Actions and Code Reusability Exercise

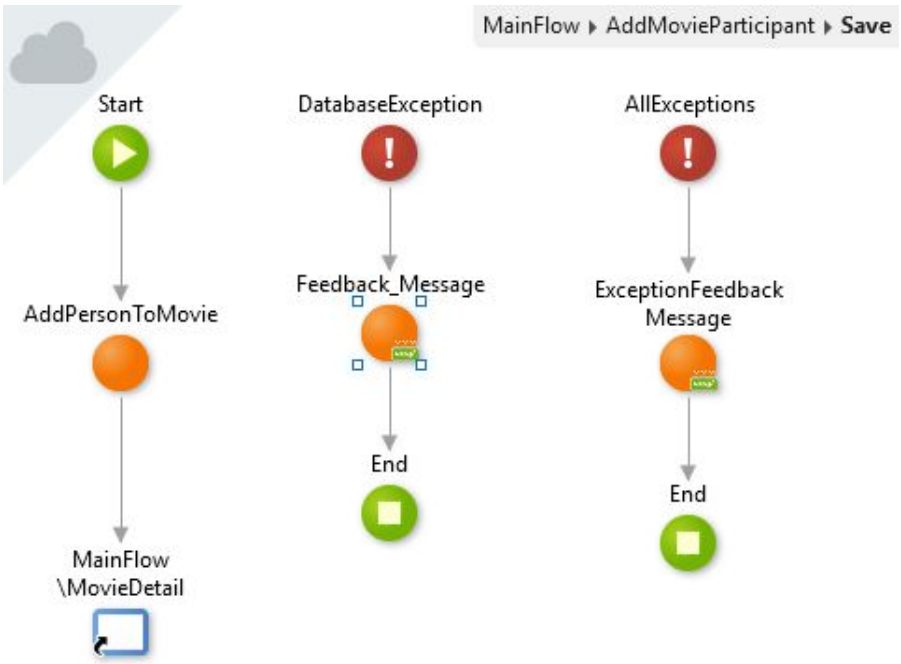


Table of Contents

Introduction

In this exercise Lab, we will create a new Server Action. This Action is a bit different from the ones we have been creating so far. It will be created under the Logic tab, meaning that it will not be associated with any Screen. This means that this Action can be used in any context of the module, namely in every Screen Action or Preparation, as well as other Server Actions and even Expressions. It can also be set as Public and shared to other modules.

Our new Server Action will be created in the OSMDb_Core module, and will be used to encapsulate the logic of assigning a role to a certain person in a given movie. This way, we will make the respective Entity as Read Only, and the only way to add this information in the database, is by calling this new Action. On the other hand, by being exposed as read only, the only Entity Action available to be used in the consumer modules is the *GetPersonMovieRole*. This means that it will not be possible to update or delete the role of a person in a given movie, unless specific Actions are created for that effect.

This pattern is actually an OutSystems Best Practice. First, we avoid that the Entity is exposed with write permissions to any consumer module, without any security or other logic around it. Second, by creating these Actions, we can add additional business logic inside them, that by calling the Entity Actions directly we would not have. As an example, we can add validation rules (e.g: user has permissions to add a role to a person?) or exception handling to these Actions.

In a future lab, we will go back to the Action and add more business logic to it. Also, we are only creating one Server Action to be exposed as Public, for the *PersonMovieRole* Entity, however we should do the same for the other Entities as well. We will leave that as an optional exercise at the end.

Finally, we will implement a Database Exception handler in the Save Action of the AddMovieParticipant Screen.

In summary, in this specific exercise lab we will:

- Create a reusable Server Actions in the OSMDb_Core module
- Make the PersonMovieRole Entity as read-only
- Replace the usages of the PersonMovieRole Entity Action in the OSMDb module, by the new Action
- Define Exception handling logic for Database Exceptions
- (Optional) Create Server Actions for creating and updating the movies and people in the database

Create an Action to assign a role to a person in a movie

We will start this Lab by creating a new Server Action. This one will be defined under the Logic tab of Service Studio. Here, we have all the reusable Server Actions, meaning that they can be used in different contexts and scopes of the module: e.g. Preparations and Screen Actions of any Screen, other Server Actions and even Expressions.

This particular Server Action will be created in the OSMDb_Core module and will be used to assign a role to a person in a given movie. This functionality is already available with the CreatePersonMovieRole Entity Action, however the PersonMovieRole Entity is being exposed to the consumer modules with write permissions. In OutSystems, it is best practice to expose the Entities as Read-only and provide some public Actions with all the functionalities we want the consumer modules to use. This way, not only we can control which Actions to expose (instead of all Entity Actions), as well as we can add business logic around it.

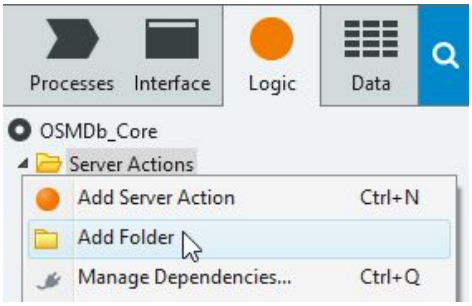
The Entity Actions are a black box, where we don't have access to its implementation. However, by using them in a Server Action Wrapper, we can build logic around it, including for instance Exception Handling, or data / permissions validations.

