

AI Coding Exercise

Build this module's ValidationClinic 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

Complete the ValidationClinic capstone. Write btnSave_Click as a fixed pipeline: reset the summary, call bindingSource.EndEdit(), stop if ValidateChildren() fails, stop if the DataAnnotations check fails, stop if ContactValidator returns business errors, stop if the contacts grid still has row errors, and only then save inside a try/catch that turns a duplicate name into a field error and any other exception into a friendly message. Create SummaryErrorProvider implementing IErrorProvider, assign it to validation.ErrorProvider, and route errors through a SetError helper that writes to both the icon ErrorProvider and the summary provider. Test required, invalid, valid and boundary values and write a short README explaining the validation design.

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

- ✓ Save pipeline that runs EndEdit, ValidateChildren, model, business and grid checks before saving
- ✓ SummaryErrorProvider class implementing IErrorProvider SetError and GetError
- ✓ SetError helper writing to both the icon ErrorProvider and the summary provider
- ✓ try/catch around the save separating a duplicate-name error from system failures
- ✓ README with the validation design and test cases for required, invalid, valid and boundary values

Why it is shaped this way

A user filling in the ValidationClinic intake form never asks which API found a mistake; they only notice whether the screen tells them what is wrong and lets them finish. After six modules the app has field handlers, extender rules, a validator service, annotated models and grid rules, but each was added on its own, and nothing yet guarantees they all run, in the right order, every time Save is clicked. This module joins the layers into one predictable Save pipeline and one visible summary, then asks you to prove the design in a 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
Save pipeline	A fixed sequence inside btnSave_Click that commits pending edits, runs each validation layer in turn and returns at the first layer that fails, so nothing reaches the repository unchecked.
Early return	Each stage ends the handler with return when it finds errors, so later stages never run against data an earlier stage already rejected.
IErrorProvider	The small interface the Wisej.NET validation system sends messages to: SetError(Control, string) to set or clear a message and GetError(Control) to read it back.
SummaryErrorProvider	Your own IErrorProvider that keeps a dictionary of control-to-message pairs and rewrites a summary label from it every time an error is set or cleared.
Dual providers	A single SetError helper that writes the same message to the built-in ErrorProvider (the icon beside the field) and to the summary provider (the list at the top).
User error vs system failure	A mistake the user can fix, such as a duplicate name, becomes a field message; an unexpected exception is logged for developers and shown as a calm, generic message.
Boundary test	A test value sitting exactly on a rule's edge, like age 0 or 120, which catches the off-by-one mistakes that ordinary valid and invalid values miss.

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 ValidationClinic 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 Validation UX, Custom Error Providers and Capstone" – of the LearnWisej Validation 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/validation-in-wisej-net/specs/module-07.md>

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

1. Project "ValidationClinic", 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: Complete the ValidationClinic capstone. Write `btnSave_Click` as a fixed pipeline: reset the summary, call `bindingSource.EndEdit()`, stop if `ValidateChildren()` fails, stop if the `DataAnnotations` check fails, stop if `ContactValidator` returns business errors, stop if the contacts grid still has row errors, and only then save inside a try/catch that turns a duplicate name into a field error and any other exception into a friendly message. Create `SummaryErrorProvider` implementing `IErrorProvider`, assign it to `validation.ErrorProvider`, and route errors through a `SetError` helper that writes to both the icon `ErrorProvider` and the summary provider. Test required, invalid, valid and boundary values and write a short README explaining the validation design.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Save pipeline that runs `EndEdit`, `ValidateChildren`, model, business and grid checks before saving
 - `SummaryErrorProvider` class implementing `IErrorProvider` `SetError` and `GetError`
 - `SetError` helper writing to both the icon `ErrorProvider` and the summary provider
 - try/catch around the save separating a duplicate-name error from system failures
 - README with the validation design and test cases for required, invalid, valid and boundary values
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 tester types a new email into the form and clicks Save immediately; the old email is saved. Which pipeline stage is missing or misplaced, and why does its position matter?
- Your `SummaryErrorProvider` shows "Email is required." twice after the user fixes and breaks the field again. Which part of `SetError` should prevent that, and what does it depend on?
- The repository throws an exception with a database message when the network drops. Walk through what the user sees and what the developer can find afterwards, and explain why this differs from the duplicate-name case.

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 tester types a new email into the form and clicks Save immediately; the old email is saved. Which pipeline stage is missing or misplaced, and why does its position matter?
- ✓ Your SummaryErrorProvider shows "Email is required." twice after the user fixes and breaks the field again. Which part of SetError should prevent that, and what does it depend on?
- ✓ The repository throws an exception with a database message when the network drops. Walk through what the user sees and what the developer can find afterwards, and explain why this differs from the duplicate-name case.

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.

- ✓ **Make it survive a bad day.** *"Review this ValidationClinic 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 ValidationClinic 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/validation-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."



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.