



DEVELOPING OUTSYSTEMS MOBILE APPS

Movies Assignment

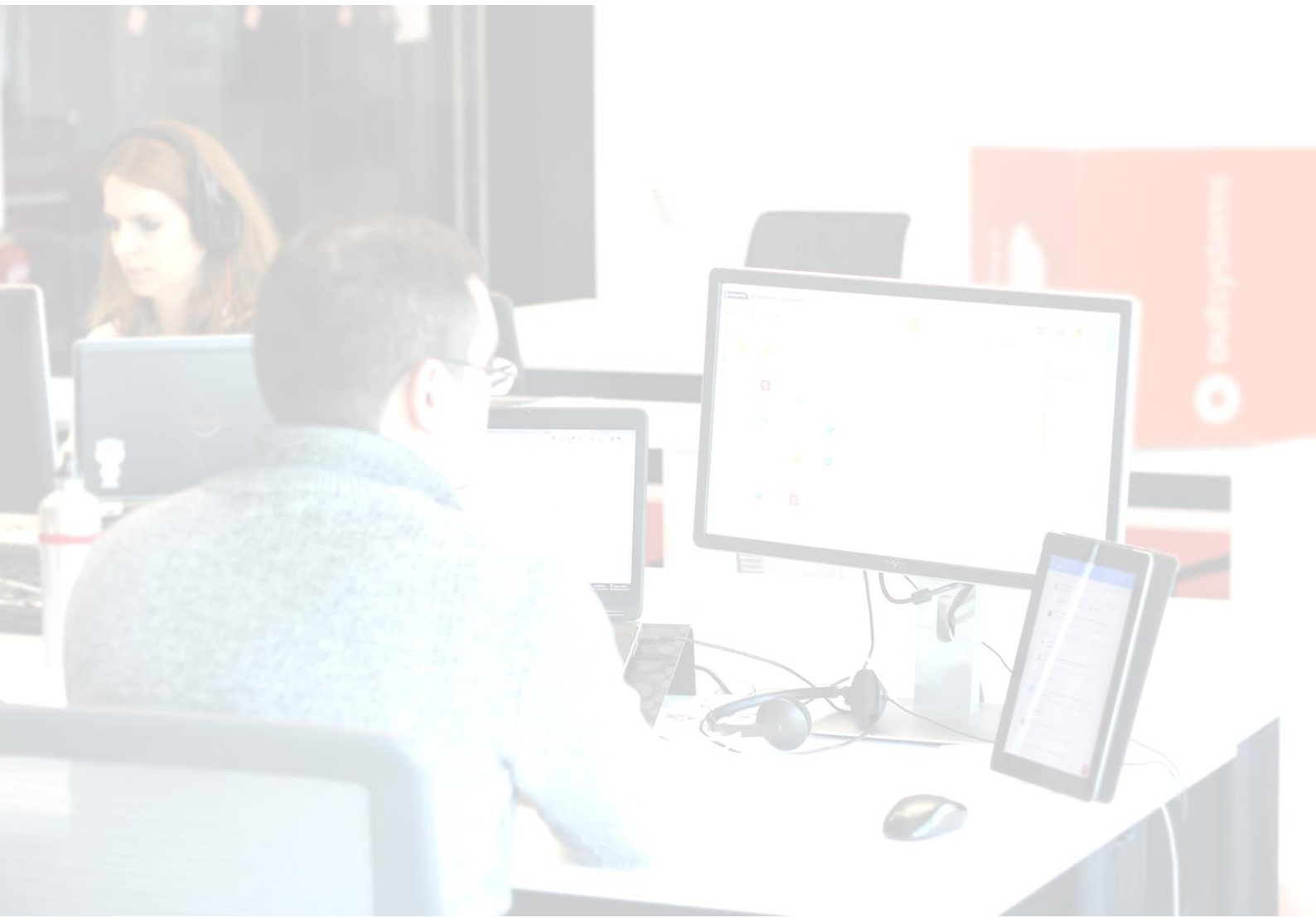


Table of Contents

Introduction: Movies App	3
Assignment 1: Movies Data	5
Assignment 2: Create the Movies App	6
Assignment 3: Basic Navigation.....	10
Assignment 4: Cinemas and Movies.....	12
Assignment 5: Movie and Cinema Details	14
Assignment 6: Buy Ticket Logic	16
Assignment 7: Buy Ticket Screen	18
List of images.....	24

Copyright

This material is owned by OutSystems and may only be used in the ways described in this Copyright Notice:

- You may take the temporary copies necessary to read this document
- You may print a single copy of this material for personal use
- You must not change any of this material or remove any part of any copyright notice
- You must not distribute this material in any shape or form

Introduction: Movies App

The goal of this Assignment is to build a **Movies** app that allows users to see the details about movies that are currently playing in cinemas and buy tickets.

Users like to view information about movies such as the title and images, as well as the release date, genres, and information about the cast and crew members. Besides movies, we also have information about cinemas and when they are playing sessions of the movies. Users should be able to buy a ticket for a specific movie session in a cinema. Authenticated users will also be able to interact with other users by submitting comments about the movies.

To allow us to display information about **Cinemas**, **Movies**, **Cast and Crew Members** and book a **Ticket**, the following **Data Model** is available in the **MoviesData** module.

