**KONGU ENGINEERING COLLEGE **

**PERUNDURAI, ERODE-638060**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**2024-2025**

**ODD SEMESTER**

**20ITE03**

**USER INTERFACE DESIGN**

**A PROJECT REPORT**

##### **MOVIE SEARCH APP**

**Submitted By**

GOKUL K

**NAME OF THE CANDIDATE**

GOKUL K

**Register Number**

22ITR026

Index

|  |  |  |
| --- | --- | --- |
| S No | Title | Page Number |
| 1 | Rubrics for Assessment | 3 |
| 2 | About Project | 5 |
| 3 | Individual Components Description with  GitHub Link | 6 |
| 4 | Web Pages - Screenshot | 6 |
| 5 | Folder Structure -Screenshot | 9 |
| 6 | Coding | 10 |

Rubrics for Assessment

|  |  |  |
| --- | --- | --- |
| Name of Concepts | Type of Implementation  (Class/ Function) | Marks |
| Components based Design (2) |  |  |
| Multiple Components Integration(2) |  |  |
| Keys and List with map method-Dynamic Content Generation(2) |  |  |
| Filter and Search(2) |  |  |
| Authentication(2) |  |  |
| Props and State(2) |  |  |
| Conditional Rendering(2) |  |  |
| Event Handling(2) |  |  |
| Form-Multiple Fields (Min 6)(2) |  |  |
| Textbox(2) |  |  |
| Checkbox(2) |  |  |
| RadioButton(2) |  |  |
| Select (2) |  |  |
| File Option(2) |  |  |
| Validation  (Regular Expression and Required Field)(2) |  |  |
| Submission and Display(2) |  |  |
| Routing(2) |  |  |
| Hooks-UseEffect(4) |  |  |
| Hooks-UseRef(2) |  |  |
| Styling and User Interaction(30) [(0-10)- Poor, (11-20)- Average -(21-30)-Good ] with GitHub Presentation and Live Demo | |  |
| Project Report(15)[(0-5)- Poor, (6-10)- Average -(11-15)-Good ] | |  |
| Presentation(15)[(0-5)- Poor, (6-10)- Average -(11-15)-Good ] | |  |
| Total (100) | |  |

Faculty Incharge HoD/IT

**MOVIE SEARCH APP**

**About the Project**

The Movie Search App is a full-stack application that allows users to search for movies, view details, and explore a variety of movie-related information using The Movie Database (TMDB) API. The app is built with ReactJS for the frontend, which provides an interactive, user-friendly experience. For user authentication, Auth0 is integrated to allow secure login and registration. The app is designed with responsiveness in mind, adapting seamlessly across different devices.

**Key Features:**

* Movie search functionality with autocomplete suggestions.
* Detailed movie information, including cast, ratings, and trailers.
* Secure user authentication using Auth0.
* Clean and responsive design for optimal user experience on both desktop and mobile devices.

**Tools and Technologies Used**

1. **React + Vite**: Vite is used as the frontend tool to build and serve the React components, offering faster build times and hot-reloading. React’js component-based architecture helps create a dynamic and interactive user interface, providing a seamless experience for movie searches, ratings, and watchlist management.
2. **MongoDB Atlas**: A cloud-based NoSQL database, MongoDB Atlas is used to store user data, favorite movies, search history, and app configurations. Its scalability and flexibility make it a great choice for managing dynamic data such as movie information and user preferences.
3. **API**: The application uses RESTful APIs to ensure smooth communication between the frontend and backend. These APIs fetch real-time movie data from TMDB and manage user authentication, search results, and watchlists securely, allowing for modular backend operations and efficient data transfer.

**Individual Components Description**

1. **Login Component:**
   * Handles user authentication with Auth0, enabling login and registration functionality.
   * Manages user sessions to ensure secure access to app features.
2. **Search Bar Component:**
   * Provides an input field for users to type movie titles, using TMDB API for real-time search suggestions.
   * Supports autocomplete for a smoother user experience.
3. **Movie List Component:**
   * Displays a list of movies based on user search queries.
   * Each movie item shows a thumbnail, title, and basic info, and allows users to click for more details.
4. **Movie Detail Component:**
   * Shows detailed information about a selected movie, such as synopsis, ratings, and cast.
   * Integrates with TMDB API to fetch additional details, such as trailers and similar movie recommendations.

