Rhythmic Tunes: Your Melodic Companion

Indroduction:

- **Project title:** Rhythmic Tunes
- **Team leader:** Madhavan.V (venketmano2004@gmail.com)
- **Team member:**
  - Madhavan.V (venketmano2004@gmail.com)
  - Madhuvanth.K (madhuvanthhhh@gmail.com)
  - Ezhilarasu.G (kgurukguru895@gmail.com)
  - Arun kumar.M (akumar13102004@gmail.com)

Project overview:

- **Purpose**:
  - Music Streaming is designed for a diverse audience, including:
  - Music Enthusiasts: People passionate about enjoying and listening Music Through out there free time to relax themselves.
- **Features:**
  - Song Listings: Display a comprehensive list of available songs with details such as title, artist, genre, and release date.
  - Playlist Creation: Empower users to create personalized playlists, adding and organizing songs based on their preferences.
  - Playback Control: Implement seamless playback control features, allowing users to play, pause, skip, and adjust volume during music playback.
  - Offline Listening: Allow users to download songs for offline listening, enhancing the app's accessibility and convenience.

- Search Functionality: Implement a robust search feature for users to easily find specific songs, artists, or albums within the app.

Architecture:

- **Component-Based:**_The application is built using React's component-based architecture. This means the UI is broken down into reusable and independent components.

  - **Client-Side Rendering (CSR):**
    - React.js is primarily used for client-side rendering. The browser fetches the initial HTML, and React takes over to render and update the UI dynamically.
  - **Frontend-Centric:**
    - The provided information focuses heavily on the frontend implementation using React.js.
    - The backend is simulated using a JSON server.
  - **API Interactions:**
    - Axios is used to make HTTP requests to the backend (JSON server) for fetching and manipulating data (songs, favorites, playlists).

- **Component Structure:**Based on the provided code, here's a likely component structure:

  - **App (src/App.js):**
    - The root component.
    - Sets up routing using BrowserRouter, Routes, and Route.
    - Contains the Sidebar and the main content area.
    - Manages the overall layout.
  - **Songs (likely a component):**
    - Displays the list of songs.
    - Handles search functionality.
    - Manages audio playback.
    - Handles adding/removing songs from the wishlist and playlist.
    - Fetches song data from the backend.
  - **Favorities (likely a component):**
    - Displays the user's favorite songs.
    - Interacts with the backend to get the users favorited songs.
  - **Playlist (likely a component):**
    - Displays the user's playlists.
    - Interacts with the backend to get the users playlist songs.

- **Sidebar (likely a component):**
  - Provides navigation links to different sections of the application (Songs, Favorites, Playlist).
- **Card(likely a component):**
  - Displays the individual song information.

- **State Management:**

  - **Local Component State:**
    - The useState hook is used extensively to manage local component state.
    - Examples:
      - items: Stores the list of songs.
      - wishlist: stores the list of favorited songs.
      - playlist: stores the list of playlist songs.
      - searchTerm: Stores the search query.
      - currentlyPlaying: Manages the currently playing audio.
  - **No Centralized State Management:**
    - Based on the code, there's no evidence of using a centralized state management library like Redux or Context API.
    - State is managed within individual components.
  - **Data Fetching State:**
    - The state variables items, wishlist, and playlist are used to store the data that is fetched from the server.

- **Routing:**

  - **React Router DOM:**
    - The application uses React Router DOM for client-side routing.
    - BrowserRouter: Provides the routing context.
    - Routes: Defines the routes.
    - Route: Specifies the path and the component to render.
  - **Routes:**
    - /: Renders the Songs component.
    - /favorities: Renders the Favorities component.
    - /playlist: Renders the Playlist component.

Setup Instructions:

- **Prerequisites:**

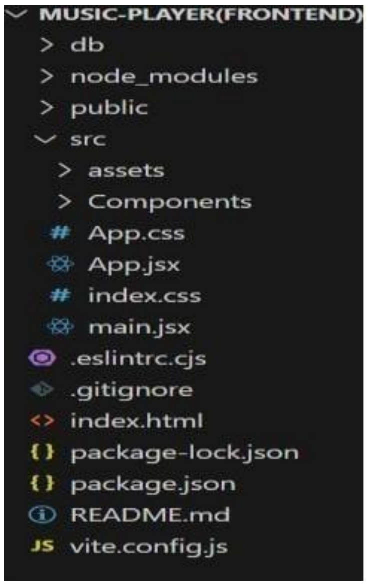
- **Node.js and npm (Node Package Manager):**
  - Download and install Node.js from: https://nodejs.org/en/download/
  - npm is included with Node.js.
- **Git (Optional but Recommended):**
  - Download and install Git from: https://git-scm.com/downloads
- **Code Editor (Recommended):**
  - Visual Studio Code: https://code.visualstudio.com/download
  - Sublime Text: https://www.sublimetext.com/download
  - WebStorm: https://www.jetbrains.com/webstorm/download

