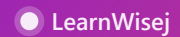


AI Coding Exercise

Build this module's VisualOperationsStudio 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.



The assistant is the fast pair of hands. You are still the architect.

An assistant can produce this module's sample in minutes. What it cannot do is decide whether the result is the right design for your application — that call stays with you. You supply the judgement: brief it properly, then check what comes back against what Module 7 just taught you.

What you are asking for

Complete the VisualOperationsStudio capstone by putting one model behind all four surfaces. Restructure the project into Models, Geometry, Renderers and Wisej adapter folders so the painted gauge, the user-painted grid cells, the Canvas topology and the PNG export all read the same OperationsModel and share the pure geometry layer, with no business state duplicated per renderer. Add a RenderMetrics helper that records render duration, output bitmap size and the number of scene objects drawn, and use it to compare the server-painted gauge against the off-screen renderer before and after one optimisation you choose and justify. Exercise the failure paths — empty data, a zero-sized surface, an invalid range, a missing font — so each degrades to a readable placeholder rather than an exception, add an accessible table that lists every value the graphics show, and write the capstone notes covering the architecture, the browser/server boundary and the test matrix.

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

- ✓ Models, Geometry, Renderers and adapter layers with one OperationsModel driving all four surfaces
- ✓ RenderMetrics helper recording render duration, bitmap size and drawn object count per surface
- ✓ Before-and-after measurements for one optimisation, with the reasoning for choosing it
- ✓ Failure paths for empty data, zero size, invalid range and a missing font that degrade instead of throwing
- ✓ CapstoneNotes.md with the architecture, the server/browser boundary, the test matrix and an accessible data table

Why it is shaped this way

VisualOperationsStudio now owns four rendering surfaces, and each was built in its own module with its own private idea of what a machine's health means. That is how a drawing subsystem rots: somebody moves a warning threshold, three surfaces agree, and the exported PNG quietly contradicts the screen it came from. This module collapses those four interpretations into one model behind one geometry layer,

then proves with measurements, deliberate failure paths and a semantic equivalent that the result survives real data, real platforms and real users.

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
Visual model	Semantic state only: readings, captions, nodes and edges, trend series, severity as a domain choice and the shared selection, so no surface needs a copy.
Geometry layer	Mostly pure maths turning model state plus the current surface size into rectangles, sweep angles, label positions and hit regions, testable with no control present.
Surface adapter	One renderer per surface rather than a common interface: System.Drawing.Graphics and Wisej.Web.Canvas keep their own primitives above a shared geometry.
Performance budget	A number agreed before tuning, such as a painted status control under 5 ms and a full Canvas redraw inside roughly 16 to 50 ms.
Render instrumentation	Recorded elapsed time, bitmap dimensions, drawn object count and payload growth per surface, so an optimisation is argued from a before and an after.
Golden-image test	A fixed-input off-screen render compared with a stored reference using a tolerance, because anti-aliasing and encoders differ across platforms.
Semantic equivalent	The label, AccessibleDescription, tooltip or companion list carrying every essential value the graphics show, so nothing important is pixels-only.

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 7 is about. You can open [module-07.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 VisualOperationsStudio 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 7 – "Production Drawing Architecture, Performance and Capstone" – of the LearnWisej Drawing & Painting in Wisej.NET 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/drawing-and-painting/specs/module-07.md>

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

1. Project "VisualOperationsStudio", 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: Complete the VisualOperationsStudio capstone by putting one model behind all four surfaces. Restructure the project into Models, Geometry, Renderers and Wisej adapter folders so the painted gauge, the user-painted grid cells, the Canvas topology and the PNG export all read the same OperationsModel and share the pure geometry layer, with no business state duplicated per renderer. Add a RenderMetrics helper that records render duration, output bitmap size and the number of scene objects drawn, and use it to compare the server-painted gauge against the off-screen renderer before and after one optimisation you choose and justify. Exercise the failure paths – empty data, a zero-sized surface, an invalid range, a missing font – so each degrades to a readable placeholder rather than an exception, add an accessible table that lists every value the graphics show, and write the capstone notes covering the architecture, the browser/server boundary and the test matrix.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Models, Geometry, Renderers and adapter layers with one OperationsModel driving all four surfaces
 - RenderMetrics helper recording render duration, bitmap size and drawn object count per surface
 - Before-and-after measurements for one optimisation, with the reasoning for choosing it
 - Failure paths for empty data, zero size, invalid range and a missing font that degrade instead of throwing
 - CapstoneNotes.md with the architecture, the server/browser boundary, the test matrix and an accessible data table
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 geometry layer takes the model and a size and returns shapes, yet one teammate wants it to return a GraphicsPath so the gauge renderer gets shorter. What does that cost the Canvas renderer and the unit tests, and where would you draw the line instead?

- The export PNG and the on-screen Canvas disagree by a few pixels on the Linux build agent but match on Windows. Which measurements and which kind of test would tell you whether that is a defect or an expected platform difference?
- A reviewer says the accessible table is redundant because every gauge already has a tooltip. Explain what the table gives a keyboard-only or screen-reader user that the tooltip does not, and what that implies for the colour-only severity chips in the grid.

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 geometry layer takes the model and a size and returns shapes, yet one teammate wants it to return a GraphicsPath so the gauge renderer gets shorter. What does that cost the Canvas renderer and the unit tests, and where would you draw the line instead?
- ✓ The export PNG and the on-screen Canvas disagree by a few pixels on the Linux build agent but match on Windows. Which measurements and which kind of test would tell you whether that is a defect or an expected platform difference?
- ✓ A reviewer says the accessible table is redundant because every gauge already has a tooltip. Explain what the table gives a keyboard-only or screen-reader user that the tooltip does not, and what that implies for the colour-only severity chips in the grid.

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.

- ✓ **Prove it actually works.** *"Go through this project deliverable by deliverable against the module specification at <https://learnwisej.com/assets/courses/drawing-and-painting/specs/module-07.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 VisualOperationsStudio 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 VisualOperationsStudio 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.