# 1.INTRODUCTION:

PROJECT TITLE: RHYTHMIC TUNES

TEAM MEMBERS AND THEIR ROLES:

NAME: HARINI S

ROLE: TEAM LEADER

EMAIL: harini.s.25.bsccs@princeshribalaji.in

NAME: ABINAYA V

ROLE: DOCUMENTATION CREATOR

EMAIL: abinaya.v.25.bsccs@princeahribalaji.in

NAME: SHAFI GANI I

ROLE: SPEAKER ABOUT THE PROJECT

EMAIL: shafigani.i.25.bsccs@princeshribalaji.in

NAME: LEN PREETHA MARY V

ROLE: TAKING NOTES ABOUT THE PROJECT

EMAIL:

lenpreethamary.v.25.bsccs@princeahribalaji.in

# 2.PROJECT OVERVIEW:

## PURPOSE:

Aims to provide users with an engaging and interactive music experience. It allows users to explore, play, and interact with music through a modern, responsive interface. The project focuses on seamless UI/UX, smooth audio playback, and dynamic features such as playlists, song recommendations, and real time visualizations.

## FEATURES:

**MUSIC PLAYER:** Play, pause, skip and control volume for seamless audio playback.

**PLAYLISTS MANAGEMENT:** Create, edit, and manage custom playlists.

**SEARCH & FILTER:** Find songs by title, artist, genre, or album.

**REAL TIME LYRICS:** Display synchronized lyrics while playing music.

**FAVOURITIES AND LIKES:** Save favorite songs and albums for quick access.

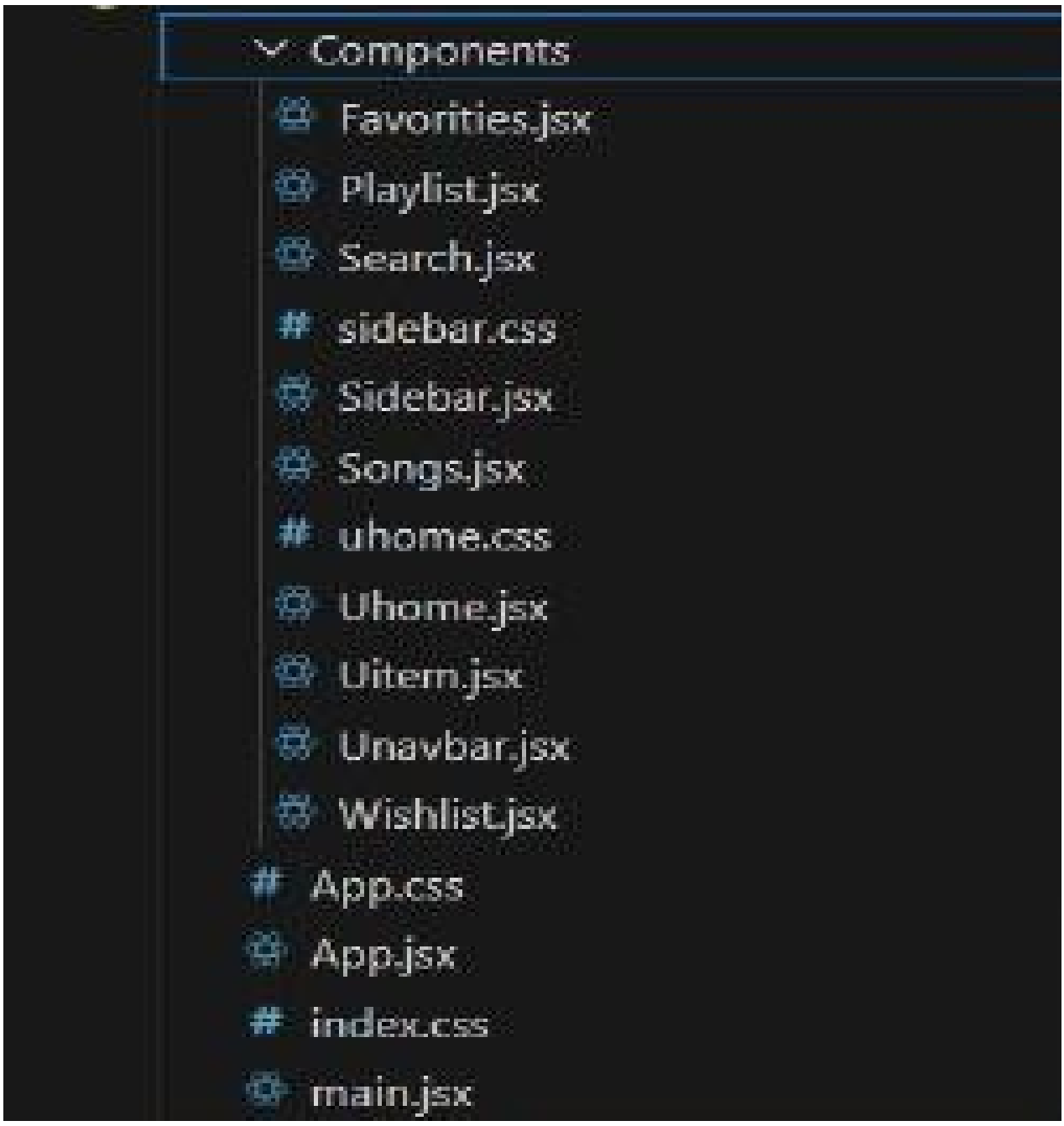
# 3.ARCHITECTURE:

Architecture, like music, is defined by rhythm and harmony, with each structure pulsating to its own beat. Foundations act as the steady bass, while beams and columns create the beats that shape the space. Light and shadow play the melodies, dancing across surfaces in a dynamic composition. The curves and angles of a building add their unique tempo, guiding movement and emotion. Together, architecture and music weave a symphony, transforming spaces into experiences.

# COMPONENT STRUCTURE:

1. **Favourites**– Displays a list of favorite songs or playlists saved by the user.
2. **Playlist.jsx** – Handles the creation, display, and management of playlists.
3. **Search.jsx** — Provides a search bar for users to find songs, albums, or artists.
4. **Sidebar.jsx** — Contains navigation links for different sections like Home, Favorites, and Playlists.

5. **Songs.jsx** — Displays a list of songs with options to play, add to playlists, or favorite them.



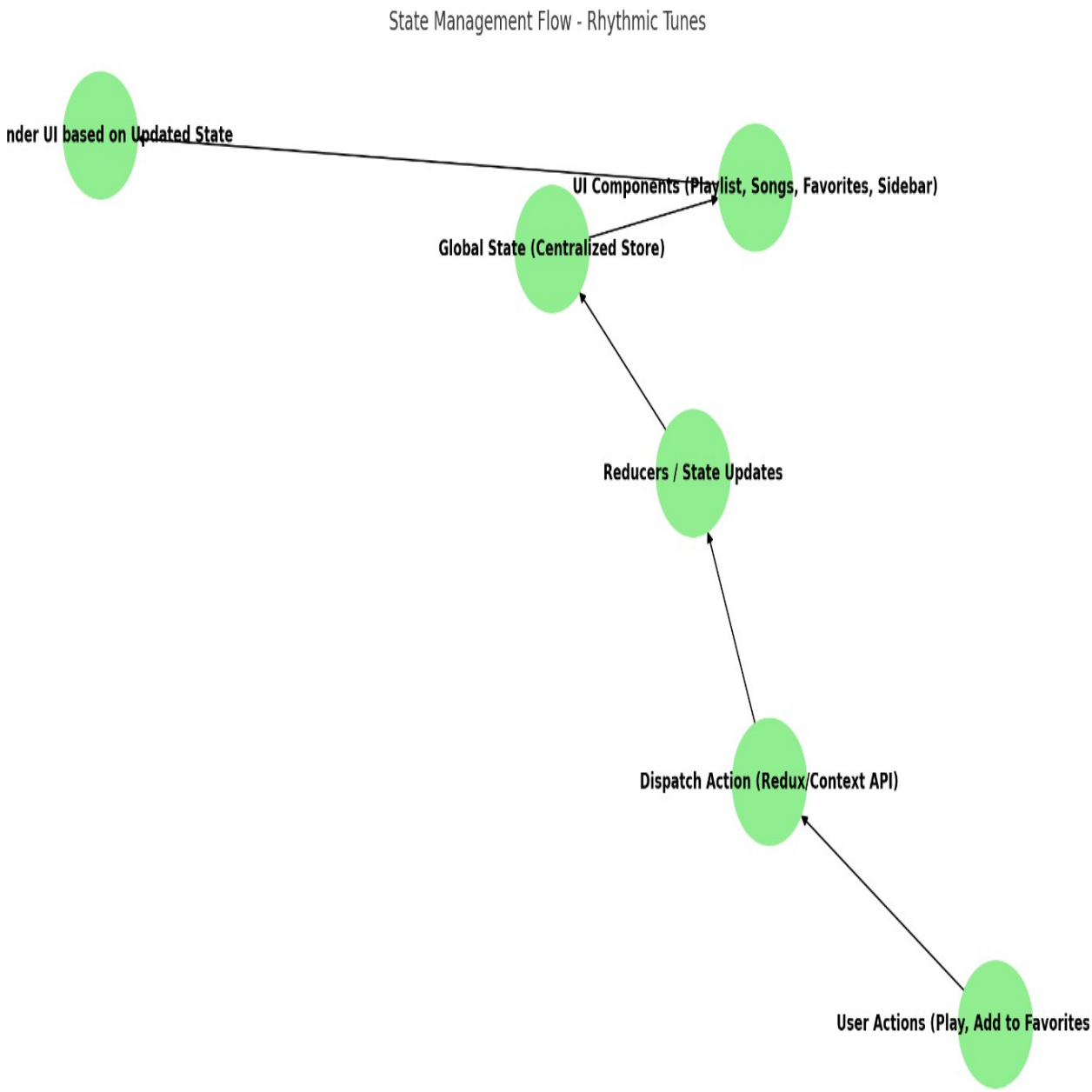
# STATE MANGEMENT:

There are three main contexts to manage different states:

**Player context:** Handles music playback

**Theme context:** Manages dark/light model toggling

**Playlist context:** Stores user’s playlists, liked songs and history


State Management Flow - Rhythmic Tunes
nder UI based on Updated State
UI Components (Playlist, Songs, Favorites, Sidebar)
Global State (Centralized Store)
Reducers / State Updates
Dispatch Action (Redux/Context API)
User Actions (Play, Add to Favorites

# ROUTING:

The Rhythmic Tunes project uses React Router for seamless navigation, defining routes for pages like Home, Playlist, Search, and Favorites. It supports dynamic routing, route parameters, and protected routes, ensuring efficient and scalable SPA navigation.

# CODING:

```
import 'bootstrap/dist./CSS/bootstrap.min.css';
import './App.css'
import {BrowserRouter, Routes, Route} from 'react-router-dom'
import Songs from ‘. /Components/Songs'
import Sidebar from ‘. /Components/Sidebar'
import Favorites from ‘. /Components/Favorites’
import Playlist from ‘. /Components/Playlist';
function App () {
  return (
  <div >
```

```
    <Browser Router>
    <div>
    <Sidebar/>
    </div>
 <div>
         <Routes>
           <Route path='/songs' element={<Songs/>}
/>
           <Route path='/favorites'
  element={<Favorites/>} />
            <Route path='/playlist' element={<Playlist/>}
/>
          </Routes>
         </div>
       </Browser Router>
</div>
  )
}
export default App
```

# 4.SETUP INSTRUCTIONS:

## PREREQUISTIES:

### 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 nm. on your development machine, as they are required to run JavaScript on the server side

### 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.

### 1.CREATE A NEW REACT APP:

npm. create Vite latest

**2.NAVIGATE THE PROJECT DIRECTORY:**
cd project-name npm install

**3.START THE DEVELOPMENT SERVER:** npm run dev

**4.HTML, CSS, JS:** Basic knowledge of html for creating the structure of your app, CSS for styling and JavaScript for client-side interactivity is essential.

**5.VERSION CONTROL:** Use git for version control enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or bitbucket can host your repository.

**6.DEVELOPMENT ENVIRONMENT:** Choose a code editor or integrated development environment that suits your preferences such as visual studio code, sublime text, or WebStorm.

# INSTALLATION:

# STEP BY STEP GUIDE TO CLONE THE REPOSITORY:

## 1. Clone the Repository

First, open a terminal and run the following command to clone the project:

Navigate into the project directory: sh

cd rhythmic-tunes

## 2. Install Dependencies

Ensure you have Node.js (LTS version) installed. Then, install the required dependencies:

sh

npm install

## 3. Start the Development Server

Once dependencies are installed and environment variables are set, start the server:

sh

npm start

## 4. Verify the Setup

- Ensure no errors appear in the terminal.

- Open http://localhost:3000/ in your browser to check if the app loads correctly.

Installation of requires tools:

- ❖ React JS
- ❖ Router Dom
- ❖ Icons
- ❖ Bootstrap
- ❖ Axios

Setup react application:

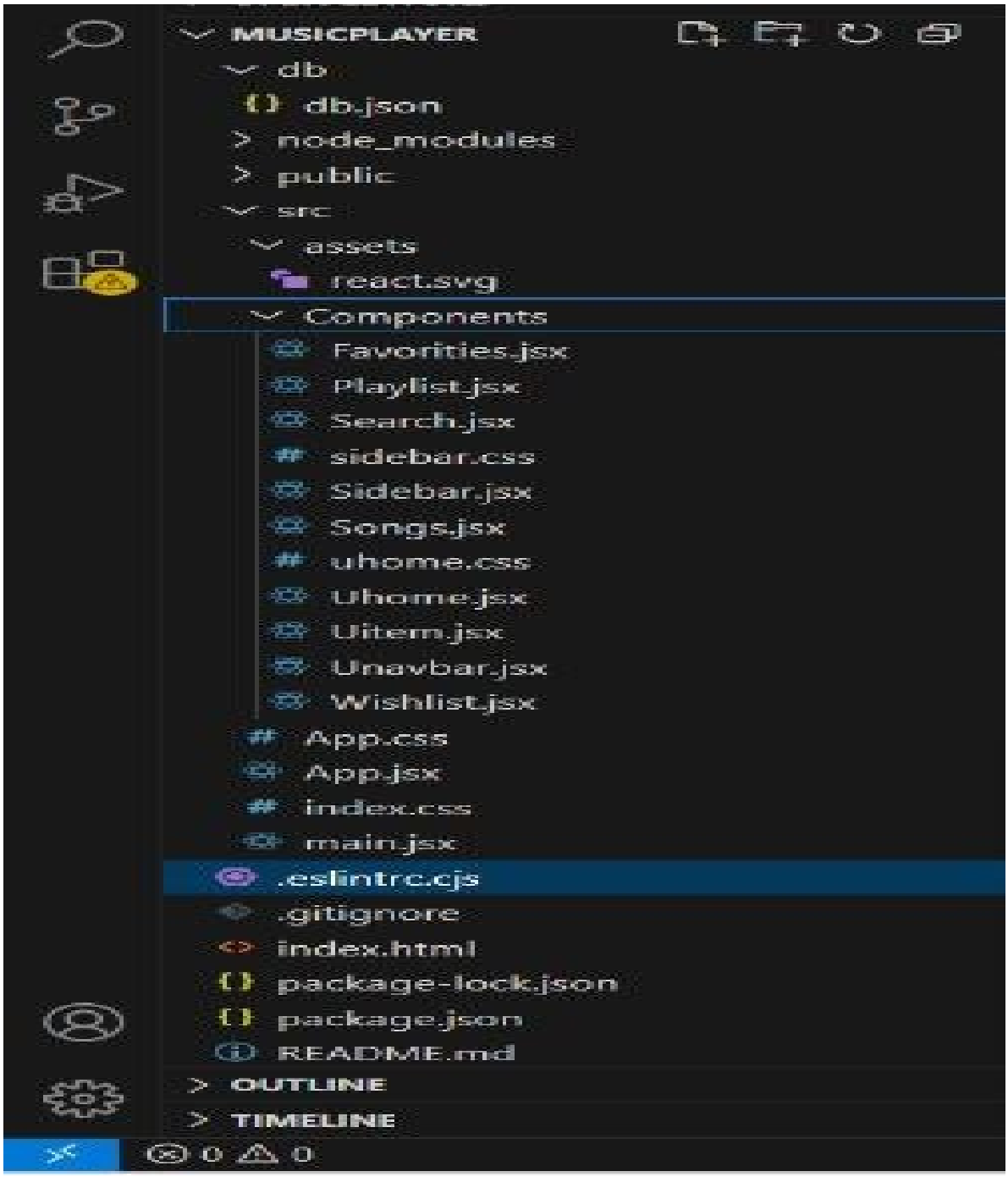
- o Create react application
- o Configure routing
- o Install required libraries

# 5.FOLDER STRUCTURE:

# CLIENT:

The Rhythmic Tunes project follows a well-structured client-side architecture using React.js to

ensure modularity, maintainability, and scalability.



# UTILITIES:

The utilities folder in the Rhythmic Tunes project contains reusable helper functions that simplify common operations.

## 1. Helper Functions (Utilities)

Stored inside the utils/ or services/ folder, these functions simplify common operations.

*Example: Formatting Time for Songs*

*Used to convert song duration (in seconds) into a MM: SS format.*

## 2. Utility Classes (For Styling)

Stored in the styles/ folder, these CSS classes ensure consistent UI design.

*Example: Reusable Button Styles (utilities.css)*

*Used to Playlist, Favourites, Search, and Song Actions.*

## 3. Custom Hooks for Reusable Logic

*Example: useAuth.js (Authentication Hook)*

*Handles user authentication state and session* storage.

Stored in the hooks/ folder, custom hooks handle stateful logic across multiple components.

# 6.TO RUN THE APPLICATION:

## FRONT-END DOCUMENTATION:

### 1. Navigate to the Client Directory

Open a terminal and move into the client folder: sh

cd rhythmic-tunes/client

### 2. Install Dependencies

Ensure all required packages are installed:

sh

npm install

### 3. Start the Frontend Server

Run the following command to launch the development server:

sh

npm start

### 4. Access the Application

Once the server is running, open your browser and visit:

http://localhost:3000/

# 7.COMPONENT DOCUMENTATION:

## KEY COMPONENTS:

### 1. Navbar Component (Unavbar.jsx)

📌 **Purpose:** Displays the top navigation bar with links to key pages like Home, Search, and Favorites.

- 🔹 **Props:**
  - user (Object) – User details for authentication display.
  - on Logout (Function) – Handles logout action.

### 2. Sidebar Component (Sidebar.jsx)

📌 **Purpose:** Provides side navigation with links to playlists, songs, and other sections.

- 🔹 **Props:**
  - playlists (Array) – List of user playlists.

### 3. Playlist Component (Playlist.jsx)

📌**Purpose:** Displays a list of songs in a selected playlist.

- 🔹 **Props:**
  - playlist (Object) – Playlist data including title

and songs.

### 4. Favourites Component (Favosites's)

📌 **Purpose:** Shows the user's favorite songs.

🔹 **Props:**

- favorites (Array) – List of favorite songs.
- on Remove (Function) – Removes a song from favorites.

### 5.Songs Component (Songs.jsx)

📌 **Purpose: Displays a list of all available songs.**

🔹 **Props:**

- songs (Array) — List of song objects.
- on Play (Function) – Handles song playback.

