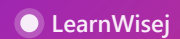


KI-Coding-Übung

Erstellen Sie das Beispielprojekt ValidationClinic dieses Moduls mit ChatGPT oder Claude aus einem fertigen Prompt, der das Modell auf die Spezifikation des Moduls verweist – und prüfen, starten und erweitern Sie dann, was Sie erhalten, auch im Vergleich mit unserem eigenen Referenz-Build auf GitHub.



Der Assistent liefert die schnellen Hände. Der Architekt bleiben Sie.

Ein Assistent erstellt das Beispielprojekt dieses Moduls in wenigen Minuten. Was er nicht kann, ist zu entscheiden, ob das Ergebnis der richtige Entwurf für Ihre Anwendung ist – diese Entscheidung bleibt bei Ihnen. Das Urteilsvermögen bringen Sie mit: Instruieren Sie ihn sorgfältig und prüfen Sie dann, was zurückkommt, anhand dessen, was Ihnen Modul 7 gerade vermittelt hat.

Worum Sie bitten

Schließen Sie das ValidationClinic-Abschlussprojekt ab. Schreiben Sie btnSave_Click als feste Pipeline: Zusammenfassung zurücksetzen, bindingSource.EndEdit() aufrufen, abbrechen, wenn ValidateChildren() fehlschlägt, abbrechen, wenn die DataAnnotations-Prüfung fehlschlägt, abbrechen, wenn ContactValidator Geschäftsfehler zurückgibt, abbrechen, wenn das Kontakt-Grid noch Zeilenfehler hat, und erst dann innerhalb eines try/catch speichern, das einen doppelten Namen in einen Feldfehler und jede andere Exception in eine verständliche Meldung umsetzt. Legen Sie SummaryErrorProvider als Implementierung von IErrorProvider an, weisen Sie ihn validation.ErrorProvider zu und leiten Sie Fehler über eine Hilfsmethode SetError, die sowohl in den Icon-ErrorProvider als auch in den Zusammenfassungs-Provider schreibt. Testen Sie Pflicht-, ungültige, gültige und Grenzwerte und schreiben Sie ein kurzes README, das das Validierungsdesign erklärt.

Das Modul verlangt diese Ergebnisse, und der Prompt unten fordert genau diese vom Assistenten an:

- ✓ Speicher-Pipeline, die EndEdit, ValidateChildren sowie Modell-, Geschäfts- und Grid-Prüfungen vor dem Speichern ausführt
- ✓ Klasse SummaryErrorProvider, die SetError und GetError von IErrorProvider implementiert
- ✓ Hilfsmethode SetError, die sowohl in den Icon-ErrorProvider als auch in den Zusammenfassungs-Provider schreibt
- ✓ try/catch um das Speichern, das einen Fehler wegen eines doppelten Namens von Systemfehlern trennt
- ✓ README mit dem Validierungsdesign und Testfällen für Pflicht-, ungültige, gültige und Grenzwerte

Warum es so aufgebaut ist

Ein Benutzer, der das Aufnahmeformular von ValidationClinic ausfüllt, fragt nie, welche API einen Fehler gefunden hat; er merkt nur, ob die Maske ihm sagt, was falsch ist, und ob er fertig werden kann. Nach sechs Modulen hat die App Feld-Handler, Extender-Regeln, einen Validator-Service, annotierte Modelle und Grid-Regeln, aber jedes davon wurde für sich hinzugefügt, und noch garantiert nichts, dass bei jedem Klick auf Speichern alle laufen, und zwar in der richtigen Reihenfolge. Dieses Modul fügt die Ebenen zu einer vorhersehbaren Speicher-Pipeline und einer sichtbaren Zusammenfassung zusammen und lässt Sie das Design dann in einem Abschlussprojekt belegen.

Diese Begriffe muss der Assistent richtig verwenden. Geht die Antwort nachlässig mit ihnen um, hat er das Modul nicht verstanden:

Begriff	Was er hier bedeutet
Speicher-Pipeline	Eine feste Abfolge in btnSave_Click, die offene Bearbeitungen übernimmt, jede Validierungsebene der Reihe nach ausführt und bei der ersten fehlschlagenden Ebene zurückkehrt, sodass nichts ungeprüft das Repository erreicht.
Frühe Rückkehr	Jede Stufe beendet den Handler mit return, wenn sie Fehler findet, sodass spätere Stufen nie mit Daten laufen, die eine frühere Stufe bereits abgelehnt hat.
IErrorProvider	Das kleine Interface, an das das Validierungssystem von Wisej.NET Meldungen schickt: SetError(Control, string), um eine Meldung zu setzen oder zu entfernen, und GetError(Control), um sie zurückzulesen.
SummaryErrorProvider	Ihr eigener IErrorProvider, der ein Dictionary aus Control-Meldungs-Paaren führt und daraus bei jedem Setzen oder Entfernen eines Fehlers ein Zusammenfassungs-Label neu schreibt.
Doppelte Provider	Eine einzige SetError -Hilfsmethode, die dieselbe Meldung in den integrierten ErrorProvider (das Icon neben dem Feld) und in den Zusammenfassungs-Provider (die Liste oben) schreibt.
Benutzerfehler und Systemfehler	Ein Fehler, den der Benutzer beheben kann, etwa ein doppelter Name, wird zur Feldmeldung; eine unerwartete Exception wird für Entwickler protokolliert und als ruhige, allgemeine Meldung angezeigt.
Grenzwerttest	Ein Testwert genau an der Grenze einer Regel, etwa Alter 0 oder 120, der die Off-by-one-Fehler aufdeckt, die gewöhnliche gültige und ungültige Werte übersehen.

Selbst bauen – wählen Sie Ihren Assistenten

Beide Buttons öffnen einen neuen Tab, in dem **derselbe fertige Prompt** bereits eingetragen ist. Der Prompt verweist das Modell auf die Spezifikation dieses Moduls – eine einzelne Markdown-Datei mit der Lektion, der Lab-Aufgabe und den geforderten Ergebnissen –, sodass es mit dem echten Material arbeitet, statt zu raten, worum es in Modul 7 geht. Sie können [module-07.md](#) selbst öffnen, um genau zu sehen, was es lesen wird.



ChatGPT verwenden

Öffnet chatgpt.com mit dem Prompt dieses Moduls. Funktioniert mit GPT-5 oder Codex; jeder Tarif mit Websuche ruft die Spezifikation ab.



Claude verwenden

Öffnet claude.ai mit demselben Prompt. Wenn Sie Claude Code nutzen, können Sie ihn stattdessen in einem leeren Ordner in ein Terminal einfügen.



Unverändert abgeschickt, erhalten Sie ein Projekt, kein Code-Fragment

Der Prompt verlangt am Ende die gesamte Projektmappe als einzelne [.zip](#) -Datei mit Anleitung zum Kompilieren und Starten. Laden Sie sie herunter, öffnen Sie die Projektmappe in Visual Studio, drücken Sie F5, und ValidationClinic startet auf [localhost](#). Sie benötigen das .NET SDK und das NuGet-Paket [Wisej-4](#); alles andere soll der Assistent laut Prompt selbst schreiben.

Ich arbeite gerade Modul 7 – „Validierungs-UX für den Produktivbetrieb, eigene Error-Provider und Abschlussprojekt“ – des LearnWisej-Kurses „Validierung in Wisej.NET“ durch und möchte das Beispielprojekt des Moduls gemeinsam mit dir erstellen. Bitte antworte auf Deutsch.

Lies zuerst die Modulspezifikation. Sie ist eine einzelne Markdown-Datei (auf Englisch) mit der vollständigen Lektion, der Lab-Aufgabe und den geforderten Ergebnissen:

<https://learnwisej.com/assets/courses/validation-in-wisej-net/specs/module-07.md>

Erstelle dann ein vollständiges Wisej.NET-4-Projekt, das ich in Visual Studio öffnen kann:

1. Projekt „ValidationClinic“, C#, net8.0 oder neuer, NuGet-Paket Wisej-4 4.1.x, Kestrel-Host, der `app.UseWisej()` aufruft.
2. Setze um, was das Lab der Spezifikation verlangt: Schließen Sie das ValidationClinic-Abschlussprojekt ab. Schreiben Sie `btnSave_Click` als feste Pipeline: Zusammenfassung zurücksetzen, `bindingSource.EndEdit()` aufrufen, abbrechen, wenn `ValidateChildren()` fehlschlägt, abbrechen, wenn die DataAnnotations-Prüfung fehlschlägt, abbrechen, wenn `ContactValidator` Geschäftsfehler zurückgibt, abbrechen, wenn das Kontakt-Grid noch Zeilenfehler hat, und erst dann innerhalb eines try/catch speichern, das einen doppelten Namen in einen Feldfehler und jede andere Exception in eine verständliche Meldung umsetzt. Legen Sie `SummaryErrorProvider` als Implementierung von `IErrorProvider` an, weisen Sie ihn `validation.ErrorProvider` zu und leiten Sie Fehler über eine Hilfsmethode `SetError`, die sowohl in den `Icon-ErrorProvider` als auch in den Zusammenfassungs-Provider schreibt. Testen Sie Pflicht-, ungültige, gültige und Grenzwerte und schreiben Sie ein kurzes README, das das Validierungsdesign erklärt.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - Speicher-Pipeline, die `EndEdit`, `ValidateChildren` sowie Modell-, Geschäfts- und Grid-Prüfungen vor dem Speichern ausführt
 - Klasse `SummaryErrorProvider`, die `SetError` und `GetError` von `IErrorProvider` implementiert
 - Hilfsmethode `SetError`, die sowohl in den `Icon-ErrorProvider` als auch in den Zusammenfassungs-Provider schreibt
 - try/catch um das Speichern, das einen Fehler wegen eines doppelten Namens von Systemfehlern trennt
 - README mit dem Validierungsdesign und Testfällen für Pflicht-, ungültige, gültige und Grenzwerte
4. Halte dich an die Spezifikation, wo immer sie eine Wisej.NET-Klasse, -Eigenschaft oder ein -Event nennt – verwende genau diese. Wo sie eine Wahl offenlässt, sag mir, welche API du gewählt hast und warum.
5. Der Server bleibt maßgeblich: Zustand und Validierung leben in .NET, der Browser stellt dar und meldet zurück. Verlagere nie eine Entscheidung nach JavaScript, nur weil es dort einfacher ist.
6. Mach das Ergebnis vorführbar: Ich muss es starten und das Verhalten sehen können, um das es in diesem Modul geht – auch, was passiert, wenn etwas schiefgeht.

Bevor du Code schreibst, fasse in eigenen Worten zusammen, was du bauen wirst und wo jedes Teil liegen wird, damit ich den Entwurf korrigieren kann, solange das noch wenig kostet.

Wenn der Code fertig ist, packe die gesamte Projektmappe in eine einzelne .zip-Datei zum Herunterladen und nenne mir die genauen Schritte, um sie zu kompilieren und auszuführen.

Beantworte zum Schluss diese Fragen aus der Spezifikation und nutze dabei deine eigene Implementierung als Beleg:

– Ein Tester tippt eine neue E-Mail-Adresse in das Formular und klickt sofort auf Speichern; gespeichert wird die alte Adresse. Welche Stufe der Pipeline fehlt oder steht an der falschen Stelle, und warum ist ihre Position wichtig?

– Ihr SummaryErrorProvider zeigt „Email is required.“ zweimal an, nachdem der Benutzer das Feld korrigiert und erneut ungültig gemacht hat. Welcher Teil von SetLastError sollte das verhindern, und wovon hängt er ab?

– Das Repository wirft beim Abbruch der Netzwerkverbindung eine Exception mit einer Datenbankmeldung. Gehen Sie durch, was der Benutzer sieht und was der Entwickler hinterher finden kann, und erklären Sie, warum sich das vom Fall des doppelten Namens unterscheidet.

Frag mich nach allem, was unklar ist, bevor du anfängst.

Prüfen Sie dann, was zurückkommt

Code, der kompiliert, ist noch lange kein richtiger Code. Die spannenden Fragen sind Designfragen: Ist das wirklich das Wisej.NET-Control, das die Aufgabe verlangt? Folgt der Code dem Lebenszyklus, den das Framework vorgibt? Liegt der maßgebliche Zustand auf dem Server – oder wurde er im Browser abgelegt, weil das bequemer war? Lesen Sie das Ergebnis als Reviewer, nicht als Kunde.

Die Checkliste liefert Ihnen dieses Modul. Beantworten Sie jeden Punkt anhand des Codes, den Sie gerade erhalten haben:

- ✓ Ein Tester tippt eine neue E-Mail-Adresse in das Formular und klickt sofort auf Speichern; gespeichert wird die alte Adresse. Welche Stufe der Pipeline fehlt oder steht an der falschen Stelle, und warum ist ihre Position wichtig?
- ✓ Ihr SummaryErrorProvider zeigt „Email is required.“ zweimal an, nachdem der Benutzer das Feld korrigiert und erneut ungültig gemacht hat. Welcher Teil von SetLastError sollte das verhindern, und wovon hängt er ab?
- ✓ Das Repository wirft beim Abbruch der Netzwerkverbindung eine Exception mit einer Datenbankmeldung. Gehen Sie durch, was der Benutzer sieht und was der Entwickler hinterher finden kann, und erklären Sie, warum sich das vom Fall des doppelten Namens unterscheidet.

