Bilkent University

Department of Computer Engineering

Senior Design Project

Musync

Analysis Report

Ahmet Çandıroğlu, Anıl Erken, Berk Mandıracıoğlu, Halil İbrahim Azak

Supervisor: Asst. Prof. Dr. M. Mustafa Özdal
Jury Members: Assoc. Prof. Dr. Özcan Öztürk, Prof. Dr. Cevdet Aykanat

Analysis Report Nov 12, 2018

1. Introduction

Music is a common interest that many people enjoy and rest their souls. It may become a cumbersome procedure to find music that many people enjoy. In our daily lives, we are surrounded with music by means of our environment such as restaurants, cafes, bars, etc. Therefore, it is even more important to create a suitable playlist to satisfy majority based on their collective music taste. However, majority can not be satisfied every time, hence it is also necessary to recommend new places that users might enjoy the general music taste and even discover new songs.

Musync aims to facilitate creation of playlists that are dynamically modified by both analysing the music tastes of users and their feedback on the current playlist. Musync is basically a digital jukebox which collects data about their music tastes from its current users and initialises a playlist. Moreover, users are able to add songs to the playlist and manipulate currently played songs by the power of bidding. Users have different values in our system, thus not every user has the same bidding power. Users that are regular customers of the place have more power in the creation of the playlist in that place.

In this report, first we will provide a description of the system we propose. After that, requirements of it will be examined under relative sections, like functional requirements. Then, we will provide scenarios and use cases for the system. After those, object and class model, dynamic models and finally user interface information will be provided in respective order.

2. Proposed system

2.1 Overview

The aim of Musync is enhancing the enjoyment of music in public areas such as restaurants, cafes, and bars. Achieving this requires owners of these areas to be able to gather feedback from theirs patrons constantly and control the playlist with frequent changes. This is a very cumbersome work from the perspectives of both owners and the patrons. The purpose of Musync is to automate most of this work and provide an easy to use service to enhance the music playing in public areas.

Musync enhances the playlist by both gathering and analyzing music taste of patrons, and providing an easy way to contribute to the playlist for patrons. At first, Musync will require place owner to setup and configure the application the way they like. It will provide options to limit music to genres owner prefers to not disturb the theme of the place. Also this preferences will be used to guide users to places with the music that satisfies their taste. After the first configuration, starting the application will be as easy as a click of a mouse. Musync will handle the remaining part of work for place owners.

Patrons of public places will be able to connect to the playlist of the place they are in easily after entering a 4 number pin with the help of geolocation. If a patron chooses to authenticate their Spotify account, their music preferences will be analysed and contribute to the playlist of the place. Musync also provides bidding system to control playlist, a funny and fair way to engage with the playlist. Patrons will be given an amount of points based on how frequently they visit the place. They will have two ways to engage with the playlist, either by adding a track of their to the playlist or bidding for the next track to be played.

Musync will also be available for individuals to host their own shared playlist. It will be available for free for a limited time in a month. The time will be adjusted to allow people use it freely for their occasional gatherings like birthday parties. In case a user wants to use it more frequently, they will need to upgrade their account to a premium account public places will use.

2.2 Functional Requirements

Place Owner Specific Requirements

- *Place owners* have to create an account with an email and password.
- Anyone with a Spotify account should be able to subscribe to our as place owner system. System should be able to support be two types of accounts: free trial and paid subscription.
    - *Free trial* allows up to 5 days of usage in a month. This account type is suitable for individuals who want to use the application occasionally for their home parties etc.
    - *Paid subscription* allows unlimited usage of our system. These users are mostly owners of a place(cafe, bar, restaurant, etc) who are willing to give their customers best music experience.
- Owners should be able to limit song requests to specific genres.

Visitor Specific Requirements

- *Visitors* should be able to use the application without an account. A unique account will be created automatically on first use.
- *Visitors* should be able to add new songs to the common playlist if it is suitable to genre set by the owner.
- *Visitors* should be able to bid on 3 songs selected by Musync out of the playlist to play them next. The bidding is done with points earned by visiting the place.
- *Visitors* should be able to add songs or bid without linking their Spotify account to use.
- *Visitors* should be able to link their Spotify account to contribute to the common playlist.
- *Visitors* should be able to earn points which can be used to add/bid for songs by being a regular customer of a place.
- *Visitors* should earn their points with respect to time they are spending in the place.
- *Visitors* should be able to earn extra point based on their visit frequency. For example, 5 visits in a month should earn more coins than 5 visits in a year.

- *Visitors* should be able to receive place recommendations according to their music tastes and location, and view them in a map.
- *Visitors* should be able to view all places with their preferred genre and currently played song in a map.
- *Visitors* should agree to share their GPS locations to our system in order to use it.
- *Visitors* should select a place based on their GPS location and verify that they are in that place by entering a 4-digit code.
- *Visitors* should be able to view a list of recommended places based on their Spotify activity,music taste, their GPS location and accuracy.
- *Visitors* should be able to see the song they have added to playlist on top of the auto generated list.

General Requirements

- Musync should be able to gather user's Spotify data by using Spotify Web API
- Musync should be able offer and display new places to user by utilizing Google Maps API.
- Musync should keep the login date of each *visitor* for each place and count them logged in for 5 hours
- Musync should keep the log of recently played songs(up to 10) for each place in order to recommend new places to visitors.
- *Place owners* should be able to host from both the computer or the phone

2.3 Nonfunctional Requirements

2.3.1 Usability

- The user interface should be simple and user-friendly.
- The users should have no difficulty to start using the system.

- The users should be able to use the system without any login procedures, if they wish.

2.3.2 Supportability

- The system should be able to work in any web browser.
- The system should be compatible with third party APIs such as Google Maps and Spotify.

2.3.3 Reliability

- System should be able to create joint playlists that will satisfy the majority, independent of the number of concurrent users.

2.3.4 Efficiency

- The delay between a user requesting a song and it being added to the playlist should not exceed 10 seconds.
- Backend response time of the website (the time starting when an HTTP request taken and ending when the server starts to send frontend data) should not exceed 200 milliseconds.
- The joint playlist should be created in a reasonable time and space complexity.

2.3.5 Security

- Information gathered from users and authentication related tokens should be stored securely.

2.3.6 Scalability

- The service provided to each place and its customers should not be affected by the amount of places and active users.
- System should be able to handle multiple song requests coming from different users simultaneously.

2.3.7 Extensibility

- The system should be able to include new features without difficulties.

2.4 Pseudo requirements

- Application must run on every popular browser properly.
- Development of the system must be completed before CS Fair 2019.

2.5 System models

2.5.1 Scenarios

2.5.1.1 User Registration

Actors: User

Entry Conditions:

- User opens registration page.
- User doesn’t have another account logged in.

Exit Conditions:

- User successfully registers OR
- User cancels registration.

Main Flow of Events:

- User enters information
- User chooses a password
- User clicks to submit button

2.5.1.2 User Login

Actors: User

Entry Conditions:

- User opens login page.
- User doesn’t have another account logged in.

Exit Conditions:

- User successfully logins OR
- User cancels login

Main Flow of Events:

- User enters login information.
- User clicks login button.

2.5.1.3 User Login with Spotify

Actors: User

Entry Conditions:

- User opens login page.
- User doesn’t have another account logged in.

Exit Conditions:

- User successfully logins through Spotify OR
- User cancels login.

Main Flow of Events:

- User clicks to Login with Spotify button.
- Application redirects user to Spotify.
- User authorizes Musync.
- Application redirects user to home page.

2.5.1.4 User Connects Spotify Account

Actors: User

Entry Conditions:

- User is logged in.
- User opens profile page.

Exit Conditions:

- User successfully connects Spotify account OR
- User cancels Spotify connection.

Main Flow of Events:

- User clicks on Login with Spotify button.
- Application redirects user to Spotify.
- User authorizes Musync.
- Application redirects user to profile page.

2.5.1.5 User Disconnects Spotify Account

Actors: User

Entry Conditions:

- User is logged in.
- User opens settings page.
- User has Spotify account connected.

Exit Conditions:

- User successfully disconnects Spotify account OR
- User cancels disconnecting Spotify.

Main Flow of Events:

- User clicks to Disconnect Spotify button.
- Application asks User if she/he is sure.

- User chooses an option.

2.5.1.6 User Connects to Place

Actors: User

Entry Conditions:

- User opens home page.
- User allows application to use Location data.

