**Frontend Development with React.js**

**RhythmicTunes: Your Melodic Companion**

**1. Introduction**

* **Project Title:** RhythmicTunes: Your Melodic Companion
* **Team Members:**
  + Katheeja Barvin M
  + Mohanapriya G
  + Uma mageshwari M
  + Vasanthasena V

**2. Project Overview**

* **Purpose:** RhythmicTunes is a React + Vite-based music player web application offering users a clean, responsive, and enjoyable music listening experience. This project serves as a full-stack simulation by using a JSON Server backend, giving real-world API-like interaction within a development environment.
* **Features:**
  + Browse songs and search by title, singer, or genre.
  + Add tracks to favorites and create personalized playlists.
  + Play songs directly from the app without interruptions.
  + Fully responsive design using Bootstrap and Tailwind CSS.

**3. Architecture**

* **Component Structure:**
  + Follows a modular architecture with reusable UI elements.
  + Key components: Favorites, Playlist, Search, Sidebar, Songs, Uhome, Uitem, Unavbar, and Wishlist.
* **State Management:**
  + Utilizes React Hooks (useState, useEffect) for local state management.
* **Routing:**
  + Implemented with React Router.
  + Routes include /, /favorites, /playlist for smooth navigation.

**4. Setup Instructions**

* **Prerequisites:**
  + Node.js (version 16+)
  + npm or yarn
  + Git
* **Installation:**
  + Clone the repository: git clone https://github.com/your-repo.git
  + Navigate to the project folder: cd rhythmic-tunes
  + Install dependencies: npm install
  + Start the JSON Server: npm run server
  + Start the frontend: npm run dev

**5. Folder Structure**

MUSIC-PLAYER(FRONTEND)/

│-- public/

│-- src/

│ │-- assets/

│ │-- Components/

│ │ │-- Favorites.jsx

│ │ │-- Playlist.jsx

│ │ │-- Search.jsx

│ │ │-- Sidebar.jsx

│ │ │-- Songs.jsx

│ │ │-- Uhome.jsx

│ │ │-- Uitem.jsx

│ │ │-- Unavbar.jsx

│ │ │-- Wishlist.jsx

│ │ │-- sidebar.css

│ │ │-- uhome.css

│ │-- App.css

│ │-- App.jsx

│ │-- index.css

│ │-- main.jsx

│-- .eslintrc.cjs

│-- .gitignore

│-- db.json

│-- index.html

│-- package-lock.json

│-- package.json

│-- README.md

│-- vite.config.js

**6. Running the Application**

* **Frontend:** Run npm run dev in the root directory.
* **Backend (Mock API):** Run npm run server to start the JSON Server.

**7. Component Documentation**

* **Key Components:**
  + Favorites.jsx: Displays user-marked favorite songs.
  + Playlist.jsx: Manages user-created playlists.
  + Search.jsx: Allows users to search for songs.
  + Sidebar.jsx: Navigation menu for the app.
  + Songs.jsx: Displays all available songs.
  + Uhome.jsx: User home screen component.
  + Uitem.jsx: UI item component.
  + Unavbar.jsx: User navigation bar.
  + Wishlist.jsx: Displays wishlisted songs.

**8. State Management**

* **Global State:**
  + JSON Server acts as the backend to persist song data.
* **Local State:**
  + Managed using React Hooks for handling UI interactions like play/pause.

**9. User Interface**

* **Design System:**
  + Uses Bootstrap and Tailwind CSS for a modern, responsive UI.
  + Persistent sidebar navigation for easy access.
* **Screenshots:** Include images showcasing different UI elements.

