RHYTHMIC TUNES

Project submitted for the partial fulfilment of the requirements for the course

<SEC 160: Full Stack Development>

Offered by the

Department Computer Science and Engineering

School of Engineering and Sciences

Submitted by

1) Mohammed Abdul Adam Akmal, AP23110010471

2) Y V S Nikhil, AP23110010453

3) K Vinay Reddy, AP23110010473

4) Shaik Sameer, AP23110010985

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

December, 2025

Contents:

1. Introduction

Step into a world of seamless melody and rhythm with **RhythmicTunes** – a modern, feature-rich music streaming web application built with the power of React.js. Designed to deliver a complete auditory experience, RhythmicTunes merges aesthetic elegance with robust functionality, redefining how users discover, organize, and enjoy their favourite tracks.

Crafted for music lovers and audiophiles alike, this application integrates a sophisticated audio engine with a responsive, intuitive user interface. From exploring top charts to curating personal playlists, RhythmicTunes offers a polished journey through sound. At its core beats React.js, ensuring a dynamic, single-page application experience where every transition is smooth, and every beat drops on time.

2. Scenario-Based Intro

Imagine it's Friday evening. You launch **RhythmicTunes** on your laptop. The application welcomes you with a personalized dashboard displaying "Popular Right Now" hits and your "Recently Added" tracks.

You feel like exploring, so you head to the **Browse** page to check out the "Top Charts." You find a new track, "*Ethereal*," and hit the play button. Instantly, the persistent **Audio Player** at the bottom of the screen comes alive, streaming the audio without a stutter. As the song plays, you navigate to the **Search** tab to find the artist "Txmy". You decide to save this track for later, so you click the "Heart" icon, adding it to your **Liked Songs** collection. Finally, you create a new private playlist and drag the volume slider up to immerse yourself fully. RhythmicTunes handles every interaction—playback, routing, and database updates—in real-time.

3. Target Audience

- **Music Enthusiasts:** Users who want a clean, ad-free environment to organize playlists and discover new genres.
- **Frontend Developers:** Engineers looking to study a reference implementation of a complex React application involving media handling, authentication, and global state management.
- **UI/UX Designers:** Designers interested in seeing how **Shadcn/ui** and **Radix UI** primitives can be composed to build accessible, dark-mode-first interfaces.

4. Project Goals and Objectives

The primary goal of **RhythmicTunes** is to offer a seamless platform for music enthusiasts, facilitating an immersive and personalized auditory experience. Our objectives include:

- **User-Friendly Interface:** Develop an intuitive interface that enables users to browse, play, and organize their music effortlessly, promoting a smooth and engaging listening environment.
- **Comprehensive RhythmicTunes:** Provide robust features for organizing and managing music content, including advanced search options for streamlined song discovery, playlist creation, and library structuring.
- **Full-Stack Simulation:** To demonstrate CRUD (Create, Read, Update, Delete) capabilities by interacting with a mock REST API (json-server) for managing users, playlists, and song data.
- **Modern Tech Stack:** Leverage cutting-edge web development technologies, such as **React.js** and **Vite**, to ensure an efficient and enjoyable user experience while navigating and interacting with the streaming application. This ensures that users can seamlessly discover, curate, and enjoy music using the latest advancements in web development tools.

5. Key Features

RhythmicTunes is engineered to provide a robust and immersive listening experience, mirroring the functionality of industry-standard streaming platforms.

- **Advanced Audio Engine:** At the heart of the application is a full-featured music player that supports play, pause, skip, and seek functionality. It features a **Smart Queue System** that intelligently queues songs based on the user's current context (e.g., an album, a playlist, or search results). The player includes gapless playback that persists across page navigation, along with Shuffle Mode.
- **Comprehensive Music Discovery:** Users can explore a library of music through dedicated **Browse** and **Artist** pages. The application includes a powerful search functionality that filters songs, artists, and albums in real-time, alongside a genre-based browsing system.
- **Personalized Library Management:** The platform empowers users to curate their own musical journey. Users can create and manage custom playlists, "Like" their favorite tracks for quick access, and view their **Recently Played** history (tracking the last 50 songs).
- **Dynamic User Profiles:** The application features dynamic user profiles that display real-time music statistics, such as total liked songs, created playlists, and estimated listening time analytics.
- **Modern & Adaptive UX:** Designed with a "Mobile-First" approach, the interface is fully responsive across desktop, tablet, and mobile devices. It includes a collapsible sidebar for adaptive navigation and a persistent **Dark/Light Theme** toggle that saves user preferences.

