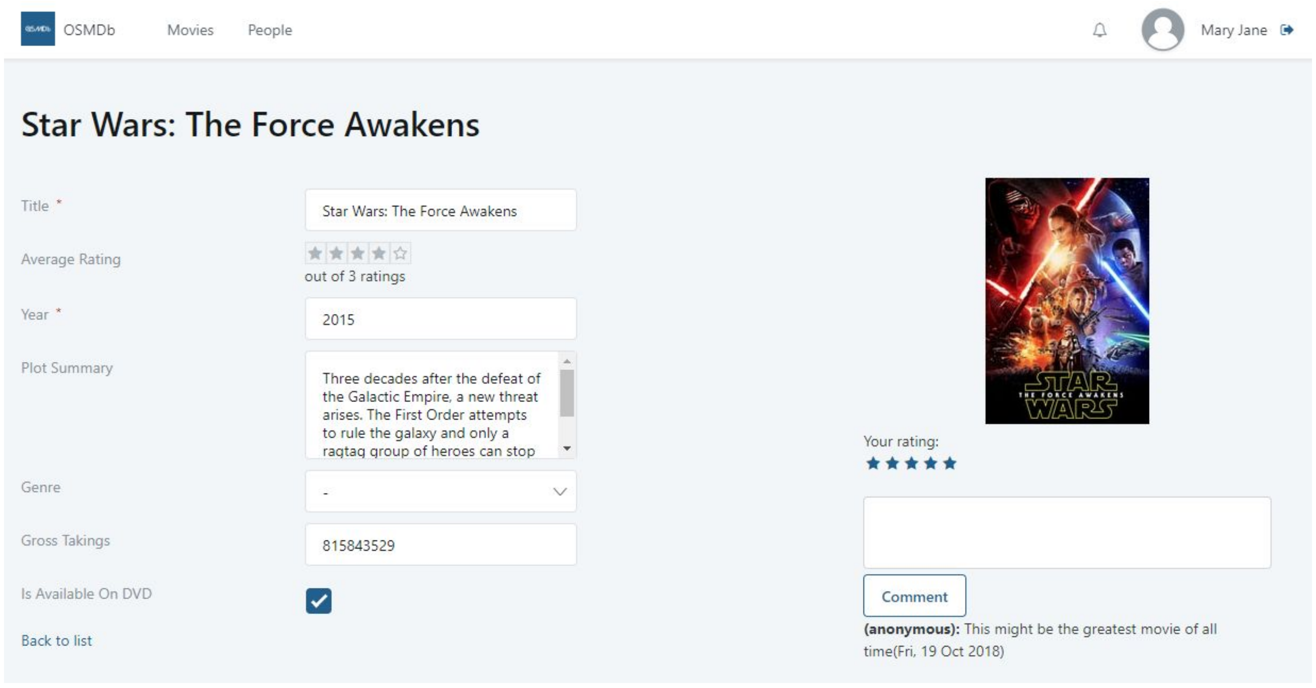
Web Services Exercise



OSMDb · Movies · People · Mary Jane

Star Wars: The Force Awakens

Title *	Star Wars: The Force Awakens
Average Rating	out of 3 ratings
Year *	2015
Plot Summary	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
Genre	-
Gross Takings	815843529
Is Available On DVD	✓

Back to list

Your rating:

Comment

(anonymous): This might be the greatest movie of all time(Fri, 19 Oct 2018)

Table of Contents

Introduction

OutSystems allows easy consumption and exposure of SOAP Web Services and REST APIs. In this exercise lab, we will use a method from a REST API. This method returns a movie poster that will be displayed in the MovieDetail Screen, alongside the remaining movie information.

When consuming a REST API (or just a method), OutSystems allows the methods to be used just like any regular Action, with drag and drop to a Preparation or Action flow. This makes it easier to use them and integrate them with our application.

Also, OutSystems offers two callbacks that we can use to manipulate the request and the response of the web service. In this lab, we will manipulate the response of the web service, to handle errors that may appear.

In this exercise lab, we will:

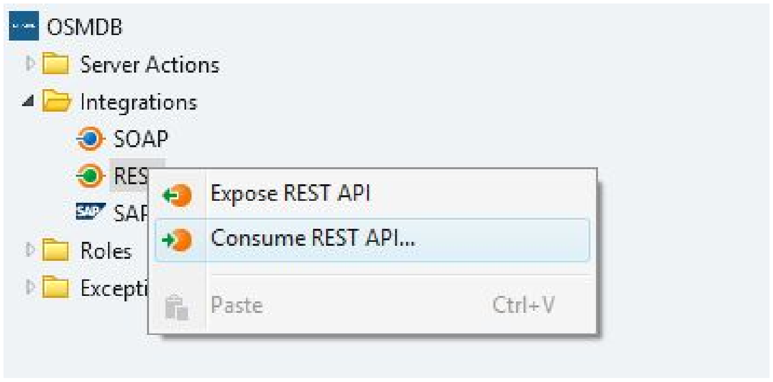
- Learn how to integrate with an existing REST API, using OutSystems.
- Execute a REST API method and modify a Screen to show the results.

Integrate with a REST API method

We will start by consuming a REST API that retrieves the information about a movie, based on a search term. Specifically speaking, we will use The Movie Database API to obtain movies by title, and from the result, extract the movie poster image URL. The poster will later be added to the **MovieDetail** Screen. This will be done in the **OSMDb** module.

1) Consume **The Movie Database** REST API (*http://api.themoviedb.org*) method that obtains the movie information based on the movie title. This is the */search* method of the API. To do so, make sure you have a look at the API documentation to find out the information the method is expecting.

    a) Switch to the Logic tab and in the Integrations folder, right click on the REST integration and select **Consume REST API...**. This opens a window to configure the REST methods that we want to consume.

    b) On the new window, click on the **Add Single Method** button, since we are just going to use one method from the The Movie Database REST API.