- **Install required tools and software**:
  - Installation of required tools:
  - Open the project folder to install necessary tools In this project, we use:
    - ✓ React Js
    - ✓ React Router Dom
    - ✓ React Icons
    - ✓ Bootstrap/tailwind css
    - ✓ Axios

Installation:

- Clone the repository
- Navigate to the project folder
- Install dependencies with npm install
- Start the development server with npm start
- Open http://localhost:5173 in a browser

Folder Structure:



```
MUSIC-PLAYER(FRONTEND)
  > db
  > node_modules
  > public
  ∨ src
    > assets
    > Components
      App.css
      App.jsx
      index.css
      main.jsx
    .eslintrc.cjs
    .gitignore
    index.html
    package-lock.json
    package.json
    README.md
    vite.config.js
```

Running the Application:

- **Node.js and npm:**
  - Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the local environment. It provides a scalable and efficient platform for building network applications.
  - InstallNode.js and npm on your development machine, as they are required to run JavaScript on the server-side.
    - Download: https://nodejs.org/en/download/
    - Installation instructions: https://nodejs.org/en/download/package-manager/

- **React.js:**
- React.js is a popular JavaScript library for building user interfaces.

- It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. Install React.js, a JavaScript library for building user interfaces.

1.Create a new React app:

```
npm create vite@latest
```

Enter and then type project-name and select preferred frameworks and then enter

2. Navigate to the project directory:

```
cd project-name

npm install
```

3. Running the React App:

With the React app created, you can now start the development server and see your React application in action.

4.Start the development server:

```
npm run dev
```

This command launches the development server, and you can access your React app at http://localhost:5173 in your web browser.

Component Documentation:

Absolutely! Let's create component documentation for the key components in the Rythimic Tunes React application based on the provided description.

1. App.js (Main Application Component)

- **Purpose:**
  - The root component of the application.
  - Sets up routing using `react-router-dom`.
  - Provides the overall layout and structure of the application.
- **Functionality:**
  - Imports necessary dependencies (React, `react-router-dom`, Bootstrap CSS, custom CSS).
  - Uses `BrowserRouter` to enable routing.
  - Defines routes for different sections of the application:
    - `/`: Renders the `Songs` component.
    - `/favorites`: Renders the `Favorites` component.
    - `/playlist`: Renders the `Playlist` component.
  - Renders the `Sidebar` component.
- **Dependencies:**
  - React
  - `react-router-dom`
  - Bootstrap CSS
  - Custom CSS (`./App.css`)
  - Sidebar.js Component
  - Songs.js Component
  - Favorites.js Component

    - Playlist.js Component
- **State:**
    - None.
- **Props:**
    - None.

2. Songs.js (Song Listing and Playback Component)

- **Purpose:**
    - Displays a list of available songs.
    - Provides search functionality.
    - Handles audio playback.
    - Manages adding/removing songs from the wishlist and playlist.
- **Functionality:**
    - Fetches song data from a JSON server (e.g., `http://localhost:3000/items`).
    - Fetches wishlist and playlist data.
    - Manages audio playback, ensuring only one song plays at a time.
    - Allows users to add/remove songs from the wishlist and playlist.
    - Filters songs based on the search term.
    - Renders a list of songs with details (title, artist, genre, image, audio player).
    - Handles search functionality.
- **Dependencies:**
    - React (`useState`, `useEffect`)
    - `axios`
    - React Bootstrap (`InputGroup`, `Form`, `Card`, etc.)
    - React Icons (`FaHeart`, `FaRegHeart`, `FaSearch`)
- **State:**
    - `items`: Array of all songs.
    - `wishlist`: Array of favorite songs.
    - `playlist`: Array of songs in the playlist.
    - `currentlyPlaying`: Audio element that is currently playing.
    - `searchTerm`: Search term entered by the user.
- **Props:**
    - None.

