**Frontend Development with React.js**

**Music App**

**1. Introduction**

* **Project Title**: Rythimic Tunes  
  A web-based application allowing users to stream music online without downloading it, offering a user-friendly interface with features like personalized playlists, music search, and music recommendations.
* **Team Leader**: *SRIDHARSHAN S*  
  The team leader responsible for overseeing the project, managing the team, and ensuring milestones are met.
* **Team Members**:
  + **SANTHOSH P**  
    Works on frontend development and UI design.
  + **PARTHIBASANKAR S**  
    Handles backend integration and API development.
  + **PREM KUMAR G**  
    Focuses on state management and optimizing app performance.
  + **KARTHIGA M**

Handles backend integration and API development.

**2. Project Overview**

* **Purpose**:  
  The purpose of this music streaming app is to provide users with an intuitive, easy-to-use platform for discovering, listening to, and managing music online. It aims to create a seamless and engaging music experience, accessible anytime, anywhere.
* **Features**:  
  Key features include:
  + **Music Playback**: Play, pause, skip, and loop music seamlessly.
  + **Personalized Playlists**: Users can create and save their playlists.
  + **Music Discovery**: Search by genre, artist, or album.
  + **Recommendations**: Suggested music based on user preferences and listening habits.
  + **User Accounts**: Authentication for saving preferences and managing playlists.
  + **Responsive Design**: Fully optimized for mobile, tablet, and desktop devices.

**3. Architecture**

* **Component Structure**:  
  The application is built using React, with components divided based on functionality:
  + **UI Components**: Buttons, sliders, and modals for playback controls.
  + **Container Components**: Handle data fetching and state management, such as music library and playlists.
  + **Presentational Components**: Display UI elements like music cards, album art, and song details.
* **State Management**:  
  The app uses **Context API** or **Redux** to manage global state, allowing the user’s preferences, playlists, and music data to be accessible throughout the app. State is shared between components for consistency across the app, such as when the user switches pages or interacts with playback controls.
* **Routing**:  
  The app uses **React Router** to manage navigation between pages such as the homepage, music player, and user profile. Routing helps create a single-page application (SPA) experience, allowing users to navigate between sections without reloading the page.

**4. Setup Instructions**

* **Prerequisites**:  
  Before setting up the application, make sure to install:
  + **Node.js** (LTS version recommended)
  + **npm** or **yarn** for dependency management.
* **Installation**:  
  To get started, follow these steps:

1. Clone the repository:  
   git clone https://github.com/Sridharshan2004/My-Project
2. Navigate into the project directory:  
   cd music-streaming-app
3. Install dependencies:  
   npm install (or yarn install)
4. Set up environment variables (e.g., API keys for third-party music APIs, if necessary).
5. Start the development server:  
   npm start (or yarn start).

**5. Folder Structure**

* **Client**:  
  The React application is organized into key folders:
  + src/: Contains all source code.
    - components/: Reusable UI components like Player, TrackCard, and SearchBar.
    - pages/: Page-level components like HomePage, LibraryPage, and ProfilePage.
    - assets/: Stores static files such as images, icons, and audio files.
    - services/: Functions that interact with external APIs or handle data fetching.
* **Utilities**:  
  Includes helper functions, custom hooks, or utility classes, such as:
  + **useAuth.js**: Custom hook for handling user authentication.
  + **useFetch.js**: Helper function for fetching data from external APIs.

**6. Running the Application**

* To run the frontend server locally, use the following command:
  + npm start (or yarn start) from the client/ directory.
* The app will be available at http://localhost:3000/ in your browser.

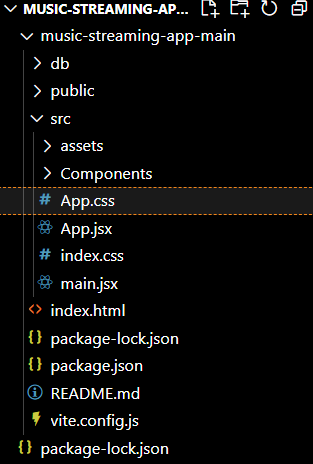
**7. Component Documentation**

* **Key Components**:  
  Here, we detail the important components of the app, for example:
  + **Player**: The main audio player that controls playback (play, pause, skip). Receives props like currentTrack and playbackStatus.
  + **TrackCard**: Displays individual song information such as album art, song title, and artist. Props include trackName, artistName, and albumArt.
* **Reusable Components**:
  + **Button**: A generic button component used throughout the app. It accepts props like onClick, text, and style.
  + **Loader**: A reusable loading spinner shown when the app is fetching data.

