SESlay

Museos
Software Architecture Document

Version 1.1

Revision History

Date	Version	Description	Author
23/11/2023	1.0	Write part 1, 2, 3, 4.0, 4.12	Lam Thanh Ngoc
23/11/2023	1.0	Write part 4.1, 4.2, 4.9, 4.10, 4.11	Nguyen Thien Tho
23/11/2023	1.0	Write part 4.3, 4.4, 4.5, 4.6, 4.7, 4.8	Tran Nguyen An Phong
05/12/2023	1.1	Write part 5, 6	Tran Nguyen An Phong

Table of Contents

Software Architecture Document

1. Introduction

This document has a purpose of providing a detailed architecture design of the Museos System by focusing on the model of the program and all its packages. These attributes were chosen based on their importance in the design and construction of the web application.

The scope of incorporating Museos Web elements within the realm of software architecture encompasses a comprehensive exploration of the design, integration, benefits, challenges, and potential applications of this innovative approach. This scope will guide the development and understanding of Museos Web within software systems.

Term list:

- Stakeholder: any person involved or affected, directly or indirectly, by this product.
- Javascript: web-browser interpreted programming language for enhancing websites in a dynamic way.
- Express.js: is a back-end web application framework for building RESTful APIs with Node.js
- Node.js: is a JavaScript runtime built on top of Chrome's V8 JavaScript engine

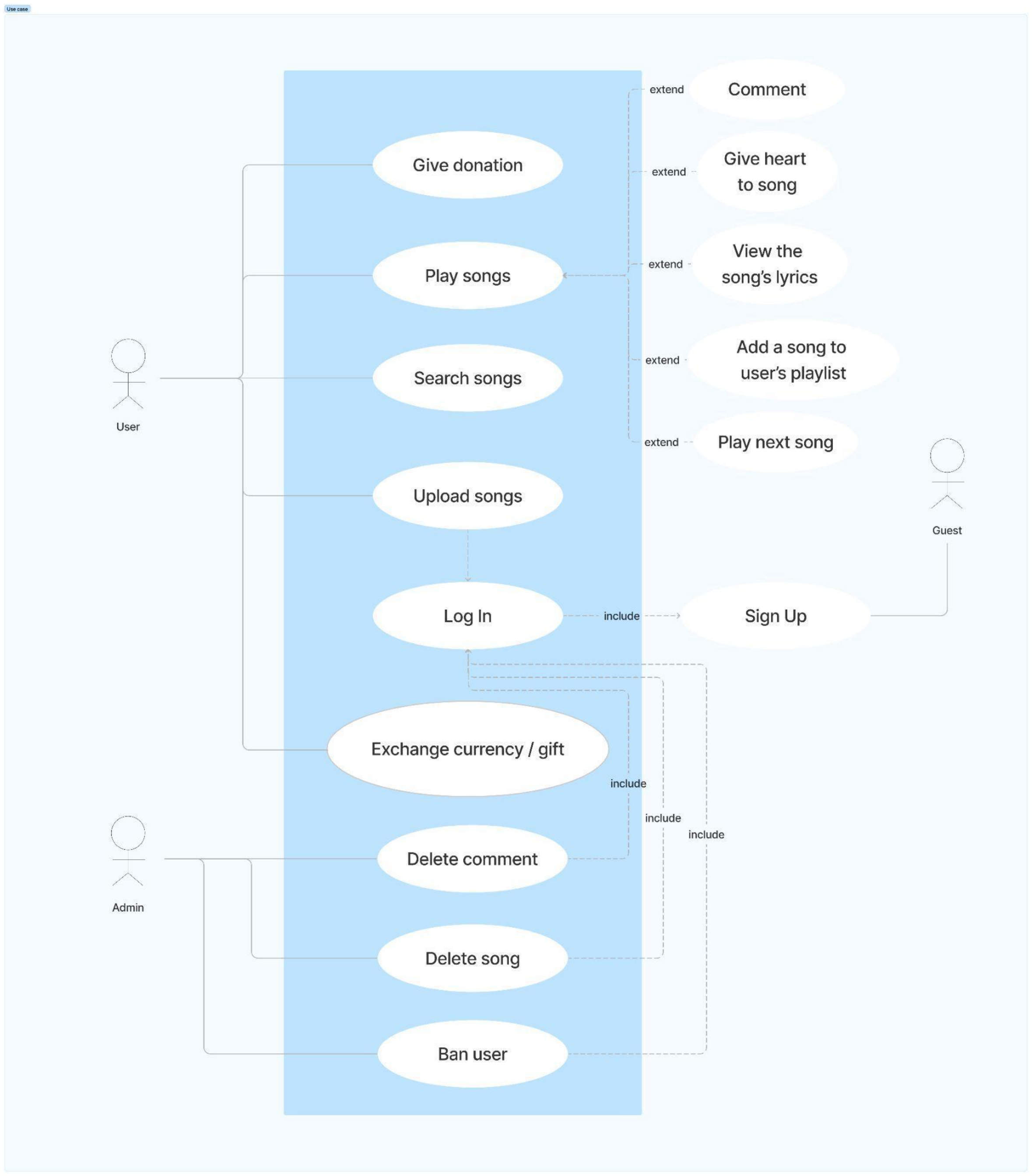
Acronym list:

- UI: User Interface
- API: Application Programming Interface

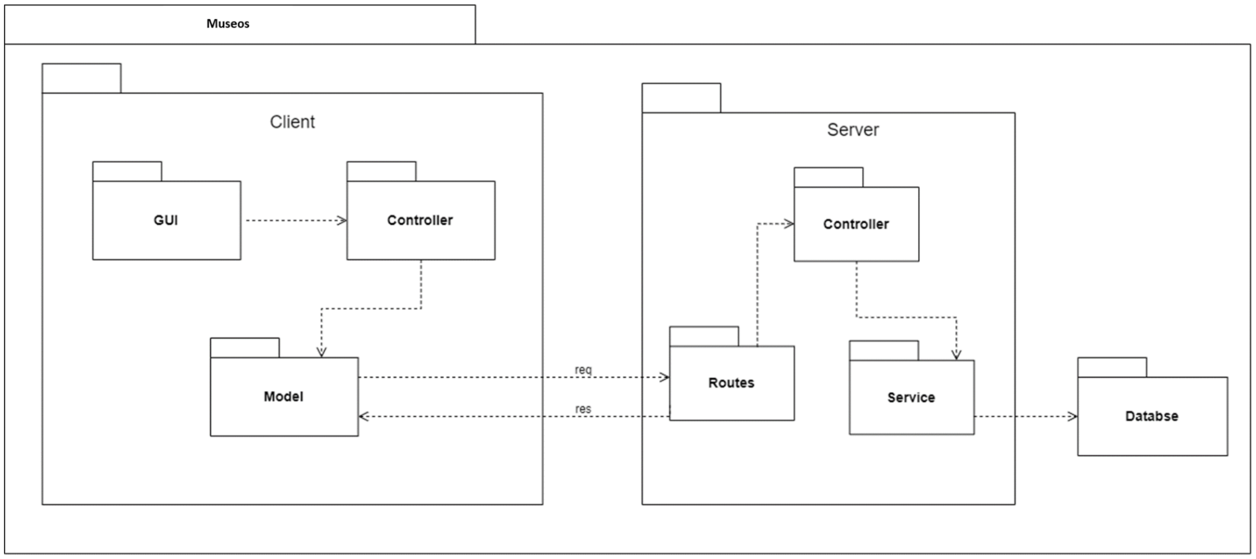
2. Architectural Goals and Constraints

- Application environment: Web.
- Programming language and platform: Javascript (Node.js, Express.js), HTML, CSS.
- Database environment: MongoDB (this is NoSQL Database, for large scale and real-time applications).
- Security: password must be encrypted before storing in the database.
- Copyright requirement: Artists must make sure that the uploaded songs are their own product.
- Responsive requirement: Laptop, PC, smartphone, tablet…

3. Use-Case Model

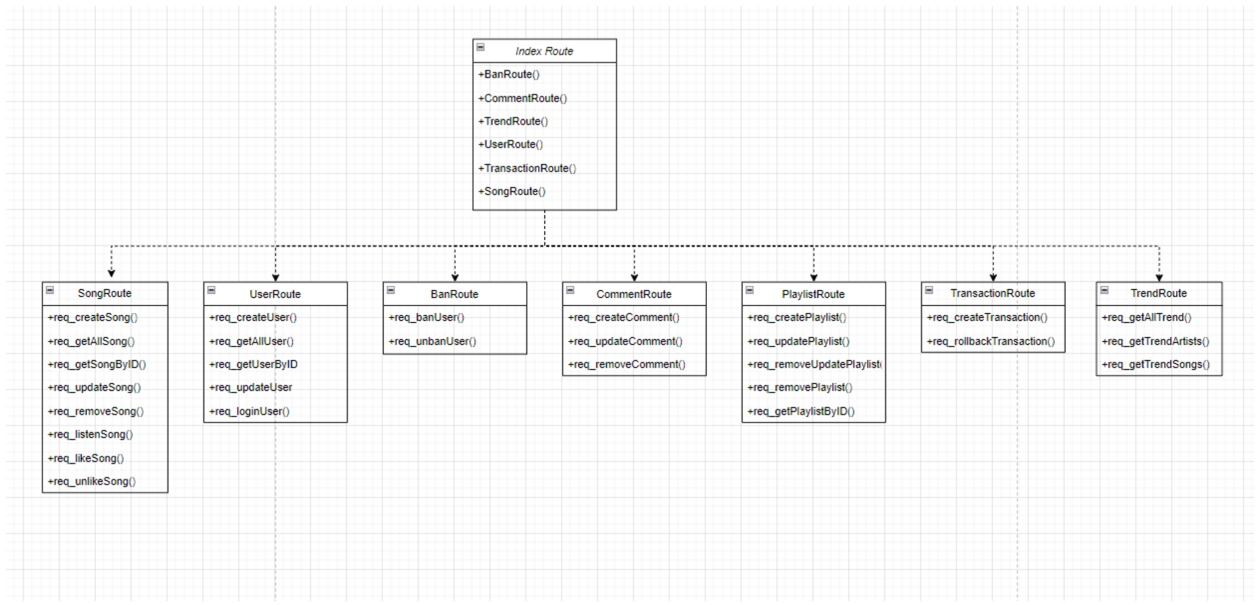


4. Logical View



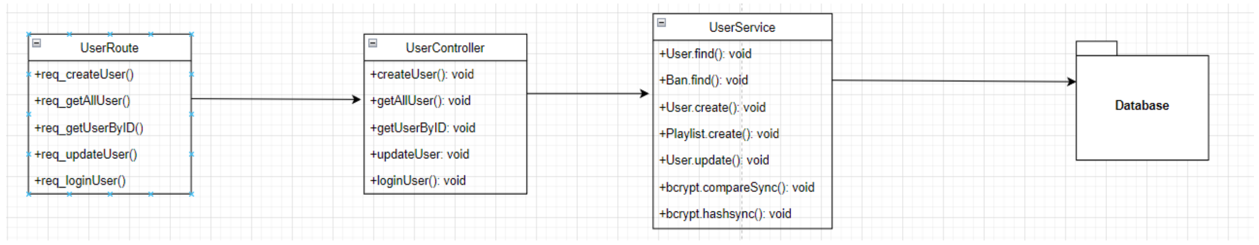
- **Client:** is a website developed in HTML/CSS and JS, the component which displays the UI for the user, receiving requests from the user then pushing requests to server handle.
  + **Service:** User services which are displayed by UI, providing any song service for users.
  + **GUI:** This displays the services in UI format which are easy for users to interact with the website.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 is necessary the change required will be dispatched through the actions.
  + **Model:** Calling API to server.
- **Server:** The component that receives the request and handles it which is sent by the client, developed on the Express.js 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:** The component that receives the request from users then passes it to the controller in the server handle.
  + **Controller:** The component that validates data gets from request then handles it then calls to service to perform corresponding tasks, and send response for the client in final.
  + **Service:** are for managing the data, the component which interacts directly with the database to query data and receives responses from the database and sends them back to service.
- **Database:** Using MongoDB

