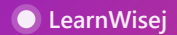


AI Coding Exercise

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.



You are not going to type this sample. You are going to direct it.

Almost nobody writes Wisej.NET code from a blank file any more — an assistant writes the first draft and the developer decides whether it is right. That only works if you know the material, which is what the rest of this module gave you. Here you hand that knowledge to ChatGPT or Claude, get this module's sample back as a working Visual Studio solution, and review it the way you would review a colleague's pull request.

What you are asking for

Add auditing and secret management to AccessOps: an AuditService that records approvals, permission changes and document downloads with user, time, address and correlation ID, an append-only audit table, a correlation ID created per command and written to every log line, connection strings and the OIDC client secret moved to environment configuration with a documented rotation procedure, and a log review that redacts sensitive fields.

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

- ✓ AuditService and append-only audit table
- ✓ Correlation ID in every command and log line
- ✓ Secrets moved to environment configuration
- ✓ Secret rotation procedure
- ✓ Log redaction review

Why it is shaped this way

AccessOps now authenticates people, resolves their permissions and guards its documents — and none of that helps on the morning somebody asks who approved request 4471, from which address, and whether anyone was refused first. The previous five modules decided what is allowed; this one decides what is recoverable afterwards. It also closes the gap that turns a small incident into a large one: a database password or an OIDC client secret in a file committed two years ago.

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
audit trail	A business record of every sensitive command — who acted, on what, when and from where — kept for a stated retention period because someone will have to answer for it.
append-only storage	An audit table the application account may only INSERT into. If the code that writes history can also rewrite it, the history proves nothing.
tamper evidence	A gapless sequence plus a hash chain, where each row's hash covers its own fields and the previous hash, so an edited row breaks the chain.
correlation ID	One identifier minted when a command starts and repeated on every log line and audit row it causes, so a single click can be reassembled later.
logging scope	An ambient property block — ILogger.BeginScope or a Serilog enricher — that attaches the correlation ID and actor to everything logged inside it.
secret store	Where a credential lives outside the build: user secrets locally, environment variables in a container, a managed vault in production — or a managed identity, with nothing stored at all.
redaction	A rule about what never reaches a log sink — passwords, tokens, cookies, connection strings and personal data — enforced by structured logging, not by memory.

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 6 is about. You can open [module-06.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 6 - "Auditing, Secrets and Configuration" - 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-06.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: Add auditing and secret management to AccessOps: an `AuditService` that records approvals, permission changes and document downloads with user, time, address and correlation ID, an append-only audit table, a correlation ID created per command and written to every log line, connection strings and the OIDC client secret moved to environment configuration with a documented rotation procedure, and a log review that redacts sensitive fields.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `AuditService` and append-only audit table
 - Correlation ID in every command and log line
 - Secrets moved to environment configuration
 - Secret rotation procedure
 - Log redaction review
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 audit table is insert-only for the application, and every row is hash-chained. Name something an attacker with the application's own database login can still do to the trail, and explain exactly how the verification pass would show it.
- An approval is started on the approval screen and finishes in a background task started with `Application.StartTask` after the user has signed out. Where does that task get its correlation ID and its actor from, and what breaks if it reads them from the session instead?
- A developer removes the OIDC client secret from the tracked configuration file and pushes the change. Why is the secret still compromised, and what has to happen before the incident is actually closed?

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 audit table is insert-only for the application, and every row is hash-chained. Name something an attacker with the application's own database login can still do to the trail, and explain exactly how the verification pass would show it.
- ✓ An approval is started on the approval screen and finishes in a background task started with `Application.StartTask` after the user has signed out. Where does that task get its correlation ID and its actor from, and what breaks if it reads them from the session instead?
- ✓ A developer removes the OIDC client secret from the tracked configuration file and pushes the change. Why is the secret still compromised, and what has to happen before the incident is actually closed?

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.

- ✓ **Prove it actually works.** *"Go through this project deliverable by deliverable against the module specification at <https://learnwisej.com/assets/courses/security-and-auth-patterns/specs/module-06.md> and show me the evidence that each one is genuinely done — the code path, what to click, and what I should see. Where a deliverable is only partly met, say so plainly instead of arguing it through. Then add whatever is missing."*
- ✓ **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."



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.