**Frontend Development with React.js**

**RhythmicTunes**

**Your Melodic Companion**

**Introduction:**

* **Project Title:** : RhythmicTunes: Your Melodic Companion
* **Team Id:** NM2025TMID41815
* **Team Leader:** SANJAIKARTHIKEYAN S K
* **Mail Id :** [sanjaikarthikeyan22@gmail.con](mailto:sanjaikarthikeyan22@gmail.con)
* **Team Members and Their Roles:**
* ROHIT S – UI/UX Desinger & Frontend Developer.
* SASIDHARAN S – Document & deployment Support.
* SANTHA KUMAR B - Testing & Quality Assurance.

**Project Overview:**

* **Purpose:** RhythmicTunes is a modern web-based music companion app built using React.js. It allows users to explore, stream, and organize music seamlessly. The project aims to provide a personalized listening experience with features like mood-based recommendations, curated playlists, and a responsive user interface.
* **Features:**

- User-friendly interface with responsive design  
- Music browsing and streaming by genres, moods, and trending tracks  
- Playlist creation and management  
- Favorites and recently played section  
- Search functionality with auto-suggestions  
- Future-ready architecture with scalable components

**Architecture:**

* **Component Structrue:**

- App.js: Root component handling routing and layout  
- Navbar: Navigation across app pages  
- Home: Landing page with featured playlists and songs  
- Player: Music player with controls (play, pause, next, previous)  
- Playlist: Displays user-created and curated playlists  
- Search: Handles song/artist search functionality  
- Footer: Global footer with mini-player view

* **State Management:** Context API is used for global state management, including user authentication, music playback state, and theme. Local states manage UI-specific behaviors within components.
* **Routing:** Implemented using React Router v6. Routes include /home, /playlist/:id, /search, and /player.

**Setup Instructions:**

* **Prerequisites:**

- Node.js (v16 or later)  
- npm or yarn package manager  
- Git for version control

* **Installation:**

1. Clone the repository: git clone <repository-url>
2. Navigate into the project: cd rhythmic-tunes
3. Install dependencies
4. npm install
5. Start development server
6. npm start

**Folder Structure:**

* **Client Folder Structure:**

- src/  
 - components/: Reusable UI components (Navbar, Player, Footer)  
 - pages/: Page-level components (Home, Playlist, Search)  
 - assets/: Images, icons, fonts  
 - context/: Context API providers  
 - hooks/: Custom React hooks  
 - App.js, index.js: Application entry points

* **Utilities:** Includes helper functions for API calls, formatting song durations, and custom hooks for managing audio playback.

**Running the Application:**

* **Provide commands to start the frontend server locally.**
* Running the Application:

Use the command: npm start  
This will run the app locally at <http://localhost:3000>

**Component Documentation:**

* **Key Components:**

- Navbar: Props for user authentication state and navigation links  
- Player: Props for track data and playback controls  
- Playlist: Props for playlist ID and track details  
- Search: Handles input and displays dynamic search results

