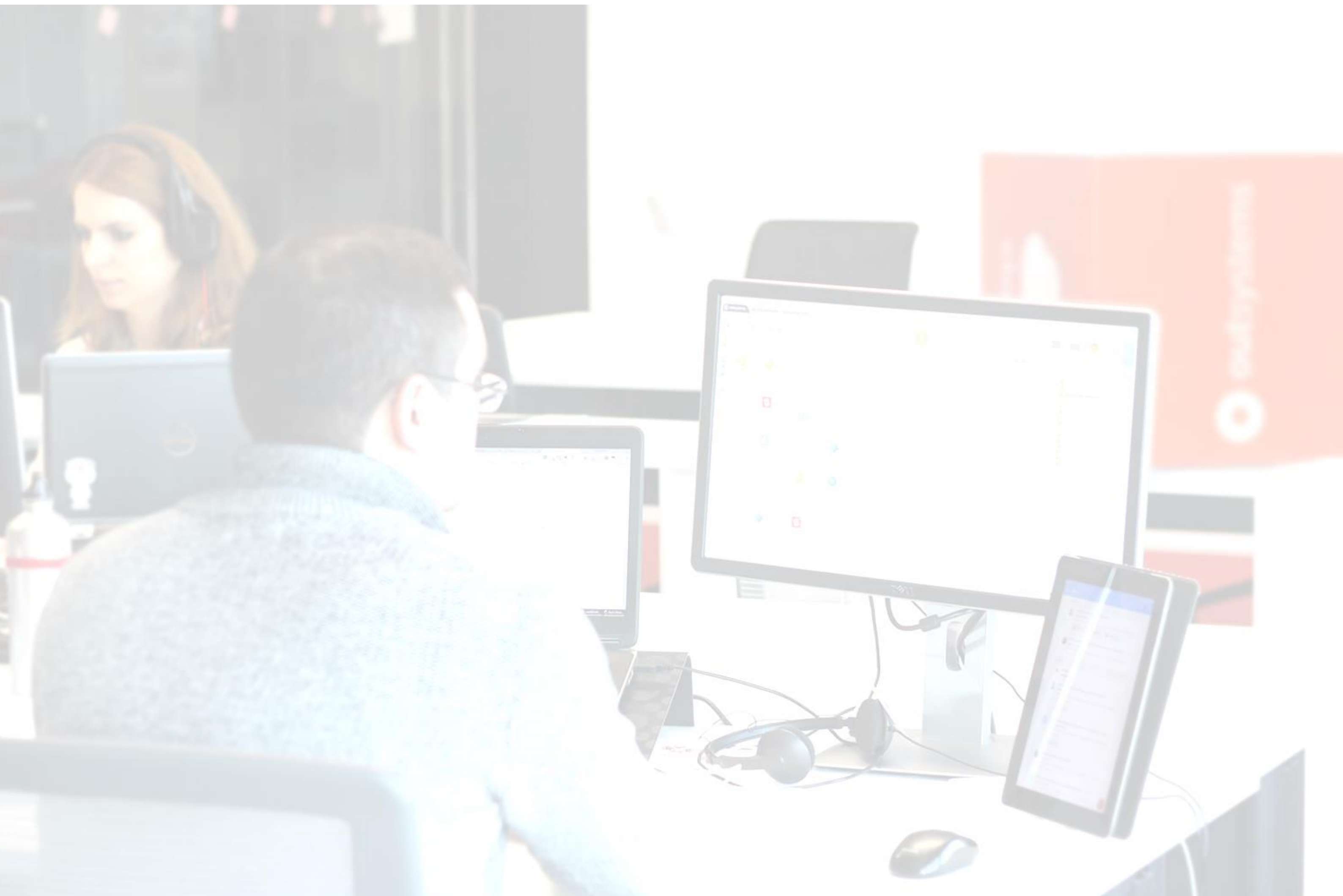
DEVELOPING OUTSYSTEMS MOBILE APPS

Local Storage



Introduction

Over the course of this set of exercise labs, you will create a mobile application. The application will focus on creating and managing To Dos. The To Dos will be persisted in a database so they can be accessed from and shared across multiple devices. To Dos will have attributes such as category, priority (low, medium or high), due date and they can be marked as important (starred) by the user.

Users of the To Do application will be able to access all of this information regardless of whether the device is online or offline. When offline, users will still be able to keep interacting with the application and changes will be saved locally in the device local storage. When the device returns to online mode, changes made while offline will automatically be synced to the server.

You constantly will be expanding your application, publishing it to the server and testing it in your mobile device. Throughout the process, you will be learning and applying new OutSystems concepts.

At the end of this set of exercise labs, you will have a small, but well-formed application, spanning multiple screens and concepts that you can easily access from your mobile device.

In this specific exercise lab, you will:

- Create Local Entities based on the Database Entities
- Fetch data from Local Storage
- Create and Update data in Local Storage
- Synchronize data to Local Storage
- Display Feedback Messages during synchronization

Table of Contents

Part 1: Create Local Entities

In this part of the exercise, you will create Local Entities based on the already existing Database (Server) Entities.

1. Create the Local Entities based on the existing Database Entities.

a) Switch to the **Data** tab and expand the **ToDo_Core** module to view the existing Entities.

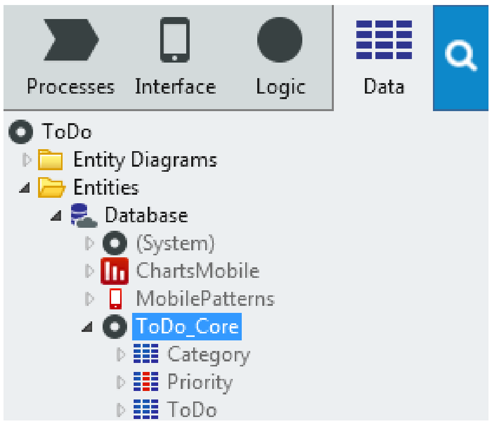


Figure 1. ToDo_Core Entities

b) Right-click the **Local Storage** element and choose **Add Entity from Database**.

