Creating a Web Application
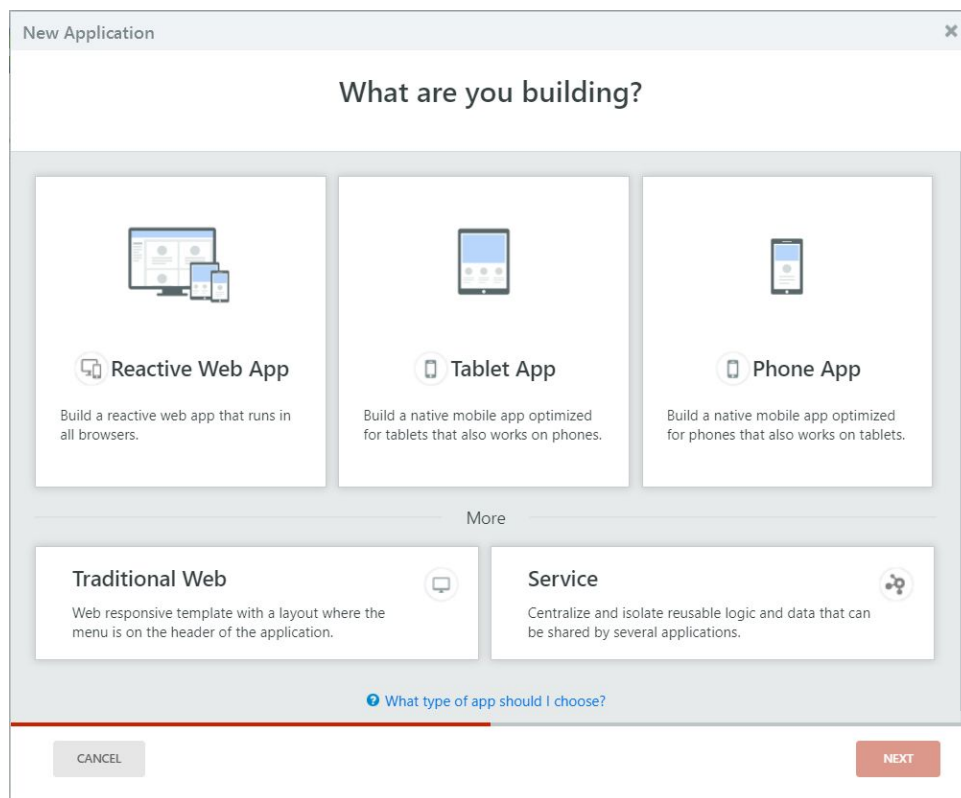
Table of Contents

Introduction

This is the first of a set of several exercise labs that will lead you on how to progressively build a web application in OutSystems. The application will focus on creating and maintaining a movie database, as well as the people involved in the film (cast and crew). Movies will have information such as title, plot summary and genre. People will have biographical information such as name and date of birth, as well as connection to the movies they worked on and in what capacity.

Users of the movie application will be able to access all of this information, but also comment on movies and rate them. These ratings will be aggregated for other users' reference, meaning that the other users can see the average ratings and all the comments about a movie.

Administrators of the movie application will be able to add new (or edit existing) movies, as also actors or members of the crew and associate them with a movie.

We will constantly be expanding, publishing and testing the application, while learning newer and more advanced OutSystems concepts in the process.

At the end of this set of exercise labs, we will have a small, but perfectly formed application, spanning multiple Screens and concepts that we can easily access from a browser.

In this specific exercise lab, we will create the web application with two modules: a traditional web module that will hold all the UI and business logic, and a Blank Core module that will hold the data model.

To test the application, we will create a scenario where the user will see a "Hello World" message when it opens the application in the browser. This scenario has two purposes:

- Reference an element from the Core module. For this purpose, we will create a Public Action in the Core module, that returns the name of the application.
- Display the result of the Action in a Screen of the UI module, so that it is visible on the browser when the user opens the application.

Create a web traditional web application with two modules

We will start by creating the Movie Database application (which we will call *OSMDb*). The *OSMDb* application will have two modules: the Core module for the data and a Traditional Web module for all the UI and business logic.

Let's start by creating the OSMDb application with a Traditional Web module.

To develop any OutSystems application, we need the OutSystems Development Environment, **Service Studio**, and an OutSystems server (or **environment**).

1) Open Service Studio and login in your personal environment (when following the online class) or the bootcamp environment (when following the classroom training).

   a) Open Service Studio from the Start Menu or by double clicking the icon.

