MaC ( Music and Chill )

Software Architecture Document

Version <1.1>

Revision History

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 10/12/2022 | <1.0> | Define software architecture | Group 07 |
| 26/12/2022 | <1.1> | Add new LogicalView, Replace new Components, Add Deployment and Implementation | Group 07 |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[**1. Introduction**](#_heading=h.gjdgxs) **4**

[**2. Architectural Goals and Constraints**](#_heading=h.30j0zll) **4**

[**3. Use-Case Model**](#_heading=h.1fob9te) **4**

[**4. Logical View**](#_heading=h.2et92p0) **4**

[4.1 Component: User](#_heading=h.tyjcwt) 5

[4.2 Component: Registration](#_heading=h.hrfdwfkmz8ib) 6

[4.3 Component: Interaction](#_heading=h.h1h1691i8nqe) 6

[4.4 Component: Login](#_heading=h.1qgtg9a9u0y8) 7

[4.5 Component: Post](#_heading=h.clfp55bxx2w2) 7

[4.6 Component: Admin](#_heading=h.mnk6k3xh2y3b) 7

[**5. Deployment**](#_heading=h.3dy6vkm) **8**

[**6. Implementation View**](#_heading=h.1t3h5sf) **8**

Software Architecture Document

# Introduction

**1.1 Scope**

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

**1.2 Scope**

This Software Architecture document provides an architectural overview of the Music Magazine WebSite System. Music magazine Website System is being developed by Music and Chill to support music listeners.