    NOTE: If the REST API has a Swagger definition you can select Add All Methods. This option will add automatically all methods of the REST API exposed.

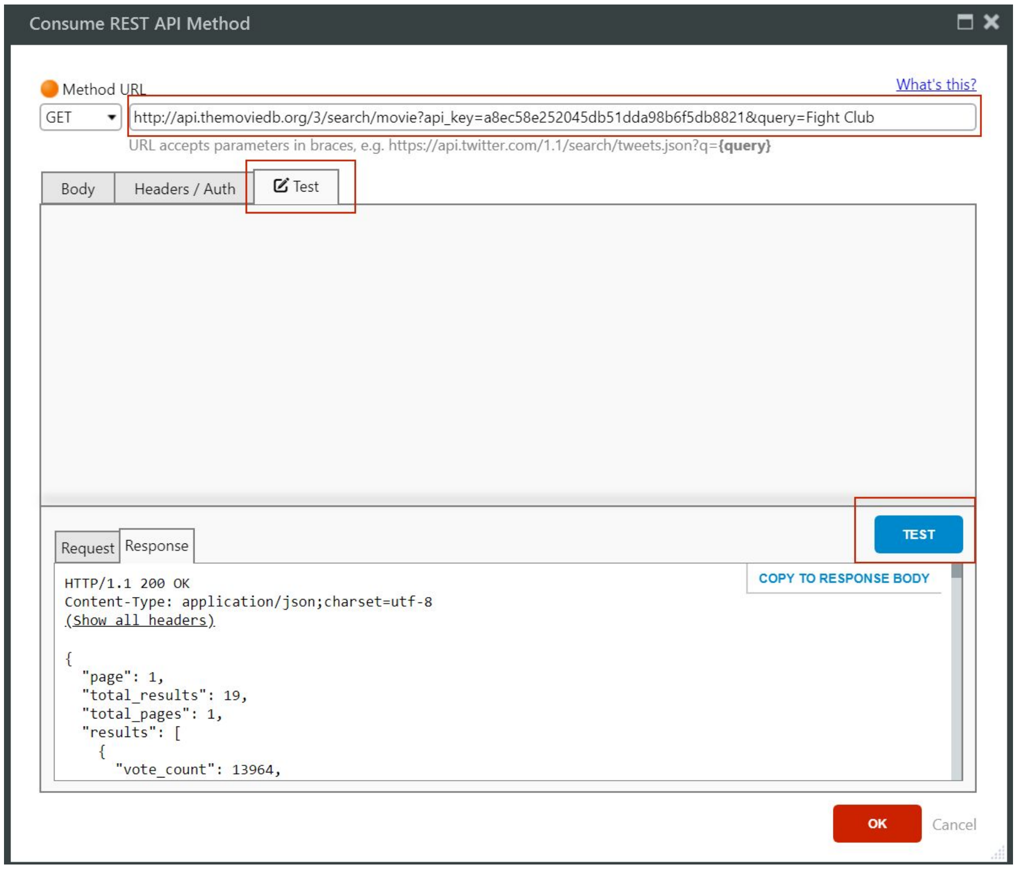
    c) Set the **Method URL** to:

    http://api.themoviedb.org/3/search/movie?api_key=a8ec58e252045db51dda98b6f5db8821&query=Fight Club

    At this step, we always need to put the URL of the method. You can find this in the API Documentation available here.

    NOTE: You can check for more information about *The Movie Database* API, in the following link: https://www.themoviedb.org/documentation/api

d) Select the **Test** tab and then click on the **TEST** button. This will test the call to the API method we are trying to consume.



http://api.themoviedb.org/3/search/movie?api_key=a8ec58e252045db51dda98b6f5db8821&query=Fight Club

URL accepts parameters in braces, e.g. https://api.twitter.com/1.1/search/tweets.json?q=**{query}**

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
(Show all headers)

