Rhythmic Tunes (React)

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.

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.

Basic Rhythm Notation

In music, rhythm notation refers to how the duration of sounds and silences are represented on the staff using various symbols. Here's a quick guide to the most common rhythmic symbols:

1. Note Values

- **Whole Note (Semibreve):**

  Duration: 4 beats.

- **Half Note (Minim):**

  Duration: 2 beats.

- **Quarter Note (Crotchet):**

  Duration: 1 beat.

- **Eighth Note (Quaver):**

  Duration: 1/2 beat.

- **Sixteenth Note (Semiquaver):**

  Duration: 1/4 beat.

2. Rest Values (Silence)

- **Whole Rest:**
  - Duration: 4 beats of silence.

- **Half Rest:**
  - Duration: 2 beats of silence.
- **Quarter Rest:**
  - Duration: 1 beat of silence.
- **Eighth Rest:**
  - Duration: 1/2 beat of silence.
- **Sixteenth Rest:**
  - Duration: 1/4 beat of silence.

3. Time Signatures

- **4/4 Time (Common Time):**
  - There are 4 beats per measure, and each quarter note gets one beat. This is the most common time signature in Western music.
- **3/4 Time:**
  - There are 3 beats per measure, and each quarter note gets one beat (used often in waltzes).
- **6/8 Time:**
  - There are 6 beats per measure, and each eighth note gets one beat.

4. Simple vs. Compound Time

- **Simple Time:**
  - Time signatures like 4/4, 3/4, or 2/4, where each beat is divided into two equal parts.

- **Compound Time:**
  - Time signatures like 6/8, 9/8, or 12/8, where each beat is divided into three equal parts (often feels like "1-2-3, 1-2-3").

These are the fundamental elements of rhythm notation. By combining these, you can create a variety of rhythms and patterns in music. Would you like to dive deeper into any of these?

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@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:

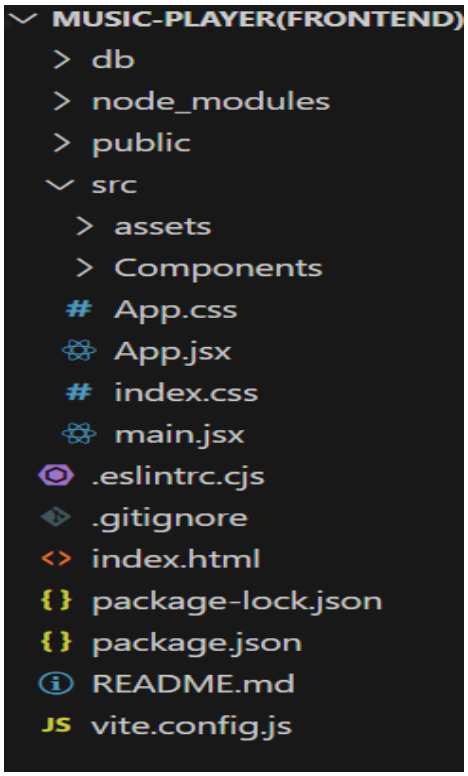
```
npm run dev
```

- **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 Bit bucket can host your repository.

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

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:-

Milestone 1: Project Setup and Configuration:

1. Install required tools and software:

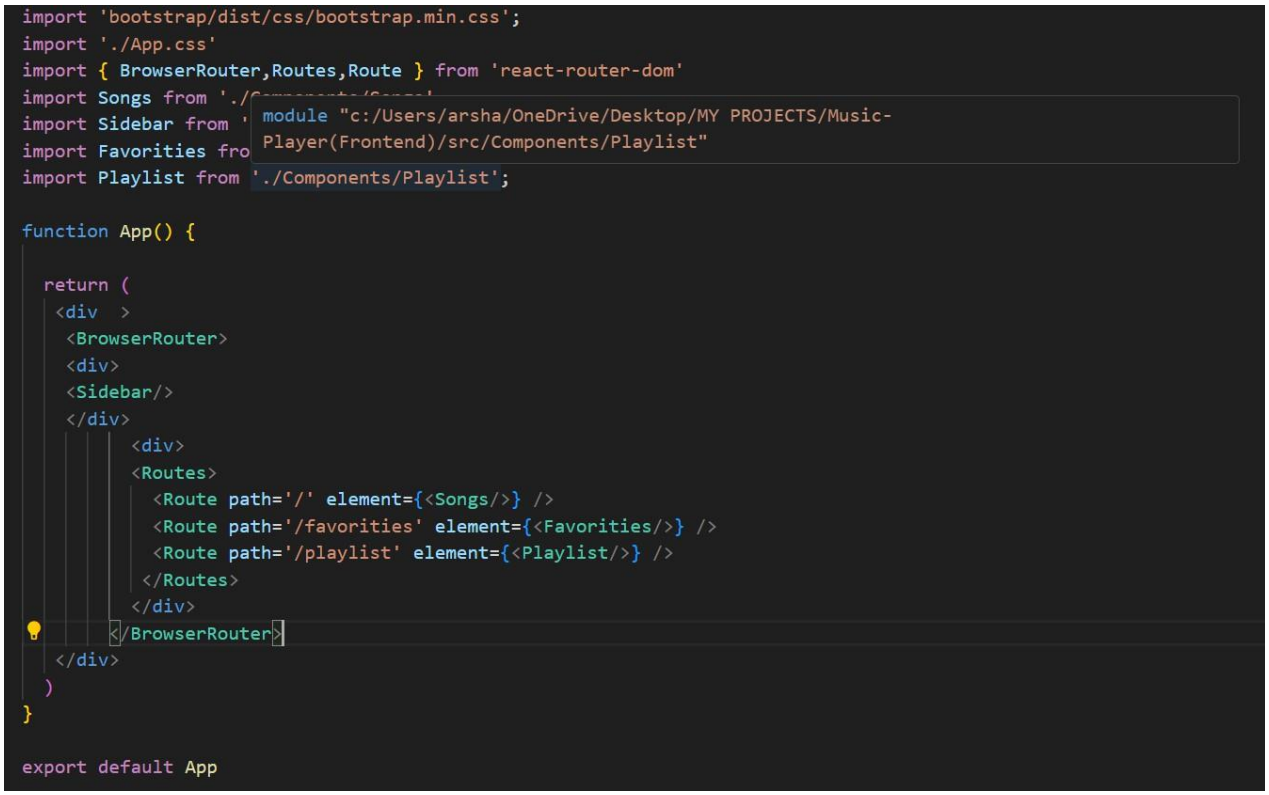
- **Installation of required tools**:

    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

Milestone 2: Project Development:

1. Setup React Application:

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



libraries. Setting Up Routes:-

Code Description:-

- ImportsBootstrap CSS (bootstrap/dist/css/bootstrap.min.css) for styling components.
- Imports custom CSS (./App.css) for additional styling.
- Imports Browser Router, 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 Browser Router as the router container to enable routing functionality.
- Includes a div as the root container for the application.
- Within Browser Router, 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='/favorities' renders the Favorities component when the /favorities 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 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:**
  - items: Holds an array of all items fetched from http://localhost:3000/items.
  - wishlist: Stores items marked as favorites fetched from http://localhost:3000/favorities.
  - 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 useEffectto 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.
  - 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.

- **Add To Wishlist (item Id):**
  - Adds an item to the wishlist (favorities) by making a POST request to http://localhost:3000/favorities.
  - Updates the wishliststate after adding an item.
- **Remove From Wishlist(itemId):**
  - Removes an item from the wishlist (favorities) by making a DELETE request to http://localhost:3000/favorities/{itemId}.
  - Updates the wishliststate after removing an item.
- **isItemInWishlist(itemId):**
  - Checks if an item exists in the wishlist (favorities) 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 wishlistand 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 wishlistand 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 divwith 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 InputGroupfrom 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 searchTermstate (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 filteredItemsarray and renders each item as a Bootstrap card ( ).
- **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.

