FRONTEND DEVELOPMENT WITH React.js- PROJECT DOCUMENTATION

1. Introduction

Project Title:

Rythmic Tunes – Music Streaming App

Team Members:

(List team members and their roles here.: Frontend Developer – Monicha.c , content creator-Aysha bee.M, video creator-Ezhilarasi.V ,Document creator- Janani.D)

2. Project Overview

Purpose:

The main purpose of the Rythmic Tunes project is to create a seamless, interactive, and user-friendly music streaming platform. It enables music enthusiasts to explore, stream, and share diverse musical experiences.

Features:

* Clean and intuitive interface for browsing, saving, and sharing favourite songs and playlists.
* Advanced search and filtering options for easy discovery of tracks.
* Responsive design for desktops and mobile devices.
* Built using a modern technology stack (React.js, React Router, Axios, Bootstrap/Tailwind CSS).

3. Architecture

Component Structure:

App.js – The root component containing BrowserRouter and all Routes.

Sidebar.js – Handles navigation between pages.

Songs.js – Displays the list of songs fetched from the JSON server.

Favorites.js – Displays and manages user’s favourite tracks.

Playlist.js – Handles playlist creation and management.

Hero.js – Hero/banner section to highlight featured content.

State Management:

Local State: Managed with React Hooks like useState and useEffect inside each component.

Global State: Can be extended using Context API or Redux for shared data like user session or playlists.

Routing:

Implemented with React Router DOM for client-side navigation:

/ → Renders the Songs component.

/favorites → Renders the Favorites component.

/playlist → Renders the Playlist component.

4. Setup Instructions

Prerequisites:

Node.js & npm (Download)

Git (Download)

Basic knowledge of HTML, CSS, JavaScript

Code editor (Visual Studio Code recommended: Download)

Installation:

1. Clone the repository:

git clone <repository-url>

cd tunes

2. Install dependencies:

npm install

3. (Optional) Start the JSON server:

json-server --watch ./db/db.json

4. Start the development server:

npm start

5. Open http://localhost:3000 in your browser.

5. Folder Structure

Client:

React application organized under src/ folder:

src/

App.js

App.css

index.js

components/

Sidebar.js

Hero.js

Songs.js

Favorites.js

Playlist.js

assets/

images/

icons/

db/

db.json

utils/

api.js

Utilities:

utils/api.js contains Axios API calls to the JSON server.

Any custom hooks (if used) are stored in src/hooks/.

Helper functions are kept separate for clarity and reusability.

6. Running the Application

Frontend:

npm start

(Run in the project root directory.)

JSON Server (if applicable):

json-server --watch ./db/db.json

Access the app at http://localhost:3000.

7. Component Documentation

Key Components:

App.js: Root component defining routing and layout.

Sidebar.js: Provides navigation to Songs, Favorites and Playlist pages.

Songs.js: Fetches and displays list of songs.

Favorites.js: Displays the user’s favourite tracks.

Playlist.js: Allows creating and managing playlists.

Hero.js: Displays a hero banner or featured content area.

Reusable Components:

Buttons styled with Bootstrap/Tailwind.

Card components for consistent display of song/playlist items.

Input fields reused across search and forms.

8. State Management

Global State:

If implemented, Context API or Redux will store shared data like user info or playlists and make it accessible across components.

Local State:

Handled with React Hooks inside individual components for UI states such as search filters, form inputs, and toggles.

9. User Interface

Screenshots or GIFs can be included here to showcase:

Homepage view with hero section.

Songs list page.

Favorites page.

Playlist management page.

10. Styling

CSS Frameworks/Libraries:

Bootstrap or Tailwind CSS for responsive design and grid layouts.

React Icons for consistent icon usage.

Theming:

Custom theming is implemented in App.css for branding and unique styling of components.

11. Testing

Testing Strategy:

Unit testing of individual components using Jest and React Testing Library.

Integration testing for verifying interaction between components (e.g. Songs + Playlist).

(Optional) End-to-end testing using Cypress or Playwright.

