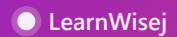


AI Coding Exercise

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.



Writing the code is the easy half now.

Getting working code for "Custom Validation: Methods, Services and ValidationRule Classes" 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

Add custom validation to ValidationClinic in three steps. Write `ValidateStartEndDates` as a method that returns an empty string when valid and a message when the start date is after the end date, and set that message on both `dateTimeStart` and `dateTimeEnd`. Move the name, email and birth-date rules into a `ContactValidator` service that returns a list of `ValidationMessage` records and map each field name back to its control in `ApplyModelValidation`. Then create `MinimumAgeValidationRule` deriving from `ValidationRule` with a `MinimumAge` property, override `OnValidating` to compute the age from the `DateTimePicker`, attach it to `dateTimeBirthDate` through the `Validation` extender, and run the service in `btnSave_Click` before anything is saved.

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

- ✓ `ValidateStartEndDates` method that sets the same error on both date pickers and clears both when fixed
- ✓ `ContactValidator` service returning `ValidationMessage` records for name, email and birth date
- ✓ `ApplyModelValidation` method mapping each field name to its control and the summary
- ✓ `MinimumAgeValidationRule` class overriding `OnValidating`, attached to `dateTimeBirthDate`
- ✓ Save handler that runs field rules, then the validator service, and only then saves

Why it is shaped this way

After Module 3 the ValidationClinic intake form knows that an email needs an @ and an age must be a whole number. It still happily saves a contact whose start date comes after the end date, or a birth date that makes the person eleven years old. The built-in rules check the shape of one value; real applications also need rules about meaning, and some of those rules involve more than one control. This module gives those rules a proper home, so they are written once, shown clearly and never skipped on Save.

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
custom validation method	A method on the form that answers one question and returns an empty string when the data is valid, or a plain-language message when it is not.
validation service	A class such as <code>ContactValidator</code> that checks a model without touching any control, so the same rules run from the form, a grid or a unit test.
ValidationMessage record	A tiny result type holding a Field name and a Message; the service returns a list of them instead of setting errors itself.
ApplyModelValidation	The form method that clears old errors, maps each field name back to its control with <code>errorProvider.SetError</code> , fills the summary and reports whether everything passed.
ValidationRule	The <code>Wisej.NET</code> base class behind every rule of the Validation extender; deriving from it turns your own check into a rule you can attach to any control.
OnValidating / OnValidated / OnControlCreated	The three overrides of a rule class: test the value and return true or false, act after a value has passed, and prepare once the control exists in the browser.
cross-field rule	A rule whose answer depends on two or more controls, such as start date before end date, so its error belongs on every control involved.

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 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 4 – "Custom Validation: Methods, Services and ValidationRule Classes" – 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-04.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: Add custom validation to ValidationClinic in three steps. Write `ValidateStartEndDates` as a method that returns an empty string when valid and a message when the start date is after the end date, and set that message on both `ctpStart` and `ctpEnd`. Move the name, email and birth-date rules into a `ContactValidator` service that returns a list of `ValidationMessage` records and map each field name back to its control in `ApplyModelValidation`. Then create `MinimumAgeValidationRule` deriving from `ValidationRule` with a `MinimumAge` property, override `OnValidating` to compute the age from the `DateTimePicker`, attach it to `ctpBirthDate` through the Validation extender, and run the service in `btnSave_Click` before anything is saved.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `ValidateStartEndDates` method that sets the same error on both date pickers and clears both when fixed
 - `ContactValidator` service returning `ValidationMessage` records for name, email and birth date
 - `ApplyModelValidation` method mapping each field name to its control and the summary
 - `MinimumAgeValidationRule` class overriding `OnValidating`, attached to `ctpBirthDate`
 - Save handler that runs field rules, then the validator service, and only then saves
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:

- The minimum-age check could live in `ContactValidator` or in `MinimumAgeValidationRule`. What does the user experience differently in each case, and when would you want both?
- Your Save handler calls `ApplyModelValidation` after setting the date errors, and the date icons never appear. Explain why, and what order fixes it.
- How would you prove that `ContactValidator` rejects an email without @ without starting the app or opening a browser?

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:

- ✓ The minimum-age check could live in `ContactValidator` or in `MinimumAgeValidationRule`. What does the user experience differently in each case, and when would you want both?
- ✓ Your Save handler calls `ApplyModelValidation` after setting the date errors, and the date icons never appear. Explain why, and what order fixes it.
- ✓ How would you prove that `ContactValidator` rejects an email without @ without starting the app or opening a browser?

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/validation-in-wisej-net/specs/module-04.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 `ValidationClinic` 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 `ValidationClinic` 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.