Rhythmic Tunes: Your Melodic Companion

|  |  |  |
| --- | --- | --- |
| **TEAM** | **NAME** | **EMAIL** |
| **Team leader** | **Manasa devi N** | **ncas2225sz1028@ncas.in** |
| **Team member1** | **Angel I** | **ncas2225sz1058@ncas** |
| **Team member2** | **Deepika V** | **ncas2225sz1432@ncas.in** |
| **Team member3** | **Devi C** | **ncas2225sz1180@ncas.in** |
| **Team member4** | **Gayathri M** | **ncas2225sz1695@ncas.in** |

**Team ID SWTID1741324396155224**

**Introduction:**

Welcome to Rhythmic Tunes: Your Melodic Companion, a revolutionary music streaming service designed to harmonize your life with the perfect melody. Imagine a world where music seamlessly adapts to your emotions, activities, and surroundings. Rhythmic Tunes makes this vision a reality. Problem Statement: Traditional music streaming services often rely on static playlists and algorithms that fail to capture the nuances of human emotion. This can lead to a disjointed listening experience, where the music doesn't quite match the mood or activity. Solution: Rhythmic Tunes addresses this challenge by incorporating advanced AI-powered algorithms that learn your listening habits, preferences, and emotional state. Our platform creates personalized playlists that dynamically adapt to your needs, ensuring a melodic companion that's always in tune with your life.

Team members role:

**Manasa devi N** - Oversees development, Ensures deadlines are met, Manages collaboration, Designs and implements UI, Works with CSS frameworks (Tailwind, Material UI, etc.) Manages global state (Redux API, Context API), Handles API calls and data flow

**Angel I** - Frontend Developer (Component Development), builds reusable components, Implements dynamic UI interactions, Conducts audits using Lighthouse & Webpage Test, Optimizes React performance (lazy loading, memorization)

**Deepika V**- Writes unit tests (Jest, React Testing Library), Performs manual testing Accessibility & Performance Expert (Secondary Role for UI Developer), Ensures UI meets WCAG accessibility standards

**Devi C**– DevOps & Deployment Specialist (Secondary Role for a Developer), Manages CI/CD pipelines (GitHub Actions, Vercel, Netlify), Handles hosting & deployment, Ensures automated testing is integrated

**Gayathri M**- Documentation & Knowledge Manager (Good for a Team Member Who Likes Writing & Organizing), Maintains project documentation. Creates a Wiki or Notion page for the team. Ensures team members follow coding conventions

**Project overview**

Purpose:

Rhythmic Tunes is a music streaming service designed to provide users with a personalized music experience. The platform aims to create a melodic companion that adapts to the user's emotions, activities, and preferences.

**Features:**

1. Mood-based Playlists: Rhythmic Tunes creates playlists based on the user's current mood, activity, or atmosphere.

2. Personalized Recommendations: The platform uses AI-powered algorithms to suggest songs, artists, and playlists tailored to the user's listening habits.

3. Music Discovery: Rhythmic Tunes features a discovery mode that allows users to explore new genres, artists, and songs.

4. Social Sharing: Users can share their favorite songs, playlists, and music moments on social media.

5. Multi-Device Support: Rhythmic Tunes is accessible on various devices, including smartphones, tablets, smart TVs, and gaming consoles.

**Architecture**

**Overview**:

Rhythmic Tunes is a music streaming service that utilizes AI-powered algorithms to provide personalized music recommendations. The architecture will consist of the following components:

**Component Structure:**

**1. App Component:**

Purpose: The top-level component that renders the entire application.

Children: Header, Main, Footer

State: None

Props: None

**2. Header Component:**

Purpose: Renders the application header with navigation links.

Children: Nav, Logo

State: None

Props: None

**3. Main Component:**

Purpose: Renders the main content area of the application.

Children: MusicPlayer, Playlist, Favorites

State: currentSong, playlist, favorites

Props: None

**4. Music Player Component:**

Purpose: Renders the music player with playback controls.

Children: PlayPauseButton, NextButton, PreviousButton

State: currentSong, playbackStatus

Props: currentSong, playbackStatus

**5. Playlist Component:**

Purpose: Renders the playlist with song items.

