# Rythimic Tunes

## 1. Introduction

**Project Title:** Rythimic Tunes

**Team Members:**

* **Team leader-R.Sakthi**
* **Team Members-D.Saran**

**-rajinish**

**-Saravana**

**-Sathish**

## 2. Project Overview

### ****Purpose:****

Rythimic Tunes is a modern music streaming application built using **React.js** and **Bootstrap**, designed to provide users with a seamless and immersive music experience. The platform allows users to browse a vast music library, create and manage playlists, and enjoy high-quality audio playback with intuitive controls. With a user-friendly interface and responsive design, the app ensures accessibility across different devices.

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.

### ****Features:****

* **Song Listings:** Browse and explore a vast collection of songs.
* **Playback Control:** Full-fledged audio player with play, pause, skip, and volume control.
* **Playlist Management:** Create, edit, and organize custom playlists.
* **Search & Filter:** Quickly find songs, artists, and albums.
* **User Authentication:** Secure sign-in and profile management.
* **Dark Mode:** Toggle between light and dark themes for a personalized experience.
* **Offline Listening:** Download songs for offline playback (planned future enhancement).

## 3. Architecture

**Component Structure:**

* App.js: Root component managing routing and layout.
* Sidebar.js: Navigation menu.
* Songs.js: Displays list of songs and handles playback.
* Playlist.js: Manages user-created playlists.
* Favorites.js: Handles user favorites.
* Player.js: Controls music playback.
* Auth.js: Manages user authentication and profile settings.

**State Management:**

* **Local State:** Managed using React’s useState hook.
* **Global State:** Context API used for shared state like user authentication and music library.
* **Caching:** Local storage is utilized for quick access to user preferences and offline songs.

**Routing:**

* Implemented using react-router-dom.
* Routes include Home, Playlist, Favorites, Profile, and Now Playing.

## 4. Setup Instructions

**Prerequisites:**

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.

**Installation:**

1. Clone the repository:

git clone https://github.com/Sakthi-Developer/music-streaming-app.git

1. Navigate to the project folder:

cd music-streaming-app-main

1. Install dependencies:

npm install

1. Start the development server:

npm run dev

**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

## 5. Folder Structure

Client

* src/components/ - UI components (buttons, cards, modals, etc.)
* src/pages/ - Application pages (Home, Playlist, Profile, etc.)
* src/assets/ - Images, icons, and styles
* src/utils/ - Utility functions and custom hooks
* db/ - JSON server data for development

### ****Utilities (****src/utils/****)****

* api.js - Handles API requests, including fetching songs, user authentication, and playlist data.
* helpers.js - Contains helper functions for tasks like date formatting, data validation, and string manipulation.
* storage.js - Manages local storage functions for caching user preferences and offline music.
* playerUtils.js - Provides utility functions for handling audio playback, such as play, pause, seek, and volume control.

## 6. Running the Application

* Open music-streaming-app-main folder
* Start the frontend server:

npm run dev

## 7. Component Documentation

**Key Components:**

* Songs.js: Displays songs and handles playback.
* Playlist.js: Manages user playlists.
* Favorites.js: Stores user favorites.
* Player.js: Controls music playback, including progress tracking.

**Reusable Components:**

* Button.js: Standardized buttons.
* Card.js: UI component for displaying song details.
* Modal.js: Used for pop-ups and dialog interactions.

## 8. State Management

In Rythimic Tunes, state management plays a crucial role in ensuring seamless data flow across the application, improving user experience and performance.

#### ****Global State Management:****

* **Context API:** The app uses React’s Context API to manage shared state globally, ensuring efficient communication between components.
* **Application-wide State:**
  + **User Authentication:** Stores user login status, profile details, and preferences.
  + **Music Library:** Keeps track of available songs, playlists, and favorites.
  + **Playback State:** Maintains information about the currently playing song, playback progress, and queue.
* **Performance Optimization:** Context is structured in modular providers to prevent unnecessary re-renders.

#### ****Local State Management:****

* **useState Hook:** Used within individual components to handle UI-related state, such as toggling modals, managing form inputs, and controlling theme settings.
* **useEffect Hook:** Manages side effects like fetching data from APIs, updating playback progress, and storing user preferences in local storage.

#### ****Caching & Persistent Storage:****

* **Local Storage:** Saves user preferences, theme settings, and last played song to improve load times.
* **Session Storage:** Temporarily holds non-essential data like UI states for a smoother user experience.

## 9. User Interface

* **Homepage:** Displays all songs with search and filter options.
* **Playlist Page:** Allows users to manage and edit playlists.
* **Favorites Page:** Users can view their liked songs.
* **Now Playing Page:** Showcases current playback details and controls.
* **Profile Page:** Displays user details and settings.