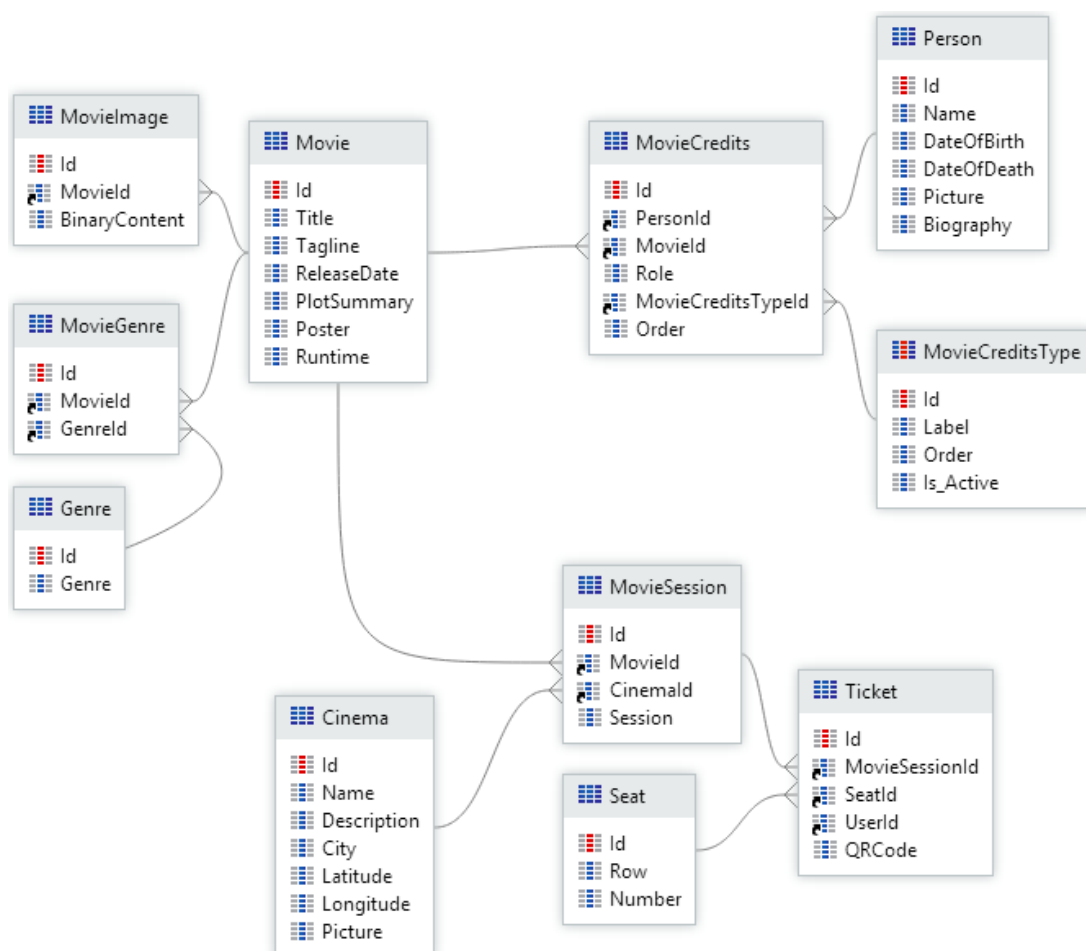


Figure 1. Data Model

The **Movie** Entity contains information about movies. This information will be displayed to the user, when the movie is selected in the App. Associated with each movie, there is also an extra set of images stored in the **MovieImage** Entity.

The **Genre** Entity contains the movie genres available, and the association between movies and genres is stored in the **MovieGenre** Entity. Notice that each movie can have more than one genre and vice-versa, thus the **MovieGenre** Entity is necessary to represent the many-to-many relationship.

The **Person** Entity stores the personal info about a cast or crew member. The details include the Name, Date of Birth, Date of Death, Picture and Biography.

The **MovieCreditsType** Static Entity defines the records **Cast** and **Crew**, to associate with a given **Person** in the App. The **MovieCredits** Entity stores the role (cast or crew member) for each **Person** that plays a role in a specific Movie. Notice that one **Person** can play more than one role in a movie (e.g. Director and Producer).

The **Cinema** Entity stores the information about cinemas. We will display the Title, Description, City and Picture to the users, and the **Latitude** and **Longitude** can be used to locate the cinema.

The **Seat** Entity has all the possible Seats available in each of the **Cinemas**. To keep things simple for the Assignment, all cinemas will have the same seat layout.

The **MovieSession** Entity has information about the available sessions for each one of the Movies, in a particular **Cinema**. It keeps the schedule information in the Session attribute, which is of DateTime type, to store the information about the Date and the Time in the same attribute.

The **Ticket** Entity will keep information about the tickets booked by a **User**. An identifier for the **MovieSession** Entity will be stored, to allow us to relate the **Ticket** with a **Session**, a **Cinema** and a **Movie**. We will also have an identifier for the **Seat** Entity to identify the **Seat** chosen. The QRCode Binary Data attribute will keep the generated QRCode image for the **Ticket**.

The following Entities have already been bootstrapped and contain data that you will be able to see and use during the assignment: **Cinema**, **MovieSession**, **Seat**, **Movie**, **MovieImage**, **MovieGenre**, **Genre**, **MovieCredits**, **Person** and **MovieCreditsType**.

The **Ticket** Entity will be populated by your application logic, when a user books a Ticket.

Assignment 1: Movies Data

NOTE: Assignment 1 can be skipped in a classroom Boot Camp. You should only do it if you are following the course Online.

1. Publish Movies Data App

- a) In the Environment tab of Service Studio, select **Open File...** from the **Environment** menu.

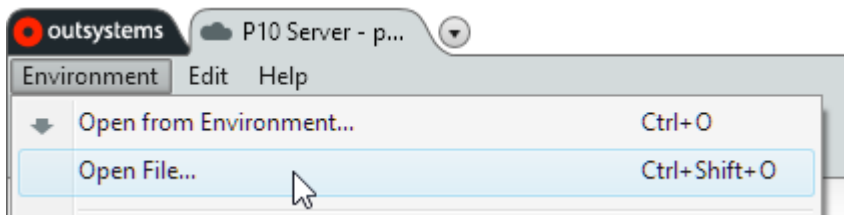


Figure 2. Open File

- b) In the **Open** dialog, change the file type to 'OutSystems Application Pack (*.oap)'.

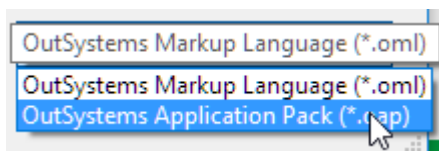


Figure 3. OutSystems Application Pack file type

- c) Locate the **MoviesData.oap** file in the Resources folder and click **Open**.
- d) When the installation of the application completes, you can proceed to the following step of the Assignment.

NOTE: The **MoviesData** module loads information from a remote server, The Movie DB, using a REST API. It will take several minutes (sometimes 15 minutes or more) to load all the data.

Assignment 2: Create the Movies App

1. Create the Application

- a) Inside Service Studio, after successfully connecting to the Server, select **New Application**.



New
Application

Figure 4. Create a New Application

- b) This is going to be a **Mobile App**, so select that type and click on the Next button.

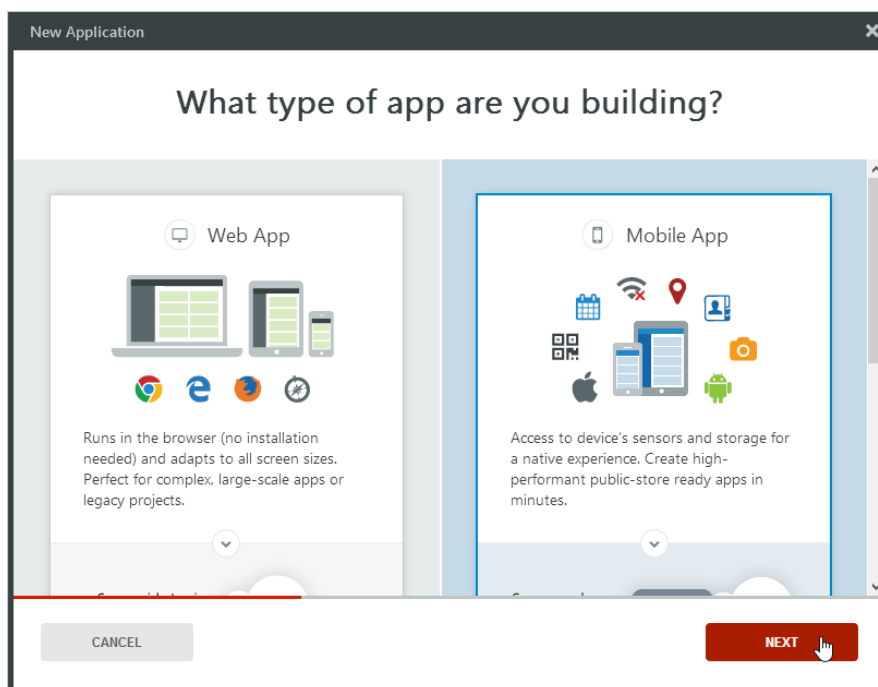


Figure 5. Select the Mobile App option

- c) Select the **Phone** template and then click the **Next** button.