* **Reusable Components:**

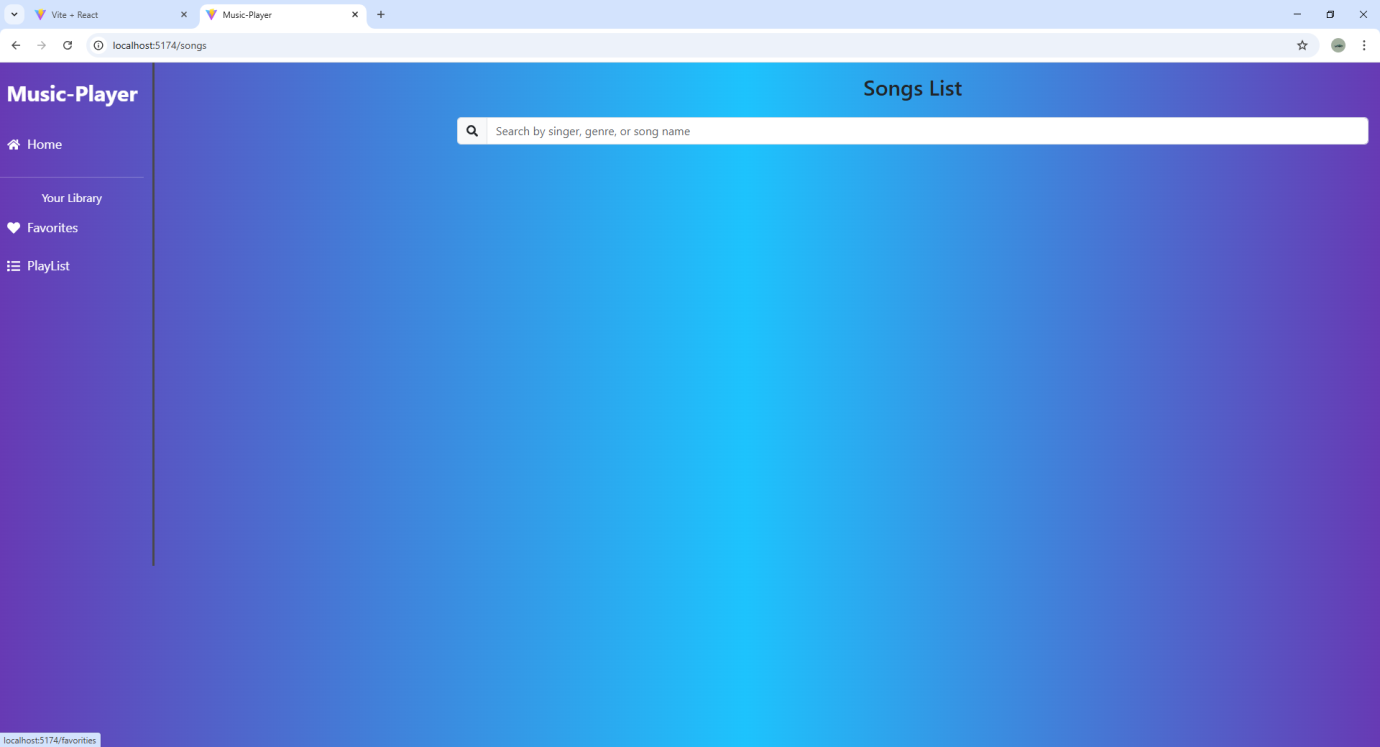
- Button: Configurable styles and actions  
- Card: Displays album/playlist info  
- Modal: Reusable popup for confirmations and settings

**Stage Management:**

* **Global State:** Managed using Context API. Handles authentication, theme, and playback state globally.
* **Local State:** Managed with React's useState hook for UI elements like dropdowns, modals, and input fields.

**User Interface:**

The UI follows a modern, minimal design with a responsive layout. Key screens include the Home page, Music Player, and Playlist view. (Screenshots can be added here in future).



