

TUNESV RHYTHMIC TUNES (REACT)

**DR.MGR-JANAKI COLLEGE OF ARTS & SCIENCE FOR
WOMEN**

B.Sc. COMPUTER SCIENCE (2022-2025)

PROJECT TITLE: TUNESV RYTHMIC TUNES(REACT)

TEAM MEMBERS:

NANDHIKA.R(asunm1423222208035)
Email:nandhika61024ramesh@gmail.com

HARINI.S(asunm1423222207994)
Email:harini6725@gmail.com

SANKARI.S(asunm1423222208020)
Email: devisankari2004@gmail.com

SUBASHREE.P(asunm1423222208022)
Email:dhinasuba02@gmail.com

INTRODUCTION

Welcome to the future of musical indulgence – an unparalleled audio experience awaits you with our cutting-edge Music Streaming Application, meticulously crafted using the power of React.js. Seamlessly blending innovation with user-centric design, our application is set to redefine how you interact with and immerse yourself in the world of music.

Designed for the modern music enthusiast, our React-based Music Streaming Application offers a harmonious fusion of robust functionality and an intuitive user interface. From discovering the latest chart-toppers to rediscovering timeless classics, our platform ensures an all-encompassing musical journey tailored to your unique taste.

The heart of our Music Streaming Application lies in React, a dynamic and feature-rich JavaScript library. Immerse yourself in a visually stunning and interactive interface, where every click, scroll, and playlist creation feels like a musical revelation. Whether you're on a desktop, tablet, or smartphone, our responsive design ensures a consistent and enjoyable experience across all devices.

SCENARIO-BASED INTRO:

Imagine stepping onto a bustling city street, the sounds of cars honking, people chatting, and street performers playing in the background. You're on your way to work, and you need a little something to elevate your mood. You pull out your phone and open your favorite music streaming app, "PROJECT NAME"

With just a few taps, you're transported to a world of music tailored to your tastes. As you walk, the app's smart playlist kicks in, starting with an upbeat pop song that gets your feet tapping. As you board the train, the music shifts to a relaxing indie track, perfectly matching your need to unwind during the commute.

PROBLEM STATEMENT:

- With the increasing demand for on-the-go music streaming, users often seek seamless, personalized, and accessible platforms to discover, manage, and enjoy their favorite music.
- However, creating such a platform involves overcoming challenges like intuitive user interfaces, efficient state management, API integration for music streaming, and ensuring responsive design across devices.
- This project aims to develop a Spotify-like music streaming application using React that replicates key features such as user authentication, playlist creation, music playback, and search functionality.
- By leveraging modern web technologies and APIs, the app will deliver a rich user experience and a robust, scalable architecture for real-world use.

PROPOSED SOLUTION:

- The (PROJECT NAME) project provides a comprehensive solution for building a music streaming web application by leveraging React for a modular and responsive UI, Spotify Web API for accessing music metadata, and Firebase or Node.js for user authentication and playlist management.
- The app integrates a fully functional music player with playback controls, a search feature for discovering songs and artists, and options to create and manage playlists.
- The solution to building a (PROJECT NAME) lies in systematically implementing the features and functionalities of a music streaming application using modern web development technologies

PROJECT OVERVIEW:

- The (project name) project is a music streaming web application designed to replicate core features of music player, providing users with an intuitive and engaging platform to discover, manage, and listen to their favorite music.
- This project serves as a hands-on implementation of modern web development concepts, including React, APIs, and responsive design, while also demonstrating how to build a scalable and user-friendly application

PROJECT 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

KEY 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.

PRE-REQUISITES:

1. 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/>

2. 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.

3. 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.

4. 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.