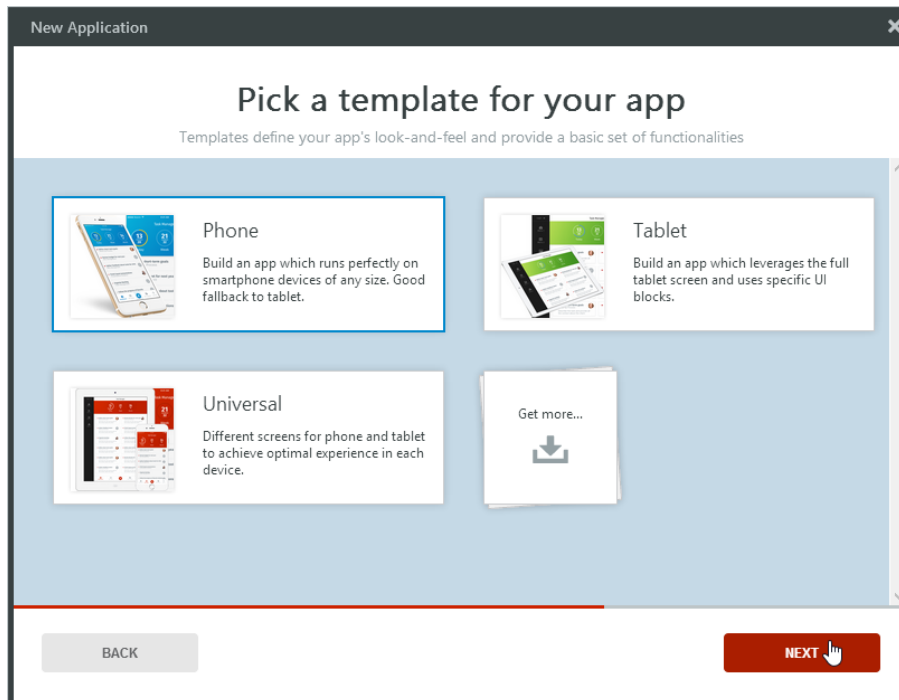


Figure 6. Select the Phone template

- d) Name the Application 'Movies_<your initials>'. Click the **Upload Icon** and select the **movies.png** image from the Resources folder. Click **Create App** to continue.

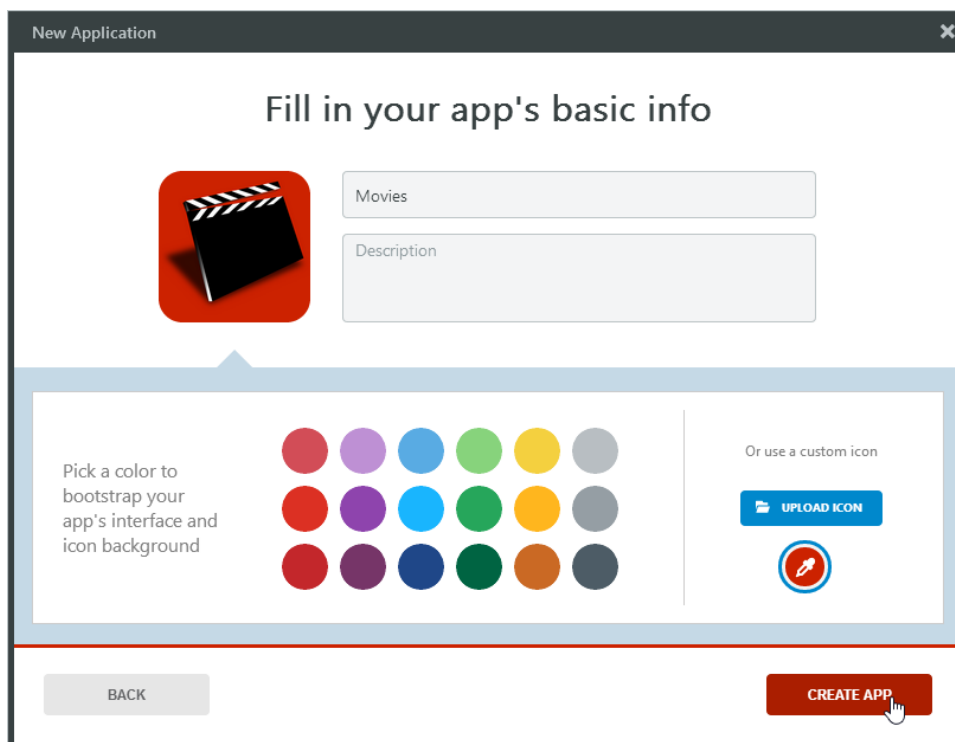


Figure 7. Give a name and logo to the Application

- e) Set the module name to 'Movies_<your initials>', verify that the Module Type is set to **Mobile** and click **Create Module** to create the module that will contain our mobile app's Screens.

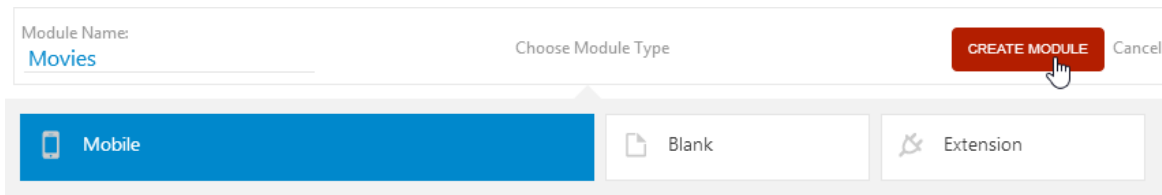



Figure 8. Creating a Movies Mobile module

2. Add References to the Data Model

In this Section, you will reference the Entities defined in the **MoviesData** App, from Assignment 1, in order to use them in your newly created Application.

- Open the **Manage Dependencies** dialog by clicking the  icon in the toolbar.
- In the list of producers' modules on the left, select the **MoviesData** module. Then, on the right side, tick the **Cinema**, **Genre**, **Movie**, **MovieImage**, **MovieGenre**, **MovieCreditsType**, **MovieCredits**, **MovieSession**, **Person**, **Seat** and **Ticket** Entities. The remaining Entities can be used to further extend the app with other interesting features, but aren't needed at this time.

