

# ***Rhythmixx'*** ***(React)***

*A fusion of rhythm and mix, perfect for a dynamic music experience*

**DR.MGR JANAKI COLLEGE OF ARTS AND SCIENCE FOR WOMEN**  
**(B.Sc., Computer Science- Final Year)**

**Presented By:**

**Haritha. U(asunm1423222207995)**

Mail id-harithaammu887@gmail.com

**Preethi. B (asunm1423222208015)**

Mail id-preethibasker5@gmail.com

**Pavithra. D(asunm1423222208013)**

Mail id-pavithradhanasekar662@gmail.com

**Suji. N.D(asunm1423222208023)**

Mail id-sujidilli84@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.

Say goodbye to the limitations of traditional music listening and welcome a world of possibilities with our React-based Music Streaming Application. Join us on this journey as we transform the way you connect with and savor the universal language of music. Get ready to elevate your auditory experience – it's time to press play on a new era of music streaming.

In this project, the foundational idea is that rhythm and melody can function as "caches" or "repositories" for words and phrases. By associating each word or concept with a specific rhythm or tune, the brain can use the established musical patterns to trigger linguistic recall. This is similar to how certain songs can evoke memories or emotions tied to specific experiences, but in this case, the musical cues are tied to the words themselves.

The words are not just stored in isolation—they are embedded within carefully designed rhythmic structures that resonate with different cognitive and emotional states. These rhythms are tailored to individuals' personal preferences, learning styles, and even emotional responses to particular stimuli.

“MELODIES THAT MOVE, RHYTHMS THAT GROOVE.”

## 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, "Rhythmonics."

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 the commute.

The stage lights flicker as **Rhythmic Tunes** takes over, seamlessly blending jazz saxophones with deep electronic basslines. The crowd moves in sync with the pulsating beats, their energy rising as the tempo shifts. Suddenly, a hip-hop verse flows effortlessly over a classical string arrangement, creating a fusion no one expected yet instantly connects with. This is the essence of **Rhythmic Tunes** a boundary-pushing musical experience that transforms sound into emotion, uniting diverse styles and audiences through the universal language of rhythm.

## Examples of Rhythmic Tunes:

1. Pop songs with catchy hooks and melodies.
2. Dance music, such as house, techno, or EDM.
3. Hip-hop and rap songs with strong rhythmic foundations.
4. Funk and soul music with emphasis on groove and rhythm.
5. A spontaneous fusion of jazz, hip-hop, and electronic beats transforms a city street into a dynamic dancefloor.
6. A VR concert where audiences interact with and remix live performances in real-time.
7. A genre-blending, adaptive soundtrack enhances storytelling in movies and video games.

## **Target Audience:-**

Music Streaming is designed for a diverse audience, including:

**Music Enthusiasts & Festival Goers** – People who enjoy live performances, experimental sounds, and immersive musical experiences.

**Electronic & Experimental Music Lovers** – Fans of EDM, synthwave, and digital soundscapes who seek unique and high-energy performances.

**Artists & Musicians** – Creative individuals inspired by cross-genre collaborations and looking to explore new musical frontiers.

## **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.

**Audience Engagement** – Connect with diverse listeners through live events, digital platforms, and interactive performances.

**Creative Collaboration** – Bring together musicians, producers, and artists to experiment with new soundscapes.

**Technology Integration** – Utilize VR, AI, and immersive tools to redefine music experiences.

**Live Performances** – Organize concerts, festival shows, and street events to showcase genre fusion.

### **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.
- **Genre-Blending Soundscapes :** Fusion of jazz, hip-hop, electronic, and classical elements for a unique musical experience.
- **Artist & Producer Collaborations:** Bringing together diverse musicians to create innovative and boundary-pushing tracks.

*“Music is the language of the spirit. It opens the secret of life bringing peace, abolishing strife.” “Where words leave off, music begins.” “Music in the soul can be heard by the universe.”*

