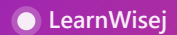


KI-Coding-Übung

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

Legen Sie die WisejPerfLab-Solution mit einem Support-Dashboard, einem Ticket-Grid und einem Kundenbaum an, die aus Seed-Daten gespeist werden, und machen Sie sie anschließend messbar. Fügen Sie einen ScenarioProbe-Service hinzu, der ein Szenario in einen Stopwatch-Scope einschließt und strukturierte PERF-Start/Ende-Logs mit dem Szenarionamen, der Benutzeraktion, den verstrichenen Millisekunden und der Zeilenanzahl schreibt, registrieren Sie ihn im Host-Builder und umschließen Sie damit drei Szenarien: Dashboard aktualisieren, Tickets suchen und Kundenbaum aufklappen. Führen Sie jedes Szenario einmal zum Aufwärmen aus, halten Sie dann die Baseline fest und schreiben Sie ein Performance-Budget, das Umgebung, Build-Konfiguration, Datenmenge und einen Abnahme-Schwellenwert pro Szenario nennt.

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

- ✓ WisejPerfLab-Solution mit Dashboard, Ticket-Grid und Kundenbaum auf Seed-Daten
- ✓ ScenarioProbe-Service, der strukturierte PERF-Start/Ende-Logs mit verstrichenen Millisekunden und Zeilenanzahl schreibt
- ✓ Drei instrumentierte Szenarien: Dashboard aktualisieren, Tickets suchen und Kundenbaum aufklappen
- ✓ Baseline-Notiz mit Umgebung, Build-Konfiguration, Datenmenge und Aufwärmverfahren
- ✓ Performance-Budget mit einem Abnahme-Schwellenwert pro Szenario und dem Tool, das ihn jeweils belegen würde

Warum es so aufgebaut ist

Ein Support-Team meldet, WisejPerfLab sei „langsam geworden“. Dieser Satz enthält keine Messung, kein Szenario und keine Schicht. Das Dashboard könnte auf SQL warten, der Aktualisierungs-Handler könnte

fünzigtausend Zellen formatieren, der Browser könnte ein Grid neu zeichnen, das nie vollständig hätte gerendert werden dürfen, oder der Server hält schlicht die Formulare von zweihundert Sessions, die vor einer Stunde geschlossen wurden. Jede dieser Ursachen hat einen anderen Fix und ein anderes Visual Studio-Tool, und wer den Profiler öffnet, bevor er entschieden hat, um welche es sich handelt, erfasst einen Trace, der die Frage nicht beantworten kann. Dieses Modul baut deshalb zuerst das mentale Modell auf: was zwischen einem Klick im Browser und einer neu gezeichneten Maske tatsächlich passiert, welcher dieser Schritte was kostet und wie Sie ein Szenario und ein Budget schriftlich festhalten, damit die nächsten sechs Module einen Maßstab haben.

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
serverseitige UI	Controls, ihre Eigenschaften und ihr Zustand leben im Serverprozess, und der Browser rendert, was der Server beschreibt. Jedes UI-Objekt, das Sie erzeugen, ist also Serverspeicher, und jede Änderung an der UI ist Server-CPU plus Payload.
Event-Roundtrip	Der vollständige Weg einer Benutzeraktion: Browser-Event, Transport zum Server, Ihr Handler, Datenzugriff, Änderungen an Controls, das serialisierte Update zurück, Neuzeichnen im Browser – die Einheit, die ein Benutzer tatsächlich spürt.
Session als Kapazitätseinheit	Eine Session besitzt Pages, Formulare, Controls, Timer, Auswahlen, Bindings und gecachte Daten. Kapazitätsfragen werden pro Session beantwortet, nicht pro Request, weil dieser Speicher zwischen den Klicks belegt bleibt.
WebSocket-Push	Der Kanal, über den der Server Updates sendet, um die der Benutzer nicht gebeten hat. Er macht die Auslieferung von Live-Daten günstig und einen ungedrosselten Hintergrund-Task teuer, weil jeder Zwischenzustand zu einem Frame wird.
die fünf Kostenkategorien	CPU in Ihrem Code, Blockieren und Warten, Allokation und Retention, Datenzugriff und I/O sowie Update-Payload – die Einordnung, die entscheidet, welches Profiler-Tool Ihr Problem überhaupt sehen kann.
Latenz vs. Durchsatz	Latenz ist, wie lange ein Benutzer wartet; Durchsatz ist, wie viele Sessions der Server tragen kann. Ein Fix, der eine Wartezeit beseitigt, hilft der Latenz; ein Fix, der Arbeit oder Speicher einspart, hilft beidem – und manche Latenz-Fixes kosten Durchsatz.
Szenario und Baseline	Eine benannte, wiederholbare Folge von Benutzeraktionen plus Umgebung, Build-Konfiguration, Datenmenge und die damit gemessenen Werte. Ohne sie ist ein Trace Rauschen und jeder spätere Vergleich bedeutungslos.

