Rhythmic Tunes (React)

Introduction:

Step into the next generation of music streaming with **Rhythmic Tunes**, a revolutionary audio platform designed to elevate your musical experience. Powered by the innovative capabilities of React.js, our app brings you an immersive world of music, offering an experience like no other.

Crafted with precision, **Rhythmic Tunes** is designed for the modern listener who craves both functionality and simplicity. Whether you're exploring the hottest new hits or enjoying old favorites, our platform creates a personalized journey through music, catering to your every mood and preference.

At the heart of this dynamic experience is React, a powerful and intuitive JavaScript library that ensures a sleek, interactive interface that responds to your every touch. Whether you're using a desktop, tablet, or smartphone, you'll enjoy a seamless experience across all devices, with smooth navigation, effortless playlist creation, and visually captivating design.

Break free from traditional music listening and embrace the boundless possibilities that **Rhythmic Tunes** offers. Join us in transforming the way you enjoy music – it's time to hit play and immerse yourself in a new era of audio streaming, powered by React.

Scenario-Based Introduction :-

As you tap open **RhythmicTunes**, the world around you begins to fade into the background. Instantly, your favorite playlist starts playing, and with each beat, you feel the energy of the city syncing with the rhythm of the music. The smooth interface of the app makes it effortless to skip through tracks, discover new songs, and even create the perfect playlist for your commute. With the power of React behind the scenes, the app's responsiveness ensures that no matter where you are—whether you're walking, waiting for the bus, or caught in traffic—**RhythmicTunes** delivers an uninterrupted, seamless listening experience that elevates your every moment. Whether you're seeking motivation or simply need a soundtrack for your day, the app effortlessly transforms your journey into a musical adventure.

Target Audience:-

Music Streaming is designed for a diverse audience, including:

● **Music Enthusiasts:** People passionate about enjoying and listening Music Through out there free time to relax themselves.

Project Goals & Objectives:-

The core aim of our **Music Streaming Application** is to offer a dynamic, accessible platform that enhances the way users interact with music. We strive to create an environment where music lovers can easily discover, enjoy, and share their musical passions without barriers. Key objectives include:

- **Intuitive Navigation**: Design a clean and simple user interface that allows for effortless navigation, ensuring that users can quickly find and organize their favorite tracks, albums, and playlists with ease.
- **Endless Music Discovery**: Implement advanced search filters and personalized recommendations, empowering users to explore an ever-expanding catalog of music from various genres, artists, and eras.
- **Robust Technology**: Leverage modern development tools, including **React.js**, to deliver a fast, responsive, and reliable music streaming experience across all devices. This ensures that our platform meets the needs of every user, whether they're using a mobile phone, tablet, or desktop computer.

Key Features:-

- **Song Listings:** Offer an extensive catalog of songs, complete with essential details such as title, artist, album, genre, and release year, making it easy for users to explore and discover new music.
- **Playlist Creation:** Give users the ability to build custom playlists tailored to their unique tastes, with drag-and-drop functionality to effortlessly organize their favorite tracks for any occasion.
- **Playback Control:** Provide smooth and intuitive playback options, including play, pause, skip, shuffle, and volume adjustment, ensuring a seamless listening experience without interruptions.
- **Offline Listening:** Enable users to download their favorite songs or entire playlists for offline playback, ensuring access to their music library even without an internet connection.
- **Advanced Search:** Incorporate a powerful search engine that allows users to quickly find and filter music based on specific criteria like song name, artist, genre, or album, making music discovery fast and efficient.

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 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.

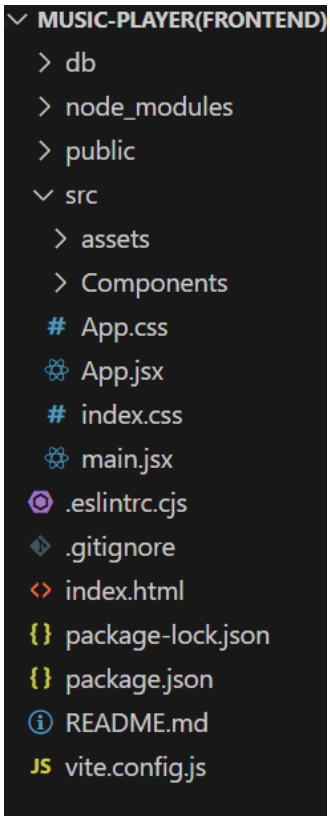
Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

• 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/1zZuq62lyYNV_k5uu0SFjoWa35UgQ4LA9/view?usp=drive_link
Use the code in:
https://drive.google.com/drive/folders/1BkYWfW_K3ek_UgtXNTAsDqlhdCuqz6nT?usp=drive_link

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.

