Frontend Development with React.js

Project Documentation format

1. **Introduction :**
   - ◆ **Project Title**: Rythmic Tunes(React)
   - ◆ **Team leader:** S.Kiruthika
   - ◆ **Team member:**

     1.C.Jaya bharathi
     2.M.aishwarya
     3. P.jaya priya

2 . Project Overview:

- ◆ **Purpose**: The Rhythmic Tunes application is designed to create and play rhythmic beats and tunes. It allows users to interact with audio files, generating custom rhythms, and offering features like play/pause, loop, and volume control. The project aims to provide an engaging music experience with simple, intuitive controls.

- ◆ **Features :**

- **Audio Playback**: Users can play or pause rhythmic tunes.
- **Loop Functionality**: Tunes can be set to loop, providing continuous playback.
- **Volume Control**: Adjust the volume of the audio playback.
- **Custom Rhythms**: Users can modify beats and rhythm through real-time adjustments.
- **Responsive Design: The app is fully responsive and works across multiple device types.**

2. **Architecture**

- **Component Structure :**

- **App**: The main component that wraps all other components and manages global state.
- **RhythmicTune**: Manages audio playback, including controls like play/pause, loop, and volume.
- **AudioControls**: A component for managing audio features such as the play/pause button, volume slider, and loop toggle.
- **Header**: Displays the title and basic app instructions.
- **Footer**: Contains information about the app and external links (e.g., GitHub repository).

- **State Management:**

- **Local State**: State is primarily managed within each component using React's `useState` hook (e.g., `isPlaying`, `volumeLevel`).
- **Global State**: The project uses the **Context API** to manage global settings like the current track, whether it's looping, and app-wide volume adjustments.

- **Routing**: **react-router-dom** is used for routing. The app has a main page for rhythm controls and an optional settings page to adjust preferences (e.g., sound quality, advanced controls).

- **Setup Instructions :**
- **Prerequisites**: **Node.js**: Ensure Node.js is installed (version 14 or above).
- **npm**: Node's package manager for managing dependencies.

  - **Installation**: Clone the repository:

```
blash
Copy
git clone https://github.com/your-repository/rhythmic-tunes.git
```

Navigate to the project directory:

```
bash
Copy
cd rhythmic-tunes
```

Install dependencies:

```
bash
Copy
npm install
```

(Optional) Configure environment variables in a `.env` file, if needed, for custom audio settings or API keys.

- **Folder Structure**

Client:Client:

```
/src
  /components
    Header.js          // Displays app title and instructions
    Footer.js          // Contains information about the app
    RhythmicTune.js    // Main audio control component
    AudioControls.js   // Handles play/pause, loop, and volume
  /pages
    Home.js            // The main page with audio controls
    Settings.js        // Optional settings page for advanced options
  /assets
    /audio
      tune.mp3         // Default audio file
    /images
      logo.png         // Logo of the app
  /context
    AppContext.js      // Global state management
  /hooks
    useAudio.js        // Custom hook to manage audio state
```

  - **Utilities**:**useAudio.js**: Custom React hook for managing audio playback, including functions like `play()`, `pause()`, and `setVolume()`.

- **helpers.js**: Utility functions for setting volume or calculating rhythm patterns.

- **Running the Application**

To start the frontend server locally:
Navigate to the project directory (if not already there):

```
bash
Copy
cd rhythmic-tunes
```

Run the following command to start the development server:

```
  bash
Copy
npm start
```

Open your browser and go to `http://localhost:3000` to view the application.

- **Component Documentation**
- **Key Components**: **RhythmicTune**: This component is responsible for the audio playback, which includes buttons to play/pause, toggle looping, and adjust volume. Props:
- `audioFile` (string): Path to the audio file to play.
- `isLooping` (boolean): Whether the audio is set to loop.

- **AudioControls**: This component contains the audio controls, such as volume sliders and play/pause buttons. Props:

- `onPlayPause` (function): Callback to trigger play/pause.
- `onVolumeChange` (function): Callback to adjust the volume.

  .
  - **Reusable Components**: **Button**: A button component that can be reused across the app to trigger actions like play, pause, and toggle settings. Props:
- `label` (string): The text displayed on the button.
- `onClick` (function): The function to execute when the button is clicked.

  - **State Management**
- **Global State**: The global state is managed using the **Context API**:
- **AppContext** stores the global state of the app, such as whether the current track is playing or paused, the volume level, and loop settings. This context is provided to all components in the app.

- **Local State**: Each component uses **local state** to handle its specific UI interactions, such as the state of the play/pause button or individual volume control sliders.

- **User Interface**

- **UI Features:**

- **Home Page**: Displays the rhythm controls, including buttons for play/pause, loop toggle, and volume slider.
- **Settings Page**: Allows users to adjust more advanced settings, such as selecting different audio files or adjusting sound quality.

- **Styling**

- **CSS Frameworks/Libraries**: **Styled-Components**: Used to style the React components with CSS-in-JS.
- **FontAwesome**: For icons like play, pause, and volume controls.
- **Theming**: The application includes a light and dark theme, which can be toggled through the settings. Custom CSS variables are used to define color schemes, and the `styled-components` library dynamically switches between themes.

- **Testing**

- **Testing Strategy**: **Unit Testing**: Components like `Button`, `AudioControls`, and `RhythmicTune` are unit tested using **Jest** and **React Testing Library**.
- **Integration Testing**: Tests ensure that components interact as expected (e.g., pressing play updates the audio state).
- **End-to-End Testing**: **Cypress** is used to simulate user interactions and verify end-to-end functionality.
- **Code Coverage**: The project uses **Jest** and **React Testing Library** to ensure high test coverage. Coverage reports are generated during the testing process to track untested components.
- **Screenshots or Demo link:https://drive.google.com/file/d/1l5Xsf3PSeLEE6QlNgJYpqj3516aHhex3/view?usp=drivesdk**

- **Known Issues**

- **Audio delay**: On certain browsers, there may be a slight delay when playing the tune for the first time.
- **Mobile UI**: The app might not be fully optimized for smaller screens.

- **Future Enhancements**

- **Audio Effects**: Add more complex audio effects, such as reverb, delay, and pitch adjustments.
- **Advanced Rhythm Generator**: Implement a system where users can create custom rhythm patterns.

- **Mobile Optimization**: Improve responsiveness on mobile devices, including better touch controls for volume and playback.