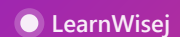


KI-Coding-Übung

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.



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 OpsMonitor-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

Reparieren Sie den Aktualisieren-Button in WisejPerfLab. Profilieren Sie das Aktualisierungsszenario mit CPU Usage, öffnen Sie das Flammendiagramm und Show Hot Path und identifizieren Sie die beiden Verursacher: einen Formatter, der bei jeder Aktualisierung pro Zelle läuft, und einen Control-Neuaufbau, der das KPI-Panel jedes Mal neu erzeugt. Ersetzen Sie beide durch ein DashboardSnapshot-View-Model, dessen Anzeigetexte einmal im Service berechnet werden, ändern Sie die Labels und die Datenquelle des Diagramms einmal innerhalb eines Probe-Scopes statt Zeile für Zeile, und halten Sie den Button während der Arbeit in einem try/finally deaktiviert. Führen Sie das identische Szenario erneut aus und erfassen Sie den neuen Hot Path.

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

- ✓ CPU Usage-Trace vorher, der den wiederholten Formatter und den Control-Neuaufbau mit ihrer gesamten und eigenen CPU-Zeit nennt
- ✓ DashboardSnapshot-View-Model mit im Service vorberechneten Anzeigetexten
- ✓ Gebündelter Aktualisierungs-Handler, der die Controls einmal innerhalb eines Probe-Scopes ändert und den Button per try/finally wieder aktiviert
- ✓ CPU Usage-Trace nachher vom identischen Szenario mit dem neuen Hot Path
- ✓ Vorher-nachher-Tabelle mit gesamter CPU-Zeit, oberster Funktion und Aufrufanzahl sowie dem verbleibenden Risiko

Warum es so aufgebaut ist

Modul 2 endete mit einem Namen: TicketFormatter.FormatRow, 68,9 % eigene CPU-Zeit, fünfzigtausendmal pro Klick aufgerufen. Das ist die Form fast jedes CPU-Problems in einer serverseitigen UI. Niemand hat eine langsame Methode geschrieben; jemand hat eine vernünftige Methode geschrieben

und dann dafür gesorgt, dass sie einmal pro Zelle, pro Aktualisierung, für immer läuft. Der zweite Verursacher im WisejPerfLab-Dashboard ist das Spiegelbild: RebuildKpiPanel wird nur einmal aufgerufen, verwirft aber ein Panel voller Controls, die sich gar nicht ändern mussten, und erzeugt es neu. Das eine ist ein Problem der Aufrufanzahl, das andere ein Problem der Arbeit pro Aufruf – und in diesem Modul geht es darum, zu erkennen, welches Sie vor sich haben, und es zu beheben, ohne zu raten.

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
Kosten eines Event-Handlers	Ein Klick, ein Auswahlwechsel oder ein Timer-Tick läuft auf dem Server und blockiert das nächste Event dieser Session. Die Menge an Logik, die der Klick auslöst, ist also die Zeit, die der Benutzer eingefroren ist.
Flammendiagramm	Dieselben Daten wie die Aufrufstruktur, gezeichnet als Balken mit proportionaler Breite. Ein einzelner breiter Frame – eine Funktion, die das Szenario auffrisst – fällt so auf den ersten Blick auf.
eigene CPU-Zeit vs. Aufrufanzahl	Hohe eigene CPU-Zeit bei wenigen Aufrufen heißt, dass der Rumpf teuer ist; geringe Kosten pro Aufruf bei enormer Aufrufanzahl heißen, dass der Fix darin besteht, seltener aufzurufen, nicht mikrozuoptimieren.
vorberechnetes View-Model	Ein kleines Objekt mit anzeigefertigen Strings und Werten, einmal von einem Service erstellt, sodass die UI-Schicht Werte zuweist, statt sie zu formatieren.
gebündelte UI-Änderung	Die Eigenschaften, die der Benutzer sehen muss, in einem Durchgang mit vorbereiteten Werten setzen, statt Controls in einer Schleife zu ändern, die nebenbei auch noch arbeitet.
Control-Neuaufbau	Controls verwerfen und neu erzeugen, um sie zu aktualisieren; dabei laufen Konstruktion, Layout und Theming erneut, und das ist fast immer teurer, als die vorhandenen Controls zu aktualisieren.
unveränderlicher Lookup-Cache	Referenzdaten, die anwendungsweit geteilt und nie verändert werden; sie lassen sich gefahrlos global cachen – anders als benutzerbezogener Zustand, der in der Session bleiben muss.

