

List and Detail Screens

Exercise - How to

Outline	1
How to	2
Movies Screen	2
MovieDetail Screen	9
Linking Both Screens	21
People Screen	24
PersonDetail Screen	25
Linking the Screens	31

Outline

In this exercise, we will start building the UI of our application. We will build four Screens, two List Screens to list the movies and the people in the database, and two Detail Screens to allow creating and editing new movies and new people.

- Movies Screen
 - a. Screen to display the list of movies in the Movie Entity.
 - b. The list of movies should be displayed in a tabular layout, with the title, year, plot summary and the gross amounts.
- MovieDetail Screen
 - a. Screen to display the details of a particular movie, identified by its id, in a Form.
 - b. Make sure that the Screen will allow creating new movies, as well as editing existing ones
 - c. Build the logic to create/update a movie in the database.
 - d. Add a Link between the MovieDetail Screen and the Movies Screen to go back to the list, when needed. In the Movies Screen, add a Link between every movie to the respective MovieDetail Screen, to display its detailed data, and a Link at the top of the Screen to allow creating a new movie.
- People Screen
 - a. Screen to display the list of people in the Person Entity.
 - b. The list of people should be displayed in a tabular layout, with the name, surname and date of birth.
- PeopleDetail Screen
 - a. Screen to display the details of a particular person, identified by its id, in a Form.
 - b. Make sure that the Screen will allow creating new people, as well as editing existing ones
 - c. Create the logic to create/update a person in the database.
 - d. Add a Link between the PersonDetail Screen and the People Screen to go back to the list, when needed. In the People Screen, add a Link between every person to

the respective PersonDetail Screen, to display its detailed data, and a Link at the top of the Screen to allow creating a new person.

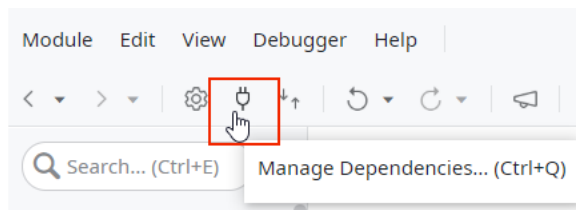
How to

In this section, we'll describe, step by step, exercise 3 - *List and Detail Screens*.

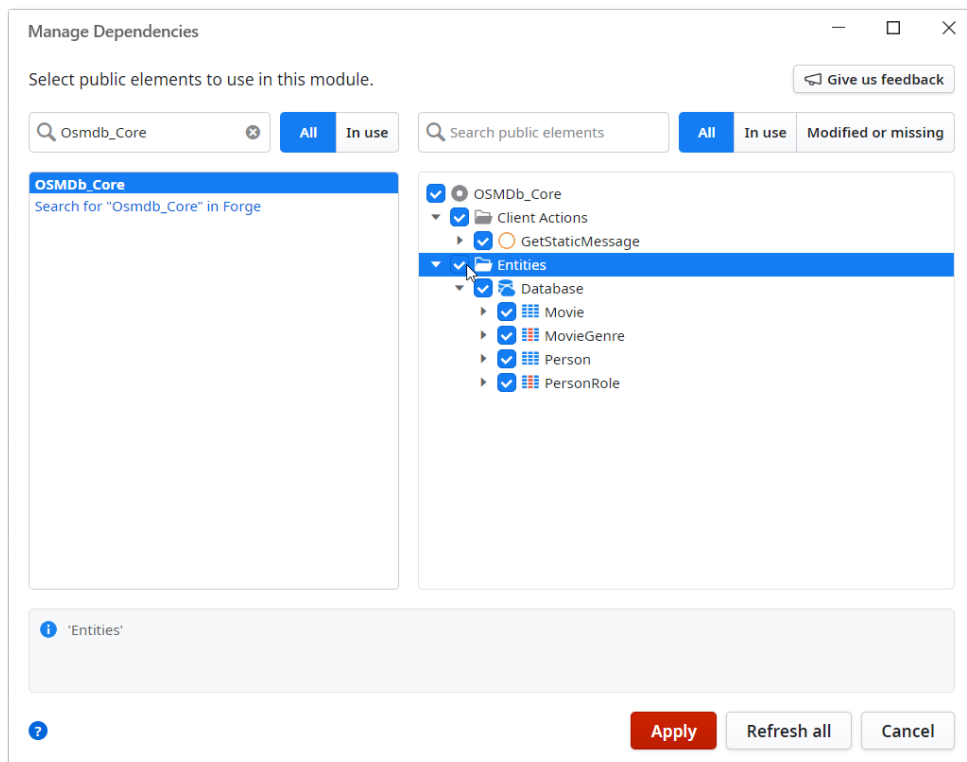
Movies Screen

We will start by creating the Movies Screen.

1. In the OSMDb module, reference the Entities and Static Entities created in the previous exercise, using the Manage Dependencies dialog.
 - a. Switch to the OSMDb module and open the **Manage Dependencies** dialog.



- b. Select your OSMDb_Core and select the four Entities: Movies, People, MovieGenre and PersonRole.

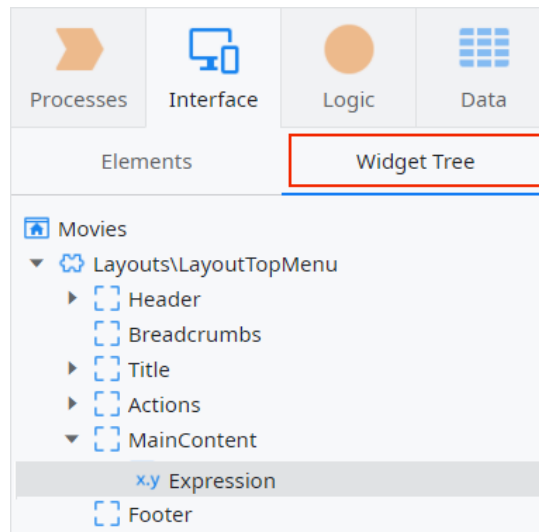


NOTE: If the Entities are not visible when trying to reference them from the OSMDb module, make sure the OSMDb_Core module is published, or that the Entities are set to Public. If they are not, we need to change them and publish the module again.

- c. Click **Apply** to close the dialog.
2. Change the HomeScreen to become the Movies Screen. Delete the Expression on the Screen and create an Aggregate to fetch all the movies from the database. These movies will be later displayed on the Screen.
 - a. Rename the **HomeScreen** to *Movies*.

The screenshot displays the OutSystems Studio interface for configuring a screen. On the left, a tree view shows the project structure: 'MainFlow' is expanded, and 'Movies' is selected. Below the tree, the 'Movies Screen' configuration panel is visible. It includes fields for 'Name' (set to 'Movies'), 'Description' (empty), 'Title' (empty), and 'Public' (set to 'No'). Under the 'Roles' section, both 'Anonymous' and 'Registered' roles are checked with blue checkmarks.

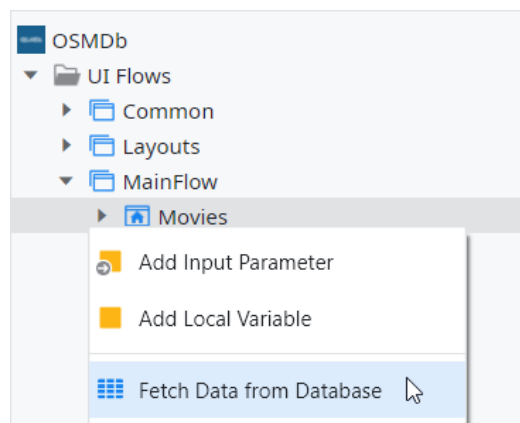
- b. Click on the **Widget Tree** button to open the structure of the Screen. Expand the **Content** to find the Expression added in the previous exercise. Right click on it and delete it.



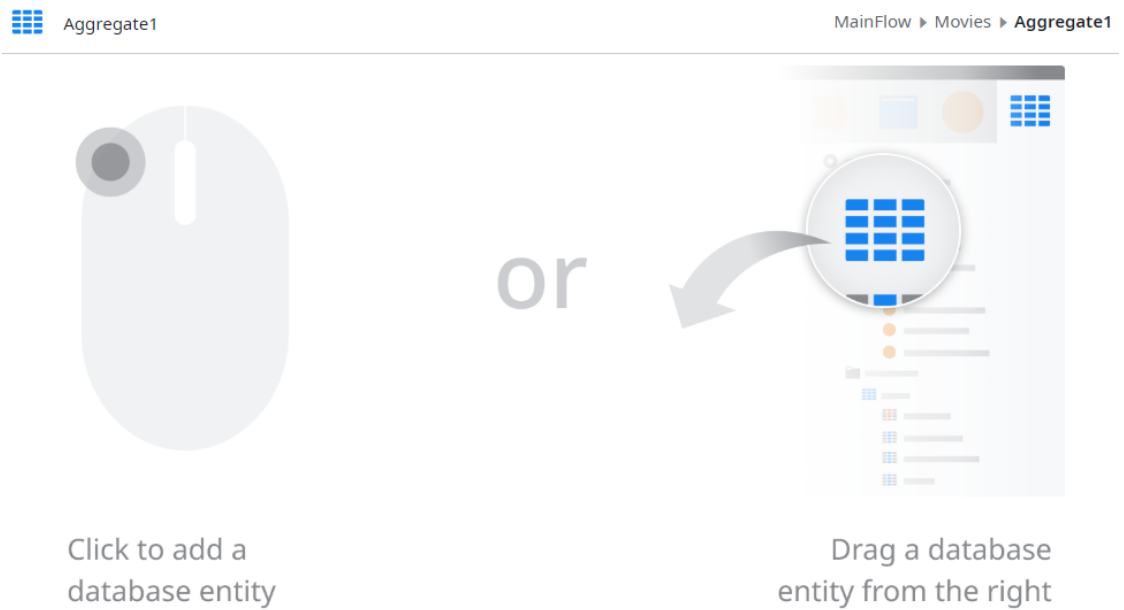
NOTE: The Widget Tree is a very helpful tool to build the UI in OutSystems. It shows an hierarchical structure of a Screen, making it easy to find every existing Screen element (and all the other elements inside it if that is the case).

It is also useful to add new Screen elements to a Screen, by helping the developer placing it exactly where it's needed, as we will see in future exercises.

