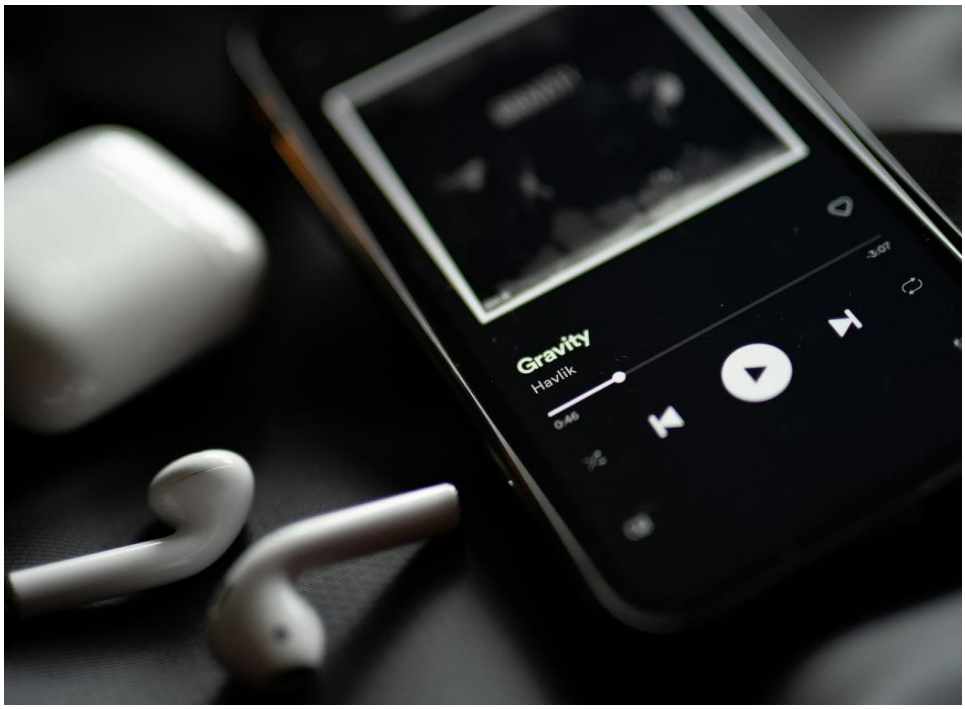


RHYTHMIC TUNES

YOUR MELODIC COMPANION

DONE BY

PREETHA R

PAVITHRA N

PAVITHRA M

RAJASHRI S

RHYTHMIC TUNES

YOUR MELODIC COMPANION

ABSTRACT:

Rhythmic Tunes is a feature-rich, React-based music streaming application designed to enhance the way users discover, listen to, and manage their favourite tracks. The application provides a seamless and personalized music experience with features such as playlist creation, offline listening, real-time lyrics, and AI-powered recommendations. Built with React.js, Node.js, and modern web technologies, it ensures a smooth and interactive user experience across all devices. Rhythmic Tunes aims to redefine how users engage with music, offering both casual listeners and music enthusiasts an all-in-one platform for an immersive auditory experience.

SYNOPSIS:

Project Title: Rhythmic Tunes – Your melodic companion

Technology Stack:
✅ Frontend: React.js, Tailwind CSS, Bootstrap
✅ Backend: Node.js, Express.js
✅ Database: MongoDB / Firebase
✅ APIs Used: Spotify API, Music match API

Problem Statement:

Traditional music players lack personalization, offline support, and interactive features. Rhythmic Tunes aims to solve these issues by offering AI-powered recommendations, social playlist sharing, and real-time lyrics**.**

Proposed Solution:

- A cloud-based music streaming app that allows users to stream, organize**,** and share music effortlessly.
- Implements modern UI/UX for a seamless and engaging experience.
- Supports multi-platform compatibility (Web, Mobile, Smart Devices).

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, "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.