**1.3 Definitions, Acronyms and Abbreviations**

None

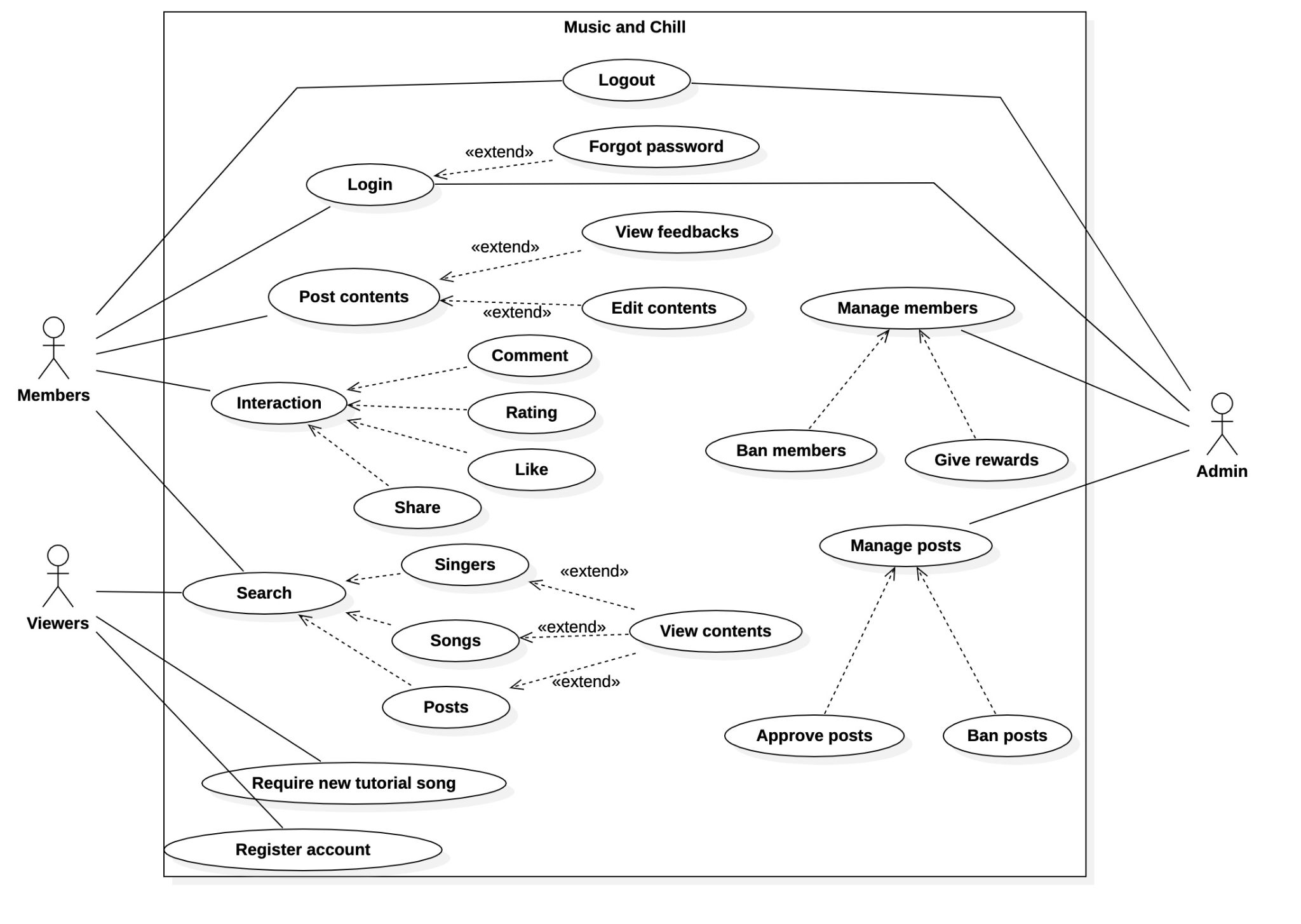
**1.4 References**

None

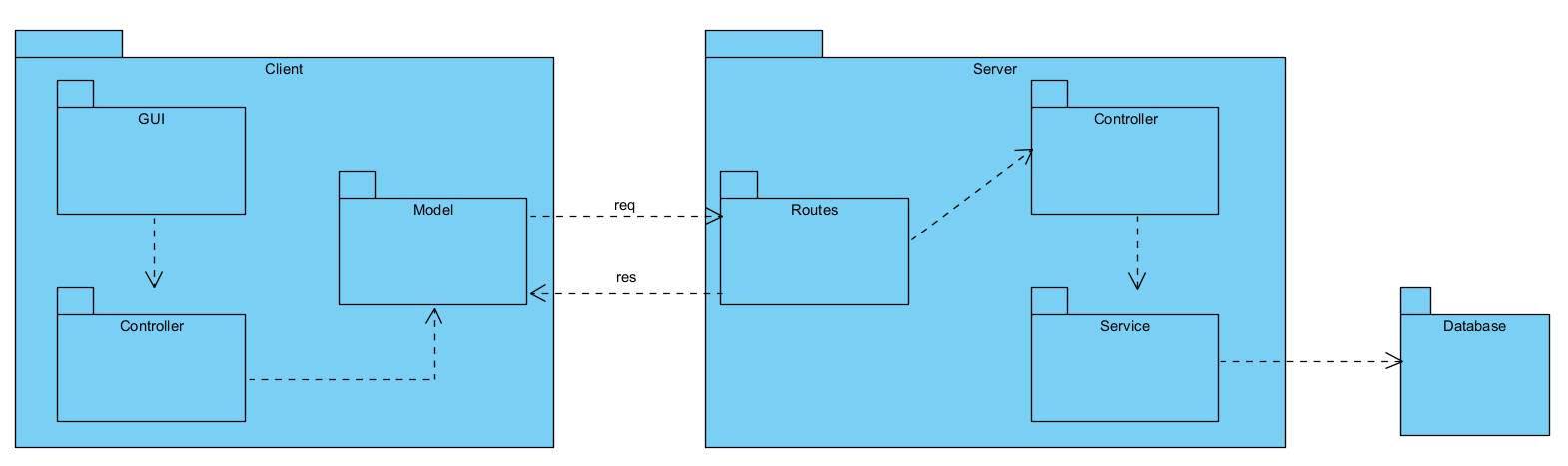
# Architectural Goals and Constraints

* Application’s environment: Web
* Programming languages and frameworks: NodeJS, ExpressJS, ReactJS, MySQL, JavaScript, HTML, CSS
* Security: password must be encrypted before storing in the database
* Valid song requests must be processed within 48 hours

# Use-Case Model

**

# Logical View

**

**- Client:** Is a website developed on NodeJS. Is the component that displays the UI for the user, receiving requests from the user.

**+ GUI:** This is where the application's UI is implemented. Components keep their own state, and between components also provide interfaces (props) to communicate with each other.

