**INTRODUCTION**

**1.1 Project Title**

**Rhythmic Tunes (React)**

**1.2 Team Members**

* Hema J - (222207824) - hema.jothi20051312@gmail.com
* Deepika R - (222207812) - rajendrandeepika735@gmail.com
* Anusha devi Inbavalli S - (222207808) - anushadevi2787@gmail.com
* Divyashree A - (222207815) – Divyaarumugam0717@gmail.com
* Dhanuja M - (222207814) - sthanu1412@gmail.com

**1.3 Project Overview**

* Purpose: To provide a seamless and dynamic music streaming experience using React.js. The application aims to revolutionize how users interact with music by offering personalized recommendations and an intuitive interface.
* Features:
  + Personalized playlists tailored to user preferences.
  + Real-time search functionality for quick music discovery.
  + Smart recommendations powered by AI-driven algorithms.
  + Offline listening for uninterrupted music playback.
  + Social sharing features to connect with friends and share favorite tracks.
  + High-performance streaming optimized for all devices, ensuring a smooth and responsive experience.

**Architecture:-**

**2.1 Component Structure**

The Rhythmic Tunes application follows a modular component structure to ensure scalability and maintainability. The main components include:-

* Navbar Component: Provides navigation across different sections of the app.
* Home Component: Displays trending music and user recommendations.
* Search Component: Enables users to find songs, albums, and artists.
* Playlist Component: Manages user-created and system-generated playlists.
* Player Component: Controls music playback, including play, pause, skip, and volume adjustments.
* Favourites Component: Allows users to save and access their favorite songs.
* Authentication Component: Handles user login, signup, and authentication states.

Each component interacts with the global state to provide a seamless music streaming experience.

* Overview of major React components and their interactions.

**2.2 State Management**

The Rhythmic Tunes application employs effective state management techniques to ensure smooth user experience and data consistency across components. The state management strategies used include:

* Global State Management: Implemented using the Context API or Redux to store essential data such as user authentication status, active playlists, and playback state.
* Local State Management: Managed within individual components for handling UI interactions, such as toggling modals, updating form inputs, and controlling player settings.
* Asynchronous State Handling: Utilizes React built-in hooks like use Effect and use State in combination with Axios for fetching and updating music data dynamically.
* Persistent State: Stores user preferences and session data in local storage or session storage to enhance user experience upon reloading the application.

This structured approach ensures seamless music streaming, reduces unnecessary re-renders, and optimizes performance across devices.

* State management approach used (e.g., Context API, Redux).

**2.3 Routing**

The Rhythmic Tunes application utilizes React Router for seamless navigation across different sections of the application. The key aspects of the routing structure include:

* Dynamic Routing: Enables users to navigate between pages such as Home, Search, Playlist, and Favourite without full page reloads.
* Protected Routes: Implements authentication-based access control to restrict certain pages (e.g., user profiles and playlist management) to logged-in users.
* Lazy Loading: Uses React .lazy and Suspense to optimize performance by loading components only when needed.
* Route Parameters: Utilizes dynamic parameters in URLs for accessing specific music tracks, artists, and playlists (e.g., /playlist/:id).
* Navigation Handling: Ensures smooth transitions and history tracking using React Router's use Navigate and use Location hooks.

This structured routing setup improves performance, enhances user experience, and maintains efficient state management throughout the application.

* Explanation of routing structure using React Router.

**Setup Instructions:-**

**3.1 Prerequisites**

* Node.js
* React.js
* HTML, CSS, and JavaScript

**3.2 Installation Guide**

1. Clone the repository.
2. Install dependencies .
3. Configure environment variables (if any).
4. Start the development server .

**Folder Structure:-**

**4.1 Client**

The Rhythmic Tunes application follows a well-structured folder hierarchy to maintain scalability and ease of development. The primary folders include:

Contains the source code for the application.

