

Project Name: Rhythmic Tunes

Project Team Members

1. **Name:** J.Surekha
Email: surekhashan2005@gmail.com
2. **Name:** S.Roshini
Email: sr3688542@gmail.com
3. **Name:** S.Swathi
Email: swathiselvan24@gmail.com
4. **Name:** R.Valarmathi
Email: v45606768@gmail.com
5. **Name:** M.Vishali
Email: vishalimohan06@gmail.com

Rhythmic Tunes (React)

Introduction:-

Welcome to the future of music! Our Music Streaming App offers an unmatched audio experience. It's built using the latest technology, React.js, ensuring top-notch performance. The app is designed with you in mind, making it easy to use and enjoy. You can discover new songs, create playlists, and stream music effortlessly. We've combined innovation with user-friendly design to make your music experience better. Whether you're at home or on the go, you'll always have access to your favourite tracks. The app loads quickly and runs smoothly, so you can enjoy uninterrupted music. It's simple, fun, and made for music lovers like you. Get ready to transform the way you enjoy 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 old ways of listening to music and step into a world full of new possibilities with our Music Streaming App, built with React. This app is designed to give you an upgraded and exciting way to enjoy music. You'll have access to endless songs, playlists, and features that make listening easier and more fun. Whether you're at home, at work, or on the go, your favourite music is always just a tap away. Our app is fast, easy to use, and constantly improving. It's perfect for music lovers who want a smooth and modern experience. We're here to change

how you connect with music and make it even more enjoyable. Get ready for a new way to discover, listen to, and experience music. It's time to press play and enjoy the future of music streaming!

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, "Rhythmic Tunes."
- 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.
- Picture this: you're walking down a bustling street, your ear buds in, and your favourite song starts playing. The beat drops, and suddenly, everything feels in sync. That's the power of music, and now imagine experiencing it with **React-based technology**.
- In a world where your musical preferences evolve as quickly as your mood, our app reacts faster than ever. Seamlessly blending vibrant design with fluid navigation, it adjusts to your rhythm, just like the beats you love. Whether you're tuning in to the latest hits or deep cuts, the app stays in tune with your preferences, delivering the perfect experience every time.
- As the music plays, the app **reacts** to your choices, giving you smart suggestions and a personalized journey through sound. From your workout jams to chill vibes, we've got you covered. So, just like hitting "play" on your favourite playlist, the adventure starts here.
- Ready to amplify your auditory experience? Get ready to feel the **extra beats** in every moment. It's time to let music take the lead, powered by React!

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.
- Who's ready to vibe? Our app is crafted for **music lovers** of all kinds, whether you're a **teen who's all about the latest trends**, or a **seasoned listener** who's got a deep catalog of classics. If you're always looking for new jams or you live for discovering hidden gems, we've got you.
- For the **commuters**, the **gym-goers**, and the **party planners** — we're here to make every moment feel like a soundtrack. Whether you're working out, cruising in your car, or just chilling at home, our app follows your flow, delivering the perfect beats every time.
- **Music creators** and **curators** will love how we let them share and organize their playlists with ease. For the **tech-savvy**, the **early adopters**, and the **innovative explorers**, React's cutting-edge power ensures you'll always be ahead of the curve.
- No matter who you are, if you're passionate about music, **this app is for you**. Get ready to connect, groove, and experience music like never before!

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.
- **Enhance User Experience** – Create a seamless, intuitive interface that allows users to easily discover, play, and manage their music, making music streaming effortless.
- **Personalized Music Discovery** – Develop algorithms to offer tailored recommendations, ensuring users always find fresh and relevant music based on their tastes and listening history.
- **High-Quality Audio Streaming** – Ensure top-notch audio quality with minimal buffering, delivering a premium listening experience for all users.
- **Cross-Device Synchronization** – Enable smooth synchronization across multiple devices, so users can pick up right where they left off, no matter which device they're using.
- **Offline Accessibility** – Allow users to download music and playlists for offline listening, ensuring access to their favorite tracks even without an internet connection.
- **Social Integration** – Integrate social sharing features so users can easily share their favorite songs, albums, or playlists with friends and followers.
- **Live Radio and Curated Content** – Offer live radio, curated playlists, and exclusive content to keep users engaged and connected to the latest trends in music.
- **Easy Playlist Management** – Empower users to easily create, manage, and customize their own playlists for every occasion or mood.
- **User-Centered Design** – Focus on a responsive, easy-to-navigate UI that adapts to different screen sizes and user needs, ensuring accessibility for everyone.
- **Continuous Improvement** – Regularly update the app with new features, bug fixes, and performance improvements based on user feedback and emerging trends in the music industry.

