.

**📄 Project Documentation – Music Player Frontend**

**Introduction**

**Project Title:** Music Player Frontend  
**Team Members:** [Add team members and roles]

**Project Overview**

**Purpose:**  
The purpose of this project is to create a responsive and modern music player frontend application using React.js. It provides users with features like music browsing, playback controls, and playlist management.

**Features:**

* Interactive music player UI
* Navigation across multiple pages using React Router
* API integration using Axios
* Reusable UI components with React Bootstrap and TailwindCSS styling
* Responsive layout compatible with multiple devices

**Architecture**

**Component Structure:**  
The application follows a modular React architecture. The root component (App.jsx) manages routing and high-level layout, while subcomponents handle features such as navigation, playback, and song display.

**State Management:**

* **Local State:** Managed via React’s built-in useState and useEffect hooks.
* **Global State:** Can be shared across components using props or Context API.

**Routing:**  
React Router DOM v6 is used to handle client-side routing. Routes are defined in App.jsx (or a dedicated routing file).

**Setup Instructions**

**Prerequisites:**

* Node.js (>= 16)
* npm

**Installation:**

# Clone the repository

git clone [your-repo-link]

# Navigate into project

cd music-player-frontend-

# Install dependencies

npm install

# Start the development server

npm run dev

**Folder Structure**

src/

├── App.jsx # Root component

├── main.jsx # Entry point, renders <App />

├── index.css # Global styles

├── App.css # App-level styles

├── assets/ # Images, icons, and static resources

└── Components/ # React components [list to be added]

**Utilities:**

* API calls handled with Axios.
* db/ contains mock backend data for json-server.

**Running the Application**

npm run dev

Runs the Vite development server locally at http://localhost:5173/.

**Component Documentation**

**Key Components:**

* App.jsx: Defines the main layout and routes.
* [Navbar.jsx]: Provides navigation across pages.
* [Player.jsx]: Manages playback controls (play, pause, skip).
* [SongList.jsx]: Displays a list of available tracks.

**Reusable Components:**

* [Button.jsx]: Styled buttons using React Bootstrap / TailwindCSS.
* [Card.jsx]: Song or playlist display.

*(Update with actual filenames once we confirm src/Components content.)*

**State Management**

* **Global State:** Routing state managed with React Router. Context API can be used if user/auth state is introduced.
* **Local State:** Hooks (useState, useEffect) manage component-specific states like current song, playback status, and UI interactions.

**User Interface**

* **Styling:**
  + TailwindCSS for utility-first styling.
  + Bootstrap & React-Bootstrap for responsive components.
  + Custom styles in index.css and App.css.
* **Theme Support:** Light/dark theming possible via Tailwind’s configuration.

**Testing**

**Testing Strategy:**

* Unit Testing with Jest & React Testing Library (to be implemented).
* Integration Testing for component interactions.
* Mock API testing with json-server.

**Code Coverage:**  
ESLint ensures coding standards. Additional coverage tools can be integrated.

**Screenshots or Demo**

[Insert screenshots or demo link once available]

**Known Issues**

* [Add known bugs here, e.g., “Playback button sometimes lags on mobile.”]

**Future Enhancements**

* Add authentication (login/signup).
* Playlist creation and saving.
* Enhanced animations for transitions.
* Dark mode toggle.
* Offline music playback support.