**Frontend Development with React.js**

**Project Documentation format**

1. **Introduction**
   * **Project Title**: Hank: music player frontend
   * **Team Members**:
     + Haifa, Jamal Naziya- Development
     + Noorul Ameera, Kavipriya - Documentation
     + Asma - video making
2. **Project Overview**
   * **Purpose**: Hank is a modern, personal music player application. It allows users to upload their own music files, organize them into playlists, and enjoy a seamless playback experience through a clean and intuitive user interface.
3. **Features**:

* **Local Music Upload**: Users can upload audio files from their device into their personal library.
* **Metadata Extraction**: Automatically reads metadata (title, artist, album, cover art) from audio files.
* **Playlist Management**: Users can create, view, and add songs to custom playlists.
* **Web Playback**: A full-featured player with controls for play/pause, next/previous track, volume adjustment, and a progress scrubber.
* **Dynamic UI**: A responsive interface with a home dashboard, library view, and detailed playlist views.
* **Component-Based UI**: Built with reusable components for a consistent and maintainable design.

1. **Architecture**

**Component Structure**: The application is built using Next.js and React, with a clear separation of concerns.

    - `AppShell`: The root component that manages the initial loading state and houses the main application layout.

    - `AppLayout`: Defines the primary three-column structure: a navigation sidebar on the left, the main content view in the center, and a "Now Playing" sidebar on the right, with a persistent player at the bottom.

    - `MusicProvider`: A React Context provider that wraps the entire application to manage global state.

    - `MainView`: Acts as a router, displaying the `HomeView`, the song library, or a selected playlist.

    - `PlayerControls`: The persistent music player component at the bottom of the screen.

**State Management**: Global application state is managed using the **React Context API**. The `MusicContext` (`src/context/music-context.tsx`) provides state and actions related to the song library, playlists, current track, and playback status to all components.

**Routing**: The application uses a client-side view management system controlled by the `currentView` state within `MusicContext`. It's not using a formal library like `react-router` but instead conditionally renders components like `HomeView` or a playlist view based on user interaction.

1. **Setup Instructions**

**Prerequisites**: You must have Node.js (v18 or newer) and npm installed on your machine.

**Installation**:

  1. Clone the repository to your local machine.

  2. Navigate to the project's root directory in your terminal.

  3. Install the required dependencies by running:

     ```bash

     npm install

     ```

1. **Folder Structure**

**src/app**: Contains the main entry point (`layout.tsx`, `page.tsx`) and global styles (`globals.css`) for the Next.js application.

**src/components**: Contains all React components.

**ui**: Reusable, low-level UI components provided by ShadCN (e.g., Button, Card, Dialog).

    - Other files in `components` are high-level components specific to this application (e.g., `AppLayout`, `PlayerControls`).

**src/context**: Includes the `MusicContext.tsx` file, which is responsible for global state management.

**src/hooks**: Contains custom React hooks, such as `useToast` for notifications.

**src/lib**: Contains utility functions (`utils.ts`) and other shared logic.

1. **Running the Application**
2. To start the development server, run the following command in the project's root directory:

```bash

npm run dev

```

1. The application will be available at **http://localhost:9002**.
2. **Component Documentation**

**Key Components**:

    - `AppShell`: Manages the startup loading screen and provides the `MusicProvider` to the app.

    - `AppLayout`: The main structural component organizing the sidebars, main view, and player.

    - `AppSidebar`: Handles navigation. It allows users to switch between Home, upload songs, and select playlists.

    - `MainView`: Renders the primary content area, showing either the home screen or a list of songs for a playlist or the main library.

    - `PlayerControls`: Provides the user interface for controlling music playback.

    - `CreatePlaylistDialog`: A dialog for creating new playlists.

**Reusable Components**: The `src/components/ui` directory contains numerous reusable components from the ShadCN library, such as `Button`, `Card`, `Dialog`, `Slider`, and `Table`, which are used throughout the application to build a consistent UI.