   OutSystems Service Studio 11.0

   b) In the **Connect to Environment** or **Switch Environment** dialog, enter the environment address, username and password you will be using to carry out the exercises, and click **Connect**.
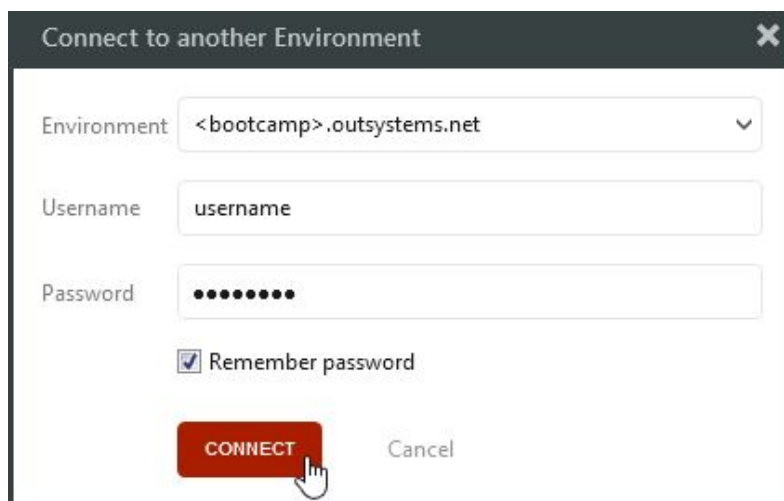

2) Create a Web application called *OSMDb_<your_Initials>* with a Traditional Web module of the same name. Use the **Traditional Web** application template and the

osmdb-icon.png file for the application icon, the, which can be found in the Resources folder.

a) In the Applications tab, select **New Application.**

b) Now we can choose from the options Web, Tablet, Phone, Traditional, and Service app. Select **Traditional Web App**, and click **Next**.
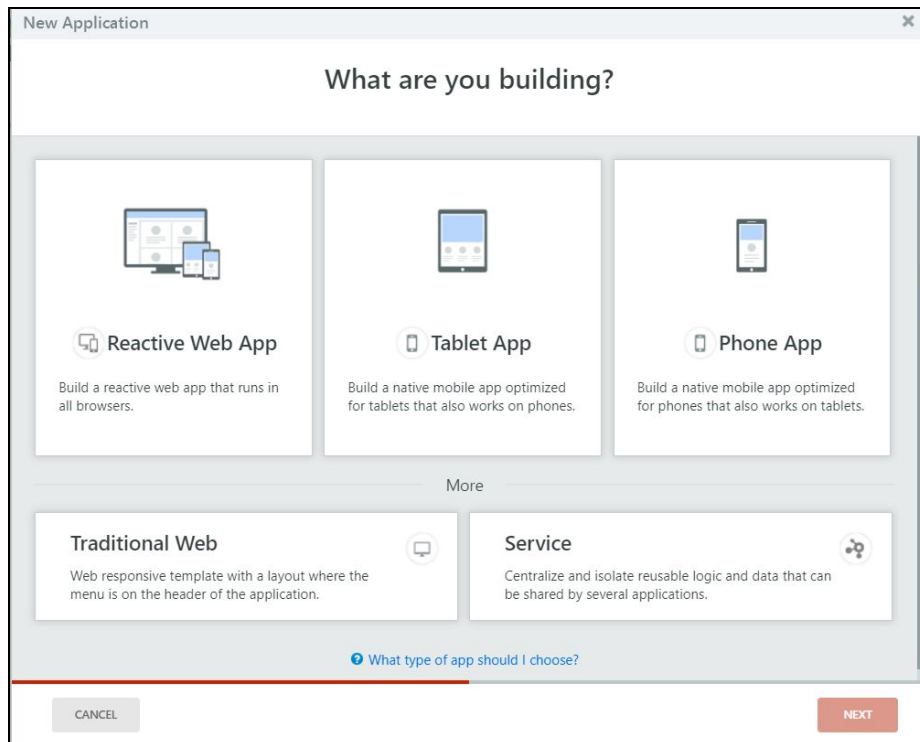
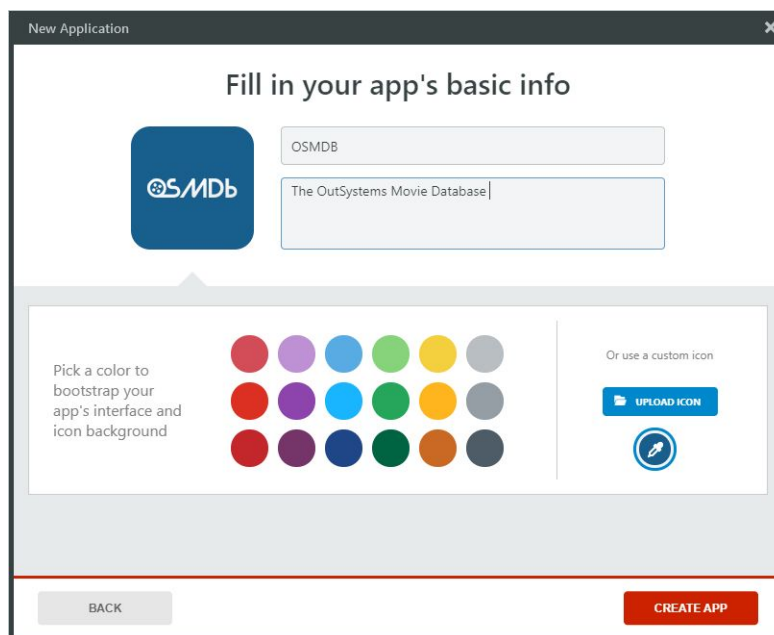c) In the next dialog, change the application's name to *OSMDb_<your_initials>.*

