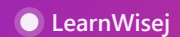


AI Coding Exercise

Build this module's OperationsConsole 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.



Brief it, read it, then break it.

This exercise has three parts, and only the first is about prompting. You brief an assistant with the module specification and get an OperationsConsole solution back; you read what it wrote and decide whether it really does what the module asked; then you push it further than the module asked. The second and third parts are where the learning is.

What you are asking for

Replace the Module 1 placeholders with the real Operations Console shell: a SplitContainer with navigation on the left, a TabControl detail surface on the right, a ToolBar for frequent commands and a StatusBar that shows real status. Move repeated UI into UserControls that expose only meaningful public properties and events, implement a narrow-profile layout with responsive properties or profile-aware logic, and document your docking and anchoring choices in LayoutNotes.md.

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

- ✓ Navigation shell built from SplitContainer, TabControl, ToolBar and StatusBar
- ✓ Repeated UI moved into UserControls with a small public surface
- ✓ Narrow-profile layout using responsive properties or profile-aware logic
- ✓ LayoutNotes.md documenting the docking and anchoring choices
- ✓ Screenshots at desktop and narrow widths

Why it is shaped this way

Containers are the architecture of the user interface: they decide how controls are grouped, resized, navigated and reused. The Module 1 Operations Console still has placeholder panels, and a screen where every control is positioned by hand becomes fragile the moment the browser is resized. Layout problems are usually design problems, not pixel problems, and this module replaces pixel-pushing with a plan: docked regions, a SplitContainer, a TabControl, a real command model and UserControls that hide their internals.

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
docking order	Docking attaches a child to one side of its container or fills what is left, and it is applied in child order, so the same four controls produce different shells depending on the order in which they were added.
anchoring	Anchoring keeps a control's edges tied to its container's edges; anchored left and right it stretches with the width, anchored on neither axis it floats while keeping its relative placement.
SplitContainer	Two adjacent regions the user can resize, the right choice whenever the person, not the developer, should decide how much room the navigation tree or the detail panel gets.
layout panels	FlowLayoutPanel arranges children side by side and wraps, TableLayoutPanel aligns rows and columns, and FlexLayoutPanel distributes and wraps like a modern responsive layout.
responsive properties	A property that holds a different value per client profile, set at design time, so a desktop layout can change a handful of properties on narrow screens without a second screen.
command model	One consistent rule for where a command lives: MenuBar, ToolBar, RibbonBar, ContextMenu or StatusBar, each with its own job, all invoking application commands rather than containing business logic.
UserControl surface	The reuse boundary: a composed control that exposes a few meaningful properties, events and methods instead of every child control, so consumers never couple to its implementation.

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 3 is about. You can open [module-03.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 OperationsConsole 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 3 – "Containers, Layouts, Navigation, and Reuse" – of the LearnWisej Mastering the Control Library 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/mastering-the-control-library/specs/module-03.md>

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

1. Project "OperationsConsole", 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: Replace the Module 1 placeholders with the real Operations Console shell: a SplitContainer with navigation on the left, a TabControl detail surface on the right, a ToolBar for frequent commands and a StatusBar that shows real status. Move repeated UI into UserControls that expose only meaningful public properties and events, implement a narrow-profile layout with responsive properties or profile-aware logic, and document your docking and anchoring choices in LayoutNotes.md.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Navigation shell built from SplitContainer, TabControl, ToolBar and StatusBar
 - Repeated UI moved into UserControls with a small public surface
 - Narrow-profile layout using responsive properties or profile-aware logic
 - LayoutNotes.md documenting the docking and anchoring choices
 - Screenshots at desktop and narrow widths
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 SplitContainer were added to the page before the ToolBar and StatusBar, what would the user see and why does the order matter?
- Which of your UserControl's members could you rename without touching any consuming page, and which could you not? What does that tell you about its public surface?
- Which properties changed on the narrow profile, and why did you not build a separate narrow screen?

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 SplitContainer were added to the page before the ToolBar and StatusBar, what would the user see and why does the order matter?
- ✓ Which of your UserControl's members could you rename without touching any consuming page, and which could you not? What does that tell you about its public surface?
- ✓ Which properties changed on the narrow profile, and why did you not build a separate narrow screen?

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



[Open the Module 3 sample on GitHub](#)

Or clone the whole set with `git clone https://github.com/iceteagroup/LearnWisej-Samples` and open `Mastering the Control Library Course/Module 3`.



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.

- ✓ **Prove it actually works.** *"Go through this project deliverable by deliverable against the module specification at <https://learnwisej.com/assets/courses/mastering-the-control-library/specs/module-03.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 OperationsConsole 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 OperationsConsole 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.