**Frontend Development with React.js**

**Project Documentation format**

1. Introduction

* **Project Title**: Rythimic Tunes
* **Team Member:**
* Team Leader:Saranya.J ([saranyajeeva2211@gmail.com](mailto:saranyajeeva2211@gmail.com))
* TeamMemembers: Varashali ([manojivaishu2027@gmail.com](file:///C:\Users\admin\Downloads\manojivaishu2027@gmail.com))
* Jennifer.S ([jenifersk2004@gmail.com)](file:///C:\Users\admin\Downloads\jenifersk2004@gmail.com))
* Haripriya .S[(haripriyasettu685@gmail.com)](C:\\Users\\admin\\Downloads\\(haripriyasettu685@gmail.com))

1. **Project Overview** 
   * **Purpose**: Rythimic Tunes aims to deliver a seamless music streaming platform that empowers users to enjoy, discover, and share diverse musical experiences, with a special focus on Tamil music. The goal is to create an intuitive, modern application tailored to music enthusiasts, young adults, and the global Tamil community.
   * **Features**: Key functionalities include song listings (predominantly Tamil), playlist creation, favorites management, playback control with YouTube integration, search functionality, responsive navigation via a sidebar, and dark theme support.
2. Architecture

* **Component Structure**: The application is built around major React components:
  + App.jsx: Central component managing routing, sidebar toggling, YouTube player integration, and error handling.
  + SongCard.jsx: Displays individual song details with playback controls, favorite/playlist toggles, and a progress bar.
  + Sidebar.jsx: Fixed, toggleable navigation linking to Songs, Favorites, and Playlist pages.
  + Header.jsx: Houses the search bar for filtering songs by title or singer.
  + Songs.jsx: Renders the main song list with search and randomization features.
  + Favorites.jsx: Displays and manages the user’s favorite tracks.
  + Playlist.jsx: Handles playlist creation and song management.  
    These components interact via React Router for navigation and API calls to an Express.js backend.
* **State Management**:
  + Local state is managed within components (e.g., playback state in SongCard.jsx, search input in Header.jsx).
  + Global state is implied through API-driven data (favorites and playlists stored in db.json), though no explicit use of Context API or Redux is mentioned.
* **Routing**: Uses react-router-dom in App.jsx to navigate between Songs (/songs), Favorites (/favorites), and Playlist (/playlist) routes.

1. Setup Instructions

* **Prerequisites**:
  + Node.js and npm (for running JavaScript and managing dependencies).
  + Git (for version control).
  + A code editor (e.g., Visual Studio Code, Sublime Text, or WebStorm).
  + Basic knowledge of HTML, CSS, and JavaScript.
* **Installation**:
  + Install Node.js from [nodejs.org](https://nodejs.org).
  + Clone the repository: git clone [your-repo-link].
  + Navigate to the project directory: cd rythimic-tunes.
  + Install dependencies: npm install.
  + (Optional) Configure environment variables if needed (not specified in reference; e.g., backend URL).

1. Folder Structure

* **Client**: The React application is organized under rythimic-tunes/src/:
  + components/: Contains Favorites.jsx, Header.jsx, Playlist.jsx, Sidebar.jsx, SongCard.jsx, and Songs.jsx.
  + assets/: Holds static files like react.svg for UI elements.
  + App.jsx: Main component with routing and layout.
  + main.jsx: Entry point for rendering the app.
  + index.css: Global styles with dark theme support.
  + App.css: Additional layout-specific styles.
* **Utilities**: No explicit custom hooks or utility classes are detailed, but yt-search in server.js serves as a helper for YouTube URL fetching, and API interactions (via axios or fetch) are assumed in components like SongCard.jsx.

1. Running the Application

* **Frontend**:
  + Run the development server: npm run dev in the rythimic-tunes directory (starts at http://localhost:5173).
* **Backend**:
  + Start the Express server: npm start in the rythimic-tunes directory (runs at http://localhost:5000).
  + Ensure the backend is running before accessing the frontend to enable API calls.

