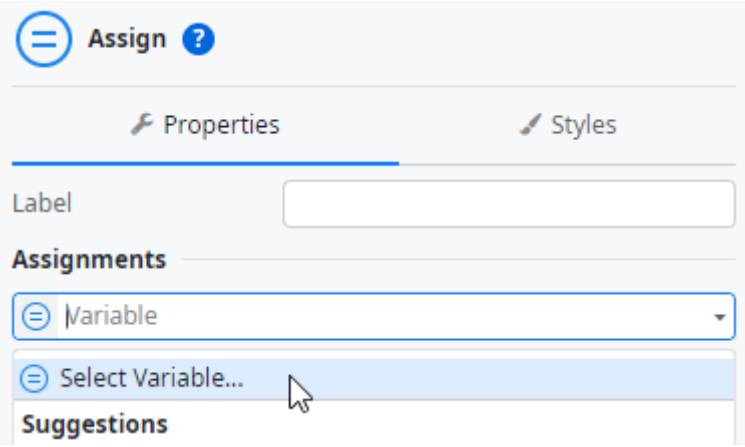
3) Inside the Action, drag an **Assign** and drop it in the flow, after the Form2.Valid.



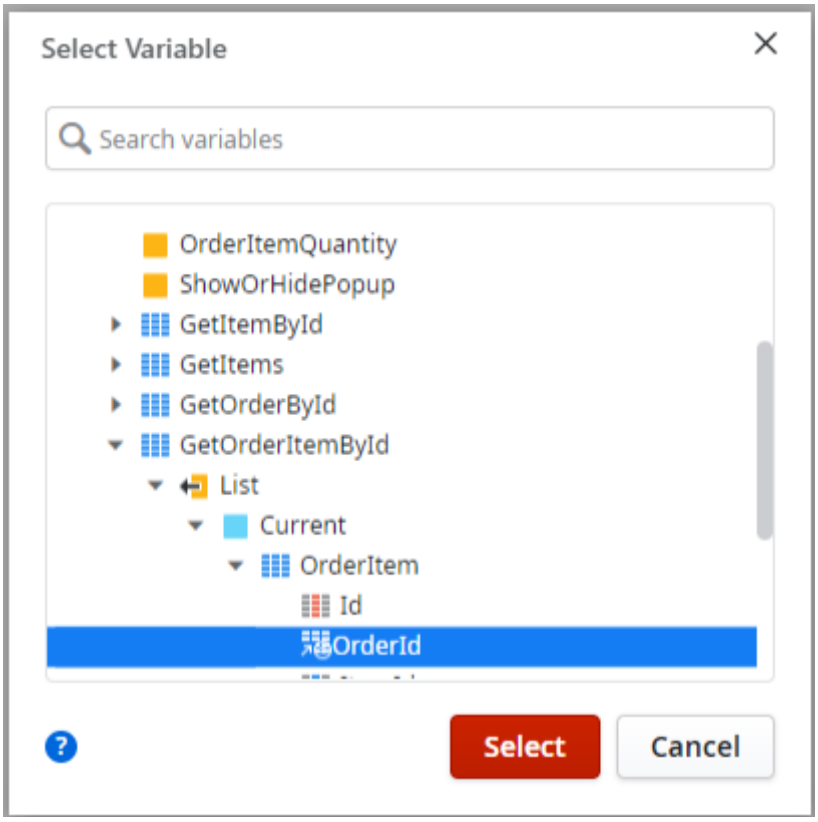
You will need to set some information here. Remember, you want to save the OrderItem. The user selects the Item and the Quantity in the Form. But, the OrderItem Entity has more fields, like the OrderId, the PricePerItem and the Amount. And that's what we need to assign here. Let's start by the OrderId.

Assigning the OrderItem to the Order

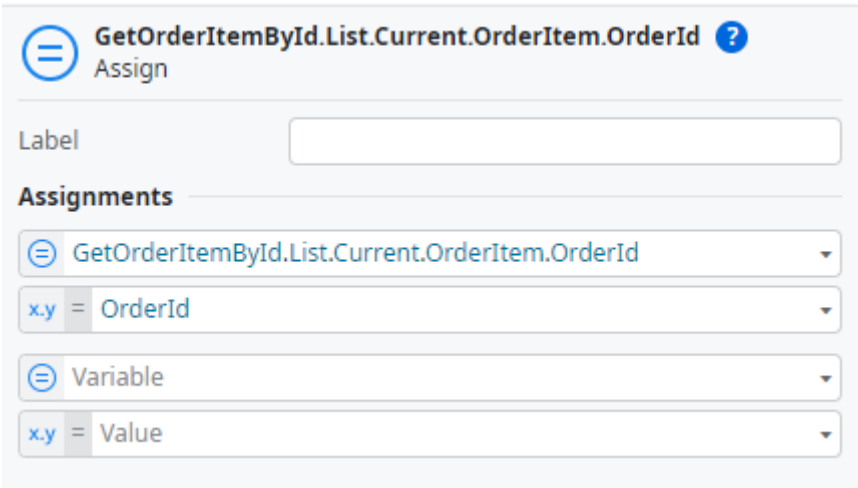
1) Click on the **Variable** property of the Assign and select the option **Select Variable**.



2) Select the `GetOrderItemById.List.Current.OrderItem.OrderId`, just like the image shows.



3) Set the **Value** to be *OrderId*.



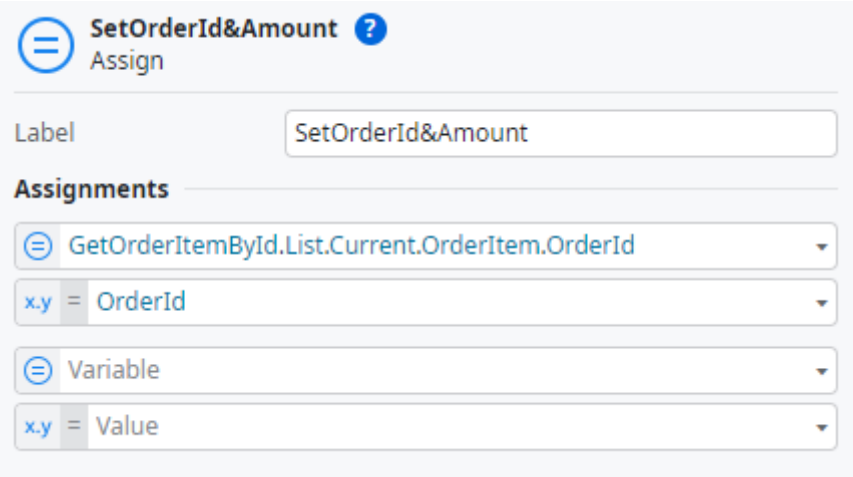
So, our OrderItem (fetched by the Aggregate GetOrderItemById) will be added to the Order that's being edited in the OrderDetail Screen. So, we can leverage the Screen's Input Parameter, OrderId, to set that value.

4) Publish to save all the work you did! You might wanna stretch your legs for a bit, but don't worry, your progress is saved!

Calculating the OrderItem Amount and the Order's Total Amount

Let's work with the OrderItem Amount now. The OrderItem Amount is pretty easy to calculate. You did it already in the UI. You just need to multiply the Item Price by the Quantity.

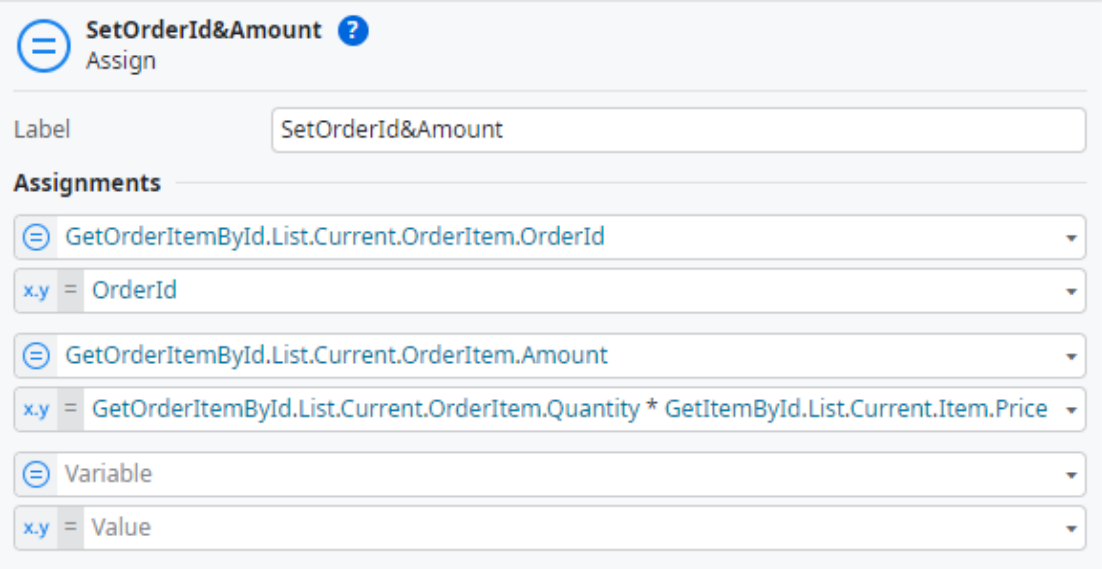
1) Set the **Name** of the Assign to *SetOrderId&Amount*.



Note: Naming the Assigns makes it easier to understand what is happening without having to actually click on the element and read all the assignments inside it. This is not a mandatory step, but it is highly recommended as you develop more complex logic.

2) In the existing Assign, set the next **Variable** field to: `GetOrderItemById.List.Current.OrderItem.Amount`. Then, set the **Value** to be:

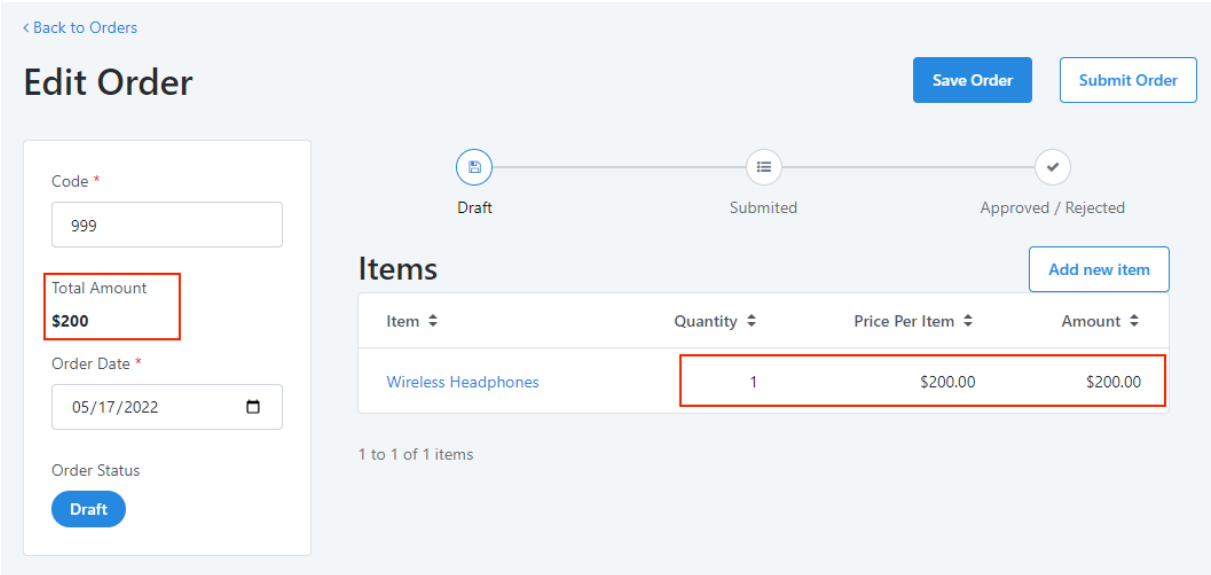
```
GetOrderItemById.List.Current.OrderItem.Quantity *
GetItemById.List.Current.Item.Price
```



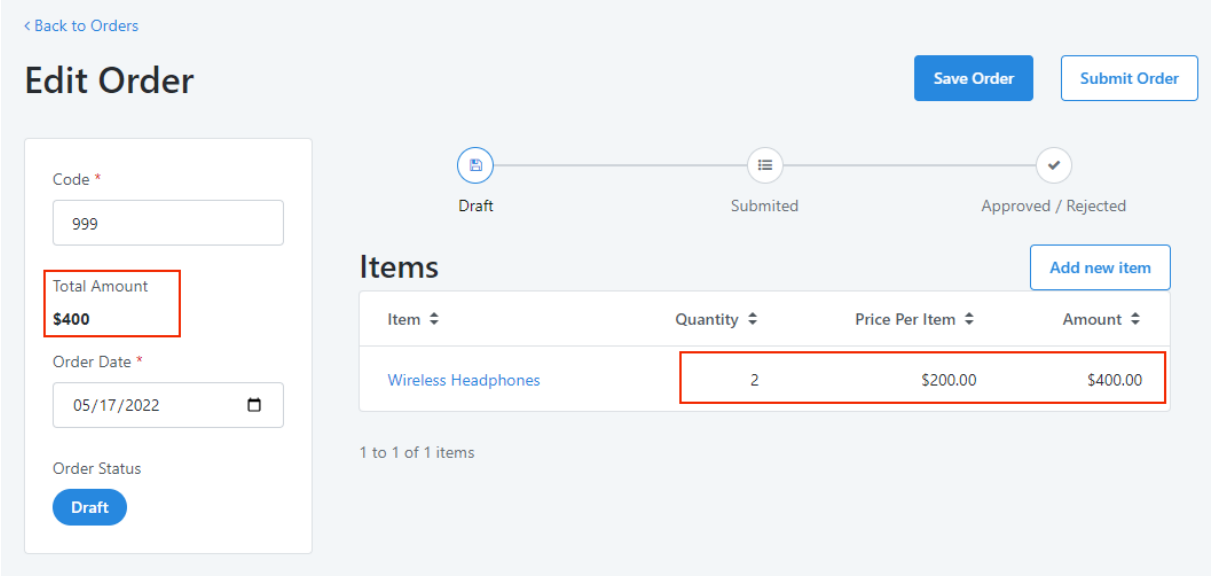
Since a new OrderItem is being created, the Order's Total Amount should also be updated.

If you don't think too much about it, the Total Amount starts with zero, and is incremented every time you add a new item. Quite simple right? If you could only add new items, that would be true. However, a user may change the quantity or even select a different item when adding / editing an order item. The app needs to be prepared for that as well! So, instead of just adding the amount of the OrderItem to the Total Amount of the Order, we also need to consider the previous Amount.

An example can really help in this scenario: Imagine you added 1 Headphone, and it costs $200. So your Total Amount is $200.

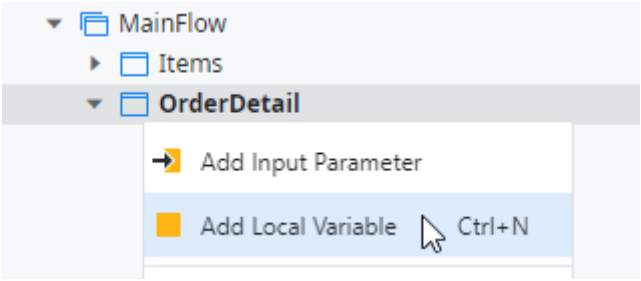


Now let's suppose you want to change the quantity to 2. You should **not** just add $400 to the Total Amount. You need to first remove the previous Amount from the Total Amount, then add the new one.

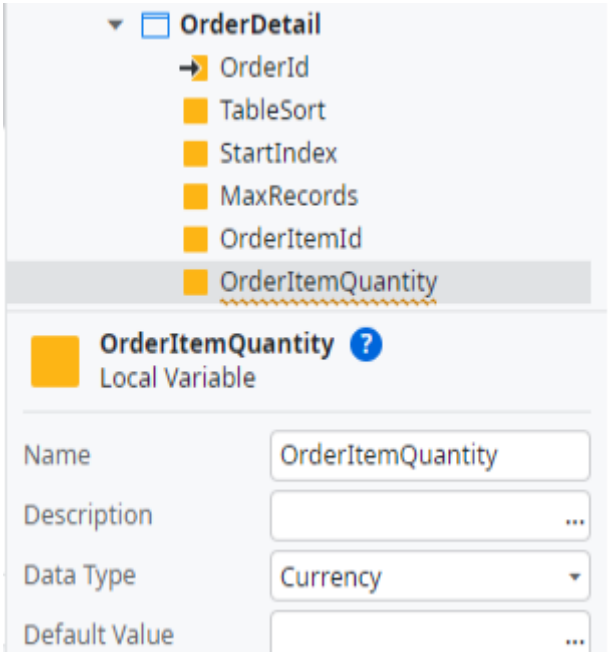


Seems tricky? We will help you!

