Blocks Exercise

Table of Contents

Introduction

In this exercise lab, we will create a Block to represent the information about a ToDo, with the Title, Due Date, Priority, and the information about if it is starred or completed.

A Block is a way to centralize and reuse UI throughout the application, and can be created just like a Screen. Then, it can be used as any widget or pattern, by dragging and dropping them to any Screen or other Block.

After creating the UI for the Block, we will define a clickable functionality to star a ToDo, and a swipeable action to set a ToDo as complete. With this functionality, we will also avoid that any completed ToDo can be modified in the server, or in the local storage.

With the Block ready, we will use it in the ToDos Screen, in the list that displays all the local to dos. Finally, we will sort that list by importance, meaning that the starred ToDos appear first, and by Due Date, starting from the closest due date to the one later in time. This will require some modifications on the Block.

As a summary, in this specific exercise lab, we will:

- Create a new Block to represent a To Do
- Define clickable and swipeable actions within the Block
- Instantiate the Block
- Prevent users from modifying completed To Dos
- Sort the ToDos by importance and due date

Create a new Block for representing To Dos

We will start this exercise by creating a Block in the application, *ToDoItem*. This Block will be created in the MainFlow and it will display the information about the ToDo. To allow that, and since a Block does not have the context of its parent, the ToDoItem will have two Input Parameters, to pass the Block information about the **LocalToDo** and **LocalPriority**.

Then, using a List Item Content and some Icon and Tag patterns, the ToDoItem Block will display the **Title**, **Due Date**, the **Priority** and if it is **starred** and **complete**.

1) Create a new Block called *ToDoItem* in the MainFlow. Add two Input Parameters to the Block: *LocalToDo* and *LocalPriority*.

   a) Under the **Interface** tab, right click the **MainFlow** and select *Add Block*.

   b) Change the **Name** of the Block to *ToDoItem*.

   c) Right-click the **ToDoItem** Block and add a *LocalToDo* Input Parameter. Verify that its **Data Type** is set to *LocalToDo*.

   d) Add a second Input Parameter named *LocalPriority*, with **Type** set to *LocalPriority*.
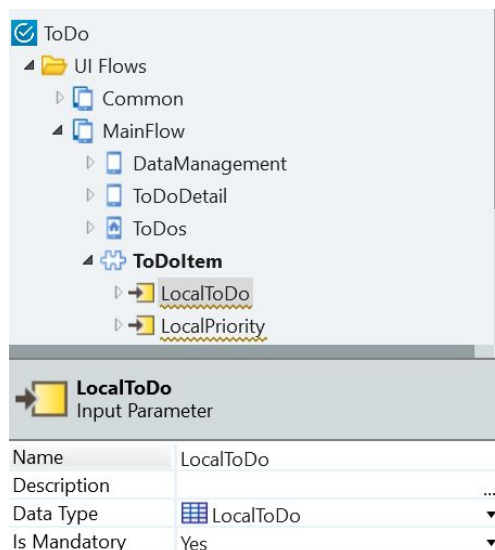

   > **NOTE:** Since this Block will be reused inside the list Screen, we need to send as Input Parameters the information that will be needed to render each instance of the Block. It would be possible to only pass the respective identifiers, but in that case we would need another Aggregate to fetch the information of each ToDo.

This would lead to an increased number of queries and could impact the application's performance.

2) We will start creating the **ToDoItem** Block content by defining how the information will be structured within the Block. The ToDo information will be displayed using a **List Item Content** inside a List Item widget. Create the Title content of the ToDoItem Block in the **Title** placeholder of the List Item Content widget. This content will be represented by the ToDo Title and a green check icon that will be displayed only if the ToDo is completed.
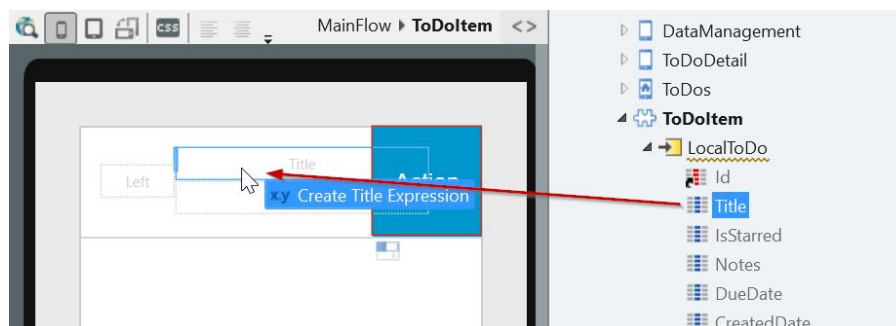
a) Open the ToDoItem, drag a **List Item** Widget from the toolbar and drop it inside the Block.
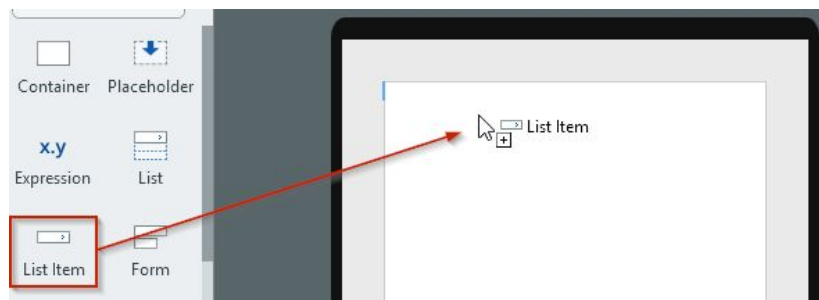
b) Drag a **List Item Content** Widget and drop it inside the **Content** placeholder of the **List Item** created in the previous step.
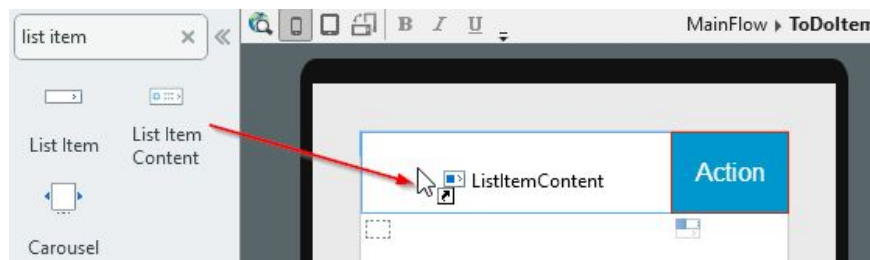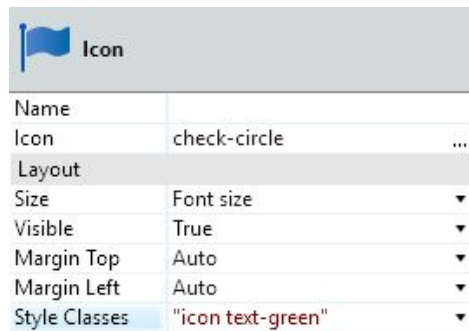