4.1 Component: Index Route (Server)



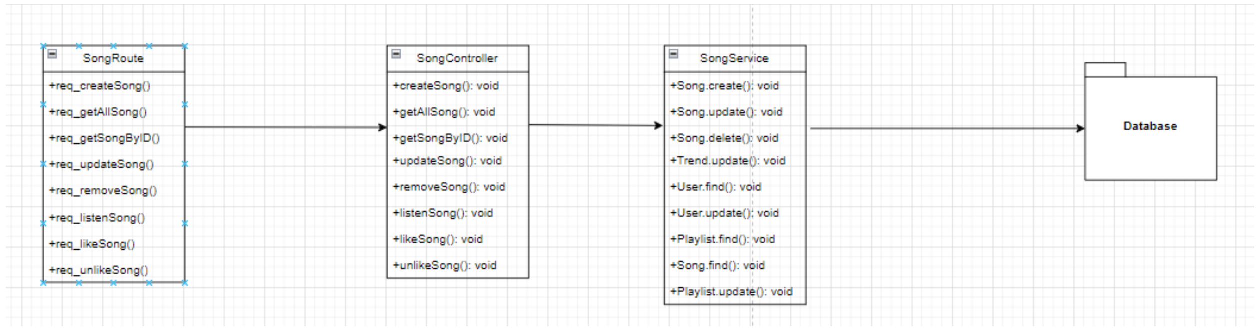
- Depending on the requests, the index router will send requests to the child routers that match that request
- Ex:
  - "/song/…" will go to SongRouter
  - "/user/…" will go to UserRouter

4.2 Component: UserRoute (Server):



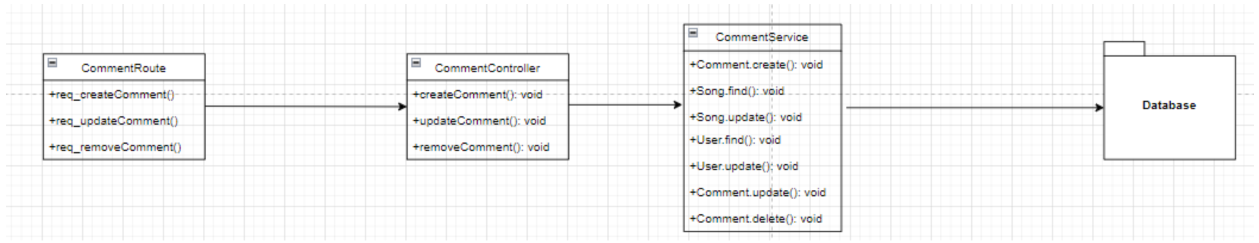
- Details:
  - Most of the schema functions like "User.find()" are MongoDB functions. These schema functions will work directly with our MongoDB database.
  - When req_createUser() is received in the UserRoute, the system will call the createUser() in the Controller. In createUser() function the user password will be hash by bcrypt.hashSync() and the MongoDB User.create() function will be executed. At the same time a personal playlist will be created by Playlist.create() and given to that user through User.update().
  - When req_getAllUser() is received in the UserRoute, the system will call the getAllUser() function in the UserController. In getAllUser() function, all users will be received by executing User.find(). It works the same for req_getUserByID.
  - When req_updateUser() is received in the UserRoute, the system will call updateUser() function in the Controller. In updateUser() function the user in the request will be found by User.find() and when a user is found, that user will be updated by User.update().
  - When req_loginUser() is received in the UserRoute, the system will call loginUser() function in the Controller. In loginUser() function the user data will be found by User.find and then compare the password from the request to the password in the database by bcrypt.compareSync().

4.3 Component: SongRoute (Server)



- Details:
  - When the req_createSong() is received in the SongRoute, the system will call the createSong() function in the Controller. In this function, the Song.create() function will be executed to create a new song based on the information from the request and some work will be done to upload the song to our database then we use Song.update() to update the song file link on the database. Then we push the song to the user's song list through User.find() – User.update(). And we also update the Trend when there is a new song uploaded by Trend.update().
  - When the req_getAllSong() is received in the SongRoute, the system will call the getAllSong() function in the Controller. This function will return a list of all songs by using Song.find(). The system works the same way with req.getSongByID() -> getSongByID(). But instead of the song list, the system will return 1 song if there is any song that matches the ID in the request.
  - When the req_updateSong() is received in the SongRoute, the system will call the updateSong() function in the Controller. This function will update the song information like lyrics, cover,…. Based on the information from the request through Song.update() function.
  - When the req_removeSong() is received in the SongRoute, the system will call the removeSong() function in the Controller. This function will remove a specific song from the database through Song.delete().
  - When the req_listenSong() is received in the SongRoute, the system will call the listenSong() function in the Controller. In this function, first the song listen counter will be updated by Song.update(). Then the song listen counter inside the trend will also be updated by Trend.update().
  - When the req_likeSong() is received in the SongRoute, the system will call the likesSong() function in the Controller. In this function, first the song heart counter will be update by Song.update(), then the song will be also added to "Liked playlist" of the user who like the song by the chain of functions User.find() – Playlist.find() – Playlist.update(). It works the opposite with the req_unlikeSong().