In this first part of the Lab, we will create a Server Action that will use the **CreatePersonMovieRole** Entity Action. This Action will later be used in the OSMDb module.

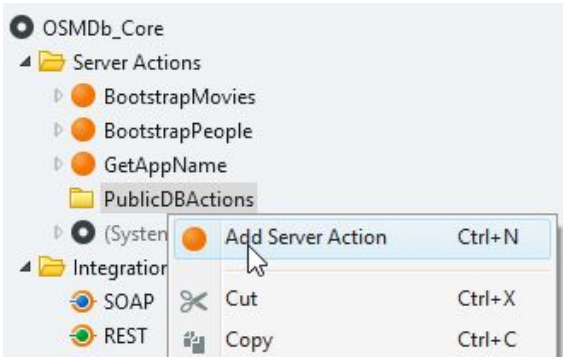
1) Add a Public Server Action to the **OSMDb_Core** module named *AddPersonToMovie*. Add three Input Parameters, *MovieId, PersonId and PersonRoleId* of types *Movie Identifier*, *Person Identifier* and *PersonRole Identifier* respectively. All of them are mandatory. Also add an Output Parameter *PersonMovieRoleId*, of type *PersonMovieRole* Identifier, to return the Id of the Record created in the database. Create this Action inside a new Folder called *Public DB Actions*.

    a) Open the OSMDb_Core module.

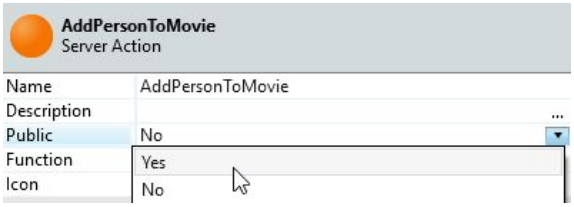
    b) Switch to the **Logic** tab, right-click the **Server Actions** folder and select **Add Folder**. Set its Name to *Public DB Actions*.



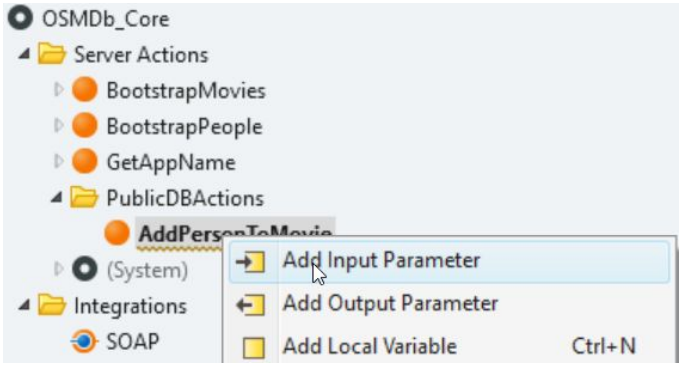
c) Right-click on the new Folder and select **Add Server Action**. Call the Action *AddPersonToMovie*.



d) Set the Action's Public property to *Yes*.

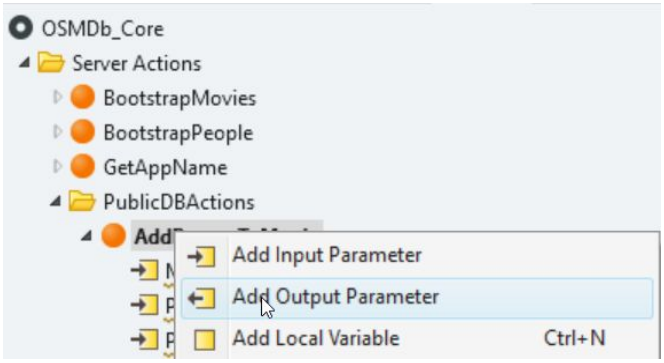


e) Add an **Input Parameter** to the **AddPersonToMovie** Action. Set its **Name** to *MovieId* and its **Data Type** to *Movie Identifier*. Set it as mandatory.



f) Repeat this process for the *PersonId* and *PersonRoleId* Input Parameters. Make sure that their types are *Person Identifier* and *PersonRole Identifier* respectively, and that they are both mandatory.