{
  "page": 1,
  "total_results": 19,
  "total_pages": 1,
  "results": [
    {
      "vote_count": 13964,
```

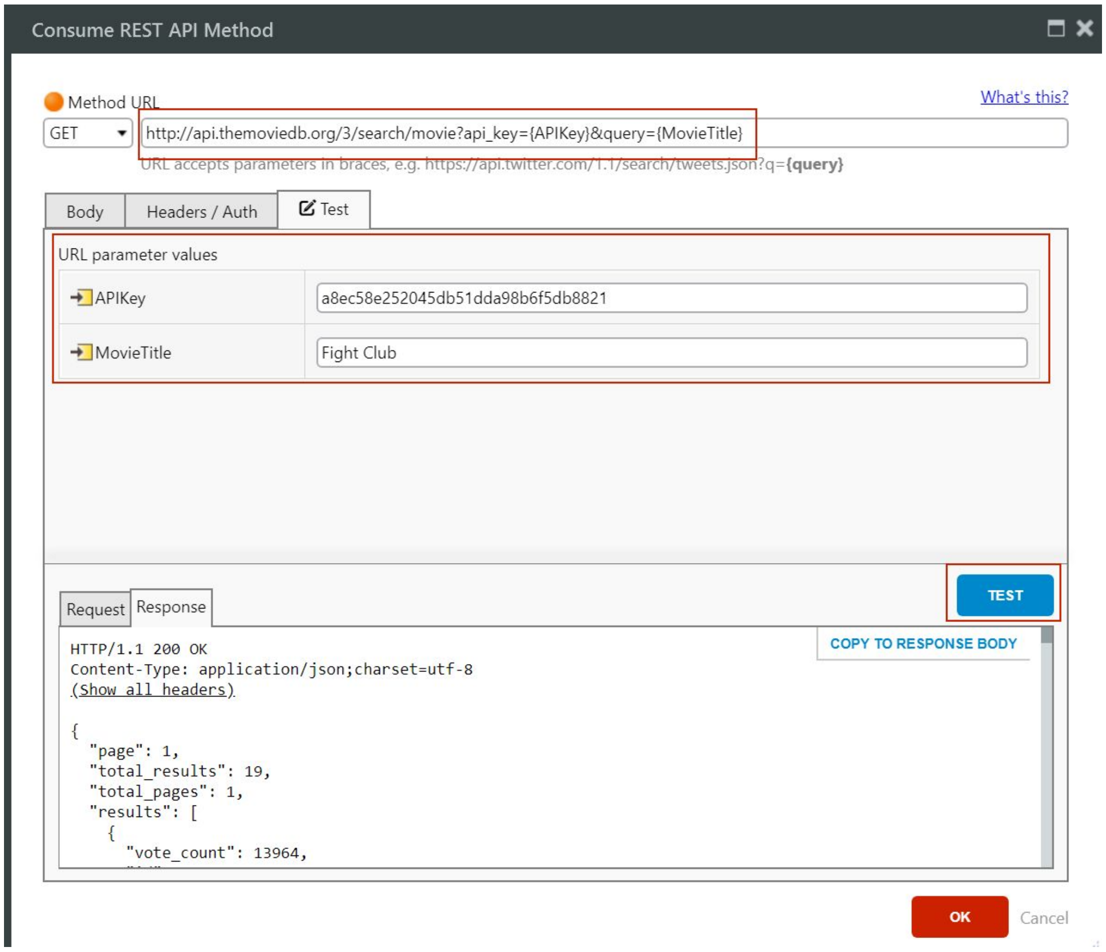
NOTE: When you click **Test**, the API is called via the OutSystems server, and not directly by Service Studio on your computer. This allows you to have the same behavior that your application will have at runtime.

e) The URL can be changed in order to search for other movies. For instance, replace the text *Fight Club* by *Star Wars*, and press **Test** again. The **Response** on the bottom should change.

f) Change the **Method URL** to have two Input Parameters, like this: *http://api.themoviedb.org/3/search/movie?api_key={APIKey}&query={MovieTitle}*

NOTE: The special syntax **{query}** allows you to define input parameters that can be set later in the application logic. This allows us to make calls to the REST API with different parameter values, instead of using a statically defined API Key and movie title. Under the **Test** tab, you can now define the values for both parameters (**APIKey** and **MovieTitle**).

g) Set some test values for both parameters, using the **APIKey** given above, and **Test** again. The Movie Database API requires you to sign up for an account to get an API Key. We have already created an account to use **ONLY** in the context of this course, with the API Key: *a8ec58e252045db51dda98b6f5db8821*



http://api.themoviedb.org/3/search/movie?api_key={APIKey}&query={MovieTitle}

APIKey	a8ec58e252045db51dda98b6f5db8821
MovieTitle	Fight Club

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
(Show all headers)

{
  "page": 1,
  "total_results": 19,
  "total_pages": 1,
  "results": [
    {
      "vote_count": 13964,
```

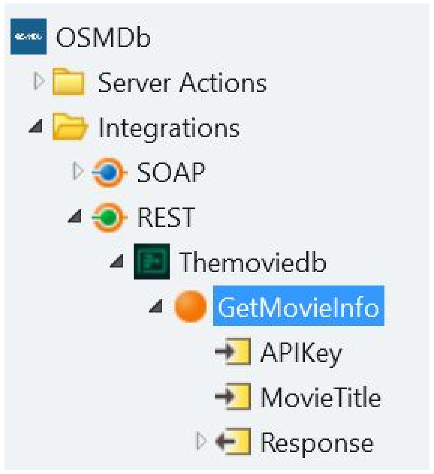
h) To make sure that OutSystems knows what is the type of response that the REST method returns, click on the blue **Copy to Response Body** link and press **OK**.

NOTE: With REST, it is necessary to provide a sample response. Service Studio will use it to define the required Structures that match the result of the REST API method. With **Copy to Response Body**, the response body created during the **Test,** is copied to the **Body** tab, thus simplifying the process of providing a sample response. Also, some API documentations provide sample responses that you can use.

---

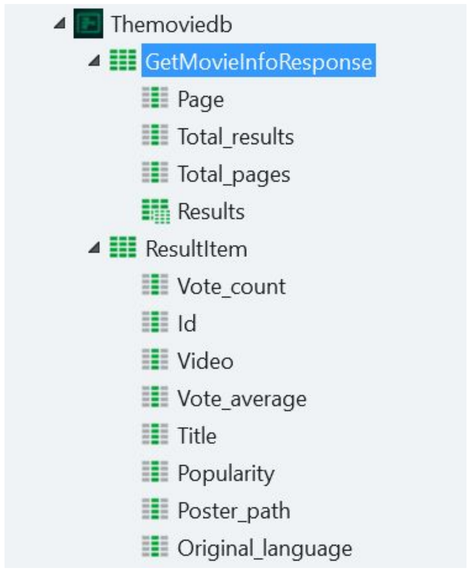
i) In the **Logic** tab, you now have a new Action that corresponds to the REST API endpoint, to obtain the movie information.

j) Change the REST API Method to *GetMovieInfo*.



NOTE: Renaming the Action **GetMovie2** will not change the REST API endpoint URL defined before. This is a name used inside Service Studio.

k) Switch to the **Data** tab and, under the **Structures** folder, you have the item **Themoviedb**. Inside you should have some Structures defined.



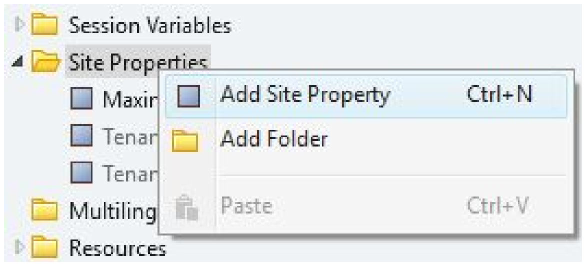
NOTE: These Structures were automatically created by Service Studio, based on the sample response that was previously inserted in the **Body** Tab. Service Studio also has defined the **Data Type** of the attributes based on the same sample response.