1. Component Documentation

* **Key Components**:
  + App.jsx: Core component; manages routing, sidebar state, and YouTube player. Props: None explicitly listed, but it wraps all routes.
  + SongCard.jsx: Displays song data (title, singer) with play/pause, favorite, and playlist buttons. Props: Likely includes song object (title, singer, youtubeUrl).
  + Sidebar.jsx: Navigation menu; toggleable via a button in App.jsx. Props: None explicitly noted.
  + Header.jsx: Search bar for filtering songs. Props: Likely includes a callback for search updates.
  + Songs.jsx: Lists songs with search and randomization. Props: Accepts filtered song data.
  + Favorites.jsx: Shows favorite tracks with toggle functionality. Props: Fetches data from /api/favorites.
  + Playlist.jsx: Manages playlists with add/remove features. Props: Fetches data from /api/playlist.
* **Reusable Components**: SongCard.jsx is reusable across Songs.jsx, Favorites.jsx, and Playlist.jsx, configured with different data and callbacks.

1. State Management

* **Global State**: Handled implicitly via API endpoints (/api/favorites, /api/playlist) and db.json. No centralized state management library (e.g., Redux) is specified.
* **Local State**:
  + SongCard.jsx: Manages playback state (play/pause, progress).
  + Header.jsx: Tracks search input.
  + Sidebar.jsx: Toggles visibility state (controlled by App.jsx).

1. User Interface

* Screenshots from the reference document:
  + **Song Listing**: Main page (Songs.jsx) with song cards and search bar.
  + **Favorites**: List of favorite songs (Favorites.jsx).
  + **Playlist**: Playlist song list (Playlist.jsx).
  + **Sidebar**: Navigation menu (Sidebar.jsx).
  + [Note: Actual screenshots or GIFs would be embedded here if provided.]

1. Styling

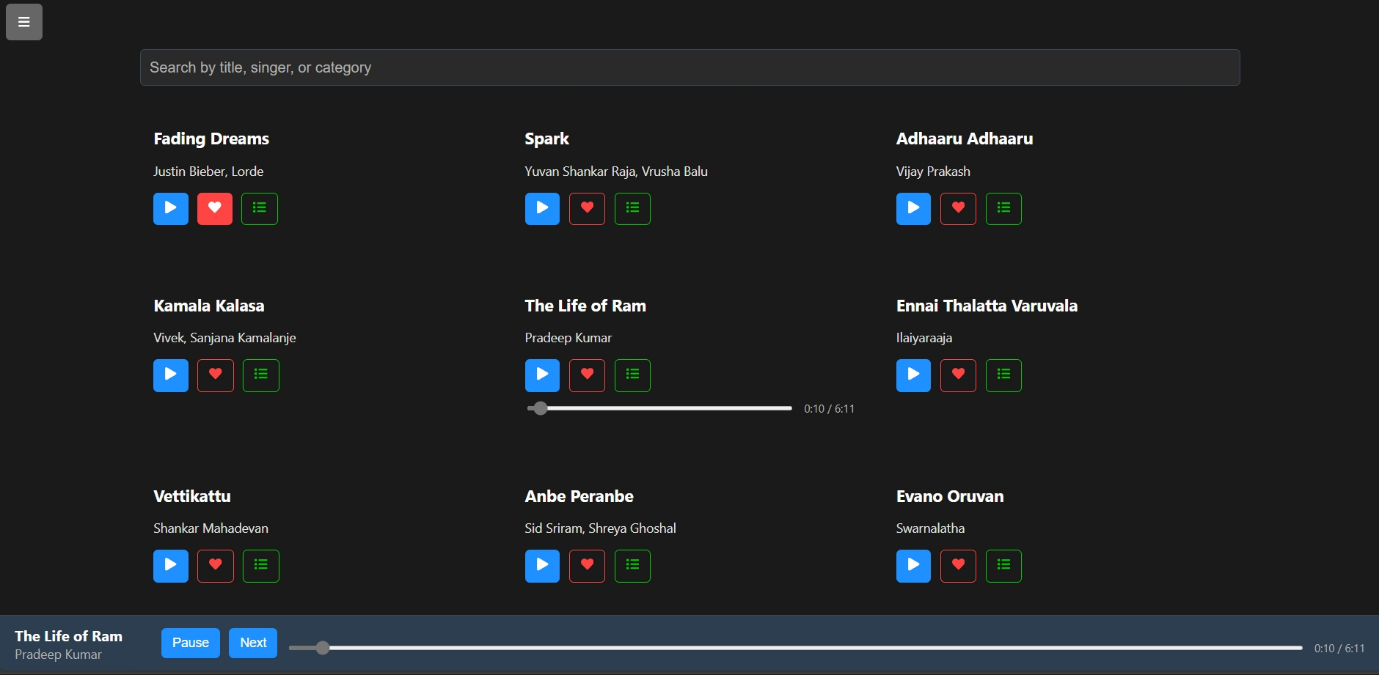
* **CSS Frameworks/Libraries**: No external frameworks (e.g., Bootstrap) are mentioned; styling is custom via index.css and App.css.
* **Theming**: Implements a dark theme using CSS variables in index.css, with potential for light theme expansion. Styles are responsive for desktops, tablets, and smartphones.

