Soundplanes



Arquitectura e Integración de Sistemas Software

Grado de Ingeniería del Software

Curso 2º

George Laurentiu Bogdan (geobog@alum.us.es)

Guillermo Diz Gil (guidizgil@alum.us.es)

Carmen Mª Muñoz Pérez (carmunper1@alum.us.es)

Francisco Rodríguez Pérez (frarodper4@alum.us.es)

Tutor: Javier Troya Castilla

Group number: 1

Application link: <http://soundplanes.appspot.com/>

Project link in projETSII, GitHub or similar: <https://github.com/GuilleX7/Soundplanes>

VERSION HISTORY

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version | Details | Participants |
| 14/03/2014 | 1.0 | - Includes introduction, prototypes of the user interfaces and UML diagrams of components and deployment | George Laurentiu Bogdan  Guillermo Diz Gil  Carmen Mª Muñoz Pérez  Francisco Rodríguez Pérez |
| 03/05/2014 | 2.0 | - Added class and sequence diagrams, | George Laurentiu Bogdan  Guillermo Diz Gil  Carmen Mª Muñoz Pérez  Francisco Rodríguez Pérez |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Index

[1 Index 4](#_Toc39437278)

[2 Introduction 6](#_Toc39437279)

[2.1 Integrated applications 6](#_Toc39437280)

[2.2 Project development 7](#_Toc39437281)

[3 User Interface Prototypes 8](#_Toc39437282)

[3.1 Landing page view 8](#_Toc39437283)

[3.2 User register view 9](#_Toc39437284)

[3.3 Location view 10](#_Toc39437285)

[3.4 Main view 11](#_Toc39437286)

[3.5 Profile menu view 12](#_Toc39437287)

[3.6 Airport menu view 13](#_Toc39437288)

[3.7 Airport chat view 14](#_Toc39437289)

[3.8 Airport lyrics view 15](#_Toc39437290)

[4 Architecture 16](#_Toc39437291)

[4.1 Component diagram 16](#_Toc39437292)

[4.2 Deployment diagram 16](#_Toc39437293)

[4.3 High-level sequence diagram 17](#_Toc39437294)

[4.4 Class diagram 18](#_Toc39437295)

[4.4.1 Landing/sign in class diagram 18](#_Toc39437296)

[4.4.2 Map class diagram 19](#_Toc39437297)

[4.4.3 Social sign in/linking class diagram 20](#_Toc39437298)

[4.5 Sequence diagram 21](#_Toc39437299)

[4.5.1 Social sign in sequence diagram 21](#_Toc39437300)

[4.5.2 Sign in sequence diagram 22](#_Toc39437301)

[4.5.3 Create airport sequence diagram 23](#_Toc39437302)

[4.5.4 Enter airport sequence diagram 24](#_Toc39437303)

[4.5.5 Get airport track sequence diagram 25](#_Toc39437304)

[4.5.6 Export playlist sequence diagram 26](#_Toc39437305)

[5 Implementation 27](#_Toc39437306)

[6 Tests 28](#_Toc39437307)

[7 User’s Manual 29](#_Toc39437308)

[7.1 Mashup 29](#_Toc39437309)

[7.2 API REST 29](#_Toc39437310)

[8 References 39](#_Toc39437311)

# Introduction

Today, people are increasingly interested in learning more about other cultures. Thanks to technologies that are continuously growing, it is becoming easier to establish this communication. On the other hand, music is a key element in our society and unites people from all over the world. Even so, with the exception of very international artists, we are not aware of the musical culture of countries other than our own. Since music is such an important cultural element in each country and in people, we intend to carry out a project that consists of offering an innovative and educational service in the form of an international online radio.

Thus, Soundplanes wants to offer an interactive online radio service with the possibility of chatting with people from anywhere in the world. It will give its users the opportunity to listen to the most listened songs from each country, as well as to offer their own playlist. In this way, users will have the opportunity to learn new songs, as well as meet many people from around the world.

## Integrated applications

We will be using the following applications:

* **Youtube:** Youtube web player will be our music player. We will also use its REST API for fetching some metadata.
* **Spotify:** we will be using Spotify as a music search site. The user will be able to search for any existing song in Spotify and play it.
* **Genius**: will be used to fetch song’s lyrics.
* **Leaflet:** will be used for displaying interactive maps in the web browser. Used in combination with *OpenStreetMap*
* **Facebook:** we will allow the user to log in with his Facebook social account and post what music are they listening to at any time.
* **Google Geocoding:** will allow us to convert addresses into geographic coordinates, so we can position the player in the map in case we were unable to geolocate him/her.
* **Instant IRC Chat:** will allow us to create different chat channels, one for each airport, so users can chat among themselves.

|  |  |
| --- | --- |
| Application name | URL API documentation |
| Youtube | <https://developers.google.com/youtube/iframe_api_reference?hl=es>  <https://developers.google.com/youtube/v3/docs> |
| Spotify | <https://developer.spotify.com/documentation/web-api/> |
| Genius | <https://docs.genius.com/> |
| Leaflet | <https://leafletjs.com/reference-1.6.0.html> |
| Facebook | <https://developers.facebook.com/docs/javascript> |
| Google Geocoding | <https://developers.google.com/maps/documentation/geocoding/intro?hl=es-419> |
| Instant IRC Chat | <https://documenter.getpostman.com/view/6300420/SzmZdgNU?version=latest> |

## Project development

We are thinking about all the different applications that we could integrate in our final application. Some of these may not work as expected and be removed, or we could change our visions about what do we want our application to be.

As of April 27, we know all the APIs we are using in the project, and we have obtained all the corresponding API keys in order to consume their services. We plan to keep the application simple but powerful and elegant. We are putting a lot of emphasis in the user experience.

As of May 3, we are thinking about the possibilities of the Facebook API. We already have included it for signing in, but we want to explore the Facebook universe in a deeper way.

# User Interface Prototypes

Our application will work as a single-page application. This means we will display all our views dynamically, without the loading of additional pages. We expect this to improve the user experience.

## Landing page view

A first view of the page, where the project is described to the user.

