RYTHMIC TUNES: YOUR MELODIC COMPANION

**INTRODUCTION**:

Rhythmic tunes are musical compositions that emphasize the pattern of beats and timing to create movement and flow in music. Rhythm is one of the fundamental elements of music, along with melody and harmony. It provides structure, pace, and energy to a piece, making it engaging and memorable.

In rhythmic tunes, the arrangement of sounds and silences in a recurring pattern helps listeners connect with the music on both a physical and emotional level. These tunes are often used in dance, traditional music, and modern genres such as hip-hop, jazz, electronic, and classical compositions.

The power of rhythmic tunes lies in their ability to influence mood, encourage motion, and bring people together through coordinated beats. Whether it’s the steady pulse of a drum, the swing of jazz, or the syncopation in folk music, rhythm serves as the heartbeat of all musical experiences.

**PROJECT TITLE: RYTHMIC TUNES:YOUR MELODIC COMPANION**

**Team Members:**

* TEAM LEADER: NANDHINI R
* SUREKA S
* KARTHIKA S
* TAMILSELVI M

**PROJECT OVERVIEW:**

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

**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.

**ARCHITECTURE:**

Architecture defines the overall design and organization of the application. It includes how components, services, and data flow interact. A well-structured architecture improves scalability, maintainability, and performance.

* **Layers**: UI (components), Business logic, Data (API/database).
* **Approach**: Can be **monolithic** (everything together) or **modular** (separate modules for flexibility).
* **Best practice**: Follow separation of concerns (UI, logic, and data should be independent).

**SETUP INSTRUCTIONS:**

Here are the key prerequisites for developing a frontend application using React.js:

* Node.js and n pm: Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the local environment. It provides a scalable and efficient platform for building network applications.
  + Install Node.js and n pm on your development machine, as they are required to run JavaScript on the server-side.
  + Download: <https://nodejs.org/en/download/>
  + Installation instructions: <https://nodejs.org/en/download/package-manager/>
* React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

* **Create a new React app:**

N pm create-react-app my-react-app

Replace my-react-app with your preferred project name.

* **Navigate to the project directory:**

**cd my-react-app**

* **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 start**

**This command launches the development server, and you can access your React app at http://localhost:3000 in your web browser.**

* HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.
* Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like Git Hub or Bit bucket can host your repository.
  + Git : Download and installation instructions can be found at: <https://git-scm.com/downloads>
* Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or Web Storm.
  + Visual Studio Code: Download from <https://code.visualstudio.com/download>
  + Sublime Text: Download from <https://www.sublimetext.com/download>
  + Web Storm: Download from [https://www.jetbrains.com/webstorm/download](https://www.sublimetext.com/download)

Follow below steps:

* **Install Dependencies:**
  + Navigate into the cloned repository directory and install libraries:

**C d tunes**

**n pm install**

* **Start the Development Server:**
  + To start the development server, execute the following command:

**N pm start**

* Access the App: Open your web browser and navigate to http://localhost:3000.
* You should see the rythamic tunes app's homepage, indicating that the installation and setup were successful.

You have successfully installed and set up the application on your local machine. You can now proceed with further customization, development, and testing as needed.

**FOLDER STRUCTURE:**

* + **1. Install required tools and software:**
    - **Installation of required tools**:
    - Open the project folder to install necessary tools

In this project, we use:

* + - React Js
    - React Router Dom
    - React Icons
    - Bootstrap/tailwind css
    - Axios

* + - For further reference, use the following resources
    - <https://react.dev/learn/installation>
    - <https://react-bootstrap-v4.netlify.app/getting-started/introduction/>
    - <https://axios-http.com/docs/intro>
    - <https://reactrouter.com/en/main/start/tutorial>

RUNNING THE APPLICTION:

* **1. Setup React Application:**

• Create React application.

• Configure Routing.

• Install required libraries.

