**RHYTHMIC TUNES**

TEAM ID : SWTID1741163103158096

TEAM SIZE : 4

TEAM MEMBERS:

Akshaya R

Gnaanadharshini K

Janani J

Krithika T.S

**PROJECT OVERVIEW**

**PURPOSE:**

Our React-based Music Streaming Application is designed to revolutionize the way users discover, experience, and interact with music. By leveraging modern web technologies, we aim to provide a seamless and immersive platform that caters to every music lover’s needs.

**GOALS AND OBJECTIVES:**

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.

**ARCHITECTURE**

**COMPONENT STRUCTURE:**

Favourites.jsx

* Fetch Favourite Songs
* Display Favourite Songs
* Play Audio with Pause Control
* Remove Songs From Favourites

Playlist.jsx

* Fetch Playlist Data
* Play/Pause a Song
* Remove a Song from the Playlist
* Play All Songs Sequentially

Search.jsx

* User input for Searching
* Displays Search Results
* User Friendly UI

Songs.jsx

* Fetch Songs,Favourites,and Playlist Data
* Search Songs
* Play Songs With Audio Control
* Manage Favourite Songs(Wishlist)
* Manage Playlist

App.jsx

* Sets up Routing
* Includes a Sidebar for Navigation
* Loads Bootstrap for Styling

Main.jsx

* Creates a Root Render
* Renders the App Component
* Includes Global Styling

**STATE MANAGEMENT:**

The project primarily uses React's built-in state management via the useState and useEffect hooks.

useState

* Local Component State
* Each component manages its own state using useState().
* Songs.jsx manages song data, wishlist, playlist, search term, and currently playing audio and Playlist.jsx and Favorities.jsx maintain their respective song lists.

useEffect

* Side Effects & Data Fetching (useEffect)
* useEffect() is used for fetching data from APIs (e.g., fetching songs, playlists, and favorites).
* It also handles event listeners for audio playback.

**ROUTING:**

The project uses React Router (react-router-dom) to handle navigation between different pages. The routing structure is defined in App.jsx using BrowserRouter, Routes, and Route.

**SETUP INSTRUCTIONS**

**PREREQUISITES:**

* Node.js and npm
* React.js
* HTML, CSS and JavaScript
* Version Control
* Development Environment (Visual Studio Code/ Sublime Text/ WebStorm)

**INSTALLATION:**

● Create a new React app:

* npm create vite@latest
* Enter and then type project-name and select preferred frameworks and then enter

● Navigate to the project directory:

* cd project-name
* npm install

● Running the React App:

* With the React app created, you can now start the
* development server and see your React application in action.

● Start the development server:

* npm run dev
* This command launches the development server, and you can access your React app at http://localhost:5173 in your web browser.

**FOLDER STRUCTURE**

**CLIENT:**

The project follows a structured folder organization, making it scalable and maintainable. Below is the typical folder structure:

/project-root

/public : Static assets (index.html, icons, etc.)

/src : Main source code

/assets : Images, icons, fonts, etc.

/components : Reusable UI components (Sidebar, Song Card, etc.)

/pages : Page-level components (Songs, Playlist, Favorities)

/utils : Utility functions or custom hooks (if any)

App.jsx : Main component (sets up routing)

index.jsx : Entry point of the application

index.css : Global styles

package.json : Dependencies and scripts

README.md : Project documentation

**UTILITIES: Helper Functions and Custom Hooks Used**

Helper Functions (/utils/helpers.js)

* Example: A function to check if a song exists in a list (used in both Favorites and Playlist).

Custom Hook (/utils/useFetch.js)

* If API fetching logic is repeated in multiple components (Songs.jsx, Playlist.jsx, etc.), a custom hook can improve reusability.

