**Rhythmic Tunes (React)**

# 1.Introduction:-

**•Project Title:**

Our project title is Rhythmic tunes. Welcome to the future of musical indulgence – an unparalleled audio experience awaits you with our cutting-edge Music Streaming Application, meticulously crafted using the power of React.js.

**•Team Members:**

|  |  |  |
| --- | --- | --- |
| Name | Email | Role |
| P. Saradhaasree | saradhaasreepachaiyappan@gmail.com | Team leader |
| L. Santhiya | santhiyalakshmanan016@gmail.com | Team member |
| V. Amirthavarshini | amiramirtha905@gmail.com | Team member |
| E. Sangeetha | sangee031004@gmail.com | Team member |

Member1- Saradhaa Sree who is a Team Leader of the project. Tracking the progress of the project, ensuring deadlines are met. Ensuring that the project's outputs meet the required quality standards. Collaborating with other team members to complete work.

Member2- Santhiya who is a member of this project. Planning, organizing, and executing the project. Troubleshooting technical issues and offering solutions.

Member3- Sangeetha who is a member of this project. Test the product for bugs, report defects, and ensure quality control. And also created the demo video for this project.

Member4- Amirtha Varshini who is a member of this project. Managed the documentation of this project. Keep the design consistent with the project's brand identity and goals.

**2.Project Overview**

* **Purpose**

The primary goal of Music Streaming is to provide a seamless platform for music enthusiasts, enjoying, and sharing diverse musical experiences. Our objectives include: User-Friendly Interface: Develop an intuitive interface that allows users to effortlessly explore, save, and share their favorite music tracks and playlists. Comprehensive Music Streaming: Provide robust features for organizing and managing music content, including advanced search options for easy discovery. Modern Tech Stack: Harness cutting-edge web development technologies, such as React.js, to ensure an efficient and enjoyable user experience while navigating and interacting with the music streaming application.

* **Features**

Song Listings: Display a comprehensive list of available songs with details such as title, artist, genre, and release date. Playlist Creation: Empower users to create personalized playlists, adding and organizing songs based on their preferences. Playback Control: Implement seamless playback control features, allowing users to play, pause, skip, and adjust volume during music playback. Offline Listening: Allow users to download songs for offline listening, enhancing the app's accessibility and convenience. Search Functionality: Implement a robust search feature for users to easily find specific songs, artists, or albums within the app.

# 3.Architecture

**• Component structure:**

A well-structured architecture component documentation should provide a clear understanding of how different components in a project interact. Below is a structured approach:

* React JS
* React Router Dom
* React Icons
* Axios

# • State Management

When documenting State Management in a project, it's essential to explain how the system handles and updates states, ensuring clarity on data consistency, state transitions, and dependencies. Exports the App component as the default export, making it available for use in other parts of the application. Explain how data moves through the system. Define what state management means in our project.

* useState:
* items: Holds an array of all items fetched from <http://localhost:3000/items>.
  + - wishlist: Stores items marked as favorites fetched from http://localhost:3000/favorities.
    - playlist: Stores items added to the playlist fetched from <http://localhost:3000/playlist>.
    - currentlyPlaying: Keeps track of the currently playing audio element.
    - searchTerm: Stores the current search term entered by the user.
* Data Fetching:
* Uses useEffect to fetch data:

▪ Fetches all items (items) from <http://localhost:3000/items>.

▪ Fetches favorite items (wishlist) from <http://localhost:3000/favorities>.

▪ Fetches playlist items (playlist) from <http://localhost:3000/playlist>.

Sets state variables (items, wishlist, playlist) based on the fetched data.

* Audio Playback Management: Sets up audio play event listeners and cleanup for each item:

▪ handleAudioPlay: Manages audio playback by pausing the currently playing audio when a new one starts.

▪ handlePlay: Adds event listeners to each audio element to trigger handleAudioPlay.

* Ensures that only one audio element plays at a time by pausing others when a new one starts playing.
* addToWishlist(itemId): Adds an item to the wishlist (favorities) by making a POST request to http://localhost:3000/favorities.
* Updates the wishlist state after adding an item.

removeFromWishlist(itemId): Removes an item from the wishlist (favorities) by making a DELETE request to http://localhost:3000/favorities/{itemId}.