6. PRE-REQUISITES

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.

  - Create a new React app:

    ```
    npm create vite@latest
    ```

    Enter and then type project-name and select preferred frameworks and then enter

  - 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.

- Backend Simulation Tools:

  Unlike a standard static site, RhythmicTunes requires a mock backend to function.

  ```
  npm run server
  ```

  launches the API at http://localhost:3001.

  JSON Server: This tool is used to simulate a full REST API using the **db.json** file. It must be running concurrently with the frontend to serve user and song data.

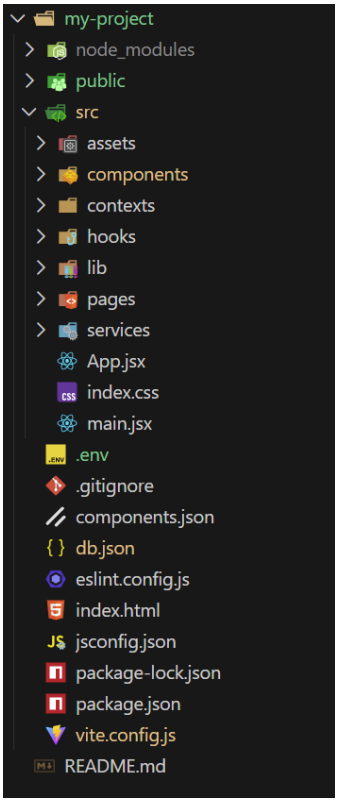
- **HTML, CSS, and JavaScript**: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

- **ES6+ Syntax:** Knowledge of modern JavaScript features (Arrows functions, Destructuring, Async/Await) is essential for working with the service layers.

- **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.
  - 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.
  - Visual Studio Code: Download from https://code.visualstudio.com/download
  - Sublime Text: Download from https://www.sublimetext.com/download
  - WebStorm:Download from https://www.jetbrains.com/webstorm/download

7. Project Structure



PROJECT FLOW
Demo & Live Application:

- **Link**: https://rhythmic-tunes.vercel.app
- **Repository**: https://github.com/akmalmohammed-1/rhythmictunes

MILESTONE 1: PROJECT SETUP AND CONFIGURATION
Technology Stack Installation

Frontend Dependencies:

- **React.js 18** – Core framework with modern hooks
- **React Router DOM** – Client-side routing
- **Tailwind CSS** – Utility-first CSS framework
- **Shadcn/ui** – High-quality component library
- **Lucide React** – Icon library
- **Axios** – HTTP client for API requests

Development Tools:

- **Vite** – Fast build tool and dev server
- **ESLint** – Code linting and formatting
- **PostCSS** – CSS processing

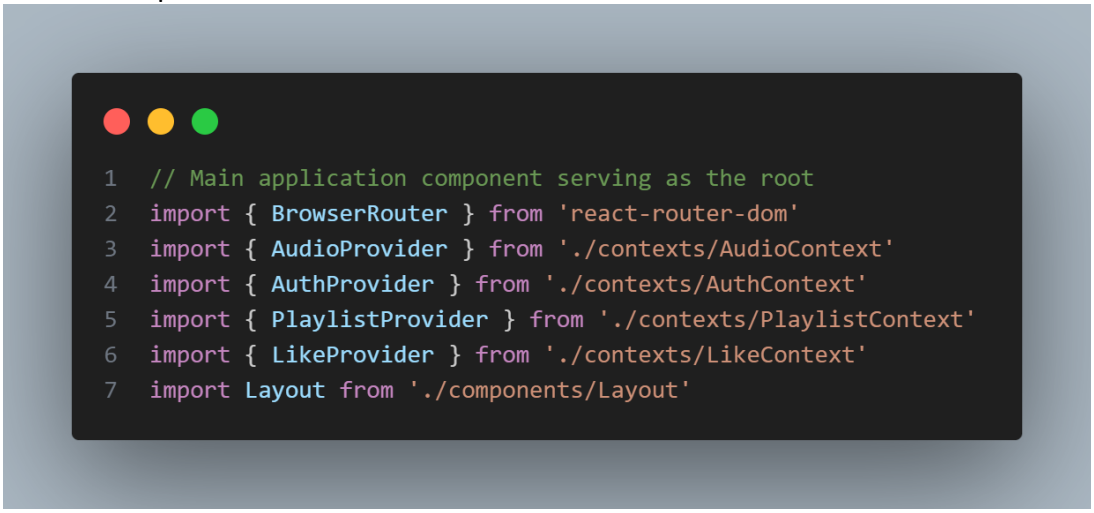
Backend:

- **JSON Server** – RESTful API simulation
- **Railway** – Backend deployment platform

MILESTONE 2: WEB DEVELOPMENT

1. Setup React Application:

Code Description:



```
// Main application component serving as the root
import { BrowserRouter } from 'react-router-dom'
import { AudioProvider } from './contexts/AudioContext'
import { AuthProvider } from './contexts/AuthContext'
import { PlaylistProvider } from './contexts/PlaylistContext'
import { LikeProvider } from './contexts/LikeContext'
import Layout from './components/Layout'
```

Context Providers

- Wraps the application with multiple context providers for state management
- Ensures all components have access to shared state
- Maintains a clear provider hierarchy

Router Setup

- BrowserRouter enables client-side routing

Layout Component

- Contains the main application layout and routing logic

Provider Hierarchy

- Guarantees shared state availability across the entire component tree

2. Context Architecture

AudioContext Provider

Code Description:

```
// Audio reducer for state management
function audioReducer(state, action) {
  switch (action.type) {
    case audioActions.PLAY_SONG:
      return {
        ...state,
        currentSong: action.payload.song,
        isPlaying: true,
        currentIndex: action.payload.index || 0,
        queue: action.payload.queue || [action.payload.song]
      }
    case audioActions.PAUSE_SONG:
      return { ...state, isPlaying: false }
    // ... other cases
  }
}

// Main play function with recently played tracking
const playSong = async (song, queue = [song], index = 0) => {
  dispatch({ type: audioActions.PLAY_SONG, payload: { song, queue, index }})

  if (user?.id && song?.id) {
    await userService.addToRecentlyPlayed(user.id, song.id)
  }
}
```

- Creates centralized audio state management using useReducer
- Manages current song, playback status, queue, volume, and repeat modes
- Provides functions for play, pause, skip, shuffle, and volume control
- Handles audio events like loading, time updates, and errors
- Integrates with HTML5 Audio API for actual playback

PlaylistContext Provider

Code Description:

```
const addSongToPlaylist = async (playlistId, songId) => {
  try {
    const updatedPlaylist = await userService.addSongToPlaylist(playlistId, songId)
    setPlaylists(prev => prev.map(p => p.id === playlistId ? updatedPlaylist : p))
    return true
  } catch (error) {
    console.error('Failed to add song:', error)
    return false
  }
}

const createPlaylist = async (playlistData) => {
  const newPlaylistData = {
    ...playlistData,
    user_id: user.id,
    created_at: new Date().toISOString()
  }
  const createdPlaylist = await userService.createPlaylist(newPlaylistData)
  setPlaylists(prev => [...prev, createdPlaylist])
}
```

- Manages user playlists and playlist operations
- Provides CRUD operations for playlist management
- Handles playlist creation, deletion, and song addition/removal
- Syncs playlist data with backend API
- Manages playlist state across the application

AuthContext Provider

Code Description:

```
const login = async (email, password) => {
  const users = await userService.getAllUsers()
  const foundUser = users.find(u => u.email === email)

  if (!foundUser || foundUser.password !== password) {
    throw new Error('Invalid credentials')
  }

  const token = `token_${foundUser.id}_${Date.now()}`
  localStorage.setItem('rhythmic-tunes-token', token)
  localStorage.setItem('rhythmic-tunes-user', JSON.stringify(foundUser))
  setUser(foundUser)
}
```

- Manages user authentication state and session persistence
- Provides login, logout, and signup functionality
- Stores user data in localStorage for session persistence
- Handles protected routes and user preferences
- Integrates with backend API for user management

3. Component Architecture

Layout.jsx Component

Code Description:

- Serves as the main application wrapper component
- Implements responsive sidebar navigation using Shadcn/ui components
- Handles route rendering with React Router
- Supports responsive design for mobile and desktop
- Integrates AudioPlayer component for persistent music playback

AudioPlayer.jsx Component

Code Description:

```jsx
// Progress bar with seeking
const handleProgressChange = (value) => {
  const newTime = (value[0] / 100) * duration
  seekTo(newTime)
}

// Main play/pause button
<Button onClick={togglePlayPause} disabled={isLoading}>
  {isLoading ? (
    <div className="animate-spin rounded-full border-2 border-current border-t-transparent" />
  ) : isPlaying ? (
    <Pause className="h-5 w-5" />
  ) : (
    <Play className="h-5 w-5" />
  )}
</Button>
```

- Implements a full-featured music player interface
- Uses AudioContext for playback control and state management
- Provides minimizable/expandable player interface
- Handles volume control, progress tracking, and playback modes
- Offers responsive, touch-optimized controls for mobile

Profile.jsx Component

```jsx
useEffect(() => {
  const loadUserStats = async () => {
    if (!user?.id) return

    const freshUserData = await userService.getUserById(user.id)
    const recentlyPlayedCount = freshUserData?.recently_played?.length || 0

    setUserStats({
      likedCount: likedSongs?.length || 0,
      playlistCount: playlists?.length || 0,
      recentlyPlayedCount,
      estimatedHours: Math.round(recentlyPlayedCount * 3.5) // 3.5 min per song
    })
  }

  loadUserStats()
}, [user, likedSongs, playlists])
```

Code Description:

- Displays dynamic user statistics and account information
- Integrates with multiple contexts (Auth, Likes, Playlist)
- Calculates real-time listening analytics
- Shows user activity metrics and account status
- Uses environment-based data fetching for accurate statistics

4. Service Layer Architecture

API Service (api.js)

Code Description:

```
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('rhythmic-tunes-token')
  if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }
  return config
})
```

- Configures Axios base instance with environment-specific URLs
- Implements request/response interceptors for authentication
- Handles error management and token refresh
- Manages API request/response formatting

User Service (userService.js)

Code Description:

```
// musicService.js - Enhanced song fetching
const getSongsWithDetails = async () => {
  const [songs, artists] = await Promise.all([
    api.get('/songs'),
    api.get('/artists')
  ])

  const artistMap = artists.data.reduce((map, artist) => {
    map[artist.id] = artist
    return map
  }, {})

  return songs.data.map(song => ({
    ...song,
    artist_name: artistMap[song.artist_id]?.name || 'Unknown Artist'
  }))
}
```

- Provides user-related API operations (CRUD)
- Manages user preferences and settings
- Handles recently played songs tracking
- Implements playlist management for users
- Integrates with authentication system

5. Advanced Features Implementation

Dynamic Statistics

- Real-time calculation of user listening metrics
- Integration with multiple context providers for live data
- Environment variable–based API configuration
- Responsive data visualization

Responsive Audio System

- HTML5 Audio API integration with React contexts
- Cross-page audio persistence

- Advanced playback controls (shuffle, repeat, queue management)
- Touch-optimized mobile controls

