

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 2 just taught you.

What you are asking for

Turn the ValidationClinic contact form into a programmatic validation form. Create one ErrorProvider for the form with ContainerControl set, NeverBlink and a right-aligned icon, write a FieldValidation record plus ValidateName, ValidateEmail and ValidatePhone helpers that return a message instead of touching the UI, and an Apply method that calls SetError for each result and fills a validationSummaryLabel that is visible only while errors exist. Use InvalidMessage on the customer code field to compare the two feedback styles, and make sure fixing one field clears only that field's error.

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

- ✓ Form-level ErrorProvider with ContainerControl, BlinkStyle and icon alignment configured
- ✓ FieldValidation record and ValidateName, ValidateEmail and ValidatePhone helper methods
- ✓ Apply method that sets or clears each control's error and returns whether all fields are valid
- ✓ validationSummaryLabel listing every current problem and hidden when there are none
- ✓ One field using InvalidMessage instead of ErrorProvider, with a note comparing the two

Why it is shaped this way

Module 1 gave the ValidationClinic intake form two Validating handlers, and that works for two fields. A real contact form has a name, an email, a phone number, a customer code and more, and if each field gets its own hand-written handler with its own way of showing a message, the form quickly becomes inconsistent: one field shows an icon, another shows nothing, a third leaves an old error on screen after the user fixed it. This module replaces that drift with one pattern: small helper methods decide what is wrong, one method shows the result, and a visible summary tells every user what to fix without making them hover over anything.

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
ErrorProvider	An extender component that draws an error icon next to any control you give a message to and shows that message as a tooltip; one instance serves the whole form.
ContainerControl	The form or user control the provider belongs to; setting it to this tells the provider which surface its icons are drawn on.
SetError(control, message)	Shows message beside control; calling it with an empty string or null removes the error for that one control only.
GetError, HasErrors, Clear()	Read one control's current message, ask whether any error remains on the form, or remove every error at once when the whole form is reset.
InvalidMessage	A property on editor controls that support the IValidation interface; the control shows the message itself as an invalid tooltip, with no separate provider involved.
FieldValidation record	A tiny result type holding the control, a field name and a message, where an empty message means valid; it lets rules return answers instead of changing the screen.
validationSummaryLabel	A plain Label that lists every current problem in text, visible only while errors exist, so feedback never depends on seeing or hovering an icon.

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 2 is about. You can open [module-02.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 2 – "Programmatic Validation and ErrorProvider Feedback" – 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-02.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: Turn the ValidationClinic contact form into a programmatic validation form. Create one `ErrorProvider` for the form with `ContainerControl` set, `NeverBlink` and a right-aligned icon, write a `FieldValidation` record plus `ValidateName`, `ValidateEmail` and `ValidatePhone` helpers that return a message instead of touching the UI, and an `Apply` method that calls `SetError` for each result and fills a `validationSummaryLabel` that is visible only while errors exist. Use `InvalidMessage` on the customer code field to compare the two feedback styles, and make sure fixing one field clears only that field's error.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Form-level `ErrorProvider` with `ContainerControl`, `BlinkStyle` and icon alignment configured
 - `FieldValidation` record and `ValidateName`, `ValidateEmail` and `ValidatePhone` helper methods
 - `Apply` method that sets or clears each control's error and returns whether all fields are valid
 - `validationSummaryLabel` listing every current problem and hidden when there are none
 - One field using `InvalidMessage` instead of `ErrorProvider`, with a note comparing the two
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 teammate's `txtName_Validating` handler calls `errorProvider.Clear()` as soon as the name becomes valid. What goes wrong for a user who also typed an invalid email, and what should the handler call instead?
- The customer code field shows its message through `InvalidMessage`. What would the summary label need to know about that field to list it too, and what does that tell you about when a shared provider is the better choice?
- A user types a well-formed email that already belongs to another contact. Is that field feedback or business-rule feedback, when should the check run, and where should the message appear?

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 teammate's `txtName_Validating` handler calls `errorProvider.Clear()` as soon as the name becomes valid. What goes wrong for a user who also typed an invalid email, and what should the handler call instead?
- ✓ The customer code field shows its message through `InvalidMessage`. What would the summary label need to know about that field to list it too, and what does that tell you about when a shared provider is the better choice?
- ✓ A user types a well-formed email that already belongs to another contact. Is that field feedback or business-rule feedback, when should the check run, and where should the message appear?

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-02.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.