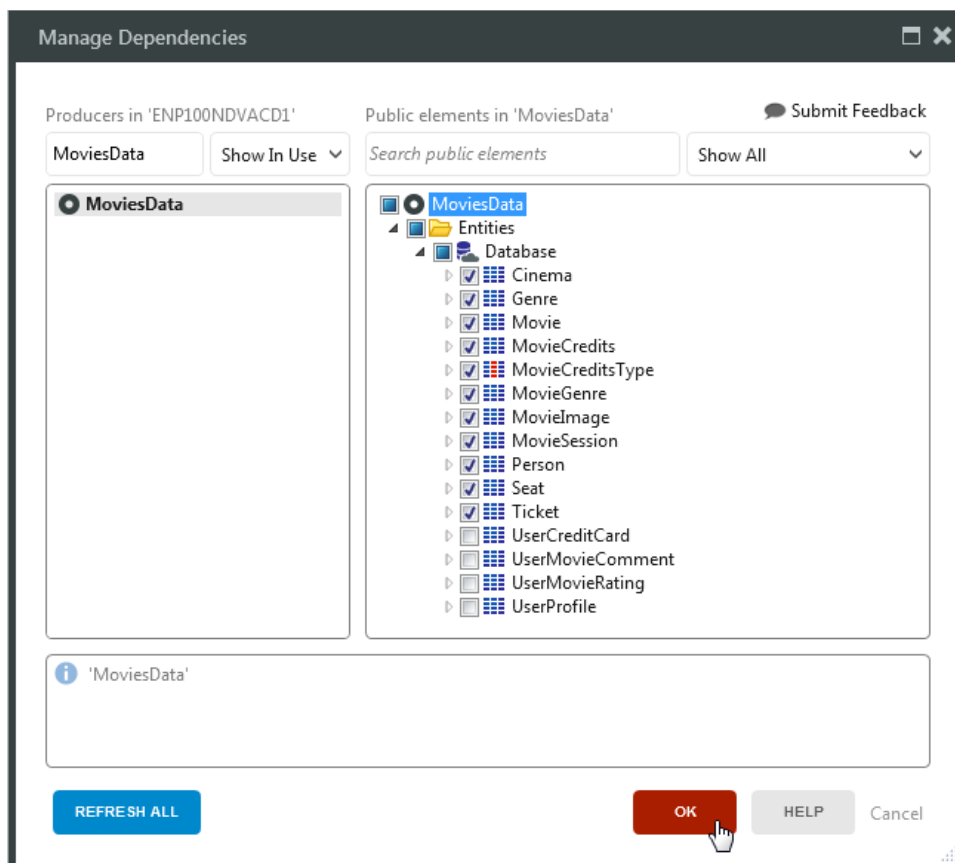


Figure 9. Reference Entities from MoviesData

- Click **Ok** to close the **Manage Dependencies** dialog.

3. Change the Splash Screen logo

In this Section, you will change the Splash Screen of the App, to make it look similarly with the remaining App.

- a) In the **Interface** tab, open the **Common** Flow and double click the **Splash** Screen to open it in the Screen Editor.
- b) Replace the existing image by the '**movies.png**' image provided in the Resources folder.



Figure 10. Splash Screen

Assignment 3: Basic Navigation

In this step of the Assignment, we will start with the basic interactions necessary to navigate through the main Screens of the Movies application. The default Screen of the App will be a list Screen to display all the **Movies**.

1. Create Screens

In this Section, you will create the Screens of the Application. One to list the Movies, one to list the Cinemas, one for the Cast and Crew of a movie, one to store the movie tickets of the user and a final one with the Settings of the user.

- In the Main Flow, rename the **HomeScreen** to 'Movies'. Also, type 'Movies' in the Title placeholder of the Screen.
- Create 4 new Screens: **Cinemas**, **CastAndCrew**, **MyTickets**, and **Settings**.
- In each Screen, set its Title to the respective name given in the previous step.
- Set the **Movies**, **Cinemas** and **CastAndCrew** Screens to be accessible by **Anonymous** users. For now, only the **Settings** and **MyTickets** Screens will be restricted to Registered user.

2. Add links to the Menu and Bottom Bar

In this Section, you will add Links on the Menu and the Bottom Bar to the recently created Screens, to make it accessible in your App.

- Open the **Menu** Block from the **Common** Flow and create Links to the Screens, by dragging and dropping them to the Block. Modify the text inside each Link accordingly.
- Add an icon to each of the Links, to make them more intuitive visually. The icons seen in the image below are 'video camera', 'film', 'group', 'ticket' and 'gear' respectively.

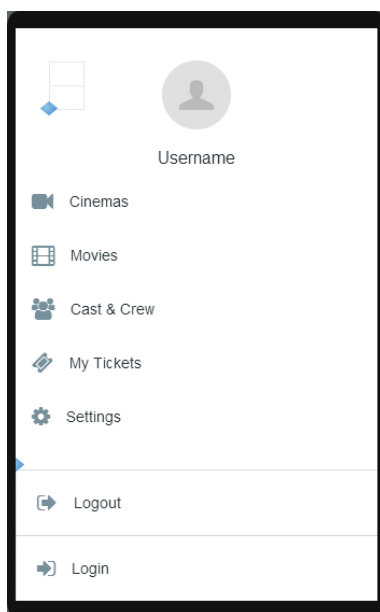


Figure 11. Menu Block

- c) In the **BottomBar** Block from the **Common** Flow, create five **Bottom Bar Items** and Link each one to the respective Screen.
- d) Your **BottomBar** Block should look like this

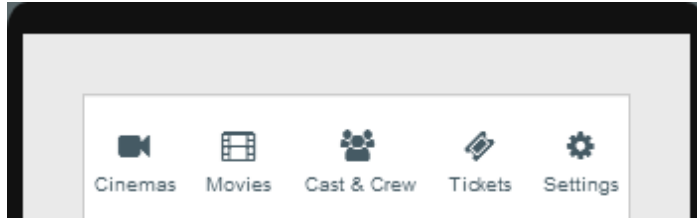


Figure 12. Bottom Bar Block

Assignment 4: Cinemas and Movies

In this exercise, we will start to build the list Screens for Cinemas and Movies, to display some basic information about these Entities.

1. Cinemas list Screen

Add an Aggregate to the **Cinemas** Screen to fetch the available Cinemas from the Database, sorted by the Cinema name. Based on the following mockup, add the required Widgets to display each Cinema. To display all the Cinemas, it is recommended to use a **List** and a **ListItemContent** Widget, to group together the Title, Location and Image like represented in the mockup. For the Picture, use an **Image** Widget and set its Type property to **Binary Data**. In the Image Content property, use the Picture attribute from the current Cinema.



