# Design and development of a social jukebox service for iOS

· Final thesis ·

Wai Ling Tam

October 2016

Software Engineering

Director: María José Casany

# Abstract

This project aims to give customers a say regarding the music that is being played in an establishment besides encouraging social interaction among people staying at the same spot. A system that allows users to vote for the tracks on the current song playlist as well as submit their own song requests is created. As a result, users are able to see the ones who have contributed, therefore look into their profiles and have the option to contact them via social media because of the common music taste or shared hang out spots.

# Resum

Aquest projecte té com a objectiu donar als clients la oportunitat de donar la seva opinió respecte la música que està escoltant i per fomentar la interacció social entre les persones que estan en un mateix espai. Un sistema que permet als usuaris votar per les cançons de la llista de cançons i també fer peticions de cançons és creada. Com a resultat, els usuaris poden veure els que han contribuït i així, visitar els seus perfils i tenir la opció contactar amb ells via les xarxes socials pel gust musical similar o els llocs en comú on han anat.

# Index

# 1. Introduction

## 1.1 Formulation of the problem

Music has always been part of human's life. Since the beginning of ancient history, music has been performed for many purposes, ranging from aesthetic pleasure, religious or ceremonial purposes, or as an entertainment product for the marketplace. After all, its main goal is to gather people at the same place and enjoy the art of sound.

The arrival of the Internet and widespread high-speed broadband has turn music into a whole new experience. The increased ease of access to music via streaming videos and music streaming services has made the choice and discovery of music unlimited.

Nowadays music is being played in almost any establishment that doesn't require silence in order to make customers feel welcomed and entertained. However, there are other visitors who prefer to listen to their own music and be disconnected from their surroundings. On the one hand, they might simply want to keep themselves concentrated in their tasks, on the other hand, assuming the less desirable cause, they are not fond of the music that is being played in their current location.

From the facts stated above, this project aims to take into account the initial end of music and its present situation in public spaces. The target is to give customers and visitors a say regarding the music they are listening to and to encourage social interaction among people staying at the same spot. A system that will allow users vote for the tracks on a currently playing song playlist as well as submit their own song requests will be created. As a result, users will most likely enjoy more their stay in an establishment and will be able to see the ones who have contributed, look into their profiles and have the option to contact them via social media because of the common music taste or shared hang out spots.

## 1.2 Objectives

Now that the problem has been set, the next step is to analyze the different objectives intended to achieve with this project.

- Help users discover new sounds and expand their music horizon.
- Encourage people reach out and meet other users of the product.
- Help visitors discover new places taking into account their music taste and hang out spots.
- Make customers' stay in a locale more enjoyable.
- Help establishments promote themselves thanks to the contributive music system.
- Create a simple and intuitive user interface which will allow a seamless usage of the product.
- Build a software that manages a locale's current song playlist which is in constant change because of the voting and the song requests.
- Build a system that handles user and establishment profiles.

## 1.3 Context

The main product of this project is a mobile application, most commonly referred as an app, it is a type of application software designed to run on a mobile device, such as a smartphone or a tablet. Apps are generally small, individual software units with limited and isolated functionality such as a game or a calculator. Nowadays, mobile phones so as mobile apps are so integrated into our lives that many people feel uncomfortable without them. It makes sense that the product of this project is a mobile app because the main point is that users will be able to use it wherever they go.

The mobile app will run on iOS, which is a mobile operating system created and developed by Apple Inc. and distributed exclusively for Apple hardware. It is the second most popular mobile operating system in the world by sales. iOS apps are written in Swift or Objective-C (with some elements optionally in C or C++) and compiled specifically for iOS and the 64-bit ARM architecture or previous 32-bit one [1].

Regarding the music catalog and management, these will be provided by Spotify. Spotify is a Swedish commercial music streaming, podcast and video service that provides digital rights management–protected content from record labels and media companies. It is the biggest name in music streaming services currently and has more than 100 million users worldwide [2] [3].
The Spotify API is based on simple REST principles, their API endpoints return metadata in JSON format about artists, albums, and tracks directly from the Spotify catalog. The API also provides access to user-related data such as playlists and music saved in users' music library, subject to their authorization [4].

## 1.4 State of art

At this moment, there are already a few social jukebox applications in the market. All of them have the common features of a social music player and other aspects that make them more distinctive.
Up next, a series of apps are going to be presented with a list of characteristics and functionalities. The purpose is to study what is already out in the market and which are the factors that differentiate them from each other. As a result, a unique application can be build and it will have the chance to stand out from its competitors.

- **Rockbot** [5]
  - **Technology:** iOS and Android app, webapp for hosts (people in charge of the playlist)
  - **Music provider:** They have their own selection of licensed commercial free music (14 million songs)
  - **Common:** make requests, vote
  - **Regular:**
    - Import your own playlists from Spotify or iTunes.
    - Download songs that are currently playing.
  - **Unique:**
    - Learn the preferences of managers and guests over time and continually improve the quality of the background music.
    - Rockbot's algorithms create balance in the soundtrack and guarantee that songs are never overplayed.
    - Rockbot automatically plays different music at different times of the day to match the mood of the business.

- Disable explicit lyrics or disable it based on time of day.
- Ban any artist or song.
- Pick from hundreds of stations tailored to a specific industry or design new playlists.
    - A centralized online dashboard and instant remote control via mobile.
    - A remote control for the store managers. The Rockbot Manager App enables them to easily skip a song, change playlists, or adjust the volume on the fly.
    - Set permissions at any level. Give every location, franchisee, manager and employee as much or little control.
    - Users need credits to request songs.
    - Share song plays on social media with photos and 'tagged' friends.
    - Sync music from a device, Facebook, Spotify and Last.fm.
    - Link social networks to connect with users and businesses.
    - Unlock rewards, enter giveaways, and view current specials at the venue.

- **OutLoud** [6]
    - **Technology:** iOS app and webapp
    - **Music provider:** Spotify
    - **Common:** make requests, vote and downvote
    - **Regular:**
        - Mix the user's Spotify and iTunes playlists with the ones from friends.
        - Add songs from your own Spotify, Apple Music and OutLoud playlists or from Spotify's and OutLoud's top charts.
    - **Unique:**
        - Display the OutLoud playlist onto the TV via Google Chromecast or the web-based TV app through a laptop.
        - Every playlist has a customizable web link making it very easy to join.
        - Party social network. Chat and share pictures with everyone on the playlist. They're only visible to those at the party and when the host closes the playlist, they all disappear.
        - See what songs friends added and voted for.

- **Jukestar** [7]
  - **Technology:** Android app and webapp
  - **Music provider:** Spotify
  - **Common:** make requests, vote and downvote
  - **Regular:**
    - Hosts can overrule any song and force any song to play next.
  - **Unique:**
    - The upcoming queue is distributed and updated automatically based on who requested it, when they requested it and what other guests think of the songs.
    - Guest requests will be spread out in the queue so everyone's music gets played.
    - If enough guests veto a song it disappears from the playlist, even the currently playing song.
    - Jukestar will make sure the same song isn't repeated on the night.
    - Share playlist with guests via social media

- **SecretDJ** [8]
  - **Technology:** iOS and Android apps
  - **Music provider:** local music catalog
  - **Common:** make requests
  - **Regular:**
    - Option to preview a song, download from iTunes or save to Spotify.
  - **Unique:**
    - Location savvy: Secret DJ shows nearby venues that are part of the Secret DJ network and what music is on.
    - Check-in at the venues and let friends know where you are.
    - In order to request songs, users have to buy credits.
    - Secret DJ rewards the venue's loyal customers with extra songs. If a user selects enough tunes and he might become the DJ of the bar and always jump the queue.
    - Allows the user share all the songs, people and places he likes via Twitter and Facebook.
    - It has a music related news feed.
    - Establishment profile: address, likes, current music, Facebook account, Twitter account, webpage, photo, future events

- Users profile: venues visited, tunes requested, favorite tunes, last seen venue, profile picture

After reviewing the features from the applications above, the following conclusions are extracted:

- None of the applications addresses the objectives of this project in the entirety.
- SecretDJ is the one that comes closer to the app to be built. It has establishment and user profiles but it doesn't allow voting neither has a way to contact other users.
- OutLoud and Jukestar have Spotify as their music provider, however, these apps have their focus on private parties, not public places.
- Rockbot has plenty of interesting and useful features. But it seems to be more centered on the host side, on the management of the music and not the exploration of new music and places.
- The app of this project will gather characteristics from each of them and others that haven't appeared yet.

# 2. Scope

First of all, there are two types of users in the main audience of the application: the hosts of establishments and the customers. An iOS application will be developed, which is the way users will interact with the whole system. The application must be install

On the backend, a REST API implemented with the Django framework will give access to a database where information in relation to users, establishments and the song playlist management will be located. There will be an admin site for people in charge of the administration of the application, which the Django framework provides by default. Everything stated (admin site, API and database) is deployed on Heroku, a cloud Platform-as-a-Service (PaaS).

As to the music player, the establishment owners must use Spotify as its music streaming service since the application will fetch data from the Spotify music catalog and manage the user's current song playlist.

## 2.1. In scope

- The audience of the app are establishment owners and customers.
- A user can vote and request songs of a playlist currently playing in an establishment.
- A user has a profile with his establishment check-ins, favorite artists, favorite genres, requested songs, voted songs and social media links.
- A user can visit other users' profile.
- A user can see the voters of a song and the user who requested the song if given the case.
- An establishment has a profile with its address, currently playing song and current song playlist.
- A same user can perform as a host and a guest.
- A user has to sign up and log in in order to use the app.
- A host can set if songs with explicit lyrics are allowed.
- The app uses geolocation in order to display user's current location.
- The app uses Spotify as its music provider.
- The app has an admin site.

- It is a mobile application with iOS as its operating system.
- Worldwide availability.

## 2.2. Out of scope

- Down voting of songs is not available.
- The app doesn't have a chat.
- The app is not addressed to private events.
- The app is not multilingual.
- There won't be a user report system.
- The host can't ban artists or songs.
- The user doesn't need credits in order to vote or make a request.

## 2.3. Possible obstacles and solutions

### 2.3.1. Business risks

- **Obstacle**: Lack of users in some establishments, therefore the social interaction factor is lessened.
    - **Solution**: Promote the app and the establishments which are involved with it.
- **Obstacle**: Users abundance in some establishments, to the point that affects the efficiency of the servers and the user experience.
    - **Solution**: Upgrade and/or acquire more servers and improve the client-side of the app.
- **Obstacle**: Users who don't use the app correctly and vote and make requests carelessly.
    - **Solution**: Limit the number of votes and requests per hour.

### 2.3.2. Technical risks

- **Obstacle**: The Heroku platform in under maintenance, as a result, developers won't be able to work properly.
    - **Solution**: Execute tasks that don't require Heroku.
- **Obstacle**: Lack of knowledge regarding the development tools, programming language and APIs.
    - **Solution**: look for help in the available documentation or on the Internet.

- **Obstacle**: Unauthorized access to the database.
    - **Solution**: Implement an authentication method to the admin site. All user passwords must be encrypted.

### 2.3.3. Development risks

- **Obstacle:** Requirement changes.
    - **Solution:** Application of an agile methodology.
- **Obstacle**: Insufficient risks management
    - **Solution**: Revise and improve the risks list.

## 2.4. Methodology and rigor

Because of the amount of time available to carry out the project, following an agile approach, specifically the SCRUM methodology, will be the best choice.
Although agile methodologies are very team-oriented, there are many concepts and ideas from which an individual project can take advantage of.

### 2.4.1. Short development cycles

Brief iterations provide flexibility to changes, fast development and results in less time than any other ordinary methodology. Plus, it facilitates the prediction of a more accurate time and reduces some specific risks.

### 2.4.2. Constant feedback

From the project management to the implementation of the project, a very important part of the process is receiving constant information and responses of the client. In this case, the GEP tutor and the project director, who give support on how to improve the project and mention the mistakes that must be fixed. As a result, chances of misunderstandings are greatly reduced and, given the case they occur, they are corrected much sooner. To sum up, any kind of feedback has a positive impact on the project.

## 2.5. Monitoring Tools

As this project is about implementing a software, a suitable monitoring tool would be an online repository, so an easy access to the code from any platform and work station is guaranteed.

On the one hand, Git will be used to store the code and document it, due to the fact that every time a commit is done a message has to be left. Besides this toughens the idea of short iterations. Github, a web-based Git repository hosting service is used, which also offers a ticket system, ideal for setting milestones and tracking the project's progress.

## 2.6. Validation methods

Thanks to the agile methodology, there is going to be periodic and constant validations for a correct evolution of the project. There will be scheduled meetings at the end of each work iteration with the project director, which will assure that the project is following the right path and methods.

# 3. Time planning

Firstly, the development of this project had an estimated duration of 5 months at the beginning, although due to various reasons (explained in a following section), it ended up having a duration of approximately 8 months, starting from February 15th until October 17th 2016. Therefore, the schedule has been shaped and modified given these time constraints. The planning of the project is divided in different phases, thus in the upcoming sections, details of each phase as well as the duration and the human and material resources required are specified.

## 3.1. Phases

### Phase I: Project management

This is the stage of the project where the elaboration of documents related to the project management takes place.

Its parts are:

- Scope definition
- Time planning
- Economic management and sustainability
- State of art and references

Resources needed:

- Human
    - Project manager
- Material
    - Hardware: MacBook Pro Retina 15”
    - Software: OSX El Capitan, Microsoft Office 2016

### Phase II: Analysis and design

The goal of this section is to do an in-depth analysis of the project and specify the final design.
As for the analysis, the objectives that this project aims to complete are set during this process, together with the requirements, the features and the use cases. Furthermore, the market research done beforehand comes in handy during the analysis of the competitors’ app features.
Lastly, a software architecture and a graphical user interface are created in relation to the design of the project.

**Previous requirements:** Project management
**Resources needed:**

- Human
    - Designer, Analyst
- Material
    - Hardware: MacBook Pro Retina 15”
    - Software: OSX El Capitan, Microsoft Office 2016

### Phase III: Project iterations and task specification

Due to the fact that the methodology followed in this project is agile, a strict list of tasks for the different iterations cannot be listed at once. Whereas a set of objectives are set each week adjusting the process to possible deviations that could surface. Nevertheless, a set of general tasks with a strict sequence can be defined.

1. **Task specification**
   Throughout the different iterations, a list of tasks is set before the execution of development tasks. The previous iteration is reviewed, which involves the amount of tasks completed, then the work to be done, time left and the available budget is considered in order to arrange the new tasks.

   **Previous requirements:** Project management, Analysis and design
   **Resources needed:**
   - Human
       - Analyst
   - Material
       - Hardware: MacBook Pro Retina 15”
       - Software: OSX El Capitan, Microsoft Office 2016, Github

2. **Working environment configuration**
   This iteration is oriented to prepare the environment, install and configure the necessary frameworks and programs to develop correctly the application.

   **Previous requirements:** Task specification
   **Resources needed:**
   - Human
       - Programmer
   - Material

- Hardware: MacBook Pro Retina 15”, iPhone 5c, iPad mini
- Software: OSX El Capitan, iOS 9, Xcode 7, PyCharm, pgAdmin, Github, Heroku

3. **Development of the data access layer (the back end)**

   This is the main stage of the project. It covers all tasks related to the implementation and the testing of the application. The development begins with the back end, more specifically tasks related to the creation of users and establishments (registration, log in, profile).

   Next, tasks related to the Spotify API and music management are completed, such as obtaining the establishment’s current song playlist and more importantly, the entire voting system and the addition of songs from user requests.

   **Previous requirements:** Task specification, Working environment configuration

   **Resources needed:**

   - Human
     - Programmer
   - Material
     - Hardware: MacBook Pro Retina 15”
     - Software: OSX El Capitan, PyCharm, pgAdmin, Github, Heroku

4. **Development of the presentation layer (the front end)**

   Following up is the creation of the application’s user interface and connecting it to the back end.

   **Previous requirements:** Task specification, Working environment configuration, Back end development

   **Resources needed:**

   - Human
     - Programmer
   - Material
     - Hardware: MacBook Pro Retina 15”, iPhone 5c, iPad mini
     - Software: OSX El Capitan, iOS 9, Xcode 7, Github

5. **Final tests and debugging**

   Lastly, the final testing and debugging is carried out. Although during the execution of the implementation both tasks are done whenever necessary.

   **Previous requirements:** Task specification, Working environment configuration, Back end development, Front end development

**Resources needed:**

- Human
    - Programmer, Tester

- Material
    - Hardware: MacBook Pro Retina 15”, iPhone 5c, iPad mini
    - Software: OSX El Capitan, iOS 9, Xcode 7, PyCharm, pgAdmin, Github, Heroku

### Phase IV: Final documentation and delivery

In the last phase, parts missing in the report are completed, then it is presented alongside the appendix, where the code, the manual of the application, plus other relevant documents are included.
It is important to note that the documentation is updated throughout the completion of the different phases.

From beginning to end of the execution of the project, learning is a big part of it. It consists of getting to know the technology used as well as the design and parts of the documentation.

**Previous requirements:** Task specification, Working environment configuration, Back end development, Front end development, Final tests and debugging
**Resources needed:**

- Human
    - Project manager
- Material
    - Hardware: MacBook Pro Retina 15”
    - Software: OSX El Capitan, Microsoft Office 2016

## 3.2. Estimated duration

| Task | Estimated time (hours) | Time invested (hours) |
|---|---|---|
| Project management | 90 | 90 |
| Analysis and design | 60 | 60 |
| Task specification | 20 | 20 |
| Environment configuration | 20 | 20 |
| Back end development | 100 | 130 |
| Front end development | 80 | 110 |
| Testing and debugging | 30 | 30 |
| Learning | 50 | 100 |
| Documentation | 50 | 50 |
| **Total** | **500** | **610** |

Table 1: Summary of the time spent in each task.

