Rhythmic Tunes:Your Melodic

Companion

Team Members:

Yuvasree .M(212205311) - Coding

Uma Maheshwari .A(212205305) - Video

Pavithra .T(212205275) - Documentation

Divya .D(212205223) - Video

**Project overview:**

1. Purpose

Rhythmic Tunes is a dynamic and user-friendly web application designed to provide an immersive music streaming experience. The goal of the project is to allow users to discover, listen to, and interact with their favorite music. The application aims to streamline the process of music playback, create personalized playlists, and deliver a visually appealing and responsive interface. With an intuitive design and powerful features, Rhythmic Tunes enhances the overall music experience for users across various devices.

2. Features

* The frontend of Rhythmic Tunes offers several key features to enhance the user experience:
* Music Player: A sleek and interactive music player that allows users to play, pause, skip tracks, and adjust volume.
* Track Listings: Browse and search through a library of tracks, with detailed information like song title, artist, album, and duration.
* Playlists: Create, edit, and manage custom playlists to organize favorite tracks.
* Search Functionality: Easily search for songs, artists, or albums and quickly find content.
* User Authentication: Secure login and user account management, enabling users to save their preferences and playlists.Track Previews: Listen to previews of songs before adding them to playlists or starting playback.
* Responsive Design: The app is fully responsive, ensuring a seamless experience across desktops, tablets, and smartphones.
* Track Information: View detailed information about tracks including artist, album cover, and more while playing songs.

**Architecture:**

component structure of Rhythmic Tunes:

Rhythmic Tunes Component Structure

1. \*Header\*: Song title, artist, album art

2. \*Audio Player\*:

1. Play/Pause button

2. Progress bar

3. Volume control

3. \*Song Details\*:

1. Artist name

2. Album name

3. Release year

4. \*Controls\*:

1. Previous/Next buttons

2. Repeat/Shuffle buttons

5. \*Footer

1. Song duration

2. Current time

**State management of Rhythmic Tunes:**

Rhythmic Tunes State Management

States:

1. \*Song List\*: Array of songs with metadata (title, artist, album)

2. \*Current Song\*: Currently playing song's index and metadata

3. \*Playback Status\*: Playing/Paused state

4. \*Progress\*: Current playback position (time)

5. \*Volume\*: Current volume level

Actions:

1. \*Play/Pause\*: Toggle playback status

2. \*Next/Previous\*: Update current song index

3. \*Seek\*: Update progress (time)

4. \*Volume Change\*: Update volume level

State Updates:

1. Update song list when new songs are added/removed

2. Update current song when user selects a new song

3. Update playback status and progress when user interacts with playback controls

**Rhythmic Tunes Architecture Routing**

Routes:

1. \*Home\*: `/` (display featured songs/playlists)

2. \*Song List\*: `/songs` (display all songs)

3. \*Playlist\*: `/playlist/:id` (display specific playlist)

4. \*Song Details\*: `/song/:id` (display specific song details)

5. \*Search\*: `/search` (display search results)

Routing Mechanism:

1. \*Client-side Routing\*: Using React Router or similar library

2. \*Server-side Routing\*: Using Node.js/Express or similar framework

Navigation:

1. \*Header Navigation\*: Links to main routes (Home, Song List, etc.)

2. \*Breadcrumb Navigation\*: Displays current route hierarchy

**Rhythmic Tunes Setup Instructions**:

Ensure you have the following installed:

Node.js (latest stable version)

npm or yarn (package manager)

Git (for cloning the repository)

Installation Steps

Clone the Repository

bash

Copy code

git clone https://github.com/your-repo/rhythmic-tunes.git

cd rhythmic-tunes

Install Dependencies

bash

Copy code

npm install

# or

yarn install

Configure Environment Variables

Create a .env file in the project root.

Add required environment variables based on .env.example.

Run the Application

bash

Copy code

npm start

# or

yarn start

**Folder Structure:**

**Client (React Application)**

The React frontend is organized as follows:

* **src/** – Main source folder
  + **components/** – Reusable UI components (e.g., buttons, modals, music player)
  + **pages/** – Main application views (e.g., Home, Playlist, Settings)
  + **assets/** – Static files like images, icons, and fonts
  + **context/** – Global state management using React Context API
  + **routes/** – Application routing configuration
  + **styles/** – Global styles and theme settings

**2. Utilities**

* **utils/** – Helper functions for formatting, API calls, and common logic
* **hooks/** – Custom React hooks (e.g., useAuth, useFetch) for reusable logic
* **constants/** – Stores static values such as API endpoints and theme settings

**Running the Rhythmic Tunes Application:**

To start the frontend server locally, follow these steps:

Navigate to the Client Directory

bash

Copy code

cd client

Start the Frontend Server

bash

Copy code

npm start

This will launch the React application on http://localhost:3000/ by default.

