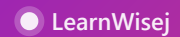


AI Coding Exercise

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.



Writing the code is the easy half now.

Getting working code for "Browser Culture, Session Culture and Runtime Language Switching" 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

Give GlobalDesk a language switch that finishes the job. Set culture to auto in Default.json so the session starts from the browser language, add a language ComboBox offering English, German and one language-region culture, and set Application.CurrentCulture from CultureInfo.GetCultureInfo in its SelectedIndexChanged handler without ever storing the choice in a static field. Handle Application.CultureChanged to call an ApplyTextResources method that refreshes every resource-backed caption and an UpdateCulturePreview method that reformats the date, number and currency samples, and recreate the designer-localized customer editor in the same handler. Verify the same page through the lang URL parameter, then write a lab note describing what a static culture field would do to a second user signed in at the same time.

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

- ✓ Default.json with culture set to auto, plus a note on when a fixed culture is the right choice
- ✓ Language ComboBox handler setting Application.CurrentCulture from CultureInfo.GetCultureInfo
- ✓ Application.CultureChanged handler calling ApplyTextResources and UpdateCulturePreview
- ✓ Designer-localized editor recreated after the switch so no caption stays in the old language
- ✓ Verification through the lang URL parameter and a note on why culture must not live in a static field

Why it is shaped this way

After Module 3, GlobalDesk has a neutral resource file, a German one beside it, a Texts helper, a designer-localized customer editor and even one Wisej system label of its own. What it still has no answer for is the question a real user asks first: how do I get this in my language, and what happens to the screen when I ask? Everything so far was decided while the application was being built; this module is about the two moments when the decision is actually made — when a session starts, and when a user changes their mind halfway through it.

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
automatic browser detection	Wisej.NET can read the language the browser reports and start the session with it, so each user gets a sensible culture without one server-wide setting deciding for everybody.
culture in Default.json	The application setting that decides where a new session's culture comes from; it defaults to auto, and a fixed name such as de-DE pins every session to one locale instead.
rightToLeft in Default.json	Its neighbor setting, whose auto value follows whatever culture the session ends up with; Module 5 builds on it, but you meet it in the same file here.
Application.CurrentCulture	Gets or sets the culture of the current Wisej.NET session, which is exactly the scope a multi-user web application needs: one user's choice must never repaint another user's screen.
lang URL parameter	A supported way to ask for a language on the way in, for example?lang=de, which makes test links and language-specific entry URLs trivial to produce.
Application.CultureChanged	Raised when the current culture changes from the one the session started with; the single place to reapply resource-backed text and reformat anything already on screen.
designer-localized controls	Their localized values are applied when the controls are created, so a later culture change does not rewrite them — you recreate the container or reapply the values in code.

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 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 4 – "Browser Culture, Session Culture and Runtime Language Switching" – 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-04.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: Give GlobalDesk a language switch that finishes the job. Set culture to auto in `Default.json` so the session starts from the browser language, add a language `ComboBox` offering English, German and one language-region culture, and set `Application.CurrentCulture` from `CultureInfo.GetCulturesInfo` in its `SelectedIndexChanged` handler without ever storing the choice in a static field. Handle `Application.CultureChanged` to call an `ApplyTextResources` method that refreshes every resource-backed caption and an `UpdateCulturePreview` method that reformats the date, number and currency samples, and recreate the designer-localized customer editor in the same handler. Verify the same page through the `lang` URL parameter, then write a lab note describing what a static culture field would do to a second user signed in at the same time.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `Default.json` with culture set to auto, plus a note on when a fixed culture is the right choice
 - Language `ComboBox` handler setting `Application.CurrentCulture` from `CultureInfo.GetCulturesInfo`
 - `Application.CultureChanged` handler calling `ApplyTextResources` and `UpdateCulturePreview`
 - Designer-localized editor recreated after the switch so no caption stays in the old language
 - Verification through the `lang` URL parameter and a note on why culture must not live in a static field
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:

- Your `ApplyTextResources` method refreshes the dashboard perfectly, yet the customer editor keeps its English captions after a switch to German. Explain what is different about those captions, and name two strategies that fix it.
- A colleague suggests caching the picked `CultureInfo` in a static field "to avoid looking it up twice". Describe what a second user, signed in from another country at the same moment,

would actually see.

– The culture preview shows a currency amount. After a switch from en-US to de-DE, what has changed in memory and what has only changed on screen – and which of the two would you ever write to the database?

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:

- ✓ Your ApplyTextResources method refreshes the dashboard perfectly, yet the customer editor keeps its English captions after a switch to German. Explain what is different about those captions, and name two strategies that fix it.
- ✓ A colleague suggests caching the picked CultureInfo in a static field "to avoid looking it up twice". Describe what a second user, signed in from another country at the same moment, would actually see.
- ✓ The culture preview shows a currency amount. After a switch from en-US to de-DE, what has changed in memory and what has only changed on screen — and which of the two would you ever write to the database?

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.

- ✓ **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."

- ✓ **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."*



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.