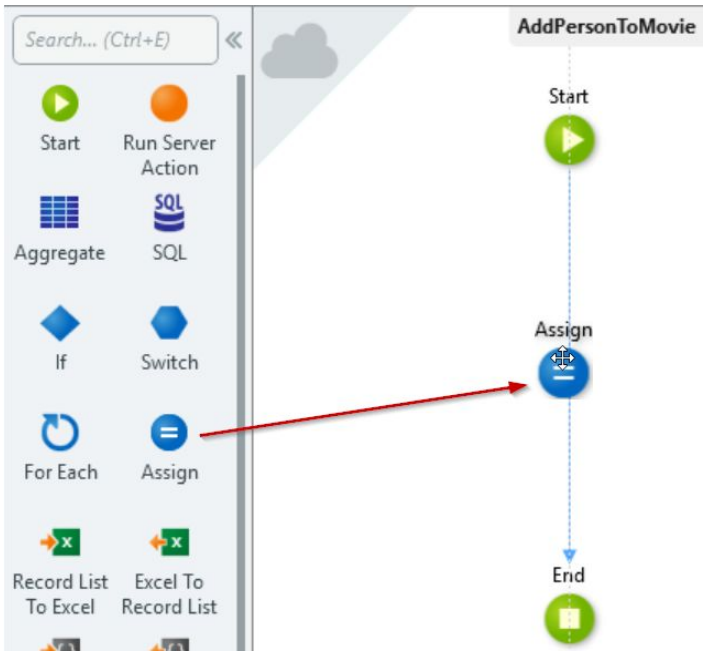
g) Add an Output Parameter to the Action. Set its **Name** to *PersonMovieRoleId* and its **Data Type** to *PersonMovieRole* Identifier.



2) Now that we have the Action created, it's time to define the logic within it. Initially, this action will add a PersonMovieRole record to the database, using the **CreateOrUpdate** Entity Action and the input parameters of the **AddPersonToMovie** Server Action.

a) Add a **Local Variable** to the **AddPersonToMovie** Action. Set its **Name** to *PersonMovieRole* and ensure its **Data Type** is *PersonMovieRole*. This Variable will be used to temporarily hold the new Record to be added to the database (with the information of the role of a person in a movie).

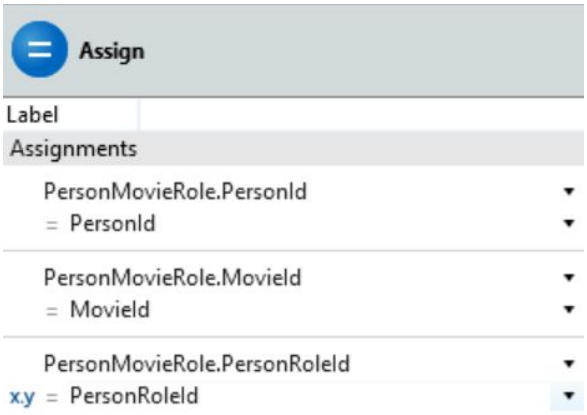
b) Drag and drop an **Assign** statement to the Action flow.



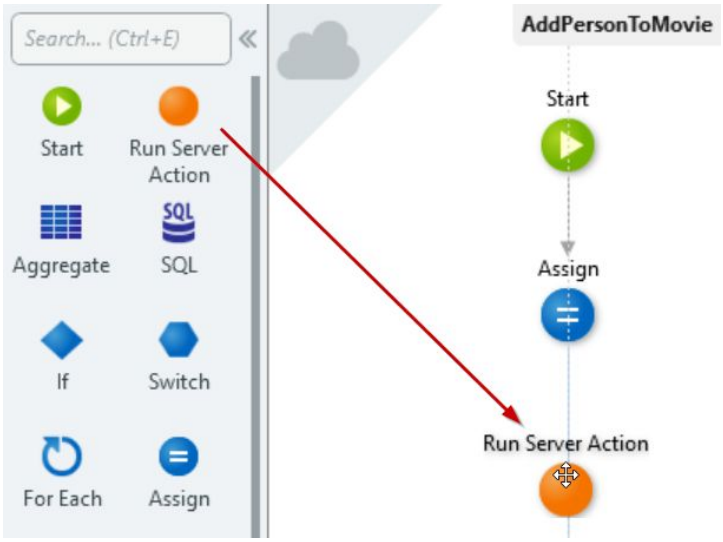
c) Define the following assignments:

PersonMovieRole.PersonId = PersonId
PersonMovieRole.MovieId = MovieId
PersonMovieRole.PersonRoleId = PersonRoleId

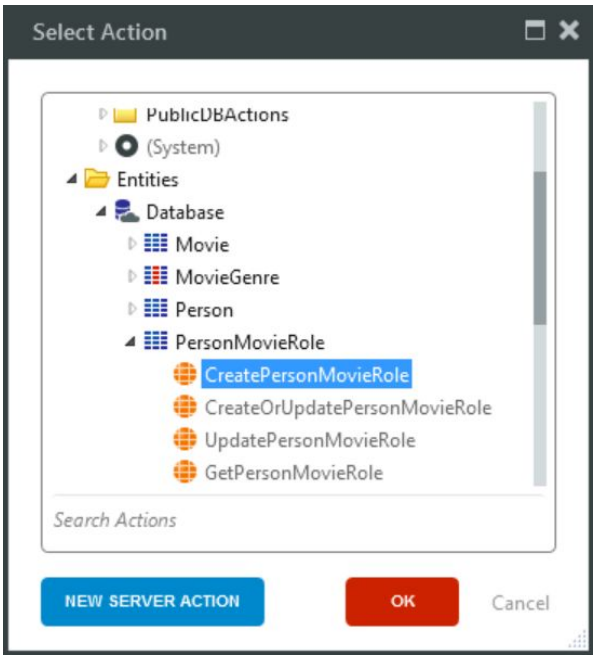
These assignments make sure that the Record to be added to the database, PersonMovieRole has all the information filled in.