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 favourite 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:

User Authentication & Profile Management

- Secure login with OAuth (Google, Facebook)
- Personalized user dashboard

Music Discovery & AI-Powered Recommendations

- Smart song suggestions based on mood, genre, and history
- Trending charts and new releases

Playlist Management & Social Sharing

- Create, edit, and organize custom playlists
- Share playlists via WhatsApp, Instagram, or Twitter

Offline Listening & Download Feature

- Download songs for offline playback
- Seamless integration with Indexed DB

Advanced Music Player Controls

- Play, pause, shuffle, repeat, and volume adjustments
- Mini player mode for multitasking

Real-time Lyrics Display

- Live karaoke-style synchronized lyrics
- Integration with Music match API

Search & Filter Functionality

- Advanced search by song title, artist, genre, or mood
- Voice search for hands-free control

Dark Mode & UI Customization

- Light/dark theme switching
- UI customization for a personalized experience

.

PRE-REQUISITES:-

Here are the key prerequisites for developing a frontend application using React.js:

NODE JS.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@LEAST

  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 JAVA SCRIPT: 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.

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

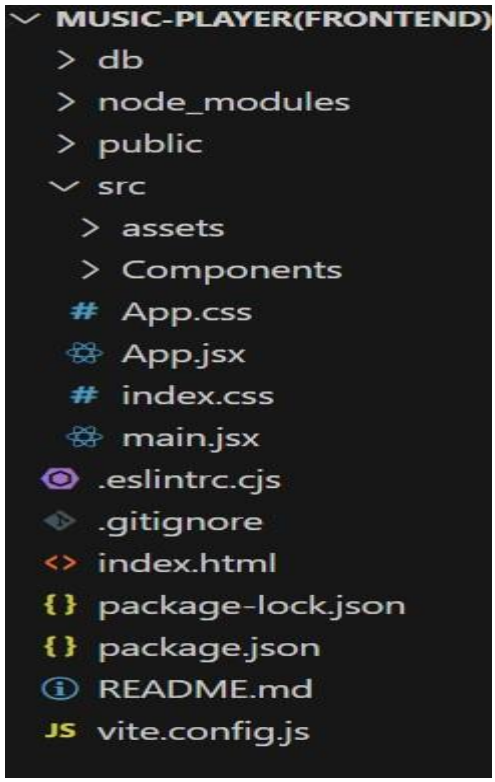
Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or Web Storm.

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, source /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.

Demo link:
https://drive.google.com/file/d/1zZuq62lyYNV_k5uu0SFjoWa35UgQ4LA9/view?usp=driv e_link

Use the code in:

https://drive.google.com/drive/folders/1BkYWfW_K3ek_UgtXNTAsDqlhdCuqz6nT?usp= drive_link

Milestone 1: Project Setup and Configuration:

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 o React Router Dom o React Icons o Bootstrap/tailwind CSS
   - AXIOS

For further reference, use the following resources
https://react.dev/learn/installation
o https://react-bootstrap-v4.netlify.app/getting-started/introduction/ 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 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='/favourite' renders the Favourite component when the /favourite 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:-

Code Description:-

- **USE STATE:**
    - ITEMS: Holds an array of all items fetched from http://localhost:3000/items.
    - WISHLIST: Stores items marked as favourites fetched from http://localhost:3000/favorities.
    - PLAYLIST: Stores items added to the playlist fetched from http://localhost:3000/playlist. Currently Playing: Keeps track of the currently playing audio element. Search Term: Stores the current search term entered by the user.

