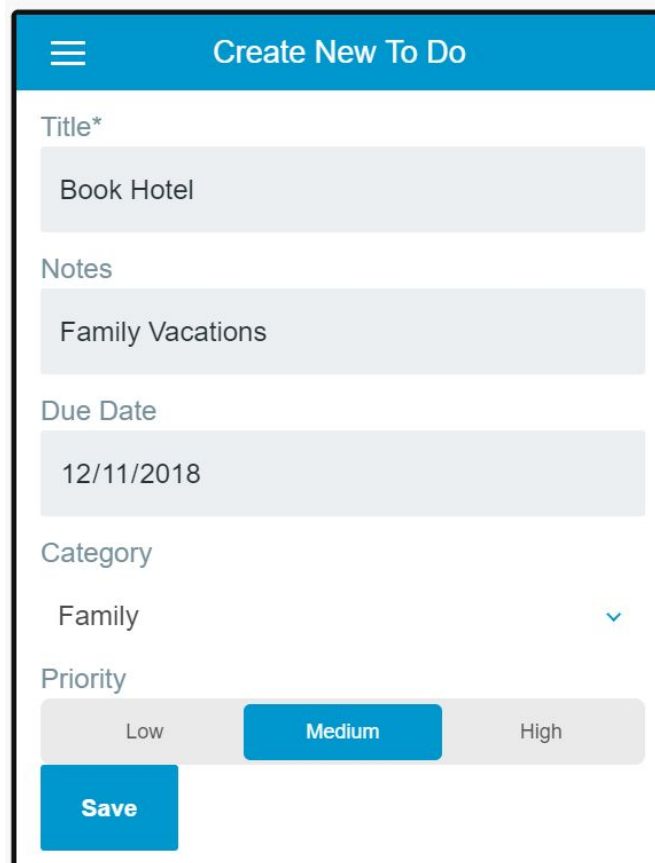


List and Detail Screens Exercise



☰ Create New To Do

Title*

Book Hotel

Notes

Family Vacations

Due Date

12/11/2018

Category

Family ▾

Priority

Low Medium High

Save

Table of Contents

| | |
|-------------------------------|-----------|
| Table of Contents | 2 |
| Introduction | 3 |
| Create a List Screen | 4 |
| Create a Detail Screen | 19 |
| Link the Screens | 30 |
| Publish and Test | 33 |
| End of Lab | 35 |

Introduction

In the previous exercise lab, we created a data model for the ToDo application, in the `ToDo_Core` module. Now, we will use the Entities created and build two Screens: the **ToDo**s Screen, to list all the **ToDo**s of a user, and the **ToDoDetail** Screen, to allow creating and updating a ToDo in the database.

First, we will add a reference to the Entities from the `ToDo` module.

Then, we will create the **ToDo**s Screen, with an Aggregate to fetch all the **ToDo**s of a certain user and a List widget to display all of them.

The **ToDoDetail** Screen will follow, which will also have an Aggregate to fetch the specific **ToDo** we want to edit. The Screen will then have a group of Inputs inside a Form, to allow users to submit information about the **ToDo**. Then, by clicking on a button that data will be submitted to the server.

Finally, at the end of the exercise lab, we will link the two Screens together, to enable an easy navigation between the Screens.

In this specific exercise lab, we will:

- Reference the Entities from the `ToDo` module
- Build the To Do list and detail Screens
- Create navigation links between the two Screens

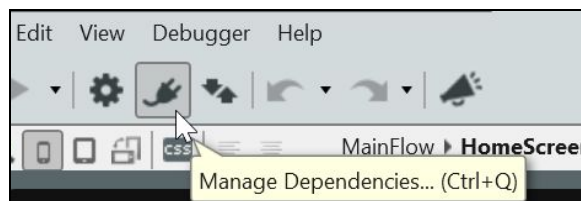
Create a List Screen

In this part of the exercise lab, we will create the first Screen of the application, the Todos Screen. This will be the Home Screen of the app, and it will list all the Todos of the logged in user. For that we will use an Aggregate to fetch all the Todos and a List widget to display them.

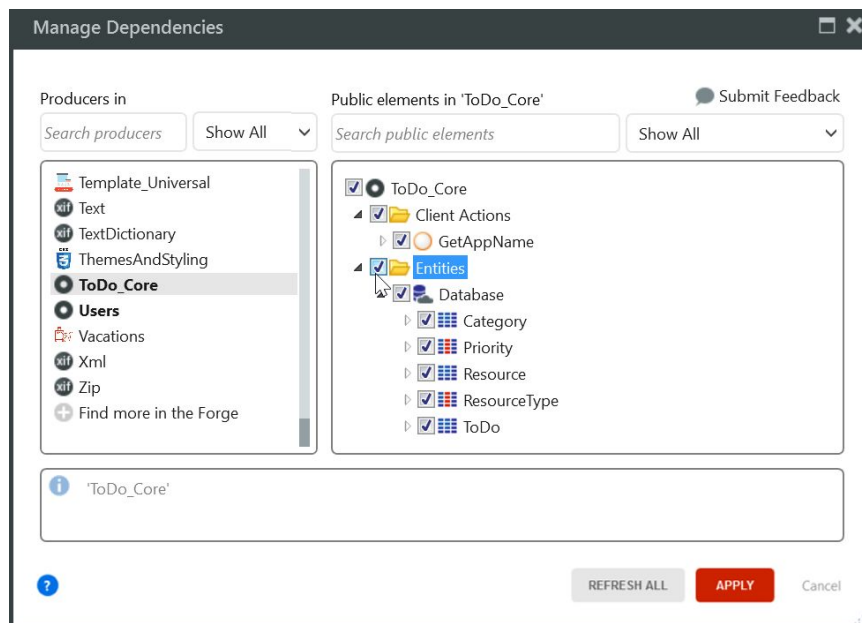
Also, to make sure that the Screen always displays some information to the user, even if there are no Todos to display, we will also add BlankSlate Patterns to the page, displaying some images and messages to inform the user what is happening.

Before we actually start creating the Screen, we need the Entities defined in the previous lab, in the `ToDo_Core` module. For that, we need to add a dependency to the Entities in the `ToDo` module, just like we did before for the `GetAppName` Action.

- 1) In the `ToDo` module, reference all the Entities created in the `ToDo_Core` module. These Entities will be used to build our UI.
 - a) In the `ToDo` module, click on the **Manage Dependencies** icon.



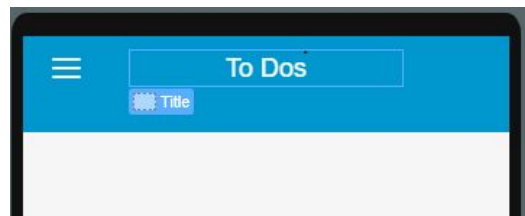
- b) In the Manage Dependencies dialog, select the **ToDo_Core** module on the left, and then select all Database Entities. Click **Apply** to add the dependencies.



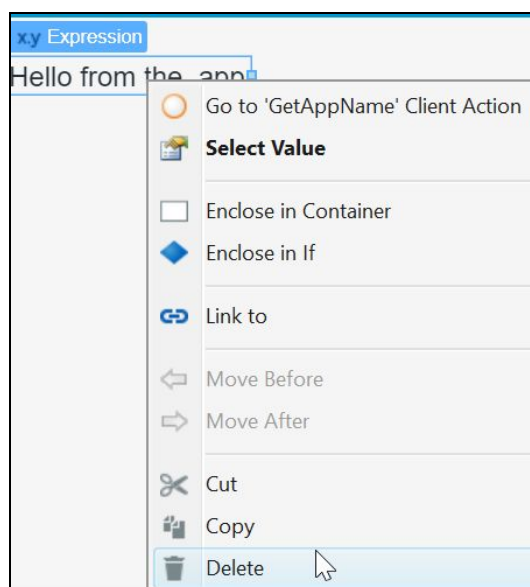
- 2) Change the **HomeScreen** to be the new *ToDo*s Screen and make it empty.
 - a) Switch to the **Interface** tab, expand the **MainFlow** and select the HomeScreen.
 - b) In the properties area, change the Screen **Name** to *ToDo*s.

| ToDos Screen | |
|---------------|-------------------------------------|
| Name | ToDos |
| Description | ... |
| Title | |
| Roles | |
| Anonymous | <input type="checkbox"/> |
| Registered | <input checked="" type="checkbox"/> |
| Events | |
| On Initialize | ▼ |
| On Ready | ▼ |
| On Render | ▼ |
| On Destroy | ▼ |

- c) Double click the **ToDo**s Screen to open it, select the **Title** placeholder at the top of the Screen and type *To Dos*. This will be the Title of the Screen that the end-users will be able to see.

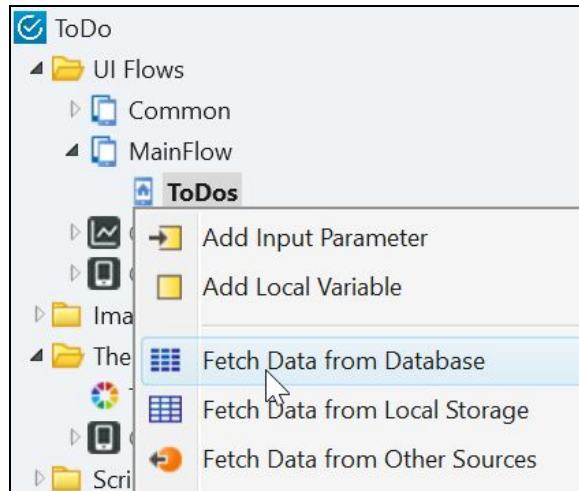


- d) Select the "Hello" Expression created before and delete it.

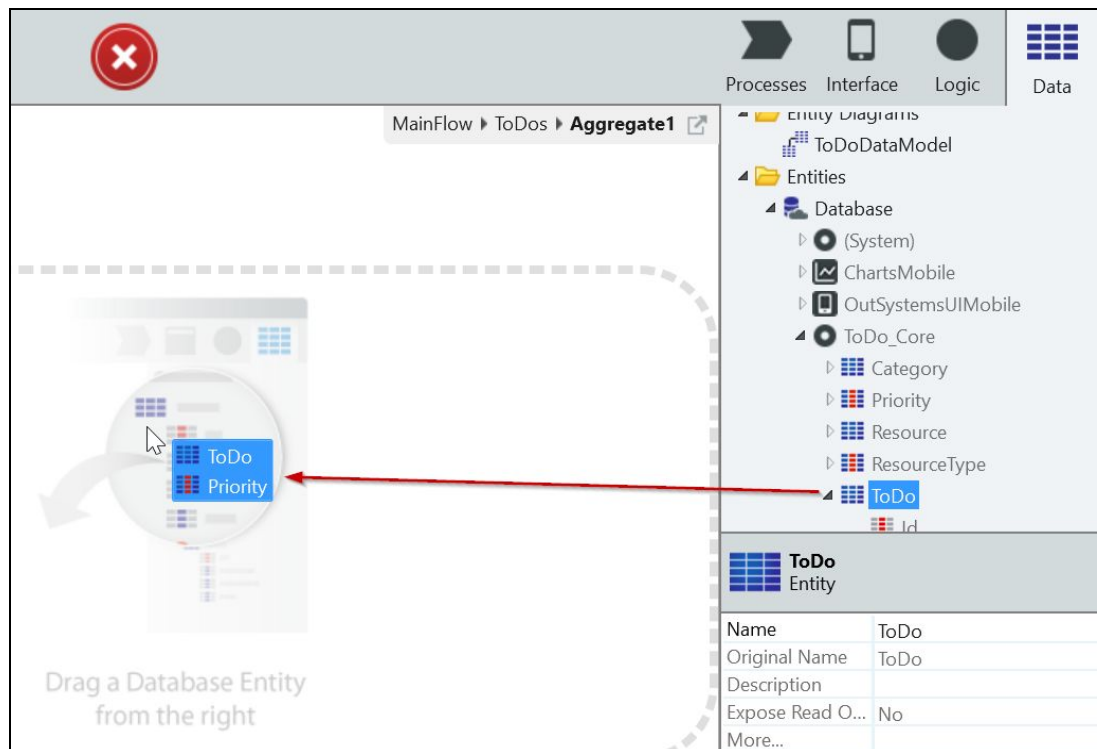


3) Fetch the Todos of the logged in user from the database, using a Screen Aggregate. Don't forget to filter it by the logged in user. This will fetch the necessary data to be displayed in the Todos Screen.