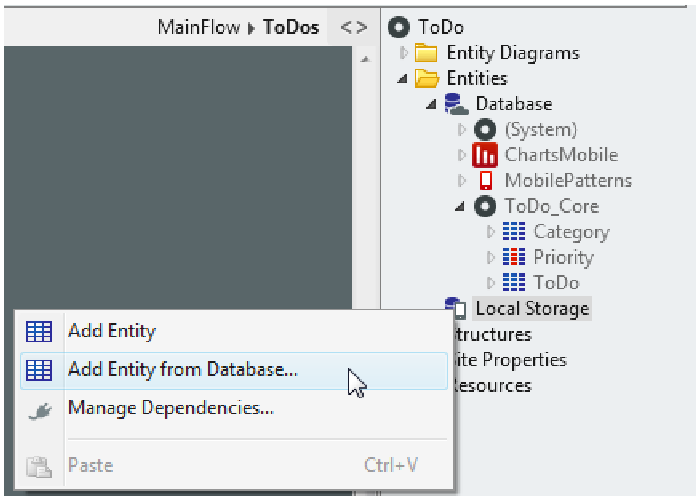


Figure 2. Add Entity from Database

c) In the **Select Entity** dialog, choose the **Category** Entity and then click **Ok**.

d) Notice that a new **LocalCategory** Entity was created and has the same structure as the **Category** Database Entity.

e) Repeat the previous steps to create the **LocalPriority**, **LocalResource**, **LocalResourceType** and **LocalToDo** Local Entities.

NOTE: The **Data Type** of the Entity Identifiers from Local Storage have the Database Entity identifier type.

f) Expand the **LocalToDo** Entity and delete the **UserId** attribute.

NOTE: Local Entities exist in the device of each user, therefore there is no need to also have the **UserId** attribute, as it would match the identifier of the logged in user. In the **LocalToDo** Entity, we will only store data from the current logged in user.

Part 2: Fetch Data from Local Storage

In this part of the exercise, you will modify the **ToDos** and **ToDoDetail** Screens to fetch data from the Local Storage, instead of using the Database.

1. Modify the **ToDos** Screen to fetch data from the Local Storage, instead of using the Database.

a) Switch to the **Interface** tab and expand the **ToDos** Screen.

b) Delete the **GetTodosByUserId** Aggregate.

NOTE: At this point, you will see several errors in the **TrueChange** tab. We will fix them later during this exercise.

c) Right click the **ToDos** Screen and choose **Fetch Data from Local Storage**.

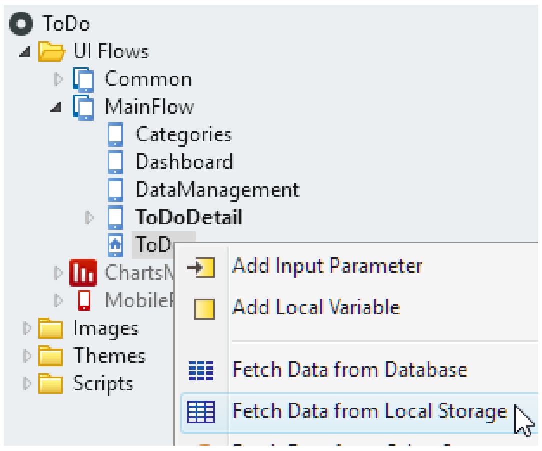
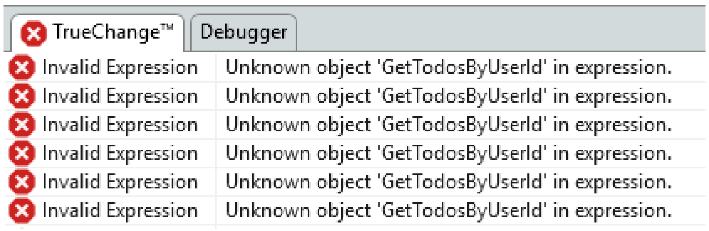


Figure 3. Fetch Data from Local Storage

d) From the **Data** tab drag the **LocalToDo** Entity and drop it inside the Aggregate Editor.

NOTE: The Local Storage Aggregate is created with the name 'GetToDos'. This happens because the **LocalToDo** Entity was created based on the **ToDo** Database Entity, which has a customized plural label 'ToDos'. As it is a customized label, the new **LocalToDo** Entity inherits it. Entities with automatic / default plural labels will have the 'Local' prefixed to the plural, e.g. Local Categories.

e) Open the **TrueChange** tab to view the existing errors. You should have the following errors