Figure 13. Cinemas Screen mockup

2. Movie Item Block

Create a new Block named 'MovieItem' to display the information for a Movie. Add an Input Parameter named **Movie** with Data Type set to 'Movie'.

Use a **ListItemContent** Widget in your Block, and place the Movie poster in the **Left** Placeholder. Since not all movies may have a poster, set the **Default Image** property of the **Image** Widget to use the 'not-yet-available.png' image located in the Resources folder.

Based on the following mockup, add the remaining movie information (Title, Tagline, Runtime and Release Date) to the Block. Notice that some Movies may not have the complete information, such as the Tagline, thus make sure that null values are not displayed, if they exist.



Figure 14. Movie Item mockup

3. Movies list Screen

Add an Aggregate to the **Movies** Screen to fetch the available Movies from the Database sorted by Release Date (descending). Use a **List** Widget in your Screen, and set its **Source** property to created **Aggregate**. Inside the **List** place a **ListItem** Widget, and in its Content Placeholder add an instance of the **MovieItem** Block.

Since we have quite a lot of Movies, create the On Scroll Ending action of the **List** Widget. You don't need to worry about the Logic in this Action, as it is automatically created for you, with the objective of allowing you to continue to scroll down for the remaining movies.

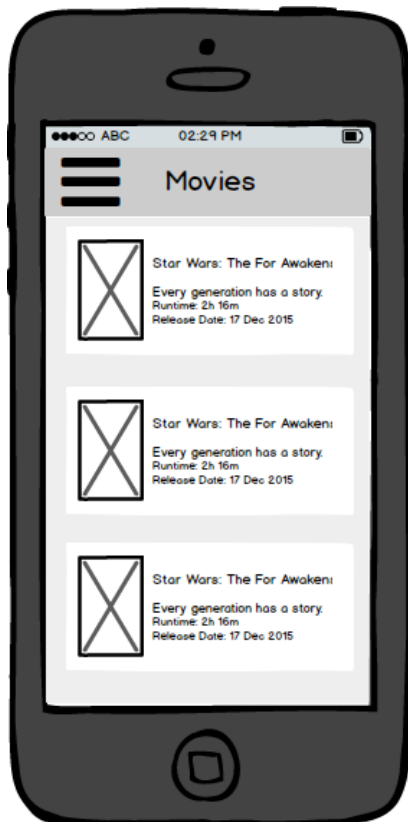


Figure 15. Movies Screen mockup

4. Publish and test the Movies App

1 Publish your eSpace (F5)

Assignment 5: Movie and Cinema Details

In this exercise, you will create the **MovieDetail** Screen, to display more information about each movie. Also, by clicking an item in the **Movies** Screen, the user will be redirected to this new Screen.

1. MovieDetail Screen

Create another Screen in your app named **MovieDetail**, and add an Input Parameter named **Movied** with Data Type set to 'Movie Identifier'. Add an **Aggregate** to the **MovieDetail** Screen, to fetch the movie information from the **Movie** Database Entity. Do not forget to add a Filter to the Aggregate to filter the **Movie** Entity, using the **Movied** Input Parameter.

In the Title placeholder of the Screen, add an **Expression** that displays the movie Title returned by the Aggregate.

Add a **Tabs** Widget to the Screen and name the 4 tabs 'Info', 'Cast', 'Crew', and 'Comments'. For now, you will only implement the content for the **Info** tab. Later, you will have the opportunity to implement the remaining tabs.

Inside the **Tab Content 1** Placeholder, use **Labels** and **Expressions** to display the attributes in a way that resembles the mockup below. Set the **OnClick** Event of the 'Watch this Movie' Button to redirect the user to the **Cinemas** Screen.

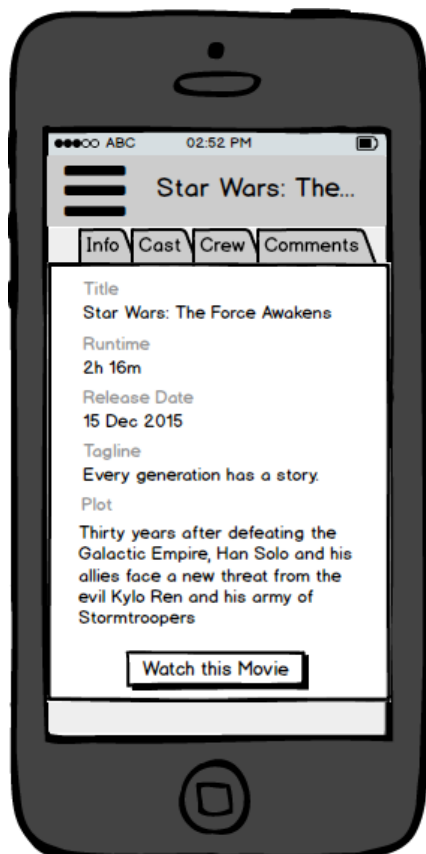


Figure 16. MovieDetail Screen mockup

2. CinemaDetail Screen

Create another Screen named **CinemaDetail**, and add an Input Parameter named **Cinemald** with Data Type set to 'Cinema Identifier'. Similar to the **MovieDetail** Screen, create an **Aggregate** to fetch the information about the **Cinema** that has its Id equal to the **Cinemald** Input Parameter. You also need to add **Labels** and **Expressions**, besides the **Image** on top, to display the information about the cinemas.

Below you can find a mockup of the Screen. The 'Select Movie to Watch' button will redirect the user to the **Movies** Screen.



Figure 17. CinemaDetail Screen mockup

3. Link List Screens to Detail Screens

In the **Cinemas** Screen, add a **Link** that enables users to click on each item in the list and navigate to the **CinemaDetail** Screen, with the Cinema Identifier sent as Input Parameter.

Also, add a Link in the **Movies** Screen that allows users to click on each item in the list and navigate to the **MovieDetail** Screen, in order to view the full movie information.

4. Publish and test the Movies App

1

Publish your eSpace (F5)

Assignment 6: Buy Ticket Logic

In this exercise, we will create the basic navigation for users to select a Cinema and a Movie and then be able to buy a ticket. We will allow users to start by selecting the Cinema, or the Movie, and then selecting the other. For instance, a user can start by selecting a Cinema and then the Movie that he wants to see, or the other way around. After selecting both, the user will be redirected to the **Buy Ticket** Screen. For now, we will modify the **Movies** App to support the described interaction. Later, we will deal with the remaining process of buying a ticket (select session and seat).

1. Create the BuyTicket Screen

Create a new Screen named 'BuyTicket' and add two Input Parameters. One named **MovieId** with type 'Movie Identifier', and a second one named **CinemaId** with type 'Cinema Identifier'. Tick the Anonymous role so all users can access the Screen.

In the Title placeholder of the new screen write 'Buy Ticket'. The remaining parts of the Screen will be developed in the next exercise of the assignment.