5. Development Environment:

Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- Visual Studio Code: Download from <https://code.visualstudio.com/download>
- Sublime Text: Download from <https://www.sublimetext.com/download>
- WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

PROJECT STRUCTURES:



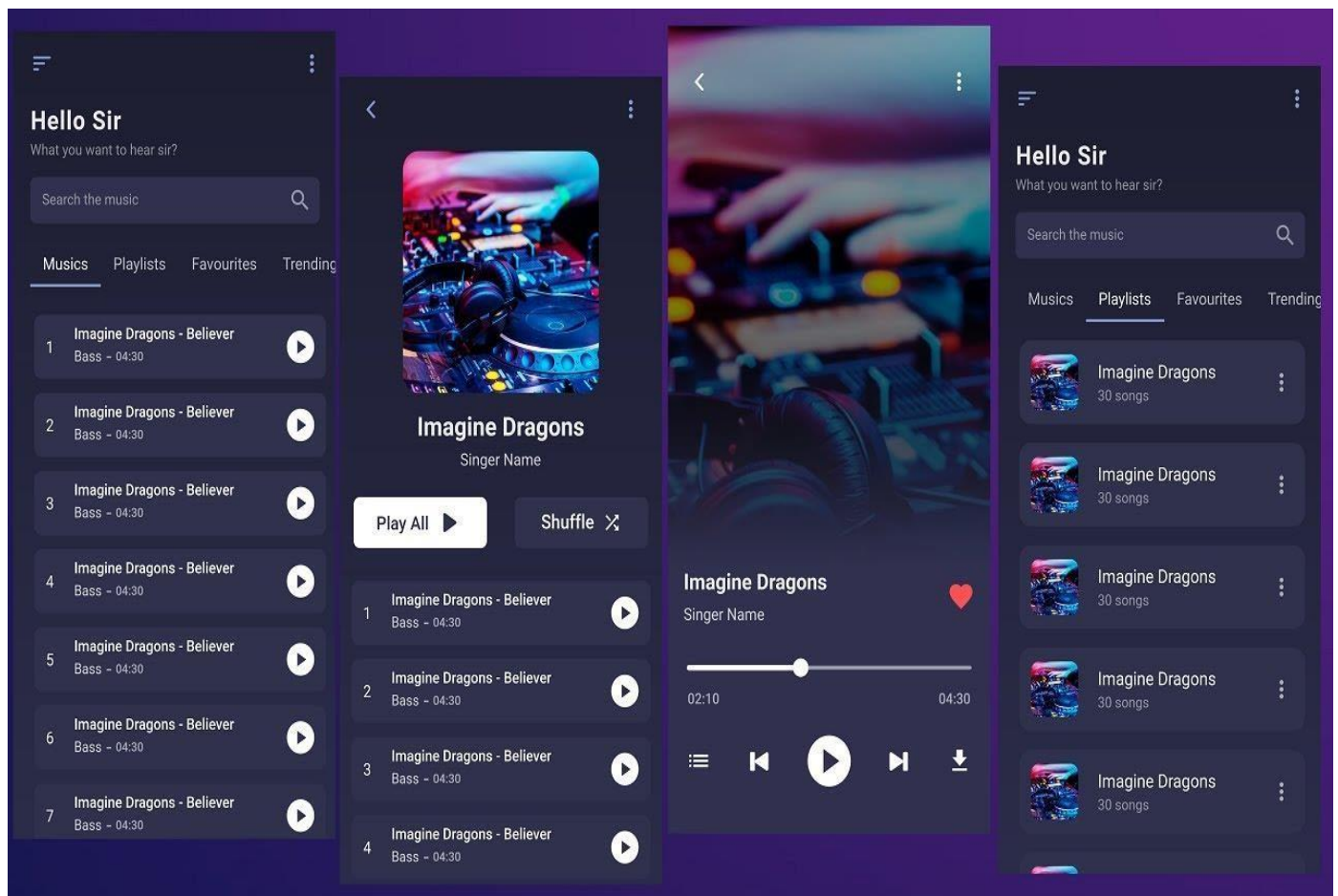
The project structure may vary depending on the specific library, framework, programming language, or development approach used. It's essential to organize the files and directories in a logical and consistent manner to improve code maintainability and collaboration among developers.

app/app.component.css, src/app/app.component: These files are part of the main App Component, which serves as the root component for the React app. The component handles the overall layout and includes the router outlet for loading different components based on the current route.