**Styling:**

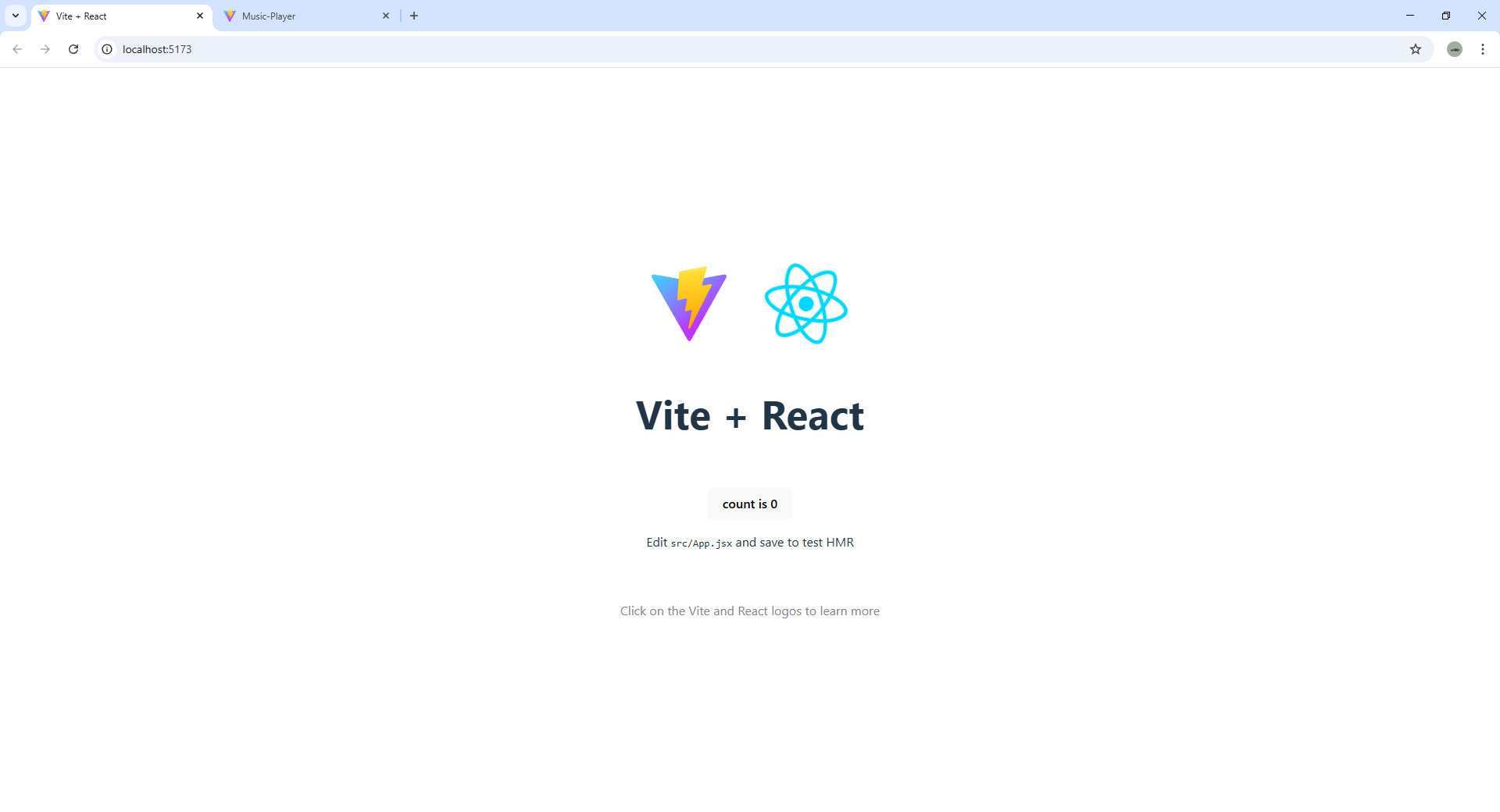
* **Styling:** Tailwind CSS is used for styling, ensuring responsive design and reusable utility classes. Custom themes are supported for dark/light modes.
* **Theming:** The app supports light and dark themes, toggleable by the user.