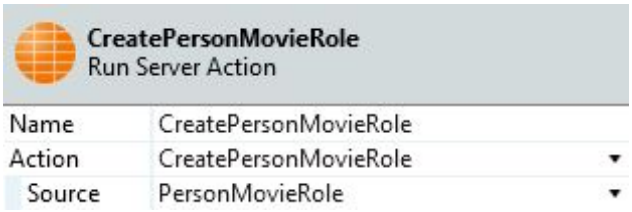
d) Drag and drop a **Run Server Action** statement below the Assign.



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



f) In the **Source** property of the Action, select the *PersonMovieRole* Local Variable.

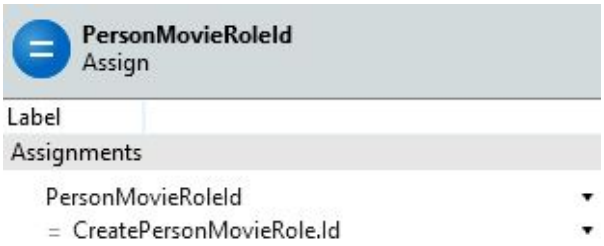


g) Drag another Assign and drop it after the **CreatePersonMovieRole** Action.

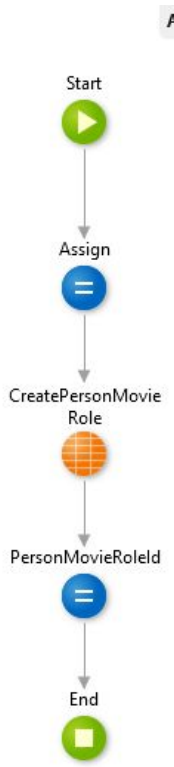
h) Set the assignment to:

PersonMovieRoleId = *CreatePersonMovieRole.Id*

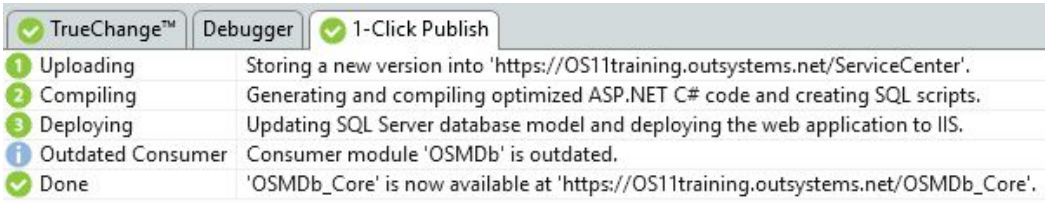
This assigns the *Id* of the PersonMovieRole created to the Output Parameter.



i) The Action should look like the following screenshot



j) Publish the **OSMDb_Core** module to the server



Ignore the outdated message for now. We will address this later in this lab.

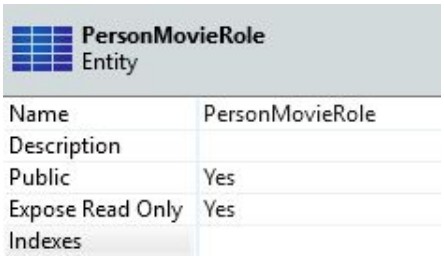
Make the PersonMovieRole Entity Read-only

Now that the Action was created, we can continue implementing the pattern by exposing the PersonMovieRole Entity as Read-only. This way, every consumer module of this Entity will only have access to the **GetPersonMovieRole** Entity Action.

Since we are changing the producer module, the consumer (OSMDb) will be outdated. So, we need to go back to the OSMDb module and refresh the dependencies to the Entity, as well as add the new dependency to the **AddPersonToMovie** Action.

Then, we can use this new Action to add participants to a movie.

1) Change the **Expose Read Only** property of the PersonMovieRole Entity to *Yes* and publish the OSMDb_Core module.