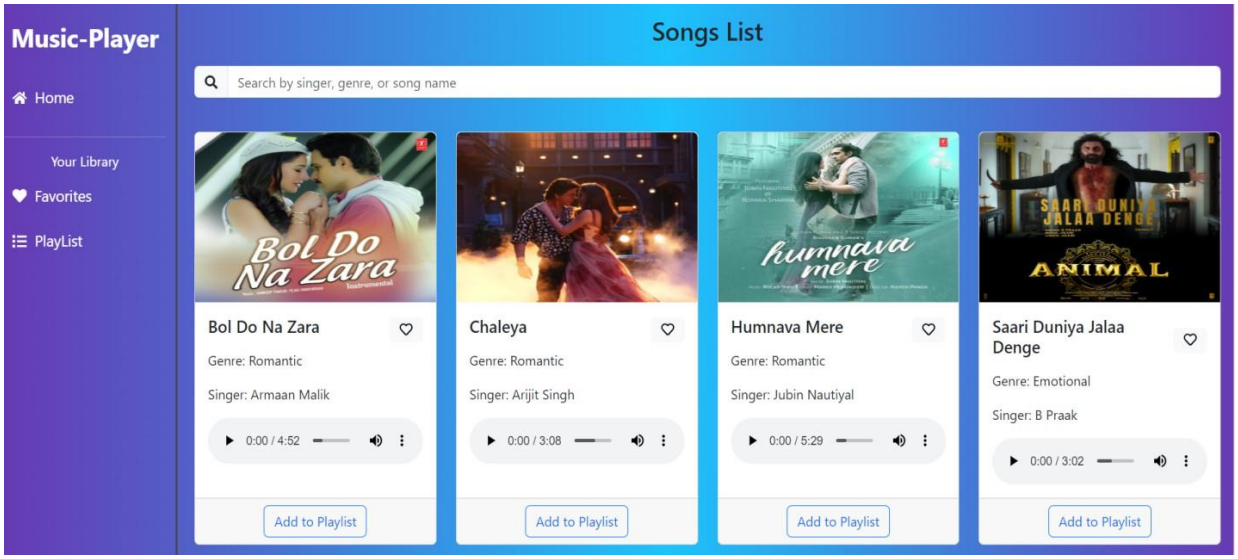
Project Execution:

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

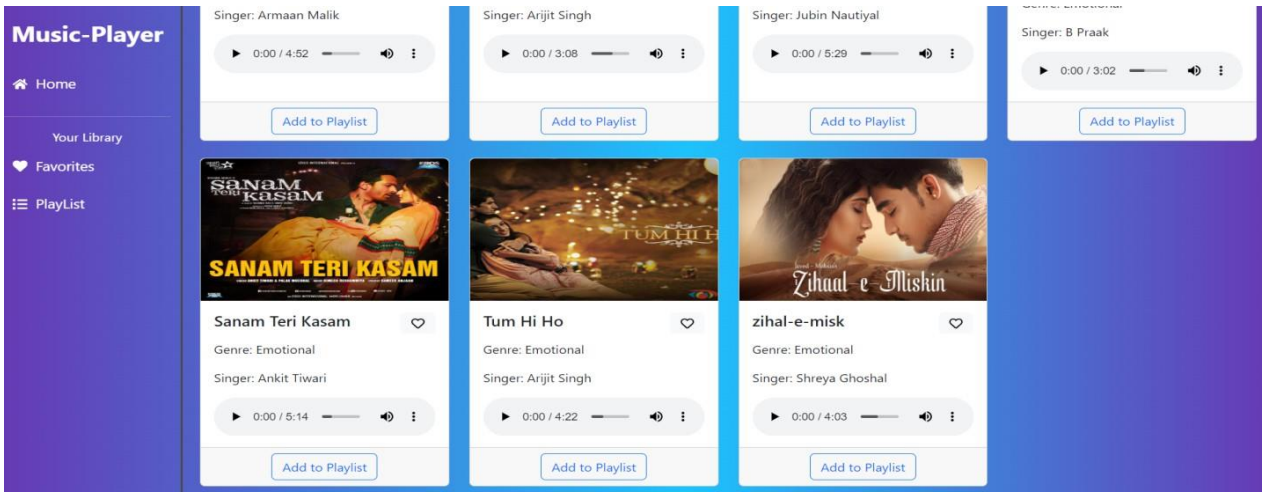
And the Open new Terminal type this command “json-server --watch ./db/db.json” to start the json server too.

