A logo with text and a picture of a book

Description automatically generated

**SOFTWARE CONSTRUCTION**

**PROJECT REPORT**

|  |  |
| --- | --- |
| **PROJECT NAME** | **MUSIC PLAYER - WEB APPLICATION** |
| **GROUP MEMBERS** | ZAHRA MARYAM  BISMA YASEEN  HUDA ZAHRA  YASHFA HAROON |
| **COURSE** | BESE 27C |
| **SUBMISSION DATE** | 15 January 2023 |

**DEPARTMENT OF COMPUTER SOFTWARE ENGINEERING**

**Military College of Signals**

**National University of Sciences and Technology**

**MUSIC PLAYER**

1. **INTRODUCTION**
   1. **Project Overview**
   2. **Objectives**
2. **PROJECT SCOPE**
3. **TOOLS USED**

* **React.js**
* **Tailwind**
* **Node.js**
* **Mongo DB**

1. **FUNCTIONALITIES**
   1. **User Authentication**
   2. **Search Songs**
   3. **Song Playback**
   4. **Add Song**
   5. **Create and Add Song to Playlist**
   6. **Playlist collaborations**
2. **PROJECT IMPLEMENTATION**
3. **RESULTS**
4. **APPENDICES**
5. **REFRENCES**
6. **INTRODUCTION**

**1.1 Project Overview**

The Music Player Web Application is a platform that offers users an immersive music experience. Using JavaScript in front-end using **React.js** and **Tailwind CSS** for the frond-end designing, **Node.js** for the backend functionalities and utilizing **MongoDB** for the data storage, it ensures a smooth experience for users to manage their music preferences.

**1.2 Objectives**

The project was initiated to create a user-friendly music player web application, enabling user authentication, implementing a music player, adding songs by artist, enabling song search, and delving into collaborative playlist creation.

1. **PROJECT SCOPE**

The Music Player is designed to offer a complete music streaming experience through a web browser. The project focuses on user management, music search, playback controls, song addition, playlist creation, and collaboration features. This includes registering and managing user accounts, implementing an easy-to-use search function, creating an intuitive music player interface, allowing users to contribute new songs, and enabling the creation and collaboration of playlists. The project aims to provide a seamless and comprehensive music experience for users.

1. **TOOLS USED**

The tools used for the development of Music Player to enhance the overall user experience, several tools are used:

* **React.js:**

The component-based architecture facilitates modularity and reusability, enabling us to create an interactive UI. By adopting a simple and declarative approach, we can briefly describe the application's desired state, allowing React.js to efficiently manage updates and changes. As users navigate and engage with the application, React.js ensures a fluid and enjoyable interaction, contributing significantly to the overall success of our Music Player Web Application.

* **Tailwind CSS:**

Tailwind CSS was chosen as our primary styling framework due to its simplicity and efficiency. It allowed us to craft a visually appealing user interface with ease. The utility-first approach of Tailwind CSS streamlined the design process, facilitating the creation of a modern and responsive user interface.

* **Node.js:**

Node.js, paired with Express.js, formed the backbone of our server-side scripting and request handling. This combination offered an optimal solution for managing the backend of our application. The integration of Node.js with Express.js ensured smooth and efficient communication between the client and the server, enhancing the overall performance and reliability of the application.

* **MongoDB:**

MongoDB was selected as our database solution, primarily for its flexibility in storing and retrieving data efficiently. As a NoSQL database, MongoDB is well-suited for the dynamic and evolving nature of our music application. Its document-oriented model enables us to seamlessly and flexibly store various types of data related to songs, playlists, and user profiles, ensuring a scalable and adaptable architecture for our application.

* **Cloudinary:**

Cloudinary to seamlessly upload, store, manipulate, and deliver media assets, enhancing web application performance and user experience. With features like dynamic image transformations, automatic optimization, and on-the-fly format conversion, Cloudinary enables developers to deliver high-quality visuals while ensuring optimal load times.

1. **FUNCTIONALITIES**

**4.1 User Authentication**

User Authentication is designed to offer a secure and user-friendly experience. To provide security and for enhanced safety measures jwt package is used. A JSON Web Token is a compact and URL-safe way of passing a JSON message between two parties. Passport package and bcrypt packages are used for added encryption. With each request from user, jwt token is necessarily passed. Users have two options for accessing the platform:

* **Login:**

Existing users can log in using their username and password.

* **Signup:**

New users can sign up for a free account, providing the following details:

* Email
* Password
* Username
* First name
* Last name

Upon successful authentication, users are directed to the home page, where they access main features of the Music Player Web Application. The home page is crafted to provide intuitive environment for users to navigate and interact with the various functionalities of the platform. Users get Logout feature to gracefully log out.

**4.2 Search Songs**

To enhance user experience, search feature is implemented to allow specific and directed search for desired songs. Tailored for finding favorite tunes within the application's library, search returns a list of desired related songs. Users can search for songs using the song name or the songs of a particular artist using the artist’s name, allowing for a quick and targeted exploration of extensive music collection. This search functionality aims to streamline the process of discovering and accessing specific songs, ensuring provision of personalized and efficient music discovery journey for users. Dynamic searching allows retrieval of songs from DB without entering exact full name(regex used here).

**4.3 Song Playback**

Our Music Player Web Application provides a user-friendly interface for music lovers. With simple controls users are able to play and pause songs at their convenience, providing them flexibility. Additionally, the application offers the functionality to 'like' songs, allowing users express their musical preferences. Users are able to loop songs to repeat songs. Functionality of playing nextSong and prevSong allows moving forward or backward in a playlist. Playback speed option allows us to increase or decrease the speed of song by 0.25 seconds. User can also download the song on their system. Provision of these intuitive features, ensures a more engaging and customized experience for every user.

**4.4 Add Songs**

Users have the ability to contribute to the music library by adding a new song. When adding a song, users must provide following details:

* **Song Name:** enter the title of the song
* **Thumbnail:** provide an image that represents the song
* **Track:** specify the audio file associated with the song.
* **Artist (User Itself):** automatically captured as the user adding the song

This feature enables users to actively participate in expanding the application's music collection, promoting a collaborative musical environment. By allowing users to add songs with key details, a wide range of musical preferences is catered to, fulfilling users’ desires accordingly.

**4.5 Create and Add Song to Playlist**

Our Music Player Web Application empowers users to choose and create their personalized music experience by creating playlists. Following details are required when creating a custom playlist:

* **Playlist Name:** enter name for the playlist
* **Thumbnail:** add an image to present playlist visually
* **Owner (User Itself):** automatically registered as the user creating the playlist
* **Songs:** populate the playlist by adding desired songs

Users can continuously modify playlists by adding or removing songs. This allows a more flexible music-sharing environment within the application.

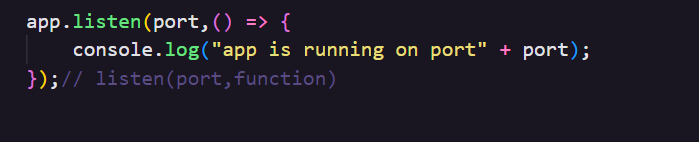
1. **PROJECT IMPLEMENTATION**

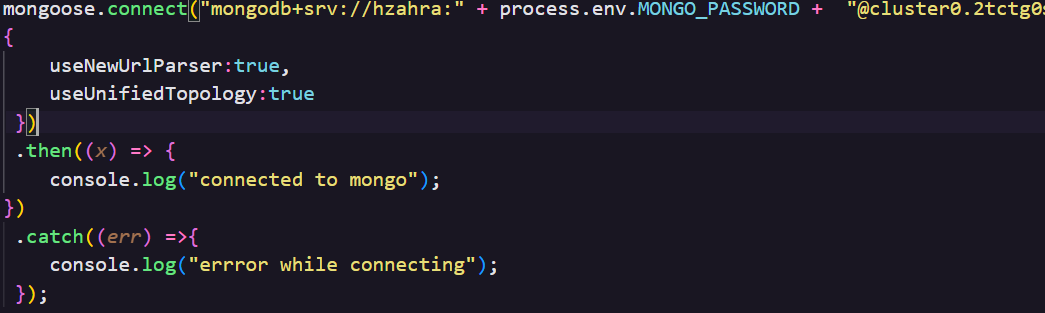
The project is mainly developed in three stages, backend development, testing of APIs and Frontend development and setup.

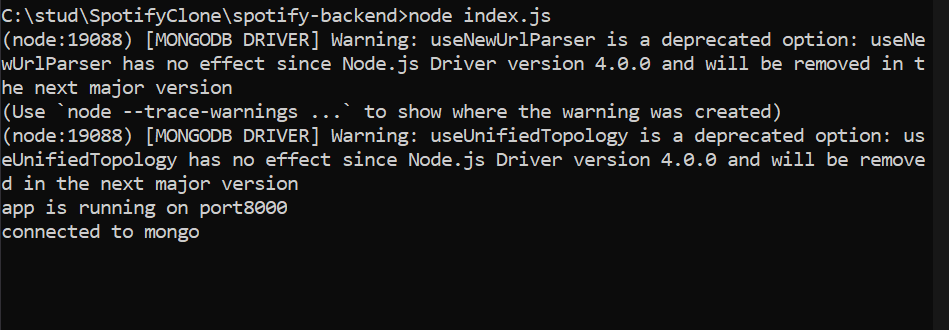
**Backend**

For Backend Node.js with Express.js is used to provide a simple and flexible development environment. Web based MongoDB Atlas is used, where our DB is on a cloud server, accessed and connected via IP Address connection.

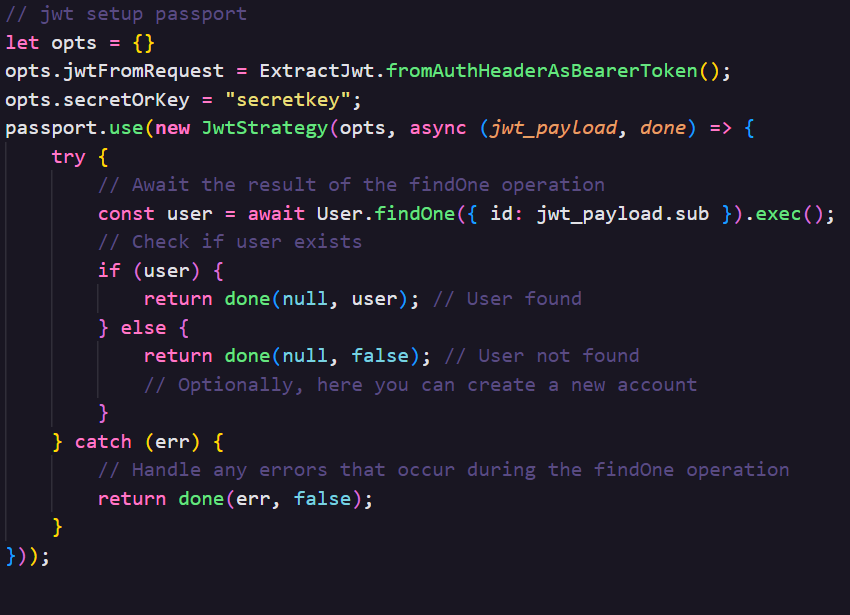
* The application starts from **index.js** where port number, DB Conn, JWT Auth setup and API responses are setup.
* **Port Number :**The port number setup in index.js defines where your application will listen for incoming network requests.

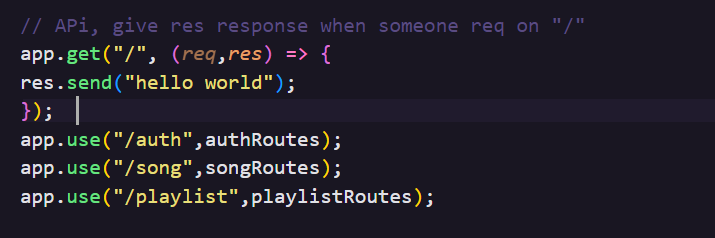


* **DB Conn:** To persistently store and retrieve data, MongoDB Atlas is used. This setup in index.js typically involves configuring connection parameters and establishing a connection to the database. **.env** is used to store DB Password.



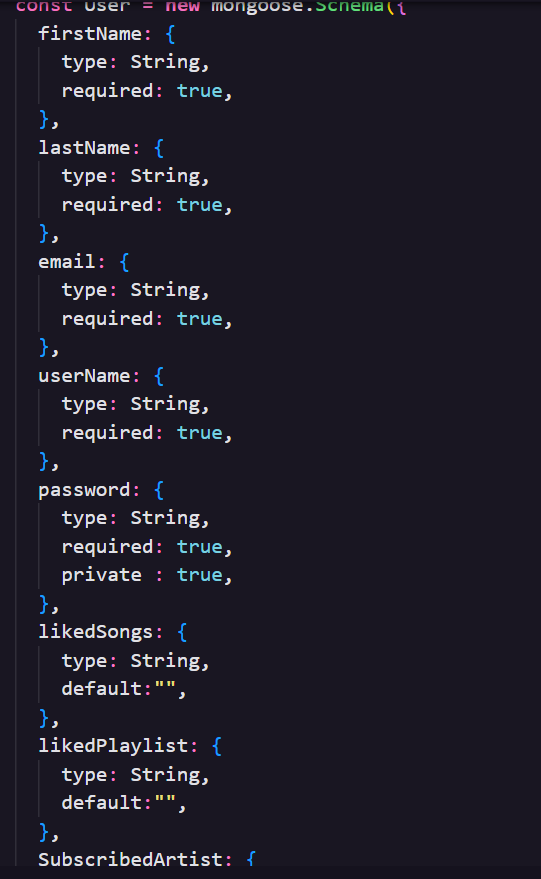
* **JWT (JSON Web Token) Authentication Setup:** JWT authentication is a method to secure your application by verifying the identity of users. In index.js, the setup typically involves middleware that validates JWTs sent by clients, ensuring secure access to certain routes or functionalities in your application.

