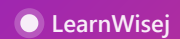


KI-Coding-Übung

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.



Instruieren, lesen, dann auf die Probe stellen.

Diese Übung hat drei Teile, und nur der erste dreht sich um den Prompt. Sie instruieren einen Assistenten mit der Modulspezifikation und erhalten eine SupportDesk-Projektmappe zurück; Sie lesen, was er geschrieben hat, und entscheiden, ob es wirklich tut, was das Modul verlangt; dann treiben Sie es weiter, als das Modul verlangt hat. Im zweiten und dritten Teil steckt das eigentliche Lernen.

Worum Sie bitten

Bauen Sie den Ticket-Browser des Support Desk: die Records TicketListItem und TicketSearchCriteria, eine Methode SearchTicketsAsync, die AsNoTracking, Where, OrderBy, Skip, Take und Select vor ToListAsync zusammensetzt, ein DataGridView, das über eine BindingSource an TicketListItem gebunden ist, Lookup-ComboBoxen für Status und Kunde, die vor der ersten Suche geladen werden, die Buttons Search, Next Page und Previous Page, eine Anzeige der Gesamtzahl und der Seitengröße sowie einen Lade-Schutz, der die UI auch nach Fehlern benutzbar hält.

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

- ✓ Records TicketListItem und TicketSearchCriteria mit SearchTicketsAsync
- ✓ DataGridView über eine BindingSource an TicketListItem gebunden
- ✓ Filterung und Paging in der Abfrage zusammengesetzt, sodass die Datenbank die Arbeit erledigt
- ✓ Lookup-ComboBoxen, die vor der ersten Suche geladen werden, Namen anzeigen und Schlüssel speichern
- ✓ Buttons Search, Next Page und Previous Page mit Gesamtzahl und Lade-Schutz

Warum es so aufgebaut ist

Die Bearbeiter im Support Desk brauchen ein Ticket-Grid, das schnell sucht, ohne die ganze Datenbank in den Speicher des Servers zu ziehen. In einer Wisej.NET-Anwendung hält jede Session ihre eigenen Formulare, BindingSources und ausgewählten Zeilen auf dem Server; eine Oberfläche, die pro Benutzer Tausende getrackte Ticket -Entitäten lädt, multipliziert diese Kosten also mit der Zahl der offenen Browser. Dieses Modul baut den Ticket-Browser richtig: die Abfrage zusammensetzen, asynchron ausführen, eine kleine projizierte Seite materialisieren und diese Liste über eine BindingSource an die UI übergeben.

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
asynchrone Ausführungsmethode	ToListAsync, CountAsync und SingleOrDefaultAsync sind die Aufrufe, die tatsächlich SQL senden; wer sie abwartet, hält den Server-Thread frei, während die Datenbank arbeitet.
Projektion	Ein Select in einen TicketListItem -Record liefert genau die Spalten, die das Grid zeigt, statt der Ticket-Entität samt Graph aus Customer, Agent und Comments.
Abfrage ohne Tracking	AsNoTracking weist den Context an, keine Snapshots der Ergebnisse anzulegen: weniger Speicher, keine versehentlichen Updates und eine Liste, die von jeder Unit of Work zum Bearbeiten unabhängig ist.
BindingSource	Die Wisej.NET-Komponente zwischen einer Daten-Collection und den daran gebundenen Controls; sie verwaltet den aktuellen Eintrag, die Navigation und Listenoperationen und hält Grid und Detail-Controls synchron.
IQueryable	Ein Rezept für eine Datenbankabfrage, keine stabile Liste: Es muss ausgeführt und materialisiert werden, bevor der UI irgendetwas zugewiesen wird.
Lade-Schutz	Ein Flag, deaktivierte Buttons und ein Status-Label rund um jeden abgewarteten Datenbankaufruf, wiederhergestellt in finally, sodass immer nur eine Operation läuft und die Page nach einem Fehler benutzbar bleibt.
Lookup-Binding	Eine ComboBox mit DisplayMember für den angezeigten Namen und ValueMember für den gespeicherten Schlüssel, geladen, bevor die erste Suche läuft oder das Edit-Modell zugewiesen wird.

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 3 geht. Sie können [module-03.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 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 3 – „Daten in BindingSource, DataGridView und Lookup-Controls laden“ – 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-03.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 den Ticket-Browser des Support Desk: die Records `TicketListItem` und `TicketSearchCriteria`, eine Methode `SearchTicketsAsync`, die `AsNoTracking`, `Where`, `OrderBy`, `Skip`, `Take` und `Select` vor `ToListAsync` zusammensetzt, ein `DataGridView`, das über eine `BindingSource` an `TicketListItem` gebunden ist, `Lookup-ComboBoxen` für Status und Kunde, die vor der ersten Suche geladen werden, die Buttons `Search`, `Next Page` und `Previous Page`, eine Anzeige der Gesamtzahl und der Seitengröße sowie einen Lade-Schutz, der die UI auch nach Fehlern benutzbar hält.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - Records `TicketListItem` und `TicketSearchCriteria` mit `SearchTicketsAsync`
 - `DataGridView` über eine `BindingSource` an `TicketListItem` gebunden
 - Filterung und Paging in der Abfrage zusammengesetzt, sodass die Datenbank die Arbeit erledigt
 - `Lookup-ComboBoxen`, die vor der ersten Suche geladen werden, Namen anzeigen und Schlüssel speichern
 - Buttons `Search`, `Next Page` und `Previous Page` mit Gesamtzahl und Lade-Schutz
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` in `SearchTicketsAsync` gewählt, und was würde kaputtgehen, wenn die Page einen Context für ihre gesamte Lebensdauer am Leben hielte?
- Welche Daten hält der Server nach einer Suche für diese Session vor, und welche bleiben nur in der Datenbank? Was kostet ein zweiter, gleichzeitig arbeitender Mitarbeiter?
- Wenn der Kunden-Lookup auf fünfzigtausend Zeilen anwüchse – welche Teile der Page würden Sie ändern, und welche Teile des Abfrage-Service blieben genau so, wie sie sind?

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 in SearchTicketsAsync gewählt, und was würde kaputtgehen, wenn die Page einen Context für ihre gesamte Lebensdauer am Leben hielte?
- ✓ Welche Daten hält der Server nach einer Suche für diese Session vor, und welche bleiben nur in der Datenbank? Was kostet ein zweiter, gleichzeitig arbeitender Bearbeiter?
- ✓ Wenn der Kunden-Lookup auf fünfzigtausend Zeilen anwüchse – welche Teile der Page würden Sie ändern, und welche Teile des Abfrage-Service blieben genau so, wie sie sind?

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 3 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 3 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 3](#).



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.

- ✓ **Prüfen Sie es wie einen Pull-Request.** *"Führ mich durch diesen SupportDesk-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 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."*



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.