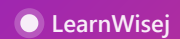


KI-Coding-Übung

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.



Den Code zu schreiben ist heute die leichtere Hälfte.

Funktionierenden Code für „Bidirektionales Binding und CRUD-Editoren“ 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

Bauen Sie die modalen Formulare Add Ticket und Edit Ticket für die Support Desk Data Console: ein TicketEditModel, ein TicketEditorForm mit einer BindingSource und einem ErrorProvider, TextBox-, ComboBox-, DateTimePicker- und CheckBox-Eigenschaften, die an das Modell gebunden sind, LoadEditorAsync für neue und bestehende Tickets, SaveAsync mit EndEdit, einem Platzhalter für die Validierung, Mapping und SaveChangesAsync, Delete mit Bestätigung und frisch geladener Entität sowie ein übergeordnetes Grid, das sich nach DialogResult.OK aktualisiert.

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

- ✓ TicketEditModel und TicketEditorForm mit einer BindingSource und einem ErrorProvider
- ✓ TextBox-, ComboBox-, DateTimePicker- und CheckBox-Eigenschaften an das Edit-Modell gebunden
- ✓ LoadEditorAsync für neue und bestehende Tickets und SaveAsync mit EndEdit, Mapping und SaveChangesAsync
- ✓ Delete mit Bestätigung, frisch geladener Entität und sauberer Behandlung bereits gelöschter Datensätze
- ✓ Übergeordnetes Grid nach DialogResult.OK aktualisiert, und Cancel speichert nichts

Warum es so aufgebaut ist

Die Bearbeiter im Support Desk wollen Oberflächen für Add Ticket und Edit Ticket, die sich wie Desktop-Formulare verhalten: ein modaler Dialog, die Buttons Save, Cancel und Delete und ein Grid, das die Änderung zeigt, sobald sich der Dialog schließt. Bidirektionales Binding macht diese Oberfläche schnell gebaut, aber Save ist eine bewusste Operation und kein Nebeneffekt des Tippens. Liest der Editor einen gebundenen Wert zu früh, speichert er eine Grid-Zeile ohne Tracking, als wäre sie getrackt, oder lässt er

einen Doppelklick zweimal absenden, sieht das Formular weiterhin korrekt aus, während es stillschweigend Daten verliert oder dupliziert. Dieses Modul macht Speichern, Abbrechen, Löschen und Neuladen explizit, sodass sich der Editor wie ein Desktop-Formular anfühlt und trotzdem den Regeln für sicheren Datenzugriff im Web folgt.

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
bidirektionales Binding	Eine Control-Eigenschaft wie <code>TextBox.Text</code> ist an eine Eigenschaft des aktuellen Eintrags gebunden, sodass das Control das Modell anzeigt und das Modell erhält, was der Benutzer eingegeben hat.
<code>DataSourceUpdateMode</code>	Das Argument von <code>DataBindings.Add</code> , das festlegt, wann ein Control seinen Wert in das Modell schreibt: bei der Validierung (wenn der Fokus das Control verlässt) oder bei jeder Eigenschaftsänderung.
EndEdit und CancelEdit	<code>BindingSource.EndEdit</code> übernimmt ausstehende Control-Änderungen in den aktuellen Eintrag; <code>CancelEdit</code> verwirft sie. Save beginnt immer mit <code>EndEdit</code> .
AddNew und RemoveCurrent	Listenoperationen der <code>BindingSource</code> , die einen neuen aktuellen Eintrag anlegen oder den ausgewählten aus der gebundenen Liste entfernen; keine von beiden berührt die Datenbank, bevor Sie speichern.
direktes Binding an Entitäten	Controls direkt an eine getrackte EF-Core-Entität binden, sodass der Change Tracker bereits weiß, was sich geändert hat. Produktiv für kleine, kurzlebige Formulare mit privatem Context.
Edit-Modell	Eine reine UI-Klasse wie <code>TicketEditModel</code> , die gebunden, validiert und dann auf die Entität gemappt wird – so erhalten Validierung, Formatierung und das Ausblenden sicherheitsrelevanter Felder eine eigene Grenze.
Speichern und Löschen mit frischem Context	Jedes Save und Delete erzeugt seinen eigenen <code>DbContext</code> aus der Factory, lädt die Entität über den Schlüssel, ändert sie und ruft <code>SaveChangesAsync</code> auf – so ist die Datenbank die maßgebliche Quelle, nicht die Grid-Zeile.

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



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 4 – „Bidirektionales Binding und CRUD-Editoren“ – 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-04.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: Bauen Sie die modalen Formulare Add Ticket und Edit Ticket für die Support Desk Data Console: ein `TicketEditModel`, ein `TicketEditorForm` mit einer `BindingSource` und einem `ErrorProvider`, `TextBox`-, `ComboBox`-, `DateTimePicker`- und `CheckBox`-Eigenschaften, die an das Modell gebunden sind, `LoadEditorAsync` für neue und bestehende Tickets, `SaveAsync` mit `EndEdit`, einem Platzhalter für die Validierung, `Mapping` und `SaveChangesAsync`, `Delete` mit Bestätigung und frisch geladener Entität sowie ein übergeordnetes Grid, das sich nach `DialogResult.OK` aktualisiert.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - `TicketEditModel` und `TicketEditorForm` mit einer `BindingSource` und einem `ErrorProvider`
 - `TextBox`-, `ComboBox`-, `DateTimePicker`- und `CheckBox`-Eigenschaften an das Edit-Modell gebunden
 - `LoadEditorAsync` für neue und bestehende Tickets und `SaveAsync` mit `EndEdit`, `Mapping` und `SaveChangesAsync`
 - `Delete` mit Bestätigung, frisch geladener Entität und sauberer Behandlung bereits gelöschter Datensätze
 - Übergeordnetes Grid nach `DialogResult.OK` aktualisiert, und `Cancel` speichert nichts
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:

- Welche Lebensdauer haben Sie für den `DbContext` im Editor gewählt, und was müsste für das Formular gelten, damit ein Context mit Formular-Lebensdauer akzeptabel wäre?
- Welche Werte sind an die UI gebunden, welche leben nur im Edit-Modell, welche bleiben nur in der Datenbank – und warum ist Number nicht bearbeitbar?
- Welche Fehlerfälle haben Sie explizit behandelt (Doppelklick, bereits gelöscht Ticket, Ablehnung durch die Datenbank, gleichzeitige Bearbeitung), und was sieht der Benutzer jeweils?

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:

- ✓ Welche Lebensdauer haben Sie für den DbContext im Editor gewählt, und was müsste für das Formular gelten, damit ein Context mit Formular-Lebensdauer akzeptabel wäre?
- ✓ Welche Werte sind an die UI gebunden, welche leben nur im Edit-Modell, welche bleiben nur in der Datenbank – und warum ist Number nicht bearbeitbar?
- ✓ Welche Fehlerfälle haben Sie explizit behandelt (Doppelklick, bereits gelöscht Ticket, Ablehnung durch die Datenbank, gleichzeitige Bearbeitung), und was sieht der Bearbeiter jeweils?

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 [Data Binding with EF Core 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.

- ✓ **Tauschen Sie ein Teil aus, ohne die Schnittstellen zu verschieben.** *"Wähle den Teil dieses SupportDesk-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."*
- ✓ **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."*



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.