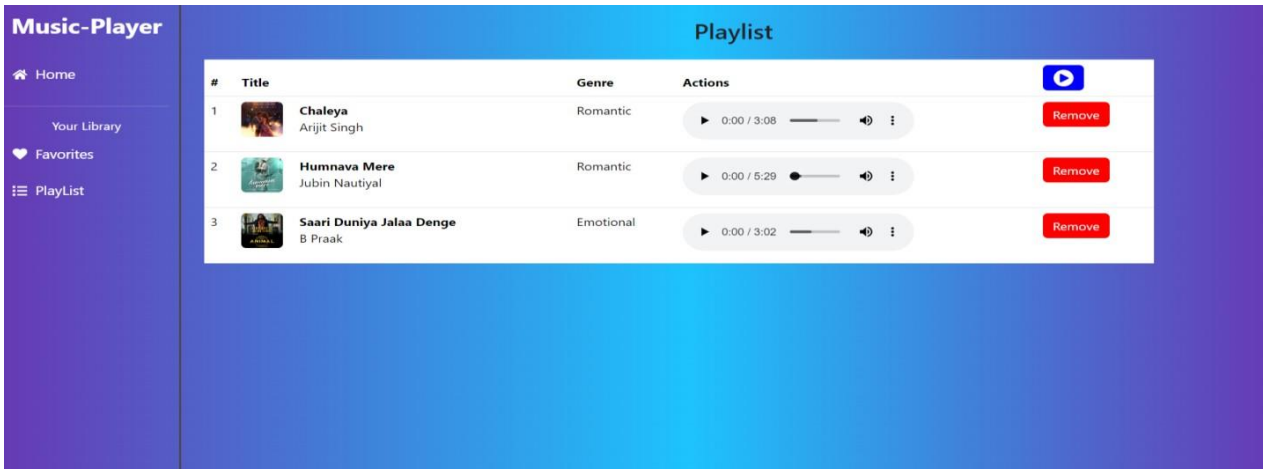
After that launch the Rythimic Tunes.

Here are some of the screenshots of the application.

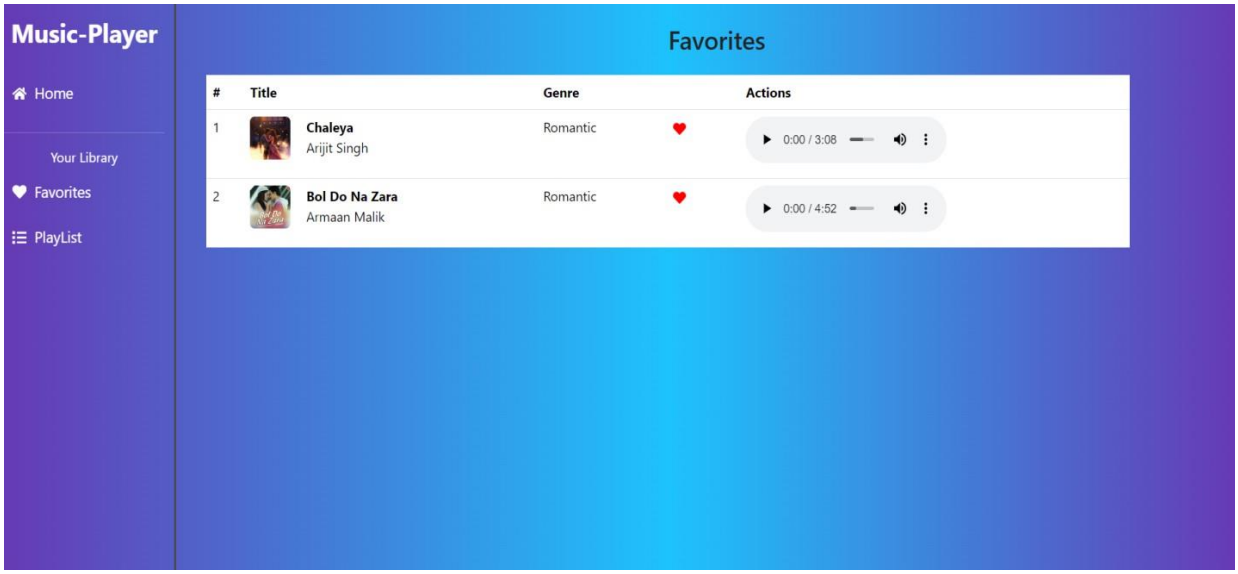


Hero components





Playlist
Favorites



conclusion: Rhythmic Tunes

Rhythmic tunes form the backbone of music, providing structure, energy, and flow to a composition. By understanding the basic elements of rhythm notation—such as note values, rests, time signatures, and rhythmic patterns—musicians can create compelling and dynamic musical pieces. Whether using simple patterns like quarter and eighth notes or more complex structures like polyrhythms and syncopation, rhythm is the key to expressing movement and emotion in music.

From classical symphonies to modern genres like jazz, funk, and electronic, rhythm plays a central role in shaping the listener's experience. Mastery of rhythmic techniques enables composers and performers to craft music that resonates deeply, whether through a steady groove or a complex, unpredictable rhythm.