4.4 Component: CommentRoute (Server)

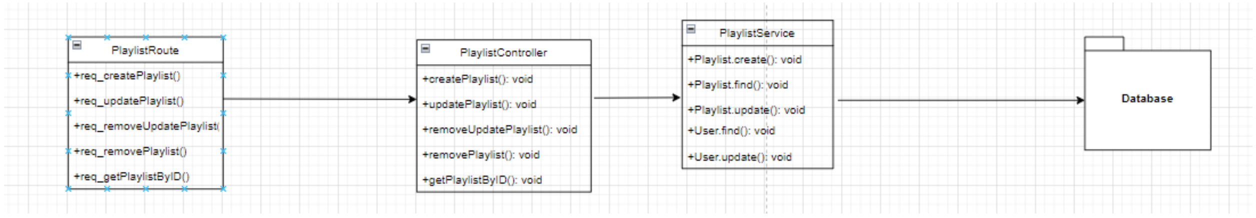


- Details:
  - When the req_createComment() is received in the commentRoute, the system will call the createComment () function in the Controller. In this function, the Comment.create() will be trigger to create a new comment. After that we will update the comment section of a song through

Song.find() – Song.update() and update the comment list of a user through User.find() and User.update().

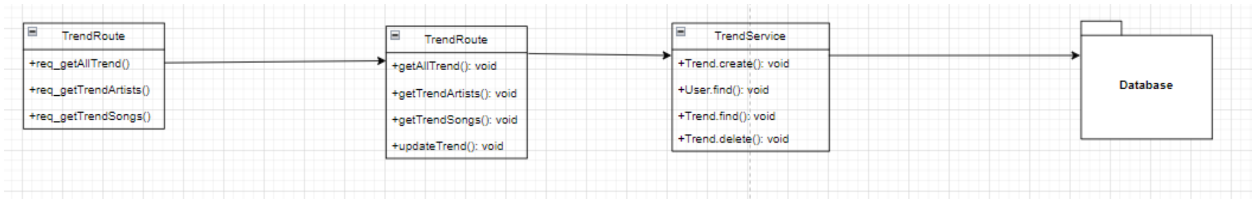
- ○ When the req_updateComment() is received in the commentRoute, the system will call the updateComment() function in the Controller. In this function, the requested comment will be updated by the Comment.update() function.
- ○ When the req_removeComment() is received in the commentRoute, the system will call the removeComment () function in the Controller. In this function, the requested comment will be deleted by the Comment.delete() function.

4.5 Component: PlaylistRoute (Server)



- Details:
  - ○ When the req_createPlaylist is received in the PlaylistRoute, the system will call the createPlaylist() function in the Controller. This function will create a new playlist by Playlist.create() and put that playlist to the user's playlist by the combo of User.find() and User.update().
  - ○ When the req_updatePlaylist is received in the PlaylistRoute, the system will call the updatePlaylist() function in the Controller. In this function, a playlist will be updated by calling the Playlist.update() function.
  - ○ When the req_removeUpdatePlaylist is received in the PlaylistRoute, the system will call the removeUpdatePlaylist() function in the Controller. This function will update the playlist by removing a song from the playlist. This will also be executed by Playlist.update() .
  - ○ When the req_removePlaylist is received in the PlaylistRoute, the system will call the removePlaylist() function in the Controller. In this function, a playlist will be deleted from the database by Playlist.delete().
  - ○ When the req_getPlaylistByID is received in the PlaylistRoute, the system will call the removeUpdatePlaylist() function in the Controller. This function will return the result of finding a playlist by ID by Playlist.find() function.

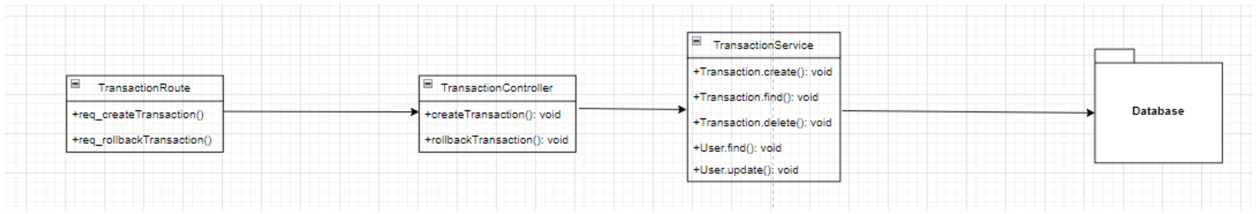
4.6 Component: TrendRoute (Server)



- Details:
  - ○ Before routing to any trend request, the TrendRoute will trigger the updateTrend() in the Controller to normalize the trend data first. In this function, a trend will be created if it is not showing up on the database yet by the function Trend.create(). If there is already a trend, the function will check if that trend is correct to the current date or not. If not, the trend will be deleted by Trend.delete() and create a new trend by Trend.create().
  - ○ When the req_getAllTrend() is requested in the TrendRoute, the system will call the getAllTrend() function in the Controller. This function will return both artists trend and song trends through Trend.find() function.
  - ○ The req_getTrendArtists() and req_getTrendSongs work with the almost same flow like req_getAllTrend but it will return the specific trend artists or trend song.

