

AI Coding Exercise

Build this module's EnterpriseOps 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.



Writing the code is the easy half now.

Getting working code for "JavaScript Object Model, Browser APIs, and Secure Interop Contracts" out of a modern assistant takes one good prompt. Judging it, correcting it and extending it takes everything this module covered. That is the split this exercise is built around — you supply the specification and the review, the assistant supplies the typing.

What you are asking for

Add an advanced keyboard command palette and browser capability panel. Use JavaScript to collect browser features and trigger server-side commands, then enforce permissions server-side before running the command.

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

- ✓ Interop contract document
- ✓ Command palette script
- ✓ Server callback methods
- ✓ Browser capability panel
- ✓ Security review notes

Why it is shaped this way

Wisej.NET lets developers add JavaScript where it is valuable, but advanced interop can become fragile unless every boundary is treated as a small API.

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
App object model	The client-side representation of Wisej.NET components that JavaScript can reach. Use it deliberately, through documented entry points, not by poking at internals.
client event	A browser-side event that is forwarded to the server. Only meaningful events should cross the wire, with compact payloads.
remote method	A server method the browser can call directly, such as a WebMethod, which returns a result to JavaScript. It needs the same validation as any endpoint.
browser feature detection	Checking what the browser supports before using an API. Use it for browser-side value only, while business rules stay on the server.
interop contract	A named, documented description of what JavaScript sends to the server and what it receives back, so both sides can be tested and versioned.
lifecycle timing	Knowing when a client widget is created, rendered, updated, and destroyed, so scripts run at the right moment and clean up after themselves.
input validation	Treating every value that arrives from the browser as untrusted and validating it on the server, because client code can be modified by anyone.

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 9 is about. You can open [module-09.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 EnterpriseOps 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 9 – "JavaScript Object Model, Browser APIs, and Secure Interop Contracts" – of the LearnWisej Enterprise Wisej.NET: Architecture to Cloud 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/azure-docker/specs/module-09.md>

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

1. Project "EnterpriseOps", 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 an advanced keyboard command palette and browser capability panel. Use JavaScript to collect browser features and trigger server-side commands, then enforce permissions server-side before running the command.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Interop contract document
 - Command palette script
 - Server callback methods
 - Browser capability panel
 - Security review notes
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 happens before the target widget exists?
- Which values from JavaScript are trusted?
- Which business rules remain server-side?

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 happens before the target widget exists?
- ✓ Which values from JavaScript are trusted?
- ✓ Which business rules remain server-side?

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



[Open the Module 9 sample on GitHub](#)

Or clone the whole set with `git clone https://github.com/iceteagroup/LearnWisej-Samples` and open [Enterprise Wisej.NET Course/Module 9](#).



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 EnterpriseOps 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/azure-docker/specs/module-09.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 EnterpriseOps 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.