Exit Conditions:

- User successfully connects to place OR
- User cancels connection.

Main Flow of Events:

- Application shows possible places based on location.
- User chooses a place.
- User enters a place specific 4-number pin.

2.5.1.7 User Requests Song

Actors: User

Entry Conditions:

- User is logged in.
- User is connected to a place.

Exit Conditions:

- User adds a song to the playlist OR
- User cancels request.

Main Flow of Events:

- User clicks on Add Song button on homepage.
- User searches for a song in Song Search Screen.
- User selects the song.

2.5.1.8 User Bids on Next Songs

Actors: User

Entry Conditions:

- User is connected to a place.

Exit Conditions:

- User successfully bids on a song OR
- User cancels bidding.

Main Flow of Events:

- User chooses a song.
- User enters the amount of points she/he wants to bid.
- Application warns user if there isn’t available amount of points.

2.5.1.9 User Creates New Place

Actors: User

Entry Conditions:

- User is logged in with a registered account.
- User opens Create Place page.

Exit Conditions:

- User successfully creates a new place OR
- User cancels creating a new place.

Main Flow of Events:

- User enters place information.
- User chooses place location on map.
- User chooses whether the place will be a permanent place or not.
- Application warns user if he doesn’t have premium account and tries to create a permanent place.

2.5.1.10 User Connects Place to Spotify

Actors: User

Entry Conditions:

- User is logged in.
- User has a place created.
- User opens settings page.

Exit Conditions:

- User successfully connects place to Spotify OR
- User cancels Spotify connection.

Main Flow of Events:

- User chooses his/her place to connect.
- User clicks Connect Place to Spotify button.
- Application redirects user to Spotify.
- User authorizes Musync.

2.5.1.11 User Disconnects Spotify from a Place

Actors: User

Entry Conditions:

- User is logged in.
- User has a place connected to Spotify.
- User opens settings page.

Exit Conditions:

- User successfully disconnects the place from Spotify.
- User cancels disconnection.

Main Flow of Events:

- User chooses his/her place to disconnect from Spotify.
- Application asks user if she/he is sure.

- User chooses an option.

2.5.1.12 User Searches Places

Actors: User

Entry Conditions:

- User is logged in.

Exit Conditions:

- User clicks on home button.

Main Flow of Events:

- User clicks on search places button on Home screen.
- User searches for places and view them as list or on map.

2.5.1.13 User Starts a new Shared Playlist

Actors: User

Entry Conditions:

- User has an active Spotify account.
- User is logged in as *place owner*

Exit Conditions:

- Place owner started the system and initial playlist is auto generated

Main Flow of Events:

- *Place owner* logins to system through the desktop client or the browser client
- *Place owner* configures the set of music genres to play in the place.
- *Place owner* starts to host the system.
- *Visitor* goes to Musync browser client url.
- Musync checks the cookies of the visitor
    - If they don’t have account a new account is created automatically
    - If they have an account they login automatically

- If the user has linked their Spotify account, Musync gathers relevant music to the genre of the place and the current playlist and adds it to playlist.

2.5.2 Use case model

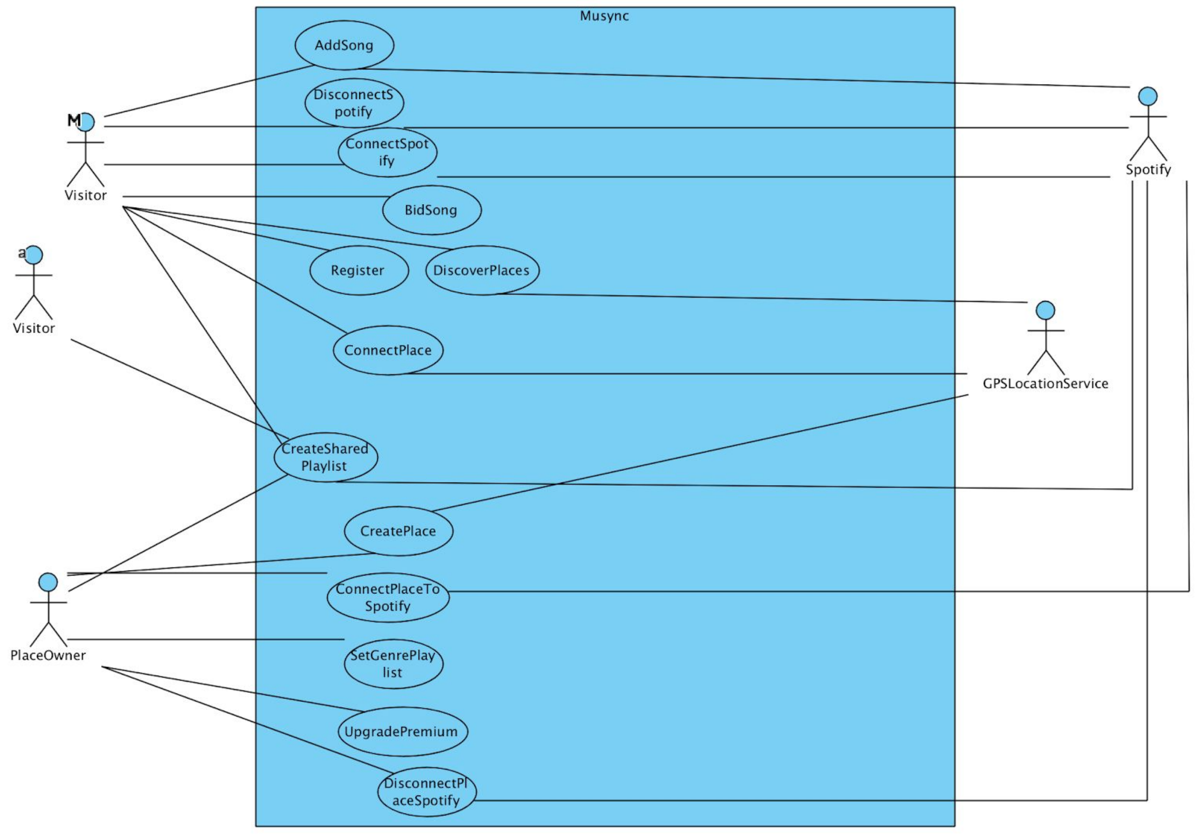


Figure 1: Use case model

2.5.3 Object and class model

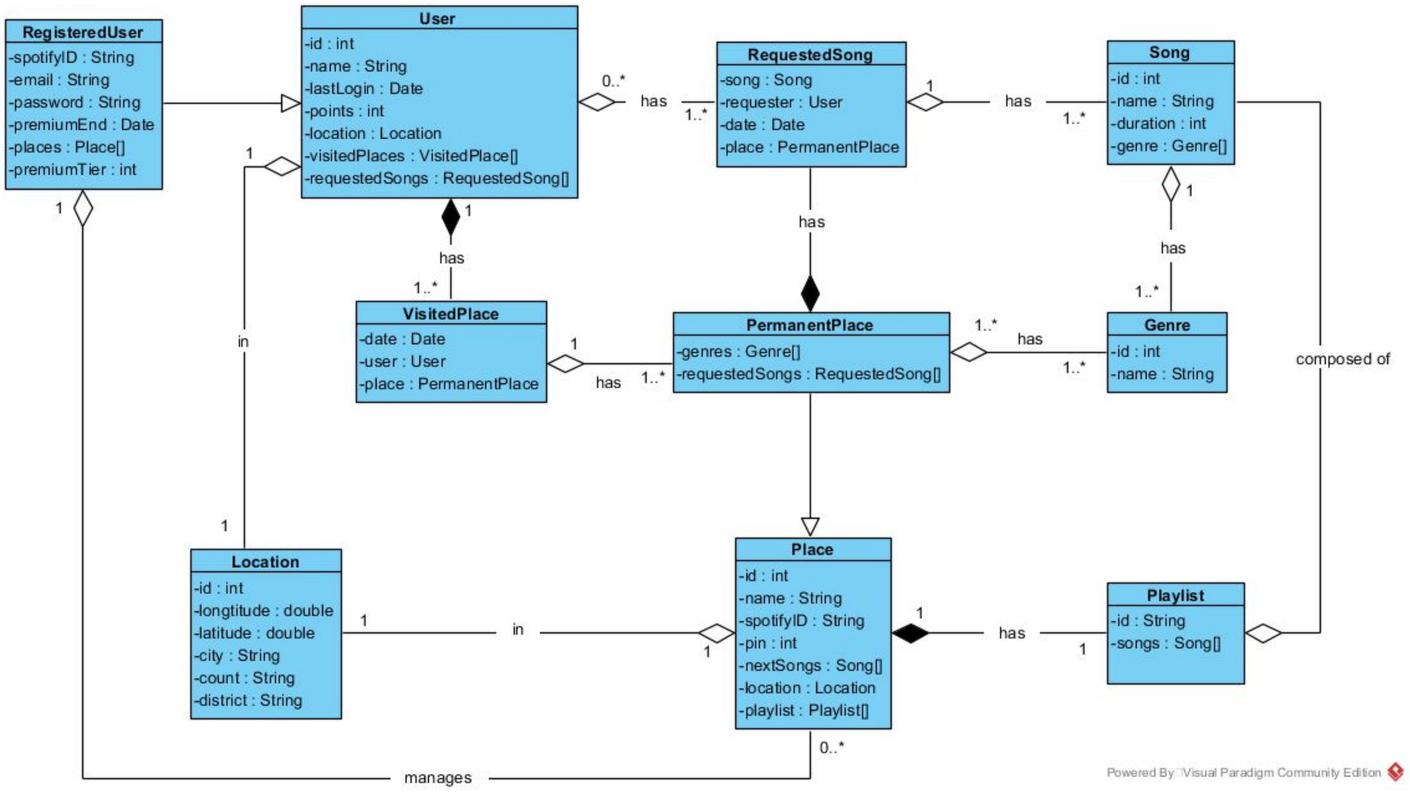


Figure 2: Model classes

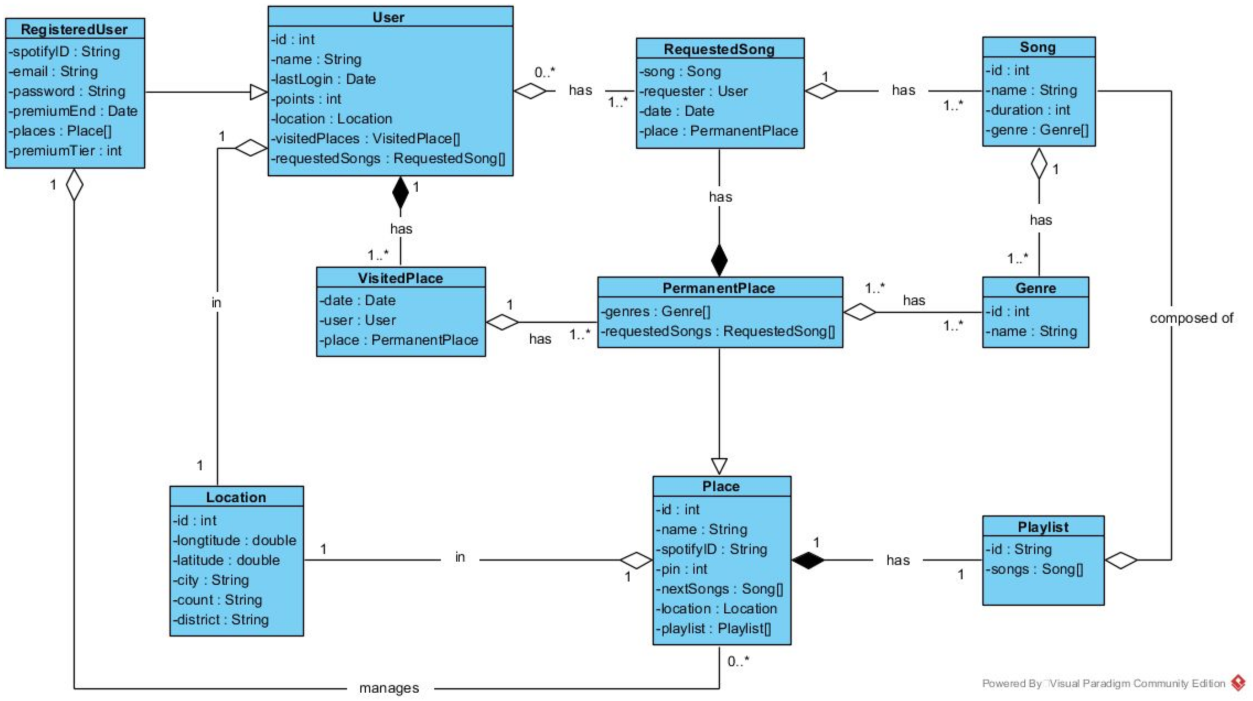


Figure 3: Controller classes

MainController

This controller class is main controller class of the application. It is responsible for storing and providing access to other controller classes. It also stores shared model class instances such as Song and Genre instances.

SpotifyController

SpotifyController class provides methods for Spotify API related operations.

UserController

This controller class is responsible for storing User classes and operations involving them. For example checking if a user exists and creating a new user will be done through this class.

PlaceController

PlaceController is responsible for storing Place classes and operations involving them. Creating and controlling places will be done through this class.

LocationController

This class will store and control locations. Operations related to locations, such as finding nearby locations, will be done through this class.

User

This class is the base class of all user types. It has the basic attributes such as password, id, location, last login date informations. Moreover, User has the points attribute in order to enable the visitor to bid on music. User has visited places and requested songs attributes in order to keep track of the visitor activity and increase points if the visitor is a regular visitor of the place. Aforementioned attributes serve as necessary data to make place recommendations to the user.

RegisteredUser

This class inherits User class. Users that registered through either email or Spotify will be a registered user. Registered users will be able to create and control places. To allow this functionality, RegisteredUser stores user information such as email and Spotify auth token, and place related information such as owned places and premium account information.

Place

Place class is the base class for place related classes. It holds the basic information such as location, Spotify token, pin and song bid information. This class also holds the playlist currently playing in the place.

PermanentPlace

PermanentPlace inherits Place class. The difference between the two classes is that permanent places corresponds to public places that will be used frequently and will have a history of songs played. PermanentPlace holds genres allowed in place and requested songs.

VisitedPlace

VisitedPlace serves as the data for keeping track of visitor - place interaction. VisitedPlace enables musync to check if a user is a regular visitor of the place and increase their points accordingly. Moreover, Musync should be able to make recommendations based on the previously visited places of the visitors which can be learned from VisitedPlace class.

Genre

This class is utilized in PermanentPlace class which enables to limit a place’s allowed genres to play. Moreover, this class is used by Song to keep the list of genres a song belongs.

Song

This class is used to represent songs. It stores id, name, duration and genre of the song.

RequestedSong

This class is used to represent requested songs in a place. It stores the song instance, the user who requested it, the date and the place. This information will be used to reach the songs a user requests, so that her/his music taste will be understood and place recommendations will be made accordingly.

Playlist

Playlist has an id and a song array. Playlists are used inside Place classes to represent joint playlist of the place.

Location

Location is used by the User and Place classes to determine where they are placed on a map. Therefore, this class enables to verify if the user is in fact in a place and their locations overlap. Moreover, this class is used by the PlaceController to make relevant place recommendations to the users.

2.5.4 Dynamic models

2.5.4.1 Sequence Models

2.5.4.1.1 User Registration and Spotify Connection

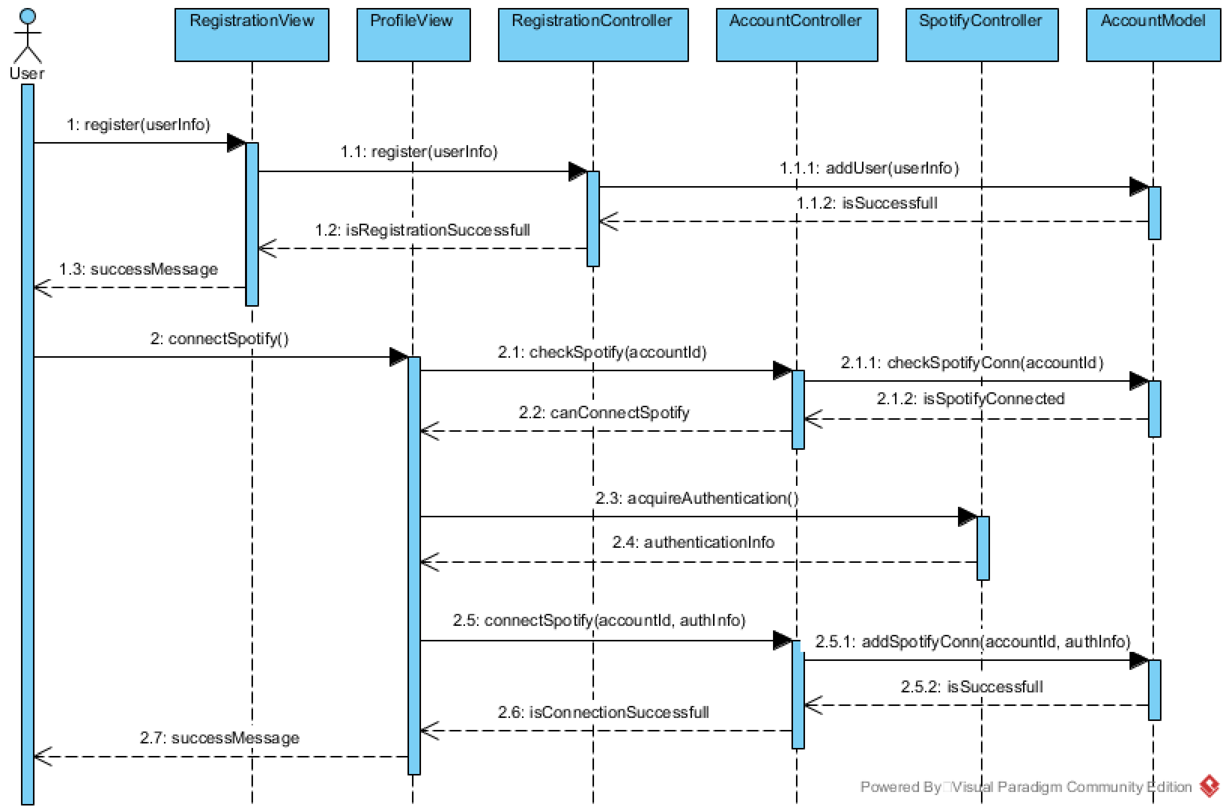


Figure 4: User registration and spotify connection

This sequence diagram explains new registration followed by connecting Spotify account. User will start to process by visiting the registration page that is controlled by *RegistrationView*. After user enters required information and submits it, it will be send to *RegistrationController*. This controller will check if the information is valid and after that will send it to *AccountModel*. *AccountModel* controls the user data. It will add user if it can and return error information if it cannot, in situations such as account information that needs to be unique already existing in database. After that, *RegistrationController* will notify view about the result and *RegistrationView* will inform the user.

After user registers a new account, she/he will be redirected to profile page controlled by *ProfileView*. In this page user can choose to connect Spotify account. If she/he does so, it will be checked first that if an existing connection exists through *AccountController* and *AccountModel*. After that *ProfileView* will redirect user to *SpotifyController* that will control the acquiring of Spotify authentication. If the authentication is succesfull, *ProfileView* will send authentication information to *AccountController*. *AccountController* will return result after sending information to *AccountModel* and getting result. Finally *ProfileView* will show result to user.

2.5.4.1.2 Next Songs Panel and Bidding On a Song

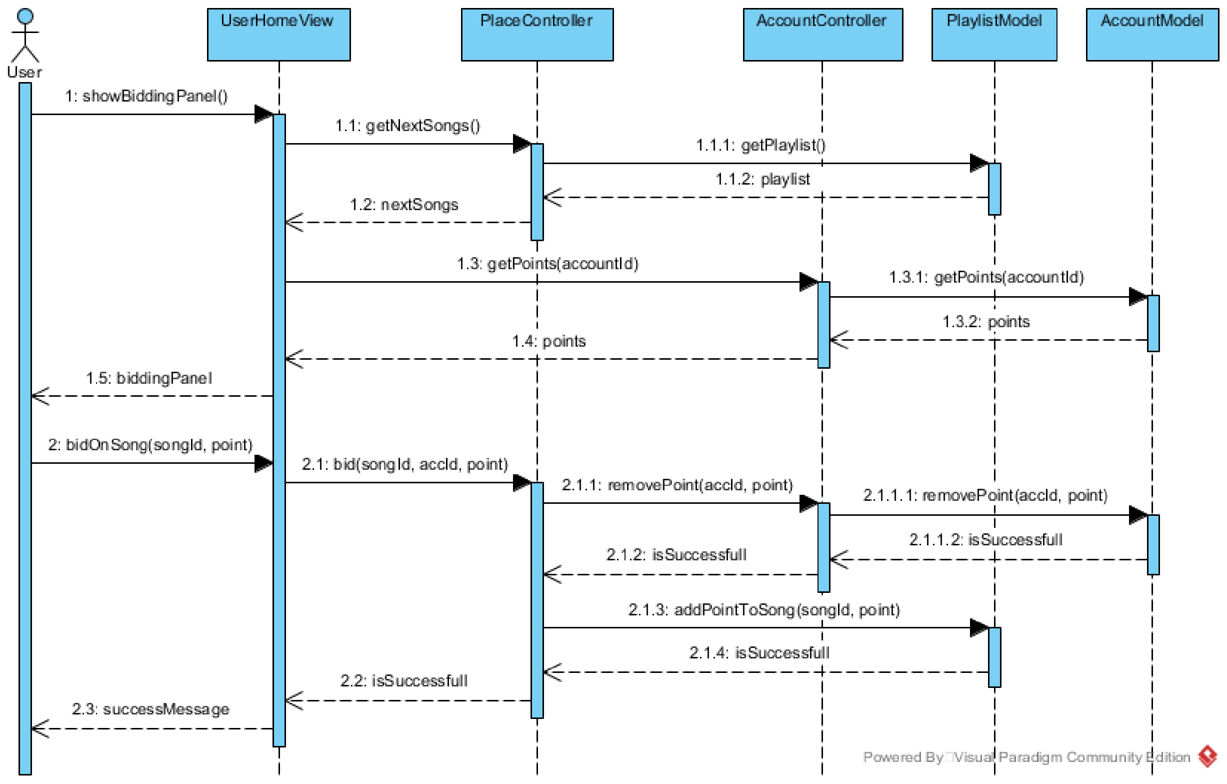


Figure 5: Next songs panel and bidding on a song

This sequence diagram explains how users will be able to bid on next songs. Users will be able to see a bidding panel on Home page if they are logged in and connected to a place. After user clicks on bidding panel button, *UserHomeView* will fetch next 3 songs on the playlist through *PlaceController*. The controller will get this data through *PlaylistModel*. *UserHomeView* will also fetch user points through *AccountController* which will fetch data through *AccountModel*. Then *UserHomeView* will show bidding panel consisting of this data.

When a user wants to bid on a song, *UserHomeView* will send this request to *PlaceController*. This controller will remove the points first through *AccountController* and *AccountModel*. If it is successfull, it will add points to the song through *PlaylistModel* and return result of the operation. Finally *UserHomeView* will notify user with the result.

2.5.4.1.3 Song Request Panel and Requesting a Song

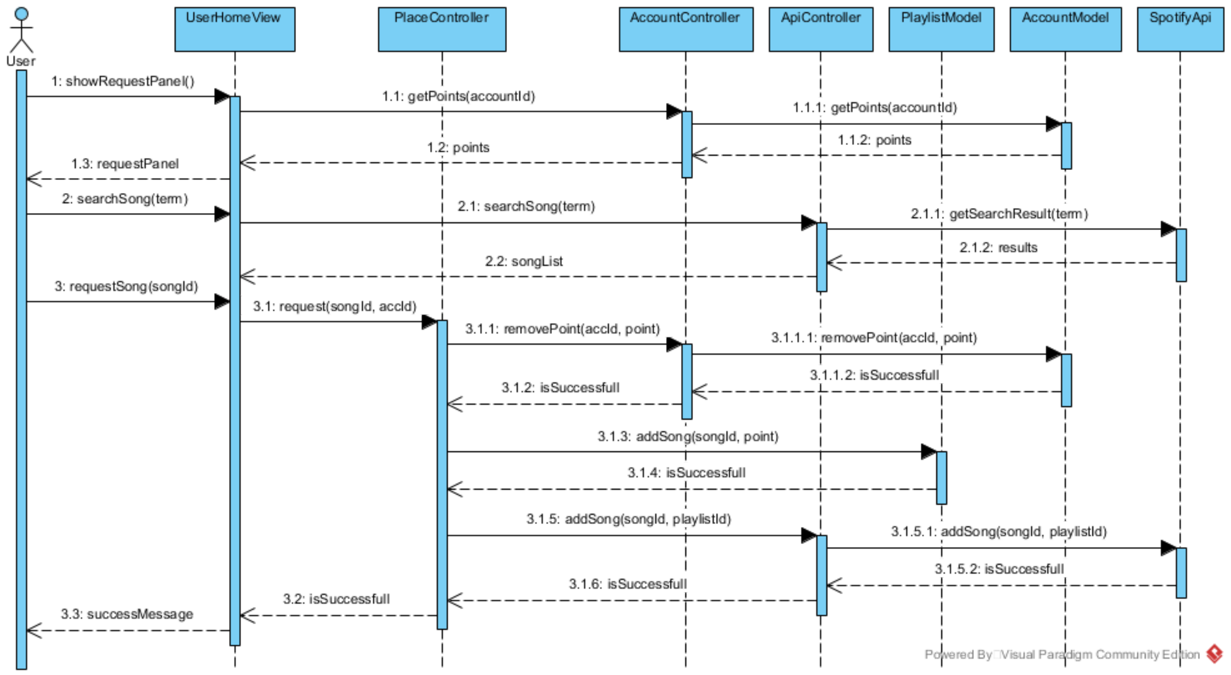


Figure 6: Song request panel and requesting a song

This sequence diagram explains how users will be able to request a song. Like bidding panel, request panel will be shown on Home page. *UserHomeView* will only fetch user points. If user has enough points, she/he will be able to search for songs to add to the playlist. *UserHomeView* will send search term to *ApiController*. *ApiController* will fetch results from *SpotifyApi*. The results will be return back to *ApiController* and *UserHomeView* in order.

After user finds a song to add and clicks to request button, *UserHomeView* will redirect this request to *PlaceController*. This controller will remove required points through *AccountController* and *AccountModel* first. If it is successfull, the song will first be added to our database through *PlaylistModel*. After that song will be added to Spotify playlist through *ApiController* and *SpotifyApi*. The result of the operation will be shown to the user.

2.5.4.1.4 User Login

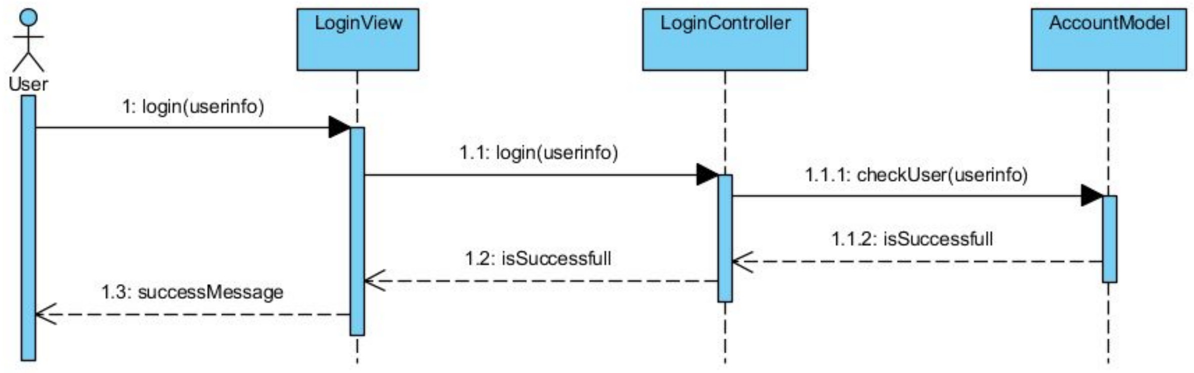


Figure 7: User login

This sequence diagram explains how users will be able to login to system. User will enter the login credentials on login page. After that, credentials are sent to *LoginController*. There, credentials will be checked to see if they are valid. If they are valid, data is sent to *AccountModel* to check if such user exists. Based on the output, a message is returned to user and she is redirected to homepage if login is successful.

2.5.4.1.5 User Login with Spotify

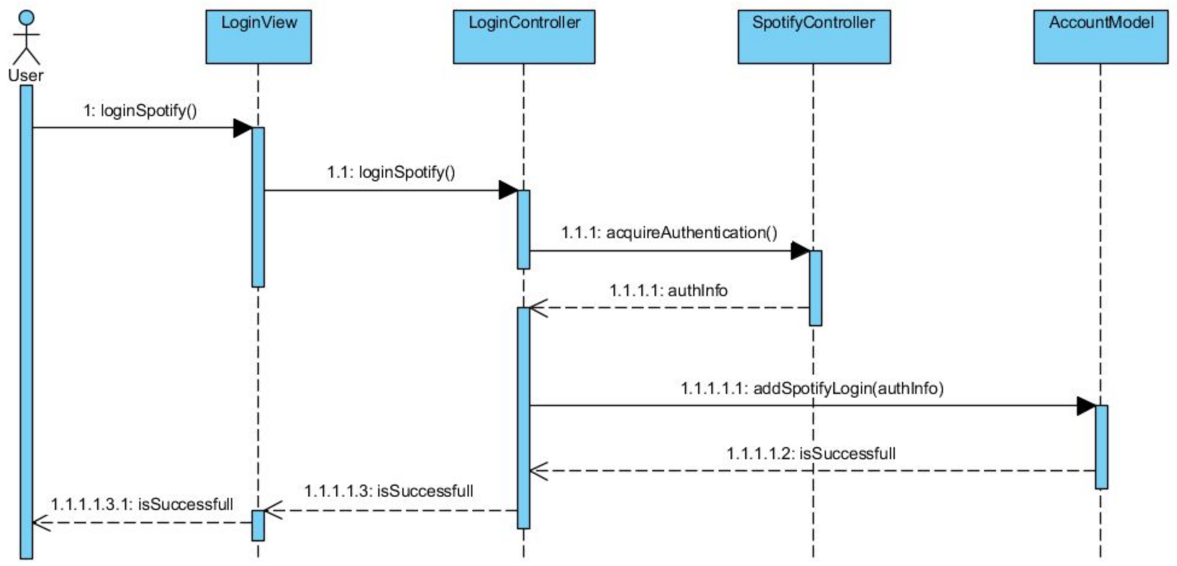


Figure 8: User login with Spotify

This sequence diagram explains how user will be able to login with her/his Spotify account. User will select Spotify login option on login panel and this request will be sent to *LoginController*. This controller will request Spotify login from *SpotifyController* which will connect user to Spotify and will return the authentication info after user logins to Spotify. Authentication info is sent to *AccountModel* to realize the login in our system. After that, a message is returned to user and redirection to homepage is done if login is successful.

2.5.4.1.6 User Connects to a Place

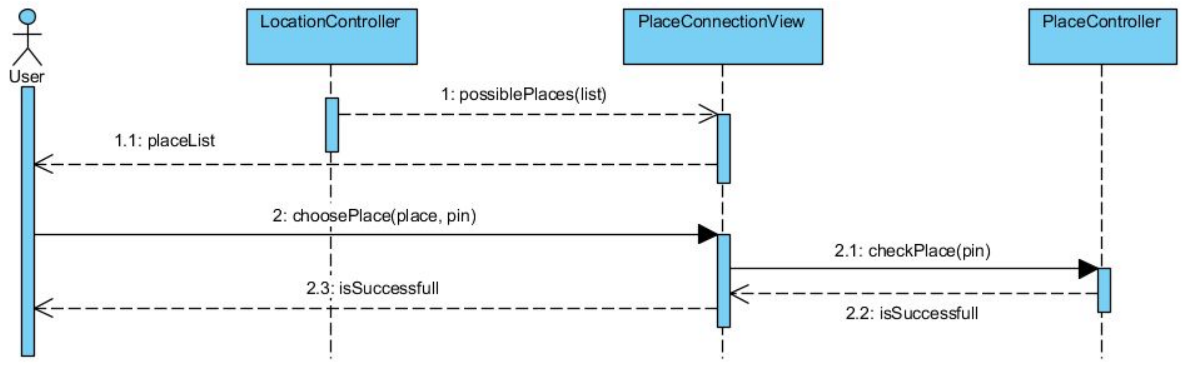


Figure 9: User connects to a place

This diagram explains how user can connect to a place to contribute to playlist and request songs. When user enters place connection page, *LocationController* finds the possible places based on user's location, and sends the list of possible places to *PlaceConnectionView.* There, place list is displayed to user, and user selects the one she/he wants to connect and enters its 4-digit pin. This data is sent to *PlaceController* to check if the pin is correct. If pin is true, connection is established and a success message is returned to user.

2.5.4.1.6 User Creates a New Place

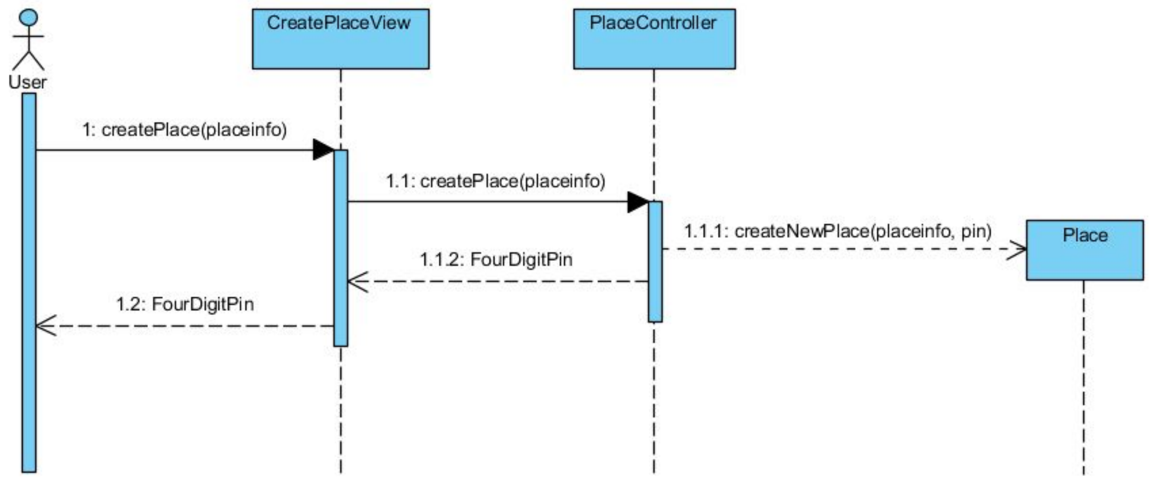

User
CreatePlaceView
PlaceController
Place
1: createPlace(placeinfo)
1.1: createPlace(placeinfo)
1.1.1: createNewPlace(placeinfo, pin)
1.1.2: FourDigitPin
1.2: FourDigitPin

Figure 10: User creates a new place

This diagram explains the creation of a new place. User enters place information on place creation window. This information is sent to *PlaceController* which checks the validity of given info and creates a new *Place* instance if it is valid. Unique 4-digit pin of the new place is returned to user.

2.5.4.2 Activity Diagram

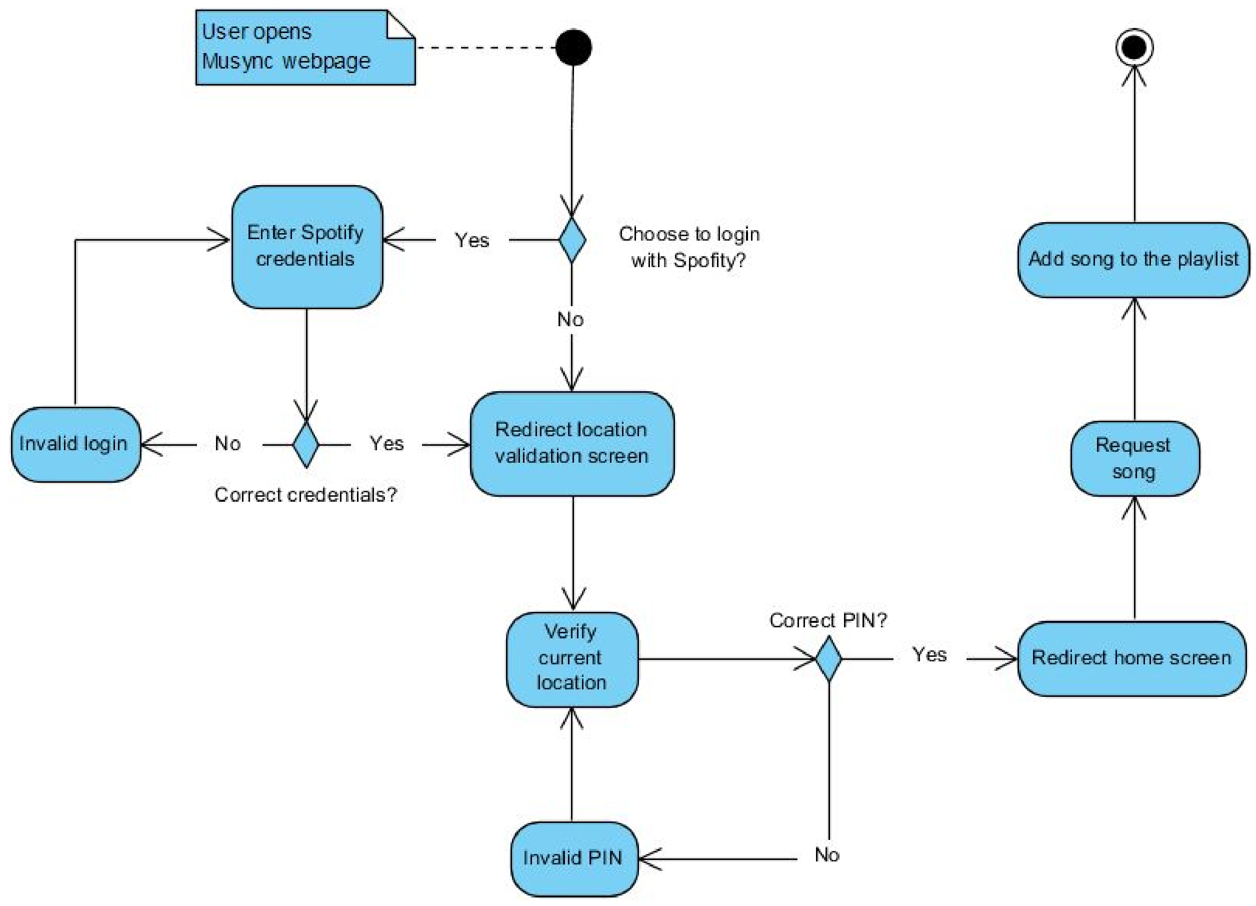


Figure 11: Activity diagram

This activity diagram shows the process of user adding a song to the playlist step by step. User can login with or without Spotify as they prefer. They have to validate their location by entering a 4-digit unique PIN code which are given by the place owner. After validation, user can click on Add Song button which would take user to the Song Search Screen. They can select the song they want and it would be added.

2.5.5 User interface - navigational paths and screen mock-ups

Our aim is to maximize the user experience and for this reason user interface will be as simple as it can be. Users will be able to use the application without having to manually authenticate. There will be no login screen, users will be authenticated automatically whenever they open the webpage. Users can use their Spotify accounts to authenticate, If they want.

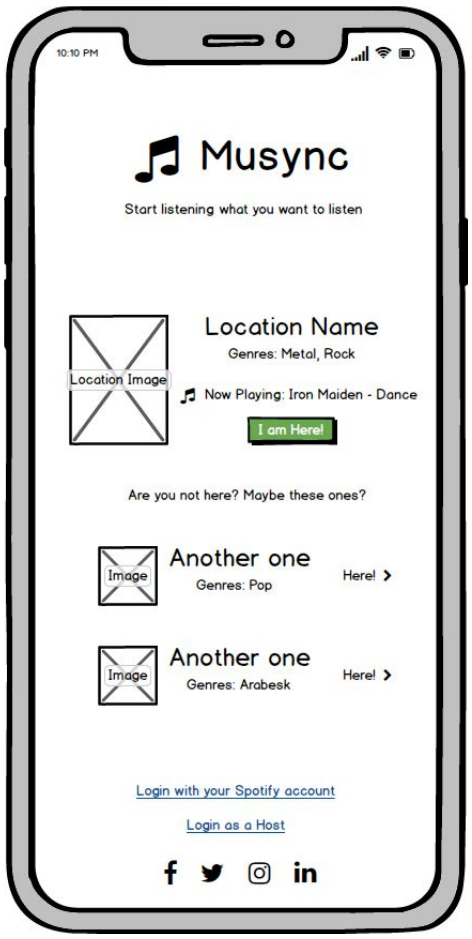


Figure 12: Location detection screen

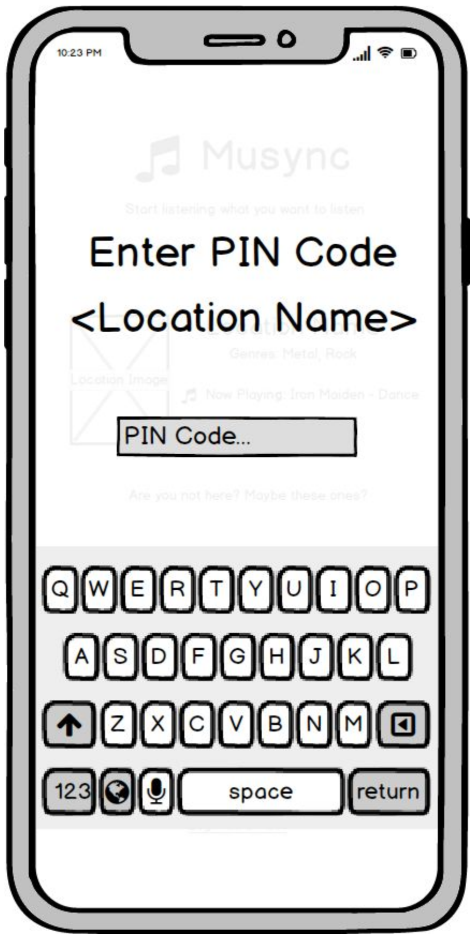


Figure 13: Location validation screen

Users will select their location and they will enter a PIN code to validate their location. The PIN code will be unique for each location and the place owner will let users to know their PIN code.

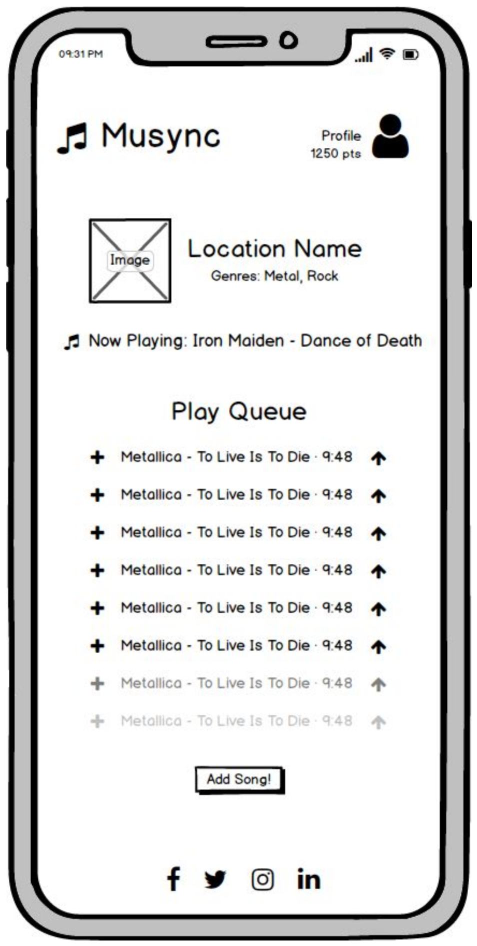

09:31 PM
Musync
Profile
1250 pts
Image
Location Name
Genres: Metal, Rock
Now Playing: Iron Maiden - Dance of Death
Play Queue
Metallica - To Live Is To Die · 9:48
Metallica - To Live Is To Die · 9:48
Metallica - To Live Is To Die · 9:48
Metallica - To Live Is To Die · 9:48
Metallica - To Live Is To Die · 9:48
Metallica - To Live Is To Die · 9:48
Metallica - To Live Is To Die · 9:48
Metallica - To Live Is To Die · 9:48
Add Song!

Figure 14: Home screen

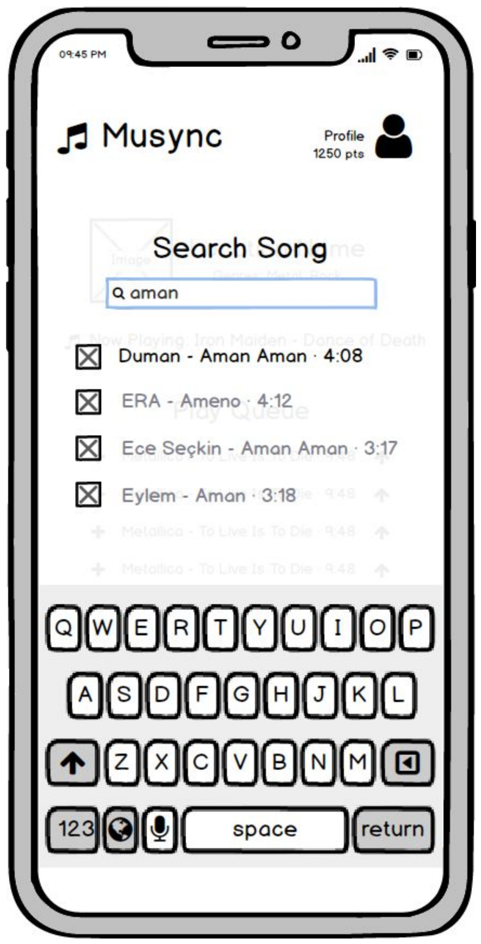
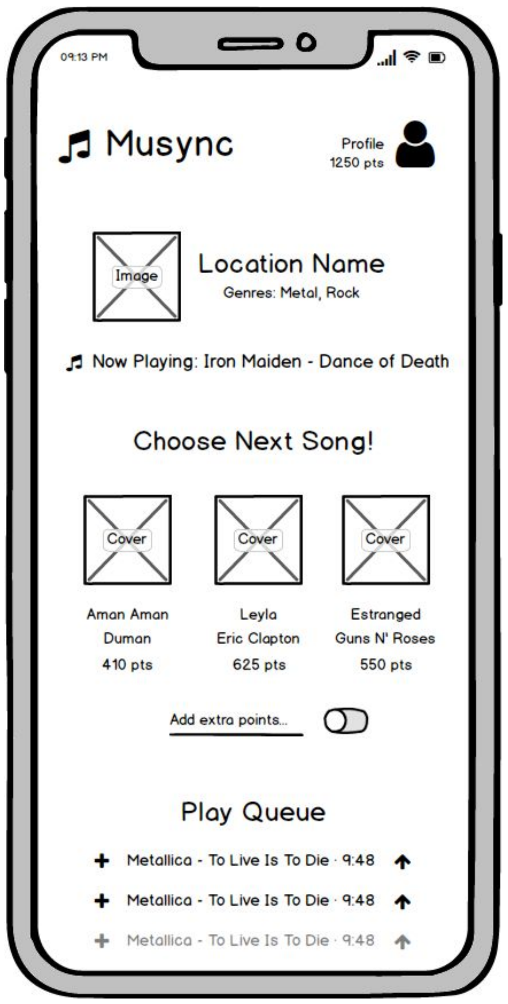


Figure 15: Song search screen

Users will see currently playing song and play queue in the home screen. They can add a song to the queue by simply searching the song. Anyone can add a song to the playlist. There will be a bidding procedure to select the next song. Users will be able to use extra points if they want.



09:13 PM

Musync

Profile
1250 pts

Image

Location Name
Genres: Metal, Rock

Now Playing: Iron Maiden - Dance of Death

Choose Next Song!

Cover | Cover | Cover

Aman Aman
Duman
410 pts

Leyla
Eric Clapton
625 pts

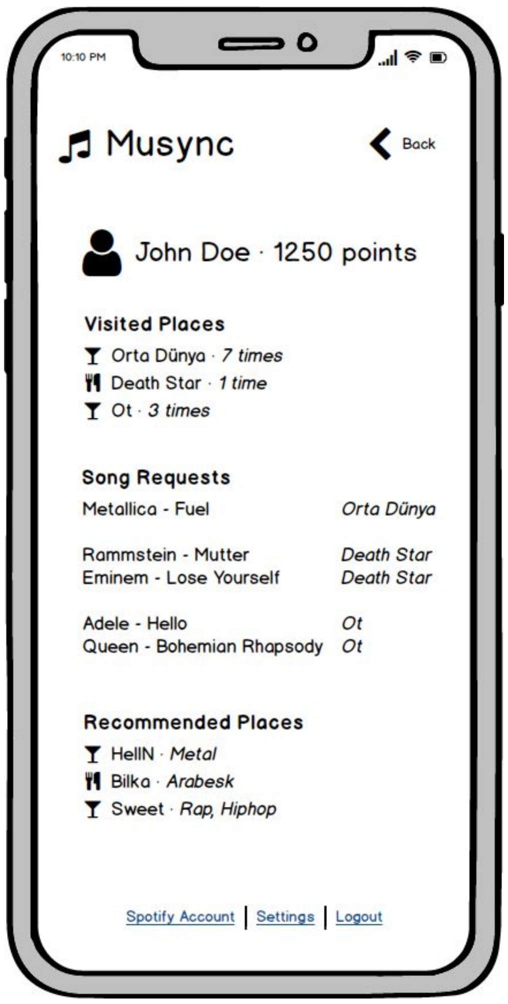
Estranged
Guns N' Roses
550 pts

Add extra points...

Play Queue

Metallica - To Live Is To Die · 9:48
Metallica - To Live Is To Die · 9:48
Metallica - To Live Is To Die · 9:48

Figure 16: Bidding screen



10:10 PM

Musync

Back

John Doe · 1250 points

Visited Places

Orta Dünya · *7 times*
Death Star · *1 time*
Ot · *3 times*

Song Requests

Metallica - Fuel	*Orta Dünya*
Rammstein - Mutter	*Death Star*
Eminem - Lose Yourself	*Death Star*
Adele - Hello	*Ot*
Queen - Bohemian Rhapsody	*Ot*

Recommended Places

HellN · *Metal*
Bilka · *Arabesk*
Sweet · *Rap, Hiphop*

Spotify Account | Settings | Logout

Figure 17: User profile screen

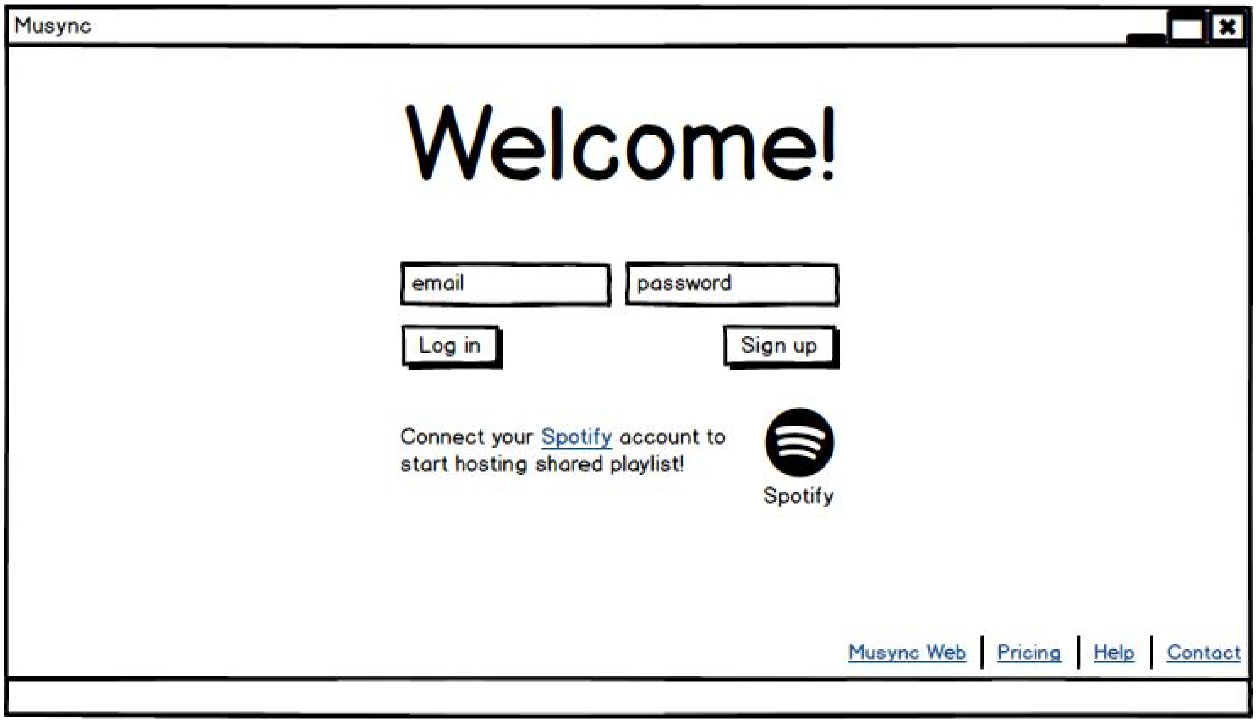
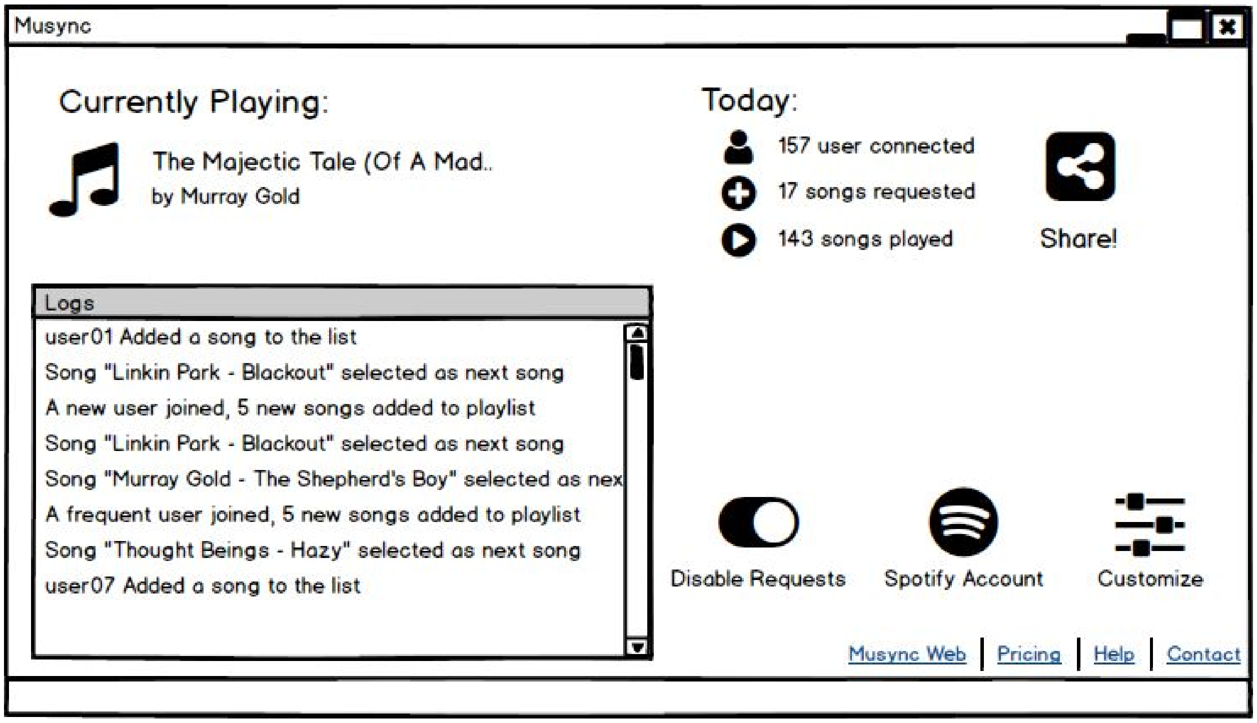


Figure 18: Desktop application login screen

Place owners will have an option to use desktop application. The desktop application will require logging in. Also users will be able to connect their Spotify account through desktop application like they could connect through web application.



```
user01 Added a song to the list
Song "Linkin Park - Blackout" selected as next song
A new user joined, 5 new songs added to playlist
Song "Linkin Park - Blackout" selected as next song
Song "Murray Gold - The Shepherd's Boy" selected as nex
A frequent user joined, 5 new songs added to playlist
Song "Thought Beings - Hazy" selected as next song
user07 Added a song to the list
```

Figure 19: Desktop application home screen

Place owners will be able control their shared playlist easily through desktop application. Since the playlist will be on Spotify, anything related to playlist itself can be done through Spotify like an individual would. Owners can see what the logs of playlist on home page of desktop application, share the numbers on their social media accounts, disable song requests temporarily if they want or customize various things like the bidding system.

3. References

[1] *Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition*, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.