PROJECT MODEL: -

(IMAGES TO ADD)

- Here is an image of a visually appealing login and sign-up page designed for a Spotify clone project.
- It features a modern dark theme with green accents, a split-screen layout, and an intuitive user interface for both login and registration



PROJECT FLOW:

1. Project Setup and Configuration:

A. Installation of required tools:

- React js
- React router dom
- React icons
- Bootstrap/tailwind css
- Axios

B. For further reference, use the following resource,

Link1: <https://react.dev/learn/installation>

Link2: <https://axios-http.com/docs/intro>

Link3: <https://reactrouter.com/en/main/start/tutorial>

Project Development:

- Create React application.
- Configure Routing.
- Install required libraries

1. 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=/'favorites'` renders the `Favorites` component when the `/favorites` 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.

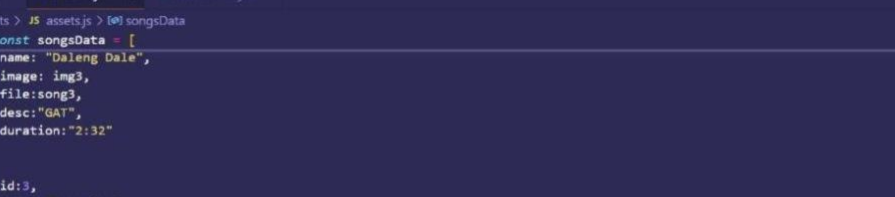
2. Fetching Songs:-

```

File Edit Selection View Go Run ... → ↺ ↻
DisplayAlbum.jsx JS assets.js DisplayNav.jsx X
tunesvi > src > components > DisplayNav > DisplayNav.jsx @ default
1 import React from 'react'
2 import { useNavigate } from 'react-router-dom';
3 import { assets } from '../assets/assets';
4
5 const DisplayNav = () => {
6
7     const navigate = useNavigate();
8
9     return (
10         <>
11             <div className='w-full flex justify-between items-center font-semibold'>
12                 <div className='flex items-center gap-2'>
13                     <img onClick={() => navigate(-1)} className='w-8 bg-black p-2 rounded-2xl cursor-pointer' src={assets.arrow_left} alt='' />
14                     <img onClick={() => navigate(1)} className='w-8 bg-black p-2 rounded-2xl cursor-pointer' src={assets.arrow_right} alt='' />
15                 </div>
16                 <div className='flex items-center gap-4'>
17                     <p className='bg-white text-black text-[15px] px-4 py-1 rounded-2xl hidden md:block'>Explore Premium</p>
18                     <p className='bg-black px-4 py-1 rounded-2xl text-[15px]'>Install App</p>
19                     <p className='bg-purple-500 text-black w-7 h-7 rounded-full flex items-center justify-center'>DX</p>
20                 </div>
21             </div>
22             <div className='flex items-center gap-2 mt-4'>
23                 <p className='bg-white text-black px-4 py-1 rounded-2xl cursor-pointer'>All</p>
24                 <p className='bg-black px-4 py-1 rounded-2xl cursor-pointer'>Music</p>
25                 <p className='bg-black px-4 py-1 rounded-2xl cursor-pointer'>Podcasts</p>
26             </div>
27         </>
28     )
29 }
30

```

Ln 31, Col 26 Spaces: 4 UTF-8 LF JavaScript JSX Go Live



```
122 export const songsData = [
141   name: "Daleng Dale",
142   image: img3,
143   file: song3,
144   desc: "GAT",
145   duration: "2:32"
146 },
147 {
148   id: 3,
149   name: "Dynamite",
150   image: img17,
151   file: song4,
152   desc: "BTS",
153   duration: "2:50"
154 },
155 {
156   id: 4,
157   name: "Nee Kavithaigalai",
158   image: img5,
159   file: song5,
160   desc: "Pradeep",
161   duration: "3:10"
162 },
163 {
164   id: 5,
165   name: "Run Bts",
166   image: img14,
167   file: song6,
168   desc: "Jungkook",
169   duration: "2:45"
```

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/favorites`.
 - 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/favorites`.
 - 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 (favorites) by making a POST request to `http://localhost:3000/favorites`.
 - Updates the wishlist state after adding an item.