* + components/: Houses reusable UI components such as Navbar, Player, and Playlist.
  + pages/: Contains different pages like Home, Search, Favorites, and Playlists.
  + assets/: Stores static assets such as images, icons, and audio samples.
  + context/: Manages global state using Context API or Redux.
  + hooks/: Custom React hooks for managing reusable logic.
  + services/: Handles API calls using Axios for fetching and updating music data.

This structured organization ensures modularity and makes maintenance easier as the project grows.

* Organization of the React application (components, pages, assets, etc.).

**4.2 Utilities**

The Rhythmic Tunes application includes various utility functions and custom hooks to streamline development and enhance performance. Key utilities include:

* Custom Hooks:
  + Use Auth(): Manages user authentication status and token storage.
  + Use Fetch(): Handles API requests and data fetching efficiently.
  + Use Audio Player(): Controls music playback state and interactions.
* Helper Functions:
  + Format Time(): Converts audio duration into a user-friendly format.
  + Filter Songs(): Filters songs based on user preferences and search queries.
  + debounce(): Optimizes search performance by reducing unnecessary API calls.
* API Services:
  + musicService.js: Handles communication with the backend for fetching music data.
  + authService.js: Manages authentication-related API requests.

These utilities help maintain clean and modular code, improving scalability and performance.

* Explanation of custom hooks, helper functions, or utility classes.

**Running the Application:-**

To run Node.js, follow these steps:-

1. Install Node.js

Download and install Node.js from:

Node.js Official Download

Verify installation:

node -v

npm -v

This should return the installed versions of Node.js and npm.

3.Install Dependencies

Run the following command inside your project folder:

npm install

This will install all necessary packages listed in package.json

4.Development Environment

* Git: <https://github.com/divyashreearumugam/project.git>
* Visual studio code: <https://code.visualstudio.com/download>
* Demo video: <https://drive.google.com/file/d/1zZuq62lyYNV_k5uu0SFjoWa35UgQ4LA9/view?usp=drivesdk>
* Document: <https://drive.google.com/file/d/14vGzt_tDkp62WOAPV8VanVXdPf_hgnFF/view?usp=drivesdk>

**Component Documentation:-**

**6.1 Key Components**

* **Navbar Component**
* Purpose: Provides navigation between different sections of the application.
* Props: user Logged In (Boolean ), on Logout (function).
* **Home Component**
* Purpose: Displays trending music, personalized recommendations, and playlists.
* Props: user Preferences (object), trending Songs (array).
* **Search Component**
  + Purpose: Allows users to search for songs, artists, and albums.
  + Props: search Query (string), on Search (function), search Results (array).
* **Playlist Component**
  + Purpose: Displays user-created and system-generated playlists.
  + Props: playlist Data (array), on Play (function), on Add To Favorites (function).
* **Player Component**
  + Purpose: Controls music playback, including play, pause, skip, and volume adjustments.
  + Props: current Track (object), is Playing (Boolean), on Play Pause (function), on Next (function).
* **Favorites Component**
  + Purpose: Allows users to view and manage their favorite songs.
  + Props: favorite Songs (array), on Remove Favorite (function).
* **Authentication Component**
  + Purpose: Handles user login, signup, and authentication status.
  + Props: on Login (function), on Signup (function), user Data (object).

Each of these components is designed to interact seamlessly with the application's state management and API services to provide an optimal user experience.

* Major components, their purpose, and the props they receive.

**6.2 Reusable Components**

The Rhythmic Tunes application includes several reusable components to maintain consistency and improve efficiency in development. These components are designed to be modular and adaptable across different parts of the application:

* **Button Component**
  + Purpose: A customizable button used across various sections.
  + Props: label (string), on Click (function), variant (string: primary/secondary).
* **Input Component**
  + Purpose: Handles user input for forms and search fields.
  + Props: type (string), placeholder (string), value (string), on Change (function).
* **Card Component**
  + Purpose: Displays music tracks, albums, or playlists in a structured format.
  + Props: title (string), image (URL), description (string), on Click (function).
* **Modal Component**
  + Purpose: Provides pop-up dialogues for playlist creation and song details.
  + Props: is Open (Boolean), on Close (function), content (JSX element).
* **Loader Component**
  + Purpose: Displays a loading animation during data fetching.
  + Props: size (string: small/medium/large), colour (string).