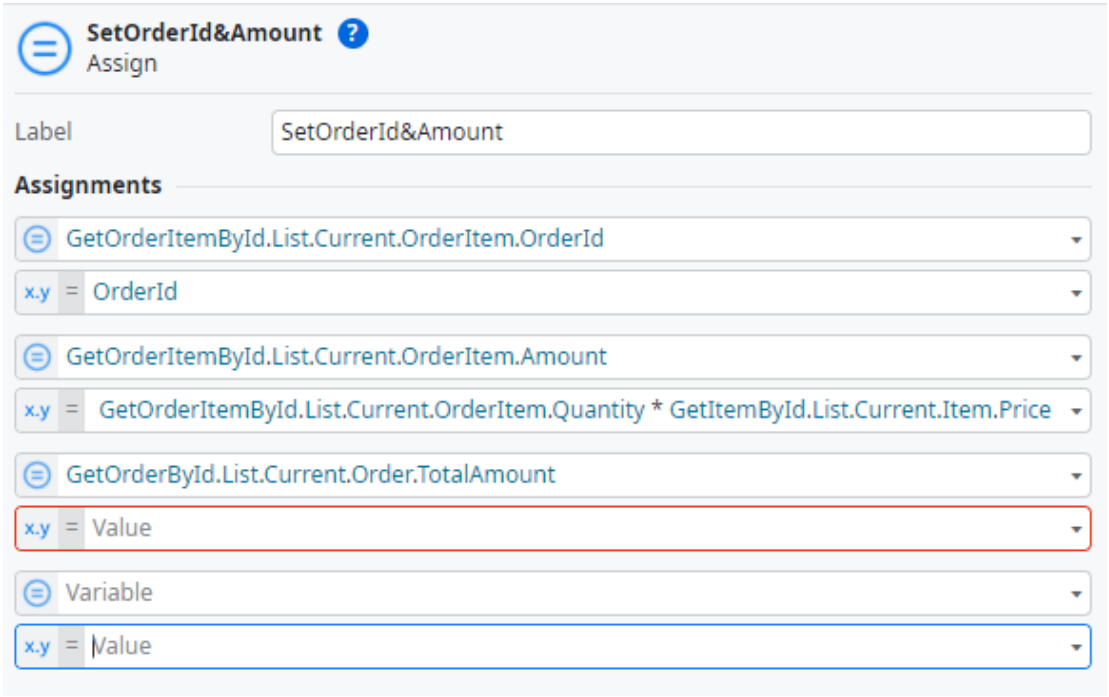
1) Since you will work with different quantities (the new one and the previous one) to calculate the total amount, we need a new Local Variable to help us save the old quantity selected. Right-click the **OrderDetail** Screen and select **Add Local Variable**.



2) Set its **Name** to *OrderItemQuantity* and change the **Data Type** to **Currency**.

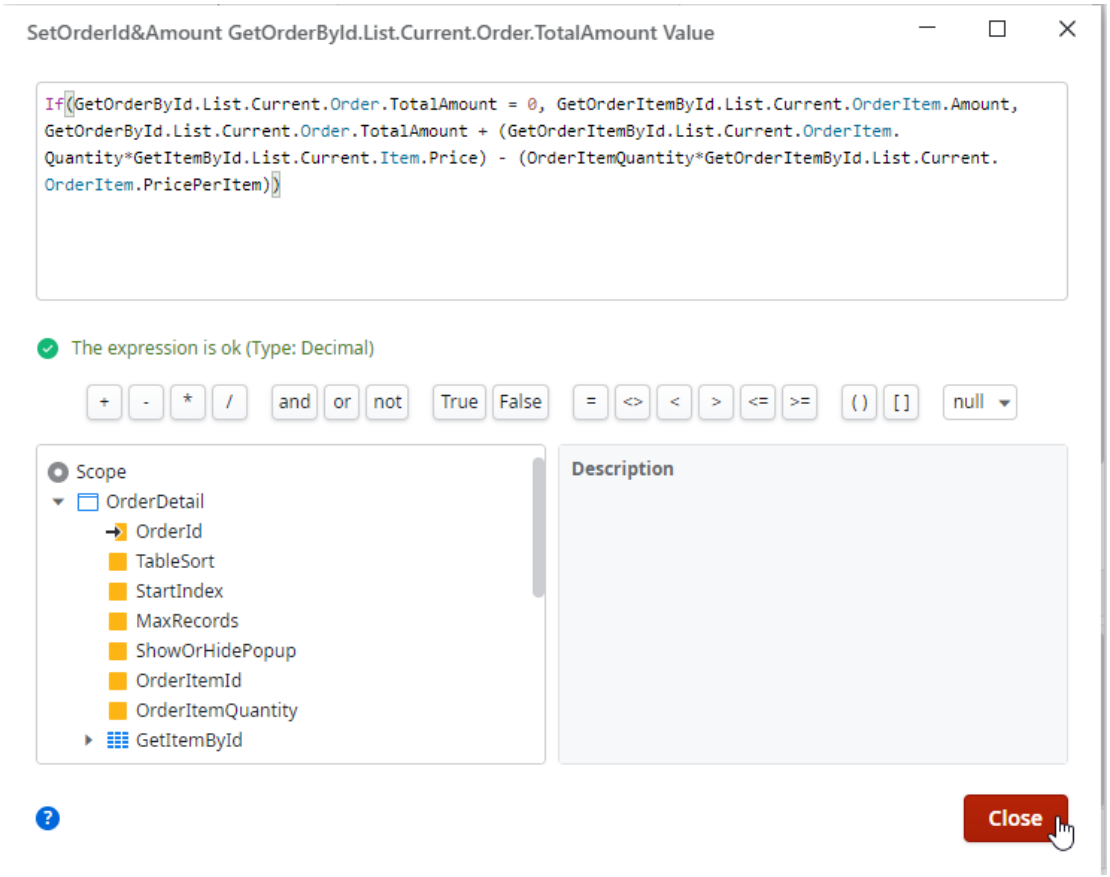


3) Back on our Action, select the **SetOrderId&Amount** Assign. Set the next **Variable** property to: `GetOrderById.List.Current.Order.TotalAmount`.



4) Set the **Value** to:

```
If(GetOrderById.List.Current.Order.TotalAmount = 0,
GetOrderItemById.List.Current.OrderItem.Amount,
GetOrderById.List.Current.Order.TotalAmount +
(GetOrderItemById.List.Current.OrderItem.Quantity *
GetItemById.List.Current.Item.Price) - (OrderItemQuantity *
GetOrderItemById.List.Current.OrderItem.PricePerItem))
```



It's a long formula right?! But it's "just" math! If the Total Amount of the order is zero, then its value is the amount of the order item (since we don't have anything else to consider).

If not, then it' the *Total Amount + ((new quantity x new price) - (old quantity x old price))*. The new quantity is what the user selected in the Form, the new price is given by the Aggregate GetItemById we worked on earlier, the old quantity will be saved in the variable we just created and the old price is still saved in the order item, since we didn't update the order item yet in the database (it will only happen after the assign). Don't worry if it seems too much now. You will have plenty of time to come back and explore.

Updating the Price Per Item

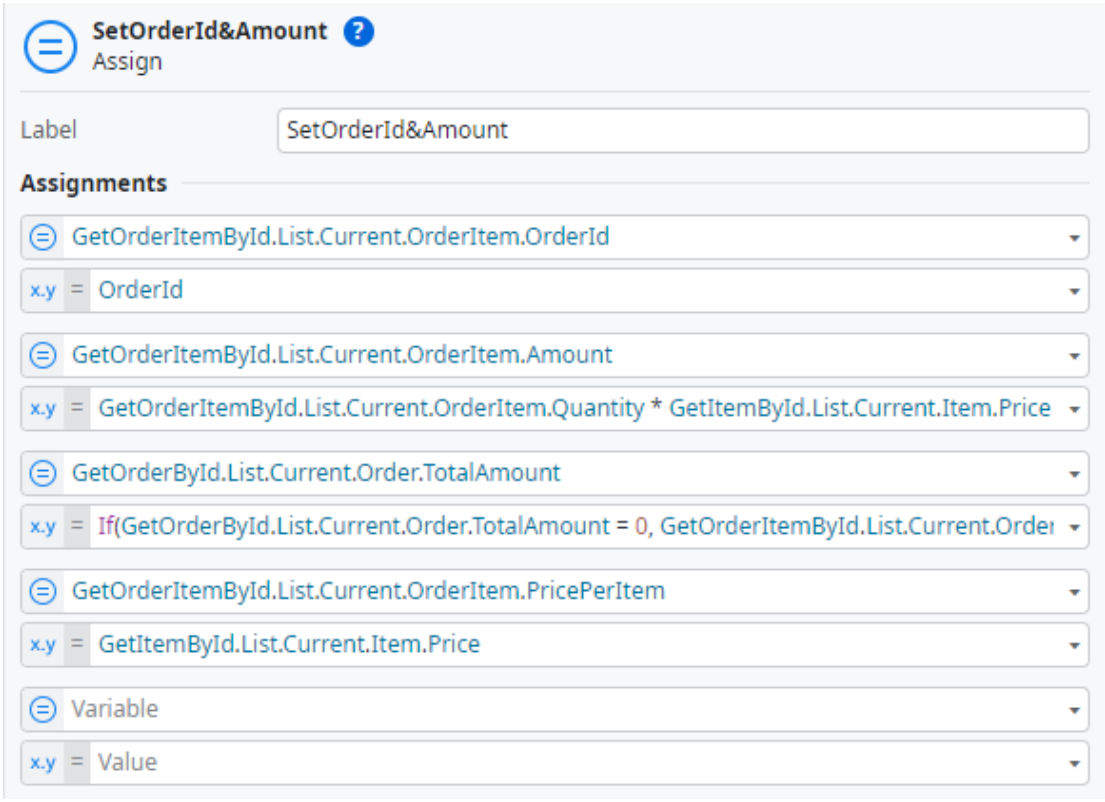
Now, we have one final field to update: the price per item of the OrderItem. In the previous assign, we used this value as the "older price" because we have not changed it yet. You will do it now, because the total amount is already calculated.

1) In the same Assign, set the next **Variable** field to be:

   `GetOrderItemById.List.Current.OrderItem.PricePerItem`

2) Set the **Value** to be:

```
GetItemById.List.Current.Item.Price
```

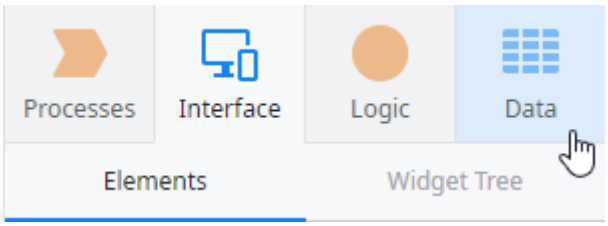


All the fields are set! We can continue and save the OrderItem in the database.

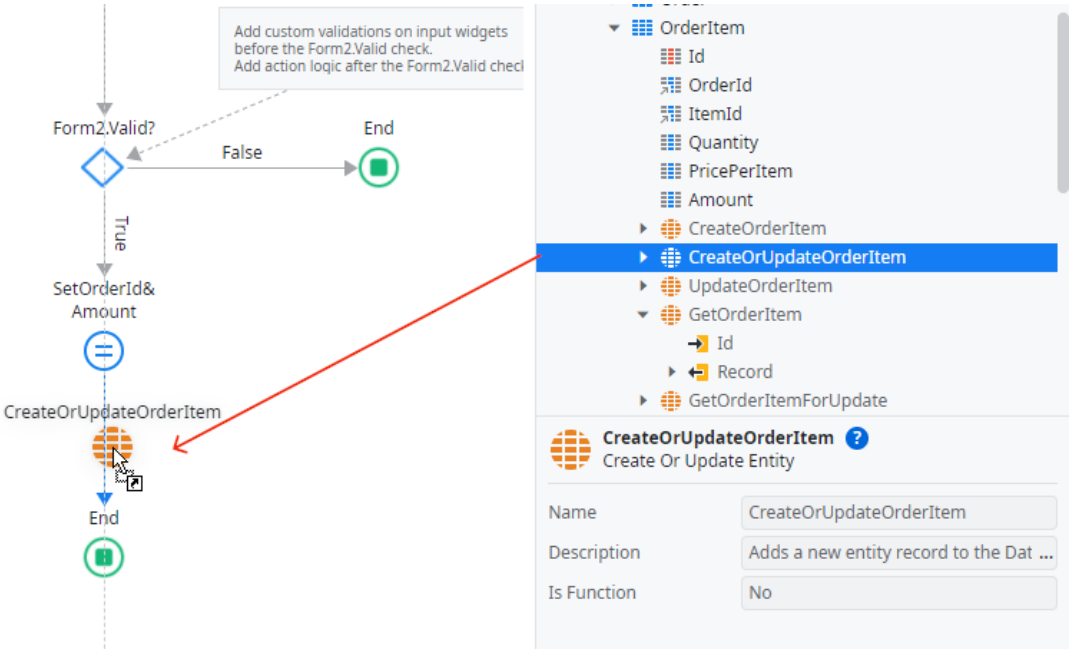
Writing the new values in the Database

The assignments are ready, but we are not writing the OrderItem in the Database yet. You will use some Actions that already exist to do the heavy lifting for you.

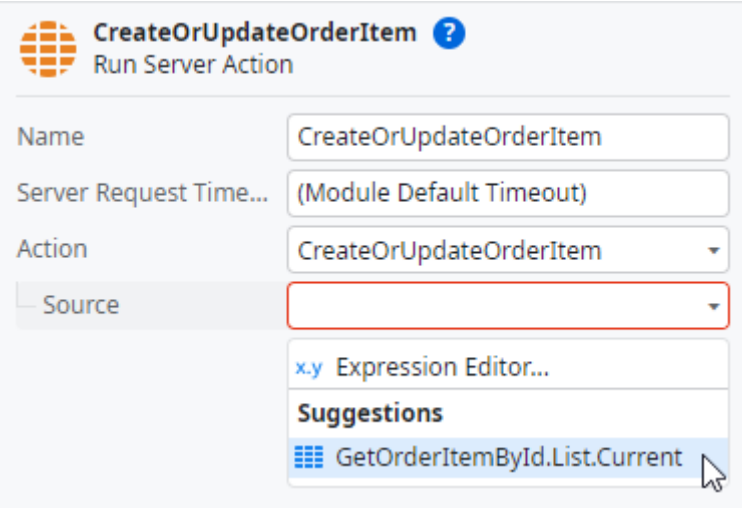
1) Click on the **Data** tab to open it.



2) Expand the **OrderItem** Entity, select the **CreateOrUpdateOrderItem** Action, then drag and drop it in the Action flow under the Assign.



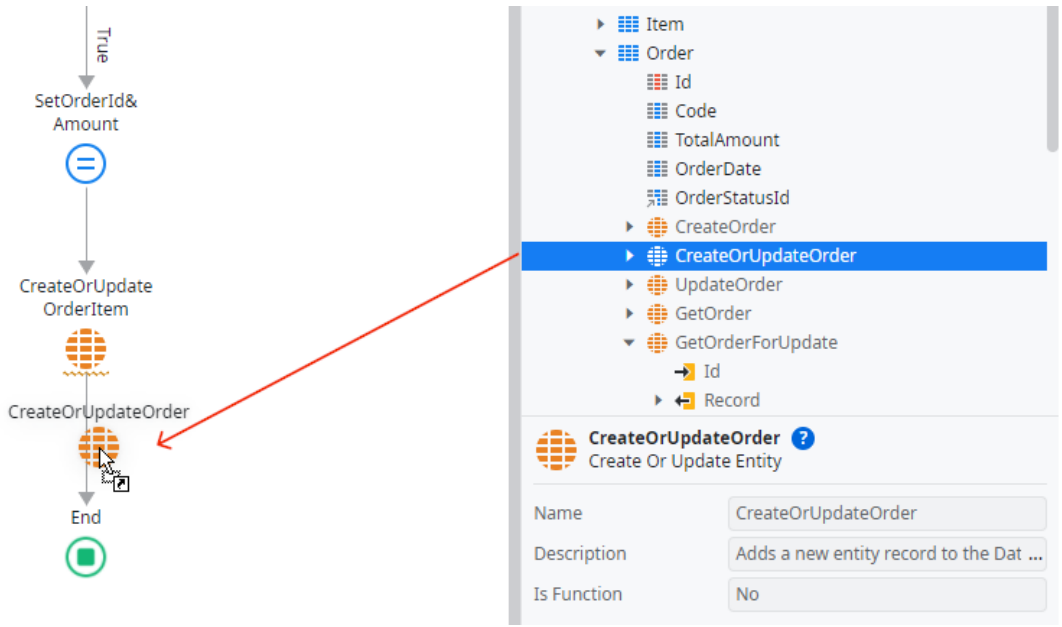
3) Set the **Source** property and select `GetOrderItemById.List.Current`.



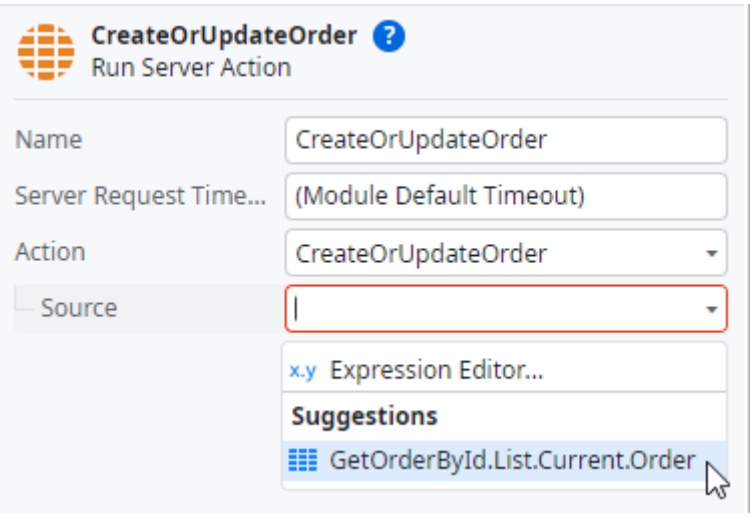
All the Entities you created in the app come with some Actions that execute common operations, such as create, update, get and delete records on the corresponding Entity. So, this Action will update in the database the Order Item being modified in the Popup.

