

Detail Screens

Exercise - How to

Outline	2
How to	3
MovieDetail Screen	3
Linking Both Screens	17
Edit Movie Link	17
New Movie Link	20
Back to Movies Link	21
PersonDetail Screen	22
Linking the Screens	29

Outline

In this exercise, we will build two detail Screens to allow creating and editing movies and people in the app.

1. MovieDetail Screen

- a. Screen to display the details of a particular movie, identified by its Id, in a Form.
- b. Make sure the Screen allows 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 to the MovieDetail Screen in every movie to display its detailed data, and a Link at the top of the Screen to allow creating a new movie.

2. PeopleDetail Screen

- a. Screen to display the details of a particular person, identified by their Id, in a Form.
- b. Make sure the Screen allows 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 to the PersonDetail Screen in every person to display its detailed data, and a Link at the top of the Screen to allow creating a new person.

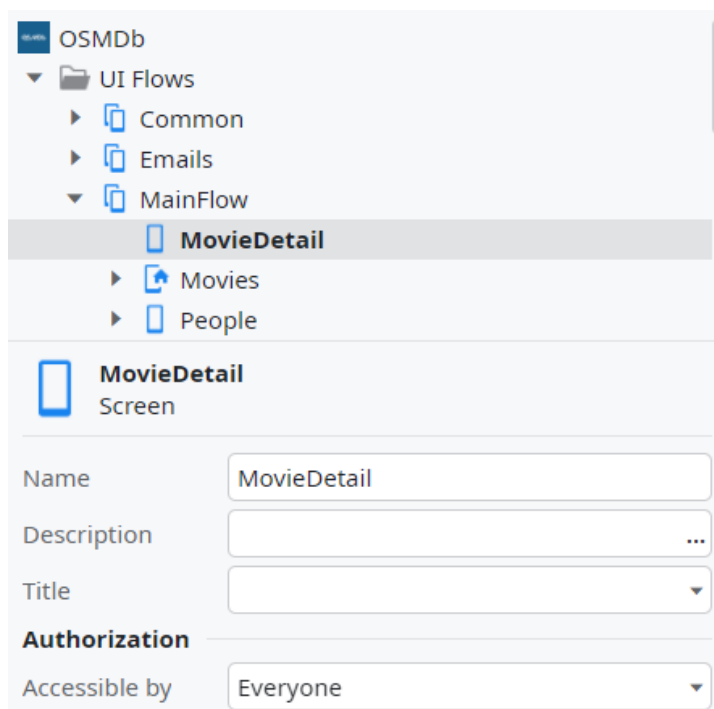
How to

In this section, we'll describe exercise 4.3 – *Detail Screens*, step by step.

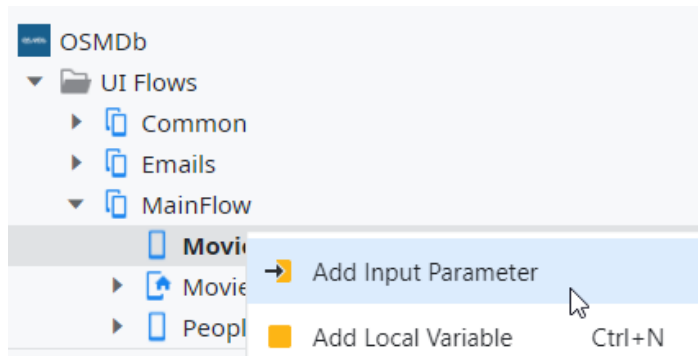
MovieDetail Screen

In the previous exercise, we created the Movies Screen to list all movies. Now, we want to create a new Screen, based again on an Empty template, that allows displaying the detailed information of a movie while also allowing editing that information. Also, the same Screen should be used to add a new movie to the database. To do that, we need to fetch data from a specific movie, giving its identifier, and we need to use a Form widget to represent the information on the Screen. Let's do it in several steps!

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 and name it *MovieDetail*. Like the first screen we created, this one should also be accessible by **Everyone**.



- b. Right-click the *MovieDetail* Screen and select the option **Add an Input Parameter**.

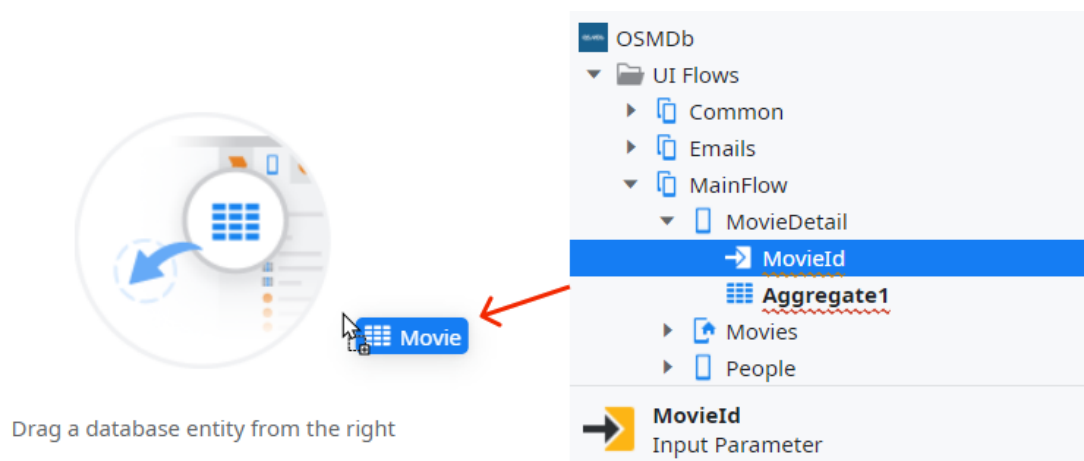


- c. Set the **Name** of the parameter to *MovieId*, with **Data Type** set to *Movie Identifier*.

The screenshot shows the configuration form for the 'MovieId' Input Parameter. The form has the following fields:

- Name:** MovieId
- Description:** (empty field with a help icon)
- Data Type:** Movie Identifier
- Is Mandatory:** Yes
- Default Value:** (empty field)

- d. Create a new Aggregate by right-clicking on the Screen and selecting **Fetch Data from Database**.
- e. Drag the **MovieId** Input Parameter into the Aggregate.



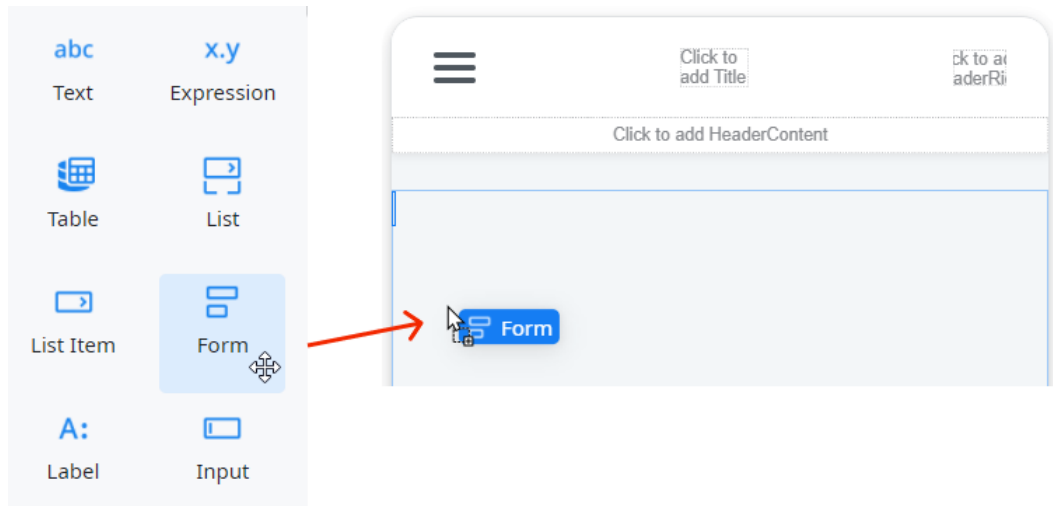
Notice that the Aggregate was automatically renamed as *GetMovieById*, the Movie Entity was added as the source, and a new Filter using the MovieId Input Parameter was created.

The screenshot shows the configuration interface for an aggregate named 'GetMovieById'. The interface is organized into several sections:

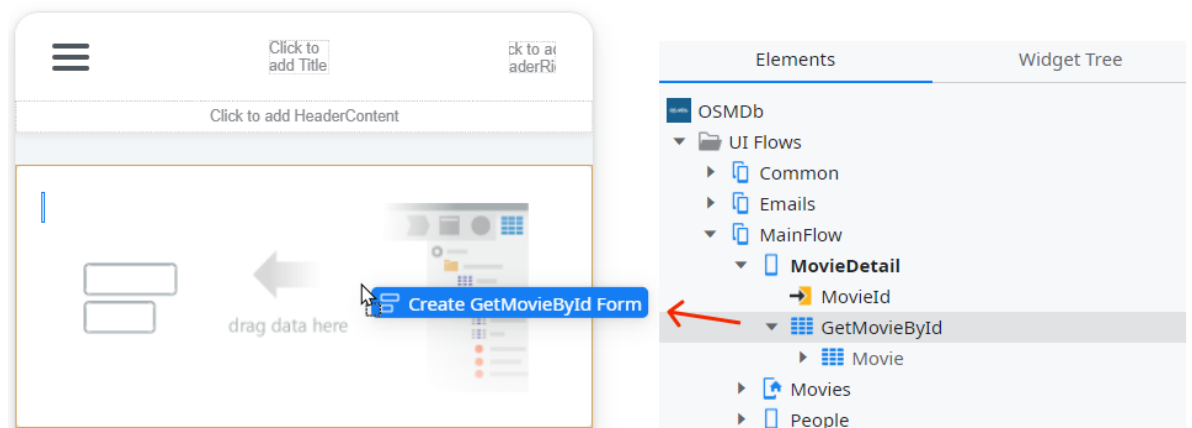
- General Properties:**
 - Name:** GetMovieById
 - Description:** (Empty field)
 - Server Request Timeout:** (Module Default Timeout)
 - Start Index:** (Dropdown menu)
 - Max. Records:** 50
 - Fetch:** At start
- Events:**
 - On After Fetch:** (Dropdown menu)
 - Executed SQL:** (Empty field)
- Sources:**
 - Movie
- Filters:**
 - Movie.Id = MovieId
- Test Values:**
 - MovieId

This accelerator is fully based on the data type of the Input Parameter. The Aggregate can then be adjusted to our needs if needed. Remember, you can basically customize whatever is automatically generated by OutSystems.