3. Favorites.js (Favorites/Wishlist Component)

- **Purpose:**
    - Displays the user's favorite songs (wishlist).
    - Allows removal of songs from the favorites list.
- **Functionality:**
    - Fetches the wishlist data from the JSON server.
    - Displays the list of favorite songs.
    - Provides functionality to remove songs from the wishlist.
- **Dependencies:**

    - React (useState, useEffect)
    - Axios
- **State:**
    - Wishlist: array of favorite songs.
- **Props:**
    - None.

4. Playlist.js (Playlist Component)

- **Purpose:**
    - Displays the user's playlist.
    - Allows removal of songs from the playlist.
- **Functionality:**
    - Fetches the playlist data from the JSON server.
    - Displays the list of songs in the playlist.
    - Provides functionality to remove songs from the playlist.
- **Dependencies:**
    - React (useState, useEffect)
    - Axios
- **State:**
    - Playlist: array of songs in the playlist.
- **Props:**
    - None.

5. Sidebar.js (Navigation Sidebar)

- **Purpose:**
    - Provides navigation links to different sections of the application.
- **Functionality:**
    - Renders navigation links to the Songs, Favorites, and Playlist components.
- **Dependencies:**
    - React
    - React Router Dom (Link)
- **State:**
    - None.
- **Props:**
    - None.

State Management:

- **Key State Variables and Their Purposes:**

1. **items (in Songs.js):**

- **Purpose:** Holds an array of all song items fetched from the backend (e.g., http://localhost:3000/items).
- **Management:**
  - Initialized using useState([]).
  - Updated asynchronously using useEffect and axios.get when the component mounts.
  - Represents the main source of song data.
- This is the main list of songs that will be displayed.

2. **wishlist (in Songs.js and Favorites.js):**
   - **Purpose:** Stores an array of song items that the user has marked as favorites.
   - **Management:**
     - Initialized using useState([]).
     - Updated asynchronously using useEffect and axios.get to fetch favorites.
     - Updated with axios.post, and axios.delete when items are added or removed.
     - Used to display the user's favorite songs.
   - This state is shared between the Songs and Favorites components.

3. **playlist (in Songs.js and Playlist.js):**
   - **Purpose:** Stores an array of song items that the user has added to their playlist.
   - **Management:**
     - Initialized using useState([]).
     - Updated asynchronously using useEffect and axios.get to fetch the playlist.
     - Updated with axios.post, and axios.delete when items are added or removed.
     - Used to display the user's playlist.
   - This state is shared between the Songs and Playlist components.

4. **currentlyPlaying (in Songs.js):**
   - **Purpose:** Keeps track of the audio element that is currently playing.
   - **Management:**

        - Initialized using useState(null).
        - Updated when a new audio element starts playing, pausing the previous one.
        - Used to ensure that only one song plays at a time.

5. **searchTerm (in Songs.js):**
    - **Purpose:** Stores the search term entered by the user.

    - **Management:**
        - Initialized using useState('').
        - Updated when the user types in the search input field.
        - Used to filter the items array and display search results.

- ## State Management Patterns:

    - **Component-Level State:**
        - Most of the state is managed within the Songs, Favorites, and Playlist components using the useState hook.
        - This approach is suitable for managing state that is specific to a single component.
    - **Asynchronous State Updates:**
        - useEffect is used to fetch data from the backend asynchronously, and the state is updated when the data is received.
        - This pattern is essential for handling data fetching in React components.
    - **State Synchronization (Implicit):**
        - The wishlist and playlist states are synchronized between the Songs component and their respective display components (Favorites and Playlist) by fetching the same data from the backend.
        - When a change is made in one component (e.g., adding to the wishlist in Songs), the change is reflected in the other component (Favorites) after the next data fetch.
    - **Controlled Components:**
        - The search input field is a controlled component, where the input's value is bound to the searchTerm state.

- **Potential Improvements:**
    - **Context API or Redux/Zustand:**
        - For larger applications with more complex state management needs, consider using React's Context API or a state management library like Redux or Zustand.
        - These tools can help to manage shared state more effectively and avoid prop drilling.
    - **Error Handling:**
        - Expand error handling for API calls. Displaying user friendly error messages would improve user experience.
    - **Optimistic Updates:**
        - For a more responsive user experience, consider implementing optimistic updates when adding or removing items from the wishlist or playlist. This involves updating the UI immediately and then reconciling with the server response.
    - **Caching:**
        - Implement caching of API responses to reduce the number of requests and improve performance.

User interface:

Overall Layout:

- The application uses a layout with a sidebar for navigation and a main content area for displaying songs, favorites, or playlists.
- The application is responsive, so it will adjust to different screen sizes.