Setting Up Routes:-

**Code Description:-**

* + Imports Bootstrap CSS (bootstrap/dist/css/bootstrap.min.css) for styling components.
  + Imports custom CSS (./App.css) for additional styling.
  + Imports BrowserRouter, Routes, and Route from react-router-dom for setting up client-side routing in the application.
  + Defines the App functional component that serves as the root component of the application.
  + Uses BrowserRouter as the router container to enable routing functionality.
  + Includes a div as the root container for the application.
  + Within BrowserRouter, wraps components inside two div containers:
    - The first div contains the Sidebar component, likely serving navigation or additional content.
    - The second div contains the Routes component from React Router, which handles rendering components based on the current route.
    - Inside Routes, defines several Route components:
      * Route with path='/' renders the Songs component when the root path is accessed (/).
      * Route with path='/favorities' renders the Favorities component when the /favorities path is accessed.
      * Route with path='/playlist' renders the Playlist component when the /playlist path is accessed.
  + Exports the App component as the default export, making it available for use in other parts of the application.

**Fetching Songs:-**

**Code Description:-**

* + **Use State:**
    - items: Holds an array of all items fetched from http://localhost:3000/items.
    - Wish list: Stores items marked as favorites fetched from http://localhost:3000/favorities.
    - Play list: Stores items added to the playlist fetched from http://localhost:3000/playlist.
    - Currently Playing: Keeps track of the currently playing audio element.
    - Search Term: Stores the current search term entered by the user.
  + **Data Fetching:**
    - Uses use Effect to fetch data:
      * Fetches all items (items) from http://localhost:3000/items.
      * Fetches favorite items (wish list) from http://localhost:3000/favorities.
      * Fetches playlist items (playlist) from http://localhost:3000/playlist.
    - Sets state variables (items, wish list, playlist) based on the fetched data.
  + **Audio Playback Management:**
    - Sets up audio play event listeners and cleanup for each item:
      * Handle Audio Play: Manages audio playback by pausing the currently playing audio when a new one starts.
      * Handle Play: Adds event listeners to each audio element to trigger handle Audio Play.
    - Ensures that only one audio element plays at a time by pausing others when a new one starts playing.
  + **Add To Wish list (item Id):**
    - Adds an item to the wish list (favorities) by making a POST request to http://localhost:3000/favorities.
    - Updates the wish list state after adding an item.
  + **Remove From Wish list(item Id):**
    - Removes an item from the wish list (favorities) by making a DELETE request to http://localhost:3000/favorities/{itemId}.
    - Updates the wish list state after removing an item.
  + **Is Item In Wish list(item Id):**
    - Checks if an item exists in the wish list (favorities) based on its item Id.
  + **Add To Playlist(item Id):**
    - Adds an item to the playlist (playlist) by making a POST request to http://localhost:3000/playlist.
    - Updates the playlist state after adding an item.
  + **Remove From Playlist(item Id):**
    - Removes an item from the playlist (playlist) by making a DELETE request to http://localhost:3000/playlist/{itemId}.
    - Updates the playlist state after removing an item.
  + **Is Item In Playlist(item Id):**
    - Checks if an item exists in the playlist (playlist) based on its item Id.
  + **Filtered Items:**
    - Filters items based on the search Term.
    - Matches title, singer, or genre with the lowercase version of search Term.
  + **JSX:**
    - Renders a form with an input field (Form, Input Group, Button, FaSearch) for searching items.
    - Maps over filtered Items to render each item in the UI.
    - Includes buttons (FaHeart, FaRegHeart) to add/remove items from wish list and playlist.
    - Renders audio elements for each item with play/pause functionality.
  + **Error Handling:**
    - Catches and logs errors during data fetching (axios.get).
    - Handles errors when adding/removing items from wish list and playlist.