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.
- **Personalized Recommendations** – Get music suggestions based on your listening history and preferences, ensuring every session is unique and tailored to your taste.
- **Custom Playlists** – Create and organize your own playlists, from daily mixes to special occasion soundtracks.
- **High-Quality Audio** – Enjoy crystal-clear sound with options for different audio qualities, perfect for any environment or device.
- **Offline Listening** – Download your favorite tracks and playlists to enjoy them even when you're offline or without a stable connection.
- **Seamless Cross-Device Sync** – Access your music on any device with automatic syncing, so you can pick up right where you left off.
- **Dynamic Search** – Find songs, albums, and artists with ease using an intuitive search function that adapts to your preferences.
- **Social Sharing** – Share songs, playlists, and musical moments with friends across social media platforms, keeping everyone in the loop.

- **Curated Playlists and Live Radio** – Explore curated playlists and tune into live radio stations for a diverse and evolving music experience.
- **User-Friendly Interface** – Enjoy a clean, easy-to-navigate design that makes music discovery and streaming fun and simple.
- **Real-Time Updates** – Stay updated with the latest tracks, albums, and music news, ensuring you're always in the loop with what's trending.

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.
- **HTML:** In **Rhythmic Tunes**, HTML provides the structure for the app. It organizes content like song lists, playlists, and media controls. Elements such as `` and `` display music, while the `<audio>` tag embeds playable tracks. Navigation is handled through `<nav>` and `<a>` tags, allowing users to move between sections. HTML works with CSS and JavaScript for responsive design and real-time updates, ensuring a smooth, dynamic music experience across devices.
- **CSS:** In **Rhythmic Tunes**, CSS is used to style the app and make it visually appealing. It defines the layout, colors, fonts, and spacing, ensuring a sleek and user-friendly design. CSS also enables responsive design, so the app looks great on any device, whether it's a

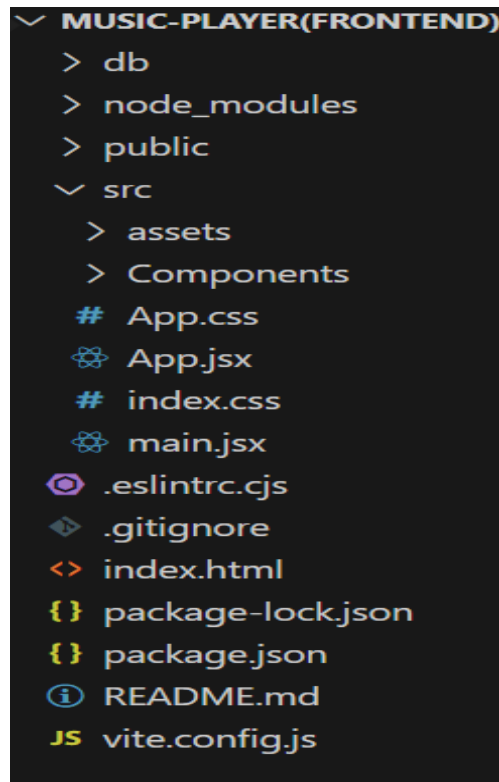
phone, tablet, or desktop. With CSS, elements like buttons, song cards, and menus are styled for a smooth and attractive music streaming experience.

- **JAVA SCRIPT:** In **Rhythmic Tunes**, JavaScript adds interactivity and functionality. It powers features like music playback controls (play, pause, skip), dynamic playlist updates, and real-time song recommendations. JavaScript also works with React to update the app's content without reloading the page, providing a smooth user experience. It ensures that user actions, like adjusting volume or searching for songs, happen instantly and seamlessly.