c) Expand the **LocalToDo** Input Parameter of the Block, then drag the **Title** attribute and drop it inside the **Title** placeholder.
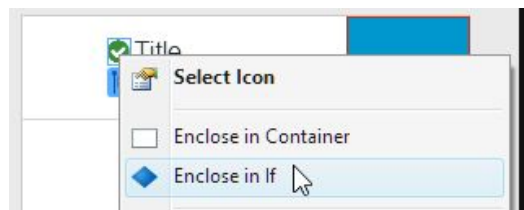

d) Drag an **Icon** Widget and drop it just before the **Title** Expression.
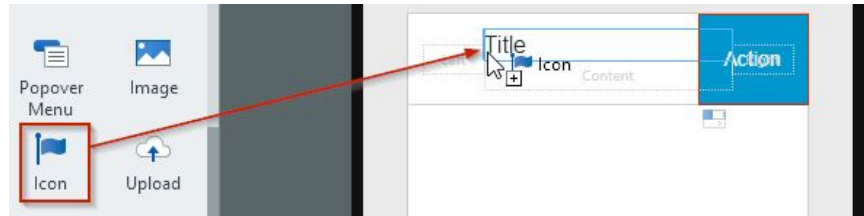
e) In the **Pick an Icon** dialog, select the *check circle* icon.

f) Change the **Size** property to *Font size* and add the *"text-green"* style to the **Style Classes** property.

g) Right-click the **Icon** Widget and choose *Enclose in If*.

h) Set the **Condition** property of the If to

   LocalToDo.CompletedDate <> NullDate()

   This will ensure that the icon will only appear if the Completed Date of the ToDo is different than the Null Date, meaning that the ToDo is already completed.

3) Define the **Content** of the List Item Content widget to display the ToDo's **Due Date**. Make sure it appears only if the Due Date exists.

   a) Expand the LocalToDo Input Parameter, drag the **DueDate** attribute and drop it in the **Content** placeholder.

   b) Right-click the **DueDate** Expression and choose *Enclose in If*.

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

   LocalToDo.DueDate <> NullDate()

This will ensure that the Due Date will only appear if the ToDo has a Due Date.

d) Add the text *Due* (with an empty space after) just before the **Due Date** Expression, but still inside the If.

4) Define the **Left** placeholder of the List Item Content widget to display if the ToDo is starred or not. Additionally, we should enable the user to be able to star a ToDo.

a) Drag an **Icon** Widget and drop it in the **Left** placeholder of the List Item Content widget. Then, choose the filled *star* icon.

b) Add the *"gold-color"* style to the **Style Classes** property.

c) Right-click the **star** Icon and choose *Enclose in If*.

d) Set the **Condition** property of the If to

> *LocalToDo.IsStarred*

e) Drag another **Icon** Widget and drop it in the **False** branch. Then, choose the hollow *star* icon. Add the *"gold-color"* style to the Icon **Style Classes** property.


5) Define the **Right** placeholder of the List Item Content to display the Priority of the ToDo. This representation will use colors (Red, Orange and Green), together with the corresponding Label (High, Medium and Low). **Hint:** Use a Tag Pattern to help representing the Priority.

a) Select the **List Item** Widget then hide its placeholders by clicking the *Collapse* circle.
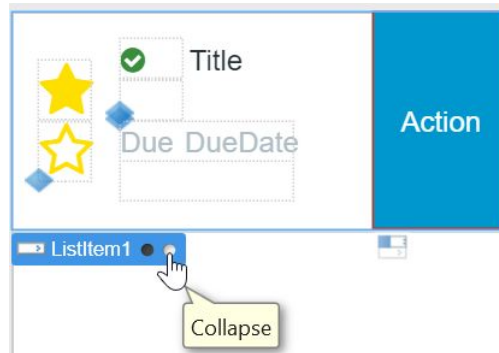
b) From the Widget toolbar, drag and drop the **Tag** Widget and drop it in the **Right** placeholder. If there is a Widget already in the placeholder, delete it first.
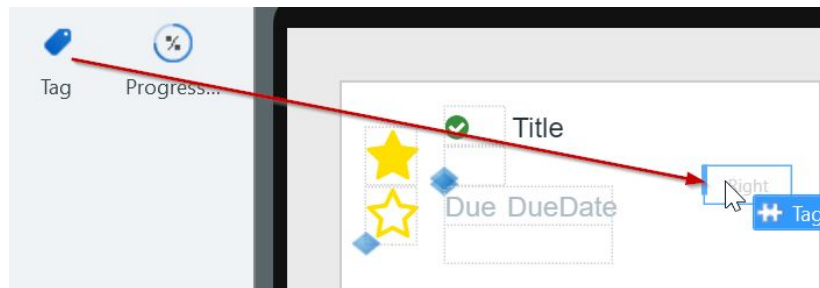

c) Double-click the **BackgroundColor** property to open the Expression Editor, then build the following expression inside it