Seeing the time estimation (*Table 1*) and the total number of hours needed is 500, the chance of finishing the project in 19 weeks seemed high if no major deviation appears unexpectedly.

## 3.3. Time deviation

Taking into account that the estimated time was calculated in a temporary planning, task durations, order and other attributes have varied during the execution of the project.

The main reason why the project duration has differed is due to the increased time invested in learning and a constant search of solutions to doubts during the development stage of the app. The small amount of knowledge regarding the selected technologies has been the cause of the time deviation. The expense on time to get familiar with them has surpassed the initial estimation.

Although the time available to develop the project has increased because of the extension of a total of three months, the list of goals to achieve has not been altered because the additional time wasn't spent on adding features to the product. To sum up, the cost on time has been increased considerably.

## 3.4. Resources

In order to carry out the project proposed, different human, hardware and software resources are used. Here are the lists along with the tasks that each resource is in charge of.

### 3.4.1. Human

- Project director: involved in every phase of the project
- Project manager: *Phase I – Project management*, *Phase IV – Final documentation and delivery*
- Analyst: *Phase II – Analysis and design, Phase III – Project iterations and task specification*
- Designer: *Phase II – Analysis and design*
- Programmer: *Phase III – Project iterations and task specification*
- Tester: *Phase III – Project iterations and task specification*

### 3.4.2. Hardware

- MacBook Pro Retina 15": used in every stage of the project.
- iPhone 5c: used during app testing.
- iPad mini: used during app testing.

### 3.4.3. Software

- OSX El Capitan: used in every task of the project.
- iOS 9: used during app development and testing.
- Xcode 7: used during the project user interface development.
- PyCharm: used during back end development.
- Microsoft Office 2016: used during the elaboration of the report.
- pgAdmin: used during back end development.
- Github: used during task specification and main development.
- Heroku: used during back end development.

# 4. Budget

This part of the document is going to provide the estimated cost of the project taking into consideration the human and material resources and the indirect costs.

In order to not surpass the estimated budget, the figures has been updated after each iteration. Therefore, the final budget will include the genuine working hours and the final cost.

## 4.1. Budget on human resources

This project is carried out by one person taking responsibility for five different roles, which are project manager, analyst, designer, programmer and tester. In the table below, the amount of hours of work needed in order to complete the project's tasks and the hourly rate for each role are shown.

| Role | Hours | Hourly rate | Total cost |
|---|---|---|---|
| **Project manager** | 170 | 45€ | 7650€ |
| **Designer** | 30 | 35€ | 1050€ |
| **Analyst** | 54 | 35€ | 1890€ |
| **Programmer** | 320 | 35€ | 11200€ |
| **Tester** | 36 | 25€ | 900€ |
| **TOTAL** | 500 | | **22690€** |

Table 2: Budget on human resources.

## 4.2. Budget on material resources

### 4.2.1. Budget on hardware

A series of hardware are required to complete tasks like documentation, implementation and tests.
Take into account that the reparation costs might vary but they should be around the values displayed.

| Product | Price | Units | Lifespan | Amortization |
|---|---|---|---|---|
| MacBook Pro retina 15” | 2799€ | 1 | 5 years | 100€ |
| iPhone 5c | 450€ | 1 | 3 years | 70€ |
| iPad mini | 329€ | 1 | 3 years | 60€ |
| TOTAL | 3578€ | 3 | | **230€** |

Table 3: Budget on hardware resources.

### 4.2.2. Budget on software

Software is going to be a significant part too. These are needed on every task of the project. Notice that if the lifespan is not specified, it means that the product has no expiration date. Plus, the updates are all free of charge.

| Product | Price | Units | Lifespan | Amortization |
|---|---|---|---|---|
| OSX El Capitan | 0€ | 1 | - | 0€ |
| iOS 9 | 0€ | 1 | - | 0€ |
| Xcode 7 | 0€ | 1 | - | 0€ |
| PyCharm | 199€ | 1 | 1 year | 159€ |
| Heroku | 0€ | 1 | - | 0€ |
| Microsoft Office 2016 | 149€ | 1 | - | 0€ |
| pgAdmin | 0€ | 1 | - | 0€ |
| Github | 0€ | 1 | - | 0€ |
| TOTAL | 348€ | 6 | | **159€** |

Table 4: Budget on software resources.

### 4.2.3. Indirect costs

Every computer related project involves indirect expenses of power and an Internet connection among other irrelevant expenses.

| Product | Price | Units | Estimated cost |
|---|---|---|---|
| **Power** | 0.3€/kWh | 2 kW/day and 5 days/week (180 days in 36 weeks) | 108€ |
| **Internet** | 36.20€/month | 8 months | 289.60€ |
| **TOTAL** | | | **397.60€** |

Table 5: Indirect costs of the project.

## 4.3. Total cost

Lastly, all the budgets are added up and as a result the total expected cost of the project is obtained.

| Subject | Estimated cost |
|---|---|
| **Human resources** | 22690€ |
| **Hardware** | 230€ |
| **Software** | 159€ |
| **Indirect costs** | 397.60€ |
| **TOTAL** | **23476.60€** |

Table 6: Total estimated cost of the project.

# 5. Sustainability

## 5.1 Economic impact

An assessment of the project budget has been done. Material and human resources were taken into consideration, as well as amortizations and deviations.
Even though it is not sure its cost could make it competitive in the market, yet it might mainly depend on how it is going to be distributed. Selling it as a free application would make it very competitive, but then a way to get an income would have to be defined. On the other hand, it could be a paid application, however its price should not be too high.

## 5.2 Social impact

This project has its place in the software area, specifically, the world of mobile applications. The mobile apps market in Spain is not huge but it has been gaining relevance more and more lately.
The final product of this project is addressed to a large audience, in fact, anyone who cares about the music in his surroundings besides being eager to meet new people. And of course, has a compatible device to install the app.
This app can improve the experience of customers in their frequented establishments making them enjoy more their stay and giving them a chance to connect with others. Furthermore, they can discover new music thanks to the contributions from others.

## 5.3 Environmental impact

In this case, being a software project, it only affects the environment indirectly. Electricity is normally generated at power plants that convert some other kind of energy into electrical power. There are different systems and each of them has advantages and disadvantages, but many of them pose environmental concerns.

## 5.4 Sustainability matrix

| Sustainability | Economic | Social | Environmental |
|---|---|---|---|
| **Analysis at** | 5.1 | 5.2 | 5.3 |
| **Rating** | 7 | 8 | 6 |

Table 7: Sustainability matrix following the UPC sustainability guidelines.

# 6. Analysis of requirements

Here is a set of relevant requirements which have been rated by difficulty and relevance. These requirements have been listed with the idea of pushing the app to the market in mind.
The completion of the requirements increases the probability of success among users.

| # | Description | Difficulty | Relevance |
|---|---|---|---|
| **1** | Simple, attractive and user-friendly design | Medium | High |
| **2** | Multilingual | Medium | Low |
| **3** | Product available 24/7 | Low | High |
| **4** | Product compatible with old iOS versions | Medium | Medium |
| **5** | Availability in the App store | Medium | High |
| **6** | Release at least an update every three months | Medium | High |
| **7** | The application manual will be available online | Low | High |
| **8** | Users will have a contact mail to solve doubts and ask for help | Low | High |
| **9** | The development team won't take more than a week to fix an important bug | High | High |
| **10** | The product will have a two-step authentication | High | High |
| **11** | User will be able to reset his password | Medium | High |
| **12** | The system will protect user data taking into account the laws regarding privacy and will inform the user on sign up. | Medium | High |

Table 8: Requirements of product with their respective difficulty and relevance

# 7. Specification

On the following sections the conceptual model and the application's use cases are presented.

## 7.1. Conceptual model

The conceptual model has the goal of identifying the knowledge (the most significant concepts of the domain) and the behavior that system must have.

### 7.1.1. Conceptual scheme

In the conceptual scheme, the classes of objects that compose the system, the associations between them, their attributes and graphical and textual integrity restrictions are shown.

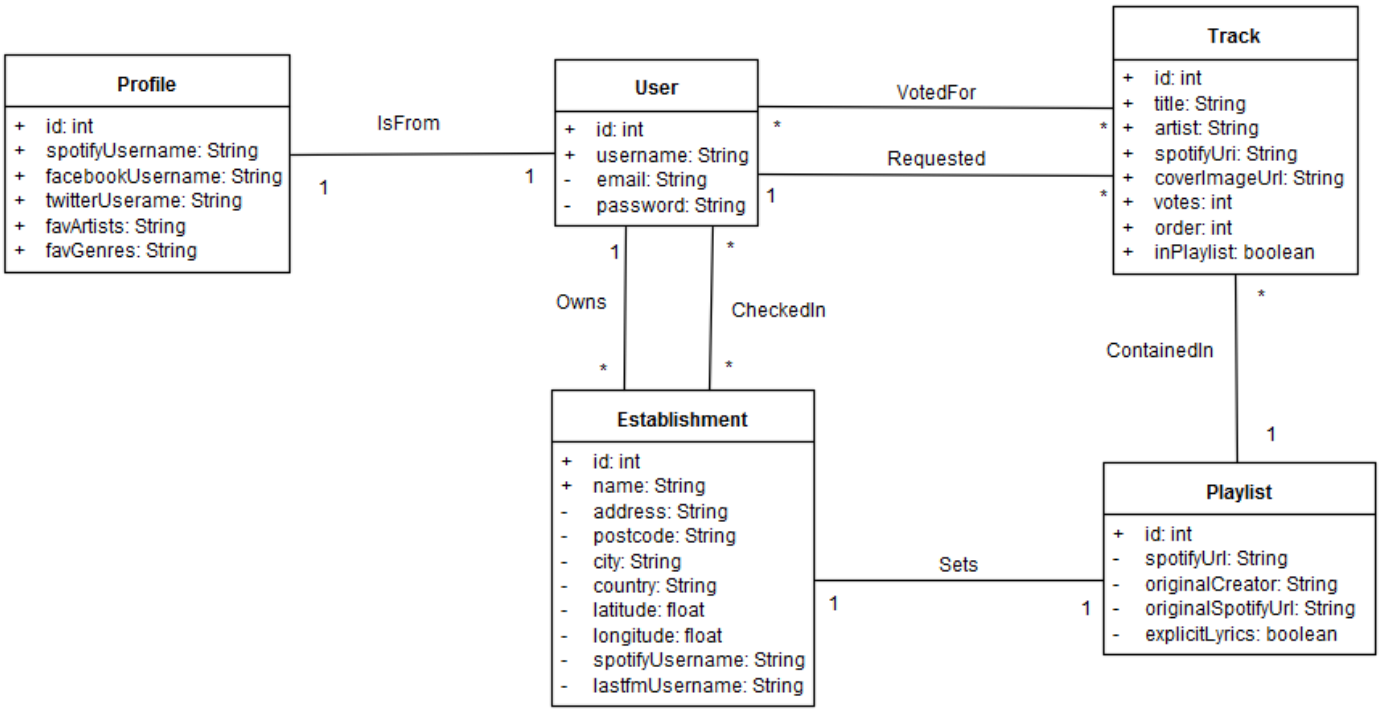



Figure 1. Conceptual scheme

### 7.1.2. Textual integrity restrictions

**Primary keys**

- User (id)
- Profile (id)
- Establishment (id)
- Playlist (id)
- Track (id)

**Alternative keys**

- User (email)
- User (username)
- Establishment (name)
- Establishment (address, postcode, city, country)

**Textual restrictions**

**RT1** A playlist cannot have more than one track with the same *spotifyUri* and *inPlaylist* as true.

### 7.1.3. Description of classes, attributes and relations

- **User**

  Represents a registered user of the app.
  - **id**: Identifier of a user
  - **username**: Username of a user. It is the user's identifier to other users.
  - **email**: Email of the user's account
  - **password**: Password of the user's account

- **Profile**

  Represents the information that is shown in the user's profile
  - **id**: Identifier of the profile
  - **spotifyUsername**: User's Spotify username
  - **facebookUsername**: User's Facebook username
  - **twitterUsername**: User's Twitter username
  - **favArtists**: User's favorite artists
  - **favGenres**: User's favorite music genres

- **Establishment**

  Represents an establishment that has registered to the service and has it available in their establishment.
  - **id**: Identifier of the establishment
  - **name**: Name of the establishment
  - **address**: Address of the establishment location
  - **postcode**: Postcode of the establishment location
  - **city**: City where the establishment is located
  - **country**: Country where the establishment is located
  - **latitude**: Latitude of the establishment's location, calculated from the given address
  - **longitude**: Longitude of the establishment's location, calculated from the given address
  - **spotifyUsername**: Spotify account from which the music is going to be played
  - **lastfmUsername**: Lastfm account is needed to get the currently playing song

- **Playlist**

  Represents an establishment playlist. It is related to the tracks that are currently playing in the establishment and the ones that were in the playlist in the past.
  - **id**: Identifier of the playlist
  - **spotifyUrl**: Spotify link to the currently set playlist. Its content is a copy of the original playlist
  - **originalCreator**: Creator of the currently set playlist
  - **originalSpotifyUrl**: Original Spotify link to the currently set playlist
  - **explicitLyrics**: Allowance of tracks with explicit lyrics in the establishment

- **Track**

  Represents tracks which can be currently playlist or not
  - **id**: Identifier of the track
  - **title**: Title of the track
  - **artist**: Artist of the track
  - **spotifyUri**: Identifier of track in Spotify
  - **coverImageUrl**: Link to the cover image of the track
  - **votes**: Number of votes from users
  - **order**: Track's order in the current playlist
  - **inPlaylist**: Presence of the track in the currently playing playlist

- **IsFrom** (Profile – User)
  Every user owns a profile which is public to other users.

- **VotedFor** (User – Track)
  A user votes for songs that are being played in an establishment.

- **Requested** (User – Track)
  A user submits song requests to get them added to a playlist.

- **Owns** (User – Establishment)
  A user is the owner of an establishment.

- **CheckedIn** (User – Establishment)
  A user can check-in to establishments they have been.

- **Sets** (Establishment – Playlist)
  A playlist is set to an establishment by the owner.

- **ContainedIn** (Track – Playlist)
  Tracks are part of a playlist.

## 7.2. Use cases

### 7.2.1. #UC1 Login

**Primary actor:** User
**Precondition:** User has signed up to the app
**Trigger:** User wants to log in to his account
**Main success scenario:**

1. User inputs his username and password in the corresponding fields.
2. User presses the *Log in* button.
3. The system validates the input data and logs in to the user account.

**Extensions:**

3a. The input credentials are incorrect.
3a1. The system displays an alert saying that the credentials are wrong.
3a2. Back to step 1.

### 7.2.2. #UC2 Sign up

**Primary actor:** User
**Precondition:** -
**Trigger:** User wants to sign up for an account in the app
**Main success scenario:**

1. User inputs his username, email and password twice in the corresponding fields
2. User presses the *Sign up* button.
3. The system validates the input data and creates a new user account with them.
4. The system displays a form to enter the social media information.
5. User inputs his Spotify username, Facebook username and/or Twitter username.
6. User presses the *Save* button.
7. The system saves the social media information to his user account.
8. The system navigates to the Login screen.

**Extensions:**

3a. The username or email is already taken.
3a1. The system displays an alert saying that the username or email are already taken.
3a2. Back to step 1.

3b. The introduced passwords don't match.
3b1. The system displays an alert saying that the passwords don't match.
3b2. Back to step 1.

5a. User presses the *Skip* button.

5a1. Go to step 8.

### 7.2.3. #UC3 Log out

**Primary actor:** User

**Precondition:** User is logged in.

**Trigger:** User wants to log out of his account.

**Main success scenario:**

1. User clicks on the *Profile* tab item.
2. The system displays the user's own profile.
3. User presses the *Log out* button.
4. The system displays the Login screen.

### 7.2.4. #UC4 Edit user's own profile

**Primary actor:** User

**Precondition:** User is logged in.

**Trigger:** User wants to change his account data.

**Main success scenario:**

1. User clicks on the *Profile* tab item.
2. The system displays the user's own profile.
3. User presses the *Edit profile* button.
4. The system displays the user's current profile data.
5. User modifies any field he desires.
6. User presses the *Save* button.
7. The system saves the new profile data to the user's account.

**Extensions:**

5a. User presses the *cancel* or back button.

5a2. The system displays the user profile view.

### 7.2.5. #UC5 Select an establishment from map

**Primary actor:** User

**Precondition:** User is logged in.

**Trigger:** User wants to select an establishment from map.

**Main success scenario:**

1. User clicks on the *Map* tab item.
2. The system displays the map centered on the user's current location along with the establishments in the displayed map area.
3. User finds the desired establishment in map and click on the establishment's pin.
4. The system displays a bubble with the establishment's name and address.

5. User presses the information bubble.
6. System navigates to the selected establishment profile.

**Extensions:**

5a. User presses the map.
5a2. The system hides the displayed information bubble.
5a3. Back to step 3.

### 7.2.6. #UC6 Search establishment by name or address

**Primary actor:** User
**Precondition:** User is logged in.
**Trigger:** User wants to search an establishment by name or address.
**Main success scenario:**

1. User clicks on the *Search* tab item.
2. The system displays the search view.
3. User inputs the desired establishment name or address.
4. The system displays a list of possible establishments with their corresponding name and address.
5. User selects an establishment from the list of search results.
6. System navigates to the selected establishment profile.

**Extensions:**

4a. User presses the cancel button at the search bar.
4a2. The system hides the previous search results.
4a3. Back to step 3.