Define Site Properties for the API

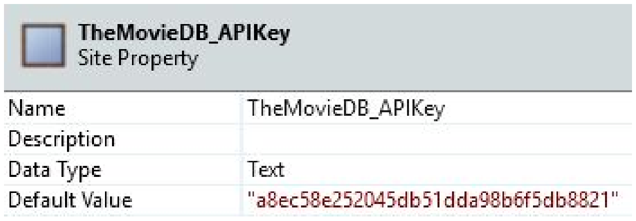
In this part of the exercise, we will define two new **Site Properties**. One will be used to store the **API Key**, and the second will store the base URL for movie posters.

We will define the first Site Property called *TheMovieDB_APIKey*, with **Data Type** *Text* and with **Default Value** set as *"a8ec58e252045db51dda98b6f5db8821"*. The second Site Property will be called *TheMovieDB_BaseURL*, with **Data Type** *Text* and with **Default Value** set as *"http://image.tmdb.org/t/p/w154"*.

1) Switch to the **Data** tab, right-click the **Site Properties** folder and select *Add Site Property*.



2) Set the Site Property **Name** to *TheMovieDB_APIKey*. Make sure that the **Data Type** is set as *Text*, and set the **Default Value** to *"a8ec58e252045db51dda98b6f5db8821"*

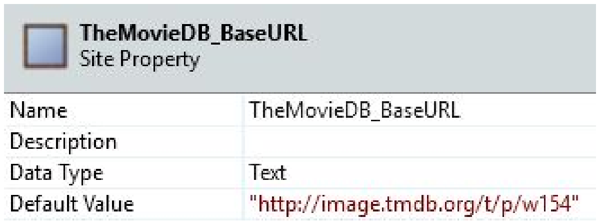


TheMovieDB_APIKey
Site Property

Name	TheMovieDB_APIKey
Description	
Data Type	Text
Default Value	"a8ec58e252045db51dda98b6f5db8821"

NOTE: When a Site Property is created, it is set by the developer with a default value. At runtime, someone with privileges can change the Site Property value for a specific environment in Service Center as we did in the Session Handling Exercise.

3) Add a new **Site Property** and set the **Name** to *TheMovieDB_BaseURL*. Make sure that the **Data Type** property is set to *Text*, and set the **Default Value** to *"http://image.tmdb.org/t/p/w154"*



TheMovieDB_BaseURL
Site Property

Name	TheMovieDB_BaseURL
Description	
Data Type	Text
Default Value	"http://image.tmdb.org/t/p/w154"

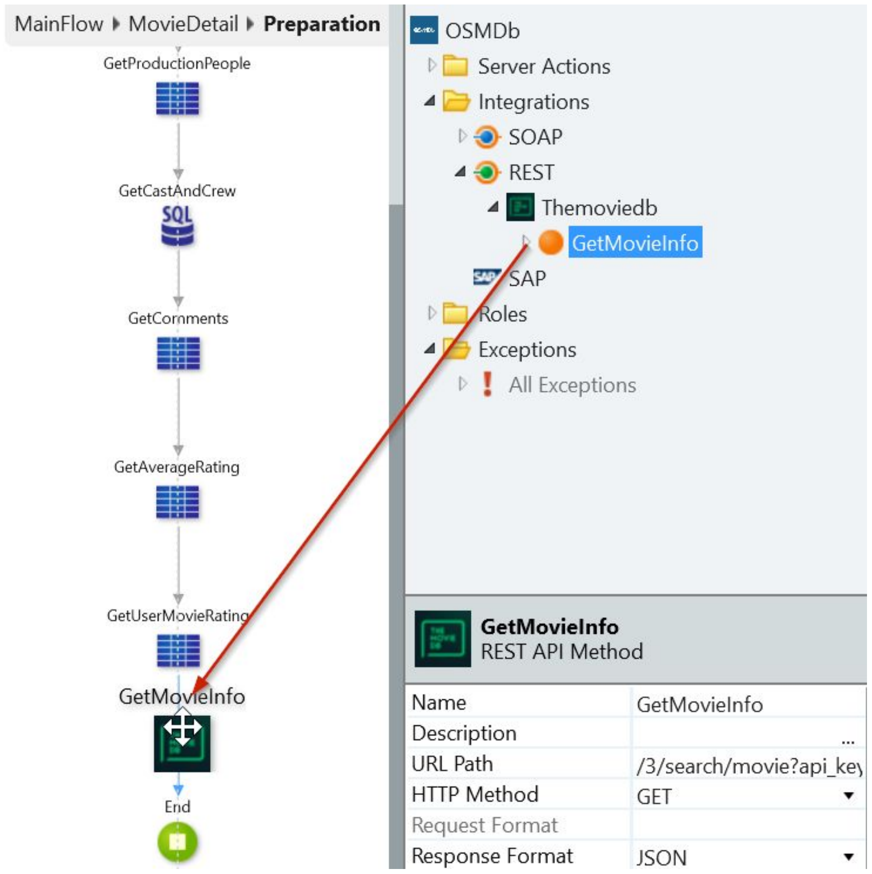
Call the GetMovieInfo API REST Method

Now that we have the method ready to be used, we will modify the **MovieDetail** Screen to display the movie poster image. For this, we need to call the **GetMovieInfo** REST API Method to retrieve the information about a movie, which includes the poster. Based on the result returned from the GetMovieInfo Method, we will extract the movie poster URL, and then display it in the Sidebar of the MovieDetail Screen.

1) Call the REST API Action to retrieve the movie information for the current movie.