- a) In the Interface tab, right-click the **ToDo** Screen item and select *Fetch Data from Database*.



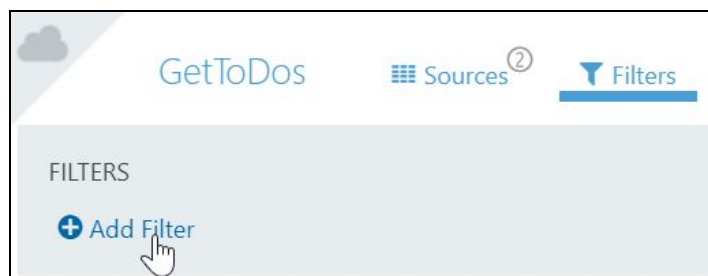
- b) A new Aggregate named **Aggregate1** was created and opened. From the Data tab, locate the ToDo Entity, under the ToDo_Core, and drag it into the Aggregate.



Notice that the Priority Static Entity is automatically added as well. This happens because there is a relationship between the Entities, and the Priority is a Static Entity. This way, the Aggregate has access to the **Label** of the Priority. Remember that the ToDo Entity has a *PriorityId* field, which is a number, so the Label is useful to distinguish what is the Priority of a given ToDo.

NOTE: When a second Entity is added to an Aggregate, the Join between the Entities is also automatically created, under the **Sources** tab of the Aggregate. At any time, we can remove a Source (which means removing the Entity from the Aggregate) or change the Join between the Entities. This is mentioned in more detail in the course lesson: **Advanced Data Queries**.

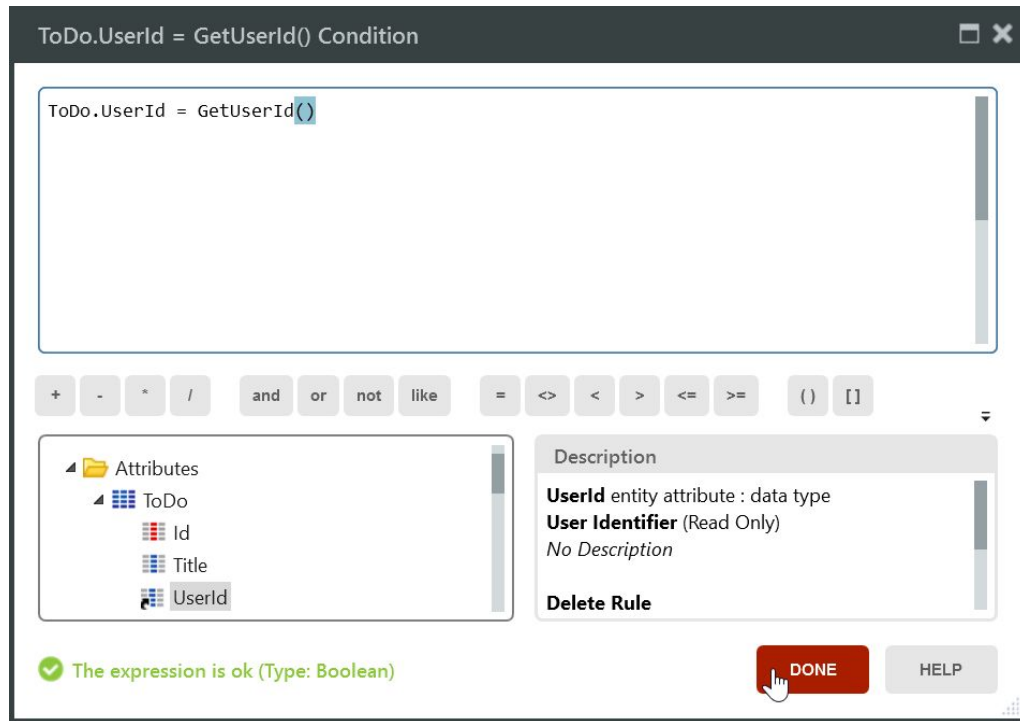
- c) Notice that the Aggregate's **Name** has changed to *GetToDos*.
- d) In the Aggregate, select the **Filters** tab and click the **+Add Filter** option to create a new filter.



- e) Set the Filter Condition to:

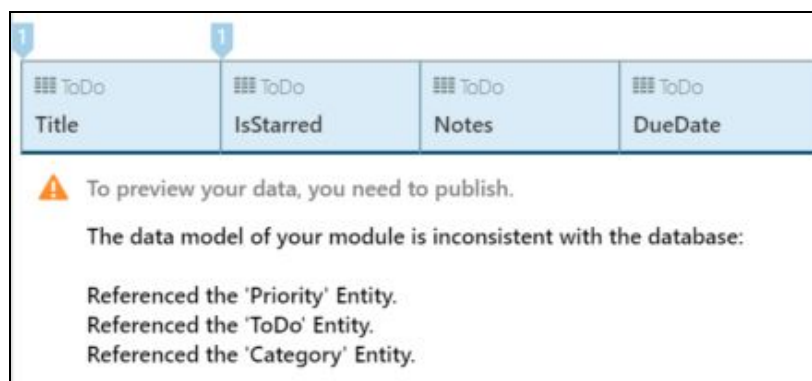
ToDo.UserId = GetUserId()

This will guarantee that the Aggregate will be filtered to only return the ToDos created by the user currently logged in. The *UserId* attribute of the *ToDo* (*ToDo.UserId*) gives us the user associated with the *ToDo*, while the *GetUserId()* function gives us the Id of the user currently logged in. Click on **Done** to close the Expression Editor.



NOTE: The function **GetUserId()** returns the identifier of the user that is currently authenticated with the server, or *NullIdentifier()* if the user is not authenticated.

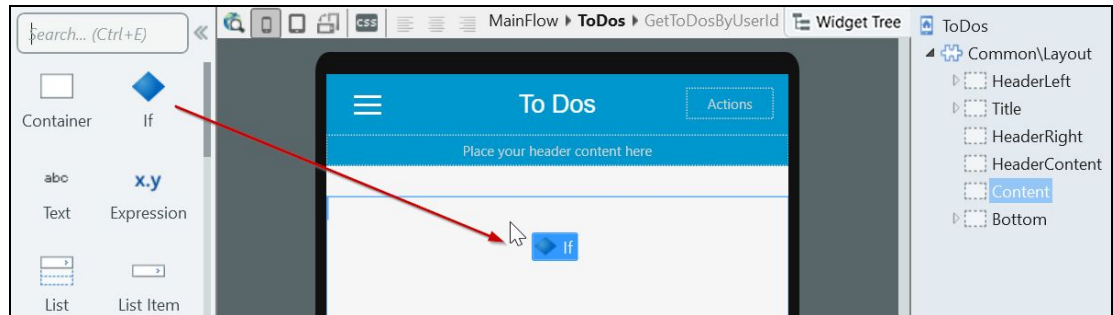
- f) Notice that the Aggregate name has changed to *GetTodosByUserId*. Also notice that no data appears on the Aggregate. That happens because we haven't published the module yet. This does not mean that the exercise is not correct, so we can proceed. If there's the case in the future where the previewer does not display the data, it is always advisable to publish and test in the browser / device.



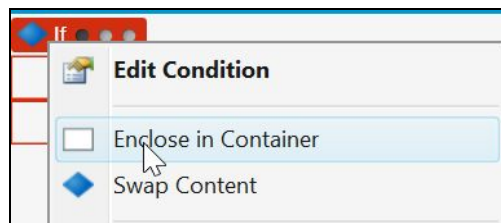
- 4) In this step, we will start adding some UI to display visual cues to the user, when there are no Todos being displayed on the Screen. The first of these scenarios happens where there was an error fetching data. We will use a **Blank Slate** widget and an **If** widget to

help us display an appropriate icon and message to the user, when in fact there was an error fetching data. **Hint:** Use the Aggregate Output Parameter **HasFetchError**. Return to the Todos Screen.

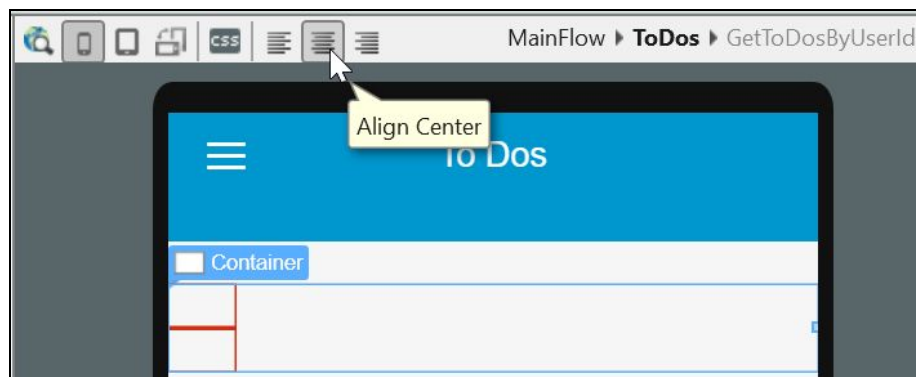
- a) Drag a new If widget and drop it in the main area of the Canvas, in the Content placeholder of the Todos Screen, as shown in the following screenshot



- b) Right-click the **If** Widget and enclose it in a **Container**.



- c) Select the newly created Container and center it on the Screen, using the option **Align Center**.

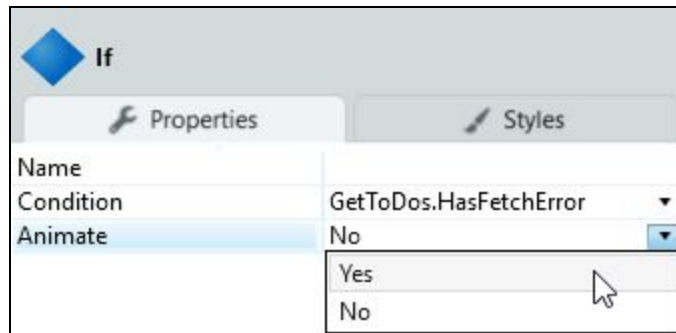


- d) Set the Condition of the If inside the Container to determine if the Screen Aggregate had any error fetching data:

GetTodosByUserId.HasFetchError

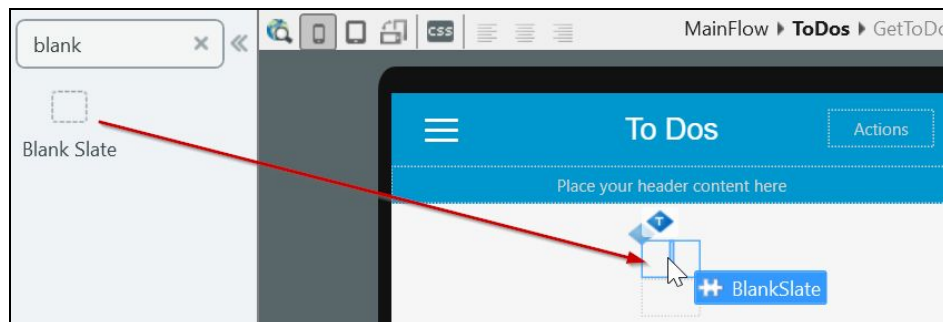
NOTE: The **HasFetchError** is an Output Parameter of the Database Aggregate. This value is True when there is an error during the data fetching, due to a server error or communication timeout.