And what about the actual Order? If the OrderItem changes, the total amount changes, so we also need to update the Order.

4) Expand the **Order** Entity in the Data tab. Select the *CreateOrUpdateOrder*, then drag and drop it in the Action flow **under** the *CreateOrUpdateOrderItem* Action.



5) Set the **Source** property to `GetOrderById.List.Current.Order`.

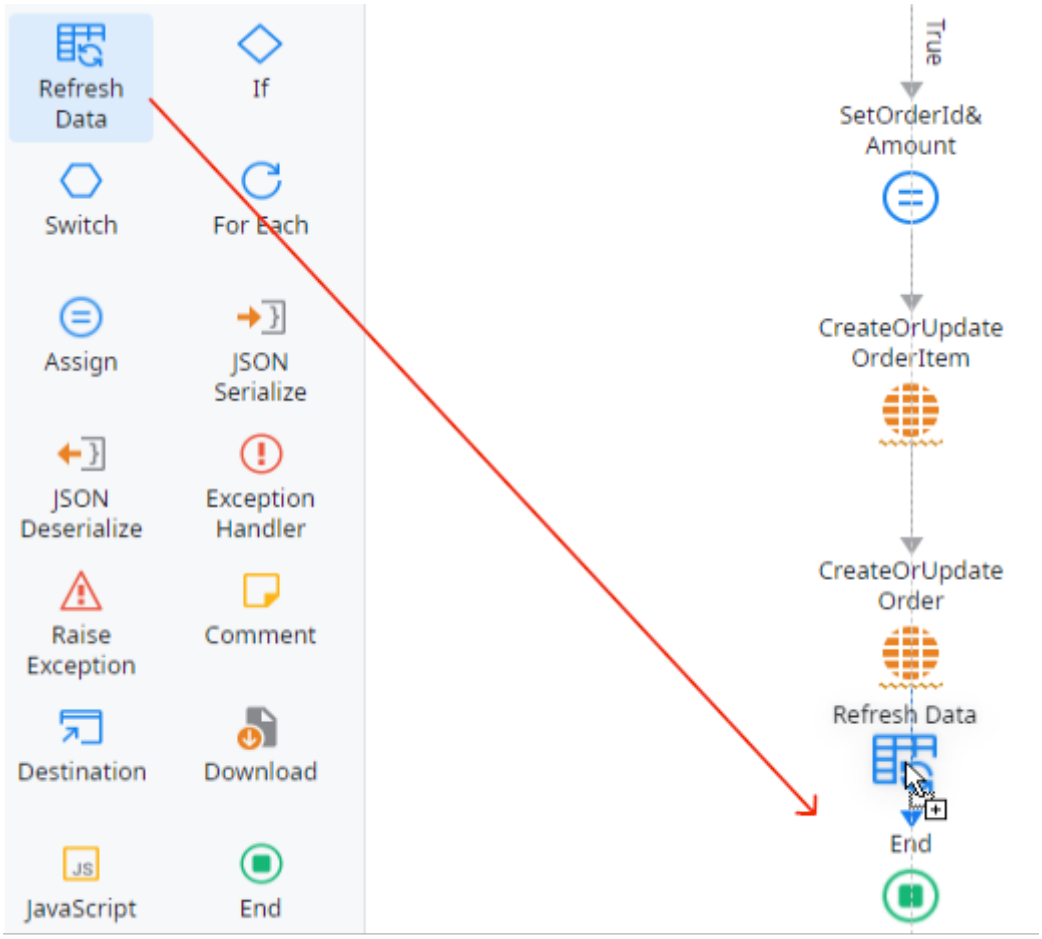


Wrapping up the Logic

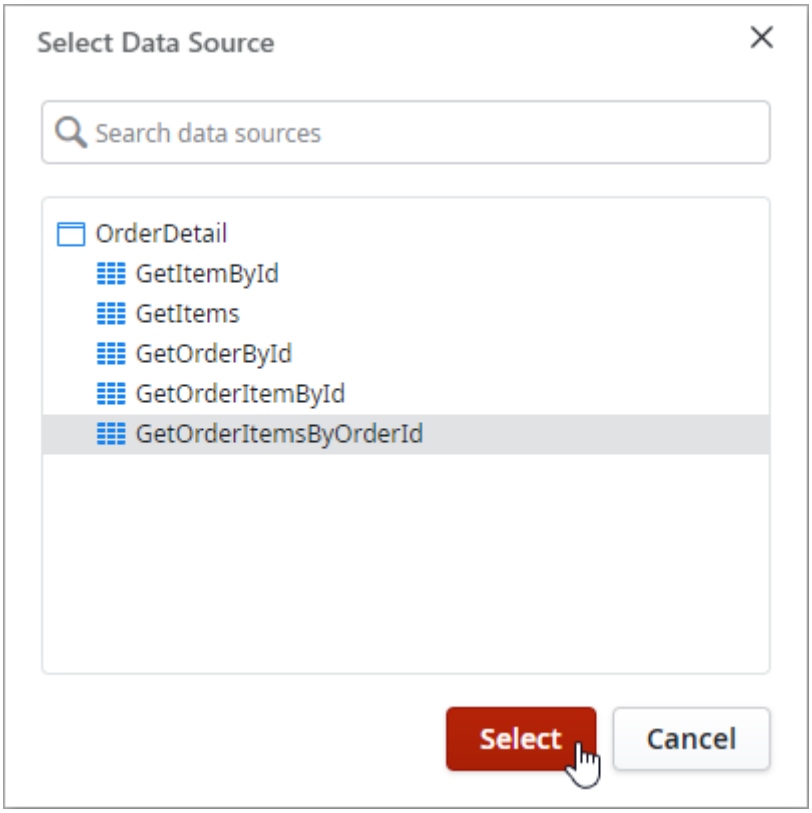
Most of the Action is ready, but there are still some final touches that need to be done. Now that you're finally adding order items, you will need to refresh the data on the Screen to make them appear on the Table of items when the popup closes. Also, the total amount of the order must be updated. And finally, we want the user to see a

message informing that everything went well and the popup is closed. Seems too much? It isn't!

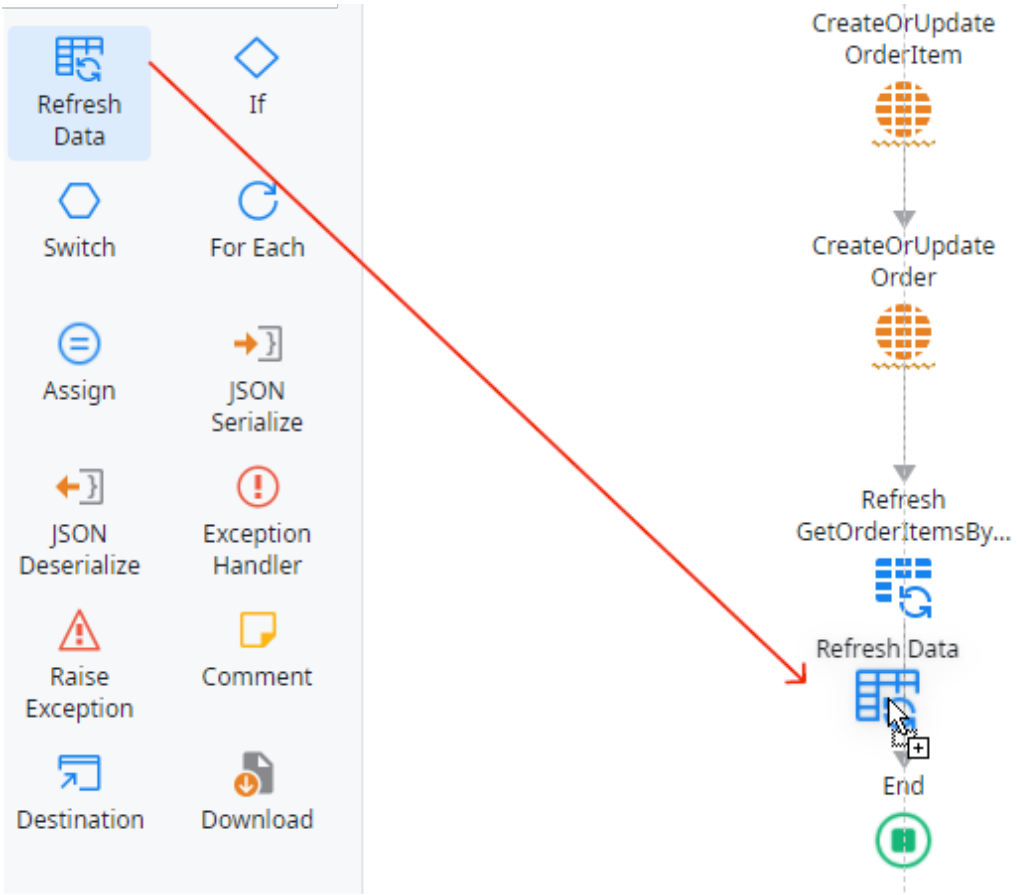
1) Drag a **Refresh Data** element from the left sidebar and drop it under the last Action you added.



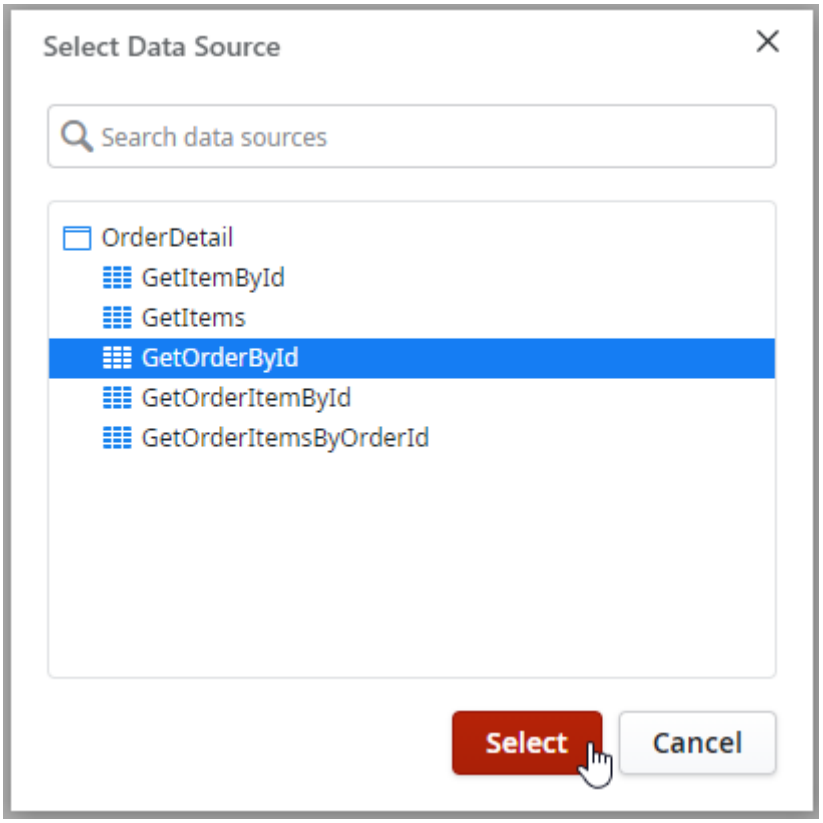
2) Select the Aggregate **GetOrderItemsByOrderId** to refresh the data that is used to populate the Table of Items in the OrderDetail Screen.



3) Drag another **Refresh Data** element and drop it under the last one.

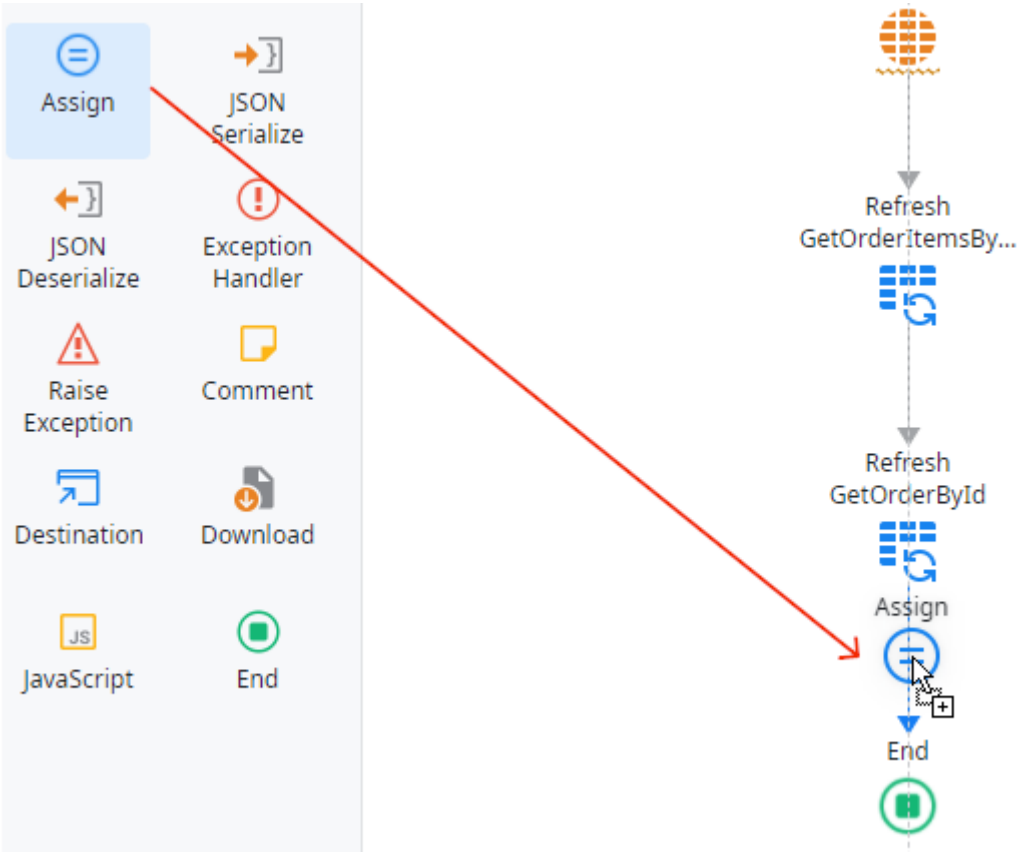


4) Select the **GetOrderById** Aggregate to refresh the order's total amount in the OrderDetail Screen.

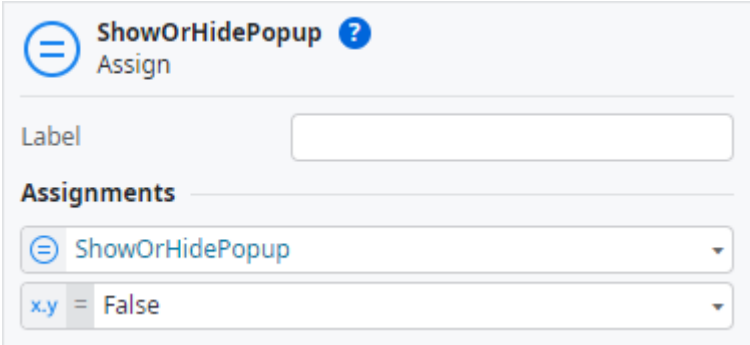


The data is refreshed, now it's time to close the Popup.

5) Drag an **Assign** and drop it between the last Refresh Data and the End element.

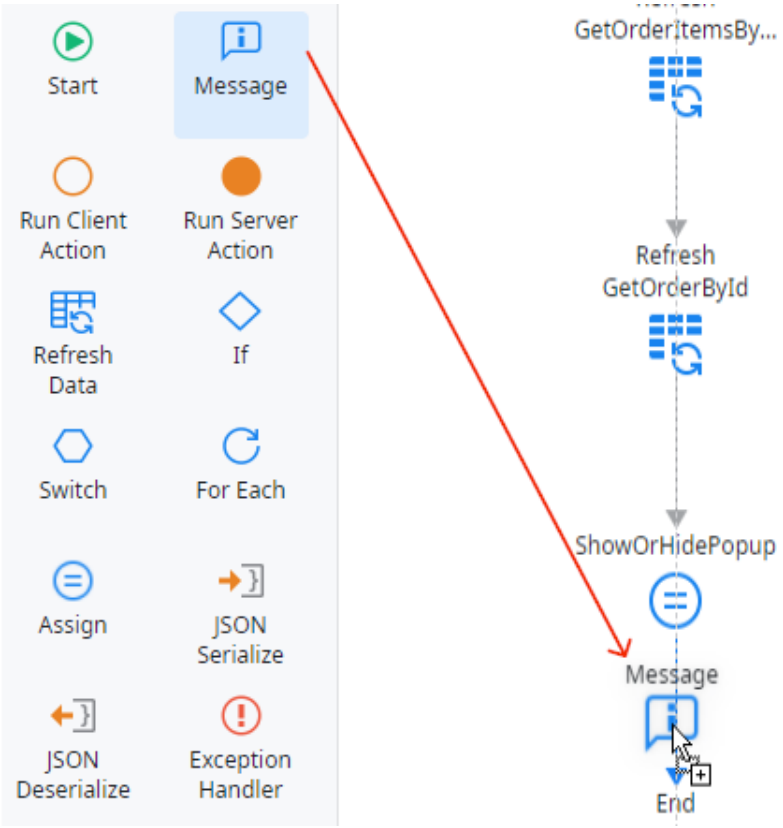


6) Set the **Variable** field to **ShowOrHidePopup**, the Local Variable you created before, and the **Value** field to *False*.