Rhythmic Tunes Component Documentation

1. Key Components

These are the main components used in the application:

* Player – Handles music playback, controls (play, pause, skip), and volume adjustments.
* Props: track (object), isPlaying (boolean), onPlayPause (function)
* TrackList – Displays a list of available tracks.
* Props: tracks (array), onSelectTrack (function)
* Navbar – Navigation bar with links to different sections of the app.
* Props: None
* 2. Reusable Components
* Button – A customizable button for various UI actions.
* Props: label (string), onClick (function), variant (string)
* Modal – Displays pop-up messages or forms.
* Props: isOpen (boolean), onClose (function), children (ReactNode)
* Loader – Shows a loading spinner while fetching data.
* Props: size (string), color (string)

**Rhythmic Tunes State Management:**

1. Global State

Managed using React Context API or Redux (depending on the project setup).

Stores and shares application-wide data such as:

User Authentication (e.g., login state)

Current Track Information (e.g., playing song, queue)

Theme & Settings (e.g., dark mode, volume level)

State flows through context providers or Redux store, allowing components to access and update data efficiently.

2. Local State

Managed using React's useState hook within individual components.

Used for component-specific data like:

UI interactions (e.g., modals, dropdowns, form inputs)

Playback controls (e.g., play/pause button state)

User inputs (e.g., search bar text)

Ensures components remain independent and performant.

**Rhythmic Tunes User Interface:**

* The User Interface (UI) of Rhythmic Tunes is designed for a smooth and engaging music experience. It includes:
* Homepage: Displays featured tracks, playlists, and recommendations.
* Music Player: Includes play, pause, skip, and volume controls.
* Track List Page: Shows all available songs with search and filter options.
* User Profile: Allows users to manage preferences and settings.
* Dark/Light Mode: Toggle between themes for better accessibility.

**Rhythmic Tunes Styling:**

1. CSS Frameworks/Libraries

Uses Tailwind CSS for utility-first styling and rapid UI development. Additional styles may be implemented with Styled-Components for dynamic styling within React components. Supports CSS Modules for scoped styles in specific components.

2. Theming

Implements Dark/Light Mode using CSS variables or context-based theming. Custom design system ensures consistent typography, colors, and spacing. Theme settings are stored in global state, allowing users to toggle themes dynamically.

**Rhythmic Tunes Testing:**

1. Testing Strategy

Unit Testing: Uses Jest and React Testing Library to test individual components and functions.

Integration Testing: Ensures different components interact correctly using React Testing Library. End-to-End (E2E) Testing: Uses Cypress or Playwright to test user flows, such as playing a track or navigating pages.

2. Code Coverage

Jest Coverage Reports: Measures how much of the code is tested.

Coverage Thresholds: Ensures key functionalities (e.g., playback, authentication) have sufficient test coverage. CI Integration: Automates tests with GitHub Actions or similar CI/CD tools.

**Rhythmic Tunes Screenshots or Demo:**

To showcase the application's features and design, you can:

Provide Screenshots: Include images of key UI elements such as the homepage, music player, track list, and settings page.

Share a Demo Link: Host a live version of the app using Vercel, Netlify, or GitHub Pages and provide the link.

Create a Video/GIF: Demonstrate interactions like playing music, navigating pages, or switching themes.

**Rhythmic Tunes Known Issues:**

Below are some known bugs or issues that users and developers should be aware of:

* Playback Delay: Slight lag when switching between tracks, especially on slower networks.
* Mobile Responsiveness: Some UI elements may not be fully optimized for smaller screens.
* Dark Mode Glitch: Theme toggle may not persist after page refresh.
* Authentication Timeout: Users may be logged out unexpectedly due to session expiration.
* Cross-Browser Compatibility: Minor styling inconsistencies in certain browsers.

**Rhythmic Tunes Future Enhancements:**

Here are some planned improvements and new features:

* Enhanced Animations – Smooth transitions and interactive UI elements for a better user experience.
* Offline Mode – Allow users to download and play tracks without an internet connection.
* AI-Powered Recommendations – Personalized song suggestions based on listening history.
* Playlist Sharing – Users can create and share playlists with friends.
* Improved Mobile Responsiveness – Better UI optimization for small screens.
* Voice Commands – Enable voice controls for hands-free navigation