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 1 geht. Sie können [module-01.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 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 1 – „Das Performance-Modell von Wisej.NET“ – 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-01.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: Legen Sie die WisejPerfLab-Solution mit einem Support-Dashboard, einem Ticket-Grid und einem Kundenbaum an, die aus Seed-Daten gespeist werden, und machen Sie sie anschließend messbar. Fügen Sie einen ScenarioProbe-Service hinzu, der ein Szenario in einen Stopwatch-Scope einschließt und strukturierte PERF-Start/Ende-Logs mit dem Szenarionamen, der Benutzeraktion, den verstrichenen Millisekunden und der Zeilenanzahl schreibt, registrieren Sie ihn im Host-Builder und umschließen Sie damit drei Szenarien: Dashboard aktualisieren, Tickets suchen und Kundenbaum aufklappen. Führen Sie jedes Szenario einmal zum Aufwärmen aus, halten Sie dann die Baseline fest und schreiben Sie ein Performance-Budget, das Umgebung, Build-Konfiguration, Datenmenge und einen Abnahme-Schwellenwert pro Szenario nennt.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - WisejPerfLab-Solution mit Dashboard, Ticket-Grid und Kundenbaum auf Seed-Daten
 - ScenarioProbe-Service, der strukturierte PERF-Start/Ende-Logs mit verstrichenen Millisekunden und Zeilenanzahl schreibt
 - Drei instrumentierte Szenarien: Dashboard aktualisieren, Tickets suchen und Kundenbaum aufklappen
 - Baseline-Notiz mit Umgebung, Build-Konfiguration, Datenmenge und Aufwärmverfahren
 - Performance-Budget mit einem Abnahme-Schwellenwert pro Szenario und dem Tool, das ihn jeweils belegen würde
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:

- Ihre Dashboard-Aktualisierung loggt 1.900 ms, aber die CPU des Servers bewegt sich während des Szenarios kaum. Welche der fünf Kategorien schließt das aus, auf welche deutet es hin, und was ist die günstigste nächste Messung?
- Die Probe misst den Handler. Nennen Sie zwei Kosten, die der Benutzer beim selben Klick

erlebt und die Ihr elapsedMs -Wert nicht enthält, und sagen Sie, wie Sie jede davon messen würden.

– Warum muss die Baseline-Notiz Datenmenge und Build-Konfiguration festhalten, obwohl Szenarioname und verstrichene Zeit der interessante Teil zu sein scheinen?

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:

- ✓ Ihre Dashboard-Aktualisierung loggt 1.900 ms, aber die CPU des Servers bewegt sich während des Szenarios kaum. Welche der fünf Kategorien schließt das aus, auf welche deutet es hin, und was ist die günstigste nächste Messung?
- ✓ Die Probe misst den Handler. Nennen Sie zwei Kosten, die der Benutzer beim selben Klick erlebt und die Ihr elapsedMs -Wert nicht enthält, und sagen Sie, wie Sie jede davon messen würden.
- ✓ Warum muss die Baseline-Notiz Datenmenge und Build-Konfiguration festhalten, obwohl Szenarioname und verstrichene Zeit der interessante Teil zu sein scheinen?

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 1 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 1 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 1`.



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-01.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.