**GitHub Repository Link**

Find the complete source code and project documentation on GitHub:

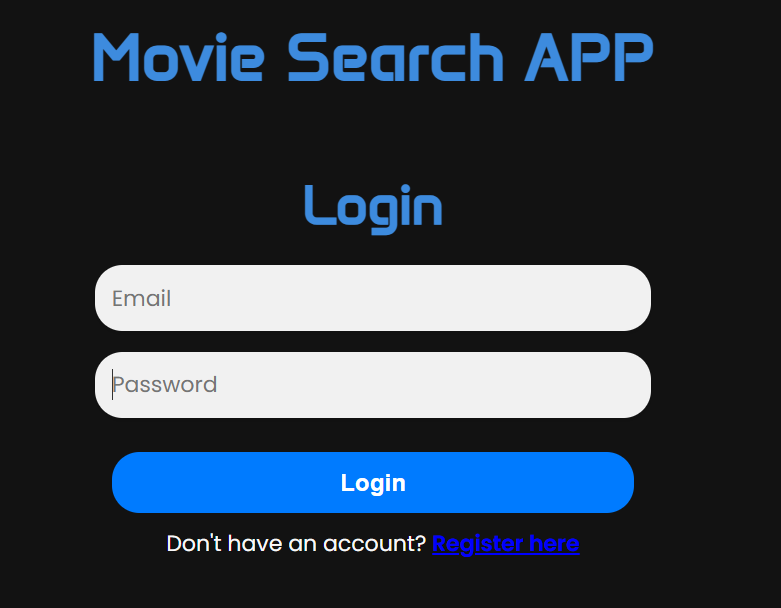
**Repository URL**: <https://github.com/gokulk-226/UID>

**Web Pages – Screenshots**

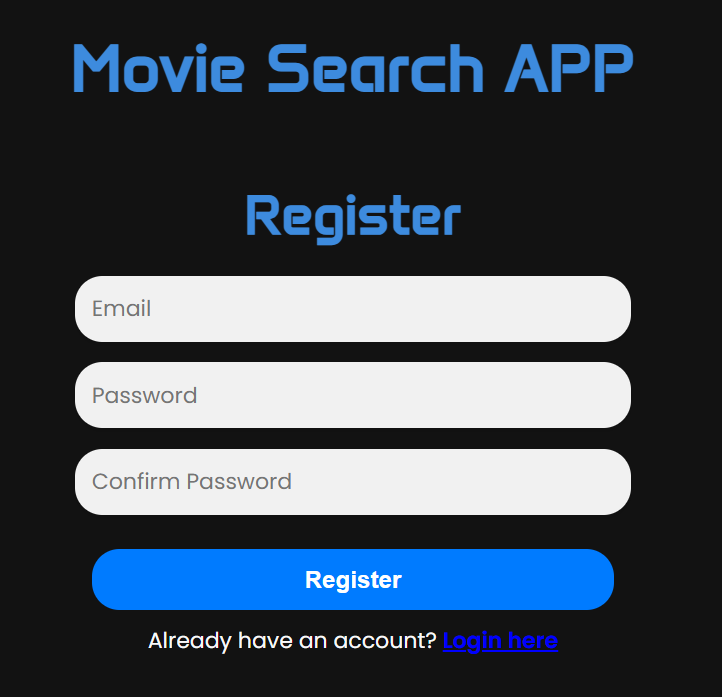
* Login Page: Displays Auth0 login options with a clean, accessible layout.
* Search Page: Main interface for searching movies, featuring an input field and responsive suggestions.

* Movie Detail Page: Displays detailed information about a selected movie, including trailer links, cast details, and similar movie suggestions.