****

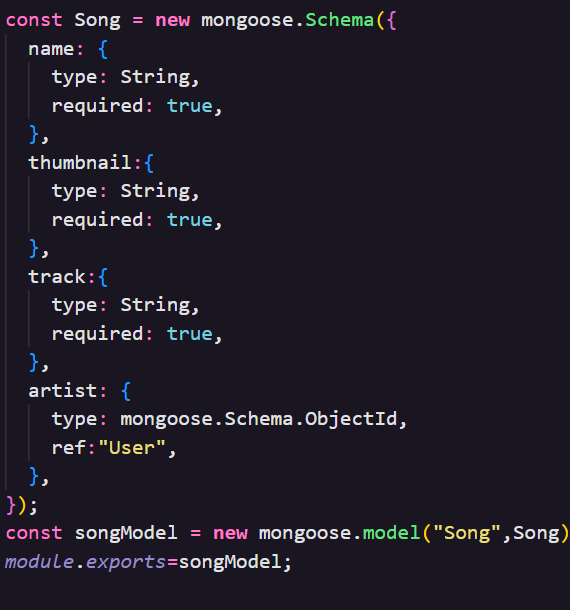
* **API Responses Setup:** In index.js, setting up API responses involves defining routes and their corresponding actions.
* **Models:**

Models are used in structuring and managing data. They are part of the **MVC** (Model-View-Controller) architecture, commonly used in web development. Three models are used in backend namely Playlist.js, Song.js, User.js for managing the Playlists, Songs and Users respectively. All models are exported for usage. Models have schema for instances. For data validity mongoose package in installed. Mongoose enables developers to define data models with fields, data types, validation rules, and default values, making it easier to ensure data consistency.

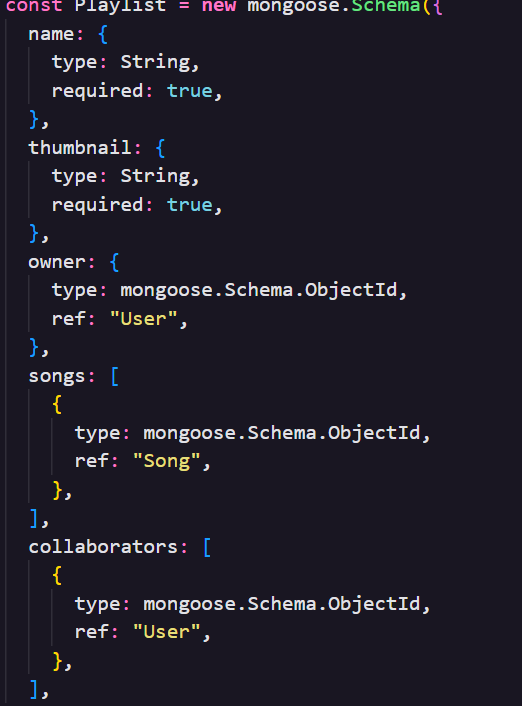
* **User.js**



* **Song.js**



* **Playlist.js**

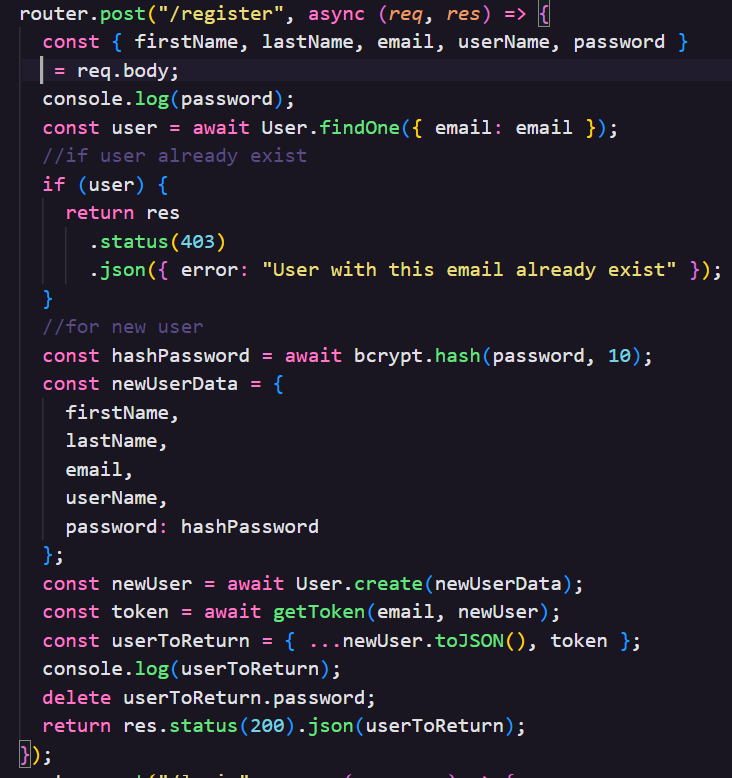
****

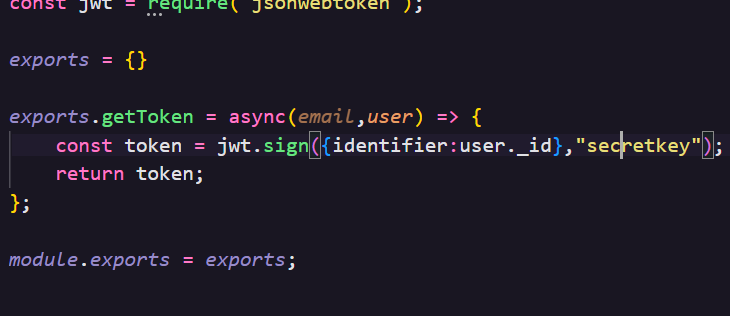
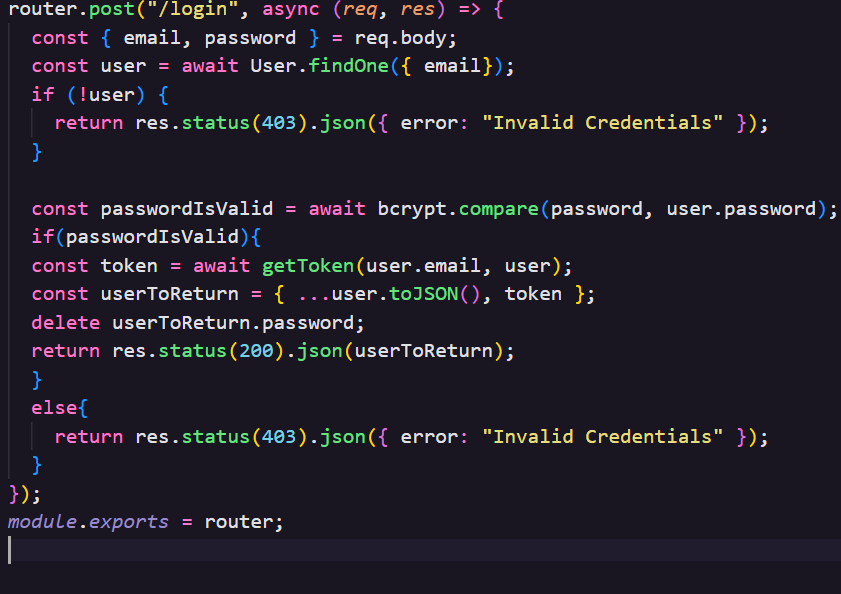
* **Routes:**

Routes direct the flow of data between the server and the client, and play a key role in the functionality of a web application. They respond to client request by responding with inquired routes. Routes use HTTP method (GET, POST, PUT, DELETE, etc.), which specifies the action to be performed. They enable the server to respond appropriately to retrieve data, accepting user input or performing operations. Authentication and other middleware task is performed by routes.

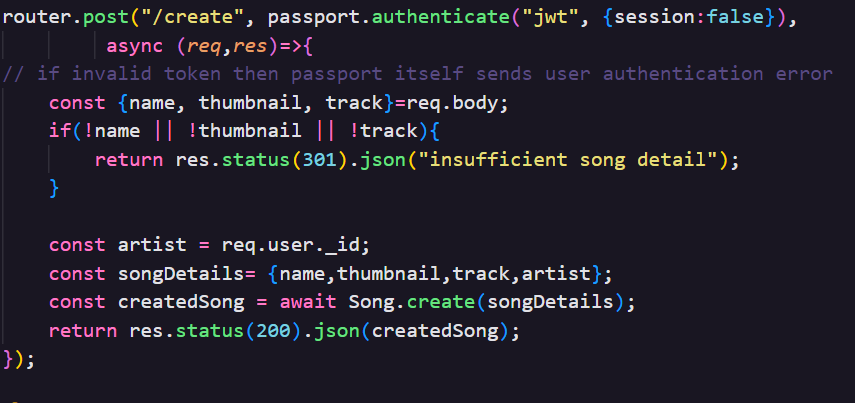
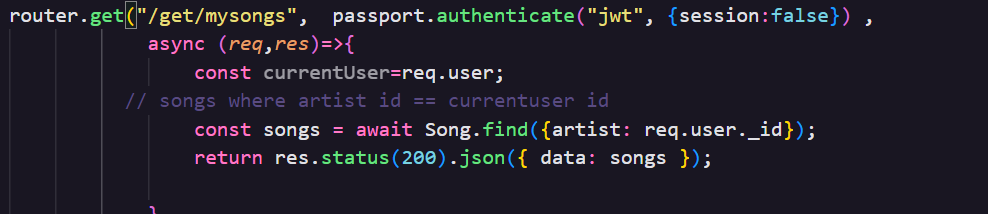
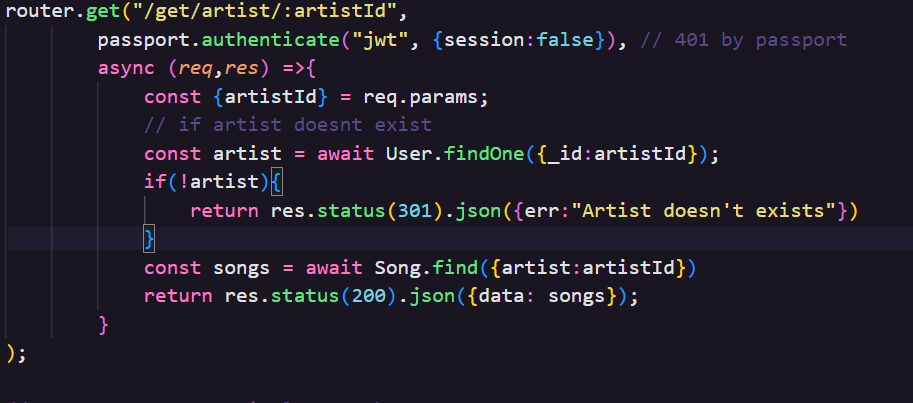
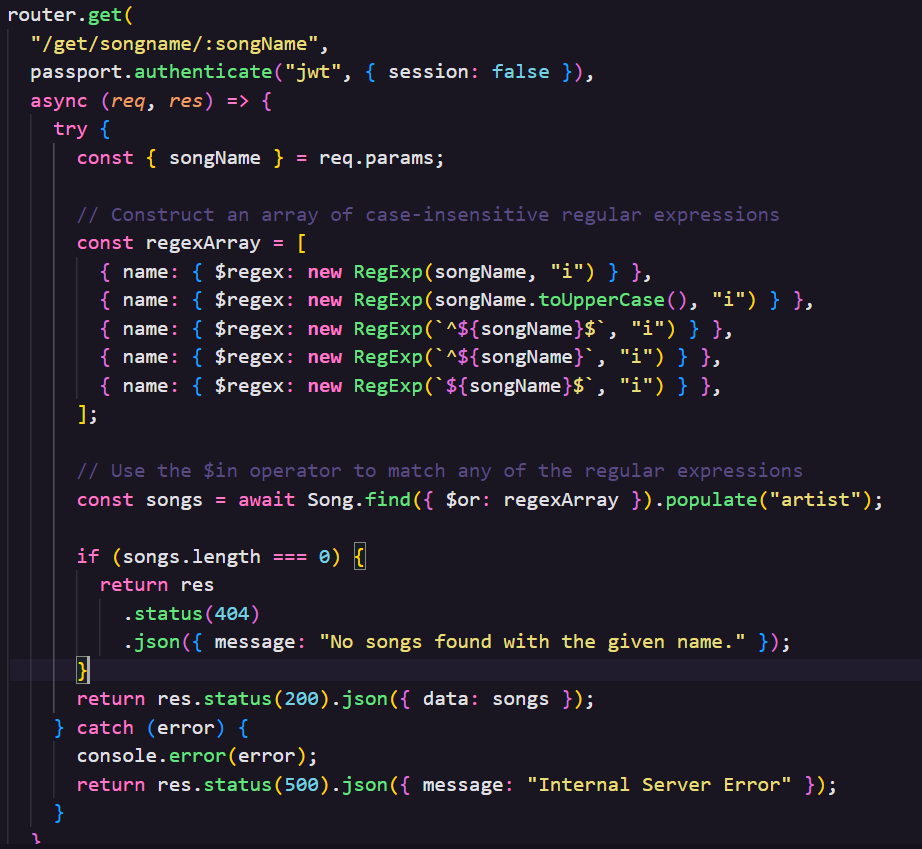
1. **auth.js** : Auth.js registers a new user. For enhanced safety and security, hashed passwords are used to encrypt passwords. Bcrypt, password and jwt packages are used for this purpose. 200 and 403 statuses are used to return responses. Auth.js helps to register new user and login existing user.