2. Create a **Form** on the MovieDetail Screen with an input field for each attribute in the Movie Entity.
 - a. Double-click on the **MovieDetail** Screen to open it in the preview area.
 - 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.



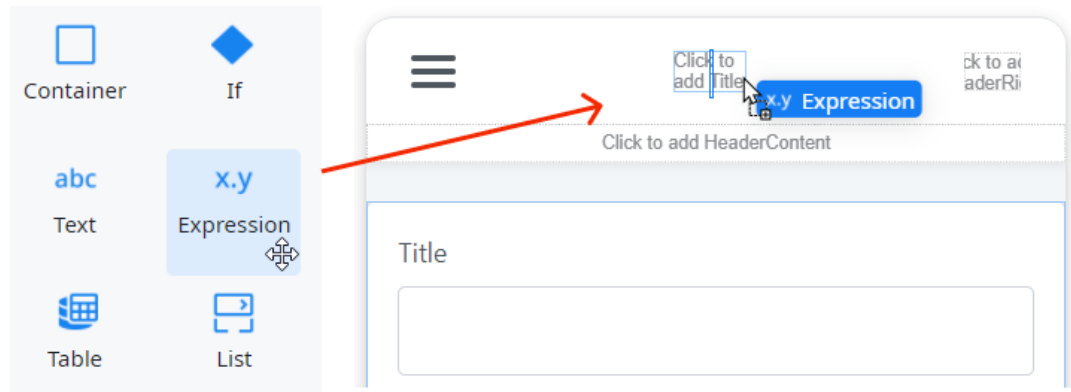
All the Movie Entity's attributes were automatically added as input fields in the Form. Also, a new Save button was created. It has an error, but don't worry about it now, we will fix it later. The end-user will use this Form and the input fields to submit information about movies to the database.

The image shows a vertical form with the following elements from top to bottom:

- A label "Title" above a rectangular text input field.
- A label "Year" above a rectangular text input field.
- A label "Plot Summary" above a rectangular text input field.
- A label "Gross Takings" above a rectangular text input field.
- A label "Is Available On DVD" above a blue square checkbox containing a white checkmark.
- A blue rectangular button with the white text "Save".

3. Now we will do something new with the **Title** of the Screen. When we are creating a new movie, the Title should display *New Movie*, while when we are editing an existing movie, it should display the movie title. You might be wondering “*How do we know if we’re creating a new movie or not?*”. Well, it all depends on the MovieId Input Parameter of the MovieDetail Screen. Every movie in the database has a numeric identifier that uniquely identifies the movie. That identifier will **never be zero**. So, if the MovieId parameter has a value that is not zero, it means that we’re editing a movie that exists in the database. If it is zero, it means that we’re creating a new movie, since zero is not a valid identifier in the database.

- a. Drag an **Expression** to the Title section of the Screen.

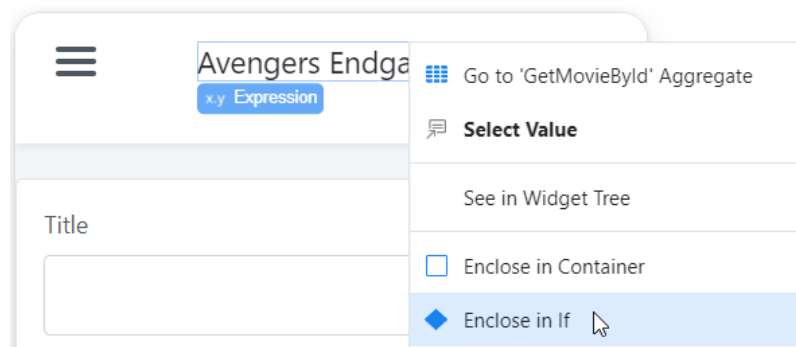


Note: Since we want a dynamic title that needs to be evaluated at runtime, we need to use an Expression instead of plain text.

- b. In the new dialog that appears, set the Expression Value to `GetMovieById.List.Current.Movie.Title`

This determines that the Expression will display the title of the movie fetched from the database. The Aggregate, filtered by the movie identifier, will guarantee that the movie fetched is the one we want to edit.

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



Note: The If widget allows displaying content based on a Condition. We can display an Expression with the title of the movie if the Condition is met, otherwise it shows something else.

- d. Set the Condition of the If to `MovieId <> NullIdentifier()`

If

Properties

Name:

Condition:

Animate:

So, what does this Condition mean? It compares the MovieId Input Parameter of the Screen to *NullIdentifier()*. This *NullIdentifier()* value is built-in OutSystems and returns the null value of the identifier, which, in this case, is zero since the identifier is an integer. So, if the MovieId is different from zero, it means it is a valid identifier that corresponds to a movie in the database. If it is equal to *NullIdentifier()*, then that movie does not exist. Simple as that!

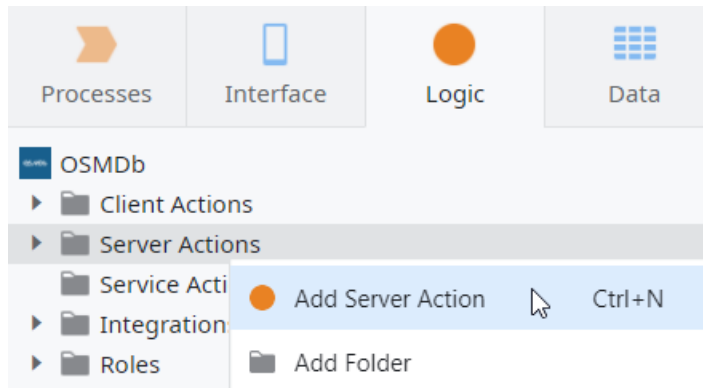
- e. Type *New Movie* in the False branch of the If.

Avengers Endgame
New Movie

Title

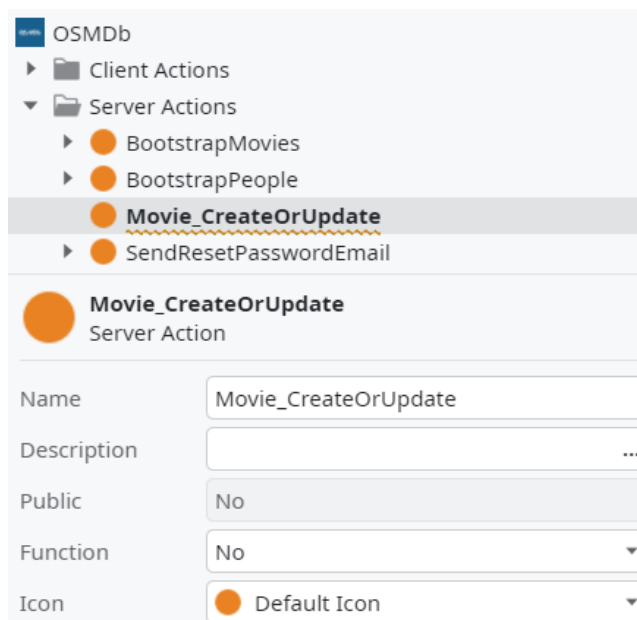
If the movie identifier is in fact equal to *NullIdentifier()*, then we're creating a new movie.

- 4. Now it's time to define the logic to create/update a movie in the database and associate it with the **Save** button. Let's start with the server-side logic that will use an Entity Action to create/update the movie information in the database, returning the movie identifier of the movie created/updated in an Output Parameter.
 - a. Switch to the **Logic** tab and right-click on the **Server Actions** folder and click on **Add Server Action**.

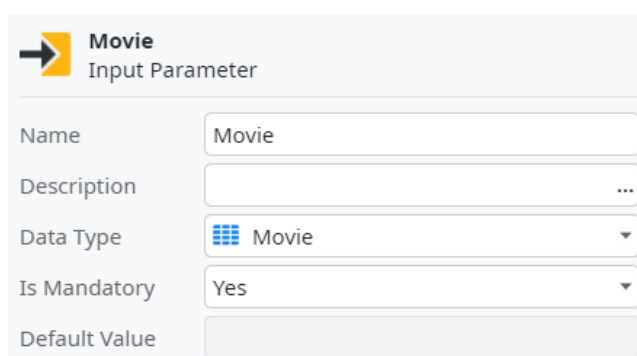


The database is hosted on the server, so we will create a Server Action to build all the server-side logic to update the database.

- b. Name it *Movie_CreateOrUpdate*.



- c. Add one Input Parameter to the Action and name it *Movie*. Confirm that the **Data Type** is *Movie*.