**8. State Management**

* **Global State**:  
  Uses **Redux** or **Context API** for managing global state, such as the list of songs, playlists, and user data. This allows multiple components to access and update shared data.
* **Local State**:  
  Each component handles its own local state, for example:
  + TrackCard manages whether the track is favorited or not.
  + SearchBar maintains the current search query entered by the user.

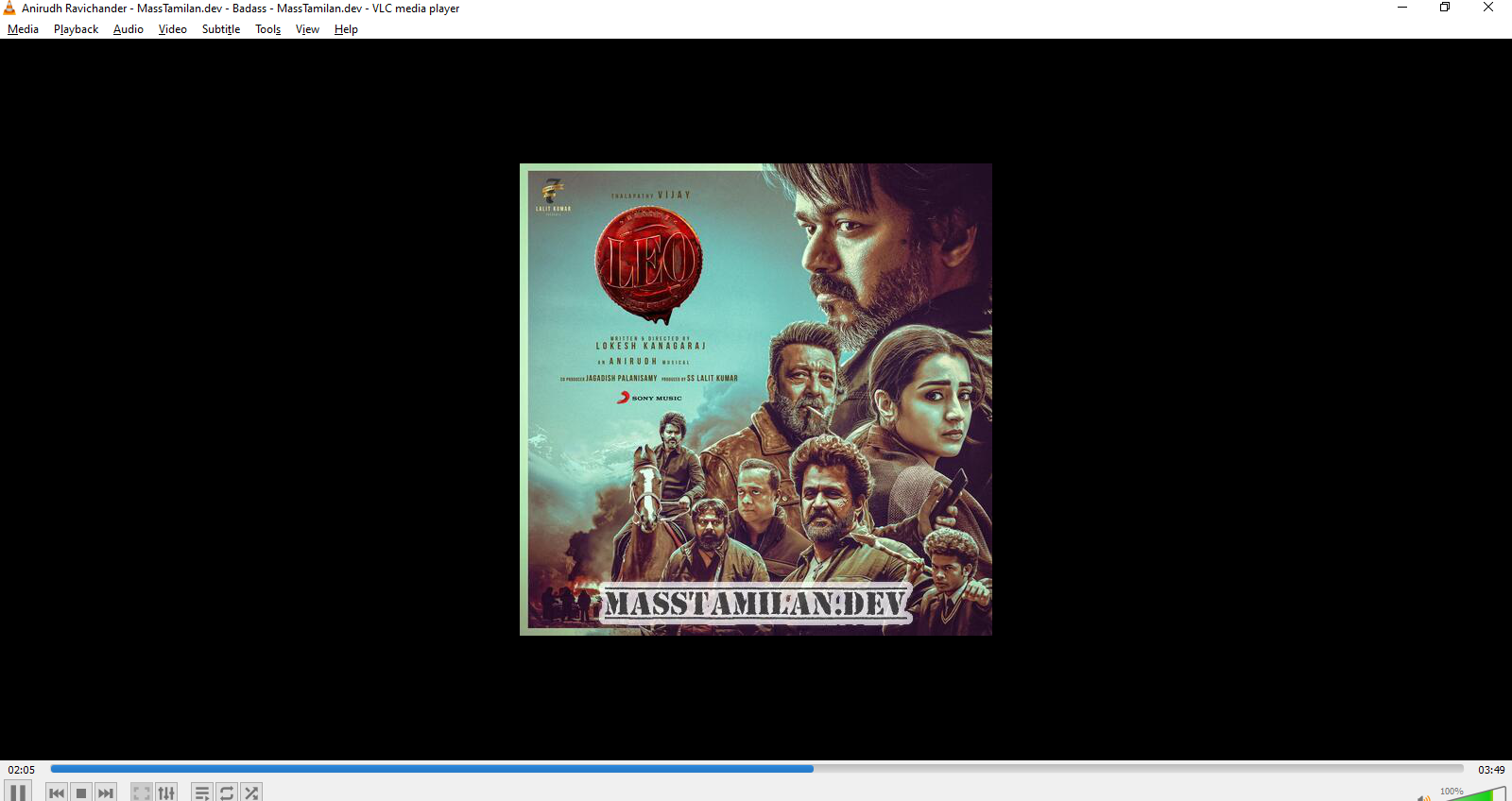
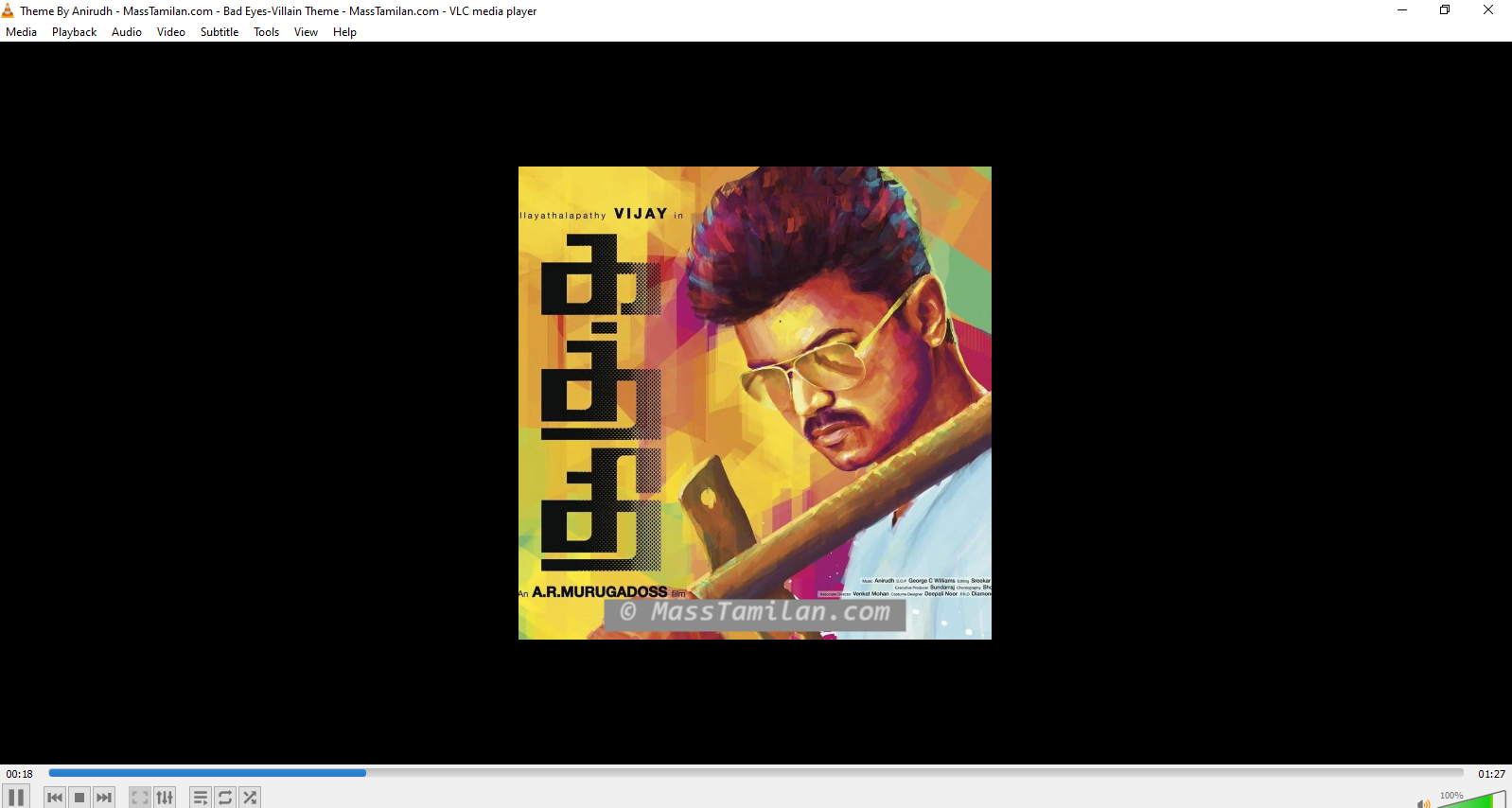
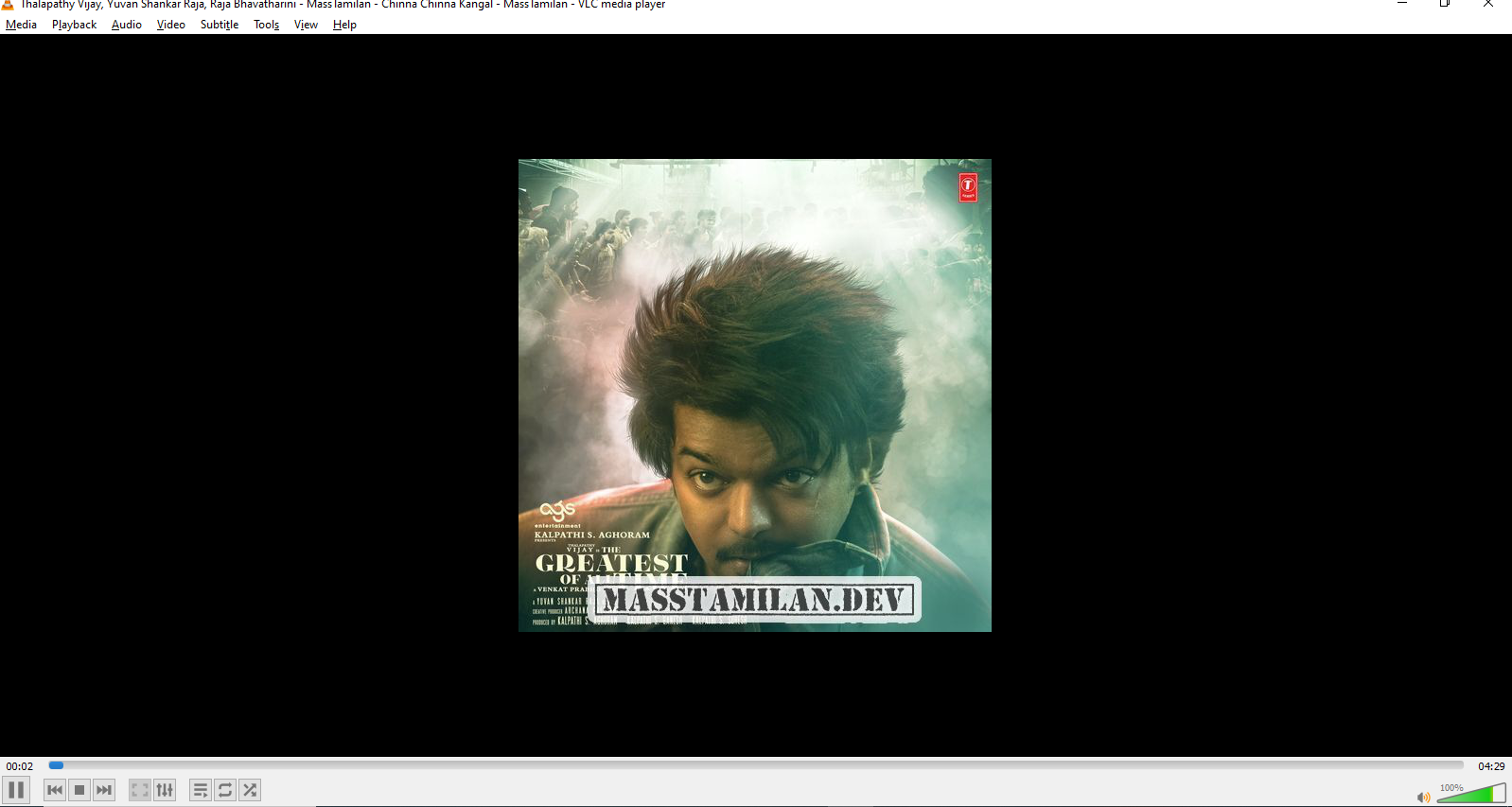
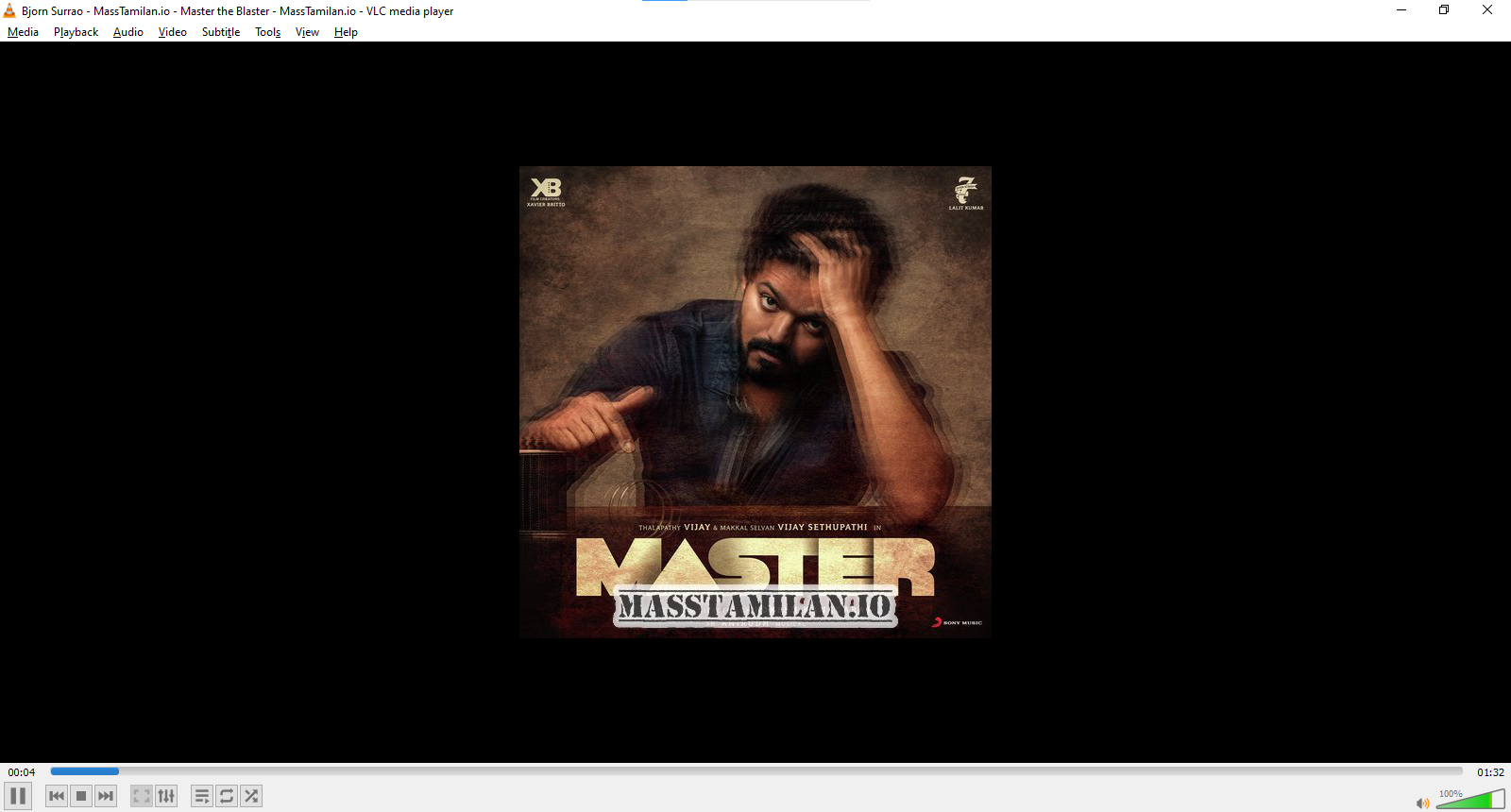
**9. User Interface**