Key UI Components and Features:

1. **Sidebar:**
    - Provides navigation to the main sections of the application:
        - "Songs" (Displays the list of all available songs)
        - "Favorites" (Displays the user's favorite songs)
        - "Playlist" (Displays the user's custom playlist)
    - Likely uses navigation links (e.g., Link from react-router-dom).
2. **Songs List View (/ route):**
    - **Header:**
        - Displays a heading such as "Songs List".
    - **Search Bar:**

        - Includes an input field for searching songs by title, artist, or genre.
        - Uses a search icon (e.g., FaSearch).
    - **Song Cards:**
        - Displays songs in a grid layout using Bootstrap cards.
        - Each card includes:
            - Song image.
            - Song title.
            - Song genre.
            - Song artist.
            - Audio player with play/pause controls.
            - "Add to Favorites" or "Remove from Favorites" button (heart icon).
            - "Add to Playlist" or "Remove from Playlist" button.
    - **Audio Player:**
        - Embedded audio players within each song card.
        - Play/pause functionality.
        - Volume control would be a desired feature.

3. **Favorites View (/favorites route):**
    - Displays a list of songs that the user has marked as favorites.
    - Similar card layout to the Songs List View.
    - Includes a "Remove from Favorites" button for each song.
4. **Playlist View (/playlist route):**
    - Displays the user's custom playlist.
    - Similar card layout to the Songs List View.
    - Includes a "Remove from Playlist" button for each song.

Styling and Responsiveness:

- The application uses Bootstrap or Tailwind CSS for styling and layout.
- The UI is designed to be responsive, adapting to different screen sizes (desktop, tablet, mobile).
- The application uses react-icons for icons.

1.User Interaction:

- ✓ Users can navigate between sections using the sidebar.
- ✓ Users can search for songs using the search bar.
- ✓ Users can play/pause songs using the audio players.
- ✓ Users can add/remove songs from their favorites and playlists.
- ✓ The user interface is designed to be intuitive and user-friendly.

2.Visual Design:

- ✓ The visual design is likely clean and modern.

- ✓ Uses consistent typography and color schemes.
- ✓ Uses icons to enhance usability.
- ✓ Card based layout is used to display song information.

3.Overall User Experience:

- ✓ The application provides a seamless and enjoyable music streaming experience.
- ✓ Users can easily discover, manage, and listen to their favorite music.
- ✓ The UI is designed to be intuitive and responsive.

Styling:

- **Bootstrap or Tailwind CSS:**
  - ➢ The project leverages either Bootstrap or Tailwind CSS for its styling. This indicates a focus on responsive design and pre-built UI components.
  - ➢ Bootstrap provides a comprehensive set of CSS classes for layout, components, and utilities, while Tailwind CSS offers a utility-first approach.

- ➢ **Custom CSS:**
  - o The project also includes custom CSS (./App.css) for additional styling. This allows for fine-tuning the look and feel of the application beyond the capabilities of the chosen CSS framework.
- ➢ **Inline Styles:**
  - o The code description mentions the use of inline styles (e.g., style={{display:"flex", justifyContent:"flex-end"}}). This is used for specific, localized styling adjustments.

- **Styling Characteristics:**
  - ➢ **Responsive Design:**
    - o The use of Bootstrap or Tailwind CSS implies a strong emphasis on responsiveness. The UI is designed to adapt to various screen sizes, ensuring a consistent experience across desktops, tablets, and mobile devices.
    - o Bootstrap grid system is mentioned in the code description.
  - ➢ **Card-Based Layout:**
    - o The project utilizes a card-based layout for displaying song information. This provides a clean and organized presentation of song details.

    - Bootstrap cards are used.
- **Clean and Modern Design:**
    - The overall design is intended to be clean and modern, focusing on usability and visual appeal.
- **Consistent Typography and Color Schemes:**
    - The project likely maintains consistent typography and color schemes throughout the application, contributing to a cohesive user experience.
- **Iconography:**
    - React Icons (react-icons) are used to enhance the UI with icons. This improves usability by providing visual cues for actions and elements.
    - Heart icons for favorites, and search icons are used.
- **Search Input Styling:**
    - The search input field is styled with the class name "search-input".
- **Header Styling:**
    - Headers are styled with text classes such as "text-3xl font-semibold mb-4 text-center".