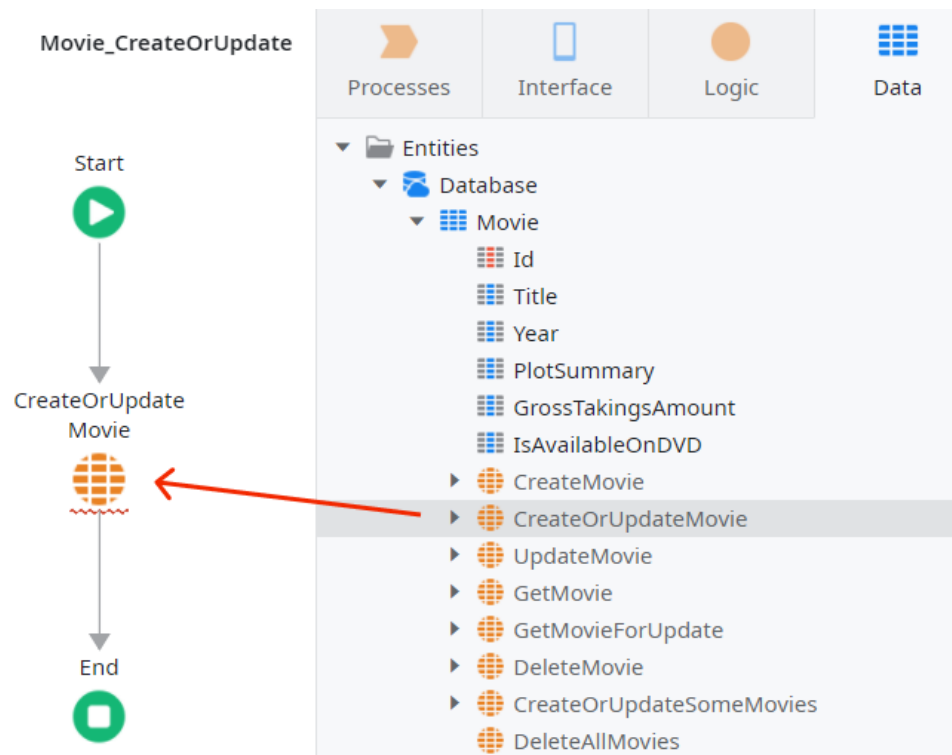


- d. Add an Output Parameter and name it *MovieId*. Confirm that the **Data Type** is *Movie Identifier*.

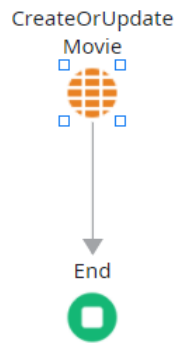
The screenshot shows the configuration for an output parameter named **MovieId**. The form includes fields for Name, Description, Data Type, and Default Value. The Data Type is set to **Movie Identifier**.

MovieId Output Parameter	
Name	MovieId
Description	
Data Type	Movie Identifier
Default Value	

- e. Switch to the Data tab, find the **CreateOrUpdateMovie** Entity Action in the Movie Entity, then drag it to the Action flow of the *Movie_CreateOrUpdate*.



- f. Click on the **CreateOrUpdateMovie** element dragged in the previous step and set its source to **Movie** (the Input Parameter of the Action).

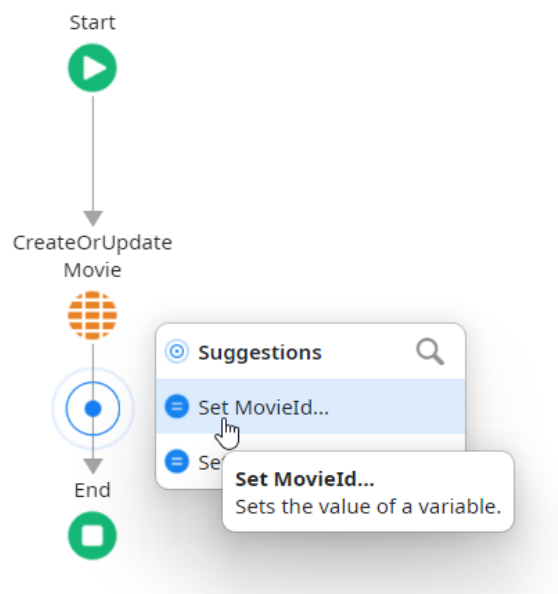


CreateOrUpdateMovie
 Run Server Action

Name	CreateOrUpdateMovie
Action	CreateOrUpdateMovie
Source	Movie

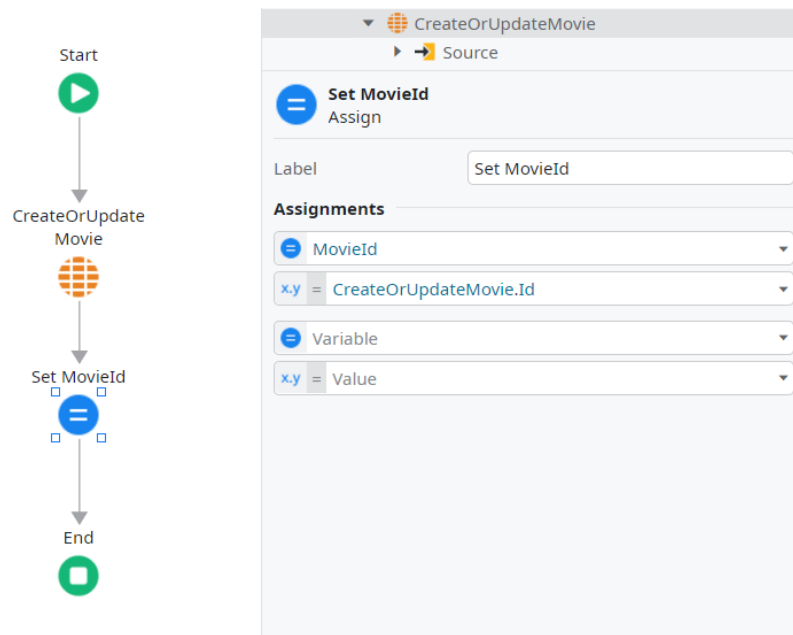
At this point, we're using the Entity Action that creates or updates the movie information in the database. We use the Input Parameter that contains the movie's information to be sent to the database.

- g. Hoover the mouse over the last arrow of the flow and click on the blue icon. Choose **Set MovieId...** to create the *MovieId* assignment.



This automatically creates an Assign element with the *MovieId* Input Parameter.

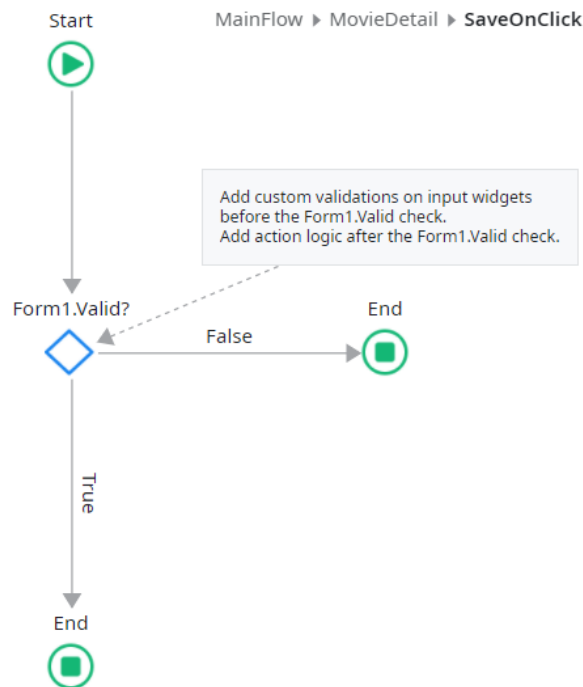
- h. Choose the **Value** of the Assign to be *CreateOrUpdateMovie.Id*



This sets the value of the Output Parameter from the Server Action as the output of the Entity Action, which returns the identifier of the movie that was created or updated. And that's it! We will come back to this Action in a later exercise.

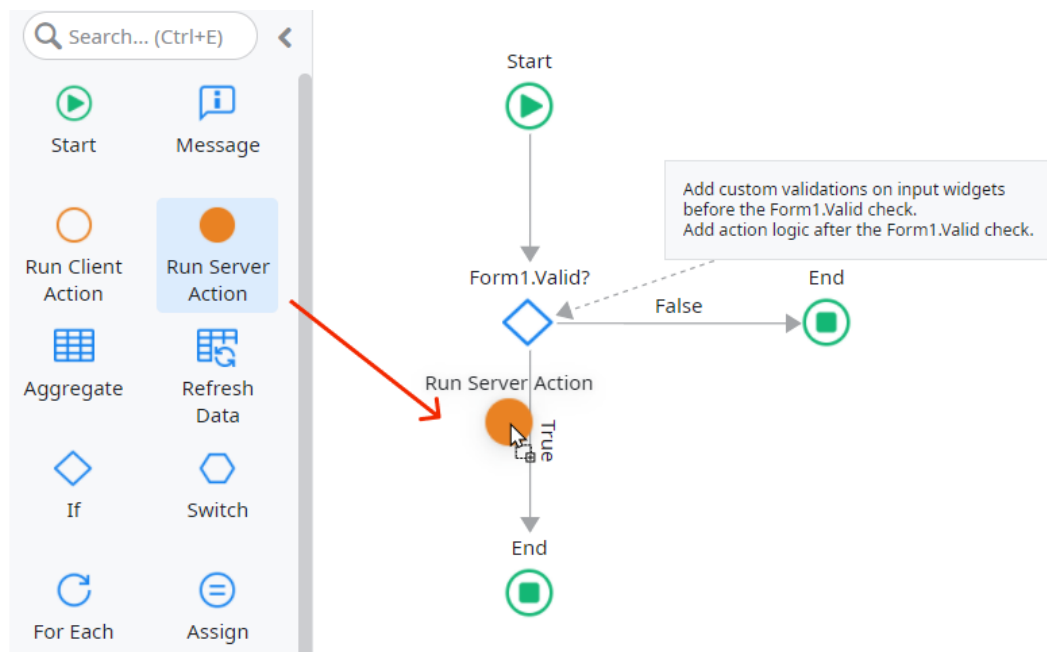
Note: This is an example of how we can leverage AI with some recommendations for the next steps in an Action flow. In this case, since the Action has an Output Parameter, OutSystems recommended an Assign to set its value. This varies from app to app and from the patterns most commonly used by developers.