> *If(LocalToDo.PriorityId = Entities.Priority.High, Entities.Color.Red,*
> *If(LocalToDo.PriorityId = Entities.Priority.Medium, Entities.Color.Orange,*
> *Entities.Color.Green))*

This property defines the color of the Tag element. With this condition, we assign the red color when the ToDo has a High Priority, orange if it has Medium Priority, or Green if it has a Low Priority.

d) Expand the **LocalPriority** Input Parameter, then drag the **Label** attribute and drop it inside the **Text** placeholder of the **Tag** Widget.
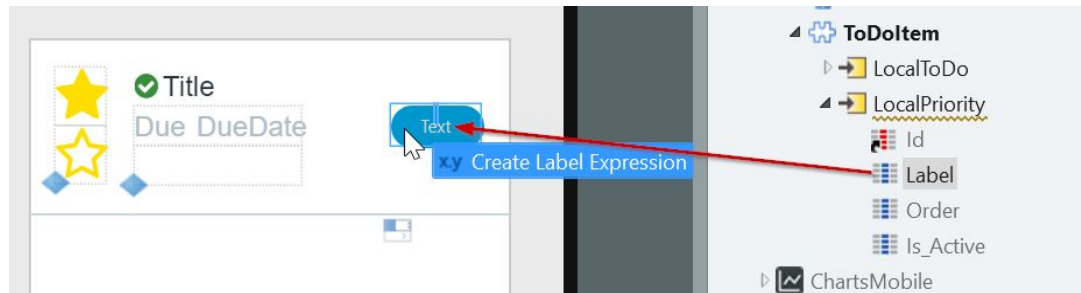
6) Make the **List Item** clickable and set its **Destination** to the **ToDoDetail** Screen. Set its Input Parameter to *LocalToDo.Id*

Setting a ToDo as starred and as complete

Now that the layout of the **ToDoItem** Block is defined, we will add some logic in the Block. First, we will allow users to star / unstar a ToDo, by clicking on the star icon within the Block. Then, we will allow a user to set a ToDo as completed, by swiping the ToDo on the ToDos Screen from right to left.

These two Actions inside the Block should follow the same logic for synchronization than the previous labs, saving the ToDo on the server and on the local storage when the app is online, and only on local storage when it is offline. In this latter case, the sync status of the ToDo record should be properly updated.

1) Define an Action with the functionality to mark a ToDo as starred or non starred. The stars will need to be clickable, which will trigger the Action. In the Action, the **IsStarred** attribute needs to be changed and the LocalToDo updated in the local storage and on the server, if the app is online. Don't forget to update the Sync Status accordingly, if the app is offline.

   a) Select the **If** Widget that contains both star icons, then right click on it and choose *Link to > (New Client Action)*
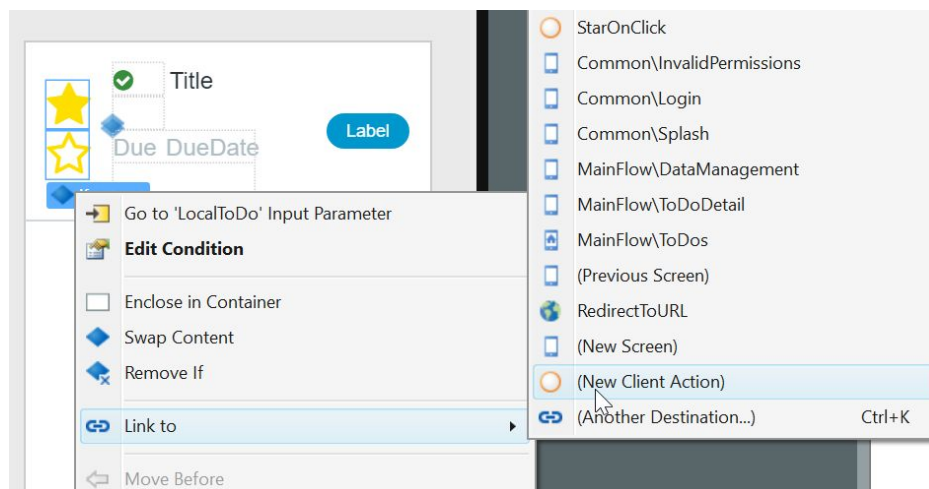
   b) Rename the Client Action to *StarOnClick*.

   c) Drag an **Assign** statement and drop it on Action flow. Add the following assignment

   LocalToDo.IsStarred = not LocalToDo.IsStarred

   This will change the IsStarred attribute accordingly. If the attribute has the value *True*, it will became *False* and vice-versa.

d) Drag an **If** statement and drop it after the Assign. Set its **Label** to *Offline?* and then set its **Condition** to

> *not GetNetworkStatus()*

This will divide the flow in two branches, one for when the app is online, and one for when the app is offline.
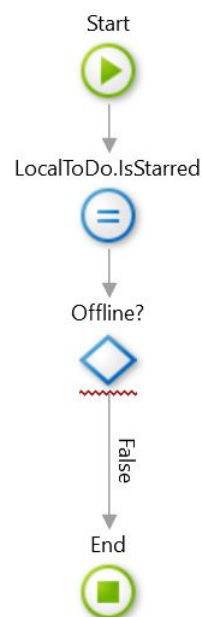
e) Drag and drop a new If on the right of the **Offline?** If. Connect the *True* branch of the **Offline?** If to the new one.

f) Set the **Label** of the new If to *Added SyncStatus?* and set its **Condition** to