**Testing:**

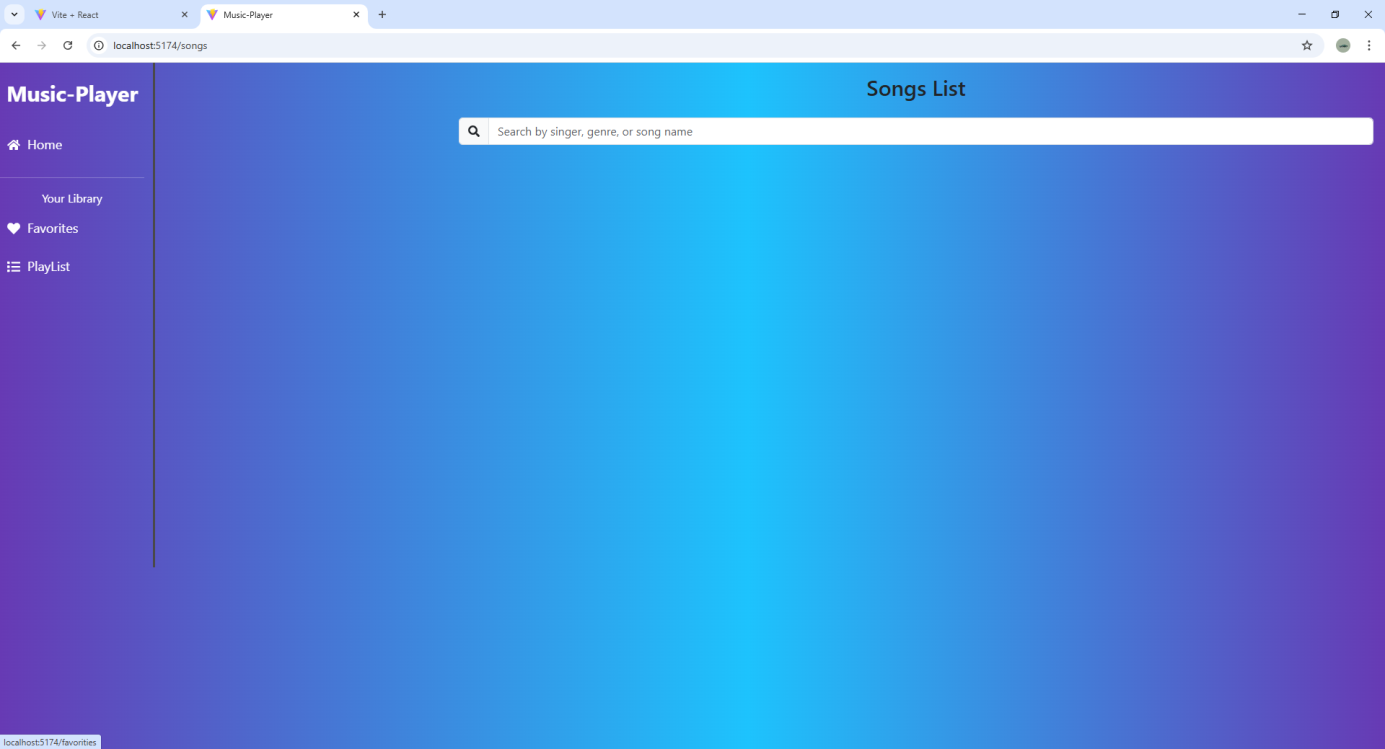
* **Testing Strategy:** Component-level unit testing with Jest and React Testing Library. Integration testing is applied for routing and playback functionality.
* **Code Coverage:** Jest coverage reports are generated to ensure test completeness.