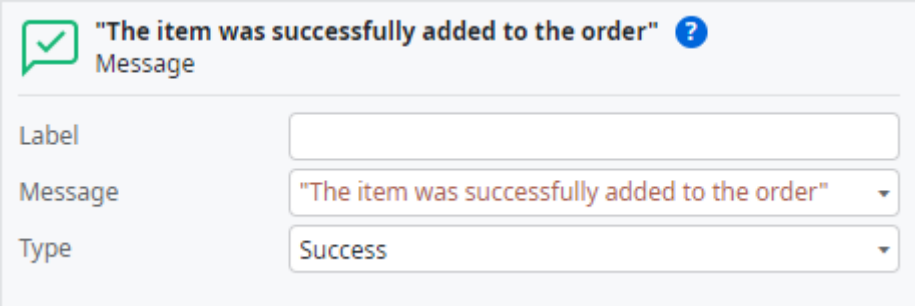


This will close the Popup.

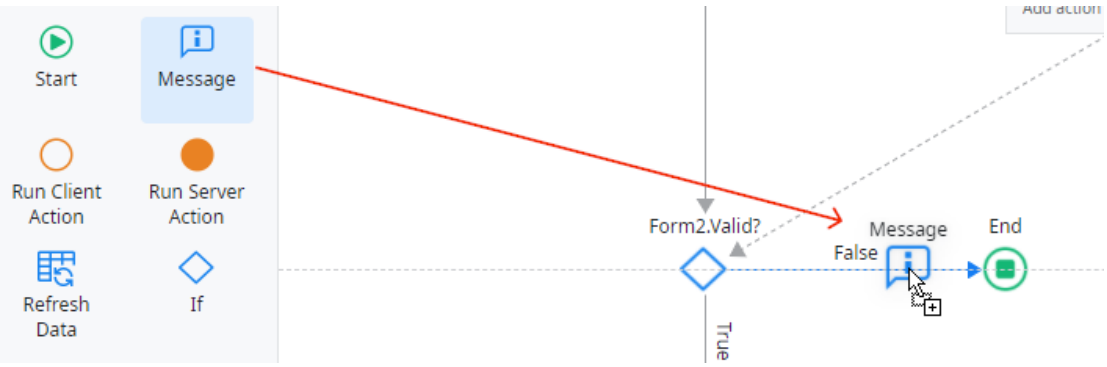
7) Drag a **Message** and drop it under the ShowOrHidePopup Assign.



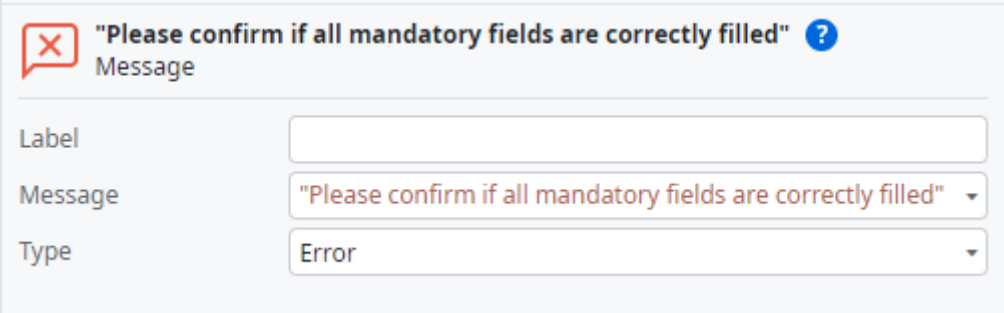
8) Set the **Type** property to **Success** and add the following **Message**: *"The item was successfully added to the order"*.



9) Drag another **Message** element, but this time drop it on the False branch of the Form2.Valid? If (the one on the right side).



10) Set the **Type** property to **Error** and the **Message** to: *"Please confirm if all mandatory fields are correctly filled"*.

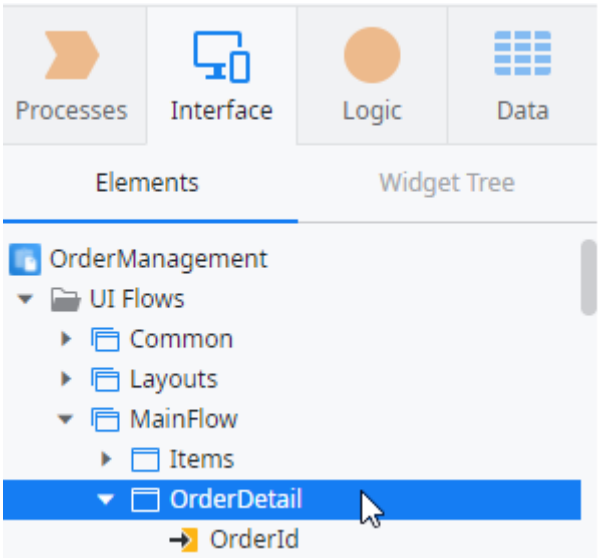


11) Publish the module to save what you've done so far.

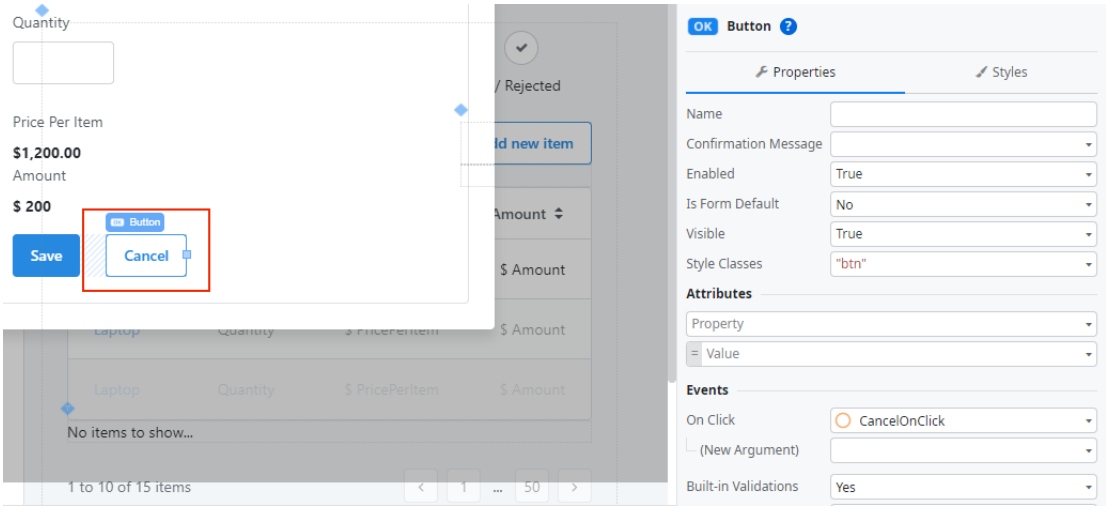
Creating the Logic to Cancel an Order Item

Now that you already implemented the logic for the Save button, it is time to implement the logic for the Cancel button. It is much simpler, you will see!

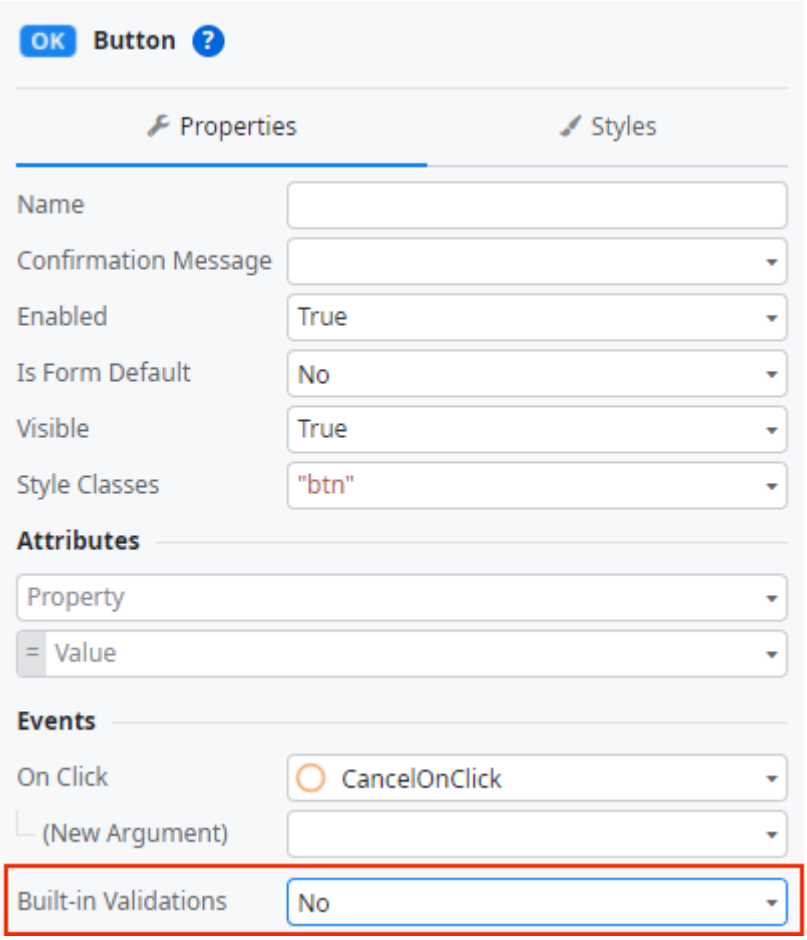
1) Open the **OrderDetail** Screen again.



2) Click on the **Cancel** Button to open its properties.

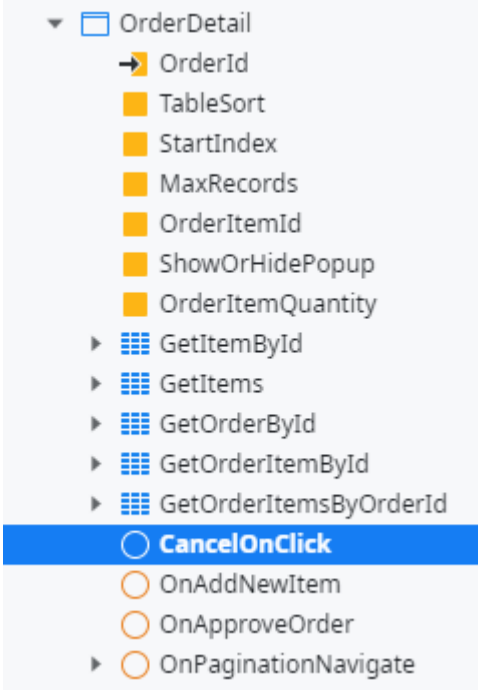


3) Change the **Built-in Validations** property to *No*.

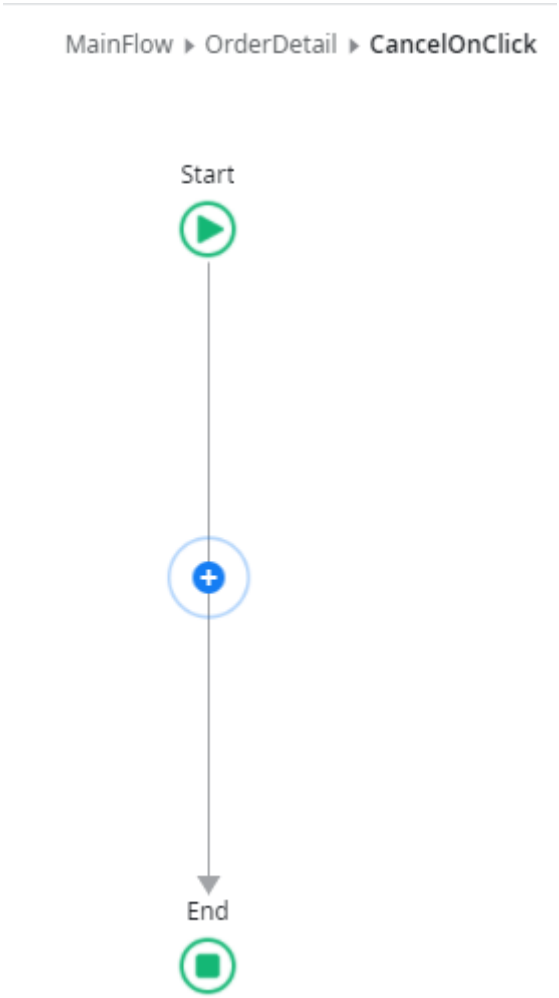


Since this button is to cancel what you're doing, then the built-in validations should not be performed.

4) Double-click on the **CancelOnClick** Action on the right sidebar, inside the OrderDetail Screen, to open it.

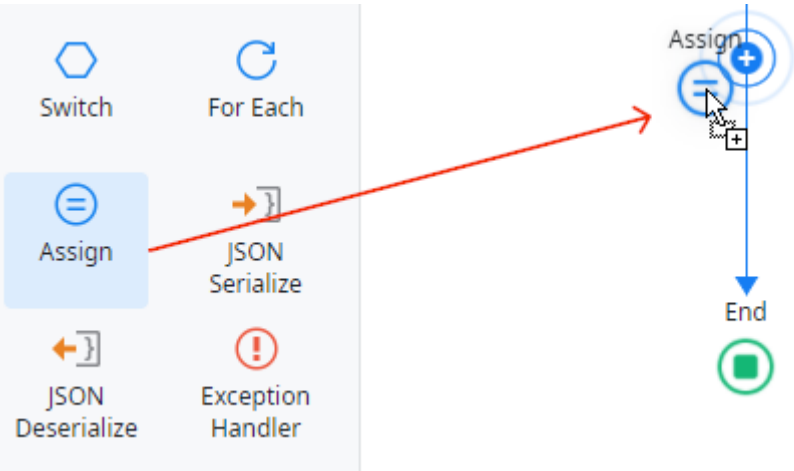


5) Right-click the **If** and delete it. Since this logic is to cancel, we don't want OutSystems to validate any data. Also, delete the End node and the grey message box on the right of the flow.

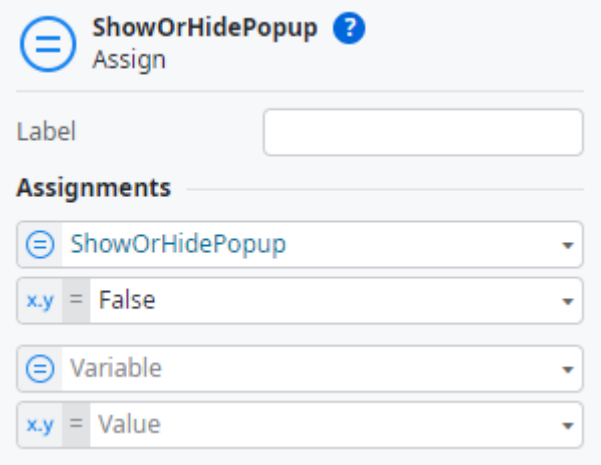


Now, the Popup should close then the user clicks on the Cancel button. And that's it!

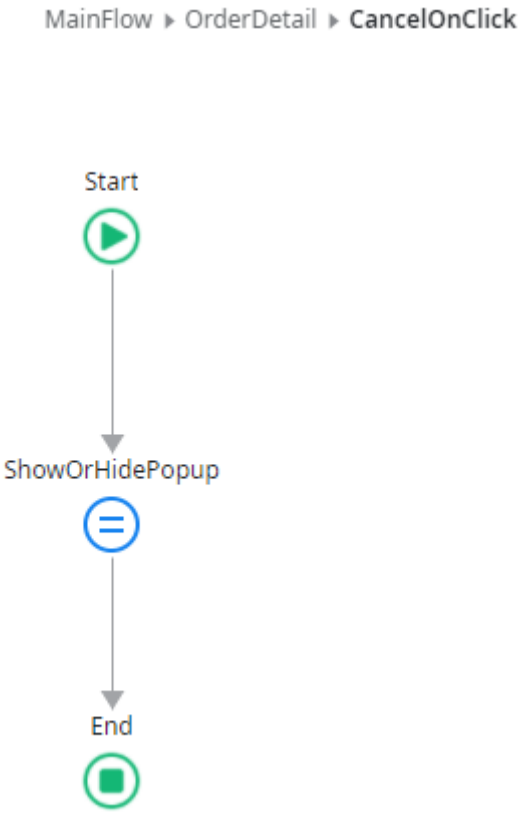
6) Drag an **Assign** and drop it on the Action flow.



7) Set the **Variable** field to the **ShowOrHidePopup** variable and set the **Value** field to *False*.