* Updates the wishlist state after removing an item.
* isItemInWishlist(itemId): Checks if an item exists in the wishlist (favorities) based on its itemId.
* addToPlaylist(itemId): Adds an item to the playlist (playlist) by making a POST request to http://localhost:3000/playlist. Updates the playlist state after adding an item.
* removeFromPlaylist(itemId): Removes an item from the playlist (playlist) by making a DELETE request to [http://localhost:3000/playlist/{itemId}](http://localhost:3000/playlist/%7bitemId%7d).
* Updates the playlist state after removing an item.

● isItemInPlaylist(itemId): Checks if an item exists in the playlist (playlist) based on its itemId. ● filteredItems: Filters items based on the searchTerm. Matches title, singer, or genre with the lowercase version of searchTerm.

● JSX: Renders a form with an input field (Form, InputGroup, Button, FaSearch) for searching items. Maps over filteredItems to render each item in the UI. Includes buttons (FaHeart, FaRegHeart) to add/remove items from wishlist and playlist. Renders audio elements for each item with play/pause functionality.

● Error Handling: Catches and logs errors during data fetching (axios.get). Handles errors when adding/removing items from wishlist and playlist.

# •Routing

Routing is the process of determining how requests are directed to different parts of an application. In software and web development, routing defines how URLs ,APIs, or network requests are handled and mapped to specific functionalities, pages, or components.

● Imports BrowserRouter, Routes, and Route from react-router-dom for setting up client-side routing in the application.

● Defines the App functional component that serves as the root component of the application.

● Uses BrowserRouter as the router container to enable routing functionality.

● Includes a div as the root container for the application.

● Within BrowserRouter, wraps components inside two div containers:

o The first div contains the Sidebar component, likely serving navigation or additional content.

o The second div contains the Routes component from React Router, which handles rendering components based on the current route.

o Inside Routes, defines several Route components:

o Route with path='/' renders the Songs component when the root path is accessed (/).

o Route with path='/favorities' renders the Favorities component when the /favorities path is accessed.

o Route with path='/playlist' renders the Playlist component when the /playlist path is accessed.

● Exports the App component as the default export, making it available for use in other parts of the application.

## 4.Setup Instructions

**•Prerequisites:**

Here are the key prerequisites for developing a frontend application using React.js: Node.js and npm: Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the local environment. It provides a scalable and efficient platform for building network applications. Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

● Download: <https://nodejs.org/en/download/>

● Installation instructions: https://nodejs.org/en/download/package-manager/ React.js: React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications. Install React.js, a JavaScript library for building user interfaces.

● Start the development server: npm run dev This command launches the development server, and you can access your React app at http://localhost:5173 in your web browser. HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential. Version Control: Use Git for version co ntrol, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

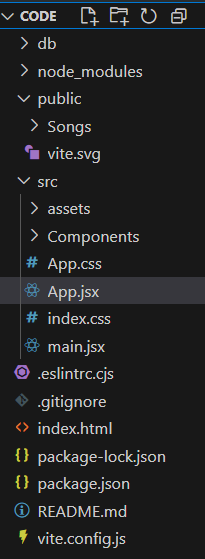
• Git: Download and installation instructions can be found at: https://git-scm.com/downloads Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

**•Installation:**

● Installation of required tools: 1. Open the project folder to install necessary tools In this project, we use: o React Js o React Router Dom o React Icons o Bootstrap/tailwind css o Axios

**5.Folder Structure**

# Client



# •Utilities

In a system, utilities refer to software tools or programs designed to help manage, maintain, and optimize computer hardware, software, or the operating system. These utilities perform essential system functions, such as ways.

* Adjusting frequencies for better sound quality.
* Removing background noise for clarity.
* Adding depth to tunes.
* Helps maintain consistent beats.
* Adjusting speed.
* Generating rhythmic patterns.

## 6. Running the application

# •Frontend

● Navigate to the project directory: cd project-name npm install

● Running the React App: With the React app created, you can now start the development server and see your React application in action.

● Start the development server: npm run dev This command launches the development server, and you can access your React app at http://localhost:5173 in your web browser.

**7. Component Documentation**

# •Key Components

The key components of a Rhythmic Tunes project depend on its functionality, whether it’s a music player, beat generator, or AI-based rhythm analyzer. Below are the essential components categorized based on software, hardware (if applicable), and user interface:

— Container Setup.

— Header.

— Search Input.

— Card Layout.

