

Rythimic Tunes (React)

A Music Streaming Application Build with
React.js

TEAM 4:

R.Vanitha-212203815

K.KavyaShri-212203803

P.Swathi-212203813

G.Dhanush-212203728

Introduction:-

Welcome to the next evolution of music streaming!
Rythimic Tunes is designed to offer an immersive audio experience, leveraging the power of React.js to create a dynamic and engaging platform.

Our application brings together modern technology and user-focused design, making it easy for you to explore, listen to, and manage your favorite tracks. With Rythimic Tunes, you can effortlessly discover new artists, create

personalized playlists, and enjoy seamless playback across multiple devices.

Whether you're looking for trending hits or nostalgic classics, our platform ensures a rich and enjoyable musical experience tailored to your preferences. Built with React.js, our application provides a visually appealing and highly interactive interface, ensuring smooth navigation and a seamless listening experience.

No matter where you are—on a desktop, tablet, or smartphone—you can access your favorite music anytime, anywhere. Get ready to embrace the future of music streaming with Rhythmic Tunes. Hit play and experience a whole new way to enjoy your favorite tunes!

Abstraction in Rhythmic Tunes (React)

Abstraction in Rhythmic Tunes (React) plays a crucial role in simplifying complex functionalities by breaking them into modular, reusable components. Instead of handling intricate audio interactions directly within the UI, the application abstracts key elements like sound playback, rhythmic patterns, and user controls into well-defined structures.

This separation ensures that each component handles a specific task, such as managing beats, adjusting tempo, or controlling audio playback, making the system more maintainable and scalable.

State management is another layer of abstraction that helps in synchronizing user interactions with real-time music generation.

Using React's state and context APIs, the application efficiently updates UI elements without unnecessary re-renders, ensuring smooth performance. Abstraction also extends to handling API calls or integrating external sound libraries, allowing seamless enhancements without disrupting core functionalities.

By implementing abstraction, Rhythmic Tunes keeps its codebase structured, reduces redundancy, and enhances flexibility. This modular approach allows for easier debugging, future upgrades, and the integration of additional features like customizable sound packs or AI-generated rhythms, making the application both efficient and user-friendly.

Scenario-Based Intro:-

Picture yourself walking through a lively city street, surrounded by the hum of conversations, honking cars, and street musicians. You take out your phone and open Rythimic Tunes, eager to enhance the moment with the perfect soundtrack.

As soon as you start the app, a personalized playlist begins. A high-energy pop song sets the tone for your morning walk.

When you step onto the subway, the playlist seamlessly transitions into a mellow acoustic track, matching the calmness of your journey.

No matter the situation—whether you're working out, studying, or unwinding after a long day—Rythimic Tunes intelligently adapts to your mood and preferences, ensuring that the right music is always playing at the right time.

Target Audience

Rythimic Tunes is designed to cater to a broad spectrum of music lovers, including:

-

Casual Listeners – Those who love playing background music while working or relaxing.

Music Aficionados – Enthusiasts who enjoy curating playlists and exploring new genres.

Frequent Travelers – Users who need music on the go, with offline listening capabilities.

Tech-Savvy Individuals – People interested in trying the latest in music streaming technology.

Project Goals and Objectives:-

Our primary aim is to develop a user-friendly platform that enhances the way people discover, stream, and organize their music. We have set the following objectives to achieve this goal:

Simple and Intuitive Interface– Ensure effortless navigation for users of all experience levels.

Seamless Music Streaming – Provide uninterrupted playback with smooth transitions.

Advanced Search Capabilities – Enable users to find specific songs, albums, and artists quickly.

Modern Development Stack – Utilize cutting-edge technologies like React.js to enhance performance and scalability.

Key Features

Rythimic Tunes offers a variety of features to create a superior listening experience:

Extensive Song Library– Access a vast collection of songs across different genres and artists.

Custom Playlists – Build, save, and share playlists with your personalized selections.