**ScreenShot or Demo:**

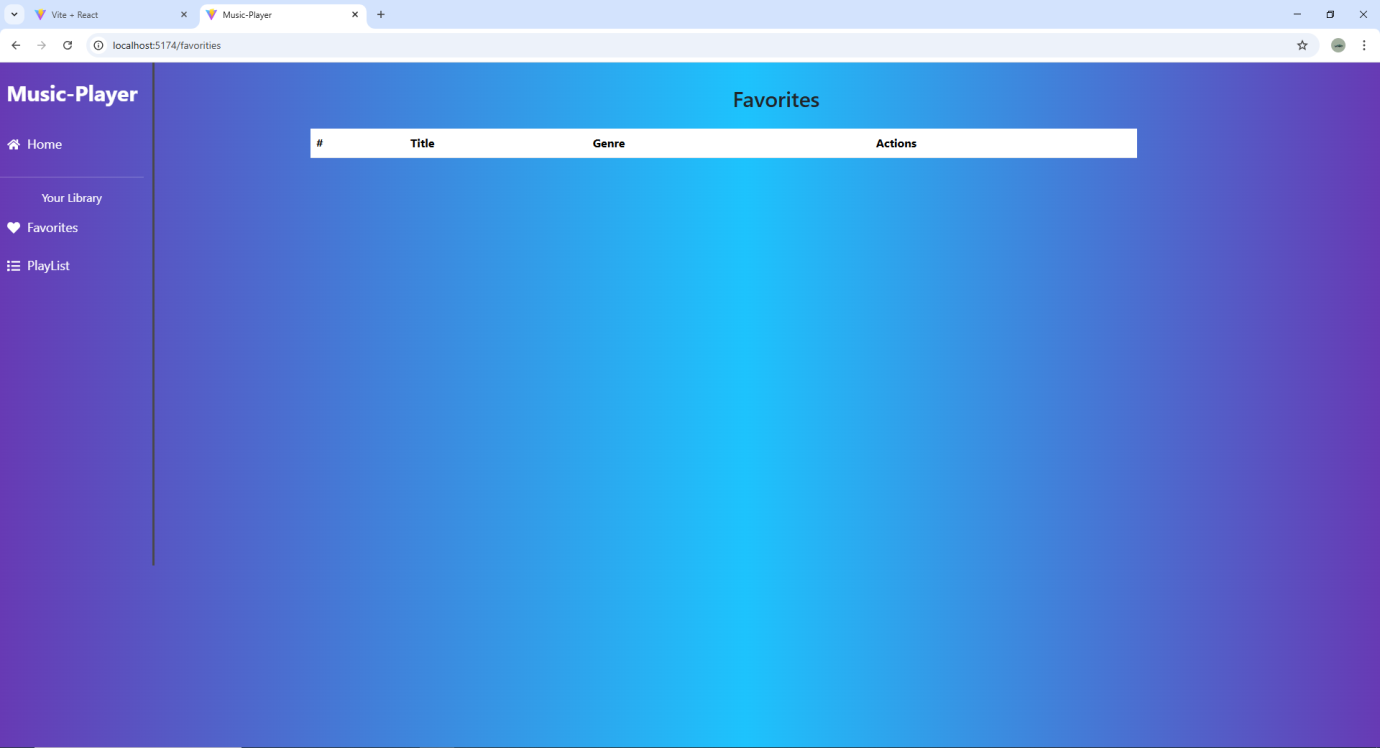
* **VITE AND REACT OUTPUT:**

****

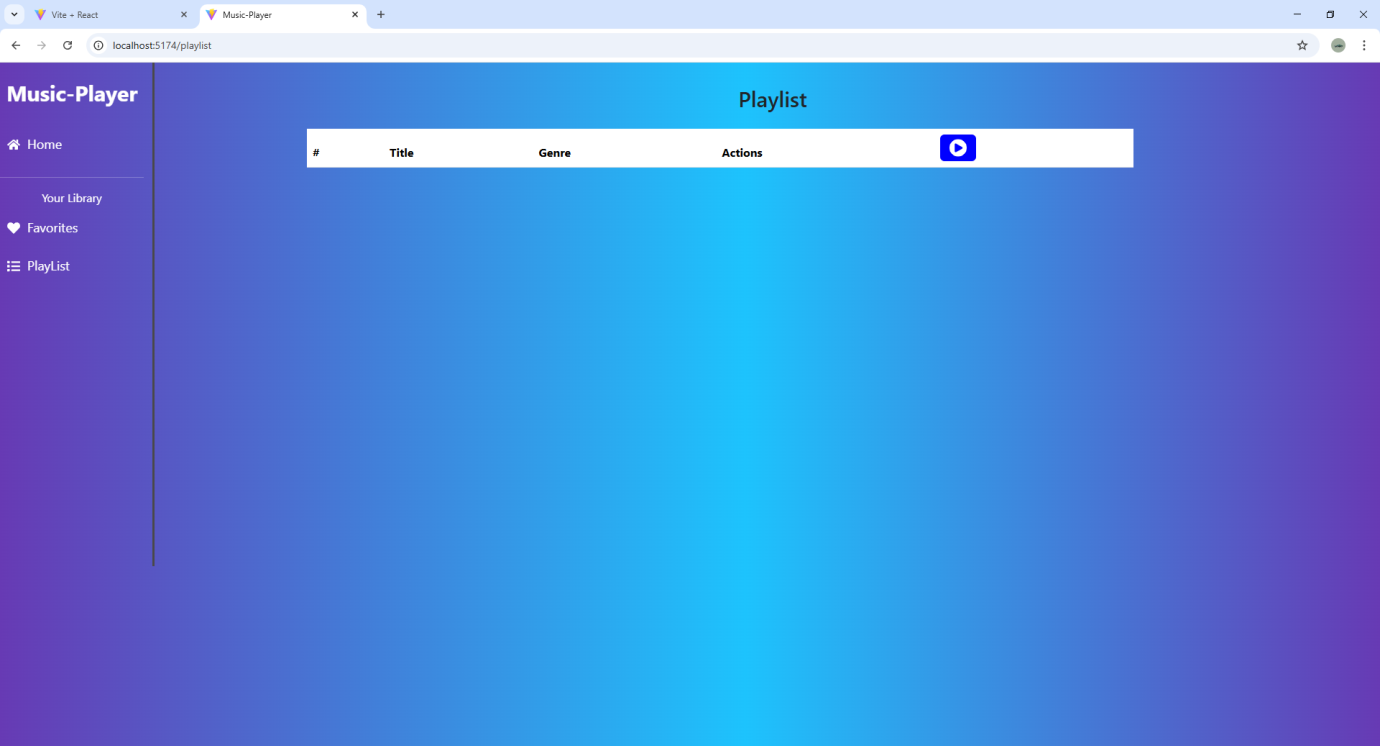
* **MAIN INTERFACE:**

****

* **FAVORITES SCREEN:**

****

* **PLAYLIST SCREEN:**

****

**Known Issues:**

* Data persistence is limited to local storage; no cloud sync is available.
* No backend integration for multi-device access or real-time updates.
* Limited authentication and user role management.
* Music recommendations are basic and not fully personalized.
* Dependency on third-party APIs for content and licensing.

**Future Enhancement:**

* AI-based song recommendations
* Offline playback support
* Social features for sharing playlists
* Mobile-first PWA version
* Advanced equalizer and audio visualizations
* Add voice-assisted controls for hands-free navigation and playback.
* Enable lyrics integration with karaoke and synchronized.