Bilkent University

Department of Computer Engineering

Senior Design Project

Musync

Final Report

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

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

Final Report May 9, 2019

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project II course CS492.

Table of Content

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. It is also necessary to recommend new places that people might enjoy according to their music taste and even discover new songs.

The main purpose of this system is to let people choose what they listen according to their music taste in public places such as cafes and bars. 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 an automated digital jukebox which collects data about people's music tastes from its current users and initialises a playlist. The playlist is dynamic and changes as people come on go so that everybody can listen what they generally like. Moreover, users are able to add songs to the playlist and have a chance to choose the next song by the power of bidding. Also, users are able to see nearby locations along with their music preferences, so they can choose where they want to hang out.

This report describes the low-level architecture and the design of the Musync. Report comprises Object design trade-offs, Engineering standards, Packages and Class interfaces sections. Finally, the report is concluded with the class diagrams and the detailed explanations of software components.

1.1 Object design trade-offs

1.1.1 Usability vs. Functionality

In Musync, our first goal is usability. Firstly, we want our users to start using it without any annoying, time consuming processes like downloading an app and registration. Then, we will provide a clean and simple menu that user will know how to use it at first sight, and fast navigation to make it easy to use. Therefore we will not implement too much functionality that will lead to complex, slow menus and make the application difficult to quickly understand. Even registration is not required, since we think that aim of our app is simple

and users don’t want to spend too much time for that. That’s why we want to keep it “as simple as possible, but not simpler”.

1.1.2 Efficiency vs. Accuracy

One of our goals it to make a common playlist for each place which contains songs from different users’ playlists. This procedure will be repeated for each place throughout the day and can be very difficult to compute when there are many users. Also, the playlist will be dynamic as it will be altered for each user. For these reasons, we aim to have an efficient algorithm instead of a highly accurate one.

1.1.3 Security vs. Usability

While high level of usability is one of our goals, we needed to lose some usability to provide more security. As in real life or any other field, being more secure may require sacrificing some conveniences. An example of this is ending sessions of users after a time of inactivity. We determined this duration to be not too long to prevent both an unintended person accessing to a user’s account on a device after the actual user stopped using it and also an ill intentional user continuing to affect the music flow of place long after leaving the place. This will require users to reconnect to the place if they stayed inactive long enough for their session to expire.

1.1.4 Reliability vs. Compatibility

Musync aims to provide a robust and reliable system where users can enjoy the music experience without any system crashes. Moreover, if the system is allowed to be compatible with more than one OS then the rate of failures and maintenance would increase and therefore the system could potentially be interrupted more than necessary. The ambition of Musync is to provide the optimal user experience and satisfaction, thus a reliable system has the utmost priority.

1.2 Interface documentation guidelines

In this report; all classes, attributes and methods are named in the camel case format. Class names start with a capital letter, while others start with lowercase. Class interface descriptions are given in the following format:

class ClassName
Explanation of the class
Attributes
typeOfAttribute nameOfAttribute
Methods
returnType methodName(parameters) Method explanation if necessary

1.3 Engineering standards

We have used the UML guidelines in this report for the class interfaces, diagrams, scenarios, subsystem compositions [1]. UML is a commonly used way to generate these diagrams, easy to use and since it is the method taught at Bilkent University, we chose to utilize it in the following pages. The report follows IEEE's standards for the citations [2]. Once again, this is a commonly used method and it is the preferred one in Bilkent University.

1.4 Definitions, acronyms, and abbreviations

API: Application Programming Interface.

MVC: Model View Controller architecture.

UI: User Interface.

2. Final Architecture and Design

In this section the subsystem decomposition, subsystem services, client and server architectures are demonstrated.

2.1 Subsystem Decomposition

2.1.1 Server Side

2.1.1.1 Model

Model system is responsible for storing, accessing and managing data. Database connection and interactions with the database are handled in this subsystem. Model subsystem consists of Database Management classes, Data Cache classes and finally Object Modelling classes. Database Management classes encapsulate database interactions. They establish and control database connection and provide functionalities to store and manage data.

Data Cache classes are responsible for storing and managing database objects during the use. Frequently used database objects are cached and served from cache to decrease load on database and gain performance.

Object Modelling classes are responsible for providing an easy to use interfaces for types of objects and data structures stored in database. Retrieving, storing, providing and validating data are managed automatically with these classes. Object Modelling classes hide interactions with database and Data Cache classes from users of the Object Modelling classes to satisfy Abstraction and Encapsulation principles of Object Oriented Programming.

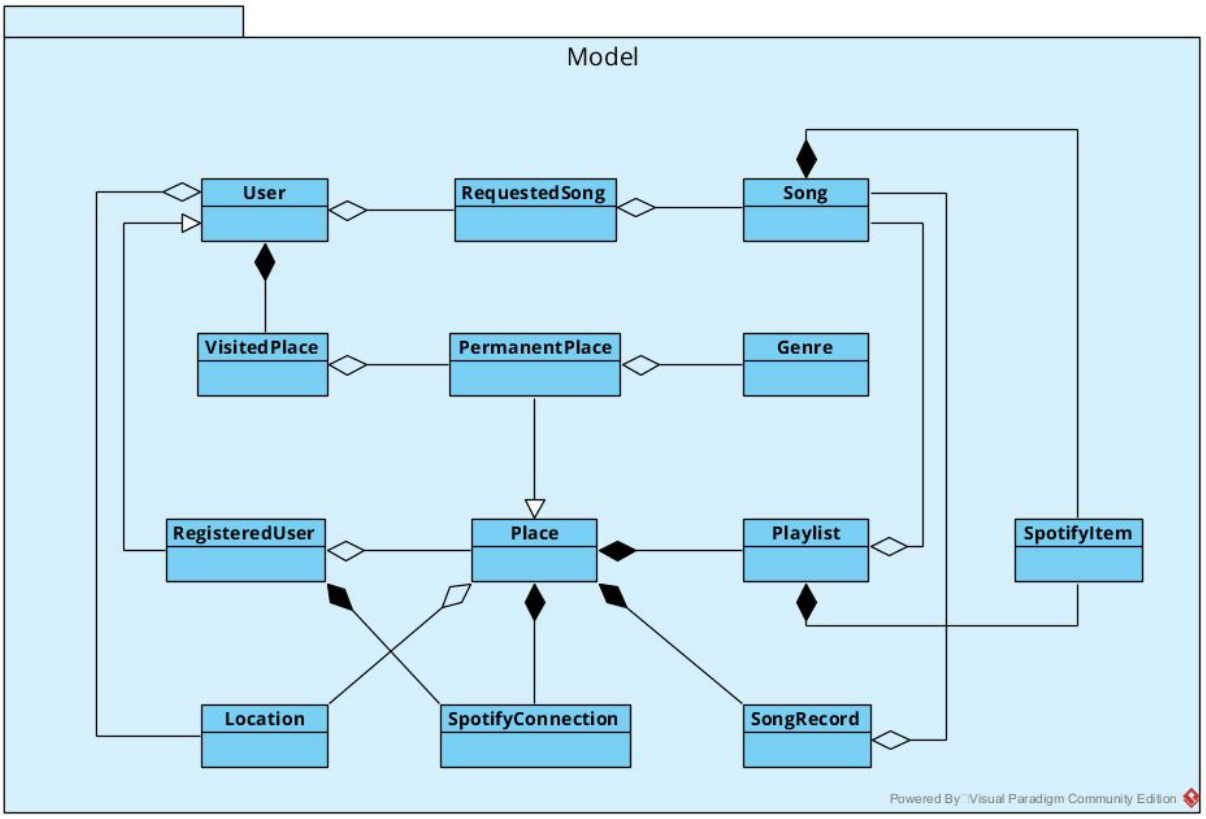

Figure 1: Model package

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: This class represents places. A place in the context of this report is a point on earth that has a shared list of songs that people can participate in controlling if they are in required proximity of the point. Musync allows both commercial place owners and individual users to create such a place and share their list. A *Place* needs a name, a location represented by

Location class and a shared playlist represented by *Playlist* class. This class also stores other data like pin to connect, votes for songs, records of previously played songs to provide functionalities Musync offers.

PermanentPlace: This class inherits the *Place* class. Owners of commercial places can create permanent places that can provide service continuingly. Also a permanent place can have a list of allowed music genres to both guide customers on what to expect and restrict the type of music requested by customers.

VisitedPlace: This class represents a record of a user visiting a place. It stores the place, the user and the date of visit to provide users with data like their previous visits.

Song: This class represents the songs as an entity in the entire system. It has the basic attributes (duration, genre, name, etc.) and the functionalities to make the system work and play music. Whenever a song is searched, added or requested to the system an instance of this class will be created.

RequestedSong: This class is basically a different representation of a *Song* instance such that it has a date, a place and user associated with it. It is the most essential entity of the system. Whenever a user requests a song, an instance of this class is initiated and assigned to a playlist. Later, users can bid on requested songs to play them as next *Song* which is *SongRecord* class.

SongRecord: This class represents songs that are played in a place. An instance of this class is initiated whenever a novel song is played in the playlist of a place. This class keeps track of the statistical information about songs so that when a user views a place they can see the genres of songs played in a place and trending songs.

SpotifyItem: This class stores data corresponding to Spotify counterparts of items like songs and playlists.

SpotifyConnection: This class stores Spotify connection related data.

Playlist: This class represents Spotify playlist of the *Place*. Requested songs will be added to the instance of this class and next songs will be selected from those songs.

Genre: This class represents genre of music and will be compatible with genres on Spotify (e.g. Metal, Rock). This class is used to determine genres of *Songs* and *Places.*

Location: This class represents physical location of *Places* and *Users.* This class will be used to determine in which *Place*, *Users* are. Also, this class will be compatible with *Google Maps* as we will be using it as main location data source.

2.1.1.2 Controller

Controller subsystem are responsible for providing a connection between Model and Frontend (View) of the project. Controller subsystem consists of router classes that handle requests made to server and utilization classes that provide additional functionalities like communicating with Spotify Web API.

Router classes handle requests made to the server. The backend server designed and developed based on REST API principles. Main logic of the backend of the project is controlled through these router classes with a request-driven approach. Router classes communicate with Model subsystem and/or Spotify to serve the data Frontend requires.

Utilization classes provide additional functionalities to other Controller classes such as handling data updates and Spotify Web API functionalities and not communicated directly from View subsystem. Controller class for Spotify handles the communication between other classes and Spotify Web API.

MainController : This class is responsible for managing the input received from the user and collect necessary information to be displayed to user. This class utilizes other controller classes in order to convey the user information about places, playlist, songs and their account. This class is the essential element of the system in order to maintain information flow between user and system.

UserController: This controller class is responsible for doing user specific operations like register and login.

PlaceController: This class is responsible for creating a new place and joining user to a place.

SpotifyController: This controller class is responsible for providing functionalities to communicate with Spotify.

LocationController: This class will be communicating with *Google Maps API* to determine locations of *Users* and *Places* and handling location related issues.

2.1.2 Client Side

2.1.2.1 View

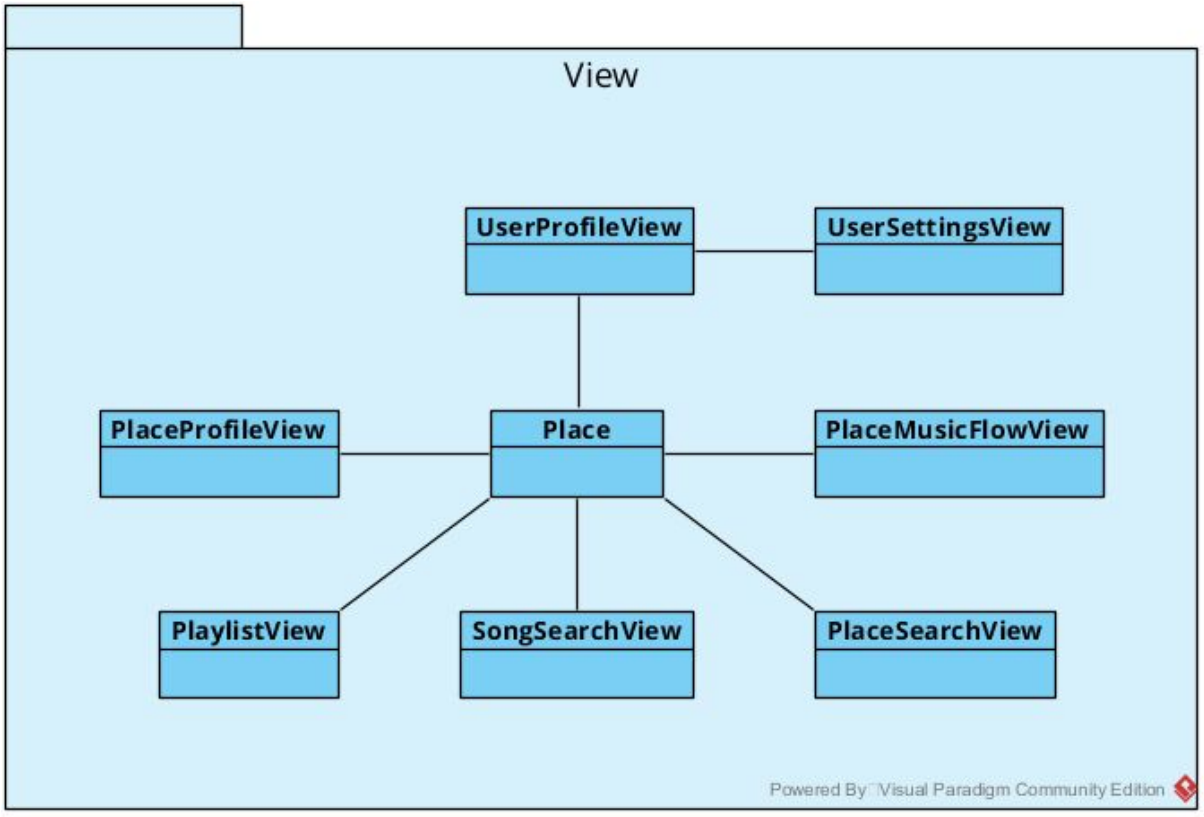

Figure 3: View package

UserProfileView: This view class will show details of the *User* and will be common for all kinds of users. However, details will be shown differently for different kinds of users. For instance, *RegisteredUsers* will be able to see and modify their email, password and Spotify information.

UserSettingsView: This view class will show details of *User* information and let them change their information such as password or Spotify account.

Place: This class will show the general information about a place such as the location so that users can see whether that place is nearby. Moreover, users will be able to see if the place has more than one branch by viewing their owner.

PlaceProfileView: This class will represent detailed information about a place so that a user can have an overall idea about the music taste of that place. Users are able to view the trending songs and general music genres of the place.

PlaceMusicFlowView: This view class will show user the music flow of the place he/she joined. It will display the currently playing song, the songs that are being voted to play next, their votes and ending time of the voting.

PlaylistView: This view class will display the current playlist of the place.

SongSearchView: This view class will be used for displaying the results of song search.

PlaceSearchView: This view class will be used for displaying the results of place search.

2.2 Class Interfaces

2.2.1 Server Side

2.2.1.1 Model

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.
Attributes
private int id
private String name
private Date lastLogin
private int points
private Location location

private List<VisitedPlace> visitedPlaces
private List<RequestedSong> requestedSongs
Methods
Getter and setter methods.

class 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.
Attributes
private SpotifyConnection spotifyConnection
private String email
private String password
private Date premiumEnd
private List<Place> places
private int premiumTier
Methods
Getter and setter methods.

class Place

Stores data related to places with shared lists.
Attributes
private int id
private String name
private SpotifyConnection spotifyConnection
private int pin
private Song[] votedSongs
private int[] votes
private Location location
private Playlist playlist
private SongRecord[] songRecords
private int initialPoint
Methods
Getter and setter methods

class PermanentPlace extends Place
Places that are permanent, like restaurants.
Attributes
private Genre[] genres
Methods
Getter and setter methods

class VisitedPlace
Stores record of a user visiting a place at a date.
Attributes
private DateTime date
private int visitCount
private PermanentPlace place
Methods
Getter and setter methods

class Song
Stores data related to songs.
Attributes
private String name
private int duration
private List<Genre> genres
private SpotifyItem spotifySong
private String songUri
private String artistName
Methods
Getter and setter methods

class RequestedSong

Stores data related to requested songs.
Attributes
private Song song
private Date date
private PermanentPlace place
Methods
Getter and setter methods

class SongRecord
Stores data related to played songs.
Attributes
private Song song
private int listenCount
Methods
Getter and setter methods

class SpotifyItem
This class stores data corresponding to Spotify counterparts of items like songs and playlists.
Attributes
private String id
private String uri

private String name
private String description
Methods
Getter and setter methods

class SpotifyConnection
This class stores Spotify connection related data.
Attributes
private String accessToken
private String refreshToken
private int expiresIn
private String userId
Methods
Getter and setter methods

class Playlist
This class represents Spotify playlist of the *Place*. Requested songs will be added to the instance of this class and next songs will be selected from those songs.
Attributes
private int id
private List<Song> songs
private int currentSong

private Date currentSongStartTime
private SpotifyItem spotifyPlaylist
private boolean isPlaying
private String snapshotId
Methods
Getter and setter methods

class Genre
This class represents genre of music and will be compatible with genres on Spotify (e.g. Metal, Rock). This class is used to determine genres of *Songs* and *Places.*
Attributes
private int id
private String name
Methods
Getter and setter methods

class Location
This class represents physical location of *Places* and *Users.* This class will be used to determine in which *Place*, *Users* are. Also, this class will be compatible with *Google Maps* as we will be using it as main location data source.
Attributes
private double latitude
private double longtitude

private String district
private String city
private String country
Methods
Getter and setter methods

2.2.1.2 Controller

class MainController
Handles requests made for Spotify API from View
Attributes
private UserController userController
private PlaceController placeController
private SpotifyController spotifyController
Methods
public List<String> searchSong() Searches song names sent with request
public String addSong() Adds song to specified playlist
public List<String> allPlaces() Returns combined information for all places

class UserController
This controller class is responsible for handling user specific requests like register and login.
Attributes

Methods
public User getUser()
public String checkConnection()
public String registerUser()
public String loginUser()
public String updateUser()
public String logoutUser()
public String connectSpotify()
public List<String> getUserHistory()
public List<String> getUserPlaylists()
public List<String> getRecommendedPlaces()

class PlaceController
This class is responsible for handling requests related to places such as creating a new place and joining user to a place.
Attributes
Methods
public String createPlace()
public String getPlace()
public String getPlaylistOfPlace()
public String changePlaceSettings()
public String connectToPlace()

public String getPlaybackInfo()
public String getVoteStatus()
public String voteForSong()

class SpotifyController
This class provides functionalities to communicate with Spotify Web API.
Attributes
Methods
public String reorderTrack()
public String addSong()
public String removeSong()
public String playSong()
public String searchSong()
public String searchArtist()
public String getLibrary()
public SpotifyConnection getSpotifyConnection()
public SpotifyConnection refreshSpotifyConnection()
public String getCurrentUser()
public List<String> getPlaylists()
public String getPlaylist()
public String createPlaylist()
public String getSong()

public List<String> getFavouriteTracks()
public String getAlbum()
public String getArtist()
public String getCurrentlyPlaying()
public String parseSpotifyResponse()

class UpdateController
This class is responsible for controlling regular updates to data.
Attributes
Methods
public int updatePlaylist(String placeId)

2.2.2 Client Side

2.2.2.1 View

class UserProfileView
This view class will show details of the *User* and will be common for all kinds of users. However, details will be shown differently for different kinds of users. For instance, *RegisteredUsers* will be able to see and modify their email, password and Spotify information.
Attributes
private int id
private DateTime premiumStartDate
private DateTime premiumEndDate
private Map<Place, Integer> visitHistory

private Map<Song, Place> requestHistory
private List<Place> recommendedPlaces
Methods
private void goToUserSettingsView()

class UserSettingsView
This view class will show details of *User* information and let them change their information such as password or Spotify account.
Attributes
private int id
private String spotifyId
private String email
private DateTime premiumStartDate
private DateTime premiumEndDate
Methods
private void changeEmail()
private void changeSpotifyAccount()
private void removeSpotifyAccount()
private void subscribeToPremium()

class PlaceMusicFlowView

This view class will show user the music flow of the place he/she joined. It will display the currently playing song, the songs that are being voted to play next, their votes and ending time of the voting.
Attributes
private String currentlyPlaying
private String[] nextSongs
private int[] nextSongVotes
private int voteEndingTime

class PlaylistView
This view class will display the current playlist of the place.
Attributes
private int playlistId
private List<Song> songs
private int currentSongNumber

class SongSearchView
Stores search term and the results of song search.
Attributes
private String searchTerm
private SpotifyItem[] results

class PlaceSearchView
Stores search term and the results of place search.
Attributes
private String searchTerm
private String[] results
private int[] resultIds

class Place
This class will show the general information about a place such as the location so that users can see whether that place is nearby. Moreover, users will be able to see if the place has more than one branch by viewing their owner.
Attributes
private int placeId
private int spotifyId
private String name
private String address
private double[] coordinates
private boolean isUserOwner

class PlaceProfileView

This class will represent detailed information about a place so that a user can have an overall idea about the music taste of that place. Users are able to view the trending songs and general music genres of the place.
Attributes
private Place place
private String[] preferredGenres
private String[] popularSongs

3. Impact of Engineering Solutions

In this section social and economic impacts of Musync is discussed.

3.1 Social Impact

Music industry has grown so dramatically that the music trends and market is changing constantly. Therefore, people are in constant search for music that they prefer. Some applications such as Shazam enables users to discover the music that is played in their environment however, such applications depends on the sound quality and if there are noise they do not find the music accurately. Musync enables users to interact with the music and increase their possibility to find new music that cater to their music interests. Users can directly view the playlist of a place from their mobile devices and make modifications such as adding a new song. Moreover, users are able to control the music they are exposed to such that they have a better music experience anywhere. As a result users are satisfied and have an amazing music experience while enjoying their favorite coffee or food in their favorite restaurants.

On the other hand, Musync facilitates the generation of playlist by creating and modifying a playlist from the user's music preferences automatically. Therefore, place owners need not to be worried whether their customers will like their playlist because it is all covered by Musync.

3.2 Economic Impact

Musync creates a new branch of industry where music is shared more openly. As a result, the producers that joined to music industry recently can potentially grow faster by introducing

their music to the users through Musync. They can even get feedback from users because if they seem to like it they would vote for their song to be played next. Therefore, Musync enables music producers to share their music easier.
Moreover, the place owners can potentially profit from Musync because their place would stand out among other places since users would have better music experience and be a loyal customer. Place owners can also sell extra vote credits to users and profit.

4. Contemporary Issues

In this section the privacy issues are discussed which is an important and delicate part of the application.

The following are the list of permission that Musync has over a spotify account.

- Your email
- Your public playlists
- Playlists you've made and playlists you follow
- What you've saved in Your Library
- The type of device you're listening on
- Your top tracks and artists
- The track you're playing
- Add and remove items in Your Library
- Create, edit, and follow playlists
- Create, edit, and follow private playlists
- Control Spotify on your devices

Moreover, Musync asks for location data of a user to be used to connect to a place. Location data consists of geographical position (longitude and latitude) of user.

Musync application does not use visitor users' Spotify connection for operations that require modification permissions. It is only used for reading data to determine a good selection of tracks for user through accessing data such as favorite tracks of the user. Furthermore, all the information gathered for this purpose is not stored in any way after use. Musync respects privacy of its users and doesn't store and share any data that is not publicly available.

For the place owners, permission to modify user data such as playlists is required. Musync does not use or store any unnecessary data. The control Musync has over the Spotify account connected to place is only over the playlist that is either created by Musync or selected by the owner. Users also are suggested to use a separate, institutional Spotify accounts for their controlled places. It should also be noted that communication with Spotify

requires secret authorization tokens and made over a secure connection, preventing access from third parties.

Location data acquired from user is only used and stored when a user wants to connect to a place. Musync doesn’t gather or use Location data for any other purpose and previous data of a user is not stored to provide privacy.

5. Tools and Technologies Used

5.1 Node.js

Node.js is a server application that supports JavaScript as a language [3]. It is widely used for applications based on REST principles.

5.2 Express.js

Express.js is a web application framework for Node.js [4]. Express.js provides ready-to-use functionality for common use cases such as handling requests made to a web server and managing user sessions.

5.3 MongoDB

MongoDB is a NoSQL database [5]. A NoSQL database is a non-relational database that allows more freedom in in the structure of data. MongoDB is a widely used database that also provides driver for connecting to database and managing data in JavaScript programming language.

5.4 React.js

React is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies [6].

React can be used as a base in the development of single-page or mobile applications, as it's optimal only for its intended use of being the quickest method to fetch rapidly changing data that needs to be recorded. However, fetching data is only the beginning of what happens on a web page, which is why complex React applications usually require the use of additional libraries for state management, routing, and interaction with an API.

5.5 Material-UI

Material design is a design language that Google created to have consistency across multiple different applications [7].
Material-UI is a library that provides components which are designed according to Material Design and can be used in React.js applications as building blocks. It also provides user interface support across multiple devices [8].

5.6 GitHub

GitHub is a web-based hosting service for version control using Git. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project. We used it for version control and to host our source code [9].

5.7 Heroku

Heroku is a cloud platform as a service (PaaS) supporting several programming languages. Heroku supports Java, Node.js, Scala, Clojure, Python, PHP, and Go. For this reason, Heroku is said to be a polyglot platform as it has features for a developer to build, run and scale applications in a similar manner across most languages. We used Heroku to deploy our application as it supports Node.js and React.js and can be deployed using only a few commands [10].

5.8 Axios

Axios is a promise based HTTP client for the browser and Node.js [11]. Axios has a wide range of browser support and can be used with ease. Additionally, it supports send custom requests and cancel prior requests on demand which was very useful for our purposes.

5.9 React-Select

React Select is a flexible and beautiful select input control for React.js with multiselect, autocomplete, async and creatable support [12]. React Select is very easy to use especially for multiple selections and has a nice and clean user interface which requires almost no customization.

5.10 React-Google-Maps

React Google Maps is a library which provides React components to wrap Google Maps JavaScript API [13]. It provides a nice looking Google Maps component while doing basic but sufficient things such as marking on the map, returning the coordinates and searching for addresses.

5.11 React-FontAwesome

React Font Awesome is a library which provides React components for widely used free Font Awesome icons [14]. React Font Awesome makes using multiple icons on different pages very easy with their icon library feature.

6. Final Status of the Project

Currently, Musync can be found at https://musync-app.herokuapp.com/. To use the application, user needs to give us the permission to reach his/her location. If permission is given, user is going to see places close to him/her. User can connect one of those places by entering its pin code which is set by owner of the place. User can search and add songs to playlist of the connected place. However, genre of the added song must be one of the preferred genres of the place. Preferred genres are set by the place owner.

In a place, next song to play is determined by voting. Each time, a place has 3 voted songs. The song with highest vote is played next.

User can register to Musync with his/her Spotify account. If user does so, some of his/her favourite songs at Spotify are automatically added to playlist of the place he/she connects. These songs also have to be from one of the preferred genres of the place. A user does not have to register for adding songs manually.

A user can create a place. To do so, he/she must connect a Spotify account. User can choose name, pin code, preferred genres and location of the place. These settings can be changed later.

User can see all places registered to Musync on a map. On this map user can see preferred genres of these places and currently playing songs there. User can also receive place recommendations based on the genres of his/her previously visited places.

References

[1] IBM, "UML - Basics," June 2003. [Online]. Available: http://www.ibm.com/developerworks/rational/library/769.html. [Accessed 08 May 2019].

[2] IEEE, "IEEE Citation Reference," September 2009. [Online]. Available: https://m.ieee.org/documents/ieeecitationref.pdf. [Accessed 08 May 2019].

[3] Foundation, Node.js. Node.js. Accessed May 09, 2019. https://nodejs.org/en/.

[4] "Node.js Web Application Framework." Express. Accessed May 09, 2019. https://expressjs.com/.

[5] "The Most Popular Database for Modern Apps." MongoDB. Accessed May 09, 2019. https://www.mongodb.com/.

[6] "React – A JavaScript Library for Building User Interfaces." – A JavaScript Library for Building User Interfaces. Accessed May 09, 2019. https://reactjs.org/.

[7] "Homepage." Material Design. Accessed May 09, 2019. https://material.io/.

[8] "The World's Most Popular React UI Framework - Material-UI." Material. Accessed May 09, 2019. https://material-ui.com/.

[9] "Build Software Better, Together." GitHub. Accessed May 09, 2019. https://github.com/about/.

[10] "'heroku' Tag Wiki." Stack Overflow. Accessed May 09, 2019. https://stackoverflow.com/tags/heroku/info.

[11] Axios. "Axios/axios." GitHub. May 07, 2019. Accessed May 09, 2019. https://github.com/axios/axios.

[12] "Select." React. Accessed May 09, 2019. https://react-select.com/home.

[13] React Google Maps Style Guide. Accessed May 09, 2019. https://tomchentw.github.io/react-google-maps/#introduction.

[14] FortAwesome. "FortAwesome/react-fontawesome." GitHub. January 15, 2019. Accessed May 09, 2019. https://github.com/FortAwesome/react-fontawesome.