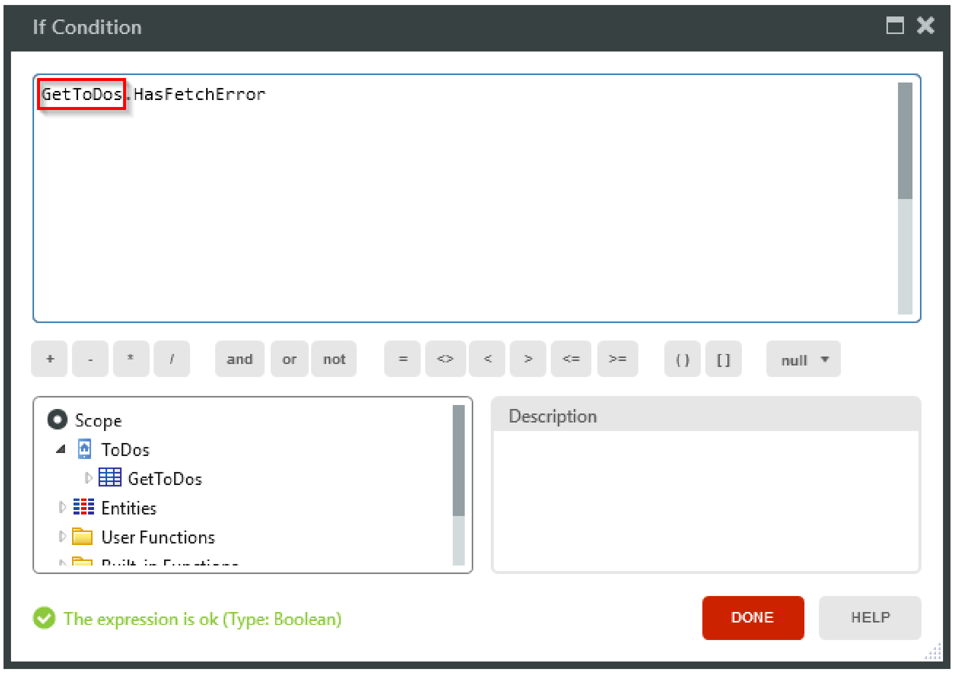


Invalid Expression Unknown object 'GetTodosByUserId' in expression.
Invalid Expression Unknown object 'GetTodosByUserId' in expression.
Invalid Expression Unknown object 'GetTodosByUserId' in expression.
Invalid Expression Unknown object 'GetTodosByUserId' in expression.
Invalid Expression Unknown object 'GetTodosByUserId' in expression.
Invalid Expression Unknown object 'GetTodosByUserId' in expression.

Figure 4. TrueChange errors

NOTE: The **TrueChange** errors were caused by deleting the **GetToDosByUserId** Aggregate, as it was used in several Expressions. To fix the errors, you need to change the references to the old Aggregate, to use the recently created Local Storage Aggregate, **GetToDos**.

f) Double-click each one of the errors and replace the **GetToDosByUserId** for **GetToDos**, in all Expressions.

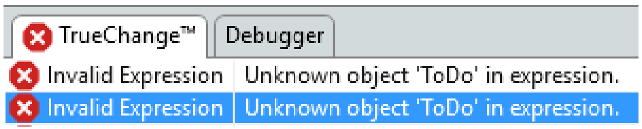


```
GetToDos.HasFetchError
```

The expression is ok (Type: Boolean)

Figure 5. Fix errors in Expressions

g) After fixing the errors, the **TrueChange** tab should display two new errors.



Invalid Expression Unknown object 'ToDo' in expression.
Invalid Expression Unknown object 'ToDo' in expression.

Figure 6. New TrueChange errors

NOTE: These new **TrueChange** errors were caused because the Local Storage Aggregate uses the **LocalToDo**, while the **GetToDosByUserId** used the **ToDo** Database Entity. To fix the errors, you need to change the references from **ToDo** to the Local Storage Entity, **LocalToDo**.

---

h) Double-click each one of these new errors and replace the **ToDo** by **LocalToDo**.

2. Modify the **ToDoDetail** Screen to fetch data from the Local Storage instead of using the Database.

a) Switch to the **Interface** tab and expand the **ToDoDetail** Screen.

b) Delete the **GetCategories** Aggregate.

c) Right click the **ToDoDetail** Screen and choose 'Fetch Data from Local Storage'.

d) From the **Data** tab drag the **LocalCategory** and drop it inside the Aggregate Editor.

e) Rename the **ToDoId** Input Parameter of the **ToDoDetail** Screen to 'LocalToDoId', and change the **Data Type** to 'LocalToDo Identifier'

f) Delete the **GetToDoById** Aggregate.

g) Right-click the **ToDoDetail** Screen and choose 'Fetch Data from Local Storage'.

h) Drag the **LocalToDoId** Input Parameter to the Aggregate Editor.

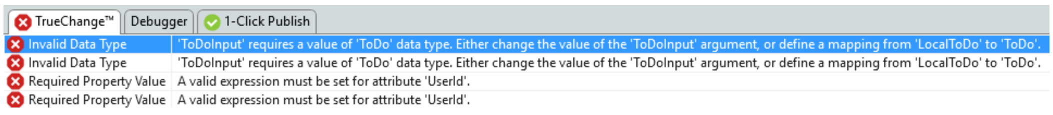
NOTE: By dragging a **LocalToDo Identifier**, the Platform automatically added the **LocalToDo** Entity as a Source and created a filter for the To Dos by Id.

---

i) In the **TrueChange** tab you should have several errors.

j) Fix the errors regarding Expressions and Variables, by double-clicking them and replacing **GetToDoById** with **GetLocalToDoById**, **ToDo** with **LocalToDo**, **GetCategories** with **GetLocalCategories** and **Category** with **LocalCategory**.

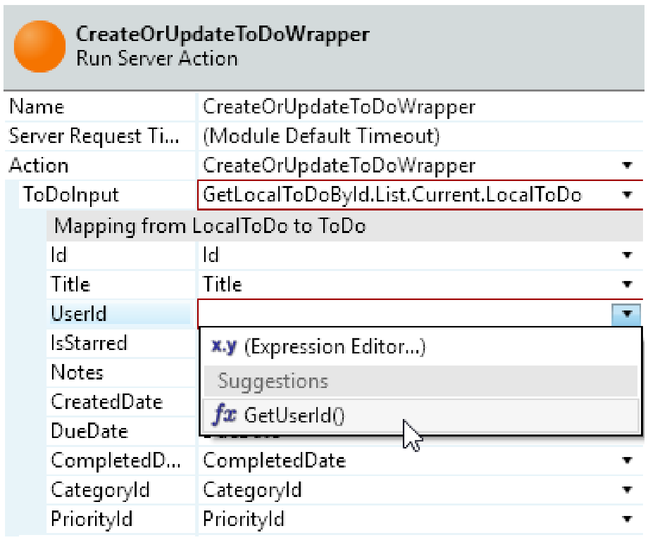
k) The following errors require a bit more attention to be fixed



Invalid Data Type	'ToDoInput' requires a value of 'ToDo' data type. Either change the value of the 'ToDoInput' argument, or define a mapping from 'LocalToDo' to 'ToDo'.
Invalid Data Type	'ToDoInput' requires a value of 'ToDo' data type. Either change the value of the 'ToDoInput' argument, or define a mapping from 'LocalToDo' to 'ToDo'.
Required Property Value	A valid expression must be set for attribute 'UserId'.
Required Property Value	A valid expression must be set for attribute 'UserId'.

