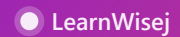


AI Coding Exercise

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.



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

Every other section of this module explains how "Interactive Canvas: State, Input, Hit Testing and Performance" 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

Turn the Canvas playground into a topology editor for VisualOperationsStudio. Create NodeModel and EdgeModel classes with stable IDs and world-space geometry, hold the scene in a TopologyScene object stored per session rather than in a static field, and keep RenderScene as the only method that draws. Add ToWorld and ToScreen conversions for the current pan offset and zoom factor, then handle MouseDown to hit-test node bounds in world coordinates and set Selected, MouseMove to drag the selected node or pan the viewport, and MouseUp to end the gesture — each of them changing state and then calling RenderScene. Add wheel or button zoom about the pointer, skip nodes outside the visible rectangle when rendering, give keyboard users a way to select and nudge a node, and record the request traffic a drag produces plus the point at which you would move the loop to client JavaScript.

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

- ✓ NodeModel, EdgeModel and a per-session TopologyScene with stable IDs and no mutable static state
- ✓ ToWorld and ToScreen conversions used for hit testing so selection stays correct after pan and zoom
- ✓ MouseDown, MouseMove and MouseUp handlers that change state and then call the single RenderScene method
- ✓ Viewport culling that skips nodes outside the visible rectangle, plus a keyboard path for selecting and nudging
- ✓ InteractionNotes.md with the request traffic measured during a drag and the client-side cutoff you would apply

Why it is shaped this way

Module 4 left VisualOperationsStudio with a Canvas playground: a DrawPlayground method that the Redraw event calls, rebuilding every path, gradient and transform from scratch. It is correct, and it is

completely inert. Operations staff want to click a pump, drag it somewhere sensible, pan across the plant and zoom into a cluster — and a drawing method cannot answer any of those requests, because the browser canvas holds pixels and pixels do not know which shape is a pump. The moment a surface becomes interactive it stops being a renderer and becomes a small state machine that happens to render.

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
Model, Input, State, Render	Four layers with one rule between them: input changes state, and only then does a single render method turn the current state into drawing calls.
Scene objects	C# classes such as NodeModel and EdgeModel that carry the stable ID, world-space bounds, status and Selected flag the pixels can only suggest.
Per-session scene state	The scene belongs to one user's session; a mutable static list would give every visitor the same nodes and the same selection.
Control input events	Canvas derives from Control, so mouse down/move/up, click, double click and the touch and gesture events such as TouchStart, Tap, Pinch, Rotate, Swipe and Track are all available.
Model-based hit testing	The documented Wisej Canvas API exposes no <code>isPointInPath</code> or <code>isPointInStroke</code> , so "what did the user click?" is answered by comparing a point against geometry you own.
Screen-to-world conversion	If rendering applies $\text{screen} = \text{world} \times \text{zoom} + \text{pan}$, then every incoming pointer coordinate must be run back through the inverse before it is compared with anything in the model.
Viewport culling	Skipping scene objects whose bounds do not intersect the visible world rectangle, so a large topology costs only what is actually on screen.

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 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 5 – "Interactive Canvas: State, Input, Hit Testing and Performance" – 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-05.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: Turn the Canvas playground into a topology editor for VisualOperationsStudio. Create `NodeModel` and `EdgeModel` classes with stable IDs and world-space geometry, hold the scene in a `TopologyScene` object stored per session rather than in a static field, and keep `RenderScene` as the only method that draws. Add `ToWorld` and `ToScreen` conversions for the current pan offset and zoom factor, then handle `MouseDown` to hit-test node bounds in world coordinates and set `Selected`, `MouseMove` to drag the selected node or pan the viewport, and `MouseUp` to end the gesture – each of them changing state and then calling `RenderScene`. Add wheel or button zoom about the pointer, skip nodes outside the visible rectangle when rendering, give keyboard users a way to select and nudge a node, and record the request traffic a drag produces plus the point at which you would move the loop to client JavaScript.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `NodeModel`, `EdgeModel` and a per-session `TopologyScene` with stable IDs and no mutable static state
 - `ToWorld` and `ToScreen` conversions used for hit testing so selection stays correct after pan and zoom
 - `MouseDown`, `MouseMove` and `MouseUp` handlers that change state and then call the single `RenderScene` method
 - Viewport culling that skips nodes outside the visible rectangle, plus a keyboard path for selecting and nudging
 - `InteractionNotes.md` with the request traffic measured during a drag and the client-side cutoff you would apply
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:

- A colleague reports that selection is fine until they zoom in, after which clicks land on the wrong pump or on nothing. Nothing about the drawing code changed. Which single conversion is missing, and why does the defect stay invisible at zoom 1.0 with no pan?

- Two operators open the topology at the same time and each sees the other's node move under their cursor. What did the scene get stored in, and what does that tell you about where interactive state belongs in a server-side web framework?
- Your drag produces a request for every pointer move. At what measured point would you stop optimising the server loop and move the interaction into client JavaScript instead, and what would still have to stay authoritative on the server?

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:

- ✓ A colleague reports that selection is fine until they zoom in, after which clicks land on the wrong pump or on nothing. Nothing about the drawing code changed. Which single conversion is missing, and why does the defect stay invisible at zoom 1.0 with no pan?
- ✓ Two operators open the topology at the same time and each sees the other's node move under their cursor. What did the scene get stored in, and what does that tell you about where interactive state belongs in a server-side web framework?
- ✓ Your drag produces a request for every pointer move. At what measured point would you stop optimising the server loop and move the interaction into client JavaScript instead, and what would still have to stay authoritative on the server?

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.

- ✓ **Make it survive a bad day.** *"Review this VisualOperationsStudio 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 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-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.