* For registering new user :



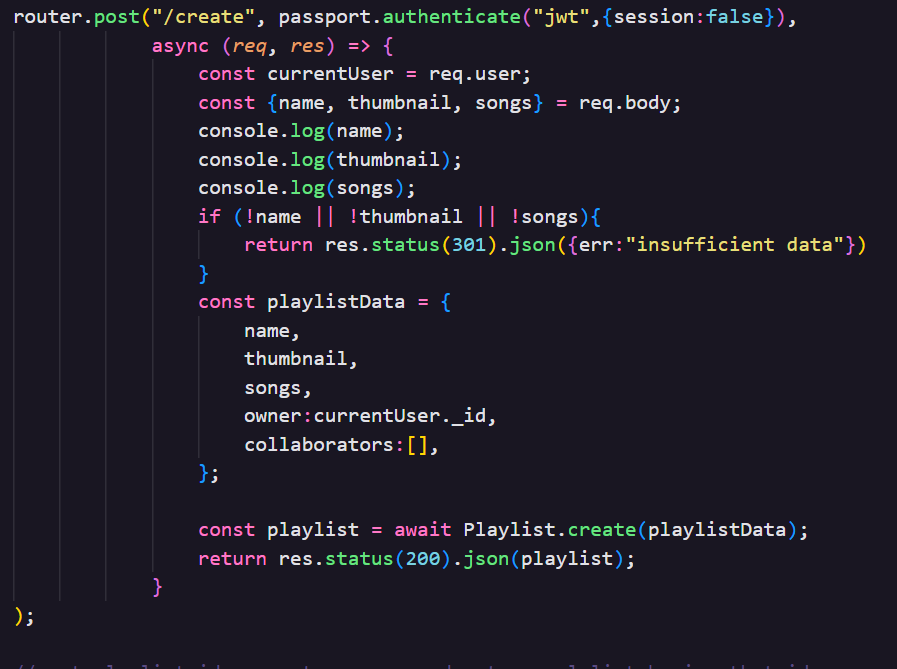
* Hashed password and secret key is used for encryption. It is accessed from utils/helper.js
* To login existing user:

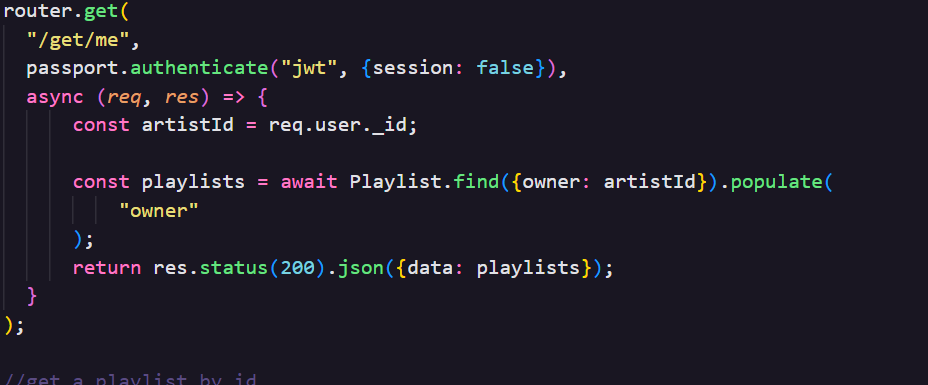
1. **songs.js:** Songs.js allow retrieval of songs from DB from Client request.

* To create songs:
* To get a users’ own song:
* To get songs of any artist by searching name of that artist, if artist doesn’t exist an error will be thrown:
* To get songs by searching song name(dynamic search : use of regex patterns):

1. **playlist.js:** Playlist.js allows retrieval and creation of playlist.

* Creation of playlist

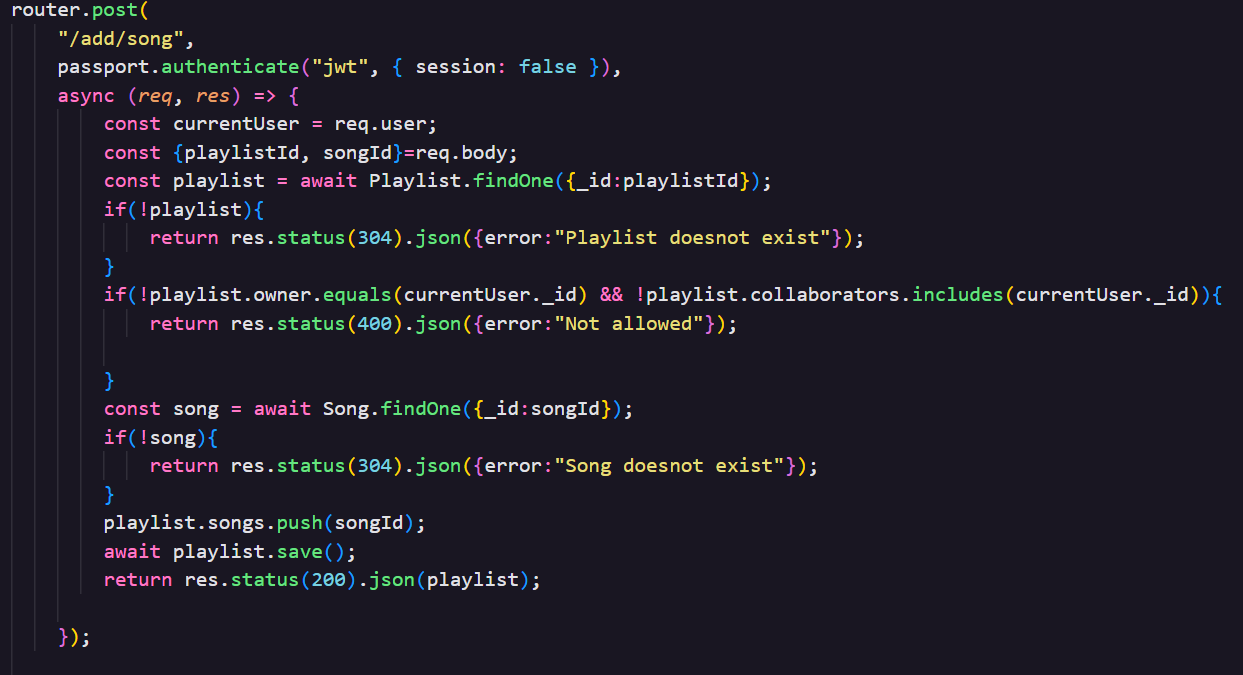
****



* Search for playlist using playlist name



*  Search for playlist using artistname
* Owner of playlist is able to add songs to playlist

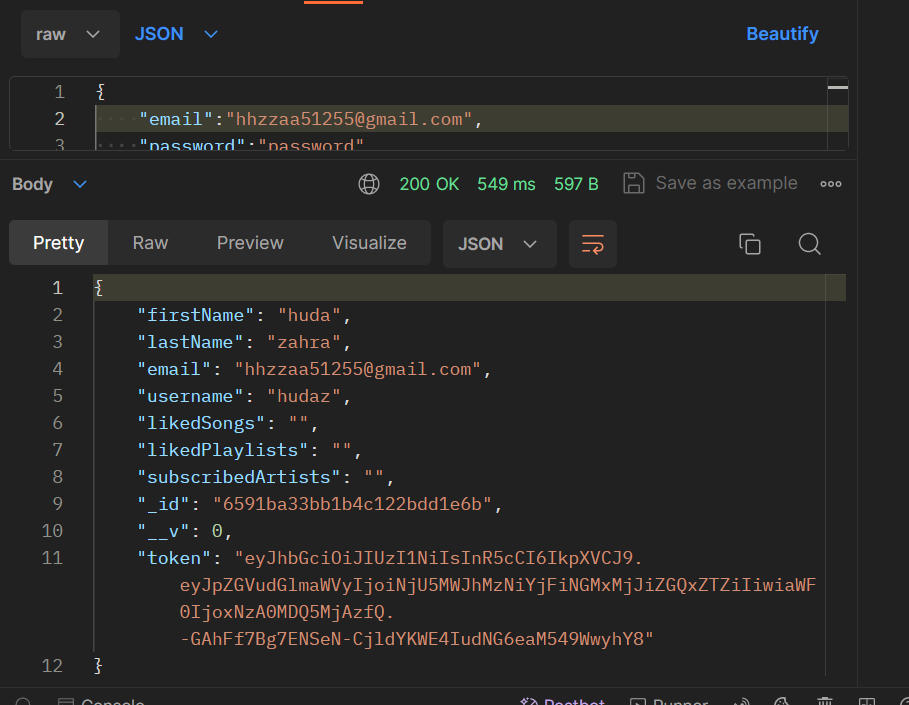


**API Testing:**

To perform API testing for backend, postman was used for our music player. API testing in **Postman** ensures that application's interfaces operate correctly. After installing Postman, related API requests into are grouped collections for easy management. When setting up requests, JWT is given for authorization for each request. JWT is generated uniquely for each user. Execution of these requests use Postman's testing scripts to validate the responses, ensuring that your API meets the necessary performance, security, and functionality standards. This approach not only identifies issues but also reinforces the overall quality of your application's interfaces.

* **Auth API Testing:**

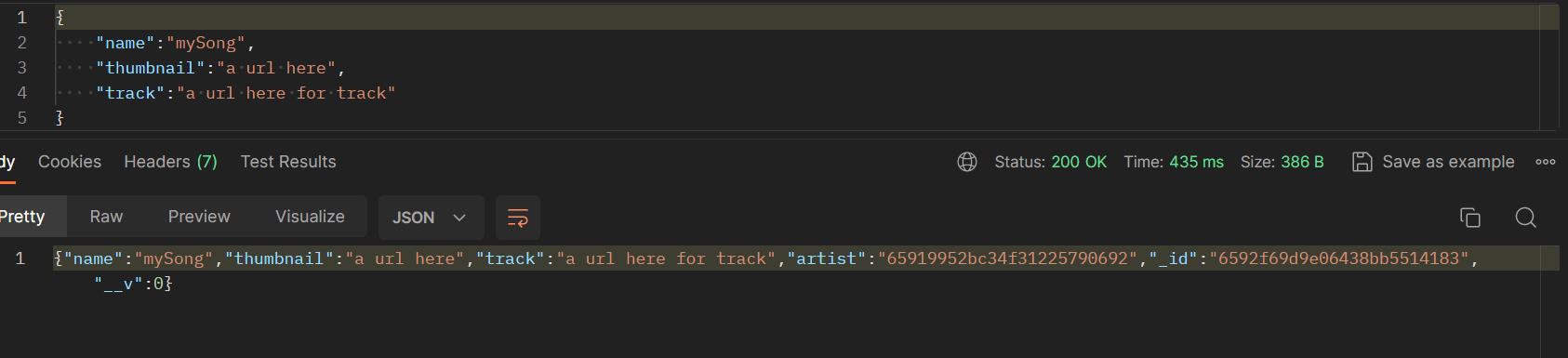
1. **Creation of user**



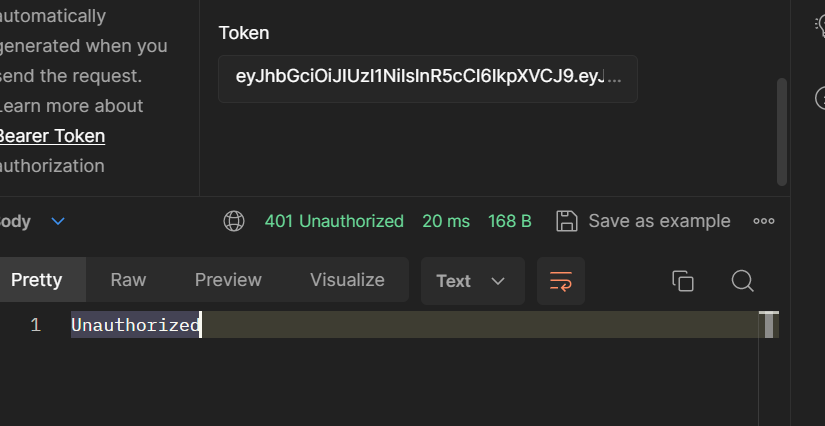
1. **Login API**

* **Songs API:**

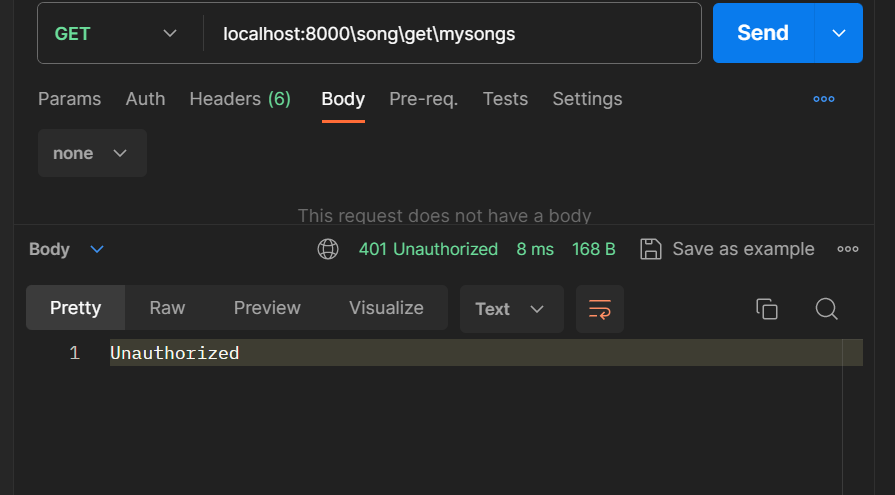
1. **Creation of song**
   * **with Auth**