5. We have the server-side logic ready. However, we still have an error in the app in the Save Button. The error is caused by not having the implementation of the Save button. Every Button widget needs to have the behavior that occurs when the user clicks on the Button set up. This Save button in particular will trigger the logic to save the movie information in the database. In the end, a success message should be displayed to the end-user and the app will navigate back to the Movies Screen.
 - a. Switch back to the **Interface** tab and double-click on the **MovieDetail** Screen to open it in the preview area.
 - b. Double-click on the **Save** button to create the **SaveOnClick** Screen Action.

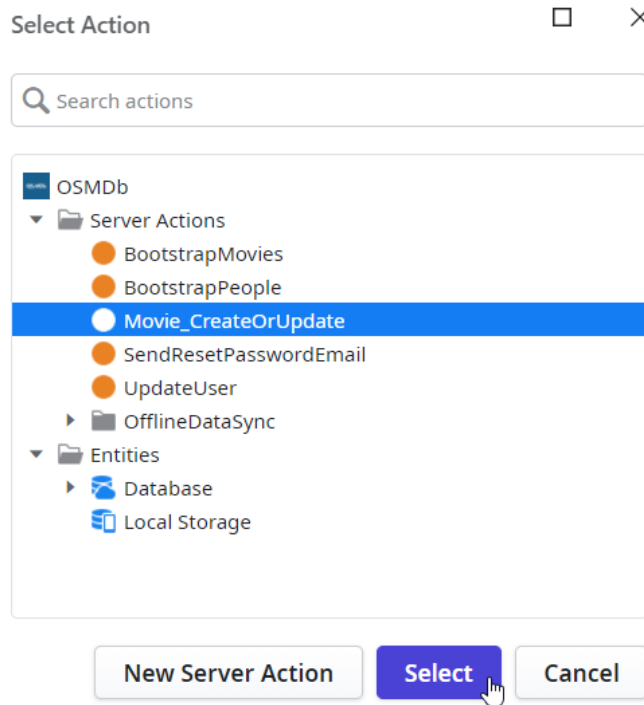


This automatically creates (and opens) a new Client Action, in the scope of the Screen. This Action will be triggered whenever the user of the app clicks on the Save button. This Action needs to call the Server Action created earlier with the logic to save the information in the database.

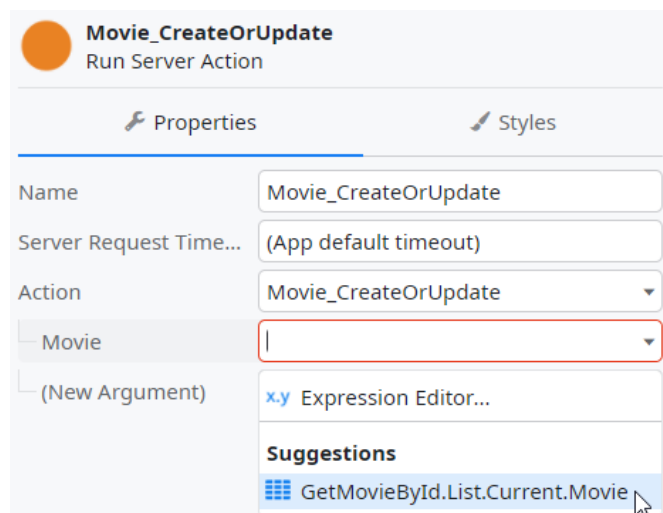
- c. Drag a **Run Server Action** node to the flow and drop it below the If.



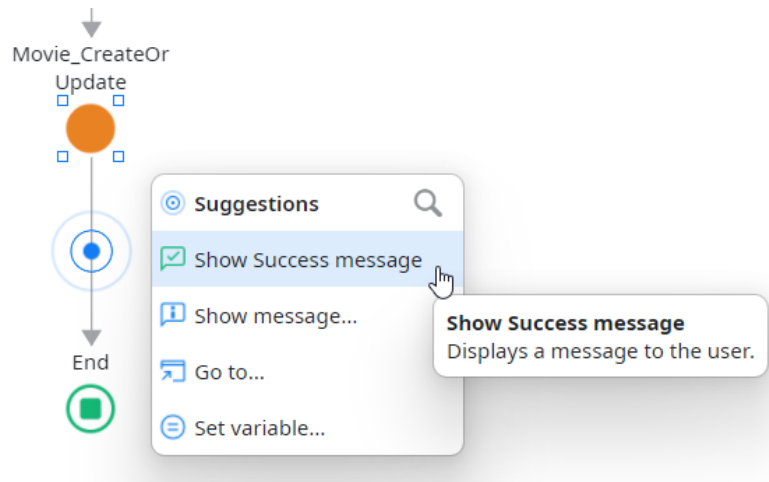
- d. In the new dialog, select the **Movie_CreateOrUpdate** and click Select.



- e. In the properties of the Action, set the **Movie** Input Parameter of the Action to *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 information to be created/updated to the **Movie_CreateOrUpdate** Action.



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



- g. Set the **Message** to *"Movie created/updated successfully"* and set the **Type** to **Success**.

"Movie created/updated successfully"
Message

Properties Styles

Label

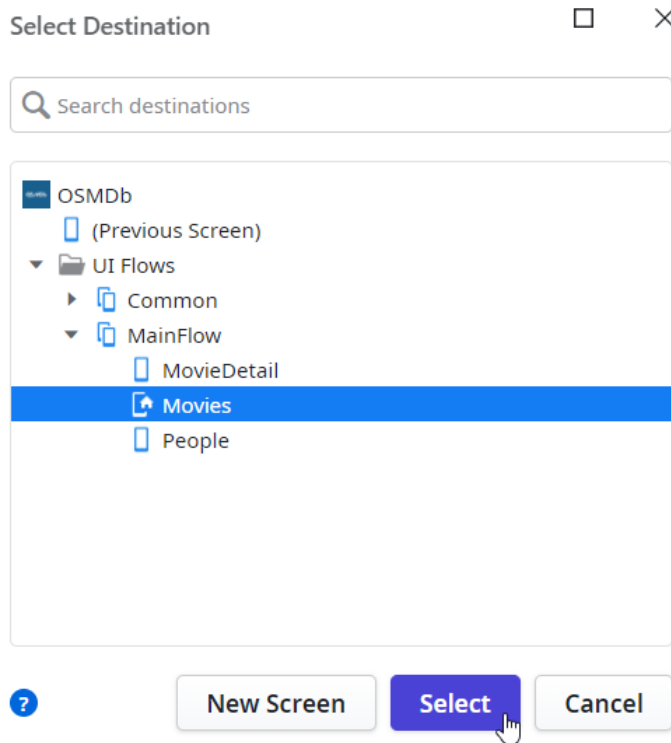
Message "Movie created/updated successfully" ▼

Type Success ▼

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



- i. Select the **Movies Screen** and click **Select**. We're done!



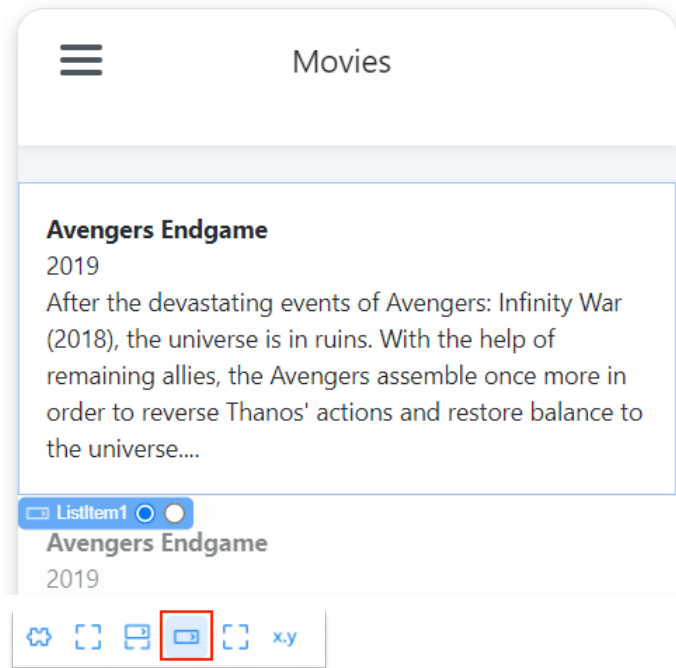
Linking Both Screens

Now that we have created the Movie and the MovieDetail Screens, we want to make sure we define proper navigation between them. The Movies Screen should have two links to the MovieDetail Screen, one to create new movies and one to for each movie in the List to allow editing information about them. We should have a link in the MovieDetail Screen that allows returning to the Movies Screen without saving or creating a movie.

