

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.

 LearnWisej



Brief it, read it, then break it.

This exercise has three parts, and only the first is about prompting. You brief an assistant with the module specification and get a GlobalDesk solution back; you read what it wrote and decide whether it really does what the module asked; then you push it further than the module asked. The second and third parts are where the learning is.

What you are asking for

Grow the GlobalDesk shared resources. Add Strings.de.resx beside Resources/Strings.resx carrying exactly the same keys with translated values, add a Customer.LastOrder entry that holds two placeholders, and build the sentence with string.Format and Application.CurrentCulture instead of concatenating fragments. Route every message the customer editor shows through the Texts helper, including the save confirmation and the two validation messages, and have the save service return a stable result key such as CustomerEditor.Saved rather than an English sentence the UI prints directly. Then add a Resources-de.resx file that overrides one Wisej system label, prove it in a MessageBox, and write a short note on the difference between your product text and Wisej system text.

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

- ✓ Strings.de.resx carrying the same keys as the neutral file with translated values only
- ✓ Customer.LastOrder resource built with string.Format and the active culture, never concatenated
- ✓ Save service returning a resource key or result code instead of an English sentence
- ✓ Resources-de.resx overriding at least one Wisej system label, shown in a MessageBox
- ✓ Lab note separating product text, Wisej system text and diagnostic text

Why it is shaped this way

Module 2 left the GlobalDesk customer editor looking convincing in German: every caption the designer owns changes with the culture and the layout survives. Then the user presses Save and reads Customer saved. in English, because that sentence is not a property of any control — it is a string literal in a click handler, beside This field is required. and the status line that counts loaded contacts.

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
shared application resource file	Resources/Strings.resx holds the user-visible text that no single designed screen owns: save confirmations, validation messages, status lines, report headings and service results.
culture-specific sibling	Strings.de.resx and Strings.it.resx sit next to the neutral file and carry the same keys; a translator changes values, never key names.
stable semantic key	A name such as CustomerEditor.Saved describes the meaning of the text, so rewording the English sentence never forces a code change or breaks the translations.
Wisej.Resources.ResourceManager	The lookup object that follows the .NET resource model: given a base name and an assembly it finds the value for the active culture and falls back toward the neutral file.
missing-key marker	A helper that returns [Key] when a lookup fails turns an invisible defect into something you can see and search for, instead of a blank label.
format string with placeholders	The whole sentence with its {0} and {1} lives in the resource and string.Format fills it, because languages order the name, the date and the verb differently.
Resources-[LANG].resx	A separate Wisej.NET system resource file in the project root or a /Resources folder that overrides or adds framework labels such as \$Ok, \$Cancel and \$Offline.

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 3 is about. You can open [module-03.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 3 – "Application Resources, ResourceManager and Wisej System Text" – 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-03.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: Grow the GlobalDesk shared resources. Add `Strings.de.resx` beside `Resources/Strings.resx` carrying exactly the same keys with translated values, add a `Customer.LastOrder` entry that holds two placeholders, and build the sentence with `string.Format` and `Application.CurrentCulture` instead of concatenating fragments. Route every message the customer editor shows through the Texts helper, including the save confirmation and the two validation messages, and have the save service return a stable result key such as `CustomerEditor.Saved` rather than an English sentence the UI prints directly. Then add a `Resources-de.resx` file that overrides one Wisej system label, prove it in a `MessageBox`, and write a short note on the difference between your product text and Wisej system text.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `Strings.de.resx` carrying the same keys as the neutral file with translated values only
 - `Customer.LastOrder` resource built with `string.Format` and the active culture, never concatenated
 - Save service returning a resource key or result code instead of an English sentence
 - `Resources-de.resx` overriding at least one Wisej system label, shown in a `MessageBox`
 - Lab note separating product text, Wisej system text and diagnostic text
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:

- Delete one key from `Strings.de.resx` and run under German. Describe what the user sees, why nothing crashes, and how you would notice this in a real application with two hundred keys.
- Your save service currently returns `CustomerEditor.Saved`. What would break in the unit tests, and in the German screen, if it returned the sentence itself instead?
- A colleague adds the word "Cancel" to `Resources-de.resx` under a key of their own invention. Explain why the `MessageBox` button does not change, and where that string actually belonged.

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:

- ✓ Delete one key from Strings.de.resx and run under German. Describe what the user sees, why nothing crashes, and how you would notice this in a real application with two hundred keys.
- ✓ Your save service currently returns CustomerEditor.Saved. What would break in the unit tests, and in the German screen, if it returned the sentence itself instead?
- ✓ A colleague adds the word "Cancel" to Resources-de.resx under a key of their own invention. Explain why the MessageBox button does not change, and where that string actually belonged.

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.

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