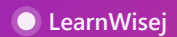


KI-Coding-Übung

Erstellen Sie das Beispielprojekt OpsMonitor 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

Profilieren Sie die beiden wartelastigen Szenarien in WisejPerfLab. Führen Sie die Ticketsuche unter dem Database-Tool aus und ordnen Sie jede Abfrage der UI-Aktion zu: Das Laden des Grids setzt pro angezeigter Zeile eine Kundenabfrage ab, und die Detailabfrage holt große Textspalten, die niemand anzeigt. Ersetzen Sie beide durch eine einzige No-Tracking-Projektionsabfrage, die nur die Grid-Spalten auswählt und das Ergebnis seitenweise liefert. Führen Sie dann den CSV-Export unter File I/O und .NET Async aus, finden Sie den blockierenden .Result-Aufruf im Klick-Handler und die wiederholten kleinen Schreibvorgänge, und verlagern Sie die Erzeugung in einen Application.StartTask-Hintergrund-Workflow, der die Datei streamt und gedrosselten Fortschritt meldet, statt jeden Schritt zu pushen.

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

- ✓ Database-Trace, der Abfragen den UI-Aktionen zuordnet, mit der Anzahl der N+1-Abfragen vor dem Fix
- ✓ Eine einzige No-Tracking-Projektionsabfrage mit Paging anstelle der Abfragen pro Zeile und der zu breiten Abfragen
- ✓ File I/O- und .NET Async-Belege für den Export, die den blockierenden Aufruf und das Schreibmuster nennen
- ✓ Export in einen Hintergrund-Task verlagert, der die Ausgabe streamt und Fortschritt nur bei sinnvollen Schritten pusht
- ✓ Abfrageanzahl, Abfragedauer und Wall-Clock-Zeit des Exports vorher und nachher, mit verbleibenden Risiken

Warum es so aufgebaut ist

Modul 2 hat 1.840 ms verstrichene Zeit gemessen und nur 300 ms CPU-Zeit gefunden. Modul 3 hat den Großteil dieser CPU-Zeit beseitigt, und die Maske ist immer noch nicht schnell, denn die fehlenden 600 ms waren nie Berechnung – sie waren Warten. Das ist die am seltensten richtig diagnostizierte Kategorie in einer Geschäftsanwendung: Der Server ist untätig, der CPU-Graph ist flach, und der Benutzer schaut auf einen Ladeindikator. Irgendwo läuft eine Abfrage einmal pro Zeile, ein Detailformular holt eine Spalte, die niemand anzeigt, oder ein Klick-Handler blockiert auf.Result, während ein Thread-Pool-Thread nichts tut. Die Tools, die das sehen, sind Database, File I/O,.NET Async und Instrumentation, und in diesem Modul bekommt die verbleibende Latenz von WisejPerfLab endlich einen Namen.

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
Database-Tool	Zeichnet die ADO.NET- und EF Core-Abfragen eines Szenarios mit Dauer und Reihenfolge auf, sodass sich jede Abfrage der UI-Aktion zuordnen lässt, die sie ausgelöst hat.
N+1-Abfrage	Eine Abfrage für die Liste, dann eine weitere pro zurückgegebenem Element; mit zehn Zeilen fühlt sich die Liste gut an, bei tausend bricht sie ein – und für CPU-Profiling ist das unsichtbar.
Over-Fetching	Spalten oder Zeilen abfragen, die die Maske nie anzeigt – große Text- und Binärspalten sind die üblichen Verursacher und kosten Abfragezeit, Übertragung und Speicher.
No-Tracking-Abfrage	Eine schreibgeschützte Abfrage ohne Change Tracking, passend für Grids und Listen, die Daten anzeigen statt bearbeiten.
Tool.NET Async	Zeigt Async-Aktivitäten und wo sie warten – so wird eine versehentlich sequenzielle oder blockierte Kette sichtbar.
Blockieren auf Async	.Result oder.Wait() aus einem Handler aufrufen; das hält während des Wartens einen Thread fest und erzeugt unter Last Symptome, die wie Thread-Pool-Starvation aussehen.
Export im Hintergrund	Lange Datei- oder Berichtserzeugung mit Application.StartTask aus dem unmittelbaren Klickpfad herausgelöst, mit Fortschrittsmeldungen in sinnvollen Abständen statt bei jedem Schritt.