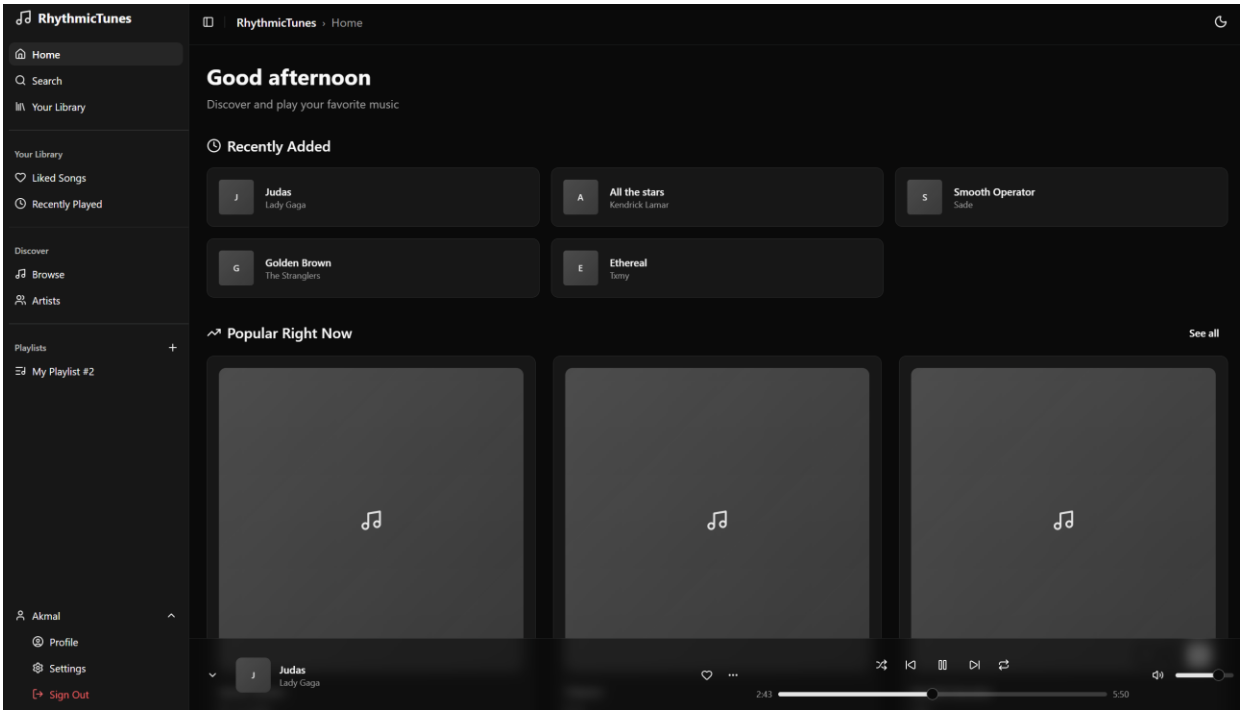
Modern UI/UX

- Dark/light theme toggle with persistence
- Responsive design using Tailwind CSS
- Smooth animations and transitions
- Accessible component design with ARIA labels

