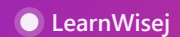


AI Coding Exercise

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.



Writing the code is the easy half now.

Getting working code for "Lists, Trees, Repeaters, and Hierarchical Data" 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

Build the Operations Console document explorer: a `TreeView` of categories or folders with a stable ID attached to every node, a `ListView` of documents tied to the selected node, lazy loading or virtual mode on at least one list or tree path, icons or image keys for the different item types, and a detail `UserControl` that loads through a service or view-model layer whenever the selection changes.

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

- ✓ `TreeView` of categories with a stable ID stored on every node
- ✓ `ListView` of documents tied to the selected tree node
- ✓ Lazy loading or virtual mode on at least one list or tree path
- ✓ Detail `UserControl` loaded through a service or view-model layer on selection
- ✓ Icons or image keys for the different item types

Why it is shaped this way

List and tree controls are often treated as simple selection widgets, but in a line-of-business application they are the user's map: folders, customers, orders, procedures, roles, tasks, documents, product catalogs and the modules of the application itself. The Operations Console document explorer you build in this module has to stay fast when the category tree is deep and the document list is long, and every click has to map to a real business record rather than to a piece of display text.

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
data shape	The first design question: is the data flat, grouped, hierarchical, repeated as cards, or a set of properties? The answer selects the control.
ListView views	One control that shows the same collection as LargeIcon, SmallIcon, Details or Tile, the natural choice for file-like and catalog-like screens.
virtual mode	A ListView mode in which the application sets VirtualListSize and answers RetrieveVirtualItem, so items exist only for the rows the control asks for.
lazy loading	Loading only the top levels of a tree at startup and fetching children when a node expands, so startup stays cheap and the UI responsive.
virtual scrolling and prefetch	Rendering only the visible rows of a large list or tree in the browser, with PrefetchItems pre-rendering a few extra rows for smooth scrolling.
stable identity	A unique business ID stored in the node's or item's Tag or Name, or in a view-model map, never derived from displayed text.
selection as an event	A selection change resolves an ID and asks a service for data; no query ever runs inside the code that builds nodes or items.

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 4 is about. You can open [module-04.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 4 – "Lists, Trees, Repeaters, and Hierarchical Data" – 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-04.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 document explorer: a `TreeView` of categories or folders with a stable ID attached to every node, a `ListView` of documents tied to the selected node, lazy loading or virtual mode on at least one list or tree path, icons or image keys for the different item types, and a detail `UserControl` that loads through a service or view-model layer whenever the selection changes.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `TreeView` of categories with a stable ID stored on every node
 - `ListView` of documents tied to the selected tree node
 - Lazy loading or virtual mode on at least one list or tree path
 - Detail `UserControl` loaded through a service or view-model layer on selection
 - Icons or image keys for the different item types
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:

- Two categories under different customers are both called "Contracts". What exactly does your selection handler use to tell them apart, and where does that value live?
- If the document count under one category grew from 300 to 300,000, which part of your explorer would break first, and which property or event would you change to fix it?
- Which lines of your selection handler could be moved into a unit test without a browser, and which lines belong to the detail `UserControl` instead?

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:

- ✓ Two categories under different customers are both called "Contracts". What exactly does your selection handler use to tell them apart, and where does that value live?
- ✓ If the document count under one category grew from 300 to 300,000, which part of your explorer would break first, and which property or event would you change to fix it?
- ✓ Which lines of your selection handler could be moved into a unit test without a browser, and which lines belong to the detail UserControl instead?

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



[Open the Module 4 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 4`.



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 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."*
- ✓ **Swap a piece without moving the seams.** *"Pick the part of this OperationsConsole 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 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."*



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.