- **DATA FETCHING:**
  - Uses use effect to fetch data:
    - Fetch all items (items) from http://localhost:3000/items.
    - Fetch favourite items (wish list) from http://localhost:3000/favorities.
    - Fetches playlist items (playlist) from http://localhost:3000/playlist. Sets state variables (ITEMS, WISHLIST, PLAY LIST) based on the fetched data.
- **AUDIO PLAYBACK MANAGEMENT**:
  - Sets up audio play event listeners and clean up for each item:
    - Handle Audio Play: Manages audio playback by pausing the currently playing audio when a new one starts.
    - Handle Play: Adds event listeners to each audio element to trigger handle Audio Play.
  - Ensures that only one audio element plays at a time by pausing others when a new one starts playing.
- **ADD TO WISHLIST(ITEMID):**
  - Adds an item to the wish list (favourites) by making a POST request to http://localhost:3000/favorities. Updates the wish list state after adding an item.
- **REMOVE FROM WISHLIST(ITEMID):**
  - Removes an item from the wish list (favourites) by making a DELETE request to http://localhost:3000/favorities/{itemId}
  - Updates the wish list state after removing an item.
- **IS ITEM IN WISHLIST(ITEMID):**
  - Checks if an item exists in the wish list (favourites) based on its ITEMID

- **ADD TO PLAYLIST(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.
- **REMOVE FROM PLAYLIST(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.
- **IS ITEM IN PLAYLIST(ITEMID):**
  - O Checks if an item exists in the playlist (playlist) based on its ITEMID
- **FILTERED ITEMS:**
  - Filters items based on the search Term.
  - Matches title, singer, or genre with the lowercase version of search Term.
- **JSX:**
  - Renders a form with an input field (Form, Input Group, Button, Fa Search) for searching items.
  - Maps over filtered Items to render each item in the UI.
  - Includes buttons (Fa Heart, REG Heart) to add/remove items from wish list 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).Handles errors when adding/removing items from wish list 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", justify Content "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" (class Name="text-3xl font-semi bold mb-4 text -").
- **Search Input:**
  - Utilizes Input Group 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 search Term state (value={search Term}) and updates it on change(on Change={(e) => set Search Term (e. target .value)}).
  - Styled with class Name="search-input".

- **Card Layout:**
  - Uses Bootstrap grid classes (row, col) to create a responsive card layout (class Name="row row-cols-1 row-cols-md-2 row-cols-lg-3 row-cols-xl-4 g-4").
  - Maps over filtered Items array and renders each item as a Bootstrap card
    - (<div class Name="card h-100">
    -

  Card Content:
  - Displays the item's image (<IMAGE>), title (<h5 Class Name="card-title">), genre (<p class Name="card-text">), and singer (<p class Name="card-text">).

- Includes an audio player (<audio controls class Name="w-100" id= {audio-${item.id}}>) for playing the song with a source (<source source= {item. Song URL} />).

- **Wish list and Playlist Buttons:**
  - Adds a heart icon button (<Button>) to add or remove items from the wish list
    (IS Item In Wish list (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
    (is Item In Playlist(item.id)).

- **Button Click Handlers**:
  - Handles adding/removing items from the wish list
    (add To Wish list(item.id), remove From Wish list(item.id)).
  - Manages adding/removing items from the playlist
    (add To Playlist(item.id), remove From Playlist(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:

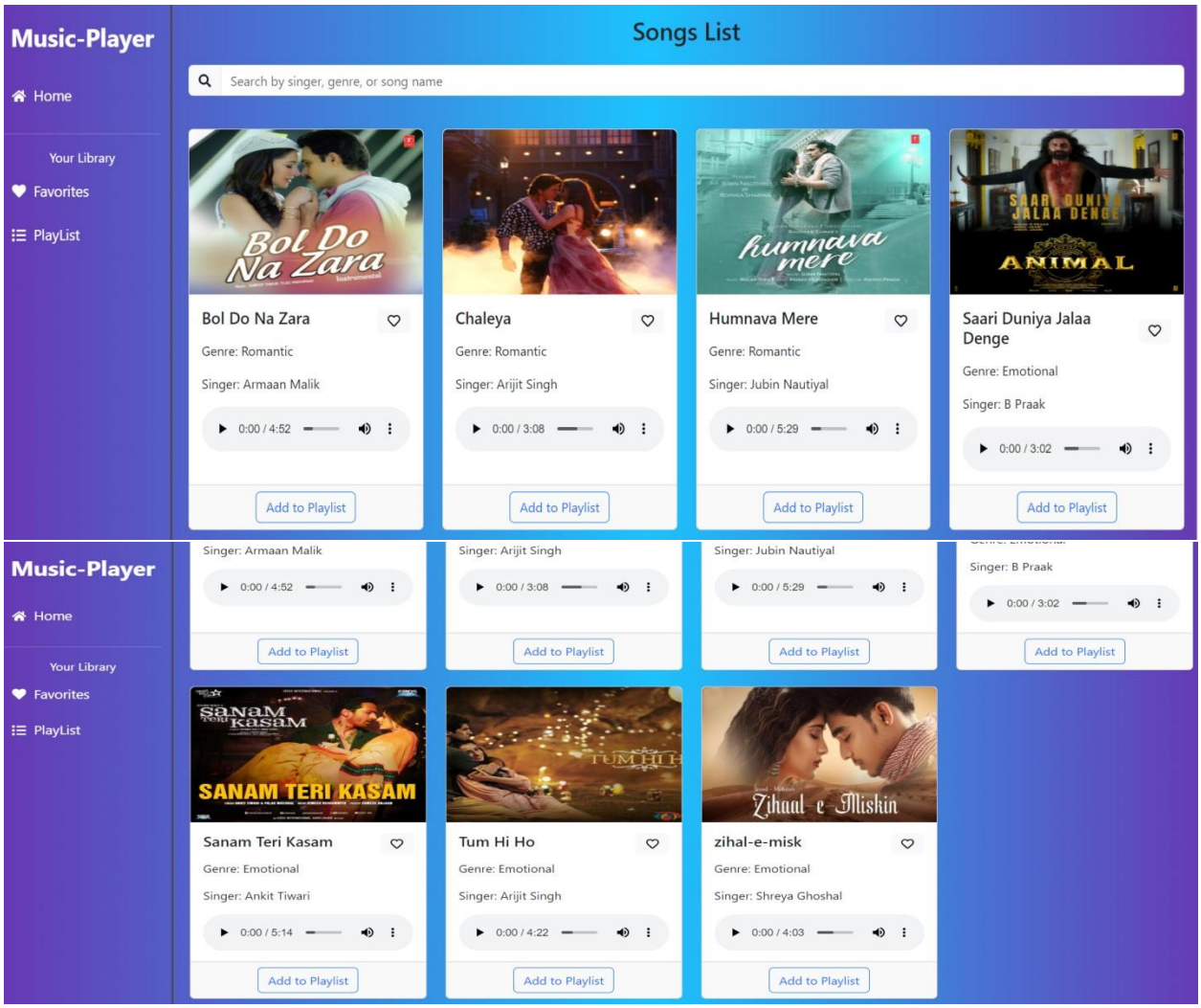
After completing the code, run the react application by using the command “NPM start” or “NPM run dev” if you are using vite.js