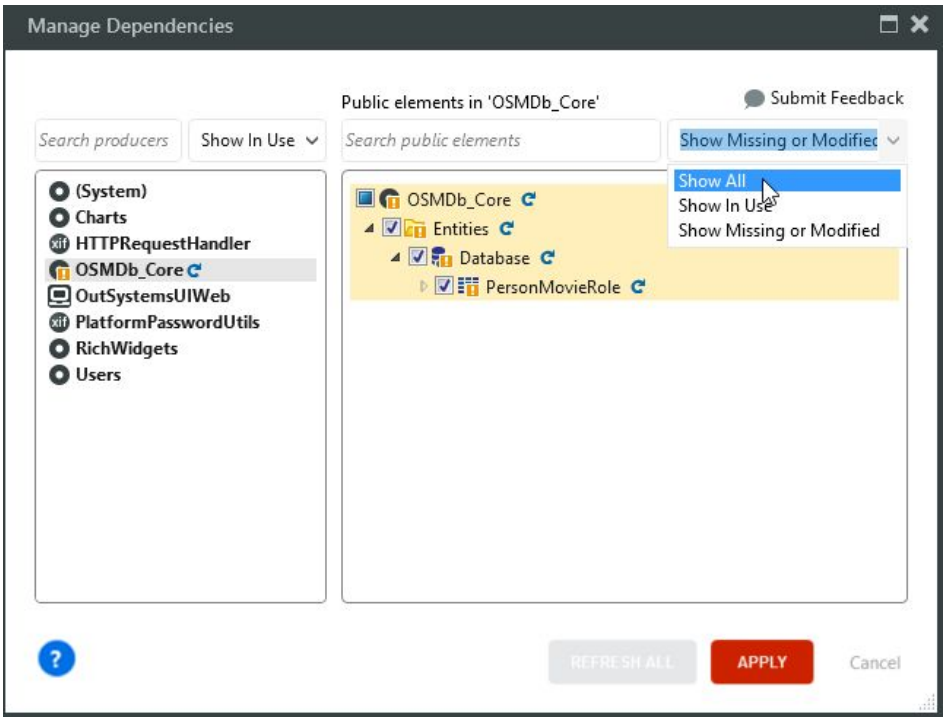


2) Now that the Entity is exposed as read-only, we need to go to the OSMDb module, refresh the dependencies and add the new ones (**AddPersonToMovie** Action). This will cause some errors that we need to fix, since the CreatePersonMovieRole Action is no longer available in the OSMDb module.

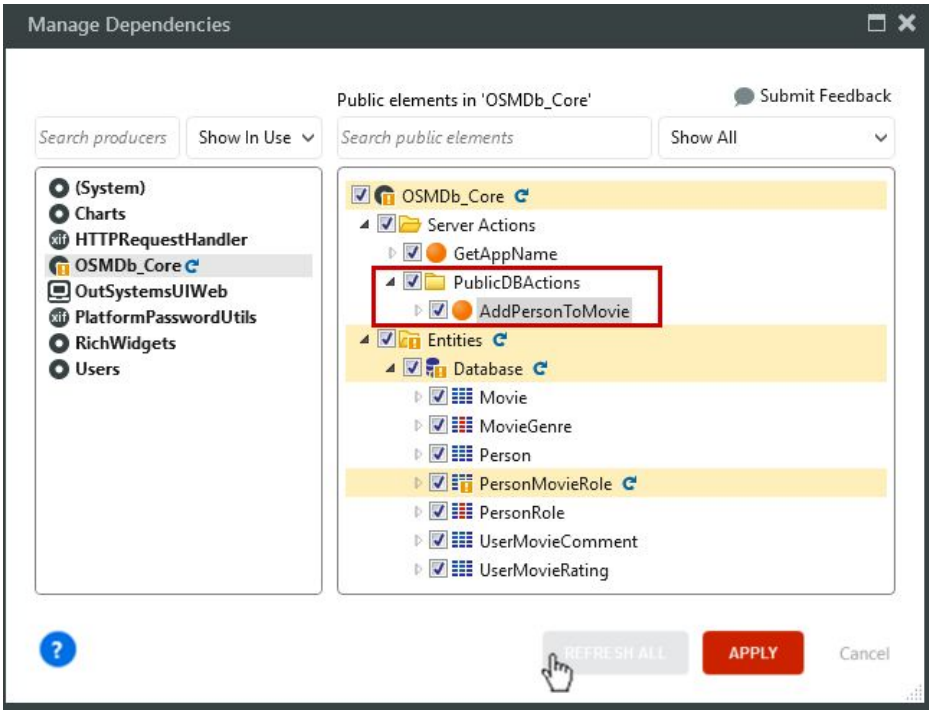
    a) Open the OSMDb module.

    b) Open the Manage Dependencies window.

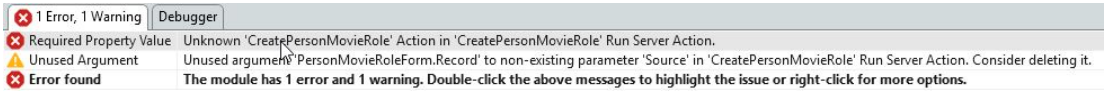
c) Select the OSMDb_Core on the Producers section on the left. Then, on the Public elements on the right, select the Show All option in the dropdown.



d) Select the **AddPersonToMovie** Action and **Refresh All** dependencies. Click **Apply** to exit. This will cause one error that we need to fix.

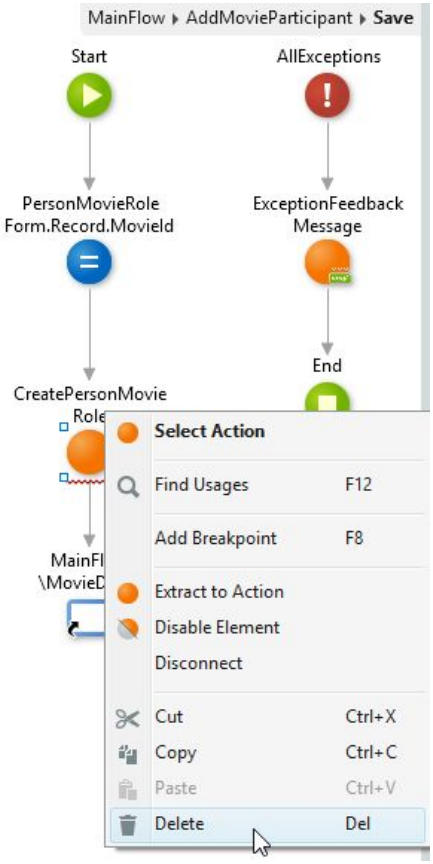


e) Double-click on the error to open its location in the module.