Figure 7. TrueChange errors

l) To fix these errors you need to update the mapping from **LocalToDo** to **ToDo**. In the mapping set the **UserId** attribute to `GetUserId()`.



CreateOrUpdateToDoWrapper
Run Server Action

Name	CreateOrUpdateToDoWrapper
Server Request Ti...	(Module Default Timeout)
Action	CreateOrUpdateToDoWrapper
ToDoInput	GetLocalToDoById.List.Current.LocalToDo
Mapping from LocalToDo to ToDo	
Id	Id
Title	Title
UserId	
IsStarred	
Notes	
CreatedDate	
DueDate	
CompletedD...	CompletedDate
CategoryId	CategoryId
PriorityId	PriorityId

Figure 8. Mapping from LocalToDo to ToDo

m) Repeat the previous step for the remaining error.

Part 3: Modify Data also in Local Storage

In this part of the exercise, you will edit already existing Client Actions so that they also update the data that is in Local Storage.

1. Modify the **SaveOnClick** Client Action of the **ToDoDetail** Screen to also update Local Storage.

a) Open the **SaveOnClick** Client Action of the **ToDoDetail** Screen.

b) Drag an **Assign** statement and drop it between the **CreateOrUpdateToDoWrapper** and the **Message** statements.

c) In the new **Assign** statement define the following assignment

```
GetLocalToDoById.List.Current.LocalToDo.Id =
CreateOrUpdateToDoWrapper.ToDoId
```

NOTE: This **Assign** will make sure that the Id of To Do created locally, will be exactly the same as in the To Do created in the server.

d) Set the **Label** property of the Assign statement to 'Set LocalToDo.Id'.

e) Drag a **Run Client Action** statement and drop it between the Assign and the Message statements.

f) In the **Select Action** dialog, choose the **CreateOrUpdateLocalToDo** Action.

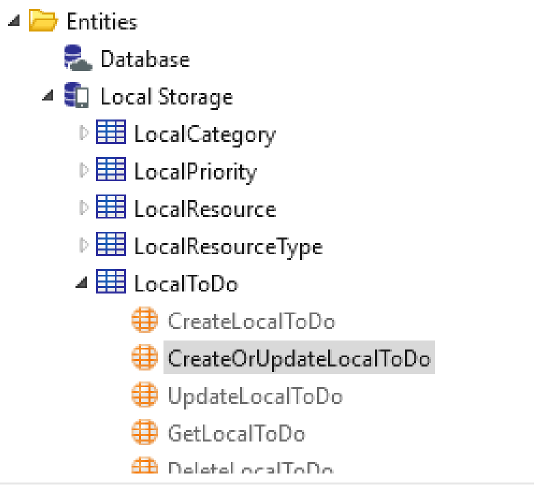


Figure 9. CreateOrUpdateLocalToDo Action

g) Set the **Source** property to 'GetLocalToDoById.List.Current'.

2. Modify the **StarOnClick** Client Action of the **ToDoDetail** Screen to also update Local Storage.

a) Open the **StarOnClick** Client Action of the **ToDoDetail** Screen.

b) Drag a **Run Client Action** statement and drop it between the **CreateOrUpdateToDoWrapper** and End.

c) In the **Select Action** dialog, choose the **CreateOrUpdateLocalToDo** Client Action.

d) Set the **Source** property to ‘GetLocalToDoById.List.Current’.

Part 4: Synchronize Data to Local Storage

In this part of the exercise, you will create the logic that allows the synchronization of data between the server and Local Storage.

1. Create buttons in the **DataManagement** Screen to allow to sync data with the server and to clear Local Storage.

a) Open the **DataManagement** Screen from the **Interface** tab.

b) Drag a **Button** Widget, drop it inside the Content placeholder of the Screen and center it in the Screen.

c) Change the text of the Button to 'Clear Local Storage'.

d) Change its **Style Classes** property to "btn btn-danger"**.**

e) Double-click the Button to create the **ClearLocalStorageOnClick** Client Action.

f) Drag a **Run Client Action** statement and drop it between Start and End, in the recently created Action flow.

g) In the **Select Action** dialog choose **New Client Action**.

NOTE: The **Action1** Client Action is also accessible from the **Logic** tab, under the Client Actions folder.

h) Drag a **Message** statement and drop it between the **Action1** statement and End.

i) Set the **Message** property to "All data cleared!", and the **Type** to 'Success'.

j) Double-click the **Action1** statement to open it.

k) Rename the Client Action to 'ClearAllLocalStorage'.

l) From the **Data** tab drag the **DeleteAllLocalResources** Action, located under the **LocalResource** Entity in Local Storage, and drop it just after the Start.

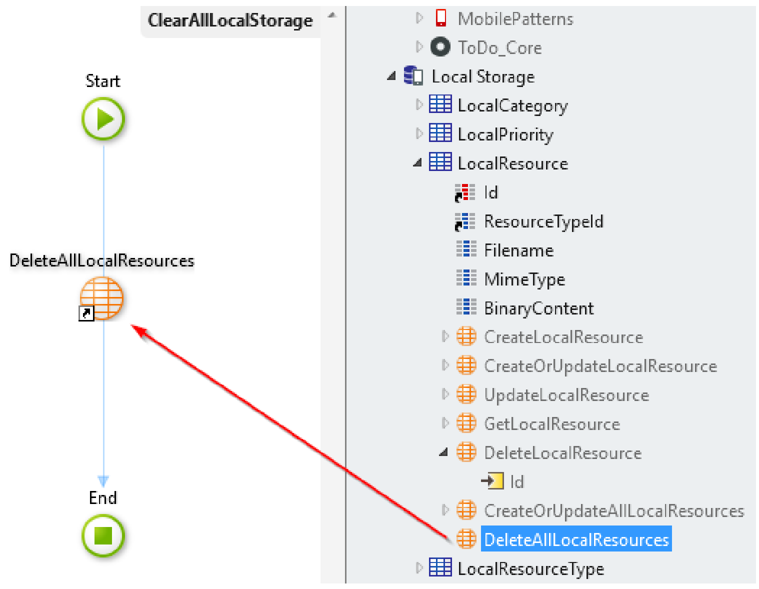


Figure 10. DeleteAllLocalResources Entity Action

m) Drag the **DeleteAllLocalToDoes** Entity Action and drop it after the previous added Action.