1. Testing

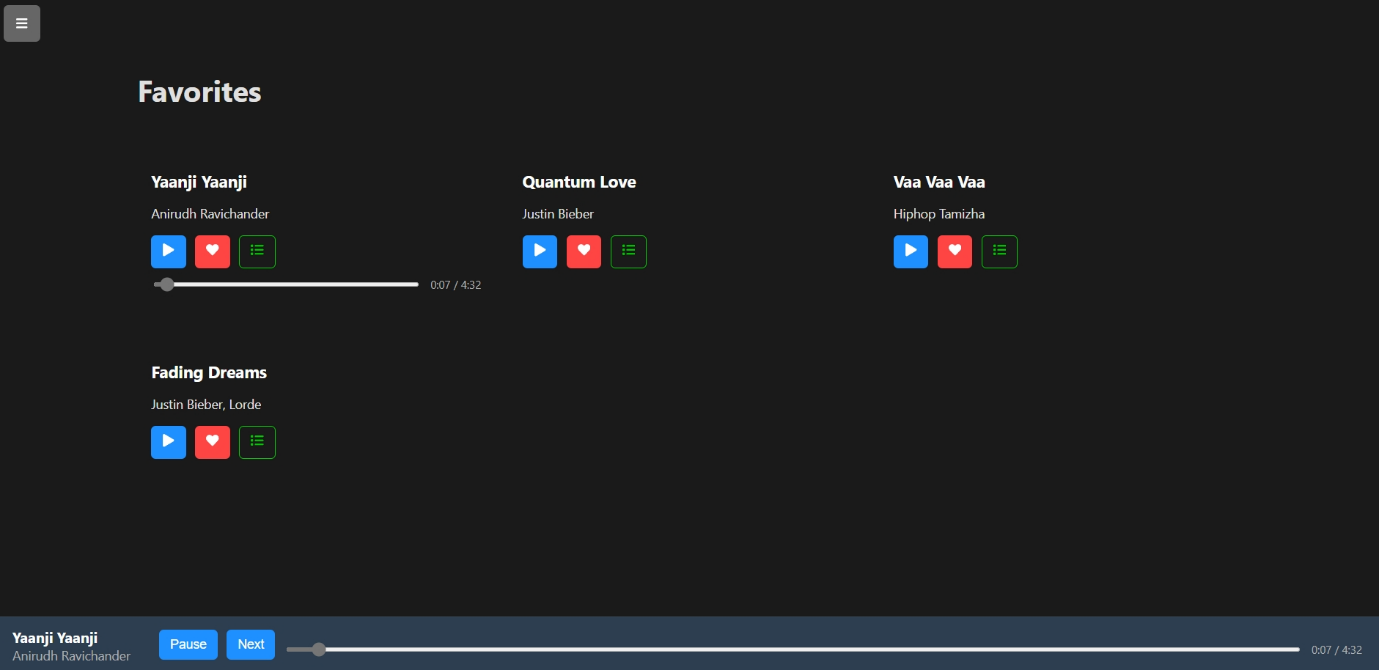
* **Testing Strategy**: Not detailed in the reference document. A suggested approach could include:
  + Unit testing components (e.g., SongCard.jsx) with Jest and React Testing Library.
  + Integration testing for API interactions (e.g., fetching songs).
  + End-to-end testing with tools like Cypress for user flows (search, playback).
* **Code Coverage**: No tools or metrics specified; could use Jest’s built-in coverage reports if implemented.