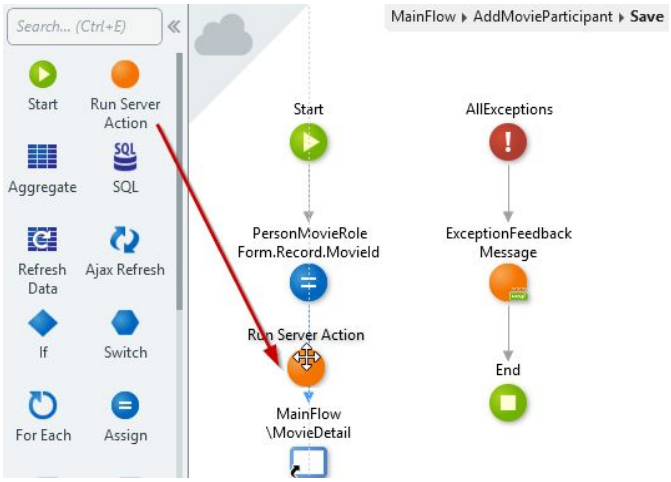


f) Since the PersonMovieRole Entity is Read-only, we don't have access to its Create Entity Action in the consumer module. So, in the Save Action of the **AddMovieParticipant** Screen we have the error that this Action is unknown. For that reason, we can delete the call to the **CreatePersonMovieRole** Action.

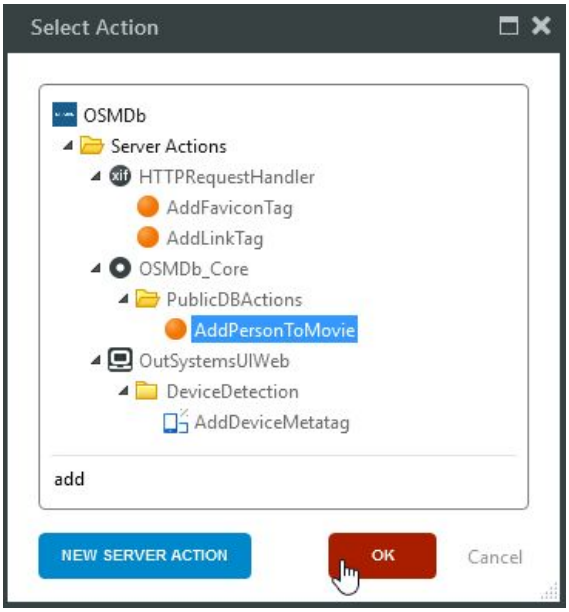
g) Delete the Action from the flow.



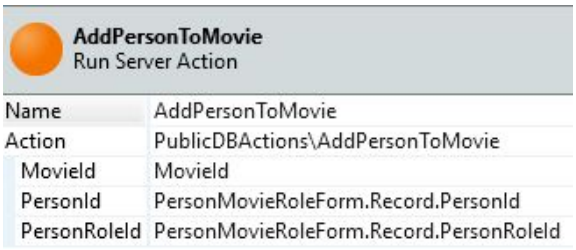
h) Drag a new Run Server Action statement below the Assign and before the End.



i) In the Select Action dialog, select the AddPersonToMovie Action.

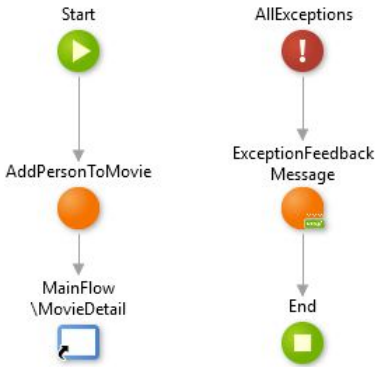


j) Set the Inputs of the Action as in the following screenshot. This guarantees that the *MovieId* passed as Input of the Screen, and the data submitted by the user in the Form is passed to the **AddPersonToMovie** Action.



k) The Assign in the Save Action is not needed anymore, since it is done inside the AddPersonToMovie Action. So, let's delete it.

l) The Action should look like this:



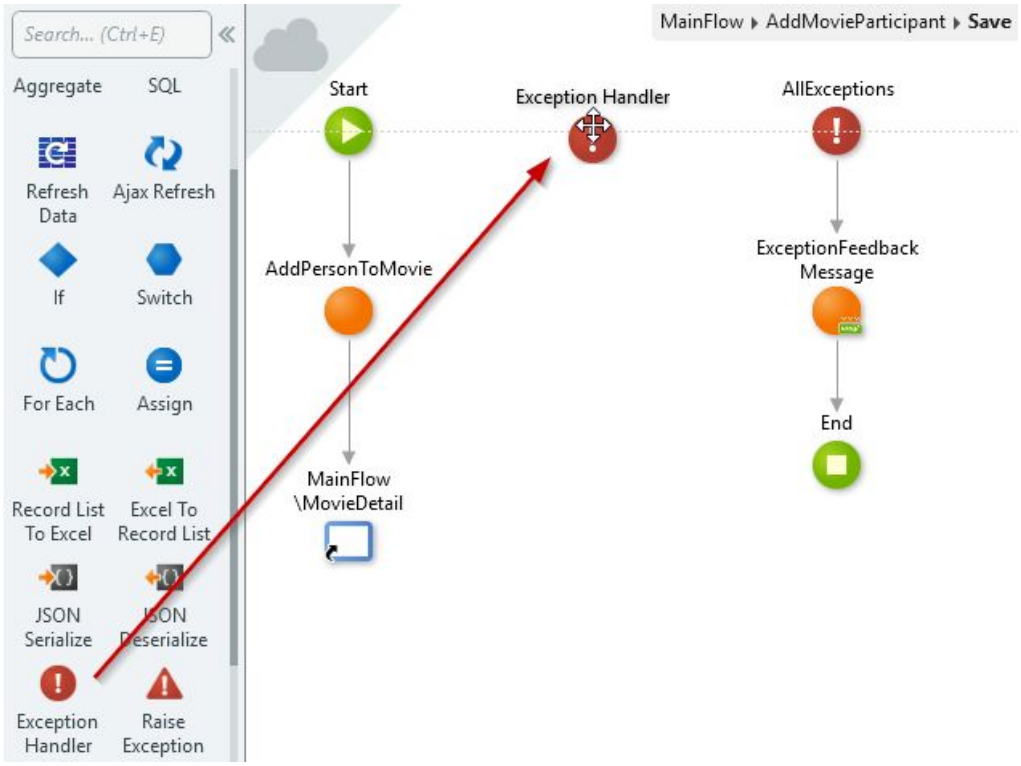
m) Publish the module and make sure that the application still works, by assigning a new Role to a Person in a Movie.

Handle Database Exceptions

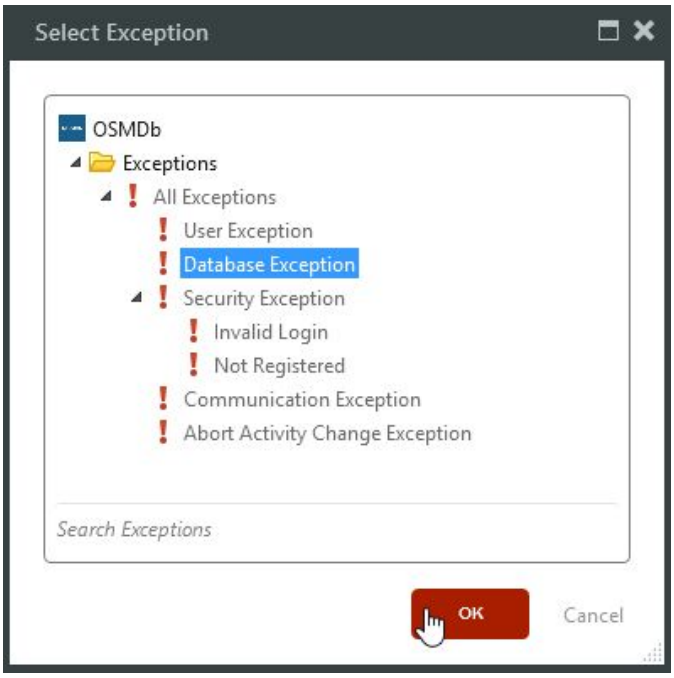
In a previous lab, we could see a red Feedback Message appearing, when we were trying to add a new PersonMovieRole, without having all the mandatory inputs filled. That happens because the logic is still trying to add the record to the database, even if those inputs are missing. Let's make that message more clear to the end user, by providing a more meaningful (or at least readable) message. We will do that, by creating an exception handler to the Database Exception, inside the **Save** Action of the **AddMovieParticipant** Screen..

1) Open the Save Action of the AddMovieParticipant Screen.