Smart Search – Find music effortlessly with intelligent filters and suggestions.

Playback Controls – Play, pause, skip, and adjust volume with ease

Offline Mode– Download songs for listening without an internet connection.

 db
Modified 28 Dec 2023 

 public
Modified 28 Dec 2023 

 src
Modified 28 Dec 2023 

 .eslintrc.cjs
 Modified 16 Dec 2023 

 .gitignore
 Modified 16 Dec 2023 

 index.html
 Modified 16 Dec 2023 

 package-lock.json
 Modified 16 Dec 2023 

 package.json
 Modified 16 Dec 2023 

 README.md
 Modified 16 Dec 2023 

 vite.config.js
 Modified 16 Dec 2023 

Pre-Requisites

To develop the frontend of Rythimic Tunes using React.js, you need the following:

- Node.js and npm– Install these to manage dependencies and run the development environment. Download: [Node.js Official Website](<https://nodejs.org/en/download/>)
- React.js – The core library for building the UI. Install it using: `sh npm create vite@latest`
- Basic Web Technologies – Knowledge of HTML, CSS, and JavaScript is essential.
- Version Control – Use Git to track changes and collaborate with others. - Download Git: [Git Official Website](<https://git-scm.com/downloads>)
- Code Editor – Recommended editors include VS Code, Sublime Text, or WebStorm.

Project Structure

Proper organization of files is key to maintaining and scaling the application. Here's a basic structure:

src/components– Houses reusable UI components.

src/pages– Contains the main pages of the application.

src/assets – Stores images and static files.

Db/ – Includes a local JSON database for testing.

package.json – Manages project dependencies. This structure ensures modularity, making it easy to maintain and extend the application

1. Setup React Applications:

```
App.jsx X
src > App.jsx > ...
1  import 'bootstrap/dist/css/bootstrap.min.css';
2  import './App.css'
3  import { BrowserRouter, Routes, Route } from 'react-router-dom'
4  import Songs from './Components/Songs'
5  import Sidebar from './Components/Sidebar'
6  import Favorities from './Components/Favorities'
7  import Playlist from './Components/Playlist';
8
9
10 function App() {
11
12   return (
13     <div>
14       <BrowserRouter>
15       <div>
16         <Sidebar/>
17       </div>
18
19       <div>
20         <Routes>
21           <Route path='/songs' element={}<Songs/> } />
22           <Route path='/favorities' element={}<Favorities/> } />
23           <Route path='/playlist' element={}<Playlist/> } />
24         </Routes>
25       </div>
26     </BrowserRouter>
27
28   </div>
29 )
30 }
31
32 export default App
33
```

2.HTML Code:

```
index.html X
index.html > ...
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.16/dist/tailwind.min.css" rel="stylesheet">
8     <title>Music-Player</title>
9   </head>
10  <body>
11    <div id="root"></div>
12    <script type="module" src="/src/main.jsx"></script>
13  </body>
14 </html>
15
```

3. Frontend Code for Displaying Songs:

```
Songs.jsx X
src > Components > Songs.jsx > ...
1 import React, { useState, useEffect } from 'react';
2 import { Button, Form, InputGroup } from 'react-bootstrap';
3 import { FaHeart, FaRegHeart, FaSearch } from 'react-icons/fa';
4 import axios from 'axios';
5 import './sidebar.css'
6
7 function Songs() {
8   const [items, setItems] = useState([]);
9   const [wishlist, setWishlist] = useState([]);
10  const [playlist, setPlaylist] = useState([]);
11  const [currentlyPlaying, setCurrentlyPlaying] = useState(null);
12  const [searchTerm, setSearchTerm] = useState('');
13
14  useEffect(() => {
15    // Fetch all items
16    axios.get('http://localhost:3000/items')
17      .then(response => setItems(response.data))
18      .catch(error => console.error('Error fetching items: ', error));
19
20    // Fetch favorites items
21    axios.get('http://localhost:3000/favorites')
22      .then(response => setWishlist(response.data))
23      .catch(error => {
24        console.error('Error fetching Favorites: ', error);
25        // Initialize wishlist as an empty array in case of an error
26        setWishlist([]);
27      });
28
29    // Fetch playlist items
30    axios.get('http://localhost:3000/playlist')
31      .then(response => setPlaylist(response.data))
32      .catch(error => {
33        console.error('Error fetching playlist: ', error);
34        // Initialize playlist as an empty array in case of an error
35        setPlaylist([]);
36      });
37    // Function to handle audio play
```

