Rythimic Tunes (React)

1. Introduction:

Project Title: RhythmicTunes: Your Melodic Companion

Team Members:

Merlin Meronika S – Team Leader

Meena R - Team Member 1

Aarthi B - Team Member 2

Swathi M - Team Member 3

2. Project Overview

Purpose:

The primary goal of Music Streaming is to provide a seamless platform for music enthusiasts, enjoying, and sharing diverse musical experiences. Our objectives include:

User-Friendly Interface: Develop an intuitive interface that allows users to effortlessly explore, save, and share their favorite music tracks and playlists.

Comprehensive Music Streaming: Provide robust features for organizing and managing music content, including advanced search options for easy discovery.

Modern Tech Stack: Harness cutting-edge web development technologies, such as React.js, to ensure an efficient and enjoyable user experience while navigating and interacting with the music streaming application.

Features:

Song Listings: Display a comprehensive list of available songs with details such as title, artist, genre, and release date.

Playlist Creation: Empower users to create personalized playlists, adding and organizing songs based on their preferences.

Playback Control: Implement seamless playback control features, allowing users to play, pause, skip, and adjust volume during music playback.

Offline Listening: Allow users to download songs for offline listening, enhancing the app's accessibility and convenience.

Search Functionality: Implement a robust search feature for users to easily find specific songs, artists, or albums within the app.

3. Architecture

Component Structure:

The application follows a **modular component-based structure**, ensuring reusability and maintainability. Below is the outline of major components and their interactions:

- **App Component (App.js)**

Root Component that manages the entire application.

Wraps the application with BrowserRouter for routing.

Includes the **Sidebar**, **Routes**, and the main content area.

- **Sidebar Component (Sidebar.js)**

Provides **navigation** options for different sections of the app.

Contains links to **Songs, Favorites, and Playlists** using NavLink from react-router-dom.

- **Songs Component (Songs.js)**

Fetches and displays a **list of songs** from the backend (db.json).

Provides **search functionality** for filtering songs by title, artist, or genre.

Implements **audio playback controls** with useState for tracking the currently playing song.

Allows users to **add/remove** songs from the **wishlist and playlist**.

- **Favorites Component (Favorites.js)**

Displays songs marked as **favorites** by the user.

Fetches data from http://localhost:3000/favorities.

Allows users to **remove songs** from favorites.

- **Playlist Component (Playlist.js)**

Shows **user-created playlists**.

Allows **adding and removing** songs from playlists.

Fetches data from http://localhost:3000/playlist.

- **Search Component (Search.js)**

Provides a **search bar** that filters songs dynamically.

Integrated within the Songs.js component.

- **AudioPlayer Component (AudioPlayer.js)**

Handles **song playback** with audio HTML element.

Ensures that **only one song plays at a time**.

- **Button Components (FavoriteButton.js, PlaylistButton.js)**

Handles actions like **adding/removing songs** to/from the wishlist and playlist.

State Management:

The application uses **React's built-in useState and useEffect hooks** for managing state.

Global & Local State Handling

- **Songs List (useState)**

  items: Stores all fetched songs.

  wishlist: Holds favorite songs.

  playlist: Stores playlist songs.

- **Audio Playback (useState)**

  currentlyPlaying: Keeps track of the currently playing audio element.

- **Search Functionality (useState)**

  searchTerm: Stores the user-inputted search term to filter songs dynamically.

- **Data Fetching (useEffect)**

Uses **Axios (axios.get)** to fetch songs, favorites, and playlist data from db.json.

- **CRUD Operations (useState + API Calls)**

  addToWishlist(itemId): Adds a song to favorites via a POST request.

  removeFromWishlist(itemId): Removes a song from favorites via a DELETE request.

  addToPlaylist(itemId): Adds a song to the playlist via a POST request.

  removeFromPlaylist(itemId): Removes a song from the playlist via a DELETE request.

State Management Libraries:

Since the app is **small-to-medium scale**, **Redux or Context API** isn't strictly necessary.

If scalability is required, **Context API** could be introduced for managing **favorites & playlists** globally.

Routing (React Router):

The application uses **React Router DOM** for client-side navigation.