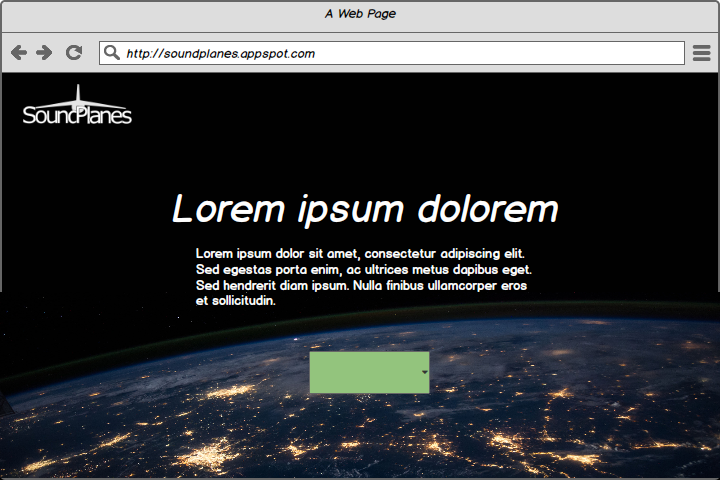


Figure 1. Landing page view’s user interface prototype

## User register view

User is requested his nickname to be displayed in-game, after that it has two options, to allow the use of user’s location in order to improve his experience or to choose the location manually, this is where its base will be located.

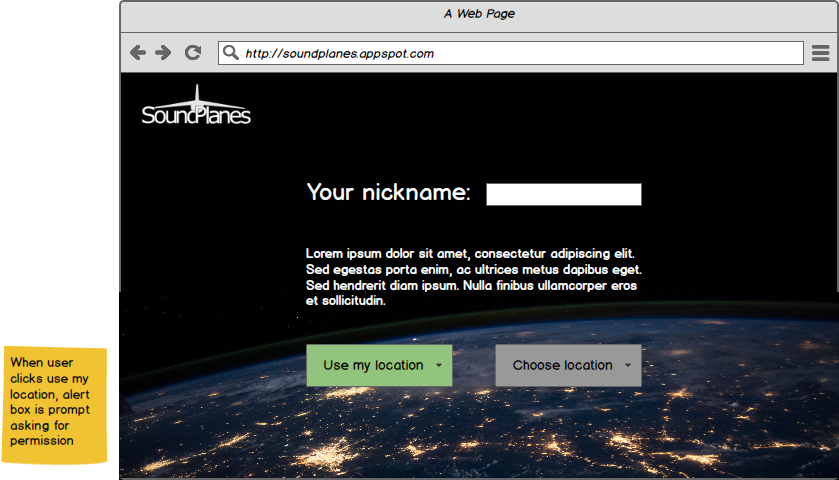


Figure 2. Setup view’s user interface prototype

## Location view

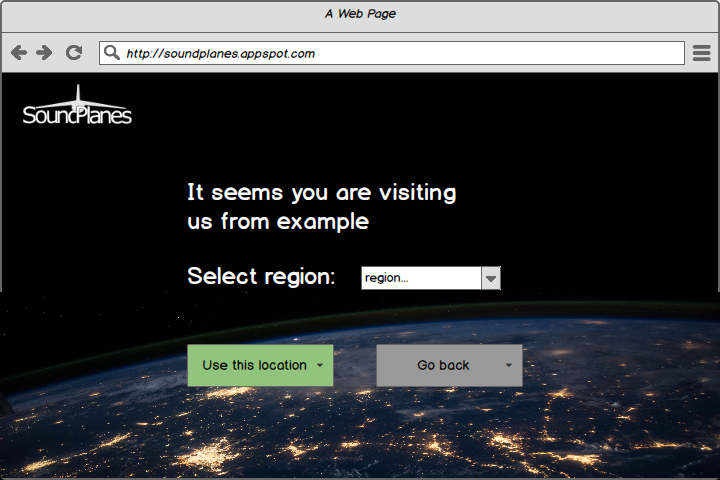
If user want to choose location manually, it’ll show the country from where he is visiting our site, and he will pick the region of the country. If he wants to use its location automatically, he can go back.

Figure 3. Location view's user prototype interface

## Main view

This will be the main view of the player, the region where the player is will be displayed, the plane is user’s position and he can see nearby airports.

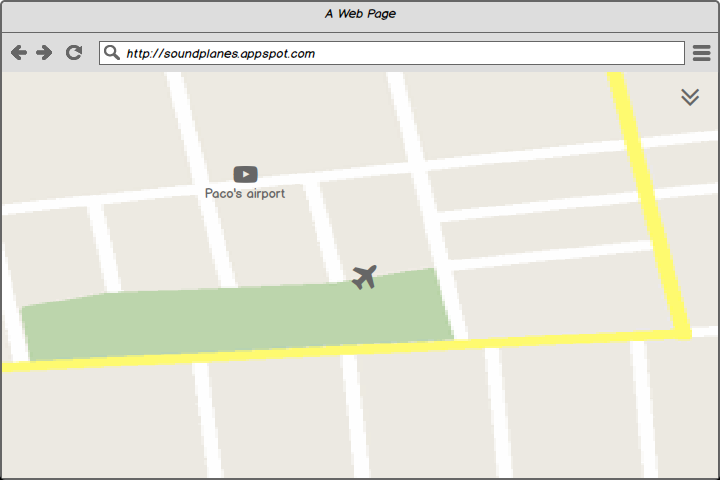


Figure 4. Main view's user prototype interface

## Profile menu view

In this tab it is displayed user information, where he can change his nickname or link his account with Spotify or Facebook. You can also enter your airport’s menu from here.