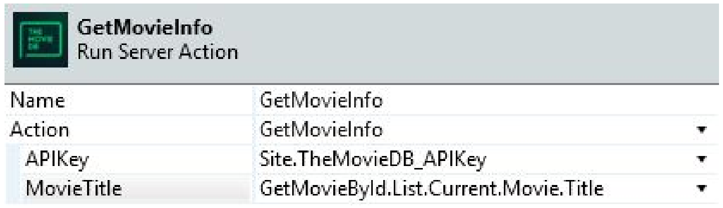
   a) Open the Preparation of the **MovieDetail** Screen.

   b) Switch to the Logic tab and drag the **GetMovieInfo** REST API Method and drop it before the End statement.



Name	GetMovieInfo
Description	...
URL Path	/3/search/movie?api_key
HTTP Method	GET
Request Format	
Response Format	JSON

c) Set the **GetMovieInfo** Action properties as follows



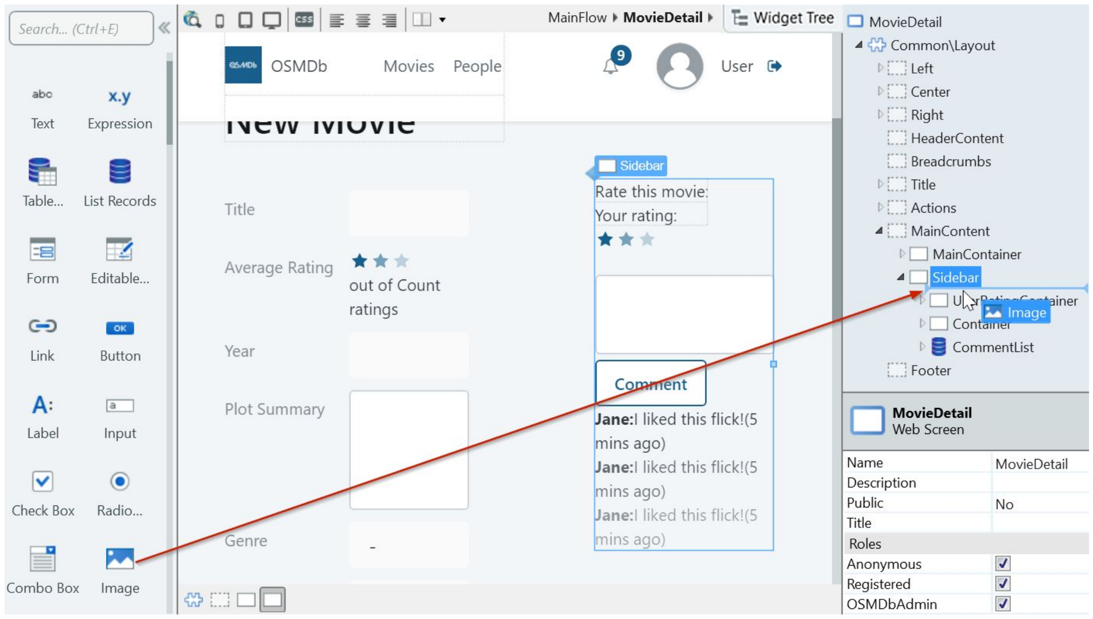
GetMovieInfo
Run Server Action

Name	GetMovieInfo
Action	GetMovieInfo
APIKey	Site.TheMovieDB_APIKey
MovieTitle	GetMovieById.List.Current.Movie.Title

2) Add the movie poster image to the **MovieDetail** Screen, in the Sidebar.

a) Open the **MovieDetail** Screen.

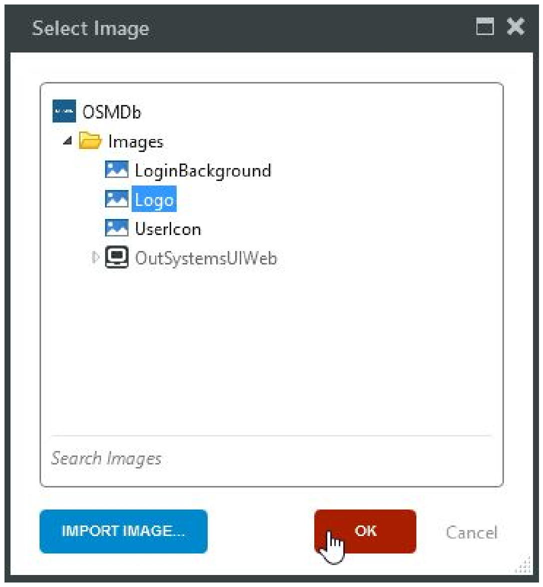
b) Drag an **Image** Widget above the **UserRatingContainer**, still inside the **SideBar**.



MovieDetail
Web Screen

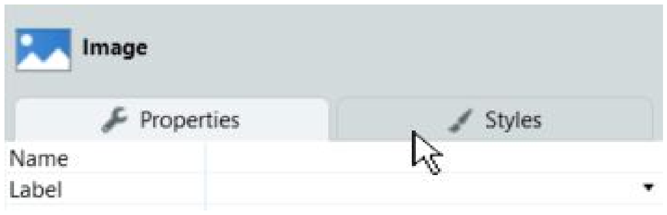
Name	MovieDetail
Description	
Public	No
Title	
Roles	
Anonymous	✓
Registered	✓
OSMDbAdmin	✓

c) In the **Select Image** dialog, select the **Logo** image resource.

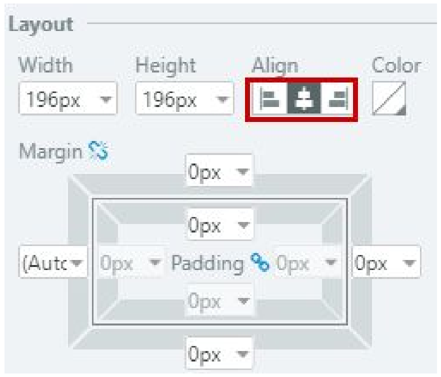


NOTE: Later, you will define the image URL of the movie poster, therefore the image selected here is just a placeholder for visual purposes in the development environment. Optionally, you could add a sample movie poster as an image resource, and use it here to make spacing more accurate.

d) With the image selected, in the properties area, change the view from the properties to the Styles Editor.