**+ Controller:** Are payloads of information that send data from your application to your store. If any state change necessary the change required will be dispatched through the actions

**- Model:** Calling API to server

**- Server:** Is a component that receives the request and handles it which is sent by the client, developed on the NodeJS environment. The server acts as a RESTful API server, providing an interface to receive requests by client, process and send responses back to the client.

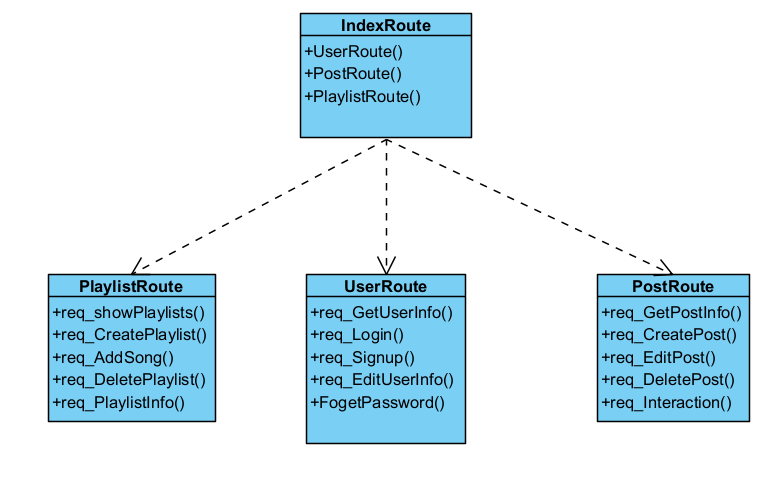
**+ Routes:** is for receive request and pass request to specification controller to process request

**+ Controllers:** validate data, transform data, then call to service to perform corresponding tasks. After that send response to client

**+ Services:** are for managing the data. It interacts directly with the database to query data and receives responses from the database and sends them back to service.

**- Database:** Using SQL database - MySQL

## Component: Index Route ( Server )

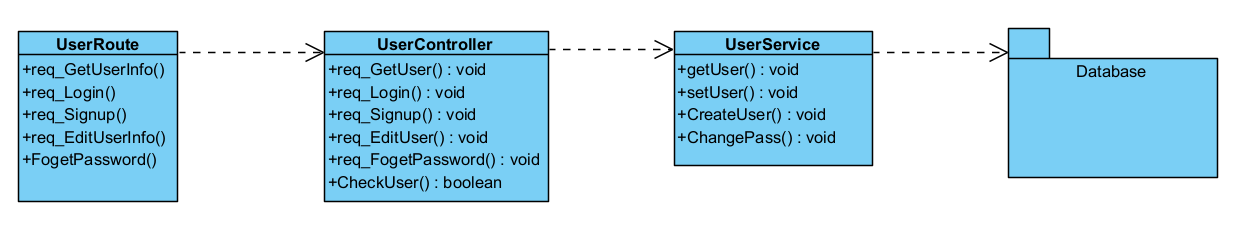
****

* Depending on the requests, the index router will send requests to the child routers that match that request
* Ex:

- “/user/…” will go to UserRouter

- “/post/…” will go to PostRouter

## Component: User ( Server )

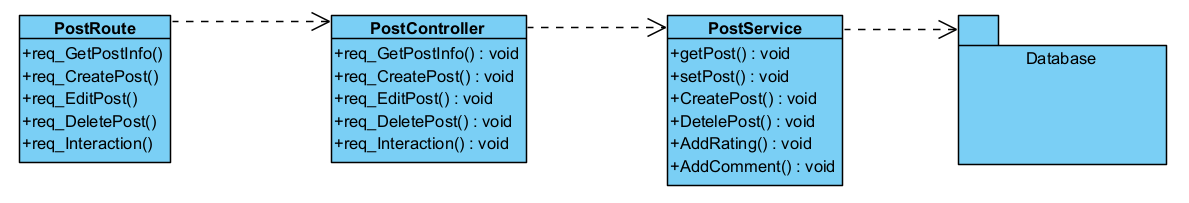


* **Details:**
* When a User login, System will call req\_Login() in UserRoute, UserRoute call req\_Login() in UserController, UserController will send username and password to CheckUser(), compare with data from getUser() in UserService and check user existence. If successful, the function returns true and sends back to UserController, else it will send an error message back

.

