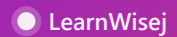


KI-Coding-Übung

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.



Den Code zu schreiben ist heute die leichtere Hälfte.

Funktionierenden Code für „Eigene Validierung: Methoden, Services und ValidationRule-Klassen“ bekommen Sie von einem modernen Assistenten mit einem guten Prompt. Ihn zu beurteilen, zu korrigieren und zu erweitern, erfordert alles, was dieses Modul behandelt hat. Genau auf dieser Aufteilung baut die Übung auf: Sie liefern Spezifikation und Review, der Assistent die Tipparbeit.

Worum Sie bitten

Ergänzen Sie ValidationClinic in drei Schritten um eigene Validierung. Schreiben Sie `ValidateStartEndDates` als Methode, die bei gültigen Werten einen leeren String und eine Meldung zurückgibt, wenn das Startdatum nach dem Enddatum liegt, und setzen Sie diese Meldung sowohl auf `ctpStart` als auch auf `ctpEnd`. Verschieben Sie die Regeln für Name, E-Mail und Geburtsdatum in einen `ContactValidator-Service`, der eine Liste von `ValidationMessage-Records` zurückgibt, und ordnen Sie in `ApplyModelValidation` jeden Feldnamen wieder seinem Control zu. Legen Sie dann `MinimumAgeValidationRule` als Ableitung von `ValidationRule` mit einer Eigenschaft `MinimumAge` an, überschreiben Sie `OnValidating`, um das Alter aus dem `DateTimePicker` zu berechnen, hängen Sie die Regel über den `Validation-Extender` an `ctpBirthDate` und führen Sie den Service in `btnSave_Click` aus, bevor irgendetwas gespeichert wird.

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

- ✓ Methode `ValidateStartEndDates`, die denselben Fehler auf beide Datumsauswahlen setzt und bei der Korrektur beide zurücksetzt
- ✓ `ContactValidator-Service`, der `ValidationMessage-Records` für Name, E-Mail und Geburtsdatum liefert
- ✓ Methode `ApplyModelValidation`, die jeden Feldnamen seinem Control und der Zusammenfassung zuordnet
- ✓ Klasse `MinimumAgeValidationRule`, die `OnValidating` überschreibt und an `ctpBirthDate` hängt
- ✓ Speichern-Handler, der zuerst die Feldregeln, dann den Validator-Service ausführt und erst danach speichert

Warum es so aufgebaut ist

Nach Modul 3 weiß das Aufnahmeformular von ValidationClinic, dass eine E-Mail-Adresse ein @ braucht und ein Alter eine ganze Zahl sein muss. Trotzdem speichert es anstandslos einen Kontakt, dessen Startdatum nach dem Enddatum liegt, oder ein Geburtsdatum, nach dem die Person elf Jahre alt ist. Die integrierten Regeln prüfen die Form eines Werts; echte Anwendungen brauchen aber auch Regeln zur Bedeutung, und manche dieser Regeln betreffen mehr als ein Control. Dieses Modul gibt solchen Regeln ein ordentliches Zuhause, damit sie einmal geschrieben, klar angezeigt und beim Speichern nie übersprungen werden.

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
eigene Validierungsmethode	Eine Methode im Formular, die genau eine Frage beantwortet und einen leeren String zurückgibt, wenn die Daten gültig sind, oder eine verständliche Meldung, wenn nicht.
Validierungs-Service	Eine Klasse wie ContactValidator, die ein Modell prüft, ohne ein Control anzufassen, sodass dieselben Regeln aus dem Formular, einem Grid oder einem Unit-Test laufen.
ValidationMessage -Record	Ein winziger Ergebnistyp mit einem Field -Namen und einer Message; der Service gibt eine Liste davon zurück, statt selbst Fehler zu setzen.
ApplyModelValidation	Die Formularmethode, die alte Fehler entfernt, jeden Feldnamen mit <code>errorProvider.SetError</code> wieder seinem Control zuordnet, die Zusammenfassung füllt und meldet, ob alles bestanden wurde.
ValidationRule	Die Wisej.NET-Basisklasse hinter jeder Regel des Validation-Extenders; leiten Sie davon ab, wird Ihre eigene Prüfung zu einer Regel, die Sie an jedes Control hängen können.
OnValidating / OnValidated / OnControlCreated	Die drei Überschreibungen einer Regelklasse: den Wert prüfen und true oder false zurückgeben, handeln, nachdem ein Wert bestanden hat, und Vorbereitungen treffen, sobald das Control im Browser existiert.
feldübergreifende Regel	Eine Regel, deren Antwort von zwei oder mehr Controls abhängt, etwa Startdatum vor Enddatum, sodass ihr Fehler an jedes beteiligte Control gehört.

