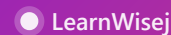


AI Coding Exercise

Build this module's AccessOps sample with ChatGPT or Claude from a ready-made prompt that points the model at this module's specification, then review, run and extend what it gives you.



The assistant is the fast pair of hands. You are still the architect.

An assistant can produce this module's sample in minutes. What it cannot do is decide whether the result is the right design for your application — that call stays with you. You supply the judgement: brief it properly, then check what comes back against what Module 7 just taught you.

What you are asking for

Complete the AccessOps capstone: enforce HTTPS with HSTS and set security headers, mark cookies Secure and HttpOnly with a strict SameSite policy, set session timeout and request size limits, add rate limiting on login and share links, run a dependency audit and record the results, then walk the full checklist from this course against the app and fix every finding.

The module asks for these deliverables, and the prompt below asks the assistant for exactly them:

- ✓ HTTPS, HSTS and security header configuration
- ✓ Cookie and session hardening settings
- ✓ Rate limiting on login and share links
- ✓ Dependency audit report
- ✓ Completed security review checklist with fixes

Why it is shaped this way

AccessOps now verifies passwords against a salted hash, resolves permissions in a service, streams documents only through DocumentService and audits every approval. None of that helps if the first request travels in clear text, if a script can read the session cookie, or if the build ships a package with a published advisory. The last mile belongs to the host: the certificate, the response headers, the cookie flags, the request limits, and the version of everything in the dependency graph.

These are the terms the assistant has to get right. If what comes back uses them loosely, it has not understood the module:

Term	What it means here
HSTS	A Strict-Transport-Security header telling the browser to refuse plain HTTP for this host for a stated time, so a later visit is never intercepted on the redirect.
Content-Security-Policy	A header listing the origins a page may load scripts, styles and frames from; it turns an injected script into a blocked request instead of code running in a live session.
frame-ancestors	The CSP directive (X-Frame-Options is the older equivalent) that stops another site framing AccessOps and tricking a manager into clicking Approve.
cookie flags	Secure keeps the session cookie off plain HTTP, HttpOnly keeps it away from JavaScript, and SameSite stops it riding a request another site started.
request limits	Host-level caps on body and header size that reject an oversized upload before a line of AccessOps code runs.
token bucket	A counter that grants an attempt only while it holds a token and refills at a fixed rate, so traffic may burst a little and then slows to a crawl.
dependency audit	A repeatable check of every direct and transitive package, and of the runtime, against published advisories — recorded with a date and a decision.

Build it — pick your assistant

Both buttons open a new tab with the **same ready-made prompt** already filled in. The prompt points the model at this module's specification — a single Markdown file holding the lesson, the lab brief and the deliverables — so it works from the real material instead of guessing what Module 7 is about. You can open [module-07.md](#) yourself to see exactly what it will read.



Use ChatGPT

Opens [chatgpt.com](#) with this module's prompt loaded. Works with GPT-5 or Codex; any plan that can browse the web will fetch the spec.



Use Claude

Opens [claude.ai](#) with the same prompt loaded. Claude Code users can paste it into a terminal in an empty folder instead.



Send it as-is and you get a project, not a snippet

The prompt ends by asking for the whole solution as a single [.zip](#) with build-and-run steps. Download it, open the solution in Visual Studio, press F5 and AccessOps comes up on [localhost](#). You need the .NET SDK and the [Wisej-4](#) NuGet package; everything else the prompt asks the assistant to write itself.

The prompt, in full — read it, then edit it if you want a different sample

I am working through Module 7 – "Hardening Checklist and Capstone" – of the LearnWisej Security & Auth Patterns course, and I want to build the module sample with you.