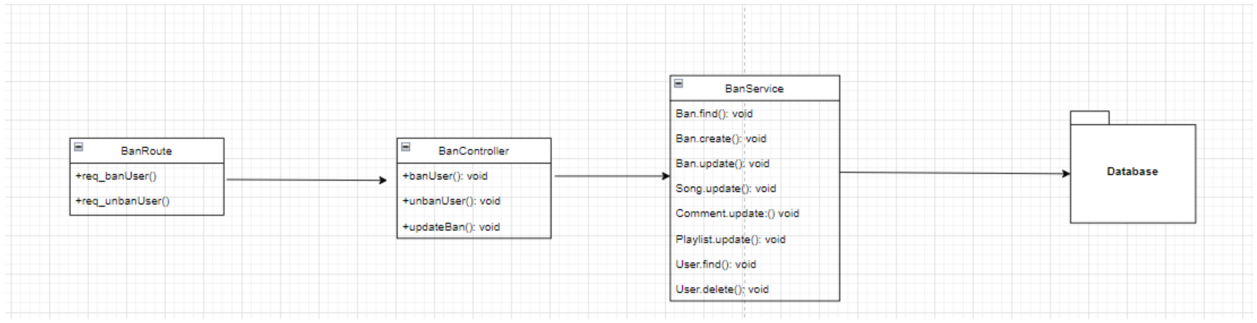
- When the req_getAllTrend() is requested in the TrendRoute, the system will call the getAllTrend() function in the Controller

4.7 Component: TransactionRoute (Server)


TransactionRoute
+req_createTransaction()
+req_rollbackTransaction()
TransactionController
+createTransaction(): void
+rollbackTransaction(): void
TransactionService
+Transaction.create(): void
+Transaction.find(): void
+Transaction.delete(): void
+User.find(): void
+User.update(): void
Database

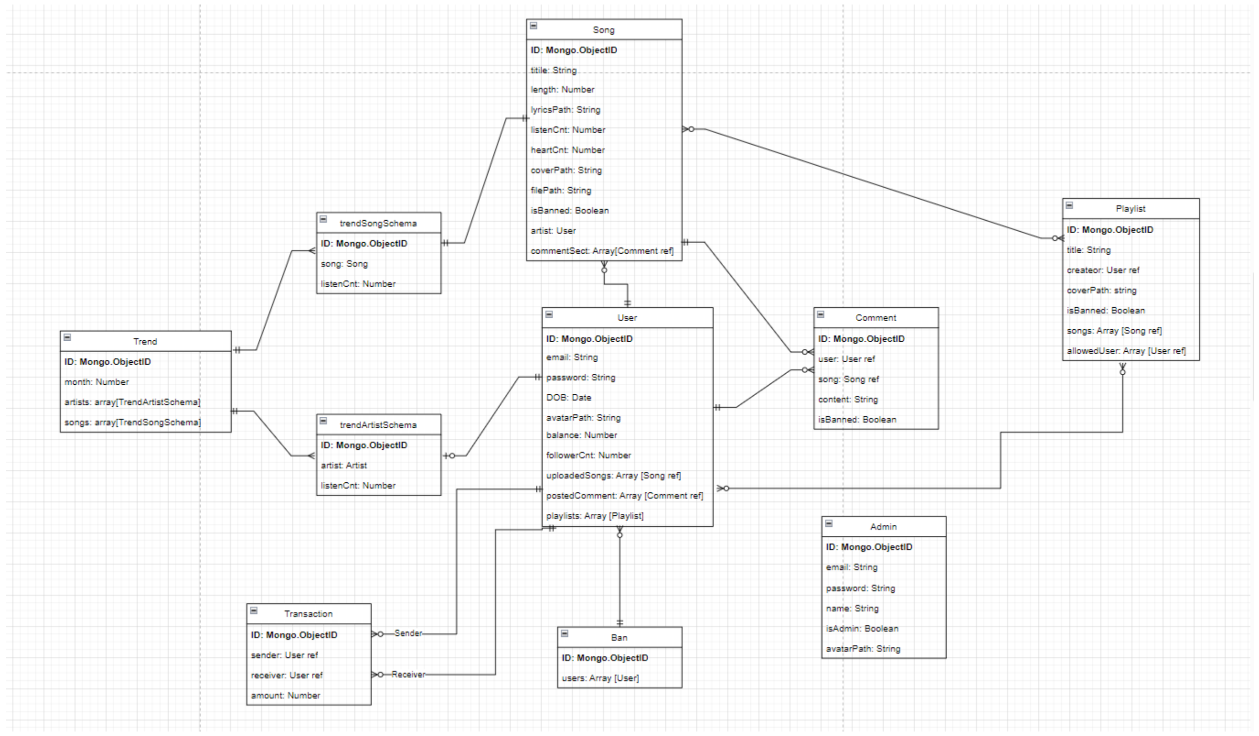
- Details:
  - When the req_createTransaction is requested on the TransactionRoute, the system will call the createTransaction function in the Controller. This function will create a new transaction with Transaction.create() and modify the balance of the User corresponding to the transaction value. The modification should be done with a User.find() and User.update().
  - When the req_rollbackTransaction is requested on the TransactionRoute, the system will call the rollbackTransaction function in the Controller. This function will first find the needed transaction by Transaction.find() and then rollback all the values by doing the same User.find() – User.update(). Finally, the system will call the function Transaction.delete() to delete the transaction from the database.

4.8 Component: BanRoute (Server)



- Details:
  - Before routing any request. The system will call the updateBan() function in the Controller to normalize the Ban table. First it will check if there is any Ban instance yet by using Ban.find(). If no Ban instance exists yet, the system will create a Ban instance by Ban.create().
  - When the req_banUser is requested on the BanRoute, the system will call the banUser function in the Controller. In this function, a user will be ban by putting by first retrieving all the user related data by User.find() and then after having the user-related song, comment, playlist data the system will update the ban status by a chain of Song.update() – Comment.update() and Playlist.update() .After that the system will put user in the ban table instead of the User table by Ban.update() and User.delete().
  - When the req_unbanUser is requested on the BanRoute, the system will call the unbanUser function in the Controller. The unbanUser() works exactly the same way as banUser() but with the opposite value from ban to unban.

4.9 Component: Database



Each class stores information in the database.

Class Song: Store information about a song, including the artist who uploaded the song and the comment section of that song.