NOTE: From now on, all screenshots or references to the app will use the *OSMDb* name, for simplicity.

d) Click the **Upload Icon** button, select the **osmdb-icon.png** file from the Resources folder, and then click **Open** to select the image. This will be the icon of the OSMDb application.

e) Type in a simple description for the application, and click **Create App**.

f) In the **MODULES** area, the list of modules of the OSMDb application can be found. Initially, a suggestion is made for the name of the first module, which is the name of the app. Make sure the module is **Traditional Web** and click the **Create Module** button.
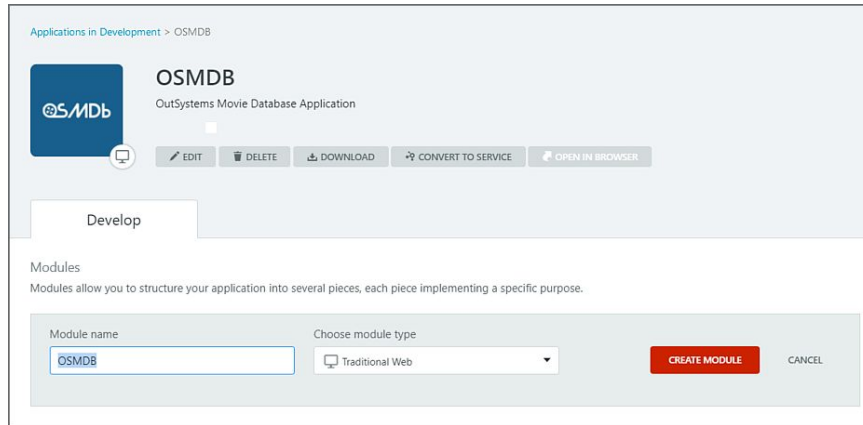
3) Create the *OSMDb_Core<your_initials>* module, the second module of the application. This should be a **Blank** module, without any UI. Don't forget to add your initials in front of the module's name, specially if you are following a classroom training.
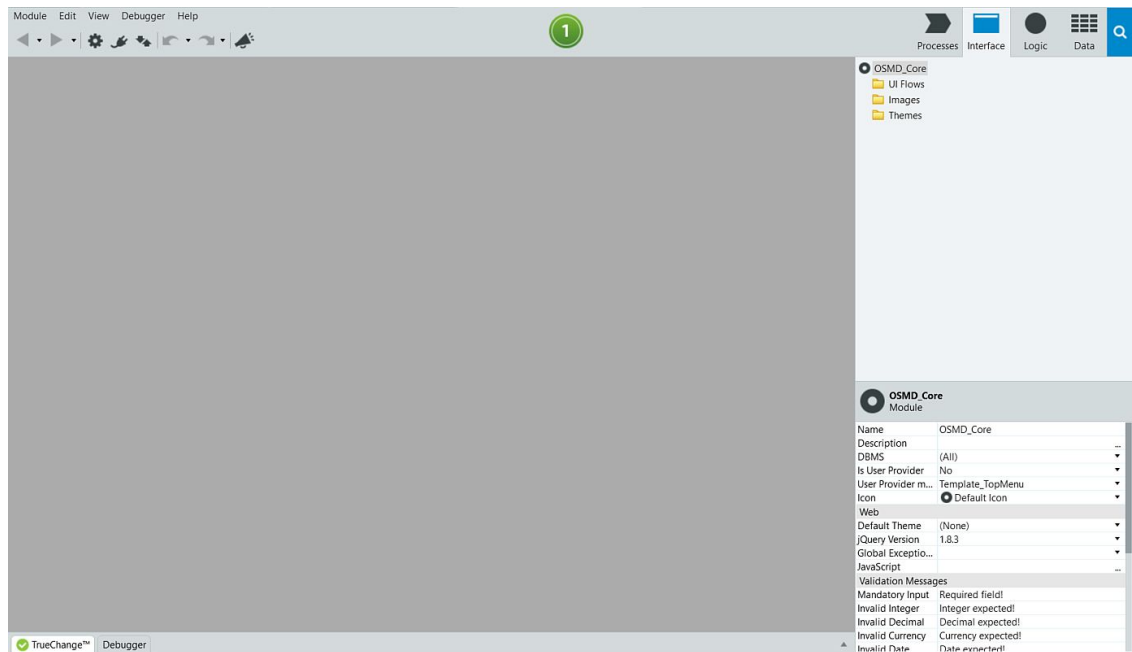
    a) Go back to the Application tab.
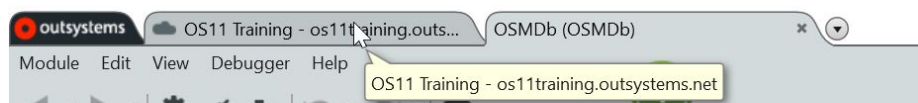
    b) Click on the **Add Module** button to create a new module.
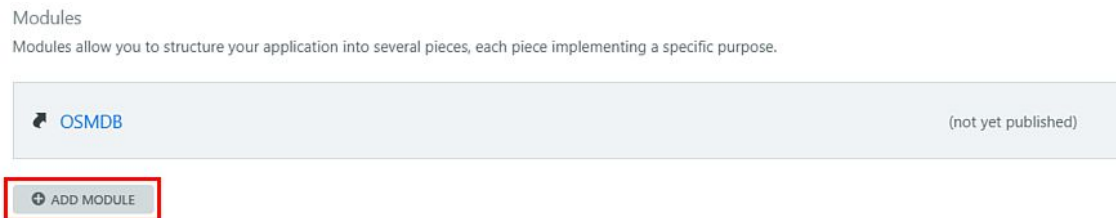
    c) Call it *OSMDb_Core_<your_initials>*.