n) Next, drag and drop the **DeleteAllLocalCategories**, **DeleteAllLocalPriorities** and **DeleteAllLocalResourceTypes**.

o) The **ClearLocalStorageOnClick** should look like this

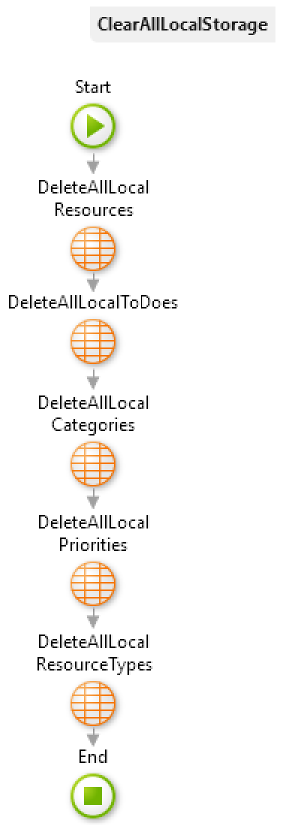

ClearAllLocalStorage
Start
DeleteAllLocal Resources
DeleteAllLocalToDoes
DeleteAllLocal Categories
DeleteAllLocal Priorities
DeleteAllLocal ResourceTypes
End

Figure 11. ClearAllLocalStorage Client Action

p) Return to the **DataManagement** Screen.

q) Drag another **Button** Widget and drop it below the **Clear Local Storage** Button.

r) Select the Button, align it to the center of the Screen and change its text to 'Sync to Local Storage'.

s) Set the **Style Classes** property to "btn btn-success".

t) Double-click the Button to create the **SynctoLocalStorageOnClick** Client Action.

u) Drag a **Run Client Action** and drop it between Start and End.

v) In the Select Action dialog choose **TriggerOfflineDataSync**.

2. Clear the Local Storage when the user logs out.

a) From the **Interface** tab, open the **ClientLogout** Client Action of the **Menu** Block.

b) Drag a **Run Client Action** statement and drop it between the **DoLogout** statement and the **Login** destination.

c) In the **Select Action** dialog, choose **ClearAllLocalStorage** Client Action.

3. Create the server synchronization logic to obtain all user data.

a) Switch to the **Logic** tab and open the **ServerDataSync** Server Action located under the **OfflineDataSync** folder.

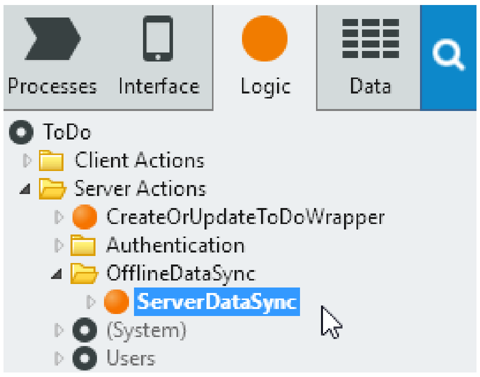


Figure 12. ServerDataSync Server Action

NOTE: The **ServerDataSync** Server Action is automatically added to your module in mobile apps. This Server Action is used to transfer the application data from the server Database to the devices' Local Storage.

b) From the **Data** tab, drag the **Category** Database Entity and drop it in the flow below the **LogMessage** statement to create the **GetCategories** Aggregate.

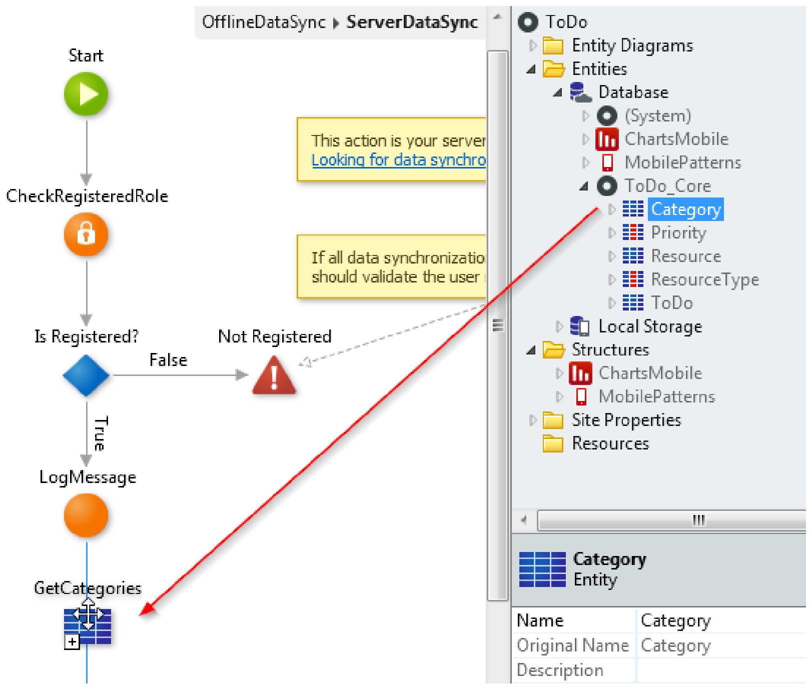


Figure 13. GetCategories Aggregate

c) Repeat the previous step for the **Priority, ResourceType, ToDo** and **Resource** Entities.

d) Double-click the **GetToDos** Aggregate to open the Aggregate Editor, and add the following filter

```
ToDo.UserId = GetUserId()
```

NOTE: By adding such filter, we ensure that a device will only receive the To Dos that belong to the logged in user.

e) Return to the **ServerDataSync** Server Action.

f) Double-click the **GetResources** Aggregate to open the Aggregate Editor.

g) In the Sources tab, you should have the **ToDo** and **Resource** Entities, and the respective join condition between both Entities.