**RUNNING THE APPLICATION**

To start the frontend application

* npm start

**COMPONENT DOCUMENTATION**

**KEY COMPONENTS:**

Songs.jsx (Main Songs Page)

* Purpose: Displays a list of songs with features like search, favorites, and playlist management.
* Props: None (fetches data internally from API).
* Main Functionalities:

✔ Fetches songs from http://localhost:3000/playlist.  
 ✔ Allows users to remove songs from the playlist.  
 ✔ Includes a **Play All** button to play all songs sequentially.  
 ✔ Ensures only one song plays at a time.

Favorities.jsx (Favorites Page)

* Purpose: Displays a list of favorite songs
* Props: None (fetches data internally from API).
* Main Functionalities:  
  ✔ Fetches user's favorite songs from http://localhost:3000/favorities.  
  ✔ Allows users to remove songs from favorites.  
  ✔ Ensures only one song plays at a time.

Playlist.jsx (User Playlist Page)

* Purpose: Displays a custom playlist created by the user.
* Props: None (fetches data internally from API)
* Main Functionalities:  
  ✔ Fetches songs from http://localhost:3000/playlist.  
  ✔ Allows users to remove songs from the playlist.  
  ✔ Includes a Play All button to play all songs sequentially.  
  ✔ Ensures only one song plays at a time.

Search.jsx (Search Bar Component)

* Purpose: Allows users to search for songs by title, singer, or genre.
* Props: None (search logic is handled internally).
* Main Functionalities:  
  ✔ Takes user input and filters songs dynamically.  
  ✔ Displays matching results in a grid format.

Sidebar.jsx (Navigation Sidebar)

* Purpose: Provides navigation links to different sections.
* Props: None.
* Main Functionalities:  
  ✔ Contains links to Songs, Favorites, and Playlist pages.  
  ✔ Stays persistent across all pages.

**REUSABLE COMPONENTS:**

SongCard.jsx (Reusable Song Display Card)

* Purpose: Displays individual song details in a card format.
* Main Functionalities:  
  ✔ Displays song image, title, genre, and singer.  
  ✔ Includes play/pause functionality for the song.  
  ✔ Shows favorite (❤️) and playlist (📂) icons to manage songs.  
  ✔ Used in Songs, Favorities, and Playlist pages.

AudioPlayer.jsx (Reusable Audio Player Component)

* Purpose: Handles song playback, ensuring only one song plays at a time.
* Main Functionalities:  
  ✔ Ensures only one song plays at a time.  
  ✔ Displays a customized audio player.  
  ✔ Used in Songs, Favorites, and Playlist pages.

FavoriteButton.jsx (Reusable Favorite Button)

* Purpose: Toggles a song in and out of the favorites list.
* Main Functionalities:  
  ✔ Shows filled ❤️ (if in favorites) or empty 🖤 (if not).  
  ✔ Calls the provided toggleFavorite function on click.  
  ✔ Used in Songs and Favorities pages.

PlaylistButton.jsx (Reusable Playlist Button)

* Purpose: Toggles a song in and out of the user's playlist.
* Main Functionalities:  
  ✔ Shows "Add to Playlist" or "Remove from Playlist" based on state.  
  ✔ Calls the provided togglePlaylist function on click.  
  ✔ Used in Songs and Playlist pages.

**STATE MANAGEMENT**

**GLOBAL STATE: How State Flows Across the Application**

The project does not use a centralized global state like Redux or Context API. Instead, it relies on:  
✅API-driven state management → Each component fetches and manages its own state from a backend API (http://localhost:3000/).  
✅ Prop drilling (Minimal) → Some state (like song ID or functions) is passed as props to child components.

**LOCAL STATE: How Components Handle State**

Each component manages its own state using useState().