- e) Set the **Animate** property of the **If** Widget to *Yes*

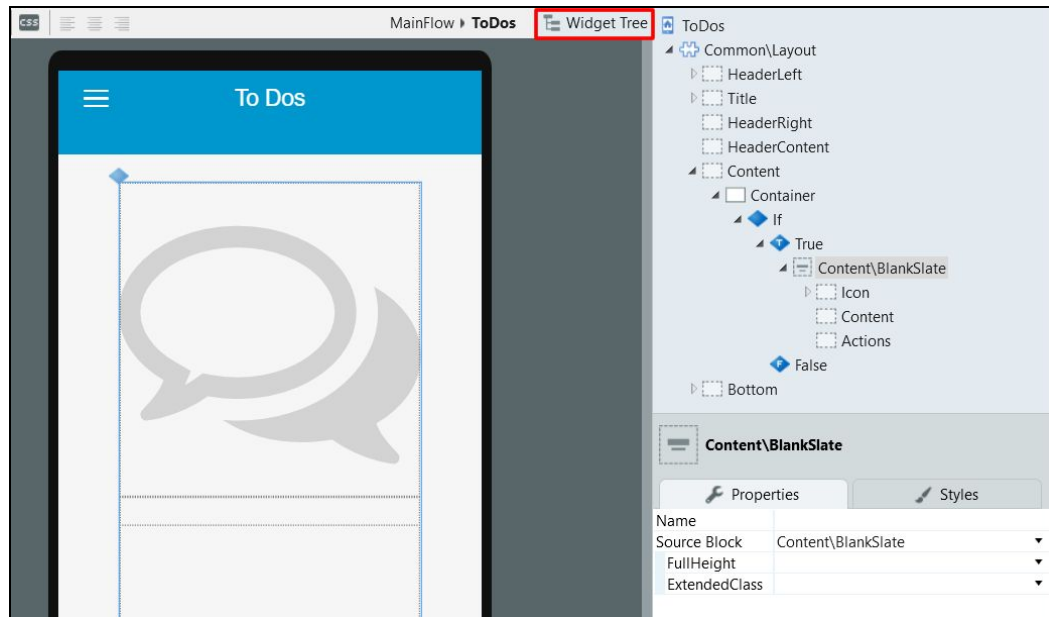


NOTE: The **Animate** property performs an animation on the content when the If condition changes its value.

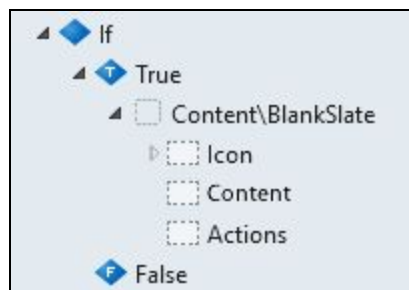
- f) Drag a **Blank Slate** Widget and drop it inside the **True** branch of the **If** Widget.



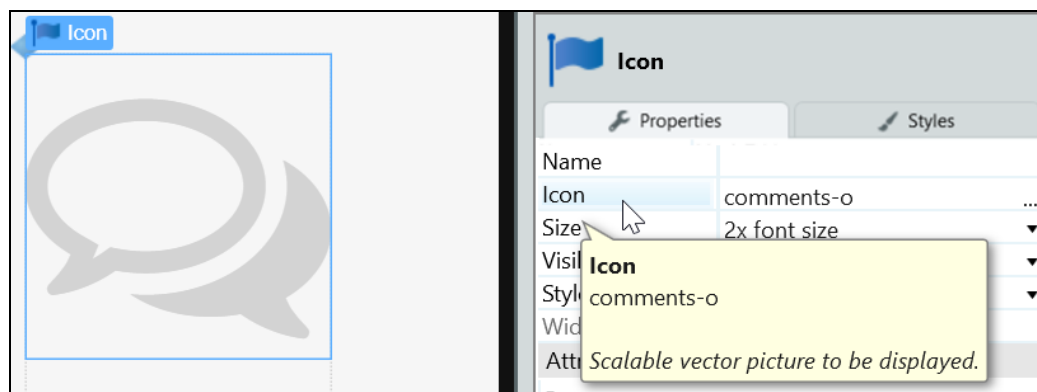
- g) Click on the **Widget Tree Icon** on the top right of the canvas to open the Widget Tree. The Widget Tree shows the structure of the page in detail.



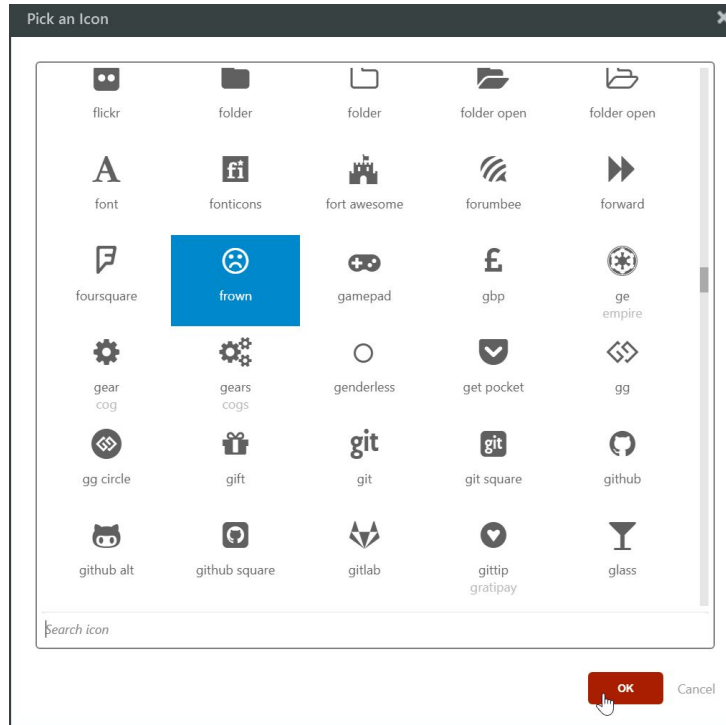
Notice that the BlankSlate has three placeholders: **Icon**, **Content** and **Actions**.



- h) Select the **Icon** Widget in the **Icon** placeholder of the Blank Slate. In the properties editor, double-click the Icon property.



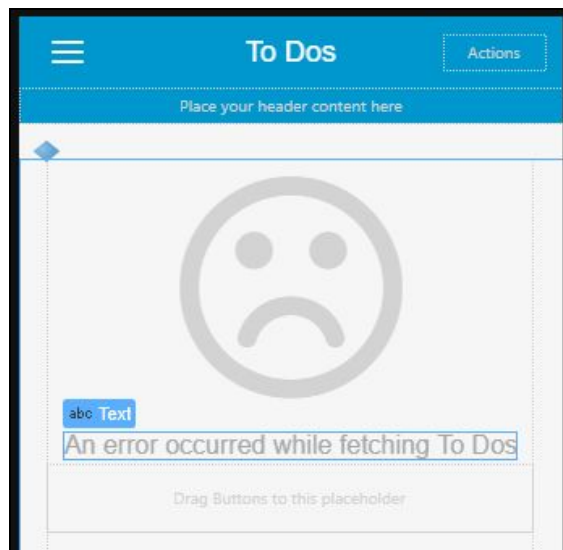
- i) In the new dialog, select the unhappy smiley face (frown) and click **Ok**. This will change the icon in the Blank Slate to the frowning face.



- j) Inside the **Description** placeholder of the Blank Slate type:

An error occurred while fetching To Dos

- k) The Screen should look like this

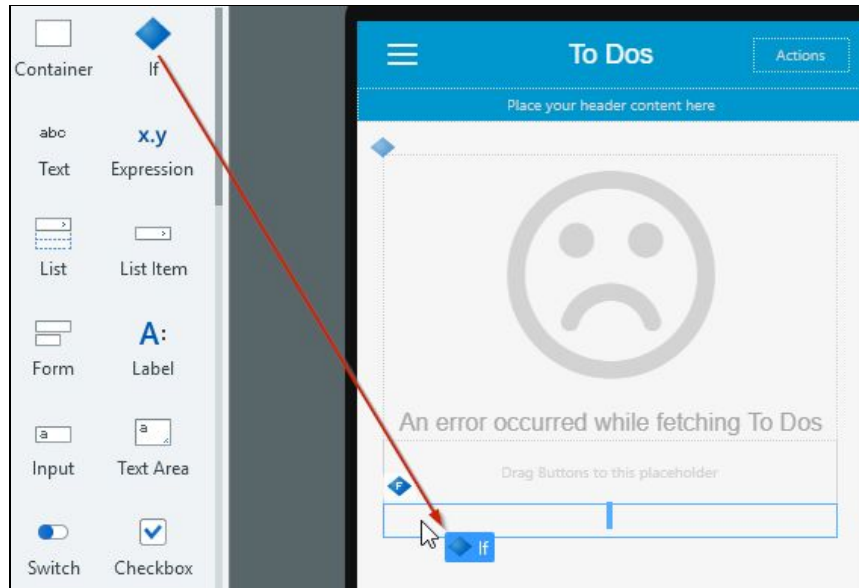


- 5) Now, let's create a second scenario, using the same logic and same patterns, for the case where the data is still being pulled from the database, like a *Loading* situation. This UI

should never appear together with the first one (frowning face), it is one or the other.

Hint: Use the Output Parameter **IsDataFetched** from the Screen Aggregate.

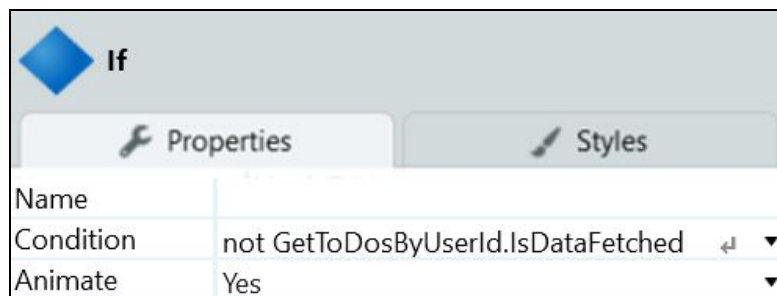
- a) Drag another **If** Widget and drop it inside the **False** branch of the existing If.



- b) In the properties of the new If, set the **Animate** property to **Yes** and the **Condition** to

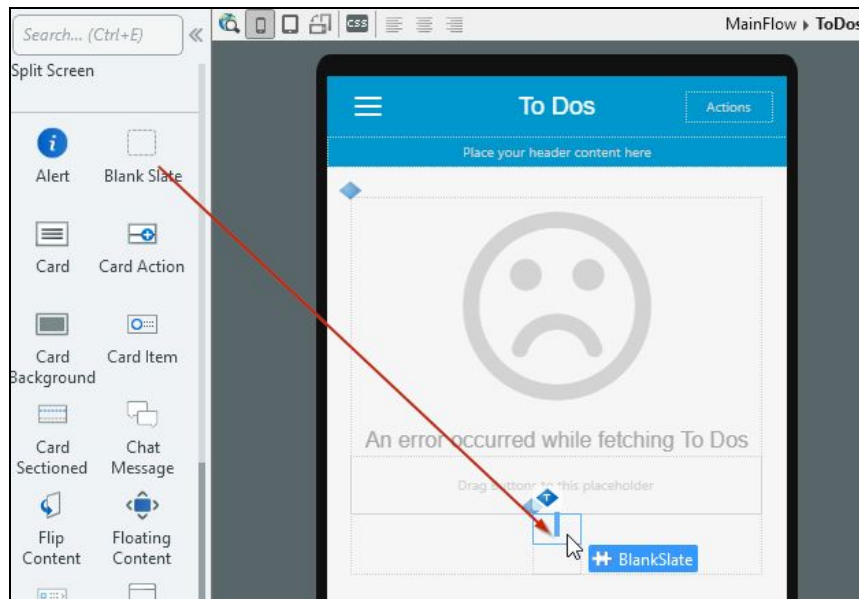
not GetToDosByUserId.IsDataFetched

This verifies if the data has been fetched already or not.