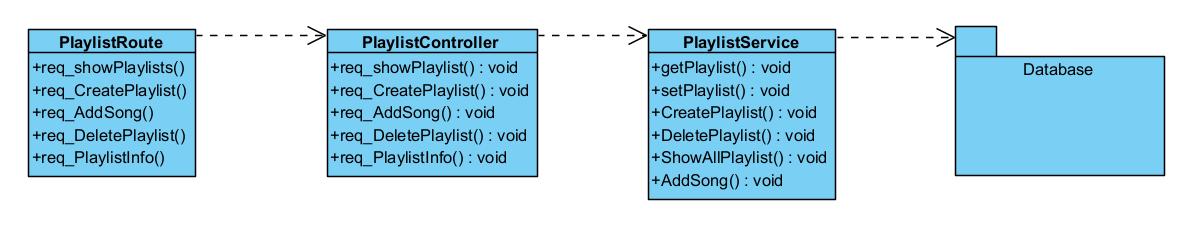
* When User choose Sign up, System will call req\_Signup() in UserRoute, UserRoute call req\_Signup() in UserController, UserController send request to CreatePass() in UserService. Before that, UserController will check in CheckUser(), if the User exists, UserController sends an error message, else continue creating User.
* When User choose to see User Info, System will call req\_GetUserInfo() in UserRoute, UserRoute call req\_GetUser() in UserController, UserController send request getUser() in UserService to get user from database and send back.
* When User forgot password, System will call req\_ForgetPassword() in UserRoute, UserRoute call req\_ForgetPassword() in UserController, UserController will confirm user info and send request to ChangePass() in UserService.
* When User want to change some information in User Info, System will call req\_EditUserInfo() in UserRoute, UserRoute call req\_EditUser() in UserController, UserController will get info from getUser() in UserService, change information and update in database through setUser() in UserService.

## Component: Post ( Server )



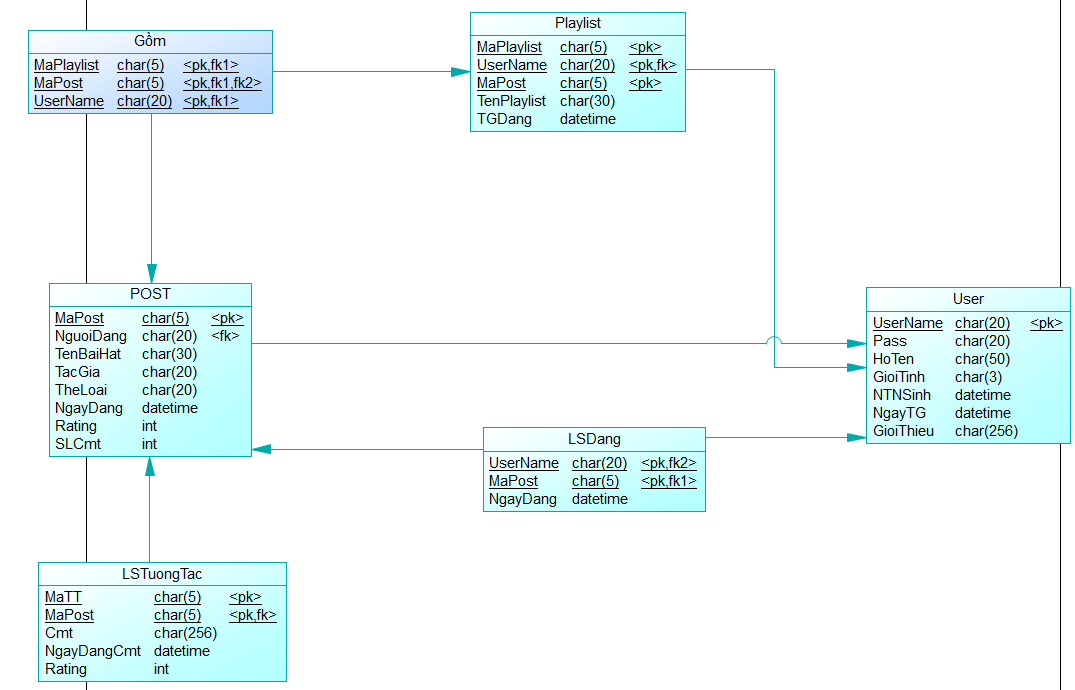
* **Detail:**
* When User creates a new Post, System will call req\_CreatePost() from PostRoute, PostRoute will call req\_CreatePost() in PostController, PostController receives information about the post and requires CreatePost() in PostService.
* When User want to see a Post, System will call req\_GetPostInfo() from PostRoute, PostRoute will call req\_GetPostInfo() in PostController, PostController require information about the post from getPost() in PostService and send back.
* When User choose edit Post, System will call req\_EditPost() from PostRoute, PostRoute will call req\_EditPost() in PostController, PostController call getPost() in PostService to get information about the Post and editing. After that, the new Post will be sent to setPost() and updated to the database.
* When the User interacts with the Post, System will call req\_Interaction() from PostRoute, PostRoute will call req\_Interaction() in PostController. If the Interaction is Rating, PostController will call AddRating() in PostService to add or update rating of User to this Post. If the Interaction is Comment, PostController will call AddComment() to add a new comment of User to this Post.
* When the User Delete Post, System will call req\_Delete() from PostRoute, PostRoute will call req\_Delete() in PostController. PostController will call DeletePost() in PostService to delete Post in Database.