**10. Styling**

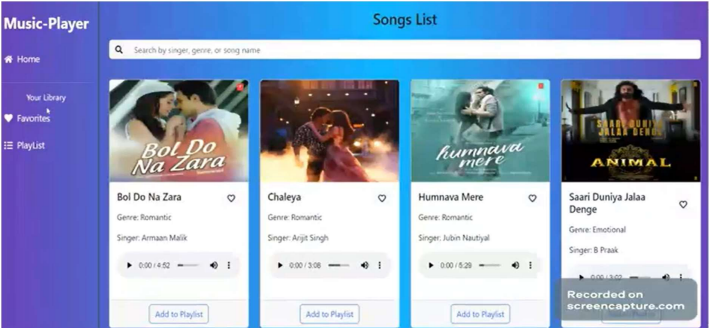
* **CSS Frameworks/Libraries:**
  + Tailwind CSS for utility-first styling.
  + Bootstrap for grid layout and components.
* **Theming:**
  + Light and dark mode support (to be added in future enhancements).

**11. Testing**

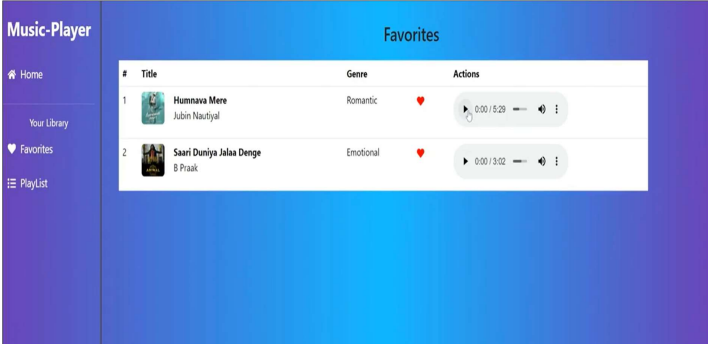
* **Testing Strategy:**
  + Unit testing with Jest.
  + Component testing with React Testing Library.
* **Code Coverage:**
  + Uses Istanbul for tracking test coverage.

**12. Screenshots**

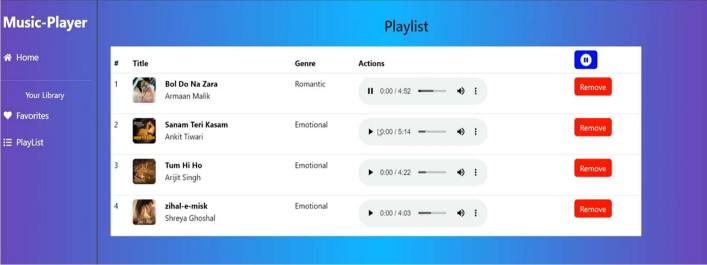
• Home (Songs Page)



• Favorites Page



• Playlist Page



**13. Known Issues**

* **Current Bugs:**
  + Some UI components need optimization for mobile screens.
  + JSON Server resets data on restart (needs real backend integration).

**14. Future Enhancements**

* Implement user authentication for private favorites and playlists.
* Replace JSON Server with a real music API like Spotify or Last.fm.
* Add lyrics display during song playback.
* Introduce dark and light mode toggle.
* Enable offline downloads for playing songs without internet.
* Add AI-based song recommendations based on listening history.

**15. Conclusion**

RhythmicTunes successfully demonstrates how to build a complete, user-centric music player using modern web development techniques. Combining React’s component-based structure with React Router for navigation and Axios for data fetching, it simulates a fully functional music streaming app.

The use of JSON Server ensures that developers get hands-on experience working with backend-style data management. The responsive design, powered by Bootstrap and Tailwind CSS, ensures accessibility across devices.

Overall, RhythmicTunes is a strong portfolio project showcasing both technical skills and real-world application structuring.

**16. References**

* [React Documentation](https://react.dev/)
* [Vite Documentation](https://vitejs.dev/)
* [React Router](https://reactrouter.com/)
* [JSON Server](https://www.npmjs.com/package/json-server)
* [Tailwind CSS](https://tailwindcss.com/)
* [Bootstrap](https://react-bootstrap.github.io/)