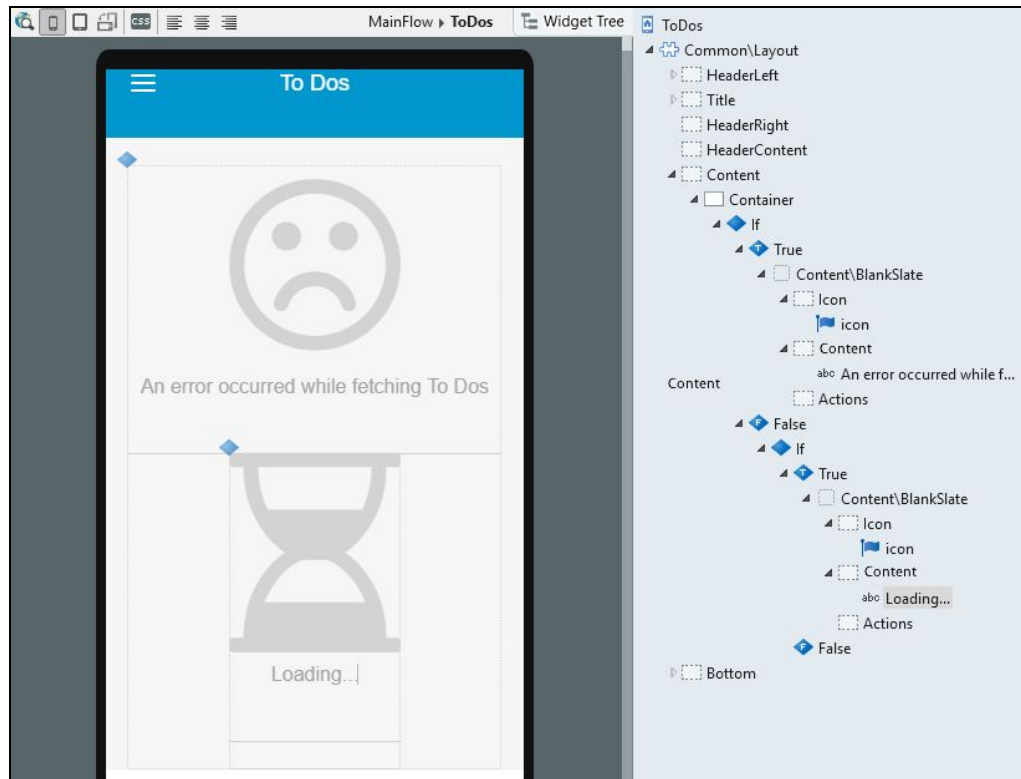


NOTE: **IsDataFetched** is a runtime property of the Aggregate, which is *True* when data has been fetched from the database and it is ready to be used. In this particular case, we want the opposite, and that is why the If condition has the **not**. This way, the True branch will display the UI that is going to appear on the Screen when the data is not yet ready to be used.

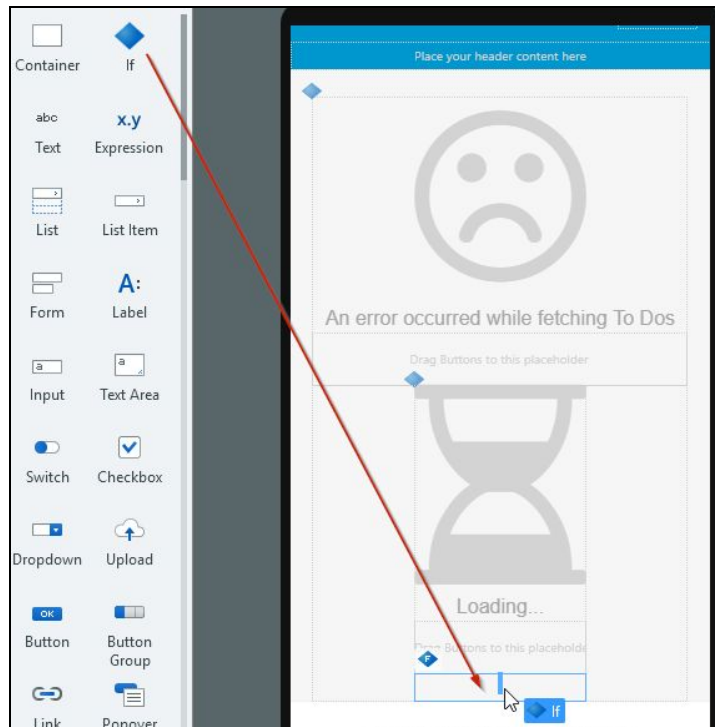
- c) Drag a new **Blank Slate** Widget to the **True** branch of the newly create If.



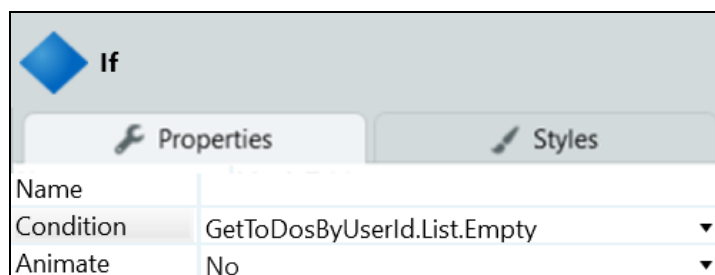
- d) Replace the **Icon** Widget in the Blank Slate by the *hourglass-half*, using the same technique as for the frowning face.
- e) Type *Loading...* in the **Content** placeholder. The resulting Screen and Widget Tree should look like the following screenshot



- 6) In the next scenario, we will create the UI to display information when there are no ToDos to display. This will follow the same strategy used in the previous two cases.
- a) Drag a new **If** Widget and drop it inside the **False** branch of the last **If**.

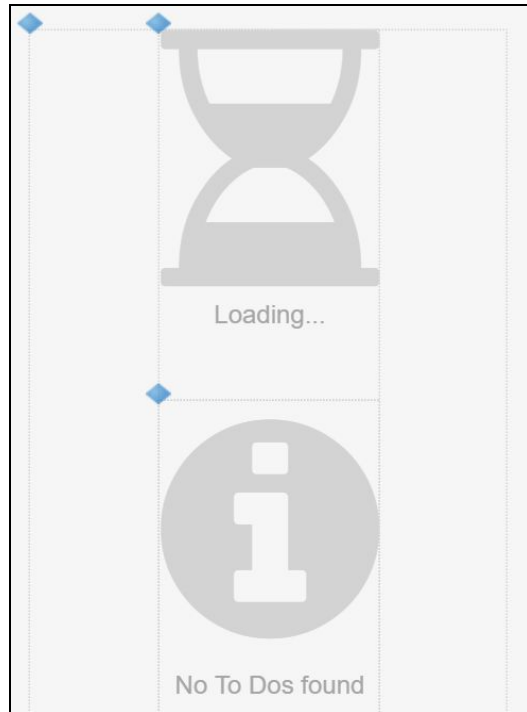


- b) Set the **Animate** property to *Yes* and the **Condition** to *GetToDosByUserId.List.Empty*



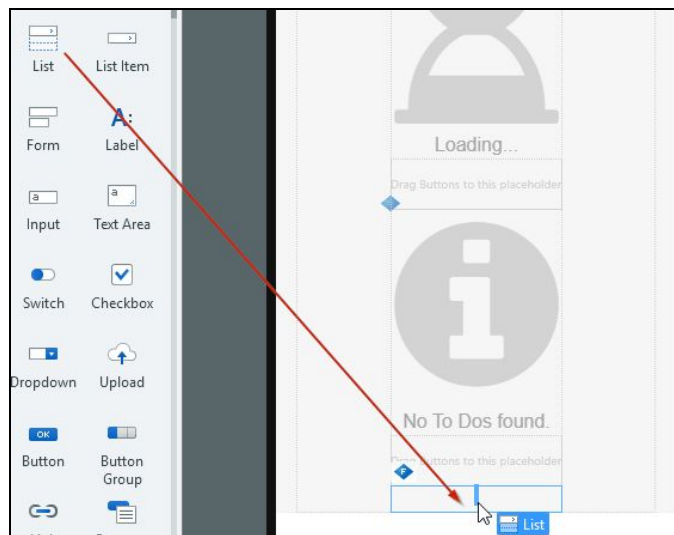
This Condition evaluates if the List of Aggregate is empty or not. If it is, it means that there are no ToDos in the database, associated with the user logged in.

- c) Drag another **Blank Slate** and drop it inside the **True** branch of the If added just before.
- d) Change the existing **Icon** Widget in the new Blank Slate to be the *info circle*.
- e) Type *No To Dos found* in the **Content** placeholder of the Blank Slate.

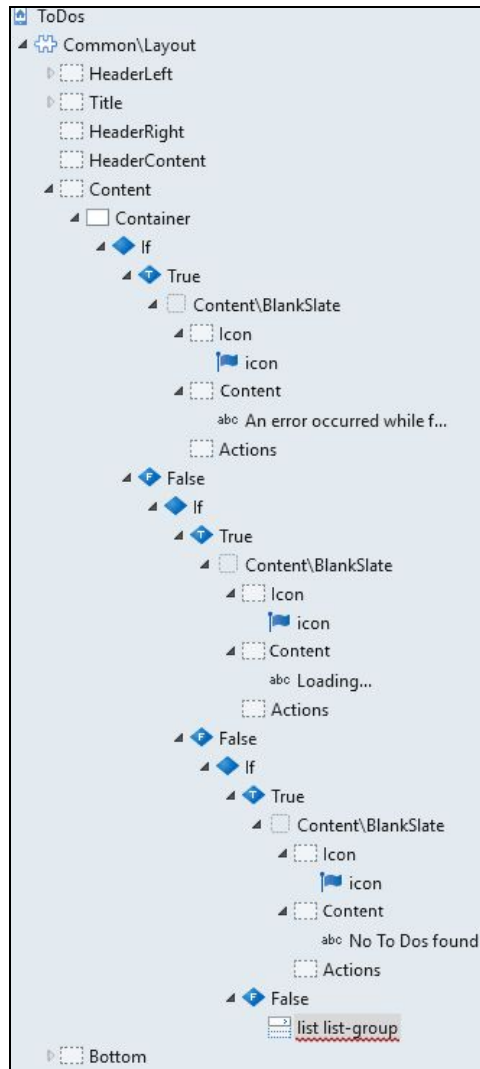


7) Finally, it is time to define the UI to display the list of To Dos returned by the Aggregate, using a List widget. The List should be added to the False of the last If, since it will only be displayed if none of the Conditions in the previous step meet.

a) Drag a **List** Widget and drop it inside the empty **False** branch.

