Rythimic Tunes (React)

Team Leader: Vinitha M

Team Members: Umamakeshwari V

Swetha V

Thilagavathi B

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 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.

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 favourite music streaming app, "RythimicTunes."

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.

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 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:-

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.

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@late

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.

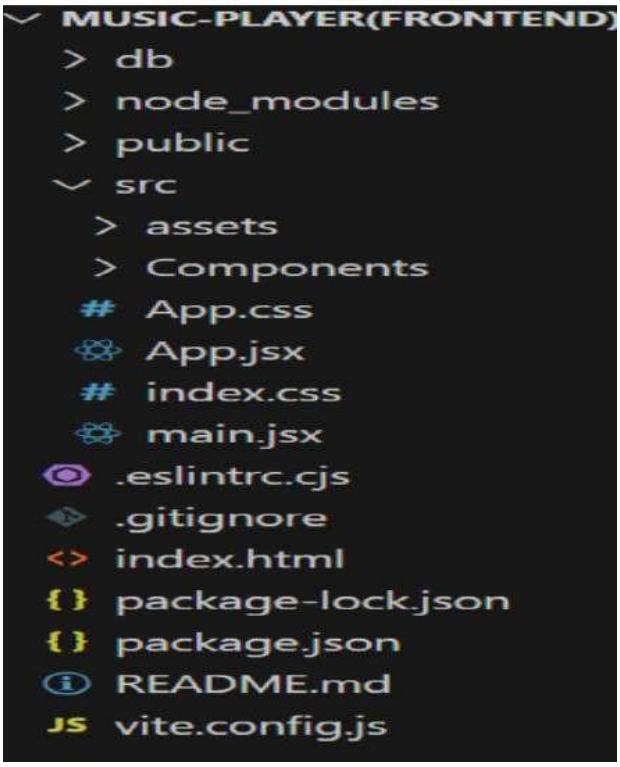
• Git: Download and installation instructions can be found at: https://git-scm.com/downloads

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:



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:**

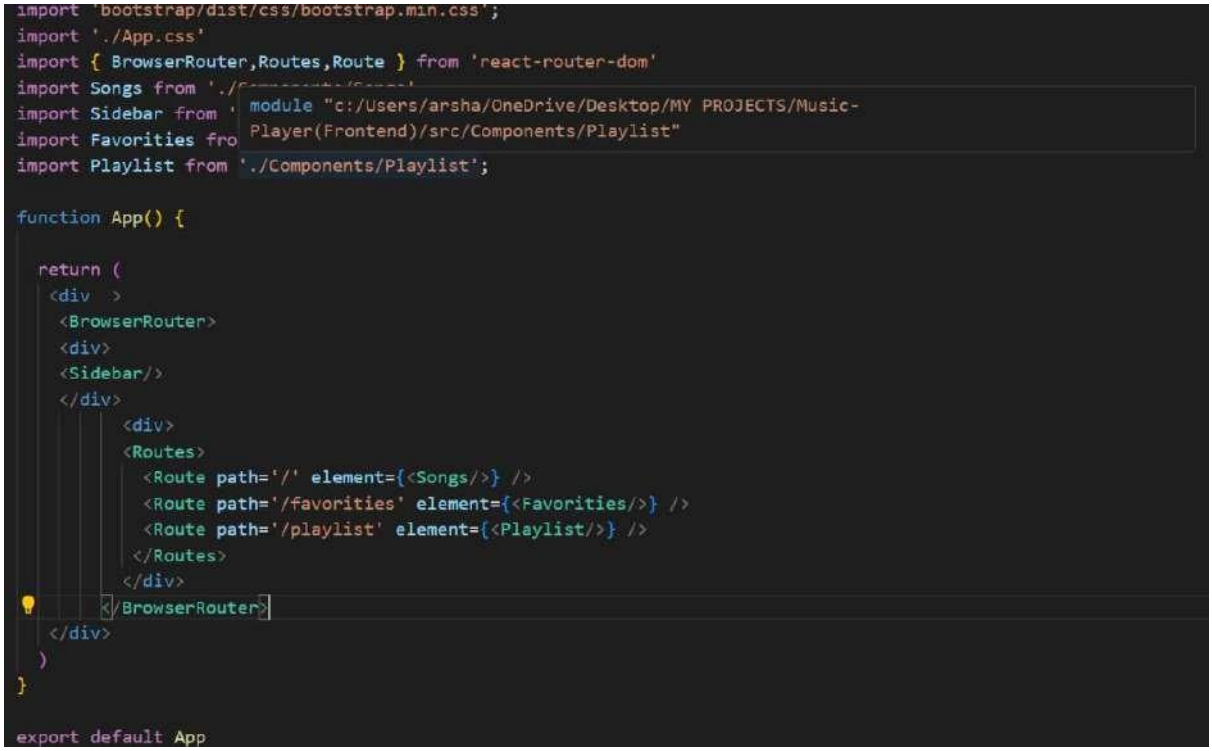
1. Setup React Application:

• Create React application.