- **Styling Implementation:**
    - The project likely uses a combination of:
        - Pre-defined classes from Bootstrap or Tailwind CSS for layout and common UI elements.
        - Custom CSS rules to override or extend the framework's styles.
        - Inline styles for specific, localized styling adjustments.

- **Potential Styling Enhancements:**
    - **Theme Customization:**
        - Consider implementing a theming system to allow users to customize the appearance of the application.
    - **Accessibility:**
        - Ensure that the styling adheres to accessibility guidelines, such as providing sufficient color contrast and keyboard navigation support.
    - **Animations and Transitions:**
        - Adding subtle animations and transitions can enhance the user experience and make the application feel more interactive.
    - **Consistent Spacing and Alignment:**
        - Maintain consistent spacing and alignment throughout the UI to create a polished and professional look.

Testing:

- **Testing Strategies:**
  - **Unit Testing:**
    - Focus: Testing individual components and functions in isolation.
    - Tools: Jest, React Testing Library.
    - What to Test:
      - Component rendering: Verify that components render correctly with expected props and state.
      - Functionality: Test functions like `addToWishlist`, `removeFromWishlist`, `addToPlaylist`, `removeFromPlaylist`, and filtering logic.
      - Event handlers: Ensure that event handlers (e.g., `handlePlay`, search input changes) behave as expected.
      - Test the audio player functionality.
    - Benefits: Provides quick feedback on code changes, ensures that individual parts of the application work correctly.
  - **Integration Testing:**
    - Focus: Testing the interactions between different components and modules.
    - Tools: React Testing Library, Jest.
    - What to Test:
      - Component interactions: Verify that components communicate with each other correctly (e.g., `Songs` component interacting with `Favorites` and `Playlist`).
      - Data flow: Test the flow of data between components and how state changes propagate.
      - API interactions: Test how components interact with the backend API (e.g., fetching song data, updating wishlist and playlist).
      - Routing: test that the react router is working as expected.
    - Benefits: Ensures that different parts of the application work together seamlessly.
  - **End-to-End (E2E) Testing:**
    - Focus: Testing the entire application from the user's perspective, simulating real user interactions.
    - Tools: Cypress, Playwright.
    - What to Test:
      - User workflows: Test complete user scenarios, such as searching for a song, adding it to the playlist, and playing it.
      - Navigation: Verify that navigation between different sections of the application works correctly.
      - UI interactions: Test user interactions with UI elements, such as buttons, input fields, and audio players.
      - Responsiveness: Test the application on different screen sizes and browsers.
    - Benefits: Provides confidence that the application works correctly in a real-world environment.
  - **API Testing:**

    - Focus: Testing the backend API endpoints that the application relies on.
    - Tools: Jest, Supertest, Postman.
    - What to Test:
        - Endpoint functionality: Verify that API endpoints return the correct data and status codes.
        - Data validation: Test that the API handles different types of data correctly.
        - Error handling: Ensure that the API handles errors gracefully.
    - Benefits: Ensures that the backend API is reliable and functional.

- **Specific Testing Considerations:**

  - **Audio Playback:**
    - Test that audio playback starts and stops correctly.
    - Verify that only one audio element plays at a time.
    - Test volume control (if implemented).
  - **Data Fetching:**
    - Test that song data, wishlist data, and playlist data are fetched correctly from the backend.
    - Handle error cases when data fetching fails.
  - **State Management:**
    - Verify that state variables (e.g., items, wishlist, playlist, searchTerm) are updated correctly.
    - Test the synchronization of shared state between components.
  - **Search Functionality:**
    - Test that search results are filtered correctly based on the search term.
    - Handle cases with no search results.
  - **Wishlist and Playlist:**
    - Test adding and removing songs from the wishlist and playlist.
    - Verify that the wishlist and playlist are updated correctly in the UI.
  - **Responsive Design:**
    - Test the application on different screen sizes to ensure that the layout and UI elements are responsive.

- **Testing Tools:**

  - **Jest:** A popular JavaScript testing framework.
  - **React Testing Library:** A library for testing React components.
  - **Cypress/Playwright:** End-to-end testing frameworks.
  - **Supertest:** A library for testing HTTP APIs.
  - **Postman:** A tool for API testing.

Future enchancement:

1.Enhanced Audio Features:

- **Volume Control:**
  - Implement a volume slider for each audio player to allow users to adjust the playback volume.
- **Audio Equalizer:**
  - Add an audio equalizer to allow users to customize the sound output based on their preferences.
- **Gapless Playback:**
  - Implement gapless playback to ensure a smooth transition between songs.
- **Crossfade:**
  - Add a crossfade feature to smoothly transition between songs.
- **Playback Speed Control:**
  - Allow users to adjust the playback speed of songs.

