**PROJECT DOCUMENTATION**

**RHYTHMIC TUNES**

1. **INTRODUCTION**

**PROJECT TITLE : RHYTHMIC TUNES**

* **TEAM ID:NM2025TMID39835**
* **TEAM LEARDER:**

**THAMIZHMATHI .N([thamizhmathin262006@gmail.com](mailto:thamizhmathin262006@gmail.com))**

* **TEAM MEMBERS:**
* ABIRAMI.K( **abik6685@gmail.com)**
* ABINAYA RANI.D( **d.k.abinayarani20052006@gmail.com)**
* HEMASRI.M (**hemasri05012007@gmail.com)**

**2.PROJECT OVERVIEW**

* **PURPOSE:**

* **Rhythmic Tunes is a music-based web application designed to provide users with an engaging and interactive experience. It allows users to listen to rhythmic sound patterns, create playlists, and explore various tunes with a simple and user-friendly interface.**

* **FEATURES**

* **Music streaming with rhythmic effects**
* **Playlist management**
* **Interactive and responsive user interface**
* **Admin panel and users**

**3.ARCHITECTURE**

* **The architecture of the Rhythmic Tunes project follows a three-tier structure consisting of the Frontend, Backend, and Database layers. Each layer is responsible for specific tasks, ensuring modularity and scalability.**

* **FRONTEND**

* **Built using React.js with Bootstrap/Material Ul for styling.**
* **Provides an interactive and responsive user interface.**
* **Handles user actions like login, registration, song browsing, and playlist management.**
* **Communicates with backend APIs via HTTP reque Axios/Fetch)**

* **BACKEND(SERVER-SLIDE)**

* **Developed with Node.js and Express.js.**
* **Manages all application logic and routes.**
* **Provides RESTful API endpoints for user authentication, playlist management, and music streaming.**
* **Implements JWT-based authentication for secure login.**
* **Middleware protects private routes and ensures data security.**

* **DATABASE(DATA LAYER)**

* **MongoDB is used as the primary database.**
* **Stores user details, playlists, songs, and activity logs.**

**4.SETUP INSTRUCTIONS**

* **PREREQUISITES**

* **Before running the project, make sure you have installed the following software/tools:**
* **Node.js (for running backend and frontend build)**
* **MongoDB (for storing data such as users, playlists, songs)**
* **Git (for cloning the repository)**
* **React.js (frontend framework)**
* **E**xpress**.js & Mongoose (backend framework and database connection)**
* **Visual Studio Code (or any code editor)**

**5.FOLDER STRUCTURE**

* **The Rhythmic Tunes project is organized into two main parts: the client (frontend) and the server (backend).**

**Rhythmic-Tunes/**

**│**

**├── client/ # React frontend**

**│ ├── public/ # Static files (index.html, images, icons)**

**│ ├── src/ # Main source code**

**│ │ ├── components/ # Reusable UI components (Navbar, Footer, Player, etc.)**

**│ │ ├── pages/ # Application pages (Home, Login, Register, Playlist, etc.)**

**│ │ ├── assets/ # CSS, images, and static assets**

**│ │ ├── App.js # Main app file**

**│ │ └── index.js # Entry point**

**│ └── package.json # Frontend dependencies and scripts**

**│**

**├── server/ # Node.js backend**

**│ ├── routes/ # API routes (user.js, playlist.js, songs.js, etc.)**

**│ ├── controllers/ # Request handling logic**

**│ ├── models/ # MongoDB models (User.js, Playlist.js, Song.js)**

**│ ├── middleware/ # Auth middleware (JWT verification, etc.)**

**│ ├── config/ # Database configuration**

**│ ├── server.js # Entry point of backend**

**│ └── package.json # Backend dependencies and scripts**

**│**

**├── .gitignore # Files ignored by Git**

**├── README.md # Project description**

**└── package.json # Root project file (if using combined setup)**

**6.RUNNING THE APPLICATION**

* **FRONTEND:**

**Bash**

**Copy code**

**Cd client**

**Npm start**

* **This will start the React.js frontend of the Rhythmic Tunes project. By default, it runs at <http://localhost:3000> and provides the user interface for browsing and playing music.**

