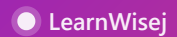


KI-Coding-Übung

Erstellen Sie das Beispielprojekt SupportDesk 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.



Dasselbe Modul – so gebaut, wie Sie in der Praxis wirklich bauen.

Alle anderen Abschnitte dieses Moduls erklären, wie „Validierung, ErrorProvider und Benutzer-Feedback“ funktioniert. Dieser fordert Sie auf, es herzustellen: indem Sie einen Assistenten mit der Spezifikation des Moduls steuern und dann prüfen, ob das Ergebnis läuft und tut, was die Lab-Aufgabe beschreibt. Es ist der Arbeitsablauf, den Sie in echten Projekten nutzen werden – geübt an etwas, dessen richtige Lösung Sie bereits kennen.

Worum Sie bitten

Ergänzen Sie den Ticket-Editor des Support Desk um Validierung: DataAnnotations auf TicketEditModel, einen TicketValidator, der Fehler auf Feldebene und zusammenfassende Fehler liefert, einen ErrorProvider, der vor jedem Durchlauf geleert und für Title, Customer, Category und DueDate gesetzt wird, ein Zusammenfassungs-Label für feldübergreifende Regeln, ein deaktiviertes Save, solange das Modell ungültig ist, einen DbUpdateException-Handler, der bei einer doppelten Ticketnummer eine verständliche Meldung anzeigt, und mindestens fünf Negativ-Testfälle.

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

- ✓ DataAnnotations auf TicketEditModel und ein TicketValidator, der Fehler auf Feldebene und zusammenfassende Fehler liefert
- ✓ ErrorProvider vor jedem Durchlauf geleert und für Title, Customer, Category und DueDate gesetzt
- ✓ Zusammenfassungs-Label für feldübergreifende Regeln und Save deaktiviert, solange das Modell ungültig ist
- ✓ DbUpdateException mit einer verständlichen Meldung für eine doppelte Ticketnummer behandelt
- ✓ Mindestens fünf Negativ-Testfälle

Warum es so aufgebaut ist

Der Ticket-Editor aus Modul 4 speichert, was auch immer der Benutzer eingegeben hat. Ein leerer Titel, ein Ticket ohne Kunde oder ein geschlossenes Ticket, das nächste Woche fällig ist – all das läuft durch SaveChangesAsync und wird, wenn überhaupt, von der Datenbank gestoppt, die mit einer Provider-

Exception antwortet, die kein Support-Mitarbeiter lesen kann. Benutzer brauchen sofortige, verständliche Meldungen, bevor die Datenbank einen Datensatz ablehnt, und die Anwendung braucht eine letzte Verteidigungslinie, die auch dann greift, wenn die UI durch einen Import, eine zweite Oberfläche oder einen Bug umgangen wird.

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
drei Validierungsebenen	Die UI-Validierung gibt sofortiges Feedback, die Domänenvalidierung setzt Regeln durch, die überall gelten müssen, und Datenbank-Constraints garantieren die abschließende Konsistenz; jede Ebene fängt ab, was die darüberliegende übersehen hat.
DataAnnotations auf dem Edit-Modell	Attribute wie [Required] und [StringLength] auf TicketEditModel beschreiben Eingaberegeln an einer deklarativen Stelle, getrennt von der EF-Core-Entität.
explizite Validierung	Validator.TryValidateObject führt die Annotationen bei Bedarf aus; EF Core erledigt das nicht vor SaveChangesAsync für Sie, also muss das Formular es aufrufen.
TicketValidator und ValidationMessage	Ein Service, der ValidationMessage(FieldName, Message) -Einträge liefert, sodass der Regelsatz testbar ist und das Formular nur noch Meldungen den Controls zuordnet.
ErrorProvider	Der Wisej.NET-Extender, der über SetError ein Icon mit Tooltip neben dem ungültigen Control anzeigt und auch an die Datenfehler des aktuellen Eintrags gebunden werden kann.
Validierungszusammenfassung	Ein Label oder eine MessageBox, die feldübergreifende Fehler auflistet, die zu keinem einzelnen Control gehören, etwa ein geschlossenes Ticket mit einem Fälligkeitsdatum in der Zukunft.
DbUpdateException	Die Exception, die EF Core auslöst, wenn die Datenbank die Änderung ablehnt; sie ist die letzte Sicherheitsebene und muss in eine verständliche Meldung übersetzt werden, niemals roh angezeigt.