Class User: Store information about a user, including the songs and comments that user has uploaded and that user's playlist. This class have:

Class Playlist: Store information about a playlist, including songs that belong to that playlist and users who are allowed to access the playlist.

Class Comment: Store information about a comment.

Class Ban: Store information about a list of banned users.

Class trendArtistSchema and trendSongSchema: Store information about an artist and a song listen counter.

Class Trend: Store information about overall platform trend, including artist trend and song trend.

Class Transaction: Store information about a donation transaction between two users.

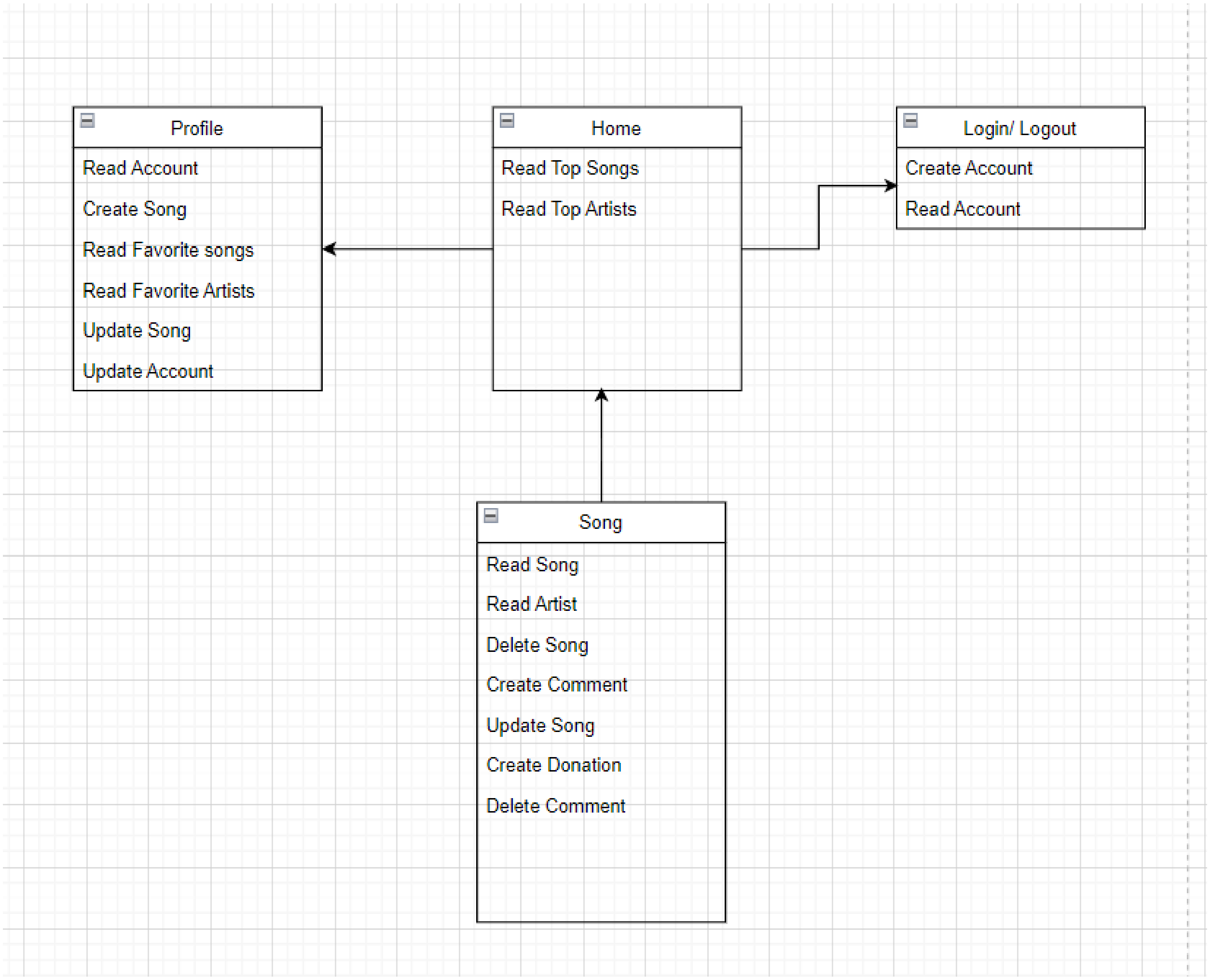
Class Admin: Store information about an admin.

Relational between class:

- One to many optional relationship between class Song and User as one user may upload none or many songs and a song must be uploaded by a user.
- One optional to one relationship between class trendArtistSchema and User as a user can be an artist and through can be in a trend and a trendArtist must be about an artist.
- One to many optional relationship between class Comment and User as a user may upload none or many comments.

- One to many optional relationship between class Transaction and User as a user may participate in none or many transactions in the form of a sender or a receiver and a transaction must have sender or receiver.
- Many optional to one relationship between class Ban and User as many or no users can be banned.
- Many optional to many optional relationship between class Playlist and Song as many songs can be in a playlist and a playlist may have many songs.
- One to one relationship between class Song and trendSongSchema as a song will be in a trend.
- One to many optional relationship between class Song and Comment as a song may have many comments and a comment must belong to a song.
- One to many between class Trend and trendArtistSchema/trendSongSchema as a trend is about many artists and songs.

