CPSC 2650 Assignment 8: JWT Auth with HttpOnly Cookies

By the end of this assignment, you should be able to:

- Refactor the existing REST-based authentication flow to use **HttpOnly** and **secure** cookies instead of storing JWTs in localStorage.
- Use **cors** and **cookie-parser** to handle cookies and cross-origin requests.

Setup Instructions:

- Note: This assignment assumes you have completed AS 7 before starting this one.

Task 1: Install Dependencies, set up `cookie-parser` and `cors`

We use cookie-parser to read cookies sent by the browser (like login tokens). Cors lets your frontend (on a different port) safely send requests to your backend.

Both are needed to support secure login with cookies in a MERN stack app.

In summary, (1) cookie-parser helps read cookies and (2) cors allows the frontend (React) to talk to the backend (express) server.

Ensure your backend has these installed:

```
npm install cookie-parser cors
```

Now, enable **`cookie-parser`** middleware and set up **`cors`** with credentials in your server entry point

Code Snippets	Explanation:

`const cookieParser = require('cookie-parser');`
`app.use(cookieParser());`
	We add cookie-parser so our server can read cookies from incoming requests.

`const cors = require('cors');`
`app.use(cors({`
`  origin: 'http://localhost:5173', // your React frontend`
`  credentials: true`
`}));`
	We use CORS to allow our React frontend to talk to the backend. Setting **`credentials: true`** lets cookies (like login tokens) be sent with requests.

Also, add an environment variable `NODE_ENV=development` to tell your app it's running in development mode. This will be used in the later steps.

Task 2: Modify your auth routes to use cookies

Previously, in the **`/login`** route, you sent the JWT directly to the client using `res.json({ token })`, and possibly during **`/signup`** as well.

To improve security, we'll now set the JWT as an **HttpOnly cookie** during login—this keeps it hidden from JavaScript and safe from XSS attacks. During signup, we'll skip sending a token entirely and just return a success response.

We'll also update the auth middleware to read the token from **`req.cookies.token`** instead of the Authorization header.

Code Snippets	Explanation:
`// login route change` `res.cookie('token', token, {` `httpOnly: true,` `secure: process.env.NODE_ENV === 'production',` `sameSite: 'Lax',` `maxAge: 7 * 24 * 60 * 60 * 1000` `})` `.json({ user }); // Send back user object directly`	We're setting a secure cookie named 'token' that stores the JWT. **httpOnly:** true means JavaScript can't access it (helps prevent XSS). **secure:** process.env.NODE_ENV === 'production' sends the cookie only over HTTPS in production. **sameSite:** 'Lax' helps prevent CSRF but still allows basic navigation. **maxAge:** sets the cookie to last 7 days. After setting the cookie, we send back the logged-in user object in the response.
`// signup route change` `res.status(201).json({ error: 'New User Created' })`	During signup, we don't need to send a JWT — logging in comes next. So instead of sending a token, we just return a **201 Created status** with a message like 'New User Created' to confirm the signup was successful.
`// /me route change` `//replace this` `const token = req.headers.authorization?.split(' ')[1];` `// with this` `const token = req.cookies.token;`	In the /me route, instead of reading the token from the Authorization header, we now read it from a cookie using **req.cookies.token**. This works because we're storing the token as an HttpOnly cookie during login.

```
// Add a logout route handler


router.post('/logout', (req, res) => {
  res.clearCookie('token', {
    httpOnly: true,
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'Strict'
  });
  res.json({ message: 'Logged out successfully' });
});

module.exports = router;
```

This POST /logout route clears the token cookie from the browser using res.clearCookie.

It ensures the cookie is removed securely and then sends back a message confirming the logout.

sameSite: 'Lax' lets the cookie be set during login from your frontend.

sameSite: 'Strict' blocks cross-site access, making logout more secure.

Task 3: Update Front-end React Components

In this section, you'll stop handling JWTs manually. Instead, you'll let the browser manage authentication using secure cookies — just make sure all your fetch requests include **`credentials: 'include'`** to send and receive them properly.