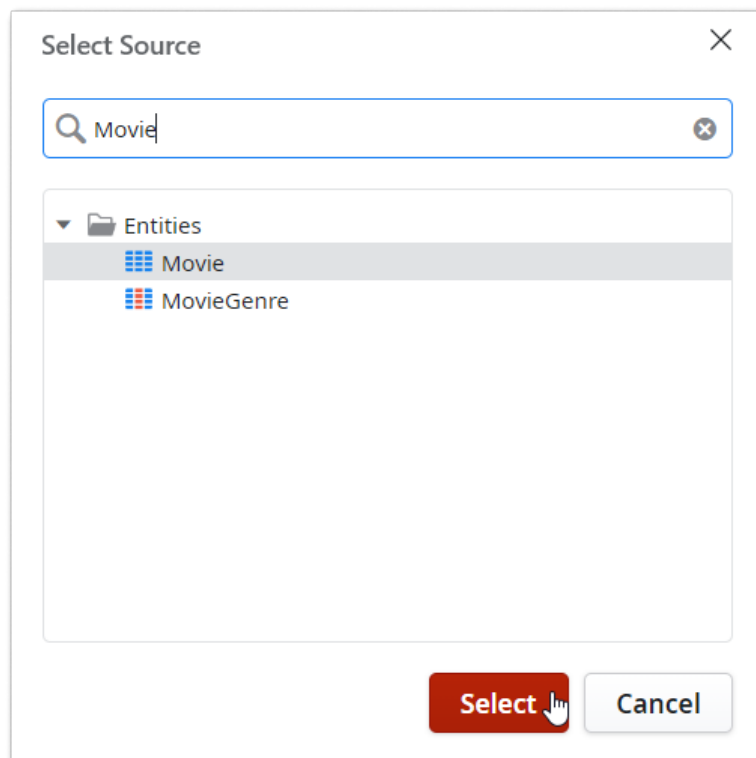
- c. Back in the Elements tab, Right click the **Movies** Screen and choose **Fetch Data from Database**



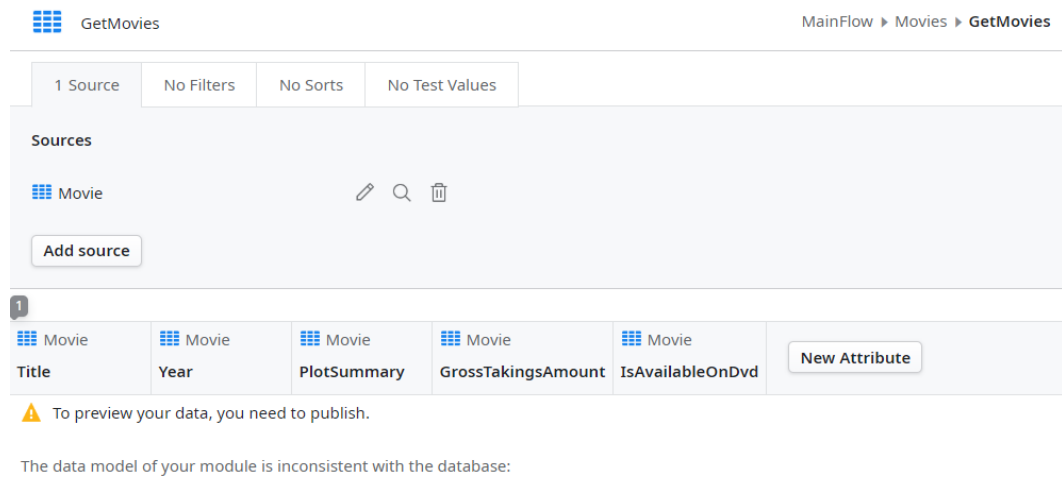
- d. Like the instructions on the Screen indicate, click on the main area to add a source to the Aggregate.



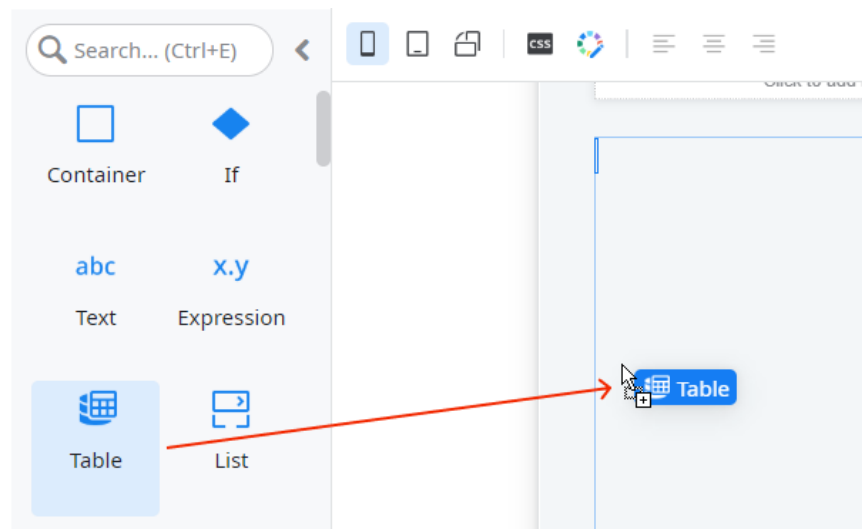
- e. Select the **Movie** Entity and click **Select**. The Aggregate is automatically renamed to *GetMovies*



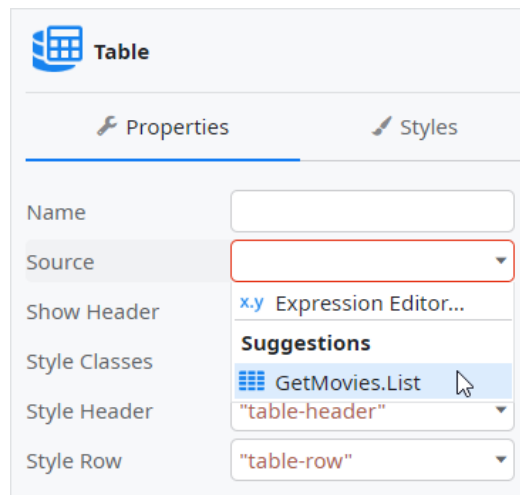
f. You should get something like the following screenshot



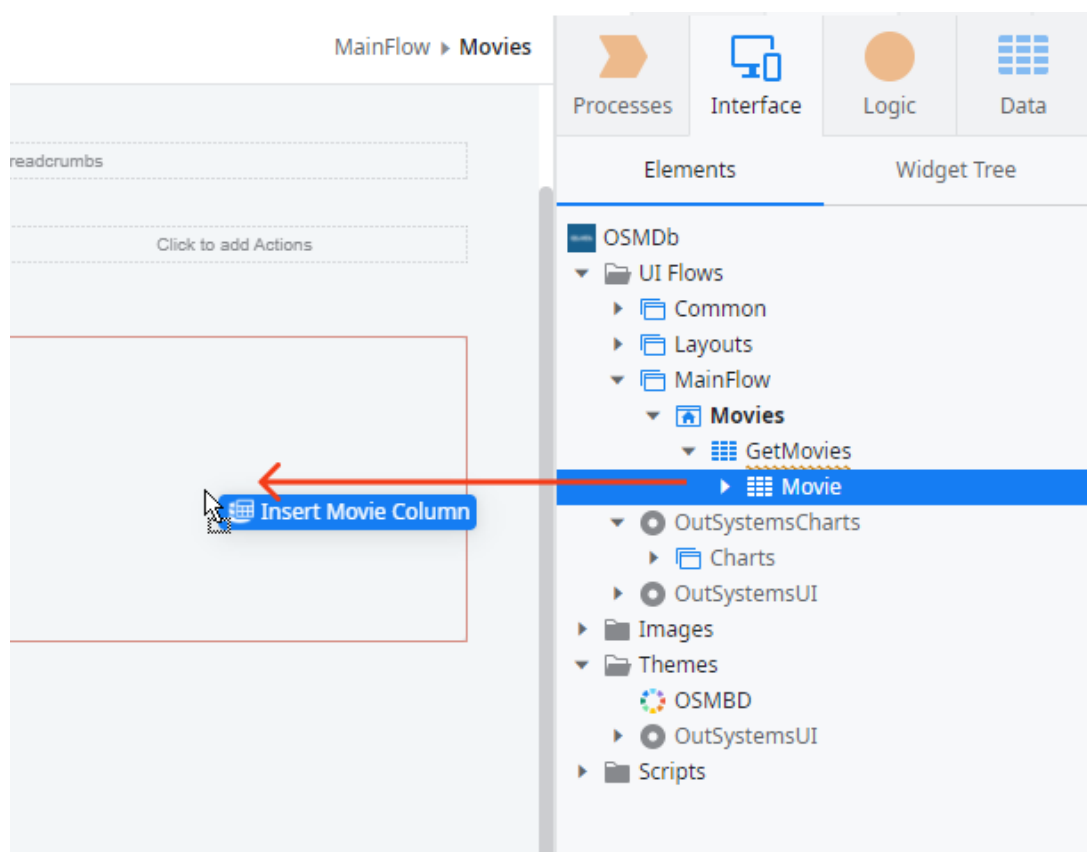
3. Define the UI of the Movies Screen by using a **Table** that will display all the Movies, with the **Title**, **Year**, **PlotSummary** and **GrossTakingsAmount** information. Also, make sure the Title of the Screen (in the UI) is set to *Movies*.
 - a. Double click the **Movies** Screen to open it.
 - b. Find the Table widget, then drag and drop it on the main content of the Screen.



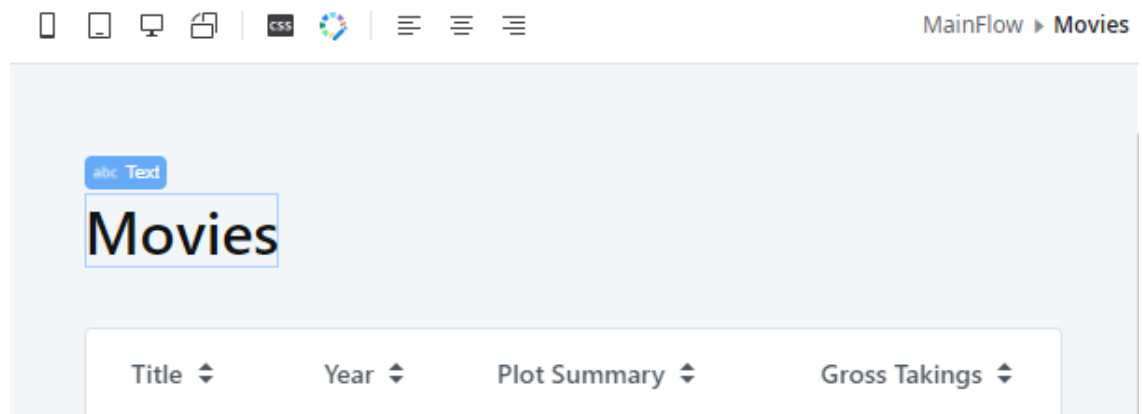
- c. Set the **Source** property of the Table to the output of the GetMovies Aggregate.



- d. Expand the GetMovies Aggregate and drag the Movie Entity to the Table.



- e. Set the Title of the Screen to be **Movies**



- f. **Publish** the application and open it in the browser. Make sure all the movies appear on the Screen.

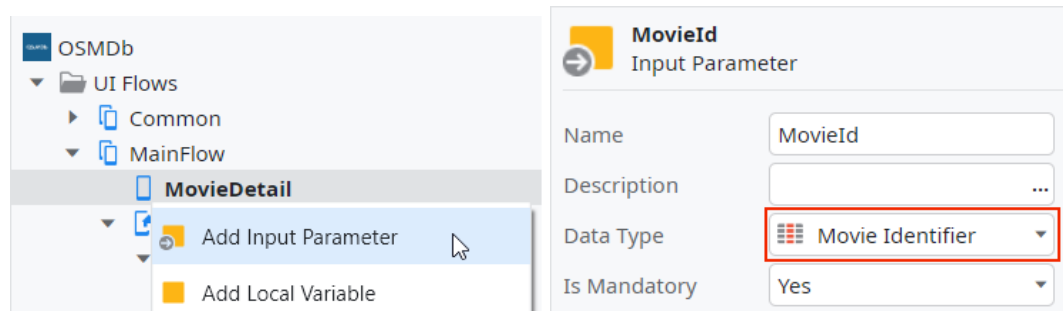


The screenshot shows the published application in a browser. The title 'Movies' is at the top. Below it is a table with four columns: 'Title', 'Year', 'Plot Summary', and 'Gross Takings'. The table contains five rows of movie data.