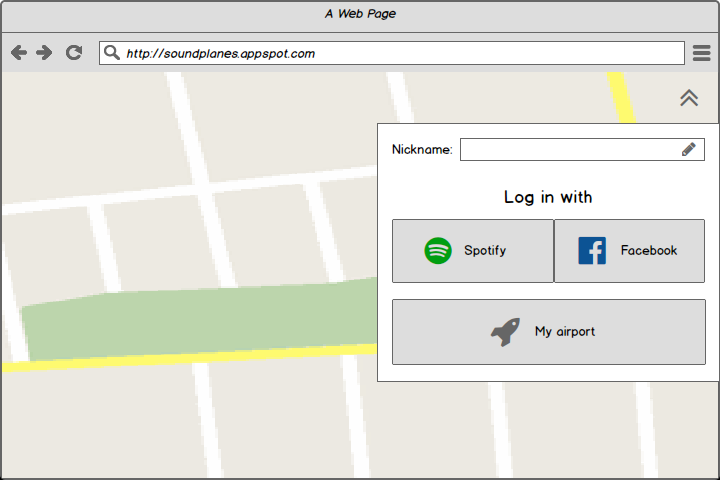


Figure 5. Profile menu view's user interface prototype

## Airport menu view

Here is where it’s displayed user’s airport information. User can change airport’s name, see the number of visitors that are currently in the airport, and the total number of visitors overall. User can also change the actual playlist of the base or delete the airport.

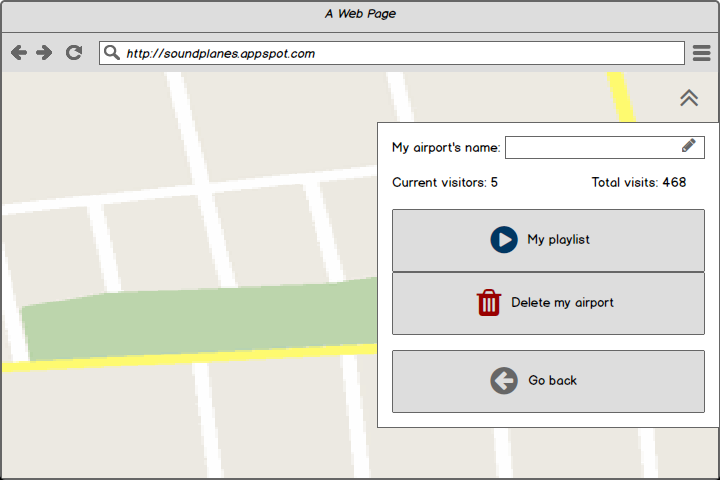


Figure 6. Base menu view's user interface prototype

## Airport chat view

Here is the chat of the airport where users can chat with each other.

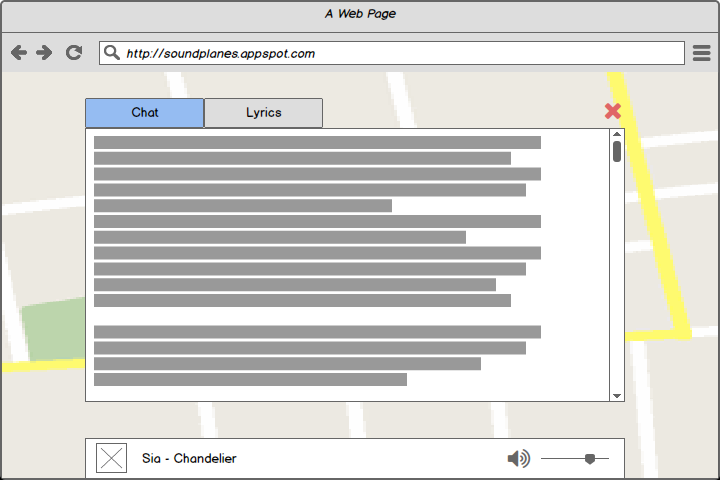


Figure 7. Airport chat view's user interface prototype

## Airport lyrics view

Lyrics of the song that is currently playing in the airport.