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 5 geht. Sie können [module-05.md](#) selbst öffnen, um genau zu sehen, was es lesen wird.



ChatGPT verwenden



Claude verwenden

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

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 SupportDesk 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 5 – „Validierung, ErrorProvider und Benutzer-Feedback“ – des LearnWisej-Kurses „Data Binding mit EF Core“ 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/ef-core-binding/specs/module-05.md>

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

1. Projekt „SupportDesk“, 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 den Ticket-Editor des Support Desk um Validierung: DataAnnotations auf `TicketEditModel`, einen `TicketValidator`, der Fehler auf Feldebene und zusammenfassende Fehler liefert, einen `ErrorProvider`, der vor jedem Durchlauf geleert und für Title, Customer, Category und DueDate gesetzt wird, ein Zusammenfassungs-Label für feldübergreifende Regeln, ein deaktiviertes Save, solange das Modell ungültig ist, einen `DbUpdateException`-Handler, der bei einer doppelten Ticketnummer eine verständliche Meldung anzeigt, und mindestens fünf Negativ-Testfälle.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - DataAnnotations auf `TicketEditModel` und ein `TicketValidator`, der Fehler auf Feldebene und zusammenfassende Fehler liefert
 - `ErrorProvider` vor jedem Durchlauf geleert und für Title, Customer, Category und DueDate gesetzt
 - Zusammenfassungs-Label für feldübergreifende Regeln und Save deaktiviert, solange das Modell ungültig ist
 - `DbUpdateException` mit einer verständlichen Meldung für eine doppelte Ticketnummer behandelt
 - Mindestens fünf Negativ-Testfälle
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:

- Wenn Sie alle DataAnnotations von `TicketEditModel` entfernen, sie aber auf der Entität Ticket belassen: Welche Ihrer fünf Negativ-Tests würden dann noch vor `SaveChangesAsync` fehlschlagen, und warum?
- Die Regel für geschlossene Tickets meldet DueDate als Feld. Was würde sich für den Benutzer ändern, wenn sie gar kein Feld meldete, und wann wäre das die bessere Wahl?
- Zwei Agents legen in derselben Sekunde ein Ticket an, und das zweite Speichern löst

DbUpdateException aus. Was sieht der zweite Agent, was protokolliert der Server, und was davon würden Sie einem Kunden zeigen?

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:

- ✓ Wenn Sie alle DataAnnotations von TicketEditModel entfernen, sie aber auf der Entität Ticket belassen: Welche Ihrer fünf Negativ-Tests würden dann noch vor SaveChangesAsync fehlschlagen, und warum?
- ✓ Die Regel für geschlossene Tickets meldet DueDate als Feld. Was würde sich für den Benutzer ändern, wenn sie gar kein Feld meldete, und wann wäre das die bessere Wahl?
- ✓ Zwei Agents legen in derselben Sekunde ein Ticket an, und das zweite Speichern löst DbUpdateException aus. Was sieht der zweite Agent, was protokolliert der Server, und was davon würden Sie einem Kunden zeigen?

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 5 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 5 auf GitHub öffnen](#)

Oder klonen Sie alle Beispiele mit `git clone https://github.com/iceteagroup/LearnWisej-Samples` und öffnen Sie [Data Binding with EF Core Course/Module 5](#).



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 SupportDesk-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 SupportDesk-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/ef-core-binding/specs/module-05.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.