### 6.Search Component (Search.jsx)

📌 **Purpose:** Provides a search bar to find songs and playlists.

🔹 **Props:**

- on Search (Function) – Handles search input changes.
- results (Array) – Displays search results.

### 7. Wishlist Component (Wishlist.jsx)

📌 **Purpose:** Displays songs added to the Wishlist for later listening.

- **Props:**
  - Wishlist (Array) – List of songs saved for later.
  - On Remove (Function) – Removes a song from the wish list.

## REUSABLE COMPONENTS:

### 1. Button Component (Button's)

📌 **Purpose:** A customizable button used across the app for actions like play, pause, and add to playlist.

- **Props:**
  - label (String) – Button text.
  - on Click (Function) – Click event handler.
  - variant (String) – Defines button style (primary, secondary, etc.).
- **UsageExample:**

jsx

<Button label="Play" on Click= {handle Play}

variant="primary" />

- **Configuration(Button. Jsx):**

jsx

```
const Button = ({label, on Click, variant = "primary”})
=> {
  return (
    <button class Name= {`btn ${variant}`}
onClick={onClick}>
      {label}
    </button>
  );
};
export default Button;
```

## 2. Modal Component (Modal. Jsx)

📌 **Purpose:** Displays pop-up dialogs for actions like adding to a playlist or confirming song deletions.

- **Props:**
  - is Open (Boolean) — Controls visibility.
  - on Close (Function) – Closes the modal.

- title (String) – Modal heading.

◆ **Usage Example:**

jsx

```
<Modal is Open= {show Modal} on Close= {handle Close} title="Add to Playlist">
  <p>Select a playlist to add this song. </p>
</Modal>
```

◆ **Configuration (Modal. Jsx):**

jsx

```
const Modal = ({is Open, on Close, title, children}) => {
  if (! is Open) return null; return (
    <div class Name="modal-overlay">
      <div class Name="modal-content">
        <h2>{title}</h2>
        {children}
        <button onClick= {on Close}>Close</button>
      </div>
    </div>
  );
```

};

export default Modal;

### 3. Input Component (Input. Jsx)

📌 **Purpose:** A reusable input field used in forms (e.g., search, login, playlist creation).

- **Props:**
  - type (String) – Input type (text, password, etc.).
  - placeholder (String) – Placeholder text.
  - value (String) – Input value.
  - on Change (Function) – Handles input changes.
- **Usage Example:**

jsx

<Input type="text" placeholder="Search songs..." value={query} on Change= {handle Search} />

- **Configuration (Input. Jsx):**

jsx

const Input = ({type, placeholder, value, on Change}) => {

return (

<input type={type} placeholder={placeholder} value={value} on Change= {on Change} />

);

};

export default Input;

## 4. Card Component (Card.jsx)

📌 **Purpose:** A reusable card layout used for songs, playlists, and albums.

- **Props:**
  - image (String) – Thumbnail or album art.
  - title (String) – Card title (e.g., song name).
  - description (String) – Subtitle or details.
  - onClick (Function) – Click handler (e.g., play song).
- **UsageExample:**

jsx

<Card image="song.jpg" title="Song Name" description="Artist Name" onClick= {play Song} />

- **Configuration(Card.jsx):**

jsx

Copyedit

```
const Card = ({image, title, description, onClick})
=> {
  return (
    <div class Name="card" onClick={onClick}>
      <img src={image} alt={title} />
      <h3>{title}</h3>
      <p>{description}</p>
    </div>
  );
};
export default Card;
```

## 5. Loader Component (Loader. Jsx)

📌 **Purpose:** Displays a loading animation when fetching data.

- 🔹 **Props:**
  - message (String) — Custom loading message (optional).
- 🔹 **Usage Example:**

jsx

{is Loading && <Loader message="Loading Songs..." />}

- Configuration(Loader. Jsx):

jsx

```
const Loader = ({message = "Loading...”}) => {
  return <div class Name="loader">{message}</div>;
};
```

# 8.STATE MANAGEMENT:

# GLOBAL STATE:

## Global State Management in Rhythmic Tunes

In Rhythmic Tunes, global state management ensures smooth data flow and synchronization across components. This helps manage user authentication, playlists, song playback, and favorites efficiently.

- **State Management Approach**

We use React Context API (or Redux, if required) to manage the global state. This allows components to access shared state without excessive prop drilling.

**Key Global States Managed**

1. **User Authentication State** – Stores user login information.
2. **Current Playing Song** — Tracks which song is playing.
3. **Playlist Data** — Manages user playlists.
4. **Favourites & Wishlist** – Stores user-favorite songs.
5. **Search & Filter State** — Handles search queries.

- **How State Flows Across the Application**

➤ State is stored globally in the Context Provider (AppContext.js).

➤ Components subscribe to the global state using the use context hook.

➤ State updates trigger re-renders in components that depend on the updated data.

