Tunify 🎵

Introduction:

Get into the rhythm of the future with **Tunify** – a sleek and immersive music streaming application crafted with the power of React.js. Designed for music enthusiasts, Tunify offers a seamless auditory experience, merging modern design with robust functionality.

Built for the modern listener, Tunify's React-based interface ensures a smooth and responsive journey through your favourite tunes. From discovering new genres to curating the perfect playlist, our platform provides a comprehensive musical ecosystem tailored to your unique taste.

The core of Tunify beats with React, a dynamic JavaScript library that powers our visually stunning and interactive interface. Whether you're on a desktop or mobile device, Tunify guarantees a consistent and enjoyable listening experience.

Scenario-Based Intro:

Imagine a typical evening. You want to unwind with some music. You open **Tunify**. As the application loads, you are greeted by a vibrant, Soft-themed interface that sets the mood. You navigate to the "Browse" section, where a curated list of songs awaits. You search for Chill Songs, and a list of soothing tracks appears. You hit play on the first song. The integrated player at the bottom of the screen comes to life, allowing you to control volume, skip tracks, or loop your favourite Song. You decide to save this track to your "Liked List" for later. With Tunify, every interaction is smooth, intuitive, and designed to keep you in the flow.

Target Audience:

Tunify is tailored for:

- **Music Enthusiasts**: People who love exploring new music and organizing their libraries.
- **Playlist Curators**: Users who enjoy creating custom playlists for every mood.

Project Goals and Objectives:

The primary goal of Tunify is to offer a premium music streaming experience. Our objectives include:

- **User-Friendly Interface**: Develop an intuitive UI that makes navigation and playback effortless.

- **Robust Player Functionality**: Implement a fully featured audio player with queue management, shuffle, and loop capabilities.
- **Modern Tech Stack**: Leverage React.js, Vite, and Tailwind to ensure high performance and a modern aesthetic.

Key Features:

- **Live Audio Player**: A persistent player that continues playing while you navigate the app, featuring Play/Pause, Next/Prev, Shuffle, and Loop controls.
- **Music Discovery**: Browse songs by categories and genres.
- **User Authentication**: Secure Login and Registration to save user preferences.
- **Personalized Library**: Create playlists and "Like" songs to build your personal collection.

PRE-REQUISITES:

Here are the key prerequisites for developing or running this application:

➢ **Node.js and npm**

Node.js is required to run the development environment.

Download: nodejs.org

➢ **React.js & Vite**

This project is built using React with Vite for a fast development experience.

Create a new app: npm create vite@latest

- **Navigate to the project directory**:

cd tunify/client

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.

- **Run the Mock Backend Server:**

Create the JSON extension file to store the site data. For example, “db.json”

To run this mock backend server:

npx json-server db.json

- **Dependencies:**

Key libraries used in this project:

- **react-router-dom:** For client-side routing.
- **tailwindcss:** For utility-first styling.
- **json-server:** For a mock backend API.
- **axios:** For making HTTP requests.
- **Frame-motion**: To make animations in the UI.

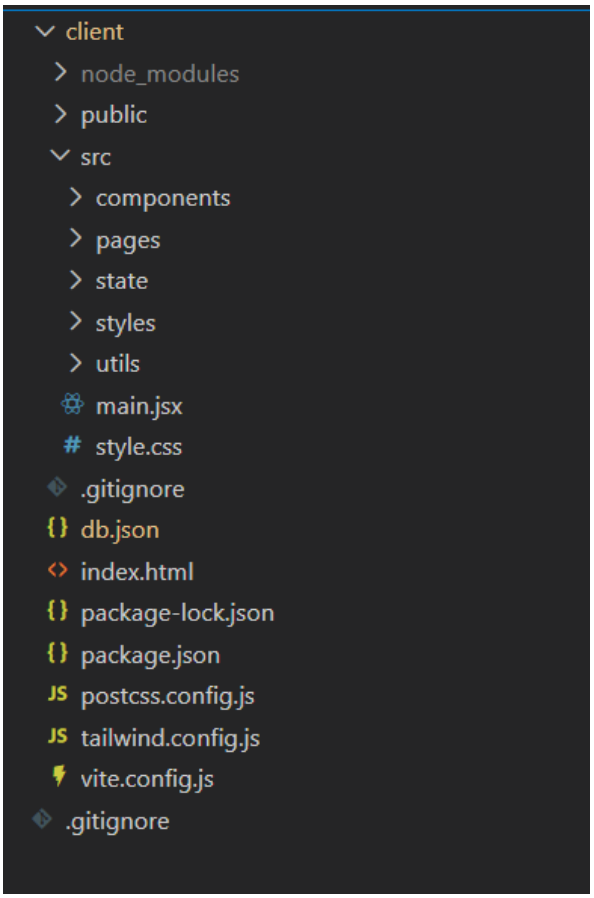
- **Version Control:** Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