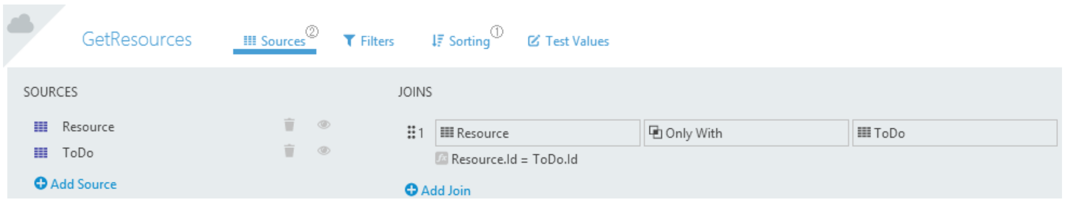


Figure 14. GetResources sources

h) Add the following filter

```
ToDo.UserId = GetUserId()
```

NOTE: Recall that the join condition is set as **Only With,** which means that will return only Resources associated to To Dos of the logged in user.

i) In the **Logic** tab, right-click the **ServerDataSync** Server Action and choose **Add Output Parameter**.

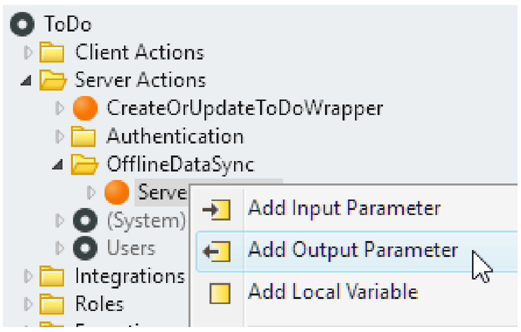


Figure 15. Add Output Parameter to ServerDataSync Server Action

j) Set the Output Parameter **Name** to 'LocalCategories' and verify that the **Data Type** is set to 'LocalCategory List'.

k) Add another Output Parameter, name it 'LocalPriorities' and verify that the **Data Type** is 'LocalPriority List'.

l) Add another Output Parameter, name it 'LocalResourceTypes' and verify that the **Data Type** is 'LocalResourceType List'.

m) Add another Output Parameter, name it 'LocalToDos' and verify that the **Data Type** is 'LocalToDo List'. Note here that the type inference did not work, because the **LocalToDo** plural Label is 'ToDos', as the Label in the respective Database Entity (**ToDo**) was changed.

n) Add another Output Parameter, name it 'LocalResources' and verify that the **Data Type** is 'LocalResource List'.

o) In the **ServerDataSync** Server Action flow, add an **Assign** statement between the **GetResources** Aggregate and the End.

p) Define the following assignments and respective mappings

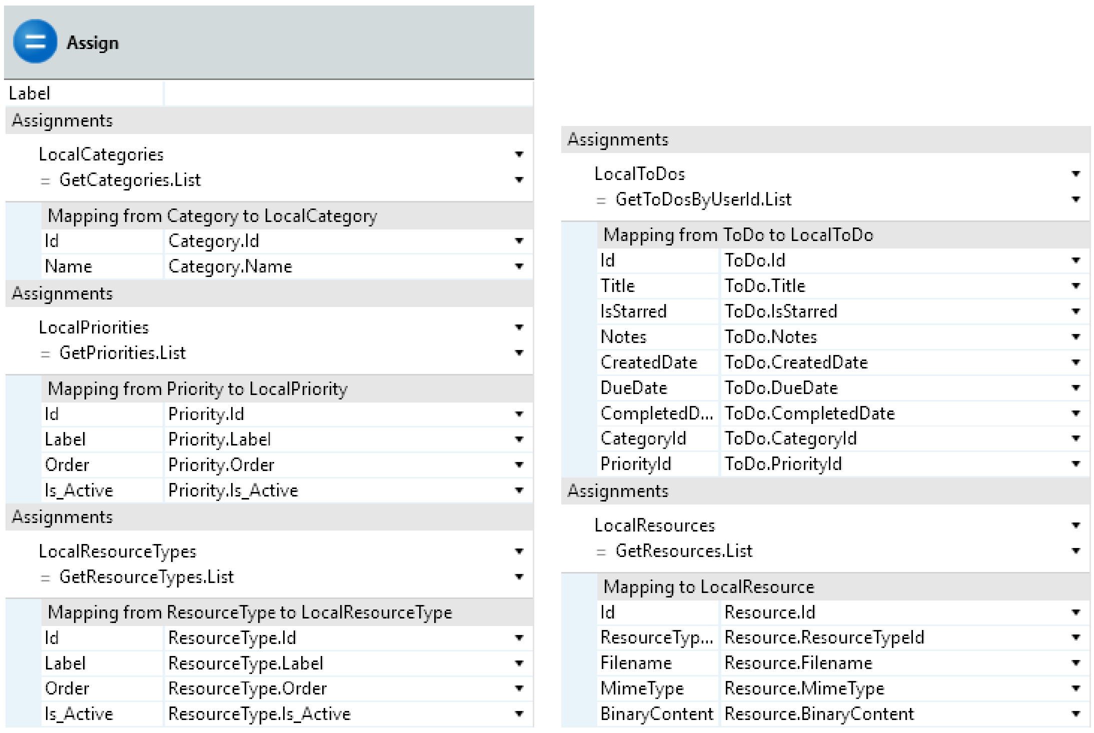
```
LocalCategories = GetCategories.List

LocalPriorities = GetPriorities.List

LocalResourceTypes = GetResourceTypes.List

LocalToDos = GetToDosByUserId.List

LocalResources = GetResources.List
```