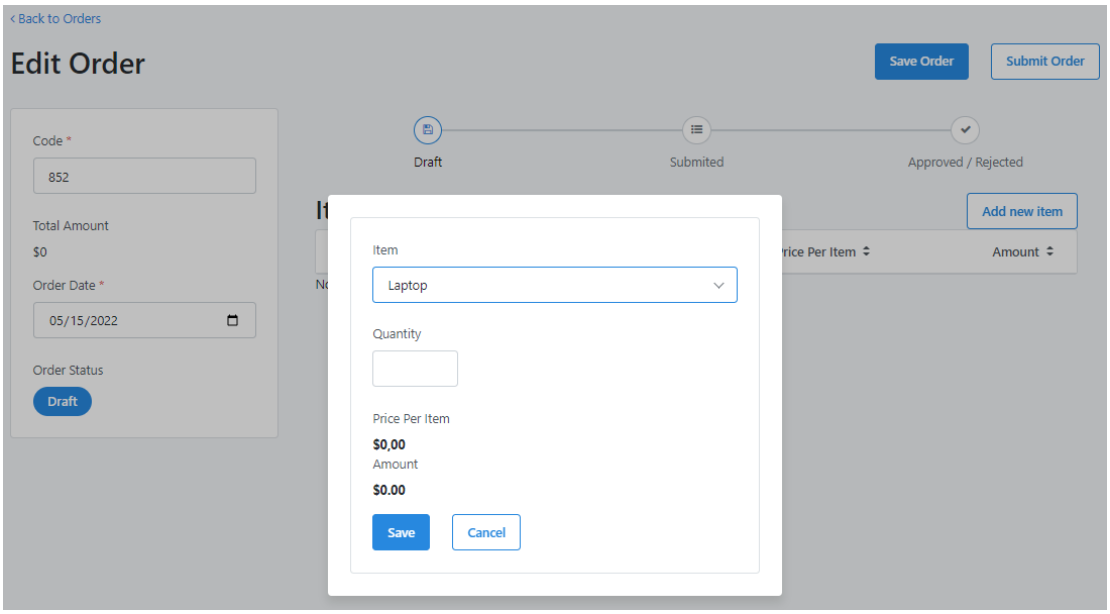
That's it! Your Cancel Action is ready! We told you it'd be easy...



8) Publish the module and open the application in the browser.

9) Try to add items to an order. What happens? The information in the popup is not updating right? That's because you doing all the logic, but not refreshing

the item information in the UI when the user chooses the item in the dropdown.

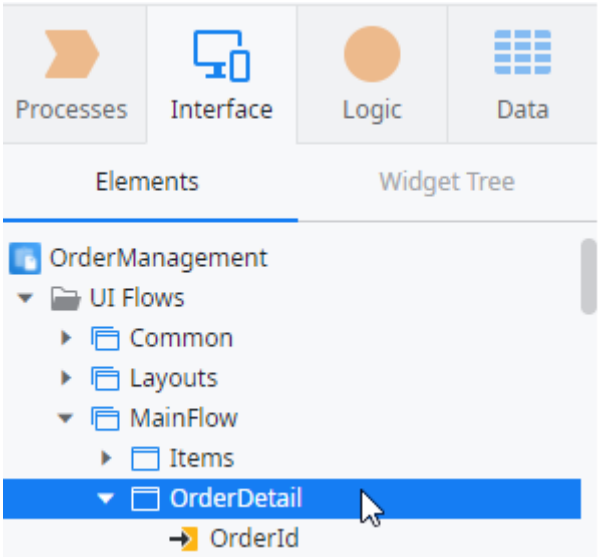


Note: Don't forget that the *Add new item* Button is only visible if the Order Status is Draft.

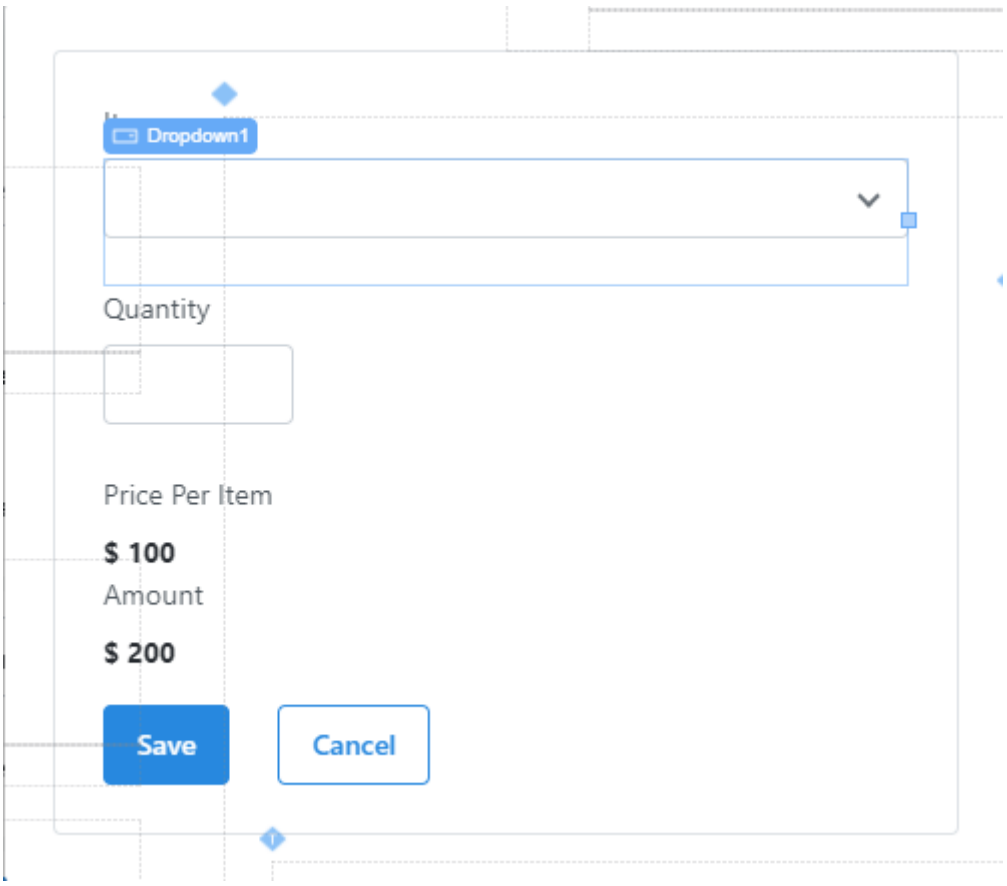
Refreshing Item Data

If an item is selected in the dropdown, the price and amount should be updated. So, you need to refresh the data according to the Item selected in the Dropdown.

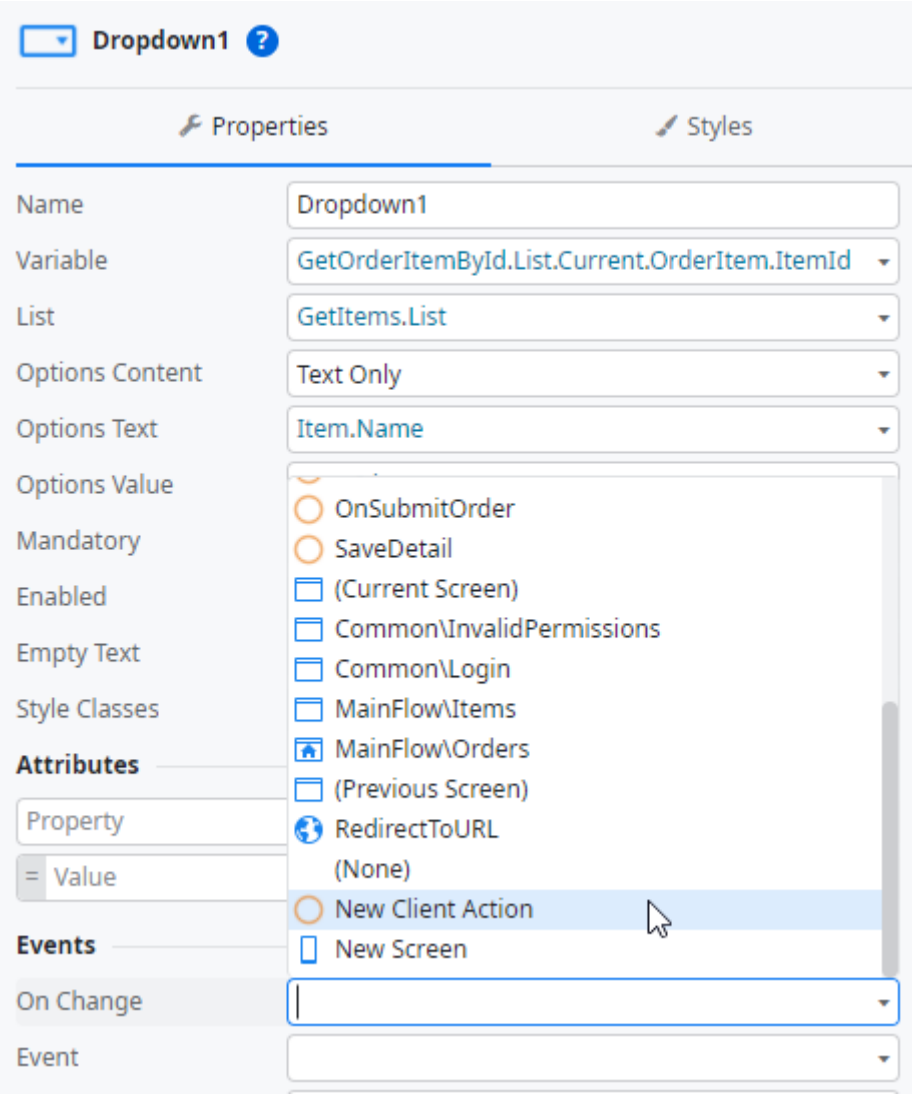
1) Go back to the **OrderDetail** Screen.



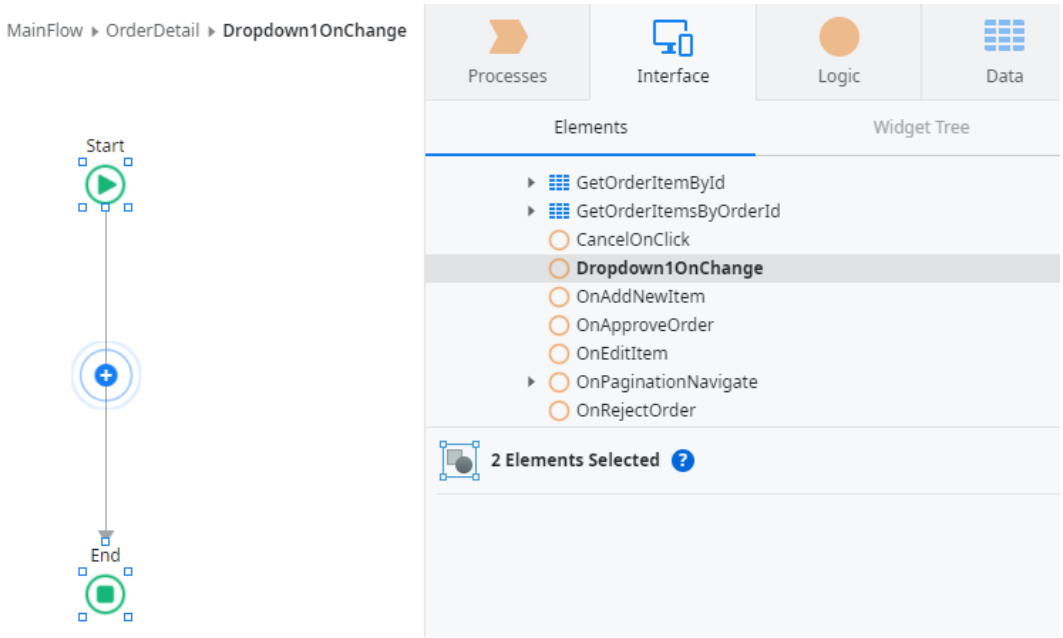
2) Click on the **Items** Dropdown to open its properties on the right sidebar.



3) In the **Events** properties, set the **OnChange** property to **New Client Action**.

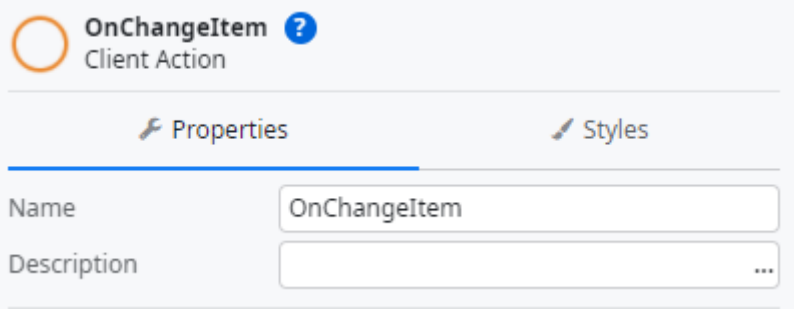


A new Client Action was created with the name *Dropdown1OnChange*.

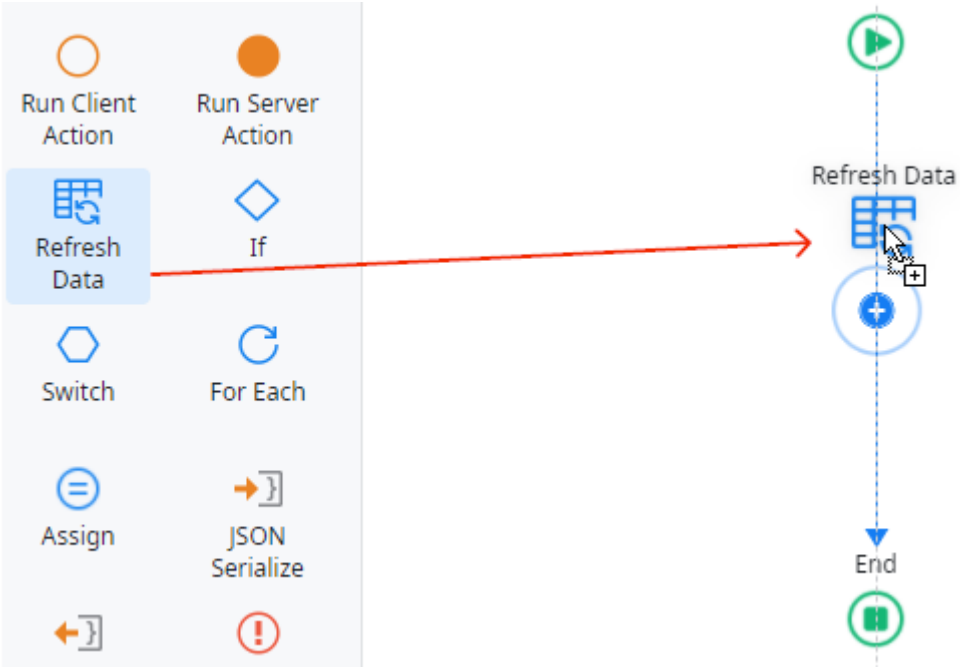


The **OnChange** property sets the behavior that will happen when a new option is selected on the dropdown. So, when a user chooses an item, automatically this Action will run.

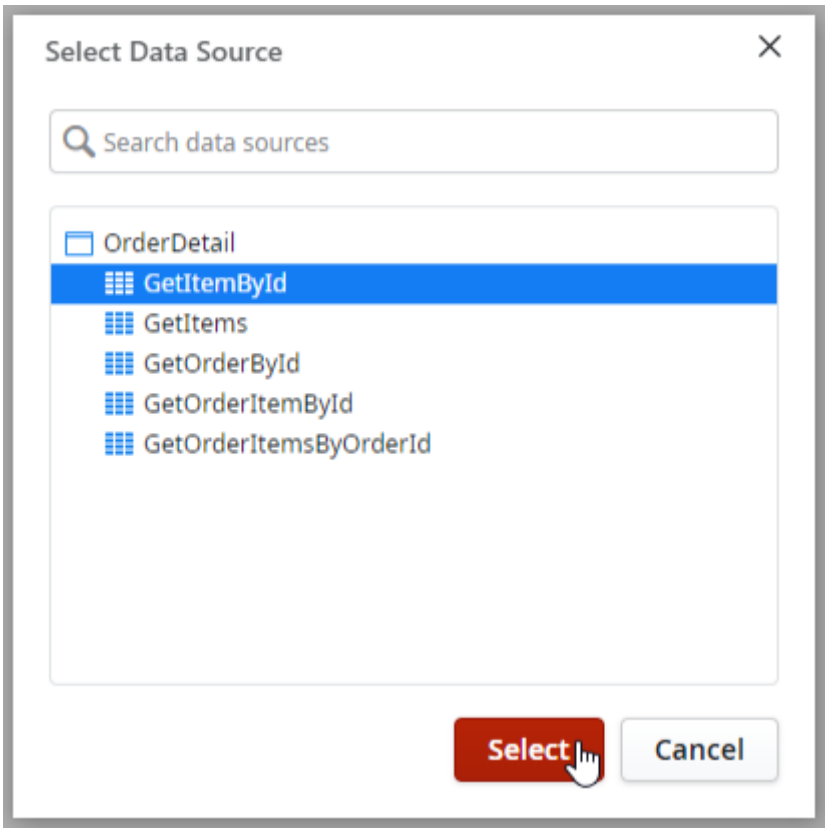
4) Rename the Action to *OnChangeItem*.



5) Inside the Action, drag and drop a **Refresh Data** element from the left sidebar to the Action Flow.