d) Select the **Blank** module option. Click on the **Create Module** button.

NOTE: If you see something visual in the workspace, it is because you created a **Traditional Web** module. In that case, you should delete the module and create a new **Blank** module.

Publish the application modules

In this section, we will publish the modules in the server for the first time. This will create the first version of the modules in the application server. From now on, every new publish creates a new version of the module.

The publishing process uploads the module's information to the server. Then, it proceeds to generate and compile the necessary code and create the required database scripts. Finally, it uses the scripts to update the database and then deploys the application to the server.

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

2) Notice the **1-Click Publish tab** that appears near the bottom. This tab provides progress updates on the publishing process.

NOTE: Once published, module Screens become available at the URL displayed in the step **Done**. In general, that URL will be *https://hostname/ModuleName*

Since the current module has no UI, you won't be able to navigate to it using your browser, thus the 1-Click Publish will still be green after publishing.

3) Open the *OSMDb (OSMDb) tab* and publish the **OSMDb** module to the server.

Display a "Hello from the OSMDb app" message

Now that we have the application created, with a Traditional Web and a Blank module, we want to display a simple message to the end-user that opens the application in the browser.

We will create a simple **Action** that returns the name of the application, **in the Core module**, and then will display it next to a "Hello World" message in the Home Screen of the application. This section will require an element from one module to be used in a different module. For that, we need to create a dependency between the two modules.

A lot of the steps we are doing is to help us navigate and get familiar with Service Studio. All of these concepts will be tackled in detail in the further lessons and labs, which will help clarify all the steps being done.

1) In the Core module, create a *GetAppName* Server Action that returns the application's name (*OSMDb*). The *GetAppName* Server Action should be **Public**, so it can be reused in other modules, and a **Function** so it can be used in Expressions.

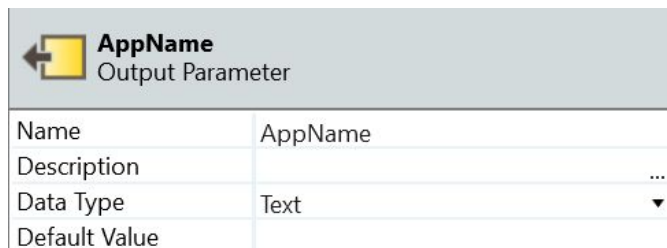   a) Switch to the OSMDb_Core module and open the **Logic** tab.
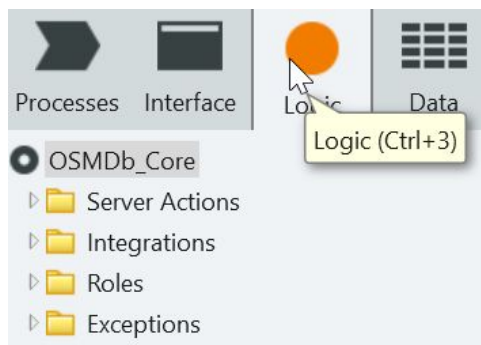
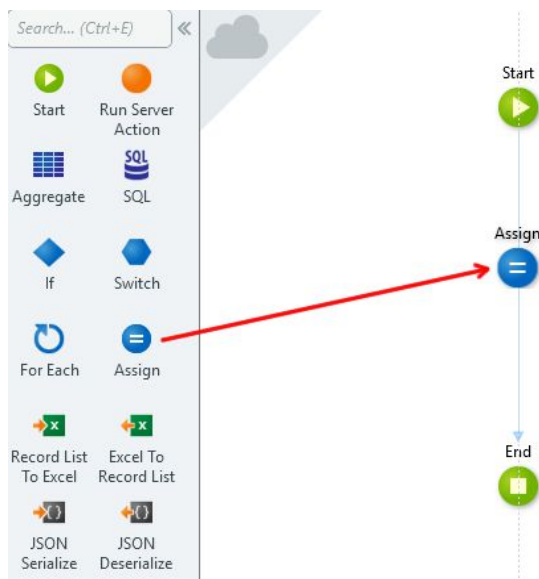   b) Right-click the **Server Actions** folder and select **Add Server Action**.
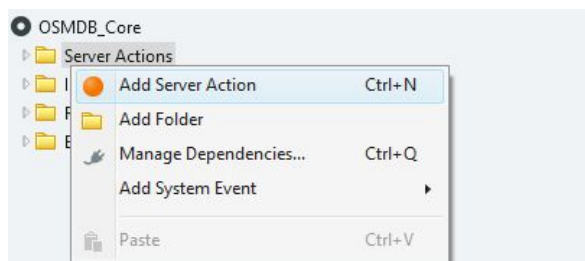
   c) Type in *GetAppName* to change the **Name** of the Action.

