

# **Frontend Development Project Documentation**

## **1. Introduction**

**Project Title:**

Music Player App

**Team Member:**

MANIKANDAN S  
KAMALESH S  
KARTHIKEYAN P R  
KATHIRAVAN S

The Music Player App is a frontend-based project developed using **React.js** with the primary goal of creating a seamless, modern, and interactive way for users to listen to music. This application not only focuses on essential music playback features but also emphasizes **user experience (UX)** and **scalable architecture**.

The app is designed with **responsiveness in mind**, ensuring it works across devices ranging from desktops to smartphones. By incorporating intuitive controls, playlist management, and real-time search, it mirrors real-world music applications while being lightweight and customizable.

## **2. Project Overview:**

### **Purpose:**

The primary purpose of the Music Player App is to provide users with a smooth and responsive

interface for audio playback while demonstrating advanced **React development practices**.

## **Core Objectives:**

- Deliver a **user-friendly interface** with intuitive navigation.
- Showcase the **use of Context API** for managing application-wide states.
- Implement **search and filtering mechanisms** for large music libraries.
- Demonstrate responsive UI design using **CSS Flexbox and Grid**.

## **Features (Expanded):**

### **1. Audio Playback with Controls**

- Play, pause, forward, and rewind functionality.

- Visual indicators for the currently playing track.

## 2. Search Functionality

- Real-time filtering of songs by **title, genre, or artist**.
- Search optimized for scalability in larger datasets.

## 3. Favorites & Playlist Management

- Add/remove songs to a dedicated **Favorites** list.
- Create and manage multiple **custom playlists**.

## 4. Navigation

- Clear routing structure with separate views: **Home, Library, Favorites,**

## Playlist.

### 5. Modern UI

- Attractive card layouts with album artwork.
  - Responsive design across different screen sizes.
- 

### 3. Architecture

The Music Player App follows a **component-based architecture**, which promotes modularity and reusability.

#### **Component Hierarchy:**

- **App.js** → Root component that integrates routing and global state.
- **Sidebar.js** → Navigation links for Home, Favorites, and Playlists.
- **SearchBar.js** → Search input for filtering songs.
- **SongCard.js** → Displays song details (cover, title, artist) with embedded audio controls.
- **Playlist.js, Favorites.js, Library.js** → Dedicated pages for managing user content.

## State Management:

- Managed using **Context API** to avoid prop drilling.
- Global states include:

- Favorites
  - Playlists
  - Search Query
- Local states handle UI-level toggles (e.g., button hover, play/pause).

## Routing (react-router-dom):

- `/` → Home
- `/favorites` → Favorites Page
- `/playlist` → Playlist Page

---

## 4. Setup Instructions

### Prerequisites:



- **Node.js** ( $\geq 14.x$ )
- **npm** or **yarn**

## **5. Folder Structure (Expanded with Explanations)**

**music-player-app/**

| — public/           # Static files (index.html, icons, manifest.json)

| — src/

| | — assets/       # Logos, album art, and images

| | — components/   # Core reusable UI components (Sidebar, SongCard, SearchBar)

| | — context/       # Context providers for global state (Favorites, Playlists)

| | — pages/         # Different views (Home, Favorites, Playlist, Library)

| | — styles/        # Centralized CSS or styled-components

| | — App.js         # Root component managing layout and routing

| | — index.js       # Entry point for ReactDOM rendering

**Utilities:**

- **Custom Hooks:** Handle audio state (e.g., play, pause, progress tracking).
- **Utility Functions:** Searching, filtering, and formatting song metadata.

## **6. Running the Application**

- **Development Mode:**

**npm start**

**Runs on `http://localhost:3000/` with hot reloading.**

**Production Build:**

**`npm run build`**

## **7. Component Documentation (Expanded)**

**SongCard Component:**

- **Props:** `title`, `genre`, `artist`, `audioSrc`, `imageSrc`

- **Features:**

- **Play/Pause functionality.**
- **Add/Remove from Favorites.**
- **Add to Playlist with a modal selection.**

## **Sidebar Component:**

- **Provides navigation between Home, Library, Favorites, and Playlist.**
- **Highlights the active section for clarity.**

## **Reusable Components:**

- **AudioPlayer: Standalone player with playback controls.**

- **Button:** Customizable UI button for actions.
  - **SearchBar:** Input field with search icon and placeholder text.
- 

## **8. State Management**

**Global State:**

- **favorites**: Stores user's favorite songs.
- **playlist**: Stores songs added to custom playlists.
- **searchQuery**: Keeps track of current search input.

## Local State:

- Component-specific states such as:
  - Current song playing.
  - Toggle states for UI elements.

## Why Context API?

- Avoids the complexity of external libraries like Redux.

- **Lightweight and sufficient for this project's scope.**
- 

## **9. User Interface (Expanded with Description)**



- **Left Sidebar: Permanent navigation menu.**
  - **Top Search Bar: Allows instant song lookup.**
  - **Main Content Area: Displays songs in a grid layout.**
  - **Responsive Design:**
    - **Desktop → Sidebar always visible.**
    - **Mobile → Collapsible sidebar with hamburger menu.**
- 

## **10. Styling**

- **CSS Frameworks: Built with plain CSS, Flexbox, and Grid.**
  - **Theme Guidelines:**
    - **Gradient background (blue → purple).**
    - **Consistent typography with modern sans-serif font.**
    - **Rounded card corners and hover effects.**
-

## Strategy:

- **Unit Tests:** For core components using Jest.
- **Integration Tests:** For playlist/favorites logic with React Testing Library.

## Code Coverage:

- **Minimum 80% coverage targeted.**
  - **Coverage reports generated automatically in `/coverage`.**
-

## **12. Screenshots or Demo**

**(Include multiple UI screenshots: Home, Playlist, Favorites, Mobile View).**

---

### **13. Known Issues**

- 1. Song duration does not update dynamically during scrubbing.**
  - 2. Favorites and Playlist reset after page refresh (no persistence).**
  - 3. Limited error handling for invalid audio file formats.**
-

## **14. Future Enhancements**

- **Implement user authentication (Firebase/Auth0).**
- **Persistent state with localStorage or a backend API.**
- **Support song upload and custom metadata.**
- **Theme toggle (Dark/Light mode).**
- **Advanced filters like genre categories, sorting by popularity, etc.**
- **Add drag-and-drop playlist reordering.**