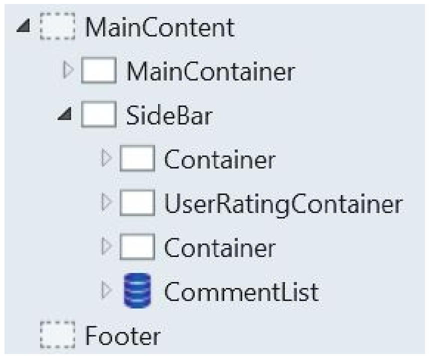


e) Align the image to the center, by selecting the corresponding option in the Styles Editor area.



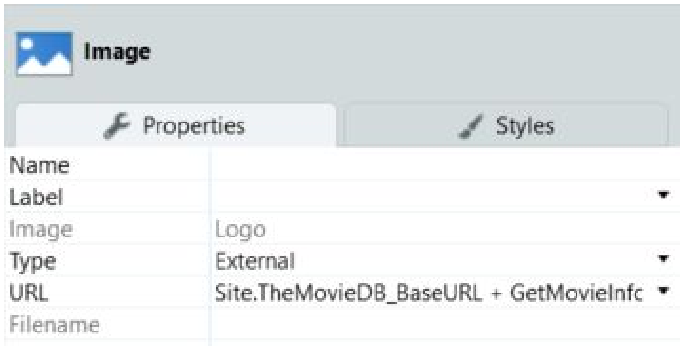
NOTE: Changing the alignment of an image, encloses the image in a new Container. The container created has the **Align** property set to the selected alignment (Center in this case).

f) In the Widget Tree, the structure of your Screen should look like this



```
MainContent
    MainContainer
    SideBar
        Container
        UserRatingContainer
        Container
        CommentList
Footer
```

g) Change the Image **Type** property to *External*, and set the **URL** property to *Site.TheMovieDB_BaseURL + GetMovieInfo.Response.Results.Current.Poster_path*

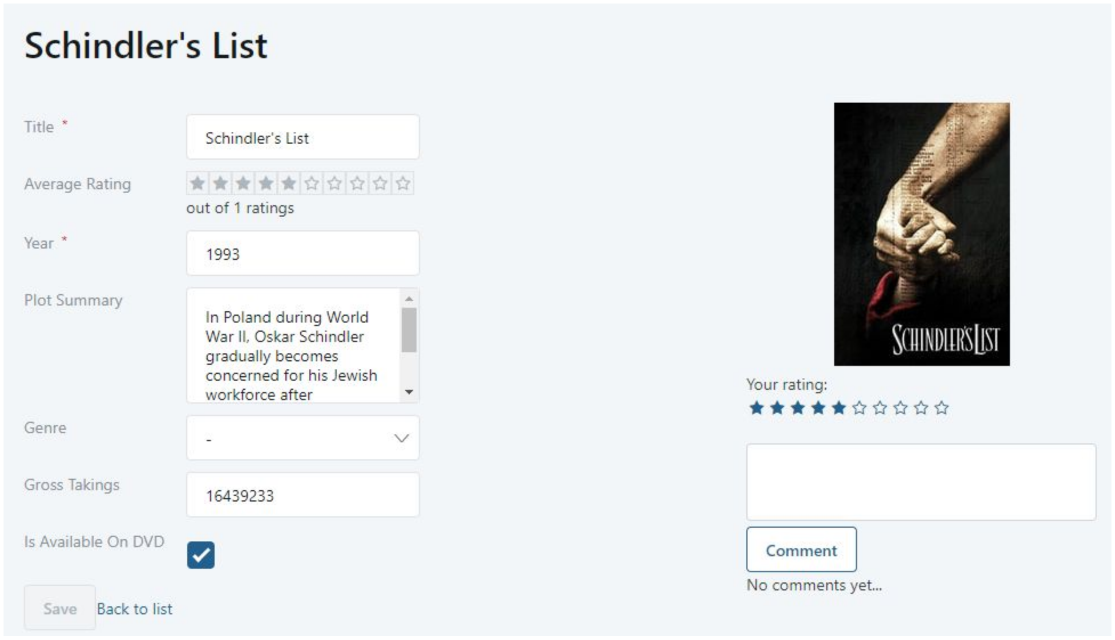


Name	
Label	
Image	Logo
Type	External
URL	Site.TheMovieDB_BaseURL + GetMovieInfc
Filename	

This will make sure that the source for the image comes from the response of the REST API method.

h) Click the **1-Click Publish** button to publish the application, and access it using your browser.

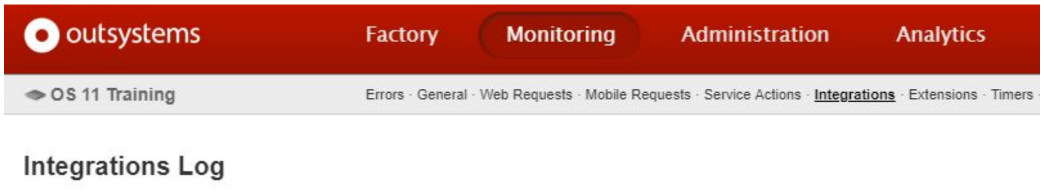
i) Navigate to the **MovieDetail** Screen by selecting an existing movie. The Screen should look like this