4. Terminal:

Project Flow

Before starting development, take a look at the project demo

[Demo Link](
)

Milestone 1:

Project Setup and Configuration

1. Install necessary tools like React.js, React Router, Bootstrap, and Axios.
2. Set up the project using ``npm create vite@latest``.
3. Initialize version control using Git.

Milestone 2:

Developing Core Features

1. Implement navigation and routing.

2. Create components for song listings, playlists, and favorite songs.
3. Integrate search functionality and music playback controls.

Milestone 3:

Testing and Deployment

1. Test responsiveness and cross-browser compatibility.
2. Optimize performance for smooth streaming.
3. Deploy the final version on Vercel or Netlify

Fetching and Managing Songs

Rhythmic Tunes retrieves and organizes songs dynamically:

Data Fetching— Uses Axios to load songs from the database.

State Management – Utilizes useState to handle playlists and favorites.

Audio Playback— Ensures smooth transitions when switching between tracks.

Example API endpoints:

GET /items` -Fetch all songs

POST /favorites` – Add a song to favorites

DELETE /favorites/:id` – Remove a song from favorites

Frontend Code for Displaying Songs

The song list is displayed using React Bootstrap for a clean and modern UI. jsx

```
{filteredItems.map((item) => (
```

```
{item.title}
```

```
{item.artist}
```

```
))}
```

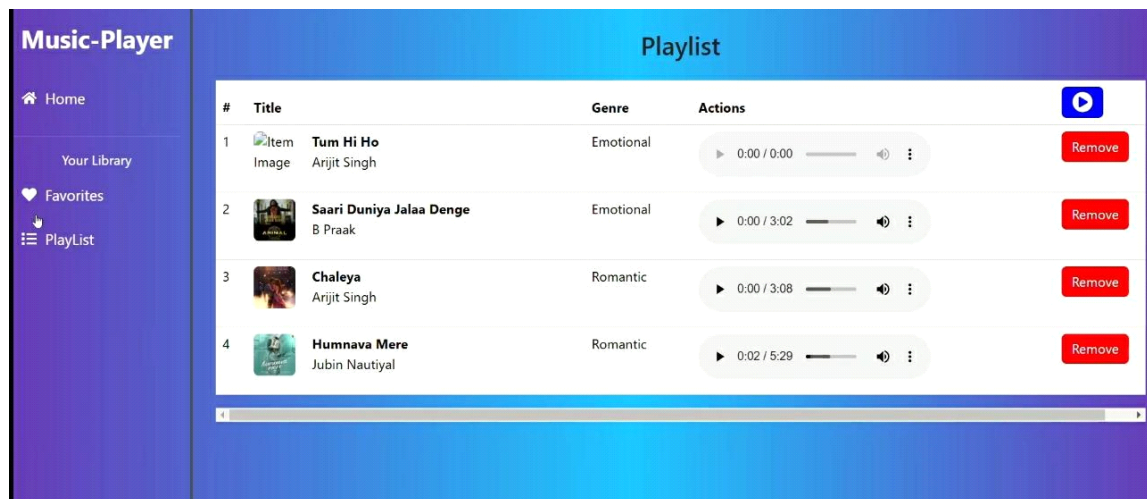
This layout ensures an elegant and responsive music catalog.

```
Songs.jsx X
src > Components > Songs.jsx > ...
1 import React, { useState, useEffect } from 'react';
2 import { Button, Form, InputGroup } from 'react-bootstrap';
3 import { FaHeart, FaRegHeart, FaSearch } from 'react-icons/fa';
4 import axios from 'axios';
5 import './sidebar.css'
6
7 function Songs() {
8   const [items, setItems] = useState([]);
9   const [wishlist, setWishlist] = useState([]);
10  const [playlist, setPlaylist] = useState([]);
11  const [currentlyPlaying, setCurrentlyPlaying] = useState(null);
12  const [searchTerm, setSearchTerm] = useState('');
13
14  useEffect(() => {
15    // Fetch all items
16    axios.get('http://localhost:3000/items')
17      .then(response => setItems(response.data))
18      .catch(error => console.error('Error fetching items: ', error));
19
20    // Fetch favorites items
21    axios.get('http://localhost:3000/favorites')
22      .then(response => setWishlist(response.data))
23      .catch(error => {
24        console.error('Error fetching Favvorities:', error);
25        // Initialize wishlist as an empty array in case of an error
26        setWishlist([]);
27      });
28
29    // Fetch playlist items
30    axios.get('http://localhost:3000/playlist')
31      .then(response => setPlaylist(response.data))
32      .catch(error => {
33        console.error('Error fetching playlist: ', error);
34        // Initialize playlist as an empty array in case of an error
35        setPlaylist([]);
36      });
37    // Function to handle audio play
```