> *LocalToDo.SyncStatusId = Entities.SyncStatus.Added*

Just like in the Data Synchronization exercise, here we are in the offline branch, meaning that the app is offline. Here, we check if the LocalToDo already has the SyncStatus of Added, meaning that it was created already while the app is offline. If so, we keep the SyncStatus as it is, since it will be added as a new ToDo in the server, when the synchronization is triggered.

g) Drag a **Run Client Action** statement below the Offline? If. Select the **UpdateLocalToDo** Action.
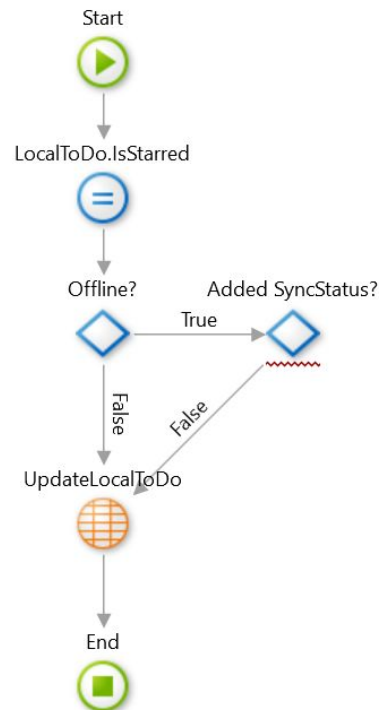
h) Set its **Source** as the *LocalToDo* Input Parameter of the Block.

i) Connect the True branch of the **Added SyncStatus?** If to the **UpdateLocalToDo** statement. This will make sure that if the Sync Status is already as *Added*, it will update immediately the LocalToDo in the Local Storage, without changing the Sync Status.

j) Drag an **Assign** and drop it right next to the **Added SyncStatus?** If. Connect the *False* branch of the If to the new Assign.

k) Add the following assignment to the new Assign statement

> *LocalToDo.SyncStatusId = Entities.SyncStatus.Updated*

If the ToDo was not marked as *Added* yet, it means that we are updating it while we are offline. Thus, we set its Sync Status as *Updated*.

l) Connect the Assign with the **UpdateLocalToDo** statement.

m) Now, we need to create the logic for when the app is online. Drag a new **Run Server Action** and place it between the **Offline?** If and the **UpdateLocalToDo** statement. In the next dialog, select the **UpdateToDoWrapper** Action.

n) Set the Input Parameters of the Action accordingly, using the attributes of the **LocalToDo** Input Parameter.

o) Drag a new Assign statement between the Offline? If and the Wrapper Action. Set the Assignment to

LocalToDo.SyncStatusId = Entities.SyncStatus.None

This will guarantee that the Sync Status stays as None, since the app is online and there is no need to save any status for future synchronizations.

p) The Action should look like this
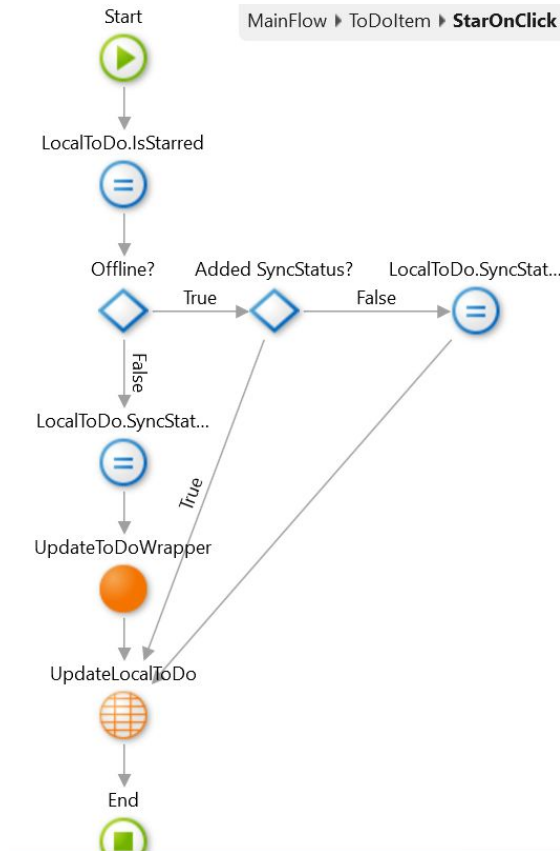
2) Make sure the **Link** for the stars is only **Enabled** when the ToDo is not completed yet, using the following Condition

   LocalToDo.CompletedDate = NullDate()
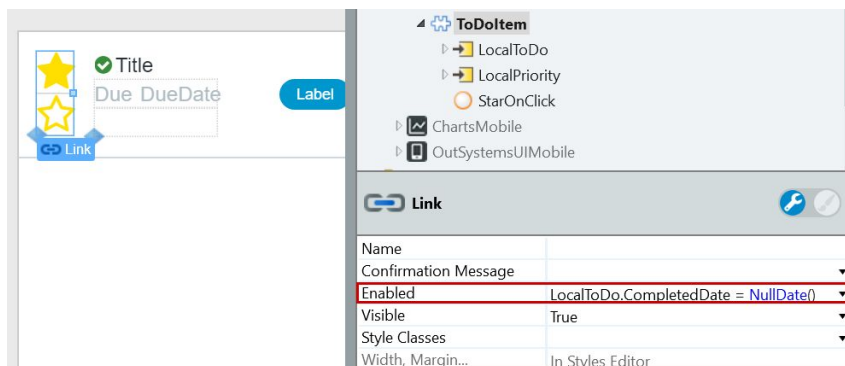
3) Define an Action with the functionality to mark a ToDo as completed, using a swipe movement. Don't forget to update the Sync Status accordingly, if the app is offline.