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



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 OpsMonitor 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 – „Datenbank, Datei-I/O, Async und externe Wartezeiten“ – des LearnWisej-Kurses „Performance und Profiling“ 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/performance-and-profiling/specs/module-06.md>

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

1. Projekt „OpsMonitor“, 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: Profilieren Sie die beiden wartelastigen Szenarien in `WisejPerfLab`. Führen Sie die Ticketsuche unter dem Database-Tool aus und ordnen Sie jede Abfrage der UI-Aktion zu: Das Laden des Grids setzt pro angezeigter Zeile eine Kundenabfrage ab, und die Detailabfrage holt große Textspalten, die niemand anzeigt. Ersetzen Sie beide durch eine einzige No-Tracking-Projektionsabfrage, die nur die Grid-Spalten auswählt und das Ergebnis seitenweise liefert. Führen Sie dann den CSV-Export unter File I/O und .NET Async aus, finden Sie den blockierenden `.Result`-Aufruf im Klick-Handler und die wiederholten kleinen Schreibvorgänge, und verlagern Sie die Erzeugung in einen `Application.StartTask-Hintergrund-Workflow`, der die Datei streamt und gedrosselten Fortschritt meldet, statt jeden Schritt zu pushen.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - Database-Trace, der Abfragen den UI-Aktionen zuordnet, mit der Anzahl der N+1-Abfragen vor dem Fix
 - Eine einzige No-Tracking-Projektionsabfrage mit Paging anstelle der Abfragen pro Zeile und der zu breiten Abfragen
 - File I/O- und .NET Async-Belege für den Export, die den blockierenden Aufruf und das Schreibmuster nennen
 - Export in einen Hintergrund-Task verlagert, der die Ausgabe streamt und Fortschritt nur bei sinnvollen Schritten pusht
 - Abfrageanzahl, Abfragedauer und Wall-Clock-Zeit des Exports vorher und nachher, mit verbleibenden Risiken
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 Suche setzt nun eine Abfrage statt zweihundertundeiner ab, aber die verstrichene Zeit hat sich nur halbiert. Wo steckt der Rest der Zeit, und mit welchem Tool würden Sie es

herausfinden?

- Ihr Kollege sagt, der Export sei in Ordnung, weil.Result „nur einen Thread blockiert“. Erklären Sie, was bei fünfzig gleichzeitigen Exporten passiert und bei welchen Masken der Benutzer eine Verlangsamung bemerken würde.
- Welche der Dinge, die dieses Lab cacht, lassen sich gefahrlos zwischen Sessions teilen und welche nicht, und nach welcher Regel haben Sie das entschieden?

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 Suche setzt nun eine Abfrage statt zweihundertundeiner ab, aber die verstrichene Zeit hat sich nur halbiert. Wo steckt der Rest der Zeit, und mit welchem Tool würden Sie es herausfinden?
- ✓ Ihr Kollege sagt, der Export sei in Ordnung, weil.Result „nur einen Thread blockiert“. Erklären Sie, was bei fünfzig gleichzeitigen Exporten passiert und bei welchen Masken der Benutzer eine Verlangsamung bemerken würde.
- ✓ Welche der Dinge, die dieses Lab cacht, lassen sich gefahrlos zwischen Sessions teilen und welche nicht, und nach welcher Regel haben Sie das entschieden?

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 `Performance and Profiling 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 OpsMonitor-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/performance-and-profiling/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 OpsMonitor-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.