### 7.2.7. #UC7 Check in to an establishment

**Primary actor:** User
**Precondition:** User is logged in, viewing an establishment profile and hasn't checked in to this establishment yet
**Trigger:** User wants to check in to the establishment displayed
**Main success scenario:**

1. User clicks on the button with a checkmark icon.
2. The system adds the establishment as a checked-in of the user and changes the button to a green check mark.

### 7.2.8. #UC8 Vote for a song

**Primary actor:** User
**Precondition:** User is logged in, a playlist is currently playing in the establishment he's viewing, user hasn't voted for the desired song yet
**Trigger:** User wants to vote for a song
**Main success scenario:**

1. User clicks on the *like* button next to the song he wants to vote for.
2. The system adds a vote to the selected song, adds the user as a voter of the song, reorders the playlist taking into account the number of votes and updates it.

### 7.2.9. #UC9 Request a song

**Primary actor:** User
**Precondition:** User is logged in and a playlist is currently playing in the establishment he's viewing
**Trigger:** User wants to request a song for the establishment's playlist.
**Main success scenario:**

1. User clicks on the *Request* button.
2. The system navigates to the song search view.
3. User inputs the song title or artist in the search bar.
4. The system displays a list of tracks given the search parameters.
5. User selects a song from the list of search results.
6. The system displays an alert asking the user's confirmation of the selected song.
7. User confirms the selection by clicking the *Yes* button.
8. The system adds the requested song to the current playlist with a vote from the user and displays a successfully added song message to the user.
9. User clicks on the *Ok* button
10. The system navigates back to the establishment profile.

**Extensions:**

6a. User presses the *No* button.
6a2. The system dismisses the alert.
6a3. Back to step 4.

8a. The system displays an alert with a message saying that the requested song has explicit lyrics and it isn't allowed at the establishment.
8a2. User clicks on the *Ok* button.
8a2. Back to step 4.

### 7.2.10. #UC10 View a song's voters

**Primary actor:** User
**Precondition:** User is logged in and is viewing an establishment that has a playlist set
**Trigger:** User wants to see the voters of a song.
**Main success scenario:**

1. User clicks on the button that displays the number of voters of the song.
2. The system navigates to a view displaying the list of voters.

### 7.2.11. #UC11 View a song's voter profile

**Primary actor:** User
**Precondition:** User is logged in and is viewing the list of voters of a song
**Trigger:** User wants to see a voter's profile
**Main success scenario:**

1. User clicks on the user he's interested to view the profile of.
2. The system navigates to the selected voter profile.

### 7.2.12. #UC12 View profile of the user who requested a song

**Primary actor:** User
**Precondition:** User is logged in and is viewing an establishment that has a playlist set
**Trigger:** User wants to see the user that has requested a certain song.
**Main success scenario:**

1. User clicks on the *R* button that displays the number of voters of the song.
2. The system navigates to a view displaying the list of voters.

### 7.2.13. #UC13 View user's check-ins

**Primary actor:** User
**Precondition:** User is logged in and is viewing a user profile
**Trigger:** User wants to see the check-in places of the profile owner
**Main success scenario:**

1. User clicks on the number of check-ins of the profile owner.
2. The system displays a list of establishment names.

### 7.2.14. #UC14 View user's voted songs

**Primary actor:** User
**Precondition:** User is logged in and is viewing a user profile
**Trigger:** User wants to see the profile owner has voted for
**Main success scenario:**

1. User clicks on the number of voted songs of the profile owner.
2. The system displays a list of song titles and artists.

### 7.2.15. #UC15 View user's requested songs

**Primary actor:** User
**Precondition:** User is logged in and is viewing a user profile
**Trigger:** User wants to see the profile owner has voted for
**Main success scenario:**

1. User clicks on the number of voted songs of the profile owner.
2. The system displays a list of song titles and artists.

### 7.2.16. #UC16 Register an establishment

**Primary actor:** User
**Precondition:** User is logged in.
**Trigger:** User wants to register an establishment.
**Main success scenario:**

1. User clicks on the *My joogpoints* tab item.
2. The system displays a list of establishments the user is owner of.
3. User clicks on the *Plus* icon button.
4. The system displays an establishment registration form.
5. User completes the registration form and clicks on the *Register* button.
6. The system adds the new establishment.
7. System navigates back to the list of establishments the user owns.

**Extensions:**

5a. User presses the *cross* icon button or the *Cancel* button.
5a2. The system displays an alert asking if the user is sure about the decision.
5a3a. User presses the *No* button.
5a3a2. The system dismisses the alert.
5a3a3. Back to step 5.

5a3b. User presses the *Yes* button.
5a3a2 The system dismisses the alert and navigates back to the list of establishments.

6a. The system displays a message saying that all fields have to be filled in.
6a2. User clicks on the *Ok* button.
6a3. The system dismisses the alert.
6a4. Back to step 5.

### 7.2.17. #UC17 Edit owned establishment profile

**Primary actor:** User and owner of the establishment
**Precondition:** User is logged in and is owner of the establishment.
**Trigger:** User wants to edit the profile of an establishment he owns.
**Main success scenario:**

1. User clicks on the *My joogpoints* tab item.
2. The system displays a list of establishments the user is owner of.
3. User clicks on the establishment he's interested to modify the profile of.
4. The system displays the management view of the selected establishment.
5. The user clicks on the *Edit profile* button.
6. The system displays an establishment data form with the current establishment info.
7. User modifies the fields he desires and clicks on the *Save* button.
8. The system saves any changes on the establishment profile info.
9. System navigates back to the establishment management view.

**Extensions:**

7a. User presses the *cross* icon button or the *Cancel* button.
7a2. The system navigates back to the establishment management view.

8a. The system displays a message saying that all fields have to be filled in.
8a2. User clicks on the *Ok* button.
8a3. The system dismisses the alert.
8a4. Back to step 7.

### 7.2.18. #UC18 Remove an establishment

**Primary actor:** User and owner of the establishment
**Precondition:** User is logged in and is owner of the establishment.
**Trigger:** User wants to remove an establishment he owns.
**Main success scenario:**

1. User clicks on the *My joogpoints* tab item.
2. The system displays a list of establishments the user is owner of.
3. User clicks on the establishment he wants to remove.
4. The system displays the management view of the selected establishment.
5. The user clicks on the *Edit profile* button.
6. The system displays an establishment data form with the current establishment info.
7. User clicks on the *Remove establishment* button.
8. The system displays an alert asking for the user's confirmation of the removal.
9. User clicks on the *Y*es button.

10. The system dismisses the alert and removes the establishment.
11. System navigates back to the *My joogpoints* view.

**Extensions:**

9a. User presses the *No* button.
9a2. The system dismisses the alert.
9a3. Back to step 6.

### 7.2.19. #UC19 View owned establishment current playlist

**Primary actor:** User and owner of the establishment
**Precondition:** User is logged in, is owner of the establishment and has set a playlist for the establishment
**Trigger:** User wants to view the current playlist of an establishment he owns.

**Main success scenario:**

1. User clicks on the *My joogpoints* tab item.
2. The system displays a list of establishments the user is owner of.
3. User clicks on the establishment he's interested to view the current playlist of.
4. The system displays the management view of the selected establishment.
5. The user clicks on the *Current playlist* button.
6. The system displays the current playlist of the establishment.

### 7.2.20. #UC20 Set establishment playlist

**Primary actor:** User and owner of the establishment
**Precondition:** User is logged in and is owner of the one establishment.
**Trigger:** User wants to set the playlist of an establishment he owns.

**Main success scenario:**

1. User clicks on the *My joogpoints* tab item.
2. The system displays a list of establishments the user is owner of.
3. User clicks on the establishment he's interested to set playlist of.
4. The system displays the management view of the selected establishment.
5. User clicks on the *Set playlist* button.
6. The system displays a pop-up with two fields for a Spotify playlist link and the creator of the playlist.
7. User fills in the playlist info and presses the *Confirm* button.
8. The system sets the input Spotify playlist as the establishment's playlist and dismisses the pop-up.

**Extensions:**

6a. User presses the *Cancel button*.
6a2. The system dismisses the pop-up.

8a. The system displays an alert saying that the same playlist is already set.
8a2. User clicks on the *Ok* button.
8a3. The system dismisses the alert.

### 7.2.21. #UC21 Reset establishment playlist

**Primary actor:** User and owner of the establishment
**Precondition:** User is logged in, is owner of the establishment and has set a playlist for the establishment
**Trigger:** User wants to reset the establishment's current playlist.
**Main success scenario:**

1. User clicks on the *My joogpoints* tab item.
2. The system displays a list of establishments the user is owner of.
3. User clicks on the establishment he's interested to reset the playlist of.
4. The system displays the management view of the selected establishment.
5. User clicks on the *Reset playlist* button.
6. The system resets the playlist to its original set of songs and clears the votes and displays a message informing that the reset was done successfully.

### 7.2.22. #UC22 Clear playlist's votes

**Primary actor:** User and owner of establishments
**Precondition:** User is logged in, is owner of the establishment and has set a playlist for the establishment
**Trigger:** User wants to clear the votes of the establishment's current playlist.
**Main success scenario:**

1. User clicks on the *My joogpoints* tab item.
2. The system displays a list of establishments the user is owner of.
3. User clicks on the establishment he's interested to.
4. The system displays the management view of the selected establishment.
5. User clicks on the *Clear votes* button.
6. The system clears the votes of all tracks in the playlist and displays a message informing that the process was done successfully.

### 7.2.23. #UC23 Remove establishment's current playlist

**Primary actor:** User and owner of the establishment
**Precondition:** User is logged in, is owner of the establishment and has set a playlist for the establishment
**Trigger:** User wants to remove the establishment's current playlist.
**Main success scenario:**

1. User clicks on the *My joogpoints* tab item.
2. The system displays a list of establishments the user is owner of.
3. User clicks on the establishment he's interested to remove the playlist of.
4. The system displays the management view of the selected establishment.
5. User clicks on the *Remove playlist* button.
6. The system shows a pop-up asking for the user's confirmation.
7. User clicks on the *Yes* button.
8. The system removes the playlist from establishment and displays an alert with a message saying that the removal was done successfully.

**Extensions:**

7a. User presses the *No button*.

7a2. The system dismisses the pop-up.

### 7.2.24. #UC24 Set song with explicit lyrics allowance

**Primary actor:** User and owner of establishments

**Precondition:** User is logged in and is owner of the establishment

**Trigger:** User wants to modify the explicit lyrics allowance of the establishment

**Main success scenario:**

1. User clicks on the *My joogpoints* tab item.
2. The system displays a list of establishments the user is owner of.
3. User clicks on the establishment he's interested to.
4. The system displays the management view of the selected establishment.
5. User clicks on the *Explicit lyrics* button.
6. The system sets explicit lyrics to *allowed* if it wasn't, otherwise the contrary.

# 8. Design

## 8.1. Design models

In this section, the design of each use case is detailed. The Model View Controller pattern has been applied in the process.

There are three levels of permissions that are checked before each action:

- **None**: the process of login and sign up don't need any permission.
- **IsAuthenticated**: the user only needs to have an account in the app.
- **IsOwner**: the user must be the owner of such element (profile or establishment in this case) in order to make certain modifications.

There is a *permissionController* that has the responsibility of checking if a user has the right to make an action. Its usage is shown at #UC3. Its application is assumed from #UC3 and forward.

