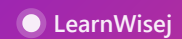


AI Coding Exercise

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.



Same module, built the way you will actually build things.

Every other section of this module explains how "Protecting Data, Uploads, Downloads and Handlers" works. This one asks you to produce it — by directing an assistant with the module's own specification, then verifying the result runs and does what the lab brief describes. It is the workflow you will use on real work, practised on something whose right answer you already know.

What you are asking for

Secure the AccessOps documents feature: uploads limited by size and type with content sniffing, stored outside the web root with a random name and linked to the owning department, downloads that go through DocumentService authorization and Application.Download, a custom handler for a signed share link that expires in one hour, and a review that every query is parameterised and every user-provided string is encoded before display.

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

- ✓ Validated upload pipeline with quarantine folder
- ✓ Authorized download through the service
- ✓ Signed, expiring share-link handler
- ✓ Parameterised query and output encoding review
- ✓ Test with a forged download request

Why it is shaped this way

The confidential documents area is the part of AccessOps an attacker actually wants: salary letters, disciplinary files, signed contracts, board packs. Every other screen moves rows between grids; this one moves bytes across the boundary to a browser you do not control, and back. Identity and permissions were settled in the earlier modules, and this one is about the places where a file, a URL or a string walks around them.

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-streamed download	Calling <code>Application.Download</code> from server code streams a file to the browser during an event you already authorized, so it has no address to bookmark, share or guess.
static file exposure	Anything under the web root is served before a line of your code runs, so the file's permissions are exactly "whoever knows the URL".
Uploaded event	The Upload control raises it on the server with the incoming file streams — the only place a file can be inspected before it joins your system.
content signature check	Comparing a file's first bytes with the format's known header, because the extension and the declared content type are supplied by the client.
quarantine storage	Holding the accepted upload outside the web root under a generated name until the scan is clean, and only then linking it to the owning department.
custom HTTP handler	A server class registered with the application that receives raw requests outside the event flow, so it has no session identity and must authenticate and validate everything itself.
signed expiring link	A URL carrying a document id, an expiry and an HMAC computed over both with a server-held key, so the values cannot be edited and the link stops working on its own.

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 5 is about. You can open [module-05.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 5 – "Protecting Data, Uploads, Downloads and Handlers" – 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-05.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: Secure the AccessOps documents feature: uploads limited by size and type with content sniffing, stored outside the web root with a random name and linked to the owning department, downloads that go through `DocumentService` authorization and `Application.Download`, a custom handler for a signed share link that expires in one hour, and a review that every query is parameterised and every user-provided string is encoded before display.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Validated upload pipeline with quarantine folder
 - Authorized download through the service
 - Signed, expiring share-link handler
 - Parameterised query and output encoding review
 - Test with a forged download 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:

- Your upload handler checks the extension against an allow-list and the file passes. Explain what an attacker can still send, and what the signature check adds that the extension check cannot.
- A manager forwards a share link to somebody outside the company before it expires. Which parts of your design limit the damage, and what would you have to add to revoke that single link without changing the key for everyone?
- Your download button is hidden for users without the read permission, and the forged request test still passes. Explain precisely which line of code caused it to pass, and why the hidden button had nothing to do with it.

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 upload handler checks the extension against an allow-list and the file passes. Explain what an attacker can still send, and what the signature check adds that the extension check cannot.
- ✓ A manager forwards a share link to somebody outside the company before it expires. Which parts of your design limit the damage, and what would you have to add to revoke that single link without changing the key for everyone?
- ✓ Your download button is hidden for users without the read permission, and the forged request test still passes. Explain precisely which line of code caused it to pass, and why the hidden button had nothing to do with it.

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.

- ✓ **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-05.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."*



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.