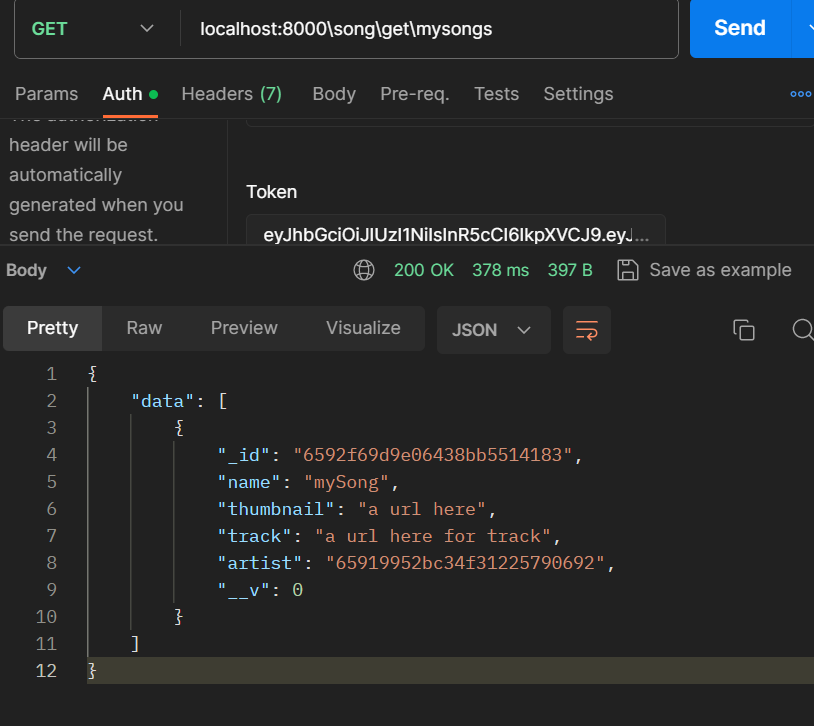
* + **without Auth**



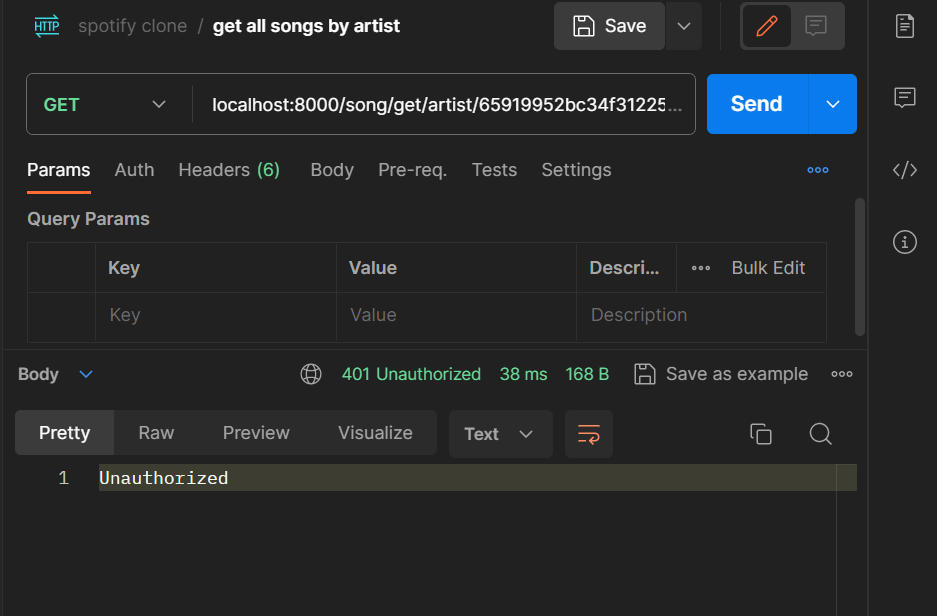
1. **Get All songs** 
   * **without Auth**



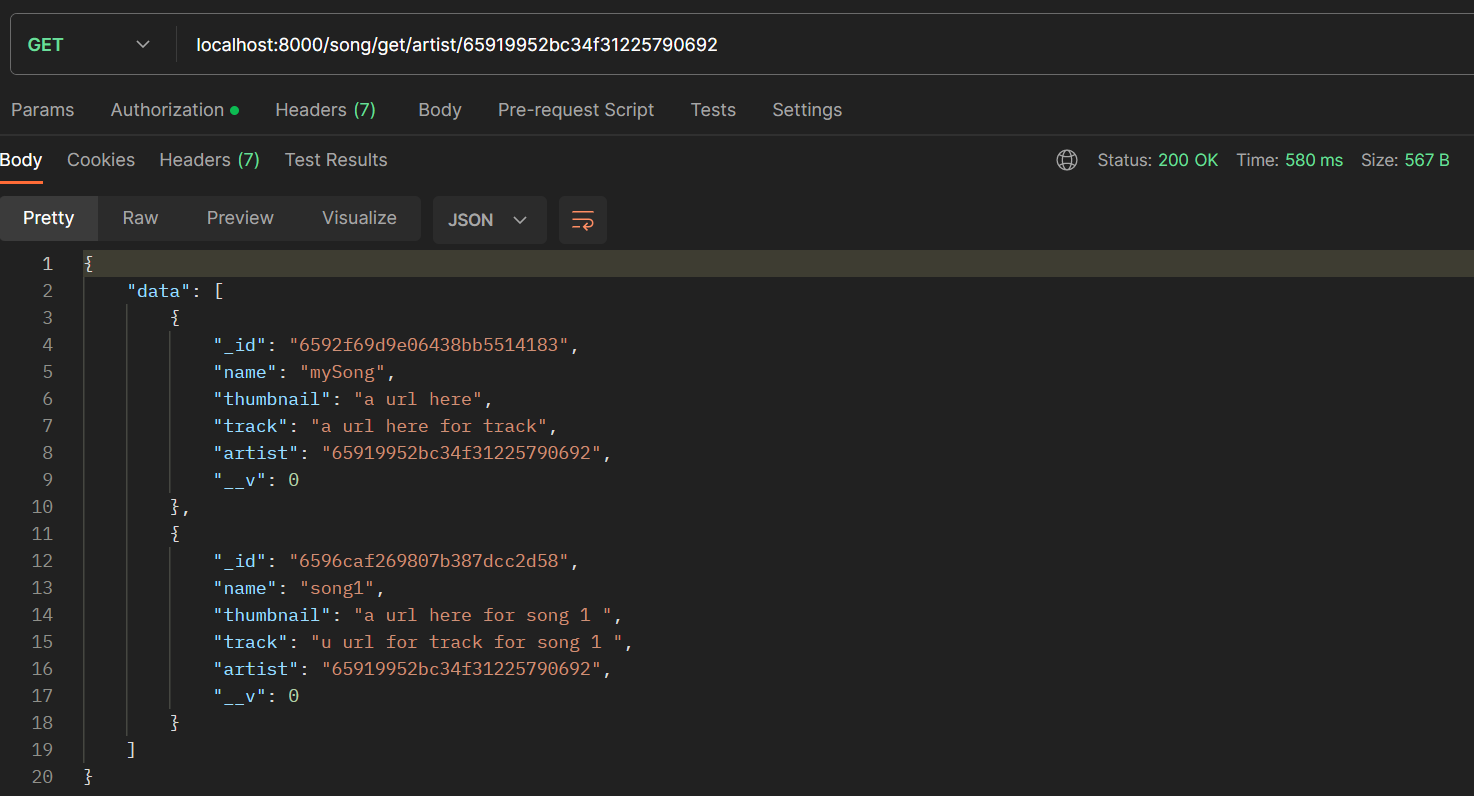
* + **with Auth**



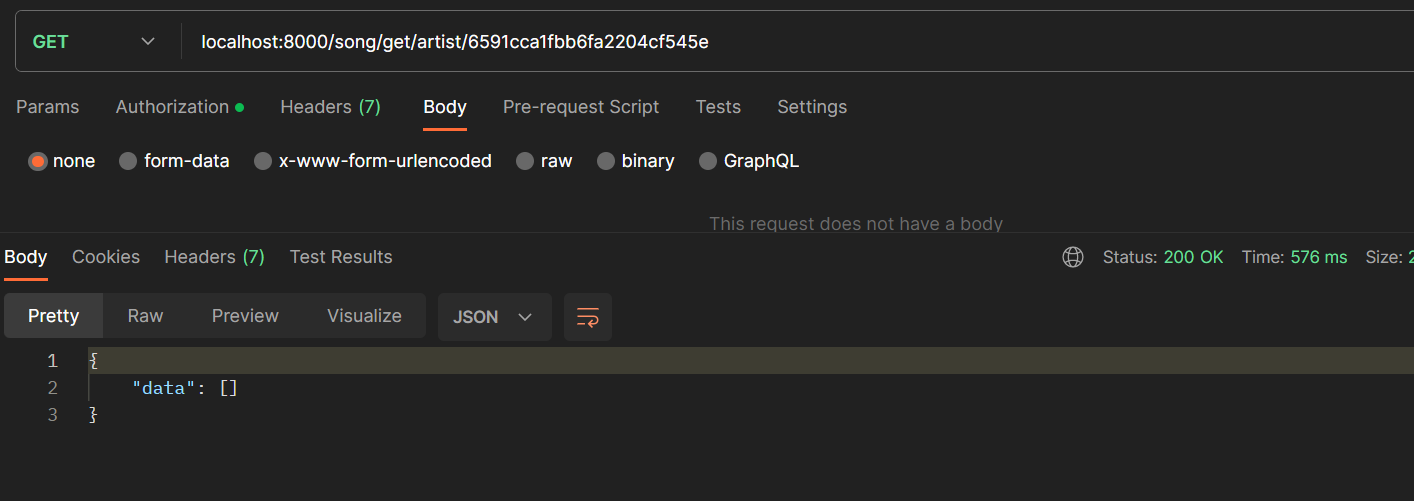
1. **Get songs by artistID**
   * **without Auth**



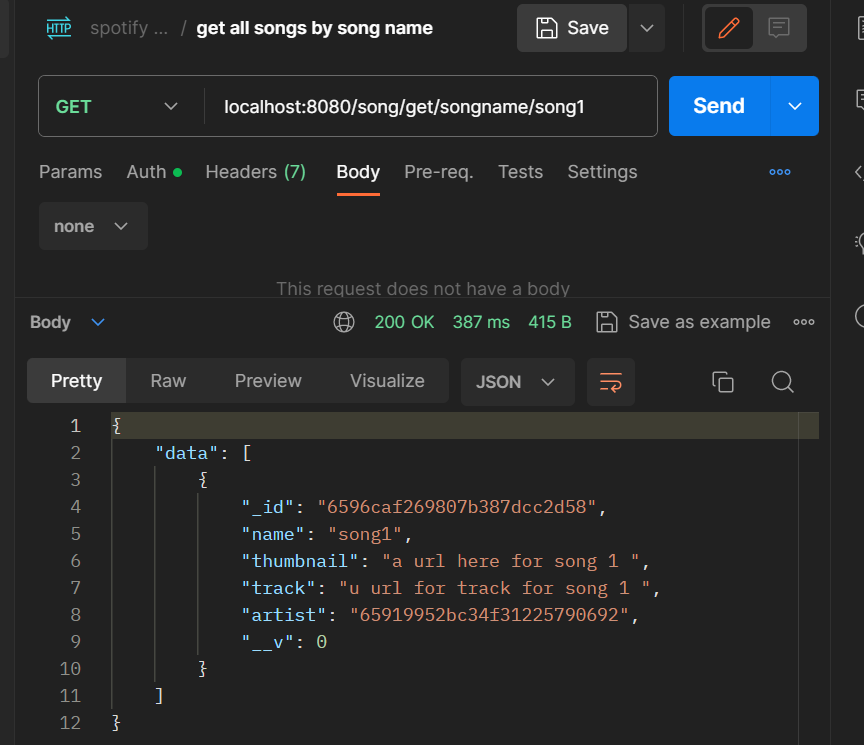
* + **with Auth(JWT Token given)using artistID**