Edit Movie Link

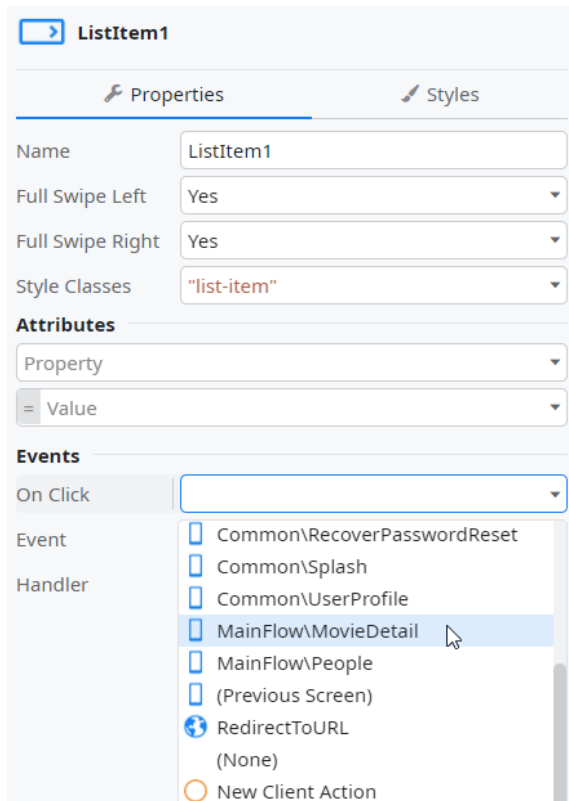
The first Link that we will work on is to edit the information of an existing movie. This means that every movie in the List will have a Link around its Title to navigate to the respective MovieDetail Screen. When the user gets to the MovieDetail Screen, the information about the movie clicked on should appear in the Form.

1. In the Interface Tab, double-click on the **Movies** Screen to open it.
2. Click on the List on the Screen, and by using the **Widget Breadcrumb** in the area selected on the next screenshot, select the **ListItem1** widget, which represents each record being displayed on the List.

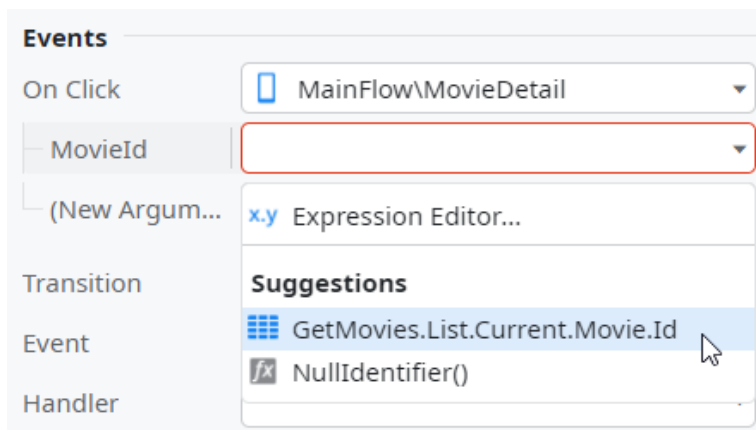


Note: The Widget Breadcrumb area appears whenever we click on a widget on a Screen. It shows the hierarchy of widgets of what we are selecting. For instance, if we click on the title of the movie, the expression is selected. But, on the Widget Breadcrumb, it will also appear the ListItem and the List widgets, which are parents of the Expression, since the Expression was placed inside them. This makes it easier to select the widget we want and work on it.

3. In the ListItem1 properties, expand the **OnClick** property and select the **MovieDetail** Screen. This means that whenever a user clicks on the ListItem widget corresponding to a Movie, it will open the MovieDetail Screen.



4. In the input parameter of the Link's **OnClick** Destination select *GetMovies.List.Current.Movie.Id*.

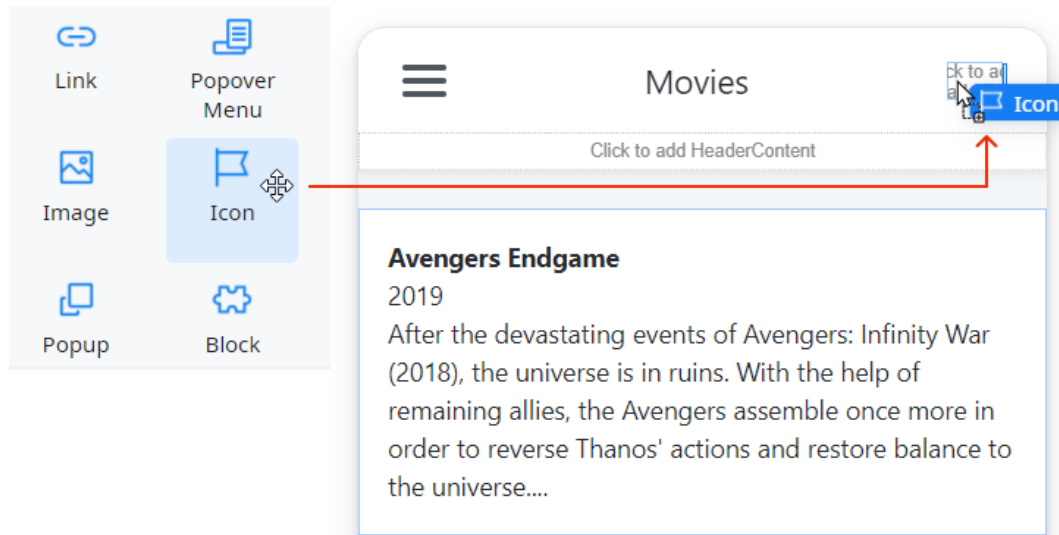


Note: This guarantees that the Id of the movie selected is passed to the MovieDetail Screen. In this case, the MovieDetail will display the details of this movie and the end-user will be able to edit it. Considering the way we implemented the MovieDetail Screen, this is the final step that guarantees that everything will work as expected.

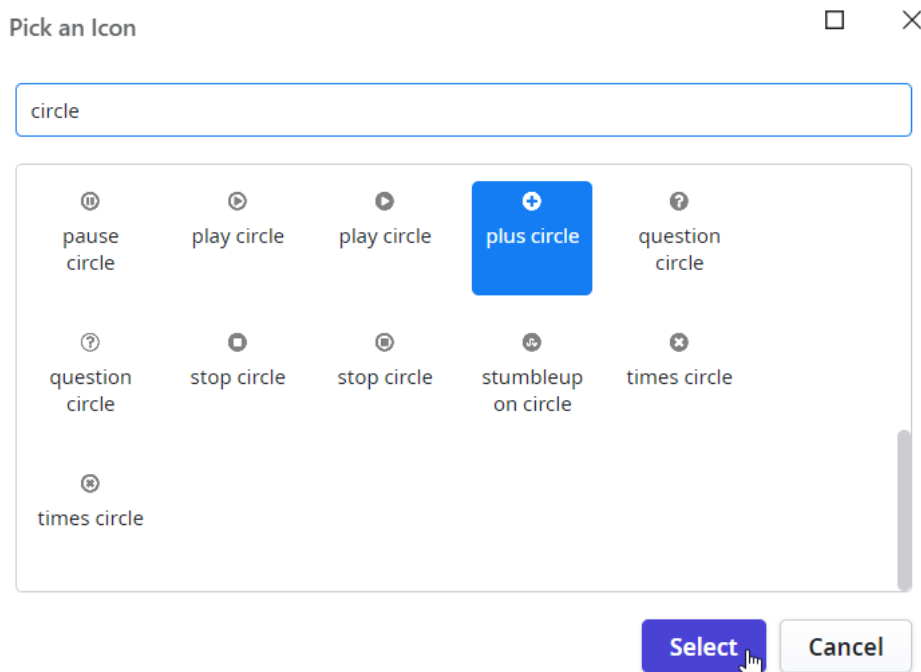
New Movie Link

Now, we want a link to create a new movie on the top right of the Screen.

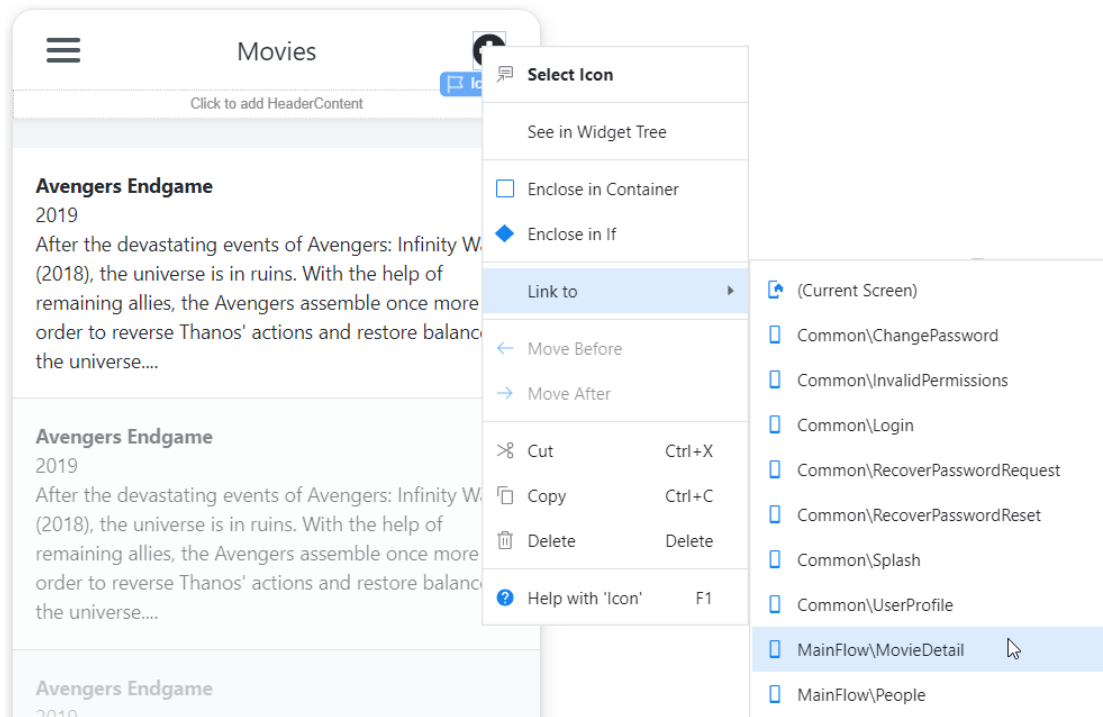
1. Drag an **Icon** widget and drop it on the top right of the Movies Screen, in the **Actions** area.