   a) Select the **List Item** Widget, then click the *Expand* circle to display the placeholders.
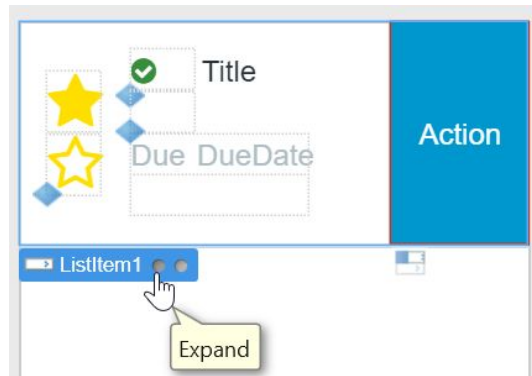
b) On the **List Action** on the right, change the **Action** text to *Complete*.

c) Select the **List Action** Widget, and set the **Visible** property to

   LocalToDo.CompletedDate = NullDate()

   This makes sure that the swipable action only appears if the ToDo has not been yet completed.
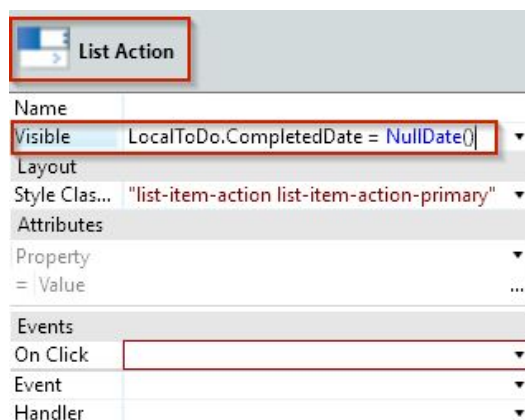
d) Set the **Style Classes** property to *"list-item-action background-green"*, to change the background color of the button to green.

e) Set the **On Click** Destination to *(New Client Action)*.
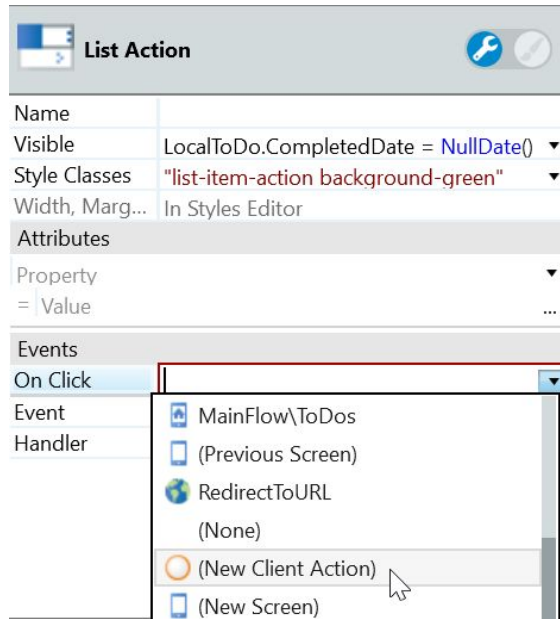
f) Define the newly created Action similarly to what was done for the StarOnClick. Here, instead of assigning the IsStarred attribute, we want to change the Completed Date to be equal to the current date:

LocalToDo.CompletedDate = CurrDate()

Make sure that the Sync Status is properly updated, when needed. The Action flow should look like this

Start

LocalToDo. Co...

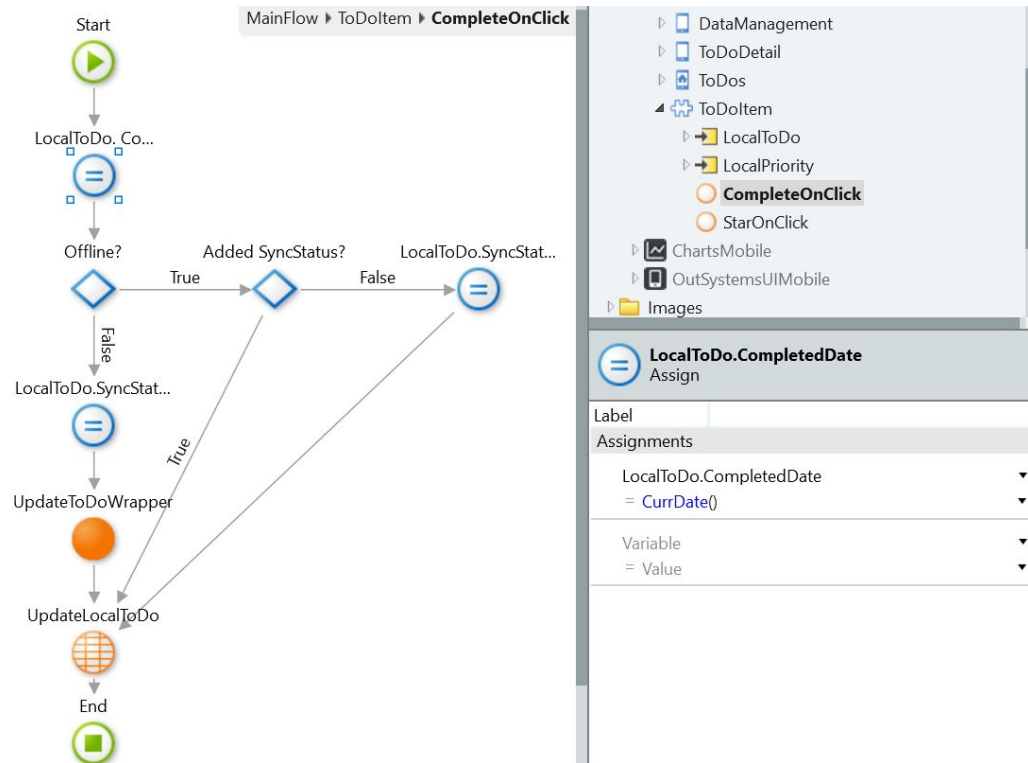Offline?   Added SyncStatus?   LocalToDo.SyncStat...
        True   False

False

LocalToDo.SyncStat...