  • 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

Project Structure:



The project is organized to promote maintainability and scalability:

- client/src/main.jsx: The entry point of the application. It handles the root rendering and sets up the global providers (AuthProvider, PlayerProvider) and the AppRouter.
- **client/src/components/:** Contains reusable UI components like Navbar, PlayerModal, SongCard, etc.
- **client/src/pages/:** Contains page-level components like HomePage, BrowsePage, LoginPage, etc.
- **client/src/state/:** Contains Context API files for global state management (AuthContext, PlayerContext).
- **client/src/utils/:** Utility functions and API configuration.

PROJECT FLOW: -

Project demo:

Before starting to work on this project, let's see the demo.

Demo link:

https://drive.google.com/file/d/1Jkbc54EFcRC5H0jQveXd28IdXHeNZEny/view?usp=drive_link

Use the code in:

https://github.com/ganesh-gonuguntla/tunify-react-frontend-app-course-project.git

Code Description:

main.jsx (Entry Point)

This file serves as the root of the React application.

- **Router**: It uses BrowserRouter to manage navigation.
- **Routes**: Defines paths for /login, /, /browse, /liked, etc., and handles protected routes (redirecting unauthenticated users to login).
- **Providers**: Wraps the application in AuthProvider and PlayerProvider to ensure authentication state and player state are accessible globally.

PlayerContext.jsx (State Management)

This context manages the entire audio playback logic.

- **State**: Tracks queue, currentIndex, isPlaying, volume, shuffle, and loop.
- **Audio Ref**: Uses a useRef to hold the Audio object, allowing direct control over playback without re-rendering unnecessarily.
- **Functions**:
    - playSongs(songs, startIndex): Replaces the queue and starts playing a new list of songs.
    - next () / prev(): Handles navigation through the queue, respecting shuffle mode.
    - play () / pause (): Controls the current playback state.
- **Effects**: useEffect hooks listen for audio events like ended (to auto-play the next song) and timeupdate (to update the progress bar).

AppRouter Component (in main.jsx)

- **Conditional Rendering**: Checks if a user is authenticated (isAuthed) to decide whether to show the Navbar and PlayerModal.
- **Route Guards**: Protects private routes like /profile and /liked by checking the user's auth status.

Project Execution:

To run the Tunify application locally:

1. **Install Dependencies**: Navigate to the client directory and install the necessary packages.

   cd client

   npm install

2. **Start the Backend Server**: Run the JSON server to mock the API.

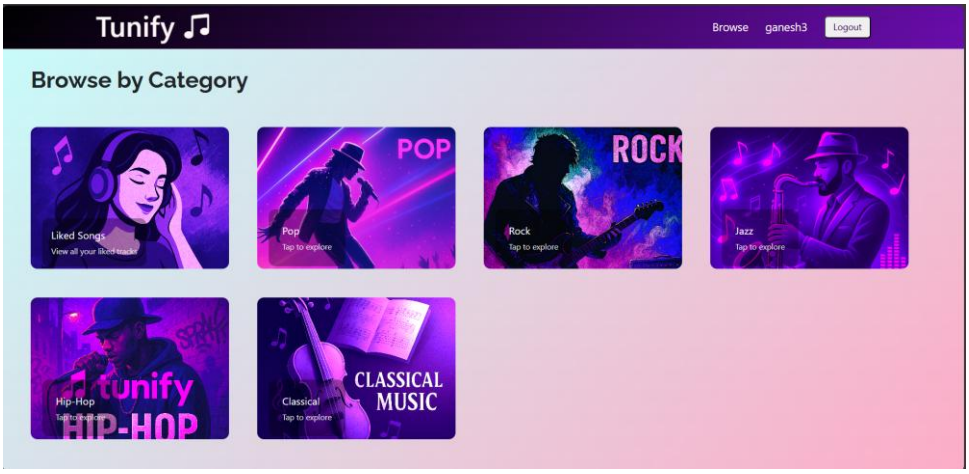
   npx json-server db.json

3. **Start the React App**: Launch the json-server(Mock DataBase).