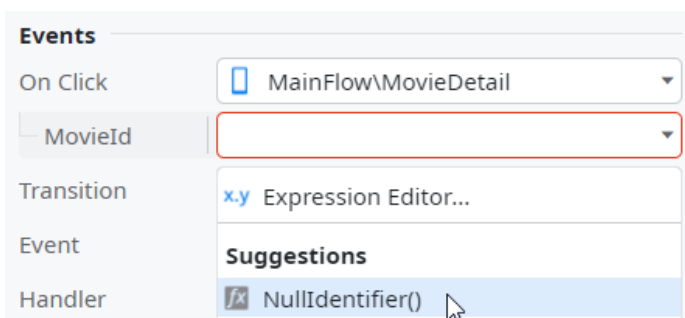
2. In the new dialog, type circle and select the *“plus circle”* Icon.



3. Right click the Icon and select **Link to** -> **MainFlow\MovieDetail**.



4. Using the same logic as the previous Link, we need to set the value of the **MovieId** Input Parameter. In this case, we'll set the value as *NullIdentifier()*.

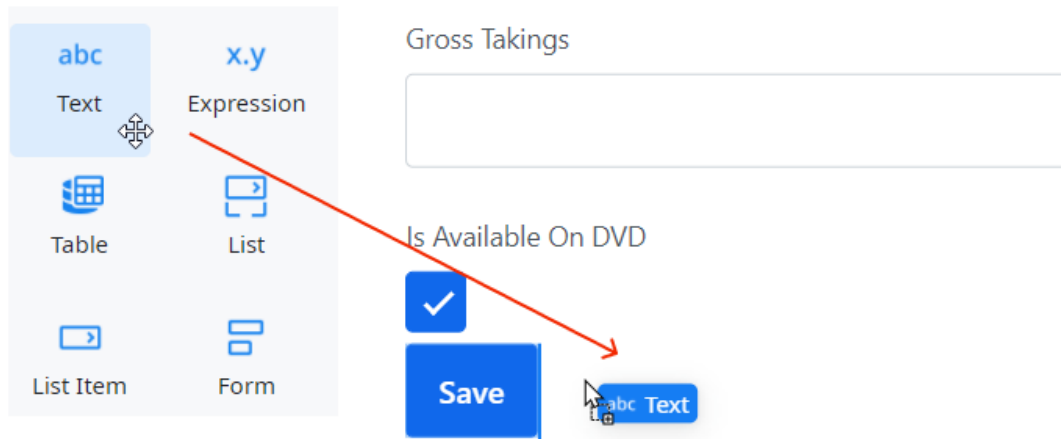


Note: Remember that, since it's a new movie, it does not have an Id yet and that's why we passed the value *NullIdentifier()*.

Back to Movies Link

There's only one link missing. We should have a Back Link in the MovieDetail Screen to allow users to navigate back to the Movies Screen.

1. Double-click on the MovieDetail Screen to open it in the preview area.
2. Drag a **Text** widget and drop it next to the Button.



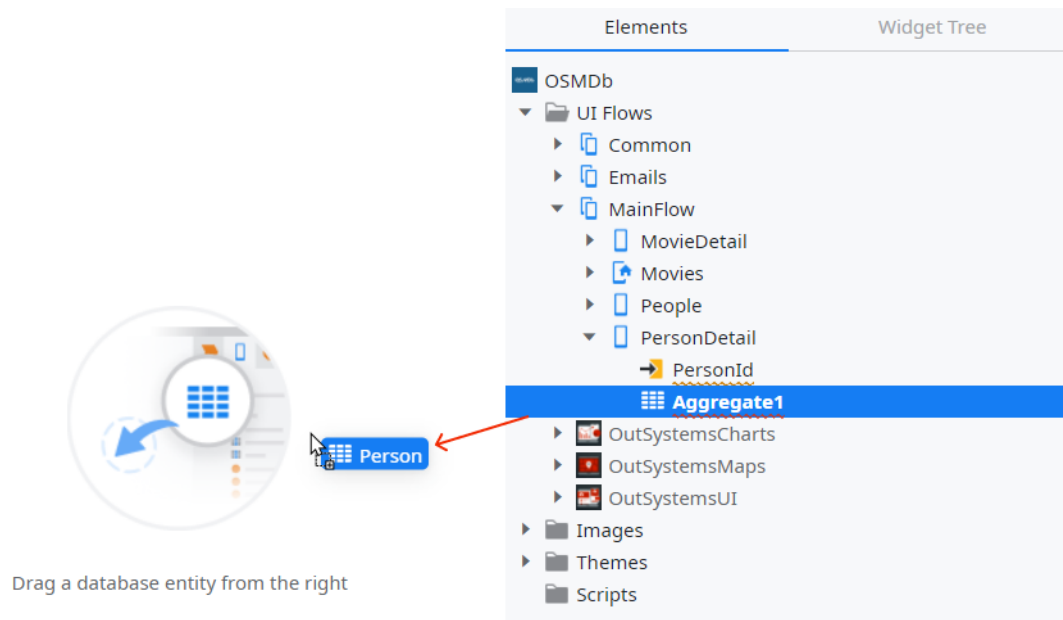
3. Set the Text to be *Back to Movies*.
4. Right-click on the Text and create a Link to the **Movies** Screen.
5. **Publish** the app and test it in the browser!



PersonDetail Screen

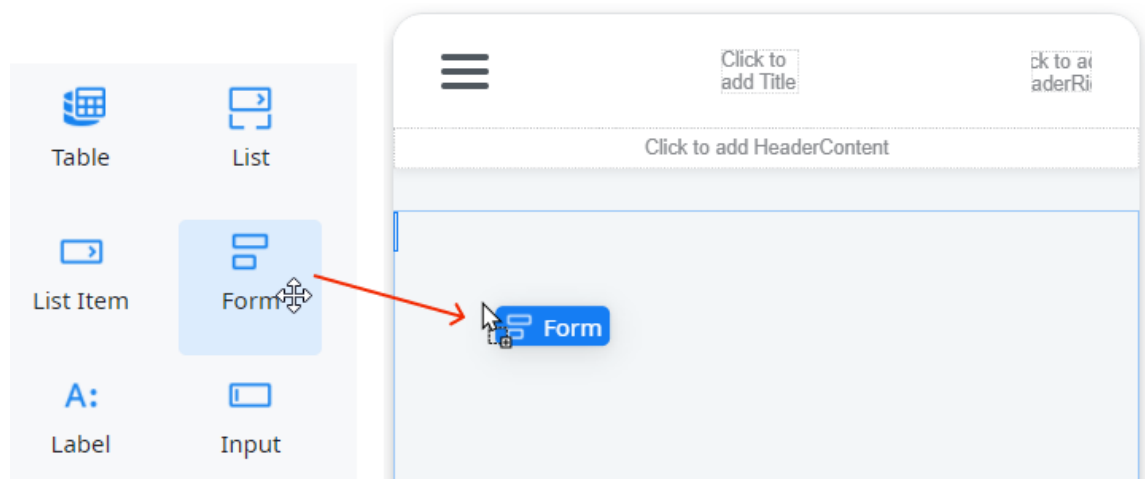
Just like we did for the movies, now it's time to create the PersonDetail Screen. This Screen should have a Form with input fields for all the attributes in the Person Entity. The Screen should also expect the Id of the Person as input. That input should be used to fetch the information about the Person. Also, the logic to create/update the Person in the database should be triggered by a Save button.

1. Create a new **Empty** Screen and name it *PersonDetail*. 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*. Just like the other screens, this also should be accessible by **Everyone**.
 - b. Add an Input Parameter called *PersonId* and set the **Data Type** to *Person Identifier*.
 - c. Create a new Aggregate and use the **PersonId** Input Parameter to only return the Person whose Id matches the value in the Input Parameter. The Aggregate should change the name to *GetPersonById*.