2. Modify CinemaDetail Screen

Add a new Input Parameter to the **CinemaDetail** Screen named **MovieId** with data type set to 'Movie Identifier'. Tick the Anonymous role so all users can access the Screen.

In the **CinemaDetail** Screen, enclose the 'Select Movie to Watch' Button in an **If** Widget. Set the Condition of the **If** to 'MovieId = NullIdentifier()'. In the **True** branch, you should have the 'Select Movie to Watch' button. In the **False** branch, add a new 'Buy Ticket' Button and set its **OnClick** destination to the **BuyTicket** Screen. Set the Input Parameters to the current Screen's **MovieId** and **CinemaId** Input Parameters.

3. Modify MovieDetail Screen

Add a new Input Parameter to the **MovieDetail** Screen, named **CinemaId**, with Data Type set to 'Cinema Identifier'.

Similarly to the previous step, enclose the 'Watch this Movie' Button in an **If** Widget. Set the Condition of the **If** to 'CinemaId = NullIdentifier()'. In the **True** branch, you should have the 'Watch this Movie' Button. In the **False** branch, add a new 'Buy Ticket' Button and set its **OnClick** destination to the **BuyTicket** Screen. Set the Input Parameters to the current Screen's **MovieId** and **CinemaId** Input Parameters.

4. Navigation between list and detail Screens

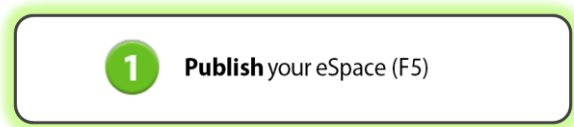
Both **Cinemas** and **Movies** Screens link to their respective detail Screen, which now have an extra Parameter. Since users can now reach the **Cinemas** Screen, after selecting a Movie to watch (and vice-versa), we need to store the identifier of the selected Movie (or Cinema).

Add an Input Parameter named **MovieId**, with Data Type set to 'Movie Identifier', to the **Cinemas** Screen. In the case of the **Movies** Screen, add an Input Parameter named **CinemaId** with Data Type set to 'Cinema Identifier'.

Since new Input Parameters were added to several of the Screens, we need to set them accordingly. In the **BottomBar** Links for the **Movies** and **Cinemas** Screens, set the Input Parameter to `'NullIdentifier()'`. This also needs to be done for the Links in the **Menu**.

In the **Cinemas** Screen, the link for the list item that navigates to the **CinemaDetail** Screen has a **Movied** Input Parameter. It should be set to the **Movied** Input Parameter that was passed into the **Cinema** Screen. The same applies to the Link in the **Movies** Screen. In the case of the **CinemaDetail** and **MoviesDetail** Screens, the **On Click** Event destination of the Buttons should also use the Input Parameter (**Cinemaid** and **Movied** respectively).

5. Publish and test the Movies App



Assignment 7: Buy Ticket Screen

In this exercise, you will implement the **BuyTicket** Screen, so users can select the movie session and seat after choosing the Cinema and Movie.

1. Display Cinema and Movie information

In the **BuyTicket** Screen, create two new Aggregates to gather the **Movie** information based on the **MovieId** Input Parameter, and the **Cinema** information based on the **CinemaId** Input Parameter. Use the results of the Aggregates to build the section above the movie sessions, like in the mockup below

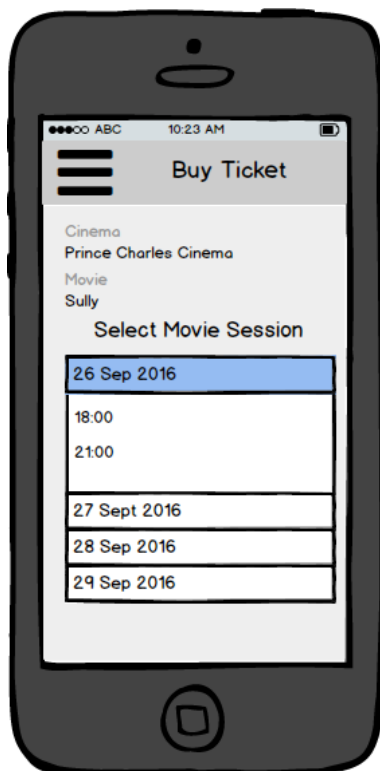


Figure 18. Select Movie Session mockup

2. Select Movie Session

Add a new Local Variable to the **BuyTicket** Screen named **SelectedMovieSessionId**, with Data Type set to 'MovieSession Identifier'.

Add a new **Aggregate** to the Screen. In this Aggregate, fetch data from the **MovieSession** Entity and filter it based on the **CinemaId** and **MovieId**. Create a new calculated attribute and call it **SessionDate** and define it to be the following expression

```
DateTimeToDate (MovieSession.Session)
```

Perform a **Group By** on this calculated attribute to display only the days that have available sessions.

Add a new **List** Widget to the Screen, and set its Source to the new Aggregate list. Add a **Section Expandable** Widget inside the **List**. In the **Title** placeholder of the **Section Expandable**, add an expression to display the **SessionDate** attribute.

In the **Content** placeholder, we want to display all the available Movie Sessions for that particular day. To do this, create a new Block called **SessionsPerDay**.

The new **SessionsPerDay** Block should have an Input Parameter called **SessionDate**, and also the **Cinemald** and **MovieId**. Inside the Block, add a new Aggregate that retrieves the **MovieSession** data filtered by the **Cinemald** and **MovieId**, and the sessions for that day, by using the following expression as a filter

```
DateTimeToDate (MovieSession.Session) = SessionDate
```

We also need to add an Event to the Block called **SelectSessionEvent** with an Input Parameter named **MovieSessionId**. This Event should be triggered when the user selects a movie session. The **BuyTicket** Screen will then handle the Event, and store the **MovieSessionId** value sent in the Event in the Screen's **SelectedMovieSessionId** Local Variable.

3. Select Seat

Using the **SelectedMovieSessionId** Local Variable, create an **If** Widget around the Screen section that enables the movie session selection. When the movie session has not been selected yet (`SelectedMovieSessionId = NullIdentifier()`), the Screen should display the available sessions. In the other branch of the **If**, when the session is already selected, you will add the Seat Selection map. You can use **Icons** to build the structure to display the seats based on the following mockup. You may be able to create the seats using **Bottom Bar Items** as well, to have the structure of the **Image** with the Seat Number below.