## Pre-Requisites:-

Here are the key prerequisites 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
```

- 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:5174/> in your web browser.

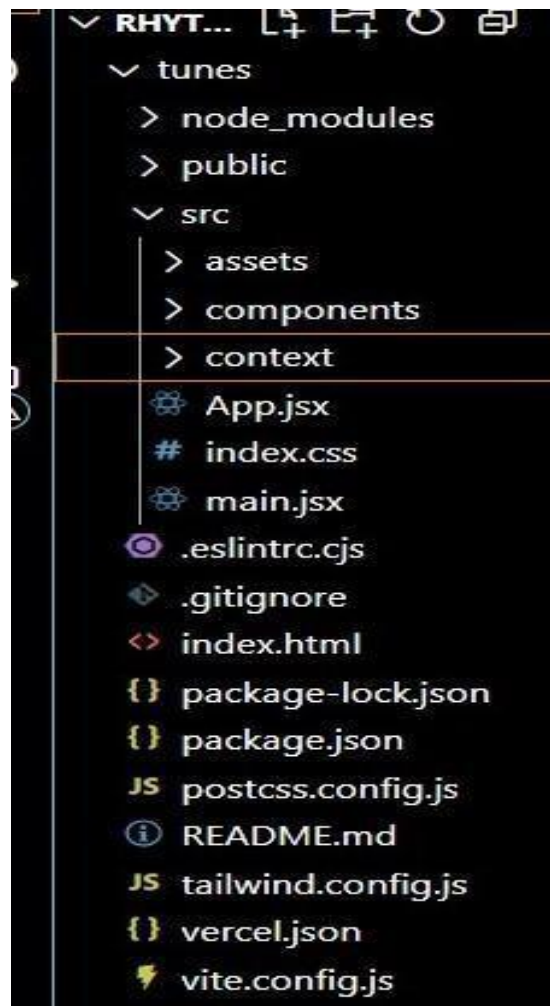
- **HTML, CSS, and JavaScript:** Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for
- client-side interactivity is essential.
- **Version Control:** Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.
- 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 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 structure:

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 `AppComponent`, 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 Flow:-**

### **Project demo:**

Before starting to work on this project, let's see the demo.

Demolink:

[https://drive.google.com/file/d/1PX2hVSpAuqskEJnyQ0vKZ1Uvc5GW0mwD/view?usp=drive\\_link](https://drive.google.com/file/d/1PX2hVSpAuqskEJnyQ0vKZ1Uvc5GW0mwD/view?usp=drive_link)

Use the code in:

[https://github.com/Preethi8015/NM-Project\\_rhythmictunes](https://github.com/Preethi8015/NM-Project_rhythmictunes)

### **Milestone 1: Project Setup and Configuration:**

#### **1. Install required tools and software:**

- **Installation of required tools:**

1. Open the project folder to install necessary tools In this project, we use:
  - o React Js
  - o React Router Dom
  - o React Icons
  - o Bootstrap/tailwind css
  - o Axios

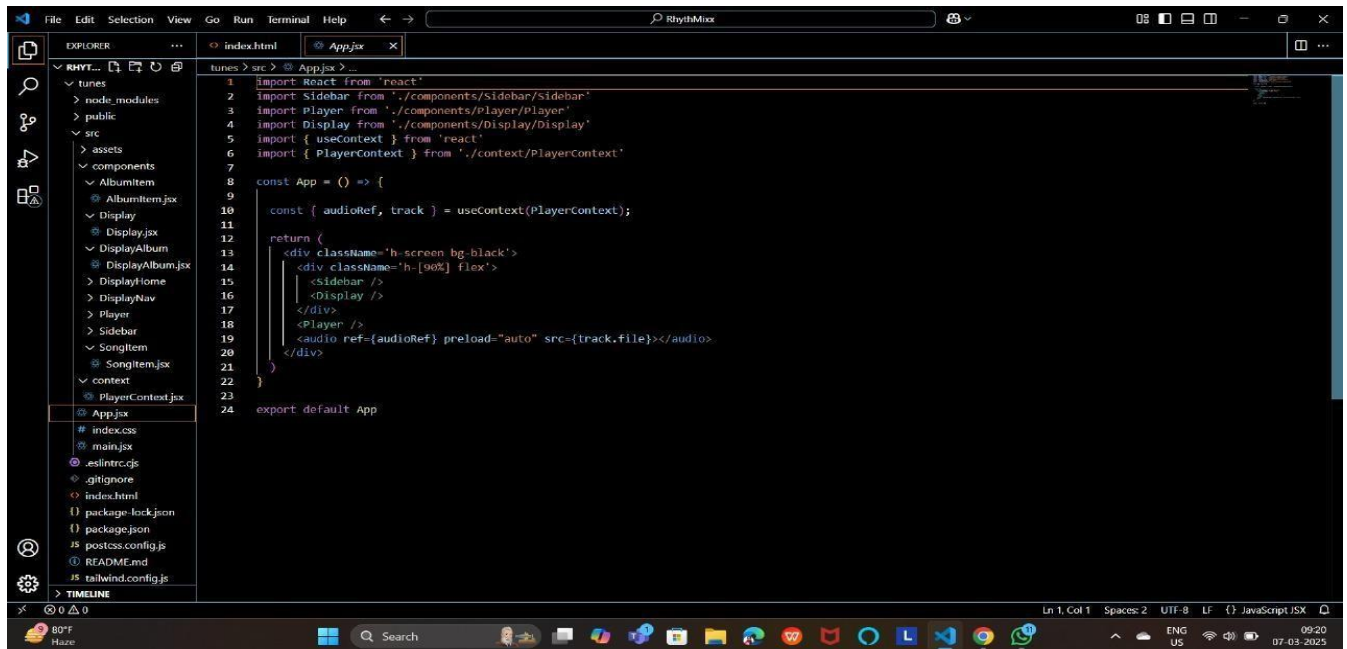
For further reference, use the following resources

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

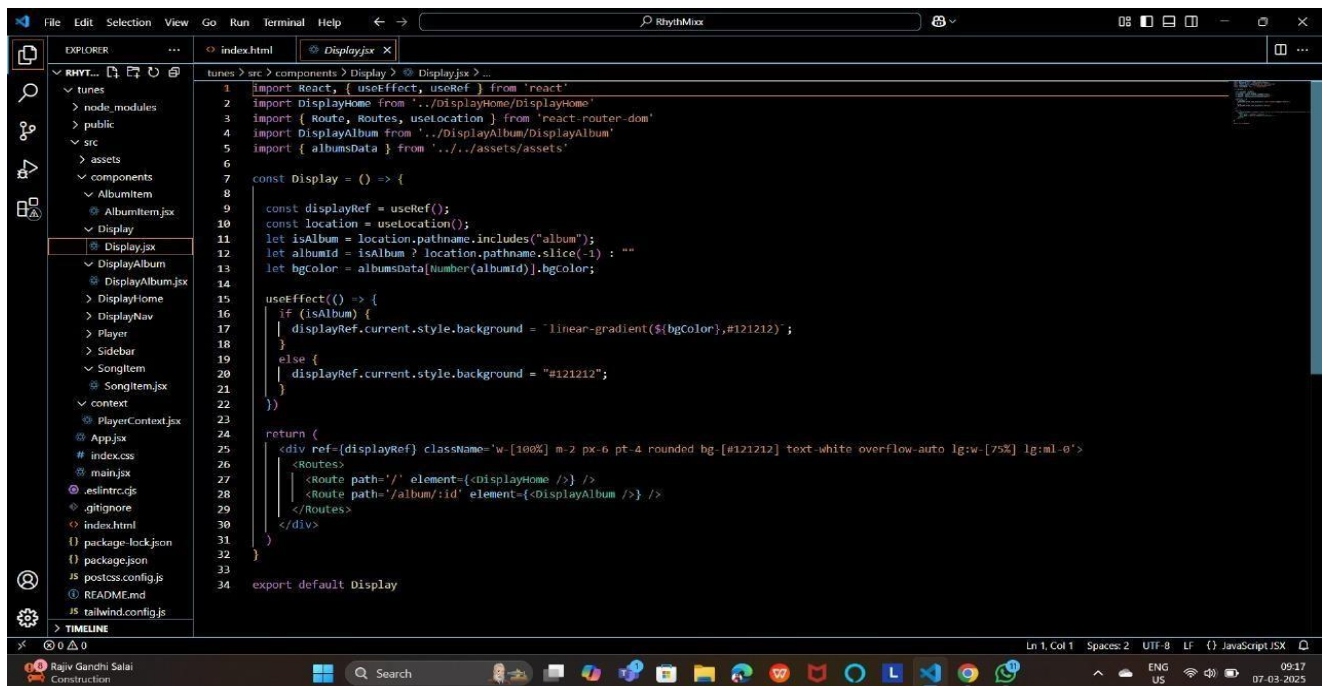
## Milestone 2: Project Development:

### 1. Setup React Application:

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