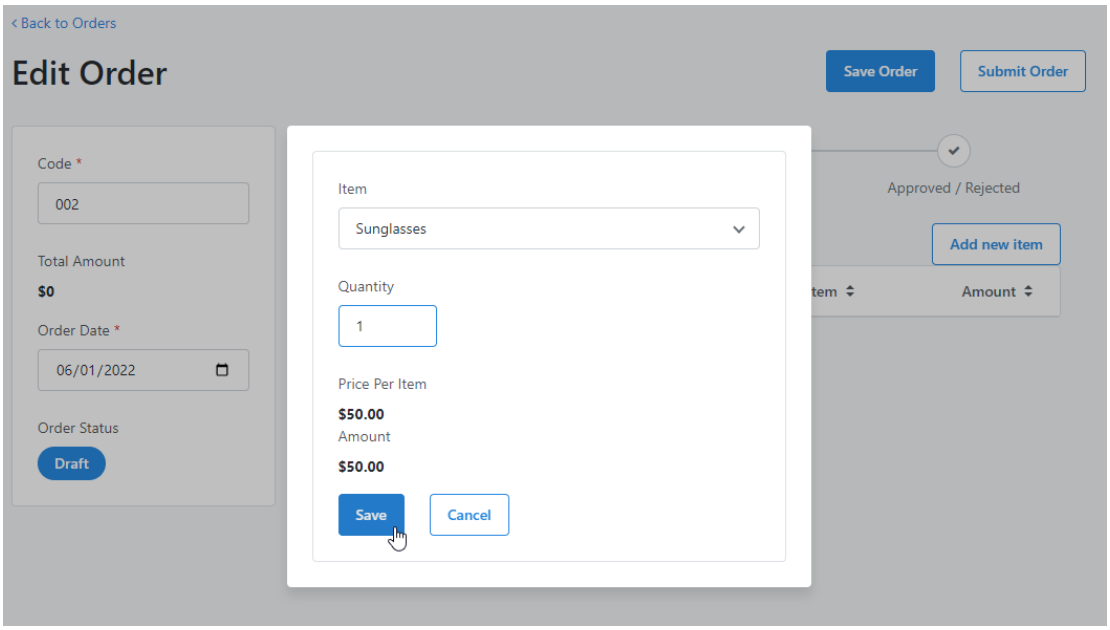
6) Select the **GetItemById** Aggregate.



This will refresh the Item data, after the user chooses it in the dropdown.

7) Publish the module and open the app in the browser.

8) Add Items to the Orders and test if everything is working.



Note: Don't forget that the *Add new item* Button is only visible if the Order Status is Draft.

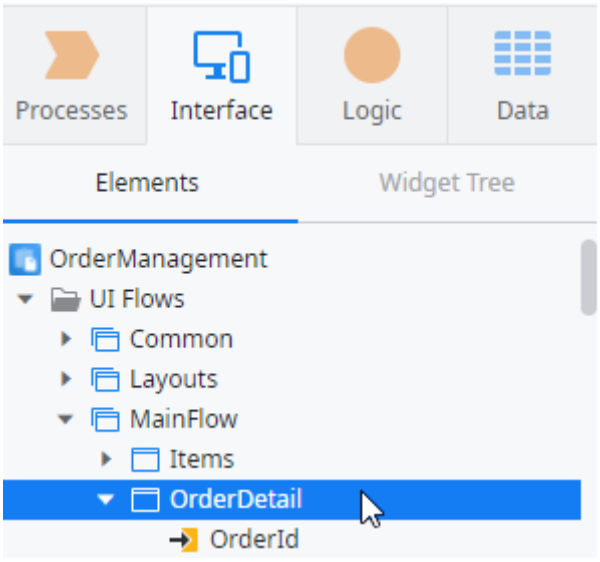
Edit Order Item

You created the logic and interface to add a new Item to an Order. Now you will take one step further and create the logic to allow users to edit an Item that was already added to the Order.

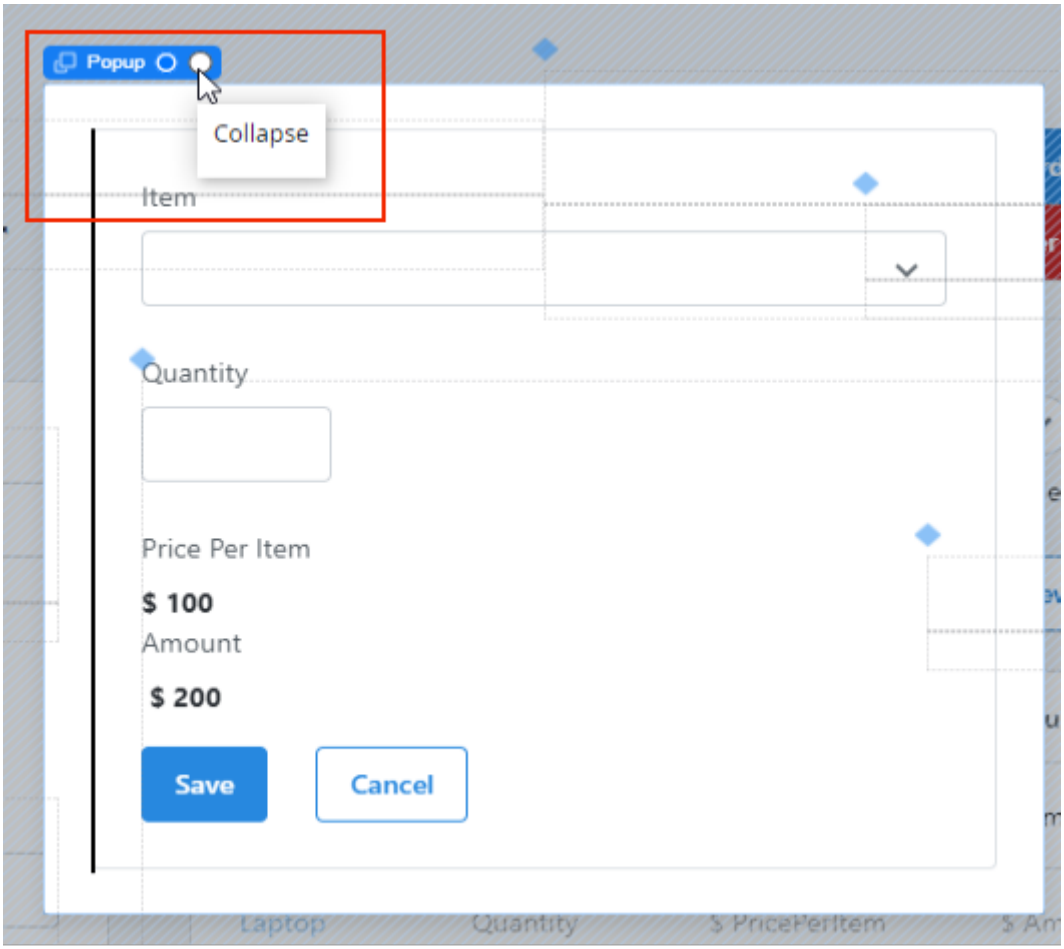
Let's start by adding a link to the name of the item that will open the popup to allow editing the order item.

Opening the Popup for Edition

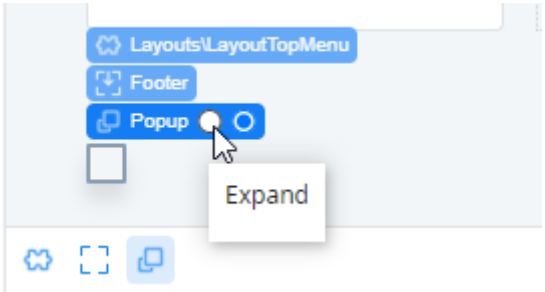
1) Go back to the **OrderDetail** Screen.



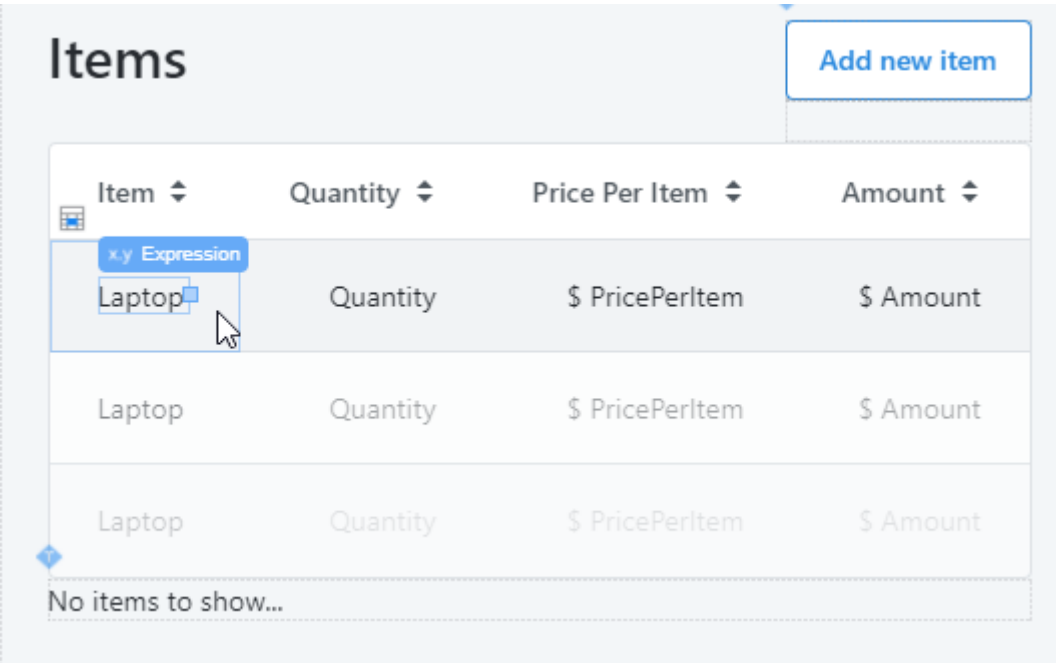
2) The Popup is covering part of the OrderDetail Screen. So, click on the Popup, then click on the second dot to collapse it.



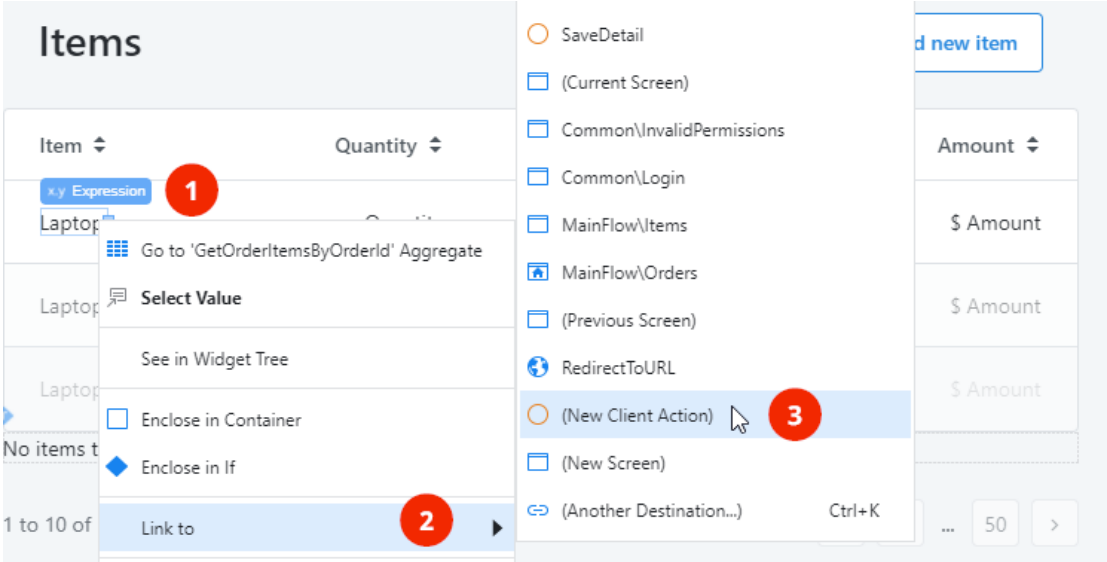
The Popup is still there and can be expanded at any moment by clicking on the first dot.



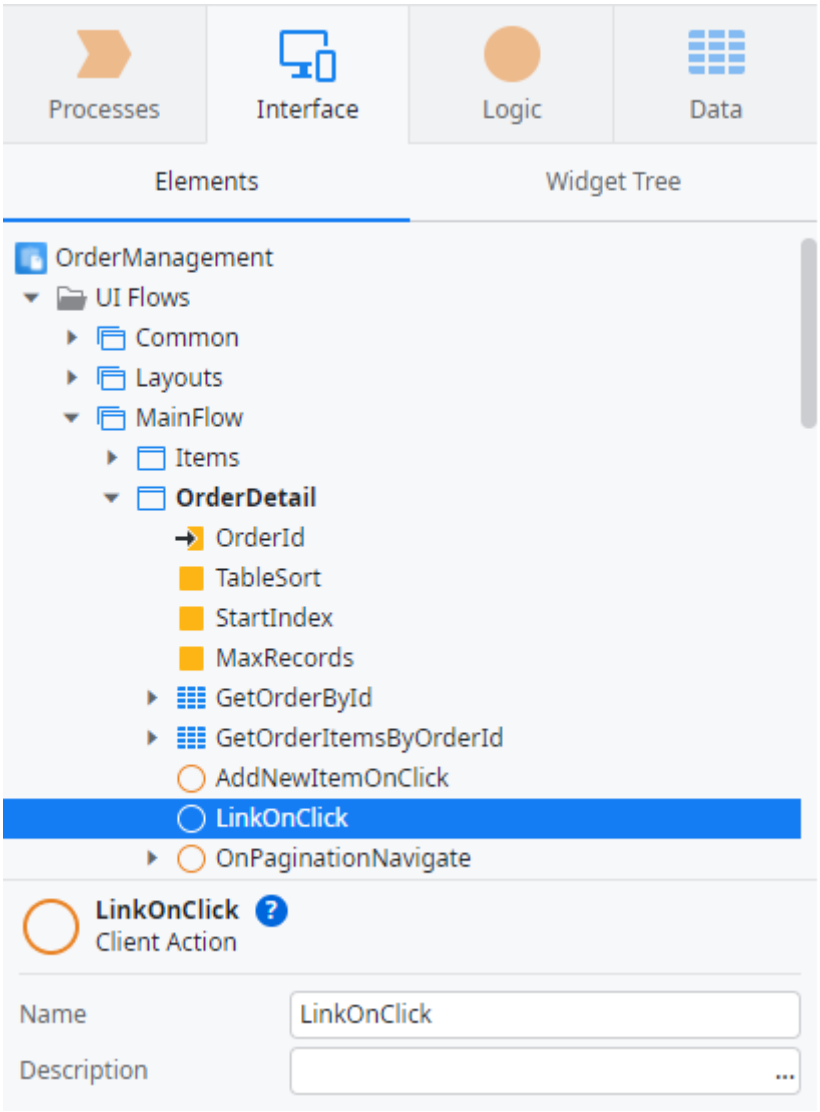
Now you will create a link in the Expression that contains the Item Name in the OrderItem table.



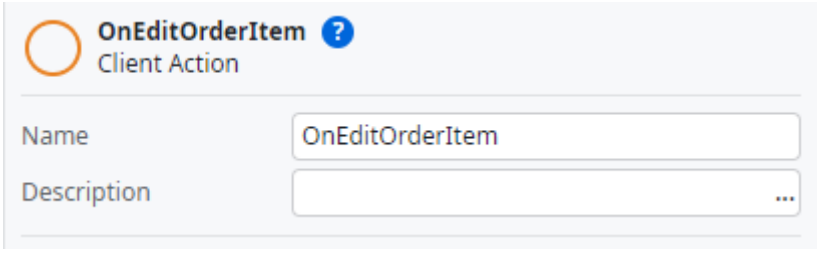
3) Right-click on the Expression, then select the **Link to**, then **(New Client Action)**.



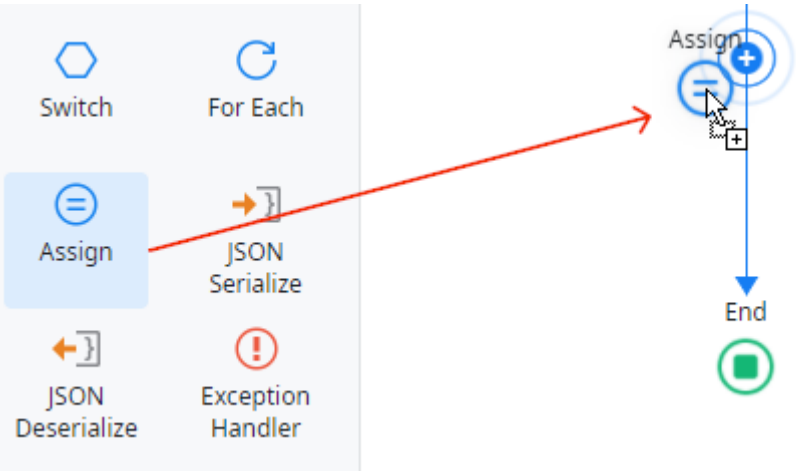
A new Action was created, and you can find it with the other elements of the Screen in the right sidebar.



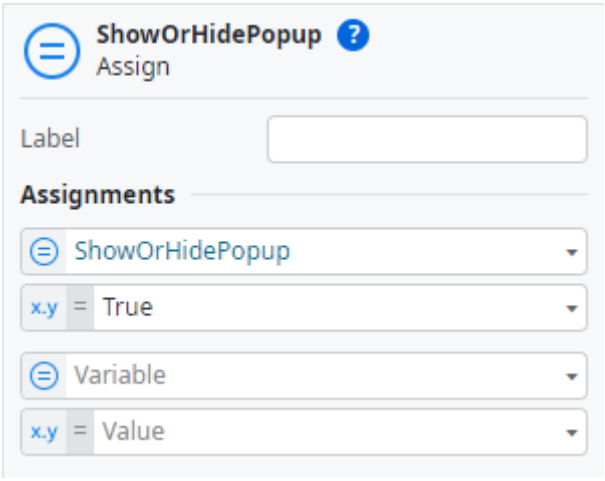
4) Change the **Name** of the Action to *OnEditOrderItem*.