- **removeFromWishlist(itemId):**

- Removes an item from the wishlist (favorites) by making a DELETE request to `http://localhost:3000/favorites/{itemId}`.
 - Updates the wishlist state after removing an item.

- **isItemInWishlist(itemId):**

- Checks if an item exists in the wishlist (favorites) 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.

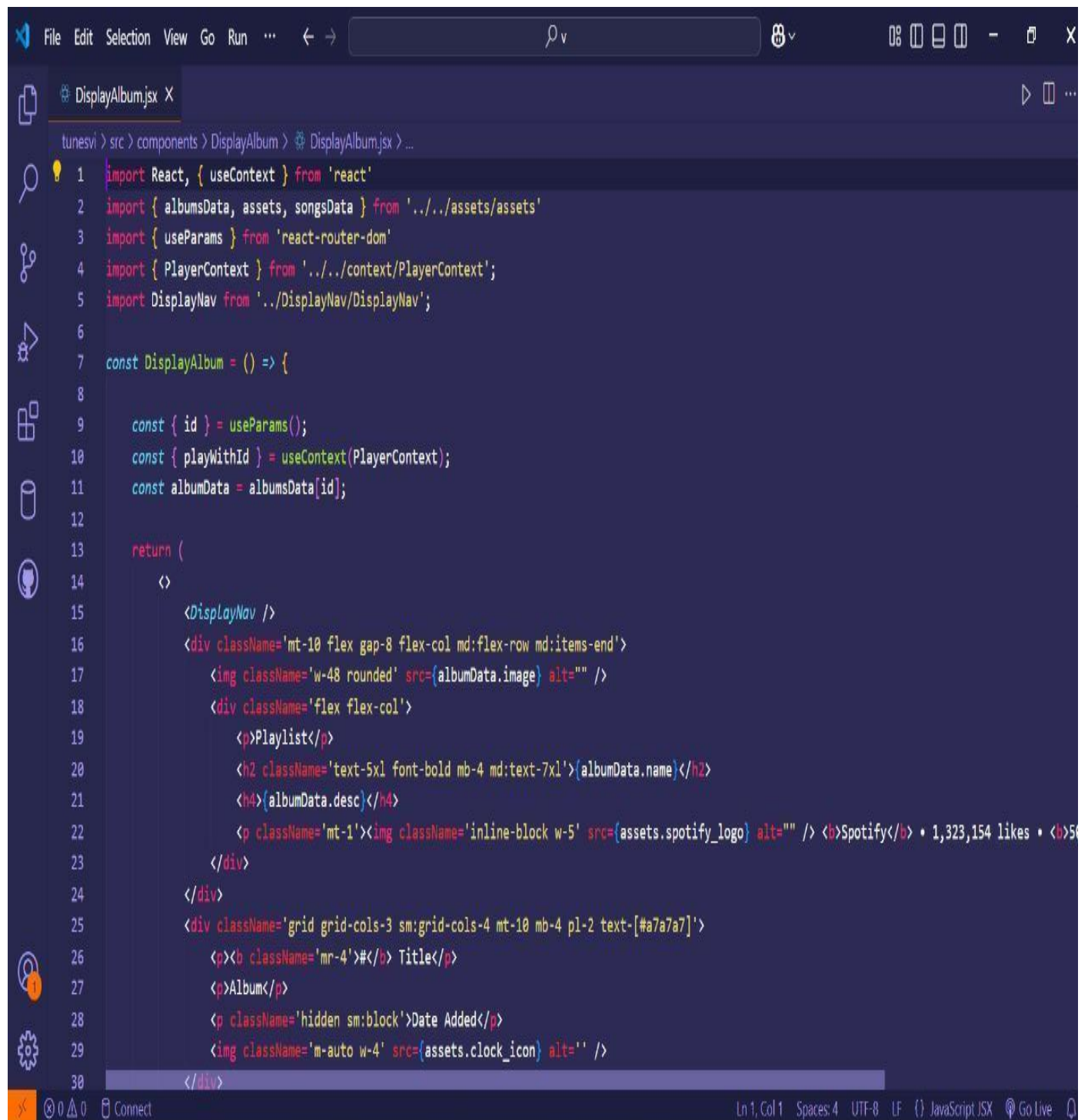
- **JSX:**

- 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 DISPLAY ABLUMS:-



```
1 import React, { useContext } from 'react'
2 import { albumsData, assets, songsData } from '../assets/assets'
3 import { useParams } from 'react-router-dom'
4 import { PlayerContext } from '../context/PlayerContext';
5 import DisplayNav from '../DisplayNav/DisplayNav';
6
7 const DisplayAlbum = () => {
8
9   const { id } = useParams();
10   const { playWithId } = useContext(PlayerContext);
11   const albumData = albumsData[id];
12
13   return (
14     <>
15       <DisplayNav />
16       <div className='mt-10 flex gap-8 flex-col md:flex-row md:items-end'>
17         <img className='w-48 rounded' src={albumData.image} alt='' />
18         <div className='flex flex-col'>
19           <p>Playlist</p>
20           <h2 className='text-5xl font-bold mb-4 md:text-7xl'>{albumData.name}</h2>
21           <h4>{albumData.desc}</h4>
22           <p className='mt-1'><img className='inline-block w-5' src={assets.spotify_logo} alt='' /> Spotify • 1,323,154 likes • <b>5</b>
23         </div>
24       </div>
25       <div className='grid grid-cols-3 sm:grid-cols-4 mt-10 mb-4 pl-2 text-[#a7a7a7]'>