## 10. Styling

Rythimic Tunes utilizes a combination of **Bootstrap, Tailwind CSS, and Styled-Components** to achieve a modern, responsive, and visually appealing user interface.

### ****CSS Frameworks/Libraries:****

* **Bootstrap**: Provides a responsive grid system and pre-designed components for faster UI development.
* **Tailwind CSS**: Used for utility-based styling, enabling a highly customizable and flexible design.
* **Styled-Components**: Allows dynamic styling through JavaScript, making it easier to implement theme-based styles.

### ****Theming:****

* Supports **Dark Mode and Light Mode** with smooth transitions.
* CSS variables and Styled-Components ensure easy theme customization.
* User preferences for themes are saved in local storage for a personalized experience.

## 11. Testing

Rythimic Tunes follows a structured testing approach to ensure the reliability and stability of the application. The testing process includes **unit, integration, and end-to-end (E2E) testing** to verify the correctness of components and user interactions.

### ****Testing Strategy:****

* **Unit Testing**: Conducted using **Jest** to test individual components and utility functions.
* **Integration Testing**: Utilizes **React Testing Library** to verify component interactions and data flow.
* **End-to-End (E2E) Testing**: Uses **Cypress** to simulate real user journeys, ensuring smooth navigation and functionality.

### ****1. Unit Testing (Jest)****

Unit tests focus on individual functions or components. Here’s an example test for the **Player** component.

#### Player.test.js ****(Unit Test)****

javascript

CopyEdit

import { render, screen, fireEvent } from '@testing-library/react';import Player from '../components/Player';

describe('Player Component', () => {

test('renders player controls', () => {

render(<Player />);

expect(screen.getByTestId('play-button')).toBeInTheDocument();

expect(screen.getByTestId('pause-button')).toBeInTheDocument();

});

test('plays the song when play button is clicked', () => {

render(<Player />);

const playButton = screen.getByTestId('play-button');

fireEvent.click(playButton);

expect(screen.getByText(/playing/i)).toBeInTheDocument();

});

test('pauses the song when pause button is clicked', () => {

render(<Player />);

const pauseButton = screen.getByTestId('pause-button');

fireEvent.click(pauseButton);

expect(screen.getByText(/paused/i)).toBeInTheDocument();

});

});

### ****2. Integration Testing (React Testing Library)****

Integration tests verify if different components work together as expected. Here's an example of testing the **Playlist and Player** components.

#### Playlist.test.js ****(Integration Test)****

javascript

CopyEdit

import { render, screen, fireEvent } from '@testing-library/react';import Playlist from '../components/Playlist';import Player from '../components/Player';

describe('Playlist and Player Integration', () => {

test('clicking a song in the playlist updates the player', () => {

render(

<>

<Playlist />

<Player />

</>

);

const songItem = screen.getByText('Song 1');

fireEvent.click(songItem);

expect(screen.getByText('Now Playing: Song 1')).toBeInTheDocument();

});

});

### ****3. End-to-End (E2E) Testing (Cypress)****

E2E tests check the full user journey in the browser. This test verifies if a user can **search for a song, play it, and see it in the "Now Playing" section**.

#### player.cy.js ****(E2E Test)****

javascript

CopyEdit

describe('Music Player E2E Test', () => {

beforeEach(() => {

cy.visit('http://localhost:3000');

});

it('allows user to search and play a song', () => {

cy.get('input[placeholder="Search"]').type('Song 1');

cy.contains('Song 1').click();

cy.get('[data-testid="play-button"]').click();

cy.contains('Now Playing: Song 1').should('be.visible');