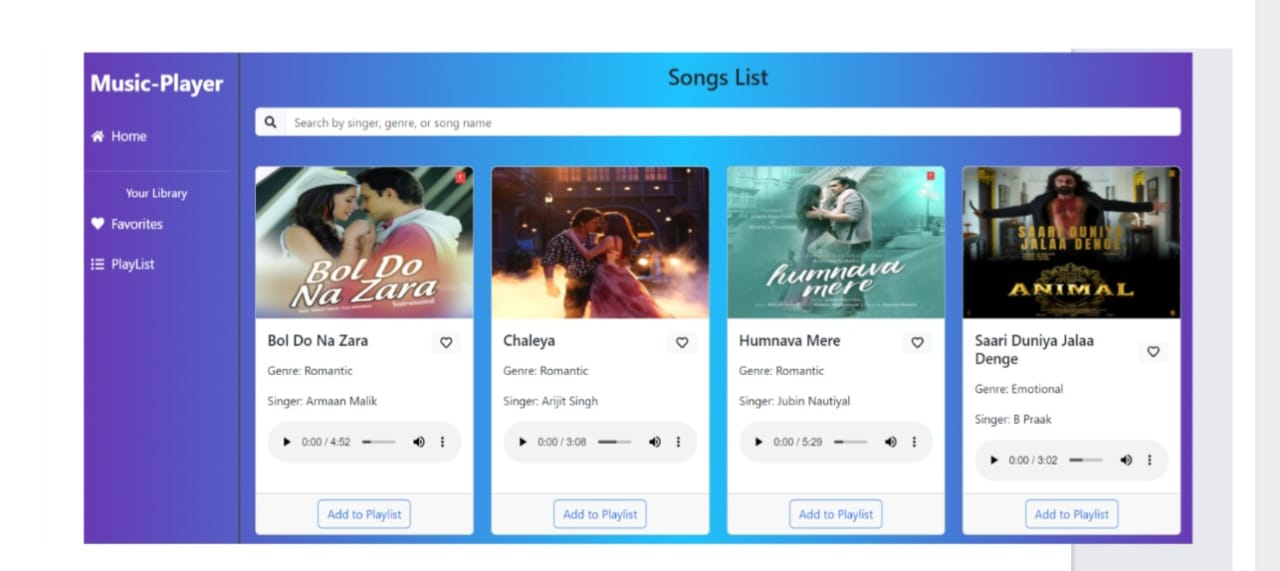
Code Coverage:

Configure Jest to generate coverage reports to ensure adequate test coverage.

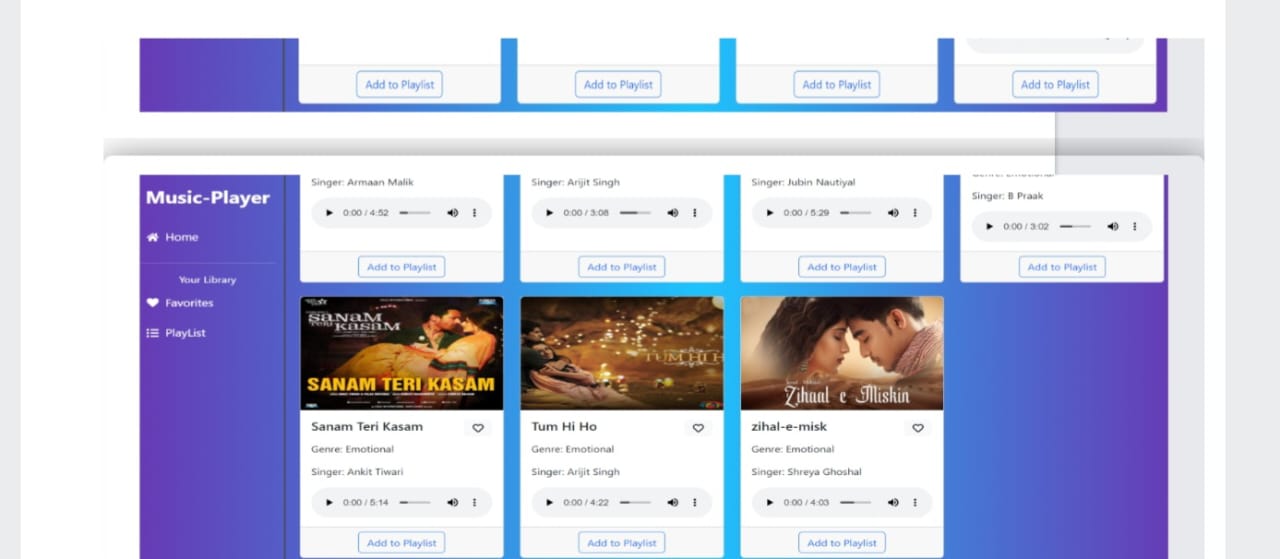
12. Screenshots or Demo

Include screenshots of all major UI sections or provide a link to a hosted demo if available. For example:

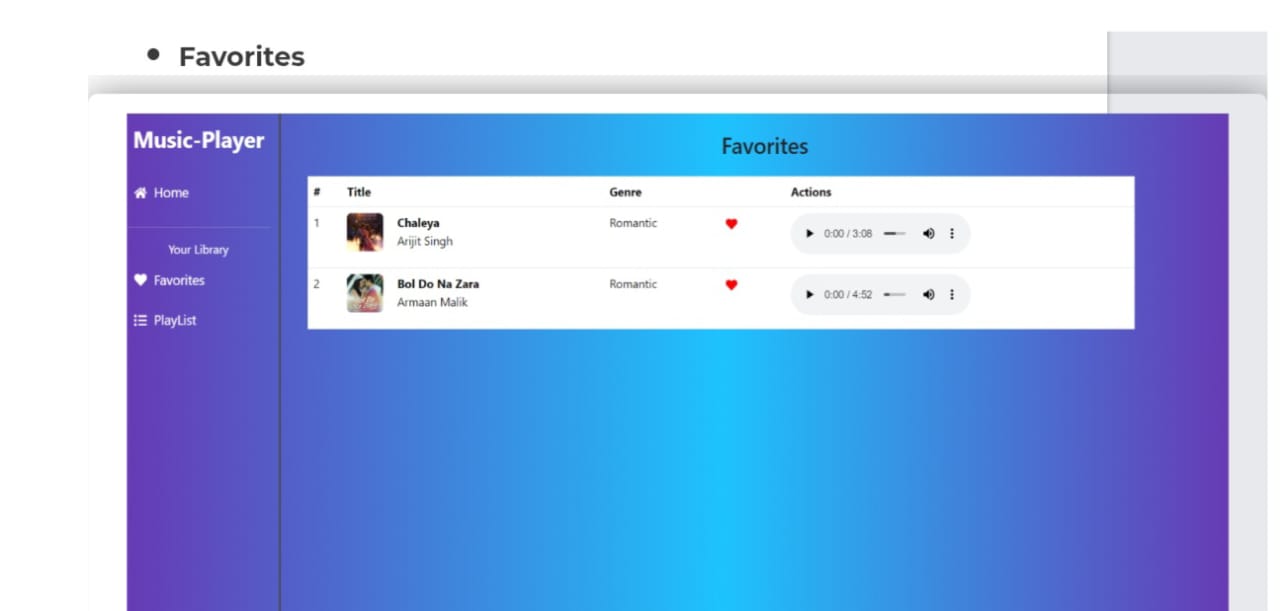
Hero section screenshot.



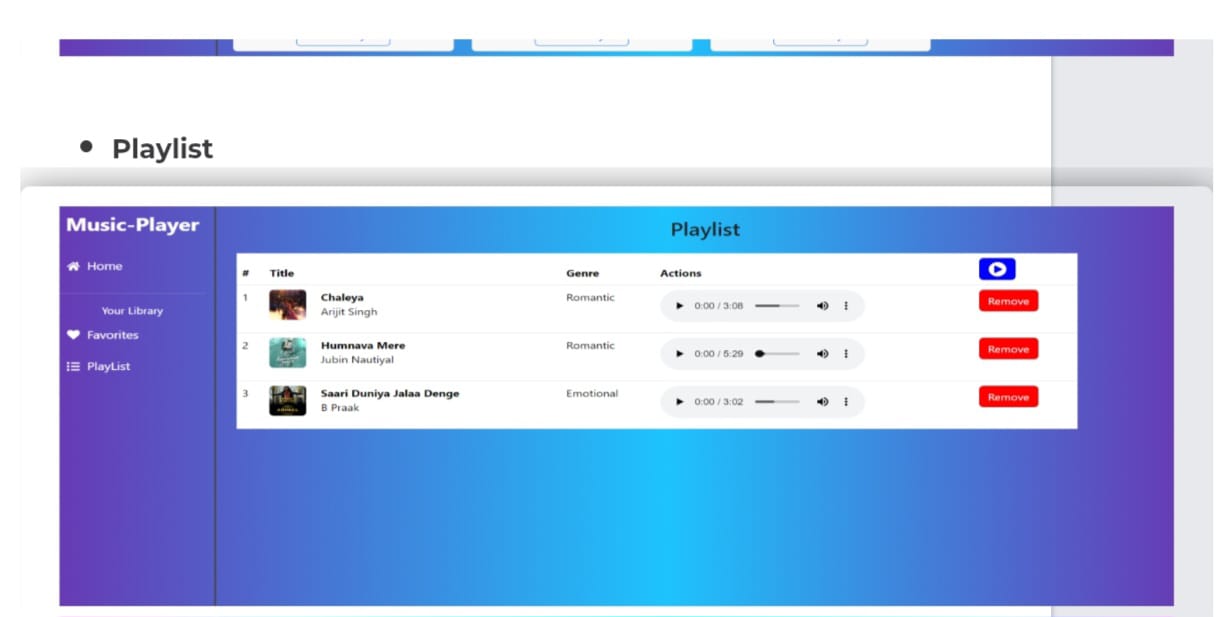
Songs page screenshot.



Favorites page screenshot.



Playlist page screenshot.



13. Known Issues

Document any known bugs or limitations. Example:

JSON server must be running to fetch data.

First API call may be slow on initial load.

14. Future Enhancements

Potential improvements and features:

Integrate a real music streaming API instead of JSON server.

Add user authentication and profiles.

Build an in-browser audio player with lyrics display.

Add animations and advanced styling for a richer user experience.