

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.

 LearnWisej



You are not going to type this sample. You are going to direct it.

Almost nobody writes Wisej.NET code from a blank file any more — an assistant writes the first draft and the developer decides whether it is right. That only works if you know the material, which is what the rest of this module gave you. Here you hand that knowledge to ChatGPT or Claude, get this module's sample back as a working Visual Studio solution, and review it the way you would review a colleague's pull request.

What you are asking for

Create the WisejPerfLab solution with a support dashboard, a ticket grid and a customer tree fed by seeded data, then make it measurable. Add a ScenarioProbe service that wraps a scenario in a Stopwatch scope and writes structured PERF start/end logs with the scenario name, the user action, the elapsed milliseconds and the row count, register it in the host builder, and wrap three scenarios in it: dashboard refresh, ticket search and customer tree expand. Run each scenario once to warm up, then record the baseline and write a performance budget that states the environment, build configuration, dataset size and an acceptance threshold per scenario.

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

- ✓ WisejPerfLab solution with a dashboard, ticket grid and customer tree over seeded data
- ✓ ScenarioProbe service writing structured PERF start/end logs with elapsed milliseconds and row counts
- ✓ Three instrumented scenarios: dashboard refresh, ticket search and customer tree expand
- ✓ Baseline note recording environment, build configuration, dataset size and warm-up procedure
- ✓ Performance budget with an acceptance threshold per scenario and the tool that would prove each one

Why it is shaped this way

A support team reports that WisejPerfLab "got slow". That sentence contains no measurement, no scenario and no layer. The dashboard might be waiting on SQL, the refresh handler might be formatting fifty thousand cells, the browser might be repainting a grid that should never have been rendered in full, or the server might simply be holding two hundred sessions worth of forms that were closed an hour ago. Every one of those has a different fix and a different Visual Studio tool, and a developer who opens the profiler before deciding which one it is will collect a trace that cannot answer the question. This module builds the

mental model first: what actually happens between a click in the browser and a repainted screen, which of those steps costs what, and how to write down a scenario and a budget so the next six modules have something to measure against.

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 UI	Controls, their properties and their state live in the server process and the browser renders what the server describes, so every UI object you create is server memory and every UI mutation is server CPU plus a payload.
event round-trip	The full path of one user action: browser event, transport to the server, your handler, data access, control mutations, the serialized update back, browser repaint — the unit a user actually feels.
session as the unit of capacity	A session owns pages, forms, controls, timers, selections, bindings and cached data; capacity questions are answered per session, not per request, because that memory stays resident between clicks.
WebSocket push	The channel through which the server sends updates the user did not ask for; it makes live data cheap to deliver and makes an unthrottled background task expensive, because every intermediate state becomes a frame.
the five buckets	CPU in your code, blocking and waiting, allocation and retention, data access and I/O, and update payload — the classification that decides which profiler tool can even see your problem.
latency vs throughput	Latency is how long one user waits; throughput is how many sessions the server sustains. A fix that removes a wait helps latency; a fix that removes work or memory helps both, and some latency fixes cost throughput.
scenario and baseline	A named, repeatable sequence of user actions plus the environment, build configuration, dataset size and the measured numbers it produced — without one, a trace is noise and a later comparison is meaningless.

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 1 is about. You can open [module-01.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



Use Claude

Opens [claude.ai](#) with the same prompt loaded. Claude Code users can paste it into

Codex; any plan that can browse the web will fetch the spec.

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 1 – "The Wisej.NET Performance Model" – 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-01.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: Create the WisejPerfLab solution with a support dashboard, a ticket grid and a customer tree fed by seeded data, then make it measurable. Add a ScenarioProbe service that wraps a scenario in a Stopwatch scope and writes structured PERF start/end logs with the scenario name, the user action, the elapsed milliseconds and the row count, register it in the host builder, and wrap three scenarios in it: dashboard refresh, ticket search and customer tree expand. Run each scenario once to warm up, then record the baseline and write a performance budget that states the environment, build configuration, dataset size and an acceptance threshold per scenario.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - WisejPerfLab solution with a dashboard, ticket grid and customer tree over seeded data
 - ScenarioProbe service writing structured PERF start/end logs with elapsed milliseconds and row counts
 - Three instrumented scenarios: dashboard refresh, ticket search and customer tree expand
 - Baseline note recording environment, build configuration, dataset size and warm-up procedure
 - Performance budget with an acceptance threshold per scenario and the tool that would prove each one
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 dashboard refresh logs 1,900 ms but the server's CPU barely moves during the scenario. Which of the five buckets does that rule out, which does it point at, and what is the cheapest next measurement?
- The probe measures the handler. Name two costs the user experiences on the same click that your `elapsedMs` number does not include, and say how you would measure each.
- Why does the baseline note have to record the dataset size and the build configuration, when the scenario name and the elapsed time seem to be the interesting part?

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 dashboard refresh logs 1,900 ms but the server's CPU barely moves during the scenario. Which of the five buckets does that rule out, which does it point at, and what is the cheapest next measurement?
- ✓ The probe measures the handler. Name two costs the user experiences on the same click that your elapsedMs number does not include, and say how you would measure each.
- ✓ Why does the baseline note have to record the dataset size and the build configuration, when the scenario name and the elapsed time seem to be the interesting part?

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



[Open the Module 1 sample on GitHub](#)

Or clone the whole set with `git clone https://github.com/iceteagroup/LearnWisej-Samples` and open `Performance and Profiling Course/Module 1`.



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.

- ✓ **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-01.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 OpsMonitor 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.