Setting Up Routes:-

```
import 'bootstrap/dist/css/bootstrap.min.css';
import './App.css'
import { BrowserRouter,Routes,Route } from 'react-router-dom'
import Songs from './
import Sidebar from '
import Favorities fro
import Playlist from './Components/Playlist';

function App() {

  return (
   <div  >
    <BrowserRouter>
    <div>
    <Sidebar/>
    </div>
          <div>
          <Routes>
            <Route path='/' element={<Songs/>} />
            <Route path='/favorities' element={<Favorities/>} />
            <Route path='/playlist' element={<Playlist/>} />
          </Routes>
          </div>
    </BrowserRouter>
   </div>
  )
}

export default App
```

module "c:/Users/arsha/OneDrive/Desktop/MY PROJECTS/Music-Player(Frontend)/src/Components/Playlist"

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: o The first div contains the Sidebar component, likely serving navigation or additional content.
o The second div contains the Routes component from React Router, which handles rendering components based on the current route.
o Inside Routes, defines several Route components: o Route with path='/' renders the Songs component when the root path is accessed (/).
o Route with path='/favorities' renders the Favorities component when the /favorities path is accessed.
o 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:-

```jsx
import React, { useState, useEffect } from 'react';
import { Button, Form, InputGroup } from 'react-bootstrap';
import { FaHeart, FaRegHeart, FaSearch } from 'react-icons/fa';
import axios from 'axios';
import './sidebar.css'

function Songs() {
  const [items, setItems] = useState([]);
  const [wishlist, setWishlist] = useState([]);
  const [playlist, setPlaylist] = useState([]);
  const [currentlyPlaying, setCurrentlyPlaying] = useState(null);
  const [searchTerm, setSearchTerm] = useState('');

  useEffect(() => {
    // Fetch all items
    axios.get('http://localhost:3000/items')
      .then(response => setItems(response.data))
      .catch(error => console.error('Error fetching items: ', error));

  // Fetch favorities items
      axios.get('http://localhost:3000/favorities')
      .then(response => setWishlist(response.data))
      .catch(error => {
        console.error('Error fetching Favvorities:', error);
        // Initialize wishlist as an empty array in case of an error
        setWishlist([]);
      });

    // Fetch playlist items
    axios.get('http://localhost:3000/playlist')
      .then(response => setPlaylist(response.data))
      .catch(error => {
        console.error('Error fetching playlist: ', error);
        // Initialize playlist as an empty array in case of an error
        setPlaylist([]);
      });
      // Function to handle audio play
    const handleAudioPlay = (itemId, audioElement) => {
        if (currentlyPlaying && currentlyPlaying !== audioElement) {
          currentlyPlaying.pause(); // Pause the currently playing audio
        }
        setCurrentlyPlaying(audioElement); // Update the currently playing audio

      };
```

```
    // Event listener to handle audio play
    const handlePlay = (itemId, audioElement) => {
      audioElement.addEventListener('play', () => {
        handleAudioPlay(itemId, audioElement);
      });
    };

    // Add event listeners for each audio element
    items.forEach((item) => {
      const audioElement = document.getElementById(`audio-${item.id}`);
      if (audioElement) {
        handlePlay(item.id, audioElement);
      }
    });

    // Cleanup event listeners
    return () => {
      items.forEach((item) => {
        const audioElement = document.getElementById(`audio-${item.id}`);
        if (audioElement) {
          audioElement.removeEventListener('play', () => handleAudioPlay(item.id, audioElement));
        }
      });
    };
  }, [items,currentlyPlaying, searchTerm]);


  const addToWishlist = async (itemId) => {
    try {
      const selectedItem = items.find((item) => item.id === itemId);
      if (!selectedItem) {
        throw new Error('Selected item not found');
      }
      const { title, imgUrl, genre, songUrl, singer, id: itemId2 } = selectedItem;
      await axios.post('http://localhost:3000/favorities', { itemId: itemId2, title, imgUrl, genre, songUrl, singer });
      const response = await axios.get('http://localhost:3000/favorities');
      setWishlist(response.data);
    } catch (error) {
      console.error('Error adding item to wishlist: ', error);
    }
  };
```

Code Description:-

● useState: `o items`: Holds an array of all items fetched from `http://localhost:3000/items`.

o wishlist: Stores items marked as favorites fetched from http://localhost:3000/favorities.
o playlist: Stores items added to the playlist fetched from http://localhost:3000/playlist.
o currentlyPlaying: Keeps track of the currently playing audio element.
o searchTerm: Stores the current search term entered by the user.

