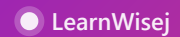


AI Coding Exercise

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.



Brief it, read it, then break it.

This exercise has three parts, and only the first is about prompting. You brief an assistant with the module specification and get a ValidationClinic solution back; you read what it wrote and decide whether it really does what the module asked; then you push it further than the module asked. The second and third parts are where the learning is.

What you are asking for

Replace the hand-written field handlers in ValidationClinic with the Validation extender. Add a Validation component and an ErrorProvider to the intake form, set ContainerControl and validation.ErrorProvider, then call SetValidationRules in a ConfigureValidationRules method: Required then Email on txtEmail, Required then Integer on txtAge, Required then Telephone on txtPhone, Currency on txtCreditLimit and a Regex rule for the customer code. Put Required first everywhere so the most helpful message wins, hook the extender's Validating event to refresh the summary, and move the rule arrays into a small reusable helper so another form can apply the same configuration.

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

- ✓ Validation component wired to the form and to an ErrorProvider display target
- ✓ ConfigureValidationRules method calling SetValidationRules for email, age, phone, credit limit and customer code
- ✓ Required rule placed first on every required field, with plain-language InvalidMessage text
- ✓ Extender Validating handler that keeps the validation summary in step with the rules
- ✓ Reusable rule configuration helper that a second form can call

Why it is shaped this way

At the end of Module 2 the ValidationClinic intake form works, but look at how much code it took: a helper for the name, a helper for the email, a helper for the phone, and each one repeats the same "is it blank, is it the right shape" checks. Add age, credit limit and a customer code and the form grows another handful of near-identical methods. The checks are not wrong; they are just typed out again on every field and every form. This module replaces that repetition with the Validation extender, where you describe what a valid field looks like and Wisej.NET runs the checks for you.

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
Validation extender	A non-visual component you add to a form; it attaches to the Validating and Validated events of every control that has rules, so you no longer write a handler per field.
extended property	A property that a component adds to other controls; once the extender is on the form, each compatible control shows a ValidationRules entry in the Properties window.
SetValidationRules	The code equivalent of the designer: <code>validation.SetValidationRules(txtEmail, new ValidationRule[] { ... })</code> gives one control its list of rules at runtime.
built-in rules	Ready-made checks for common input: <code>RequiredValidationRule</code> , <code>EmailValidationRule</code> , <code>IntegerValidationRule</code> , <code>DecimalValidationRule</code> , <code>CurrencyValidationRule</code> , <code>TelephoneValidationRule</code> and <code>RegexValidationRule</code> .
rule order	Rules run in the order you list them; the first one that fails stops the rest and cancels the control's Validating event.
display target	Where the failing rule's <code>InvalidMessage</code> appears: on the control itself, through an <code>ErrorProvider</code> icon, or as text in a <code>Label</code> , all reached through <code>validation.ErrorProvider</code> .
rule configuration helper	A small method that builds the rule arrays for common fields, so a second form gets the same checks and the same wording by calling 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 3 is about. You can open [module-03.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 3 – "Built-in Validation Rules and the Validation Extender" – 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-03.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: Replace the hand-written field handlers in ValidationClinic with the Validation extender. Add a Validation component and an ErrorProvider to the intake form, set ContainerControl and validation.ErrorProvider, then call SetValidationRules in a ConfigureValidationRules method: Required then Email on txtEmail, Required then Integer on txtAge, Required then Telephone on txtPhone, Currency on txtCreditLimit and a Regex rule for the customer code. Put Required first everywhere so the most helpful message wins, hook the extender's Validating event to refresh the summary, and move the rule arrays into a small reusable helper so another form can apply the same configuration.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Validation component wired to the form and to an ErrorProvider display target
 - ConfigureValidationRules method calling SetValidationRules for email, age, phone, credit limit and customer code
 - Required rule placed first on every required field, with plain-language InvalidMessage text
 - Extender Validating handler that keeps the validation summary in step with the rules
 - Reusable rule configuration helper that a second form can call
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 swap the order so the Email rule runs before Required on txtEmail. Describe exactly what a user sees after leaving the field empty, and why that message is less useful.
- The Module 2 helpers and the extender both end with an icon beside the field. What did you stop writing by hand, and what did you still have to decide yourself?
- A colleague wants to add "the end date must be after the start date" to ClinicRules. Why is the extender the wrong home for that check, and when would it fail to run?

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 swap the order so the Email rule runs before Required on txtEmail. Describe exactly what a user sees after leaving the field empty, and why that message is less useful.
- ✓ The Module 2 helpers and the extender both end with an icon beside the field. What did you stop writing by hand, and what did you still have to decide yourself?
- ✓ A colleague wants to add "the end date must be after the start date" to ClinicRules. Why is the extender the wrong home for that check, and when would it fail to run?

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 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-03.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."



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.