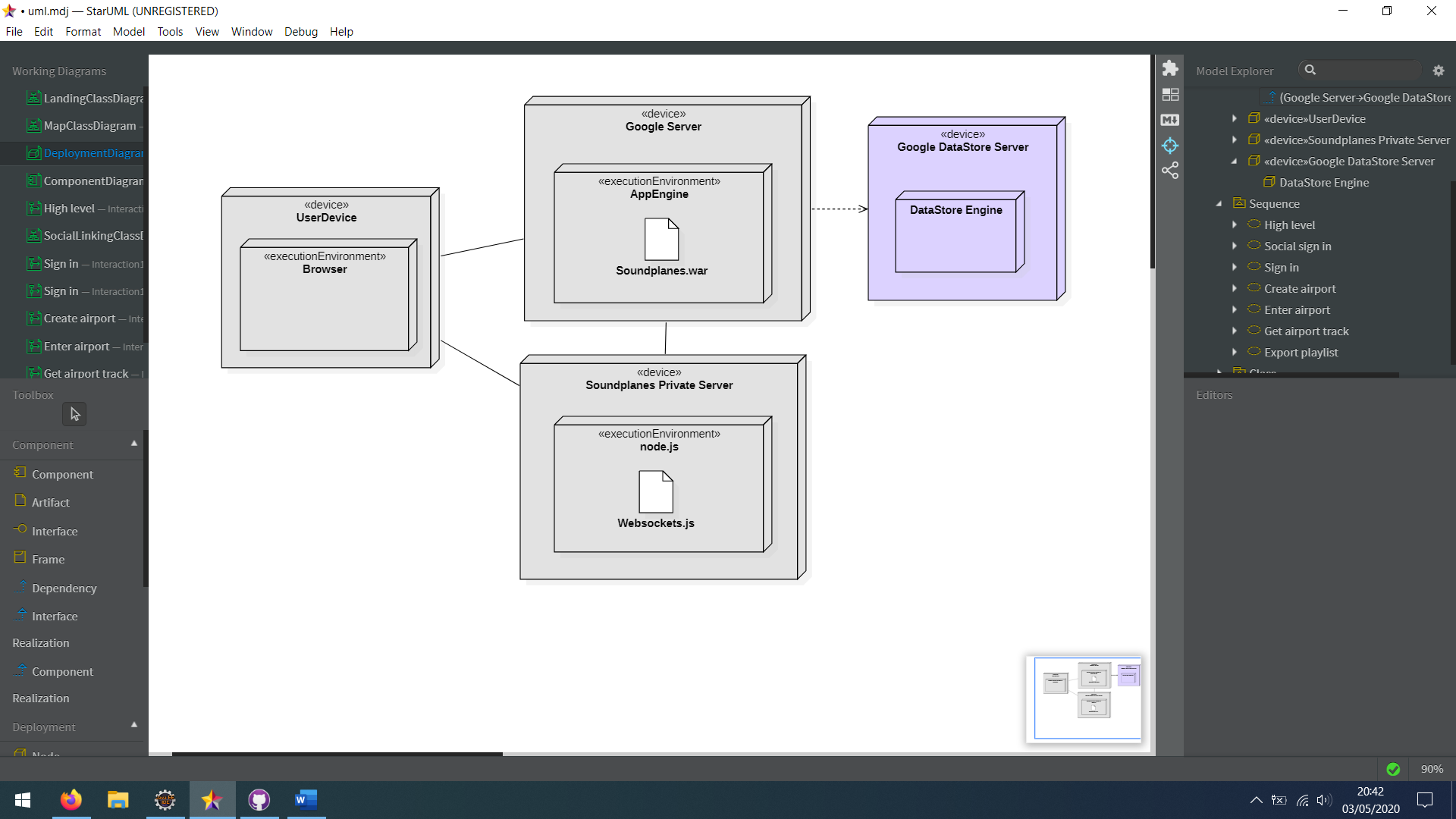


Figure 8. Airport lyrics view's user interface prototype

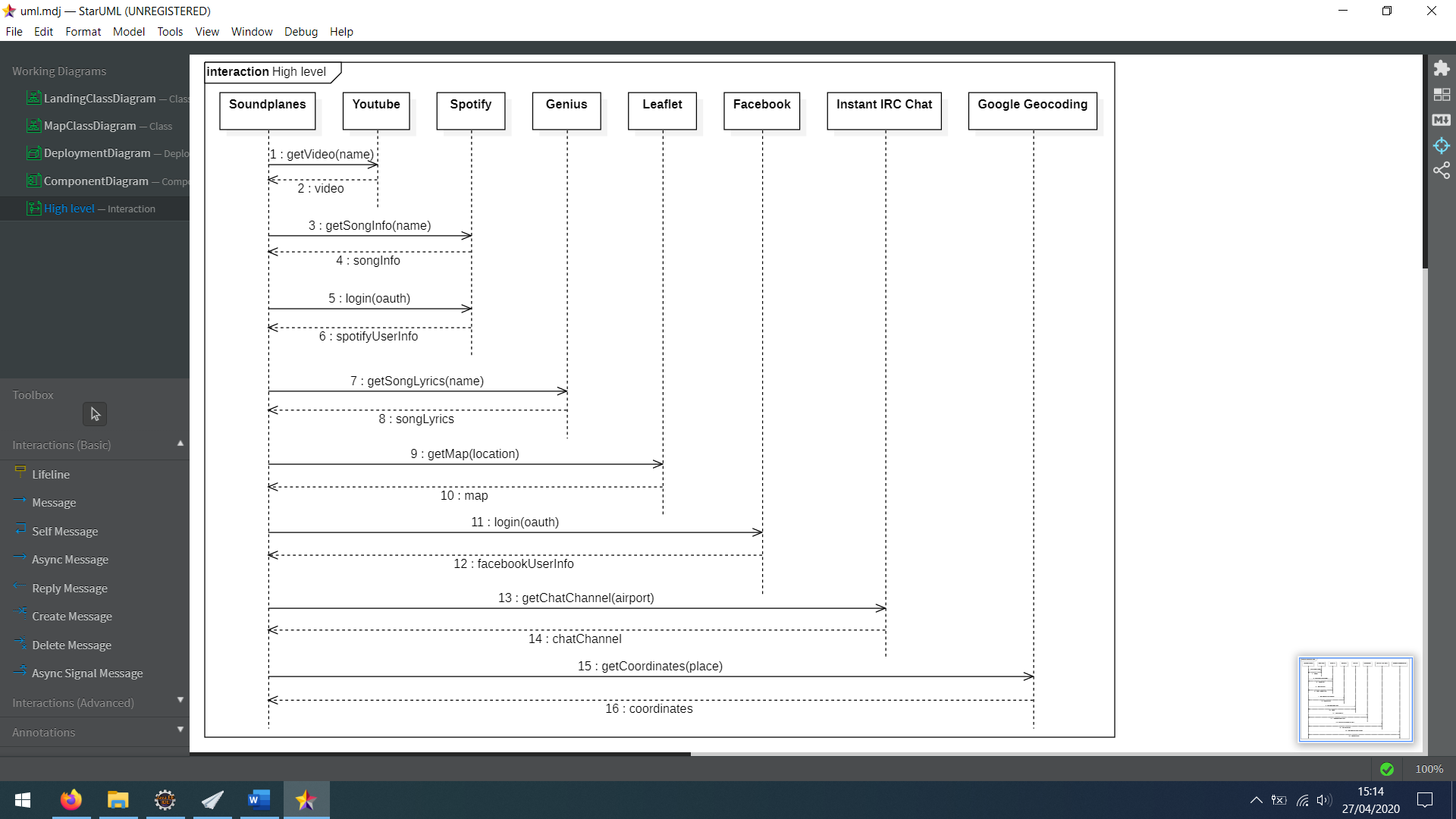
# Architecture

## Component diagram

## Deployment diagram

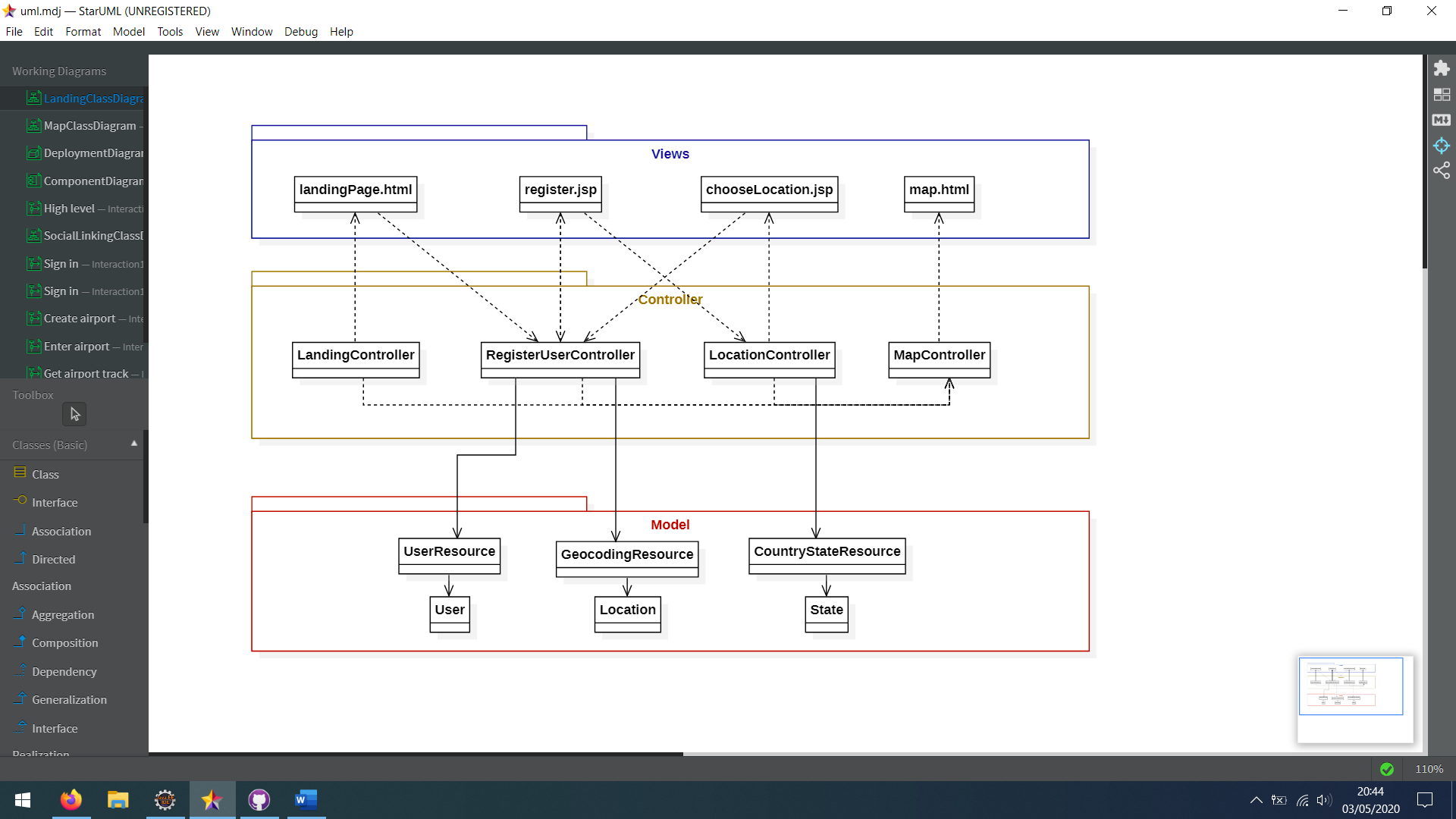


## High-level sequence diagram



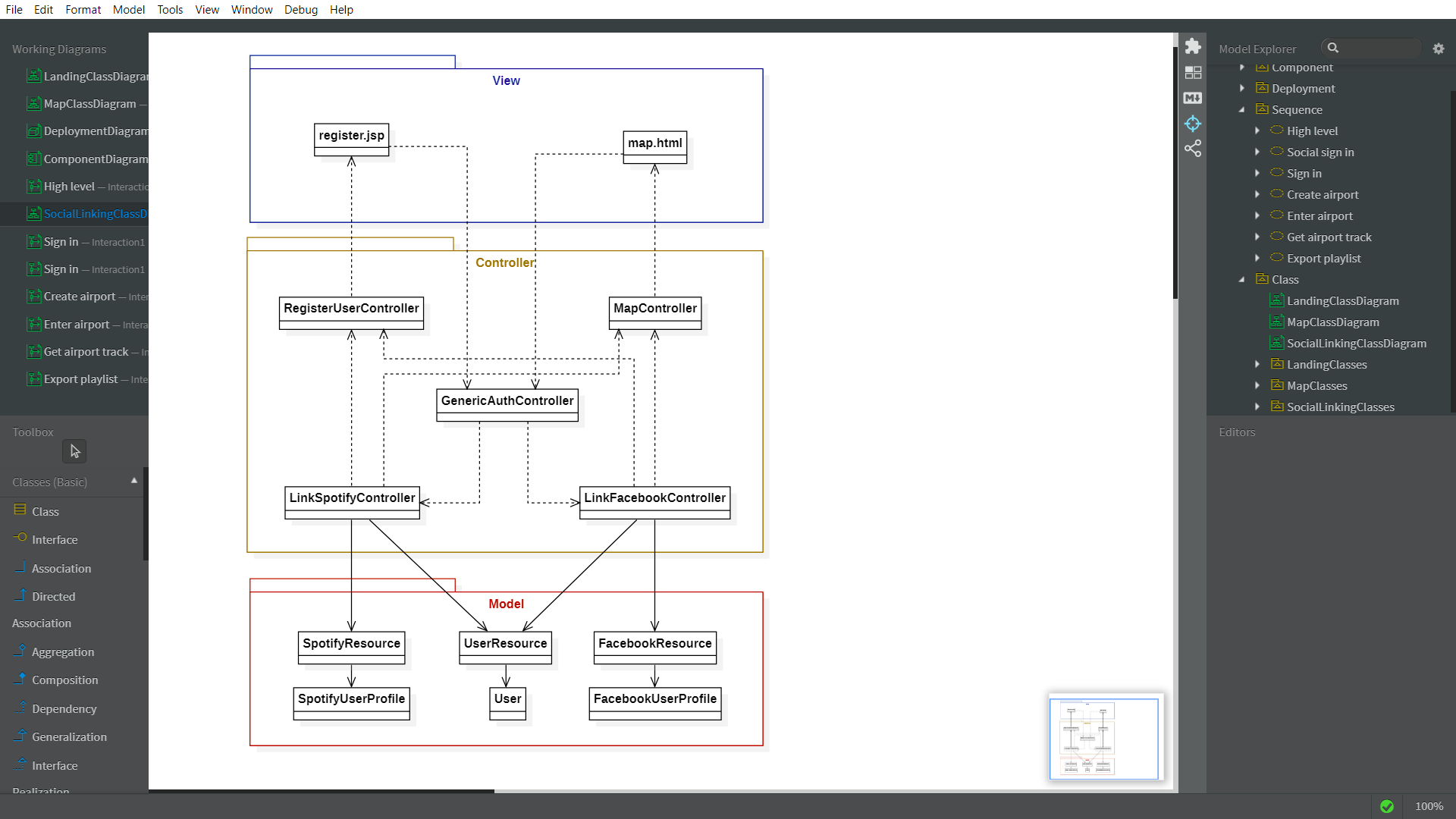
## Class diagram

### Landing/sign in class diagram



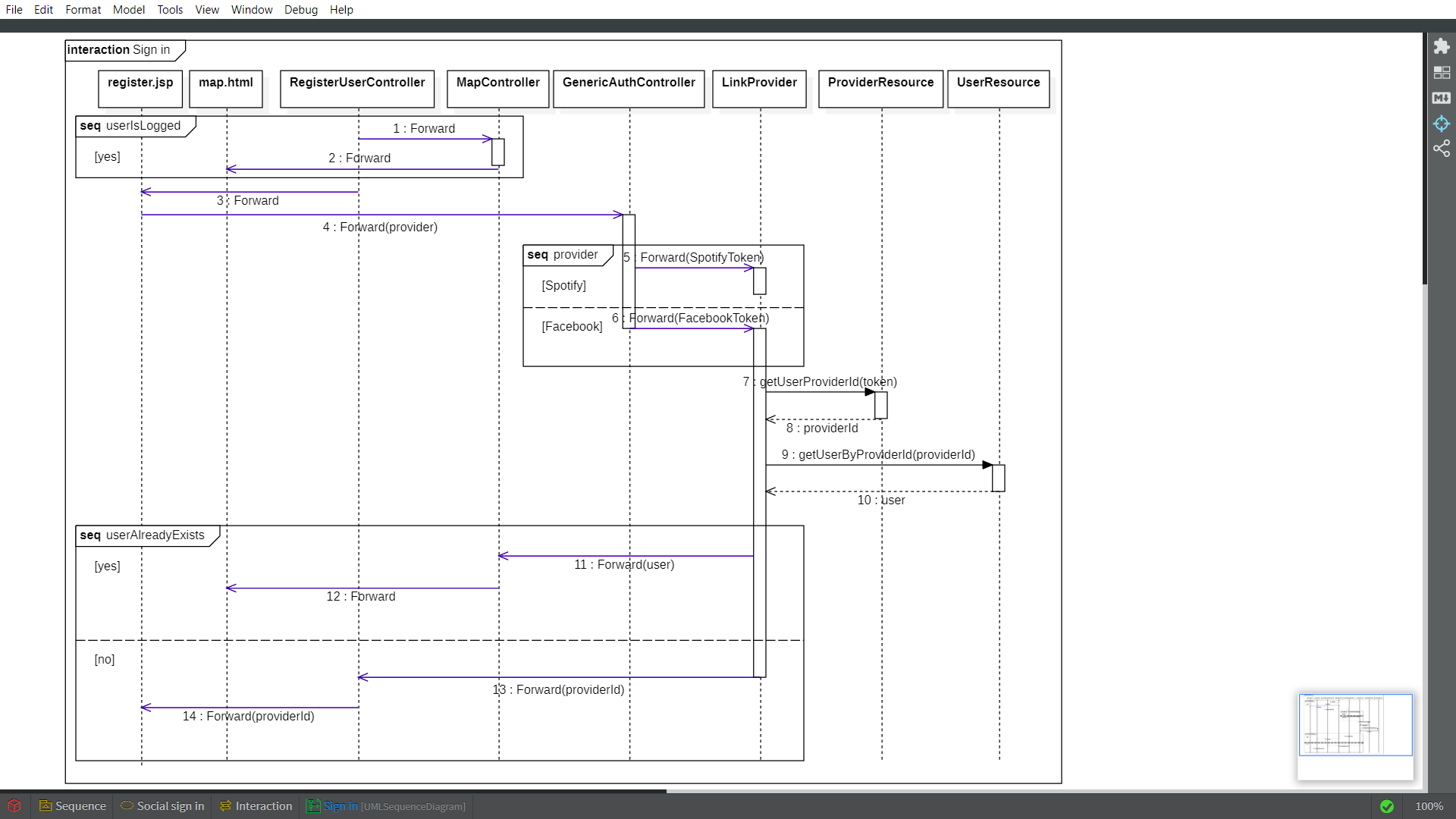
### Map class diagram

### Social sign in/linking class diagram

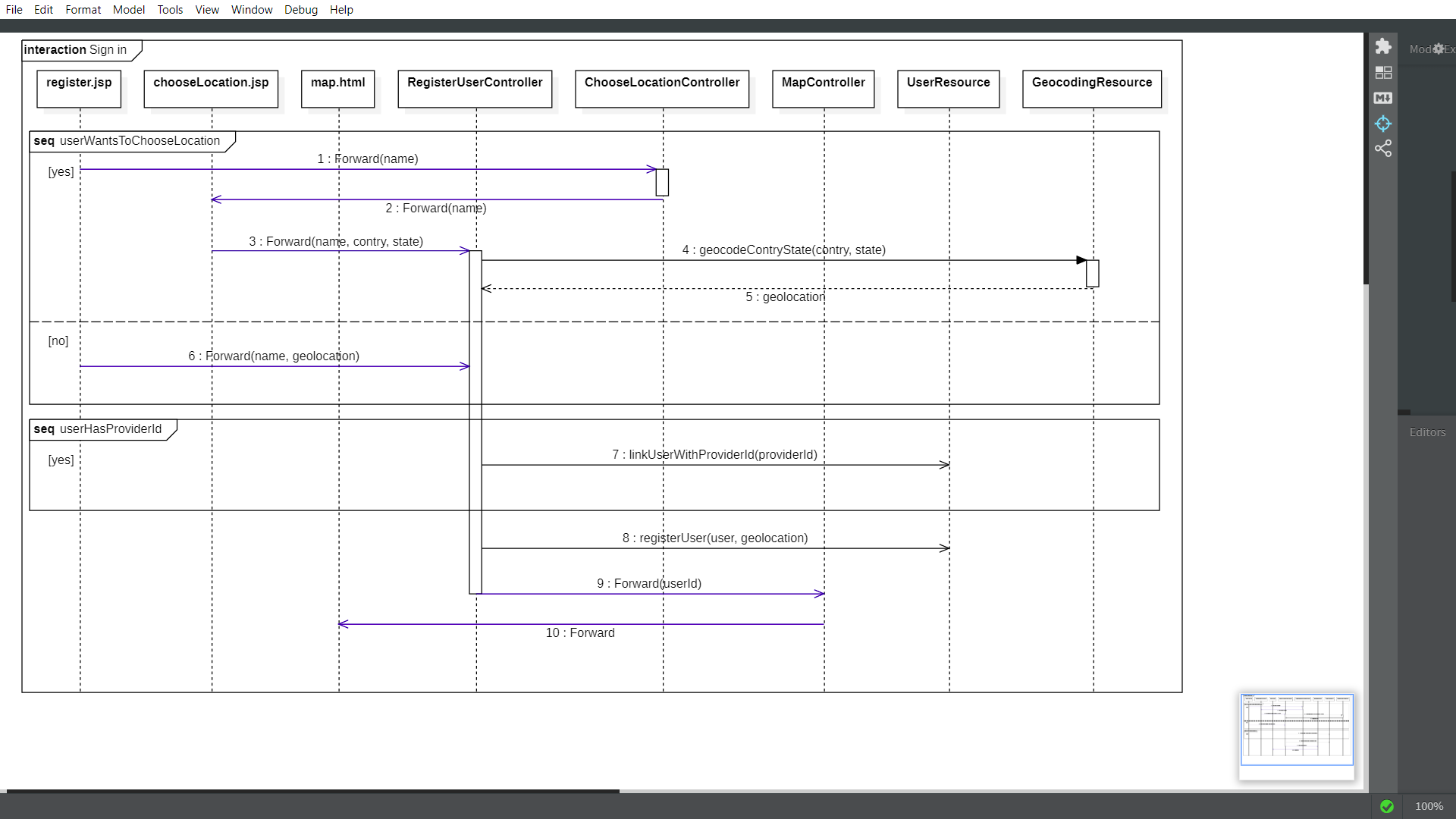


## Sequence diagram

### Social sign in sequence diagram

Some generalizations have been made in this diagram. “LinkProvider” and “ProviderResource” refers to both tuples “LinkSpotify”, “SpotifyResource” and “LinkFacebook”, “FacebookResource”. Their behaviour is the same, this is just a short name for those classes.

### Sign in sequence diagram



### Create airport sequence diagram

### Enter airport sequence diagram

### Get airport track sequence diagram

### Export playlist sequence diagram

# Implementation

Soundplanes aims to be an innovative, modern and simple way to gather people around the world. Its because of this that we needed to take our application to the next level.