```
1 import React from 'react'
2 import Sidebar from '../components/Sidebar/Sidebar'
3 import Player from '../components/Player/Player'
4 import Display from '../components/Display/Display'
5 import { useContext } from 'react'
6 import { PlayerContext } from '../context/PlayerContext'
7
8 const App = () => {
9
10   const { audioRef, track } = useContext(PlayerContext);
11
12   return (
13     <div className='h-screen bg-black'>
14       <div className='h-[90%] flex'>
15         <Sidebar />
16         <Display />
17       </div>
18       <Player />
19       <audio ref={audioRef} preload="auto" src={track.file}></audio>
20     </div>
21   )
22 }
23
24 export default App
```



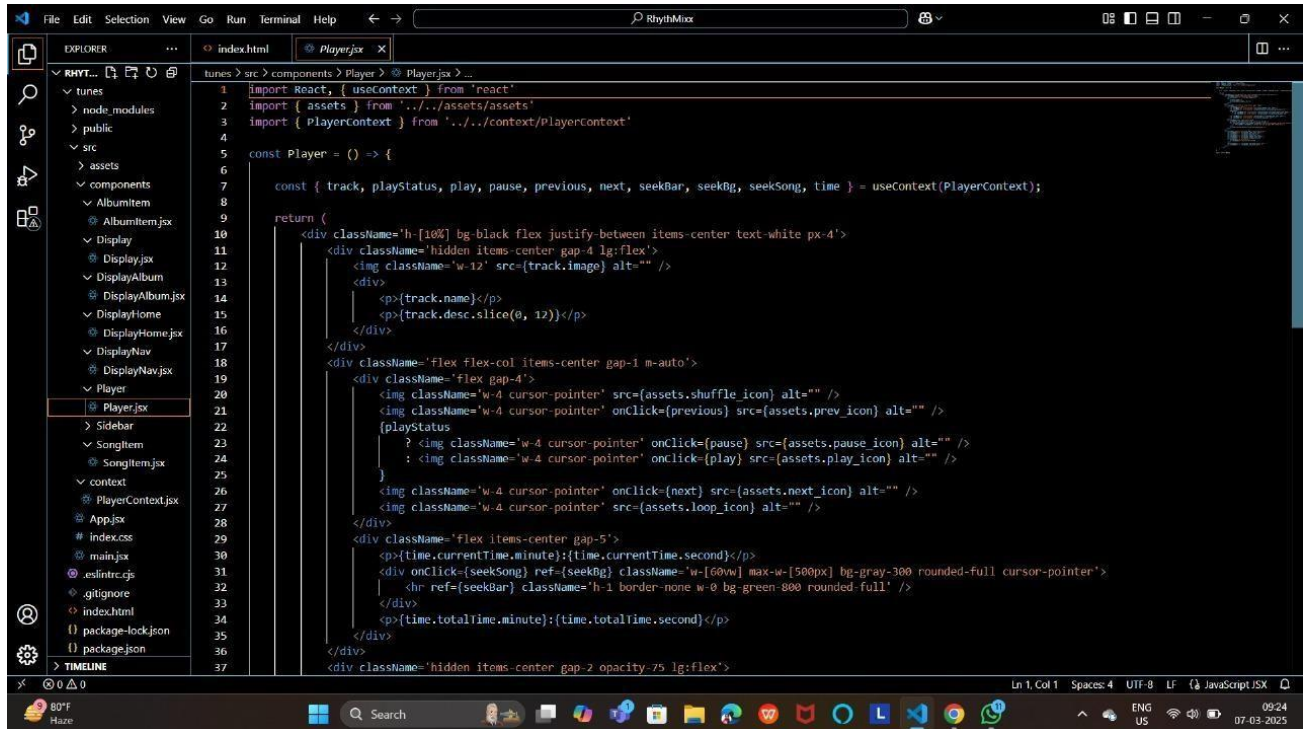
```
1 import React, { useEffect, useRef } from 'react'
2 import DisplayHome from '../DisplayHome/DisplayHome'
3 import { route, Routes, useLocation } from 'react-router-dom'
4 import DisplayAlbum from '../DisplayAlbum/DisplayAlbum'
5 import { albumsData } from '../assets/assets'
6
7 const Display = () => {
8
9   const displayRef = useRef();
10   const location = useLocation();
11
12   let isAlbum = location.pathname.includes("album");
13   let albumId = isAlbum ? location.pathname.slice(-1) : "";
14   let bgColor = albumsData[Number(albumId)].bgColor;
15
16   useEffect(() => {
17     if (isAlbum) {
18       displayRef.current.style.background = "linear-gradient(to top, #121212, #121212)";
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:ml-0'>
26       <Routes>
27         <route path="/" element={DisplayHome} />
28         <route path="/album/:id" element={DisplayAlbum} />
29       </Routes>
30     </div>
31   )
32 }
33
34 export default Display
```

## Setting Up Routers:-

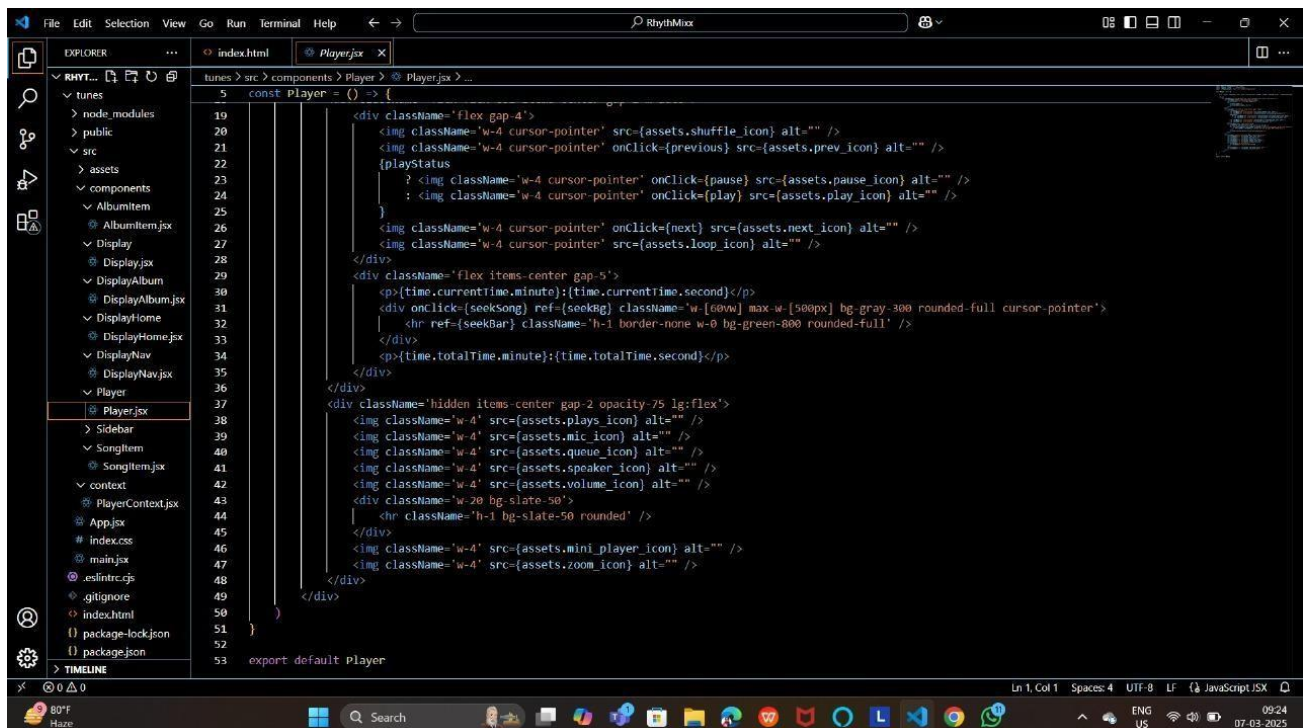
Imports `BrowserRouter`, `Routes`, and `Route` from `react-router-dom` for setting up client-side routing in the application.

- 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.

## Fetching Songs:-