### 8.1.1. #UC1 Login

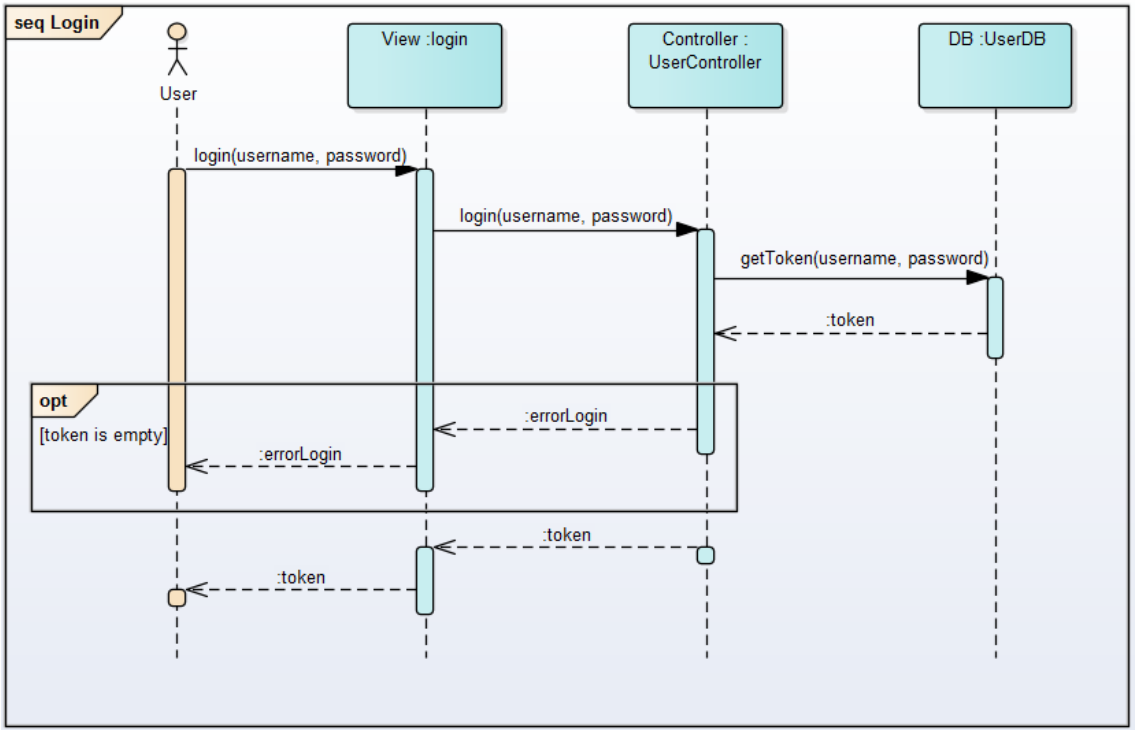



Figure 2. *#UC1 Login* design model

### 8.1.2. #UC2 Sign up

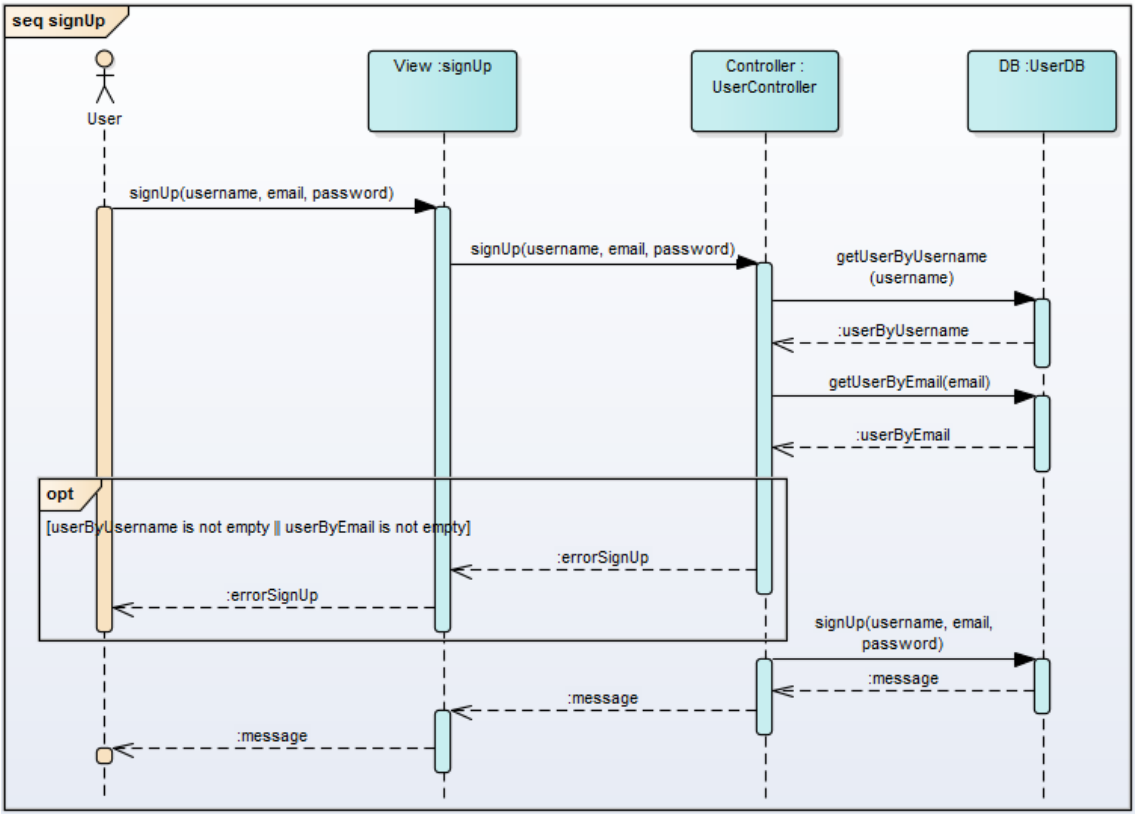



Figure 3. *#UC2 Sign up* design model

### 8.1.3. #UC3 Log out

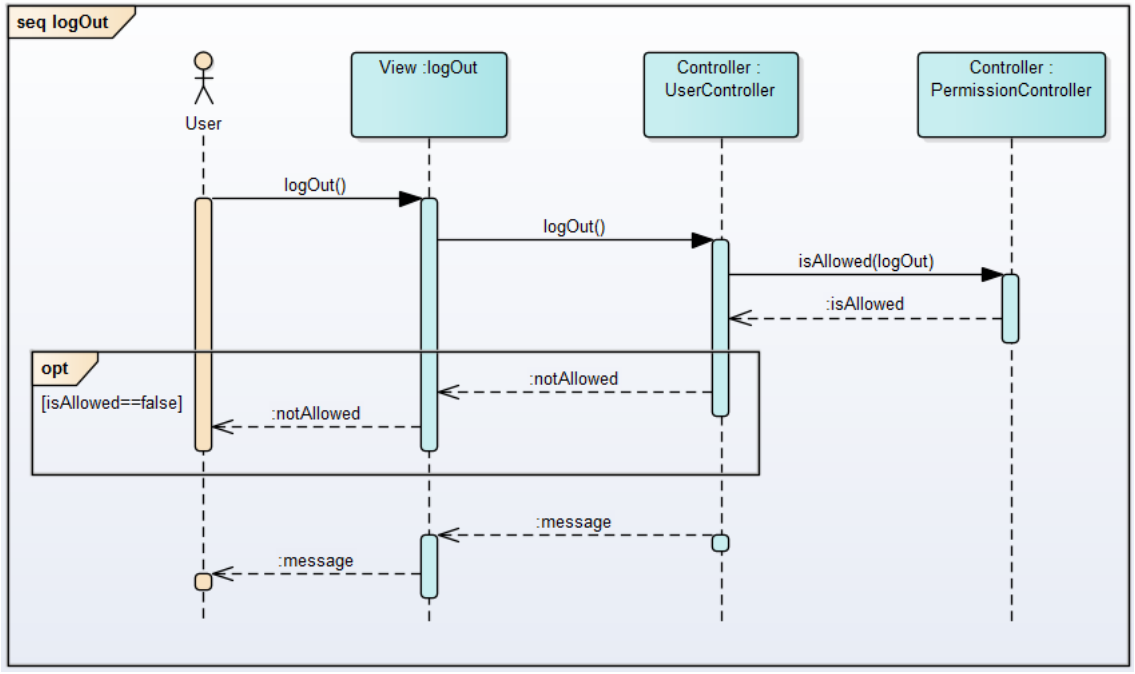



Figure 4. *#UC3 Log out* design model

### 8.1.4. #UC4 Edit user's own profile

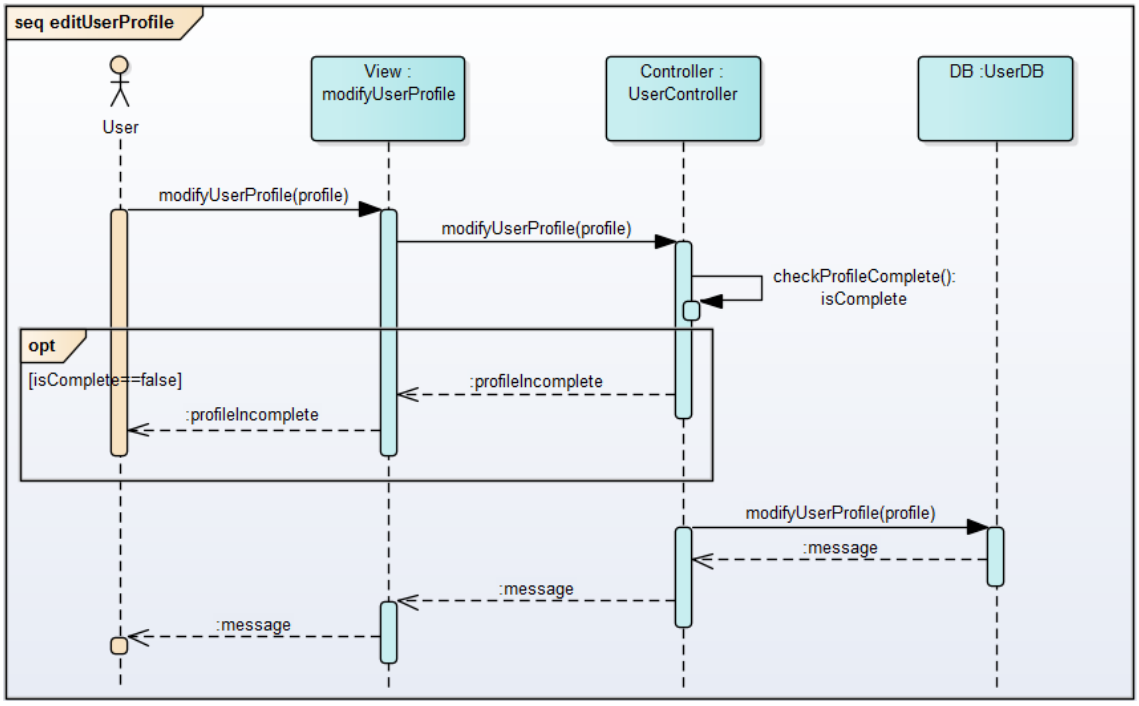



Figure 5. *#UC4 Edit user's own profile* design model

### 8.1.5. #UC5 Select an establishment from map

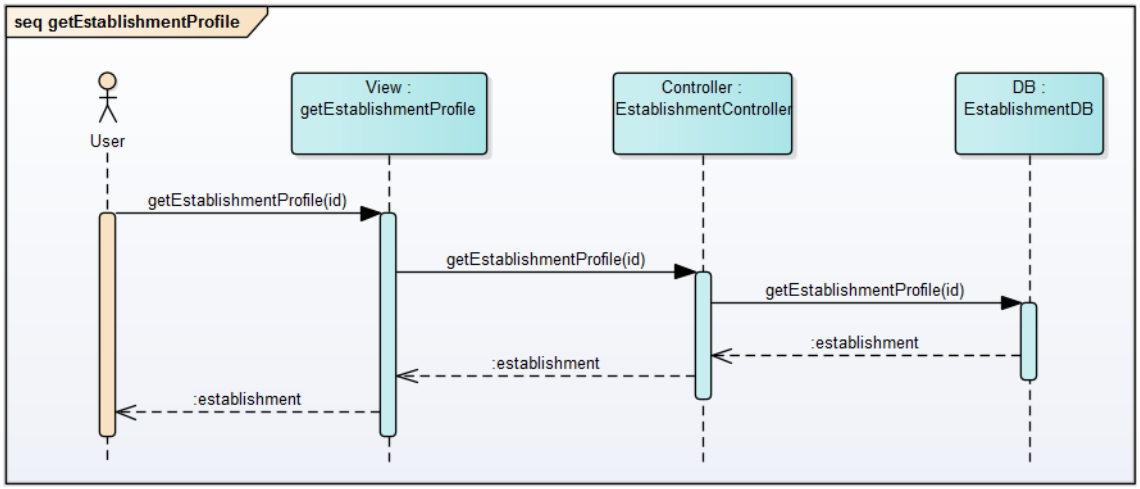



Figure 6. *#UC5 Select an establishment from map* design model

### 8.1.6. #UC6 Search establishment by name or address

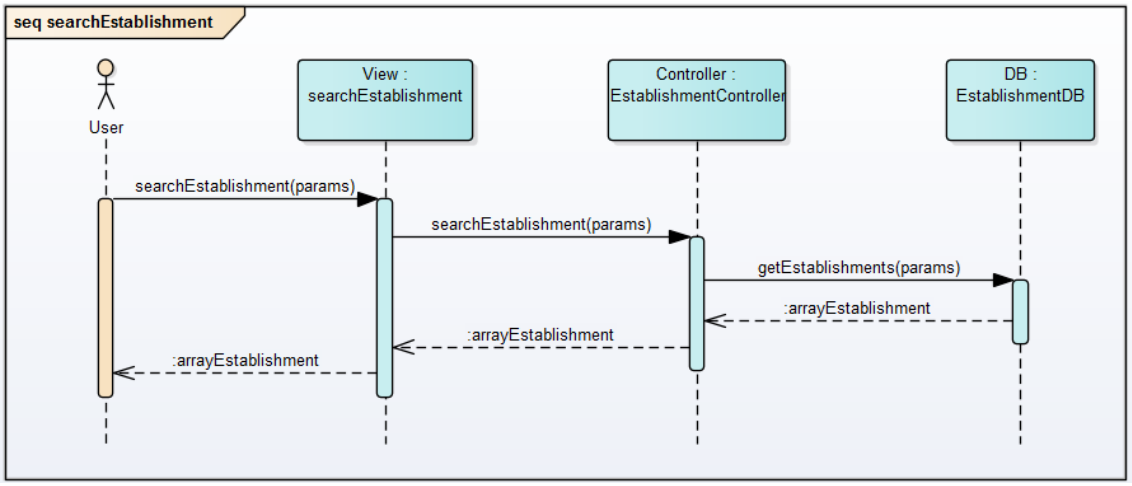



Figure 7. *#UC6 Search establishment by name or address* design model

### 8.1.7. #UC7 Check in to an establishment

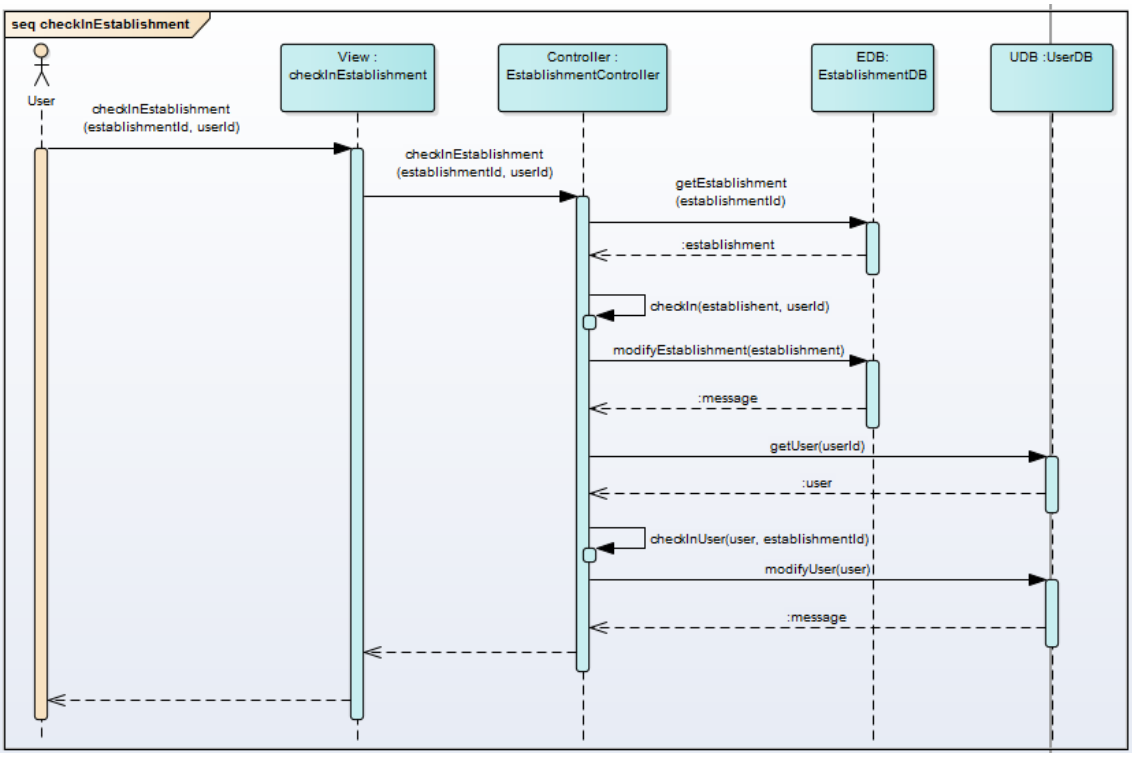



Figure 8. *#UC7 Check in to an establishment* design model

### 8.1.8. #UC8 Vote for a song

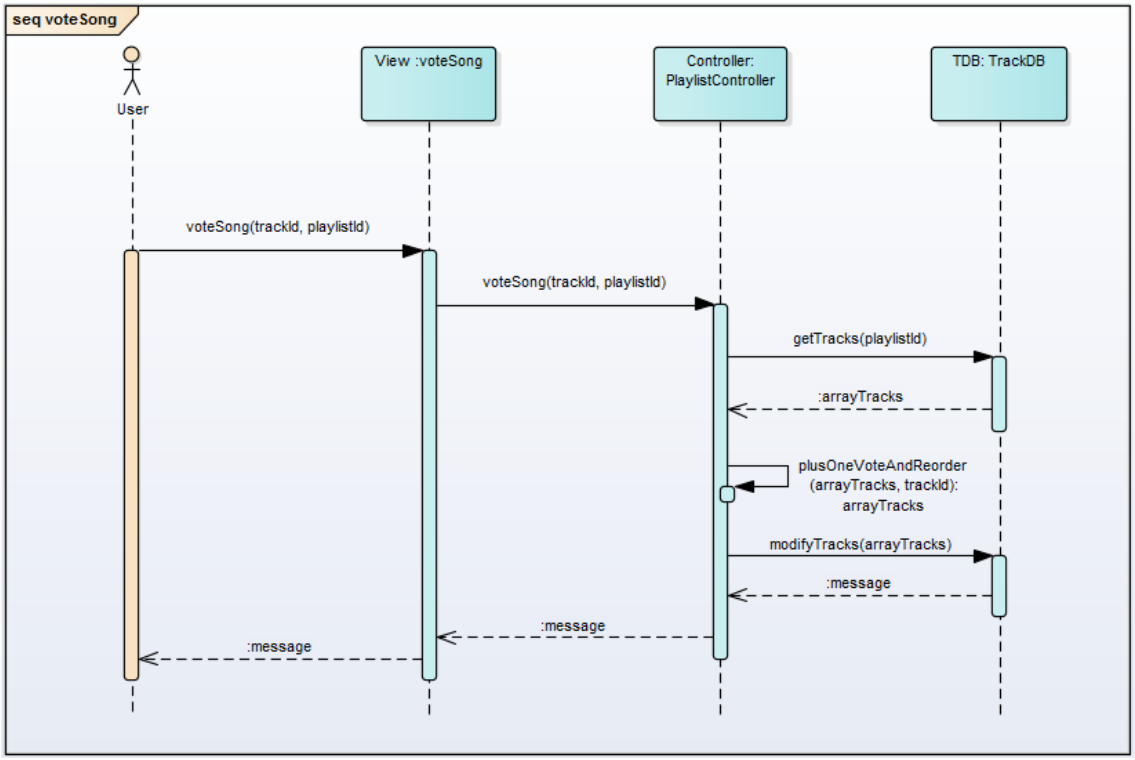



Figure 9. *#UC8 Vote for a song* design model

### 8.1.9. #UC9 Request a song

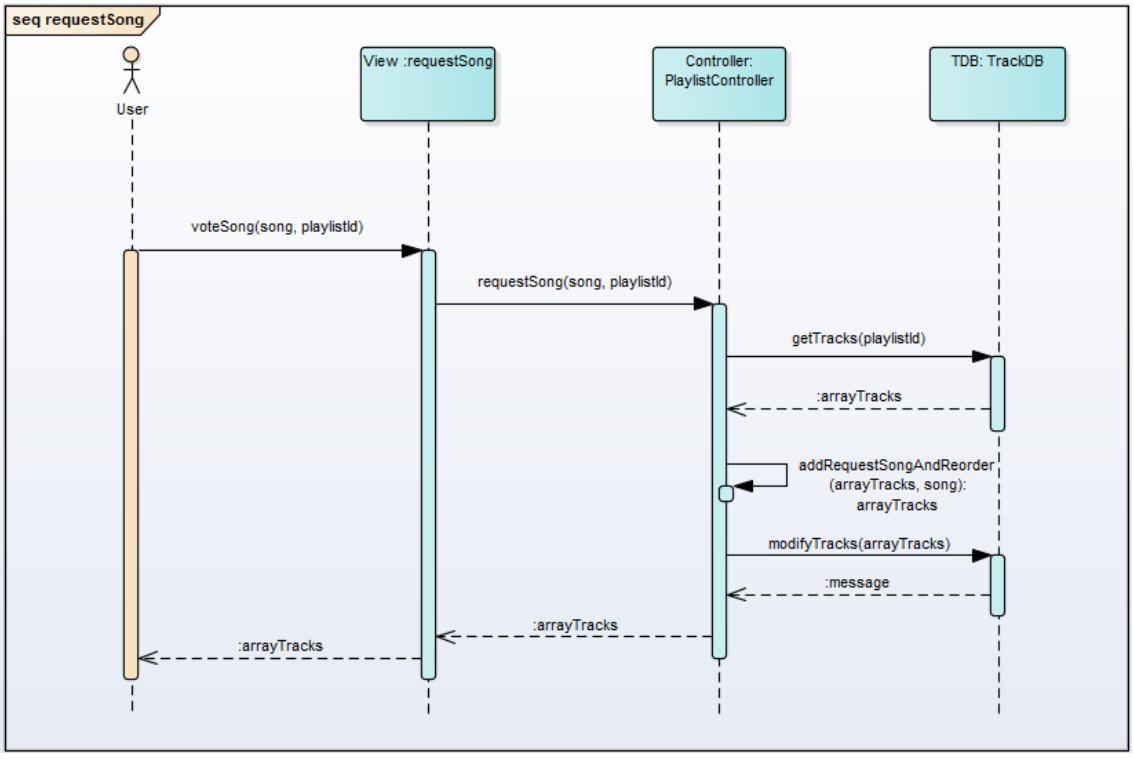



Figure 10. *#UC9 Request a song* design model

### 8.1.10. #UC10 View a song's voters

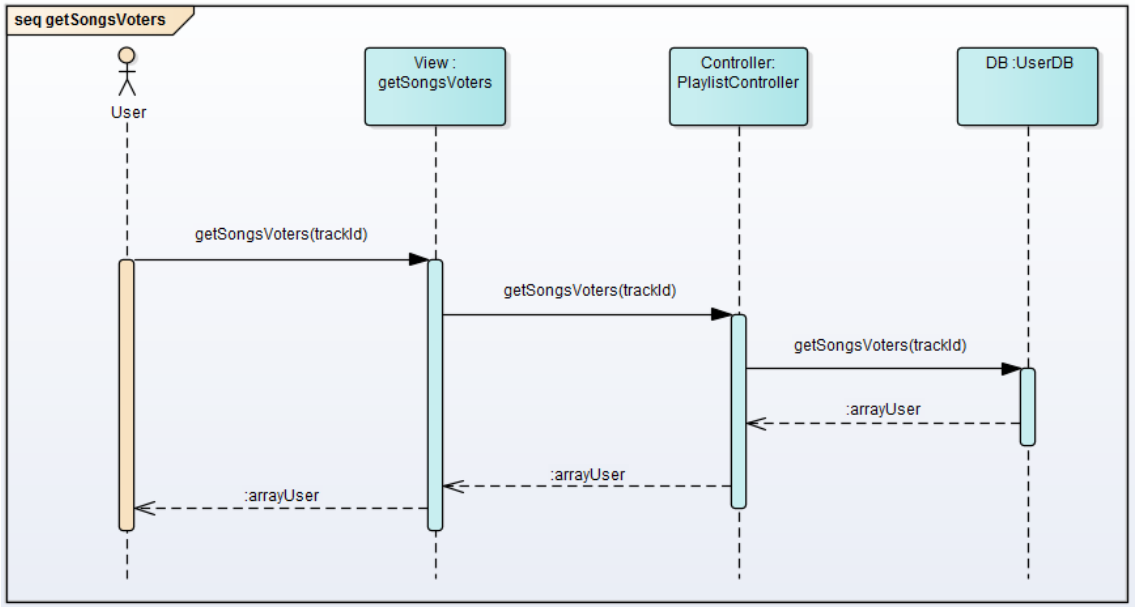



Figure 11. *#UC10 View a song's voters* design model

### 8.1.11. #UC11 View a song's voter profile, #UC12 View profile of the use who requested a song, #UC13 View user's check-ins, #UC14 View user's voted songs, #UC15 View user's requested songs

All data needed on the use cases listed in this section are part of user profile, that's the reason why they are all group together. The task to do after getting the user profile info is to extract the requested info.