The core of Soundplanes is made entirely with Java. It runs on Google AppEngine servers. However, instead of being a generic, *non-stated* mashup, we have decided to use Google DataStore API, in order to save registered users and their data. Objectify makes the hard work for us: local singleton repositories have been replaced with static objectify calls that communicates with the low-level DataStore API in order to save and retrieve data in a transparent way for our core.

We have made use of Bootstrap and JQuery for the client side. This makes our web design fully responsive. Our client will try their best to offer the best user experience using the available browser features, no matter the device.

As the nature of our data is dynamic (moving around a map, chatting…), we used websockets in the client side. These websockets opens a connection with the API servers.

# Tests

|  |  |
| --- | --- |
| Summary |  |
| Total number of tests performed | 31 |
| Number of automated tests | 25 |

|  |  |
| --- | --- |
| ID | AirportResource\_Test |
| Description | Check all CRUD (Create Read Update Delete) methods of the AirportResource are valid. |
| Input | A valid user uuid |
| Expected output | Airport and AirportPlaylist are created, read and deleted successfully. |
| Result | Success |
| Automated | Yes |

|  |  |
| --- | --- |
| ID | CountryStatesResource\_Test |
| Description | Check if states are loaded successfully |
| Input | Country Spain and invalid country |
| Expected output | Return states of Spain and 0 states for invalid country |
| Result | Success |
| Automated | Yes |

|  |  |
| --- | --- |
| ID | GeniusResource\_Test |
| Description | Retrieves the lyrics of a song |
| Input | A song name |
| Expected output | The corresponding song lyrics |
| Result | Success |
| Automated | Yes |

|  |  |
| --- | --- |
| ID | GeocodingResource\_Test |
| Description | Returns the geolocation of an address |
| Input | An address |
| Expected output | A geolocation object corresponding to the address |
| Result | Success |
| Automated | Yes |

|  |  |
| --- | --- |
| ID | IrcChatResource\_Test |
| Description | Creates a channel and retrieves valid tokens for an user |
| Input | A secret key and user information |
| Expected output | A channel and a valid token to connect to that channel |
| Result | Success |
| Automated | Yes |

|  |  |
| --- | --- |
| *ID* | SpotifyResource\_Test |
| Description | Check all CRUD (Create Read Update Delete) methods of the SpotifyResource are valid. |
| Input | An user token |
| Expected output | Playlist of that user is successfully created, read, updated and deleted |
| Result | Success |
| Automated | No |

|  |  |
| --- | --- |
| ID | FacebookResource\_Test |
| Description | Return the profile of a given user |
| Input | An user token |
| Expected output | The profile of the given user |
| Result | Success |
| Automated | Yes |

|  |  |
| --- | --- |
| ID | UserResource\_Test |
| Description | Check all CRUD (Create Read Update Delete) methods of the UserResource are valid. |
| Input | An uuid for the user |
| Expected output | User is successfully created, read, updated and deleted |
| Result | Success |
| Automated | Yes |

# User’s Manual

## Mashup

It hasn't been done yet.