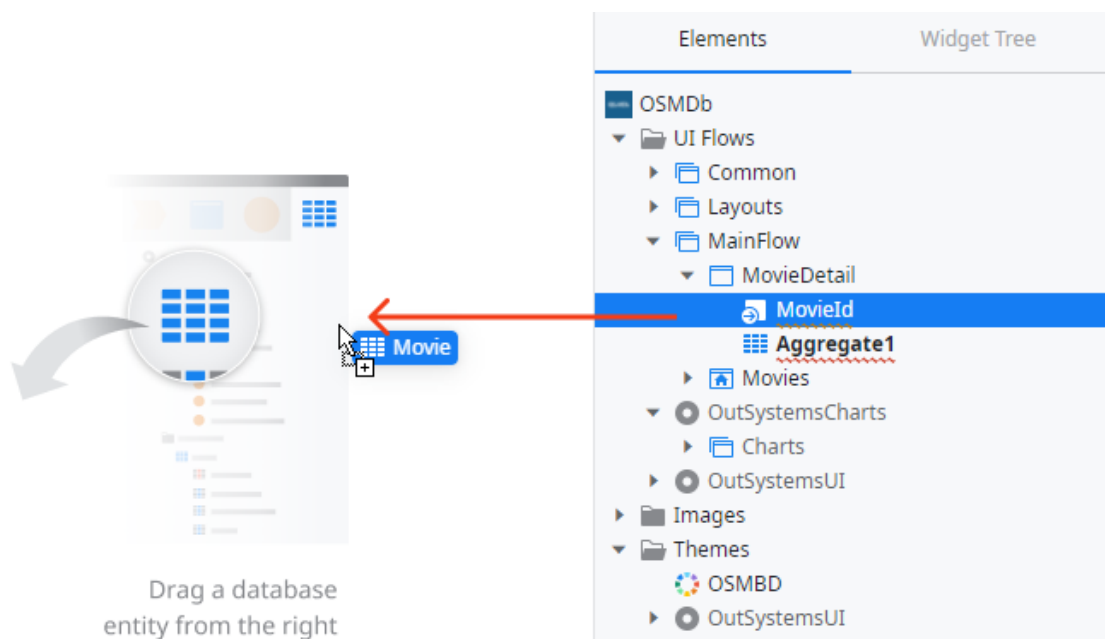
Title	Year	Plot Summary	Gross Takings
Star Wars: The Force Awakens	2015	Three decades after the defeat of the Galactic Empire, a new threat arises. The First Order attempts to rule the galaxy and only a ragtag group of heroes can stop them, along with the help of the Resistance.	\$815,843,529.00
Raiders of the Lost Ark	1981	Archaeologist and adventurer Indiana Jones is hired by the US government to find the Ark of the Covenant before the Nazis.	\$242,374,454.00
Schindler's List	1993	In Poland during World War II, Oskar Schindler gradually becomes concerned for his Jewish workforce after witnessing their persecution by the Nazis.	\$16,439,233.00
Avatar 2	2020	(unknown)	\$0.00
Indiana Jones and the Last Crusade	1989	When Dr. Henry Jones Sr. suddenly goes missing while pursuing the Holy Grail, eminent archaeologist Indiana Jones must follow in his father's footsteps and stop the Nazis.	\$197,171,806.00

MovieDetail Screen

1. Create a new Screen called *MovieDetail*, from an Empty template. This Screen should have an Input Parameter with the identifier of a movie, *MovieId*, and an Aggregate to fetch the movie with that specific Id.
 - a. Create a new Empty Screen, name it *MovieDetail* and set it to **Anonymous**
 - b. Add an Input Parameter called *MovieId*, with **Data Type** set to *Movie Identifier*



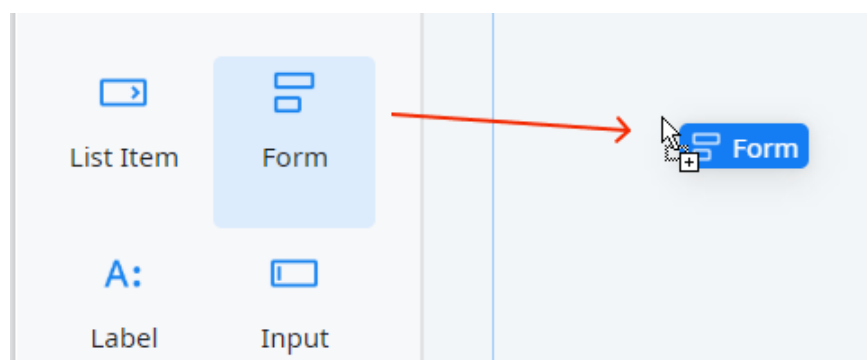
- c. Create a new Aggregate by right-clicking on the Screen and selecting the **Fetch Data from Database** option.
 - d. Drag the **MovieId** input parameter into the Aggregate



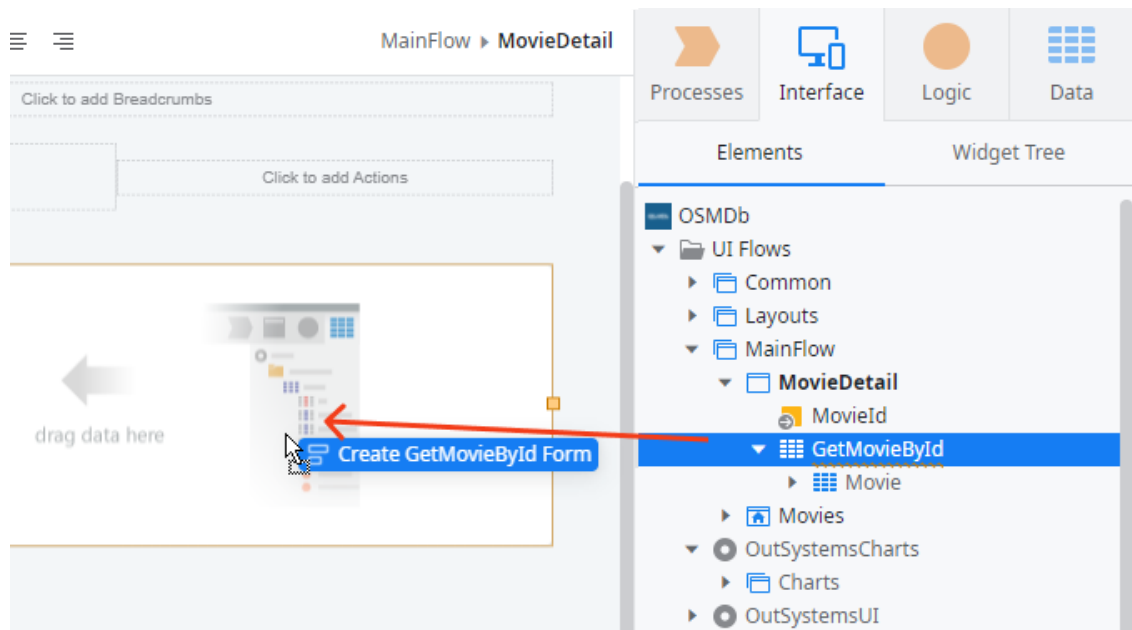
- e. Notice that the Aggregate was renamed to *GetMovieById*, the Movie Entity was added as source, and a new Filter using the MovieId input parameter was created.

The screenshot shows the configuration for the 'GetMovieById' Aggregate. The 'Name' field is set to 'GetMovieById'. The 'Description' field is empty. The 'Server Request Timeout' is set to '(Module Default Timeout)'. The 'Start Index' is set to 0. The 'Max. Records' is set to 50. The 'Fetch' is set to 'At start'. Under the 'Events' section, 'On After Fetch' is set to an empty dropdown. The 'Executed SQL' field contains 'SELECT TOP (32)'. Under the 'Sources' section, the 'Movie' entity is listed. Under the 'Filters' section, a filter is defined as 'Movie.Id = MovieId'. Under the 'Test Values' section, the 'MovieId' input parameter is listed.

2. Create a Form on the MovieDetail Screen with an input field for each attribute in the Movie Entity. Set the **Title** of the Screen to display *Create New Movie*, when the MovieId is *NullIdentifier()*, or the value of the *Title* attribute of the Movie Entity when it is not.
- a. Double click the **MovieDetail** Screen to open it.
- b. Drag a Form and drop it to the main content area of the Screen.



- c. Drag the **GetMovieById** Aggregate and drop it inside the Form



- d. All the Movie Entity attributes were added as input fields in the Form. The end-user will use them to submit information about movies.

Form1

Title

Year

Plot Summary

Gross Takings

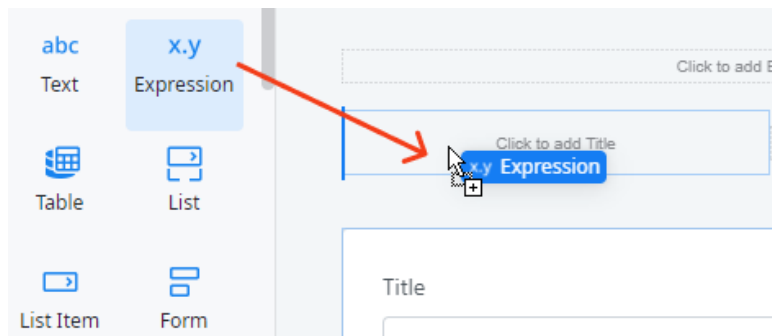
Is Available On DVD

☒

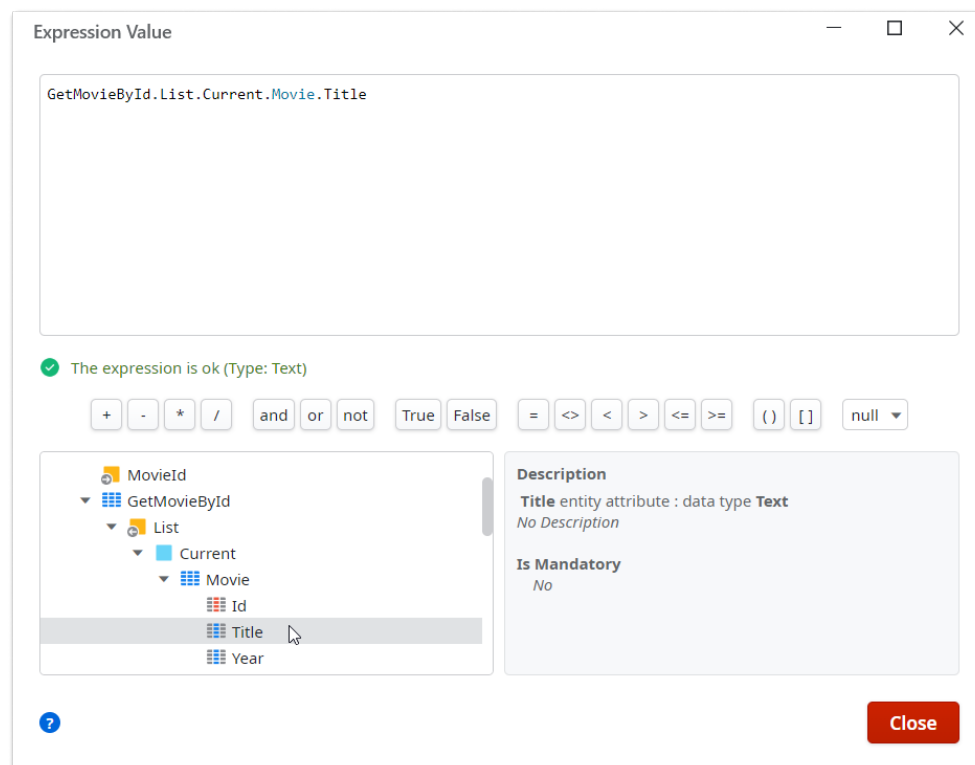
Save

NOTE: An error message will appear but we don't need to worry about it now.