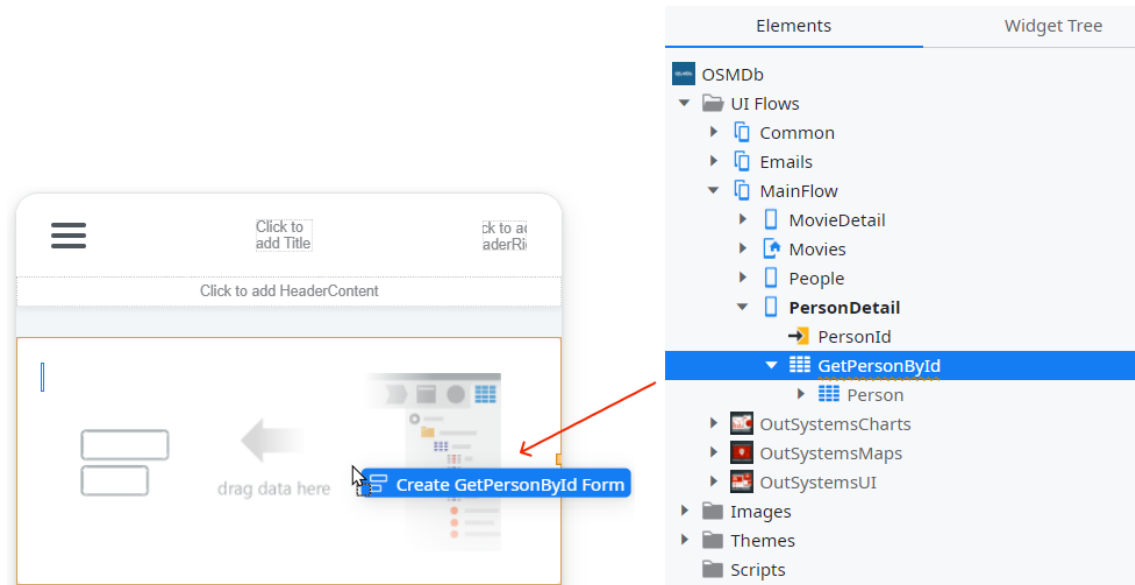


2. Now it's time to build the UI of the Screen that will have a Form with four input fields, one for each attribute of the Person Entity. Also, define the title of the Screen just like you did in the MovieDetail Screen, with the text New Person or the name of the Person.

- a. Open the PersonDetail Screen and drag a Form to the Screen.

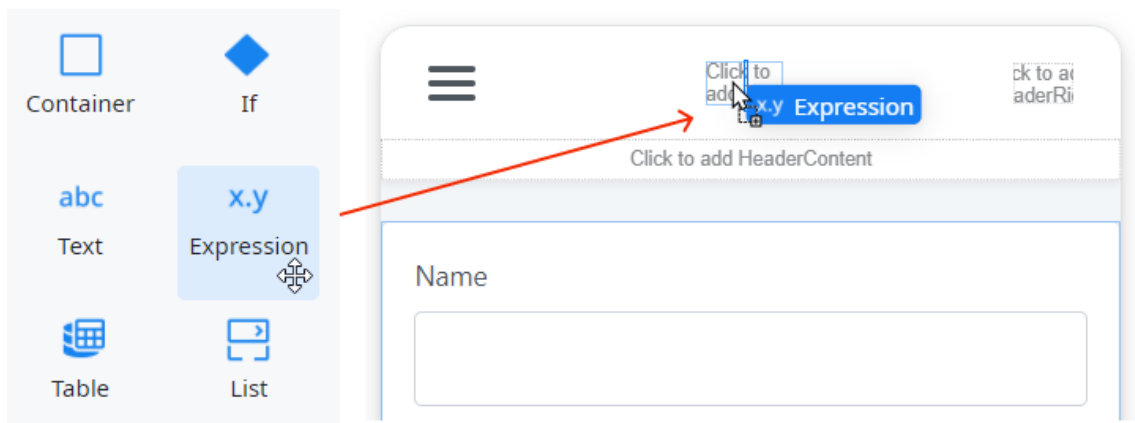


- b. Drag the **GetPersonById** Aggregate and drop it on the Form.

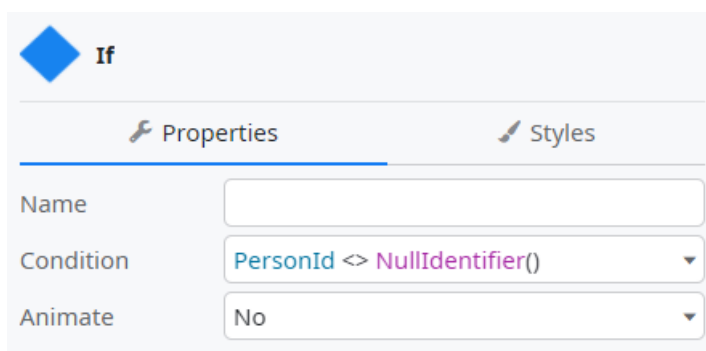


- c. Drag an **Expression** and drop it on the Title of the Screen. Set its value to:

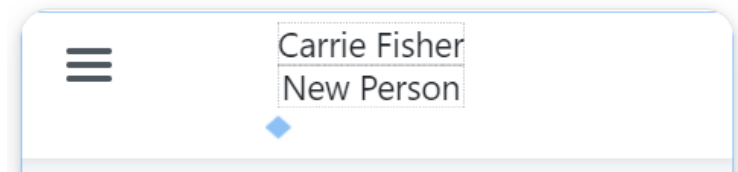
`GetPersonById.List.Current.Person.Name + " " +
GetPersonById.List.Current.Person.Surname`



- d. Enclose the Expression in an **If** and set its **Condition** to `PersonId <> NullIdentifier()`.

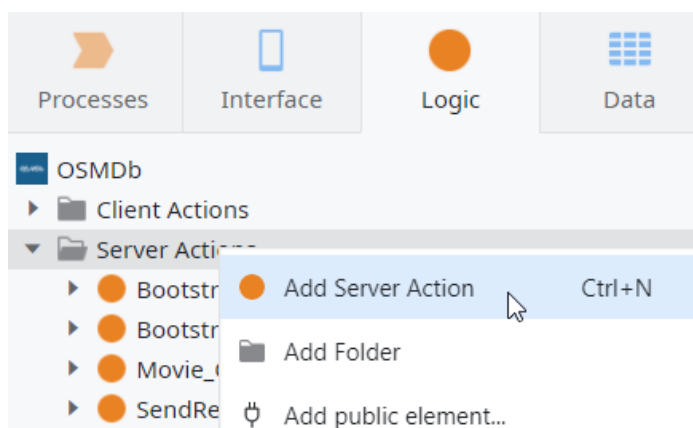


- e. In the **False** branch, type *New Person*.

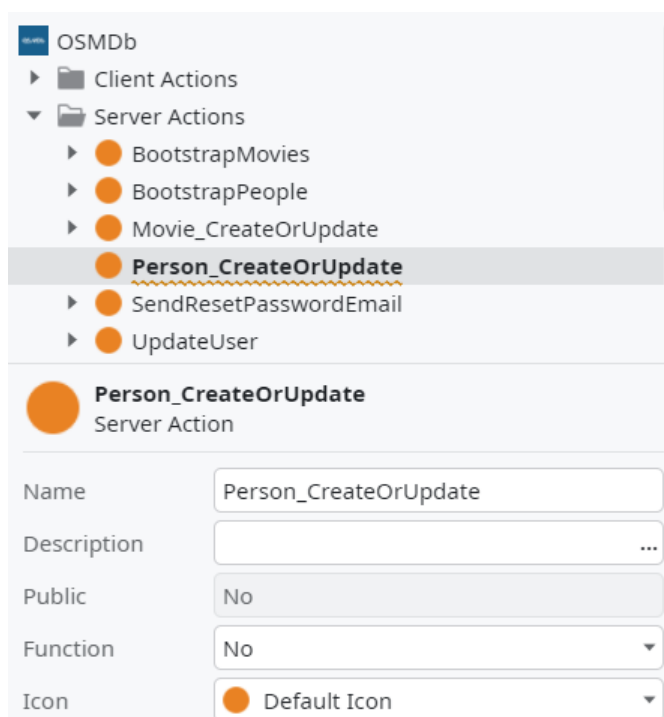


3. Now we need to create the logic to create/update a Person in the database like we did in the MovieDetail Screen.

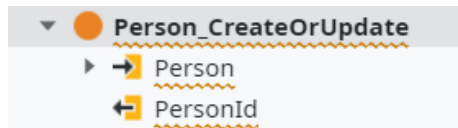
- a. Go to the **Logic** tab and right-click on the Server Actions folder and select **Add Server Action**.



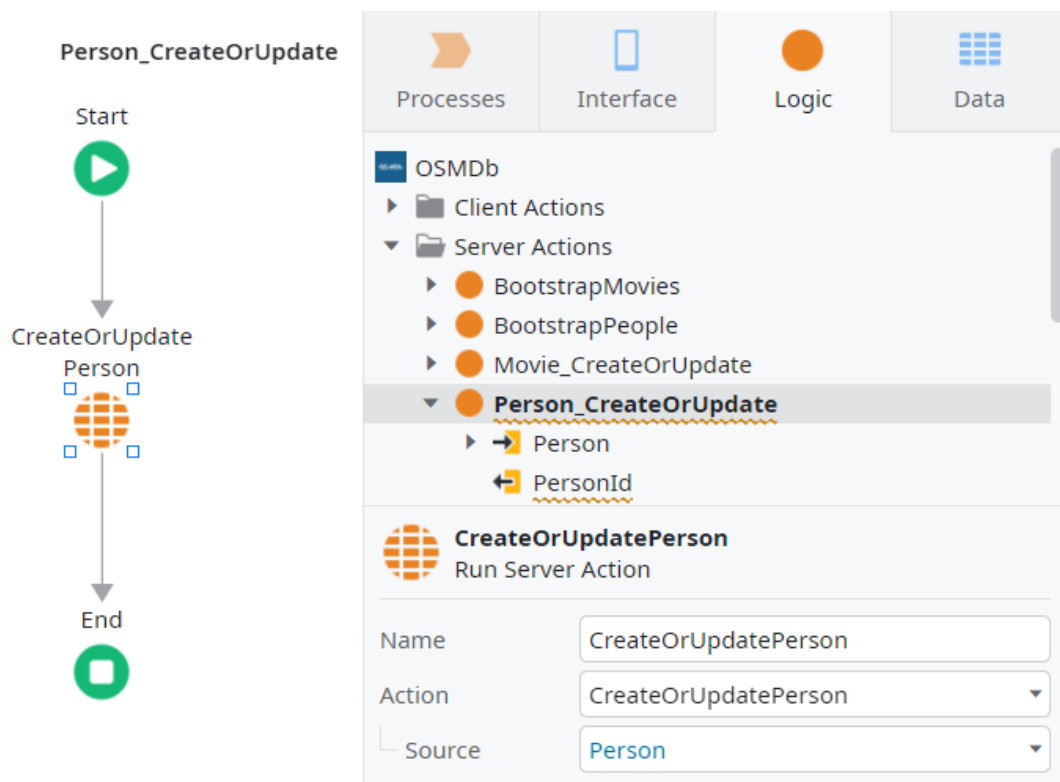
- b. Set its **Name** to *Person_CreateOrUpdate*.