26         <p><b>#</b> Title</p>
27         <p>Album</p>
28         <p className='hidden sm:block'>Date Added</p>
29         <img className='m-auto w-4' src={assets.clock_icon} alt='' />
30       </div>
31     </>
32   )
33 }
```



```
File Edit Selection View Go Run ... ← → ρ v ⚙ - □ X
Display.jsx X ▶ □ ...
tunesvi > src > components > Display > Display.jsx > ...
1 import React, { useEffect, useRef } from 'react'
2 import DisplayHome from '../DisplayHome/DisplayHome'
3 import { Route, Routes, useLocation } from 'react-router-dom'
4 import DisplayAlbum from '../DisplayAlbum/DisplayAlbum'
5 import { albumsData } from '../../assets/assets'
6
7 const Display = () => {
8
9   const displayRef = useRef();
10   const location = useLocation();
11   let isAlbum = location.pathname.includes("album");
12   let albumId = isAlbum ? location.pathname.slice(-1) : ""
13   let bgColor = albumsData[Number(albumId)].bgColor;
14
15   useEffect(() => {
16     if (isAlbum) {
17       displayRef.current.style.background = `linear-gradient(${bgColor},#121212)`;
18     }
19     else {
20       displayRef.current.style.background = "#121212";
21     }
22   })
23
24   return (
25     <div ref={displayRef} className='w-[100%] m-2 px-6 pt-4 rounded bg-[#121212] text-white overflow-auto lg:w-[75%] lg:m1-0'>
26       <Routes>
27         <Route path="/" element={<DisplayHome />} />
28         <Route path="/album/:id" element={<DisplayAlbum />} />
29       </Routes>
30     </div>
```


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 (``), 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:

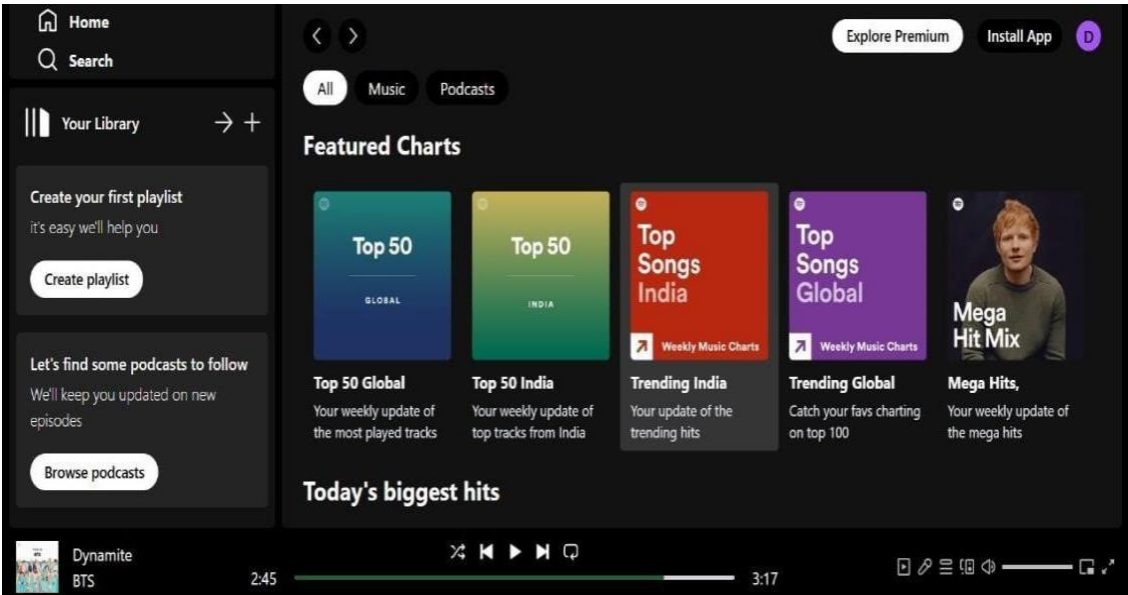
After completing the code, run the react application by using the command “npm start” or “npm run dev” if you are using vite.js

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

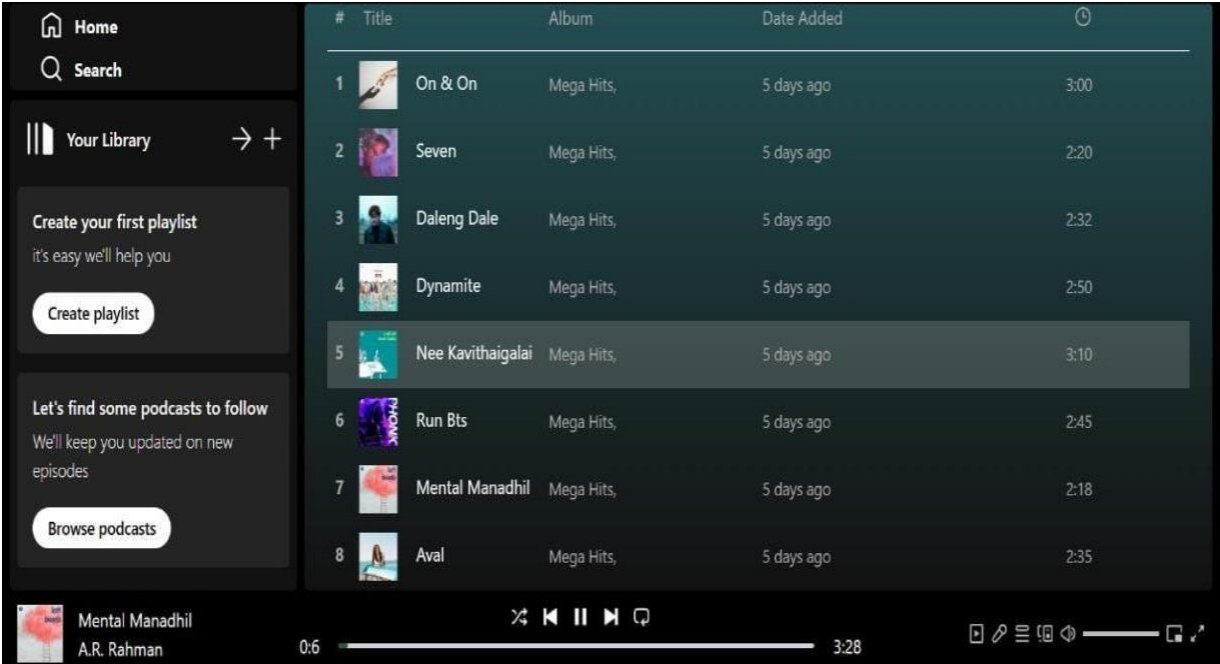
After that launch the Rhythmic Tunes.