Start by reading the module specification. It is a single Markdown file holding the full lesson, the lab brief and the deliverables:

<https://learnwisej.com/assets/courses/security-and-auth-patterns/specs/module-07.md>

Then build a complete Wisej.NET 4 project I can open in Visual Studio:

1. Project "AccessOps", C#, net8.0 or later, NuGet package Wisej-4 4.1.x, Kestrel host calling `app.UseWisej()`.
2. Build what the specification's lab asks for: Complete the AccessOps capstone: enforce HTTPS with HSTS and set security headers, mark cookies Secure and HttpOnly with a strict SameSite policy, set session timeout and request size limits, add rate limiting on login and share links, run a dependency audit and record the results, then walk the full checklist from this course against the app and fix every finding.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - HTTPS, HSTS and security header configuration
 - Cookie and session hardening settings
 - Rate limiting on login and share links
 - Dependency audit report
 - Completed security review checklist with fixes
4. Follow the specification wherever it names a Wisej.NET class, property or event – use exactly that one. Where it leaves a choice open, tell me which API you picked and why.
5. Keep the server authoritative: state and validation live in .NET, and the browser renders and reports. Never move a decision into JavaScript because it is easier there.
6. Make the result demonstrable: I should be able to run it and see the behaviour this module is about, including what happens when something goes wrong.

Before you write any code, restate in your own words what you are going to build and where each piece will live, so I can correct the design while it is still cheap.

When the code is done, package the whole solution as a single .zip I can download, and give me the exact steps to build and run it.

Finally, answer these questions from the specification, using your own implementation as the evidence:

- Middleware can rate limit the share-link handler but cannot rate limit the Sign in button. What is different about the two requests in a Wisej.NET app, and where does the login limit have to live instead?
- Your Content-Security-Policy is enforcing and the grid renders unstyled. Which directive is the likely cause, and how would report-only mode have told you before the deployment did?
- The dependency audit lists a vulnerable package nobody in AccessOps references directly. How did it get into the build, and what are the two honest ways to close the finding?

Ask me about anything that is ambiguous before you start.

Then review what came back

Code that compiles is not yet code that is right. The interesting questions are design questions: is this the Wisej.NET control the job actually calls for, does it follow the lifecycle the framework gives you, is the

authoritative state on the server rather than parked in the browser because that was convenient? Read the result as a reviewer, not as a customer.

This module gives you the checklist. Answer each one against the code you just received:

- ✓ Middleware can rate limit the share-link handler but cannot rate limit the Sign in button. What is different about the two requests in a Wisej.NET app, and where does the login limit have to live instead?
- ✓ Your Content-Security-Policy is enforcing and the grid renders unstyled. Which directive is the likely cause, and how would report-only mode have told you before the deployment did?
- ✓ The dependency audit lists a vulnerable package nobody in AccessOps references directly. How did it get into the build, and what are the two honest ways to close the finding?

Anything you had to correct is worth writing down. A short note of the conventions and decisions you settled on, pasted into the first message of your next session, is worth more than any single clever prompt.



No reference sample for this course yet — yours is the first

The [LearnWisej samples repository](#) carries worked builds for most courses, but not yet for this one. That means there is nothing to compare against and no shortcut: the specification and your own review are the whole safety net. Keep what you build — and if it turns out well, we would like to see it.

Three prompts to take it further

Point your assistant at the project you just built and try these. Each one is a review-and-extend exercise, not a rewrite.

- ✓ **Review it like a pull request.** *"Walk me through this AccessOps code as if you were reviewing a colleague's pull request and did not write it. Where does state live, what happens end to end on one user action, what would you send back for changes, and which parts would you not be comfortable putting in front of a customer? Be specific and quote the lines you mean — do not summarise the design back to me."*
- ✓ **Swap a piece without moving the seams.** *"Pick the part of this AccessOps project that is most obviously a stand-in for something real — a fake data source, an in-memory store, a simplified service — and replace it with a production-shaped implementation. The contract either side of it must not change: same types, same calls, same behaviour under failure. Tell me what the real implementation forced you to handle that the stand-in did not."*
- ✓ **Make it survive a bad day.** *"Review this AccessOps project for the failure modes it does not handle yet — bad or missing input, a dependency that is slow or unavailable, work arriving after the user navigated away, and two actions racing each other. For each one, tell me first whether it is a real risk in Wisej.NET 4 and why, and only then write the fix. Where you decide something needs no handling, say so and explain the reasoning rather than adding defensive code."*



Carry it into the lab

The hands-on lab that closes this module asks for the same deliverables you just had the assistant produce. Use what you have here as the shape of your answer and the evidence behind it — not as the answer itself. The lab is graded on your reasoning, and that part has to be yours.