## LOCAL STATE:

In Rhythmic Tunes, local state is managed using Reacts use State hook within components to handle temporary, component-specific data such as form inputs, toggles, and UI states.

- 🔹 When to Use Local State?

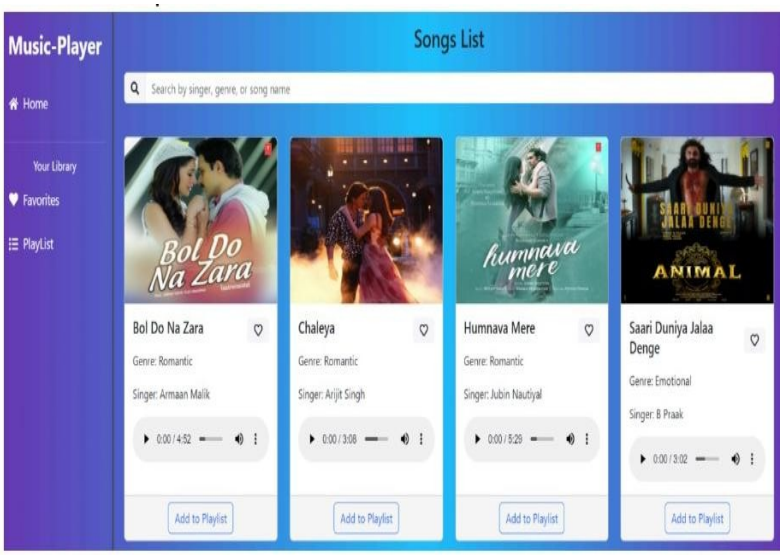
Local state is used when data doesn't need to be shared globally. Examples include:

1. **Form Inputs** – Storing user input in a search bar.
2. **UI Controls** — Managing modal visibility or dropdown selections.
3. **Component-Specific Interactions** — Like toggling favorite status.

## 9.USER INTERFACE:

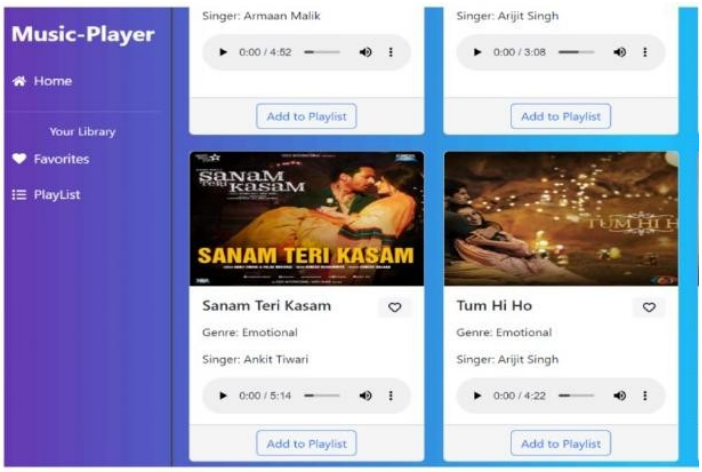
✔ **Homepage (Landing Page):**

📌 **Description:** Show the homepage layout, including featured playlists, top songs, and navigation options.



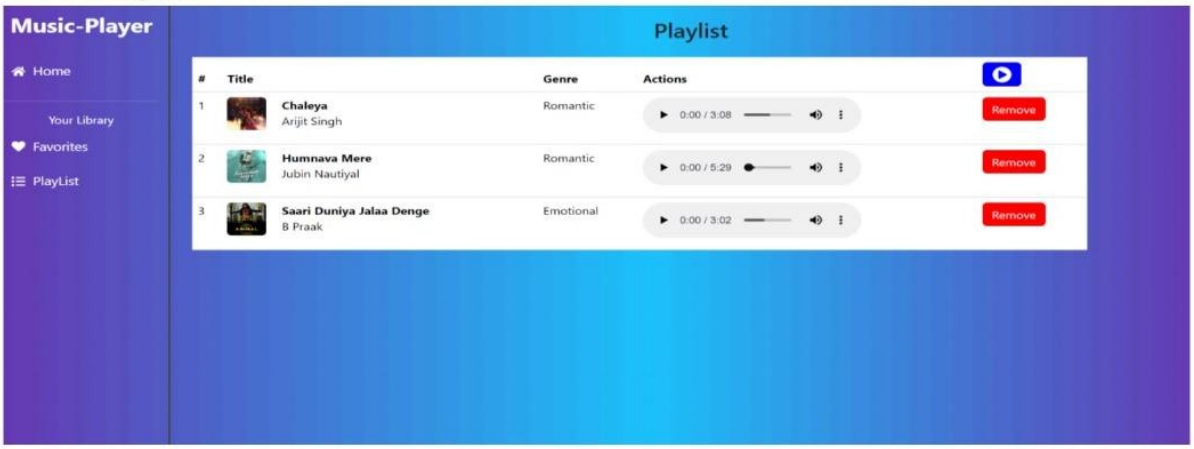

## ✔Search Functionality:

📌 Description: Showcase the real-time search feature where users type a query, and matching songs appear dynamically.



