**RYTHMIC TUNES**

**Project Documentation**

1. **Introduction**
   * **Project Title**: RYTHMIC TUNES
   * **Team Members**:

i) Samuvel P (Email id:samuvel48016@gmail.com)

ii) Abishek G (Email id:abiabishek760@gmail.com)

iii) Bharath V (Email id:bharathvenkatesan24@gmail.com)

iv) Sinmayanandhan D (Email id:sinmayanandhan1412@gmail.com)

1. **Project Overview**
   * **Purpose**
   * The primary purpose of this music application is to provide users with a comprehensive and engaging platform for discovering, listening to, and sharing music. The app aims to achieve the following objectives:
   * **Music Discovery**: To help users discover new artists, genres, and songs tailored to their personal tastes through accurate playlists, recommendations, and trending music features.
   * **Seamless Listening Experience**: To offer a high-quality audio playback experience with an intuitive music player that allows users to easily navigate their music library, create playlists, and enjoy their favourite tracks without interruptions.
   * **Social Interaction**: To foster a community of music lovers by enabling users to share their playlists, follow friends, and engage with others through social media integration, thereby enhancing the overall music experience.
   * **Personalization**: To provide personalized music recommendations based on users' listening habits, preferences, and interactions within the app, ensuring that each user feels a unique connection to the music they love.
   * **Accessibility**: To ensure that users can access their music library anytime and anywhere, whether they are using a mobile device or a web browser, making music an integral part of their daily lives.
   * **User Empowerment**: To empower users to take control of their music experience by allowing them to create, manage, and share their playlists, thus encouraging creativity and self-expression through music.
   * **Offline Listening**: To provide users with the option to download their favorite songs and playlists for offline listening, ensuring that they can enjoy music without needing an internet connection.
   * **User Engagement**: To keep users engaged through features like music quizzes, challenges, and interactive content that encourages exploration and interaction with the app.
   * **Artist Support**: To create a platform that supports emerging and independent artists by providing them with a space to showcase their music and connect with fans, thereby promoting diversity in the music industry.

­­­­­­­­

* + **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.

1. **Architecture**
   * **Component Structure**:
   * Here are some components for developing a frontend application using React.js:

**Node.js and npm:**

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 npm 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:

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.

* 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 GitHub or Bitbucket can host your repository.
  + **Routing**:

1. **Overview of Routing**

Routing allows users to navigate between different parts of the application without reloading the page.

It helps in creating a single-page application (SPA) experience, where different views are rendered based on the URL.

1. **Key Routes**

Here are some common routes you might implement in a music application:

**Home Route (/)**

Displays featured playlists, trending songs, and recommended artists.

**Search Route (/search)**

Provides a search interface for users to find songs, artists, and albums.

**Library Route (/library)**

Shows the user's music library, including their songs, albums, and playlists.

**Playlist Route (/playlist/:id)**

Displays a specific playlist based on the playlist ID in the URL.

**Song Route (/song/:id)**

Shows detailed information about a specific song, including playback options.

**User Profile Route (/profile)**

Displays the user's profile, including their playlists and listening history.

**Settings Route (/settings)**

Allows users to manage their account settings and preferences.

1. **Setting Up Routing in React**



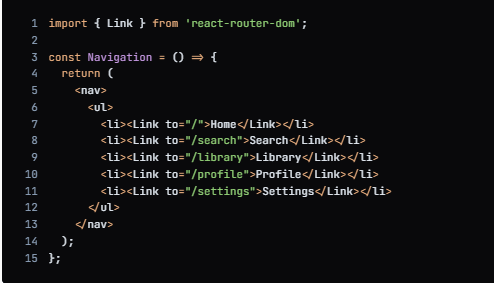
**4. Dynamic Routing**

Use dynamic routing to handle routes that require parameters, such as playlist or song IDs.

For example, the route /playlist/:id allows you to access a specific playlist based on its ID.

**5. Navigation Links**

Implement navigation links to allow users to navigate between different routes easily. You can use the Link component from React Router:



**6. Handling Not Found Routes**

Implement a catch-all route to handle 404 errors for undefined routes:

C:\Users\Admin\Pictures\music 2.PNG

1. **Setup Instructions**

● Installation of required tools:

Open the project folder to install necessary tools.In this project, we use:

* React Js
* React Router Dom
* React Icons
* Bootsrap/tailwind css
* Axios

● For further reference, use the following resources

* <https://react-bootstrap-v4.netlify.app/getting-started/introduction/>
* <https://axios-http.com/docs/intro>
* <https://reactrouter.com/en/main/start/tutorial>