— Card Content.

— Wishlist and Playlist Buttons.

— Button Click Handlers.

— Card Styling.

—Home Screen .

— Music Player

— Waveform Visualizer

— Microphone / Sound Sensor

— Speakers / Headphones

— Beat Creator / Sequencer .

— Preset Tune Library

— Settings & Preferences

# •Reusable Components

In the Rhythmic Tunes project, reusable components are key to improving efficiency, scalability, and maintainability. These components can be used multiple times across different parts of the application, especially if built with frameworks like React.js. Below are some essential reusable components:

Frontend Reusability such as:

* Used for Play, Pause, Stop, Save, and other actions.
* A modular music player with controls like volume, tempo, and waveform visualization.
* A graphical representation of beats that can be used in play mode, editor mode, and analysis mode. – Used for volume, tempo, pitch adjustments.
* Displays song details, presets, or user-created tunes in an organized format.
* For settings, confirmations, or tune-saving options.

## 8. State management

**•Global State:**

In the Rhythmic Tunes project, managing a global state is crucial for synchronizing UI components, audio processing, and user interactions across different parts of the application. Global state ensures that tune settings, playback status, user preferences, and beats data are accessible throughout the app without redundant re-fetching or re-rendering. Global state refers to data that needs to be shared across multiple components in the project. Instead of passing data through props (in React/Vue) or function calls, a centralized state management system is used to maintain consistency.

**•Local State:**

In the Rhythmic Tunes project, local state refers to data that is managed within individual components rather than shared across the entire application. These states handle temporary interactions such as button clicks, form inputs, animations, and modal visibility that do not need to be accessed globally.

Local state is component-specific and only affects the component where it is defined. It is commonly used for: Handling UI interaction. Managing temporary input values. Tracking component behaviour. Improves performance by reducing unnecessary global state updates. Keeps components independent and easier to manage. Avoids complex state management for small UI changes.

## 9. User interface

The User Interface (UI) of the Rhythmic Tunes project is the visual and interactive part of the application where users can create, play, and customize rhythms and beats. A well-designed UI ensures smooth navigation, intuitive controls, and an engaging experience. The UI of Rhythmic Tunes consists of several key sections that help users interact with the project efficiently.

— Home Screen / Dashboard

— Music Player UI

— Beat Creator / Rhythm Sequencer

— Tune Library / Saved Tracks

— Settings Page

— Additional UI Elements

## 10. Styling

**• CSS Frameworks/Libraries**

The project utilizes Tailwind CSS as the primary utility-first CSS framework. Tailwind enables rapid UI development with a mobile-first approach and promotes consistency across components through predefined utility classes. In addition to Tailwind, Sass (SCSS) is used for managing complex stylesheets, variables, and mixins to ensure maintainability and scalability.

# • Theming

The application supports light and dark themes, which users can toggle through the UI. The theming is handled using CSS variables defined in the root of the project and dynamically updated based on the selected theme. The theme system includes color palettes, typography settings, and spacing guidelines to ensure visual consistency. Custom Design System A custom design system has been implemented to maintain a consistent look and feel throughout the application. The system includes:

A standard color palette aligned with brand guidelines. Defined typography styles (font sizes, line heights, weights). Reusable UI components such as buttons, modals, cards, and forms, styled consistently and documented for ease of use.

## 11. Testing

**•Testing Strategy**

The project follows a comprehensive testing strategy to ensure the stability and reliability of all components and features. The testing process includes:

— Unit Testing Individual components and functions are tested in isolation to verify that they behave as expected. Jest is used as the primary testing framework for unit tests due to its simplicity and extensive feature set.