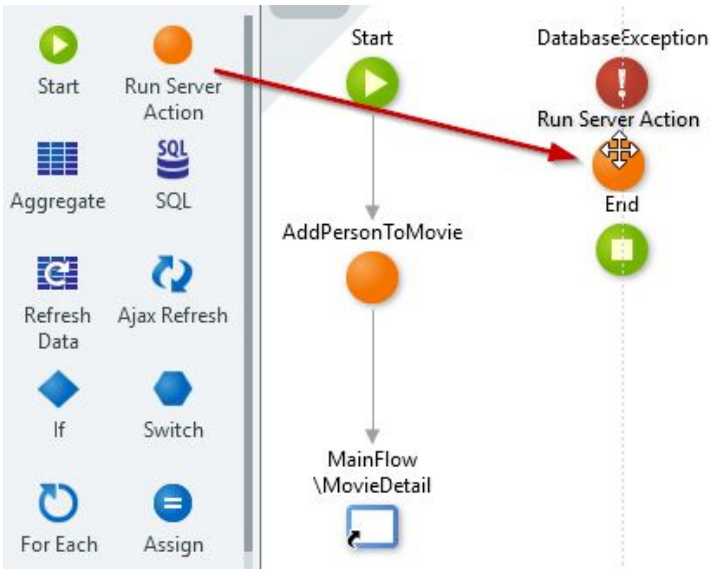
2) Drag and drop a new Exception Handler statement next to the main flow of the Action.



3) In the Select Exception dialog, select the **Database Exception**.



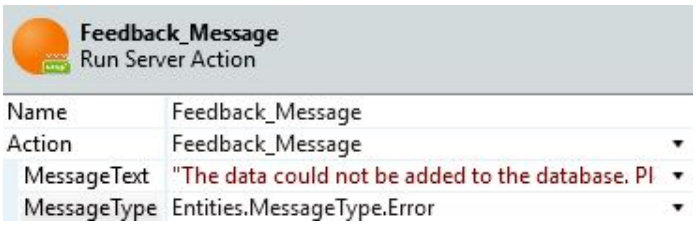
4) Drag and drop a **Run Server Action** statement to the new Exception Handler flow.



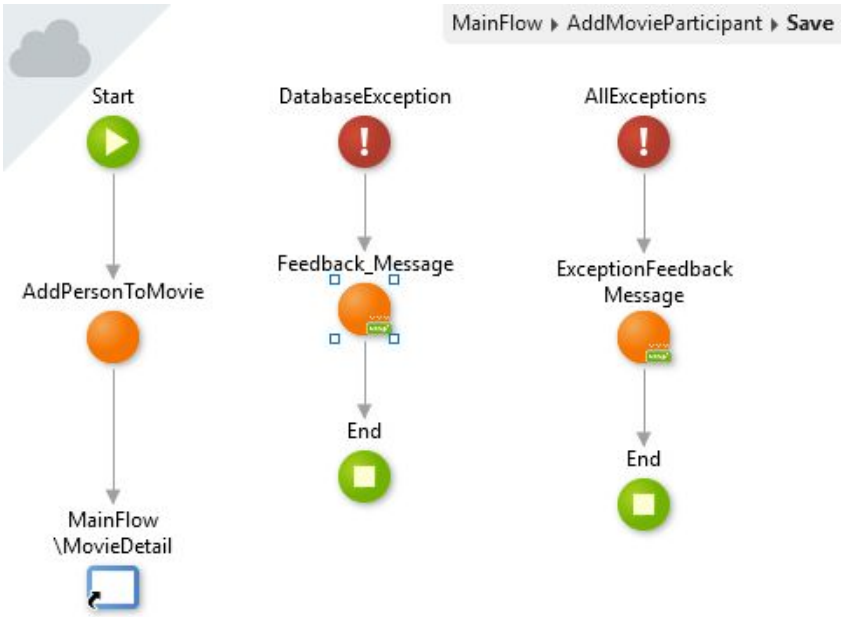
5) In the Select Action dialog, select the **Feedback_Message** under the RichWidgets module.

6) Set the Feedback_Message properties as the following:

MessageText: *"The data could not be added to the database. Please make sure all fields are properly filled"*
MessageType: *Entities.MessageType.Error*



7) The Save Action should look like this:



(Optional) Add Actions to Create or Update Movies and People

In the previous sections we implemented a pattern that is best practice in OutSystems, by exposing an Entity as Read-only, and then create Server Actions to expose functionality to change the Entity data in consumer modules.

This is also valid for the remaining Entities of the data model, which can lead to something similar to an API, with multiple Actions providing functionalities to the consumer modules, while hiding others. For instance, with the PersonMovieRole Entity exposed with write permissions, it would be possible in the consumer module to delete a Record from the Entity. This way, it is not possible.

As an optional challenge for this Lab, let's create Server Actions for Creating or Updating Movies and People in the Database, following the same strategy, with any additional logic that makes sense. Don't forget to refresh the dependencies and replace the usages of the Entity Actions by the new ones.

NOTE: The following labs will continue with the premise of this part not being completed.

End of Lab

In this exercise, we created a reusable Server Action that added participants to a movie, with a certain role. This Action was created in the OSMDb_Core and used the CreatePersonMovieRole Entity Action to add a new record to the database.

We also changed the PersonMovieRole Entity to be exposed as Read-only, to make sure that not all the Entity Actions were available to any of the consumer modules.

Then, we refreshed the dependencies in the OSMDb module, to use the new Action and to get the new version of the Entity, and adjusted the logic to use the new Server Action.

Finally, we created an Exception handler for Database Exceptions in the Save Action of the AddMovieParticipant Screen.

As an optional exercise, we could also use the same pattern for adding / updating people and movies to the database, to make sure our application followed OutSystems Best Practices.