   d) Right-click the **GetAppName** Action and select **Add Output Parameter**.
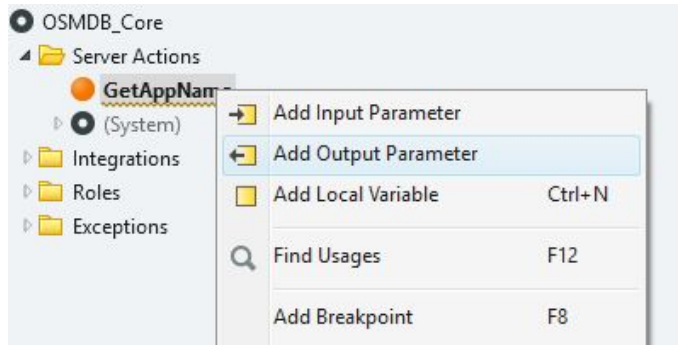
e) Set the **Name** property of the Output Parameter to *AppName*.

f) Drag an **Assign** from the toolbox and drop it on the Action flow between the Start and End nodes.

g) In the Assign properties area, select the **AppName** Parameter using the dropdown on the top of the Assignment. Then, on the bottom, just type *"OSMdb"*, without forgetting the quotes. This sets the **AppName** value to *"OSMDb"*.
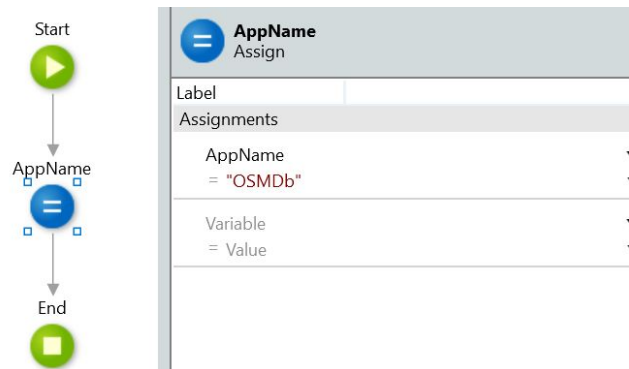
h) In the Logic tab, select the **GetAppName** Action in order to change its properties, by clicking on it once.
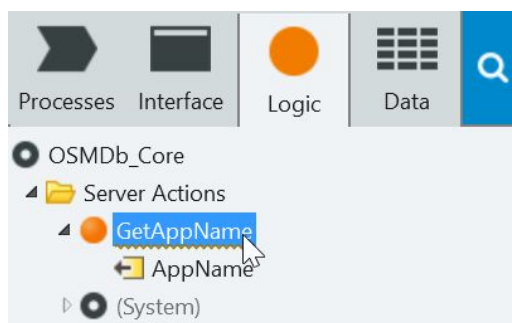
i) In the properties area of the Action, set the **Public** property to *Yes* and the **Function** property to *Yes*.
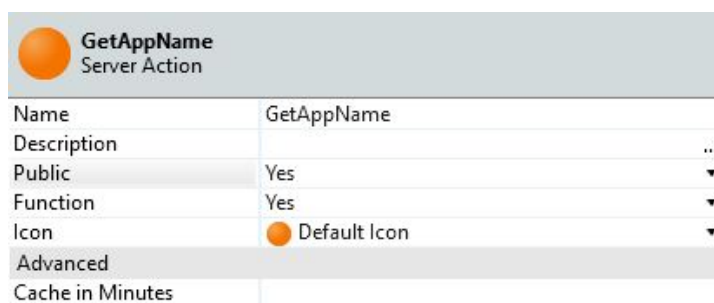
NOTE: By setting the **Function** property to *Yes*, enables the **GetAppName** Action to be used in an Expression, to be evaluated at runtime. A Function cannot have more than one Output Parameter.

2) Publish the OSMDb_Core module to the server.