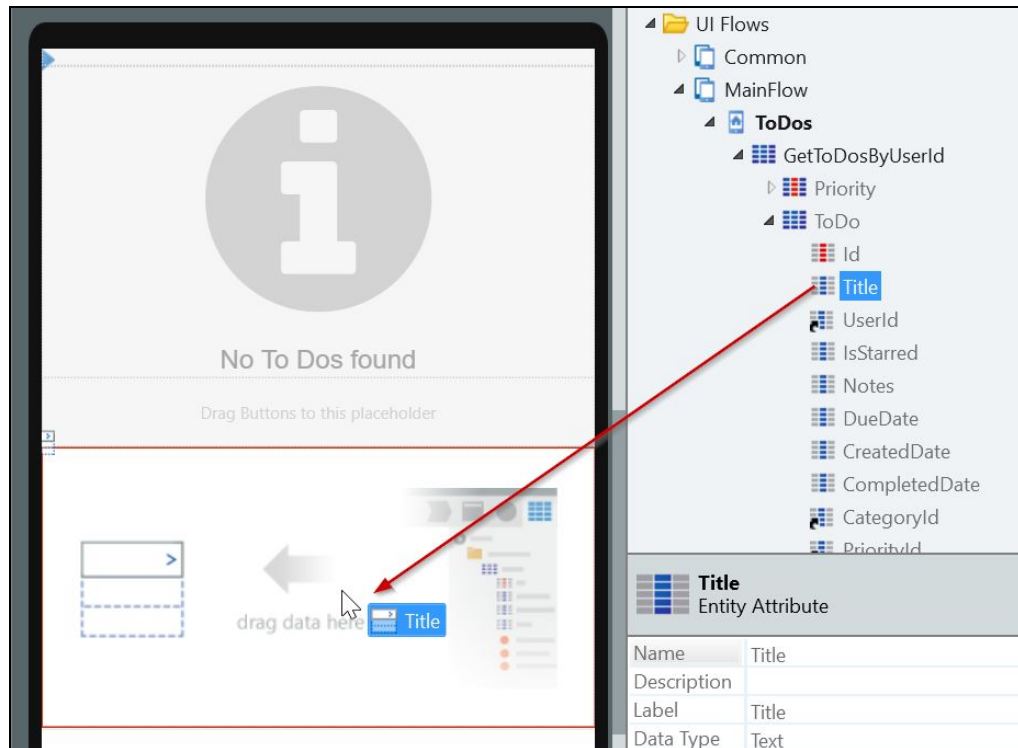


b) The Widget Tree of the **ToDos** Screen should look like this



Notice that the List widget has an error. That is caused by the Source property of the widget, which is still empty. Every List must have a Source, which will indicate what is the source of data to be displayed by the widget.

- c) Switch to the Interface Tab and expand the **GetTodosByUserId** Aggregate and locate the **Title** attribute of the **ToDo** Entity. Drag the **Title** attribute and drop it inside the List.



NOTE: By dragging an attribute from an Entity of the **GetToDosByUserId** Aggregate, the Source property of the List was automatically set to *GetToDosByUserId.List*.

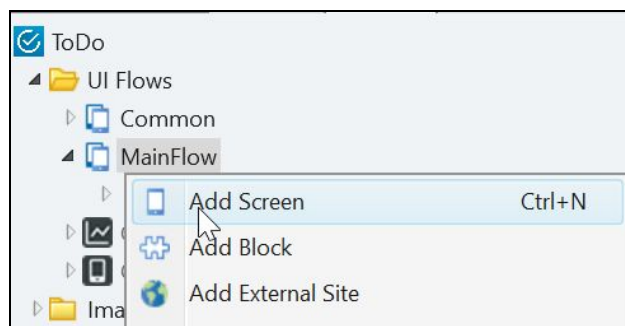
- d) Publish the module, but do not test it yet in the browser, since there are still no ToDos in the database.

Create a Detail Screen

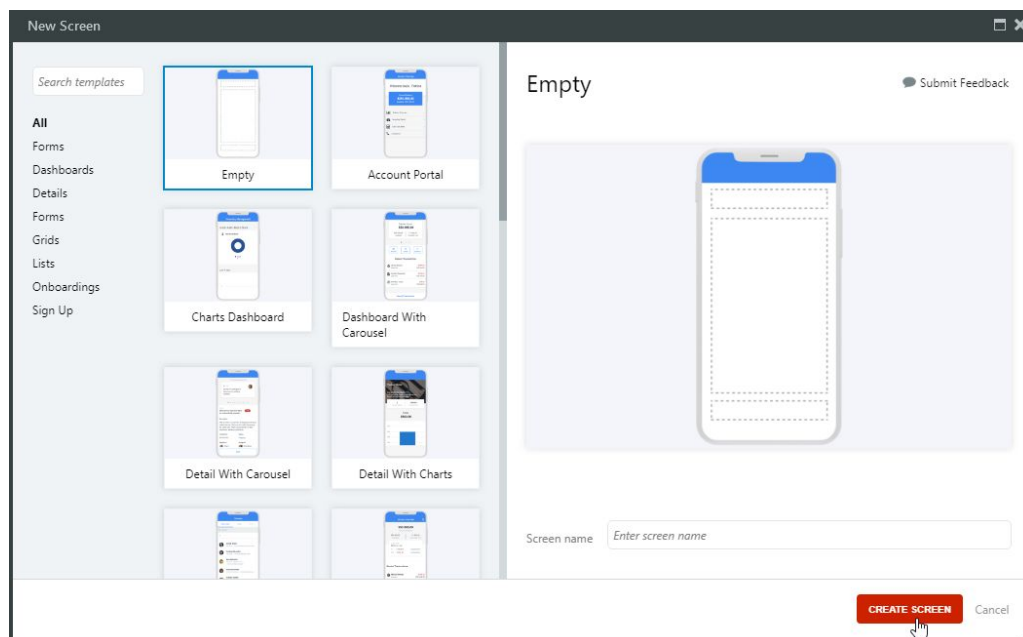
In the previous section, we created the Screen to list the Todos of a certain user. Now, we will create an additional Screen that will have two purposes: create new Todos and edit existing Todos. This new Screen will be called **ToDoDetail**.

This Screen will allow saving a new ToDo, or updating an existing one in the database, thus we will be sending information to the server for the first time.

- 1) Create a new Screen called *ToDoDetail*. Use the Empty template.
- a) Switch to the Interface Tab, right-click the **MainFlow** and choose *Add Screen*.



- b) Choose the **Empty** template Screen from the list.

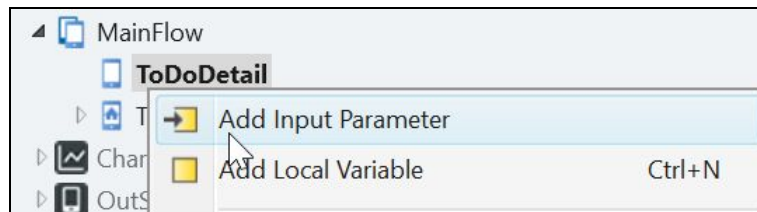


- c) Set the Screen **Name** to *ToDoDetail*.

| ToDoDetail Screen | |
|-------------------|-------------------------------------|
| Name | ToDoDetail |
| Description | ... |
| Title | |
| Roles | |
| Anonymous | <input type="checkbox"/> |
| Registered | <input checked="" type="checkbox"/> |
| Events | |
| On Initialize | ▼ |
| On Ready | ▼ |
| On Render | ▼ |
| On Destroy | ▼ |

2) Add a Screen Aggregate to the Screen that will fetch a ToDo, filtered by its identifier. This identifier will be passed to the Screen as Input Parameter.

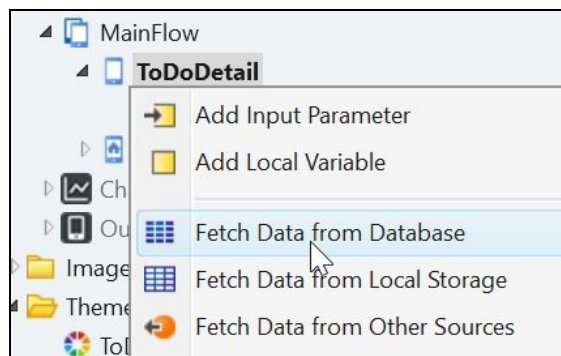
a) Right click the **ToDoDetail** Screen and select *Add Input Parameter*.



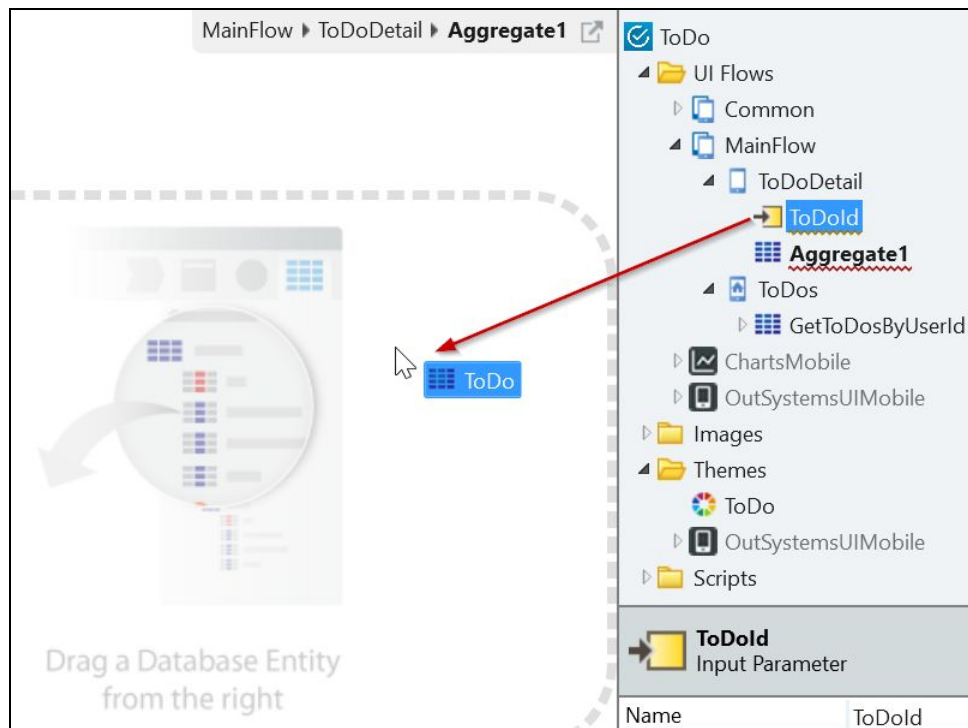
b) Set the new Input Parameter name to *ToDoid* and make sure that the **Data Type** property is set to *ToDo Identifier*.

| ToDoid Input Parameter | |
|------------------------|-------------------|
| Name | ToDoid |
| Description | ... |
| Data Type | ToDo Identifier ▼ |
| Is Mandatory | Yes ▼ |

c) Right-click the **ToDoDetail** Screen and select *Fetch Data from Database*.

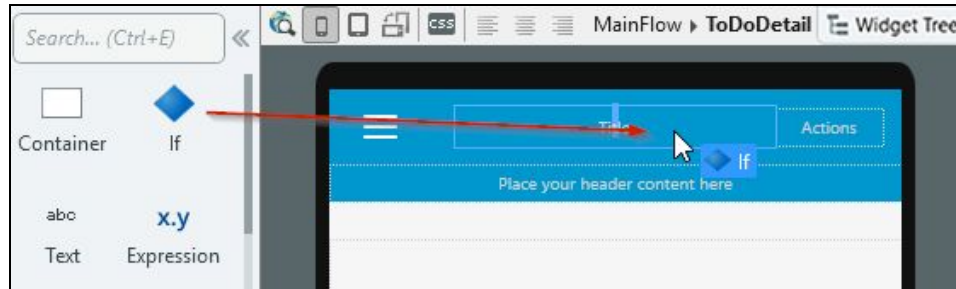


- d) Drag the **ToDold** Input Parameter and drop it inside the **Aggregate1** editor.



