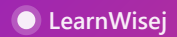


AI Coding Exercise

Build this module's AdaptiveOps 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.



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 Adaptive Operations Console solution and its first desktop shell: a main Page with a top toolbar panel, a left navigation panel, a fill workspace, a right details panel and a bottom status panel built with Dock and nested Panels. Choose a built-in base theme and record the choice, write a VisualInventory.md that lists the controls needing custom appearances, the named color and font tokens, the layout regions and the desktop, tablet and phone breakpoints, and add a status label that reports the current browser width so later modules can verify profiles.

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

- ✓ Adaptive Operations Console solution with a docked desktop shell of toolbar, navigation, workspace, details and status panels
- ✓ Base theme chosen and configured, with the reason recorded in the lab notes
- ✓ VisualInventory.md listing controls that need appearances, theme tokens, layout regions and breakpoints
- ✓ Status label that reports the current browser width
- ✓ Desktop screenshot and a note explaining which layer owns each visual decision

Why it is shaped this way

A team that treats Wisej.NET as a CSS-first framework ends up with BackColor set on two hundred controls, selectors that break on the next upgrade, and Resize handlers that recompute Bounds by hand; when a phone version is requested, the only option left is a rewrite. This module fixes where a control's look and layout really come from, names the three customization layers and which one owns each decision, and starts the Adaptive Operations Console shell that every later module extends.

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 control and client widget	You write .NET controls on the server; the browser displays JavaScript widgets that Wisej.NET keeps in sync, so a visual decision can live on either side of that line.
theme	A single JSON file of colors, images, fonts, appearances, states, styles, properties and settings that gives every standard control its identity; the contract between the two worlds.
appearance and AppearanceKey	Each control exposes an appearance key; the active theme holds an appearance for that key, and a semantic key such as action-button selects a variant.
state	A named interaction condition inside an appearance (default, hovered, pressed, focused, invalid, selected) whose styles and properties apply while it is active.
CssClass and CssStyle	Direct browser styling: a class from a scoped application stylesheet, or a raw style string on the widget element for a rare one-off value.
layout engine and Dock	Position and size come from the container hierarchy, not from the theme; docking stacks regions against the parent's edges and fills what is left.
visual inventory	The written list of controls needing appearances, color and font tokens, layout regions and breakpoints that the theme, layout and profile work is planned from.

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 AdaptiveOps 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 Visual System" – of the LearnWisej Theming & Responsive Layouts 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/theming-and-responsive-layouts/specs/module-01.md>

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

1. Project "AdaptiveOps", 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 Adaptive Operations Console solution and its first desktop shell: a main Page with a top toolbar panel, a left navigation panel, a fill workspace, a right details panel and a bottom status panel built with Dock and nested Panels. Choose a built-in base theme and record the choice, write a `VisualInventory.md` that lists the controls needing custom appearances, the named color and font tokens, the layout regions and the desktop, tablet and phone breakpoints, and add a status label that reports the current browser width so later modules can verify profiles.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Adaptive Operations Console solution with a docked desktop shell of toolbar, navigation, workspace, details and status panels
 - Base theme chosen and configured, with the reason recorded in the lab notes
 - `VisualInventory.md` listing controls that need appearances, theme tokens, layout regions and breakpoints
 - Status label that reports the current browser width
 - Desktop screenshot and a note explaining which layer owns each visual decision
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:

- The toolbar's primary command needs the brand colour on every screen. Which of the three layers owns that decision, and what would go wrong if you set `BackColor` on the button instead?
- Your width label reads 1024 px and the details panel is still visible. Which mechanism from a later module will decide whether it should be, and why is a CSS media query the wrong first answer in Wisej.NET?
- If the base theme were swapped for a different built-in theme tomorrow, which rows of your `VisualInventory.md` would change and which would not, and what does that tell you about where the inventory's decisions live?

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:

- ✓ The toolbar's primary command needs the brand colour on every screen. Which of the three layers owns that decision, and what would go wrong if you set BackColor on the button instead?
- ✓ Your width label reads 1024 px and the details panel is still visible. Which mechanism from a later module will decide whether it should be, and why is a CSS media query the wrong first answer in Wisej.NET?
- ✓ If the base theme were swapped for a different built-in theme tomorrow, which rows of your VisualInventory.md would change and which would not, and what does that tell you about where the inventory's decisions live?

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 `Theming and Responsive Layouts 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 AdaptiveOps 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/theming-and-responsive-layouts/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 AdaptiveOps 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.