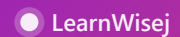


AI Coding Exercise

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.



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 AdaptiveOps 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

Add a scoped AdaptiveOps.css stylesheet to the Adaptive Operations Console with classes for metric cards and compact badges, apply them through CssClass, and use CssStyle only for one computed value such as a progress width. Assign the action-button AppearanceKey to the primary command, add a custom stale state to the ticket grid rows or a status panel, and implement a light/dark theme switch in the toolbar that changes the theme for the current session only, without modifying the shared theme.

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

- ✓ Scoped AdaptiveOps.css with metric-card and compact-badge classes applied through CssClass
- ✓ CssStyle used for exactly one computed one-off value, with the reason documented
- ✓ Primary command using the action-button AppearanceKey
- ✓ Custom stale state added through States and styled in the theme
- ✓ Session-only light/dark theme switch that leaves the shared theme untouched

Why it is shaped this way

The AdaptiveOps theme from Module 2 now owns the identity of every standard control in the Adaptive Operations Console, and the requests that arrive next are all small: a metric card needs a rounded, elevated surface, an SLA badge needs to be compact, one progress bar needs a width computed from live data, the Assign command should look like the primary action, and the night shift wants a dark variant. Each request has a tempting shortcut — a CssStyle string here, a BackColor there, the shared theme edited in place — and each shortcut quietly moves a visual decision to a layer that does not own it. Six months later nobody can say what the theme renders, what the CSS overrides and what the code patches. This module puts each of those five requests in the mechanism that owns it.

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
CssClass	One or more class names added to the widget's main element in the browser: the reusable hook a scoped application stylesheet can target without touching the theme.
CssStyle	A CSS string rendered directly as the style attribute of the widget's main element; right for a single computed value, wrong as a general styling tool because inline styles are hard to inspect, theme or hand to a designer.
scoped application stylesheet	A CSS file loaded alongside the theme whose selectors start from class names you own, such as <code>.metric-card</code> , and never reach into the internal DOM of a standard Wisej widget.
legacy StyleSheet extender	An older component that adds CSS classes to eligible controls and hosts rules in its <code>StyleSheetSource</code> property; a migration aid, unnecessary in new code because every control now has <code>CssClass</code> and <code>CssStyle</code> .
AppearanceKey	The name of an appearance in the active theme that a control renders with, so <code>C#</code> expresses intent (<code>action-button</code>) and the theme owns the colours, fonts and metrics of every state.
custom states	Names added to a control's <code>States</code> collection that the theme can style like hovered or focused, for transient UI conditions such as stale, warning or dirty that a designer should be able to see and restyle.
startup theme vs <code>Application.Theme</code>	The application-wide theme is chosen before the app runs in <code>Web.config</code> or <code>Default.json</code> ; <code>Application.Theme</code> lets one session work with a theme object at runtime without changing what other sessions see.

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 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 3 – "CSS, CssClass, CssStyle, States, and Runtime Theme Changes" – 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-03.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: Add a scoped `AdaptiveOps.css` stylesheet to the Adaptive Operations Console with classes for metric cards and compact badges, apply them through `CssClass`, and use `CssStyle` only for one computed value such as a progress width. Assign the action-button `AppearanceKey` to the primary command, add a custom stale state to the ticket grid rows or a status panel, and implement a light/dark theme switch in the toolbar that changes the theme for the current session only, without modifying the shared theme.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Scoped `AdaptiveOps.css` with metric-card and compact-badge classes applied through `CssClass`
 - `CssStyle` used for exactly one computed one-off value, with the reason documented
 - Primary command using the action-button `AppearanceKey`
 - Custom stale state added through `States` and styled in the theme
 - Session-only light/dark theme switch that leaves the shared theme untouched
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 designer asks for a smaller corner radius on every metric card. Which file changes, and how many places would you have had to edit if the radius had been set with `CssStyle`?
- Why does the Assign button need no hover or pressed code, and what exactly would you lose by replacing its `AppearanceKey` with `BackColor` and `ForeColor` values that match the palette?
- Two people are signed in and one of them switches to dark mode. Describe what each sees, and point at the line of your code that guarantees it.

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 designer asks for a smaller corner radius on every metric card. Which file changes, and how many places would you have had to edit if the radius had been set with `CssStyle`?
- ✓ Why does the Assign button need no hover or pressed code, and what exactly would you lose by replacing its `AppearanceKey` with `BackColor` and `ForeColor` values that match the palette?
- ✓ Two people are signed in and one of them switches to dark mode. Describe what each sees, and point at the line of your code that guarantees it.

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 `Theming and Responsive Layouts 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.

- ✓ **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."*
- ✓ **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."*



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.