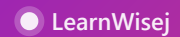


AI Coding Exercise

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.



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

Create the GlobalDesk dashboard page with a welcome heading, a Customers navigation button and a small culture preview panel. Add Resources/Strings.resx as the neutral resource with at least twelve stable semantic keys such as Navigation.Customers and CustomerEditor.Save, write a static Texts helper over Wisej.Resources.ResourceManager that returns a bracketed key marker when a key is missing, and replace every hard-coded caption on the page with a Texts.Get call. Then fill the preview panel from real values: format today's date, a quantity and a currency amount with Application.CurrentCulture instead of building the strings by hand, and finish with a short lab note that separates the stored value from its culture-sensitive presentation.

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

- ✓ GlobalDesk dashboard page whose captions all come from resource keys, with no hard-coded user-visible text
- ✓ Resources/Strings.resx neutral file holding at least twelve stable semantic keys
- ✓ Texts helper over Wisej.Resources.ResourceManager that marks a missing key instead of returning blank
- ✓ Culture preview panel formatting a date, a number and a currency with Application.CurrentCulture
- ✓ Lab note explaining which values are data and which are culture-sensitive presentation

Why it is shaped this way

GlobalDesk is a small service desk written for one office. Every heading, button caption and confirmation message is an English literal typed straight into the C#, and the dashboard prints amounts and dates the way a US machine happens to. When a German branch opens, none of that text has anywhere to go —

and a developer who starts by hand-editing captions meets the second half of the problem: a German caption over an amount that still reads 1,850.75 is not a German screen.

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
localization	Adapting an application to a user's language and culture: translated text, correctly formatted dates, numbers and currency, room for longer words, and a layout that can mirror for right-to-left languages.
culture name	An identifier such as en, de-DE or fr-CA that names a language and, optionally, a region; the language part selects which resources are preferred, the whole name drives formatting conventions.
neutral resource	The complete fallback set of resources, held in a file such as Resources/Strings.resx with no culture in its name; every key the application asks for must exist here.
Wisej.Resources.ResourceManager	The object that takes a resource base name and an assembly and returns the value for a key in the best-matching culture, so your code never opens a file itself.
culture fallback	.NET's rule for finding a value: try the full culture, then the language-only culture, then the neutral resource, and use the first one that has the key.
semantic resource key	A stable name that describes meaning rather than wording, such as CustomerEditor.Save or Validation.Required, so the English text can be reworded without touching any other language.
Application.CurrentCulture	The culture of the current Wisej.NET session; two users signed in at the same time can hold two different values, which is why culture must never live in a static field.

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 1 is about. You can open [module-01.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



Use Claude

Opens [claude.ai](#) with the same prompt loaded. Claude Code users can paste it into a terminal in an empty folder instead.

Codex; any plan that can browse the web will fetch the spec.



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 1 – "Localization Fundamentals: Culture, Language and Resources" – 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-01.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: Create the GlobalDesk dashboard page with a welcome heading, a Customers navigation button and a small culture preview panel. Add `Resources/Strings.resx` as the neutral resource with at least twelve stable semantic keys such as `Navigation.Customers` and `CustomerEditor.Save`, write a static `Texts` helper over `Wisej.Resources.ResourceManager` that returns a bracketed key marker when a key is missing, and replace every hard-coded caption on the page with a `Texts.Get` call. Then fill the preview panel from real values: format today's date, a quantity and a currency amount with `Application.CurrentCulture` instead of building the strings by hand, and finish with a short lab note that separates the stored value from its culture-sensitive presentation.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - GlobalDesk dashboard page whose captions all come from resource keys, with no hard-coded user-visible text
 - `Resources/Strings.resx` neutral file holding at least twelve stable semantic keys
 - `Texts` helper over `Wisej.Resources.ResourceManager` that marks a missing key instead of returning blank
 - Culture preview panel formatting a date, a number and a currency with `Application.CurrentCulture`
 - Lab note explaining which values are data and which are culture-sensitive presentation
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:

- You change the English wording of the Save button from "Save" to "Save customer". Which files have to change if your key is `CustomerEditor.Save`, and which files would have had to change if the key had been the English sentence itself?
- Your `Texts` helper returns `[Preview.Amount]` for one label. Name two different mistakes that produce exactly that marker, and say how you would tell them apart without a debugger.
- A colleague suggests storing the order total in the database as the string "1.850,75 €" so German users always see it correctly. Explain what breaks, and which part of that string is

data and which part is presentation.

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:

- ✓ You change the English wording of the Save button from "Save" to "Save customer". Which files have to change if your key is CustomerEditor.Save, and which files would have had to change if the key had been the English sentence itself?
- ✓ Your Texts helper returns [Preview.Amount] for one label. Name two different mistakes that produce exactly that marker, and say how you would tell them apart without a debugger.
- ✓ A colleague suggests storing the order total in the database as the string "1.850,75 €" so German users always see it correctly. Explain what breaks, and which part of that string is data and which part is presentation.

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.

- ✓ **Add the second case.** *"Extend this GlobalDesk 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/localization-in-wisej-net/specs/module-01.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."*



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.