Assign

Label	
Assignments	
LocalCategories	
= GetCategories.List	
Mapping from Category to LocalCategory	
Id	Category.Id
Name	Category.Name
Assignments	
LocalPriorities	
= GetPriorities.List	
Mapping from Priority to LocalPriority	
Id	Priority.Id
Label	Priority.Label
Order	Priority.Order
Is_Active	Priority.Is_Active
Assignments	
LocalResourceTypes	
= GetResourceTypes.List	
Mapping from ResourceType to LocalResourceType	
Id	ResourceType.Id
Label	ResourceType.Label
Order	ResourceType.Order
Is_Active	ResourceType.Is_Active
Assignments	
LocalToDos	
= GetToDosByUserId.List	
Mapping from ToDo to LocalToDo	
Id	ToDo.Id
Title	ToDo.Title
IsStarred	ToDo.IsStarred
Notes	ToDo.Notes
CreatedDate	ToDo.CreatedDate
DueDate	ToDo.DueDate
CompletedD...	ToDo.CompletedDate
CategoryId	ToDo.CategoryId
PriorityId	ToDo.PriorityId
Assignments	
LocalResources	
= GetResources.List	
Mapping to LocalResource	
Id	Resource.Id
ResourceTyp...	Resource.ResourceTypeId
Filename	Resource.Filename
MimeType	Resource.MimeType
BinaryContent	Resource.BinaryContent

Figure 16. Output assignments with mappings

NOTE: Each Aggregate returns a list of the Database Entity type, and the Outputs are set to lists of the Local Storage Entity types, therefore it is required to define a mapping between both.

4. Retrieve server data and update Local Storage.

a) In the **Logic** tab, locate and open the **OfflineDataSync** Client Action located under the **OfflineDataSync** folder in the **Client Actions**.

NOTE: The **OfflineDataSync** Client Action already has a Run Server Action statement to call **ServerDataSync**. Recall that this Server Action returns the data from the Database to sync to the Local Storage of the device.

b) Drag a **Run Client Action** and drop it between the **ServerDataSync** statement and End.

c) In the **Select Action** dialog, choose the **ClearAllLocalStorage** Action.

NOTE: The synchronization pattern being implemented here is the 'Read Only'. The first step is to clear all data form Local Storage, and then all data will be fetched from the server and recreated in the Local Storage.

d) Drag another **Run Client Action** and drop it between the previous one and the End.

e) In the **Select Action** dialog, choose the **CreateOrUpdateAllLocalPriorities** local Entity Action.

f) Set the **SourceList** parameter to 'ServerDataSync.LocalPriorities'.

g) Drag another **Run Client Action** and drop it between the previous one and the End.

h) Choose the **CreateOrUpdateAllLocalCategories** local Entity Action, and set the **SourceList** parameter to 'ServerDataSync.LocalCategories'.

i) Repeat the previous two steps for the **LocalResourceType**, **LocalToDo** and **LocalResource** local Entities.

j) Drag a **Message** statement and drop it just before the End.

k) Set the **Message** parameter to "Server data synched to Local Storage." and the **Type** to 'Info'.

l) The **OfflineDataSync** Client Action should look like this

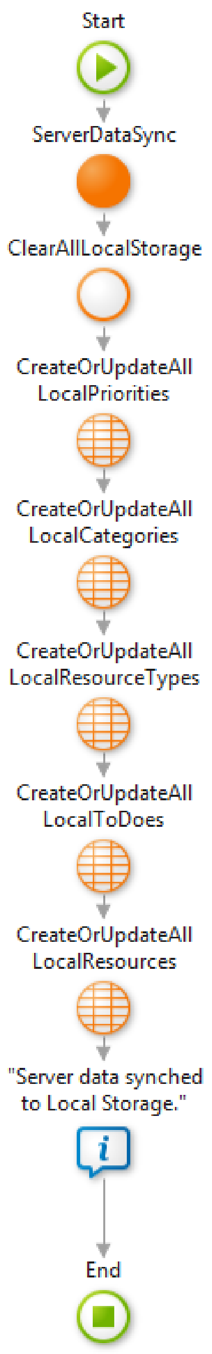

Start
ServerDataSync
ClearAllLocalStorage
CreateOrUpdateAll LocalPriorities
CreateOrUpdateAll LocalCategories
CreateOrUpdateAll LocalResourceTypes
CreateOrUpdateAll LocalToDoes
CreateOrUpdateAll LocalResources
"Server data synched to Local Storage."
End

Figure 17. OfflineDataSync Client Action

Part 5: Publish and Test

In this part of the exercise, you will publish the **ToDo** application to the server, and then you will test the synchronization to Local Storage.

1. Publish and test the synchronization to Local Storage.

a) Click the **1-Click Publish** button to publish the module to the server.

b) Click the **Open in Browser** button to open the application.

c) The list of To Dos should be empty since the Local Storage is empty at this point.

d) Navigate to the **Data Management** Screen, then click the **Sync to Local Storage** Button.

e) After some seconds, you should see the synchronization feedback message.

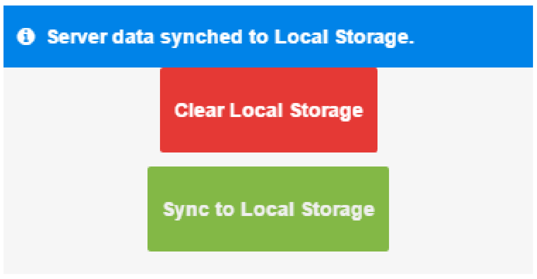


Figure 18. Synchronization feedback message

f) Navigate to the **ToDos** Screen and verify that the To Dos appear.

Part 6: Is Syncing Feedback

In this part of the exercise, you will develop the logic to display a feedback message to the user while the application is synching.

1. Display a feedback message to the user when the application is synching.

a) Switch to the **Interface** tab and open the **Layout** block.

b) Add a new **Local Variable** to the **Layout** Block and name it **IsSyncing**.