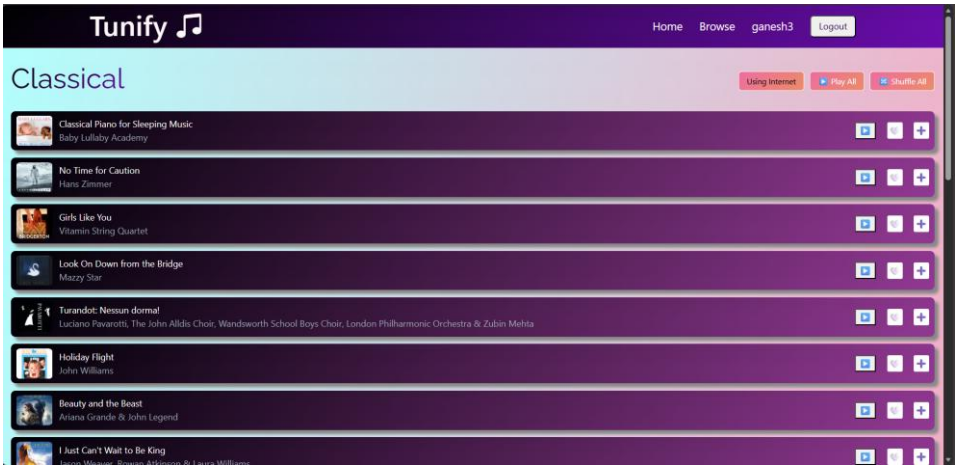
   npm run dev

4. **Access the App**: Open your browser and go to http://localhost:5173 (or the port shown in your terminal).

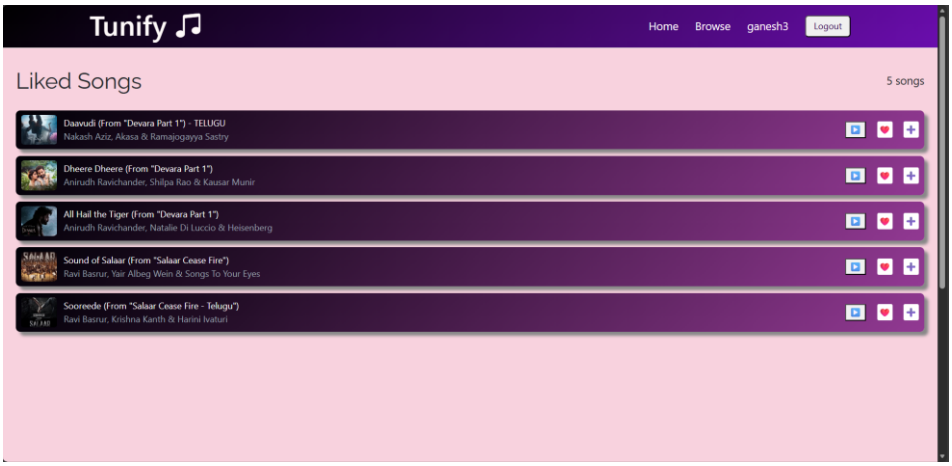
- **Home Page**:



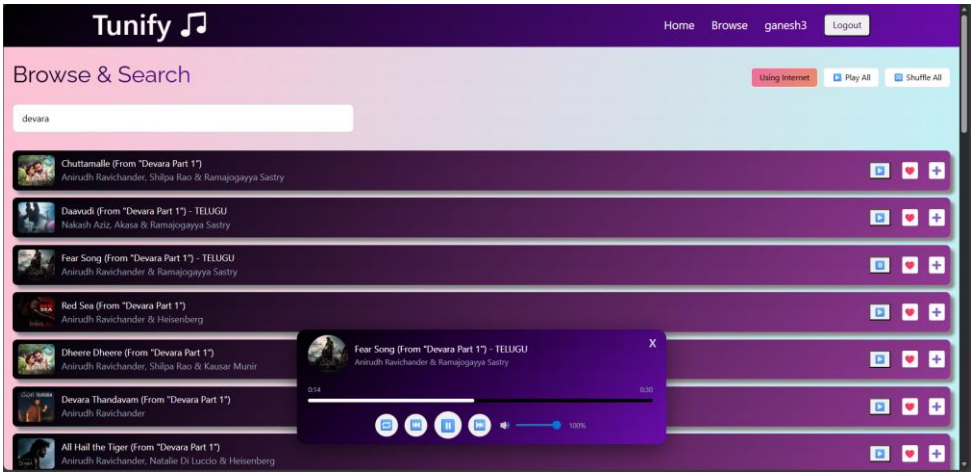
- **Category page:**



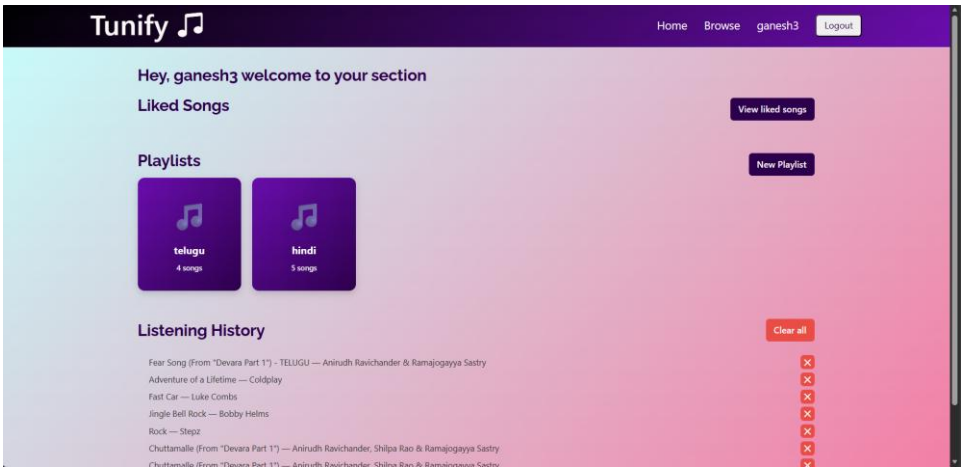
- **Liked Songs Page:**



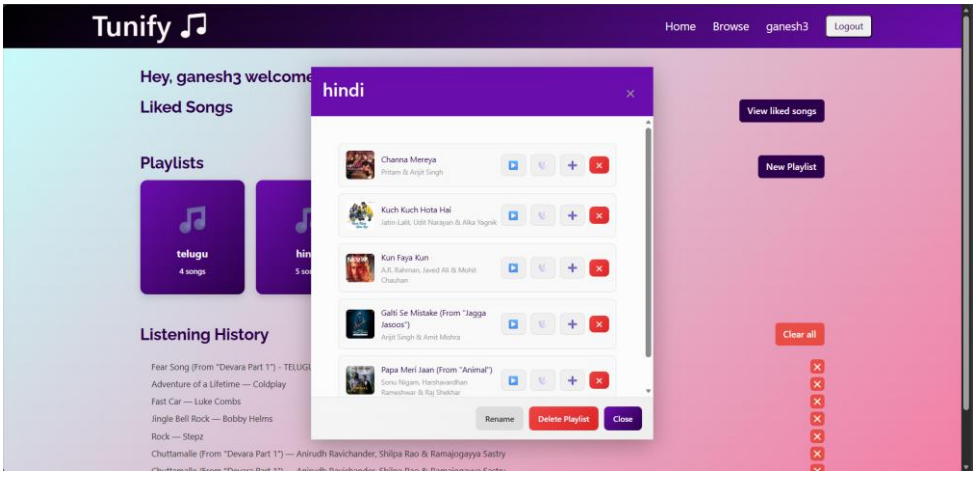
- **Song Player Modal Component and Browser Page:**



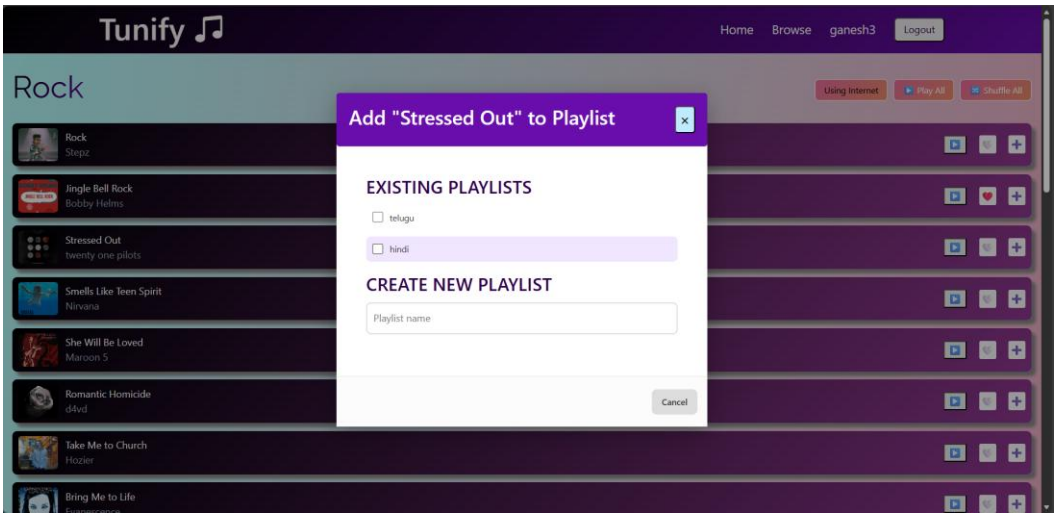
- **Profile Page (includes Playlists,Listening History and redirect button to Liked Page):**