Uses BrowserRouter to wrap the entire application.
Defines **routes** using Routes and Route:

/ → Loads **Songs Component**.

/favorites → Loads **Favorites Component**.

/playlist → Loads **Playlist Component**.

Navigation is handled by the Sidebar component.

5. Setup Instructions

Prerequisites:

Before setting up the project, ensure you have the following installed on your system:

Node.js & npm – Install from Node.js official site

Git – Version control system (Download Git)

Code Editor – (e.g., VS Code)

JSON Server – For managing local data

Installation Steps:

Clone the Repository:

```
git clone <repository_url>
cd rythimic-tunes
```

Install Dependencies:

```
npm install
```

Configure Environment (if required):

Create a .env file in the project root and add necessary API keys or configurations.

Start the Development Server:

If using **Vite.js:**

```
npm run dev
```

If using **React Create App:**

```
npm start
```

Start the JSON Server (for local data):

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

Access the Application:

Open http://localhost:5173 in your browser (for Vite).

Open http://localhost:3000 if running on a standard React server.

5. Folder Structure

Client: (React Application Organization)

The project follows a modular folder structure for better maintainability:

```
/rythimic-tunes
├── /src
│   ├── /components       # Reusable UI components (Navbar, Footer, Player, etc.)
│   ├── /pages            # Page components (Home, Playlist, Favorites, etc.)
│   ├── /assets           # Static files (images, icons, etc.)
│   ├── /context          # Context API for global state management
│   ├── /hooks            # Custom hooks for reusable logic
│   ├── /utils            # Helper functions and utilities
│   ├── App.js            # Main application component
│   ├── index.js          # Entry point for React
├── /public               # Public assets (favicon, index.html)
├── /db                   # JSON Server database file
├── package.json          # Project dependencies and scripts
├── README.md             # Project documentation
```

Utilities: (Helper Functions & Custom Hooks)

Helper Functions (/utils)

Format timestamps, API request handlers, and data processing functions.

Custom Hooks (/hooks)

useFetch() – Handles API data fetching.

useAudioPlayer() – Manages audio playback state.

Context API (/context)

Provides global state for playlist management and user preferences.

6. Running the Application

To run the application locally, follow these steps:

Starting the Frontend Server:

Navigate to the project folder:

cd rythimic-tunes

Install dependencies:

npm install

Start the development server:

npm start # If using Create React App

OR

npm run dev # If using Vite

Open the application in your browser at:
http://localhost:5173 (for Vite)
OR
http://localhost:3000 (for Create React App)

Starting the JSON Server (Mock Backend):

If the project uses JSON Server for handling data:

Open a new terminal and run:

json-server --watch ./db/db.json --port 3000

The mock API will be available at **http://localhost:3000**.

7. Component Documentation

Key Components:

App.js – Root component managing routing and layout.

SongList.js – Displays a list of songs fetched from the API.

Props: songs (array), addToPlaylist (function), addToFavorites (function).

Playlist.js – Shows user-created playlists.

Props: playlist (array), removeFromPlaylist (function).

Favorites.js – Displays favorited songs.

Props: favorites (array), removeFromFavorites (function).

AudioPlayer.js – Handles song playback and controls.

Props: songUrl (string), onPlay (function).

Reusable Components:

Navbar.js – Navigation bar with links to Home, Playlist, and Favorites.

Button.js – Custom button component for consistent styling.

Props: label (string), onClick (function), variant (string).

SearchBar.js – Allows users to search for songs.

Props: onSearch (function).

SongCard.js – Displays individual song details.

Props: title, artist, genre, imageUrl, onPlay, onAddToPlaylist, onAddToFavorites.

8. State Management

Global State:

The application uses **React Context API** for managing global state, ensuring seamless state sharing across components. The global state includes:

Songs Data – Maintains the list of available songs.

Favorites List – Tracks favorited songs for easy access.

Playlist Data – Stores user-created playlists.

State is managed using useContext and useReducer, ensuring efficient updates and avoiding prop drilling.

Local State:

Each component handles its own local state using useState for UI interactions, including:

Search Term – Stored in SearchBar.js to filter songs dynamically.