```
1 import React, { useContext } from 'react'
2 import { assets } from '../assets/assets'
3 import { PlayerContext } from '../context/playercontext'
4
5 const Player = () => {
6
7   const { track, playStatus, play, pause, previous, next, seekBar, seekBg, seekSong, time } = useContext(PlayerContext);
8
9   return (
10     <div className='h-[100] bg-black flex justify-between items-center text-white px-4'>
11       <div className='hidden items-center gap-4 lg:flex'>
12         <img className='w-12' src={track.image} alt="" />
13         <div>
14           <p>{track.name}</p>
15           <p>{track.desc.slice(0, 12)}</p>
16         </div>
17       </div>
18       <div className='flex flex-col items-center gap-1 m-auto'>
19         <div className='flex gap-4'>
20           <img className='w-4 cursor-pointer' src={assets.shuffle_icon} alt="" />
21           <img className='w-4 cursor-pointer' onClick={previous} src={assets.prev_icon} alt="" />
22           {playStatus}
23           ? <img className='w-4 cursor-pointer' onClick={pause} src={assets.pause_icon} alt="" />
24             : <img className='w-4 cursor-pointer' onClick={play} src={assets.play_icon} alt="" />
25         </div>
26         <img className='w-4 cursor-pointer' onClick={next} src={assets.next_icon} alt="" />
27         <img className='w-4 cursor-pointer' src={assets.loop_icon} alt="" />
28       </div>
29       <div className='flex items-center gap-5'>
30         <p>{time.currentTime.minute}:{time.currentTime.second}</p>
31         <div onClick={seekSong} ref={seekBg} className='w-[600px] max-w-[500px] bg-gray-300 rounded-full cursor-pointer'>
32           <div ref={seekBar} className='h-1 border-none w-0 bg-green-800 rounded-full' />
33         </div>
34         <p>{time.totalTime.minute}:{time.totalTime.second}</p>
35       </div>
36     </div>
37     <div className='hidden items-center gap-2 opacity-75 lg:flex'>
```



