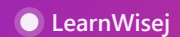


AI Coding Exercise

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.



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

Wrap a small JavaScript rating or indicator widget with the Widget control, or configure a Wisej.NET extension package: load its JavaScript and CSS packages, raise a server-side event from the client with the widget event pattern, call a client-side function from the server with CallAsync or Eval, theme it to match the console, then polish the complete Operations Console capstone and prepare a 5-minute demo.

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

- ✓ Widget or extension-based screen with its JavaScript and CSS packages loaded
- ✓ Client-to-server event raised with the widget event pattern or a native extension event
- ✓ Server-to-client call through CallAsync, Eval or the extension API
- ✓ Documented client/server contract: packages, properties, events and payload format
- ✓ Polished Operations Console capstone with a 5-minute demo script

Why it is shaped this way

Sooner or later a product needs something the native library does not ship: a specialised browser capability, a third-party JavaScript library, a custom visualisation, or a reusable integration. The temptation is to drop into JavaScript and lose the Wisej.NET model: server state, theming, the designer and an event flow another developer can follow. This module integrates a client widget without giving any of that up, then finishes the Operations Console capstone.

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
native control	A control that implements both the server side and its browser widget, so it already has design-time support, theming, events and server state; always the first stop.
extension types	Control, Widget, Component, Extender Provider and Icon Pack: naming the type stops "reusable" from being confused with "new native control".
Widget control	A general-purpose client container that loads JavaScript and CSS packages, runs an initialisation script and exchanges events and data with server code.
Packages and InitScript	The package list declares the client files the widget depends on; the InitScript creates the JavaScript object, wires its client events and exposes only the functions the server needs.
WidgetEvent	The server event raised when client code calls fireWidgetEvent; its type and data payload arrive from the browser and must be validated like any other user input.
CallAsync / Eval	The server-to-client half of the bridge: call a function exposed by the widget, or evaluate a script, without leaving C#.
widget contract	The documented properties, events, payload format, package dependencies and theming rules that let another developer, or a grounded AI assistant, maintain the widget.

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 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 7 – "Custom Widgets, Extensions, Theming, and Capstone" – 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-07.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: Wrap a small JavaScript rating or indicator widget with the Widget control, or configure a Wisej.NET extension package: load its JavaScript and CSS packages, raise a server-side event from the client with the widget event pattern, call a client-side function from the server with `CallAsync` or `Eval`, theme it to match the console, then polish the complete Operations Console capstone and prepare a 5-minute demo.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Widget or extension-based screen with its JavaScript and CSS packages loaded
 - Client-to-server event raised with the widget event pattern or a native extension event
 - Server-to-client call through `CallAsync`, `Eval` or the extension API
 - Documented client/server contract: packages, properties, events and payload format
 - Polished Operations Console capstone with a 5-minute demo script
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:

- What would happen if a user edited the `ratingChanged` payload in the browser's developer tools before it reached your handler, and which line of your code stops it?
- Your widget renders blank after the Widgets tab is hidden and shown again: which three things do you check, in which order, and why is the startup HTML not the fix?
- Which rule in your widget could you move to JavaScript as an optimisation, and which one must never leave the server?

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:

- ✓ What would happen if a user edited the ratingChanged payload in the browser's developer tools before it reached your handler, and which line of your code stops it?
- ✓ Your widget renders blank after the Widgets tab is hidden and shown again: which three things do you check, in which order, and why is the startup HTML not the fix?
- ✓ Which rule in your widget could you move to JavaScript as an optimisation, and which one must never leave the server?

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 `Mastering the Control Library 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.

- ✓ **Add the second case.** *"Extend this OperationsConsole 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."*
- ✓ **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-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 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."*



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.