Schindler's List

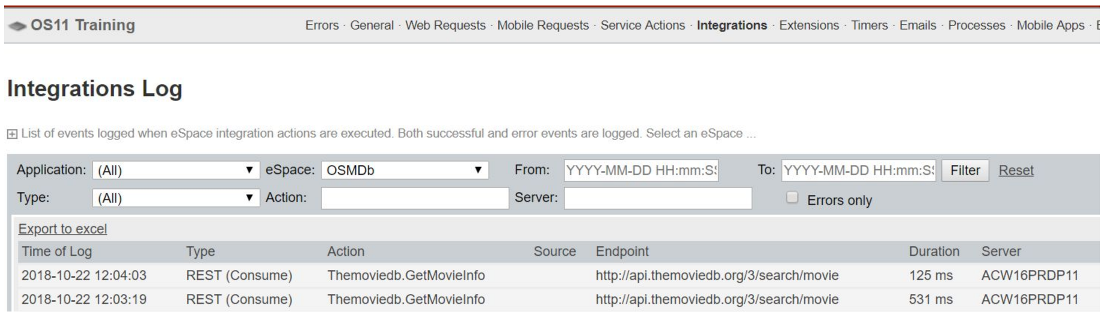
Title *	Schindler's List
Average Rating	out of 1 ratings
Year *	1993
Plot Summary	In Poland during World War II, Oskar Schindler gradually becomes concerned for his Jewish workforce after
Genre	-
Gross Takings	16439233
Is Available On DVD	✓

j) Open Service Center management console by accessing the following URL *https://<your_server>/ServiceCenter*

k) In the **Monitoring** tab, select the **Integrations** tab.


outsystems
Factory
Monitoring
Administration
Analytics
OS 11 Training
Errors · General · Web Requests · Mobile Requests · Service Actions · Integrations · Extensions · Timers ·
Integrations Log

l) Use the filter to search for your module (eSpace). You should see the calls made to the REST service.


OS11 Training
Errors · General · Web Requests · Mobile Requests · Service Actions · Integrations · Extensions · Timers · Emails · Processes · Mobile Apps ·

Integrations Log

List of events logged when eSpace integration actions are executed. Both successful and error events are logged. Select an eSpace ...

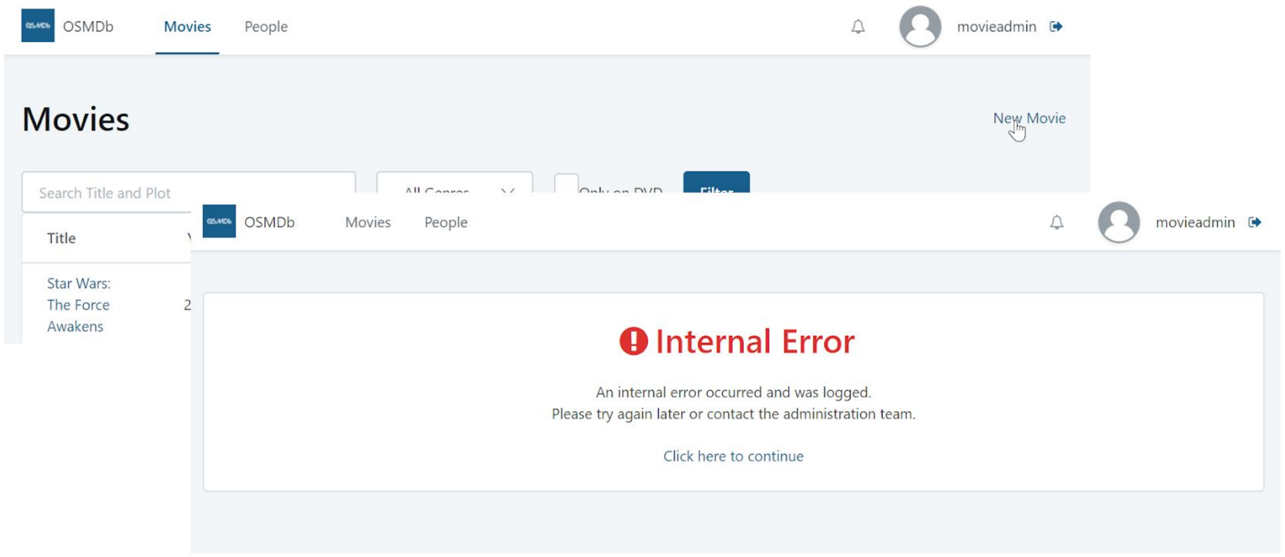
Application: (All) eSpace: OSMDb From: YYYY-MM-DD HH:mm:S: To: YYYY-MM-DD HH:mm:S: Filter Reset
Type: (All) Action: Server: Errors only

Export to excel

Time of Log	Type	Action	Source	Endpoint	Duration	Server
2018-10-22 12:04:03	REST (Consume)	Themoviedb.GetMovieInfo		http://api.themoviedb.org/3/search/movie	125 ms	ACW16PRDP11
2018-10-22 12:03:19	REST (Consume)	Themoviedb.GetMovieInfo		http://api.themoviedb.org/3/search/movie	531 ms	ACW16PRDP11

Implement the On After Response Callback

At this point, when someone tries to open the MovieDetail Screen to add a new movie, an exception will be thrown and an error will be displayed to the end-user.



Internal Error

An internal error occurred and was logged.
Please try again later or contact the administration team.