* **Screenshots/GIFs**:

**10. Styling**

* **CSS Frameworks/Libraries**:  
  The project uses **Styled-Components** for component-level styling and **CSS modules** for scoped styles.
  + External libraries like **Bootstrap** or **Material-UI** may be used for ready-made UI components and layouts.
* **Theming**:  
  The app can implement theming with **Styled-Components** to allow users to switch between light and dark modes, or a custom design system for consistent styling across the application.

**11. Testing**

* **Testing Strategy**:  
  The application is tested using:
  + **Jest**: For unit testing React components and utility functions.
  + **React Testing Library**: To test component rendering and user interactions.
  + **Cypress**: For end-to-end testing, simulating full workflows like logging in, creating a playlist, and playing music.
* **Code Coverage**:  
  We use **Jest's** built-in coverage tool to ensure that tests cover key parts of the app and prevent regressions.
* **12. Screenshots or Demo**
* 
* 
* 
* 
* 
* **13. Known Issues**
* Document any issues or bugs that are currently known, such as:
  + **Playback Bug**: Sometimes the music skips or pauses unexpectedly.
  + **Search Lag**: There’s a slight delay when searching for songs.

**14. Future Enhancements**

* **Future Features**:
  + Integration with more music APIs (e.g., YouTube, Deezer).
  + Social sharing features allowing users to share playlists or songs on social media.
  + Offline mode to listen to downloaded songs.
* **Improvements**:
  + Enhancing the UI with animations for transitions between pages.
  + Adding voice command support to control music playback.