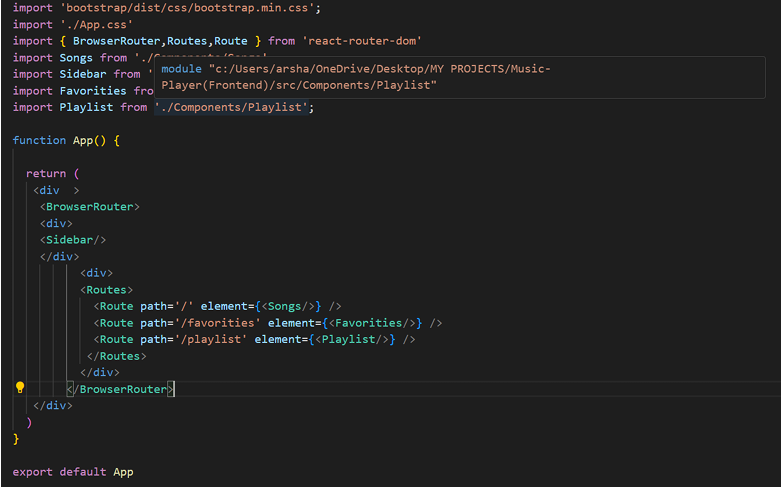
●Setup React Application:

• 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.

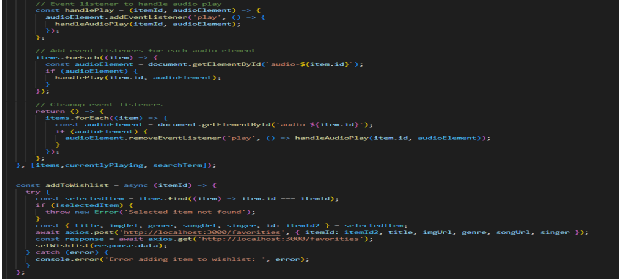
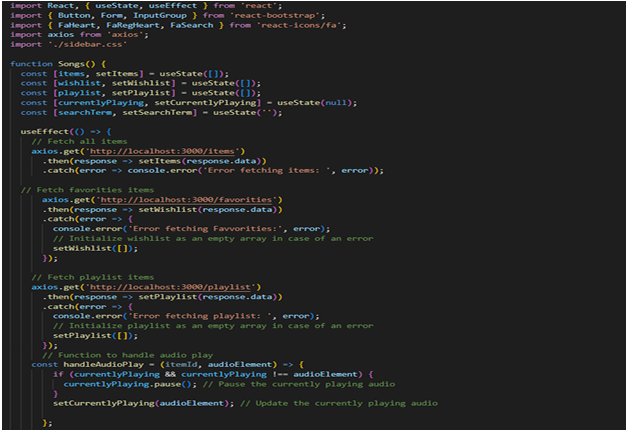
● Route with path='/playlist' renders the Playlist component when the

/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:-**

● useState:

●items: Holds an array of all items fetched from

http://localhost:3000/items.

● wishlist: Stores items marked as favorites fetched from

http://localhost:3000/favorities.

● playlist: Stores items added to the playlist fetched from

http://localhost:3000/playlist.

● currentlyPlaying: Keeps track of the currently playing audio element.

●searchTerm: Stores the current search term entered by the user.

● Data Fetching:

●Uses useEffect to fetch data:

▪ Fetches all items (items) from http://localhost:3000/items.

▪ Fetches favorite items (wishlist) from

http://localhost:3000/favorities.

▪ Fetches playlist items (playlist) from

http://localhost:3000/playlist.

●Sets state variables (items, wishlist, playlist) based on the fetched data.

● Audio Playback Management:

● Sets up audio play event listeners and cleanup for each item:

▪ handleAudioPlay: Manages audio playback by pausing the currently

playing audio when a new one starts.

▪ handlePlay: Adds event listeners to each audio element to trigger

handleAudioPlay.

●Ensures that only one audio element plays at a time by pausing others when a

new one starts playing.

● addToWishlist(itemId):

● Adds an item to the wishlist (favorities) by making a POST request to

http://localhost:3000/favorities.

● Updates the wishlist state after adding an item.

● removeFromWishlist(itemId):

●Removes an item from the wishlist (favorities) by making a DELETE request

to http://localhost:3000/favorities/{itemId}.

●Updates the wishlist state after removing an item.

● isItemInWishlist(itemId):

● Checks if an item exists in the wishlist (favorities) based on its itemId.

● addToPlaylist(itemId):

●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.

● removeFromPlaylist(itemId):

●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.

● isItemInPlaylist(itemId):

●Checks if an item exists in the playlist (playlist) based on its itemId.

● filteredItems:

● Filters items based on the searchTerm.

● Matches title, singer, or genre with the lowercase version of

searchTerm.

● JSX:

● Renders a form with an input field (Form, InputGroup, Button, FaSearch)

for searching items.

●Maps over filteredItems to render each item in the UI.

● Includes buttons (FaHeart, FaRegHeart) to add/remove items from

wishlist and playlist.