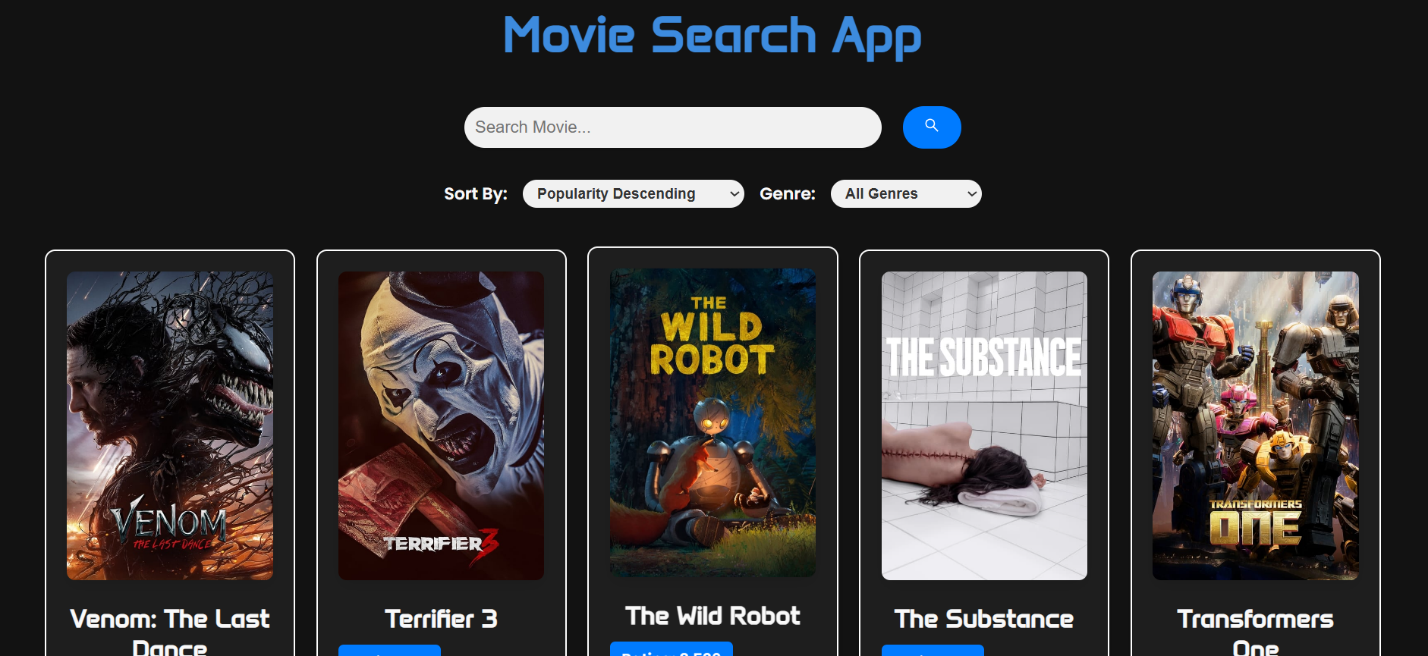
**LOGIN PAGE**

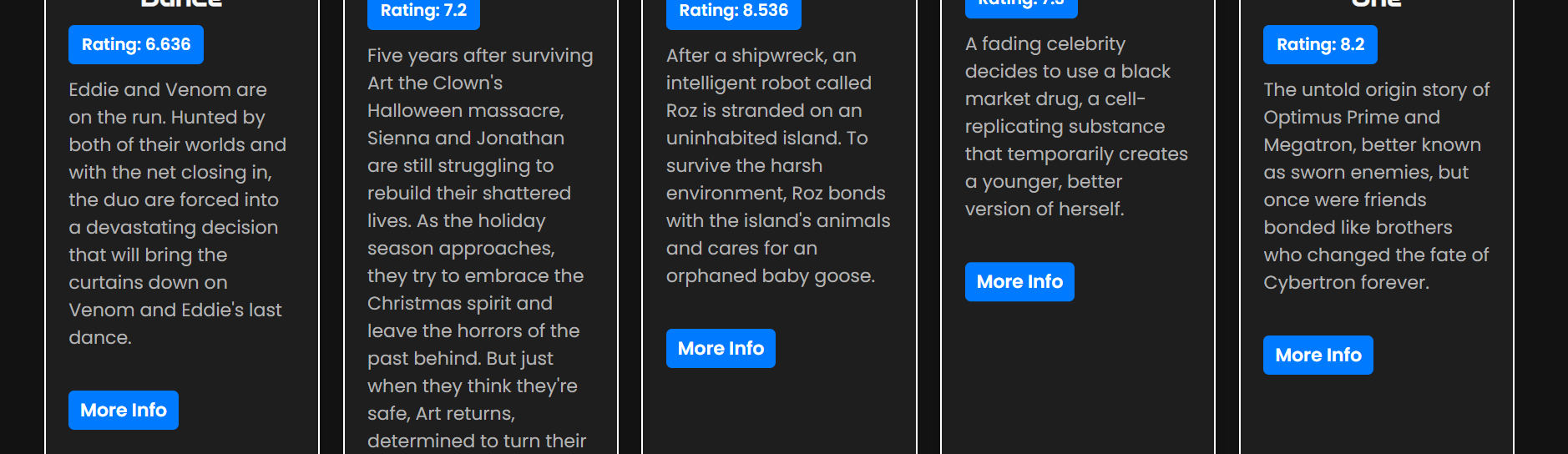


**REGISTER PAGE**

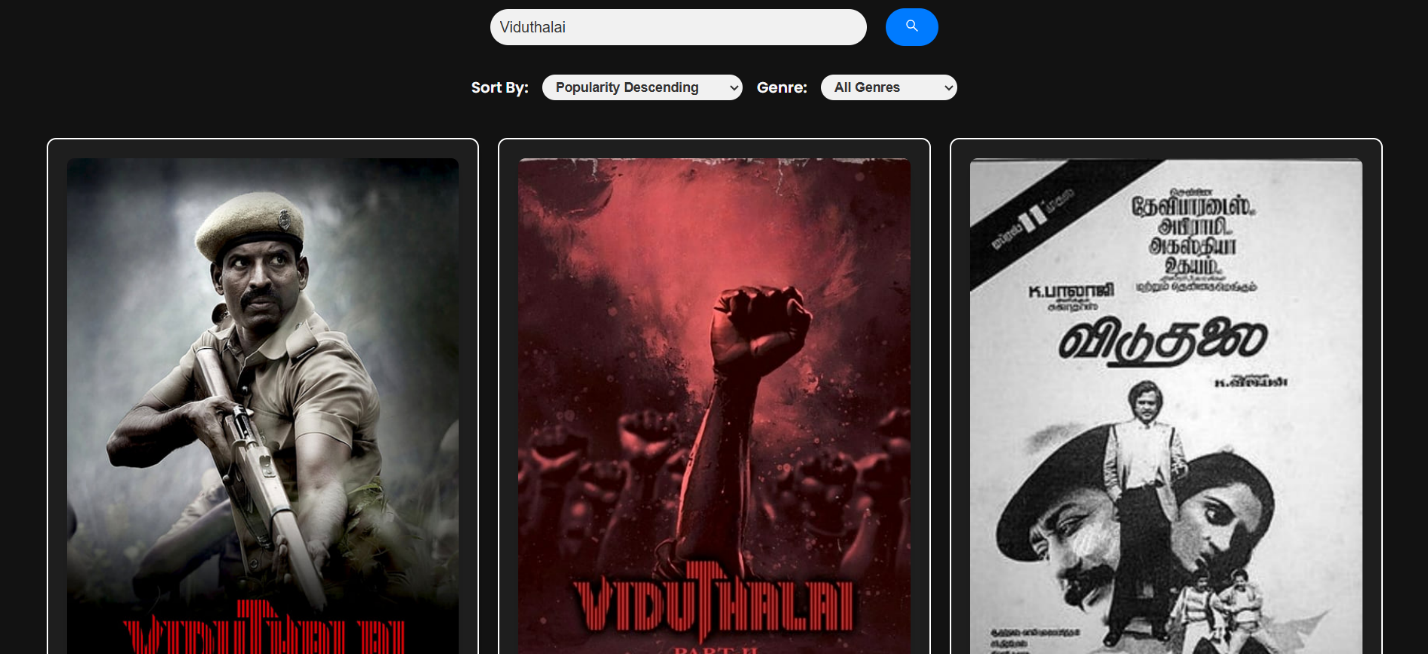


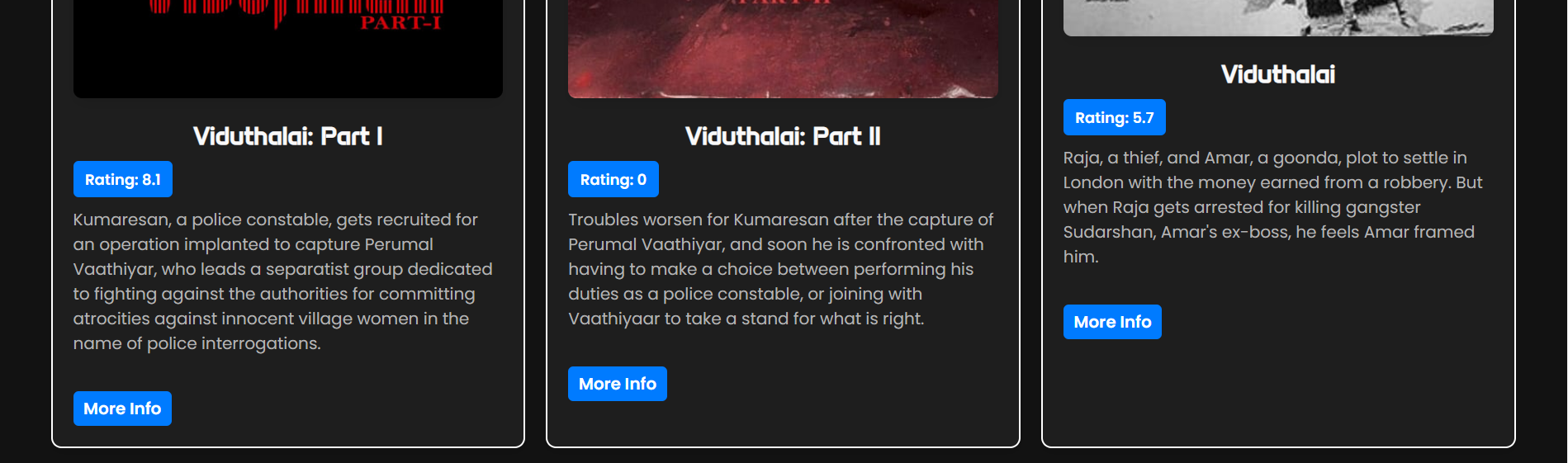
**MOVIE PAGE**





**MOVIE SEARCH PAGE**

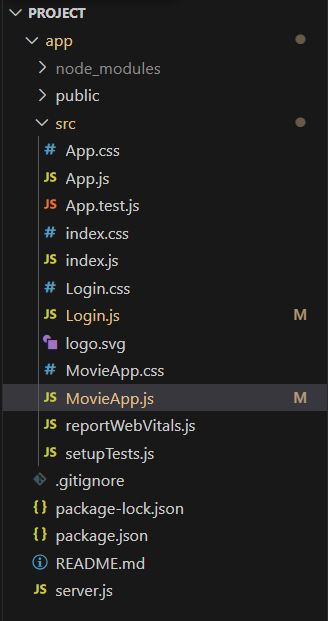




**Folder Structure – Screenshot**

Here’s the folder structure of the Movie Search App:

* 1. components
     + Login.js – Handles authentication logic.
     + MovieApp.js – Lists movie results.
  2. styles
  + Login.css – Styles for the login page.
  + App.css – Global styles for the application.
  + MovieApp.js-Styles for the MovieApp Page.
  1. App.js – Main app component that renders other components.
  2. index.js – Entry point of the application.



**Coding:**

// App.js

import React, { useState } from 'react';

import { BrowserRouter as Router, Route, Routes, Navigate } from 'react-router-dom';

import MovieApp from './MovieApp';

import Login from './Login';

export default function App() {

    const [userId, setUserId] = useState(null);

    return (

        <Router>

            <Routes>

                <Route

                    path="/"

                    element={<Navigate to={userId ? "/movies" : "/login"} />}

                />

                <Route

                    path="/login"

                    element={<Login setUserId={setUserId} />}

                />

                <Route

                    path="/movies"

                    element={userId ? <MovieApp userId={userId} /> : <Navigate to="/login" />}

                />

            </Routes>

        </Router>

    );

}

//Login.js

import React, { useState } from 'react';

import { useNavigate } from 'react-router-dom';

import './Login.css';

export default function Auth({ setUserId }) {

    const [isLogin, setIsLogin] = useState(true); // Toggle between login and register

    const [email, setEmail] = useState('');

    const [password, setPassword] = useState('');

    const [confirmPassword, setConfirmPassword] = useState('');

    const navigate = useNavigate(); // For programmatic navigation

    const handleAuth = async () => {

        if (!isLogin && password !== confirmPassword) {

            alert('Passwords do not match!');

            return;