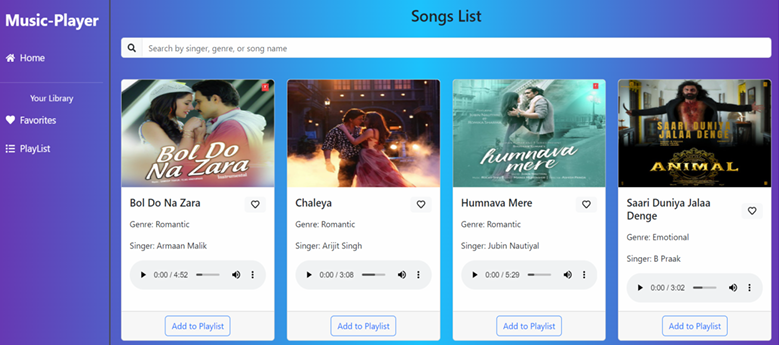
setItems() updates songs when fetched from the API.

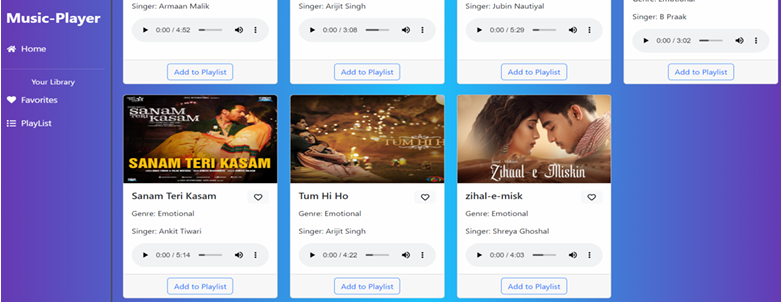
setWishlist() and setPlaylist() update state when songs are added/removed.

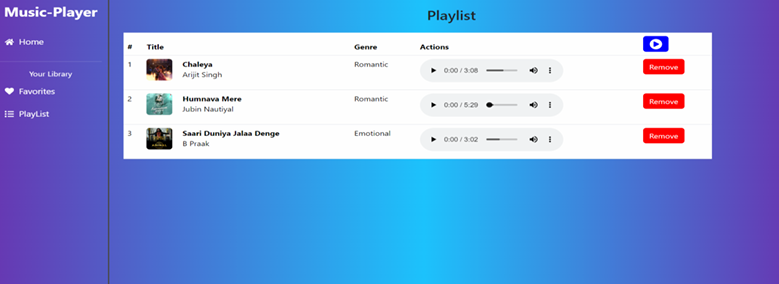
setCurrentlyPlaying() ensures only one song plays at a time.

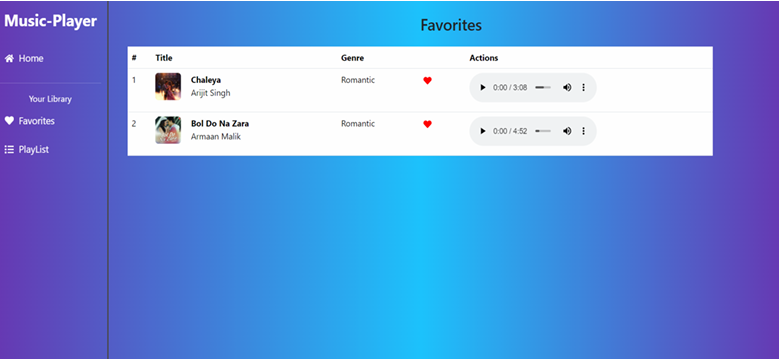
setSearchTerm() updates when the user types in the search bar.

**USER INTERFACE**









**STYLING**

**CSS Frameworks/Libraries:**

* Using Styled-Components for modular styling
* Material UI for pre-designed components

