Project Documentation: Rythmic Tunes: Your Melodic Companion

1. Introduction

- **Project Title**: Rythmic Tunes: Your Melodic Companion
- **Team Leader**: Leelakumari S
- **Team Members**:
    - Sreenidhi M
    - Sharmila L
    - Parveena L

2. Project Overview

Purpose:

The purpose of **Rythmic Tunes: Your Melodic Companion** is to provide users with a seamless and engaging music experience through an intuitive web application. The app serves as a melodic companion by offering features such as music discovery, playlists, song recommendations, and personalized playlists based on user preferences. This app aims to cater to users who enjoy discovering new music, creating custom playlists, and sharing their musical tastes with others.

Features:

- **Music Discovery**: Browse a variety of songs, genres, and albums.
- **Playlist Creation**: Create and manage custom playlists.
- **Song Recommendations**: Personalized song recommendations based on listening preferences.
- **Search Functionality**: Search for songs, artists, and albums with ease.
- **Responsive Design**: The application is fully responsive and works across different devices.
- **User Profiles**: Users can create profiles to save preferences, playlists, and liked songs.
- **Integration with Music APIs**: Fetching music-related data from external APIs.

3. Architecture

Component Structure:

The application is structured into reusable, modular React components. Major components include:

- **App.js**: The main container component that handles routing and the overall layout.
- **Header.js**: Displays navigation links and user profile options.
- **Sidebar.js**: Provides quick access to user playlists and preferences.

- **SongList.js**: Displays the list of songs or albums.
- **Player.js**: The music player that plays songs, showing controls like play, pause, next, previous, etc.
- **Playlist.js**: Allows users to manage their created playlists.

State Management:

- **Context API** is used for global state management. We use Context to store user preferences, the currently playing song, and playlist data.
- Local state is used within components to manage temporary states like toggling the song player or handling search queries.

Routing:

- **React Router** is used for navigating between pages in the app. The main routes include:
  - `/home`: The landing page with a list of songs and albums.
  - `/playlist`: A page showing the user's playlists.
  - `/profile`: The user profile page.
  - `/search`: A page dedicated to song search functionality.

4. Setup Instructions

Prerequisites:

- Node.js (version >=14.x.x)
- npm (Node Package Manager)

Installation:

1. Clone the repository:
2. `git clone https://github.com/username/rythmic-tunes.git`
3. Navigate to the `client` directory:
4. `cd rythmictunes/client`
5. Install dependencies:
6. `npm install`
7. Set up environment variables (e.g., API keys for music data, if any). Create a `.env` file and add necessary environment variables.
8. Run the development server:
9. `npm start`

5. Folder Structure

Client:

- **src**: Contains the React application code.
  - **components**: React components used in the app (e.g., Header.js, Player.js, SongList.js).
  - **pages**: Contains pages such as HomePage, PlaylistPage, ProfilePage.

    - **assets**: Images, icons, and other media files.
    - **utils**: Utility functions and custom hooks like useFetch.js, useLocalStorage.js.
    - **services**: API interaction files, e.g., apiService.js.
    - **styles**: CSS/SCSS files for styling the app.

Utilities:

- **useFetch.js**: Custom hook to fetch data from music APIs.
- **useLocalStorage.js**: Custom hook for saving data to local storage (e.g., user preferences).

6. Running the Application

To start the frontend server locally:

```
cd client
npm start
```

This will start the development server at `http://localhost:3000`.

7. Component Documentation

Key Components:

- **App.js**: Main application component that contains the routing logic and global state.
- **Header.js**: Displays navigation, search bar, and user profile options.
- **Player.js**: A component that controls the music player interface (play, pause, volume control, etc.).
- **SongList.js**: Displays a list of songs or albums fetched from the API.
- **Playlist.js**: Manages user-created playlists and allows users to add/remove songs from playlists.

Reusable Components:

- **SongCard.js**: A reusable component that displays individual song details (like album art, song title, artist, etc.).
- **Button.js**: A reusable button component used throughout the application with customizable labels, styles, and actions.

8. State Management

Global State:

The application uses **Context API** to handle global state such as:

- User preferences (e.g., theme, language).
- Current playing song.
- User-created playlists.

Local State:

Local state is managed within individual components:

- **Player** uses local state for play/pause, volume, and song progress.
- **Search** uses local state to store query strings and search results.

9. User Interface

Screenshots showcasing the application UI features:

1. **Home Page** with song list and navigation options.
2. **Music Player** interface with play/pause controls.
3. **Search Page** displaying search results.

Screenshots and GIFs will be uploaded to the shared Google Drive folder linked below.

10. Styling

CSS Frameworks/Libraries:

- **Styled-Components** is used for writing component-level CSS with support for theming and dynamic styling.

Theming:

- A custom theme is implemented for light and dark modes. This allows users to switch between a light or dark theme according to their preference.

11. Testing

Testing Strategy:

- **Unit Testing**: Jest is used for testing individual components.
- **Integration Testing**: React Testing Library is used to test component integration and interaction.
- **End-to-End Testing**: Cypress is used to simulate real user interactions and test the application flow.

Code Coverage:

- We use **Jest** and **Istanbul** for measuring code coverage. Reports are generated after each test run to ensure that all major components and functions are tested.

12. Screenshots or Demo

https://drive.google.com/drive/folders/1jhOP1fEPKwz2FRTwuQzeEE2_-GkLqXrR

13. Known Issues

- **Issue with Audio Player on Mobile**: Sometimes the audio player does not automatically resume on mobile devices after being paused.
- **Search Function Lag**: There's a slight delay in search results due to external API response time.

14. Future Enhancements

- **AI-Powered Recommendations**: Introduce machine learning algorithms to provide more accurate music recommendations.
- **Music Player Enhancements**: Adding advanced features like equalizer, shuffle, and repeat modes.
- **User Sharing Options**: Allow users to share their playlists with friends via social media or direct links.
- **Offline Support**: Enable users to download playlists and listen offline.
