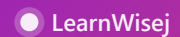


AI Coding Exercise

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.



Writing the code is the easy half now.

Getting working code for "Layout Fundamentals: Docking, Anchoring, AutoSize, AutoScroll, and Shell Composition" out of a modern assistant takes one good prompt. Judging it, correcting it and extending it takes everything this module covered. That is the split this exercise is built around — you supply the specification and the review, the assistant supplies the typing.

What you are asking for

Refactor the Adaptive Operations Console shell into nested containers: a top toolbar, a left navigation rail, a fill workspace, a right details panel and a status region, each a Panel or UserControl with its own local layout. Delete every Resize handler that sets Bounds, fix docking order in the designer, use Padding for gaps between docked regions, give the workspace and grid a MinimumSize, and make the details editor scroll with AutoScroll and hidden scrollbars at narrow widths.

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

- ✓ Shell rebuilt from nested containers with toolbar, navigation, workspace, details and status regions
- ✓ All manual Resize code that set Bounds removed
- ✓ Docking order, Padding and MinimumSize set intentionally on every region
- ✓ Details editor scrolling through AutoScroll with hidden scrollbars at narrow widths
- ✓ Navigation and details regions extracted into UserControls

Why it is shaped this way

The Adaptive Operations Console has a theme and scoped CSS now, but its shell still moves controls by hand: a Resize handler recomputes Bounds for the toolbar, the navigation rail, the ticket grid and the details editor every time the browser changes size. It works at the width the developer tested, breaks at every other one, and cannot survive the tablet and phone profiles that Module 6 introduces. Adaptive design in Wisej.NET depends on containers and layout rules, not coordinates: regions that claim edges, a workspace that fills what is left, and editors that scroll instead of overlapping.

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
default layout engine	Every Wisej.NET control has a layout engine; the default one supports absolute position, docking and anchoring, so a screen can mix precise designer placement with regions that resize themselves.
docking priority	Docked children consume space in the order they appear in the parent's control collection, which is why the same five panels produce a correct shell or an overlapping one depending on child order.
DisplayRectangle and Padding	Docked controls lay out inside the parent's DisplayRectangle, which Padding reduces; the default engine ignores Margin for docked children, so gaps come from padding or spacer controls.
anchoring	An anchor keeps a control's distance from selected parent sides, which suits form-like regions: labels top-left, text boxes left-right, command buttons bottom-right.
AutoSize and circular layout	A container that sizes itself from children that are docked or anchored back to it has no stable answer; that loop is the module's main AutoSize warning.
AutoScroll with hidden bars	AutoScroll lets a container scroll when content does not fit, and ScrollBars.Hidden keeps touch scrolling on phones without drawing scrollbars.
UserControl region	A shell region with its own local layout and a small public surface, so the shell shows, hides, docks or moves the whole region per profile instead of knowing its internal coordinates.

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 4 is about. You can open [module-04.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 4 – "Layout Fundamentals: Docking, Anchoring, AutoSize, AutoScroll, and Shell Composition" – 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-04.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: Refactor the Adaptive Operations Console shell into nested containers: a top toolbar, a left navigation rail, a fill workspace, a right details panel and a status region, each a Panel or UserControl with its own local layout. Delete every Resize handler that sets Bounds, fix docking order in the designer, use Padding for gaps between docked regions, give the workspace and grid a MinimumSize, and make the details editor scroll with AutoScroll and hidden scrollbars at narrow widths.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Shell rebuilt from nested containers with toolbar, navigation, workspace, details and status regions
 - All manual Resize code that set Bounds removed
 - Docking order, Padding and MinimumSize set intentionally on every region
 - Details editor scrolling through AutoScroll with hidden scrollbars at narrow widths
 - Navigation and details regions extracted into UserControls
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:

- If the workspace panel were docked before the status panel in the child order, what would the user see at the bottom of the grid, and which designer command fixes it without changing a single size?
- You set a 12-pixel Margin on the navigation rail and the gap did not appear. Why not, and which two mechanisms do produce a gap between docked regions?
- The details editor is a UserControl with AutoScroll. Which of its members must the shell know about to move it under the grid on a tablet, and which must it never need?

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:

- ✓ If the workspace panel were docked before the status panel in the child order, what would the user see at the bottom of the grid, and which designer command fixes it without changing a single size?
- ✓ You set a 12-pixel Margin on the navigation rail and the gap did not appear. Why not, and which two mechanisms do produce a gap between docked regions?
- ✓ The details editor is a UserControl with AutoScroll. Which of its members must the shell know about to move it under the grid on a tablet, and which must it never need?

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



[Open the Module 4 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 4`.



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.

- ✓ **Swap a piece without moving the seams.** *"Pick the part of this AdaptiveOps 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."*
- ✓ **Make it survive a bad day.** *"Review this AdaptiveOps 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 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."*



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.