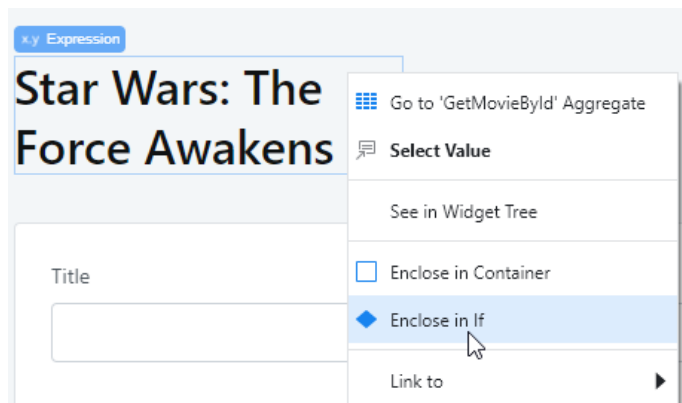
- e. Drag an Expression to the Title section of the Screen.



- f. Set the Expression to *GetMovieById.List.Current.Movie.Title*

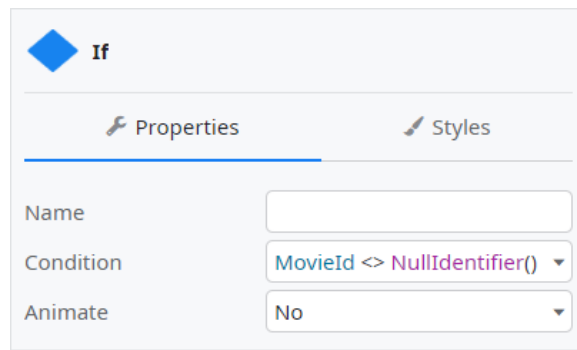


- g. Right-click on the Expression and select the option **Enclose in If**.



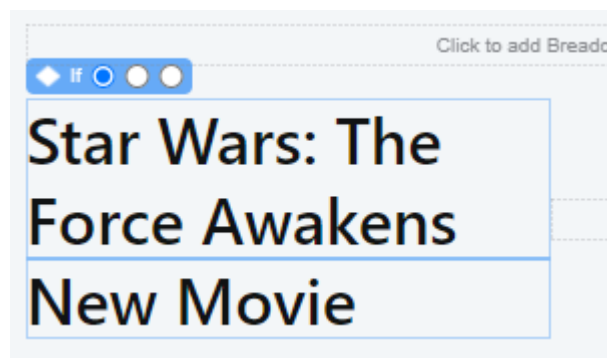
- h. Set the **Condition** of the If to

MovieId <> NullIdentifier()



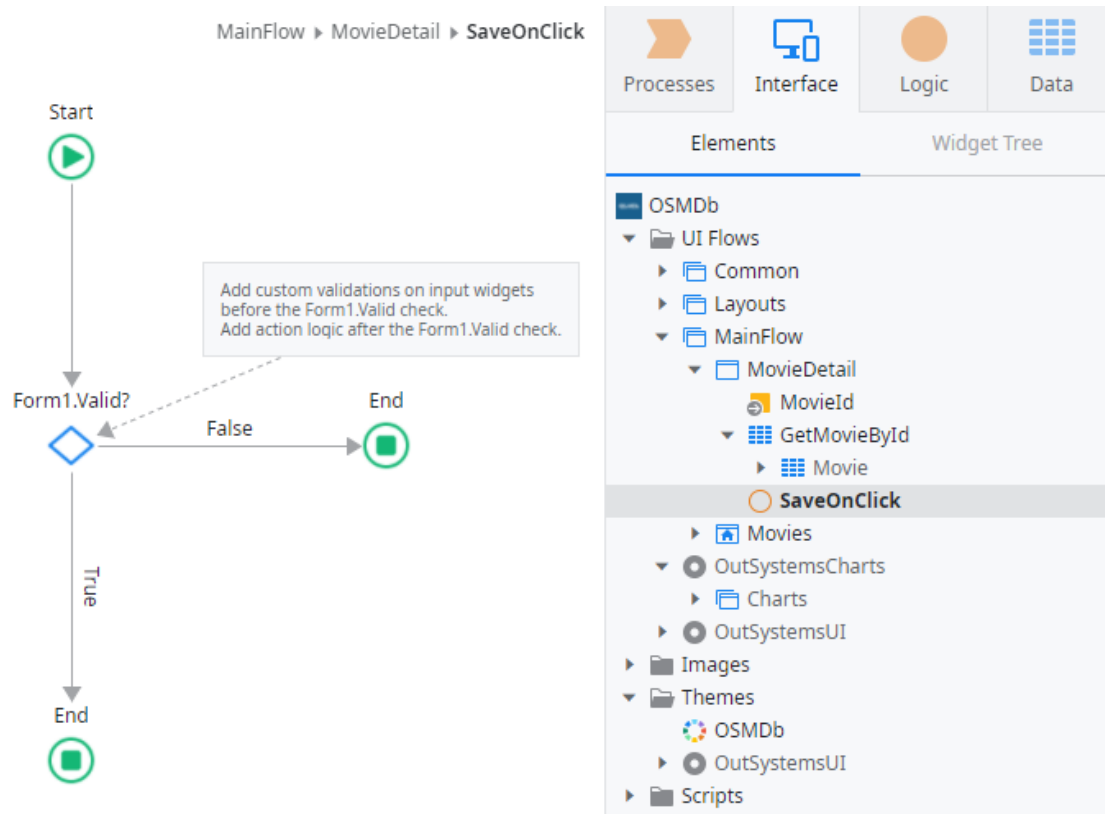
The screenshot shows the 'If' widget configuration panel. It has a 'Properties' tab selected. The 'Name' field is empty. The 'Condition' dropdown is set to 'MovieId <> NullIdentifier()'. The 'Animate' dropdown is set to 'No'.

- i. In the False branch of the If, enter *New Movie*

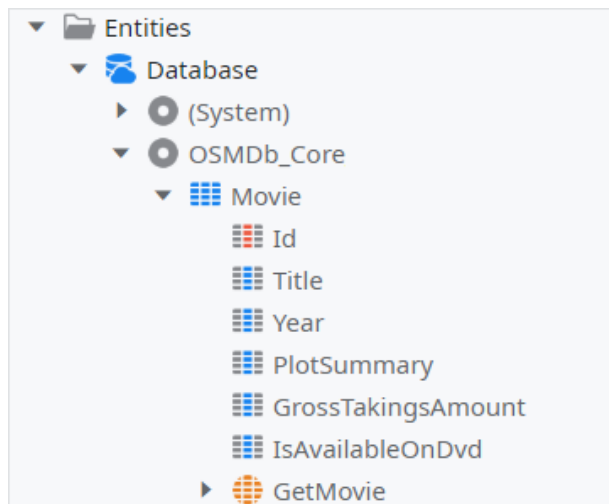


3. Define the logic to create/update a movie in the database and associate it with the Save button. The server-side logic should be implemented in the Core module, since the Entity is exposed as read-only.

- a. Double click the **Save** button to create a Screen Action

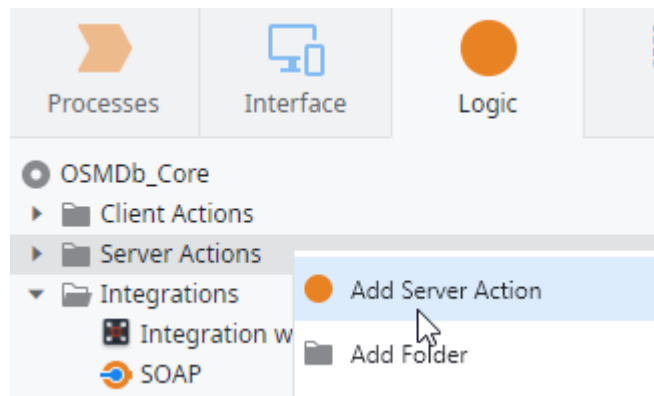


- b. Now we need to add the logic to create/update a movie in the database. Since the Entity is exposed as read-only, we only have access to the Get Entity Action in the OSMDb module. So, we need to create the necessary logic on the Core module and make it Public



- c. Switch to the OSMDb_Core module and open the Logic tab
- d.

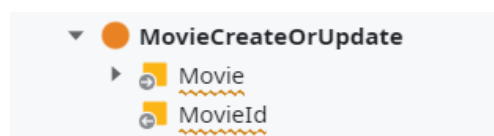
- e. Right-click on the Server Actions folder and click on **Add Server Action**



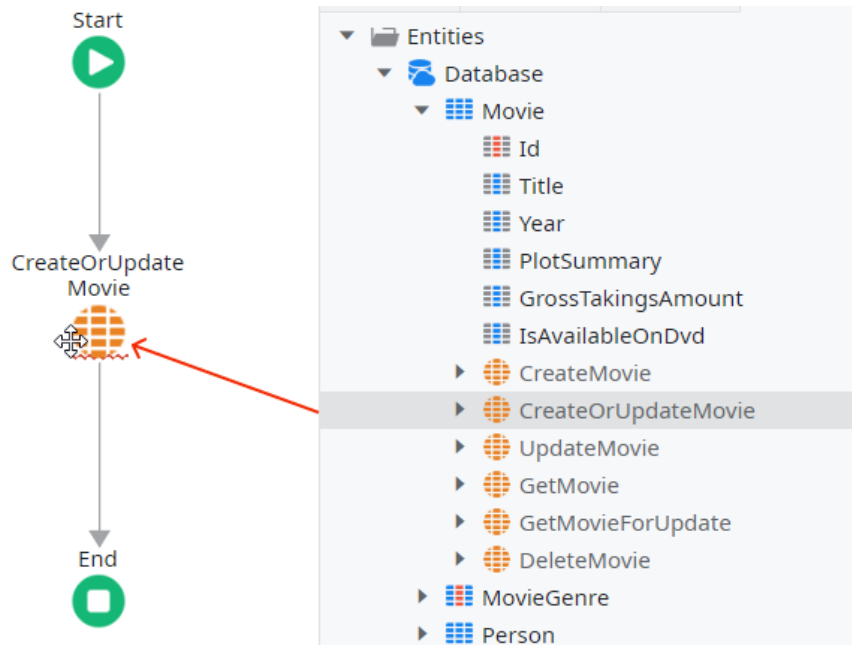
- f. Name it *MovieCreateOrUpdate* and set it as **Public**

A screenshot of the 'MovieCreateOrUpdate' Server Action configuration form. The form has a title bar with an orange circle icon and the text 'MovieCreateOrUpdate Server Action'. Below the title bar, there are five fields: 'Name' (containing 'MovieCreateOrUpdate'), 'Description' (empty with a three-dot icon), 'Public' (a dropdown menu set to 'Yes'), 'Function' (a dropdown menu set to 'No'), and 'Icon' (a dropdown menu showing an orange circle icon and the text 'Default Icon').