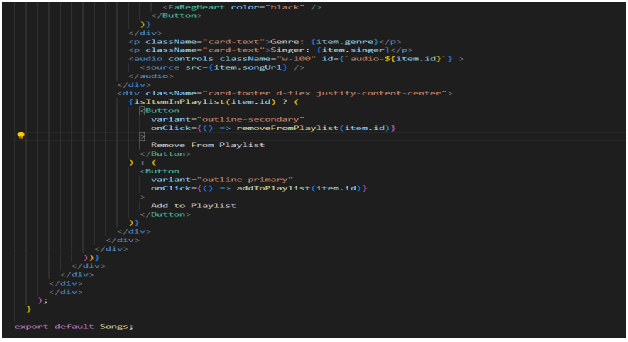
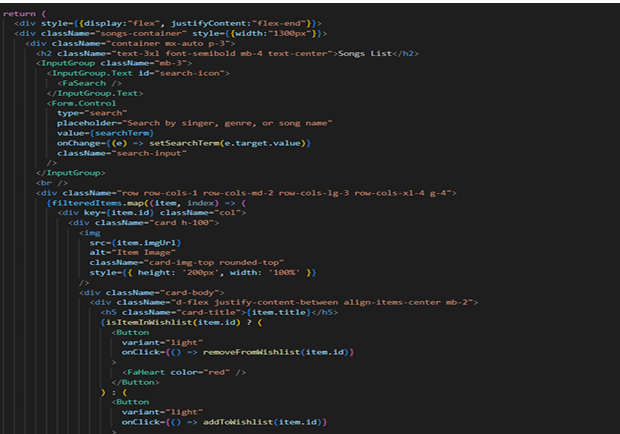
● Renders audio elements for each item with play/pause functionality.

● Error Handling:

●Catches and logs errors during data fetching (axios.get).

●Handles errors when adding/removing items from wishlist and playlist.

**Frontend Code For Displaying Songs:-**



**Code Description:-**

● Container Setup:

●Uses a div with inline styles (style={{display:"flex",

justifyContent:"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

(className="text-3xl font-semibold mb-4 text-center").

● Search Input:

● Utilizes InputGroup 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 searchTerm state (value={searchTerm})

and updates it on change (onChange={(e) =>

setSearchTerm(e.target.value)}).

● Styled with className="search-input".

● Card Layout:

● Uses Bootstrap grid classes (row, col) to create a responsive card layout

(className="row row-cols-1 row-cols-md-2 row-cols-lg-3

row-cols-xl-4 g-4").

● Maps over filteredItems array and renders each item as a Bootstrap card

(<div className="card h-100">).

● Card Content:

● Displays the item's image (<img>), title (<h5

className="card-title">), genre (<p className="card-text">),

and singer (<p className="card-text">).

● Includes an audio player (<audio controls className="w-100"

id={audio-${item.id}}>) for playing the song with a source (<source

src={item.songUrl} />).

● Wishlist 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.

**Project Execution:**

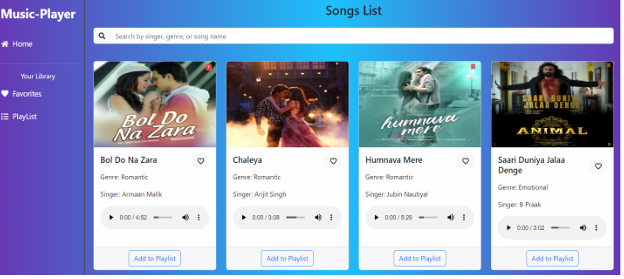
●After completing the code, run the react application by using the command “npm start” or

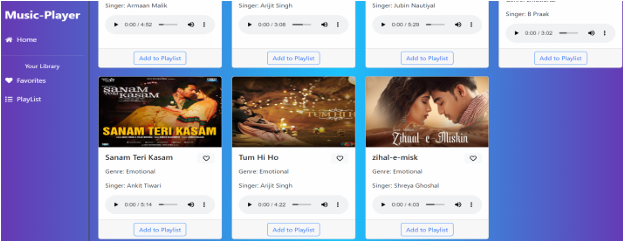
“npm run dev” if you are using vite.js

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

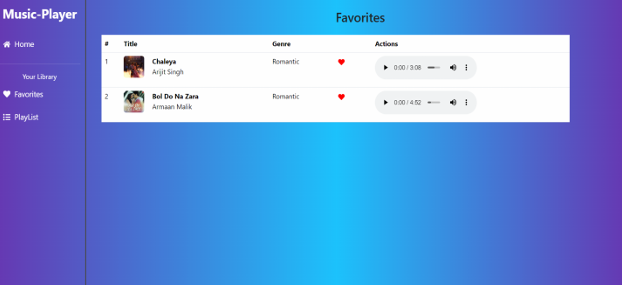
●After that launch the Rythimic Tunes.

Here are some of the screenshots of the application.

**Hero components :** 



**Favourites :**



**Playlist :**

