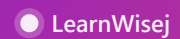


KI-Coding-Übung

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.



Dieses Beispiel tippen Sie nicht ab. Sie steuern es.

Kaum jemand schreibt Wisej.NET-Code heute noch in einer leeren Datei: Ein Assistent liefert den ersten Entwurf, und der Entwickler entscheidet, ob er stimmt. Das funktioniert aber nur, wenn Sie den Stoff beherrschen – und genau den hat Ihnen der Rest dieses Moduls vermittelt. Hier geben Sie dieses Wissen an ChatGPT oder Claude weiter, erhalten das Beispielprojekt des Moduls als lauffähige Visual-Studio-Projektmappe zurück und prüfen es so, wie Sie den Pull-Request eines Kollegen prüfen würden.

Worum Sie bitten

Optimieren Sie den Ticket-Browser des Support Desk: Aktivieren Sie das EF-Core-Logging in der Entwicklung, protokollieren Sie SQL und Laufzeit der nicht optimierten Suche, ersetzen Sie das Laden von Entitäten durch Projektion, ergänzen Sie AsNoTracking bei Grid-Abfragen, führen Sie Paging mit einer Seitengröße von 50 ein, ersetzen Sie Lazy-Navigation-Zugriffe durch Projektion oder Include und dokumentieren Sie die gemessenen Vorher-nachher-Ergebnisse in Ihren Lab-Notizen.

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

- ✓ EF-Core-Logging in der Entwicklung aktiviert, mit protokolliertem SQL und protokollierter Laufzeit der nicht optimierten Abfrage
- ✓ Grid-Abfrage als Projektion ohne Tracking neu geschrieben
- ✓ Paging mit einer Seitengröße von 50 und indizierten Filtern
- ✓ Kein N+1-Verhalten durch Lazy Loading mehr im Anzeige-Code
- ✓ Lab-Notizen, die die gemessene Verbesserung vorher und nachher dokumentieren

Warum es so aufgebaut ist

Der Ticket-Browser des Support Desk hat in Modul 3 mit fünfzig Seed-Tickets funktioniert. Mit fünfzigtausend wird er es nicht mehr, es sei denn, die Abfrage erledigt weniger Arbeit an der richtigen Stelle. Das Versagen ist leise: ein Grid, das drei Sekunden braucht, weil Formatierungscode pro Zeile eine Navigationseigenschaft angefasst hat, eine Session, die Tausende getrackte Entitäten hält, die niemand bearbeiten wird, eine Suche, die eine Tabelle scannt. In diesem Modul geht es darum, bewusst weniger Arbeit zu erledigen, und zwar in SQL – und das mit einem Log zu belegen statt mit einem Bauchgefühl.

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
Projektion	Ein Select, das genau die Felder liefert, die die Oberfläche braucht, etwa TicketListItem, sodass die Datenbank die verknüpften Namen in einer einzigen Anweisung joint.
Include und gefiltertes Include	Eager Loading verknüpfter Entitäten zusammen mit der Wurzel; ein gefiltertes Include grenzt eine Collection ein (die letzten fünf Kommentare), sodass ein Editor ein überschaubares Aggregat lädt.
explizites Laden	Eine Navigation später bei Bedarf laden, über Collection oder Reference des Entry und LoadAsync, wenn der Benutzer einen Detailbereich aufklappt.
Lazy Loading und N+1	Navigationseigenschaften, die beim ersten Zugriff transparent geladen werden; in einem Grid ist das eine versteckte Abfrage pro Zeile oder pro Navigationszugriff.
AsNoTracking und Identity Resolution	Ergebnisse, die für schreibgeschützte Oberflächen am Change Tracker vorbeigehen; AsNoTrackingWithIdentityResolution liefert trotzdem eine Instanz pro Entität, wenn dieselbe Zeile mehrfach vorkommt.
Paging auf Abfrageebene	Where, OrderBy, Skip, Take und Select, zusammengesetzt vor ToListAsync, mit einem separaten CountAsync für die Gesamtzahl.
EF-Core-Logging	Das erzeugte SQL jedes Befehls mit seiner Dauer, über LogTo oder Microsoft.Extensions.Logging.

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 6 geht. Sie können [module-06.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 6 – „Asynchrones Laden, verknüpfte Daten, Filterung und Performance“ – 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-06.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: Optimieren Sie den Ticket-Browser des Support Desk: Aktivieren Sie das EF-Core-Logging in der Entwicklung, protokollieren Sie SQL und Laufzeit der nicht optimierten Suche, ersetzen Sie das Laden von Entitäten durch Projektion, ergänzen Sie `AsNoTracking` bei Grid-Abfragen, führen Sie Paging mit einer Seitengröße von 50 ein, ersetzen Sie Lazy-Navigation-Zugriffe durch Projektion oder Include und dokumentieren Sie die gemessenen Vorher-nachher-Ergebnisse in Ihren Lab-Notizen.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - EF-Core-Logging in der Entwicklung aktiviert, mit protokolliertem SQL und protokollierter Laufzeit der nicht optimierten Abfrage
 - Grid-Abfrage als Projektion ohne Tracking neu geschrieben
 - Paging mit einer Seitengröße von 50 und indizierten Filtern
 - Kein N+1-Verhalten durch Lazy Loading mehr im Anzeige-Code
 - Lab-Notizen, die die gemessene Verbesserung vorher und nachher dokumentieren
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:

- Ihr Log zeigte 151 Anweisungen für eine Oberfläche mit 50 Tickets. Welche Codezeile hat die 150 zusätzlichen verursacht, und warum hat Sie nichts in der Anwendung gewarnt?
- Der Editor braucht das Ticket, seinen Kunden und seine Kategorie sowie die letzten fünf Kommentare; das Kommentar-Panel öffnet den vollständigen Verlauf bei Bedarf. Welche Ladestrategie bedient jeden dieser drei Bedarfe, und warum ist Projektion für den Editor die falsche Antwort?
- Ein Teamkollege schlägt für den Ticket-Browser kompilierte Abfragen und Context-Pooling vor, bevor er sich das Log angesehen hat. Was würden Sie zuerst messen, und was müssten die Zahlen zeigen, damit sich eine der beiden Änderungen lohnt?

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:

- ✓ Ihr Log zeigte 151 Anweisungen für eine Oberfläche mit 50 Tickets. Welche Codezeile hat die 150 zusätzlichen verursacht, und warum hat Sie nichts in der Anwendung gewarnt?
- ✓ Der Editor braucht das Ticket, seinen Kunden und seine Kategorie sowie die letzten fünf Kommentare; das Kommentar-Panel öffnet den vollständigen Verlauf bei Bedarf. Welche Ladestrategie bedient jeden dieser drei Bedarfe, und warum ist Projektion für den Editor die falsche Antwort?
- ✓ Ein Teamkollege schlägt für den Ticket-Browser kompilierte Abfragen und Context-Pooling vor, bevor er sich das Log angesehen hat. Was würden Sie zuerst messen, und was müssten die Zahlen zeigen, damit sich eine der beiden Änderungen lohnt?

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 6 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 6 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 6](#).



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.

- ✓ **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-06.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 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."*



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.