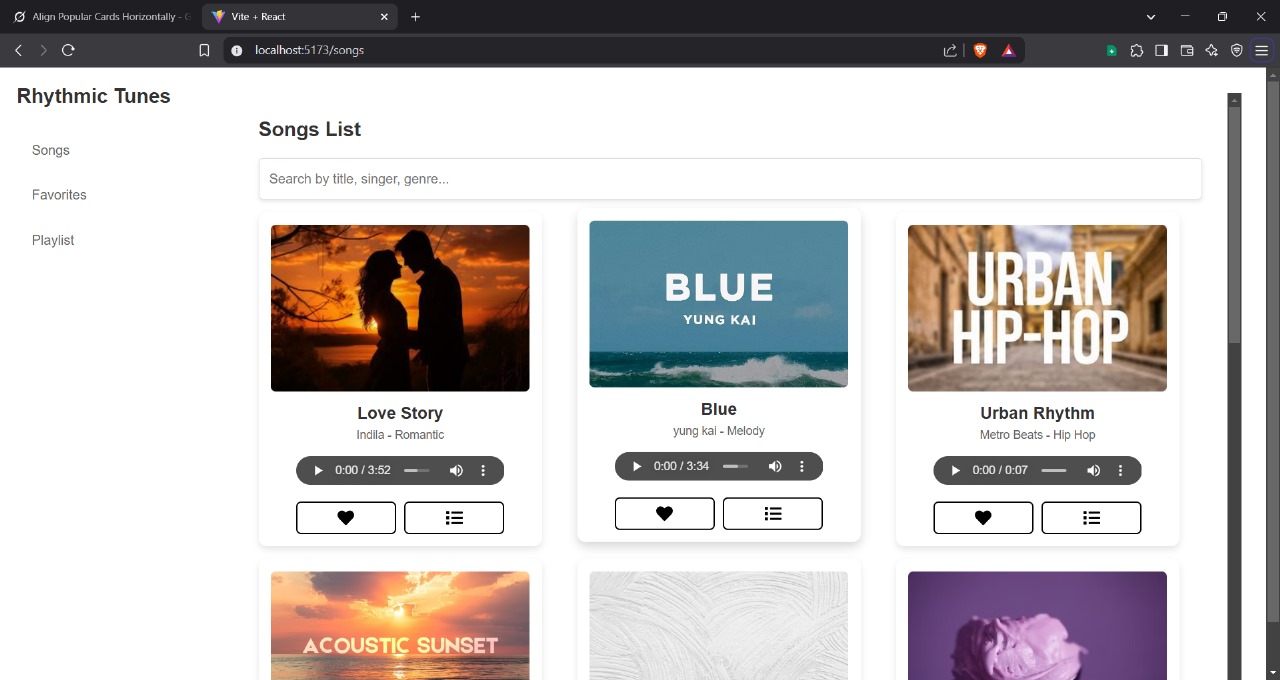
});

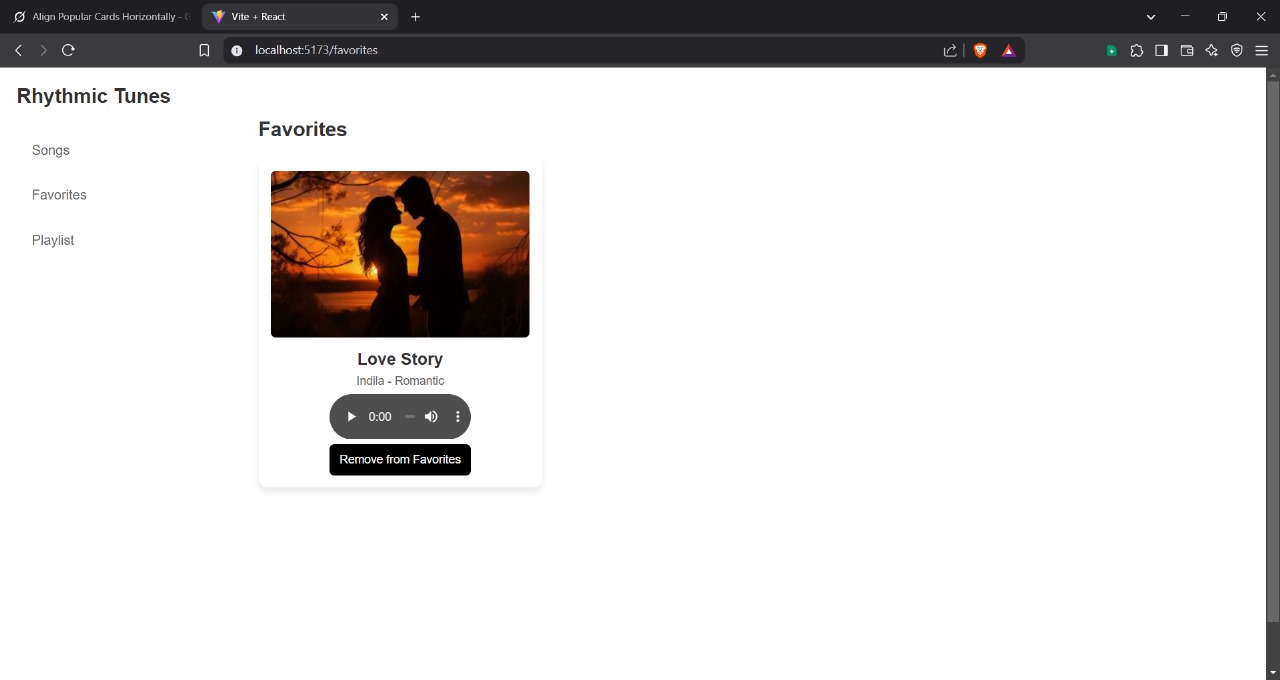
});