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

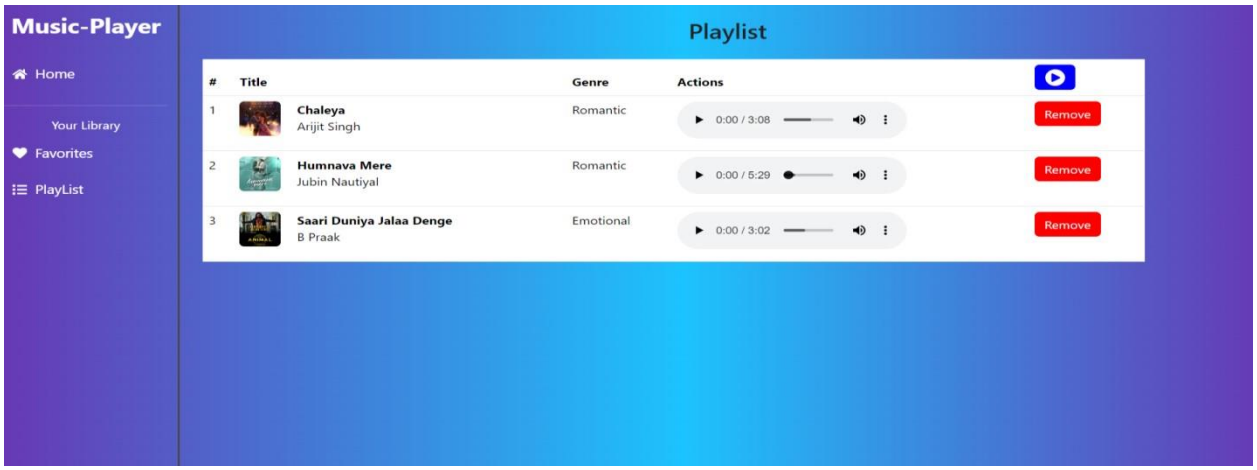
After that launch the RHYTHMIC TUNES

Here are some of the screenshots of the application.

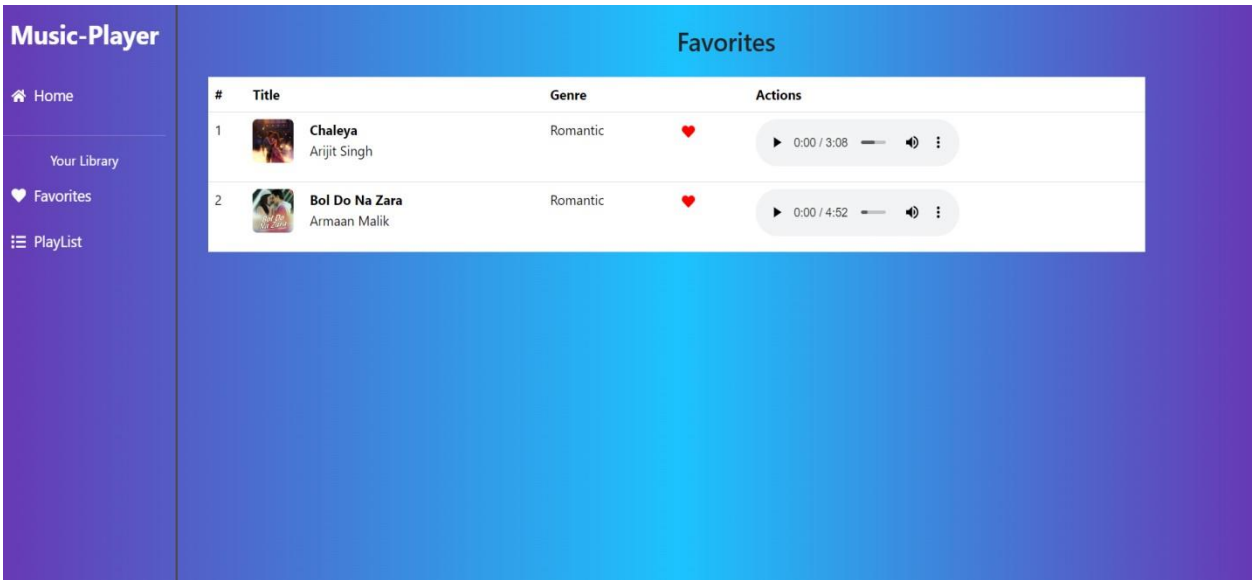
HERO COMPONENTS:



PLAYLIST:



FAVOURITES:



Demo link:

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

CONCLUSION:

Rhythmic Tunes is designed to be more than just a music streaming platform—it is a personalized, AI-powered music companion that adapts to the user's mood and preferences. With real-time lyrics, offline playback, playlist collaboration, and AI-based recommendations, the platform aims to bridge the gap between music and technology. Using React.js, Node.js, and cloud storage solutions, the app ensures scalability and high performance.

By integrating social sharing and a community-driven experience, Rhythmic Tunes encourages users to discover, enjoy, and share music like never before. 🚀 🎵

*****KEEP BUILDING AWESOME STUFF*****