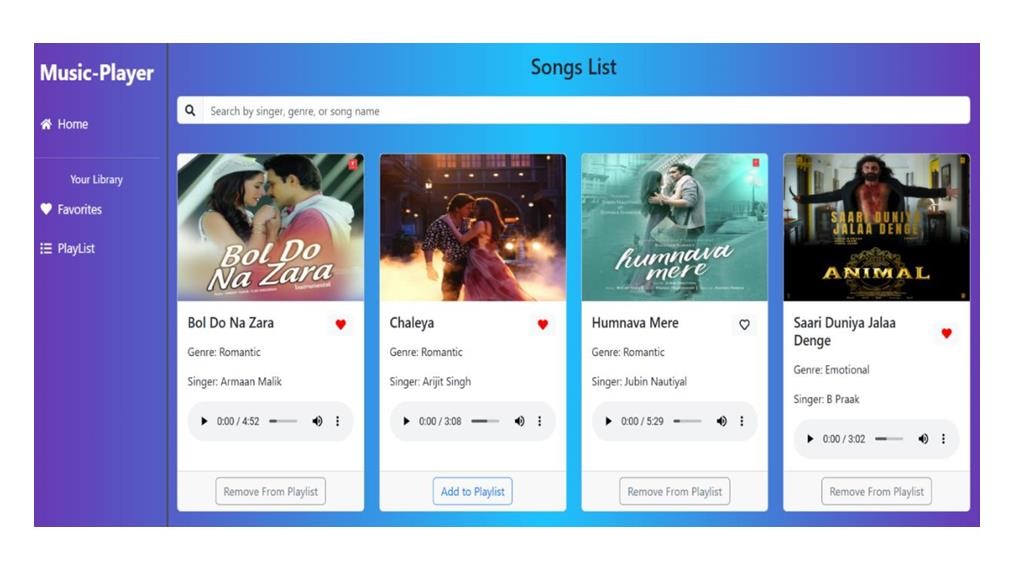
— Integration Testing Components are tested together to ensure they work Integration Testing Components are tested together to ensure they work correctly when combined. React Testing Library is used to test React components in a way that simulates real user interactions and behavior.

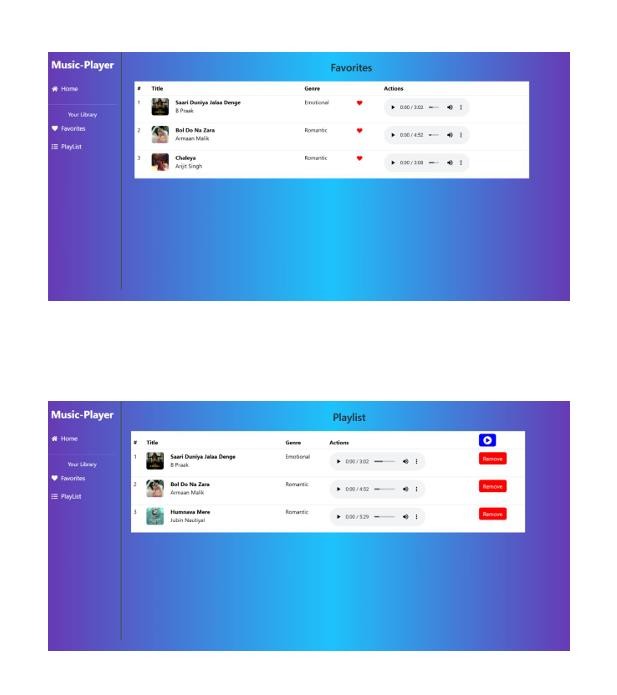
— End-to-End (E2E) Testing End-to-end tests are performed to validate the entire application workflow from the user’s perspective. Cypress is used for E2E testing, covering critical user f lows like authentication, form submission, and navigation.

# •Code Coverage

Jest is configured to generate code coverage reports to ensure that a high percentage of the codebase is tested. Coverage thresholds are enforced to maintain quality, focusing on statements, branches, functions, and lines. Reports are generated in multiple formats (HTML, text, lcov) and stored in the /coverage directory for easy review. Continuous Integration (Optional Section) Tests are automatically run as part of the CI pipeline.

## 12. Screenshots





# 13. Known Issues

User Interface (UI) Glitches Some icons or images may not load properly on lower end devices. Minor layout issues on certain screen sizes or resolution. Search Functionality Search results may not always be accurate or ranked appropriately. Delays in loading search suggestions on slow connections.

## 14. Future Enhancements

As part of our commitment to continuous improvement and providing the best user experience, several potential enhancements have

been identified for future iterations of the Music Streaming App: Add mood-based recommendations using sentiment analysis and contextual data (time of day, location, etc.). Social Features Enable users to follow friends and artists. Allow users to share playlists, songs, and activities on social media platforms. Add collaborative playlists where multiple users can contribute.

Compatibility with wearable devices and smart TVs. User-Generated Content Allow users to create and upload their own music (subject to moderation). -quality audio, and exclusive content. Include a Enable podcast hosting and streaming capabilities.