True

UpdateToDoWrapper

UpdateLocalToDo

End

▷ 🔲 DataManagement
▷ 🔲 ToDoDetail
▷ 🔲 ToDos
◢ 🧩 ToDoItem
    ▷ ➜ LocalToDo
    ▷ ➜ LocalPriority
    ○ **CompleteOnClick**
    ○ StarOnClick
▷ 📈 ChartsMobile
▷ 🔲 OutSystemsUIMobile
▷ 📁 Images

LocalToDo.CompletedDate
Assign

Label	
Assignments	

|	
LocalToDo.CompletedDate	▼
= CurrDate()	▼

Variable	▼
= Value	▼

Instantiate the ToDoItem Block

Having the **ToDoItem** Block ready, it is time to use it on the ToDos Screen, replacing the content of the List that was previously created. Then, we will avoid any user to edit a ToDo that was already marked as complete, since it does not make sense to change any of its information.

1) Modify the **ToDos** Screen to display an instance of the **ToDoItem** Block for each item in the list of To Dos.

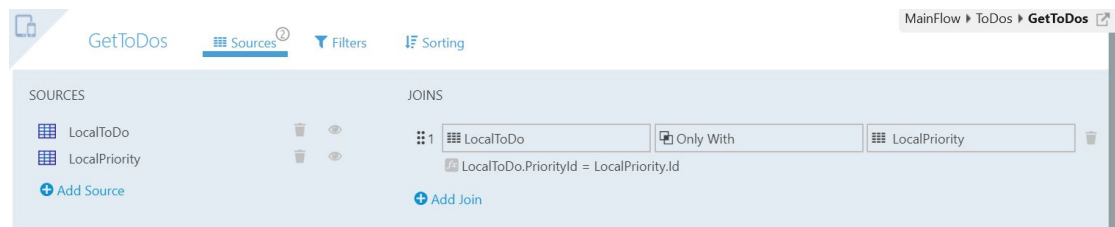   a) Open the **ToDos** Screen, select the **ListItem1** Widget and delete it.
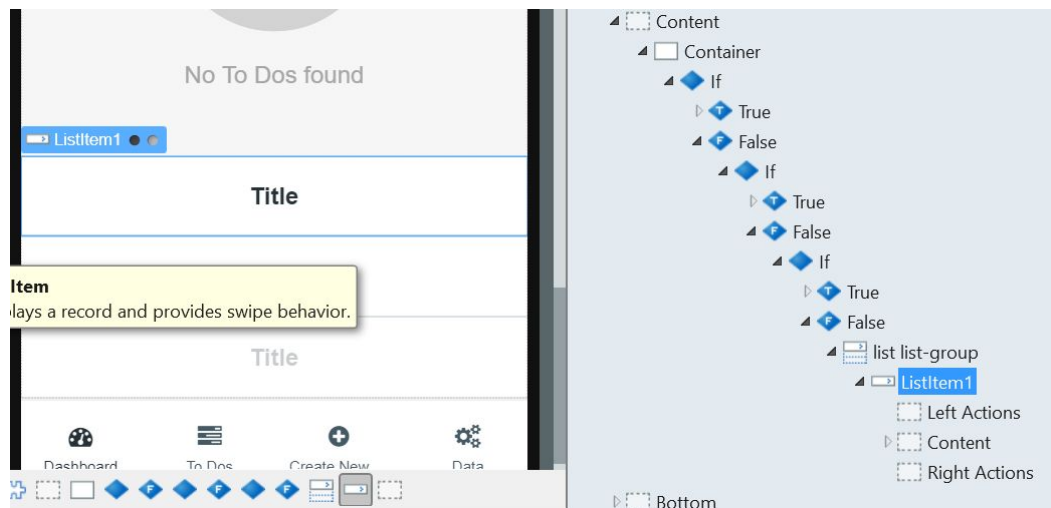
   b) Drag the **ToDoItem** Block and drop it inside the **List** Widget.
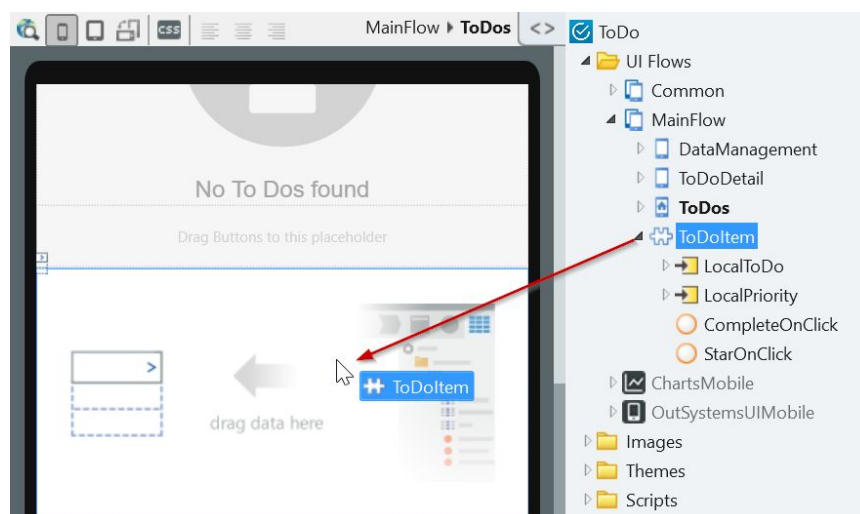
   c) Set the **LocalToDo** Input Parameter of the Block instance to *GetToDos.List.Current.LocalToDo*