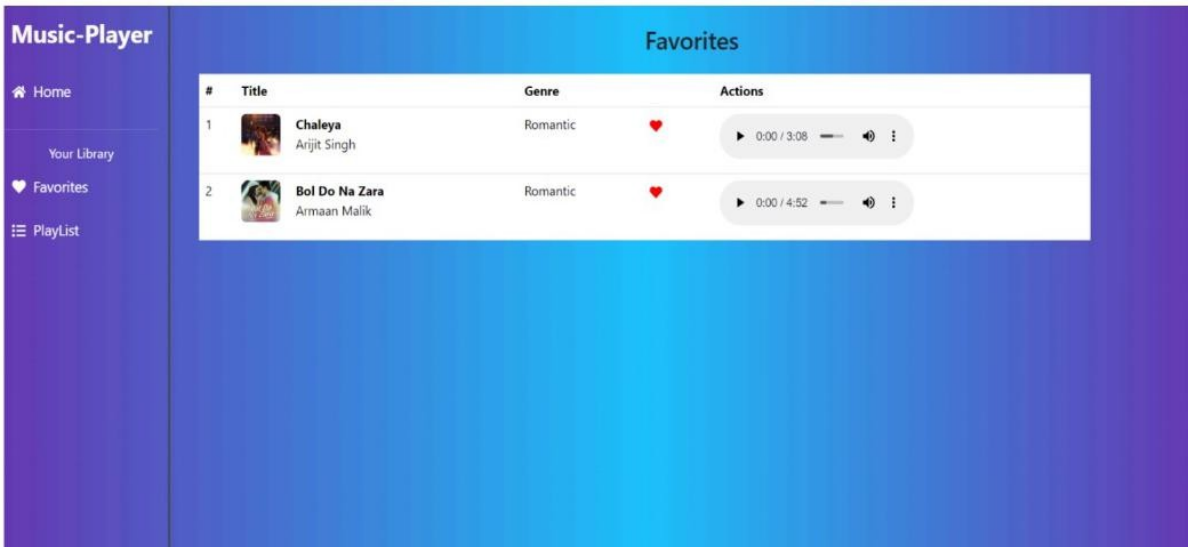
✔Playlist Management:

📌 **Description:** Demonstrate how users can create, edit, and delete playlists.




✔Favourites List in Rhythmic Tunes:

📌 **Description:** The Favorites List in Rhythmic Tunes allows users to save and manage their favorite songs, albums, or playlists for quick access.


Music-Player
Home
Your Library
Favorites
PlayList
Favorites
# Title Genre Actions
1 Chaleya Arijit Singh Romantic 0:00 / 3:08
2 Bol Do Na Zara Armaan Malik Romantic 0:00 / 4:52

# 10.STYLING:

## Styled Components

📌 A CSS-in-JS library that allows styling components directly within JavaScript files.

- **Key Features:**

* **Scoped Styles** – Styles are only applied to specific components.
* **Dynamic Styling** – Supports props-based styles.
* **No Additional CSS Files** – Everything is encapsulated in the component.

- **CSS Preprocessor Used**
  - **SASS/SCSS**

📌 A powerful preprocessor that extends CSS capabilities with variables, nesting, mixins, and inheritance.

- **Key Features:**

* **Reusable Variables** – Define colors, fonts, and sizes once.
* **Nesting** – Write styles in a structured way.
* **Mixins** — Reuse styles without repetition.

# 11.TESTING:

# TESTING STRATEGY:

To ensure reliability, performance, and a bug-free user experience, Rhythmic Tunes follows a structured testing strategy that includes:

- **Unit Testing** – Verifying individual components work as expected.
- **Integration Testing** — Ensuring components interact correctly.
- **End-to-End (E2E) Testing** – Testing the full user flow.

## 1.Unit Testing (Component-Level)

📌 **Purpose:** Tests**individual React components** to verify correct rendering, props, and behavior.

📌 **ToolUsed: Jest + React Testing Library**

## 2 . Integration Testing (Component Interactions)

📌 **Purpose:** Tests how multiple **components interact** together (e.g., forms, modals, API calls).

📌 **ToolUsed: Jest + Mock Service Worker (MSW)**

### 3.End-to-End (E2E) Testing (Full User Flow)

📌 **Purpose:** Simulates real-world **user interactions** from start to finish.

📌 **ToolUsed: Cypress / Playwright**

## CODE COVERAGE:

Ensuring adequate test coverage for rhythmic tunes requires a combination of analytical, software-based, and auditory techniques. Here are some key tools and techniques used:

### 1. Automated Audio Analysis Tools

- **Sonic Visualiser**: Helps visualize and analyses rhythm structures.
- **Essentia (by MTG/UPF)**: An open-source library for extracting rhythmic and tonal features.
- **Audio**: Detects beats, tempo, and onset points to ensure rhythmic consistency.