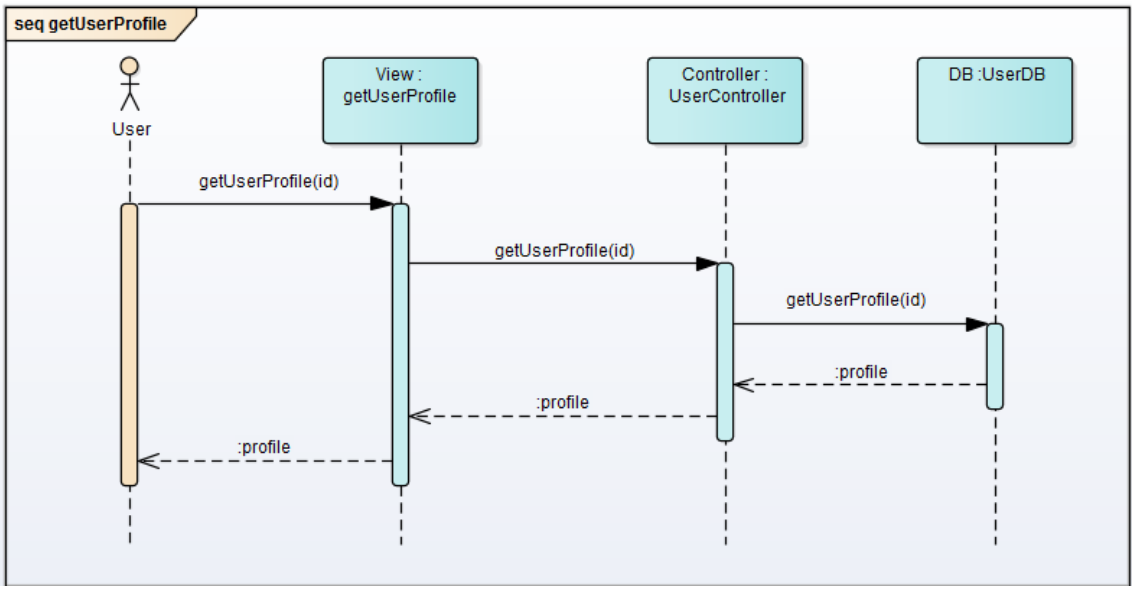



Figure 12. *#UC11, #UC12, #UC13, #UC14, #UC15* design model

### 8.1.12. #UC16 Register an establishment

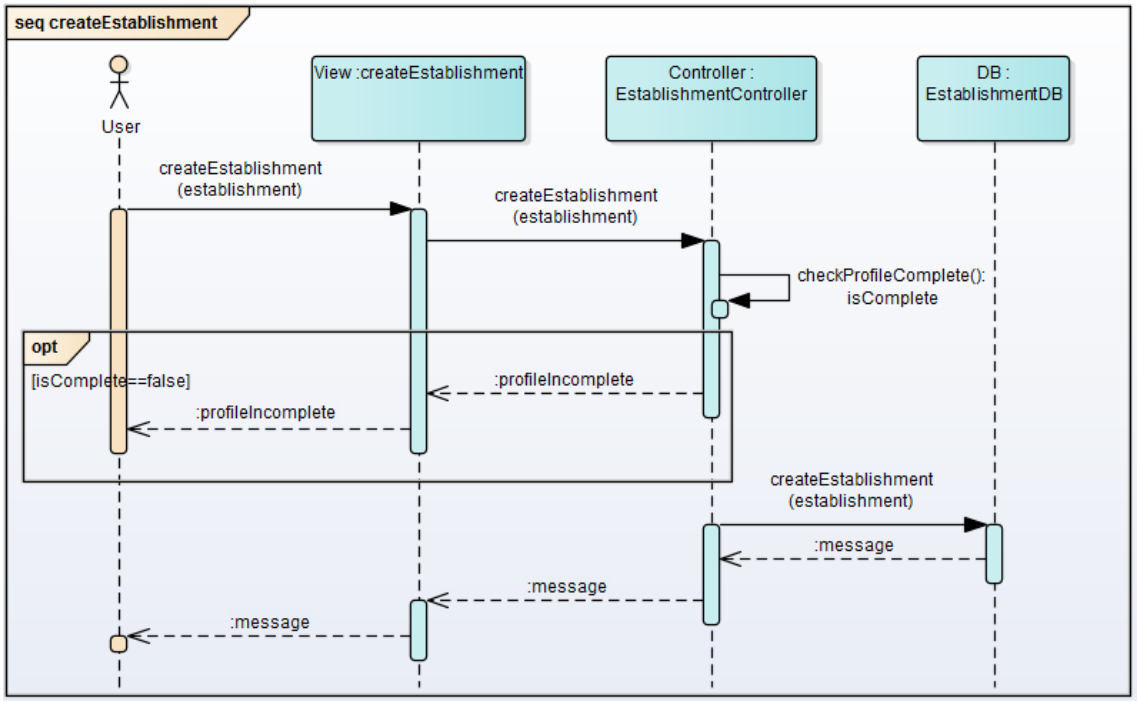



Figure 13. *#UC16 Register an establishment* design model

### 8.1.13. #UC17 Edit owned establishment profile

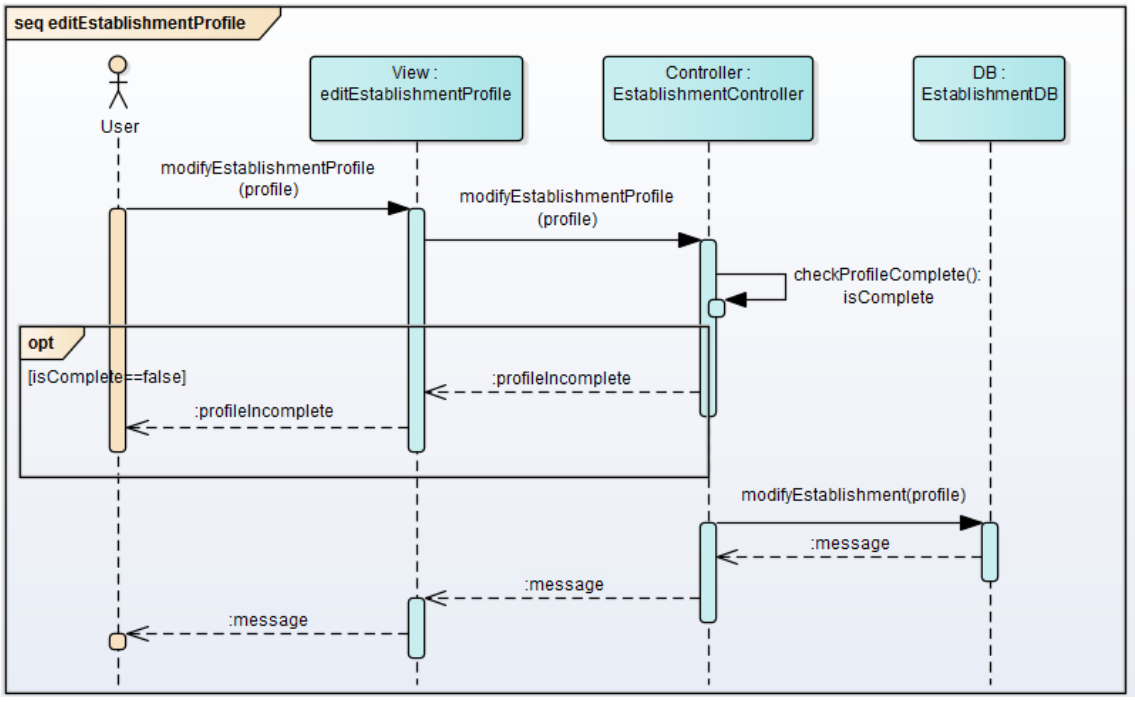



Figure 14. *#UC17 Edit owned establishment profile* design model

### 8.1.14. #UC18 Remove an establishment

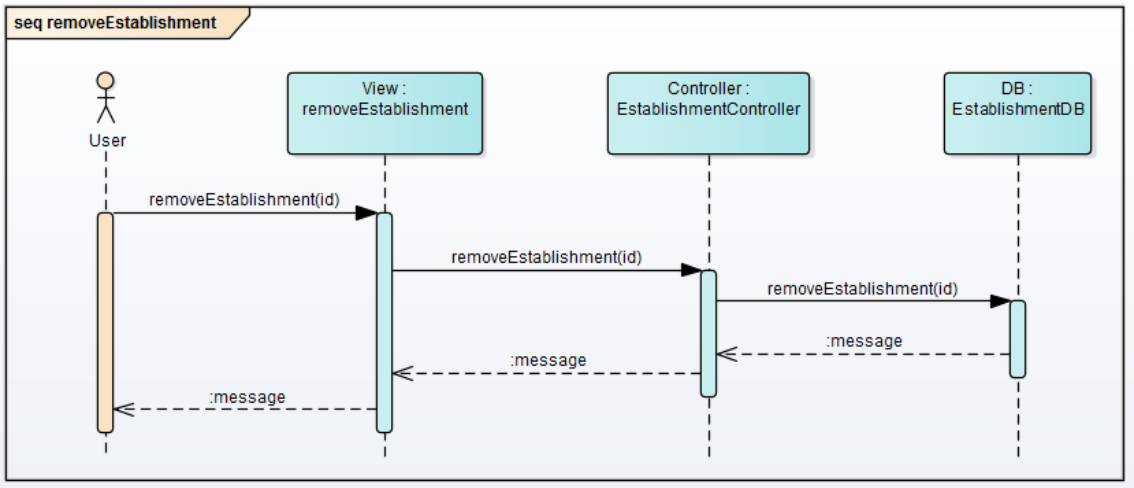



Figure 15. *#UC18 Remove an establishment* design model

### 8.1.15. #UC19 View owned establishment current playlist

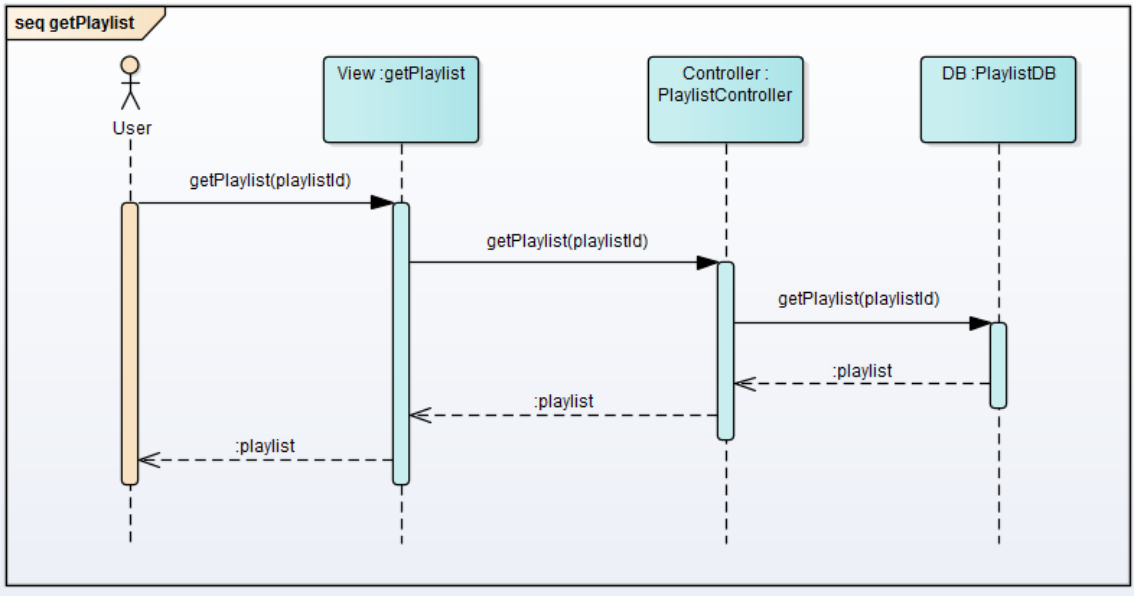



Figure 16. *#UC19 View owned establishment current playlist* design model

### 8.1.16. #UC20 Set establishment playlist

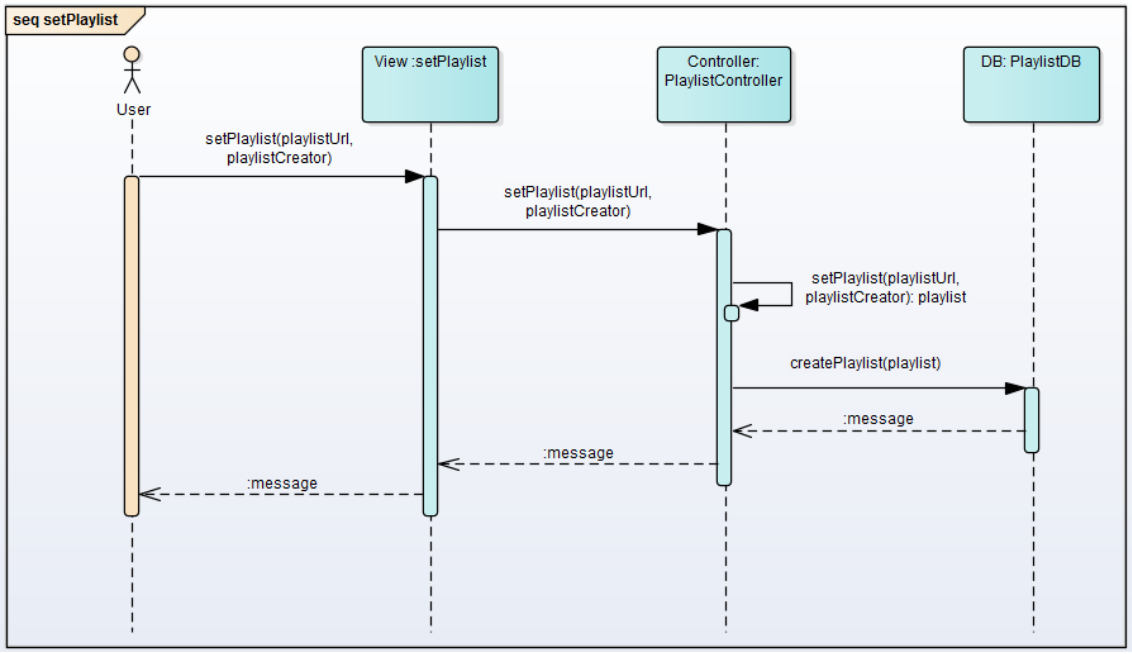



Figure 17. *#UC20 Set establishment playlist* design model

### 8.1.17. #UC21 Reset establishment playlist

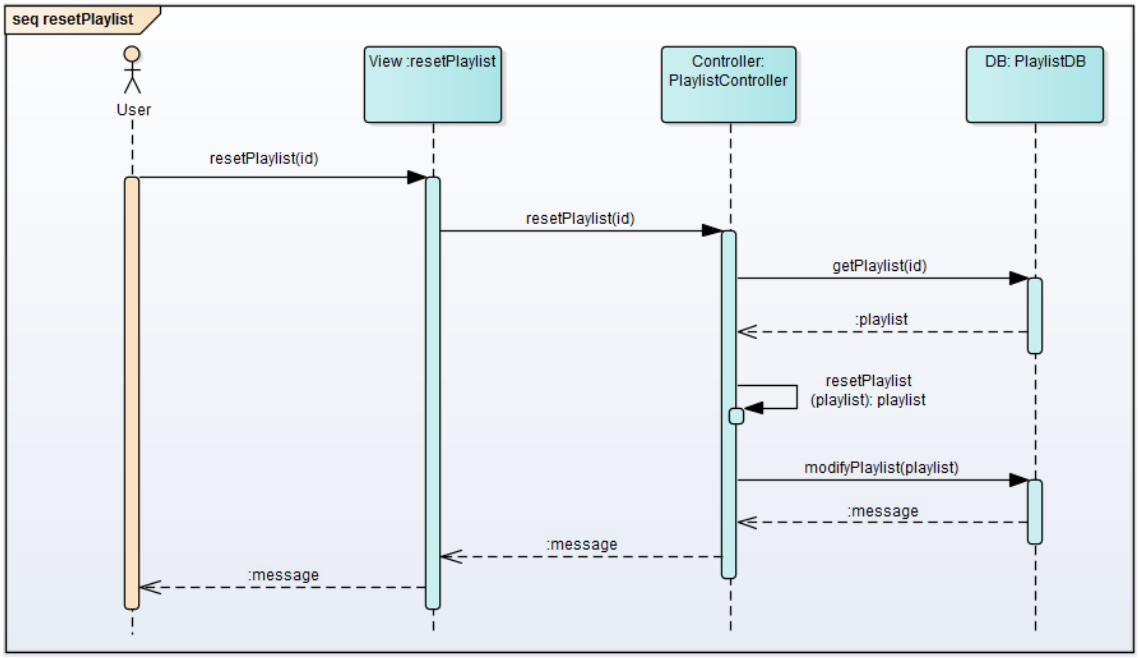



Figure 18. *#UC21 Reset establishment playlist* design model

### 8.1.18. #UC22 Clear playlist's votes

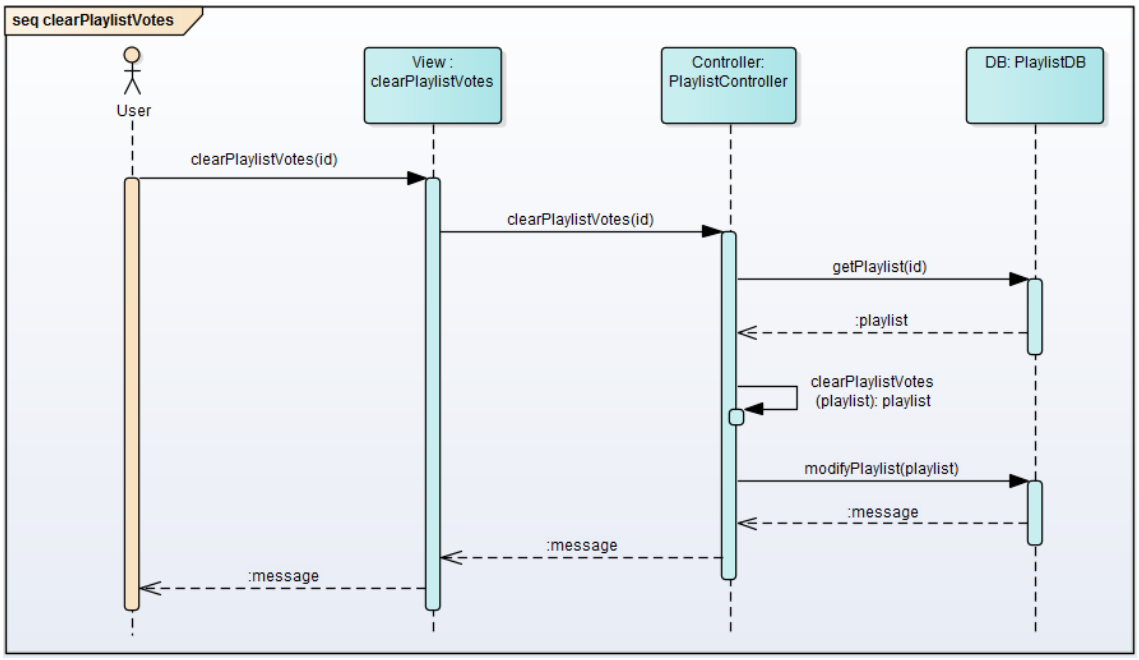



Figure 19. *#UC22 Clear playlist's votes* design model

### 8.1.19. #UC23 Remove establishment's current playlist

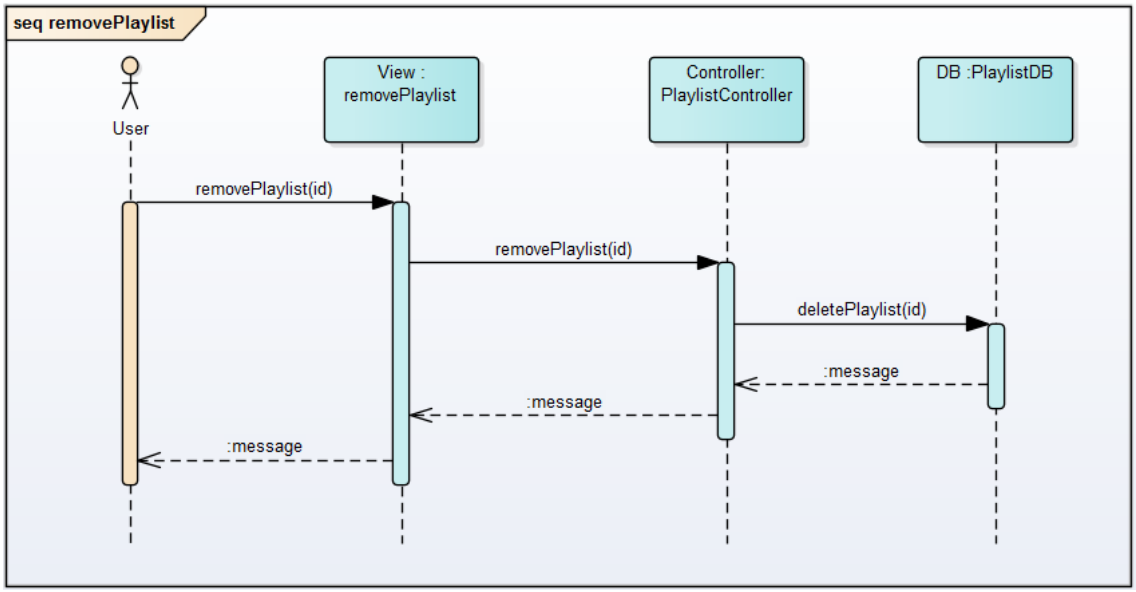



Figure 20. *#UC23 Remove establishment's current playlist* design model

### 8.1.20. #UC24 Set song with explicit lyrics allowance

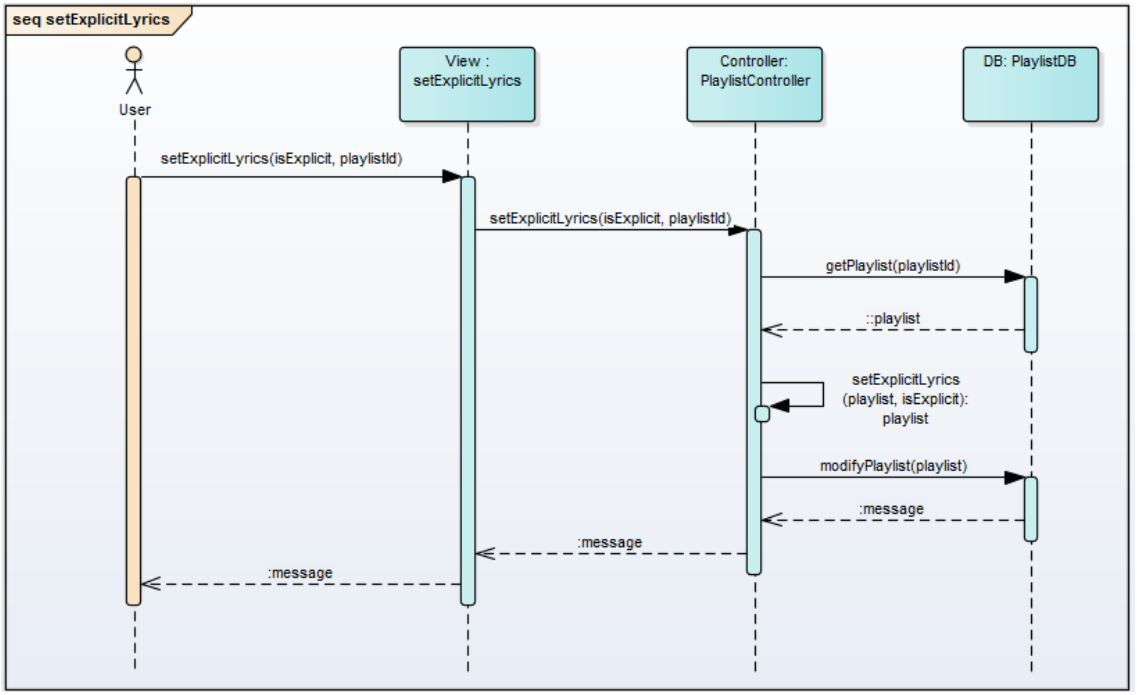



Figure 21. *#UC24 Set song with explicit lyrics allowance* design model

## 8.2. Patterns

The following patterns have been used to develop the application.

### 8.2.1. Singleton pattern

The singleton pattern is applied to the classes in charge of making requests to the database. Therefore, only a single instance of each class will be created and a unique access to it is guaranteed.

### 8.2.2. Model view controller pattern

The application has been developed using a three-tier architecture which is composed by a presentation tier, a logic tier and a data tier. The mobile app is responsible of the interaction with the user (presentation) and all data is fetched through API requests to the server, where the database is located. In this way, a clear separation of the different components is achieved, which eases the reuse of code and the development and maintenance of the app.