**Frontend Code For Displaying Songs:-**

**Code Description:-**

* + **Container Setup:**
    - Uses a div with inline styles (style={{display:"flex", justify Content:"flex-end"}}) to align the content to the right.
    - The main container (songs-container) has a fixed width (width:"1300px") and contains all the UI elements related to songs.
  + **Header:**
    - Displays a heading (<h2>) with text "Songs List" centered (class Name="text-3xl font-semi bold mb-4 text-center").
  + **Search Input:**
    - Utilizes Input Group from React Bootstrap for the search functionality.
    - Includes an input field (Form. Control) that allows users to search by singer, genre, or song name.
    - Binds the input field value to search Term state (value={search Term}) and updates it on change (on Change={(e) => set Search Term(e.target.value)}).
    - Styled with class Name="search-input".
  + **Card Layout:**
    - Uses Bootstrap grid classes (row, col) to create a responsive card layout (class Name="row row-cols-1 row-cols-md-2 row-cols-lg-3 row-cols-xl-4 g-4").
    - Maps over filtered Items array and renders each item as a Bootstrap card (<div class Name="card h-100">).
  + **Card Content:**
    - Displays the item's image (<img>), title (<h5 class Name="card-title">), genre (<p class Name="card-text">), and singer (<p class Name="card-text">).
    - Includes an audio player (<audio controls class Name="w-100" id={audio-${item.id}}>) for playing the song with a source (<source src={item.songUrl} />).
  + **Wish list and Playlist Buttons:**
    - Adds a heart icon button (<Button>) to add or remove items from the wishlist (isItemInWishlist(item.id) determines which button to show).
    - Includes an "Add to Playlist" or "Remove From Playlist" button (<Button>) based on whether the item is already in the playlist (isItemInPlaylist(item.id)).
  + **Button Click Handlers:**
    - Handles adding/removing items from the wishlist (addToWishlist(item.id), removeFromWishlist(item.id)).
    - Manages adding/removing items from the playlist (addToPlaylist(item.id), removeFromPlaylist(item.id)).
  + **Card Styling:**
    - Applies Bootstrap classes (card, card-body, card-footer) for styling the card components.

Uses custom styles (rounded-top, w-100) for specific elements like images and audio players.

**COMPONENT DOCUMENTATION :**

# 🎶 Rhythmic Tunes – Component Documentation

### ****Header & Navigation****

* Provides access to Home, Search, Playlists, Profile.

### ****Music Player****

* Core controls: play, pause, skip, volume, progress.
* Displays current track info.

### ****Search****

* Allows searching songs, albums, artists.
* Supports filtering and suggestions.

### ****Playlists****

* Shows user-created and recommended playlists.
* Add/remove tracks easily.

### ****Recommendations****

* Suggests music based on history and preferences.

### ****Authentication****

* Login, Signup, Logout.
* Profile management (name, avatar, settings).

### ****Profile****

* Displays user details and saved playlists.
* Manages preferences (themes, subscription).

**USER INTERFACE:**

 **Home Screen**

* Featured songs, trending tracks, and quick access playlists.
* Minimal, clean grid layout.

 **Navigation Bar**

* Tabs for Home, Search, Playlists, Profile.
* Sticky for easy access.

 **Music Player (Bottom Bar / Full View)**

* Compact mini-player at bottom.
* Expands to full screen with album art, controls, and lyrics.

 **Search Page**

* Search bar with live suggestions.
* Results categorized by Songs, Albums, Artists.

 **Playlist Page**

* Scrollable list of tracks with album art.
* Options for shuffle, repeat, and add/remove songs.

 **Recommendations**

* Personalized section with “For You” suggestions.

 **Profile Page**

* User info, subscription status, saved playlists.
* Settings for themes, notifications, and logout.

**STYLING**:

 **Theme**: Dark mode by default, with light mode option.

 **Colors**: Deep backgrounds, vibrant accent for active states (e.g., play button, progress bar).

 **Typography**: Clean sans-serif, bold for headings, medium for labels.

 **Layout**: Grid and flex for responsive design, balanced spacing.

 **Components**: Rounded corners, soft shadows, hover/active effects.

 **Consistency**: Uniform button styles, icon set, and transitions.

TESTING:

**Project Implementation & Execution**

After completing the code, run the react application by using the command “n pm start” or “n pm run dev” if you are using vite.js

And the Open new Terminal type this command    “j son-server --watch  ./db/db .j son” to start the j son server too.

SCREENSHOTE OR DEMO:

<https://drive.google.com/file/d/1zZuq62lyYNV_k5uu0SFjoWa35UgQ4LA9/view?usp=drive_link>

KNOWN LSSUES:

 **Playback Lag**: Minor delays when switching tracks.

 **API Limitations**: Some music data may not load if API request fails.

 **Cross-Device Sync**: Playback state may not always update instantly across devices.

**FUTURE ENHANCEMENTS:**

 **Offline Mode**: Full song downloads and playback without internet.

 **Smart Recommendations**: AI-driven suggestions based on mood and activity.

 **Social Features**: Share playlists, follow friends, and collaborative playlist creation.