## Component: Playlist ( Server )



* **Detail:**
* When the User create new Playlist, System will call req\_CreatePlaylist() from PlaylistRoute, PlaylistRoute will call req\_CreatePlaylist() in PlaylistController. PlaylistController calls CreatePlaylist() in PlaylistService to add a new Playlist in the database.
* When the User sees a Playlist, System will call req\_PlaylistInfo() from PlaylistRoute, PlaylistRoute will call req\_PlaylistInfo() in PlaylistController. PlaylistController calls getPlaylist() in PlaylistService to send back Playlist Info.
* When the User want to see a list of Playlist, System will call req\_ShowPlaylists() from PlaylistRoute, PlaylistRoute will call req\_ShowPlaylist() in PlaylistController. PlaylistController calls ShowAllPlaylist() in PlaylistService and sends all the Playlists in the database.
* When the User add a song to their Playlist, System will call req\_AddSong() from PlaylistRoute, PlaylistRoute will call req\_AddSong() in PlaylistController. PlaylistController calls AddSong() in PlaylistService to add a Song in Playlist.
* When the User delete a Playlist, System will call req\_DeletePlaylist() from PlaylistRoute, PlaylistRoute will call req\_DeletePlaylist() in PlaylistController. PlaylistController calls DeletePlaylist() in PlaylistService to delete a Playlist in the database.