● Data Fetching: o 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.

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

● Audio Playback Management: o 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.

o Ensures that only one audio element plays at a time by pausing others when a new one starts playing.

● addToWishlist(itemId): o Adds an item to the wishlist (favorities) by making a POST request to http://localhost:3000/favorities.
o Updates the wishlist state after adding an item.

● removeFromWishlist(itemId): o Removes an item from the wishlist (favorities) by making a DELETE request to http://localhost:3000/favorities/{itemId}.
o Updates the wishlist state after removing an item.

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

● addToPlaylist(itemId):

o Adds an item to the playlist (`playlist`) by making a POST request to `http://localhost:3000/playlist`.
o Updates the `playlist` state after adding an item.

● removeFromPlaylist(itemId): o Removes an item from the playlist (`playlist`) by making a DELETE request to `http://localhost:3000/playlist/{itemId}`.
o Updates the `playlist` state after removing an item.

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

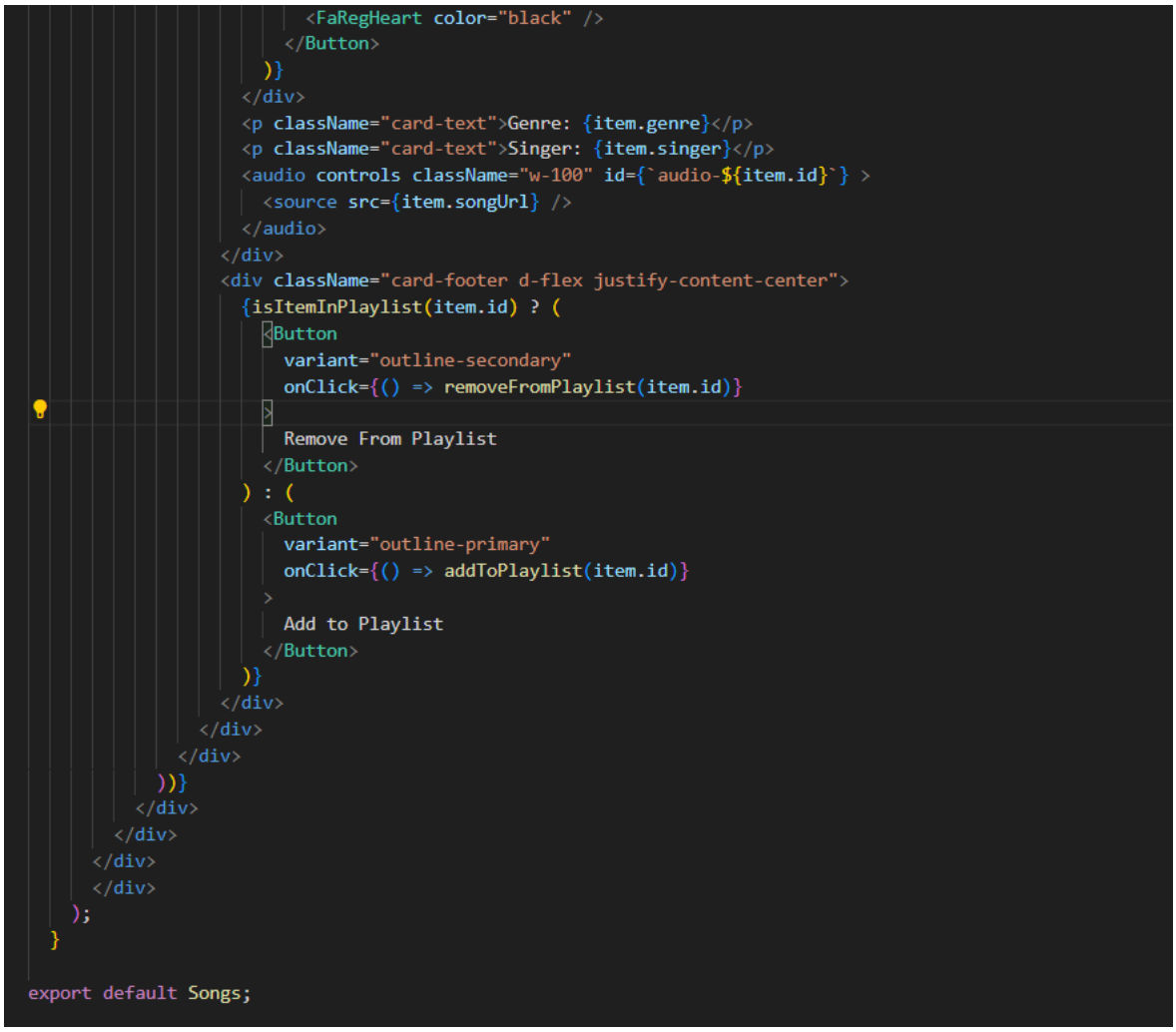
● filteredItems: o Filters `items` based on the `searchTerm`.
o Matches `title`, `singer`, or `genre` with the lowercase version of `searchTerm`.