1. **State Management**

* **Global Stat**: Managed by `MusicContext`. It holds the `songs` array, `playlists` array, `currentTrackIndex`, `isPlaying` boolean, and the current `view`. All major components consume this context via the `useMusic` hook to access and modify the application's state.
* **Local State**: Component-level state is managed using the `useState` hook for UI-specific concerns, such as the open/closed state of a dialog or form input values.

1. **User Interface**
   * Provide screenshots or GIFs showcasing different UI features, such as pages, forms, or interactions.
2. **Styling**

* **CSS Frameworks/Libraries**: The UI is styled using **Tailwind CSS**. It also uses **ShadCN UI**, which provides a set of accessible and customizable components built on top of Radix UI and Tailwind CSS.
* **Theming**: Theming is implemented using CSS variables in `src/app/globals.css`, allowing for easy customization of colors (primary, background, accent, etc.) and other design tokens like border-radius. The app is configured with a default dark theme.

1. **Testing**

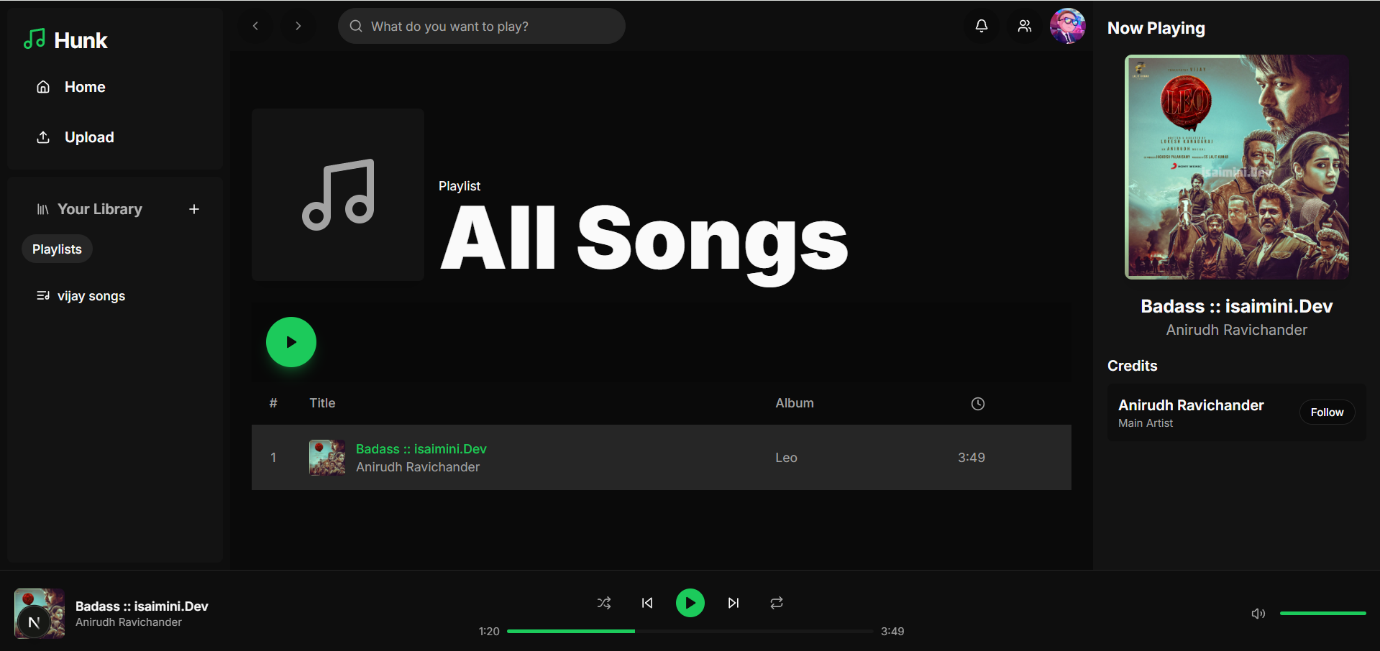
**Testing Strategy**: There is currently no formal testing strategy or framework (like Jest or React Testing Library) implemented in this project.