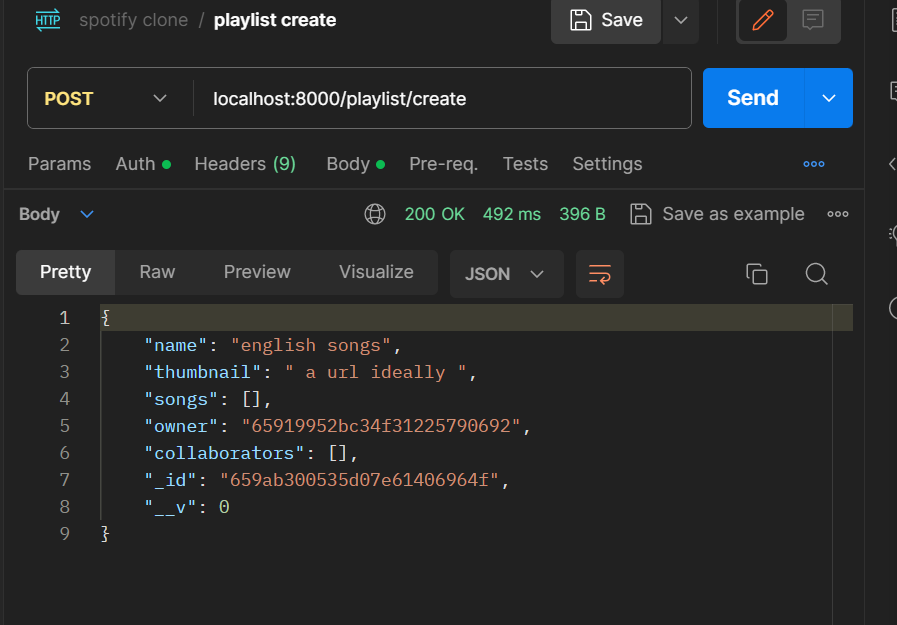
* + **with Auth using ArtistID but artist doesn’t exist**



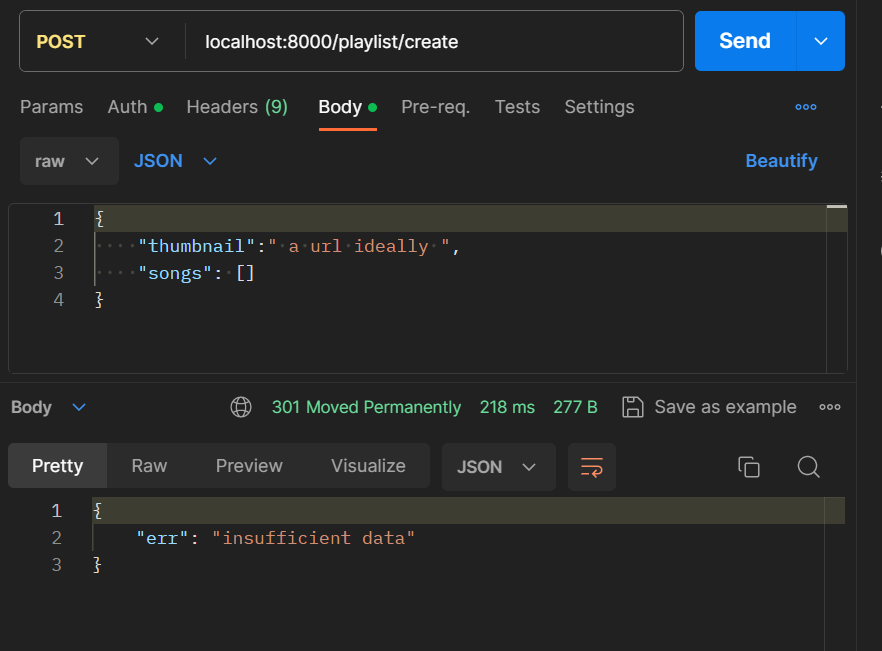
1. **Get Songs by Songname**



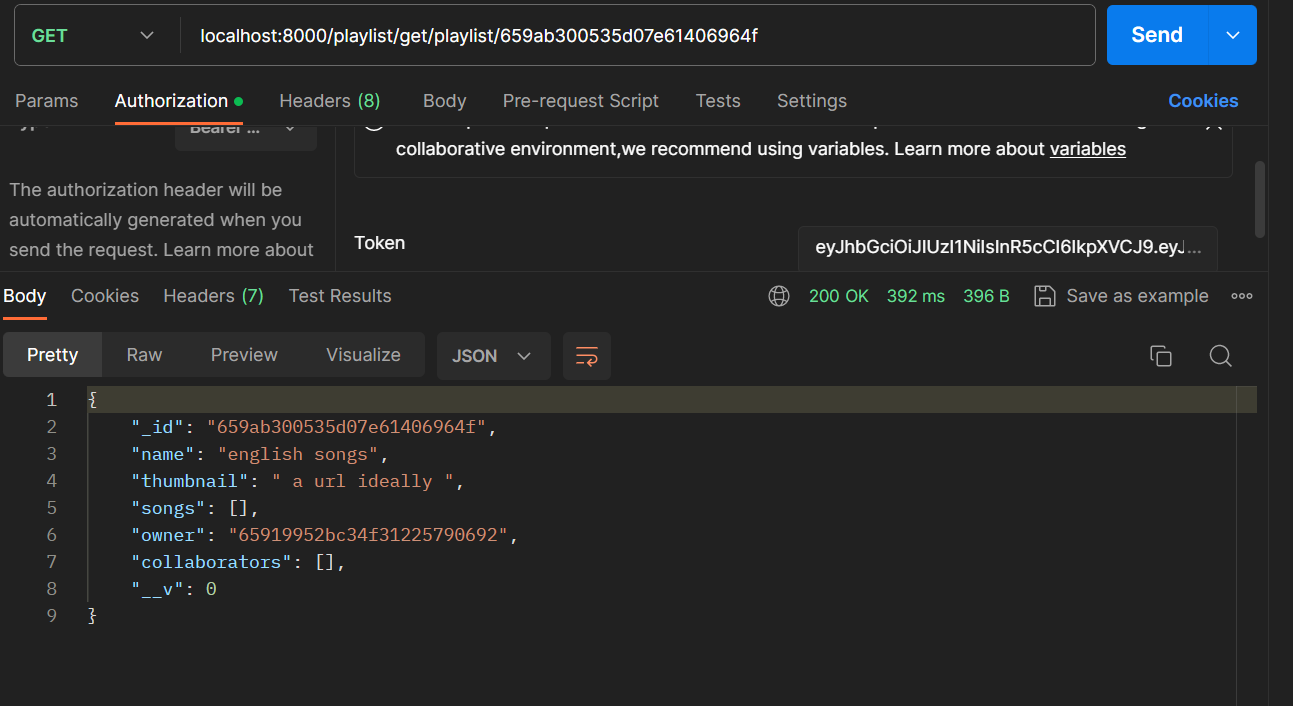
1. **Create playlist**
   * **Successful creation**



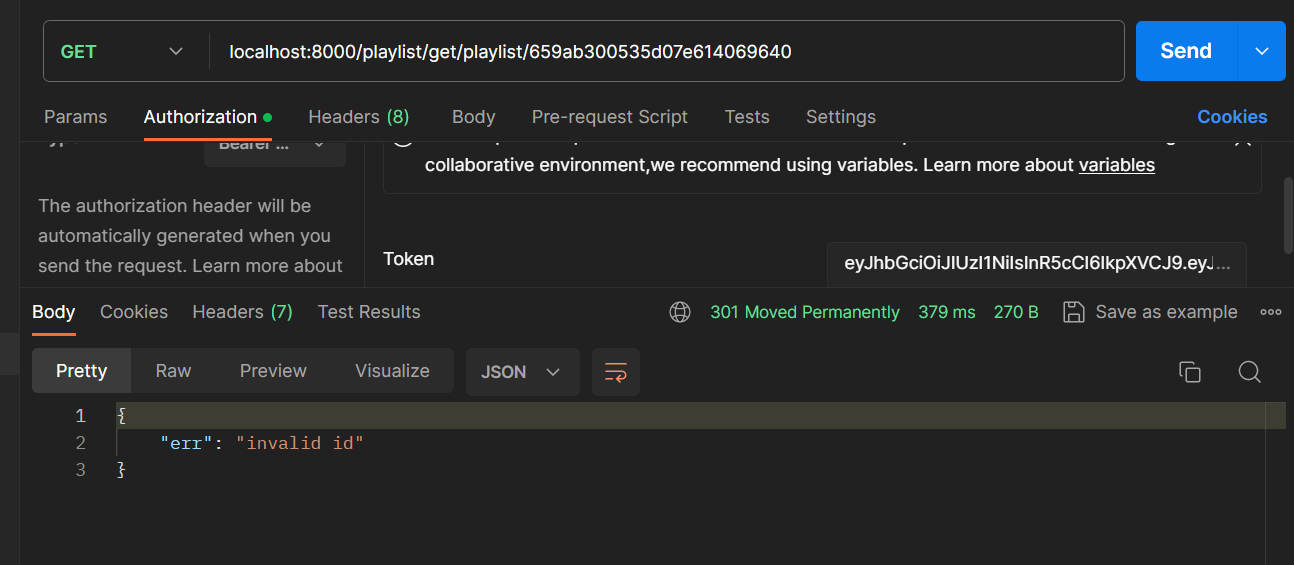
* + **Insufficient data error**



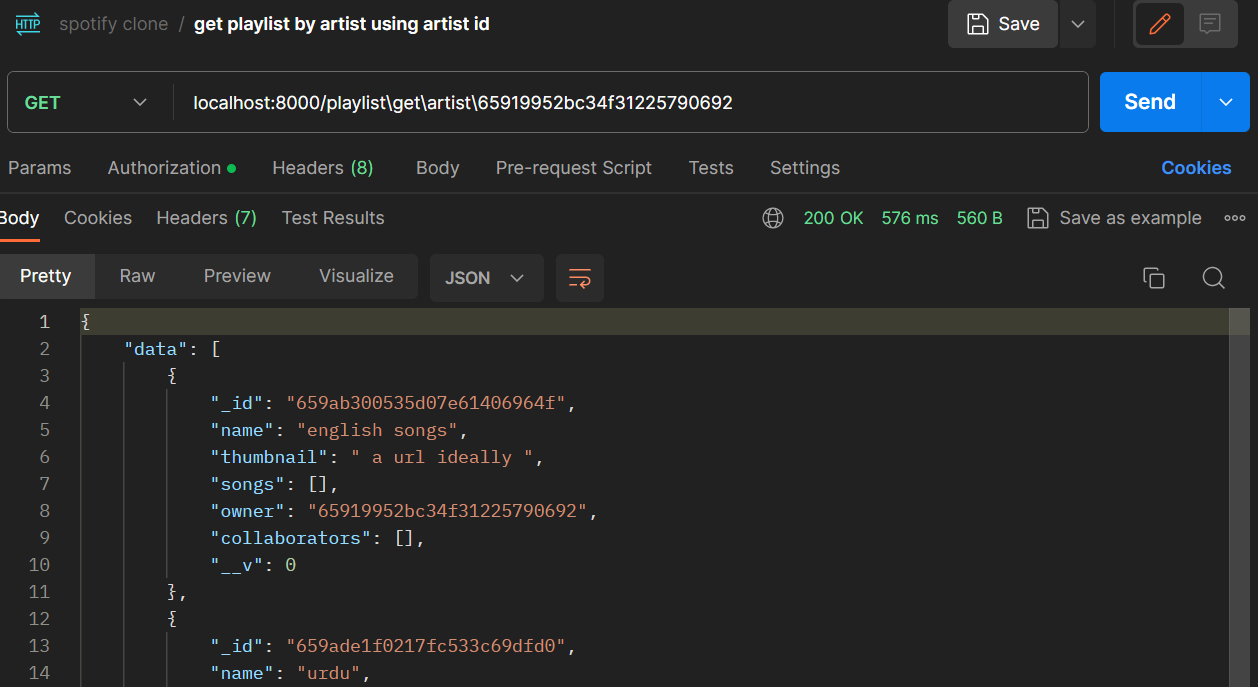
1. **Get playlist by playlistID**
   * **Successful return**



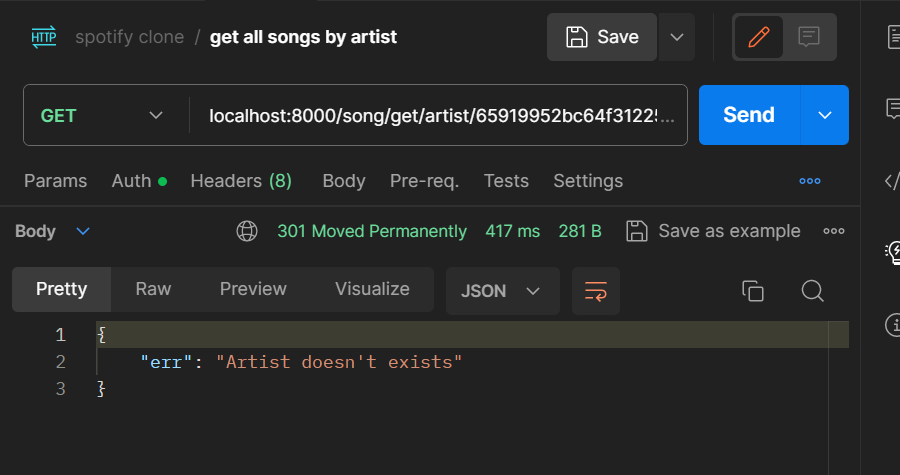
* + **Invalid ID for playlist**



1. **Get playlist by ArtistID**
   * **Successful return**



* + **Artist doesn’t exist**



1. frontend
2. **Add songs to playlist**
   * **Successful return**

A screenshot of a computer

Description automatically generated

* + **Playlist doesn't exist**

A screenshot of a computer

Description automatically generated

**Frontend**

Using React.js and Tailwind CSS for the frontend development of the music app ensures a modern and efficient approach to building a dynamic user interface. React.js facilitates the creation of reusable components, enabling a modular and organized code structure, while Tailwind CSS streamlines the styling process with its utility-first classes, allowing for a responsive and visually appealing design. The combination of these technologies empowers developers to deliver a seamless and interactive music experience for users, ensuring both flexibility in functionality and elegance in design.