5) Open the Action by double-clicking on it. Then, drag an **Assign** element from the left sidebar and drop it on the flow.



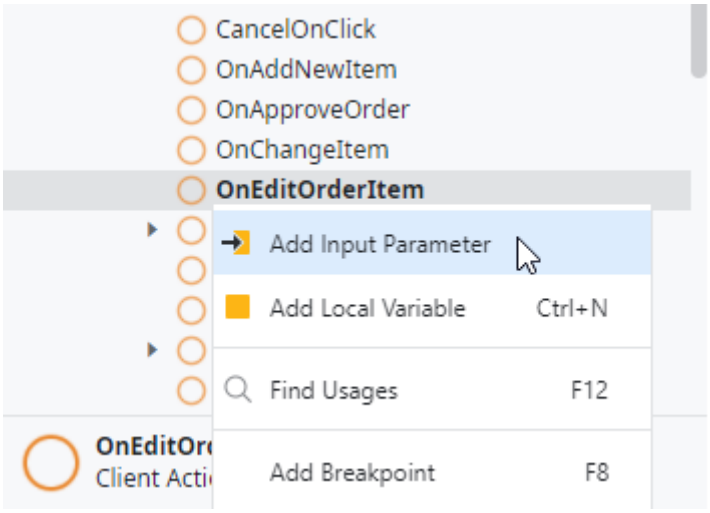
6) Select **ShowOrHidePopup** in the **Variable** and *True* in the **Value** to make sure that the popup opens when the link is clicked.



In this case, the Order Item already exists. So, the Action will need an Input Parameter to receive the OrderItem that will be modified.

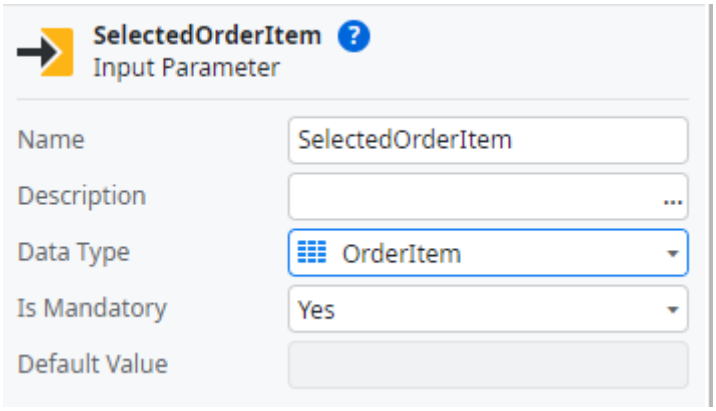
Save the Information from the Current OrderItem

1) Right-click on the **OnEditOrderItem** Action in the right sidebar and select **Add Input Parameter**.



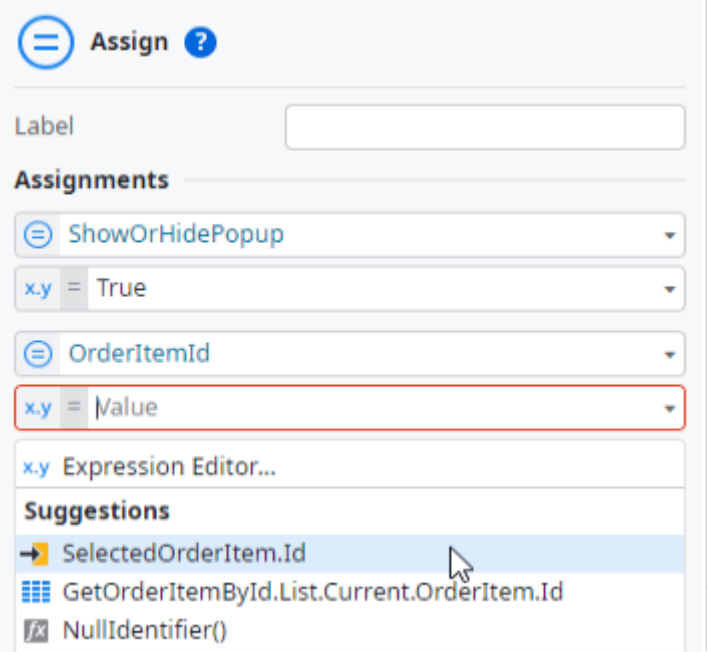
An error will appear because the Action is not receiving any value in the Input Parameter **yet**.

2) Set the **Name** of the Input Parameter *SelectedOrderItem* and set the **Data Type** to **OrderItem**.

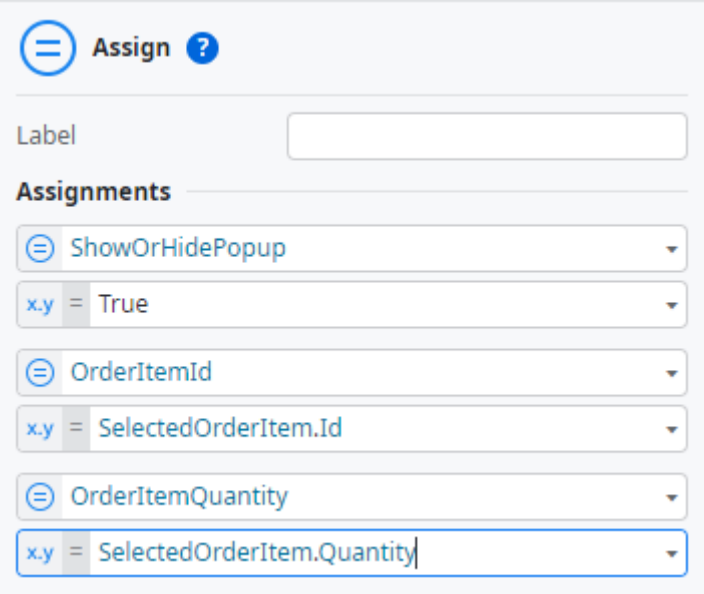


Now, you will use the input parameter to set a value to the Local Variables OrderItemId and OrderItemQuantity. This will save the information of the current OrderItem, before the user actually edits this information. Remember the Total Amount formula where we had to use the new and old quantity? So, that's why we need to save the quantity and the identifier of the OrderItem in the Local Variables.

3) Select the Assign and set the **Variable** field to *OrderItemId* and set the **Value** field to *SelectedOrderItem.Id*.

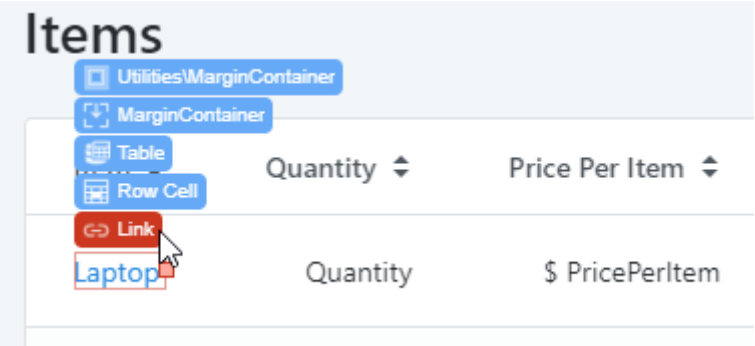


4) Set the next **Variable** field to *OrderItemQuantity* and the **Value** field to *SelectedOrderItem.Quantity*.



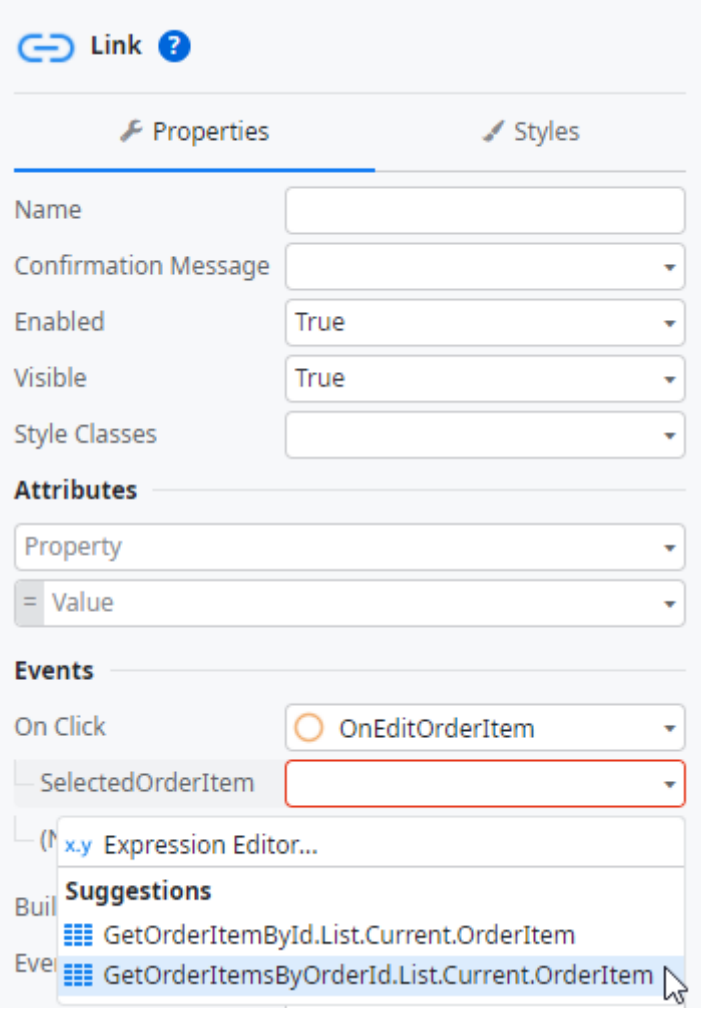
5) Go back to the **OrderDetail** Screen.

6) Click on the **Link** on the Item Table that you just added before.



The error is in the **SelectedOrderItem** property. Notice that this property did not exist before you created the Input Parameter! Now you need to send the OrderItem that is supposed to be edited to the *OnEditOrderItem* Action.

7) Set the **SelectedOrderItem** property to `GetOrderItemsByOrderId.List.Current.OrderItem`.

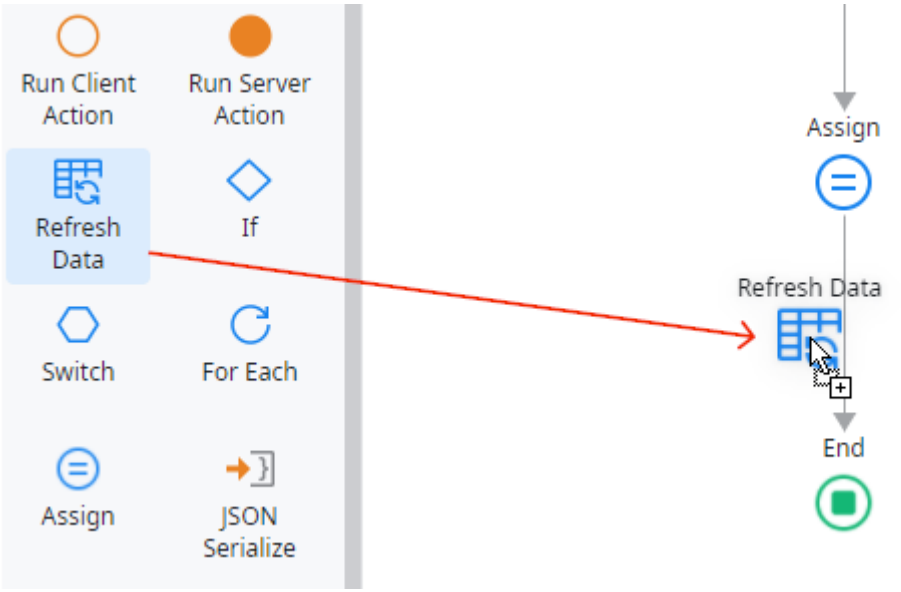


Refresh the Data to Fetch the OrderItem Information

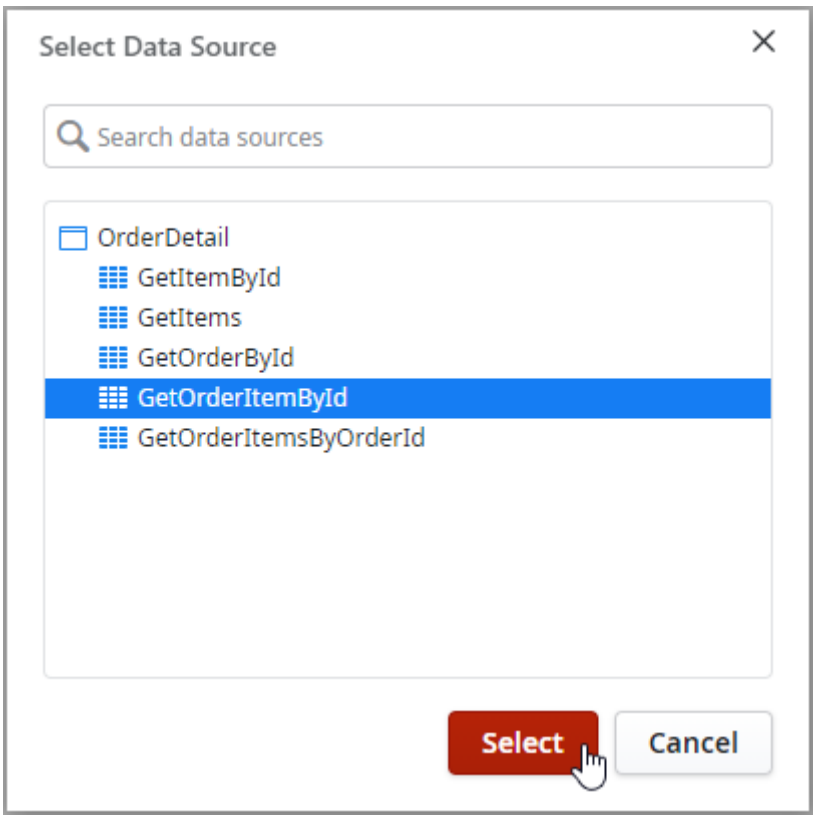
At this point, you opened the popup and you saved the current information of the order item to use it later to recalculate the total amount. But since you're open the Popup, you actually want to see something appearing on the Form. So, you need to fetch the data to be displayed on the Form, which is the info of the OrderItem and the Item itself.

Forget OutSystems for now for a while. Just think about this behavior. Whenever you want to show something on a page, or whenever you change an information that influences some data, it is important to fetch fresh data from the database, so we can get the most up to data information on the app's screens. That's why you have been refreshing data in multiple places.

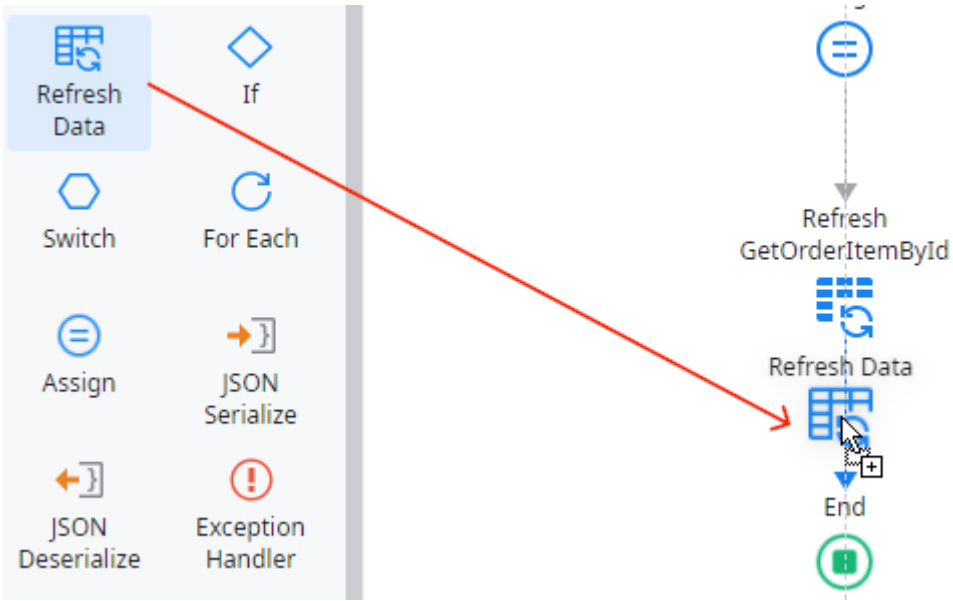
1) Double-click the **OnEditOrderItem** to open it. In the Action, drag a **Refresh Data** element and drop it below the Assign.