Children: SongItem

State: playlist

Props: playlist

**6. Favorites Component:**

Purpose: Renders the favorites list with song items.

Children: SongItem

State: favorites

Props: favorites

**State Management:**

RhythmicTunes uses Redux for state management.

**1. Store:**

The store is created using the createStore function from Redux.

**2. Reducers:**

RhythmicTunes has three reducers:

musicReducer: Handles music-related state changes (e.g., current song, playlist).

favoritesReducer: Handles favorites-related state changes (e.g., favorites list).

playlistReducer: Handles playlist-related state changes (e.g., playlist items).

**3. Actions:**

Rhythmic Tunes has several actions:

Play Song: Plays a song.

Pause Song: Pauses a song.

Next Song: Plays the next song in the playlist.

Previous Song: Plays the previous song in the playlist.

Add To Favorites: Adds a song to the favorites list.

Remove From Favorites: Removes a song from the favorites list.

**Routing:**

Rhythmic Tunes uses React Router for client-side routing.

**1. Routes:**

Rhythmic Tunes has several routes:

/: Renders the main content area with the music player and playlist.

/favorites: Renders the favorites list.

/playlist: Renders the playlist.

**2. Navigation:**

Rhythmic Tunes uses the Link component from React Router to enable navigation between routes.

**3. Route Protection:**

Rhythmic Tunes uses route protection to restrict access to certain routes based on user authentication status.

**Testing Plan**

**Unit Testing**:

**1. User Service:**

Test user registration and login functionality.

Verify user data is stored correctly in the database.

**2. Music Service:**

Test music metadata retrieval and storage.

Verify music recommendation algorithms.

**3. Search Service:**

Test search functionality for music, artists, and playlists.

**Integration Testing:**

**1. User-Music Interaction:**

Test user playback, pause, and skip functionality.

Verify music playback history is updated correctly.

**2. Search-Music Integration:**

Test search results for music, artists, and playlists.

Verify search results are relevant and accurate.

**3. User-Playlist Interaction:**

Test user playlist creation, editing, and deletion.

Verify playlist changes are reflected correctly.

**Code coverage**

**Source Code:**

**public/index.html:**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="UTF-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1.0">**

**<title>RhythmicTunes</title>**

**<link rel="stylesheet" href="styles.css">**

**</head>**

**<body>**

**<div id="root"></div>**

**<script src="bundle.js"></script>**

**</body>**

**</html>**

**UI Testing:**

**1. User Interface:**

Test UI components, such as buttons, forms, and navigation.

Verify UI is responsive and works correctly on different devices.

**2. User Experience:**

Test user workflow, such as searching, playing, and creating playlists.

Verify user experience is smooth and intuitive.

**Integration Testing:**

**API Integration:**

**1. Music API:**

Test integration with music streaming APIs.

Verify music metadata and streaming functionality.

**2. Social Media API:**

Test integration with social media platforms.

Verify social sharing and login functionality.

**Database Integration:**

**1. User Data:**

Test user data storage and retrieval.

Verify user data is consistent across the application.

**2. Music Data:**

Test music metadata storage and retrieval.

Verify music data is consistent across the application.

**Deployment Integration:**

**1. Cloud Deployment:**

Test deployment on cloud providers.

Verify application scalability and reliability.

**2. Containerization:**

Test containerization using Docker.

Verify application consistency across environments.

**3. Tools and Frameworks:**

1. Jest: For unit testing and integration testing.

2. Cypress: For UI testing and end-to-end testing.

3. Postman: For API testing and integration testing.

4. Docker: For containerization and deployment.

**4. Testing Schedule:**

1. Unit Testing: 2 weeks

2. Integration Testing: 3 weeks

3. UI Testing: 2 weeks

4. Deployment Testing: 1 week

**Setup Instructions:**

Prerequisites:

1**. Node.js**: Install Node.js (version 14 or higher) from the official website: https://nodejs.org/en/download/

2. **npm: Install npm** (version 6 or higher) from the official website: https://www.npmjs.com/get-npm

3. **Git:** Install Git from the official website: https://git-scm.com/downloads

4. **Code Editor**: Install a code editor of your choice (e.g., Visual Studio Code, Sublime Text, Atom)

**Installation:**

1. **Clone the Repository**: Run the following command in your terminal to clone the Rhythmic Tunes repository:

git clone https://github.com/your-username/RhythmicTunes.git

2. **Navigate to the Project** Directory: Run the following command to navigate to the project directory:

cd Rhythmic Tunes

3. **Install Dependencies**: Run the following command to install the project dependencies:

npm install

4. **Start the Development** Server: Run the following command to start the development server:

npm start

5**. Access the Application**: Open your web browser and navigate to http://localhost:5173 to access the Rhythmic Tunes application.