* First, we initiate the installation of Tailwind CSS and React by utilizing the CLI/terminal within our project's directory, setting the foundation for developing the front end of our web application.
* Then we setup React router making the specified changes to our App.js file in the front end folder, where we will create routes for the different pages.

**Routes:**

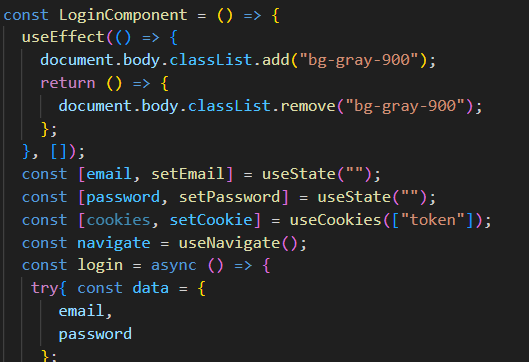
The routes for the front, login, signup, home, music, search and playlist are defined in the App.js file.



**Components**

* **Login Component**

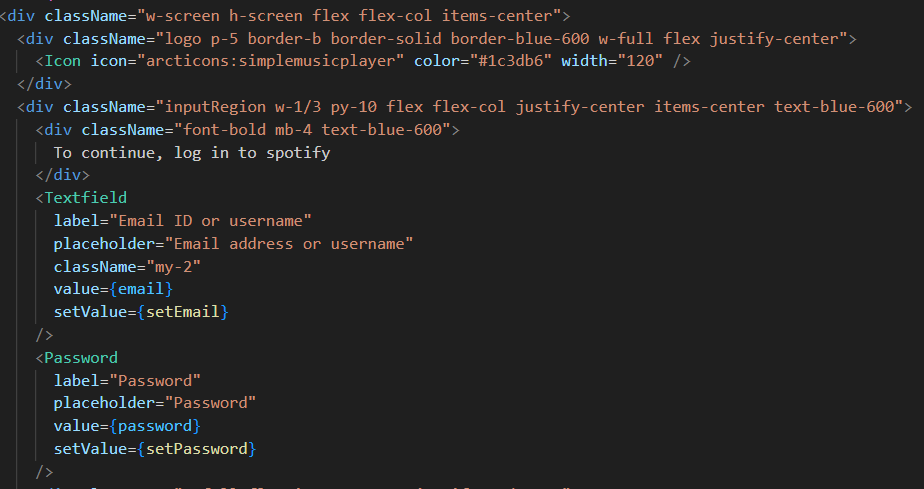
During the login process, the user's input credentials are sent to the server via a POST request that is unauthenticated. The server then verifies the validity of the provided credentials through the makeUnauthenticatedPOSTRequest function. If any errors occur during this verification, an alert mechanism is triggered to notify and guide the user through a smooth and secure login experience. Upon successful login, the user is redirected to the home page. Additionally, after a successful signup, the application generates cookies with a 30-day expiration time



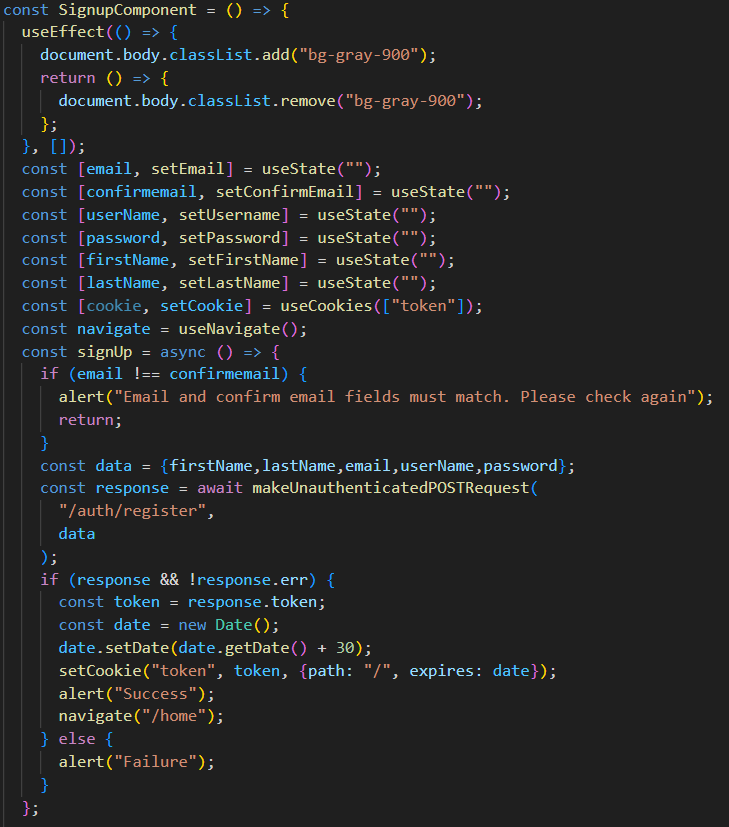
A screen shot of a computer program

Description automatically generated

The styling of the login page is done through the Tailwind classes.



* **Signup component**

During the signup process, the application checks if the user is already registered by making an unauthenticated POST request. If any discrepancies are detected, such as a mismatch between the provided email and confirmed email or missing essential credentials, an error is triggered. Once the signup is successful, the user will be directed to the home page. Additionally, upon successful signup, the application generates cookies with a 30-day expiration time.

* **HomeLoggedIn component**

The Home component retrieves secure song and playlist data using makeAuthenticatedGETRequest and displays it in dynamic cards within a LoggedInContainer. This enhances user exploration on the home screen with a visually appealing and intuitive interface for discovering and accessing content.

* **UploadSong component**

The UploadSong component, enclosed within a LoggedInContainer, enables users to upload songs through a user interface. It comprises input fields for the song's name and thumbnail, as well as an option to upload the song file via a Cloudinary upload component or by entering the file URL directly. Upon submission, the submitSong function is triggered, which sends a POST request to the server to create the new song. After a successful submission, the user is alerted, and the Home component is navigated to.

* **MyMusic component**

The MyMusic component, enclosed within a LoggedInContainer, showcases a user's personal collection of songs. It retrieves the user's songs from the server using the makeAuthenticatedGETRequest function and displays them in individual song cards using the SingleSongCard component. Each card displays essential details such as song name, artist, and thumbnail. The MyMusic component offers a visually appealing and intuitive way for users to explore and enjoy their curated music collection.

* **Library component**

The Library component is a user interface element within a music application that allows logged-in users to access their personalized playlists. It retrieves the user's playlists from the server using the makeAuthenticatedGETRequest function. The component then displays the playlists in a visually appealing grid layout using the Card component, which is a design pattern that presents content as a series of cards. Each card displays the playlist's title, thumbnail, and an option to navigate to the playlist details page by clicking on it.

* **Search Component**

The SearchComponent is a React component that enables users to search for songs within the music application. It consists of an input field, a search icon, and dynamically updates the displayed song results based on user input. The component makes authenticated GET requests to fetch song data from the server, and the search results are presented in a grid layout using SingleSongCard components. If no results are found, the component displays a corresponding message. This user-friendly interface allows users to easily discover new music within the application.

* **Home Component**

The HomeComponent in a music application is a React component presenting playlists through visually appealing cards with titles, descriptions, and images. The left sidebar offers navigation options, including "Home," "Search," and "Your Library," as well as features for playlist creation and accessing liked songs. The dynamic content in the main area allows users to scroll through diverse playlists, each represented by an image, title, and description

* **Front component**

The Front component is the main entrance to the music app, serving as the landing page with a navigation bar, carousel, and sections for showcasing different features. It has a responsive hamburger menu for mobile devices and easy-to-use links for navigation. The carousel uses a custom component to display dynamic content, such as featured songs with title, artist, and thumbnail images. Additionally, there are sections for displaying the latest releases and exploring various genres.

* **Upload Song**

The UploadSong component in our music application allows users to easily upload their own songs. It has input fields for the song name and thumbnail, as well as a Cloudinary component for uploading the actual song. Users can preview the uploaded song's details before submitting. Upon submission, the component sends an authenticated POST request to create the song on the server. If successful, the user is alerted and redirected to the home screen.

* + **LoggedInContainer:**

LoggedIn container uses React Context to centrally manage music playback interface state, ensuring seamless synchronization of playback controls and related features for an efficient user experience.

* + **SingleSongCard**

SingleSongCard dynamically displays song information and allows users to toggle play/pause using the songContext, providing an interactive and visually responsive interface tightly integrated with the music playback system.

* + **Carousel**

The Carousel component is a React component that utilizes the react-slick library to create a dynamic and visually appealing image carousel. It includes four slides featuring different background images, each contributing to an immersive user experience. The component is configured with various settings, such as autoplay, speed, and slide navigation dots. These settings enhance the overall user interaction with the carousel.