## Component: Database ( Use PowerDesigner )

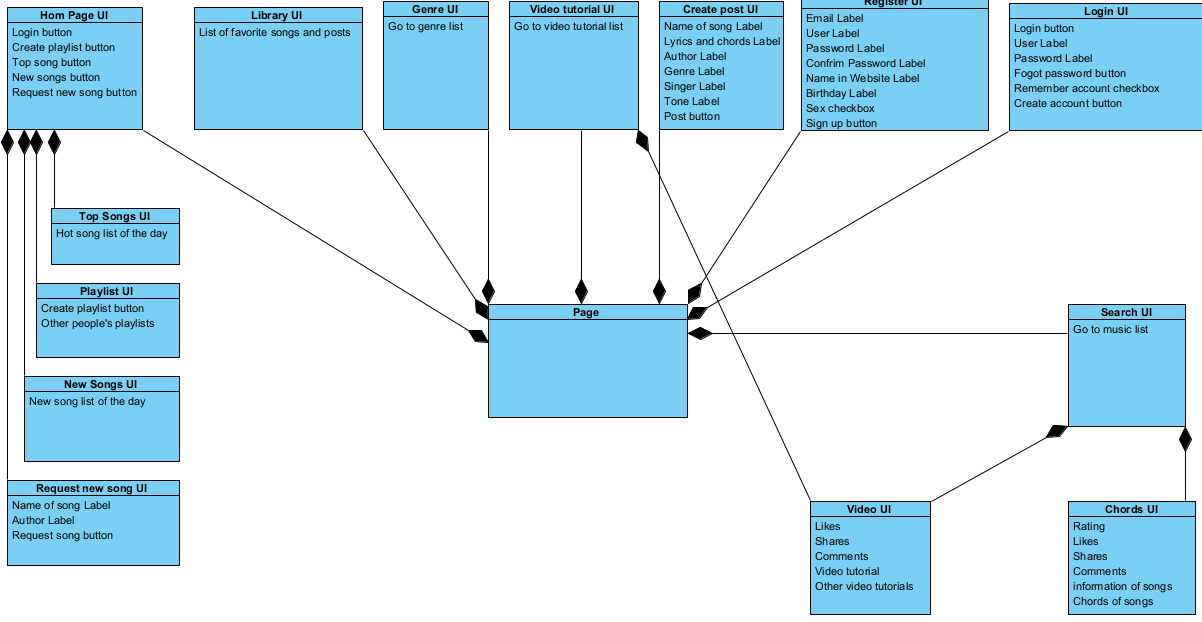


Each model class in component above is the class contains the operator to its class in database

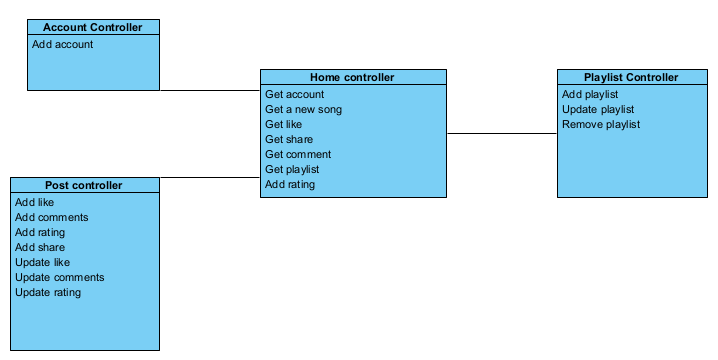
## Component: GUI ( Client )

When the user visits the interface of the website, the user will be able to choose the paths to the required task.

* Header class: Where apparently all the web pages are pointed to and the searchbar
* Page class: Is the result given after being requested (displaying post, video, song, ...)
* Footer class: Information about: contact address, website copyright information, links to social networking sites,...



## Component: Controller ( Client )

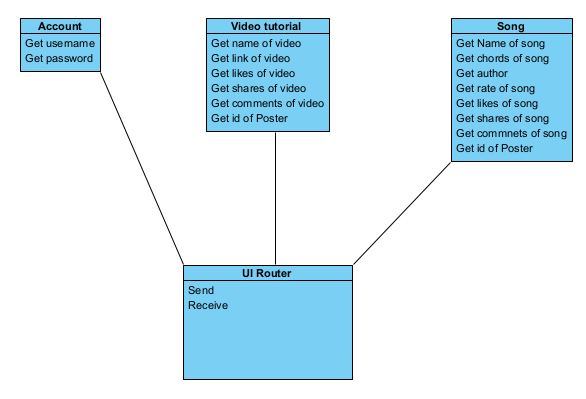


* Home controller: The source class is responsible for accessing the guest controllers when requested by the

user to navigate.