## API REST

Note: this is not our definitive API, neither the format is. This is just for the second deliverable activity.

**GET /user/:id**

Returns information about a user.

**Path Parameters**

Id: A unique user identifier

**Response Format**

On success the HTTP status code in the response header is 200 OK.

{

Id: {id},

name: {name},

country: {country},

state: {state}

}

On error the HTTP status code is 404 Not Found if the given id doesn’t exist.

**POST /user**

Creates a user.

**Body Parameters**

{

name: {name},

country: {country},

state: {state}

}

**Response** **Format**

On success the HTTP status code in the response header is 201 Created.

{

Id: {id},

location: {

lat: {latitude},

lon: {longitude}

}

}

On error the HTTP status code is 400 Bad Request if the body parameters aren’t correct.

**PUT /user/:id**

Modifies information of a given user.

**Header Fields**

Authorization: Required. A valid access token to modify the given user.

**Path Parameters**

Id: A unique user identifier

**Body Parameters**

{

name: {name}

}

**Response** **Format**

On success the HTTP status code in the response header is 200 OK.

On error the HTTP status code is 404 Not Found if the given id doesn’t exist, 401 Unauthorized if Access token isn’t valid.

**DELETE /user/:id**

Deletes a given user.

**Header Fields**

Authorization: Required. A valid access token to delete the given user.

**Path Parameters**

Id: A unique user identifier

**Response** **Format**

On success the HTTP status code in the response header is 200 OK.

On error the HTTP status code is 404 Not Found if the given id doesn’t exist, 401 Unauthorized if Access token isn’t valid.

**GET /user/:id/playlist**

Returns the playlist of the given user.

**Path Parameters**

Id: A unique user identifier

**Response** **Format**

On success the HTTP status code in the response header is 200 OK.

An array of songs

{

Playlist: [

{

Id: {songId},

name: {songName}

artist: {artistName}

},

…

]

}

On error the HTTP status code is 404 Not Found if the given id doesn’t exist, or the player doesn’t have a playlist.

**POST /user/:id/playlist**

Creates a playlist for the user

**Header Fields**

Authorization: Required. A valid access token to create a playlist for the given user.

**Path Parameters**

Id: A unique user identifier

**Body Parameters**

An array of songs.

{

Playlist: [

{

Id: {songId},

name: {songName}

artist: {artistName}

},

…

]

}

**Response Format**

On success the HTTP status code in the response header is 201 Created.

On error the HTTP status code is 404 Not Found if the given id doesn’t exist, or 400 Bad Request if the body parameters are not correct.

**PUT /user/:id/playlist**

Add songs to a user playlist

**Header Fields**

Authorization: Required. A valid access token to modify a playlist for the given user.

**Path Parameters**

Id: A unique user identifier

**Body Parameters**

An array of songs.

{

Playlist: [

{

Id: {songId},

name: {songName}

artist: {artistName}

},

…

]

}

**Response Format**

On success the HTTP status code in the response header is 200 OK.

On error the HTTP status code is 404 Not Found if the given id or playlist doesn’t exist, or 400 Bad Request if the body parameters are not correct.

**DELETE /user/:id/playlist**

Removes a user’s playlist

**Header Fields**

Authorization: Required. A valid access token to delete a playlist for the given user.

**Path Parameters**

Id: An unique user identifier

**Response Format**

On success the HTTP status code in the response header is 200 OK.

On error the HTTP status code is 404 Not Found if the given id or playlist doesn’t exist, or 401 Unauthorized if the Access token isn’t valid.

**GET /user/:id/airport**

Returns information of a user’s airport.

**Path Parameters**

Id: A unique user identifier

**Response Format**

On success the HTTP status code in the response header is 200 OK.

{

name: {airportName},

location: {

lat: {latitude},

lon: {longitude{

}

}

On error the HTTP status code is 404 Not Found if the given id or airport doesn’t exist.

**POST /user/:id/airport**

Creates an airport for the given user.

**Header Fields**

Authorization: Required. A valid access token to create an airport for the given user.

**Path Parameters**

Id: A unique user identifier

**Response Format**

On success the HTTP status code in the response header is 201 Created.

On error the HTTP status code is 404 Not Found if the given id doesn’t exist, or 401 Unauthorized if the Access token isn’t valid.

**PUT /user/:id/airport**

Change an airport information for the given user.

**Header Fields**

Authorization: Required. A valid access token to modify an airport for the given user.

**Path Parameters**

Id: A unique user identifier

**Response Format**

On success the HTTP status code in the response header is 200 OK.

On error the HTTP status code is 404 Not Found if the given id or airport doesn’t exist, or 401 Unauthorized if the Access token isn’t valid.

**DELETE /user/:id/airport**

Deletes an airport for the given user.

**Header Fields**

Authorization: Required. A valid access token to delete an airport for the given user.

**Path Parameters**

Id: A unique user identifier

**Response Format**

On success the HTTP status code in the response header is 200 OK.

On error the HTTP status code is 404 Not Found if the given id or airport doesn’t exist, or 401 Unauthorized if the Access token isn’t valid

**GET /country/:name**

Returns information about the users, songs playing…

**Path Parameters**

name: A unique country identifier in Alpha-3 code (ISO 3166)

**Response Format**

On success the HTTP status code in the response header is 200 OK.

On error the HTTP status code is 404 Not Found if the given name doesn’t exist.

**GET /country/:name/top**

Returns top songs of a given country.

**Path Parameters**

name: A unique country identifier in Alpha-3 code (ISO 3166)

**Response Format**

On success the HTTP status code in the response header is 200 OK.

On error the HTTP status code is 404 Not Found if the given name doesn’t exist.

**GET /country/:name/airport**

Returns information about the international airport of a country.

**Path Parameters**

name: A unique country identifier in Alpha-3 code (ISO 3166)

**Response Format**

On success the HTTP status code in the response header is 200 OK.

On error the HTTP status code is 404 Not Found if the given name doesn’t exist.

# References

[1] *Balsamiq*. <http://balsamiq.com/>. Accessed January 2014.

[2] J. Webber, S. Parastatidis y I. Robinson. *REST in Practice: Hypermedia and Systems Architecture.* O'Reilly Media. 2010.