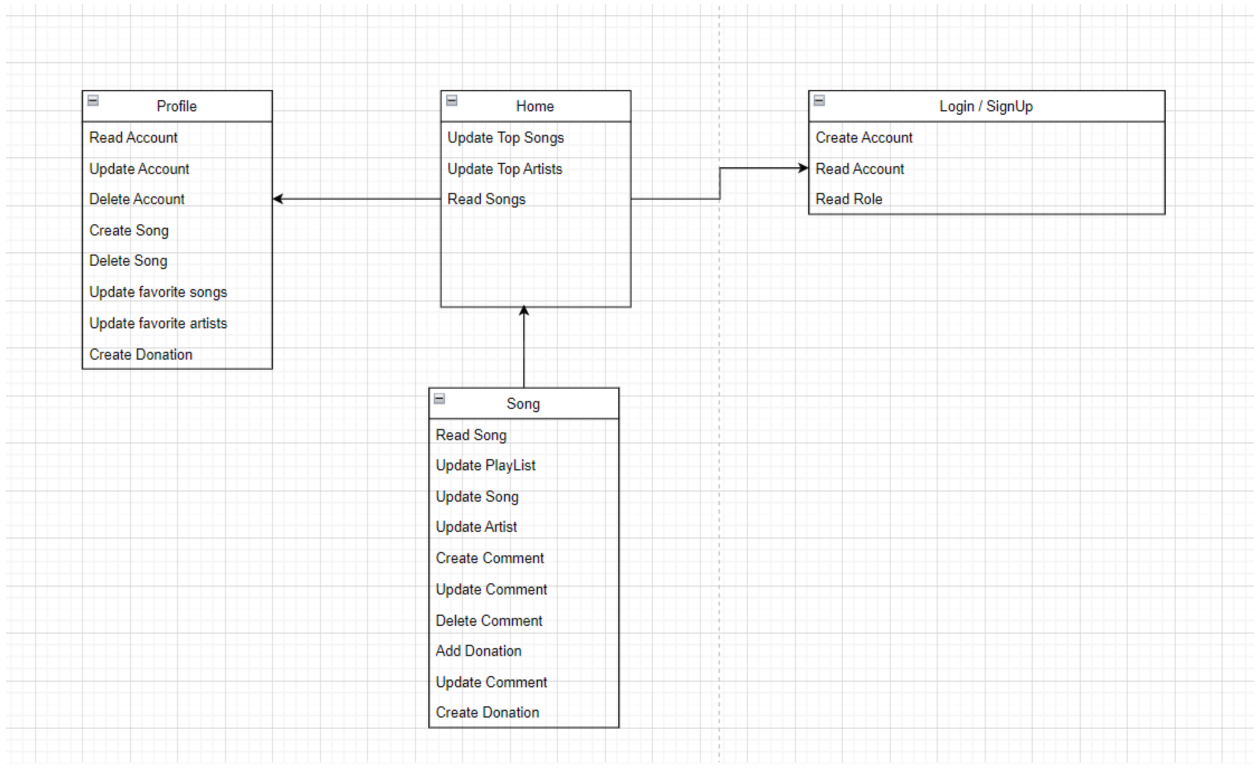
4.10 Component: Controller (Client)



- Home: It would read songs to get information showing top songs and artists.
- Profile: Manages user account details and personal content. It allows for the reading of account details and personal lists such as favorite songs and artists. Users can also create and update songs, as well as update their account information.
- Song: Manages song-related functionalities. It enables users to read information about songs and the artists who created them. Users can delete songs, create and delete comments on songs, update song details, and create donations related to the songs.

- LogIn / SignUp: Handles user authentication. It allows for the creation and reading of user accounts. It facilitates users logging in and out of the system.

4.11 Component: Controller (Server)

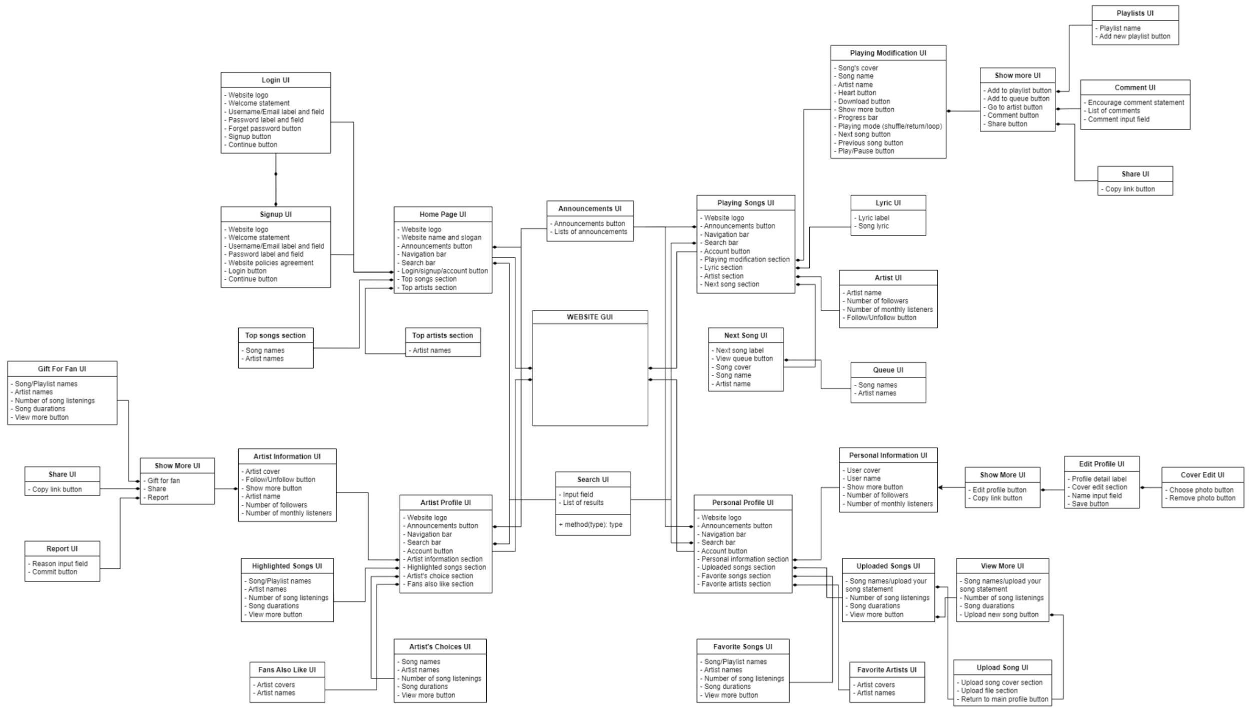


- Home : It would read songs to get information for updating top songs and artists.
- Song: It would take care of reading a song, updating playlists and songs, updating artists, creating, updating, and deleting comments, and adding donations.
- Profile: It would handle actions like reading, updating, and deleting an account, creating and deleting a song, updating favorite songs and artists, and creating donations.
- LogIn / SignUp: It would read accounts to get information and get the account's role to assign permissions. If there is no information, an account can be created if required.