NOTE: By dragging the **ToDold** Input Parameter, the **ToDo** Entity was added as a Source Entity to the Aggregate. Also, a new Filter Condition was created to filter the **ToDos** Entity, based on the To Do Identifier. Notice also that the Aggregate has been renamed to *GetToDoById*.

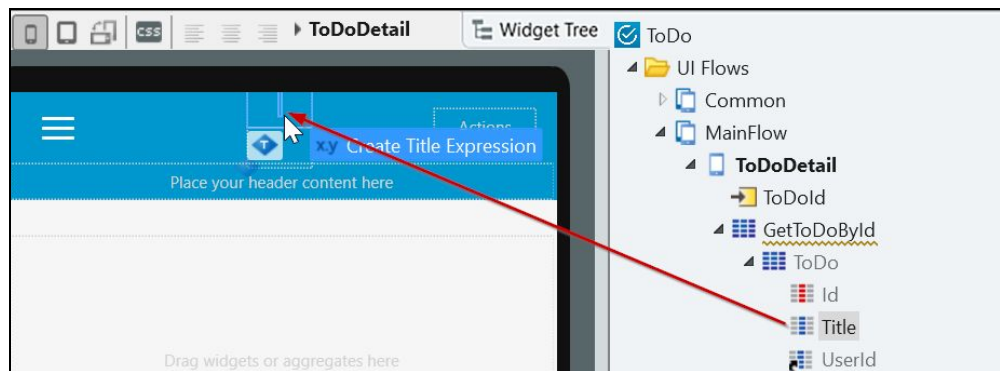
- 3) Define the Title of the Screen to depend on the value of the Input Parameter passed to the Screen. If the *ToDold* Parameter is *NullIdentifier()*, the Title of the Screen should display *Create New To Do*. If it has a value, then it should display the value of the Title attribute of the *ToDo*.
 - a) Double-click the **ToDoDetail** Screen to open it.
 - b) Drag an **If** Widget and drop it inside the **Title** placeholder of the **ToDoDetail** Screen.



- c) Set the **Condition** of the If to

ToDoid <> NullIdentifier()

- d) Expand the **GetToDoById** Aggregate, drag the **Title** attribute of the ToDo Entity and drop it on the **True** branch of the If Widget.

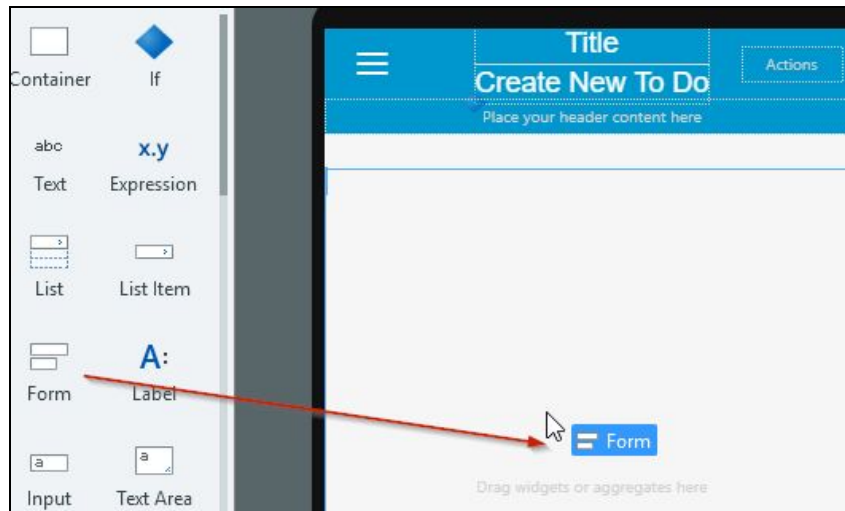


- e) Inside the **False** branch, type *Create New To Do*

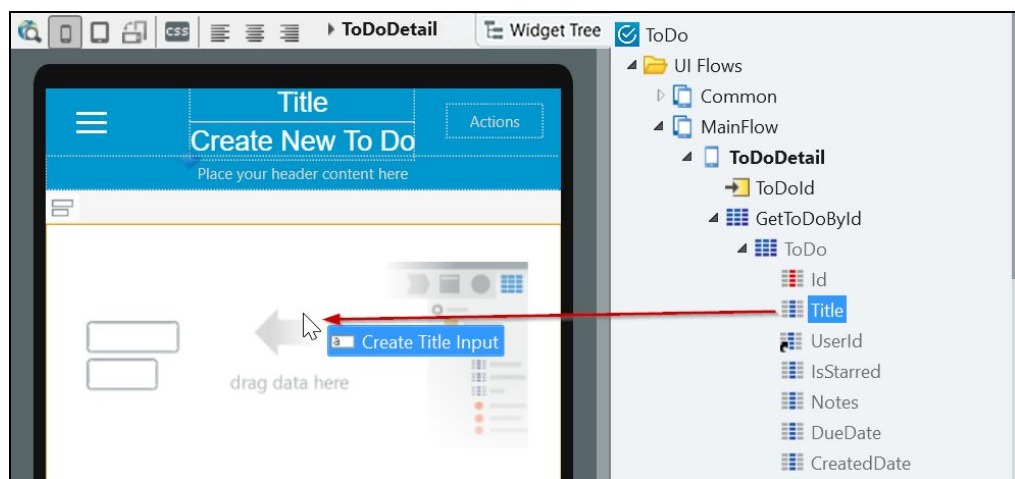


- 4) Create a **Form** to allow the **ToDoDetail** Screen to accept input from end-users, to fill information for the **ToDo** attributes. The users should be able to submit the Title, Notes, Due Date, Category (in a Dropdown) and Priority (in a Button Group), and then click on the Save Button to save the data on the database.

- a) Drag a **Form** Widget and drop it inside the Screen's **Content** placeholder.

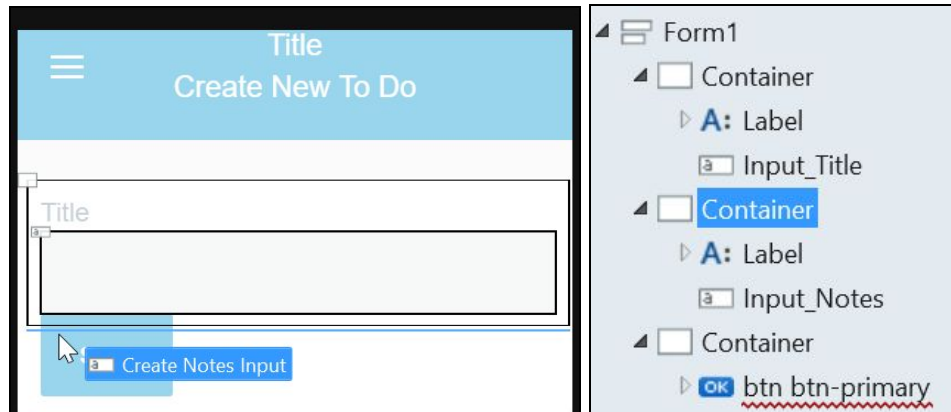


- b) Locate the **Title** attribute of the **ToDo** Entity, from the **GetToDoById** Aggregate, then drag it and drop it inside the **Form** Widget.



Notice that the Save Button was automatically created, since now the Form has at least one Input widget.

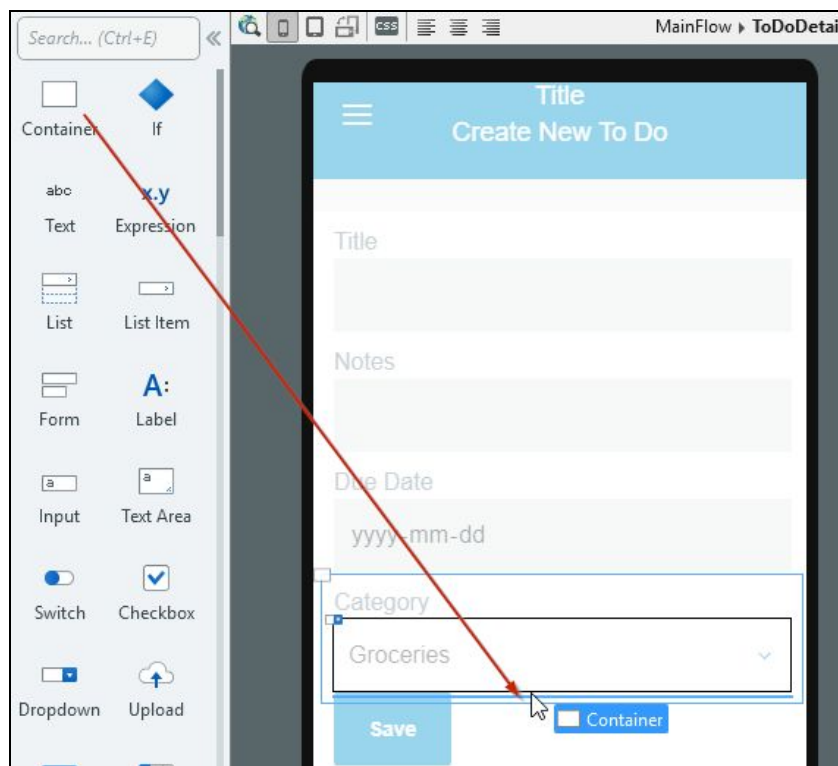
- c) Drag the **Notes** attribute and drop it between the **Title** input and the **Save** Button.



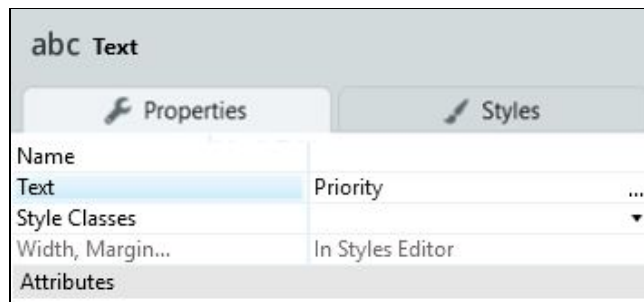
- d) Repeat the same steps for the **DueDate** and **CategoryId** attributes. Make sure they both appear after the Notes input, but before the Save Button.

NOTE: Dragging the **CategoryId** attribute created a **Dropdown**, with all the Categories options. This happened because you dragged the foreign key attribute of the **ToDo** Entity. Also, a new Aggregate, **GetCategories**, was added to the Screen to populate the Dropdown.

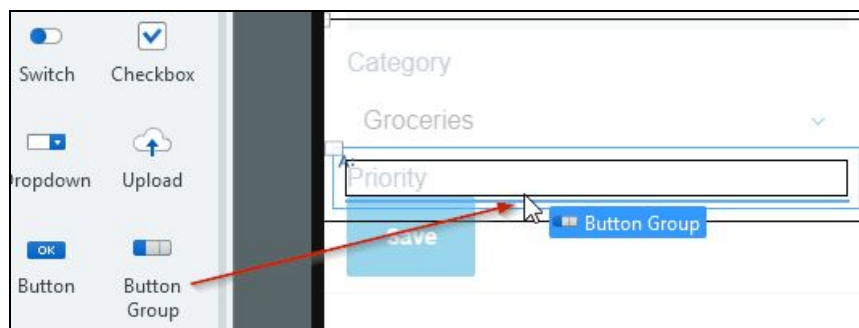
- e) Drag a **Container** Widget and drop it between the Categories Dropdown and the Save Button.