        }

        const endpoint = isLogin ? 'login' : 'register';

        try {

            const response = await fetch(`http://localhost:5000/${endpoint}`, {

                method: 'POST',

                headers: { 'Content-Type': 'application/json' },

                body: JSON.stringify({ email, password }),

            });

            const data = await response.json();

            if (response.ok) {

                if (isLogin) {

                    setUserId(data.userId);  // Set userId on successful login

                    navigate('/movies');  // Redirect to the movie app on login success

                } else {

                    alert('Registration successful! Please log in.');

                    setIsLogin(true);

                }

            } else {

                alert(data.error);

            }

        } catch (error) {

            console.error(`Error ${isLogin ? 'logging in' : 'registering'}:`, error);

        }

    };

    return (

        <div className="auth-container">

            <h1 className="project-title">Movie Search APP</h1> {/\* Project title \*/}

            <h2>{isLogin ? 'Login' : 'Register'}</h2>

            <input

                type="email"

                placeholder="Email"

                value={email}

                onChange={(e) => setEmail(e.target.value)}

            />

            <input

                type="password"

                placeholder="Password"

                value={password}

                onChange={(e) => setPassword(e.target.value)}

            />

            {!isLogin && (

                <input

                    type="password"

                    placeholder="Confirm Password"

                    value={confirmPassword}

                    onChange={(e) => setConfirmPassword(e.target.value)}

                />

            )}

            <button onClick={handleAuth}>{isLogin ? 'Login' : 'Register'}</button>

            <p>

                {isLogin ? "Don't have an account?" : 'Already have an account?'}{' '}

                <span

                    onClick={() => setIsLogin(!isLogin)}

                    style={{ color: 'blue', cursor: 'pointer' }}

                >

                    {isLogin ? 'Register here' : 'Login here'}

                </span>

            </p>

        </div>

    );

}

//MovieApp.js

import React, { useEffect, useState } from 'react';

import './MovieApp.css';

import { AiOutlineSearch } from "react-icons/ai";

import axios from 'axios';

export default function MovieApp() {

    const [movies, setMovies] = useState([]);

    const [sortBy, setSortBy] = useState('popularity.desc');

    const [searchQuery, setSearchQuery] = useState('');

    const [genres, setGenres] = useState([]);

    const [selectedGenre, setSelectedGenre] = useState('');

    // Fetch genres on component mount

    useEffect(() => {

        const fetchGenres = async () => {

            try {

                const response = await axios.get(

                    'https://api.themoviedb.org/3/genre/movie/list',

                    {

                        params: {

                            api\_key: 'afd8e97aa856fc0ccaa601681d56510e',

                        },

                    }

                );

                setGenres(response.data.genres);

            } catch (error) {

                console.error('Error fetching genres:', error);

            }

        };

        fetchGenres();

    }, []);

    // Fetch movies when search query, sort option, or selected genre changes

    useEffect(() => {

        const fetchMovies = async () => {

            try {

                const response = await axios.get(

                    'https://api.themoviedb.org/3/discover/movie',

                    {

                        params: {

                            api\_key: 'afd8e97aa856fc0ccaa601681d56510e',

                            sort\_by: sortBy,

                            page: 1,

                            with\_genres: selectedGenre,

                            query: searchQuery,

                        },

                    }

                );

                setMovies(response.data.results);

            } catch (error) {

                console.error('Error fetching movies:', error);

            }

        };

        fetchMovies();

    }, [searchQuery, sortBy, selectedGenre]);

    const handleSearchChange = (event) => {

        setSearchQuery(event.target.value);

    };

    const handleSearchSubmit = async () => {

        if (searchQuery.trim()) {

            try {

                const response = await axios.get(

                    'https://api.themoviedb.org/3/search/movie',

                    {

                        params: {

                            api\_key: 'afd8e97aa856fc0ccaa601681d56510e',

                            query: searchQuery,

                        },

                    }

                );

                setMovies(response.data.results);

            } catch (error) {

                console.error('Error searching for movies:', error);

            }

        } else {

            // Reset movies if search query is empty

            setSearchQuery('');

            setSelectedGenre('');

            setSortBy('popularity.desc');

