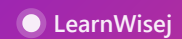


AI Coding Exercise

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.



Brief it, read it, then break it.

This exercise has three parts, and only the first is about prompting. You brief an assistant with the module specification and get an AccessOps solution back; you read what it wrote and decide whether it really does what the module asked; then you push it further than the module asked. The second and third parts are where the learning is.

What you are asking for

Add authorization to AccessOps: a PermissionService resolving roles and claims into permissions at login, an Authorize guard called by every DocumentService and ApprovalService method, a record-level rule that employees see only their department and managers only their reports, a UI that hides admin actions and disables approve buttons the user cannot use, and a test proving a forged request from a non-manager is rejected by the service.

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

- ✓ PermissionService resolving permissions into the SessionContext
- ✓ Authorization guard in every service method
- ✓ Department and manager record-level rules
- ✓ Permission-aware UI adaptation
- ✓ Negative test with a forged request

Why it is shaped this way

AccessOps now knows who is signed in: Module 2 left a verified identity in the SessionContext. Knowing who someone is answers nothing about what they may do with a confidential document or an approval request worth six figures. The failure mode in an internal portal is rarely a broken login — it is a payroll clerk who can read the legal department's contracts because the query was never scoped, or a team lead who approves their own expense request because the only check was a disabled button.

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
role	A coarse named job function such as Employee, Manager or Auditor. Roles are convenient to administer but too blunt to test against directly, because a screen needs to know about one action, not a whole job.
claim	A name/value fact about the verified identity — department 14, cost centre EU-North, manager of team 7. Claims carry the data that record-level rules need, and they come from the identity store, never from the browser.
permission	One named action the application can perform, such as ApproveRequest or ViewDepartmentDocuments. Code tests permissions; administrators assign roles; the mapping between them is configuration.
permission resolution	The single step at login that expands roles and claims into a flat, immutable permission set held in the SessionContext, so no request re-reads the role table and no two checks can disagree.
authorization guard	A call such as Require(Permission.ApproveRequest) at the top of a service method that throws before any work happens. It is the only place enforcement actually occurs.
record-level rule	A restriction on which rows a permitted action may touch — your own department, your own reports — applied inside the query or command rather than after the data has already been loaded.
UI adaptation	Hiding, disabling or explaining an action based on the same permission set the server enforces. It is a courtesy to the user and never 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 3 is about. You can open [module-03.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 3 - "Authorization: Roles, Claims and Permissions" - 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-03.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 authorization to AccessOps: a `PermissionService` resolving roles and claims into permissions at login, an `Authorize` guard called by every `DocumentService` and `ApprovalService` method, a record-level rule that employees see only their department and managers only their reports, a UI that hides admin actions and disables approve buttons the user cannot use, and a test proving a forged request from a non-manager is rejected by the service.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `PermissionService` resolving permissions into the `SessionContext`
 - Authorization guard in every service method
 - Department and manager record-level rules
 - Permission-aware UI adaptation
 - Negative test with a forged request
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:

- An auditor is granted approval rights while their AccessOps session is still open. Explain exactly when the new permission takes effect with your design, why you chose that moment, and what you would have to change to make it immediate.
- Your document query filters by department in the repository. Describe what an attacker gains if that filter is moved to the grid's `DataSource` instead, even though the visible result on screen is identical.
- Both `ApprovalService` and its repository could check that the approver manages the requester. Argue for one of them as the right home, and say what breaks if the check lives in the other, or in both.

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:

- ✓ An auditor is granted approval rights while their AccessOps session is still open. Explain exactly when the new permission takes effect with your design, why you chose that moment, and what you would have to change to make it immediate.
- ✓ Your document query filters by department in the repository. Describe what an attacker gains if that filter is moved to the grid's DataSource instead, even though the visible result on screen is identical.
- ✓ Both ApprovalService and its repository could check that the approver manages the requester. Argue for one of them as the right home, and say what breaks if the check lives in the other, or in both.

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.

- ✓ **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."*
- ✓ **Add the second case.** *"Extend this AccessOps project with a second case that follows the same pattern as the first — another entity, screen or control of the same shape, wired the same way. Factor out what is genuinely shared into a base class or service, but only where it earns its place;"*

two similar things do not always want an abstraction. Show me the diff and justify each thing you extracted."



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.