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

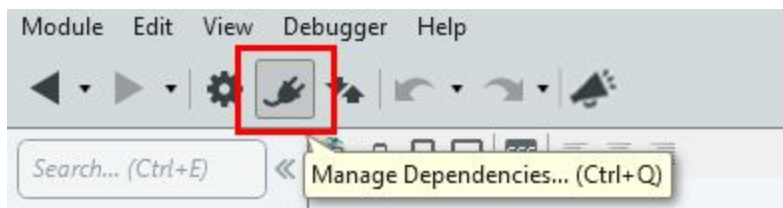b) In the 1-Click Publish tab, you should see something similar to this

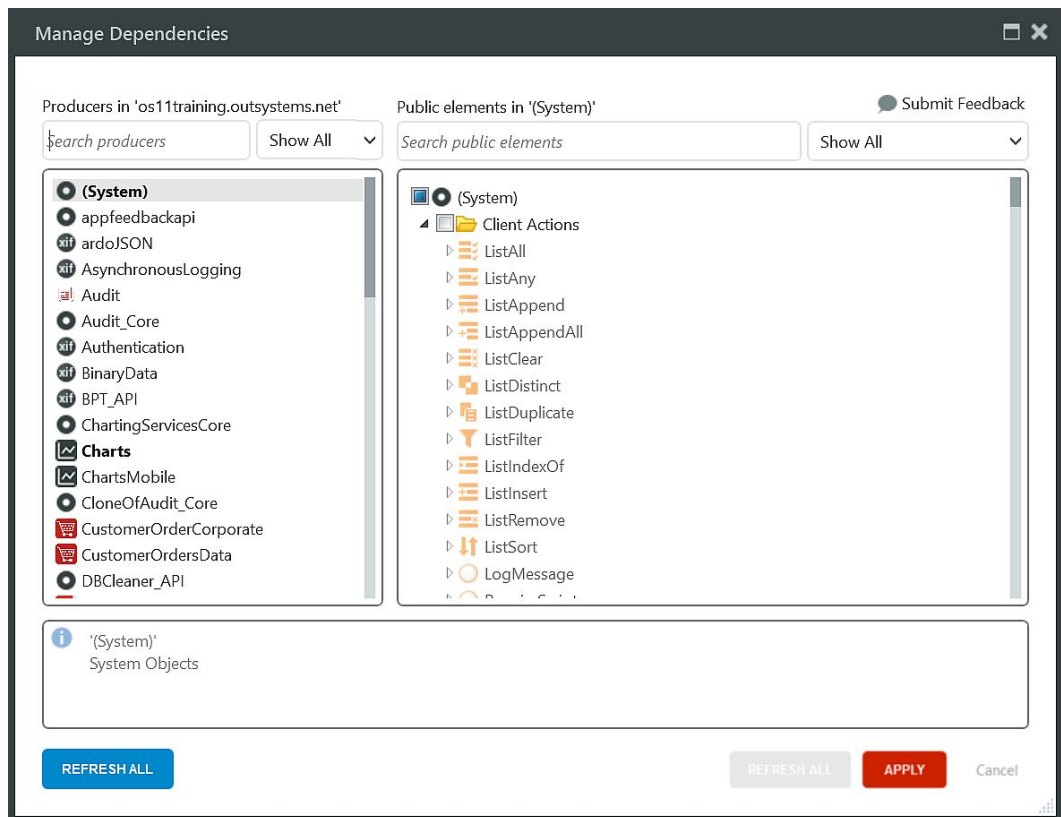3) Reference the **GetAppName** Action in the **OSMDb** Traditional Web module.

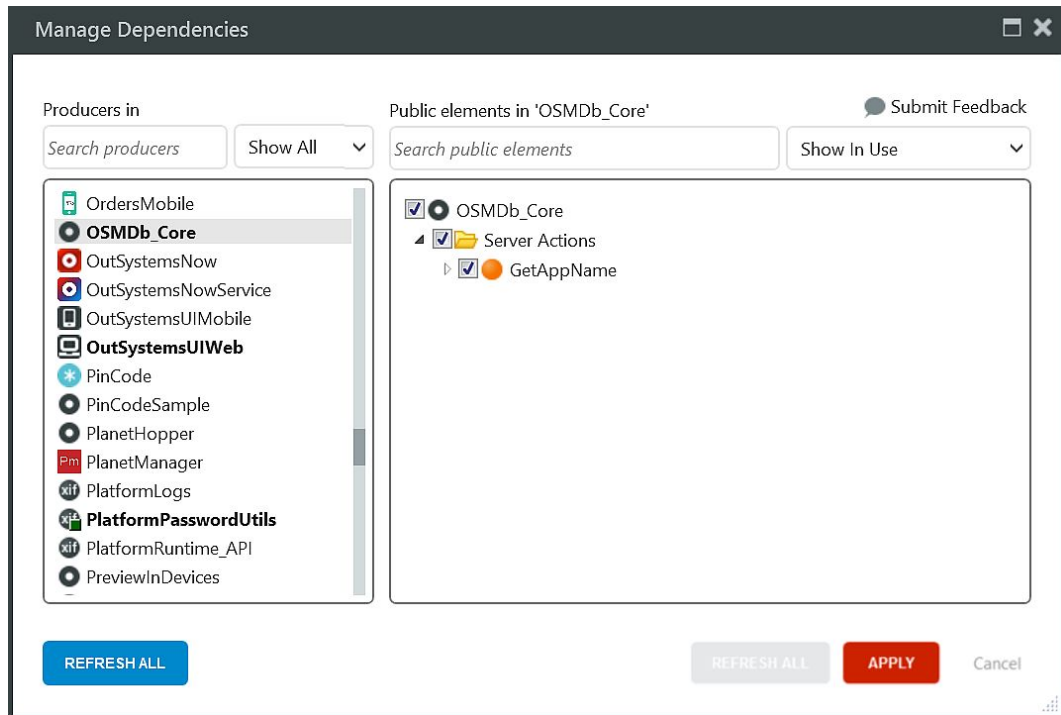a) Open the **OSMDb** module by clicking on the OSMDb (OSMDb) tab.

