Frontend Development with React.js

1. Introduction

Project Title: Rythmic Tunes : your melodic companion

Team Leader :

Ramya . G ([ramyagopi345@gmail.com](mailto:ramyagopi345@gmail.com))

Team Members:

Ramya Shankari. V ([ramyavijai92392@gmail.com](mailto:ramyavijai92392@gmail.com))

Revati Sree . P ([revatisreep12b441@gmail.com](mailto:revatisreep12b441@gmail.com))

Saranya. J ([jeyavel672005@gmail.com](mailto:jeyavel672005@gmail.com))

2. Project Overview

Purpose: To listen songs

The primary goal of music stream is to provide a seamless platform for music enthusiasts,enjoying, and sharing diverse musical experiences.our objectives includes:

* User-friendly interface.
* Comprehensive Music Streaming
* Modern Tech Stack

Features:

* Song Listings
* Play Creation
* Playback Control
* Offline Listening
* Search Funtionality

3. Architecture

Component Structure:

The application follows a component-based architecture. The major components are:

* App.jsx: The main component that includes the overall structure.
* PlayerControls.jsx: Manages play, pause, next, previous, and volume controls.
* Playlist.jsx: Displays and manages playlists.
* SongItem.jsx: Represents an individual song in the list.

State Management:

The application handles state management in two ways:

* Local State: Manages UI elements like buttons, current song, and volume settings.
* Global State: If required, React Context API or Redux can be used to manage the overall application state.
* Routing: If the application has multiple pages (e.g., Home, Playlist, Settings), React Router can be used for navigation.

4. Setup Instructions

Prerequisites:

* Ensure the following software is installed on your system:
* Node.js (for running JavaScript on the backend)
* npm (Node Package Manager) or yarn (for managing dependencies)

Installation Steps:

1. Clone the repository using Git: git clone [repository-url]

2. Navigate into the project directory: cd music-player-frontend

3. Install required dependencies: npm install

5. Folder Structure

MUSIC-PLAYER (FRONTEND)

│── db/ # (Database folder, if required)

│── node\_modules/ # (Dependencies installed via npm)

│── public/ # (Public assets like favicon, index.html)

│── src/ # (Main source code folder)

│ ├── assets/ # (Images, audio files, icons)

│ ├── Components/ # (All React components)

│ ├── App.css # (Main CSS file)

│ ├── App.jsx # (Main application component)

│ ├── index.css # (Global styles)

│ ├── main.jsx # (Application entry point)

│── .eslintrc.cjs # (Linting configuration)

│── .gitignore # (Files to ignore in Git)

│── index.html # (HTML entry point)

│── package-lock.json # (Dependency lock file)

│── package.json # (Project configuration file)

│── README.md # (Project documentation)

│── vite.config.js # (Vite configuration for fast development)

Client (src/): Contains the React application files.

Utilities: Any helper functions, utility classes, or hooks used.

6. Running the Application

To start the development server, run the following command:

* npm start
* This will launch the application on http://localhost:3000/ (or another available port).

7. Component Documentation

Key Components:

App.jsx: The main component that controls the application flow.

PlayerControls.jsx: Handles play, pause, next, and volume adjustments.

Playlist.jsx: Displays and manages user playlists.

SongItem.jsx: Represents a single song entry.

8. State Management

Global State Management: Context API or Redux can be used to store and share state across components, such as the current playing song.

Local State Management: Each component has its own useState() hooks to manage UI behaviors (e.g., play/pause, volume).

9. User Interface

Screens and UI Features:

* Home Page: Displays the list of available songs.
* Playlist Page: Allows users to create and manage playlists.
* Player Controls: Floating music player with play/pause, skip, and volume controls.

10. Styling

CSS Frameworks/Libraries:

Tailwind CSS or Bootstrap (if used) for styling.

Sass or Styled Components for advanced theming.

Theming: If the application supports light mode/dark mode, mention how it’s implemented.