```
38     <div className='hidden items-center gap-2 opacity-75 lg:flex'>
39       <img className='w-4' src={assets.play_icon} alt="" />
40       <img className='w-4' src={assets.mic_icon} alt="" />
41       <img className='w-4' src={assets.queue_icon} alt="" />
42       <img className='w-4' src={assets.speaker_icon} alt="" />
43       <img className='w-4' src={assets.volume_icon} alt="" />
44       <div className='w-20 bg-slate-50'>
45         <div className='h-1 bg-slate-50 rounded' />
46       </div>
47       <img className='w-4' src={assets.mini_player_icon} alt="" />
48       <img className='w-4' src={assets.zoom_icon} alt="" />
49     </div>
50   )
51 }
52
53 export default Player
```

## 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 wishliststate 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 playliststate 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 playliststate after removing an item.
- **isItemInPlaylist(itemId):**
  - Checks if an item exists in the playlist (playlist) based on its itemId.
- **filteredItems:**
  - Filters itemsbased 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 filteredItemsto render each item in the UI.
  - Includes buttons (FaHeart, FaRegHeart) to add/remove items from wishlistandplaylist.
  - Renders audio elements for each item with play/pause functionality.



## Error Handling:

```

return (
  <div style={{display:"flex", justifyContent:"flex-end"}}>
    <div className="songs-container" style={{width:"1300px"}}>
      <div className="container mx-auto p-3">
        <h2 className="text-3xl font-semibold mb-4 text-center">Songs List</h2>
        <InputGroup className="mb-3">
          <InputGroup.Text id="search-icon">
            <FaSearch />
          </InputGroup.Text>
          <Form.Control
            type="search"
            placeholder="Search by singer, genre, or song name"
            value={searchTerm}
            onChange={(e) => setSearchTerm(e.target.value)}
            className="search-input"
          />
        </InputGroup>
        <br />
        <div className="row row-cols-1 row-cols-md-2 row-cols-lg-3 row-cols-xl-4 g-4">
          {filteredItems.map((item, index) => (
            <div key={item.id} className="col">
              <div className="card h-100">
                <img
                  src={item.imageUrl}
                  alt="Item Image"
                  className="card-img-top rounded-top"
                  style={{ height: '200px', width: '100%' }}
                />
                <div className="card-body">
                  <div className="d-flex justify-content-between align-items-center mb-2">
                    <h5 className="card-title">{item.title}</h5>
                    {isItemInWishlist(item.id) ? (
                      <Button
                        variant="light"
                        onClick={() => removeFromWishlist(item.id)}
                      >
                        <FaHeart color="red" />
                      </Button>
                    ) : (
                      <Button
                        variant="light"
                        onClick={() => addToWishlist(item.id)}
                      >

```

```

        <FaRegHeart color="black" />
      </Button>
    )}
  </div>
  <p className="card-text">Genre: {item.genre}</p>
  <p className="card-text">Singer: {item.singer}</p>
  <audio controls className="w-100" id={`audio-${item.id}`}>
    <source src={item.songUrl} />
  </audio>
</div>
<div className="card-footer d-flex justify-content-center">
  {isItemInPlaylist(item.id) ? (
    <Button
      variant="outline-secondary"
      onClick={() => removeFromPlaylist(item.id)}
    >
      Remove From Playlist
    </Button>
  ) : (
    <Button
      variant="outline-primary"
      onClick={() => addToPlaylist(item.id)}
    >
      Add to Playlist
    </Button>
  )}
</div>
</div>
</div>
))}
</div>
</div>
</div>
);
}