• Configure Routing.

• Install required libraries.

Setting Up Routes:-



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:

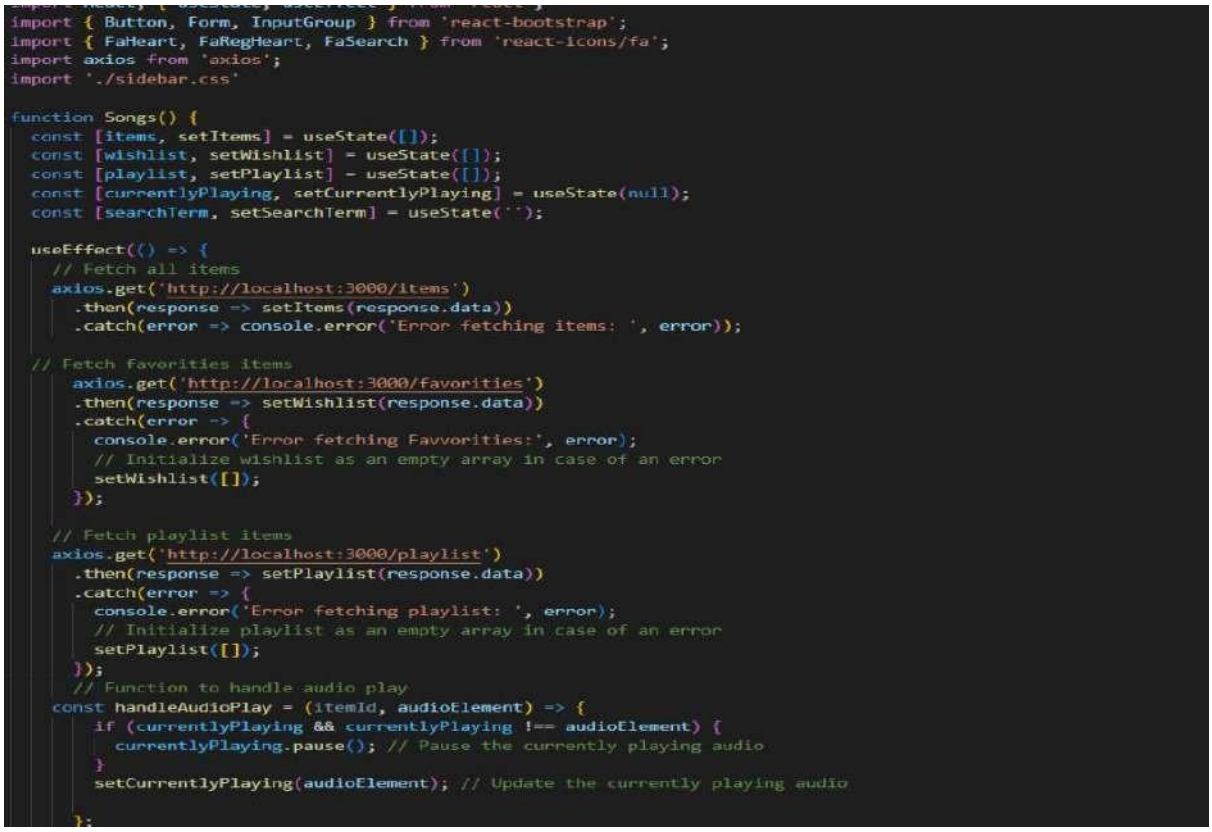
o Route with path='/' renders the Songs component when the root path is accessed (/).

o Route with path='/favourites' renders the Favourites component when the /favourites