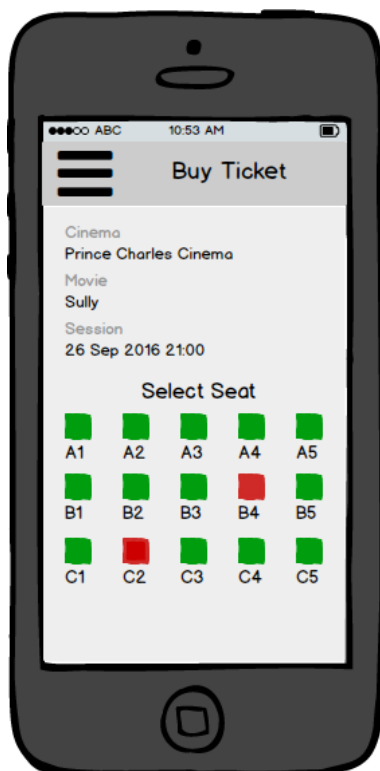


Figure 19. Select Seat mockup

To get the list of seats, you will need a new Aggregate. In this Aggregate, use the **Seat** and **Ticket** Entities. Make sure that the Join clause is set to **Seat** 'With or Without' **Ticket** and the **Join Condition** is set to the following

```
Seat.Id = Ticket.SeatId and Ticket.MovieSessionId = SelectedMovieSessionId
```

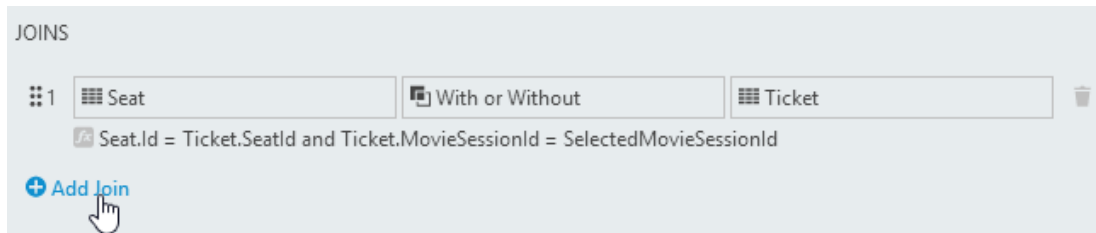


Figure 20. Join Condition

To know if a **Seat** is free, create a new calculated attribute called **IsFree** and define it to be

```
Ticket.Id = NullIdentifier()
```

With this Aggregate, you should be able to use a **List** Widget and display the seats. With the seats in the Screen, you may set the color of the icon, depending on the value of the **IsFree** attribut. Recall that the Style Class property can contain expressions, therefore you can use something like:

```
If(IsFree, "text-green", "text-red")
```

Note: You can easily get six Seats displayed per row in this Screen, but there are only 5 seats per row in this exercise. To have the five Seats per row displayed in the Screen (A1 – A5, etc.), you will need more than just this Aggregate. **Hint:** Try to use a Block to display the seats in a single row and use it to display all rows.

Clicking an empty seat (green icon) initiates the process of buying a ticket. In the Client Action to handle the **On Click** Event, you will need to call a Server Action to create and store the new ticket in the database.

After buying a ticket, the user should be redirect to the **MyTickets** Screen.

4. Optional: Use a QRCode (REST Web Service)

The **Ticket** Entity has a Binary Content attribute called **QRCode**. To fill in this attribute, you can use a REST API that returns generated QR Codes. You can check the reference to the API here:

<http://goqr.me/api/doc/create-qr-code/>.

Consume a REST Web Service in OutSystems, you should complete the following steps:

- Right-click the **REST** section., in the **Integrations** folder of the **Logic** tab, and select 'Consume REST API...'

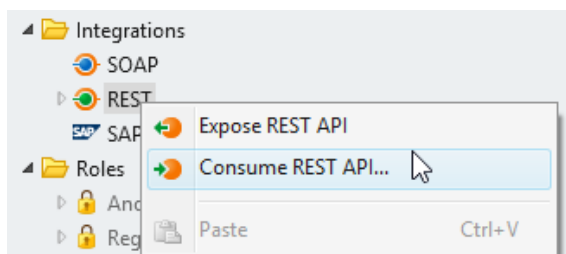


Figure 21. Consume REST API

- In the following window, select 'Add Single Method', as you will only use one Method of the API.

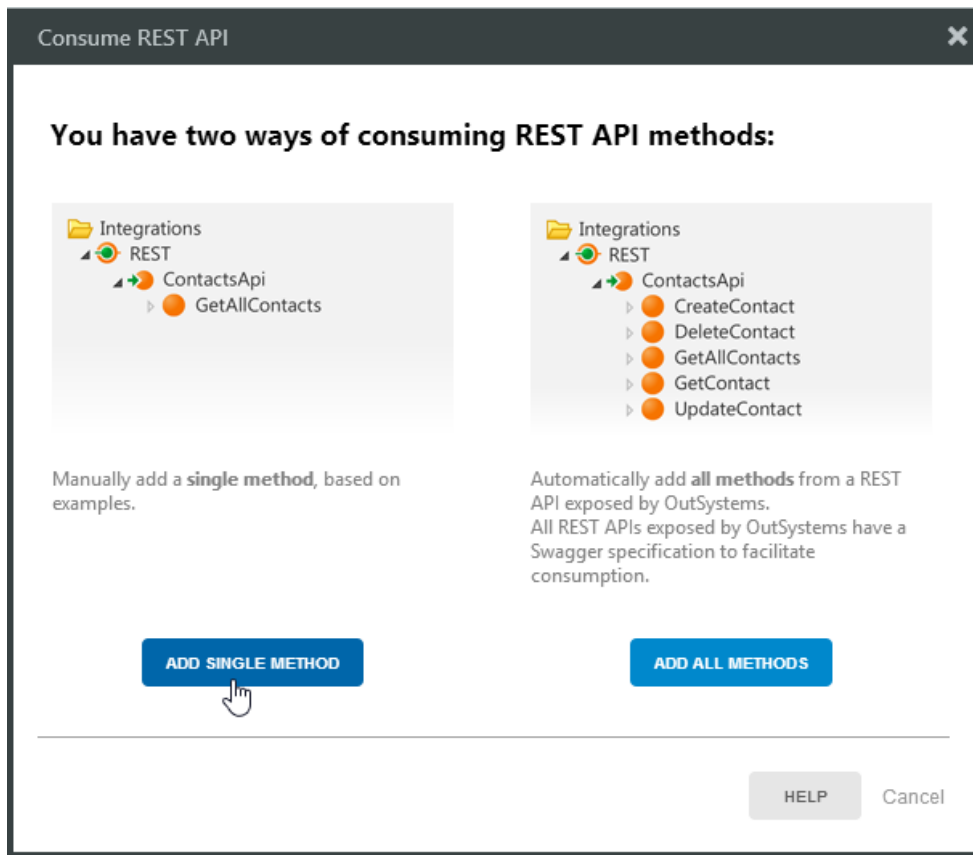


Figure 22. Add Single Method of a REST API

c) In the following window, follow the next six steps:

- Type the URL that returns the QR Code Binary Content:
<https://api.qrserver.com/v1/create-qr-code/?size=150x150&data={Data}>
- Click on the **Test** tab.
- Fill the Input Parameter **Data** with any text you want, e.g: TicketInfo.
- Click the **Test** Button.
- You will see a response in the section below the **Test** Button. Click on the **Copy to Response Body** option to define the structure of the Web Service Response, that OutSystems will use.
- Click **Ok**.