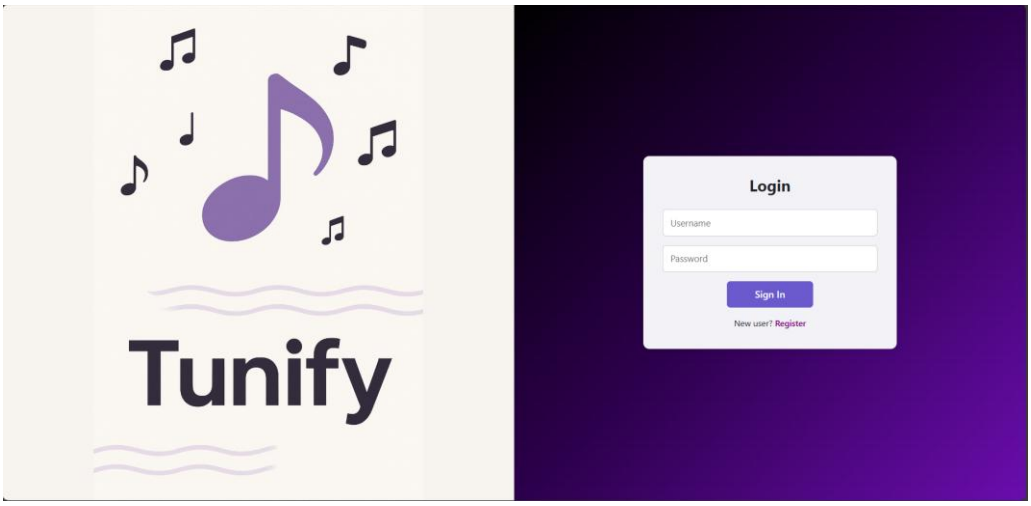
- **Playlist modal:**



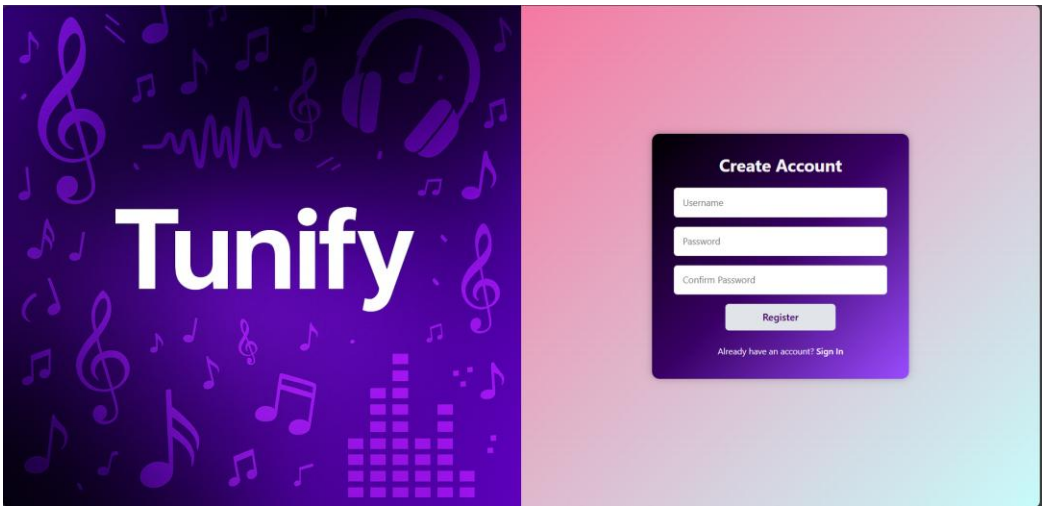
- **Add to Playlist Modal/ Create Playlist Modal:**



- **Login page:**



- **Registration Page:**



Project Demo Link:

https://drive.google.com/file/d/1Jkbc54EFcRC5H0jQveXd28IdXHeNZEny/view?usp=drive_link