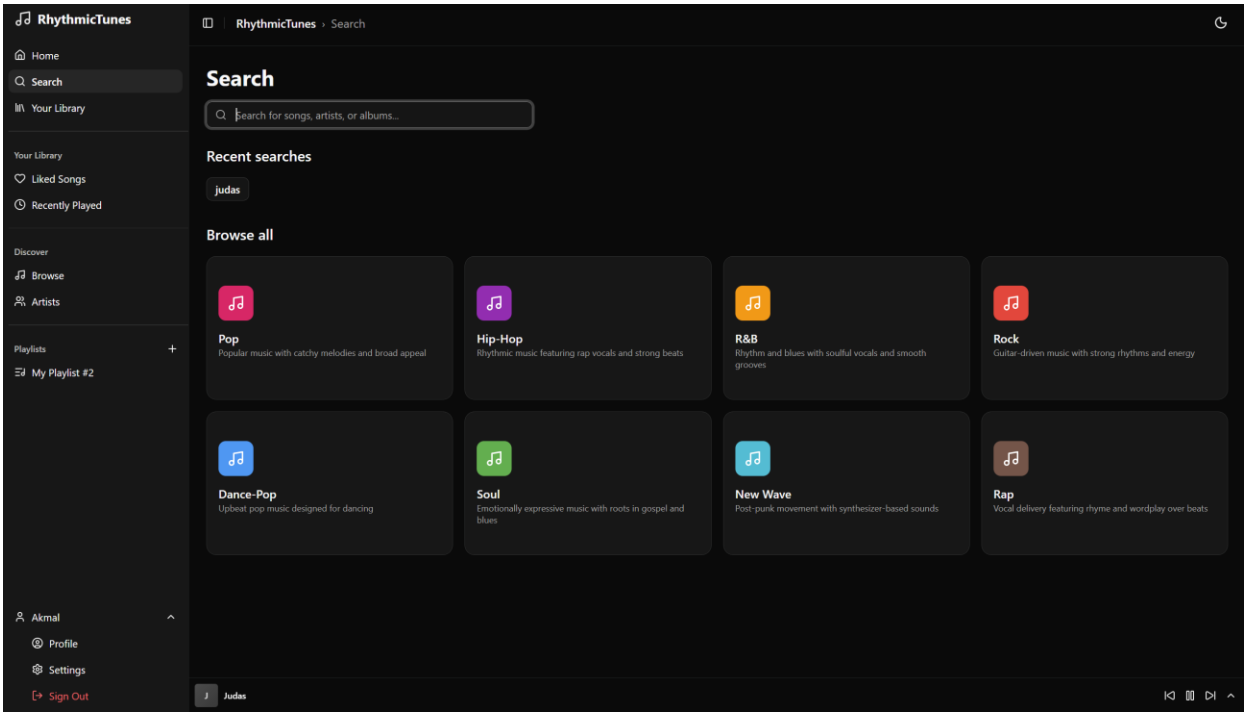
8.Project Execution

After completing the code, run the React application using the following commands:

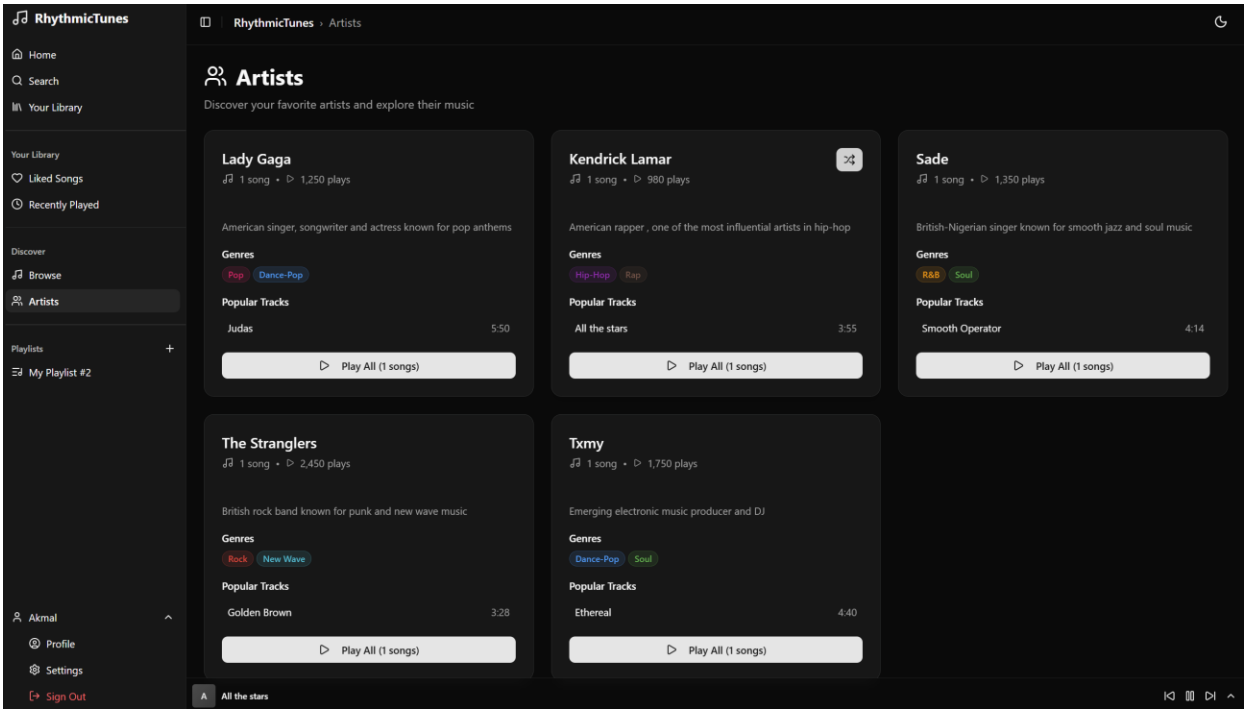
Here are some of the screenshots of the application.