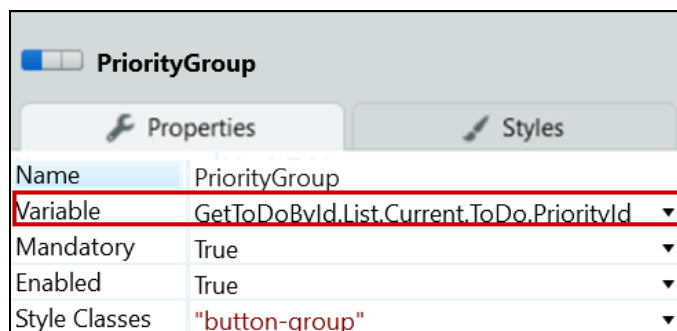
- f) Drag a **Label** Widget and drop it inside the Container created in the previous step. Set the **Text** of the Label to *Priority*.



- g) Drag a **Button Group** Widget and drop it below the **Label**, but still inside the surrounding **Container**.

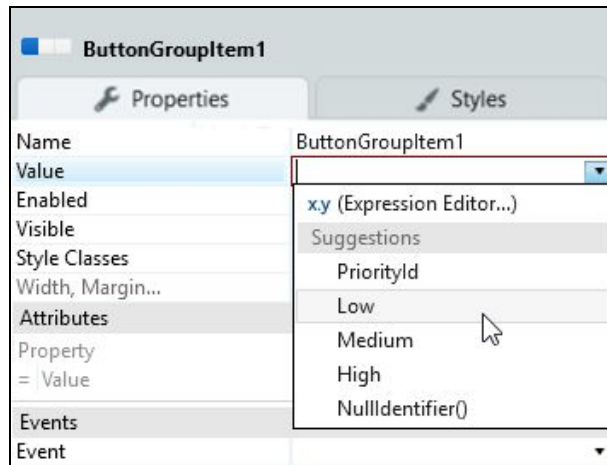


- h) Set the **Name** property of the **Button Group** to *PriorityGroup*.
i) Set the **Variable** property to *GetToDoById.List.Current.ToDo.PriorityId*



This will bind the ButtonGroup to the **PriorityId** attribute of the ToDo being displayed (or being created) in this Screen. This means that the value chosen for the Priority of the ToDo will be saved in this **Variable**.

- j) Change the text of the first **ButtonGroupItem** in the group to *Low*, the text of the second ButtonGroupItem to *Medium*, and the last one to *High*.
k) Select the first ButtonGroupItem of the **PriorityGroup** and set the **Value** property to *Low*.



- l) Set the **Values** for the two remaining ButtonGroupItems in the group to *Medium* and *High* respectively.

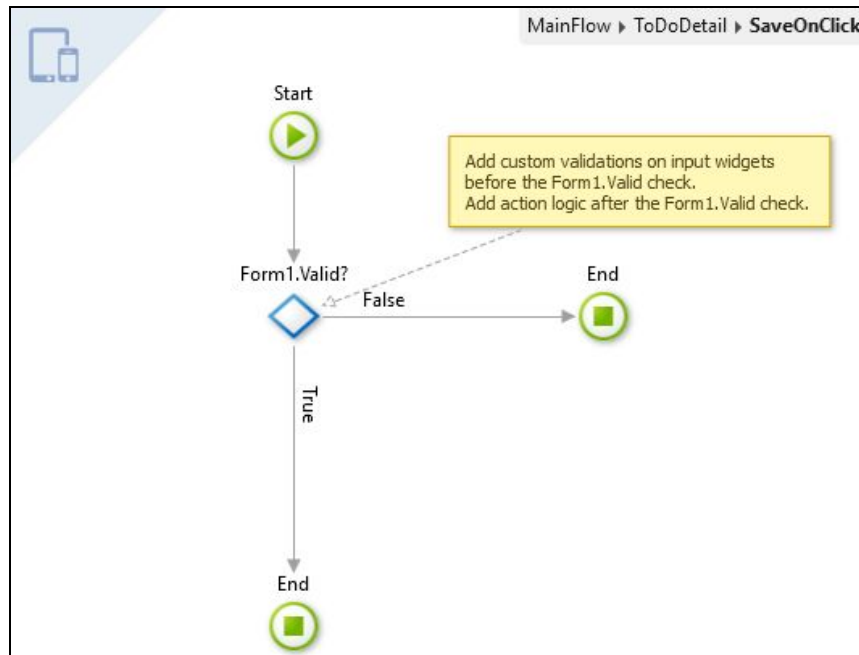
This is binding each ButtonGroupItem to a value of the **Priority** Static Entity. This way, when a user chooses the first ButtonGroupItem, the **Variable** of the **ButtonGroup** will have the value *Low*.

- m) Set the **Mandatory** property of the Priority Button Group to the following:

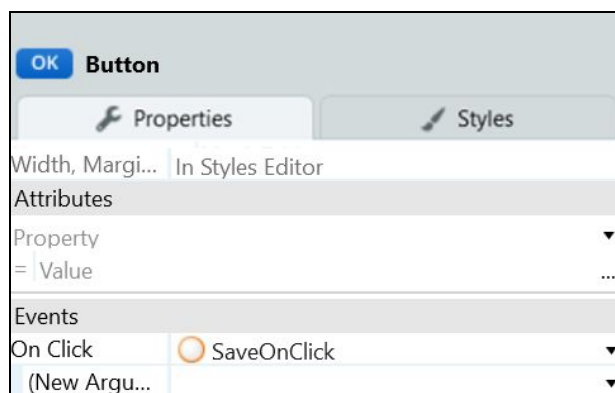
If(GetToDoById.List.Current.LocalToDo.PriorityId = NullIdentifier(), True, False)

This code is a work around that will enable the user to update the ToDo item again without re-selecting the priority button.

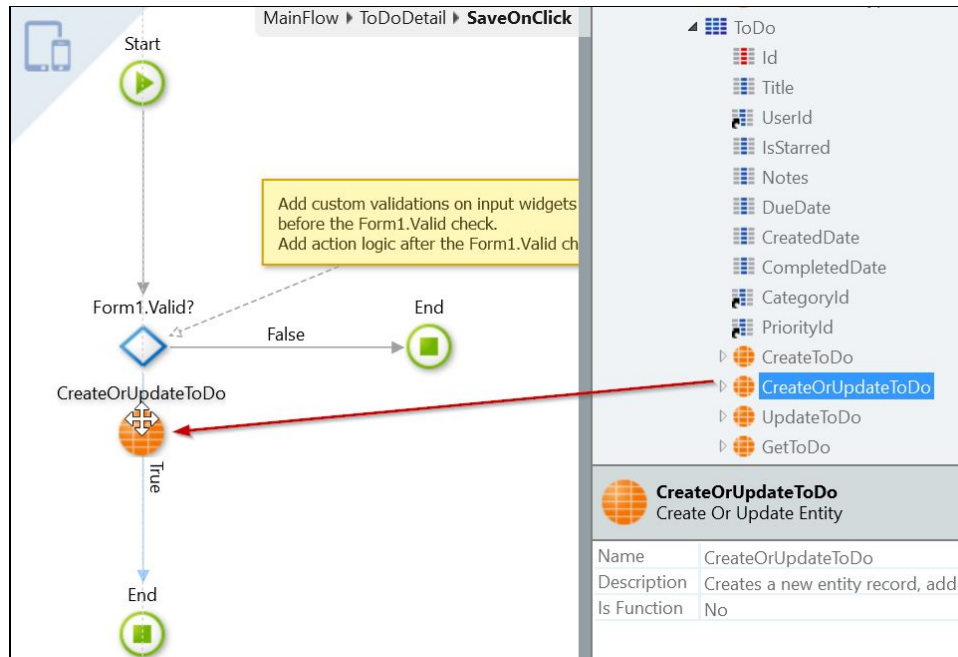
- 5) Create the Logic to create or update new To Dos in the database, triggered when the Save button is clicked. At the end, we should automatically return to the ToDos Screen.
 - a) Double-click the **Save** Button to create the **SaveOnClick** Client Action.



This will create an Action that will run when the end-user clicks on the Save Button. That happens because the OnClick Event of the Button will have the SaveOnClick Action associated to it.



- b) Switch to the Data Tab and Expand the **ToDo** Entity. Drag and drop the **CreateOrUpdateToDo** Action to the flow, on the **True** branch of the **If**.

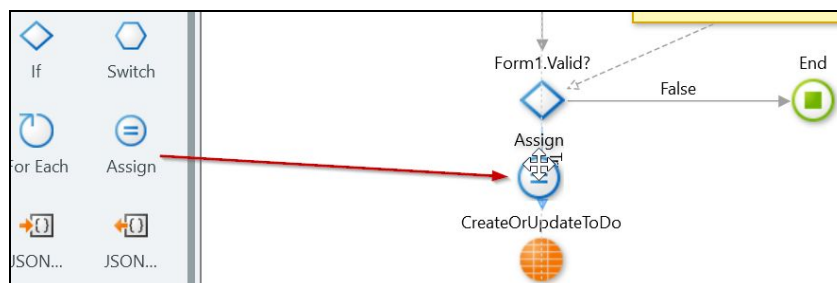


- c) Set the **Source** property of the CreateOrUpdateToDo Action to *GetToDoById.List.Current*



This will use the record that is bound to the inputs of the ToDoDetail Screen, and pass it to the Entity Action that creates the ToDo in the database. However, there is something missing! A mandatory attribute of the ToDo Entity has no value yet: the UserId. So, we need to assign it one, before we use the CreateOrUpdateToDo Action.

- d) Drag and drop an **Assign** statement, right before the CreateOrUpdateToDo.



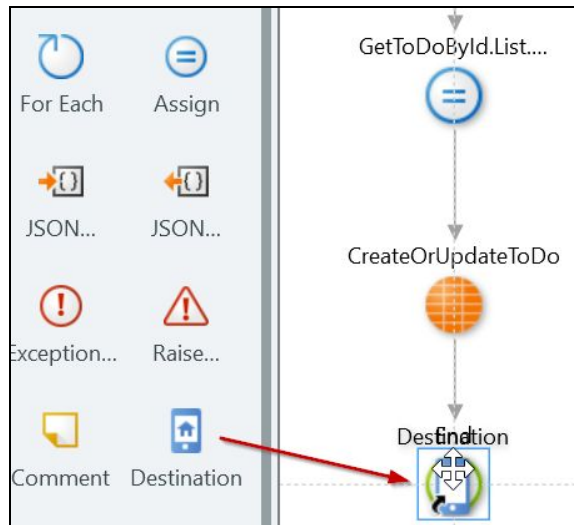
- e) Add the following assignment to the recently added Assign:

GetToDoById.List.Current.ToDo.UserId = GetUserId()

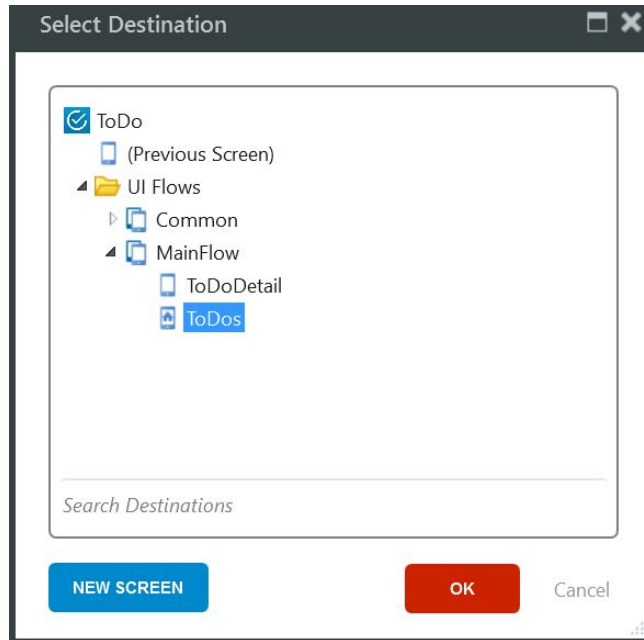
| | |
|--------------------------------------|---|
| | GetToDoById.List.Current.ToDo.UserId Assign |
| Label | |
| Assignments | |
| GetToDoById.List.Current.ToDo.UserId | ▼ |
| = GetUserId() | ▼ |

This will make sure that the Id of the logged in user is added to the record.

