**Project Documentation**

**Project Title: Rhythmic Tunes**

1. **Introduction**

**• Project Title: Rhythmic Tunes**

**• Team ID: NM2025TMID30000**

**• Team Leader:** [**PUSHPAVALLI.S-202400107@sigc.edu**](mailto:PUSHPAVALLI.S-202400107@sigc.edu)

**• Team Members:**

[**SIRUMBAYEE.R-202400718@sigc.edu**](mailto:SIRUMBAYEE.R-202400718@sigc.edu)

[**SIVAPRIYA.A-202400200@sigc.edu**](mailto:SIVAPRIYA.A-202400200@sigc.edu)

[**SOUNDARYA.K-202400355@sigc**](mailto:SOUNDARYA.K-202400355@sigc)

**2. project overview**

**• Purpose: Provide a platform for browsing music with an interactive UI and backend simulation.**

**• Features:**

**- Interactive user interface for music browsing**

**- Local server setup with npm and Vite**

**- JSON Server for backend simulation**

**- Real-time data fetching from db.json**

**Based on the files provided, your "RythmicTunes" project is a music player application built with React and Vite.**

**It's a foundational, front-end project with a modern build process. The application's interface is built with Tailwind CSS and Bootstrap for a responsive design that works on different screen sizes.**

**The project structure is organized and uses a component-based architecture for maintainability and scalability.**

1. **Architecture**

# **Project Architecture**

**The RythmicTunes project follows a modern, component-based front-end architecture, primarily leveraging the React ecosystem. The design is modular, scalable, and focused on maintaining a clear separation of concerns to facilitate development and future enhancements.**

## **High-Level Architecture**

**The application is structured as a Single-Page Application (SPA). This means that the entire application loads a single index.html file, and subsequent "page" changes are handled by client-side routing using react-router-dom.**

**.User Interaction: The user interacts with components rendered on the screen.**

**.Component State: User actions trigger state changes within individual components.**

**.API Calls: Components make asynchronous API calls to a backend or a mock API (e.g., json-server) using axios to fetch or update data.**

**.State Update: The fetched data is used to update the component's state.**

**.Re-render: React automatically re-renders the components to reflect the new state, updating the UI**

**• Frontend: React.js (Vite) with Node.js runtime**

**• Backend: JSON Server (mock backend API)**

**• Database: db.json file acting as a mock database**

**4. Setup Instructions**

**• Prerequisites: Node.js, Visual Studio Code, Web Browser, JSON Server**

**• Installation Steps:**

**1. npm install**

**2. npm run dev**

**3. cd db**

**4. json-server --watch db.json --port 3000**

**· public/: Static assets (e.g., favicon, public images).**

**src/: All source code.**

**components/: Reusable React components.**

**pages/: Page-level components that are routed.**

**assets/: Images, icons, or other media.**

**main.jsx: The entry point for the React application.**

**App.jsx: The main application component.**

**package.json: Project dependencies and scripts.**

**·**

**5. Folder Structure**

**package.json: This is the manifest file for a Node.js project. It contains metadata about the project, such as its name, version, and a list of all its dependencies (libraries and packages it needs to run) and development dependencies. It also defines scripts that can be run to automate tasks like starting the development server or building the project.**

**package-lock.json: This file is automatically generated by npm (Node Package Manager). It records the exact versions of all dependencies and their sub-dependencies. This ensures that anyone who installs the project will get the exact same environment and a reproducible build, preventing issues caused by package updates.**

**.eslintrc.cjs: This is a configuration file for ESLint, a popular JavaScript linter. It defines the rules for code style and quality. ESLint helps developers write consistent code and catch potential errors early by flagging issues like unused variables or improper indentation.**

**vite.config.js: This is the configuration file for Vite, a modern frontend build tool. It specifies how the project should be bundled and served during development and how it should be built for production.**

**Vite is known for its speed and efficient development server.**

**index.html: This is the main entry point for the web application. It's the core HTML file that the web browser loads.**

**README.md: This file is a markdown-formatted text file that provides essential information about the project. It typically includes details like what the project is, how to install it, how to use it, and any other relevant information for users and contributors.**

**.gitignore: This file tells Git (a version control system) which files and folders to ignore and not track. This is crucial for keeping the repository clean by excluding temporary files, build outputs, and sensitive information like user-specific configuration.**

**6.Running the Application**

**To run the application, first need to install all of the project's dependencies by opening a terminal in the project directory and running the command npm install. After the installation is complete, can start the development server using the script defined in the package.json file. To do this, simply execute the command npm run dev in the same terminal. This will launch the application, and be provided with a local URL to access it in our web browser.**