● JSX: o Renders a form with an input field (`Form`, `InputGroup`, `Button`, `FaSearch`) for searching items.
o Maps over `filteredItems` to render each item in the UI.
o Includes buttons (`FaHeart`, `FaRegHeart`) to add/remove items from `wishlist` and `playlist`.
o Renders audio elements for each item with play/pause functionality.

● Error Handling: o Catches and logs errors during data fetching (`axios.get`).
o Handles errors when adding/removing items from `wishlist` and `playlist`.

Frontend Code For Displaying Songs:-

```
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.imgUrl}
                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>
    </div>
  );
}

export default Songs;
```

Code Description:-

● Container Setup: o Uses a `div` with inline styles (`style={{display:"flex", justifyContent:"flex-end"}}`) to align the content to the right.
o The main container (`songs-container`) has a fixed width (`width:"1300px"`) and contains all the UI elements related to songs.

● Header: o Displays a heading (`<h2>`) with text "Songs List" centered (`className="text-3xl font-semibold mb-4 text-center"`).

● Search Input: o Utilizes `InputGroup` from React Bootstrap for the search functionality.
o Includes an input field (`Form.Control`) that allows users to search by singer, genre, or song name.
o Binds the input field value to `searchTerm` state (`value={searchTerm}`) and updates it on change (`onChange={(e) => setSearchTerm(e.target.value)}`).
o Styled with `className="search-input"`.

● Card Layout: o 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"`).
o Maps over `filteredItems` array and renders each item as a Bootstrap card (`<div className="card h-100">`).

● Card Content: o Displays the item's image (`<img>`), title (`<h5 className="card-title">`), genre (`<p className="card-text">`), and singer (`<p className="card-text">`).
o 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: o Adds a heart icon button (`<Button>`) to add or remove items from the wishlist (`isItemInWishlist(item.id)` determines which button to show).
o Includes an "Add to Playlist" or "Remove From Playlist" button (`<Button>`) based on whether the item is already in the playlist (`isItemInPlaylist(item.id)`).

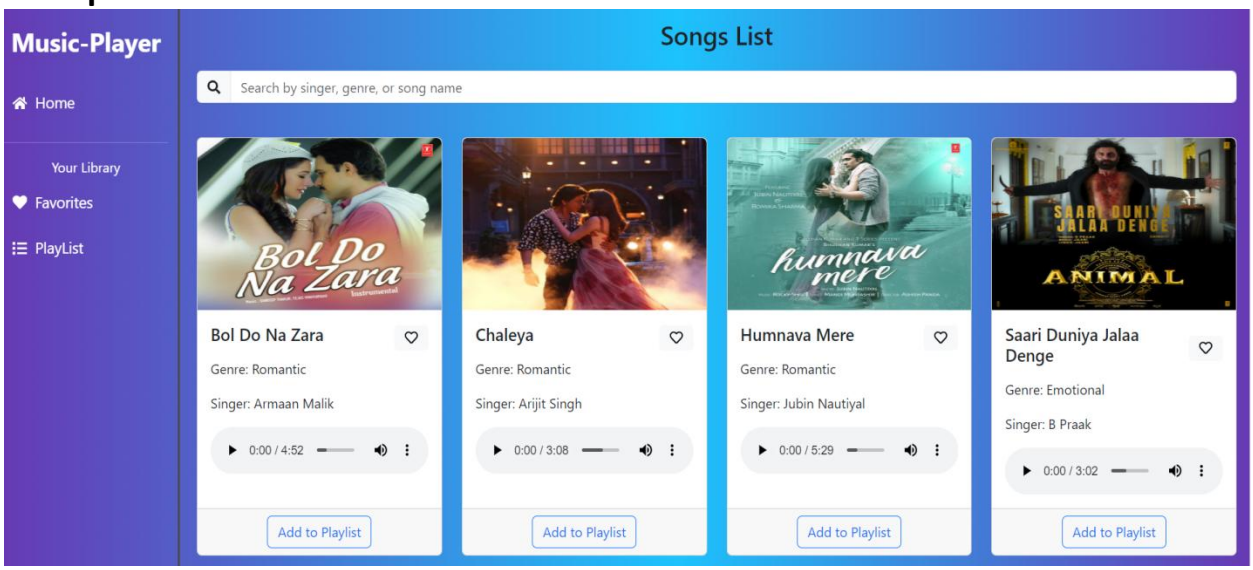
● Button Click Handlers: o Handles adding/removing items from the wishlist (`addToWishlist(item.id)`, `removeFromWishlist(item.id)`).
o Manages adding/removing items from the playlist (`addToPlaylist(item.id)`, `removeFromPlaylist(item.id)`).