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



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 3 – „CPU-Hotpaths, Event-Handler und serverseitige UI-Arbeit“ – 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-03.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: Reparieren Sie den Aktualisieren-Button in `WisejPerfLab`. Profilieren Sie das Aktualisierungsszenario mit CPU Usage, öffnen Sie das Flammendiagramm und Show Hot Path und identifizieren Sie die beiden Verursacher: einen Formatter, der bei jeder Aktualisierung pro Zelle läuft, und einen Control-Neuaufbau, der das KPI-Panel jedes Mal neu erzeugt. Ersetzen Sie beide durch ein `DashboardSnapshot-View-Model`, dessen Anzeigetexte einmal im Service berechnet werden, ändern Sie die Labels und die Datenquelle des Diagramms einmal innerhalb eines `Probe-Scopes` statt Zeile für Zeile, und halten Sie den Button während der Arbeit in einem `try/finally` deaktiviert. Führen Sie das identische Szenario erneut aus und erfassen Sie den neuen Hot Path.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - CPU Usage-Trace vorher, der den wiederholten Formatter und den Control-Neuaufbau mit ihrer gesamten und eigenen CPU-Zeit nennt
 - `DashboardSnapshot-View-Model` mit im Service vorberechneten Anzeigetexten
 - Gebündelter Aktualisierungs-Handler, der die Controls einmal innerhalb eines `Probe-Scopes` ändert und den Button per `try/finally` wieder aktiviert
 - CPU Usage-Trace nachher vom identischen Szenario mit dem neuen Hot Path
 - Vorher-nachher-Tabelle mit gesamter CPU-Zeit, oberster Funktion und Aufrufanzahl sowie dem verbleibenden Risiko
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:

- Beide Verursacher waren langsam, aber aus entgegengesetzten Gründen. Erklären Sie, welcher ein Problem der Aufrufanzahl und welcher ein Problem der Arbeit pro Aufruf war und warum die Profiler-Belege jeweils anders aussehen.
- Ein Kollege schlägt vor, den Aktualisierungs-Handler als `async` zu markieren, um ihn schneller zu machen. Was würde sich tatsächlich ändern, was nicht, und wie würde der Trace

das zeigen?

– Ihr Snapshot wird einmal pro Aktualisierung erstellt. Welches Risiko veralteter Daten oder welches Korrektheitsrisiko entsteht dadurch, und was müssten Sie ändern, wenn derselbe Snapshot über Sessions hinweg gecacht würde?

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:

- ✓ Beide Verursacher waren langsam, aber aus entgegengesetzten Gründen. Erklären Sie, welcher ein Problem der Aufrufanzahl und welcher ein Problem der Arbeit pro Aufruf war und warum die Profiler-Belege jeweils anders aussehen.
- ✓ Ein Kollege schlägt vor, den Aktualisierungs-Handler als async zu markieren, um ihn schneller zu machen. Was würde sich tatsächlich ändern, was nicht, und wie würde der Trace das zeigen?
- ✓ Ihr Snapshot wird einmal pro Aktualisierung erstellt. Welches Risiko veralteter Daten oder welches Korrektheitsrisiko entsteht dadurch, und was müssten Sie ändern, wenn derselbe Snapshot über Sessions hinweg gecacht würde?

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 `Performance and Profiling 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 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."*
- ✓ **Tauschen Sie ein Teil aus, ohne die Schnittstellen zu verschieben.** *"Wähle den Teil dieses OpsMonitor-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 OpsMonitor-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.