## 8.3. Deployment model

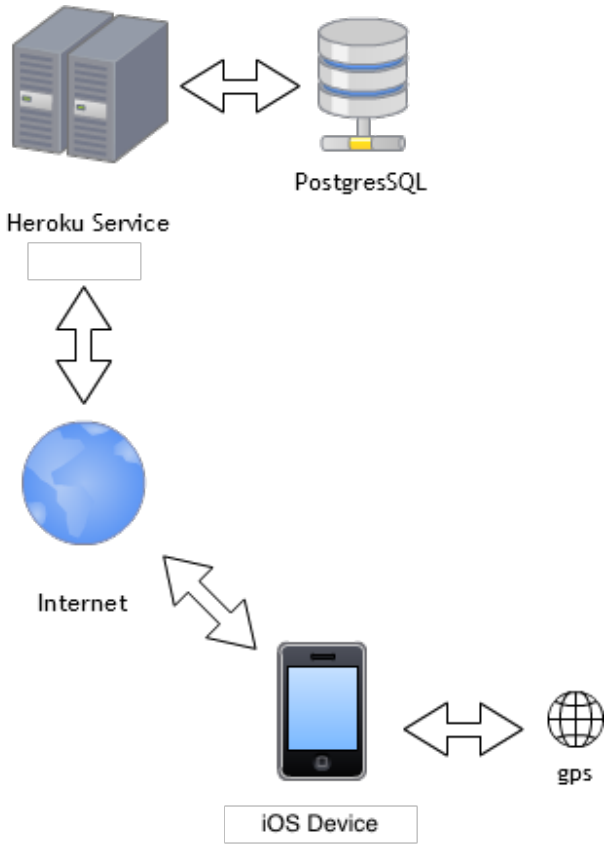



Figure 22. Deployment model

The relation between the client and the server is shown in the image above: the user has to be connected to the Internet in order to receive current data from the server deployed on Heroku. This one has the responsibility of retrieving and sending the requested data from the PostgreSQL database and storing the received data in the database.

On the other hand, the application must have access to the device's geolocation in order to display the current location of the user in the map, where establishments registered to the service are displayed.

## 8.4. Data model

In the following data model, a clear view of the relationships between the different tables of the database is displayed.

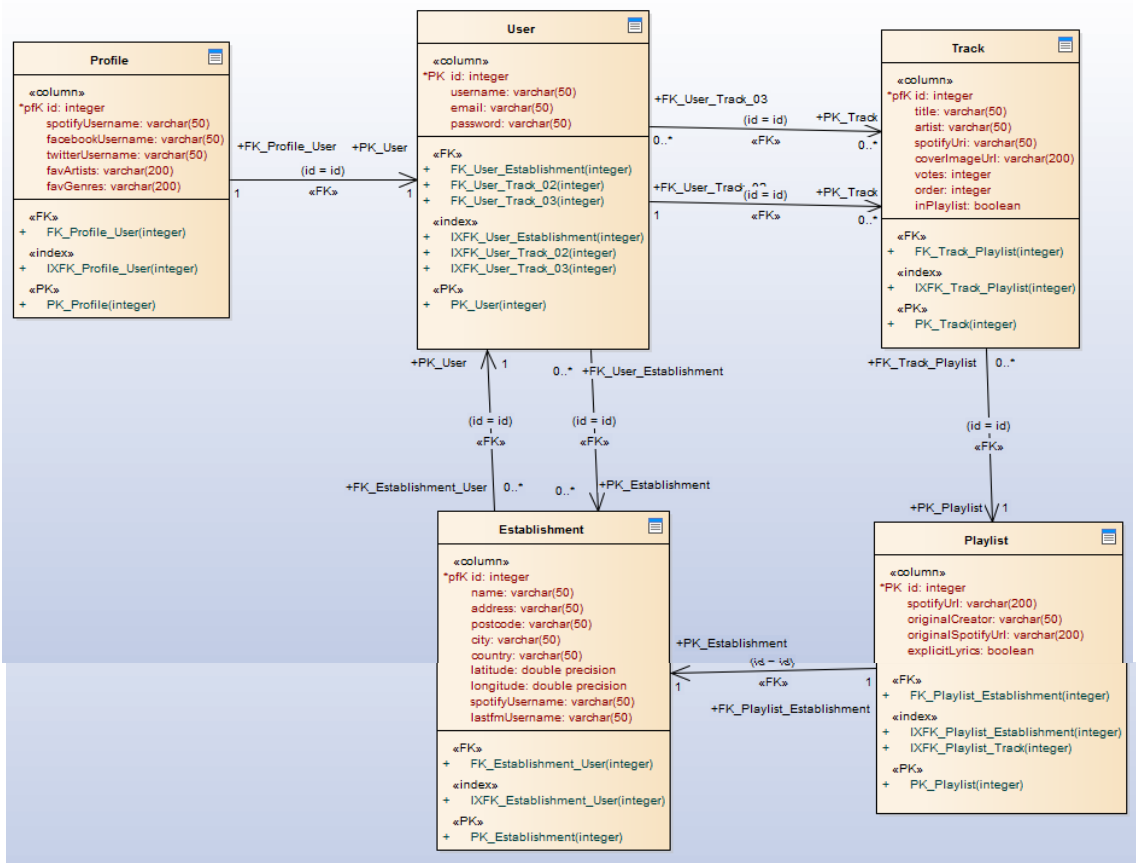



Figure 23. Data model

As it can be observed, the data model of this project is quite simple. There are only one to one, one to many and many to many relationships. The attributes and their respective type are stated in red in each model. Below them, the foreign keys of each relationship are located. All relationships between models are through their id's as seen in figure 23.

## 8.5. Navigational model

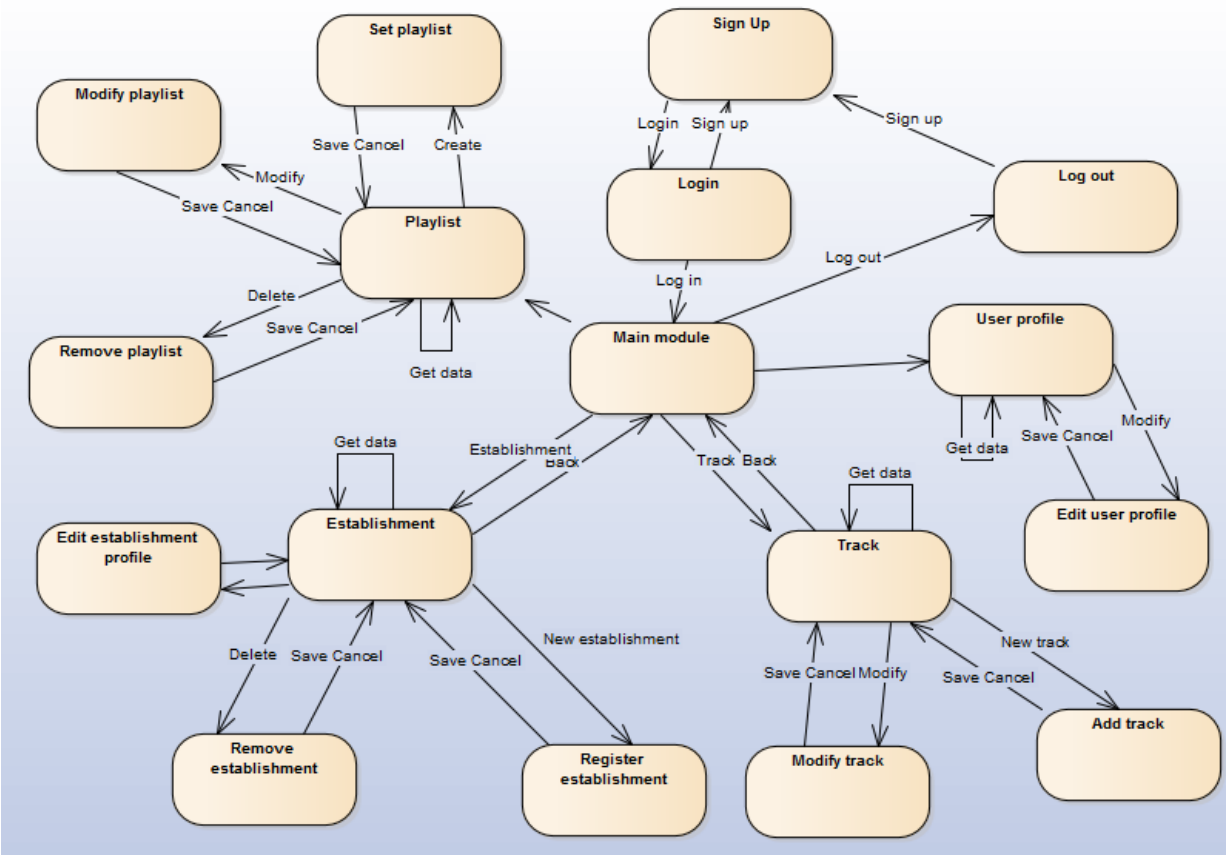



Figure 24. Navigation model

Navigation throughout the mobile app is controlled by the navigation controller of iOS. The standard *create*, *modify* and *delete* actions can be performed to each model. Only a few of them are restricted to the administrator of the product, which are:

- Delete of all models. The data are kept after the apparent removal performed by the user. They are set to disabled in reality.
- Modify: tracks. Tracks are created when a new playlist is set or a submission of a song request is done, and from that moment on, the only interaction available with tracks are through votes.
- Create: profiles. A unique profile is created for the user in the moment of sign up.
- Modify: user. A user cannot replace the email used on registration. Modification of the password is not a feature of the product at this moment.