b) Click on the **Manage Dependencies** icon at the top of your screen.

c) You should now see a dialog with all the modules that have Public elements.
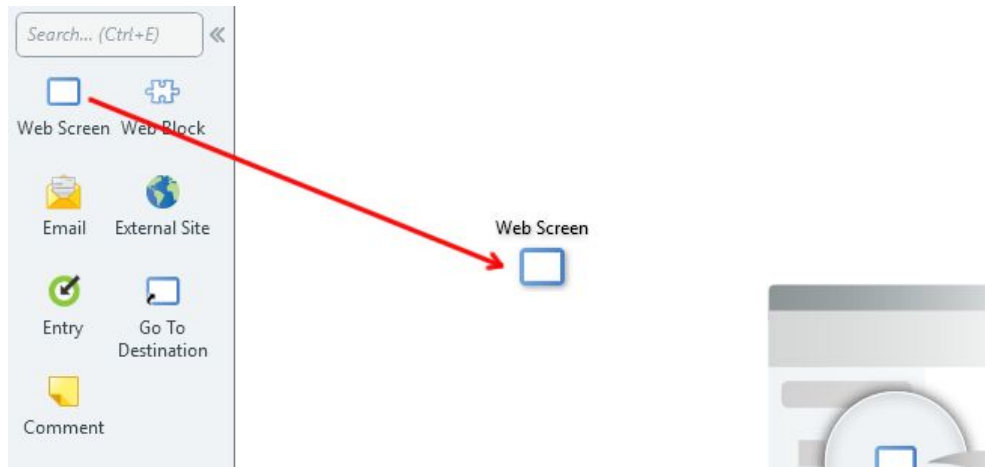
d) Select the **OSMDb_Core** module on the left, and then select the **GetAppName** Server Action on the right. Click **Apply** to confirm.
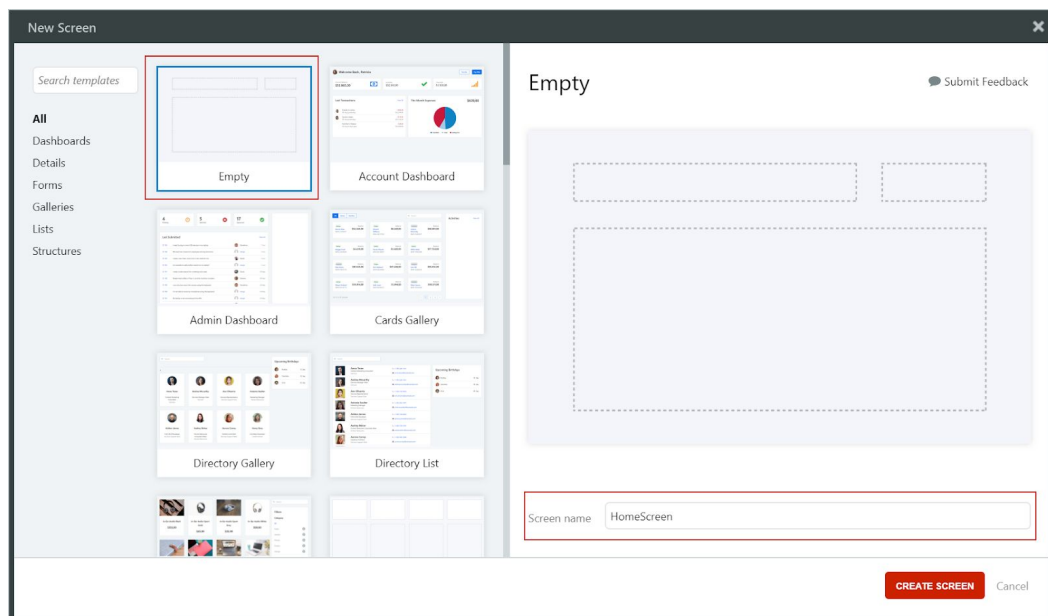