**Theming:**

* Light and Dark mode support
* Custom color palette for branding

**TESTING**

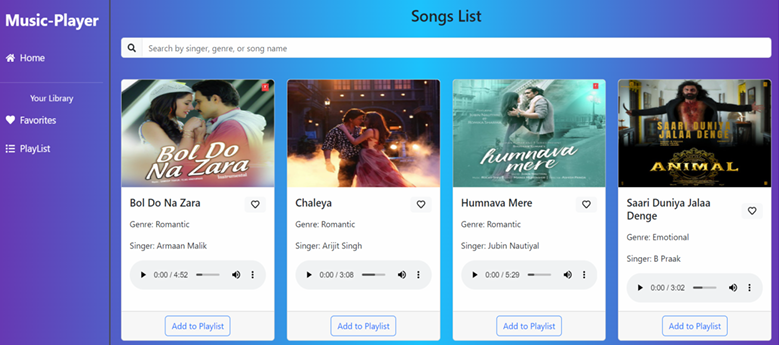
**TESTING STRATEGY:**

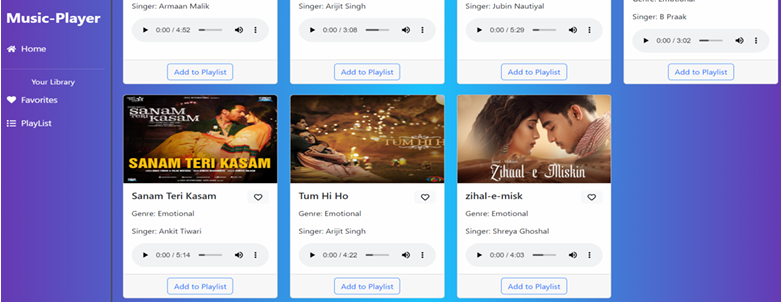
* Unit Testing using Jest
* Component Testing with React Testing Library

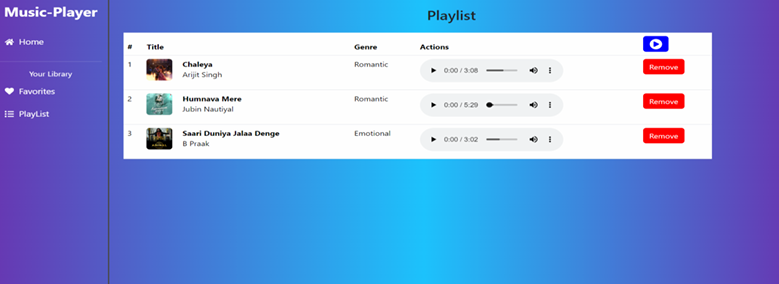
**CODE COVERAGE:**

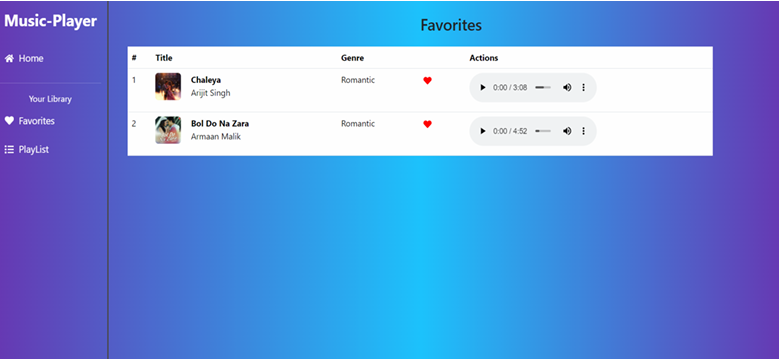
* Coverage reports generated using Jest

**SCREENSHOTS / DEMO**









**KNOWN ISSUES**

* Each time a song is added/removed from favorites or playlist, the entire list is fetched again from the API.
* This can cause unnecessary network requests and slow performance.
* The currentlyPlaying state ensures only one song plays at a time, but it may not always pause previous audio due to event listener inconsistencies.
* If an API request fails (e.g., due to a server issue or network problem), the UI does not provide feedback.
* Example: If a song fails to add to favorites, the user sees no error message.

**FUTURE ENHANCEMENTS**

* User Authentication (Sign Up/Login) using JWT or Firebase Auth.
* Store User Preferences (e.g., theme settings, last played song).
* Each user should have their own playlist & favorites stored in the database.