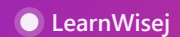


AI Coding Exercise

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.



You are not going to type this sample. You are going to direct it.

Almost nobody writes Wisej.NET code from a blank file any more — an assistant writes the first draft and the developer decides whether it is right. That only works if you know the material, which is what the rest of this module gave you. Here you hand that knowledge to ChatGPT or Claude, get this module's sample back as a working Visual Studio solution, and review it the way you would review a colleague's pull request.

What you are asking for

Build the Operations Console dashboard tab: a ChartJS line or bar chart whose labels and DataSets come from a dashboard view model, a ProgressBar or gauge-like indicator for one status value, a PdfViewer or HtmlPanel content preview, an Upload workflow, a map or content placeholder, and one refresh method that updates every dashboard control from a single model.

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

- ✓ Dashboard tab with a ChartJS line or bar chart
- ✓ Chart labels and DataSets built from a dashboard view model
- ✓ ProgressBar or gauge-like indicator for one status value
- ✓ PdfViewer or HtmlPanel content preview plus an Upload workflow
- ✓ One refresh method that updates every dashboard control

Why it is shaped this way

Dashboards are not just charts. A dashboard is a control composition that turns data into decisions, and the Operations Console dashboard tab will be the screen a service manager opens first every morning. Wisej.NET gives you native content and media controls plus extensions for charts, maps, document viewing and gauges; the intermediate skill is knowing what belongs in the Wisej.NET control model, what is better delegated to a specialised JavaScript library wrapped as an extension or Widget, and how to keep the whole screen refreshing as one unit instead of a pile of independently reloading visuals.

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
dashboard	A composition of controls whose only job is to answer a decision question; if a visual does not help someone decide something, it is clutter.
ChartJS extension	A NuGet package that exposes Chart.js as a server-side component: the server owns the chart state, the browser renders the visual, and the chart joins the same event model as every other control.
labels and DataSets	The two halves of chart data; labels name the points along the axis and each DataSet carries one series, so they are always set together.
dashboard view model	A display-shaped object such as DashboardModel that holds months, counts and a completion value, separate from any database entity and from the code that draws it.
browser boundary	The client is the browser and the server is where your Wisej.NET code runs; files, PDFs, HTML and map tiles all cross that line in one direction or the other.
Upload versus local path	Upload moves a file from the browser to the server; it is not the same as reading a desktop path, because the server can never browse the user's disk.
model-based refresh	One service call returns a complete dashboard model and one method updates every control from it, in a batch, with a visible timestamp.

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 6 is about. You can open [module-06.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 6 – "Charts, Dashboards, Content, Media, and Documents" – 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-06.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: Build the Operations Console dashboard tab: a ChartJS line or bar chart whose labels and DataSets come from a dashboard view model, a ProgressBar or gauge-like indicator for one status value, a PdfViewer or HtmlPanel content preview, an Upload workflow, a map or content placeholder, and one refresh method that updates every dashboard control from a single model.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Dashboard tab with a ChartJS line or bar chart
 - Chart labels and DataSets built from a dashboard view model
 - ProgressBar or gauge-like indicator for one status value
 - PdfViewer or HtmlPanel content preview plus an Upload workflow
 - One refresh method that updates every dashboard control
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:

- Which control on your dashboard tab is doing the most important work, and what question would go unanswered if you removed it?
- What would break first if the ticket data set became ten times larger, and why does returning aggregates from the service protect the chart?
- Where does the uploaded file exist at each step of the Upload workflow, and why can the server never open a path the user typed in?

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:

- ✓ Which control on your dashboard tab is doing the most important work, and what question would go unanswered if you removed it?
- ✓ What would break first if the ticket data set became ten times larger, and why does returning aggregates from the service protect the chart?
- ✓ Where does the uploaded file exist at each step of the Upload workflow, and why can the server never open a path the user typed in?

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



[Open the Module 6 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 6`.



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.

- ✓ **Make it survive a bad day.** *"Review this OperationsConsole 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."*
- ✓ **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-06.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."*



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.