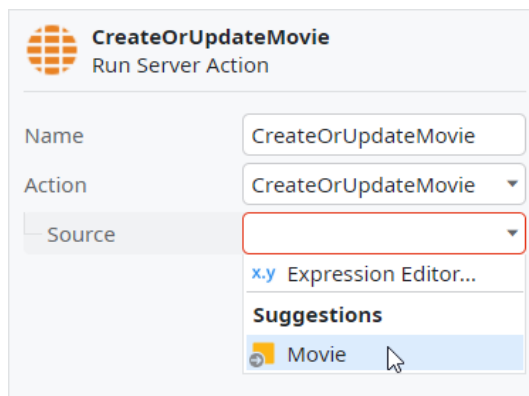
- g. Add one Input Parameter to the Action and name it *Movie*. Confirm the **Data Type** is set to *Movie*.
- h. Add an Output Parameter and name it *MovieId*. The **Data Type** should be *Movie Identifier*.



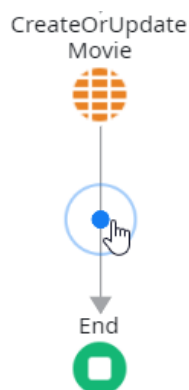
- i. Find the **CreateOrUpdateMovie** Entity Action in the tab Data, then drag it to the Action flow.



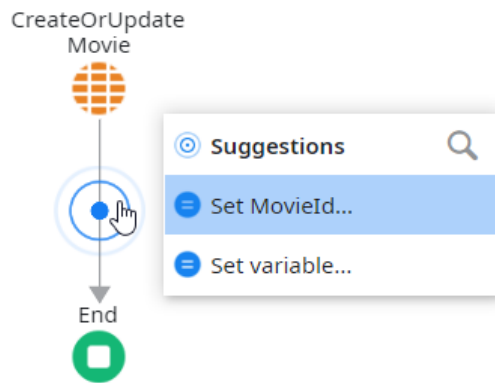
- j. Select the CreateOrUpdateMovie node dragged in the previous step and set its source to **Movie** (the input parameter of the Action).



- k. Mouse over the last arrow of the flow and click on the blue icon.

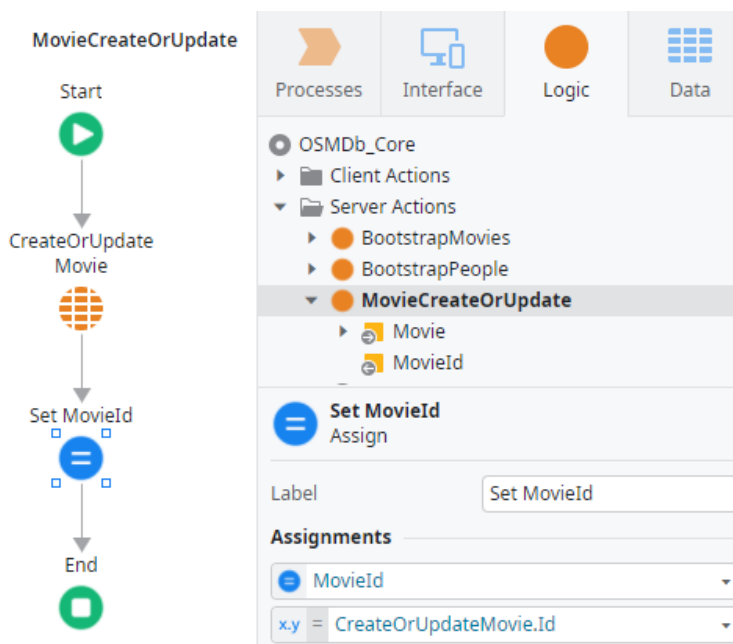


- l. Choose **Set MovieId...** and set the value field to **CreateOrUpdateMovie.Id**



If this option does not appear and instead you see a **Set MovieId to CreateOrUpdateMovie.Id** option, select that one.

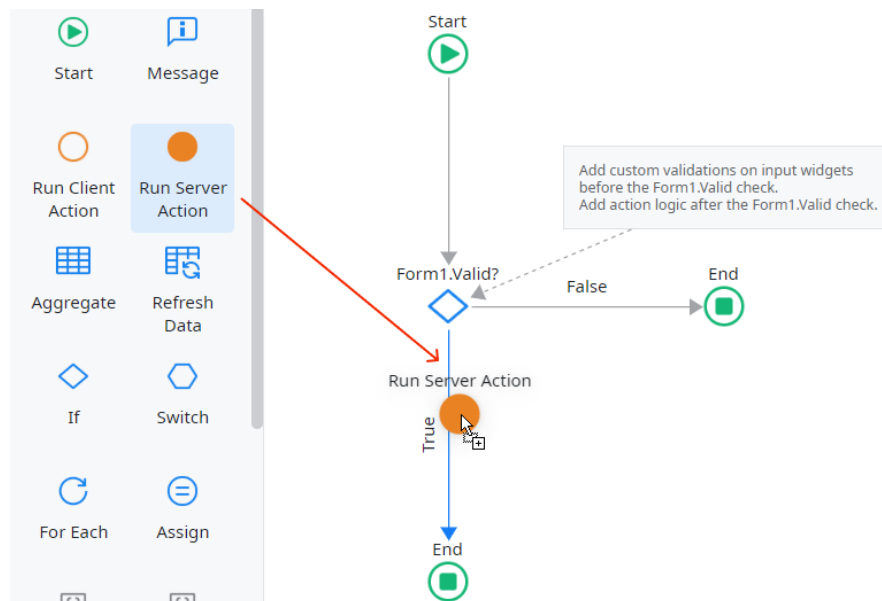
- m. Automatically, the output parameter **MovieId** was assigned to the output of the **CreateOrUpdateMovie Action**, which is the id of the movie created/updated. If you got the option **Set MovieId...** in the previous step, the Assign is created but we need to explicitly assign the *CreateOrUpdateMovie.Id* Value to the **MovieId**.



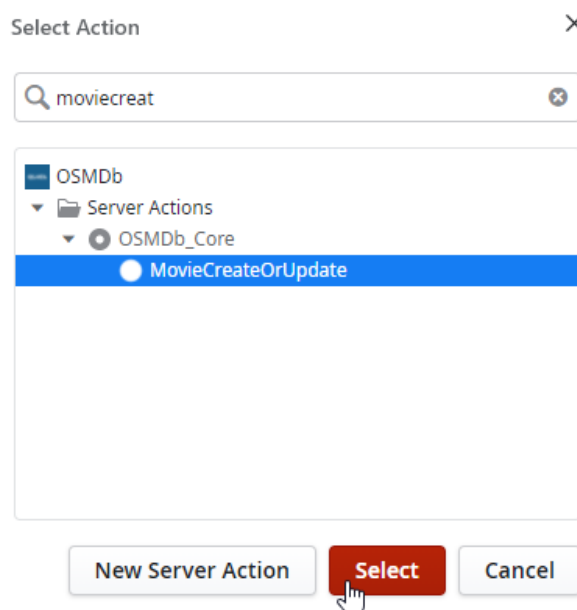
NOTE: This is an example of how we can leverage AI in Service Studio, with some recommendations of next steps in an Action flow. In this case, since the Action has an output parameter and the CreateOrUpdateMovie Action returns a value with

the same data type, Service Studio suggested the Assign. This varies from application to application and to the patterns most used by developers.

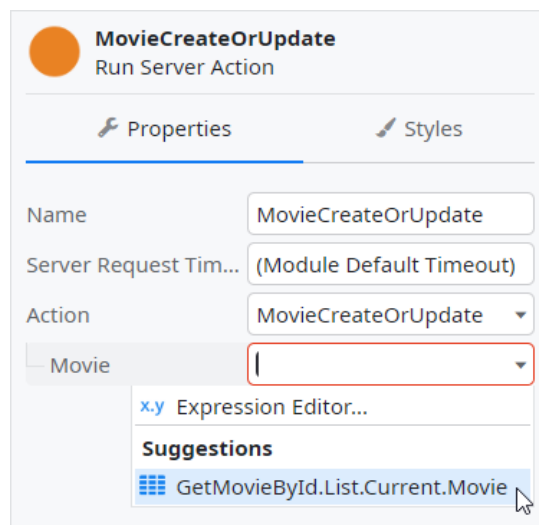
- n. Publish the **OSMDB_Core** Module.
- o. Go back to the OSMDb module and use the **Manage dependencies** to reference the Server Action **MovieCreateOrUpdate** from your core module.
- p. Open the **SaveOnClick** Screen Action of the MovieDetail Screen
- q. Drag a **Run Server Action** node to the flow and drop it below the If



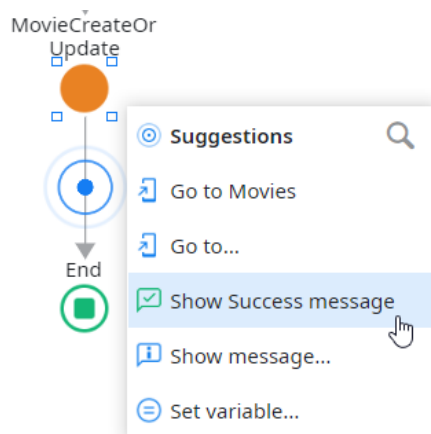
- r. In the new dialog, select the **MovieCreateOrUpdate** and click OK



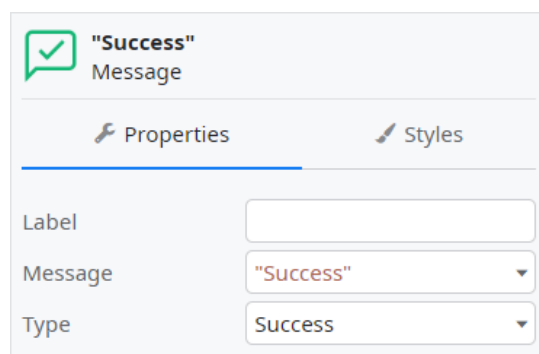
- s. In the properties of the Action, set the Movie input parameter of the Action to be *GetMovieById.List.Current.Movie*. Since this is the record returned by the Aggregate, that is being edited in the Form, we can use it to pass the movie to be created/updated to the MovieCreateOrUpdate Action.