1. Screenshots or Demo

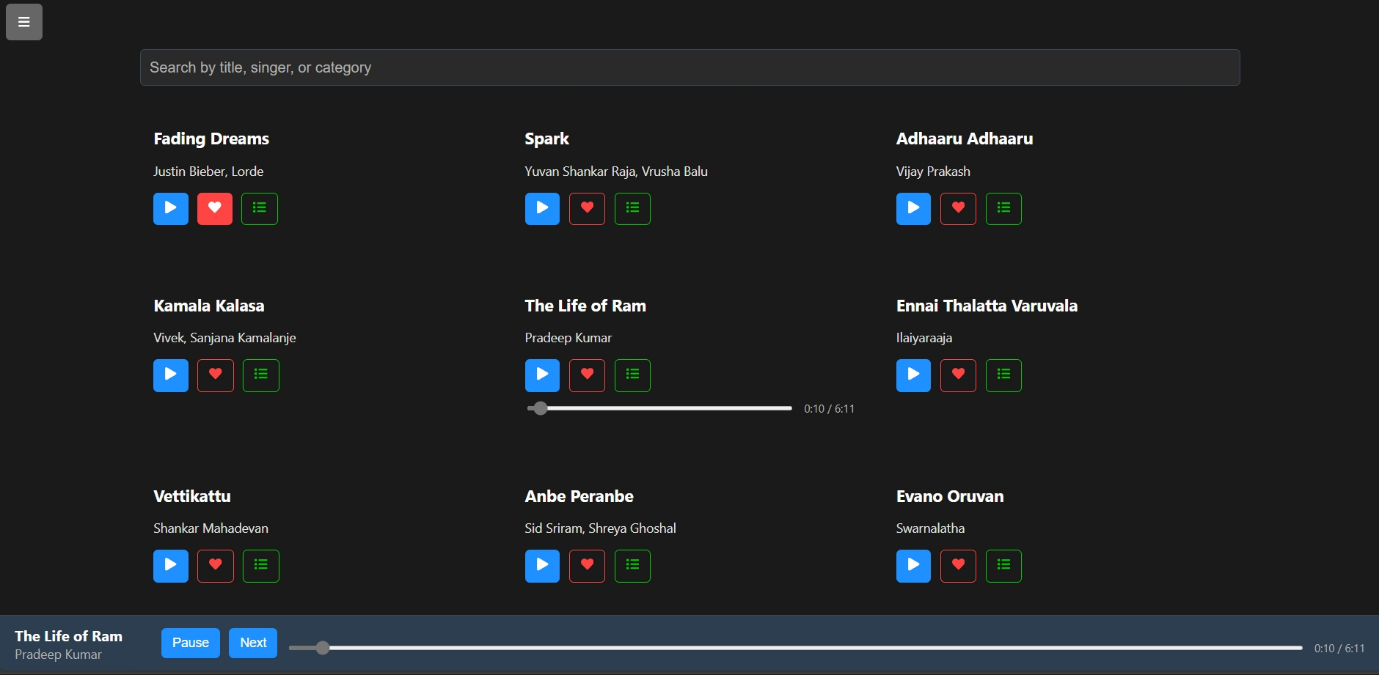
* **Screenshots**: **SONG LISTING:** Main page (Songs.jsx) with song cards and search bar.