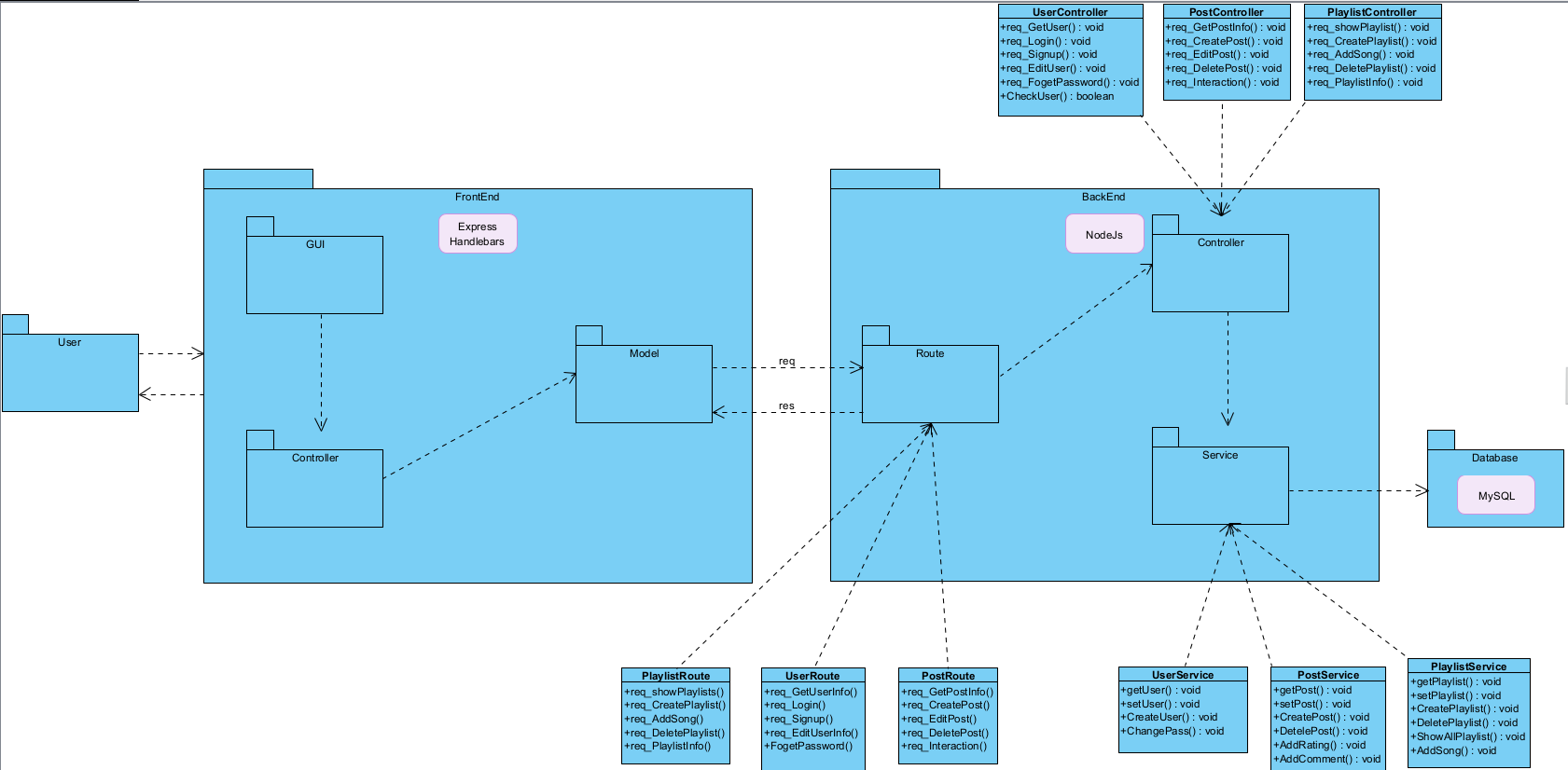
* Account controller: Users can create accounts, update their accounts.
* Post controller: Users can add and edit rating, comments in Post.
* Playlist controller: Users can add and edit playlist.

## Component: Service ( Client )



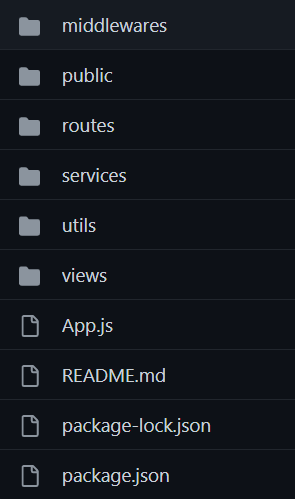
* When the user logs in, the system will send the account information and password to the server for confirmation.
* When the user logs out, the system will disconnect from the account.
* When the user registers an account, the system will send information to the server for confirmation.
* When the user views the chords of the song, the system will get the song information from the server.
* When the user watches the video tutorial, the system will get the video information from the server.

# Deployment



* Users connect to the web by Handlebars through GUI, GUI sends request to Controller in FrontEnd, Controller sends request to Model to connect to BackEnd.
* BackEnd uses NodeJs to get requests by Route and handle it by Controller, Controller manipulates data by Service.
* Service will call ( Select / Insert / Update / Delete ) by MySQL and respond to BackEnd. BackEnd response data to FrontEnd and by GUI, FrontEnd displays data to User.

# Implementation View

**

* Public is folder that contain imagine, video, … file and jquery, font awesome library to build FrontEnd
* Middlewares and Route are folder that contain screen navigation in web
* Service contains data retrieval requests in the database, Controller components in BackEnd.
* Utils contains connections to the database.
* View contains css,html, … file that build a page in FrontEnd
* App.js is a main file that start the program