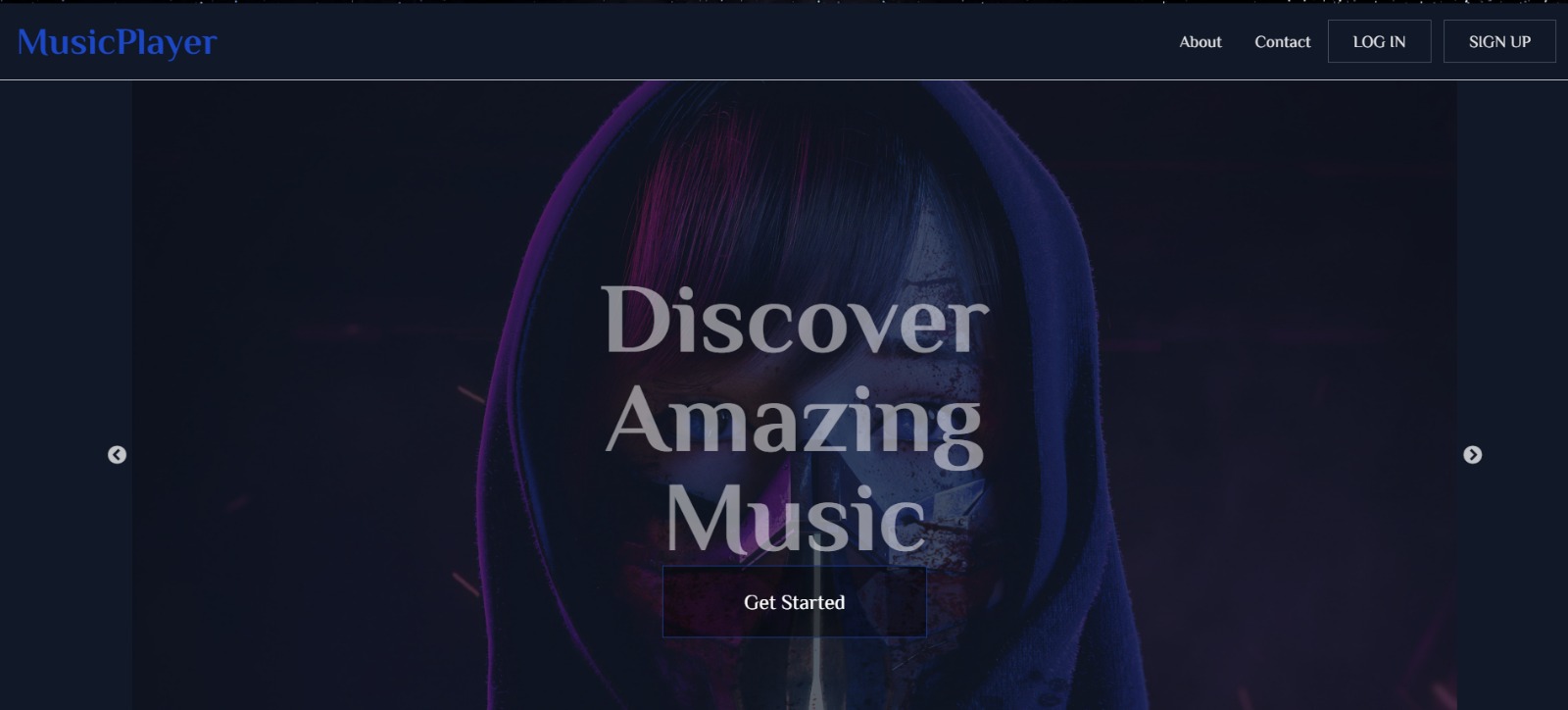
* + **CloudinaryUpload**

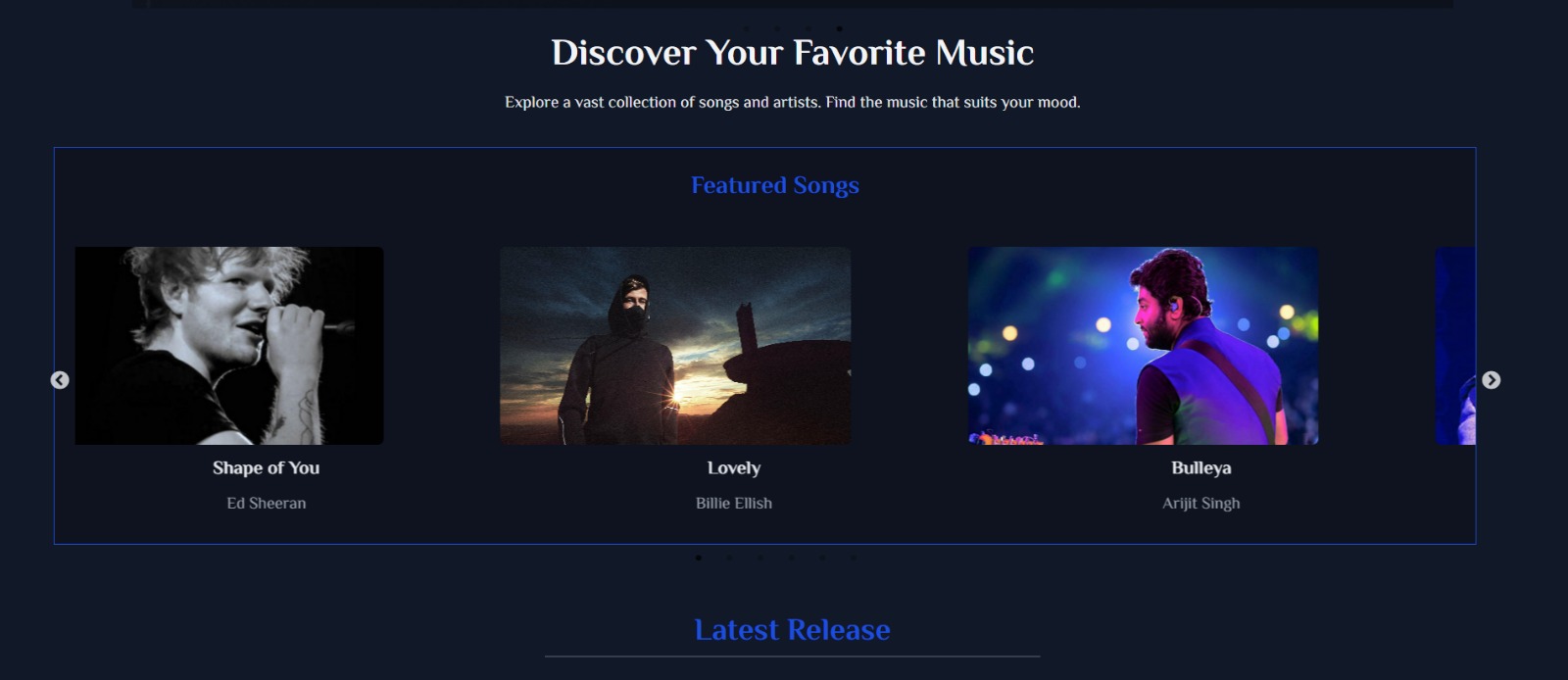
The CloudinaryUpload component is a React component designed for handling image uploads using the Cloudinary service. It utilizes the openUploadWidget function from a CloudinaryService utility and allows users to trigger the image upload process by clicking a button. Upon successful upload, the component updates the state with the uploaded image URL and name, providing a seamless integration for incorporating Cloudinary image uploads into a React application.

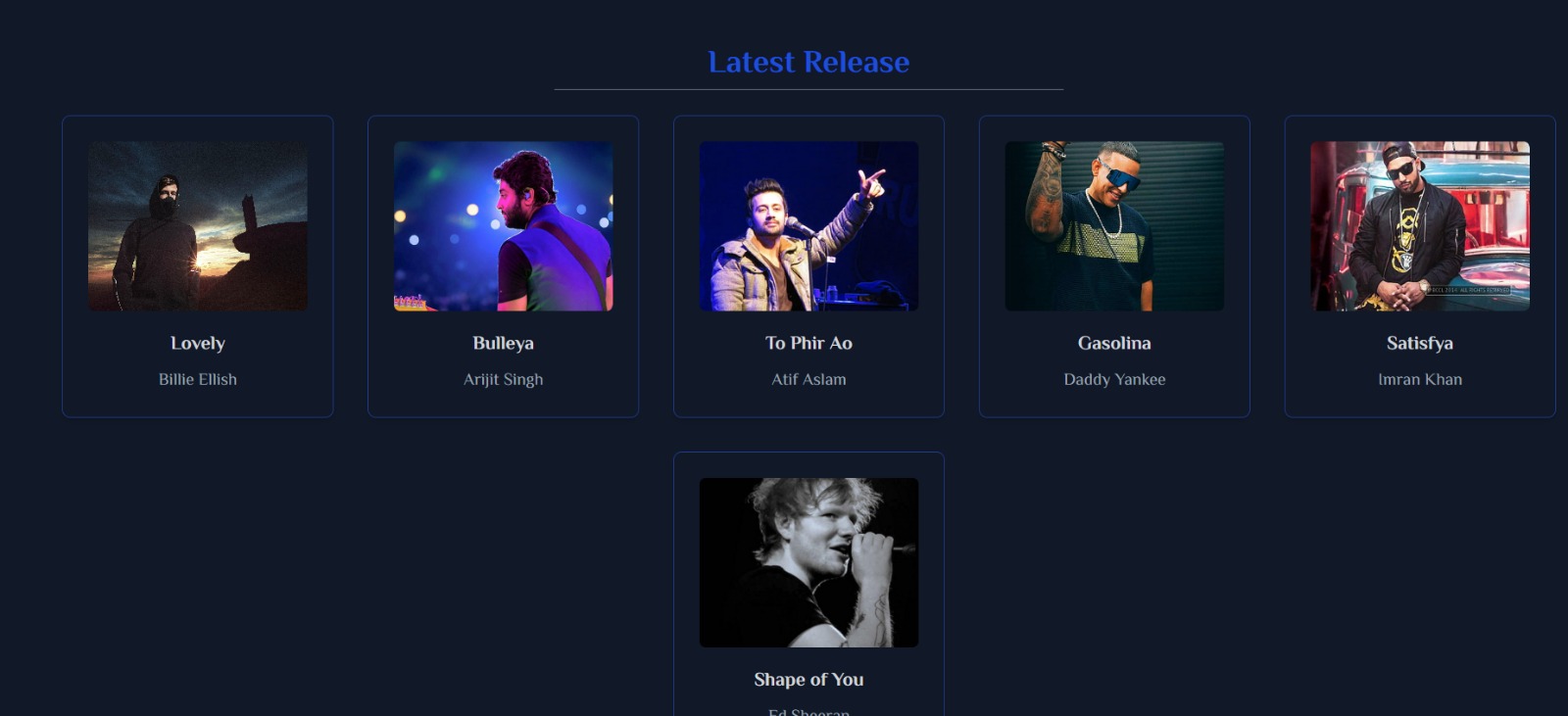
1. **USER MANUAL**

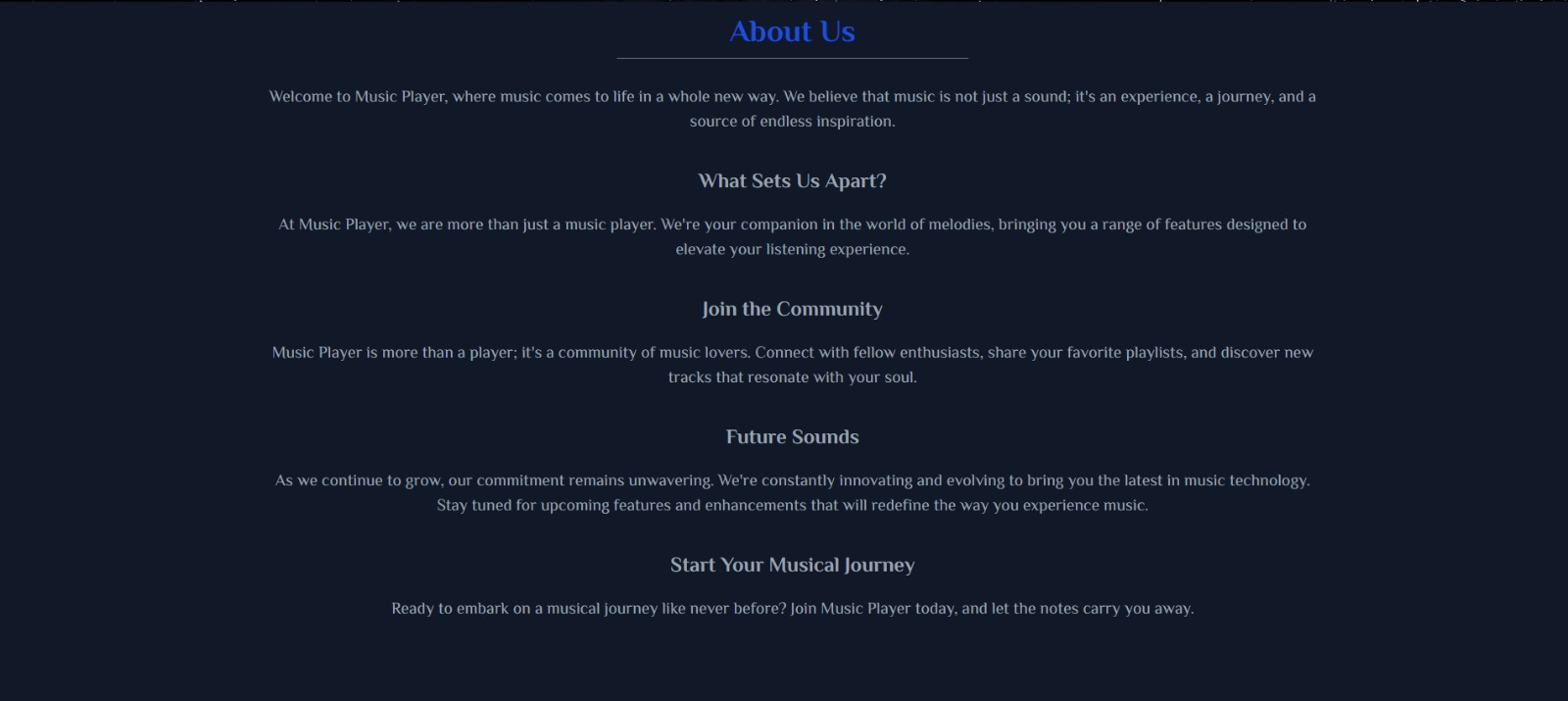
**1. Front page**

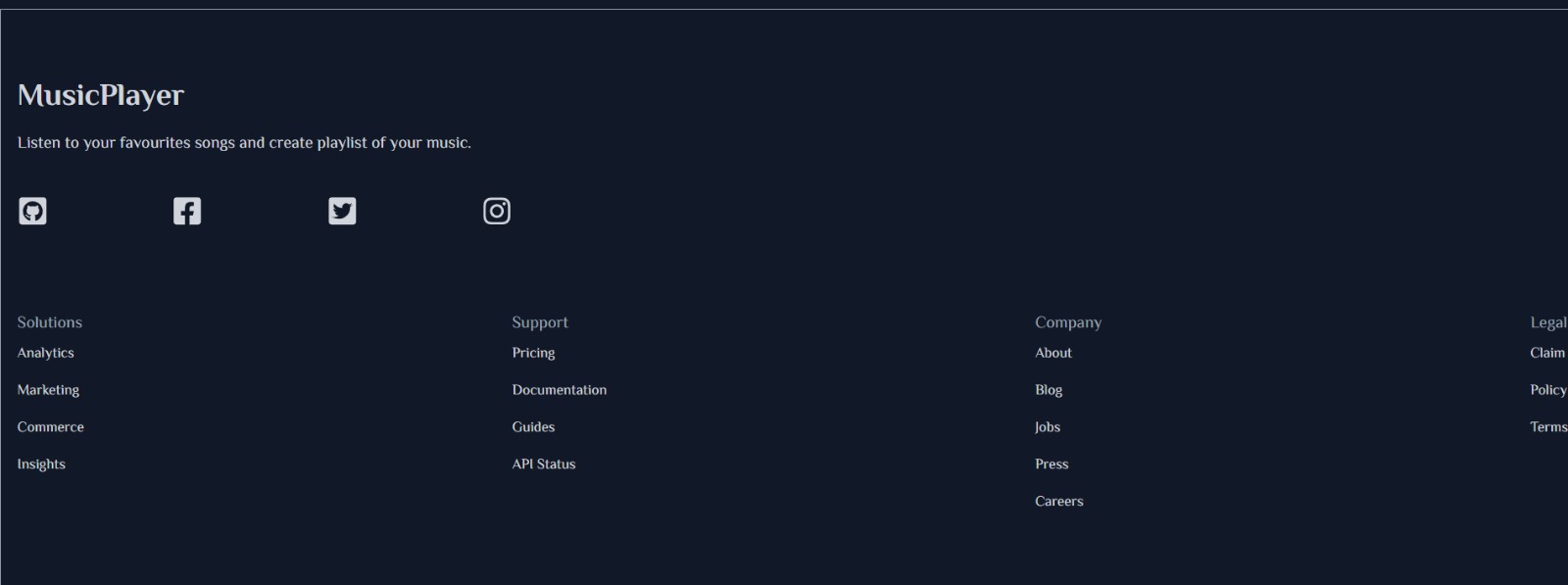
Upon opening the app, the user will see the front page with options to browse, login, and view carousels showcasing contents.





