

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.

● LearnWisej



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

Create the AccessOps solution and write its threat model: draw the trust boundaries, list the assets (employee records, documents, approvals), enumerate the entry points (login, uploads, downloads, custom endpoints, the AI assistant), rate each threat, and map every mitigation to the module of this course that implements it. Build the shell with a placeholder login page.

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

- ✓ AccessOps solution with a placeholder login page
- ✓ Trust boundary diagram
- ✓ Asset and entry point inventory
- ✓ Threat table with likelihood, impact and mitigation
- ✓ Security control placement map

Why it is shaped this way

AccessOps is an internal operations portal: departments, employees, confidential documents, approval requests and an admin console. Everyone who can reach it already works for the company, which is exactly why the security work gets postponed — and why one portal ends up holding salary bands, disciplinary letters and the approval trail for every exception the business ever granted. Before a line of authentication code exists you need three answers: what the browser is holding, where the boundaries run, and which of the things that could go wrong are worth engineering against first.

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
server-side UI model	Controls are C# objects in the server process; the browser holds a rendered mirror and an event channel, so your code and queries are never downloaded.
widget state payload	The property values Wisej.NET serialises so the browser can draw a control. Whatever you assign is on the wire, including on controls the user cannot see.
trust boundary	A line where data passes between parties of different trust. Anything crossing one is re-checked by the more trusted side, because the other side can lie.
session boundary	The frontier between one browser and its own server state in Application.Session. Decisions come from state on the server side of it, never from a browser value.
entry point	Any route in that an attacker can reach: the login page, event handlers, uploads, downloads, custom HTTP handlers, interop calls and the AI assistant's prompt.
STRIDE	Six categories for naming threats — spoofing, tampering, repudiation, information disclosure, denial of service, elevation of privilege — so a review is systematic.
control placement	Choosing the boundary at which a rule can be enforced. A rule that exists only in the UI is a suggestion; the same rule inside a service is a control.

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 1 is about. You can open [module-01.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 1 – "The Wisej.NET Security Model" – 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-01.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: Create the AccessOps solution and write its threat model: draw the trust boundaries, list the assets (employee records, documents, approvals), enumerate the entry points (login, uploads, downloads, custom endpoints, the AI assistant), rate each threat, and map every mitigation to the module of this course that implements it. Build the shell with a placeholder login page.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - AccessOps solution with a placeholder login page
 - Trust boundary diagram
 - Asset and entry point inventory
 - Threat table with likelihood, impact and mitigation
 - Security control placement map
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 employee grid is bound to full employee records and the salary column is hidden in the designer. Describe what the browser is holding, and name the change that actually removes the salary from the wire.
- Two people are signed in to AccessOps on the same server. Explain which pieces of state they share by default, and what would have to be true for one of them to see the other's current department.
- Pick the threat in your table with the highest impact and argue the opposite case: what would have to be true about AccessOps for that mitigation to be a waste of the team's time?

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 employee grid is bound to full employee records and the salary column is hidden in the designer. Describe what the browser is holding, and name the change that actually removes the salary from the wire.
- ✓ Two people are signed in to AccessOps on the same server. Explain which pieces of state they share by default, and what would have to be true for one of them to see the other's current department.
- ✓ Pick the threat in your table with the highest impact and argue the opposite case: what would have to be true about AccessOps for that mitigation to be a waste of the team's time?

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-01.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.