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 4 geht. Sie können [module-04.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 4 – „Eigene Validierung: Methoden, Services und ValidationRule-Klassen“ – 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-04.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: Ergänzen Sie `ValidationClinic` in drei Schritten um eigene Validierung. Schreiben Sie `ValidateStartEndDates` als Methode, die bei gültigen Werten einen leeren String und eine Meldung zurückgibt, wenn das Startdatum nach dem Enddatum liegt, und setzen Sie diese Meldung sowohl auf `ctpStart` als auch auf `ctpEnd`. Verschieben Sie die Regeln für Name, E-Mail und Geburtsdatum in einen `ContactValidator-Service`, der eine Liste von `ValidationMessage-Records` zurückgibt, und ordnen Sie in `ApplyModelValidation` jeden Feldnamen wieder seinem Control zu. Legen Sie dann `MinimumAgeValidationRule` als Ableitung von `ValidationRule` mit einer Eigenschaft `MinimumAge` an, überschreiben Sie `OnValidating`, um das Alter aus dem `DateTimePicker` zu berechnen, hängen Sie die Regel über den `Validation-Extender` an `ctpBirthDate` und führen Sie den Service in `btnSave_Click` aus, bevor irgendetwas gespeichert wird.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - Methode `ValidateStartEndDates`, die denselben Fehler auf beide Datumsauswahlen setzt und bei der Korrektur beide zurücksetzt
 - `ContactValidator-Service`, der `ValidationMessage-Records` für Name, E-Mail und Geburtsdatum liefert
 - Methode `ApplyModelValidation`, die jeden Feldnamen seinem Control und der Zusammenfassung zuordnet
 - Klasse `MinimumAgeValidationRule`, die `OnValidating` überschreibt und an `ctpBirthDate` hängt
 - `Speichern-Handler`, der zuerst die Feldregeln, dann den `Validator-Service` ausführt und erst danach speichert
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:

- Die Prüfung auf das Mindestalter könnte in `ContactValidator` oder in

`MinimumAgeValidationRule` stehen. Was erlebt der Benutzer jeweils anders, und wann würden Sie beides wollen?

- Ihr Speichern-Handler ruft `ApplyModelValidation` auf, nachdem er die Datumsfehler gesetzt hat, und die Datums-Icons erscheinen nie. Erklären Sie, warum, und welche Reihenfolge das behebt.
- Wie würden Sie nachweisen, dass `ContactValidator` eine E-Mail-Adresse ohne @ ablehnt, ohne die App zu starten oder einen Browser zu öffnen?

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:

- ✓ Die Prüfung auf das Mindestalter könnte in `ContactValidator` oder in `MinimumAgeValidationRule` stehen. Was erlebt der Benutzer jeweils anders, und wann würden Sie beides wollen?
- ✓ Ihr Speichern-Handler ruft `ApplyModelValidation` auf, nachdem er die Datumsfehler gesetzt hat, und die Datums-Icons erscheinen nie. Erklären Sie, warum, und welche Reihenfolge das behebt.
- ✓ Wie würden Sie nachweisen, dass `ContactValidator` eine E-Mail-Adresse ohne @ ablehnt, ohne die App zu starten oder einen Browser zu öffnen?

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 4 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 4 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 4](#).



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.

- ✓ **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-04.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."*
- ✓ **Prüfen Sie es wie einen Pull-Request.** *"Führ mich durch diesen ValidationClinic-Code, als würdest du den Pull-Request eines Kollegen prüfen und hättest ihn nicht selbst geschrieben. Wo lebt der Zustand, was passiert bei einer einzelnen Benutzeraktion von Anfang bis Ende, was würdest du zur Überarbeitung zurückgeben, und welche Teile würdest du nicht guten Gewissens einem Kunden vorlegen? Sei konkret und zitiere die Zeilen, die du meinst – fasse mir nicht den Entwurf zusammen."*
- ✓ **Tauschen Sie ein Teil aus, ohne die Schnittstellen zu verschieben.** *"Wähle den Teil dieses ValidationClinic-Projekts, der am offensichtlichsten nur ein Platzhalter für etwas Echtes ist – eine Fake-Datenquelle, ein In-Memory-Speicher, ein vereinfachter Service – und ersetze ihn durch eine praxisnahe Implementierung. Der Vertrag auf beiden Seiten darf sich nicht ändern: dieselben Typen, dieselben Aufrufe, dasselbe Verhalten im Fehlerfall. Sag mir, was du durch die echte Implementierung behandeln musstest, was der Platzhalter nicht behandelt hat."*



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.