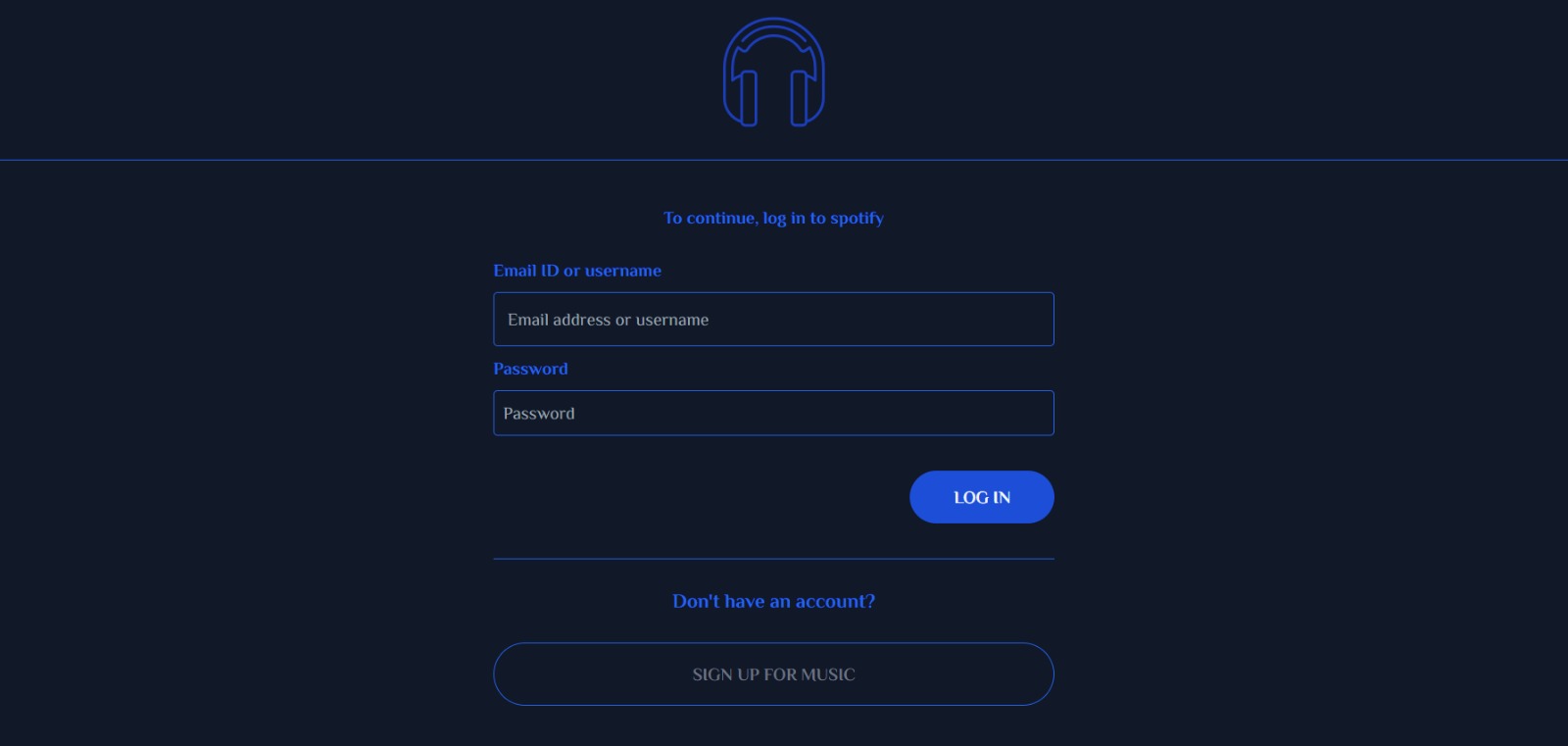






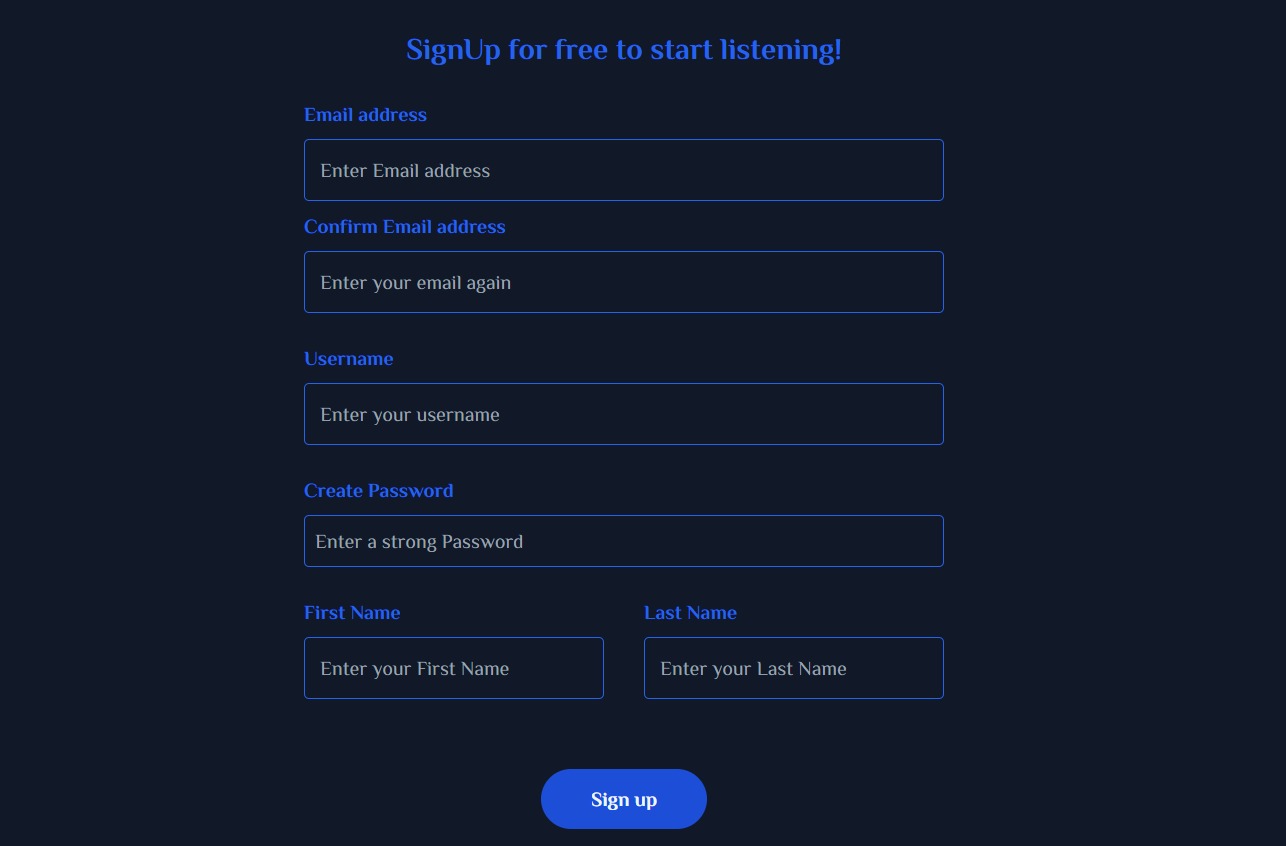
**2. Login**

If the user is already registered, they can log in by entering their username and password.



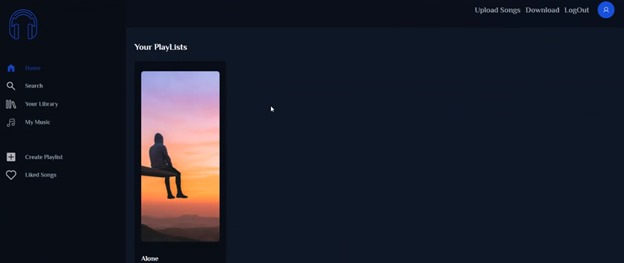
**3. Signup**

If the user is not registered, they can sign up by providing their email, confirming it, choosing a username, creating a password, and entering their first and last name. If the confirmed email is different, or the user does not enter all the required credentials the signup will not proceed.



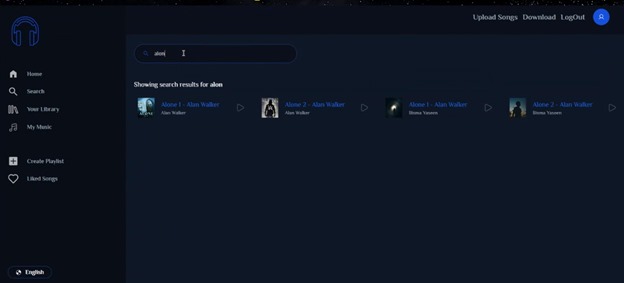
**4. Home page**

After logging in or signing up, the user will be directed to the home page where all songs are displayed.



**5. Search for a song**

The user can search for a song using either the song name or artist name using the search icon. The user can search for the song dynamically by entering a few letter for the song and the related songs will be shown.



**6. Play song**

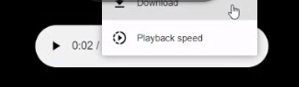
Once the song is found, it will begin playing automatically. The user can control playback using the play, pause, next, and previous buttons at the bottom of the screen.

**7. Song Playback Options**

The playback options include adjusting the volume, increasing or decreasing the speed of playback, adjusting the playbar, and replaying the song.

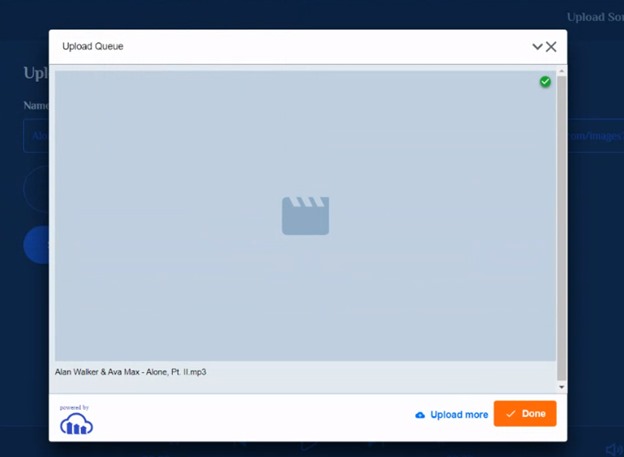
**8. Download song**

The user can also choose to download the song for offline listening by selecting the download option. The user will be directed to a new page where they can download the song.



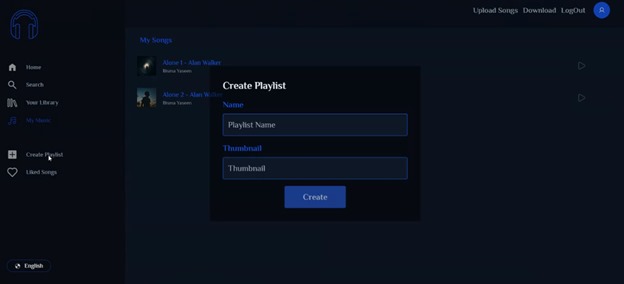
**9. Upload Song**

After logging in or signing up, the user can upload a new song by providing the song name, URL for the thumbnail, and the MP3 file of the song from their device's storage or cloud.



**10. Create Playlist**

The user can also create a new playlist by providing a name for it.



**11. Add to playlist**

User can add the currently playing song to their already created playlist by selecting the add to playlist icon and choose the playlist to add the song to.

**12. View My Music**

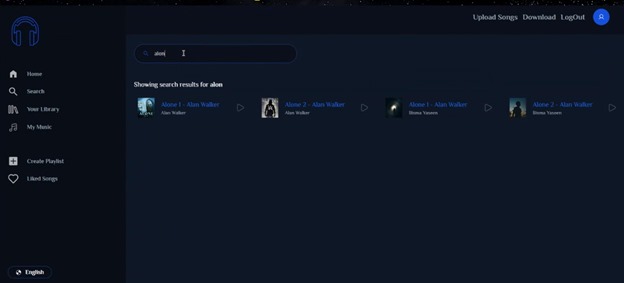
To view their songs, the user can click on the "My music" icon.

**13. Explore Playlists**

The user can view their own playlists from the Playlist icon.

**14. Logout**

To logout smoothly, the user can click on the "Logout" button in the settings menu or on the login page after finishing their session ensuring a secure and controlled user experience.



1. **RESULT**

The key outcomes and results of the project include:

* **User-Friendly Interface:**

The application provides an intuitive and visually appealing user interface, courtesy of Tailwind CSS and React.js, enhancing the overall user experience.

* **Secure and Seamless Authentication:**

Users can securely log in using their usernam and password or sign up with additional details, ensuring a smooth onboarding process.

* **Efficient Song Search:**

The implemented search feature facilitates easy exploration, allowing users to find their favorite tunes within the application's extensive music library.

* **Playback Control and Song Interaction:**

Users can play and pause songs and like specific tracks, creating a personalized and interactive music environment.

* **User Contribution to Music Library:**

The "Add Song" feature empowers users to actively contribute to the music library, allowing users as an artist to add to music library.

* **Playlist Creation**

Users can create personalized playlists with customizable details shared musical space

* **Robust Backend Infrastructure:**

Node.js serves as the backbone, providing a scalable and efficient server-side scripting environment, ensuring reliable communication between the client and server.

* **Flexible Data Storage with MongoDB:**

MongoDB provides flexibility in storing and retrieving diverse data related to songs, playlists, and user profiles.

In summary, the Music Player Web Application successfully achieves its objectives, providing users with a comprehensive and engaging music streaming platform. The combination of React.js, Tailwind CSS, Node.js, and MongoDB tailored with efficient declarative approach gives us a product enhancing user experience in their journey of music listening. Ongoing improvements and enhancements are planned with React.js facilitating a responsive and continuously improving front-end experiences.

1. **REFRENCES**

* Tailwind Install: <https://tailwindcss.com/docs/installation>
* Tailwind CSS: <https://tailwindcss.com/docs>
* React.js: <https://reactjs.org/docs/getting-started.html>
* Node.js Install: <https://nodejs.org/en>
* Node.js: <https://nodejs.org/en/docs>
* MongoDB: <https://docs.mongodb.com/>