> **NOTE:** Since the **GetToDos** Aggregate does not return the **LocalPriority** as Output, we cannot define the **LocalPriority** Block instance parameter, yet.

d) Double-click the **GetToDos** Aggregate of the **ToDos** Screen to open the Aggregate Editor.

e) From the **Data** tab, drag the **LocalPriority** Entity and drop it into the Aggregate Editor.

f) Verify that the Join between the **LocalPriority** and **LocalToDos** Entities was created.

g) Return to the **ToDos** Screen, select the **ToDoItem** Block instance, and set the **LocalPriority** parameter to *GetToDos.List.Current.LocalPriority*

h) Notice that the Title and Due Date are centered. Right-click the Block instance and choose *Enclose in Container*.
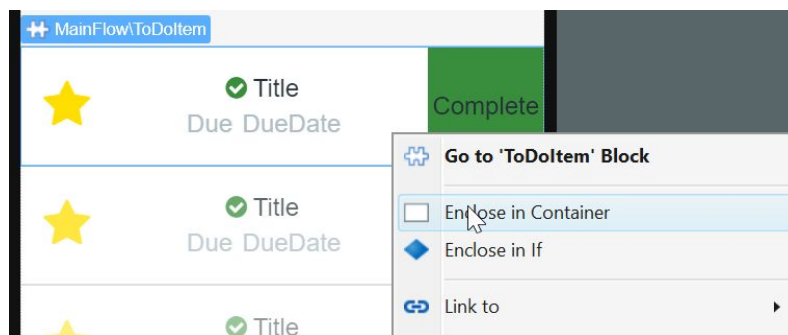
i) Align the Container to the left by using the top menu options.

> **NOTE:** These last two steps are required because in our application, the whole content of the Screen is enclosed in a Container aligned to Center, and that is being propagated to the inner Widgets.

2) Prevent users from modifying the To Dos already marked as completed, in the **ToDoDetail** Screen. Also, make sure that the Save button only appears if the ToDo is not completed, and that it is not possible to star / unstar a ToDo also if the it is already completed. **Hint:** Use the Enabled property of the Inputs.

   a) Open the **ToDoDetail** Screen.

   b) Select the **Input_Title** Widget and change the **Enabled** property to

   GetLocalToDoById.List.Current.LocalToDo.CompletedDate = NullDate()
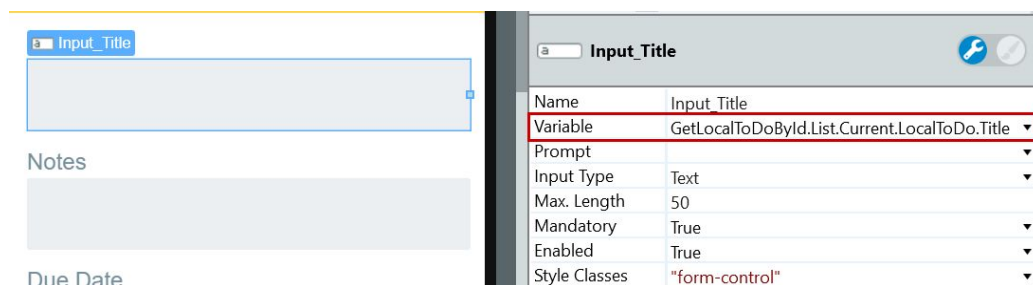
   c) Repeat the previous step for the remaining inputs (Notes, Due Date, Category and Priority).

   d) Change the **Visible** property of the **Save** Button to

   GetLocalToDoById.List.Current.LocalToDo.CompletedDate = NullDate()

   e) Select the **Link** around the star icons and change the **Enabled** property to

   GetLocalToDoById.List.Current.LocalToDo.CompletedDate = NullDate()

Publish and Test

Now that the Block is ready, it is time to publish and test the application in the browser or on the device.

1) Click the **1-Click Publish** button to publish the module to the server and open the application.

2) In the **ToDos** Screen, click the star icon of one of the ToDos that is not completed.

3) Notice that the star is now hollow, and the To Do has been updated.

4) Swipe the same To Do from the right to the left.

NOTE: In the browser, we can click and drag to emulate a swipe action.

5) The green check mark should now appear next to the title.

6) Click the completed To Do to open the detail Screen, and verify that the **Form** is disabled and the To Do cannot be modified.

Sort the ToDos by importance and due date

At this point, the list of ToDos does not have a specific sorting. In this section, we will change this by sorting the ToDos by their importance, meaning if they are starred or not, and then differentiate by their Due Date.

1) Change the **GetToDos** Aggregate in the **ToDos** Screen to sort by the **IsStarred** attribute, showing the starred ToDos before the non starred. Then, within the starred and non starred ToDos, sort them by **Due Date**, from the most recent one to the latest one in time.

   a) Open the **GetToDos** Aggregate in the **ToDos** Screen.

   b) Open the **Sorting** tab on the Aggregate and select **+Add Sort**.

   c) In the **Select Attribute** dialog, select the *LocalToDo.IsStarred* and change the **Ascending** to the *Descending* order.
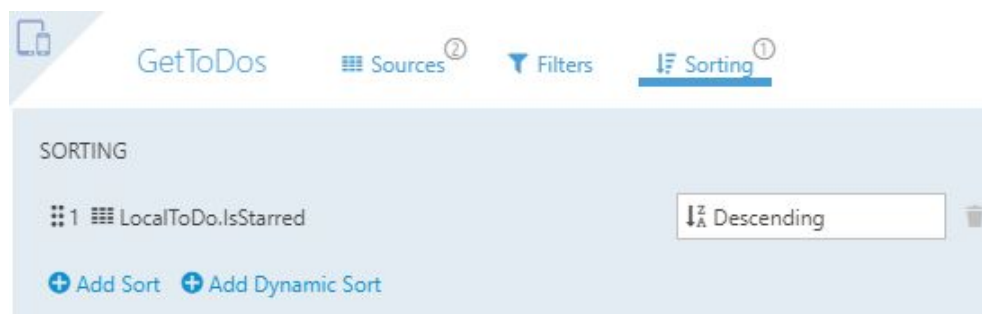

   d) Add a second Sort to the Aggregate by the *LocalToDo.DueDate*, in an *Ascending* order.