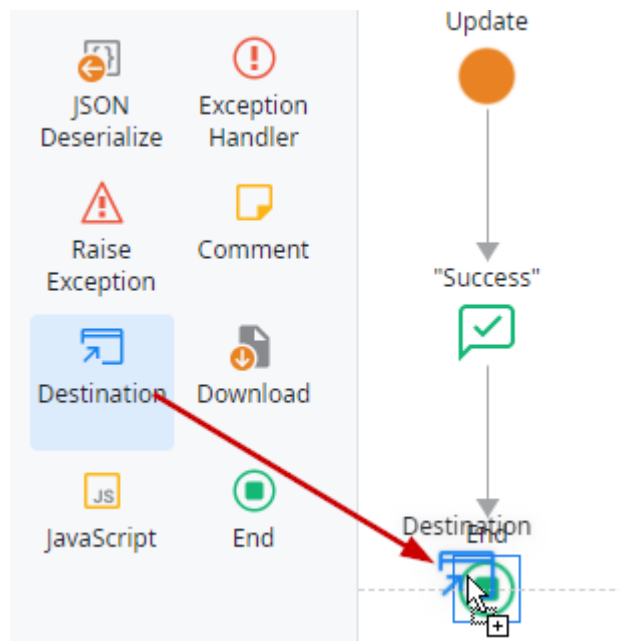
- t. Use the blue icon for suggestions to create a new Success message



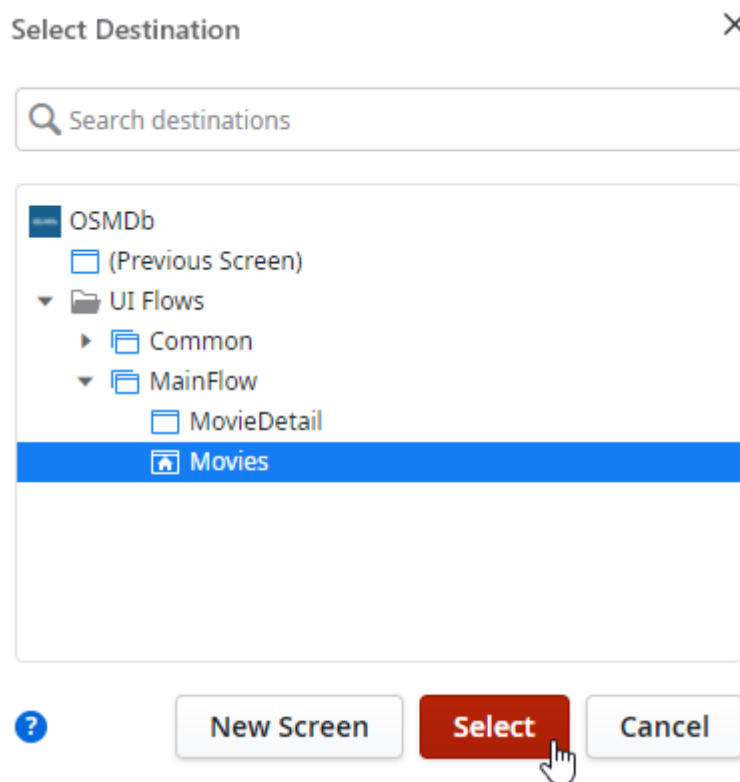
- u. Set the **Message** to something like "Success!"



- v. Drag a **Destination** node on top of the End node



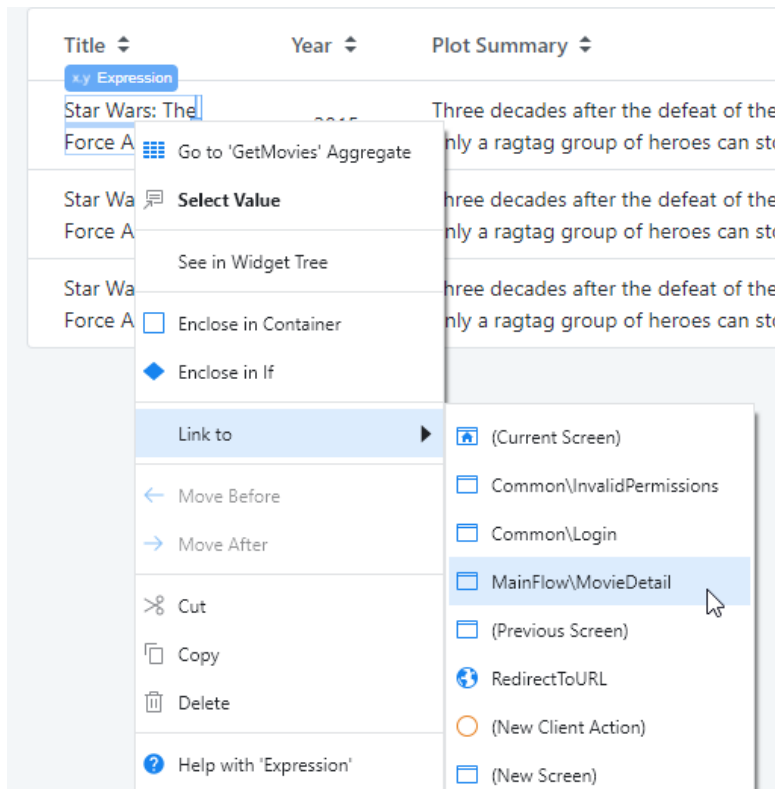
- w. Select the **Movies** Screen and click **OK**



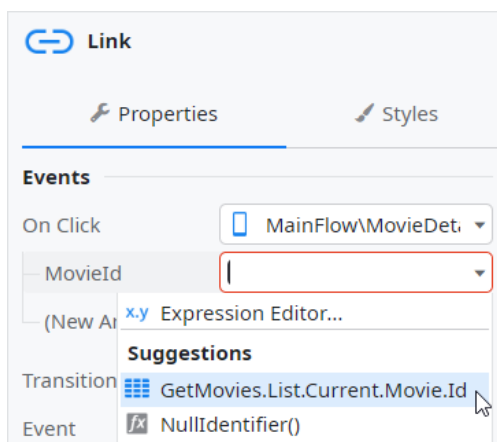
Linking Both Screens

Now that we have both Screens for movies created, we want to make sure we define a proper navigation between them. The Movies Screen should have two links to the MovieDetail Screen, one to create new movies and one per movie in the table to allow editing information about the movie. In the MovieDetail Screen, we should have a link that allows returning to the Movies Screen.

1. In the Interface Tab, double click the Movies Screen, Right click the **title** expression -> Link To -> **MainFlow\MovieDetail**

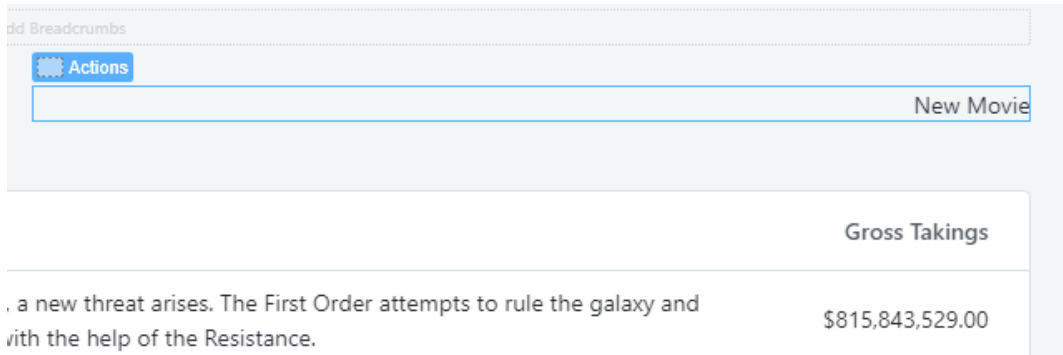


2. In the input parameter of the Link's **On Click** Destination select *GetMovies.List.Current.Movie.Id*.

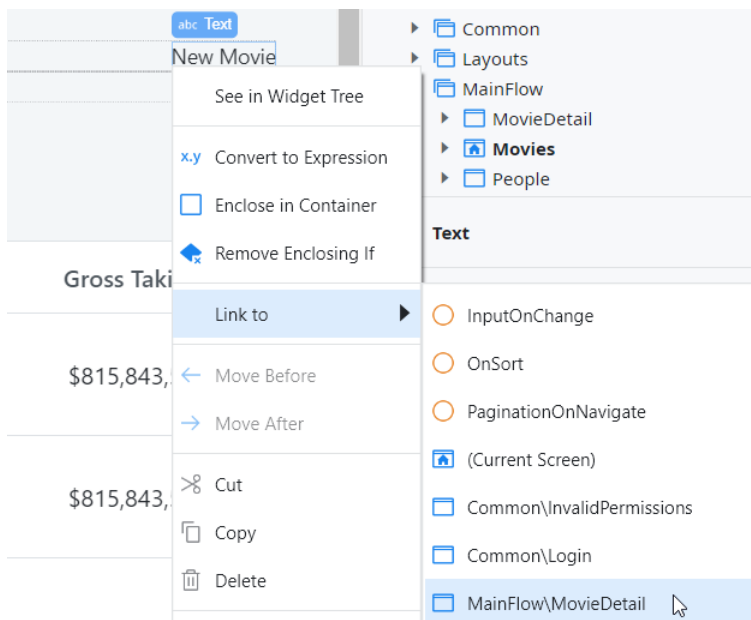


NOTE: This guarantees that the Id of the movie selected is passed to the MovieDetail Screen. In this case, the MovieDetail will display the details of this movie and the end-user will be able to edit it.

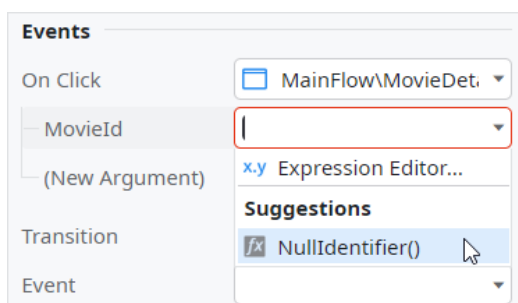
3. On the top right of the Screen, in the Actions area, type *New Movie*



4. Right click the New Movie text and select **Link to -> MainFlow\MovieDetail**

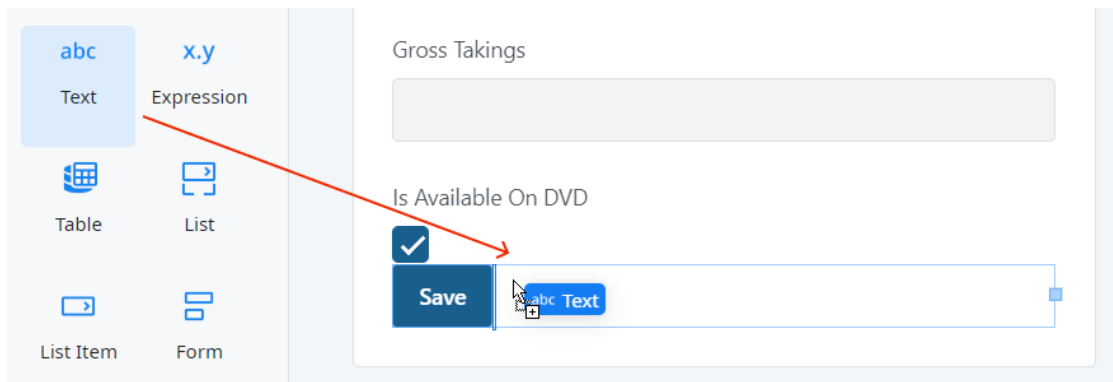


5. Use **NullIdentifier()** for the input parameter



NOTE: In this case, since it's a new movie, it does not have an Id yet, so we pass the `NullIdentifier()` value.