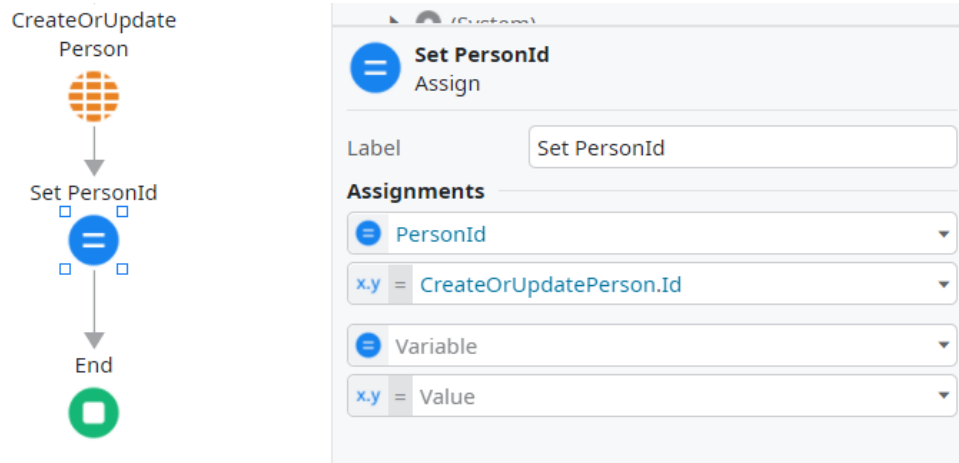
- c. Add one Input Parameter to the Action and name it *Person*. Make sure its **Data Type** is set to *Person*.
- d. Add an Output Parameter to the Action and name it *PersonId*. Make sure its **Data Type** is set to *Person Identifier*.



- e. Switch to the Data tab, drag the **CreateOrUpdatePerson** Entity Action to the Action flow, and set its **Source** to the Person Input Parameter.



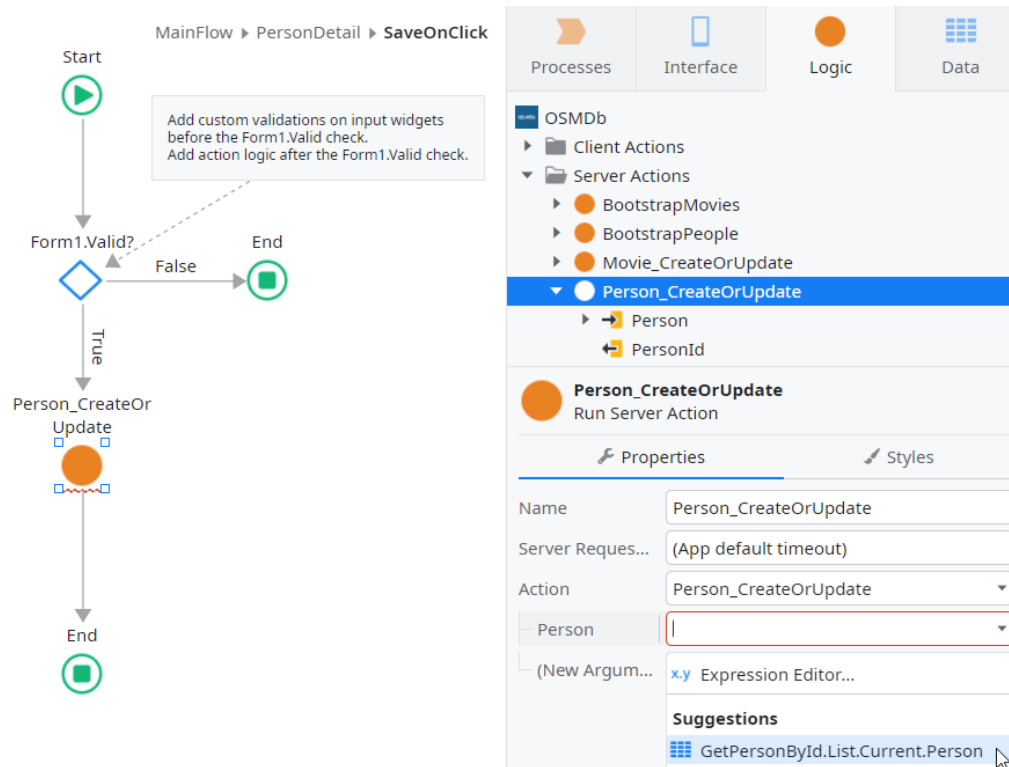
- f. Use the AI suggestions to assign the Output Parameter with the return value from the Entity Action: *PersonId* = *CreateOrUpdatePerson.Id*



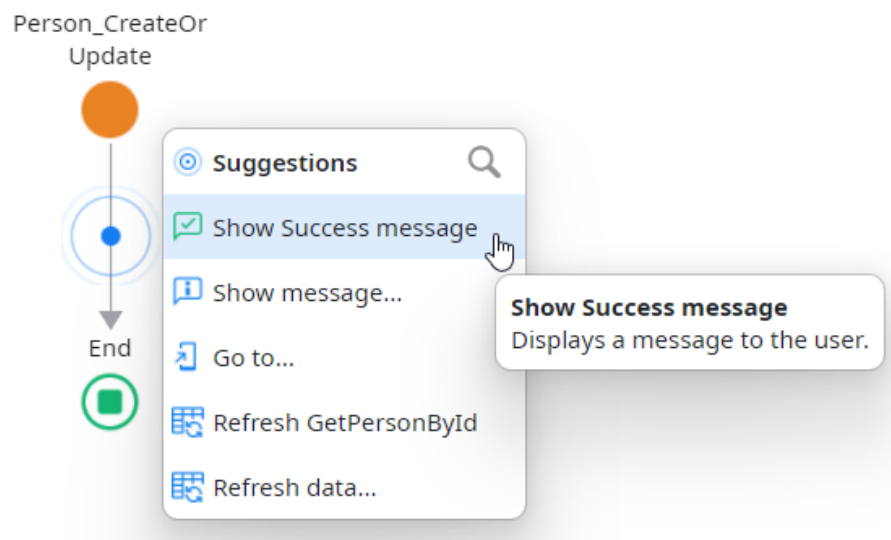
4. Create the logic in the PersonDetail Screen triggered by the Save button to create/update a person in the database.
 - a. Switch to the Interface tab, open the PersonDetail Screen, and double-click on the **Save** button to create the **SaveOnClick** Action.
 - b. Drag a **Run Server Action** and drop it after the existing If in this new Action. In the dialog, select the **Person_CreateOrUpdate** and click the **Select** button.

It looks very similar to what we've done for the movies right? And it will continue that way.

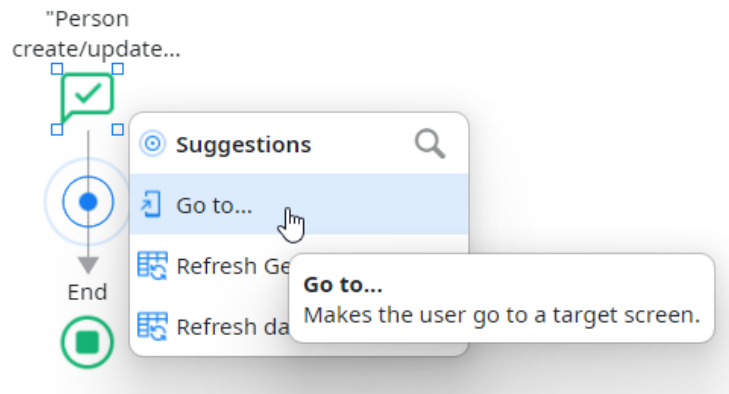
- c. Set its **Person** Input Parameter to the suggestion provided, which is the Person record being fetched from the database.



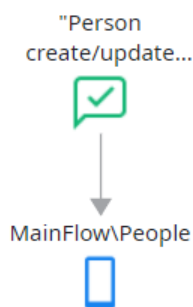
- d. Mouse over again the last arrow, click the blue icon and choose **Show Success message**.



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



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



And we're done with the logic.

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. In the Interface tab, go back to the **People** Screen and create two Links to the **PersonDetail** Screen:
 - a. One Link per **ListItem** On Click
 - b. One Link with the "plus circle" Icon in the top right 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 app and test it in the browser!