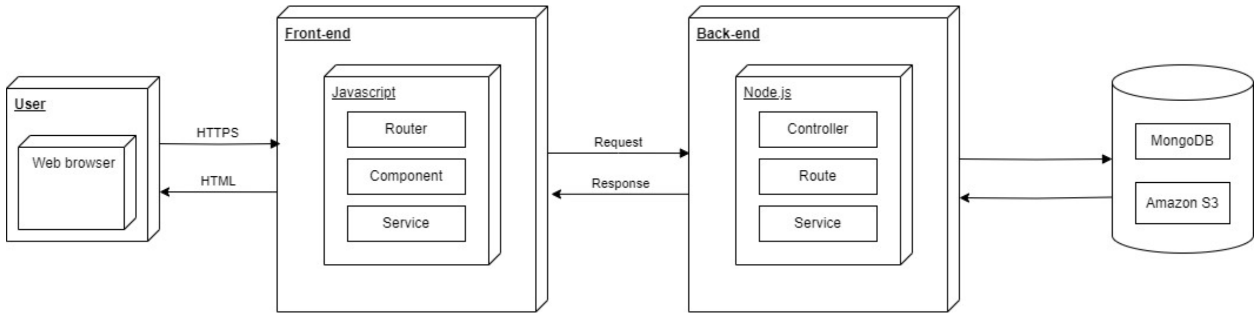
4.12 Component: GUI (Client)

- Class Home Page UI: Initially displays when the website is accessed. Top songs (3-most-listening songs) and top artists (3-most-followers artists) updated weekly are divided into 2 sections so that users can access to listen to the song or see the artist's information.
- Class Playing Song UI: Displays when a song is played. Listeners can modify the way how they enjoy music such as: choose the playing mode (shuffle, return, loop), play the next or previous song, play or pause the song, tracking the lyric, add the song to a playlist or a queue, see more detail of the song (artist, comment), add to favorite playlist, download and share the song.
- Class Artist Profile UI: Displays when an artist is accessed for more information. Users can follow or unfollow the artist, see the artist's popularity within the number of followers and listeners, reference the artist's own best songs with highlighted songs section or his/her choices. Besides, the artists with fans in common are also recommended. Users can look for the artist's "gift for fan", share the artist's profile or even report if there are any problems as well.
- Class Personal Profile UI: Users can customize their profile and track how popular they are with the

number of followers and listeners is shown below their name and cover photo. The profile can be edited by changing the name or cover photo. This page also allows users to upload their song, show their favorite songs or playlists along with their favorite artists.

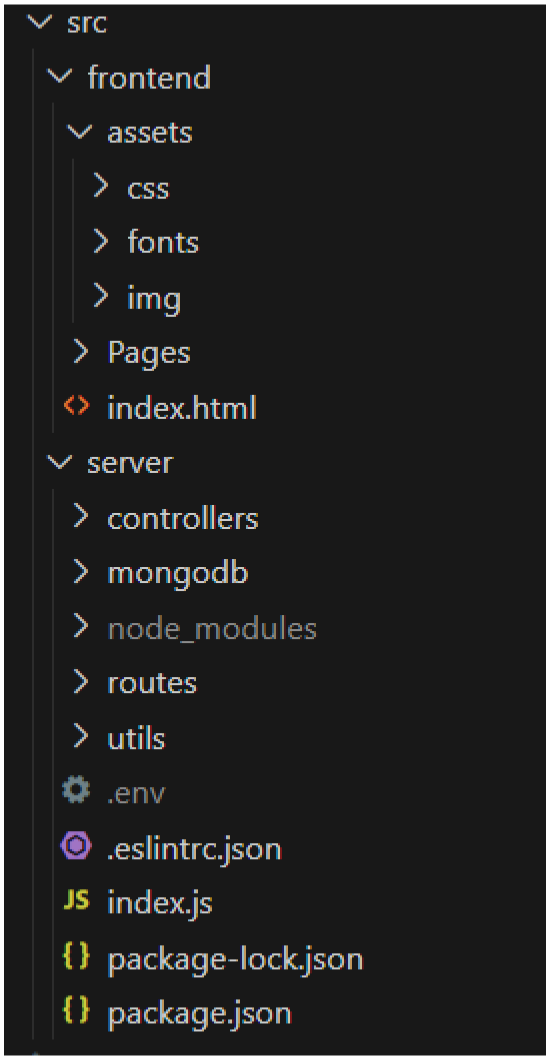


5. Deployment



- Users use a web browser to access the website and call HTTPS to the front-end.
- The front-end then sends HTTPS requests to the back-end according to what the user demands. The back-end receives these requests then query the data from MongoDB for information and files link (music and image files are being stored in Amazon S3 Storage) and return these information to backend as response.
- Finally the front-end gets the response, handles the data and shows it to the user browser in the form of HTML.

6. Implementation View



```
src
  frontend
    assets
      css
      fonts
      img
    Pages
    index.html
  server
    controllers
    mongodb
    node_modules
    routes
    utils
    .env
    .eslintrc.json
    index.js
    package-lock.json
    package.json
```