**Folder Structure:**

**Rhythmic Tunes/**

**├── public/**

**│ ├── index.html**

**│ └── favicon.ico**

**├── src/**

**│ ├── components/**

**│ │ ├── App.js**

**│ │ ├── Header.js**

**│ │ ├── Footer.js**

**│ │ ├── MusicPlayer.js**

**│ │ └── ...**

**│ ├── containers/**

**│ │ ├── AppContainer.js**

**│ │ └── ...**

**│ ├── actions/**

**│ │ ├── musicActions.js**

**│ │ └── ...**

**│ ├── reducers/**

**│ │ ├── musicReducer.js**

**│ │ └── ...**

**│ ├── utils/**

**│ │ ├── api.js**

**│ │ └── ...**

**│ ├── index.js**

**│ └── store.js**

**├── package.json**

**└── README.md**

**Client:**

**1. Music Player:**

**Description: A customizable music player that allows users to play, pause, and skip songs.**

**Features:**

**Playback controls (play, pause, skip)**

**Song progress bar**

**Volume control**

**Repeat and shuffle options**

**2. Playlist Manager:**

**Description: A utility that allows users to create, edit, and manage playlists.**

**Features:**

**Create new playlists**

**Add songs to playlists**

**Remove songs from playlists**

**Edit playlist names and descriptions**

**Utilities:**

**1. Music API:**

**- Description: A RESTful API that provides access to music metadata and streaming functionality.**

**- Endpoints:**

**- GET /songs: Retrieves a list of songs**

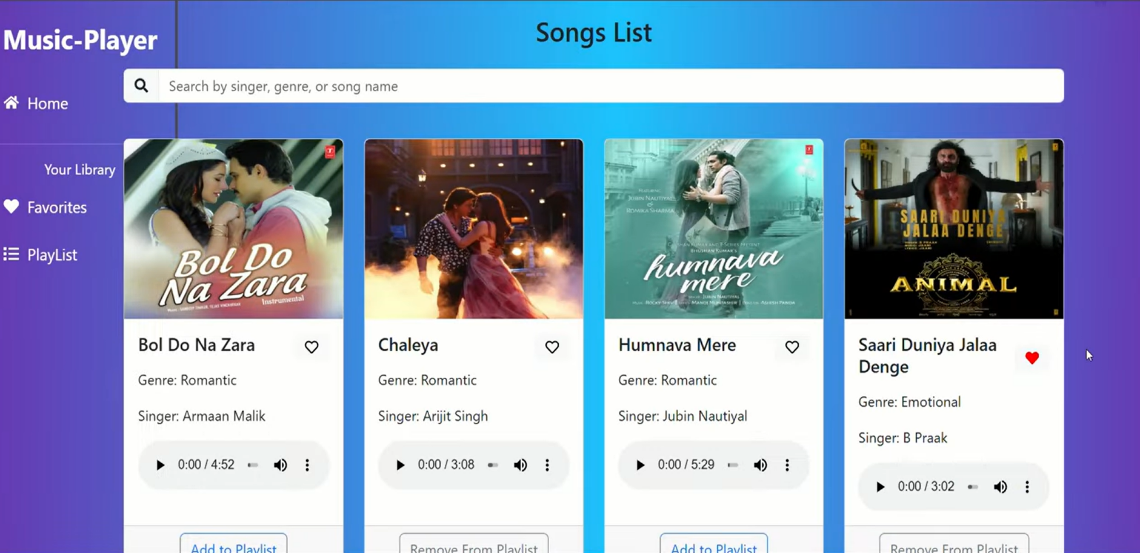
**- GET /songs/:id: Retrieves a specific song by ID**