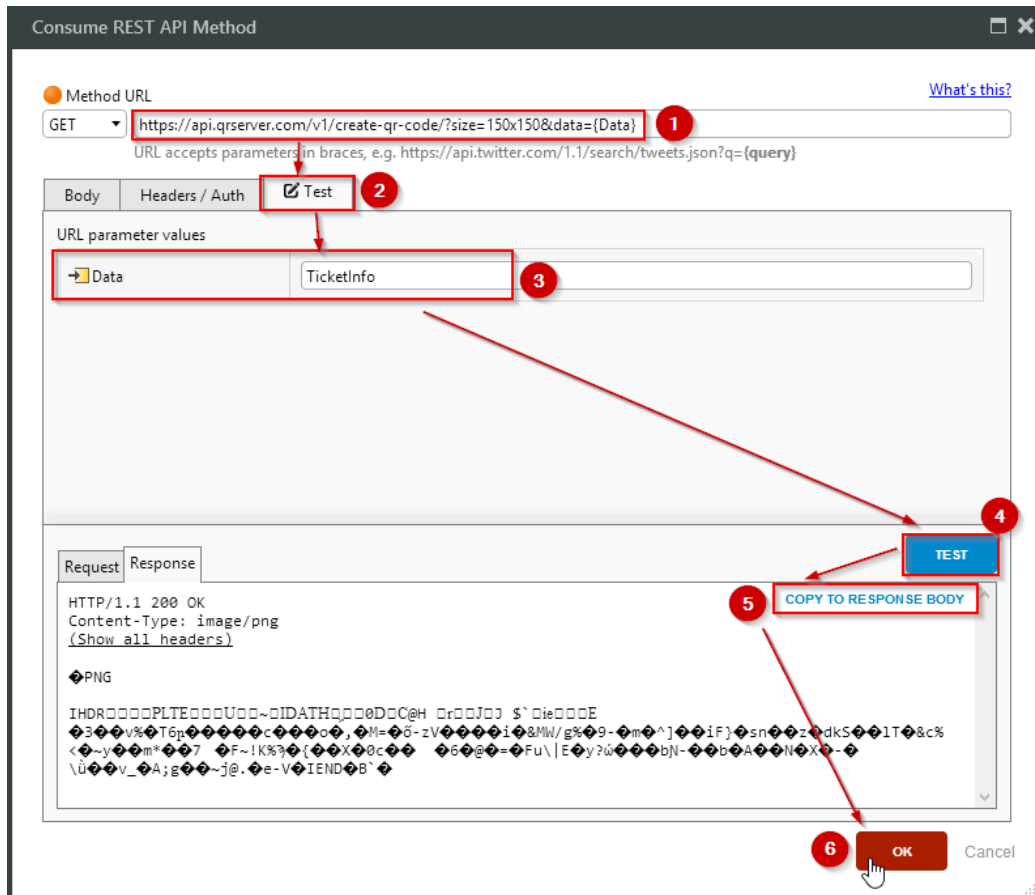


Figure 23. Consume REST API Method Steps

- d) You will see that the Method and the necessary Structures are automatically created by OutSystems.

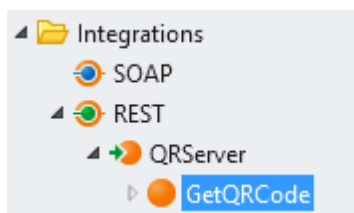


Figure 24. GetQRCode Method

- e) Change the **Response Format** of the generated method to Binary.

GetQRCode REST API Method	
Name	GetQRCode
Description	...
URL Path	/v1/create-qr-code/?size=200x200&data={Data}
HTTP Method	GET
Request Format	
Response Format	Binary
Timeout in Seconds	

Figure 25. Response Format of Web Service

5. Display Tickets

In the **MyTickets** Screen, display the list of tickets based on the following mockup

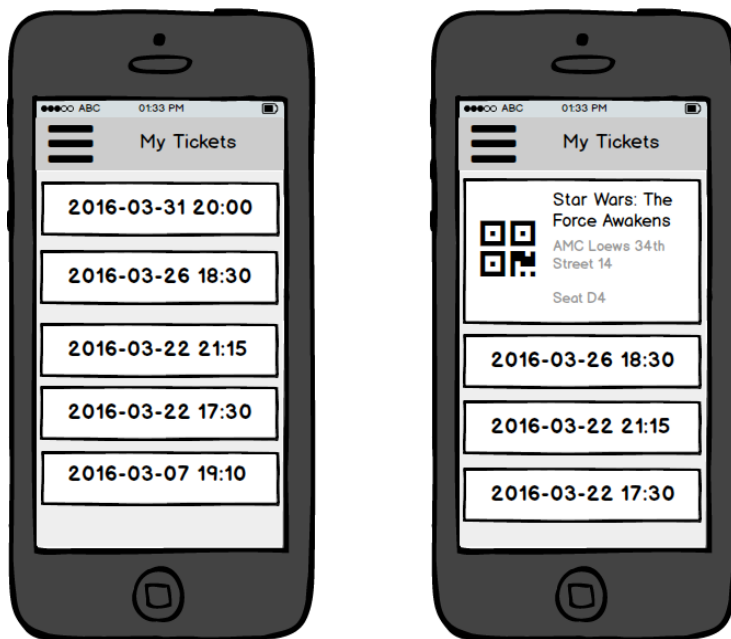


Figure 26. MyTickets Screen mockups

Use the **Flip Content** and the **ListItemContent** Widgets, to enable users to click the ticket and see its details (The QR Code is optional).

6. Publish and test the Movies App

1 Publish your eSpace (F5)

List of images

Figure 1. Data Model	3
Figure 2. Open File.....	5
Figure 3. Outsystems Application Pack file type	5
Figure 4. Create a New Application	6
Figure 5. Select the Mobile App option	6
Figure 6. Select the Phone template.....	7
Figure 7. Give a name and logo to the Application	7
Figure 8. Creating a Movies Mobile module.....	8
Figure 9. Reference Entities from MoviesData	8
Figure 10. Splash Screen	9
Figure 11. Menu Block.....	10
Figure 12. Bottom Bar Block.....	11
Figure 13. Cinemas Screen mockup.....	12
Figure 14. Movie Item mockup	13
Figure 15. Movies Screen mockup	13
Figure 16. MovieDetail Screen mockup	14
Figure 17. CinemaDetail Screen mockup	15
Figure 18. Select Movie Session mockup	18
Figure 19. Select Seat mockup.....	19
Figure 20. Join Condition.....	20
Figure 21. Consume REST API	20
Figure 22. Add Single Method of a REST API	21
Figure 23. Consume REST API Method Steps	22
Figure 24. GetQRCode Method	22
Figure 25. Response Format of Web Service	22
Figure 26. MyTickets Screen mockups	23