export default Songs;
```

## 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:

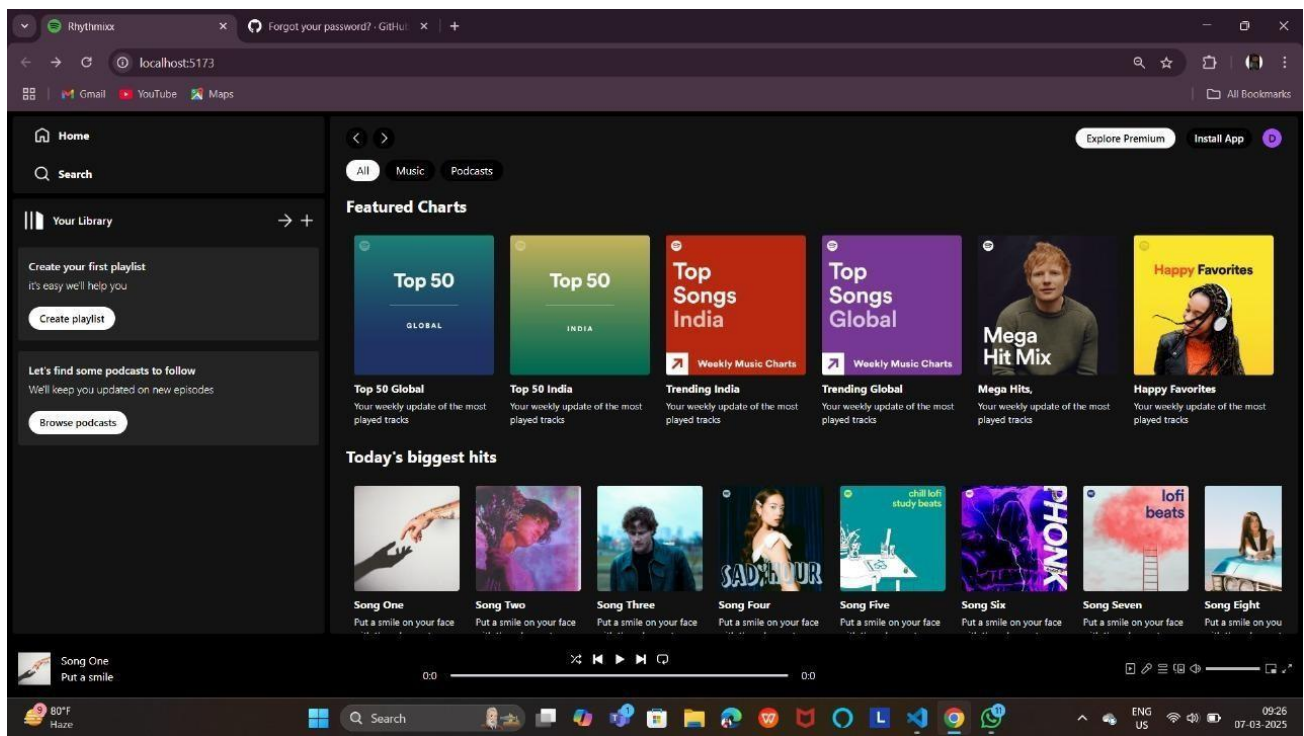
After completing the code, run the react application by using the command “npm start” or “npm run dev” if you are using vite.js

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

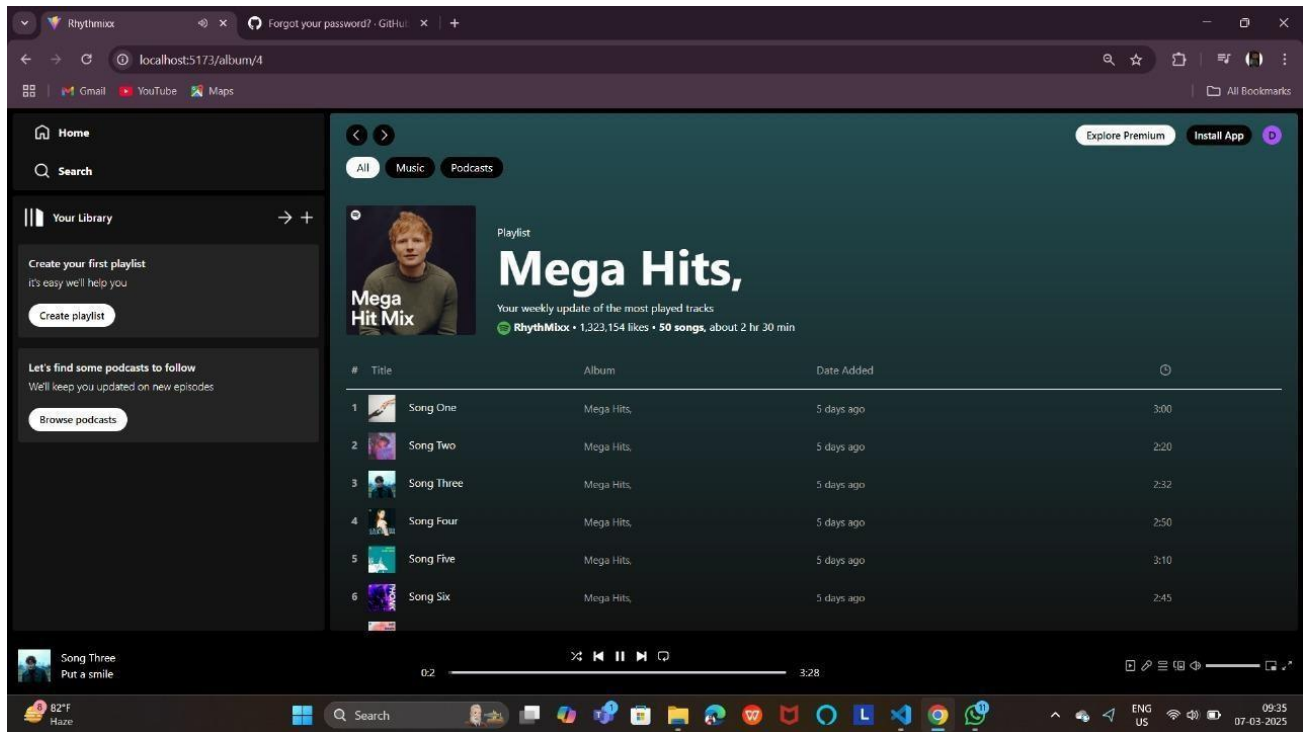
After that launch the Rythimic Tunes.

Here are some of the screenshots of the application.

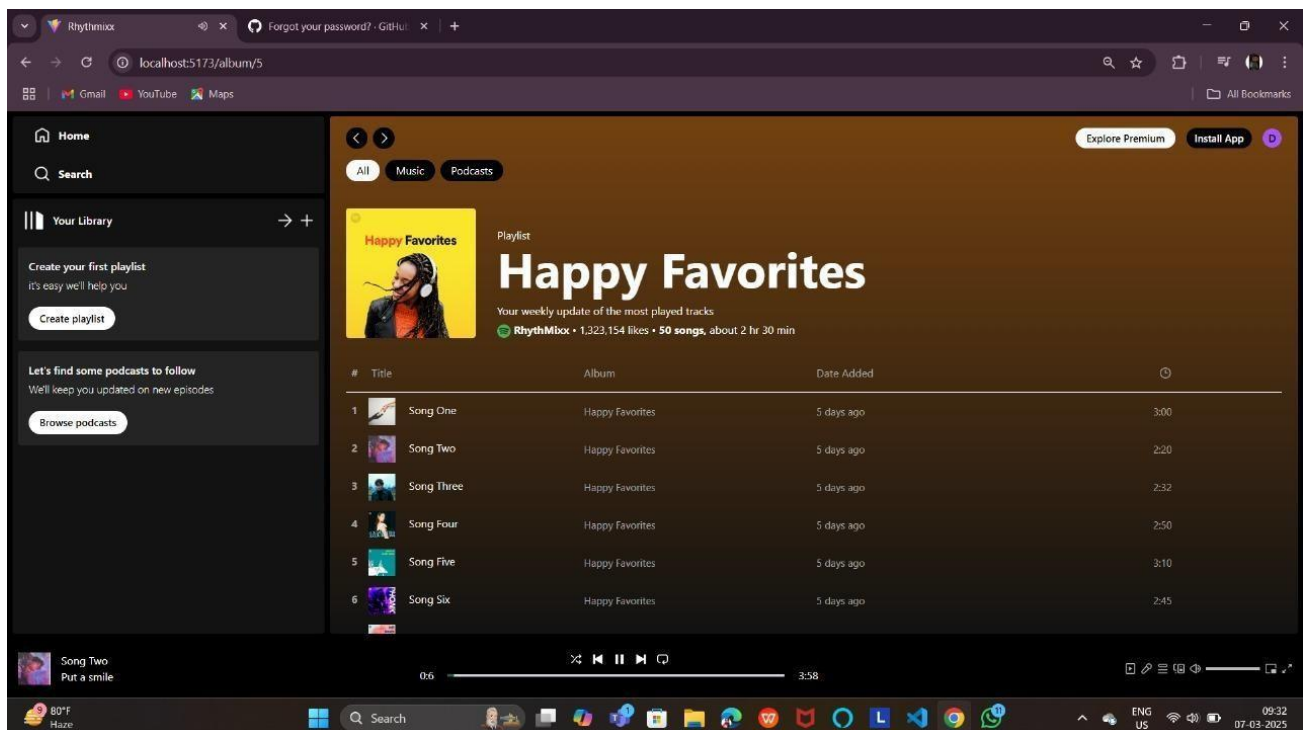
### ➤ Hero components



## ➤ playlist



## ➤ Favorites



## Conclusion

Rhythmixx is not just a music streaming app—it's a revolution in how we experience sound. By integrating cutting-edge technology with intuitive design, we aim to redefine music consumption. Join us on this rhythmic journey and stay tuned!

### Project Demo link:

[https://drive.google.com/file/d/1PX2hVSpAugskEJnyO0vKZ1Uvc5GW0mwD/view?usp=drive\\_link](https://drive.google.com/file/d/1PX2hVSpAugskEJnyO0vKZ1Uvc5GW0mwD/view?usp=drive_link)

*Stay Code-tastic!*