Here are some of the screenshots of the application.

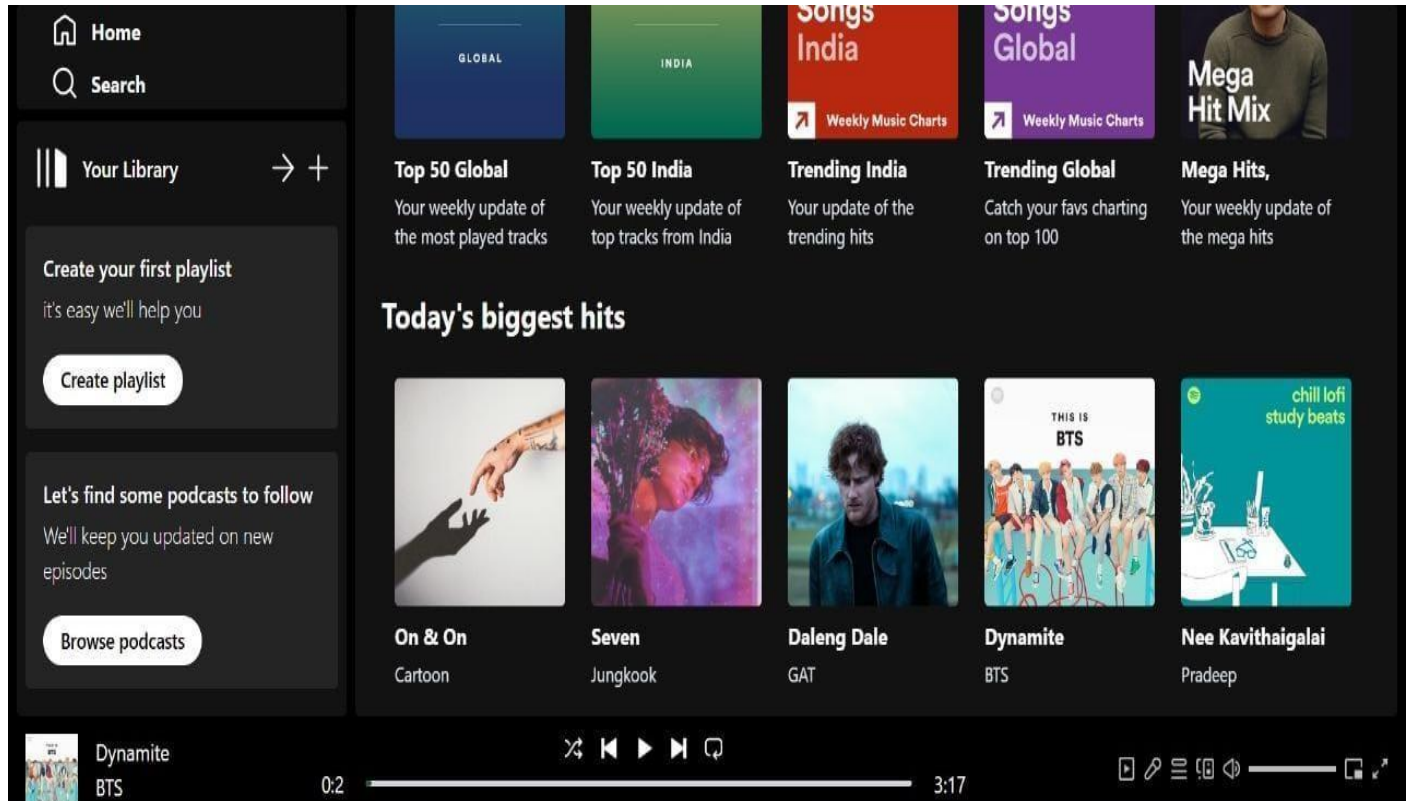
BEAT BLAST:



PLAYLIST:



FAVORITES :



PROJECT LINK:

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