2. Offline Listening:

- **Download Functionality:**
  - Implement a download feature to allow users to download songs for offline listening.
  - Manage downloaded files and storage.
- **Offline Mode:**
  - Create an offline mode that allows users to access and play downloaded songs without an internet connection.

3. Playlist Enhancements:

- **Playlist Sorting and Filtering:**
  - Allow users to sort and filter their playlists by title, artist, genre, or other criteria.
- **Collaborative Playlists:**
  - Implement collaborative playlists that allow multiple users to add and manage songs.
- **Smart Playlists:**
  - Develop smart playlists that automatically generate playlists based on user listening habits or preferences.
- **Playlist Cover Images:**
  - Allow users to add cover images to their playlists.
- **Drag and Drop Playlist reordering:**
  - Allow users to drag and drop songs within a playlist to reorder them.

4. Social Features:

- **User Profiles:**
  - Implement user profiles that allow users to customize their profile information and share their favorite songs and playlists.
- **Social Sharing:**
  - Allow users to share songs and playlists on social media platforms.
- **Follow Users:**
  - Allow users to follow other users and see their listening activity.
- **Comments and Ratings:**
  - Allow users to comment on songs and rate them.

5. Search and Discovery:

- **Advanced Search Filters:**
  - Implement advanced search filters to allow users to refine their search results by genre, release date, or other criteria.
- **Recommendations:**
  - Develop a recommendation system that suggests songs and playlists based on user listening history.
- **Artist and Album Pages:**
  - Create dedicated pages for artists and albums with detailed information and related content.
- **Genre Browsing:**
  - Improve genre browsing and add sub genre browsing.

6. User Interface and Experience:

- **Theme Customization:**
  - Allow users to customize the application's theme and appearance.
- **Accessibility Improvements:**
  - Enhance the application's accessibility to ensure that it is usable by all users.
- **Improved Mobile Experience:**
  - Optimize the application for mobile devices and consider developing a native mobile app.
- **Add Visualizations:**
  - Add audio visualizations to the audio player.

7. Backend and Performance:

- **Caching:**
  - Implement caching to improve application performance and reduce server load.

- **Scalability:**
    - Ensure that the backend infrastructure is scalable to handle a growing number of users.
- **Data Analytics:**
    - Implement data analytics to track user listening habits and improve recommendations.

8. Integrations:

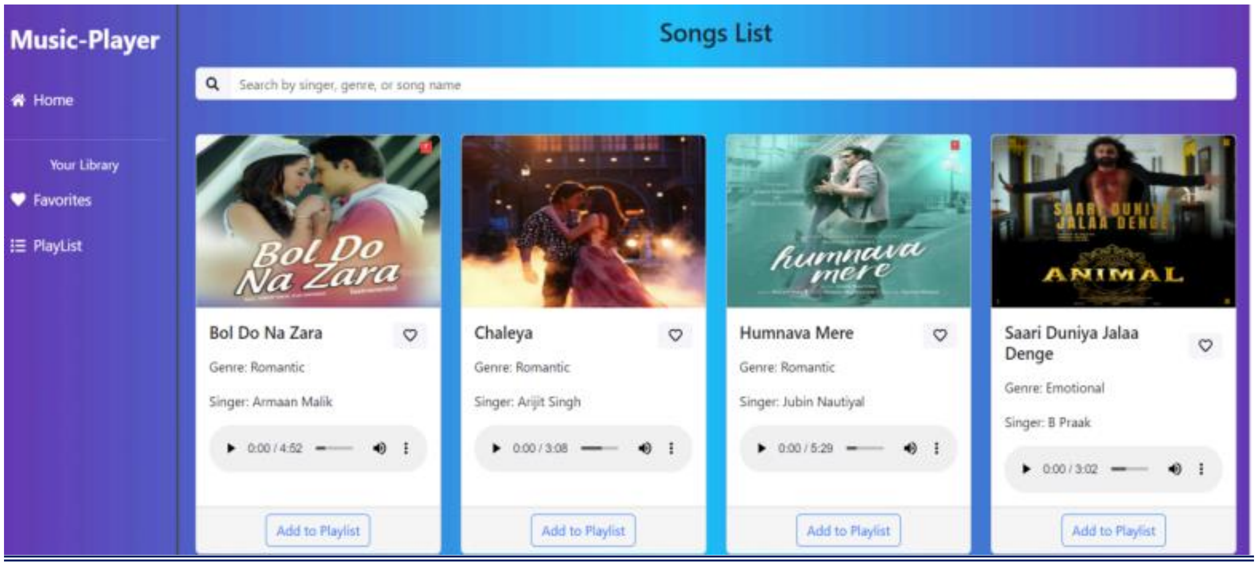
- **Integrate with Third-Party Music Services:**
    - Allow users to import playlists from other music streaming services.
- **Integrate with Smart Speakers:**
    - Allow users to control playback on smart speakers.