2) Publish the module and test the application.

   a) Click the 1-Click Publish button and open the application.

   b) Confirm that the ToDos are properly sorted, according to what was defined in the previous step.

   c) Star one ToDo that would reinforce a change of order in the listed ToDos. Confirm that the sorting didn't happen.

   This situation occurs because the option to star / unstar a ToDo exists inside the ToDoItem Block. The UI automatically refreshes, but since the GetToDos Aggregate is in the scope of the Screen, and not on the Block, it does not know that it needs to run again to update the list of results. For that matter, we need to create an Event on the Block that warns the Screen, when a star is clicked, that an important change happened. The Screen then needs to refresh the Aggregate.

3) Create an Event on the **ToDoItem** Block, called *ToDoStarred*. This Event should then be triggered when a ToDo is starred. Make sure that the parent Screen (**ToDos**) handle the Event properly, by executing again the **GetToDos** Aggregate.

    a) Under the **Interface** tab, right-click on the **ToDoItem** Block and select *Add Event*.
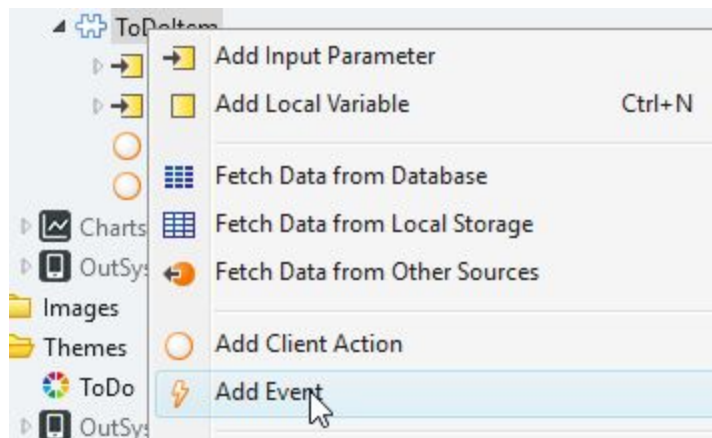
    b) Set the **Name** of the Event as *ToDoStarred*.

    c) Open the **StarOnClick** Action and drag a **TriggerEvent** statement right before the End.
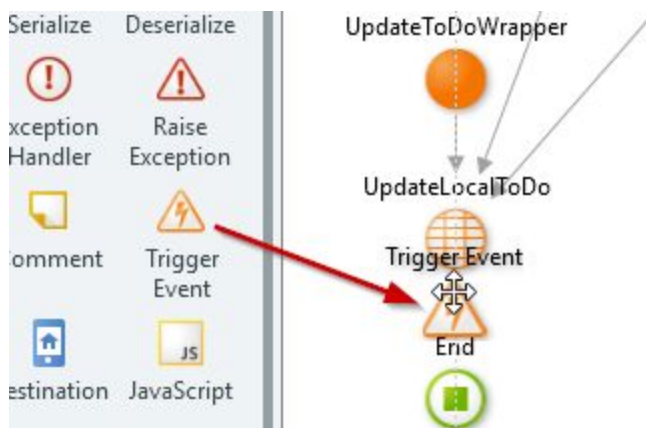
    d) In the **Select Event** dialog, choose the *ToDoStarred* Event. This will make sure that the Event is triggered to the parent, when the user stars / unstars a ToDo.

    e) In the **TrueChange** tab, double-click on the error that appears. It will open the ToDos Screen, with focus on the instance of the ToDoItem Block.

    f) Since the Block now triggers a mandatory Event, the parent needs to handle the Event. Set the Handler property to *(New Client Action)*.
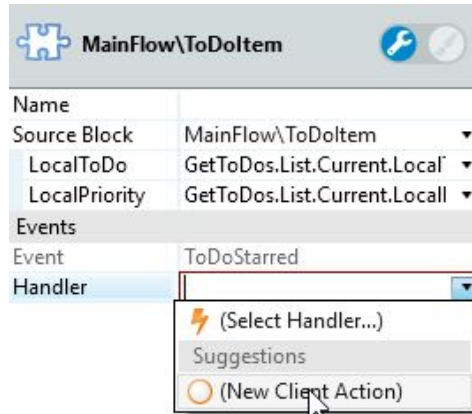
g) Set the **Name** of the Action to *RefreshToDos*.

h) Open the **RefreshToDos** Action and drag and drop a **Refresh Data** statement to the Action flow.

i) Select the **GetToDos** Aggregate. Here, we are forcing the Aggregate to run again, when the ToDoStarred Event is triggered.

4) Publish the module and test the application. Confirm that now the list of ToDos updates properly after setting a ToDo as starred or non starred.

End of Lab

In this exercise lab, we created a new Block to display the information of each To Do on the ToDos Screen. The Block uses a List Item Content, with some Expressions to display the Title and Due Date, some icons to display if the ToDo is starred and / or completed, and a tag to display the Priority of the ToDo.

We also created some logic to star a ToDo when clicking on the star icon, and to set a ToDo as complete, by swiping from right to left.

Then, we used the ToDoItem Block to display the ToDo information on the the list of To Dos, and created logic to avoid any user to edit a ToDo when it is already completed.

Finally, we sorted the list of ToDos by their importance and due date. For that to work, we created an Event in the Block, to warn the parent when a ToDo was set to starred or non starred.