- **Version Control:** Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like Github 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>

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 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 FLOW:-

Project demo:

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

Demolink:

https://drive.google.com/file/d/1zZuq62lyYNV_k5uu0SFioWa35UgQ4LA9/view?usp=drive_link

Use the code in:

https://drive.google.com/drive/folders/1BkYWfW_K3ek_UgtXNTAsDglhdCuqz6nT?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:

- 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>

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 './Components/Songs'
import Sidebar from 'module "c:/Users/arsha/OneDrive/Desktop/MY PROJECTS/Music-Player(Frontend)/src/Components/Playlist"'
import Favorities from 'module "c:/Users/arsha/OneDrive/Desktop/MY PROJECTS/Music-Player(Frontend)/src/Components/Playlist"'
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
```

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.

Fetching Songs:-

```
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 favorites items
    axios.get('http://localhost:3000/favorites')
      .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/favorites', { itemId: itemId2, title, imgUrl, genre, songUrl, singer });
      const response = await axios.get('http://localhost:3000/favorites');
      setWishlist(response.data);
    } catch (error) {
      console.error('Error adding item to wishlist: ', error);
    }
  };
}
```

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 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:**

- 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:

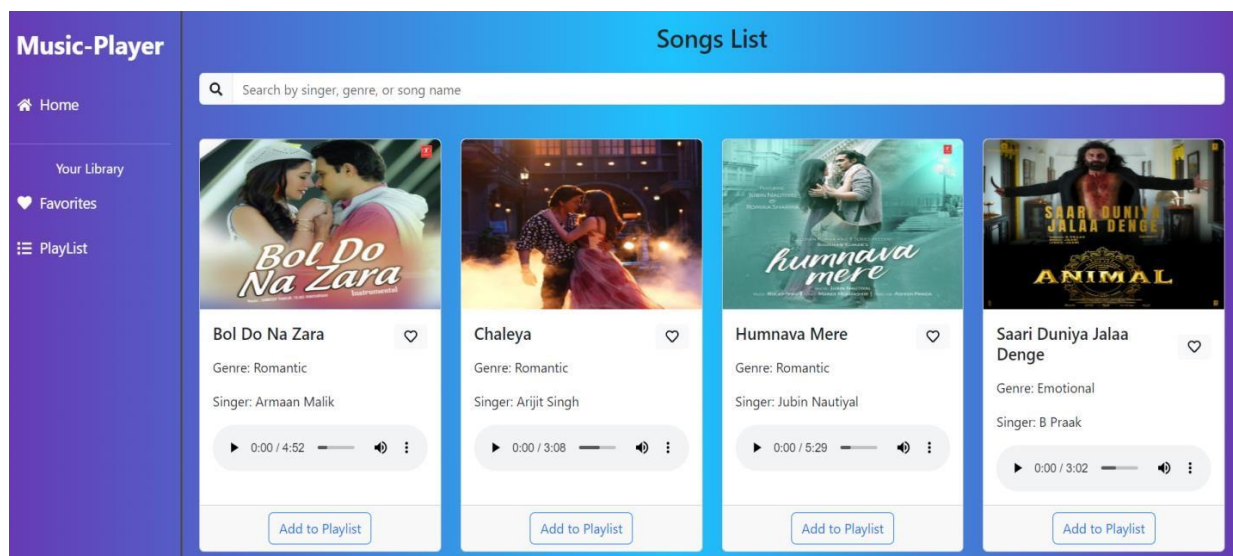
After completing the code, run the react application by using the command “npm start” or “npm run dev” if you are using vite.js

And 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



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

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-misk

Genre: Emotional

Singer: Shreya Ghoshal

0:00 / 4:03

Add to Playlist

➤ Playlist

Music-Player

- Home
- Your Library
- Favorites
- PlayList

Playlist

#	Title	Genre	Actions
1	<div> <h3>Chaleya</h3> <p>Arijit Singh</p> </div>	Romantic	<div>0:00 / 3:08</div> <div>Remove</div>
2	<div> <h3>Humnava Mere</h3> <p>Jubin Nautiyal</p> </div>	Romantic	<div>0:00 / 5:29</div> <div>Remove</div>
3	<div> <h3>Saari Duniya Jalaa Denge</h3> <p>B Praak</p> </div>	Emotional	<div>0:00 / 3:02</div> <div>Remove</div>

➤ Favorites

Music-Player



Home

Your Library

Favorites

PlayList

Favorites

#	Title	Genre		Actions
1	 Chaleya Arijit Singh	Romantic	♥	▶ 0:00 / 3:08 — 🔊 ⋮
2	 Bol Do Na Zara Armaan Malik	Romantic	♥	▶ 0:00 / 4:52 — 🔊 ⋮

Project Demo link:

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