Alles, was Sie korrigieren mussten, sollten Sie festhalten. Eine kurze Notiz zu den Konventionen und Entscheidungen, auf die Sie sich festgelegt haben – eingefügt in die erste Nachricht Ihrer nächsten Sitzung –, ist mehr wert als jeder noch so clevere Prompt.



Dieses Beispiel haben wir schon gebaut – vergleichen Sie Ihres mit unserem

Die Referenzimplementierung zu Modul 7 liegt im öffentlichen Beispiel-Repository. Bauen Sie zuerst Ihre eigene Version und öffnen Sie dann unsere: Die Unterschiede sind der interessante Teil.



[Beispiel zu Modul 7 auf GitHub öffnen](#)

Oder klonen Sie alle Beispiele mit `git clone https://github.com/iceteagroup/LearnWisej-Samples` und öffnen Sie [Validation in Wisej.NET Course/Module 7](#).



Unser Beispiel ist ein Ausgangspunkt, kein fertiges Produkt

Es ist genauso entstanden wie Ihres – per Prompt und anschließend korrigiert – und endet bewusst dort, wo das Modul endet. Rechnen Sie mit Platzhaltern, wo Produktivcode hingehört, mit knapper Fehlerbehandlung und ohne Tests. Verstehen Sie das als Einladung: Nutzen Sie, was Sie in diesem Modul gelernt haben, und dazu den Assistenten, um es weiterzuentwickeln. Code zu verbessern, den Sie nicht selbst geschrieben haben, ist die Fähigkeit, die dieser Kurs eigentlich vermittelt.

Drei Prompts für den nächsten Schritt

Richten Sie Ihren Assistenten auf das Projekt, das Sie gerade gebaut haben – oder auf unseres –, und probieren Sie diese Prompts aus. Jeder ist eine Übung zum Prüfen und Erweitern, kein Neuschreiben.

- ✓ **Machen Sie es fit für einen schlechten Tag.** *"Untersuche dieses ValidationClinic-Projekt auf Fehlerszenarien, die es noch nicht behandelt – fehlerhafte oder fehlende Eingaben, eine Abhängigkeit, die langsam oder nicht erreichbar ist, Arbeit, die eintrifft, nachdem der Benutzer die Seite verlassen hat, und zwei Aktionen, die sich gegenseitig in die Quere kommen. Sag mir für jedes zuerst, ob es in Wisej.NET 4 ein echtes Risiko ist und warum, und schreib erst dann die Korrektur. Wo du entscheidest, dass nichts behandelt werden muss, sag das und begründe es, statt defensiven Code hinzuzufügen."*
- ✓ **Ergänzen Sie den zweiten Fall.** *"Erweitere dieses ValidationClinic-Projekt um einen zweiten Fall, der demselben Muster folgt wie der erste – eine weitere Entität, Maske oder ein Control derselben Form, auf dieselbe Weise verdrahtet. Lagere wirklich Gemeinsames in eine Basisklasse oder einen Service aus, aber nur dort, wo es sich lohnt; zwei ähnliche Dinge brauchen nicht immer eine Abstraktion. Zeig mir den Diff und begründe jede Auslagerung."*
- ✓ **Beweisen Sie, dass es wirklich funktioniert.** *"Geh dieses Projekt Ergebnis für Ergebnis anhand der Modulspezifikation unter <https://learnwisej.com/assets/courses/validation-in-wisej-net/specs/module-07.md> durch und zeig mir den Beleg, dass jedes wirklich erledigt ist – den Codepfad, worauf ich klicken muss und was ich sehen sollte. Wo ein Ergebnis nur teilweise erfüllt ist, sag das klar, statt es schönzureden. Ergänze dann, was fehlt."*



Nehmen Sie es mit ins Lab

Das praktische Lab am Ende dieses Moduls verlangt dieselben Ergebnisse, die Sie gerade vom Assistenten haben erstellen lassen. Nutzen Sie, was Sie hier haben, als Gerüst für Ihre Antwort und als Beleg dafür – nicht als die Antwort selbst. Bewertet wird im Lab Ihre Argumentation, und dieser Teil muss von Ihnen stammen.