4) Create a new Screen called *HomeScreen* using the **Empty** screen template and make it **Anonymous**.

a) In the Interface tab, double-click the **MainFlow** UI Flow to open it in the canvas.
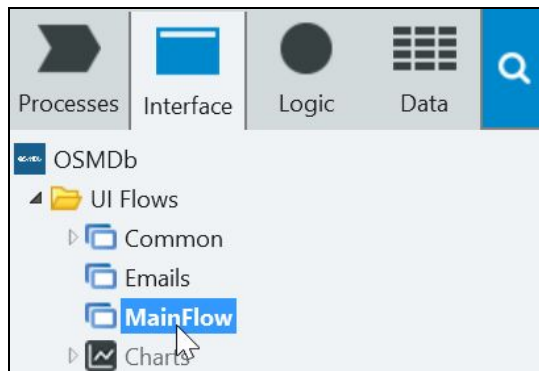
b) Drag a **Web Screen** to the workspace.

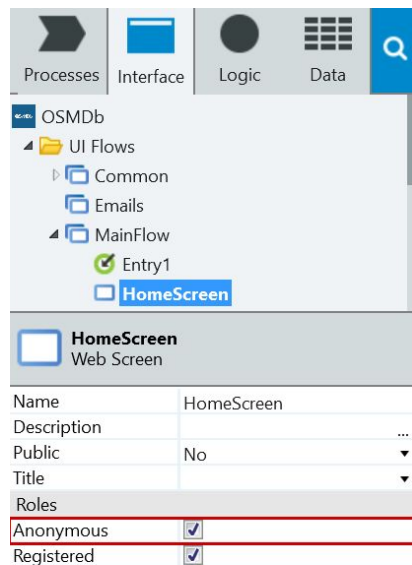

c) In the new window, select the **Empty** Template and name the Screen as *HomeScreen.* Click on the **Create Screen** button.


d) On the right, in the elements area, we can find the HomeScreen under the **MainFlow**.

e) Tick the **Anonymous** Role checkbox, to allow every user to access the Screen, even without a login.

5) Add an **Expression** to the HomeScreen to display *"Hello from the OSMDb app"*. The **GetAppName** Action can help us with the name of the app in this step.

a) Drag and drop an **Expression** from the toolbox to the main content area.
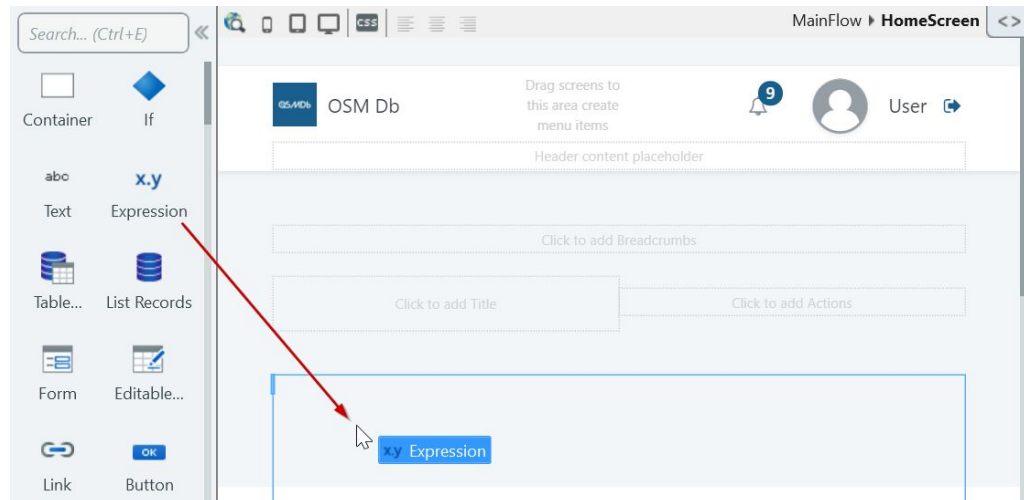
b) Set the Expression to *"Hello from the "* + *GetAppName()* + *" app."*

6) Publish the OSMDb module and see the application in the browser.

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

b) After it's done, we can see that the ① 1-Click Publish button has changed. This is the 🌐 Open in Browser button. This button appears every time a publish was successful, on modules with UI already defined.

c) Click on it to open your application in your default browser. You should see something like this

End of Lab

In this exercise lab, we created a web application to manage movies and its cast and crew, OSMDb. Since this is the first lab, we just created the web app in Service Studio and two modules: one Traditional Web, to hold all the UI of the application, and one Blank, to hold the data model of the application.

After creating these modules, we published them for the first time to the server.

To test the app in the browser for the first time, we created an Action that returns the name of the app, which is later used to display it in the HomeScreen of the application. Since the Action was created in the Blank module, and the Screen is present in the Traditional Web module, we created a dependency between the two modules.