Project Execution:

- After completing the code, run the react application by using the command “npm start” or “npm run dev” if you are using vite.js
- And the Open new Terminal type this command “json-server --watch ./db/db.json” to start the json server too.
- After that launch the Rythimic Tunes.
- Here are some of the screenshots of the application.

Screenshots of the project:

- **Hero component:**

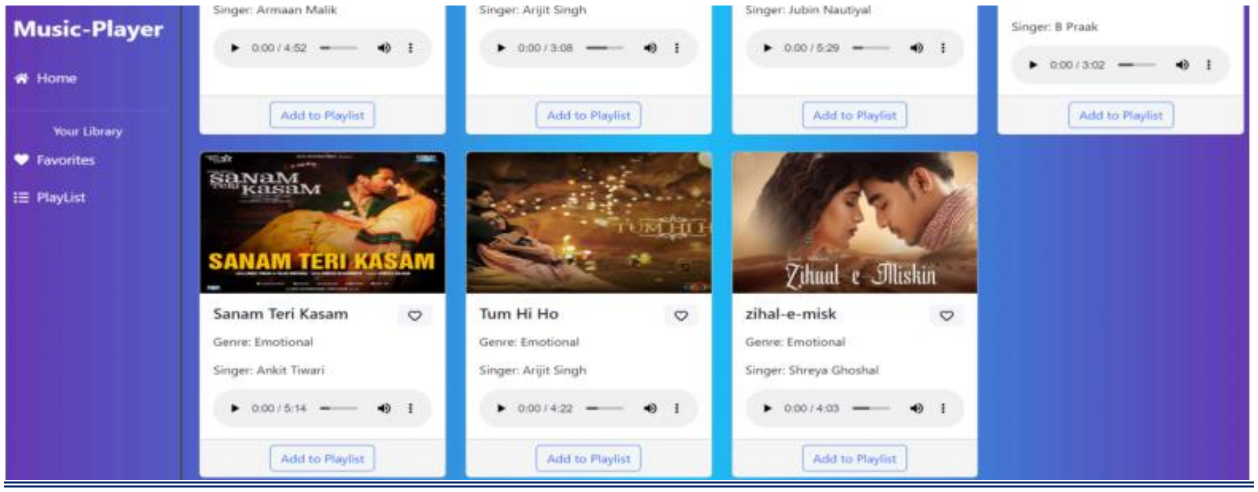


Bol Do Na Zara
Genre: Romantic
Singer: Armaan Malik
0:00 / 4:52

Chaleya
Genre: Romantic
Singer: Arijit Singh
0:00 / 3:08

Humnava Mere
Genre: Romantic
Singer: Jubin Nautiyal
0:00 / 5:29

Saari Duniya Jalaa Denge
Genre: Emotional
Singer: B Praak
0:00 / 3:02



Singer: Armaan Malik
0:00 / 4:52