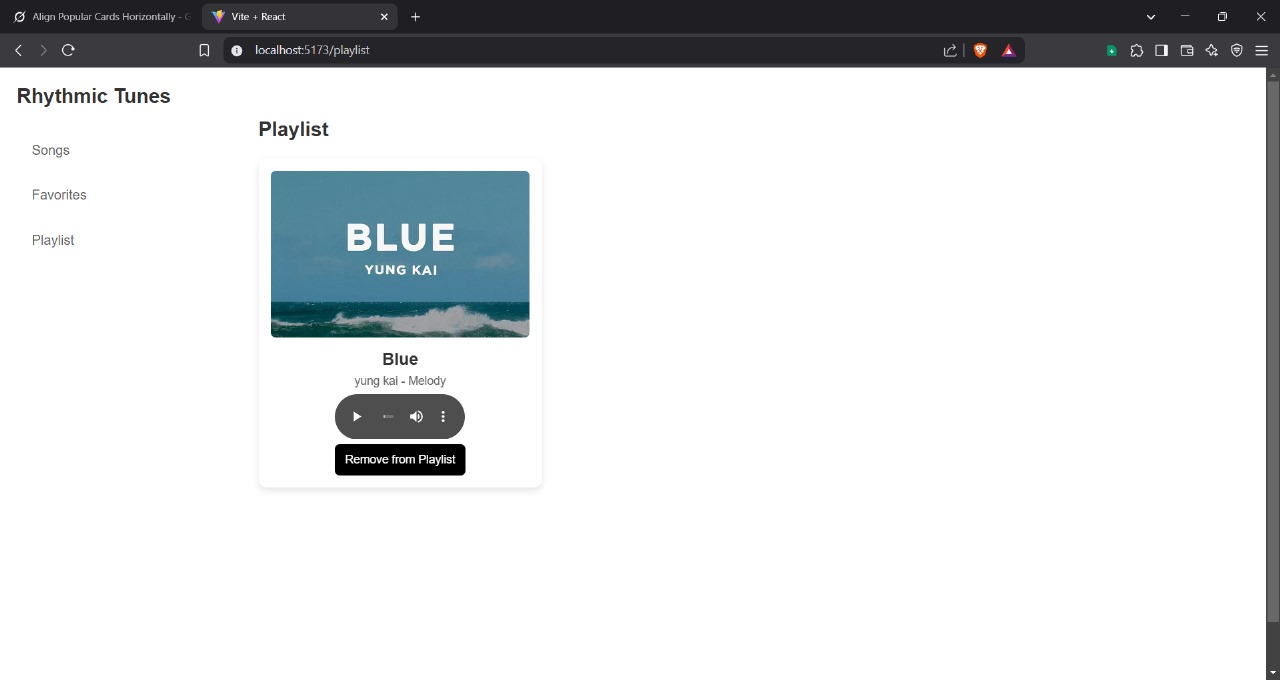
### ****Code Coverage:****

* Jest's built-in coverage tool is used to track tested components and functions.
* Automated tests are integrated into the **CI/CD pipeline** to catch issues early and ensure stability before deployment.
* End-to-end testing using Cypress.

## 12. Screenshots







**DEMO**

Demo video forRythimic Tunes

https://drive.google.com/file/d/15ESOYoaB7QmM22VROMJspv8LwU0FV8L-/view?usp=sharing

## 13. Known Issues

Despite thorough development and testing, Rythimic Tunes has a few known issues that are being actively worked on for future improvements:

* **Playback Optimization:** Some users may experience minor delays or buffering during song playback, especially on slower networks.
* **Offline Listening:** The offline mode requires enhancements to handle large music files more efficiently.
* **Search Performance:** When searching through extensive music libraries, occasional lag or delays may occur.
* **UI Responsiveness:** Some UI elements may need better optimization for smaller screens and devices.

## 14. Future Enhancements

Rythimic Tunes aims to continuously evolve by introducing new features and improvements. Some planned enhancements include:

* **AI-Powered Song Recommendations:** Implementing machine learning algorithms to suggest music based on user preferences and listening history.
* **Real-Time Lyrics Display:** Providing synchronized lyrics for a more immersive music experience.
* **Enhanced Offline Mode:** Improving the offline playback feature to support larger music libraries with better caching mechanisms.
* **UI & Animation Improvements:** Refining UI transitions and animations for a smoother and more engaging user experience.
* **Social Sharing Features:** Allowing users to share playlists and favorite songs directly with friends on social media.
* **Performance Optimization:** Further optimizing load times and search functionality for large-scale music libraries.