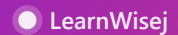


AI Coding Exercise

Build this module's EnterpriseOps 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 — including against our own reference build on GitHub.



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

Every other section of this module explains how "High-Volume Data UX, Server Filtering, and Batch Operations" 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

Create an Enterprise Work Queue screen with server-side filters, saved views, paging, sort persistence, and a batch reassignment action. Add progress and a per-row result report for partial failures.

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

- ✓ Paged query service
- ✓ Search projection model
- ✓ Saved view definition
- ✓ Batch reassignment workflow
- ✓ Performance notes

Why it is shaped this way

Enterprise users expect fast grids and reliable bulk actions. Loading everything into memory or rebinding blindly works in demos but fails under realistic data volume.

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 paging	The database returns only the requested page of rows, so a grid can browse millions of records without loading them into memory or the browser.
search projection	A lean read model containing only the columns a search screen displays, instead of full entities with every navigation property.
grid state	The current page, sort, filter, and selection of a grid, kept on the server so it survives refreshes and can be restored predictably.
virtualized thinking	Designing every list and grid as if the data set is unbounded, so rows are fetched, rendered, and released on demand.
batch command	One server operation that applies a change to many selected records at once, with validation and transaction handling done in one place.
partial failure	When some items in a batch succeed and others fail. The operation must report exactly which ones failed and why, rather than an all-or-nothing error.
responsive feedback	Progress indicators, disabled controls, and status messages that keep users informed during long operations instead of leaving the screen frozen.

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 EnterpriseOps 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 – "High-Volume Data UX, Server Filtering, and Batch Operations" – of the LearnWisej Enterprise Wisej.NET: Architecture to Cloud 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/azure-docker/specs/module-05.md>

Then build a complete Wisej.NET 4 project I can open in Visual Studio:

1. Project "EnterpriseOps", 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 an Enterprise Work Queue screen with server-side filters, saved views, paging, sort persistence, and a batch reassignment action. Add progress and a per-row result report for partial failures.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Paged query service
 - Search projection model
 - Saved view definition
 - Batch reassignment workflow
 - Performance notes
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:

- How many records are loaded for the first screen?
- What happens when one batch row fails?
- Can the user repeat the search after refresh?

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:

- ✓ How many records are loaded for the first screen?
- ✓ What happens when one batch row fails?
- ✓ Can the user repeat the search after refresh?

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.



We already built this one — compare yours with ours

The Module 5 reference implementation lives in the public samples repository. Build yours first, then open ours: the differences are the interesting part.



[Open the Module 5 sample on GitHub](#)

Or clone the whole set with `git clone https://github.com/iceteagroup/LearnWisej-Samples` and open `Enterprise Wisej.NET Course/Module 5`.



Our sample is a starting point, not a finished product

It was built the same way you are building yours — by prompt, then corrected — and it deliberately stops where the module stops. Expect stand-ins where production code would go, thin error handling and no tests. Treat that as an invitation: use what you learned in this module, plus the assistant, to push it further. Improving code you did not write is the skill this course is actually teaching.

Three prompts to take it further

Point your assistant at the project you just built — or at ours 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/azure-docker/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 EnterpriseOps 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 EnterpriseOps 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.