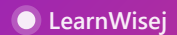


AI Coding Exercise

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 2 just taught you.

What you are asking for

Create the AdaptiveOps theme from a built-in base theme in the Theme Builder. Define a named color palette with brandPrimary, brandAccent, surface, surfaceAlt, textMain, textMuted, danger, warning, success and focusFrame plus default, heading and mono fonts, then restyle the button, panel, tab, grid header, invalid editor and tooltip appearances using those tokens. Add an action-button appearance that inherits from button and overrides only its default, hovered and pressed states, verify every change in the Theme Builder preview and load the theme in the Adaptive Operations Console.

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

- ✓ AdaptiveOps.theme created from a base theme and loaded by the application
- ✓ Named color tokens and default, heading and mono font tokens defined once and reused
- ✓ Button, panel, tab, grid header, invalid editor and tooltip appearances restyled from the tokens
- ✓ action-button appearance inheriting from button with overridden default, hovered and pressed states
- ✓ Theme Builder preview screenshots and a note on state order and inheritance decisions

Why it is shaped this way

Module 1 gave the Adaptive Operations Console a docked shell, but the base theme still owns every colour and font. The shortcut is to brand it control by control with BackColor, Font, CssStyle strings or CSS against the widgets' internal DOM. Each scatters one hex value across dozens of places, fights the theme engine that already renders every state, and breaks on the next upgrade. A theme is one JSON file that owns the identity of the standard controls; the Theme Builder is how you produce and maintain it, and this module builds the AdaptiveOps theme with 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
base theme	The built-in theme the Theme Builder copies to create yours, so hundreds of appearances, components, states, metrics, images and fonts exist before you change a value.
named color and font tokens	Entries in the colors and fonts sections (brandPrimary, danger, heading) that appearances reference by name, so a brand change is one edit.
appearance and AppearanceKey	An appearance is the theme block for one kind of widget; a control's AppearanceKey selects which block applies, which is how a semantic variant such as an action button exists.
appearance inheritance	An inherit entry lets a specialised appearance reuse a base one and override only a few states, so action-button stays a real button everywhere else.
state order	States such as default, hovered, pressed, focused, disabled and invalid are applied in list order; moving one changes which styles win, so order is behaviour.
child component	An inner part of a widget, such as a panel caption bar, a grid column header or an editor button, with its own appearance path and states.
styles versus properties	Styles are CSS-like values the theme turns into dynamic CSS classes and owns; properties are widget properties that may become inline style or behaviour and can be overridden by the application.

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 2 is about. You can open [module-02.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 2 – "Theme Builder and Theme JSON Internals" – 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-02.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 AdaptiveOps theme from a built-in base theme in the Theme Builder. Define a named color palette with `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` and `focusFrame` plus `default`, `heading` and `mono` fonts, then restyle the `button`, `panel`, `tab`, `grid header`, `invalid editor` and `tooltip` appearances using those tokens. Add an `action-button` appearance that inherits from `button` and overrides only its `default`, `hovered` and `pressed` states, verify every change in the Theme Builder preview and load the theme in the Adaptive Operations Console.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `AdaptiveOps.theme` created from a base theme and loaded by the application
 - Named color tokens and `default`, `heading` and `mono` font tokens defined once and reused
 - `Button`, `panel`, `tab`, `grid header`, `invalid editor` and `tooltip` appearances restyled from the tokens
 - `action-button` appearance inheriting from `button` with overridden `default`, `hovered` and `pressed` states
 - Theme Builder preview screenshots and a note on state order and inheritance decisions
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:

- Marketing changes `brandPrimary` to a new hex value tomorrow. Which files and which lines change in your solution, and what would the answer have been if you had set `BackColor` on each button instead?
- You moved the `pressed` state above `hovered` in the `action-button` appearance. What does a user see while holding the mouse button down, and why does the order rather than the colour decide it?
- A `TextBox` in the details editor stays grey when it is invalid even though you styled the invalid state. Which three things do you check, in which order, before editing the theme again?

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:

- ✓ Marketing changes brandPrimary to a new hex value tomorrow. Which files and which lines change in your solution, and what would the answer have been if you had set BackColor on each button instead?
- ✓ You moved the pressed state above hovered in the action-button appearance. What does a user see while holding the mouse button down, and why does the order rather than the colour decide it?
- ✓ A TextBox in the details editor stays grey when it is invalid even though you styled the invalid state. Which three things do you check, in which order, before editing the theme again?

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



[Open the Module 2 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 2`.



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-02.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.