Images of the mobile application can be found in the appendix in order to get a real impression of the final product.

# 9. Development and implementation

## 9.1. Description and justification of the selected technologies

### 9.1.1. Main frameworks, tools, services

**Django** is a high-level Python Web framework that encourages rapid development and clean and pragmatic design. Besides that, it has an excellent documentation and a very supportive community.

**Django REST Framework** is a powerful and flexible toolkit for building Web APIs. It comes with a browsable API by default and

**Heroku** is a cloud Platform-as-a-Service (PaaS) being used as a Web Application Deployment model which allows building, running and scaling applications easily and it is exactly what this project needed.

**Postgres** is a powerful, open source object-relational database system. The great side is that it is very well integrated with Heroku, called *Heroku Postgres*, which is an SQL database as a service with operational expertise built in, security by default, database forking, etc.

**iOS** is chosen because firstly, it is one of the most popular mobile OS in the market and secondly, its development environment is simple and innovative and has a great variety and very well done system libraries/APIs.

### 9.1.2. API's and libraries from third parties

**Spotipy** is a lightweight Python library for the Spotify Web API. It allows a full access to all of the music data provided by the Spotify platform and supports all of the features of the Spotify Web API including access to all end points, and support for user authorization.

**pygeocoder** is a Python library that was used to get the establishment's geolocation (latitude and longitude) from the input address so they can be displayed in the map.

**Lastfm** API is used to get the currently playing song from a Spotify account. At this moment the Spotify API does not provide this functionality. Users must have a lastfm account in order to use the app as an establishment owner.

**Alamofire** is a Swift-based HTTP networking library for iOS and Mac OS X. It provides an elegant interface on top of Apple's Foundation networking stack that simplifies a number of common networking tasks.

**SwiftyJSON** library makes dealing with JSON data with swift easier. It was specifically used for data received at the Alamofire responses.

**Locksmith** is a powerful, protocol-oriented library for working with the keychain in Swift. In this project it is used to store the token of the user which is provided after the user successfully logs in to his account.

### 9.1.3. Package and library managers

**virtualenv** is a tool to create isolated Python environments. The basic problem it addresses to is are dependencies and versions, and indirectly permissions.

**Cocoapods** is a dependency manager for Swift and Objective-C Cocoa projects. It has over 24 thousand libraries and provides a standard format for managing external libraries.

## 9.2. API

An API (Application Programming Interface) has been built to allow communication between the mobile app and the server.
The browsable API can be found at http://joogpoint.herokuapp.com/ and the admin site is located at http://joogpoint.herokuapp.com/admin/ (I will gladly provide temporary credentials to access them. Please contact me by mail).

The whole API has been documented, the format of requests and examples responses and errors can be seen at http://docs.joogpointrestapi.apiary.io/.

# 10. Testing

## 10.1. Manual tests

Manual testing has been done mostly on the mobile app part (front end). Since it is the visible part of the product, it made sense to try out the features by hand. Furthermore, the design and user experience cannot be tested automatically and has to be tested by a human in order to obtain a valuable review.

## 10.2. Automated tests

The *unittest* module built in to the Python standard library is used to write tests in Django, which is their recommended way to test code.
These tests have been done to validate that the code works as expected. When refactoring or modifying code after a significant period of time, using tests has ensured that the changes haven't affected the application's behavior unexpectedly.
With Django's test-execution framework and assorted utilities, requests simulation, test data insertion and inspection of the application's output and verification of the code have been done successfully.

# 11. Results and conclusion

The overall results of the project have been a success. The application has been designed and developed after a good amount of constant research and learning but it was all worth the time. This project has allowed me have a full vision of all the aspects involved in the development of a software project. From the necessary but tedious project management to design, development, implementation and testing of a complete product.

Often when a feature has been implemented, I can get to see that it could be improved, here and there, at what stage I should have seen that mistake or found a better way to execute and reach the goal. Practice is indeed the way you get to learn and improve on what you do. I am very pleased with the final product. Being able to choose what to build and what tools to use on every aspect has made the project a valuable experience.

## 11.1. Future enhancements

The design and style can be refined without doubts. Currently the majority of components of the app are the ones provided by Xcode by default. Also, requirements appeared during the analysis such as making the app multilingual, having a two-step authentication or being able to recover the password would be some basic but great features to be added.

The option of pushing it to the market can also be considered, although some strict rules and design principles have to be followed in order to have it published on the App Store. But it is a possibility to be considered surely.

# 12. Glossary

**ARM**: originally Acorn RISC Machine, later Advanced RISC Machine, is a family of reduced instruction set computing (RISC) architectures for computer processors, configured for various environments.

**API**: Application program interface is a set of routines, protocols, and tools for building software applications. An API specifies how software components should interact and APIs are used when programming graphical user interface (GUI) components.

**REST**: Representational state transfer (REST) or RESTful web services are one way of providing interoperability between computer systems on the internet. REST-compliant web services allow requesting systems to access and manipulate textual representations of web resources using a uniform and predefined set of stateless operations.

**PaaS**: Platform as a Service is a category of cloud computing that provides a platform and environment to allow developers to build applications and services over the internet. PaaS services are hosted in the cloud and accessed by users simply via their web browser.

**SCRUM**: Scrum is a management and control process that cuts through complexity to focus on building software that meets business needs. Management and teams are able to get their hands around the requirements and technologies and deliver working software, incrementally and empirically.

**Git**: Git is a version control system that is used for software development and other version control tasks. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows.

# 13. Sources

## Font

Quicksand Light (used in the mobile app)
http://www.dafont.com/quicksand.font

## Icons

www.icons8.com

## Swift libraries

Alamofire
https://github.com/Alamofire/Alamofire

SwiftyJSON
https://github.com/SwiftyJSON/SwiftyJSON

Locksmith
https://github.com/matthewpalmer/Locksmith

## Python libraries

spotipy
http://spotipy.readthedocs.io/en/latest/

pygeocoder
https://pypi.python.org/pypi/pygeocoder

# 14. Bibliography


[1] Wikipedia Contributors, "iOS," Wikipedia, 27 Feb 2016. [Online]. Available: https://en.wikipedia.org/wiki/IOS.

[2] "Mobile Music Streaming: Driving the Next Digital Revolution," App Annie, 1 Dec 2015. [Online]. Available: http://blog.appannie.com/mobile-music-streaming-driving-the-next-digital-revolution/.

[3] A. Allsopp, "Deezer vs Spotify vs Tidal vs Amazon Prime Music vs Apple Music comparison: What is the best music streaming service?," Tech Advisor, 11 Feb 2016. [Online]. Available: http://www.pcadvisor.co.uk/review/audio/deezer-vs-spotify-tidal-apple-music-amazon-prime-music-comparison-review-3523953/.

[4] Spotify AB, "Spotify Web API," Spotify, Feb 2016. [Online]. Available: https://developer.spotify.com/web-api/.

[5] Rockbot, "Rockbot," 2016. [Online]. Available: https://rockbot.com/.

[6] OutLoud Inc., "OutLoud," 2014. [Online]. Available: https://outloud.dj.

[7] James Litjens, "Jukestar," 2015. [Online]. Available: http://jukestar.mobi/.

[8] c-burn systems Ltd, "SecretDJ," 2012. [Online]. Available: http://www.secretdj.com/.

[9] "Geocoding and reverse geocoding," 1 5 2016. [Online]. Available: http://chrisalbon.com/python/geocoding_and_reverse_geocoding.html.

[10] J. Pradel, Software Projects Management Course Slides, Departament d'Enginyeria de Serveis i Sistemes d'Informació. ESSI, 2014.

[11] GEP Professors, "Notes and resources on Project Management,," 2016. [Online]. Available: http://atenea.upc.edu/moodle/.

[12] PES Professors, Software Engineering Project Course Slides, Departament d'Enginyeria de Serveis i Sistemes d'Informació. ESSI, 2015.

[13] ER Professors, Requirements Engineering Course Slides, Departament d'Enginyeria de Serveis i Sistemes d'Informació. ESSI, 2015.

[14] "Django REST Framework documentation," 2016. [Online]. Available: http://www.django-rest-framework.org.

[15] "Spotify Web API Endpoint Reference," 2016. [Online]. Available: https://developer.spotify.com/web-api/endpoint-reference/.

[16] "Heroku Postgres," 2016. [Online]. Available: https://www.heroku.com/postgres .

[17] "Django," 2016. [Online]. Available: https://docs.djangoproject.com/en/1.10/.

[18] "MapKit Tutorial," [Online]. Available: https://www.raywenderlich.com/90971/introduction-mapkit-swift-tutorial.

[19] "Legal guidelines for the use of location data on the web," 10 3 2016. [Online]. Available: https://www.smashingmagazine.com/2016/03/location-data-web-development-and-the-law/.

[20] "Spotify Privacy Information," [Online]. Available: https://support.spotify.com/us/article/spotify-privacy-info/.

[21] "Heroku documentation," 2016. [Online]. Available: https://devcenter.heroku.com/categories/reference.

# 15. Annex

## 15.1. API Documentation

It is attached as an additional material.

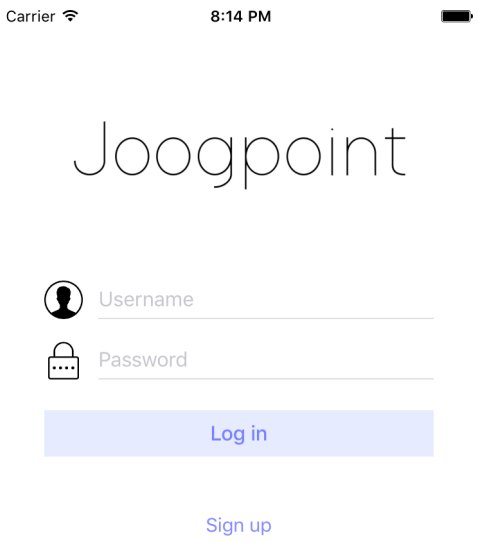
## 15.2. Backend code repository

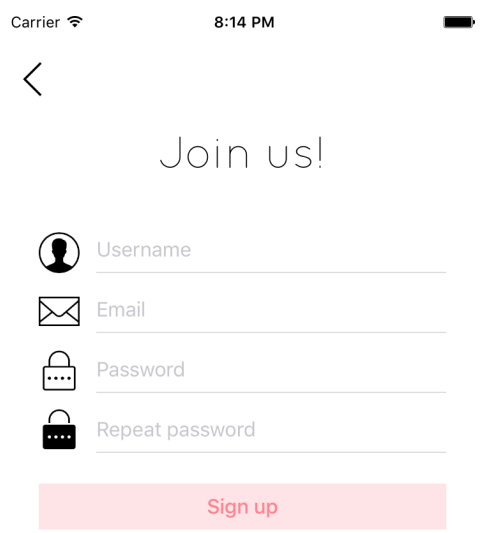
https://github.com/wailingtam/joogpoint

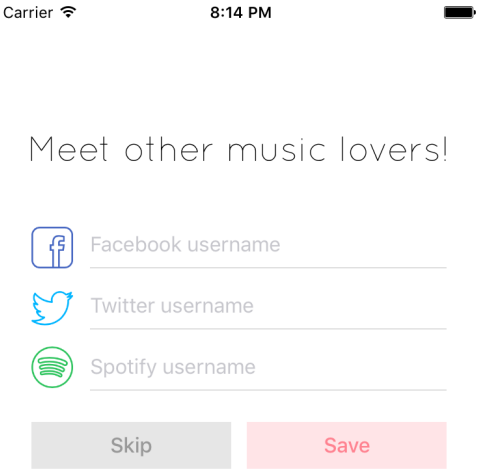
## 15.3. App code repository

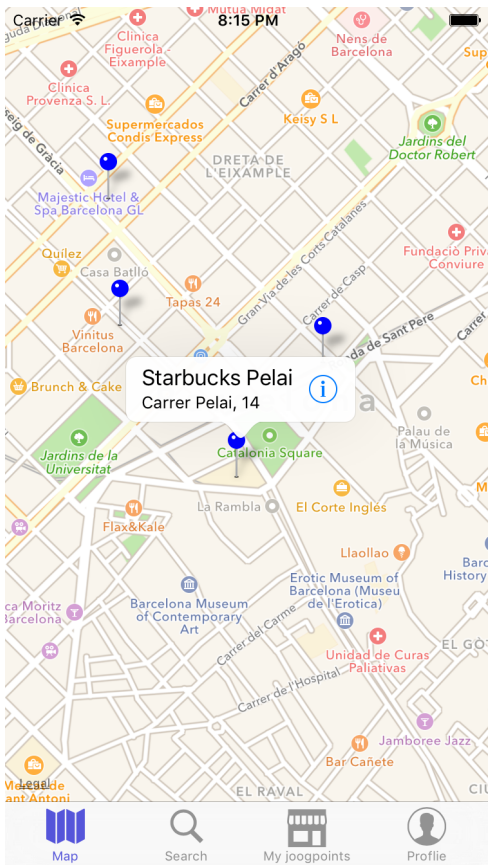
https://github.com/wailingtam/joogpointApp