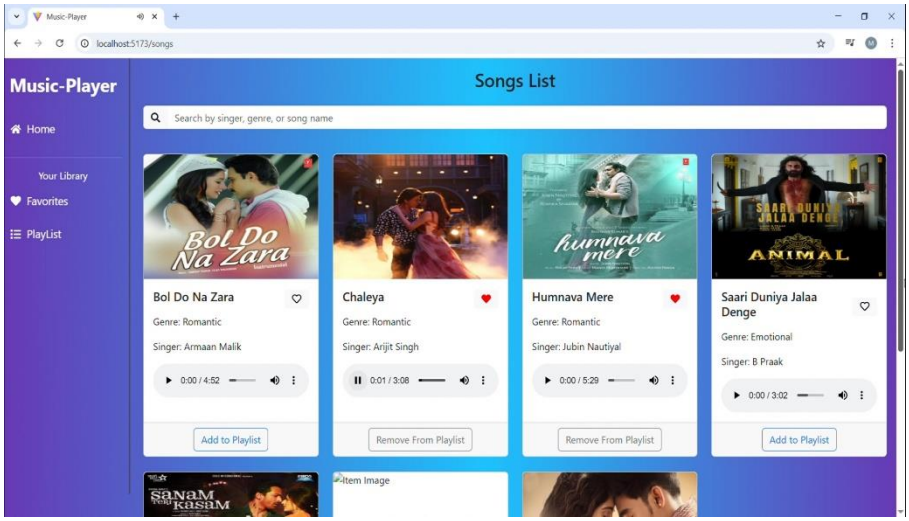
Audio Playback – Managed in AudioPlayer.js to control the currently playing song.

Form Inputs – Used in playlist creation or user preferences.

This hybrid approach balances efficiency, keeping **global data centralized** while maintaining **local control** for individual component behavior.

9. User Interface

Home page



10. Styling

CSS Frameworks/Libraries:

The project utilizes **Tailwind CSS** for rapid styling and responsiveness, alongside **Bootstrap** for prebuilt UI components. Additionally, **React Icons** enhances the UI with scalable vector icons.

Theming:

A **custom theming system** is implemented using Tailwind's utility classes, ensuring a **modern and cohesive** design. Users experience a **consistent UI** across light and dark modes, with dynamic color palettes applied globally for **branding and accessibility**.

11. Testing

The project follows a **comprehensive testing approach** using:

Unit Testing – Conducted with **Jest** and **React Testing Library** to validate individual component behavior.

Integration Testing – Ensures proper interaction between components and state management.

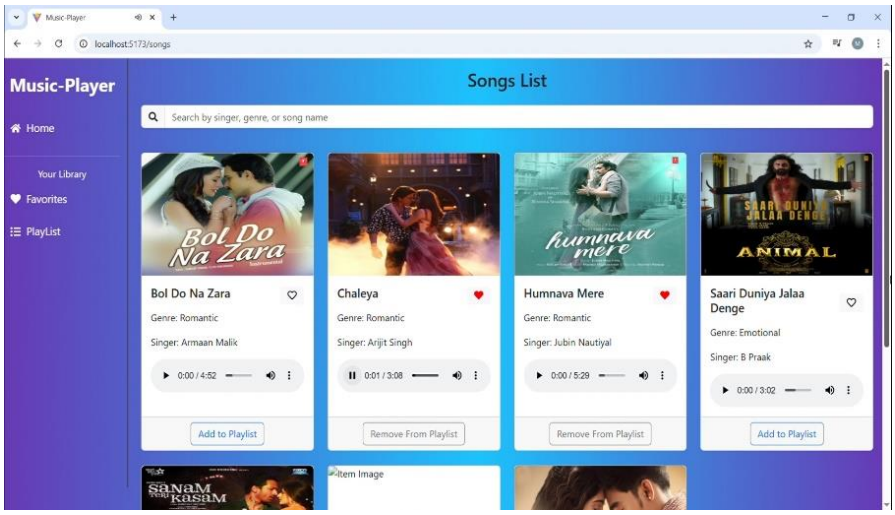
End-to-End (E2E) Testing – Uses **Cypress** to simulate real user workflows, ensuring smooth navigation and functionality.

