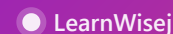


AI Coding Exercise

Build this module's OpsMonitor 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 "Data Controls, Browser Payloads and Large UI Surfaces" 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

Shrink the two largest surfaces in WisejPerfLab. The ticket grid binds 50,000 full entities with navigation properties: replace the binding with a TicketGridRow projection of just the visible columns, set VirtualMode and RowCount and serve cells from a CellValueNeeded handler backed by a paged query service. The customer tree builds every node at startup: load children on expand instead, using a count placeholder for collapsed branches. Measure each screen before and after with CPU Usage and .NET Object Allocation on the server and the browser network panel for update payload size.

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

- ✓ TicketGridRow projection bound instead of full entities, with only the displayed columns
- ✓ Virtual-mode grid with RowCount and a CellValueNeeded handler served from a paged query service
- ✓ Customer tree loading child nodes on expand with a count placeholder for collapsed branches
- ✓ Before/after CPU and allocation numbers for the grid load and the tree expand scenarios
- ✓ Browser network evidence comparing update payload size before and after, with the caveat about compression

Why it is shaped this way

Large UI surfaces are the most common source of Wisej.NET performance problems, and they are the bucket the profiler cannot fully see. The WisejPerfLab ticket grid binds fifty thousand entities with their navigation properties; the customer tree builds every node at startup. On a desktop that pattern was acceptable, because the control and the data lived in the same process as the screen. Here the same code pays four times: server memory to hold the objects, server CPU to build and serialize the UI state, network to carry the update, and browser time to render it. This module is where the best optimization is usually not a faster algorithm but less UI existing at once.

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
projection row	A small type holding only the columns a screen displays, bound instead of the entity graph, so neither the server nor the browser carries data nobody looks at.
virtual mode	The grid holds a row count rather than row objects and asks for cell values as they are needed, which removes the per-row objects and the rendering of rows nobody scrolled to.
CellValueNeeded	The event that supplies a value on demand in virtual mode; it runs very often, so it is a hot path by construction and must return quickly.
server paging and filtering	Asking the database for the page the user is looking at, instead of loading everything and filtering in UI code.
lazy node loading	Building tree children when a branch is expanded, with a count placeholder while it is collapsed, instead of materializing the whole hierarchy at startup.
update payload	The serialized difference the server sends after a handler; large or frequent payloads cost network and browser time that no Visual Studio tool reports.
compression is not free work	Wisej.NET compresses HTTP and WebSocket traffic, which shrinks bytes but does not make an unnecessary update cheap to produce, transmit or render.

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 OpsMonitor 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 – "Data Controls, Browser Payloads and Large UI Surfaces" – of the LearnWisej Performance & Profiling 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/performance-and-profiling/specs/module-05.md>

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

1. Project "OpsMonitor", 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: Shrink the two largest surfaces in `WisejPerflab`. The ticket grid binds 50,000 full entities with navigation properties: replace the binding with a `TicketGridRow` projection of just the visible columns, set `VirtualMode` and `RowCount` and serve cells from a `CellValueNeeded` handler backed by a paged query service. The customer tree builds every node at startup: load children on expand instead, using a count placeholder for collapsed branches. Measure each screen before and after with CPU Usage and .NET Object Allocation on the server and the browser network panel for update payload size.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `TicketGridRow` projection bound instead of full entities, with only the displayed columns
 - Virtual-mode grid with `RowCount` and a `CellValueNeeded` handler served from a paged query service
 - Customer tree loading child nodes on expand with a count placeholder for collapsed branches
 - Before/after CPU and allocation numbers for the grid load and the tree expand scenarios
 - Browser network evidence comparing update payload size before and after, with the caveat about compression
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:

- Virtual mode removed the row objects but your CPU trace barely improved. What does that tell you about where the cost actually was, and which measurement would you look at next?
- Your `CellValueNeeded` handler is correct but slow under fast scrolling. List three things it could be doing that a per-cell handler must never do.
- The compressed payload for the tree expand barely changed after lazy loading, but users say the screen feels faster. Give a plausible explanation that the byte count alone would hide.

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:

- ✓ Virtual mode removed the row objects but your CPU trace barely improved. What does that tell you about where the cost actually was, and which measurement would you look at next?
- ✓ Your CellValueNeeded handler is correct but slow under fast scrolling. List three things it could be doing that a per-cell handler must never do.
- ✓ The compressed payload for the tree expand barely changed after lazy loading, but users say the screen feels faster. Give a plausible explanation that the byte count alone would hide.

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 `Performance and Profiling 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.

- ✓ **Make it survive a bad day.** *"Review this OpsMonitor 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 OpsMonitor 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/performance-and-profiling/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."*



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.