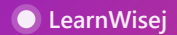


AI Coding Exercise

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.



Same module, built the way you will actually build things.

Every other section of this module explains how "Data-Bound Validation and Model Errors" works. This one asks you to produce it — by directing an assistant with the module's own specification, then verifying the result runs and does what the lab brief describes. It is the workflow you will use on real work, practised on something whose right answer you already know.

What you are asking for

Bind the ValidationClinic intake form to a ContactEditModel through a contactBindingSource and add Required, StringLength, EmailAddress and Range annotations to its Name, Email and Age properties. In btnSave_Click call contactBindingSource.EndEdit() first, run Validator.TryValidateObject with validateAllProperties set to true, and map every ValidationResult member name to its control through a field map and the validation summary. Then implement IDataErrorInfo on a second model, point the ErrorProvider at the binding source with BindToDataAndErrors, and show the same errors appearing through data binding without a hand-written SetError call.

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

- ✓ ContactEditModel with DataAnnotations bound to the form through a BindingSource
- ✓ Save handler that calls EndEdit before running Validator.TryValidateObject
- ✓ Field map that turns ValidationResult member names into control errors and summary lines
- ✓ IDataErrorInfo model whose errors appear through a data-bound ErrorProvider
- ✓ BindToDataAndErrors call used when the provider's data source changes at runtime

Why it is shaped this way

Everything ValidationClinic checks so far lives in the intake form: a Validating handler here, a rule on the Validation extender there, a ContactValidator call in btnSave_Click. But a contact does not only arrive through that form. It can be edited in the contacts grid, imported from a file or created by a service, and none of those paths run the form's handlers. This module moves the basic rules onto the ContactEditModel itself, so the data carries its own definition of "valid" and every screen can ask it the same question.

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
BindingSource	A component that sits between the controls and your object: its DataSource is the model, and each control binds to a property through it instead of being copied by hand.
EndEdit()	Pushes the value the user is still editing from the control into the model, so validation reads what is on screen rather than the last committed value.
DataAnnotations	Attributes such as [Required], [StringLength], [EmailAddress] and [Range] placed on model properties, each with an ErrorMessage written for the user.
Validator.TryValidateObject	The .NET method that runs those attributes against an object and fills a list of ValidationResult objects; it knows nothing about controls or Wisej.NET.
MemberNames	The property names a ValidationResult belongs to, which is the only link between an error message and the control that should display it.
IDataErrorInfo	An interface that lets a model answer "what is wrong with this property?" through an indexer, so the binding layer can ask for errors instead of your code pushing them.
Data-bound ErrorProvider	An ErrorProvider whose DataSource is the same binding source as the controls, so it shows the model's errors next to the bound controls without any SetError call.

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 5 is about. You can open [module-05.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 5 – "Data-Bound Validation and Model Errors" – 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-05.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: Bind the ValidationClinic intake form to a ContactEditModel through a contactBindingSource and add Required, StringLength, EmailAddress and Range annotations to its Name, Email and Age properties. In btnSave_Click call contactBindingSource.EndEdit() first, run Validator.TryValidateObject with validateAllProperties set to true, and map every ValidationResult member name to its control through a field map and the validation summary. Then implement IDataErrorInfo on a second model, point the ErrorProvider at the binding source with BindToDataAndErrors, and show the same errors appearing through data binding without a hand-written SetError call.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - ContactEditModel with DataAnnotations bound to the form through a BindingSource
 - Save handler that calls EndEdit before running Validator.TryValidateObject
 - Field map that turns ValidationResult member names into control errors and summary lines
 - IDataErrorInfo model whose errors appear through a data-bound ErrorProvider
 - BindToDataAndErrors call used when the provider's data source changes at runtime
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 and presses Enter to save, and the old, valid email is what gets checked and stored. Which line is missing, and why does the form look correct on screen while the model is wrong?
- You remove validateAllProperties: true and every save still "works". Which of your four annotations stopped running, and what bad data can now reach the repository?
- The import service creates contacts without opening the form. Which of this module's rules still protect that path, and which ones only run because a user clicked Save?

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 and presses Enter to save, and the old, valid email is what gets checked and stored. Which line is missing, and why does the form look correct on screen while the model is wrong?
- ✓ You remove `validateAllProperties: true` and every save still "works". Which of your four annotations stopped running, and what bad data can now reach the repository?
- ✓ The import service creates contacts without opening the form. Which of this module's rules still protect that path, and which ones only run because a user clicked Save?

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.

- ✓ **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."*
- ✓ **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."*



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.