* **BACKEND**

**Bash**

**Copy code**

**cd server**

**npm**

**start**

* **This will start the Node.js/Express backend of Rhythmic Tunes. It manages APIs, authentication, and database operations. By default, it runs at <http://localhost:5000>.**

* **ACCESS**

* **Once both servers are running, open a browser and visit http://localhost:3000 to access the complete Rhythmic Tunes application.**

**7.API DOCUMENTATION**

* **MUSIC APIs**

* **GET/api/music/all →Fetch all available songs from the database.**
* **GET/api/music/:id → Get details of a specific song by ID.**
* **POST/api/music/upload→ Upload a new song (admin access).**

* **PLAYLIST APIs**

* **POST/api/playlist/create Create a new playlist.**
* **GET/api/playlist/:id → View Songs in a playlist.**
* **PUT/api/playlist/:id/add → Add a song to a playlist.**
* **DELETE /api/playlist/:id/remove → Remove a song from a playlist.**

* **CHAT APIs (for community interaction)**

* **POST /api/chat/send → Send a new chat message.**
* **GET /api/chat/:roomId → Fetch chat history for a room or user.**

**8. AUTHENTICATION**

**In a full-stack application, authentication is typically handled by middleware on the backend. This middleware acts as a gatekeeper for the API, intercepting incoming requests to verify that a user is logged in before granting access to protected routes, such as managing a personal playlist.**

**9.USER INTERFACE**

**The User Interface (UI) of Rhythmic Tunes is designed to be simple, interactive, and user-friendly, allowing smooth navigation through the application’s features. It includes the following main components.**

* **LANDING PAGE**
* **The entry point of the application, where users are introduced to Rhythmic Tunes and can choose to log in or register.**

* **USER DASHBOARD**
* **A personalized space for users to explore available songs, create and manage playlists, and view recommendations.**

* **ADMIN PANEL**
* **A separate interface for administrators to upload new songs, manage users, and monitor the system.**

* **MUSI PLAYER PAGE**
* **A dedicated Player interface where users can play, pause, skip, and control songs, along with viewing lyrics or album art.**

* **PLAYLIST & FAVOURITES PAGE**
* **Allows users to view, create, and organize playlists and save favorite tracks for quick access.**

* **CHAT /COMMUNITY PAGE**
* **Provides users with the ability to chat, share thoughts, and engage in music-related discussions**

**10.TESTING**

* **Testing for the Rhythmic Tunes project was performed to ensure that all frontend features function correctly and that the application runs smoothly. The process focused on manual testing of the user interface and its interaction with the mock `json-server’ backend.**

* **USER INT (UI) Testing**

* **The main pages, including the \*\*Home Page\*\*, \*\*Music Player\*\*, and \*\*Playlist pages\*\*, were tested to ensure they render correctly and are responsive on different screen sizes.**
* **Interactive elements like buttons, search bars, and player controls were checked to confirm they respond correctly to user input**
* **FUNCTIONAL TESTING**
* **Song Playback Verified that clicking a song fetches its data and begins playback, and that the player controls (play, pause, skip) work as expected.**