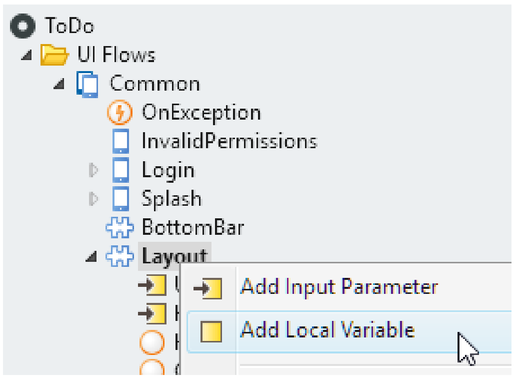


Figure 19. Add Local Variable

c) Verify that the **Data Type** of **IsSyncing** was automatically set to ‘Boolean’.

d) Drag a **Container** Widget and drop it just above the **content** Container, and at the same level as the **PullToRefresh If** Widget.

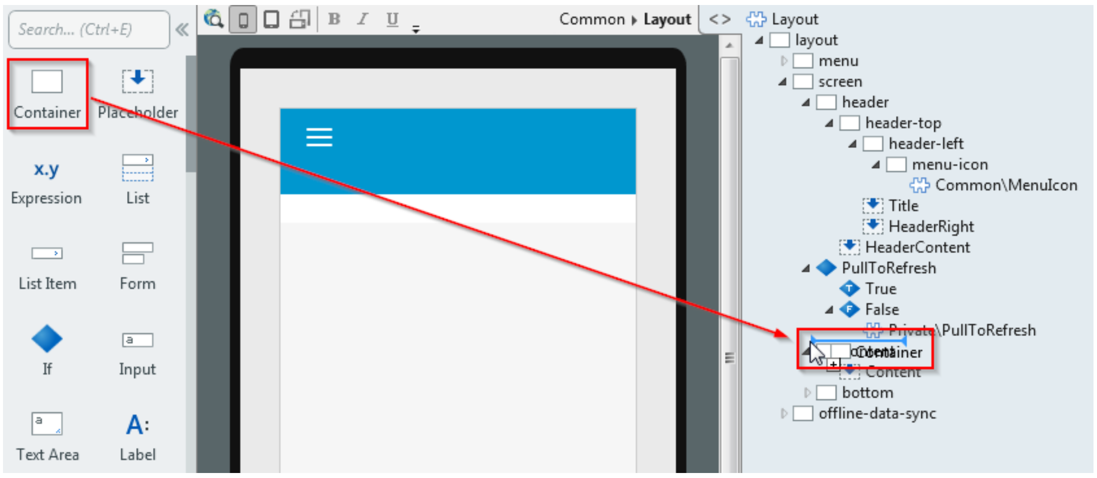
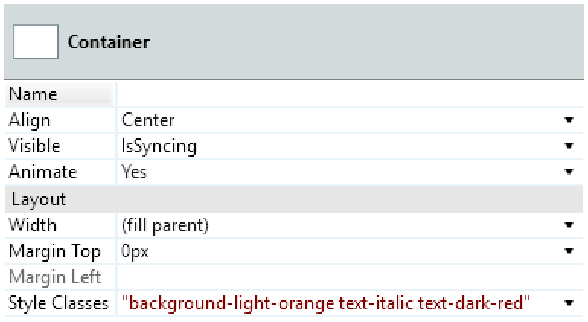


Figure 20. Drag and drop a Container to the Layout Block

e) Set the **Visible** property of the Container to the Local Variable **IsSyncing** and name the Container ‘Syncing’.

f) Set the **Align** property to 'Center', the **Margin Top** to '0px' and the **Style Classes** to `"background-light-orange text-italic text-dark-red"`.



Container

Name	
Align	Center
Visible	IsSyncing
Animate	Yes
Layout	
Width	(fill parent)
Margin Top	0px
Margin Left	
Style Classes	"background-light-orange text-italic text-dark-red"

Figure 21. Syncing Container properties

g) Drag an **Icon** Widget and drop it inside the Container, then choose the 'refresh' icon.

h) Set the **Size** property of the icon to 'Font size'.

i) Click to the right of the icon and type ' Syncing…'.

j) In the **Interface** tab, open the **ActionHandler_OnSyncStartTrigger** Client Action under the **Layout** Block.

k) Add an **Assign** statement just after the Start, and define the following assignment

```
IsSyncing = True
```

l) Open the **ActionHandler_OnSyncCompleteTrigger** Client Action, also under the **Layout** Block.

m) Add an **Assign** statement just after the Start, and define the following assignment

```
IsSyncing = False
```

n) Open the **ActionHandler_OnSyncErrorTrigger** Client Action and add an **Assign** statement as the one before.

2. Publish and test the syncing feedback container.

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

b) Navigate to the **Data Management** Screen, then click the **Sync to Local Storage** Button.

c) Notice that for a few moments the 'Syncing…' Container appears and then disappears.

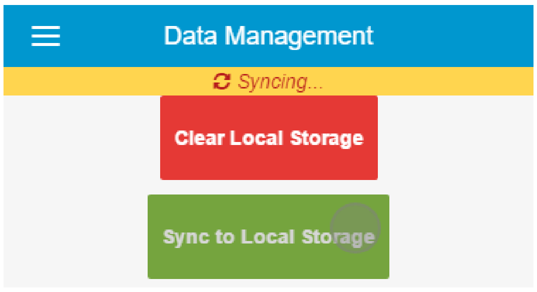


Figure 22. Syncing Container

End of Lab

In this exercise, you started by creating the Local Entities based on the existing Database Entities. You modified the **ToDos** and **ToDoDetail** Screens, so that they fetch data from Local Storage instead of using the Database Entities. Then, you edited some of the already existing Client Actions, so that the local storage data is also updated.

In the end, you have implemented the logic that allows the synchronization of data from the Database to Local Storage, and provide feedback to the user while the synchronization is running. Then, you published your application to the server and then tested the synchronization of data to Local Storage.

List of Figures

Here is the list of screenshots and pictures used in this exercise.