### 2. Manual and Computational Beat Tracking

- **Tap-Tempo Method**: Manually tapping beats

to verify tempo consistency.

- **Dynamic Time Warping (DTW)**: Compares rhythmic sequences for alignment.
- **Onset Detection Algorithms**: Identifies key beats and note attacks.

### 3. MIDI-Based Testing

- **MIDI Sequencers (e.g., Ableton Live, FL Studio, Logic Pro)**: Allows precise control and visualization of rhythmic elements.
- **MIDI Validation Scripts**: Detects timing deviations and quantization errors.

### 4. Spectrogram and Waveform Analysis

- **Waveform Editors (Audacity, Adobe Audition)**: Helps detect rhythmic inconsistencies.
- **Spectral Flux Analysis**: Identifies energy variations in beats.

### 5. Human Listening Tests

- **A/B Testing**: Compares different rhythmic variations for consistency.
- **Professional Drummers/Musicians Review**: Ensures a human feel in the rhythm.

### 6. Machine Learning & AI Techniques

- **Deep Learning Models (e.g., Magenta, Wave2Midi2Wave)**: Analyses and generates rhythm patterns.
- **Pattern Recognition Algorithms**: Detects anomalies in rhythmic sequences.

7. **Quantization and Grid Analysis**

- **DAW Quantization Tools**: Aligns notes to a rhythmic grid.
- **Swing and Groove Templates**: Ensures natural feel without mechanical stiffness.

## 12.SCREENSHOTS OR DEMO:

https://drive.google.com/file/d/1th2TWlfbIzdjLW29qASKQuKoGVBKMIJ7/views=sharing

## 13.KNOWN ISSUES:

The issues faced in rhythmic tunes are:

### 1. Transitions Between Sections:

Musicians often struggle with maintaining

rhythmic accuracy during transitions, such as moving from a verse to a chorus or changing tempos. These points can lead to timing issues and disrupt the flow of a performance.

## 2. Complex Rhythmic Patterns:

Certain rhythms, especially those with syncopation or irregular timing, can be difficult to perform accurately. Research using Steve Reich's "Clapping Music" has shown that both the inherent complexity of a rhythm and the transitions between patterns contribute to performance challenges.

## 3. Cultural Variations in Rhythm Perception:

Rhythm perception and production can vary significantly across different cultures. Projects that incorporate diverse musical traditions may face challenges in understanding and integrating these cultural differences.

### 4. Age-Related Declines in Timing Abilities:

As individuals age, there can be declines in timing and executive functions, affecting rhythm perception and production. Engaging in rhythmic musical activities has been suggested as a way to strengthen connectivity between brain networks associated with these functions.

### 5. Synchronization in Networked Performances:

In networked music performances, achieving synchronicity and tempo stability can be challenging, especially under conditions of high network latency. Studies have explored the use of global metronomes and signal source-panning to mitigate these issues.

## 14.FUTURE ENHANCEMENT:

Future enhancements in Rhythmic Tunes can focus on improving user experience, personalization, and advanced playback features. Here are some key areas for enhancements:

## 1.AI-Powered Personalized Recommendations:

- ❖ Smart Playlists: AI-based suggestions based on user listening habits.
- ❖ Mood-Based Playlists: Auto-generate playlists based on mood detection.
- ❖ Song Similarity Engine: Suggest similar songs based on tempo, genre, or user preferences.

## 2. Advanced Playback Features:

- ❖ Real-time Lyrics Sync: Show synchronized lyrics while a song plays.
- ❖ Equalizer & Audio Effects: Custom bass, treble, and surround sound settings.
- ❖ Crossfade & Gapless Playback: Smooth transitions between songs.
- ❖ Offline Mode: Download songs for offline listening.

## 3.Enhanced User Experience (UX/UI):

- ❖ Dark & Light Mode Toggle: Improved theme customization.

- ❖ Gesture Controls: Swipe gestures for skipping, play/pause, and volume control.
- ❖ Mini Player Mode: A floating mini-player for background play.

## 4.Social & Community Features:

- ❖ Collaborative Playlists: Users can create & share playlists with friends.
- ❖ Live Listening Parties: Sync music playback with friends in real time.
- ❖ User Profiles & Activity Feed: Display favorite tracks, playlists, and recent listens.

## 5. Integration with Smart Devices & Platforms:

- ❖ Smart Home Integration: Play music using Alexa, Google Home, or Siri.
- ❖ Wearable Support: Control music from smartwatches & fitness bands.
- ❖ CarPlay & Android Auto: Seamless music playback in cars.

## 6.Blockchain & NFT Music Integration:

- ❖ NFT-Based Music Ownership: Artists can sell

unique digital tracks as NFTs.

- ❖ Royalty Tracking for Artists: Blockchain-powered fair revenue distribution.

## 7.Multi-Cloud Streaming & High-Quality Audio:

- ❖ Adaptive Streaming: Adjusts quality based on internet speed.
- ❖ Res Audio Support: Loss less audio streaming for audiophiles.