        }

    };

    const handleSortChange = (event) => {

        setSortBy(event.target.value);

    };

    const handleGenreChange = (event) => {

        setSelectedGenre(event.target.value);

    };

    // Handle Enter key press for search

    const handleKeyDown = (event) => {

        if (event.key === 'Enter') {

            handleSearchSubmit();

        }

    };

    return (

        <div className='container'>

            <h1>Movie Search App</h1>

            <div className='search-bar'>

                <input

                    type='text'

                    placeholder='Search Movie...'

                    value={searchQuery}

                    onChange={handleSearchChange}

                    onKeyDown={handleKeyDown} // Add keydown event

                    className='search-input'

                />

                <button onClick={handleSearchSubmit} className='search-button'>

                    <AiOutlineSearch />

                </button>

            </div>

            <div className='filters'>

                <label htmlFor='sort-by'>Sort By:</label>

                <select id='sort-by' value={sortBy} onChange={handleSortChange}>

                    <option value="popularity.desc">Popularity Descending</option>

                    <option value="popularity.asc">Popularity Ascending</option>

                    <option value="vote\_average.desc">Rating Descending</option>

                    <option value="vote\_average.asc">Rating Ascending</option>

                    <option value="release\_date.desc">Release Date Descending</option>

                    <option value="release\_date.asc">Release Date Ascending</option>

                </select>

                <label htmlFor='genre'>Genre:</label>

                <select id='genre' value={selectedGenre} onChange={handleGenreChange}>

                    <option value="">All Genres</option>

                    {genres.map((genre) => (

                        <option key={genre.id} value={genre.id}>

                            {genre.name}

                        </option>

                    ))}

                </select>

            </div>

            <div className="movie-wrapper">

                {movies.length > 0 ? (

                    movies.map((movie) => (

                        <div key={movie.id} className='movie'>

                            <img

                                src={`https://image.tmdb.org/t/p/w500${movie.poster\_pat }}

                                alt={movie.title}

                            />

                            <h2>{movie.title}</h2>

                            <p className='rating'>Rating: {movie.vote\_average}</p>

                            <p>{movie.overview}</p> {/\* Display full overview \*/}

                            <br />

                            <a

                                href={`https://www.themoviedb.org/movie/${movie.id}`}

                                target="\_blank"

                                rel="noopener noreferrer"

                                className="more-info"

                            >

                                More Info

                            </a>

                        </div>

                    ))

                ) : (

                    <p>No movies found.</p>

                )}

            </div>

        </div>

    );

}

// server.js

const express = require('express');

const mongoose = require('mongoose');

const bcrypt = require('bcrypt');

const cors = require('cors');

const app = express();

app.use(cors({

    origin: 'http://localhost:3000',

    credentials: true,

}));

app.use(express.json());

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/movieApp', { useNewUrlParser: true, useUnifiedTopology: true })

    .then(() => console.log('MongoDB connected'))

    .catch((error) => console.log('MongoDB connection error:', error));

// User Sc

const userSchema = new mongoose.Schema({

    name: String,

    email: { type: String, unique: true },

    password: String,

});

const User = mongoose.model('User', userSchema);

// Register Route

app.post('/register', async (req, res) => {

    const { name, email, password } = req.body;

    const hashedPassword = await bcrypt.hash(password, 10);

    try {

        const user = new User({ name, email, password: hashedPassword });

        await user.save();

        res.status(201).json({ message: 'User registered successfully' });

    } catch (error) {

        res.status(400).json({ error: 'Email already exists' });

    }

});

// Login Route

app.post('/login', async (req, res) => {

    const { email, password } = req.body;

    const user = await User.findOne({ email });

    if (user && await bcrypt.compare(password, user.password)) {

        res.json({ message: 'Login successful', userId: user.\_id });

    } else {

        res.status(400).json({ error: 'Invalid email or password' });

    }

});

// Protected Route: Fetch Movies

app.get('/movies', async (req, res) => {

    const { userId } = req.query;

    // Verify user exists in the database (minimal check)

    const user = await User.findById(userId);

    if (user) {

        res.json({ message: 'Access to movies granted' });

    } else {

        res.status(401).json({ error: 'Unauthorized access' });

    }

});

app.listen(5000, () => {

    console.log('Server running on http://localhost:5000');

});