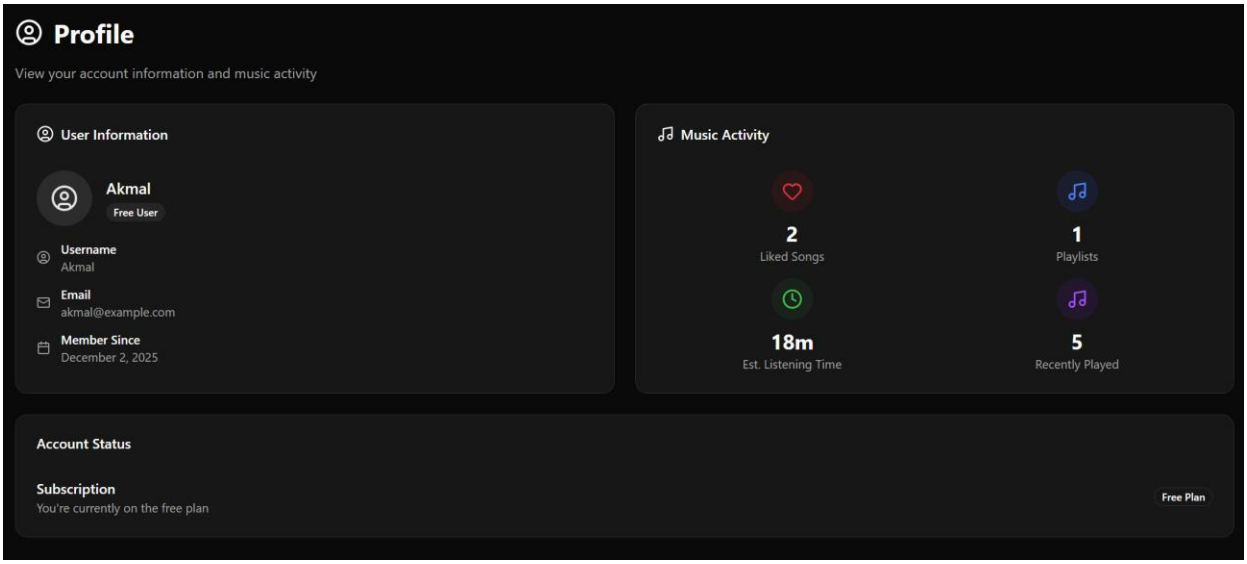
Search Page



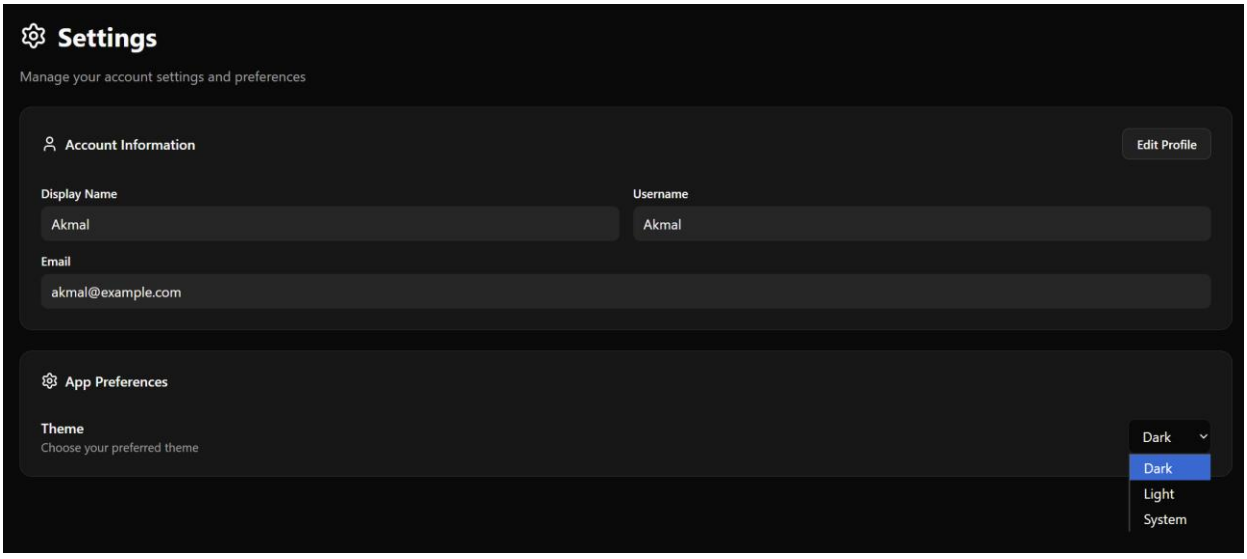
Artists Page



Profile page



Settings page



9.Project Demo link

https://drive.google.com/file/d/16K3OEqFVG8DpaXPPAzHACNN_in4YiFhs/view?usp=sharing