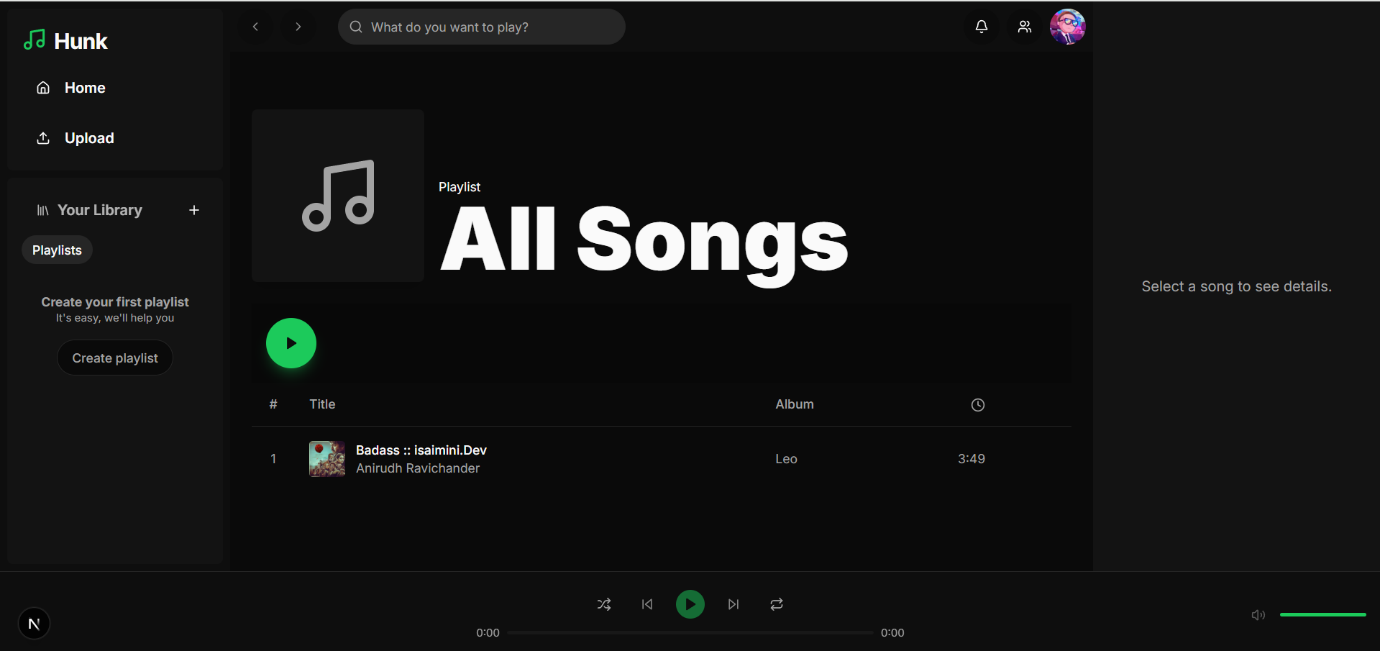
**UNIT TESTING:**  To verify individual components, that work together seamlessly

**INTEGRATING TESTING:** To ensure that different modules work together seamlessly.  
From validation was carried out to check for empty fields validation .  
Navigations buttons were verified ti ensure they scroll to the appropriate sections of the website

1. **Screenshots or Demo**







1. **Known Issues**

- The application relies on client-side state and does not persist data. Refreshing the browser will reset the library and playlists.

- The `jsmediatags` library can sometimes have issues in a Next.js environment, which may require specific webpack configuration to resolve build errors.

1. **Future Enhancements**

* **Data Persistence**: Integrate a backend or use browser `localStorage` to save user libraries and playlists between sessions.
* **AI-Powered Features**: Use the integrated Genkit framework to add features like AI-powered playlist generation or song recommendations.
* **User Accounts**: Add user authentication to support multiple user libraries.
* **Advanced Player Features**: Implement features like shuffle, repeat, and a song queue.
* **UI Animations**: Add more fluid animations and transitions to enhance the user experience.