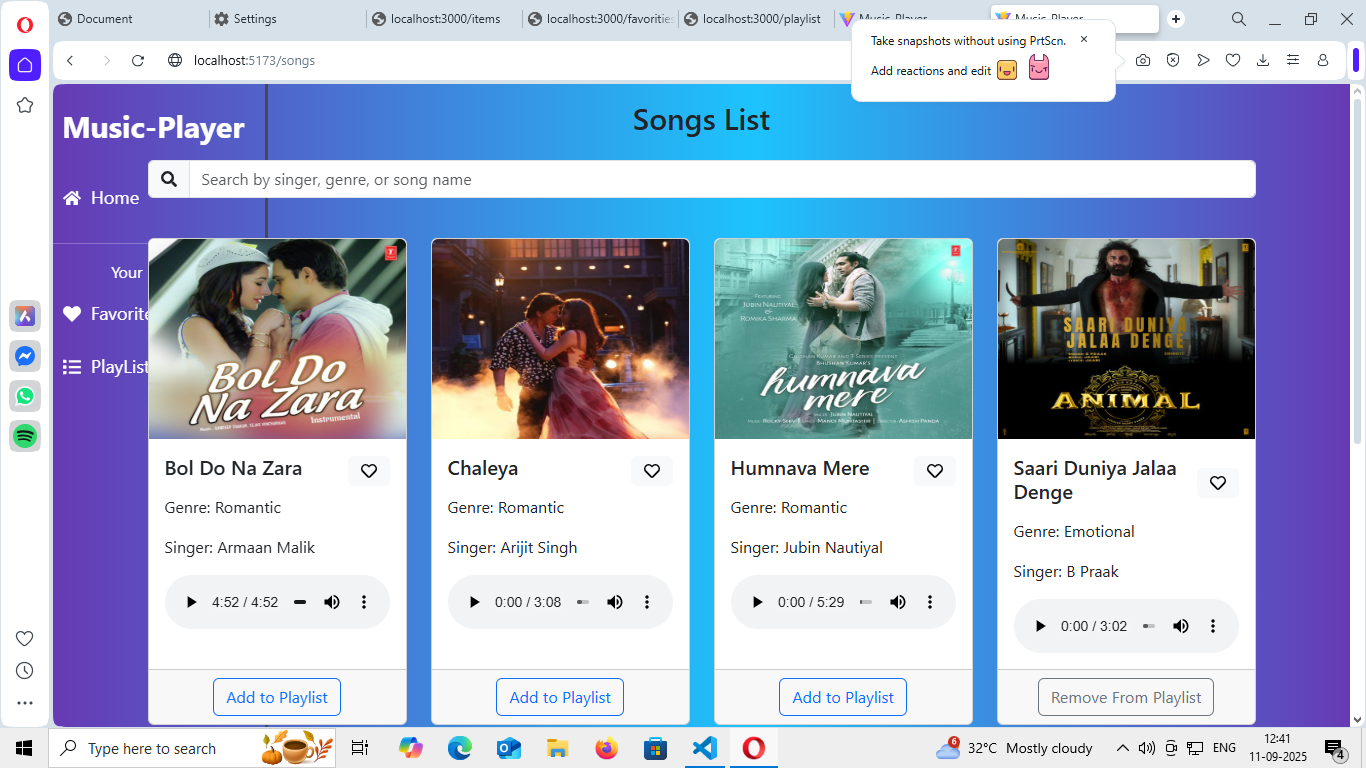
* **PLAYLIST MANAGEMENT**
* **Confirmed that users can successfully create a new playlist, add songs to it, and delete it. The UI was checked to ensure it updates correctly after each of these actions.**

* **API Integration Testing**
* **The interaction between the React frontend and the `json-server` backend was tested to ensure seamless communication.**
* **Verified that `GET` requests correctly fetch data for songs and playlists, and that `POST` and `DELETE` requests successfully update the `db.json` file.**