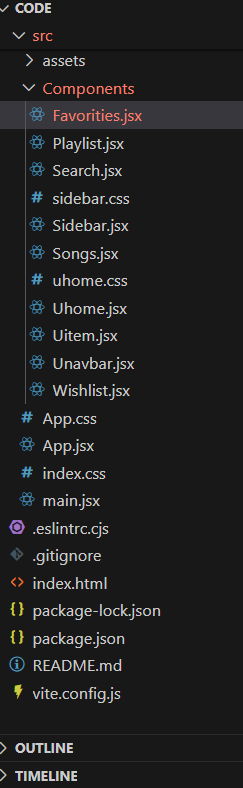
**- POST /songs: Creates a new song**

**- PUT /songs/:id: Updates a specific song by ID**

**- DELETE /songs/:id: Deletes a specific song by ID**

**Screenshots or Demo**





**Running the Application:**

1. npm start: Run the command npm start to start the development server.

2. npm run build: Run the command npm run build to build the production-ready application.

3. npm run deploy: Run the command npm run deploy to deploy the application to a hosting platform.

**Frontend Technologies:**

1. React: A JavaScript library for building user interfaces.

2. Redux: A state management library for managing global state.

3. React Router: A library for managing client-side routing.

4. Sass: A CSS preprocessor for writing more efficient and modular CSS code.

5. Webpack: A module bundler for bundling and optimizing code.

**Component Documentation:**

**Key Components**:

1. Music Player: A customizable music player that allows users to play, pause, and skip songs.

2. Playlist Manager: A utility that allows users to create, edit, and manage playlists.

3. Music Library: A centralized library that stores all the user's music.

4. Music Recommender: A utility that provides personalized music recommendations based on user listening history and preferences.

5. Navigation: A navigation component that allows users to navigate between different sections of the application.

**Reusable Components**:

1. Song Card: A reusable component that displays information about a song, such as the title, artist, and album art.

2. Playlist Card: A reusable component that displays information about a playlist, such as the name, description, and number of songs.

3. Button: A reusable button component that can be used throughout the application.

4. Input Field: A reusable input field component that can be used throughout the application.

**State Management:**

**Global State**:

1. User Data: The user's data, such as their username, email, and password.

2. Music Library: The user's music library, including all the songs and playlists.

3. Current Song: The currently playing song.

4. Current Playlist: The currently selected playlist.

**Local State:**

1. Music Player State: The state of the music player, including the current song, playback status, and volume.

2. Playlist Manager State: The state of the playlist manager, including the current playlist and any changes made to it.

3. Music Library State: The state of the music library, including any changes made to the songs or playlists..

**Styling:**

1. CSS Preprocessor: Sass (SCSS) is used as the CSS preprocessor for writing more efficient and modular CSS code.

2. CSS Framework: Bootstrap is used as the CSS framework for providing a responsive and consistent layout.

3. CSS Library: Font Awesome is used as the CSS library for providing icons and graphics.

**CSS Frameworks/Libraries:**

1. Bootstrap: A popular CSS framework for building responsive and mobile-first websites.

2. Font Awesome: A popular CSS library for providing icons and graphics.

3. Animate.css: A CSS library for providing animations and transitions.

**Theming**:

1. Color Scheme: A dark theme with a primary color of #333 and a secondary color of #666.

2. Typography: Open Sans is used as the primary font, with a font size of 16px for body text and 24px for headings.

3. Layout: A responsive layout with a maximum width of 1200px and a padding of 20px.

**Theme Switching**:

1. Light Theme: A light theme with a primary color of #FFF and a secondary color of #CCC.

2. Dark Theme: A dark theme with a primary color of #333 and a secondary color of #666.

3. Theme Switcher: A theme switcher component that allows users to switch between light and dark themes.

**Known Issues:**

Here are some known issues with the application:

The music player may not work correctly in older browsers.

The search functionality may not return accurate results.

The application may not be fully accessible for users with disabilities.

**Future Enhancements:**

Here are some future enhancements planned for the application:

Implementing a recommendation algorithm to suggest music based on user listening habits.

Adding support for multiple music streaming services.

Improving the application's accessibility features ,Adding support for offline playback.