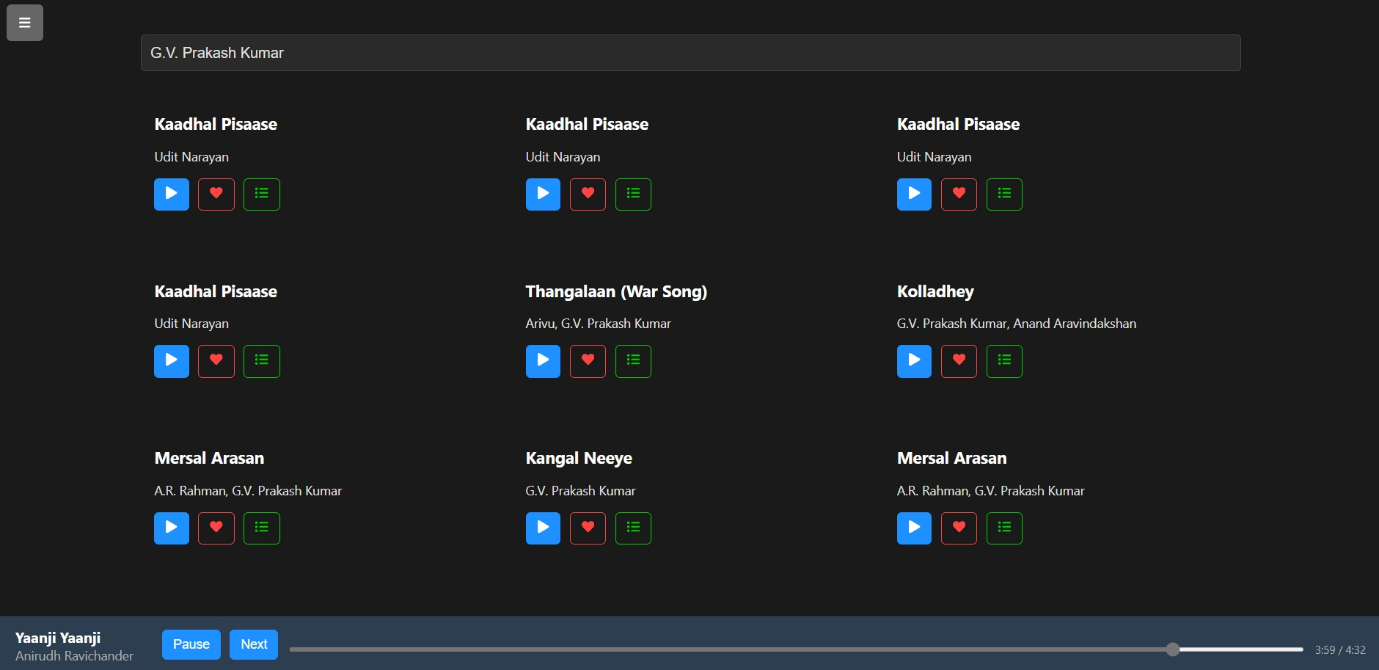
* **FAVORITES:** Favorite songs list (Favorites.jsx).



* **PLAYLIST:** Playlist songs list (Playlist.jsx).



* **SIDEBAR:** Navigation sidebar (Sidebar.jsx).



* **Demo**: [https://drive.google.com/file/d/1EaZIK9eghJ5bYes62OQr Whv4uiwIMMf/view?usp=drive\_link](https://drive.google.com/file/d/1EaZIK9eghJ5bYes62OQr%20Whv4uiwIMMf/view?usp=drive_link)

1. Known Issues

* Not explicitly listed in the reference. Potential issues could include:
  + YouTube URL fetching errors if yt-search fails (handled in App.jsx).
  + Limited scalability of db.json for large datasets.

1. Future Enhancements

* Potential features based on reference:
  + Light theme toggle alongside the dark theme.
  + TypeScript integration (suggested in README.md).
  + Advanced search filters (e.g., by genre or year).
  + Animations for UI transitions (e.g., sidebar toggle, song playback).
  + Offline song caching or expanded library beyond 1300+ Tamil songs.