

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



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 7 just taught you.

What you are asking for

Finish the GlobalDesk capstone. Wrap every resource lookup in a `LocalizationService` with `Text`, `Date`, `Number` and `Currency` methods, give it a missing-key policy that shows a bracketed key in development and logs the key in production, and make that service the only place the UI touches `ResourceManager`. Keep the domain out of it: the ticket model exposes a `TicketStatus` value and the UI maps it to a localized caption. Then run the localization test matrix over the finished application, covering the neutral culture, German, a language-region culture that falls back, a deliberately missing key, formatted dates and currency, a runtime switch, a long-string pass and the RTL culture, using a pseudo-localized resource set to expose anything still hard-coded. Confirm the culture folders survive a publish, and hand in the QA checklist as the capstone deliverable.

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

- ✓ `LocalizationService` with `Text`, `Date`, `Number` and `Currency` methods and a documented missing-key policy
- ✓ Domain code returning a `TicketStatus` value that the UI maps to a localized caption
- ✓ Pseudo-localized resource set used to find every remaining hard-coded string
- ✓ Completed localization test matrix covering fallback, formatting, runtime switching and RTL
- ✓ Published output checked for its culture folders, plus the GlobalDesk localization QA checklist

Why it is shaped this way

After six modules GlobalDesk speaks two languages, but the machinery is scattered: designer resources on the customer editor, a helper over the shared.resx files, a language picker, a `CultureChanged` handler and an RTL demonstration. Nothing yet states what happens when a key has no translation, and nobody has proved the German build still fits a narrow window. A production localization system is predictable: it knows where each string comes from, which culture is active in this session, what a missing translation looks like, and what a language change does to open screens. This module turns those four answers into one small service, one written test matrix and the finished capstone.

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
LocalizationService	A small class that centralizes your application's choices around.NET localization instead of replacing it: one resource base name, one missing-key policy, one set of formatting methods.
Missing-key policy	The chosen, tested behavior when GetString returns nothing: a visible [Key] marker while you develop, a logged key plus neutral fallback text in production.
Product, system and diagnostic text	Three families with three owners: your own.resx strings, Wisej.NET's own labels through Resources-[LANG].resx, and log or exception text that stays in one developer language.
Culture-free domain	The model exposes a value such as TicketStatus.Closed and the UI maps it to a localized caption, so business logic and its tests never depend on a language.
Localization test matrix	A written list of runs — neutral, translated, region-falling-back, missing key, formatted values, runtime switch, long strings, RTL, system labels — so coverage is a checklist rather than a memory.
Pseudo-localization	A test resource set whose values are visibly altered and lengthened, such as [!!! Save customer record!!!], built to expose clipping and hard-coded strings rather than to translate anything.
Satellite resources	.NET compiles culture-specific resources into satellite assemblies, so the culture folders produced by the build have to travel with the application when you publish or package it.

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 7 is about. You can open [module-07.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 7 – "Production Localization Architecture, Testing and Capstone" – 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-07.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: Finish the GlobalDesk capstone. Wrap every resource lookup in a `LocalizationService` with `Text`, `Date`, `Number` and `Currency` methods, give it a missing-key policy that shows a bracketed key in development and logs the key in production, and make that service the only place the UI touches `ResourceManager`. Keep the domain out of it: the ticket model exposes a `TicketStatus` value and the UI maps it to a localized caption. Then run the localization test matrix over the finished application, covering the neutral culture, German, a language-region culture that falls back, a deliberately missing key, formatted dates and currency, a runtime switch, a long-string pass and the RTL culture, using a pseudo-localized resource set to expose anything still hard-coded. Confirm the culture folders survive a publish, and hand in the QA checklist as the capstone deliverable.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `LocalizationService` with `Text`, `Date`, `Number` and `Currency` methods and a documented missing-key policy
 - Domain code returning a `TicketStatus` value that the UI maps to a localized caption
 - Pseudo-localized resource set used to find every remaining hard-coded string
 - Completed localization test matrix covering fallback, formatting, runtime switching and RTL
 - Published output checked for its culture folders, plus the GlobalDesk localization QA checklist
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 reviewer opens GlobalDesk in French, a culture you never translated, and sees English captions with correctly formatted French dates. Explain which mechanism produced each half of that screen, and say whether anything is broken.
- Your pseudo-localized run shows one label still reading "Save" in plain English. List the three places that text could be coming from, and describe how you would tell them apart

without guessing.

– A colleague proposes caching the resolved `CultureInfo` in a static field on `LocalizationService` "because looking it up on every call is wasteful". Describe what a second signed-in user would experience, and what you would suggest 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:

- ✓ A reviewer opens `GlobalDesk` in French, a culture you never translated, and sees English captions with correctly formatted French dates. Explain which mechanism produced each half of that screen, and say whether anything is broken.
- ✓ Your pseudo-localized run shows one label still reading "Save" in plain English. List the three places that text could be coming from, and describe how you would tell them apart without guessing.
- ✓ A colleague proposes caching the resolved `CultureInfo` in a static field on `LocalizationService` "because looking it up on every call is wasteful". Describe what a second signed-in user would experience, and what you would suggest 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.



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-07.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.