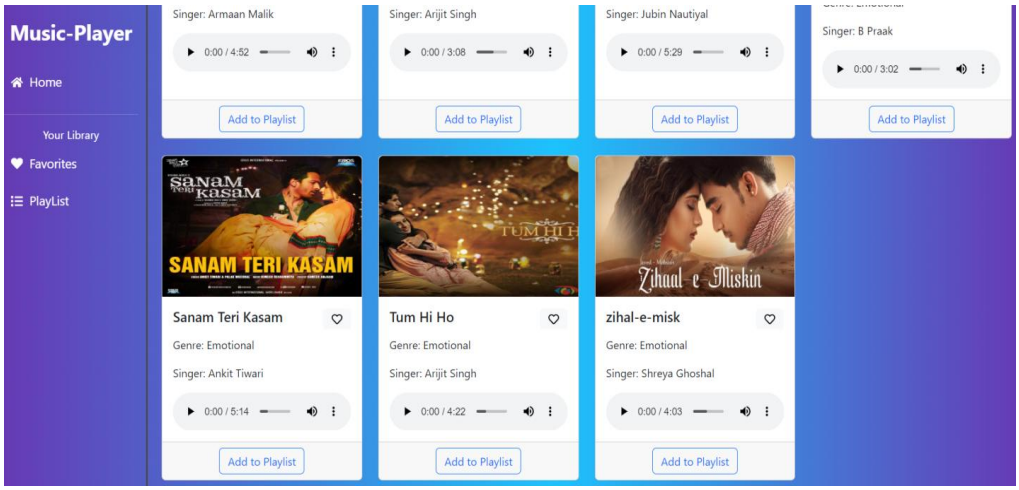
● Card Styling: o Applies Bootstrap classes (`card`, `card-body`, `card-footer`) for styling the card components.
o Uses custom styles (`rounded-top`, `w-100`) for specific elements like images and audio players.

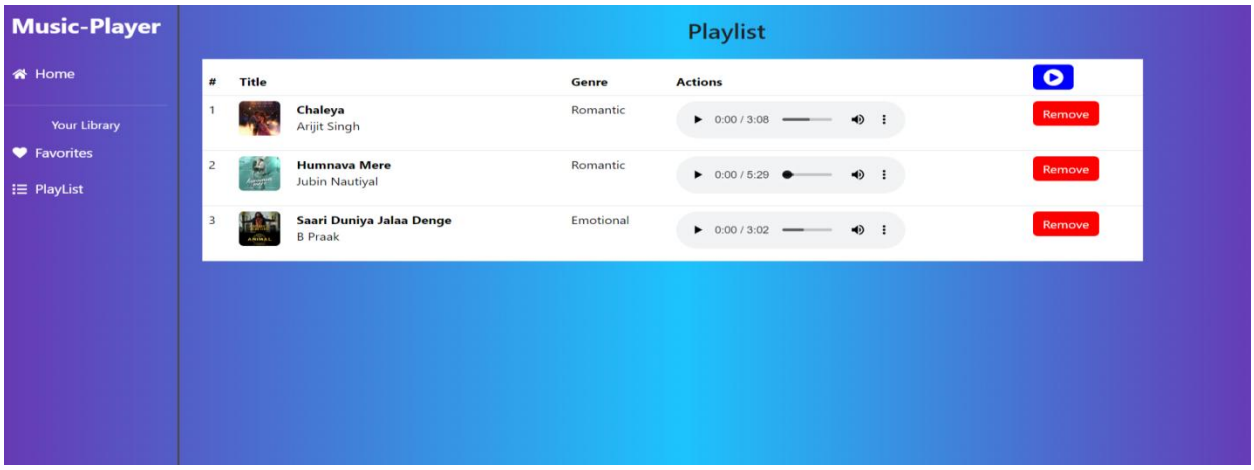
Project Execution:

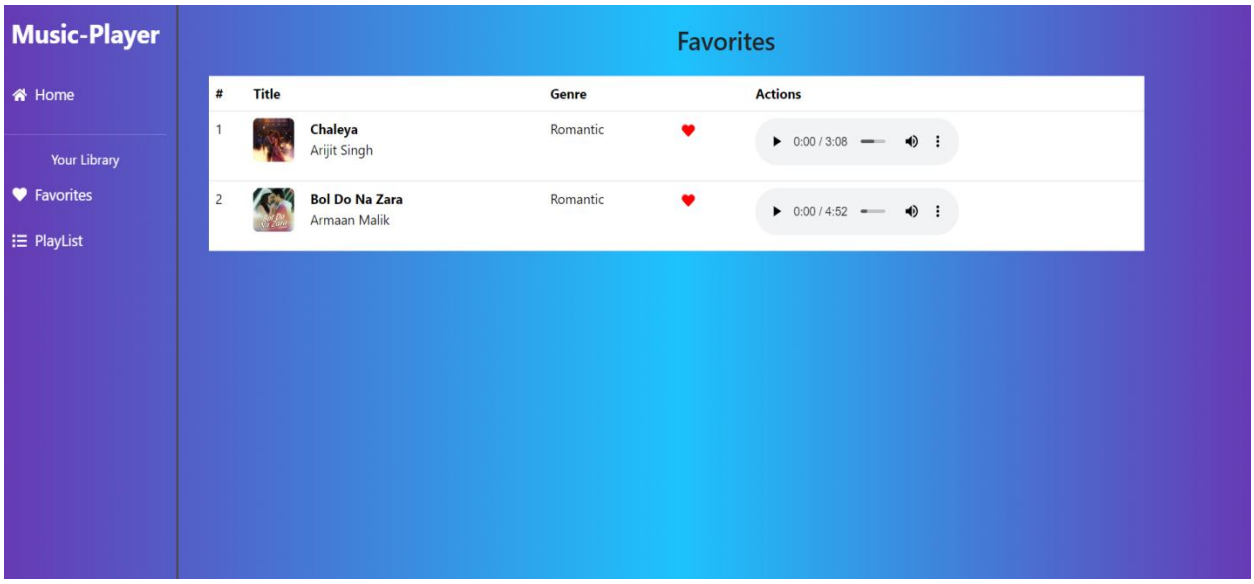
After completing the code, run the react application by using the command “npm start” or “npm run dev” if you are using vite.js
And the Open new Terminal type this command “json-server --watch ./db/db.json” to start the json server too.
After that launch the Rythimic Tunes.
Here are some of the screenshots of the application.

Head components




Music-Player
Home
Your Library
Favorites
PlayList
Singer: Armaan Malik
0:00 / 4:52
Add to Playlist
Singer: Arijit Singh
0:00 / 3:08
Add to Playlist
Singer: Jubin Nautiyal
0:00 / 5:29
Add to Playlist
Singer: B Praak
0:00 / 3:02
Add to Playlist
SANAM TERI KASAM
Sanam Teri Kasam
Genre: Emotional
Singer: Ankit Tiwari
0:00 / 5:14
Add to Playlist
Tum Hi Ho
Genre: Emotional
Singer: Arijit Singh
0:00 / 4:22
Add to Playlist
Zihaal-e-Miskin
zihal-e-misk
Genre: Emotional
Singer: Shreya Ghoshal
0:00 / 4:03
Add to Playlist


Music-Player
Home
Your Library
Favorites
PlayList
Playlist
Title Genre Actions
1 Chaleya Arijit Singh Romantic 0:00 / 3:08 Remove
2 Humnava Mere Jubin Nautiyal Romantic 0:00 / 5:29 Remove
3 Saari Duniya Jalaa Denge B Praak Emotional 0:00 / 3:02 Remove



Project Demo link:

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

*** THANK YOU ***