**Assignment 5: User Authentication in React**

**Connect React with the Express.js authentication API.**

**Implement user authentication in your React app.**

Implementing user authentication involves several steps, including setting up authentication on the server side (Express.js API) and integrating it with the React app. Below, I'll provide a basic example using JSON Web Tokens (JWT) for authentication. Note that this is a simplified example, and in a production environment, you should consider additional security measures.

Step 1: Set Up Express.js Authentication API

Assuming you have an Express.js API with authentication endpoints (e.g., /api/login), implement the authentication logic and JWT generation. For simplicity, let's use the jsonwebtoken library for JWT.

| **// express-api/authController.js**  **const jwt = require('jsonwebtoken');**  **const secretKey = 'your-secret-key'; // Replace with a strong secret key**  **const login = (req, res) => {**  **const { username, password } = req.body;**  **// Check username and password (replace this with your actual authentication logic)**  **if (username === 'user' && password === 'password') {**  **// Generate a JWT token**  **const token = jwt.sign({ username }, secretKey, { expiresIn: '1h' });**  **res.json({ token });**  **} else {**  **res.status(401).json({ error: 'Invalid credentials' });**  **}**  **};**  **module.exports = {**  **login,**  **};** |
| --- |

Step 2: Set Up React Authentication Component

Create a new component, say Login.js, to handle user authentication in your React app:

| // 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: Integrate into App

Now, integrate the Login component into your main App.js file:

| // 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: Make Authenticated Requests in React

Modify the DataFetching.js component to include the token in authenticated requests:

| // 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}`,  },  });  setData(response.data);  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, it will fetch and display data from the authenticated API endpoint.

Keep in mind that this is a basic example, and in a real-world scenario, you would implement more secure authentication mechanisms, error handling, and possibly use a state management library like Redux to handle the authentication state globally in your React app. Additionally, make sure to secure your Express.js API by implementing proper authentication middleware.