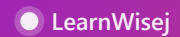


AI Coding Exercise

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 2 just taught you.

What you are asking for

Implement AccessOps authentication: a login page with lockout after five failures, password verification with a modern salted hash, a SessionContext created only after verification and stored in the session, an idle timeout that returns the user to login and clears state, an explicit Logout that ends the session, and an audit record for every login attempt.

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

- ✓ Login page with lockout and generic error messages
- ✓ Password hashing and verification service
- ✓ SessionContext with verified identity
- ✓ Idle timeout and explicit logout
- ✓ Login audit records

Why it is shaped this way

AccessOps holds employee records, confidential documents and approval requests, so the first screen anyone reaches is also the first control that matters. Everything the rest of this course enforces — permissions, record-level rules, audit trails — rests on a single assumption: that the server knows who is on the other end of this session and can prove it. Module 1 drew the trust boundaries; this module puts a verified identity inside the session boundary and keeps it from being guessed, borrowed or outlived.

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
salted slow hash	A one-way derivation of the password with a random per-user salt and a deliberately expensive iteration count, so a stolen table cannot be reversed and every guess costs real work.
constant-time comparison	Comparing two hashes in a fixed number of operations with <code>CryptographicOperations.FixedTimeEquals</code> , so the time the server takes to answer never reveals how many bytes of a guess were right.
SessionContext	The object created only after a password is verified, holding the employee id, department, roles and sign-in details, and stored in <code>Application.Session</code> as the one place any code asks who the user is.
session timeout	The idle period after which Wisej.NET disposes the server-side session and everything it holds, configured for the application in <code>Default.json</code> rather than guessed per screen.
session fixation	An attack in which the identifier that carried the anonymous login page also carries the authenticated session afterwards, so an identifier the attacker planted becomes a logged-in one.
login throttling	Counting failures per account and per client address and refusing further attempts for a lockout period, which is what turns an unlimited guessing machine into a few tries an hour.
login audit record	One row per attempt with the user name tried, the time, the client address and the real outcome — the place the specific failure reason is written, since the screen only ever shows a generic one.

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 2 is about. You can open [module-02.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 2 – "Authentication and Session Identity" – 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-02.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: Implement AccessOps authentication: a login page with lockout after five failures, password verification with a modern salted hash, a `SessionContext` created only after verification and stored in the session, an idle timeout that returns the user to login and clears state, an explicit Logout that ends the session, and an audit record for every login attempt.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Login page with lockout and generic error messages
 - Password hashing and verification service
 - `SessionContext` with verified identity
 - Idle timeout and explicit logout
 - Login audit records
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:

- Your login page answers a wrong password in 480 ms and an unknown user name in 6 ms. Explain what an attacker learns from that difference alone, and describe the two changes that remove the signal without changing the message.
- A colleague suggests keeping the signed-in employee in a static property because "it is simpler than reaching into the session every time". Describe exactly what the second concurrent user sees, and at which moment the two sessions first interfere.
- Your Logout button clears the `SessionContext` but does not end the session. Name two things that are still recoverable afterwards, and say which one you consider the more serious in a portal that shows confidential documents.

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:

- ✓ Your login page answers a wrong password in 480 ms and an unknown user name in 6 ms. Explain what an attacker learns from that difference alone, and describe the two changes that remove the signal without changing the message.
- ✓ A colleague suggests keeping the signed-in employee in a static property because "it is simpler than reaching into the session every time". Describe exactly what the second concurrent user sees, and at which moment the two sessions first interfere.
- ✓ Your Logout button clears the SessionContext but does not end the session. Name two things that are still recoverable afterwards, and say which one you consider the more serious in a portal that shows confidential documents.

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.