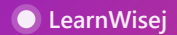


AI Coding Exercise

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.



Writing the code is the easy half now.

Getting working code for "OAuth and OpenID Connect Integration" out of a modern assistant takes one good prompt. Judging it, correcting it and extending it takes everything this module covered. That is the split this exercise is built around — you supply the specification and the review, the assistant supplies the typing.

What you are asking for

Add external sign-in to AccessOps: an OpenID Connect client using the authorization code flow with PKCE against a test identity provider, a callback handler that validates the ID token and its nonce, a claim mapping that fills the SessionContext and provisions a local employee record on first login, a fallback to local login for service accounts, and sign-out that ends both the local and the provider session.

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

- ✓ OIDC client configuration with PKCE
- ✓ Callback handling with token and nonce validation
- ✓ Claim mapping and first-login provisioning
- ✓ Local login fallback for service accounts
- ✓ Sign-out covering both sessions

Why it is shaped this way

AccessOps can already verify a password of its own, but the organisation running it does not want a second directory: staff sign in to everything else with the corporate identity provider, and joiners, leavers and multi-factor policy are administered in one place. Delegating login removes the password from AccessOps entirely, and replaces it with a different obligation — the portal is now handed an identity by a third party, through the user's own browser, and has to prove that identity is genuine, unmodified and meant for it before a single department record is shown.

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
authorization code flow	The OpenID Connect flow in which the browser only ever carries a short-lived, single-use code, while the tokens are fetched by your server over a direct HTTPS call the browser never sees.
PKCE	A random code verifier generated per sign-in, sent to the authorize endpoint only as its SHA-256 hash and revealed at the token endpoint, so an intercepted code cannot be redeemed by anyone else.
ID token	A signed JWT asserting that this provider authenticated this subject, for this client, at this time. It is the only token AccessOps consults to decide who the user is.
access and refresh tokens	The access token is a bearer credential for calling somebody else's API; the refresh token buys replacements in the back channel. Neither is evidence of identity for your own app.
state and nonce	Two random values you generate per sign-in: state ties the callback to the request you actually started, nonce ties the returned ID token to that same request so an older token cannot be replayed into it.
redirect URI	The exact, pre-registered URL the provider returns the browser to. In AccessOps it is a small server-side endpoint rather than a Wisej.NET window, because it has to run before any UI exists.
claim mapping and provisioning	Turning provider claims into an AccessOps employee: the issuer and subject pair identifies the account, and a local row is created on first sign-in carrying no permissions until someone grants them.

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 4 is about. You can open [module-04.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 4 – "OAuth and OpenID Connect Integration" – 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-04.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 external sign-in to AccessOps: an OpenID Connect client using the authorization code flow with PKCE against a test identity provider, a callback handler that validates the ID token and its nonce, a claim mapping that fills the `SessionContext` and provisions a local employee record on first login, a fallback to local login for service accounts, and sign-out that ends both the local and the provider session.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - OIDC client configuration with PKCE
 - Callback handling with token and nonce validation
 - Claim mapping and first-login provisioning
 - Local login fallback for service accounts
 - Sign-out covering both sessions
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 attacker captures the authorization code out of a browser history on a shared machine. What stops them exchanging it for tokens, and what would have failed to stop them without PKCE?
- Your provider issues group claims that look exactly like the AccessOps permission names. Why does the `PermissionService` still resolve permissions itself instead of copying those claims straight into the `SessionContext`?
- A user clicks Sign out, closes nothing, and immediately clicks Sign in again – and lands back inside AccessOps without typing anything. Which session survived, and which redirect was missing?

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 attacker captures the authorization code out of a browser history on a shared machine. What stops them exchanging it for tokens, and what would have failed to stop them without PKCE?
- ✓ Your provider issues group claims that look exactly like the AccessOps permission names. Why does the PermissionService still resolve permissions itself instead of copying those claims straight into the SessionContext?
- ✓ A user clicks Sign out, closes nothing, and immediately clicks Sign in again — and lands back inside AccessOps without typing anything. Which session survived, and which redirect was missing?

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.

- ✓ **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."*
- ✓ **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-04.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."*



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.