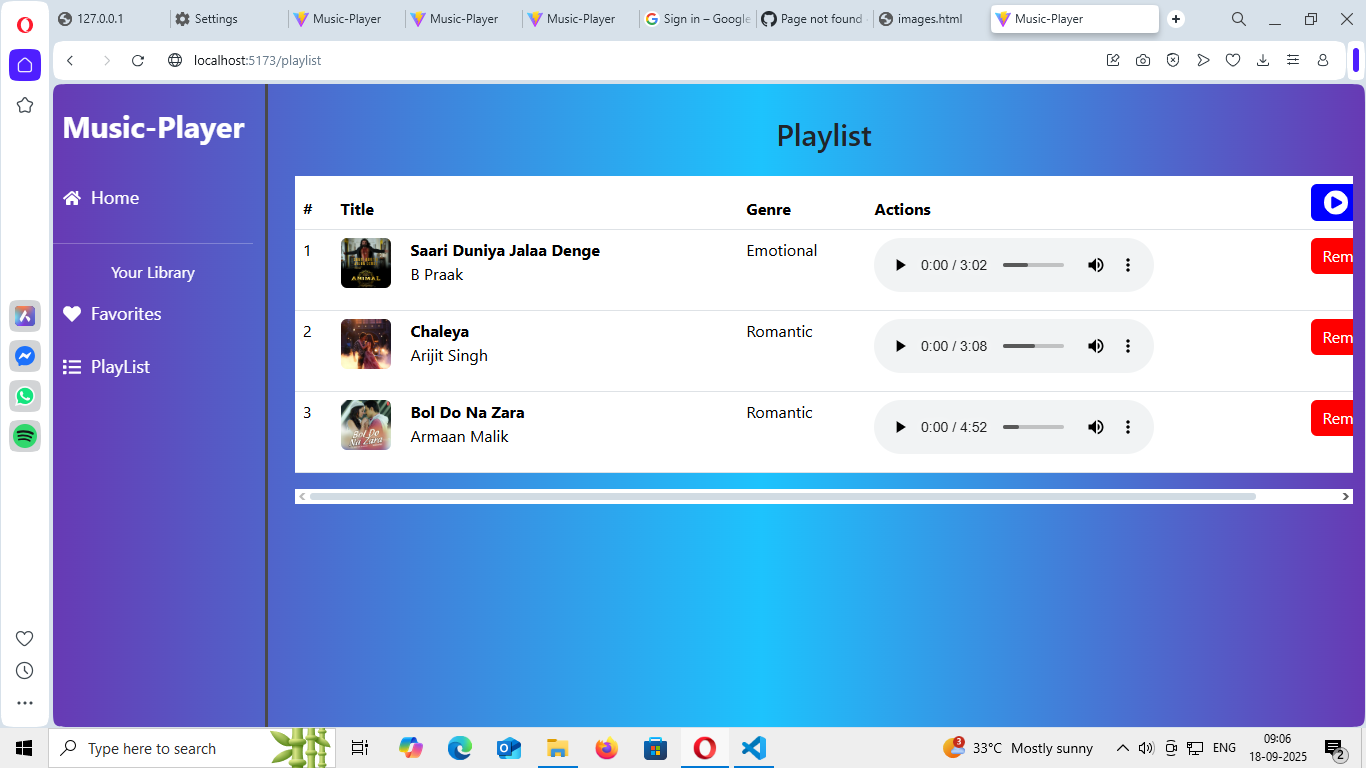
* **ERRORS HANDLING**
* **Basic error handling was checked, such as how the application behaves if the `json-server` is not running. This ensures the application does not crash and remains stable.**