11. Testing

Testing Strategy:

Unit Testing:

Test individual components using Jest or React Testing Library.

Integration Testing:

Test how components interact.

End-to-End Testing:

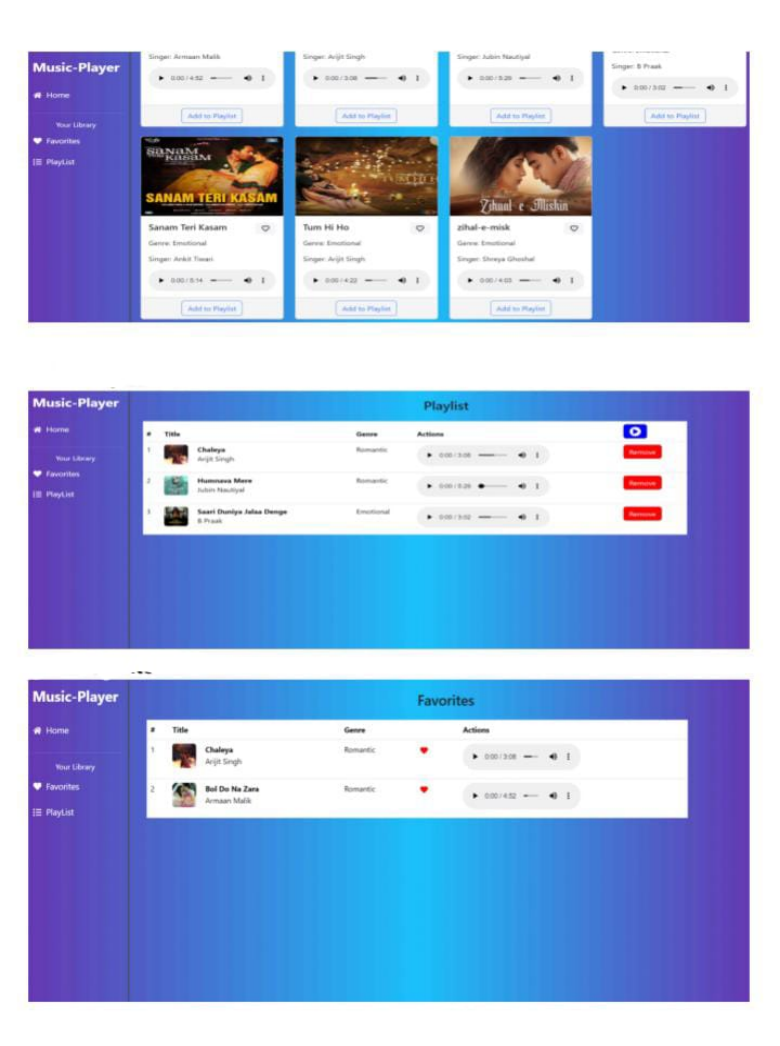
Use Cypress to test full user flows.

Code Coverage:

Measure code quality using tools like Jest coverage reports.

12. Demo Screenshot:

|  |
| --- |
|  |
|  |  |



13. Known Issues

Volume slider does not work on mobile devices.

Playback controls may be slow when switching songs quickly.

14. Future Enhancements

Planned Features:

* Offline playback support (allow users to save songs for offline listening).
* Lyrics display (fetch and display song lyrics).
* Animations and improved UI for better user experience.
* Social sharing (share favorite songs/playlists with others).
* More themes and color customization options.

Project Demo Link:

[https://drive.google.com/file/d/1O6EowPpSriQ4zM09Ltv0tmpbonvhbcYI/view?usp=drivesdk](%20https:/drive.google.com/file/d/1O6EowPpSriQ4zM09Ltv0tmpbonvhbcYI/view?usp=drivesdk)

Github Link:

[https://github.com/asunm13332213331026029/music-streaming-app](%20https:/github.com/asunm13332213331026029/music-streaming-app)