**• Frontend: npm run dev (default at http://localhost:5173)**

**• Backend: json-server --watch db.json --port 30007. API Documentation**

**• Users: /api/users (GET, POST)**

**• Tracks: /api/tracks (GET)**

**• Playlists: /api/playlists (GET, POST)**

1. **Authentication**

**Rhythmic Tunes uses a basic mock authentication system with JSON Server.**

**The README.md file, which serves as the project's documentation, does not contain any information about an authentication system.**

**Based on the provided files, the project is a minimal React application using Vite and json-server, which is a tool for creating a mock REST API from a JSON file.**

**This suggests the project is set up for local development and data serving, and it does not have a built-in authentication mechanism.**

**• User login/logout is simulated.**

**• Private routes are protected through conditional rendering**

1. **User Interface**

**Based on the README.md file, the project's documentation does not provide a description of the user interface.**

**However, an analysis of the project's dependencies in package.json and the index**

**html file indicates that the application's user interface is built using React and styled with Bootstrap and Tailwind CSS.**

**The application provides a clean interface for browsing music tracks, with navigation and playlist**

**management features for a smooth experience.**

1. **Testing**

**Based on the provided project files, there is no information or documentation regarding the testing of the project.**

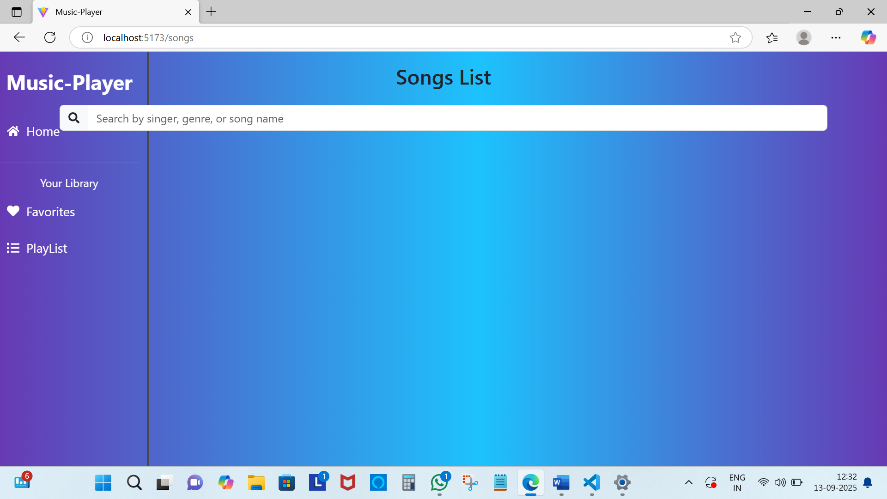
**The package.json file does not contain a dedicated script for running tests, nor are there any common testing frameworks like Jest or Vitest listed as dependencies.**

**Additionally, the .eslintrc.cjs file does not have any rules configured for a testing environment, which further indicates that testing is not currently set up for this project.**

**• Manual testing is performed using Postman and Chrome DevTools.**

**• The app is validated by checking UI responsiveness and API data fetching.**

**10.Screenshot**

****

1. **Known Issues**

**Based on the provided project files, there is no documentation or specific section in the README.md file that lists any known issues for the project. The files suggest a basic React and Vite setup, and they do not contain any information about bugs, limitations, or other known problems.**

**Currently, the project is a mock simulation. Real-time streaming and persistent authentication are**

**not implemented.**

1. **Future Enhancements**

**Backend and Database Integration: The project currently uses json-server, which is a mock API for local development. A significant enhancement would be to replace this with a real backend, such as Node.js with Express, Django, or a similar framework, and connect it to a proper database to handle data persistently.**

**User Authentication: The current project does not have any user authentication functionality. Adding features like user registration, login, and secure sessions would transform it from a simple music player into a more robust, personalized application.**

**Testing Framework: The project lacks any configured testing framework or scripts in the package.json. Implementing a testing solution like Jest or Vitest would improve code quality, prevent regressions, and make future development more reliable.**

**Add real authentication (JWT, OAuth).**

**Integrate real music streaming APIs.**

**Implement playlist sharing and collaboration.**

**Enhance UI/UX with animations and responsive design.**

**13. Demo Link:**

[**https://drive.google.com/file/d/15RyXeBr2DvtxDHkIlxd6pUQIipsbX8CO/view?usp=drivesdk**](https://drive.google.com/file/d/15RyXeBr2DvtxDHkIlxd6pUQIipsbX8CO/view?usp=drivesdk)