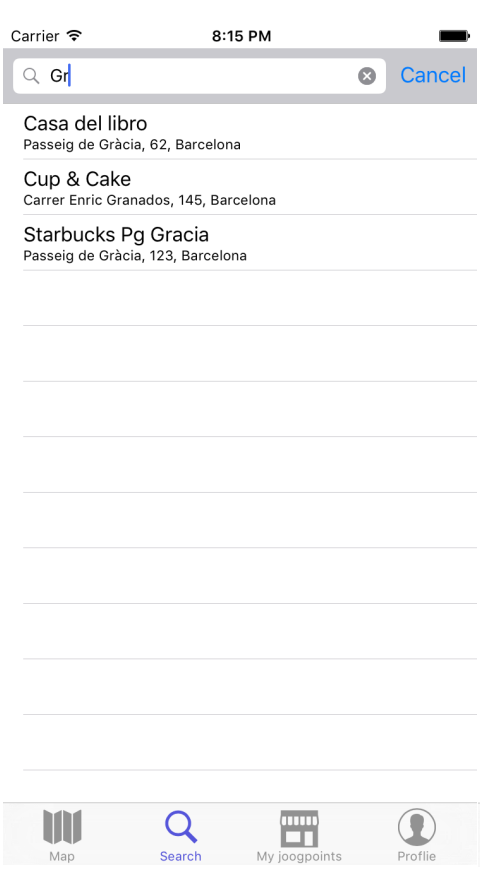
## 15.4. Application screens

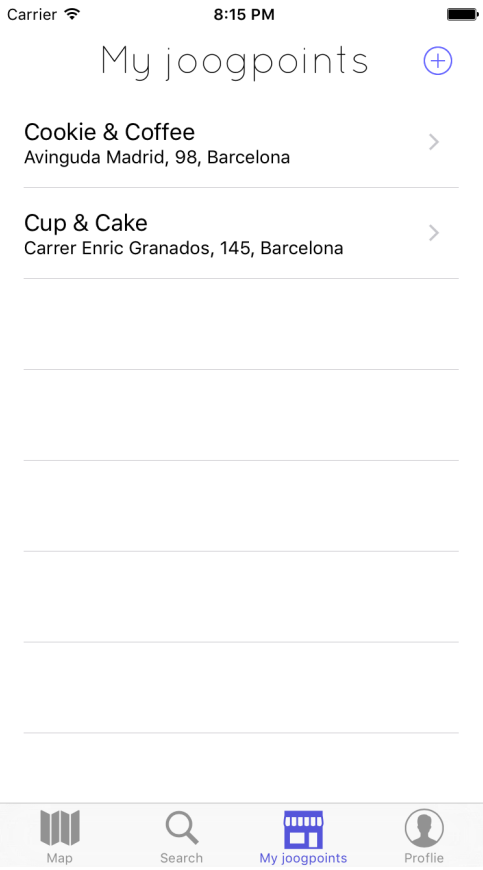


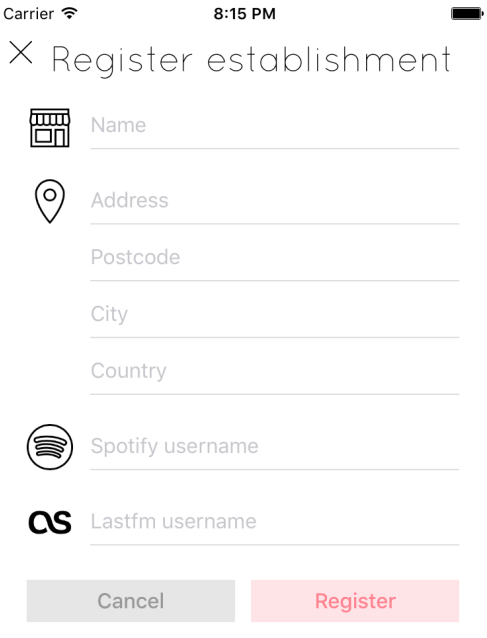



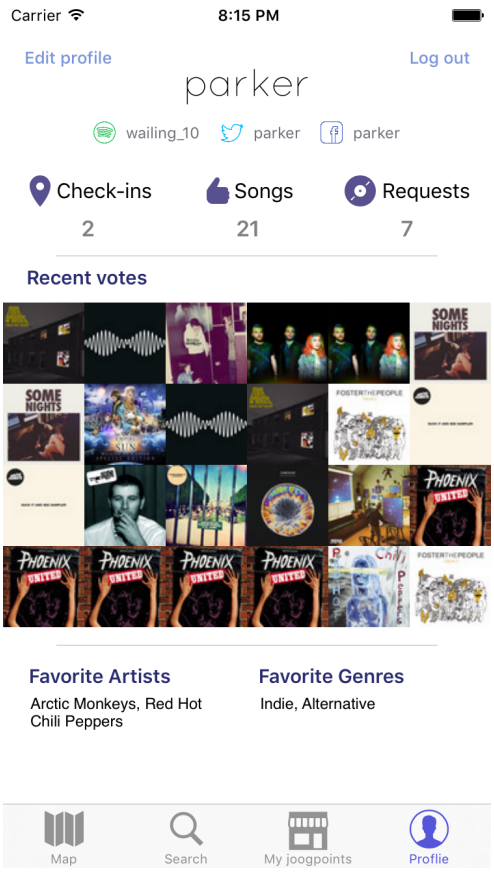



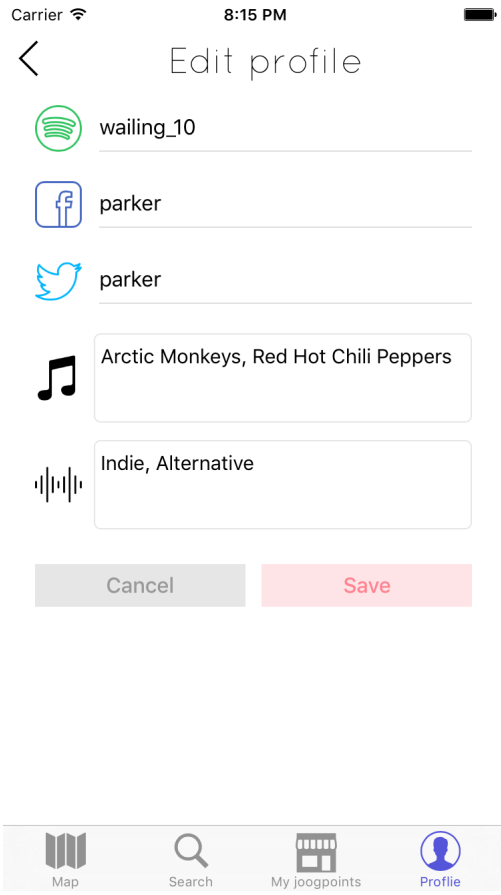



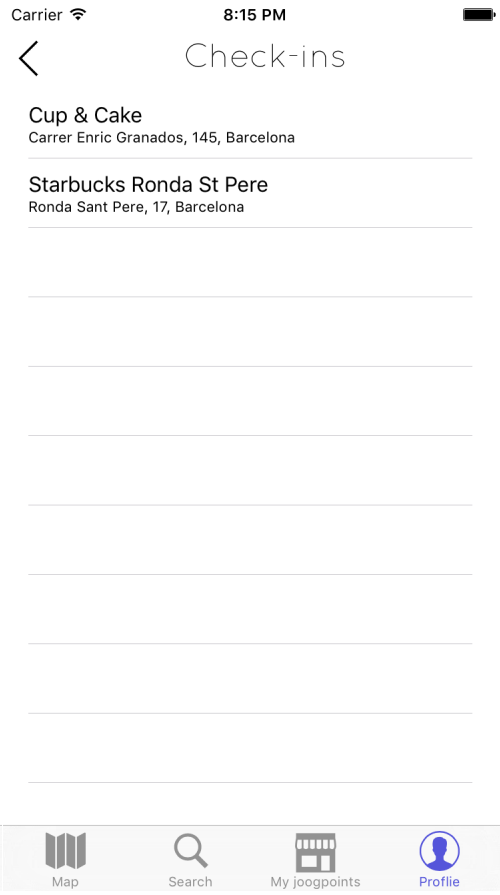

Carrier
8:15 PM
Gr
Cancel
Casa del libro
Passeig de Gràcia, 62, Barcelona
Cup & Cake
Carrer Enric Granados, 145, Barcelona
Starbucks Pg Gracia
Passeig de Gràcia, 123, Barcelona
Map
Search
My joogpoints
Profile

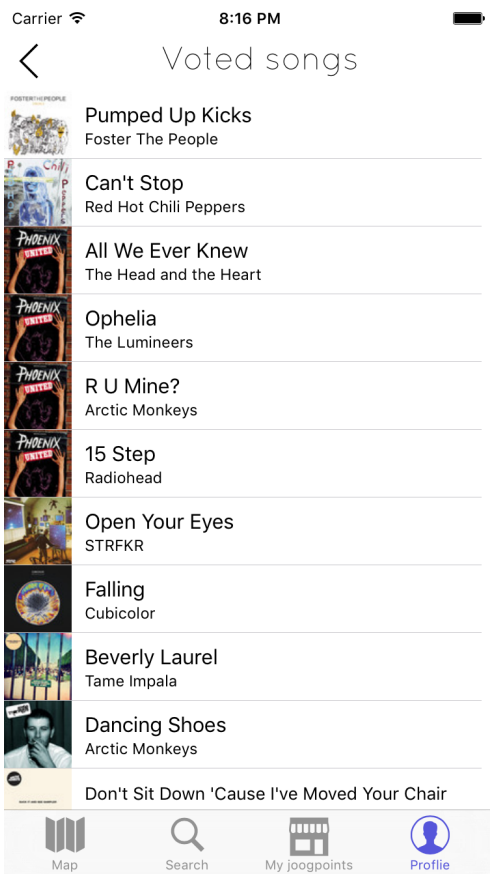

Carrier
8:15 PM
My joogpoints
Cookie & Coffee
Avinguda Madrid, 98, Barcelona
Cup & Cake
Carrer Enric Granados, 145, Barcelona
Map
Search
My joogpoints
Profile

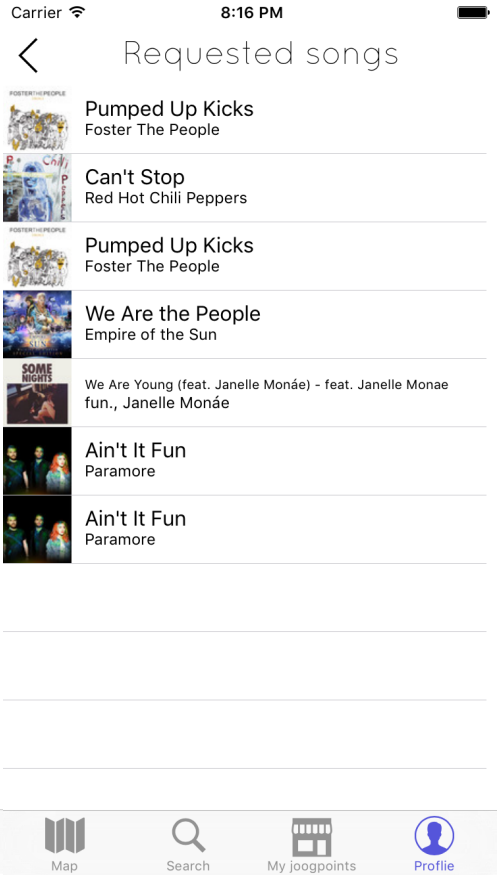

Carrier
8:15 PM
Register establishment
Name
Address
Postcode
City
Country
Spotify username
Lastfm username
Cancel
Register

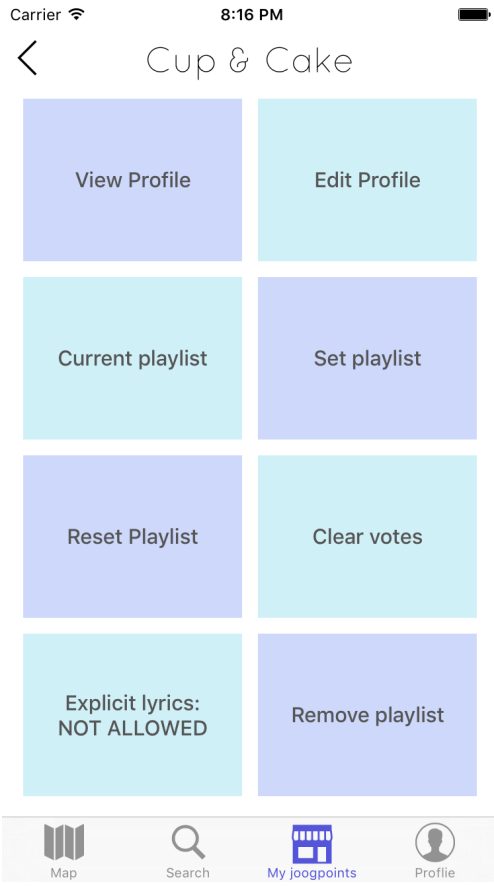

Carrier
8:15 PM
Edit profile
Log out
parker
wailing_10
parker
parker
Check-ins
2
Songs
21
Requests
7
Recent votes
Favorite Artists
Arctic Monkeys, Red Hot Chili Peppers
Favorite Genres
Indie, Alternative
Map
Search
My joogpoints
Profile

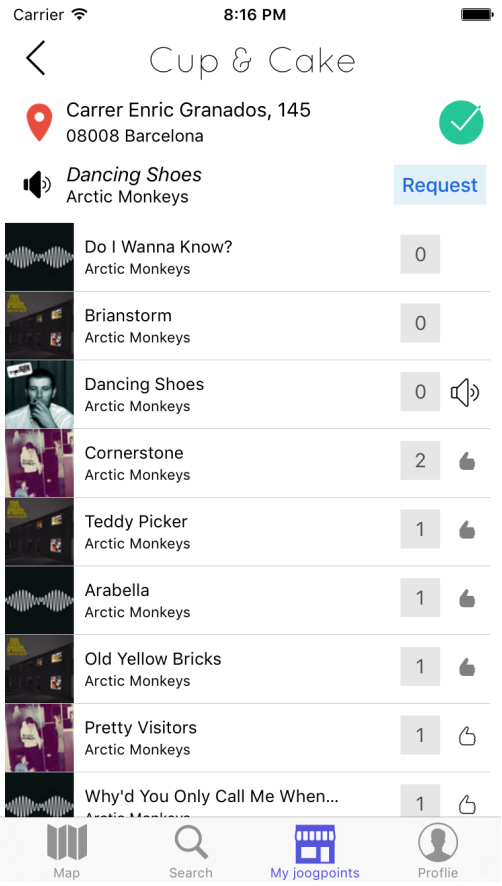

Carrier
8:15 PM
Edit profile
wailing_10
parker
parker
Arctic Monkeys, Red Hot Chili Peppers
Indie, Alternative
Cancel
Save
Map
Search
My joogpoints
Proflie

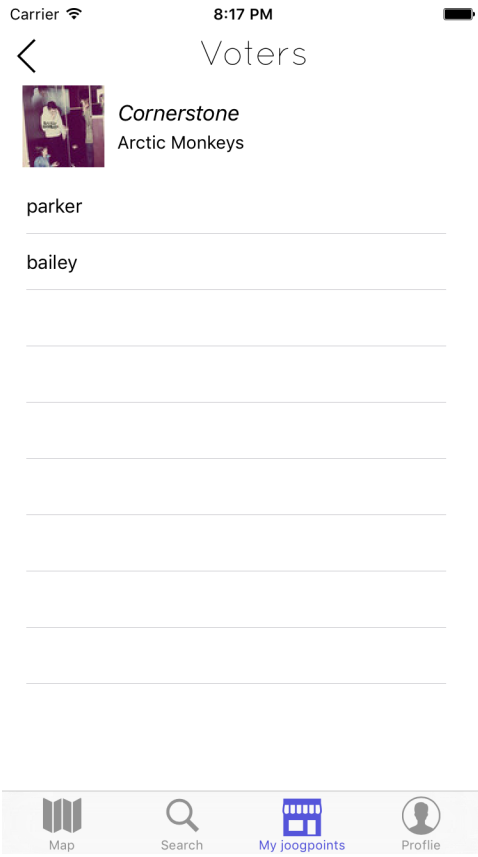

Carrier
8:15 PM
Check-ins
Cup & Cake
Carrer Enric Granados, 145, Barcelona
Starbucks Ronda St Pere
Ronda Sant Pere, 17, Barcelona
Map
Search
My joogpoints
Proflie

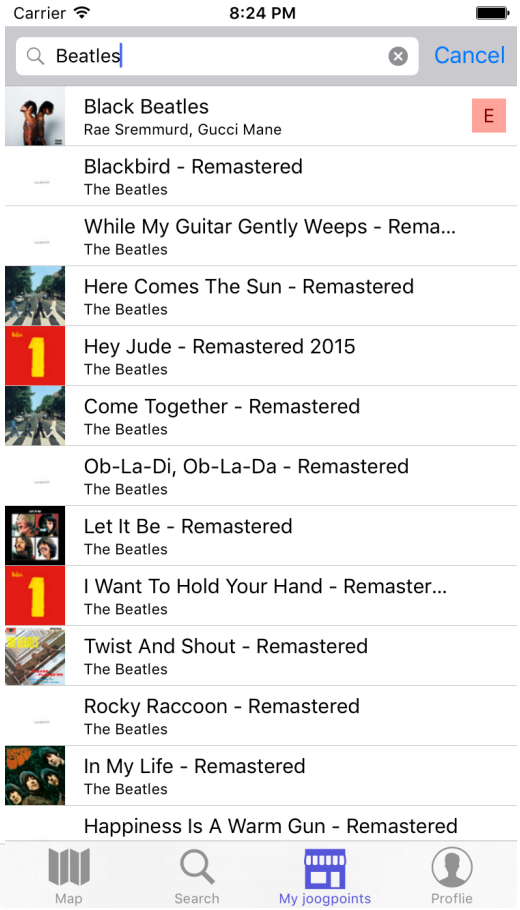

Carrier
8:16 PM
Voted songs
Pumped Up Kicks
Foster The People
Can't Stop
Red Hot Chili Peppers
All We Ever Knew
The Head and the Heart
Ophelia
The Lumineers
R U Mine?
Arctic Monkeys
15 Step
Radiohead
Open Your Eyes
STRFKR
Falling
Cubicolor
Beverly Laurel
Tame Impala
Dancing Shoes
Arctic Monkeys
Don't Sit Down 'Cause I've Moved Your Chair
Map
Search
My joogpoints
Proflie


Carrier
8:16 PM
Requested songs
Pumped Up Kicks
Foster The People
Can't Stop
Red Hot Chili Peppers
Pumped Up Kicks
Foster The People
We Are the People
Empire of the Sun
We Are Young (feat. Janelle Monáe) - feat. Janelle Monae
fun., Janelle Monáe
Ain't It Fun
Paramore
Ain't It Fun
Paramore
Map
Search
My joogpoints
Proflie


Carrier
8:16 PM
Cup & Cake
View Profile
Edit Profile
Current playlist
Set playlist
Reset Playlist
Clear votes
Explicit lyrics: NOT ALLOWED
Remove playlist
Map
Search
My joogpoints
Profile


Carrier
8:16 PM
Cup & Cake
Carrer Enric Granados, 145
08008 Barcelona
Dancing Shoes
Arctic Monkeys
Request
Do I Wanna Know?
Arctic Monkeys
0
Brianstorm
Arctic Monkeys
0
Dancing Shoes
Arctic Monkeys
0
Cornerstone
Arctic Monkeys
2
Teddy Picker
Arctic Monkeys
1
Arabella
Arctic Monkeys
1
Old Yellow Bricks
Arctic Monkeys
1
Pretty Visitors
Arctic Monkeys
1
Why'd You Only Call Me When...
1
Map
Search
My joogpoints
Profile


Carrier
8:17 PM
Voters
Cornerstone
Arctic Monkeys
parker
bailey
Map
Search
My joogpoints
Profile


Carrier
8:24 PM
Beatles
Cancel
Black Beatles
Rae Sremmurd, Gucci Mane
E
Blackbird - Remastered
The Beatles
While My Guitar Gently Weeps - Rema...
The Beatles
Here Comes The Sun - Remastered
The Beatles
Hey Jude - Remastered 2015
The Beatles
Come Together - Remastered
The Beatles
Ob-La-Di, Ob-La-Da - Remastered
The Beatles
Let It Be - Remastered
The Beatles
I Want To Hold Your Hand - Remaster...
The Beatles
Twist And Shout - Remastered
The Beatles
Rocky Raccoon - Remastered
The Beatles
In My Life - Remastered
The Beatles
Happiness Is A Warm Gun - Remastered
Map
Search
My joogpoints
Profile