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:



```
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

      };
```

```
    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 favourites fetched from http://localhost:3000/favorities. o playlist: Stores items added to the playlist fetched from http://localhost:3000/playlist. o currently Playing: Keeps track of the currently playing audio element.

o search Term: 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.

● 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.

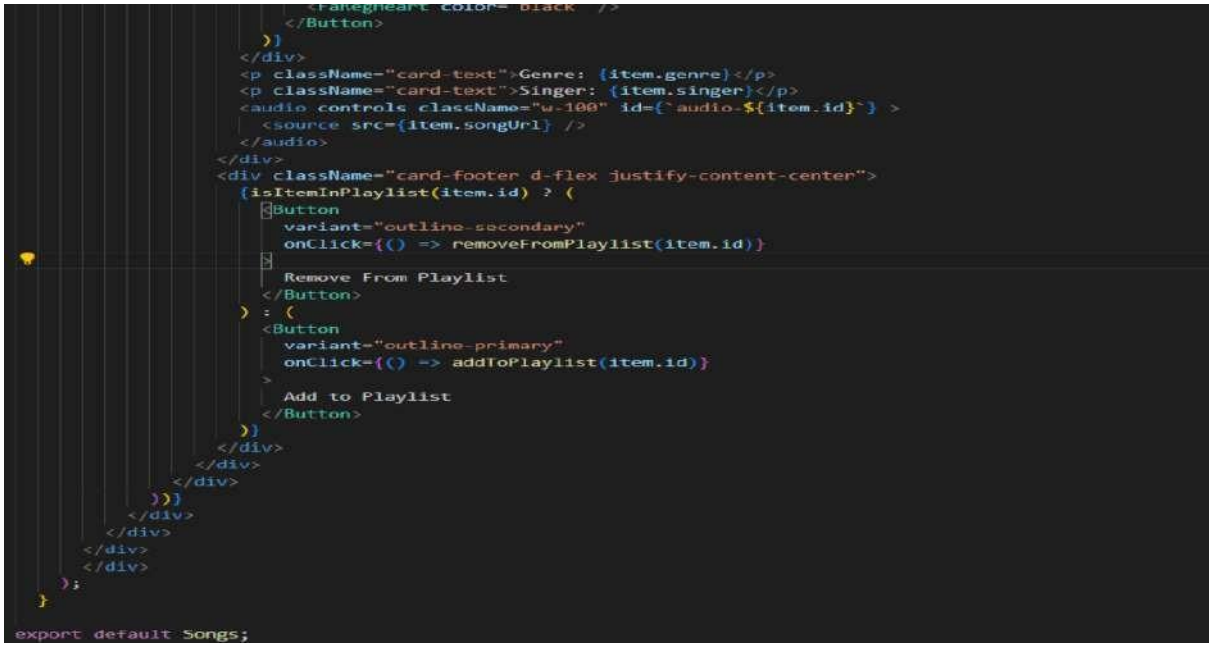
o Includes buttons (FaHeart, FaRegHeart) to add/remove items from wishlist and playlist.

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)}
```



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 fontsemibold 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 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.

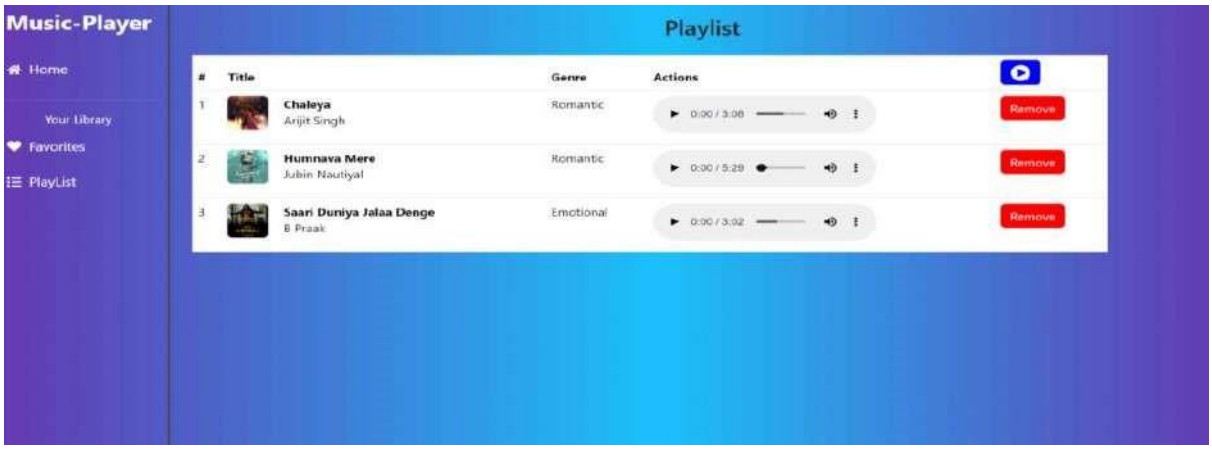
Project Execution:

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.

Playlist



Favorites