- f) Once the information is saved in the database, we want to navigate back to the ToDos Screen. Drag and drop a Destination statement over the End node, at the bottom of the flow, replacing one by the other.



- g) Set the **Destination** to *MainFlow/ToDos*, to navigate to the ToDos Screen.



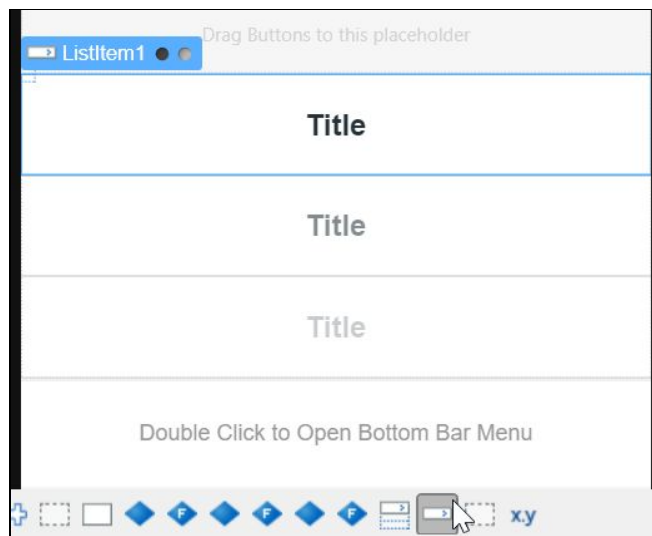
- h) Publish the module to the server, but again do not test it yet! The way the application is, it is not possible yet to access the ToDoDetail Screen.

Link the Screens

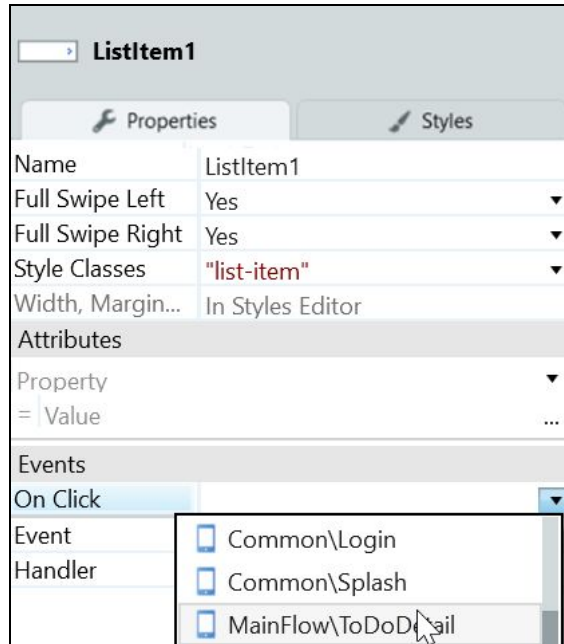
To enable navigating between the Screens, we will now create Links between both Screens, in particular give access to the **ToDoDetail** Screen from the **ToDo** Screen.

First, we will add a Link to every **ToDo** in the List, that will navigate to the **ToDoDetail** Screen, passing the Id of the **ToDo** clicked. Then, we will add an icon to the top of the Screen, to allow creating new **ToDo**s, linking it to the **ToDoDetail** Screen as well.

- 1) Add a Link from each **To Do** in the **ToDo** Screen to the **ToDoDetail** Screen.
 - a) Open the **ToDo** Screen and select the **ListItem1** Widget that contains the **Title** Expression.



- b) In the properties area, set the **On Click** Event to *MainFlow\ToDoDetail*.



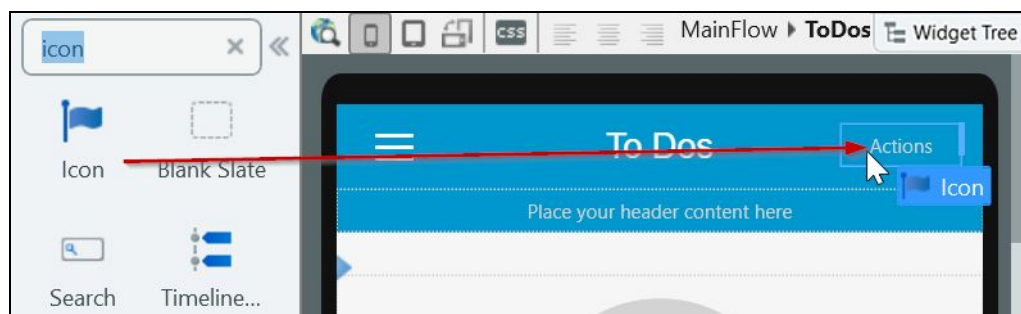
This will cause an error, since the value for the Input Parameter of the Screen is missing.

- c) Open the dropdown of suggestions for the **ToDoid** parameter and select:

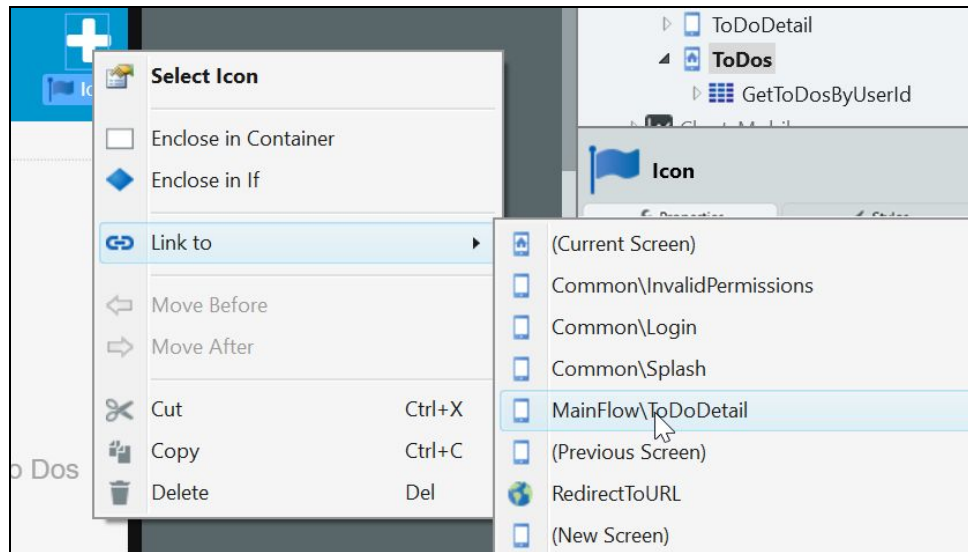
GetTodosByUserId.List.Current.ToDo.Id

This will get the Id of the ToDo that was clicked on.

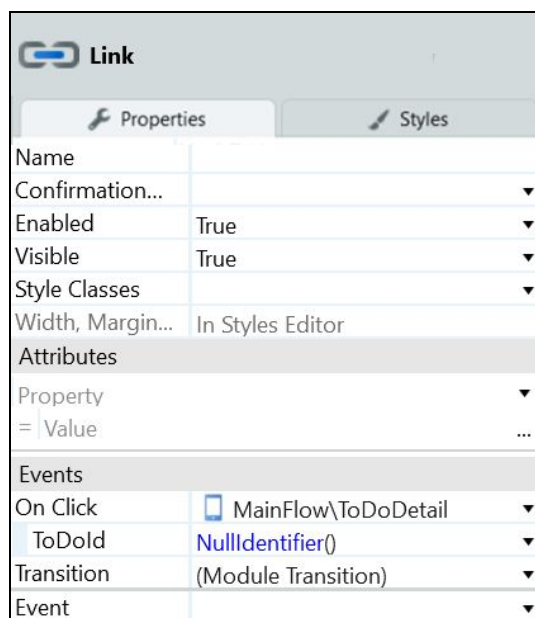
- 2) Add a Link from the **ToDos** Screen to the **ToDoDetail** Screen to create a new ToDo in the database. Use the *plus* icon and place it on the **Actions** placeholder of the ToDos Screen. Don't forget to pass the appropriate value as Input Parameter of the ToDoDetail Screen.
 - a) Drag an **Icon** Widget to the **Actions** placeholder on the top right corner of the Screen. Select the *plus* sign and click **Ok**.



- b) Right-click the newly created icon and select *Link to* and then *MainFlow\ToDoDetail*



- c) In the properties of the Link, set the **ToDoId** parameter to *NullIdentifier()*

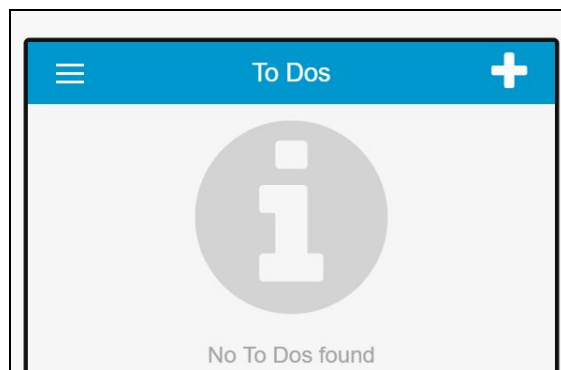


In this case, we pass *NullIdentifier()* as the value of the Input because we want to create a new ToDo. When creating a new ToDo, the ToDo **does not have an Id yet**. By passing *NullIdentifier()*, the Screen Aggregate will try to find it in the database and will always fail, since **there are no ToDos in the database with no Id** (mandatory and auto-number field). So, since no ToDo is found, the Form input fields will be empty, which is what we want. The user can then fill the inputs and click on Save to send the data to the server.

Publish and Test

Now that the Screens are built, let's publish the module and test to make sure everything works properly.

- 1) Publish the module and make sure the 1-Click Publish process was successful.
- 2) Open the application in the browser and create a new ToDo. Make sure that the new ToDo appears listed in the ToDos Screen.
 - a) Click the **Open in Browser** button to open the application.
 - b) Make sure that you see the UI indicating No To Dos found.



- c) Click the Plus sign on the top right corner to create a new ToDo.

A screenshot of a mobile application screen titled 'Create New To Do'. The screen has a blue header bar with a hamburger menu icon on the left, the title 'Create New To Do' in the center, and a plus sign icon on the right. Below the header, there is a form with the following fields: 'Title*' (a text input field), 'Notes' (a text area), 'Due Date' (a date picker showing 'yyyy-mm-dd'), 'Category' (a dropdown menu with 'Family' selected), and 'Priority' (a radio button group with 'Low', 'Medium', and 'High' options). At the bottom of the form, there is a blue 'Save' button.

- d) Fill the Form with the data you want and click Save. You should be redirected back to the ToDos Screen.

☰ Create New To Do

Title*

Book Hotel

Notes

Family Vacations

Due Date

12/11/2018

Category

Family

Priority

Low Medium High

Save

- e) In the ToDos Screen you should now see the newly created To Do.
- f) Clicking the To Do title in the ToDos Screen, should open the details of the To Do, where you can edit it (ToDoDetail Screen).

End of Lab

In this exercise, we created the **ToDos** Screen to display a list of To Dos of a given user. Then, we created the **ToDoDetail** Screen to make it possible for users to create new To Dos and edit existing ones.

For the first Screen, we have learned how to display a list of records fetched from a Database Entity, using an Aggregate. On the second Screen, we used a Form to group Inputs and a Button linked to a Screen Action to add the ToDo record to the database.

Finally, the ToDo module was published to the server and we were able to test the changes made in the device emulator.