These reusable components help streamline development, ensuring a consistent UI and reducing redundant code.

* List of reusable UI components and their configurations.

**State Management:-**

**7.1 Global State**

The Rhythmic Tunes application uses global state management to synchronize data across different components efficiently. Key global state management strategies include:

* **Context API**: Used to manage global state such as authentication status, user preferences, and music playback status.
* **Redux (if applicable)**: Centralized store for managing application-wide state changes and ensuring smooth interactions.
* **Persisted State**: Local storage or session storage is used to retain user preferences and session data.

This approach ensures that user data and preferences are seamlessly available across different parts of the application.

* Description of how state flows across the application.

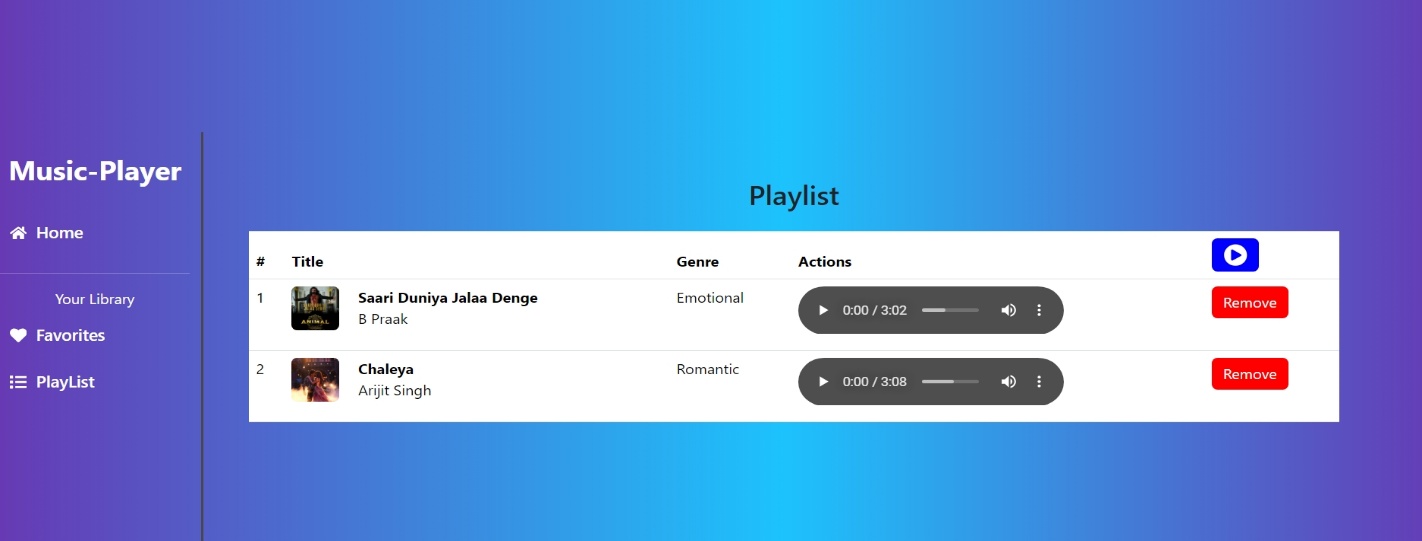
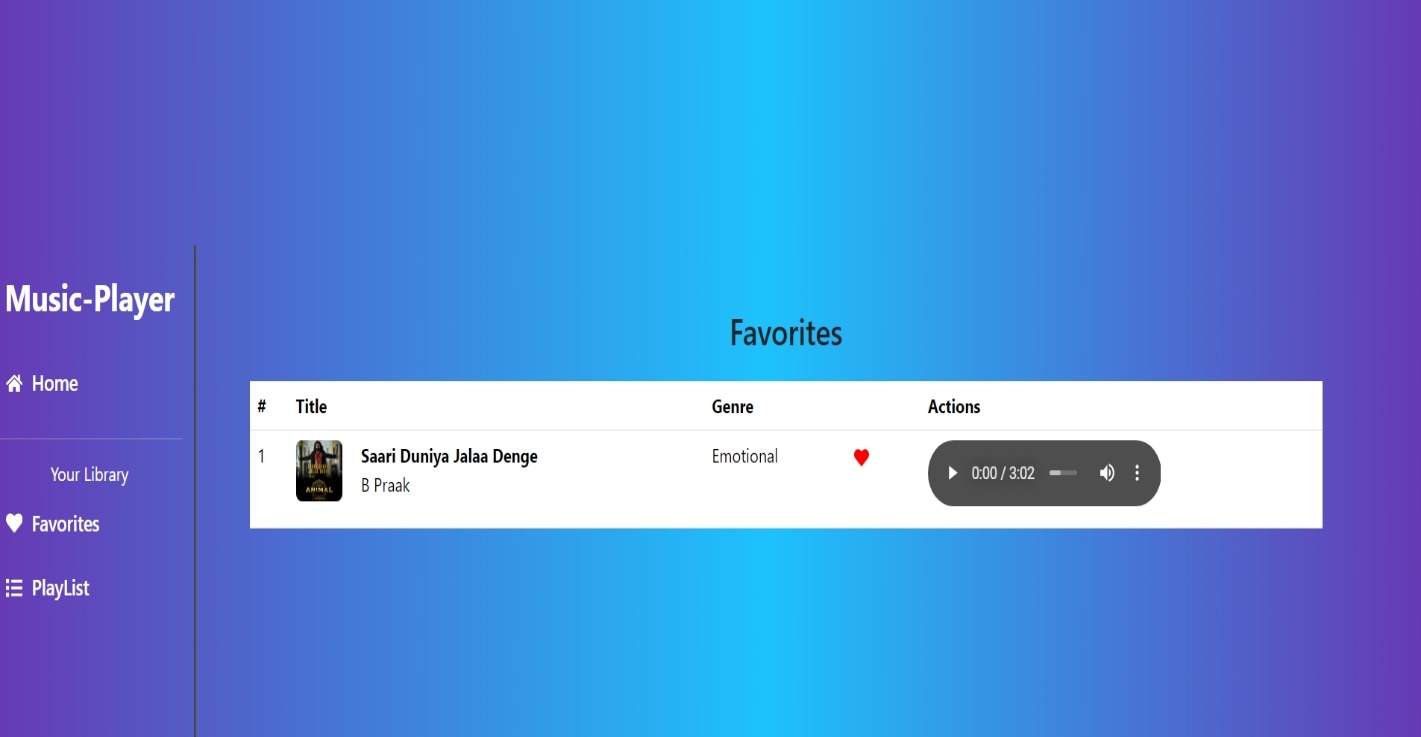
**7.2 Local State**

Local state is managed at the component level to handle UI-related interactions efficiently. Some examples include:

* **Player Controls**: Managing play, pause, and volume control states within the Player component.
* **Form Inputs**: Handling user input fields for login, signup, and search functionalities.
* **Modal Visibility**: Controlling the opening and closing of modals for playlist creation and song details.
* **Component-Specific UI Updates**: Managing animations, dropdown selections, and toggles.

Using local state optimally reduces unnecessary re-renders and enhances application performance.

* Explanation of handling local state within component.



