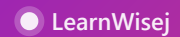


AI Coding Exercise

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.



You are not going to type this sample. You are going to direct it.

Almost nobody writes Wisej.NET code from a blank file any more — an assistant writes the first draft and the developer decides whether it is right. That only works if you know the material, which is what the rest of this module gave you. Here you hand that knowledge to ChatGPT or Claude, get this module's sample back as a working Visual Studio solution, and review it the way you would review a colleague's pull request.

What you are asking for

Make the ValidationClinic contacts grid refuse bad data. Handle `gridContacts.CellValidating`: read `e.FormattedValue` rather than the cell value, clear the cell's `ErrorText`, then reject a blank or `@`-less `EmailColumn` value and an `AgeColumn` value outside 0 to 120 with `e.Cancel = true` and a cell `ErrorText`. Add a `ValidateRow` method that sets row `ErrorText` and the `ClosedDateColumn` cell error when `Status` is `Closed` without a closed date, clear stale errors in `CellValidated`, and handle `DataError` with `ThrowException = false`, a friendly cell message and a debug log of the exception. Skip the new placeholder row until it holds user data, and block `Save` while any row still has an error.

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

- ✓ `CellValidating` handler that reads `e.FormattedValue` and validates the email and age columns
- ✓ `ValidateRow` method setting row `ErrorText` and the closed-date cell error for `Closed` contacts
- ✓ `CellValidated` handler that clears stale cell errors
- ✓ `DataError` handler with `ThrowException = false`, a friendly message and a logged exception
- ✓ `Save` guard that skips the placeholder new row and refuses to save while any row has an error

Why it is shaped this way

Modules 1 to 5 protected the ValidationClinic intake form, one contact at a time. The contacts grid is different: a user can tab through twenty rows, type an email without an `@`, enter "abc" as an age and mark a contact `Closed` without a closed date, all without opening a form. A grid cell is not a `TextBox`, so the `Validating` handlers and the `ErrorProvider` you already wrote do not reach it. This module gives the grid its own field rules, row rules and data-source rules, and makes sure none of them turns a normal typing mistake into a crash or a stuck screen.

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
CellValidating	The grid event that fires when the current cell is about to lose focus; it is the grid's version of a control's Validating event and the place to reject a single value.
e.FormattedValue	The value exactly as the user typed it in the editor, before it is committed to the cell or the data source; reading the cell's Value instead gives you the old value.
e.Cancel	Setting it to true refuses to finish the edit, so the cursor stays in the bad cell until the user corrects it or cancels the edit.
CellValidated	Fires after a cell has passed validation; a safe place to clear an error that no longer applies or refresh a summary.
Cell ErrorText	A message on one cell, shown with an error icon inside that cell; use it for a problem with a single value.
Row ErrorText	A message on the whole row, shown in the row header; use it for a rule that depends on several cells in the same row.
DataError	Raised when the grid cannot convert, format or commit a value; with e.ThrowException = false you replace the exception with a friendly message.

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 6 is about. You can open [module-06.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 6 – "DataGridView Cell Validation and Row 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-06.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: Make the ValidationClinic contacts grid refuse bad data. Handle `gridContacts.CellValidating`: read `e.FormattedValue` rather than the cell value, clear the cell's `ErrorText`, then reject a blank or @-less EmailColumn value and an AgeColumn value outside 0 to 120 with `e.Cancel = true` and a cell `ErrorText`. Add a `ValidateRow` method that sets row `ErrorText` and the `ClosedDateColumn` cell error when `Status` is `Closed` without a closed date, clear stale errors in `CellValidated`, and handle `DataError` with `ThrowException = false`, a friendly cell message and a debug log of the exception. Skip the new placeholder row until it holds user data, and block `Save` while any row still has an error.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `CellValidating` handler that reads `e.FormattedValue` and validates the email and age columns
 - `ValidateRow` method setting row `ErrorText` and the closed-date cell error for `Closed` contacts
 - `CellValidated` handler that clears stale cell errors
 - `DataError` handler with `ThrowException = false`, a friendly message and a logged exception
 - `Save` guard that skips the placeholder new row and refuses to save while any row has an error
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 classmate's email rule reads `grid.Rows[e.RowIndex].Cells[e.ColumnIndex].Value` and "works in testing". Describe the exact edit that slips through, and why.
- Your age rule checks the range 0 to 120. What should it do when the text is not a number at all, and why is a `DataError` handler still needed even if the rule rejects "abc" itself?
- The `Closed` rule could be shown as a cell error, a row error, or both. Which did you choose, and what would a user who never hovers over an icon still see?

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 classmate's email rule reads `grid.Rows[e.RowIndex].Cells[e.ColumnIndex].Value` and "works in testing". Describe the exact edit that slips through, and why.
- ✓ Your age rule checks the range 0 to 120. What should it do when the text is not a number at all, and why is a `DataError` handler still needed even if the rule rejects "abc" itself?
- ✓ The Closed rule could be shown as a cell error, a row error, or both. Which did you choose, and what would a user who never hovers over an icon still see?

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.

- ✓ **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."*
- ✓ **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."



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.