**Assignment 7: Securing API Routes**

**Secure your Express.js API routes using authentication middleware.**

**Ensure that only authenticated users can access certain routes.**

**To secure Express.js API routes, you can use middleware to authenticate users before granting access to specific routes. In this example, I'll demonstrate how to implement a simple authentication middleware to protect certain routes. We'll continue with the user authentication example from the previous assignment.**

**Step 1: Create Authentication Middleware**

**Create a new file, say authMiddleware.js, to define the authentication middleware:**

| **// express-api/authMiddleware.js**  **const jwt = require('jsonwebtoken');**  **const secretKey = 'your-secret-key';**  **const authenticateToken = (req, res, next) => {**  **const token = req.headers.authorization;**  **if (!token) {**  **return res.status(401).json({ error: 'Unauthorized: Token is missing' });**  **}**  **jwt.verify(token.replace('Bearer ', ''), secretKey, (err, user) => {**  **if (err) {**  **return res.status(403).json({ error: 'Forbidden: Invalid token' });**  **}**  **req.user = user;**  **next();**  **});**  **};**  **module.exports = {**  **authenticateToken,**  **};** |
| --- |

**Step 2: Use Authentication Middleware in Express.js Routes**

**Update your Express.js routes to use the authentication middleware for protecting certain routes. For example:**

| **// express-api/dataController.js**  **const { authenticateToken } = require('./authMiddleware');**  **const data = [**  **{ id: 1, title: 'Item 1' },**  **{ id: 2, title: 'Item 2' },**  **{ id: 3, title: 'Item 3' },**  **];**  **const getAllData = (req, res) => {**  **res.json(data);**  **};**  **const getDataForAuthenticatedUser = (req, res) => {**  **// This route is protected by authentication middleware**  **// Only authenticated users can access it**  **res.json({**  **user: req.user, // User information from the decoded token**  **data,**  **});**  **};**  **module.exports = {**  **getAllData,**  **getDataForAuthenticatedUser,**  **};** |
| --- |

**Step 3: Use Authentication Middleware in Express.js Routes**

**Modify your app.js or main Express.js file to include the authentication middleware in the routes:**

| **// express-api/app.js**  **const express = require('express');**  **const bodyParser = require('body-parser');**  **const { login } = require('./authController');**  **const { getAllData, getDataForAuthenticatedUser } = require('./dataController');**  **const { authenticateToken } = require('./authMiddleware');**  **const app = express();**  **const port = 3001;**  **app.use(bodyParser.json());**  **// Login route**  **app.post('/api/login', login);**  **// Public route**  **app.get('/api/data', getAllData);**  **// Protected route (requires authentication)**  **app.get('/api/data/authenticated', authenticateToken, getDataForAuthenticatedUser);**  **app.listen(port, () => {**  **console.log(`Express API is listening at http://localhost:${port}`);**  **});** |
| --- |

**Step 4: Run the Express.js API**

**Start your Express.js API server:**

node express-api/app.js

Step 5: Update React App to Use Authenticated Route

Modify the DataFetching.js component to access the protected route with authentication:

| // 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/authenticated', {  headers: {  Authorization: `Bearer ${token}`,  },  });  setData(response.data.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 6: Run the React App

Start your React development server:

npm start

Visit http://localhost:3000 in your browser. After successful login, the React app will fetch and display data from the protected API route, ensuring that only authenticated users can access it.

This example demonstrates how to secure your Express.js API routes using a simple authentication middleware. In a real-world scenario, you might consider more advanced authentication mechanisms and authorization checks based on user roles and permissions.