Code Snippets | Explanation:

```
// Updates to Login.jsx

fetch('/login', {
  method: 'POST',
  credentials: 'include',
  ...
});

  ...
if (res.ok) {
      onAuth(data.user); //
```

credentials: 'include' tells the browser to send and receive cookies with the request — this is how we stay logged in without storing tokens manually.

Instead of calling **onAuth(data.token)**, switch to **onAuth(data.user)** — because the token is now handled by the browser as a cookie, and we only need the user info.

```
// Updates to App.jsx

const [user, setUser] = useState(null); // user state tracking
const [message, setMessage] = useState(''); // same as before

useEffect(() => {
  fetch('http://localhost:4000/api/auth/me', {
    credentials: 'include'
  })
    .then(res => res.ok ? res.json() : null)
    .then(data => setUser(data))
}, []);
```

useState(null) starts with no user — we update it after login. useEffect runs once on page load to check if the user is already logged in by hitting /me with cookies included. This keeps the session even after refreshing!

.then(res => res.ok ? res.json() : null) — If the server responds with a success status (like 200), it parses the JSON. If not, it returns null.

```
const handleLogin = (user) => {
    setUser(user); // Update the user state
    setMessage('Logged in successfully.');
  };
```
	handleLogin(user) sets the user in state and shows a success message — no token needed anymore. No more localStorage

```
  const handleLogout = async () => {
    await fetch('http://localhost:4000/api/auth/logout', {
      method: 'POST',
      credentials: 'include'
    });

    setUser(null);
    setMessage('Logged out.');
  };
```
	handleLogout() sends a logout request, clears the user from state, and updates the message.

A few other things you will need to do are as follows:

- Remember to update any references in your Routes to token with **user**
- Remember to pass props to Navbar and Login routes in App.jsx.
- Remember to add credential: 'include' in your fetch request to /me in Profile.jsx. No need to pass the Auth header with a bearer token anymore.

❓ Check Your Understanding

1. Why do we use credentials: 'include' in our fetch requests when working with cookies?
2. What's the difference between storing a JWT in localStorage vs. setting it as an HttpOnly cookie? Which one is safer, and why?
3. In the /login route, why do we send back the user object instead of the token?
4. Why do we need a useEffect in App.jsx to call /me on page load? What happens without it?
5. What does the sameSite option in the res.cookie function control? Why do we use 'Lax' in some places and 'Strict' in others?

Hand In:

Zip your assignment directory (with your written responses text file), submit your work in the Assignment 8 folder, and hand it in.

- Remember to exclude the **node_modules** folders for each subfolder when you zip your directory. The marker will download your submission, install the node modules for each part and run your app.

Checklist:

[1.0 mark] Task 1: Install Dependencies, set up cookie-parser and cors

- Installed cors and cookie-parser using npm
- Added const cookieParser = require('cookie-parser'); in your Express app
- Called app.use(cookieParser());
- Configured cors middleware with { origin: 'http://localhost:5173', credentials: true }

[3.5 marks] Task 2: Modify your auth routes to use cookies

- In /login, set the JWT as an HttpOnly cookie using res.cookie(...)
- In /signup, removed token logic and returned res.status(201).json(...) instead
- Updated /me route to read the token from req.cookies.token instead of headers
- Updated your auth middleware (verifyToken) to use cookies instead of headers
- Implemented a /logout route that clears the cookie using res.clearCookie(...)

[3.5 marks] Task 3: Update Front-End Components

- Updated all fetch requests (e.g., /login, /me, /logout) to include **credentials: 'include'**
- Replaced any use of token with user state throughout the app
- Updated App.jsx to use useEffect to fetch /me on page load and set the user
- Made sure Navbar uses user state to show correct links after login/logout
- Updated <Login /> route to call onAuth(data.user) after successful login

[2.0 marks] Check Your Understanding

- All 5 written response questions answered correctly
- Submitted as a .txt, or .pdf file

Total: 10 marks