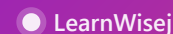


AI Coding Exercise

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.



The assistant is the fast pair of hands. You are still the architect.

An assistant can produce this module's sample in minutes. What it cannot do is decide whether the result is the right design for your application — that call stays with you. You supply the judgement: brief it properly, then check what comes back against what Module 7 just taught you.

What you are asking for

Complete the Adaptive Operations Console capstone: a custom theme with named tokens and at least four semantic appearances, scoped CsshClass usage, a desktop shell with Flow, Table and Flex regions, ClientProfiles.json with phone, tablet, small desktop and desktop behaviour, responsive properties plus an idempotent ResponsiveProfileChanged handler, and a profile indicator visible in test mode. Run the accessibility review for focus frames, invalid states, contrast, touch targets and tab order, fill in the responsive QA matrix with screenshots for every profile, and write the architecture note and grounding pack that explain each theme, layout and profile decision.

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

- ✓ Finished Adaptive Operations Console with custom theme, semantic appearances, scoped CSS and Flow, Table and Flex regions
- ✓ ClientProfiles.json, responsive properties and an idempotent ResponsiveProfileChanged handler with a test-mode profile indicator
- ✓ Accessibility review covering focus frames, invalid states, contrast, touch targets and tab order
- ✓ Responsive QA matrix with desktop, tablet and phone screenshots in both orientations
- ✓ Architecture note and AI grounding pack explaining every theme, layout and profile decision

Why it is shaped this way

Six modules of work now live in one solution: a theme derived from a base theme, scoped CSS, a nested container shell, Flow, Table and Flex regions, a ClientProfiles.json and a profile handler. What turns that into a product is not another feature. It is proof that the console is usable on every profile, that a keyboard or touch user can operate it, that a resize does not trigger expensive rebuilds, and that the next developer, or the next Wisej.NET upgrade, cannot silently break the visual system. This module is that proof, plus the grounding pack that lets an AI assistant answer questions about the decisions you made.

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
capstone screen set	The finished Adaptive Operations Console: custom theme, semantic appearances, scoped CSS, a desktop shell, an adaptive tablet shell, a phone shell, a grid, an editor, validation feedback, navigation and a profile indicator visible in test mode.
visual accessibility review	A pass over focus frames, disabled and invalid states, contrast, touch spacing, tooltips on icon-only buttons and keyboard tab order, done by operating the screen rather than reading the theme.
idempotent profile handler	A ResponsiveProfileChanged handler that assigns final values for the current profile, so running it twice, or after a refresh, produces exactly the same UI.
layout stability	MinimumSize and MaximumSize keep regions from collapsing, and AutoSize stays off large container hierarchies, so resizes are predictable and cheap.
theme governance	The theme JSON, the scoped stylesheet and ClientProfiles.json are source-controlled, reviewed as diffs, documented as tokens and never edited inside a built-in theme.
upgrade-safe customization	Documented appearances, Theme Builder workflows and server-side properties survive upgrades; CSS selectors that depend on a widget's internal DOM may not, so they are scoped, documented or avoided.
responsive QA matrix and grounding pack	The matrix records widths, orientations, expected profile and checks per scenario; the grounding pack records the decision tree, answer rules and source map an assistant answers 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 7 is about. You can open [module-07.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 7 – "Mobile-Ready Capstone, Accessibility, Performance, and Production Governance" – 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-07.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: Complete the Adaptive Operations Console capstone: a custom theme with named tokens and at least four semantic appearances, scoped `CssClass` usage, a desktop shell with Flow, Table and Flex regions, `ClientProfiles.json` with phone, tablet, small desktop and desktop behaviour, responsive properties plus an idempotent `ResponsiveProfileChanged` handler, and a profile indicator visible in test mode. Run the accessibility review for focus frames, invalid states, contrast, touch targets and tab order, fill in the responsive QA matrix with screenshots for every profile, and write the architecture note and grounding pack that explain each theme, layout and profile decision.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Finished Adaptive Operations Console with custom theme, semantic appearances, scoped CSS and Flow, Table and Flex regions
 - `ClientProfiles.json`, responsive properties and an idempotent `ResponsiveProfileChanged` handler with a test-mode profile indicator
 - Accessibility review covering focus frames, invalid states, contrast, touch targets and tab order
 - Responsive QA matrix with desktop, tablet and phone screenshots in both orientations
 - Architecture note and AI grounding pack explaining every theme, layout and profile 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:

- Your `ResponsiveProfileChanged` handler runs twice for the Tablet profile after a browser refresh. Which line of yours guarantees the second run changes nothing, and what would the user see if that line were missing?
- A reviewer asks you to remove the focus frame from action-button because it "looks dated". What do you change instead, which token drives it, and which two other states do you re-verify in the Theme Builder afterwards?
- The ticket grid header needs a tweak that only a selector on the widget's internal DOM can

reach. What do you try first, what must be true before the selector is allowed, and what do you write down so the next upgrade does not surprise the team?

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:

- ✓ Your ResponsiveProfileChanged handler runs twice for the Tablet profile after a browser refresh. Which line of yours guarantees the second run changes nothing, and what would the user see if that line were missing?
- ✓ A reviewer asks you to remove the focus frame from action-button because it "looks dated". What do you change instead, which token drives it, and which two other states do you re-verify in the Theme Builder afterwards?
- ✓ The ticket grid header needs a tweak that only a selector on the widget's internal DOM can reach. What do you try first, what must be true before the selector is allowed, and what do you write down so the next upgrade does not surprise the team?

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



[Open the Module 7 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 7`.



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/theming-and-responsive-layouts/specs/module-07.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."*
- ✓ **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."*



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.