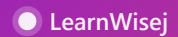


AI Coding Exercise

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.



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

Build the VisualOperationsStudio shell and render one telemetry reading on four surfaces at once. Create a VisualOperationsPage holding a single TelemetrySample model instance, then present its value as (1) a plain Label plus ProgressBar, (2) a Panel with a Paint handler that draws a bar from PaintEventArgs.Graphics and e.ClipRectangle, (3) a Wisej.Web.Canvas whose Redraw handler rebuilds the same bar from the model, and (4) an off-screen Bitmap rendered with Graphics.FromImage and shown in a PictureBox. Wire a button that changes the model value and refreshes all four surfaces through Invalidate() and a Canvas redraw, then resize the browser and record which surfaces survive and which have to be reconstructed.

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

- ✓ VisualOperationsPage holding one TelemetrySample model that every surface reads from
- ✓ Paint handler drawing from PaintEventArgs.Graphics inside e.ClipRectangle
- ✓ Canvas Redraw handler that rebuilds the whole scene from the model after a resize
- ✓ Off-screen Bitmap rendered with Graphics.FromImage and displayed in the page
- ✓ SurfaceNotes.md recording what crosses the network per surface and which visuals survive a resize

Why it is shaped this way

VisualOperationsStudio has to show the same machine reading in four places: a dial on the operations page, a health bar inside a grid row, a node on a topology map, and a PNG attached to the nightly shift report. Each one is "draw a number", yet reaching for whichever drawing API you remember from the desktop produces four pieces of code with wildly different CPU, bandwidth and resize behaviour. This module puts the choice of surface before the drawing code, because the surface decides where the pixels are produced and what travels to the browser.

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
Paint / PaintEventArgs	The control-level painting event: your C# runs on the server against the Graphics it supplies, inside the ClipRectangle it reports, and Wisej.NET turns the result into an image for the browser.
DataGridView.CellPaint	The same painting mechanism scoped to one cell; its DataGridViewCellPaintEventArgs derives from PaintEventArgs and adds RowIndex and ColumnIndex.
UserPaint	The switch a cell or a whole column needs before the grid raises CellPaint for it, which keeps painting opt-in per column instead of grid-wide.
Wisej.Web.Canvas	An immediate-mode 2D browser surface mirroring the HTML5 CanvasRenderingContext2D model; the drawing calls are executed by the client, not rendered into a picture on the server.
Redraw	The Canvas event in which the whole scene is reconstructed from your model, because the canvas keeps pixels and loses them when it is recreated or resized.
System.Drawing.Managed	Off-screen rendering with no visible control: create a Bitmap, get its Graphics from Graphics.FromImage(), and keep the image bytes for a report, an export or a download.
Invalidate()	The call that tells a painted control its visual is stale; it updates the client widget and the Visual Studio designer surface, so it belongs where state changed, not inside drawing helpers.

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 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 1 – "The Wisej.NET Drawing Architecture" – 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-01.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: Build the VisualOperationsStudio shell and render one telemetry reading on four surfaces at once. Create a VisualOperationsPage holding a single TelemetrySample model instance, then present its value as (1) a plain Label plus ProgressBar, (2) a Panel with a Paint handler that draws a bar from `PaintEventArgs.Graphics` and `e.ClipRectangle`, (3) a `Wisej.Web.Canvas` whose Redraw handler rebuilds the same bar from the model, and (4) an off-screen Bitmap rendered with `Graphics.FromImage` and shown in a PictureBox. Wire a button that changes the model value and refreshes all four surfaces through `Invalidate()` and a Canvas redraw, then resize the browser and record which surfaces survive and which have to be reconstructed.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - VisualOperationsPage holding one TelemetrySample model that every surface reads from
 - Paint handler drawing from `PaintEventArgs.Graphics` inside `e.ClipRectangle`
 - Canvas Redraw handler that rebuilds the whole scene from the model after a resize
 - Off-screen Bitmap rendered with `Graphics.FromImage` and displayed in the page
 - SurfaceNotes.md recording what crosses the network per surface and which visuals survive a resize
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:

- You set the model's reading from the button but forget the `Invalidate()` call on the painted panel. The label and the canvas update. Explain exactly why the panel does not, and what the user is looking at until something else forces it to repaint.
- Your first Canvas version draws the bar once when the page loads instead of from Redraw. Describe what the user sees after resizing the browser, and why the off-screen Bitmap in the fourth area behaves differently.
- Your reviewer asks why the first area – the plain Label and ProgressBar – is in the lab at all. Answer with the costs the other three surfaces pay that it does not, and name one requirement that would justify paying them.

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:

- ✓ You set the model's reading from the button but forget the `Invalidate()` call on the painted panel. The label and the canvas update. Explain exactly why the panel does not, and what the user is looking at until something else forces it to repaint.
- ✓ Your first Canvas version draws the bar once when the page loads instead of from Redraw. Describe what the user sees after resizing the browser, and why the off-screen Bitmap in the fourth area behaves differently.
- ✓ Your reviewer asks why the first area — the plain Label and ProgressBar — is in the lab at all. Answer with the costs the other three surfaces pay that it does not, and name one requirement that would justify paying them.

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 VisualOperationsStudio 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/drawing-and-painting/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 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."*



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.