**Assignment 6: Role-Based Access Control**

**Extend user authentication to include role-based access control.**

**Restrict certain routes or components based on user roles.**

**Implementing role-based access control (RBAC) involves extending the user authentication system to include roles for users and restricting access to certain routes or components based on these roles. In this example, I'll show you how to add roles to the authentication system and restrict access in a React app.**

**Step 1: Update Express.js API for Roles**

**Modify the Express.js API to include roles when generating JWT tokens. For simplicity, let's assume that the roles are part of the user object.**

| **// express-api/authController.js**  **const jwt = require('jsonwebtoken');**  **const secretKey = 'your-secret-key';**  **const users = [**  **{ id: 1, username: 'admin', password: 'admin123', role: 'admin' },**  **{ id: 2, username: 'user', password: 'user123', role: 'user' },**  **];**  **const login = (req, res) => {**  **const { username, password } = req.body;**  **const user = users.find((u) => u.username === username && u.password === password);**  **if (user) {**  **const token = jwt.sign({ id: user.id, username, role: user.role }, secretKey, { expiresIn: '1h' });**  **res.json({ token });**  **} else {**  **res.status(401).json({ error: 'Invalid credentials' });**  **}**  **};**  **module.exports = {**  **login,**  **};** |
| --- |

**Step 2: Update React Authentication Component**

**Modify the Login.js component to handle roles when setting the token.**

| **// src/Login.js**  **import React, { useState } from 'react';**  **import axios from 'axios';**  **const Login = ({ setToken }) => {**  **const [username, setUsername] = useState('');**  **const [password, setPassword] = useState('');**  **const handleLogin = async () => {**  **try {**  **const response = await axios.post('http://localhost:3001/api/login', {**  **username,**  **password,**  **});**  **const { token } = response.data;**  **setToken(token);**  **} catch (error) {**  **console.error('Error logging in:', error.response?.data?.error || error.message);**  **}**  **};**  **return (**  **<div>**  **<h2>Login</h2>**  **<label>**  **Username:**  **<input type="text" value={username} onChange={(e) => setUsername(e.target.value)} />**  **</label>**  **<label>**  **Password:**  **<input type="password" value={password} onChange={(e) => setPassword(e.target.value)} />**  **</label>**  **<button onClick={handleLogin}>Login</button>**  **</div>**  **);**  **};**  **export default Login;** |
| --- |

**Step 3: Update React App to Use Roles**

**Modify the App.js file to pass the token and user role to the DataFetching component.**

| **// src/App.js**  **import React, { useState } from 'react';**  **import Login from './Login';**  **import DataFetching from './DataFetching';**  **function App() {**  **const [token, setToken] = useState('');**  **return (**  **<div>**  **{token ? (**  **<DataFetching token={token} />**  **) : (**  **<Login setToken={setToken} />**  **)}**  **</div>**  **);**  **}**  **export default App;** |
| --- |

**Step 4: Update DataFetching.js to Check Roles**

**Modify the DataFetching.js component to check the user role before making an authenticated request. You can customize this based on your application's requirements.**

| **// src/DataFetching.js**  **import React, { useState, useEffect } from 'react';**  **import axios from 'axios';**  **const DataFetching = ({ token }) => {**  **const [data, setData] = useState([]);**  **const [loading, setLoading] = useState(true);**  **const [error, setError] = useState(null);**  **useEffect(() => {**  **const fetchData = async () => {**  **try {**  **const response = await axios.get('http://localhost:3001/api/data', {**  **headers: {**  **Authorization: `Bearer ${token}`,**  **},**  **});**  **// Check user role before setting data**  **if (response.data && response.data.role === 'admin') {**  **setData(response.data);**  **} else {**  **setError('Unauthorized access');**  **}**  **setLoading(false);**  **} catch (error) {**  **setError(error.message);**  **setLoading(false);**  **}**  **};**  **fetchData();**  **}, [token]);**  **return (**  **<div>**  **<h2>Data Fetching with Authentication</h2>**  **{loading ? (**  **<p>Loading...</p>**  **) : error ? (**  **<p>Error: {error}</p>**  **) : (**  **<ul>**  **{data.map((item) => (**  **<li key={item.id}>{item.title}</li>**  **))}**  **</ul>**  **)}**  **</div>**  **);**  **};**  **export default DataFetching;** |
| --- |

**Step 5: Run the App**

**Start your development server:**

**npm start**

**Visit http://localhost:3000 in your browser. You should see your React app with a login form. After successful login with the correct role (e.g., 'admin'), it will fetch and display data from the authenticated API endpoint. If the user has the role 'user', it will display an unauthorized access error.**

**This example demonstrates a basic implementation of role-based access control in a React app with user authentication. Depending on your application's complexity and requirements, you might consider more advanced solutions for RBAC and user permissions.**