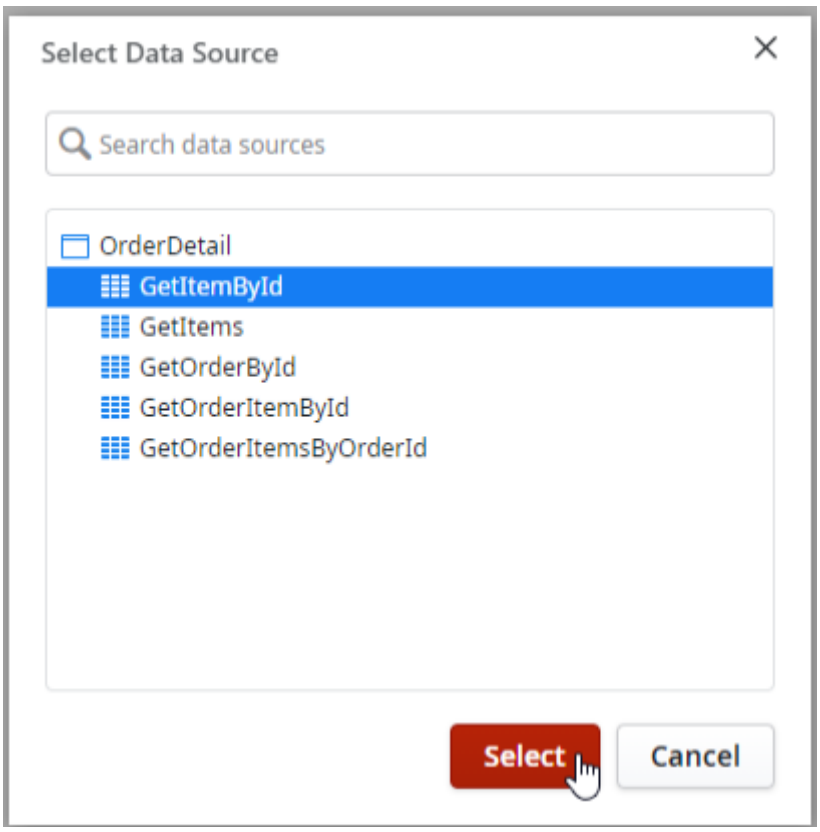
2) In the new dialog, select the Aggregate **GetOrderItemById**.



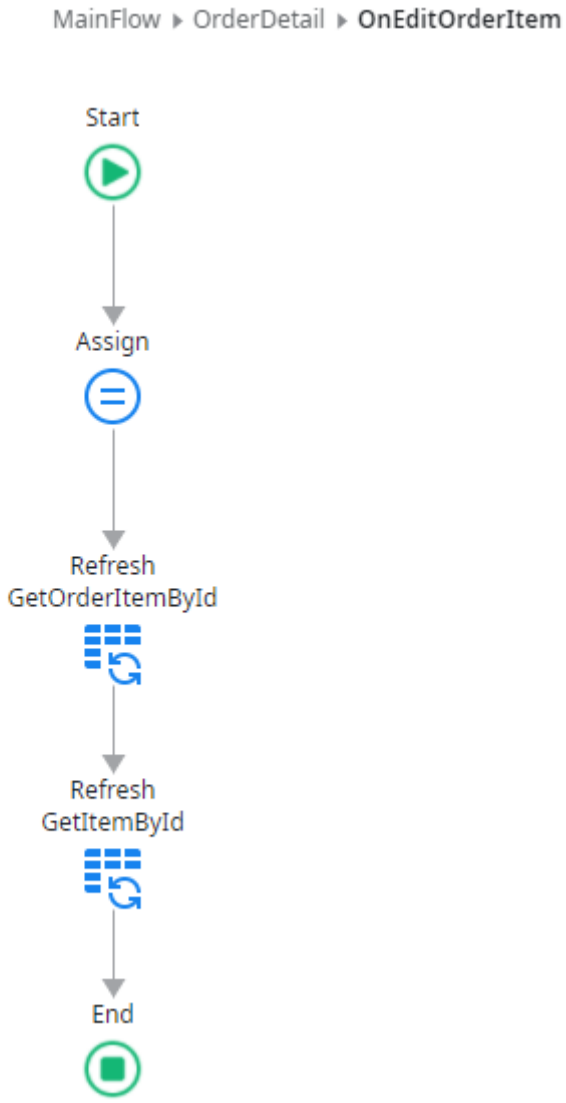
3) Drag another **Refresh Data** and drop it under the previous one.



4) Select the **GetItemById** Aggregate.



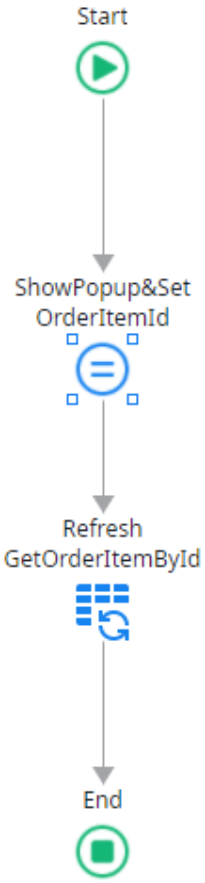
Your Edit Order Item Action is ready!

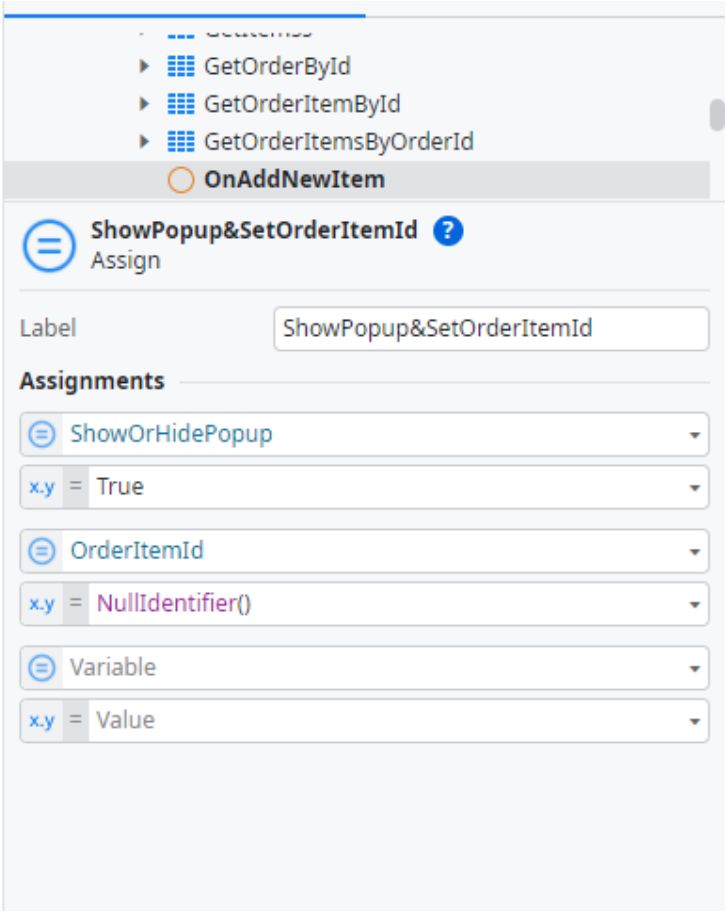


Now, when you edit the order item and click on the Save button, with the logic you already implemented, you have the new order item information available and the database will be properly updated.

Revisiting the Add New Item Logic

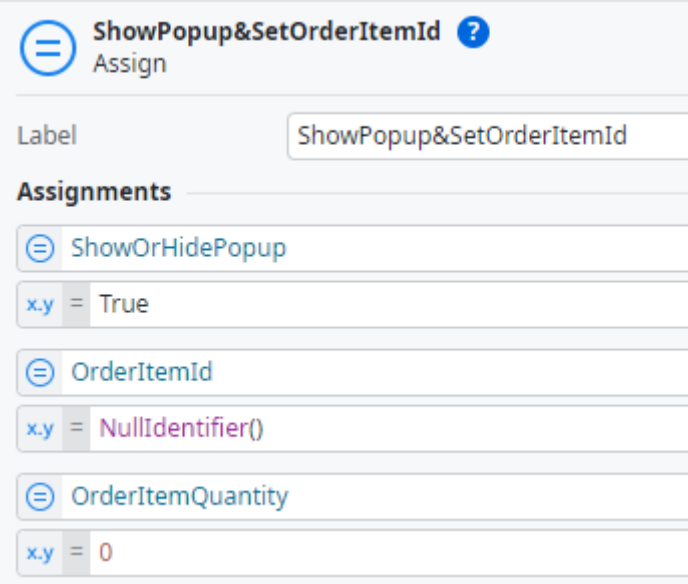
We're almost done! Remember the Add new Item logic? We have an Assign element that resets the OrderItemId to *NullIdentifier*, since we are adding a new item. The OrderItemId is useful to save the previous order item identifier, when we are editing.





So, after this, you created the **OrderItemQuantity** Local Variable and we have to follow the same logic and reset it.

1) Open the **OnAddNewItem** and select the **ShowPopup&SetOrderItemId** Assign. Set the next empty **Variable** field to **OrderItemQuantity** and the **Value** field to *0*.

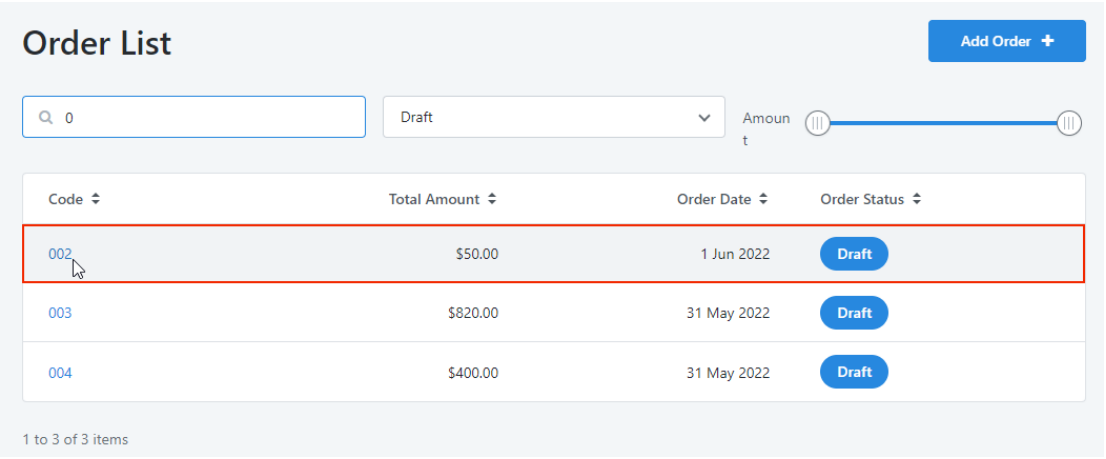


2) Publish the module and open the application in the browser.

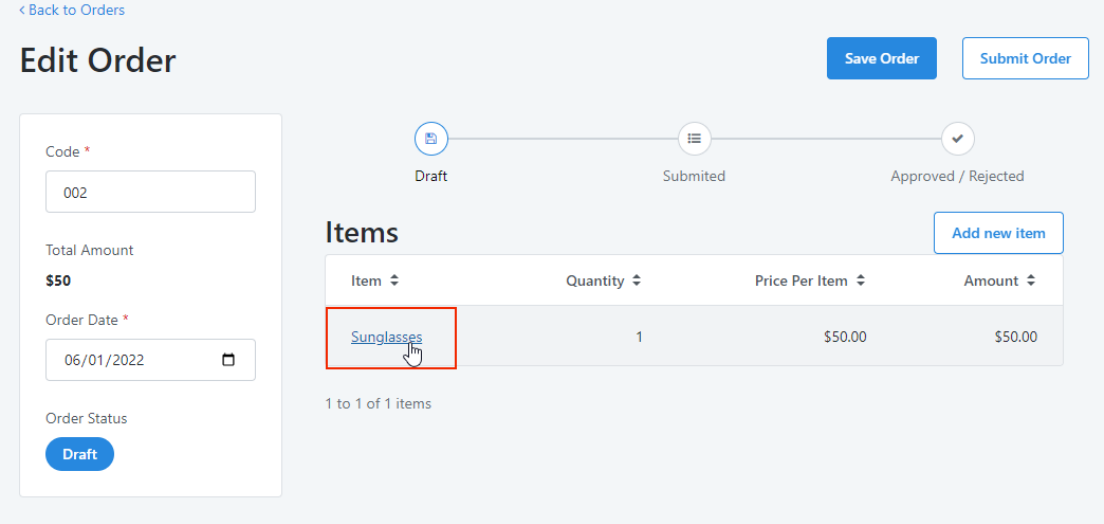
Testing the app

At this point, a user is able to add items to an Order and edit them. So, let's test it out!

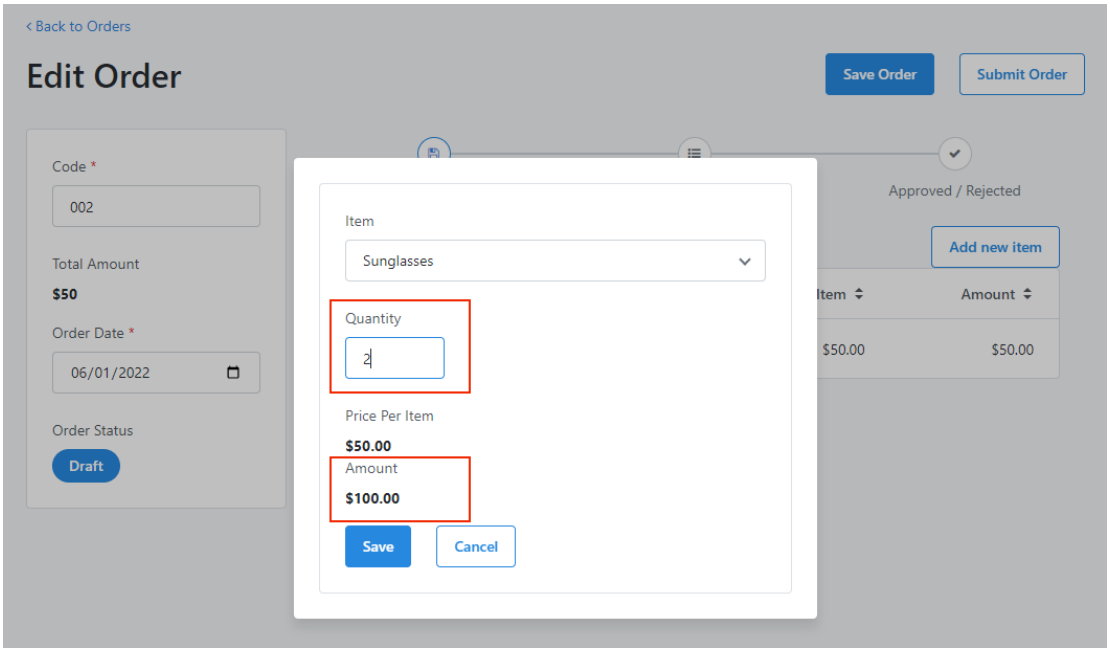
1) In the Orders Screen, click on the code of any Order with the Draft Status.



2) Click on an Item that is already added to the Order.



3) Change the quantity.

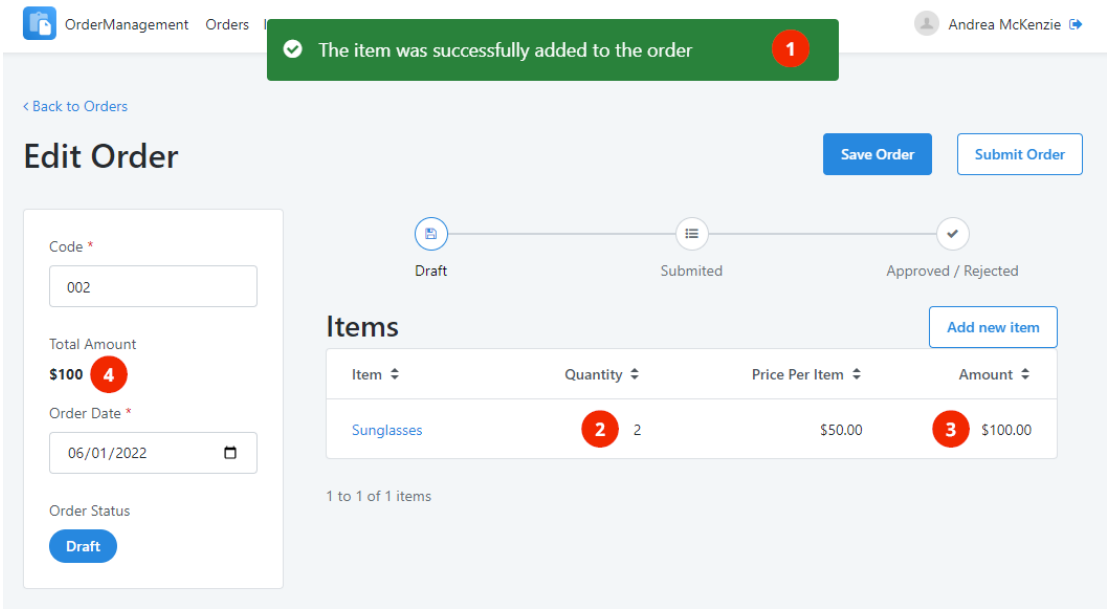


4) Add new items and check if the OrderItem Amount is correct, and also if the Total Order Amount is correct!

If the edit item functionality is working properly, you will see the following signs:

- Success Message
- Quantity of the Item has changed
- Order Amount updated

- Total Amount updated



Wrapping up

Congratulations on finishing this tutorial. The wizard is fully implemented! With this exercise, you had the chance to go through some essential aspects of OutSystems and get to know more about the platform. At this point you already have a functional app that allows users to create, edit, submit, approve and reject orders!

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) Popup
2) Designing Screens
3) Refresh Data
4) Logic Course
5) Advanced Aggregates

See you in the next tutorial!