Code Coverage:

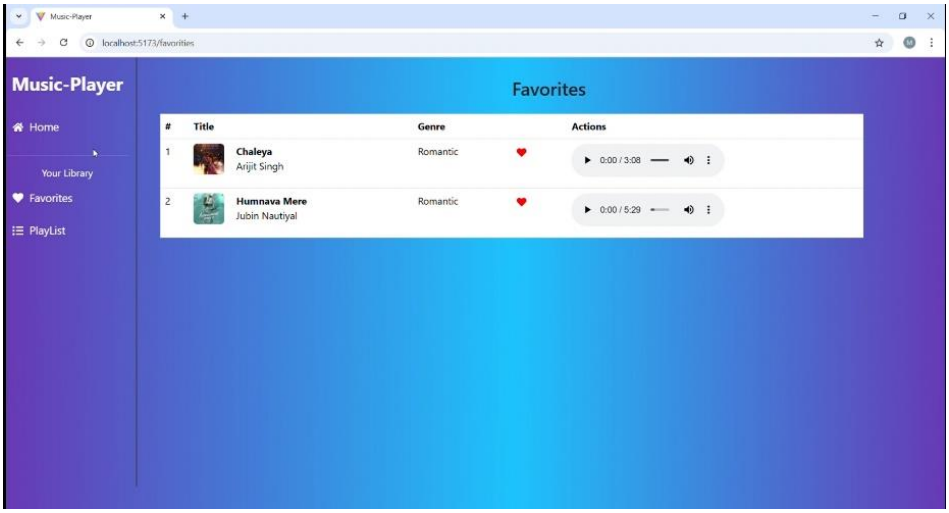
Jest's built-in coverage tool is used to measure test completeness, highlighting untested code. The goal is to maintain **high coverage across critical components** like song playback, playlist management, and search functionality.

12. Screenshots or Demo

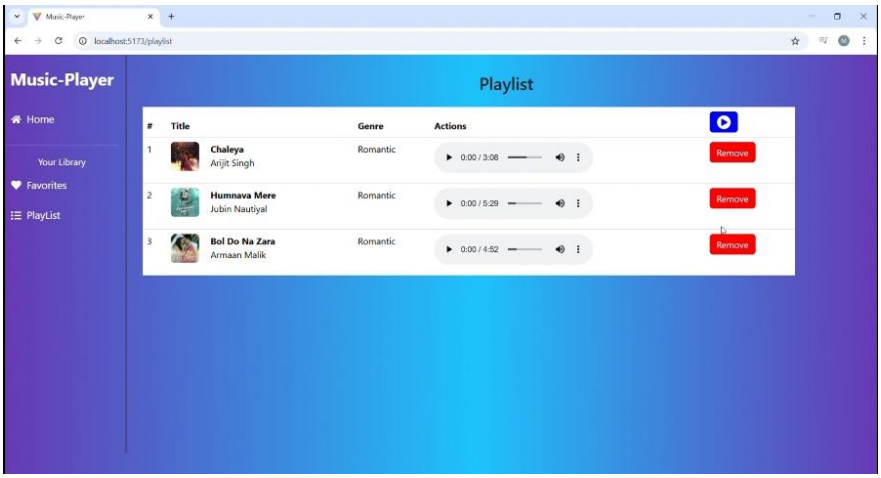
Home page



Favorites



Playlist



Demo Link:

https://drive.google.com/drive/folders/1_2c5Au4dRqbf8P2Wd5h5zdsAu4IjluCj

13. Known Issues

Audio Playback Delay – Some users may experience a slight delay when switching songs.

Offline Mode Limitations – Downloaded songs may not persist across all devices.

Search Optimization – The search function could be improved for better accuracy and speed.

UI Responsiveness – Minor layout inconsistencies on smaller screens.

14. Future Enhancements

AI-Powered Recommendations – Personalized song suggestions based on user preferences.

Dark Mode Toggle – Enhanced UI customization with a manual theme switch.

Animated Transitions – Smooth UI interactions with modern animations.

Social Features – Allow users to share playlists and follow friends.

Progressive Web App (PWA) Support – Enable offline playback and mobile-friendly experiences.

.