6. In the MovieDetail Screen, drag a Text widget and drop it next to the Button

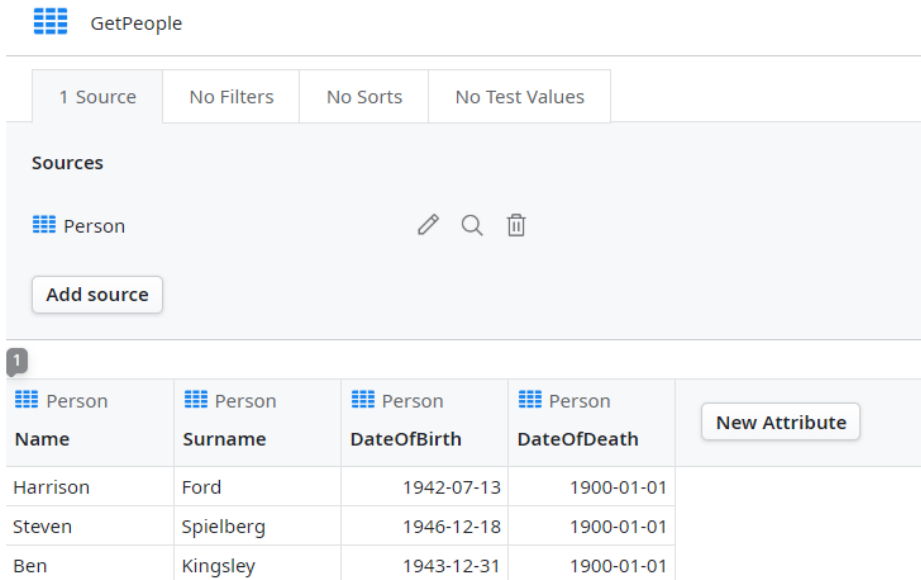


7. Set the Text to be *Back to Movies*
8. Right-click the Text and create a Link to the Movies Screen.
9. **Publish** the module and test it in the Browser!

People Screen

Now, we will create the Screens for People, just like we did for the movies. We will start by creating the People Screen based on an Empty Screen that lists all the people in the database.

1. Create an **Empty** Screen, name it *People* and set it to **Anonymous**
2. Add an Aggregate to the Screen, with the Person Entity as Source.



1 Source No Filters No Sorts No Test Values

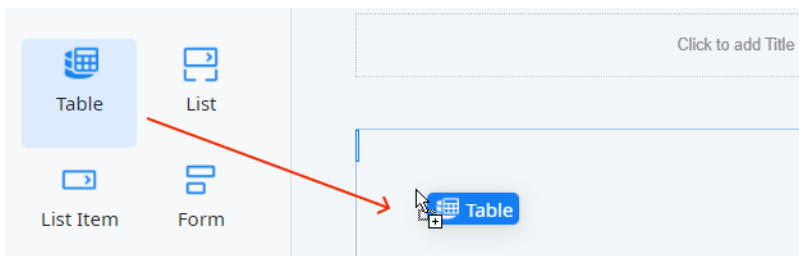
Sources

Person

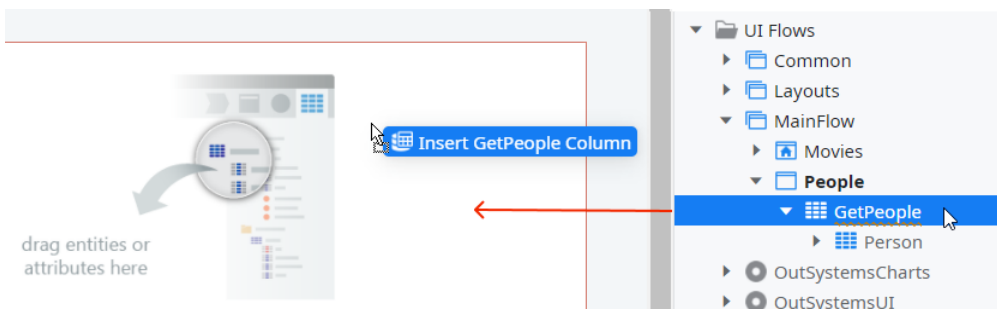
Add source

Person Name	Person Surname	Person DateOfBirth	Person DateOfDeath	New Attribute
Harrison	Ford	1942-07-13	1900-01-01	
Steven	Spielberg	1946-12-18	1900-01-01	
Ben	Kingsley	1943-12-31	1900-01-01	

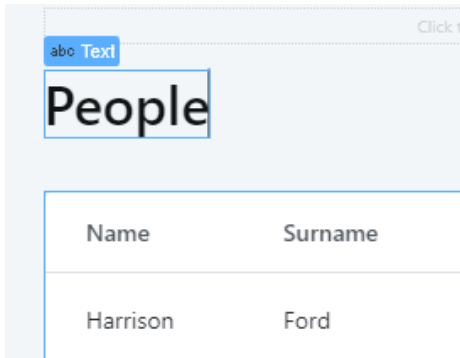
3. Drag a Table to the People Screen.



4. Drag the **GetPeople** Aggregate to the Table



5. Set the Title of the Screen to be *People*

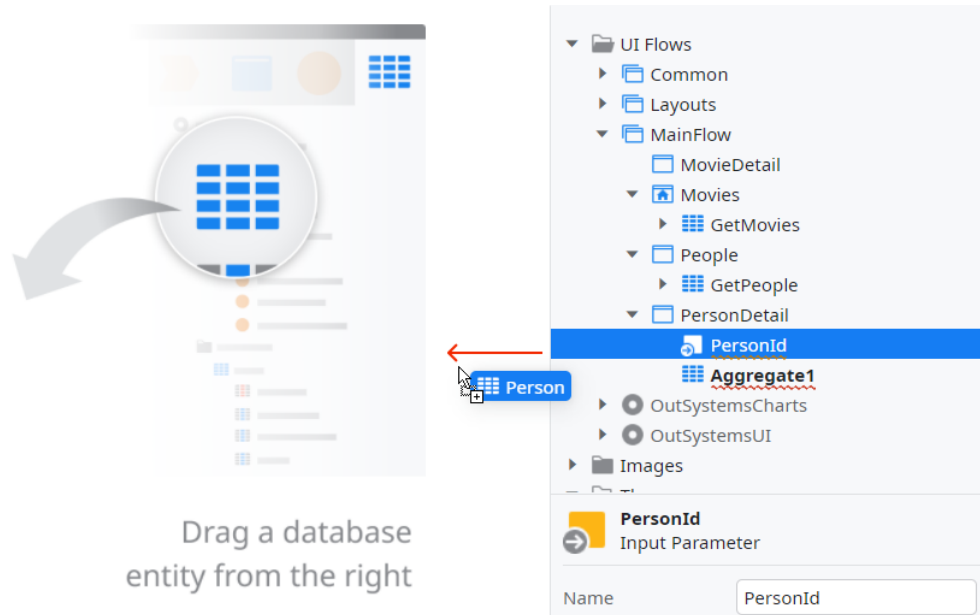


PersonDetail Screen

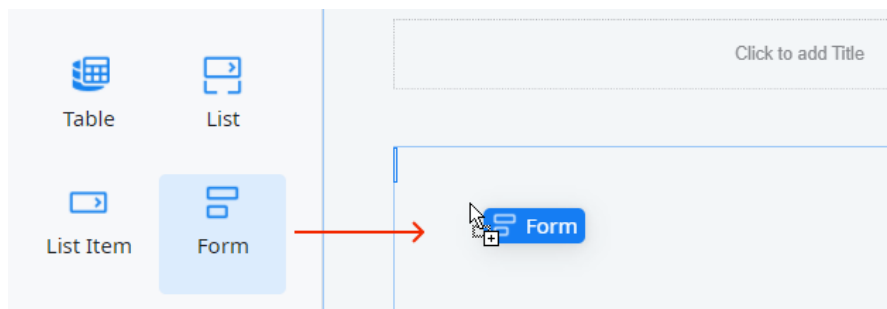
The PersonDetail Screen should have a Form with input fields for all the attributes in the Person Entity. Just like in the MovieDetail, the Screen should expect the Id of the Person as input. That input should be used to fetch the information about the Person. Also, the logic to create/update the Person in the database should be triggered by a Save button.

1. Create an Empty Screen, name it *PersonDetail* and set it to **Anonymous**. The Screen should have a *PersonId* input parameter and an Aggregate to fetch the information about the Person.
 - a. Create a new **Empty** Screen and set its name to *PersonDetail*. Set its access to **Anonymous**.
 - b. Add an input parameter called *PersonId*. Make sure its **Data Type** is set to *Person Identifier*.

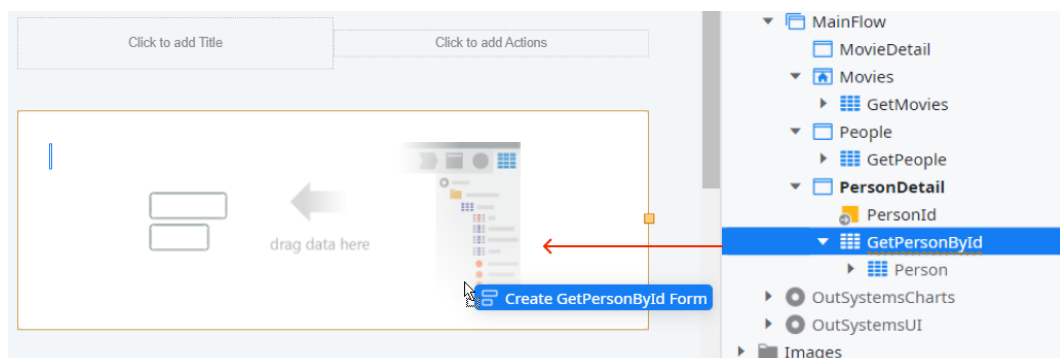
- c. Create a new Aggregate and use the **PersonId** Input Parameter to just return the Person whose Id matches the value in the input parameter. The Aggregate should change the name to *GetPersonById*.