1. **SCREENSHOT**

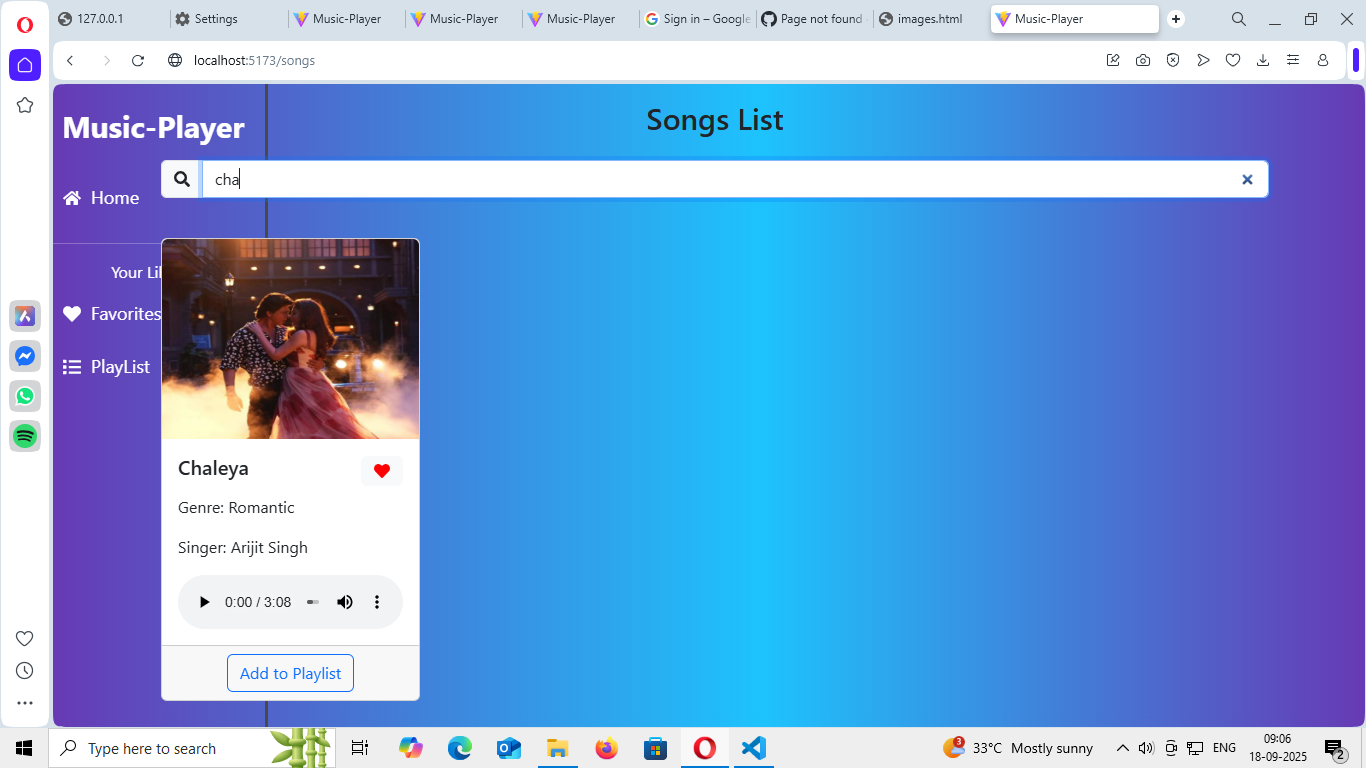
**HOME PAGE**

****

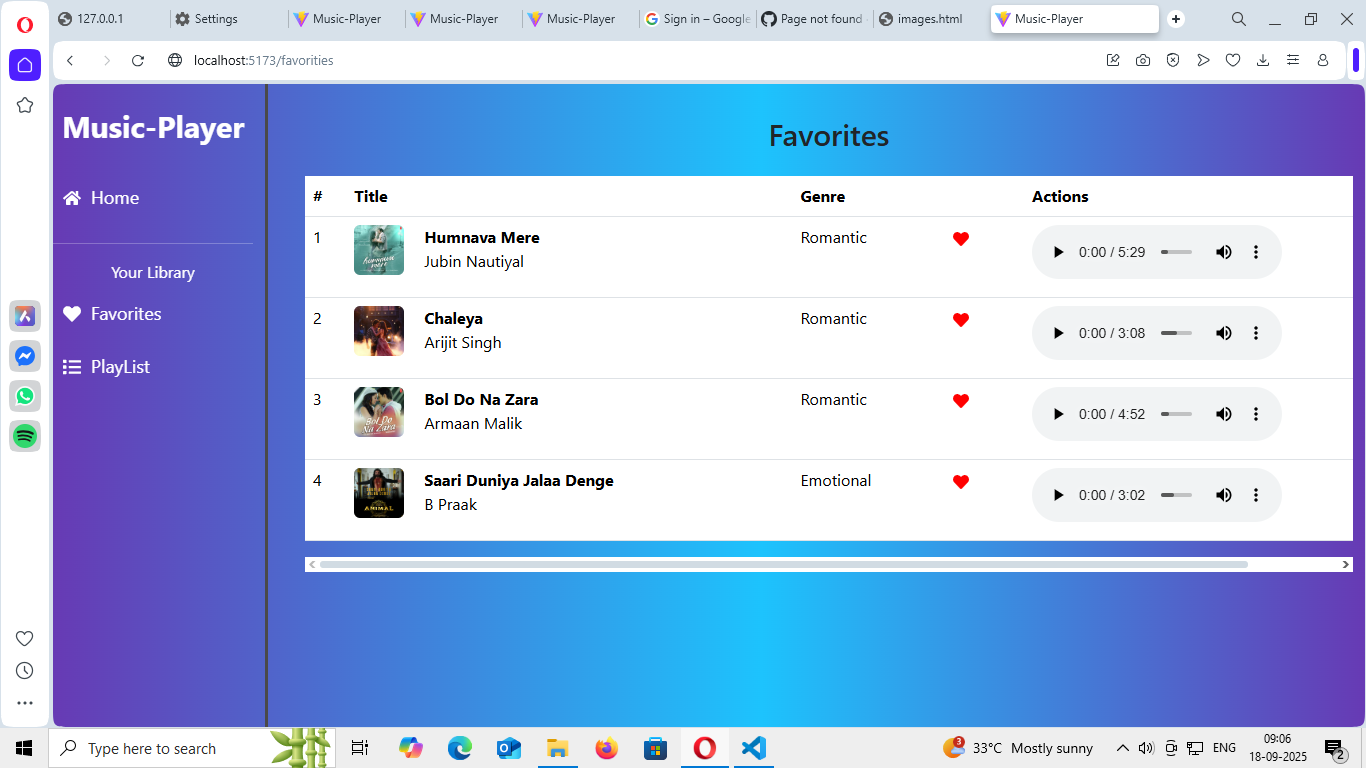
**PLAYLIST**

****

**SEARCH**

****

**FAVOURITES**

****

**12. KNOWN ISSUES**

* **The current implementation of Rhythmic Tunes relies on `json-server` for**

**its backend, which comes with certain limitations inherent to a mock server**

**environment:**

* **No Real-Time Database**
* **Data is stored in a `db.json` file. Any changes made (like creating a playlist)**

**will persist only as long as the server is running and are not stored in a**

**permanent database. If the `db.json` file is reset or replaced, all changes**

**will be lost.**

* **No User Authentication**
* **The project lacks a true user authentication system. There is no**

**functionality for user registration, login, or protected routes. All API**

**endpoints are public and accessible to anyone running the application.**

* **Limited API Logic**
* **`json-server` provides basic CRUD (Create, Read, Update, Delete)**

**operations but cannot handle complex business logic, such as validating**

**user permissions for uploading or deleting songs.**

**13.FUTURE ENHANCEMENT**

* **To evolve Rhythmic Tunes into a full-featured, production-ready**

**application, the following enhancements are planned:**

* **Develop a Full Backend**
* **Replace `json-server` with a robust backend framework like \*\*Node.js with**

**Express.js\*\*. This will allow for the implementation of custom business logic and more complex API endpoints.**

* **Integrate a Database**
* **Implement a scalable database system such as \*\*MongoDB\*\* to**

**permanently store user data, songs, and playlists.**

* **Implement User Authentication**

**Introduce a complete authentication system using \*\*JSON Web Tokens**

**(JWT)\*\*. This will include creating secure endpoints for user registration,**

**login, and profile management, ensuring that users can only access and**

**modify their own data.**

* **Admin and User Roles**
* **Create distinct roles for users and administrators. This would allow admins**

**to manage the music library (e.g., upload and delete songs), while standard**

**users would have privileges limited to their own playlists and profiles**.

* **Advanced Player Features**
* **Enhance the music player with features like shuffle, repeat, a playable**

**queue, and volume controls.**

* **Social Features**
* **Add social components, such as the ability to share playlists with other**

**users or see what friends are listening to.**