Singer: Arijit Singh
0:00 / 3:08

Singer: Jubin Nautiyal
0:00 / 5:29

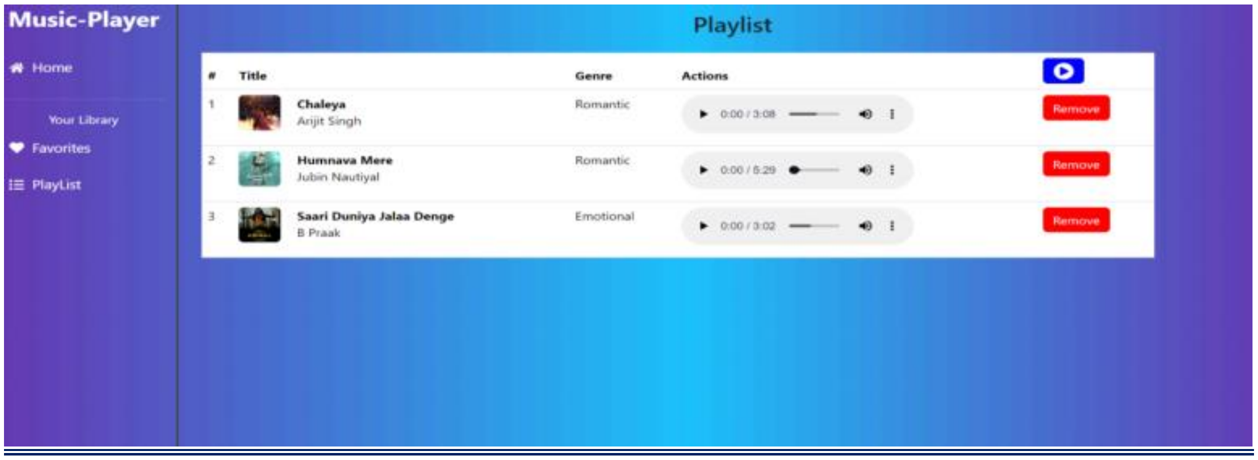
Singer: B Praak
0:00 / 3:02

Sanam Teri Kasam
Genre: Emotional
Singer: Ankit Tiwari
0:00 / 5:14

Tum Hi Ho
Genre: Emotional
Singer: Arijit Singh
0:00 / 4:22

zihal-e-misk
Genre: Emotional
Singer: Shreya Ghoshal
0:00 / 4:03

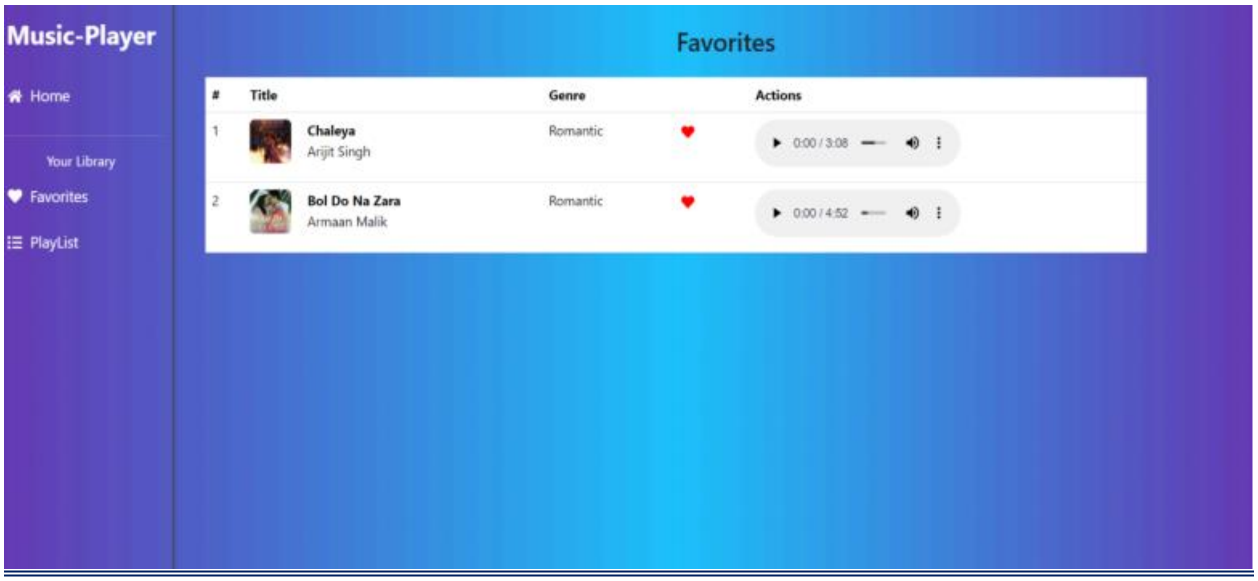
Playlist:



Playlist

#	Title	Genre	Actions	
1	Chaleya Arijit Singh	Romantic	0:00 / 3:08	Remove
2	Humnava Mere Jubin Nautiyal	Romantic	0:00 / 5:29	Remove
3	Saari Duniya Jalaa Denge B Praak	Emotional	0:00 / 3:02	Remove

Favorites:



Favorites

#	Title	Genre		Actions
1	Chaleya Arijit Singh	Romantic	♥	0:00 / 3:08
2	Bol Do Na Zara Armaan Malik	Romantic	♥	0:00 / 4:52