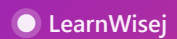


KI-Coding-Übung

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.



Der Assistent liefert die schnellen Hände. Der Architekt bleiben Sie.

Ein Assistent erstellt das Beispielprojekt dieses Moduls in wenigen Minuten. Was er nicht kann, ist zu entscheiden, ob das Ergebnis der richtige Entwurf für Ihre Anwendung ist – diese Entscheidung bleibt bei Ihnen. Das Urteilsvermögen bringen Sie mit: Instruieren Sie ihn sorgfältig und prüfen Sie dann, was zurückkommt, anhand dessen, was Ihnen Modul 2 gerade vermittelt hat.

Worum Sie bitten

Profilieren Sie die langsame Dashboard-Aktualisierung in WisejPerfLab richtig. Wechseln Sie zu Release, wärmen Sie die App einmal auf, erfassen Sie dann mit Alt+F2 einen CPU Usage-Trace genau dieses einen Szenarios, grenzen Sie die Zusammenfassungs-Timeline auf den Klick selbst ein und finden Sie mit Show Hot Path den obersten Pfad unterhalb der Dispatch-Frames von Wisej.NET. Erfassen Sie einen zweiten Instrumentation-Trace desselben Szenarios, um Aufrufanzahlen und Wall-Clock-Zeit zu erhalten, entscheiden Sie anhand der beiden Traces, ob die Kosten CPU-Zeit in Ihrem Code oder Wartezeit sind, und speichern Sie beide .diagsession-Dateien unter einem Namen, der Modul, Szenario, Build, Datenmenge und Zeitstempel enthält.

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

- ✓ Release-CPU Usage-Trace des Szenarios Dashboard-Aktualisierung, Timeline auf den Klick eingegrenzt
- ✓ Hot-Path-Screenshot, der die oberste Anwendungsfunktion und ihre eigene gegenüber der gesamten CPU-Zeit nennt
- ