Bilkent University

Department of Computer Engineering

Senior Design Project

Musync

High-Level Design 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

High-Level Design Report Dec 31, 2018

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491.

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.

1.1 Purpose of the System

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.

1.2 Design goals

1.2.1 Usability

- User interface should be simple and user friendly for both patrons and place owners.
- Users should not have any difficulty to start using the system.
- Users should be able to achieve their goals in a simple, easy and fast manner. There shouldn't be unnecessary steps.

1.2.2 Supportability

- The system should be able to work on wide range of desktop and mobile browsers.
- The system should be compatible with third party APIs such as Google Map API and Spotify API.

1.2.3 Reliability

- The system should be able to create joint playlist that will satisfy the majority and preferences of place owner regardless of the number of concurrent users.

1.2.4 Efficiency

- The delay between a user requesting a song and the song being added to an appropriate position on the list should not exceed 10 seconds.
- The server response creation time to any request should not exceed 200 milliseconds.
- The joint playlist should be created in a reasonable time and space complexity.

1.2.5 Security

- User related information should be stored securely.
- Authentication and user sessions should be handled in a secure way.
- Sensitive data like user passwords and/or credit card details should not be stored in plain text and should be encrypted securely.

1.2.6 Scalability

- The service provided to users of separate places should not be affected by the amount of all users.
- The system should be able to handle multiple requests coming from different sources simultaneously.

1.2.7 Extensibility

- New features and improvements should be able to be made to system anytime without difficulties and disrupting the service.

1.3 Definitions, acronyms, and abbreviations

API: Application Programming Interface.

MVC: Model View Controller architecture.

1.4 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. 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 using the data from their Spotify accounts, and providing an easy way to contribute to the playlist by requesting songs and bidding to requested songs.

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. Proposed software architecture

2.1 Overview

In this section proposed software architecture is presented in detail. Firstly, decomposition of 3 subsystems, which are server side for handling requests, frontend web interface for patrons and a desktop application for place owners, is shown. Then, persistent data management and database objects are described. Data control, validation and synchronization issues are discussed. Finally, initialization, termination and failure boundary conditions are described.

2.2 Subsystem decomposition

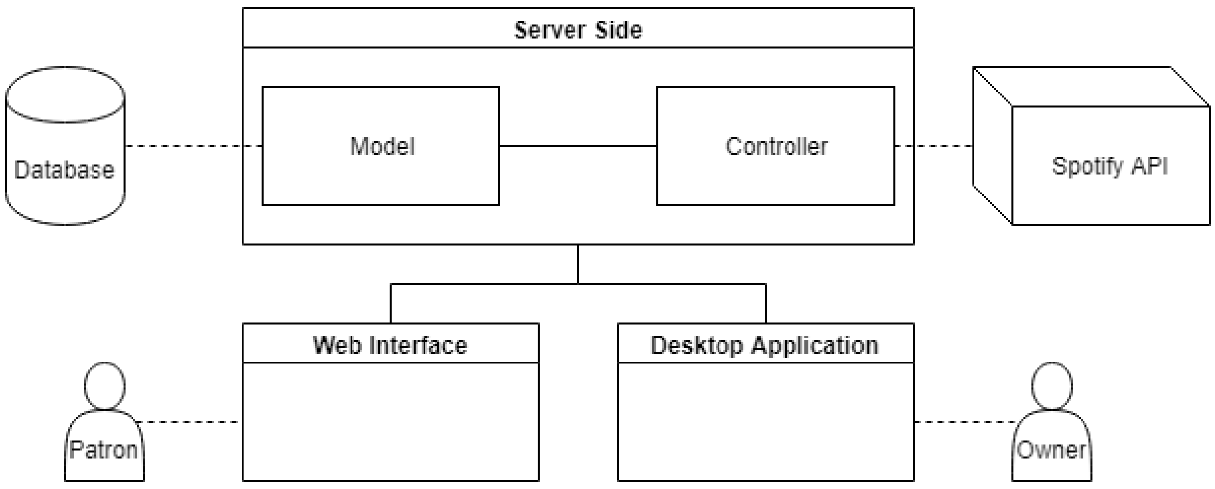


Figure 1: Subsystems overview

2.2.1 Server side

2.2.1.1 Model

Model subsystem will be responsible for storing data and provides relevant functionalities. The database connection and adding, removing and updating data will be done through Model subsystem.

2.2.1.2 Controller

Controller subsystem will control the main logic and functionalities the server side will provide. This subsystem will handle requests made to the server and will be a bridge between user interface and data. The functionalities Controller subsystem will provide includes creation of joint playlists, adding requested songs to playlists at appropriate positions and managing the bidding system and order of the songs in playlists. Also the communication between the system and Spotify API will be done through Controller subsystem.

2.2.2 Web interface

Web interface subsystem is one of the client side subsystems that users will interact with. Web interface will provide functionalities for patrons. This subsystem will consist of web pages users can open from the browser of their devices without the need of any other programs. The subsystem will communicate with Controller subsystem to provide functionalities. These functionalities will include logging in to the system, connecting to a place, making song requests and bidding on songs. In addition to the place specific functionalities, users will also be able to look for places with specific music genre preferences and see recently played songs in the places. The Web interface subsystem is the view side of the system considering the MVC architecture.

2.2.3 Desktop application

Desktop application is the other client side subsystem that will provide functionalities for place owners. It will consist of a desktop application that needs to be installed to a computer. Desktop application will provide functionalities to manage places, genre preferences of the places and other similar customizations. This subsystem will also communicate with the Controller subsystem on the Server side. This subsystem is also one of the view sides of the MVC architecture.

2.3 Persistent Data Management

Musync deals with two types of data that must be stored: User specific information and place specific information. For each user; e-mail address, password, ownership type (free trial owner/subscribed owner), owned places, points, Spotify account ID, bidding history and place visit history need to be stored in order to provide functionalities like login, contribution to shared playlist and place recommendation. For each place; name, location, owner, preferred song genre, 4-digit place specific code, recently played songs, place type (permanent or not) and connected Spotify account ID will be stored.

In order to store those, we will use a database. Database type to use is not decided yet. Currently we are thinking about factors such as data access speed to decide between a relational versus non-relational database type.

2.4 Access Control and Security

In Musync, there are different types of users and they are allowed to view and manipulate different sets of data. For example, only the owner of a place will be able to change its credentials like name, preferred genre, location etc. Only registered users will be able to create a new place. However, any visitor that is connected to a place will be able to bid on next songs and request songs. Users will not be able to access and change other users' data.

To increase the security, registration will be made with a unique, verified email address and there will be strong password requirement. Also, the passwords in the database will be encrypted.

2.5 Global software control

Musync will work on a web browser and users will not necessarily have an account. For this reason, storing and manipulating data on client side is not optimal in terms of data validation and synchronization of the data. Using MVC and client-server architectures eliminates this problem.

Changes in state which are caused by users, on client side, will be directed to the server for resolution. Controller subsystem will decide how to respond to the request according to relevant Model classes and the data which is stored in a database. If database manipulation, such as updating or deleting existing data, is needed, it will be done before sending response to the client. Then, response message will be written according to the requesting party and sent. Client side, which has view components, will show relevant information to the user according to the received response message from the server. So, the data will be validated and synchronized between users and places on the server side, and client side will only show the data provided by the server to the user.

2.6 Boundary conditions

2.6.1 Initialization

The application will require the server side to be active and running. The initialization of the server side will be done by us and only once at the very start of the service. The initialization of the server side requires setting up databases and customization of server configurations.

Patrons will need a device with an internet connection to access the web interface. To connect a place, they will also need the code provided by the place.

Place owners will need to login to the system to be able create and manage places. They will also need to connect their Spotify account. A computer connected to internet is also required to be able to use desktop application.

2.6.2 Termination

User authentication will be tracked through sessions. Sessions can end either through user logging out or the session getting timed out. If user does not log out herself, her session will get timed out after a fixed time. This time is reset every time user connects to the server through same session.

2.6.3 Failure

Internet connection loss on the user side prevents user from using the application but users will be able to reconnect and continue using until their session expires. The server side will not fail to continue providing its service to other users when a user fails to connect and use application.

Failure on the Spotify API connection will cause the service provided to a specific place to stop. This failure might be caused by user being unable to correctly connect their Spotify account or it might be a failure on the Spotify API side.

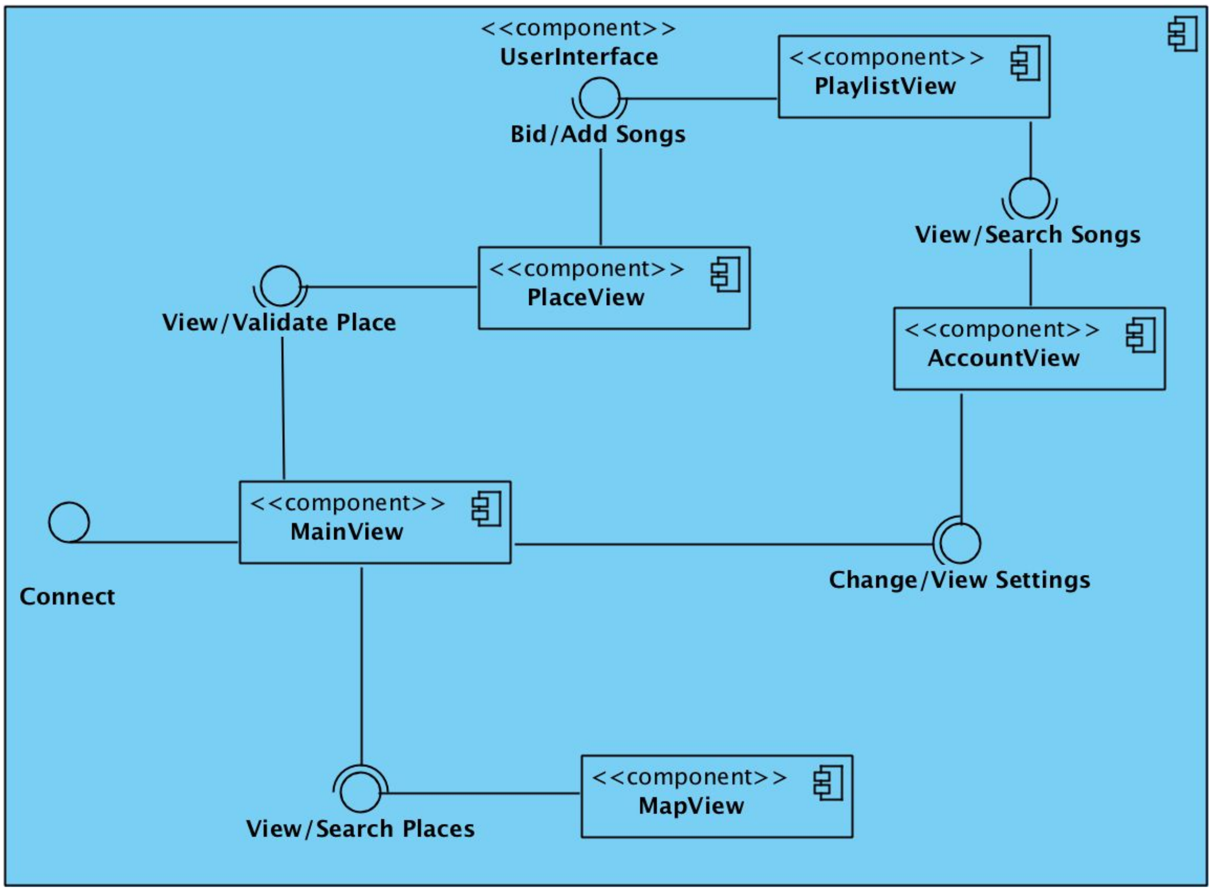
3. Subsystem services

This section describes the subsystems of Musync and the services they provide.

3.1 Client

The client corresponds to the user related functionalities of our system. It comprises Model, UserInterface subsystem.

3.1.1 UserInterface Subsystem



UserInterface Subsystem is responsible for providing user interface for Musync system.

It consists of five major components:

- MainView
- MapView

- AccountView
- PlaceView
- PlaylistView

When the user first connects to the Musync the **MainView** Component in which user can login or subscribe to our system and then interact with the system. MainView Component s the presentation level for user which they can perform main functionalities such as view settings, places, check-in to place and add/bid songs to the playlist. Moreover, if the user is a place owner they can create playlists and manage their accounts.

AccountView Component provides interface for user to change their user preferences such as their public playlists, shared songs, added songs, public profile, etc. Moreover, if the user is a place owner they can manage music preferences(list of genres and artists) of their place through this interface.

MapView Component provides interface for user to view and discover new places on map. Moreover, it enables users to view the type of music that are played around their location.

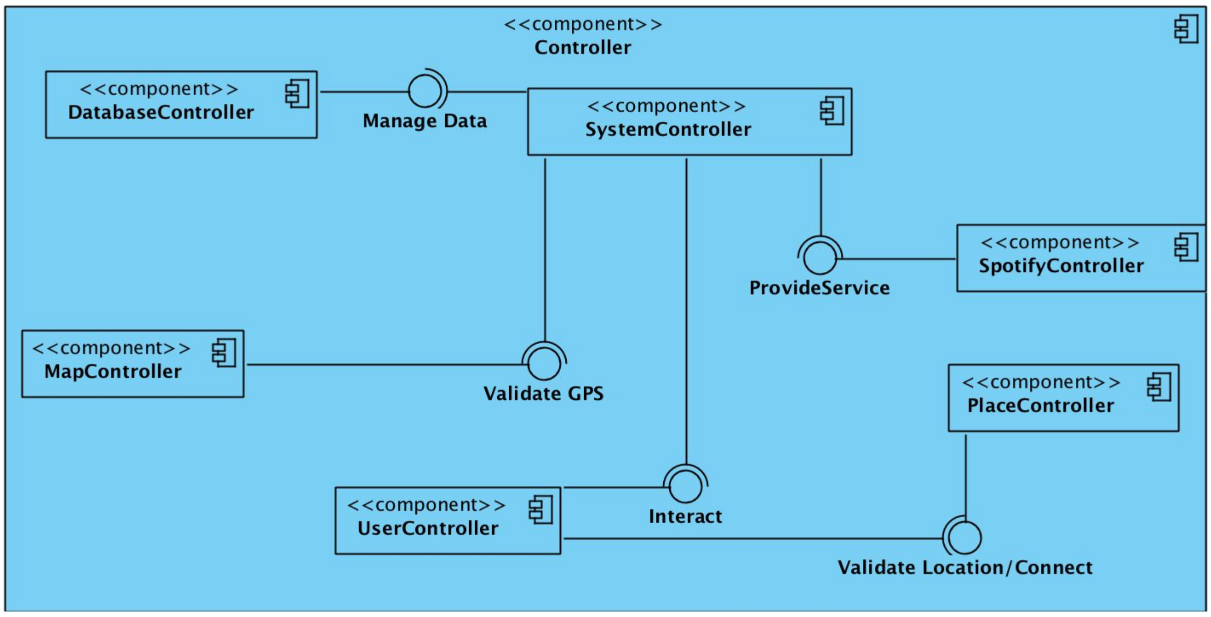
PlaceView Component provides interface for user to check-in to the place and validate their location. Then, they can see the playlist of the place and add songs to playlist. Users can bid through this interface in order to get their requested songs to be played next. Moreover, if the users are place owners they can limit the genres to be played in the playlist.

PlaylistView Component provides interface for users to search new songs and possibly add them to the playlist. They can discover new songs through this interface and add that song to their own playlists. If the user is a place owner, they can initiate an automatic playlist creation through the system. Place owners can also set limits to playlists both in terms of genres and number of songs per each genre.

3.2 Server

Server is an intrinsic part of our system. The server receives the requests from clients and reflects it to the system, thus it handles client-system communication. Server is responsible for consistent Controller of data. It comprises two subsystems Controller and Model.

3.2.1 Controller Subsystem



Controller Subsystem is responsible for controlling and handling the data and overall system.It consists of following six components:

- SystemController
- SpotifyController
- PlaceController
- UserController
- MapController
- DatabaseController

DatabaseController is responsible for controlling the data consistency throughout the system. Moreover, it keeps statistically data about user activity, playlists and genres for all places. It is an essential part of the system to be usable.

MapController is responsible for handling the location consistency and validation of the users location. Moreover, it binds the coordinates of places on the map and renders them viewable to user. It keeps track of the users location in order to avoid possible system exploitation and abuse.

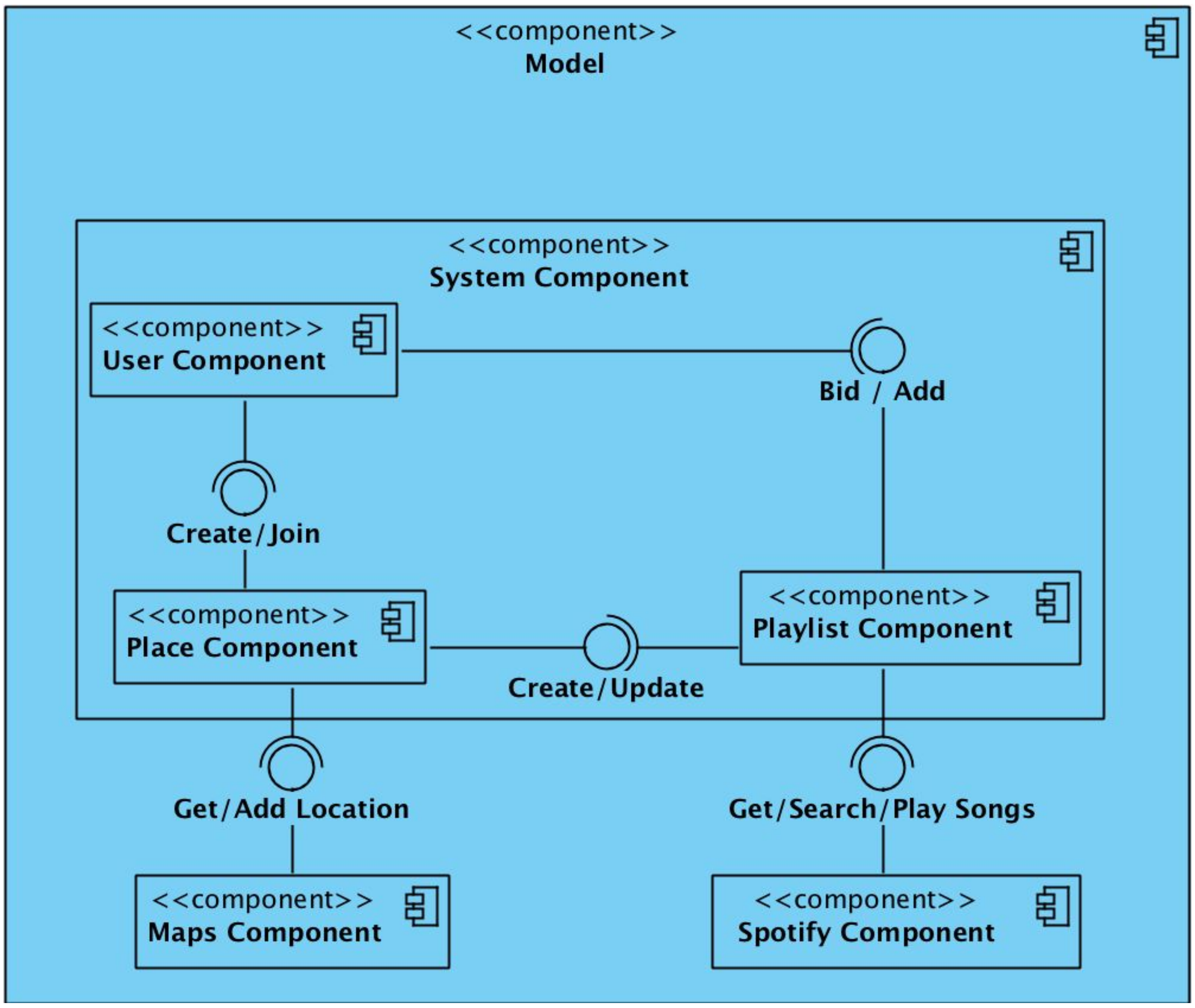
PlaceController is responsible for validating the visitors of the place by providing distinct user codes as well as checking whether their location is authentic.

SpotifyController is responsible for handling user requests and communication of Spotify API with the service. It is the most essential part of the system since playing, adding songs and creation of the playlists are performed over this system. Moreover, it collects visitors music activity in order to create the optimal playlist that is preferable by every user.

UserController is responsible for handling user activity and requests. This system is crucial for usability of the system. Users are able to perform user-related activities such as through this system.

SystemController is responsible for the consistency among all other components inside the **Controller** subsystem. Basically, it handles all the system logic to deploy and maintain the system.

3.2.2 Model Subsystem



Model Subsystem is represents our models. It consists of User, Place, Playlist Maps and Spotify components. It handles user, place and playlist updates and concurrency.

Spotify Component, enables the playlist updates, online searching, bidding and reflects it to the UserInterface subsystem. Basically, it is the dynamic data update of the music related activities coming from the Controller subsystem.

Maps Component, enables to keep track of user GPS location and updates. Therefore, it enables system to detect whether users have left the place or entered. Moreover, it handles dynamic changes in the Map view in UserInteerface subsystem. These dynamic changes may include new genres introduced to the place or new trending song emerges in the place to be reflected on the map.

Playlist Component, enables the dynamic update of user and place playlists through interacting with Spotify component and Controller subsystem. Basically, it handles and maintenance of playlist topicality.

Place Component, provides the Controller of place settings such as genres and playlist generation. Through communication with User and Playlist components, Place component provides functions like place creation/joining to a place and sets the playlist of the place.

4. References

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.