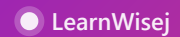


AI Coding Exercise

Build this module's GlobalDesk 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.



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

Localize the GlobalDesk customer editor through the Visual Studio designer. Finish the neutral English layout of the CustomerEditor user control first, then set Localizable to true and use the designer Language selector to add a German variant, changing only what must differ: the captions, the width of the two longest buttons and one label's Location. Run the application under de-DE and compare the real browser output with the designer view, then add a Recreate editor button that disposes and rebuilds the control so you can see for yourself why designer resources do not follow a runtime culture change. Finish with a lab note that lists the properties you overrode and the ones you deliberately left to the neutral layout.

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

- ✓ CustomerEditor user control with Localizable set to true over a complete neutral English layout
- ✓ German designer language variant overriding the captions plus at least one Size and one Location
- ✓ Anchoring, docking or AutoSize applied so the German captions cannot clip
- ✓ Recreate editor button that disposes and rebuilds the container after a culture change
- ✓ Lab note listing the overridden properties and the ones left to the neutral layout

Why it is shaped this way

Module 1 gave GlobalDesk a dashboard whose captions all come from Strings.resx, and that is the right home for text the code generates. The customer editor is different: it is a designed screen, arranged by eye in the Visual Studio designer. Routing each of its captions through a helper call means hand-editing the designer file, and it still does not solve the real problem, which is that a German Kundennummer speichern does not fit inside a button sized for Save. This module uses the localization workflow Visual Studio already offers, so a designed screen's captions and its geometry can differ per language with no hard-coded strings and no second layout file.

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
Localizable	A design-time property on a top-level container; true switches the designer into the localization workflow so values can be stored per culture.
Designer Language selector	Appears once Localizable is true; the selected culture is the version you are editing, so each change is recorded against that culture only.
The neutral language	The default version every culture falls back to, and the only one where you add, remove or restructure controls; culture views hold overrides, not a second screen.
Localizable properties	Not only Text: Size, Location, colors and images can each carry a different value per culture, because translation changes geometry as well as words.
Generated resource files	The designer writes localized values into resource files instead of the C# designer code, keeping language-specific values out of the code you read and merge daily.
Applied at creation time	Designer-localized values are applied while the controls are created, so changing the session culture later does not rewrite a container that already exists.
Translation-tolerant layout	Docking, anchoring, FlowLayoutPanel, TableLayoutPanel, FlexLayoutPanel, AutoSize and padding let text grow on its own, so you need fewer geometry overrides.

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 GlobalDesk 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 – "Localizing Wisej.NET UI in the Visual Studio Designer" – of the LearnWisej Localization in Wisej.NET 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/localization-in-wisej-net/specs/module-02.md>

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

1. Project "GlobalDesk", 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: Localize the GlobalDesk customer editor through the Visual Studio designer. Finish the neutral English layout of the CustomerEditor user control first, then set Localizable to true and use the designer Language selector to add a German variant, changing only what must differ: the captions, the width of the two longest buttons and one label's Location. Run the application under de-DE and compare the real browser output with the designer view, then add a Recreate editor button that disposes and rebuilds the control so you can see for yourself why designer resources do not follow a runtime culture change. Finish with a lab note that lists the properties you overrode and the ones you deliberately left to the neutral layout.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - CustomerEditor user control with Localizable set to true over a complete neutral English layout
 - German designer language variant overriding the captions plus at least one Size and one Location
 - Anchoring, docking or AutoSize applied so the German captions cannot clip
 - Recreate editor button that disposes and rebuilds the container after a culture change
 - Lab note listing the overridden properties and the ones left to the neutral layout
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:

- A teammate adds a new phone field to CustomerEditor while German is selected in the Language list. What is the English user likely to see, and what is the rule that would have prevented it?
- The save confirmation the editor shows after a successful update still comes from Strings.resx rather than from the designer resources. Give two reasons why that is the right place for it, and name one caption on this screen that genuinely belongs in the designer instead.
- Your Recreate editor button works, but a colleague argues that the screen should simply

reapply the localized properties instead of rebuilding. What would that code have to know that the rebuild does not, and when would their approach be the better trade?

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:

- ✓ A teammate adds a new phone field to CustomerEditor while German is selected in the Language list. What is the English user likely to see, and what is the rule that would have prevented it?
- ✓ The save confirmation the editor shows after a successful update still comes from Strings.resx rather than from the designer resources. Give two reasons why that is the right place for it, and name one caption on this screen that genuinely belongs in the designer instead.
- ✓ Your Recreate editor button works, but a colleague argues that the screen should simply reapply the localized properties instead of rebuilding. What would that code have to know that the rebuild does not, and when would their approach be the better trade?

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.



No reference sample for this course yet — yours is the first

The [LearnWisej samples repository](#) carries worked builds for most courses, but not yet for this one. That means there is nothing to compare against and no shortcut: the specification and your own review are the whole safety net. Keep what you build — and if it turns out well, we would like to see it.

Three prompts to take it further

Point your assistant at the project you just built 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/localization-in-wisej-net/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 GlobalDesk 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 GlobalDesk 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.