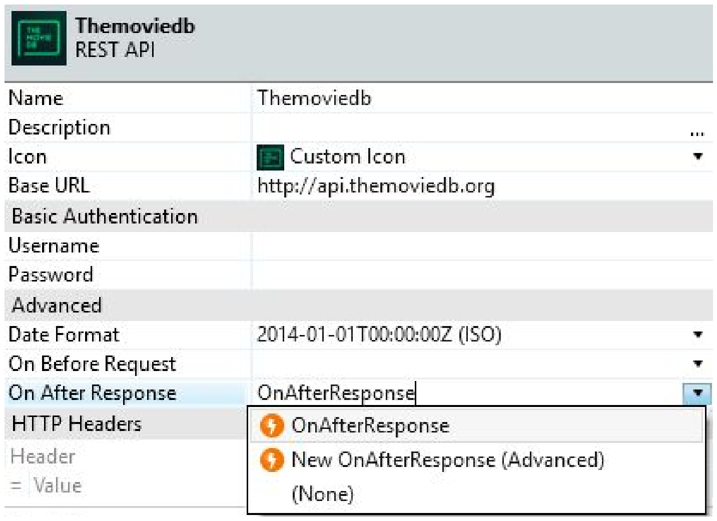
Click here to continue

This happens because in the Preparation of the MovieDetail Screen the **GetMovieInfo** is called with an empty MovieTitle. To address this issue, we will add an **On After Response** callback in the **Themoviedb** consumed REST service. The On After Response REST API Callback allows us to change the response of the **GetMovieInfo** call, before the code execution continues in the flow where this Action is being invoked. The callback to be implemented will programmatically ignore the exception that the **GetMovieInfo** throws when someone searches for "" (empty).

1) Add an **On After Response** callback, called *OnAfterResponse*, to the **Themoviedb** REST service.

   a) Switch to the **Logic** tab and select the **Themoviedb** Web Service.

b) Set the **On After Response** property to *New OnAfterResponse*. The **OnAfterResponse** callback runs after the **GetMovieInfo** call, but before the execution is returned to the Preparation of the MovieDetail Screen.



Themoviedb REST API	
Name	Themoviedb
Description	...
Icon	Custom Icon
Base URL	http://api.themoviedb.org
Basic Authentication	
Username	
Password	
Advanced	
Date Format	2014-01-01T00:00:00Z (ISO)
On Before Request	
On After Response	OnAfterResponse
HTTP Headers	
Header	
= Value	

Notice that the new OnAfterResponse REST API Callback is inside **Themoviedb**.

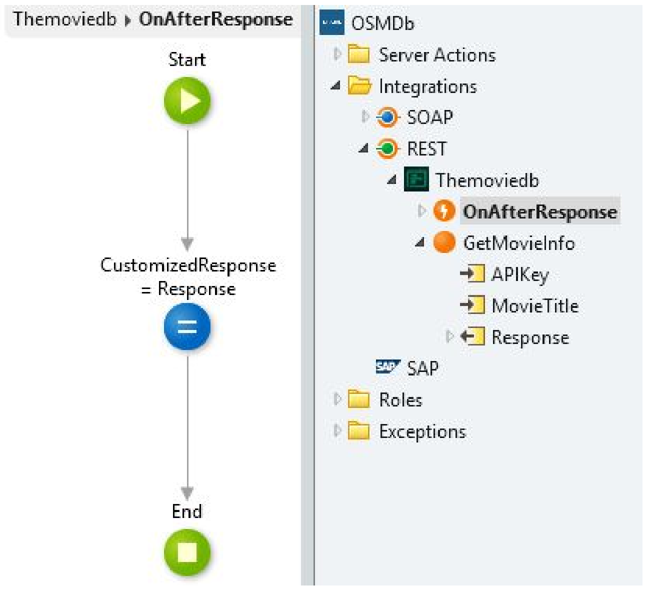
NOTE: Please ensure that you selected *OnAfterResponse* and **not** *OnAfterResponse (Advanced)*. Using the advanced version of the callback would make this exercise more complicated than needed.

---

2) In the callback, set the response **StatusCode** to *200 (OK)*, if the call returned the status / error code *422*. This will avoid the Internal Error page to appear when we are creating a new movie, since we manually assign the error code to *OK*.

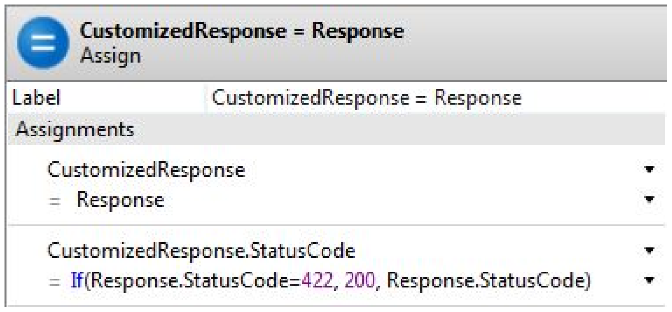
   a) Double-click the **OnAfterResponse** to open that Action.

b) Locate the **Assign** statement inside the callback Action and select it.



c) Add the following assignment (after the existing one)

CustomizedResponse.StatusCode = If(Response.StatusCode=422, 200, Response.StatusCode)


CustomizedResponse = Response
Assign

Label	CustomizedResponse = Response
Assignments	

```
CustomizedResponse
= Response

CustomizedResponse.StatusCode
= If(Response.StatusCode=422, 200, Response.StatusCode)
```

NOTE: The **OnAfterResponse** callbacks run after the web service calls, but before execution is returned to the flow where the call is made. They are designed to allow custom logging, modification of the return values etc., before normal execution continues.

The **GetMovieInfo** Action, by design, returns the error code *422*, when called with an empty search, which will be translated to an exception being thrown in the OutSystems side.

What we did above is set the status code to *200 (OK)*, if the 422 status/error code is returned by the call. This will suppress the exception, allowing the **MovieDetail** Preparation to continue its flow.

End of Lab

In this exercise, we integrated our application with an existing REST API endpoint, to retrieve information about a movie, in particular the movie poster. The results provided by the consumed API were used to display the poster in the respective **MovieDetail** Screen. This way, we learned how to use OutSystems to integrate our applications with existing REST APIs, and execute methods from a REST API with dynamic parameters.

After adding a Web Reference to a REST API endpoint, the Actions (or Action) used become available in Service Studio to be used in Screen Preparations and Server Actions, for example, just like any other Server Action.

Additionally, we learned how to implement a callback to modify the returned response before normal code execution resumes.