- d. Drag a Form to the Screen



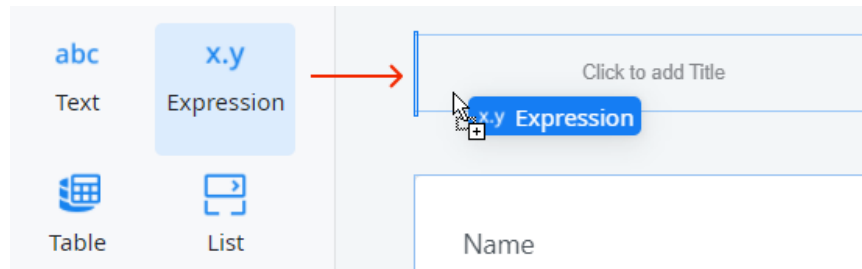
- e. Drag the **GetPersonById** Aggregate and drop it on the Form



- f. Drag an Expression and drop it on the Title of the Screen. Set its value to:

GetPersonById.List.Current.Person.Name + " " +

GetPersonById.List.Current.Person.Surname



- g. Enclose the Expression in an If and set its Condition to:

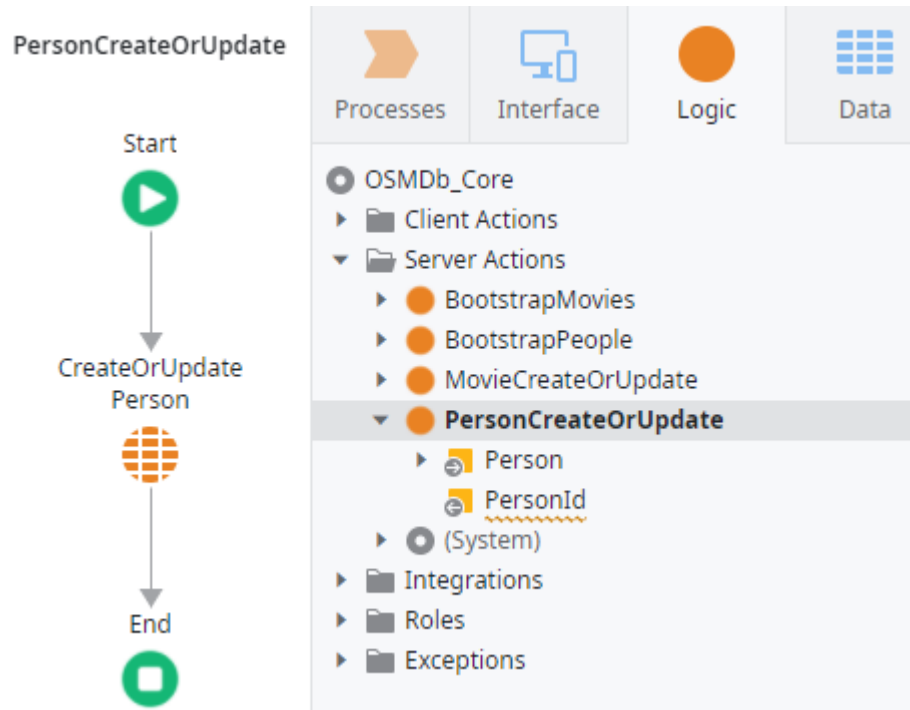
PersonId <> NullIdentifier()

- h. In the False branch, enter *New Person*



2. Create the logic to create/update a Person in the database and make sure it is triggered when clicking on the Save button.
- a. Double-click on the Save button to create the *SaveOnClick* Action.
 - b. Go back to the Core module and create a new **Server Action**. Name it *PersonCreateOrUpdate* and set it to **Public**.
 - c. Add one Input Parameter to the Action and name it *Person*. Make sure its **Data Type** is set to *Person*.
 - d. Add an Output Parameter to the Action and name it *PersonId*. Make sure its **Data Type** is set to *Person Identifier*.

- e. Drag the **CreateOrUpdatePerson** Entity Action to the Action flow and set its **Source** to the Person input parameter.

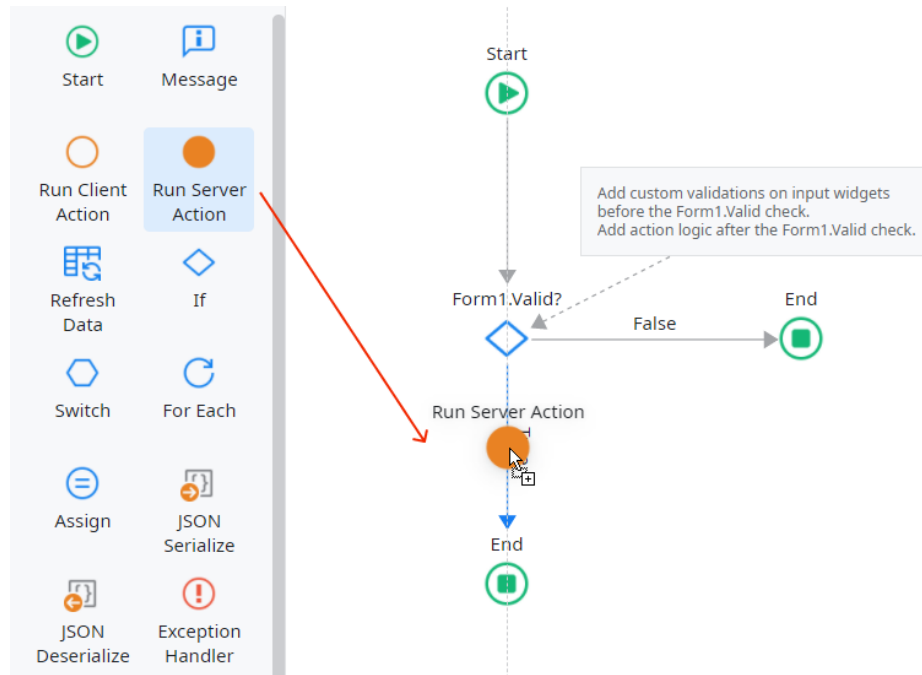


- f. Drag an Assign and drop it after the CreateOrUpdatePerson Action. This Assign should set the Output Parameter with the return value from the Entity Action

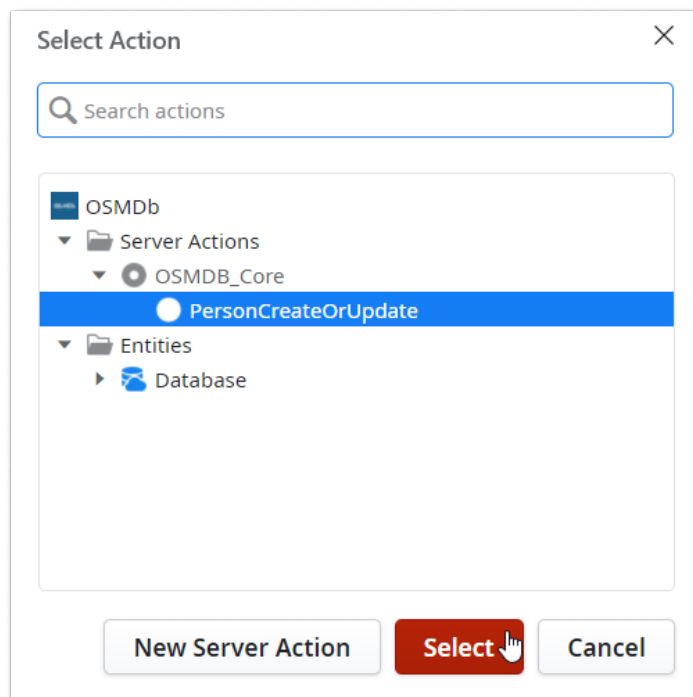
The screenshot shows the configuration for an 'Assign' action named 'Set Person Id'. The 'Label' field is set to 'Set Person Id'. Under the 'Assignments' section, there is a single assignment: 'PersonId' (selected from a dropdown) is assigned to 'CreateOrUpdatePerson.Id' (selected from a dropdown). The assignment is represented as 'x.y = CreateOrUpdatePerson.Id'.

- g. Publish the Core module
- h. Switch back to the OSMDB module and use the **Manage Dependencies** dialog to reference the newly created PersonCreateOrUpdate Action.

- i. In the **SaveOnClick** Action, under the PeopleDetail Screen, drag a **Run Server Action** node and drop it after the If.



- j. In the new dialog, choose the **PersonCreateOrUpdate**.

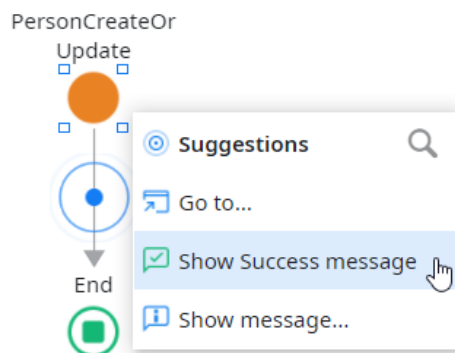


- k. Set its Person Input Parameter to the suggestion provided by Service Studio, which is the Person record being fetched from the database

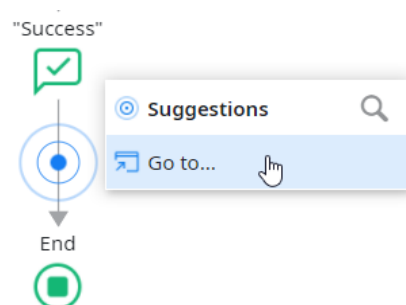
The screenshot shows the configuration for the 'PersonCreateOrUpdate' Run Server Action. The fields are as follows:

Field	Value
Name	PersonCreateOrUpdate
Server Request Time...	(Module Default Timeout)
Action	PersonCreateOrUpdate
Person	GetPersonById.List.Current.Person

- l. Mouse over again the last arrow, click the blue icon and choose **Show Success message**



- m. Define a **Message** indicating that the operation was successful.
- n. For the last time, mouse over the last arrow and now select **Go to...**

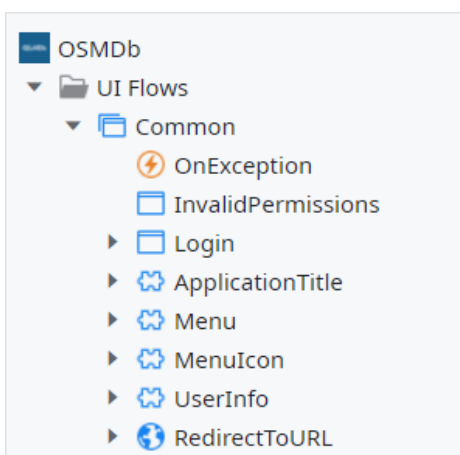


- o. Select the **People** Screen and click **OK** to create a navigation to that Screen.

Linking the Screens

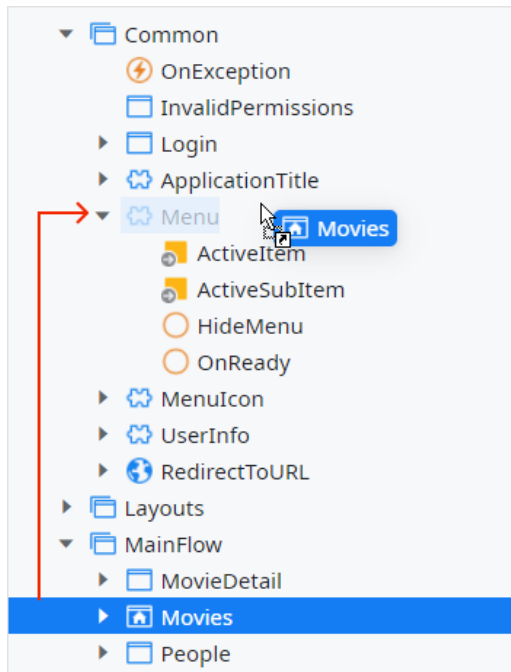
Just like we did for the movies, now we need to create a navigation between the People and the PersonDetail Screens.

1. Go back to the **People** Screen and create two Links to the **PersonDetail** Screen
 - a. One Link per **Name** in the People Table
 - b. One Link *New Person* in the Actions area of the Screen
2. In the PersonDetail Screen, create a Link to return to the People Screen with the following Text: *Back to People*
3. Publish the application and test it in the Browser!
4. At this point, you won't be able to access the People Screen. So, we need to add two entries to the application menu: Movies and People.
5. In the OSMDb module, under the Interface tab, expand the Common UI Flow.



NOTE: This UI Flow has common Screens and Blocks to the application, including the Login Screen and the Menu Block.

6. To add an entry to the Menu, drag a Screen and drop it over the Menu Block. We need to do this for the Movies and for the People.



7. Publish the application and test it in the browser.
8. Every application Screen should have the menu on top.