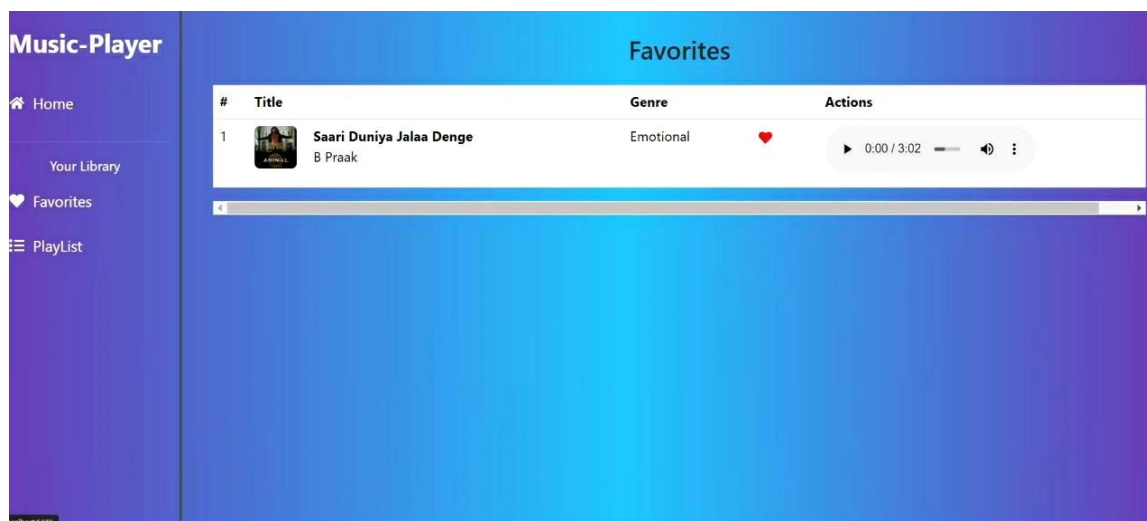
Project Execution:-

After completing development, run the application using:

sh npm run dev To enable the backend, start the JSON server: sh json-server --watch ./db/db.json Once both are running, open Rythimic Tunes and start enjoying your music!



Page 2



Project Demo:

[Watch Demo]()

Conclusion:

Rhythmic tunes are more than just patterns of sound—they are the pulse of life, bringing harmony to our emotions and energy to our movements. As each beat resonates, it connects us to the past, drives us through the present, and inspires the future. Though a song may end, its rhythm lingers, reminding us that music is a timeless force that unites and uplifts us all.