**Code Coverage**

To ensure robust testing and maintain high-quality code, the Rythimic Tunes application incorporates tools and techniques to measure test coverage:

* **Jest & React Testing Library**: Used to track component test coverage, ensuring all critical UI elements function as expected.
* **Istanbul/NYC**: Generates test coverage reports, identifying untested areas in the codebase.
* **End-to-End Testing**: Cypress is used to validate user flows and application stability.
* **Coverage Metrics**:
  + **Statement Coverage**: Ensures all statements in the code execute at least once.
  + **Branch Coverage**: Verifies all possible code branches are tested.
  + **Function Coverage**: Ensures all functions execute correctly under different conditions.

By implementing these testing strategies, we maintain a stable and reliable music streaming experience for users.

* Tools used for measuring test coverage.

**Future Enhancements**

Potential improvements such as animations, enhanced UI, or additional features.

**Rhythmic Tunes (React) Overview:-**

## Scenario-Based Introduction:

Imagine stepping onto a bustling city street, the sounds of honking cars, chatting pedestrians, and street performers filling the air. On your way to work, you open "Rhythmic Tunes," your go-to music streaming app. Within seconds, a personalized playlist uplifts your mood, adapting seamlessly to your environment—starting with an upbeat pop song and transitioning into a relaxing indie track as you board the train.

**Advantages & Disadvantages**

**2.1 Advantages**

* **Smooth & Fast Performance**: Optimized rendering for lag-free navigation and playback.
* **Cross-Platform Accessibility**: Music quality and features are maintained across devices.
* **Personalized Experience**: AI-driven recommendations enhance music discovery.
* **Engaging UI**: A clean and interactive interface improves user satisfaction.
* **Flexible Music Discovery**: Dynamic search and categorization for easy browsing.
* **Social & Collaborative Features**: Social media integration for playlist sharing and engagement.
* **Offline Listening (If Available)**: Users can save music for offline playback.
* **Freemium & Subscription Choices**: Options for free, ad-supported streaming or premium plans.

**2.2 Disadvantages**

* **Requires Internet for Streaming**: Unless offline mode is available.
* **Potential Data Consumption**: High-quality streaming can use significant mobile data.
* **Subscription Costs for Premium Features**: Ad-free listening and offline access often require a paid subscription.
* **Limited Content Availability**: Licensing restrictions may limit song availability.
* **Battery Drain on Mobile Devices**: Continuous streaming and animations may impact battery life.
* **Ads in Free Version**: Frequent ads may interrupt the listening experience.

**Sample code:**

{

"name": "music-player-frontend-",

"private": true,

"version": "0.0.0",

"type": "module",

"scripts": {

"dev": "vite",

  "build": "vite build"

"lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",

"preview": "vite preview",

"server": "json-server --watch ./db/db.json"

},

"dependencies": {

"axios": "^1.6.2",

"bootstrap": "^5.3.2",

"json-server": "^0.17.4",

"react": "^18.2.0",

"react-bootstrap": "^2.9.1",

"react-dom": "^18.2.0",

"react-icons": "^4.12.0",

"react-router-dom": "^6.20.1",

"tailwindcss": "^3.3.6"

},

"devDependencies": {

"@babel/parser": "^7.26.9",

  "@babel/template": "^7.26.9",

  "@babel/types": "^7.26.9",

  "@types/react": "^18.2.43",

"@types/react-dom": "^18.2.17",

"@vitejs/plugin-react": "^4.2.1",

"eslint": "^8.55.0",

"eslint-plugin-react": "^7.33.2",

    "eslint-plugin-react-hooks": "^4.6.0"

  "eslint-plugin-react-refresh": "^0.4.5",

  "vite": "^5.0.8"

  },

  "description": "This template provides a minimal setup to get React working in Vite with HMR and some ESLint rules.",

  "main": "vite.config.js",

"keywords": [],

"author": "",

  "license": "ISC"

}

**Conclusion:-**

Rhythmic Tunes (React) offers a next-generation music streaming experience through its seamless UI, smart recommendations, and engaging social features. Whether online or offline, users can enjoy music their way.

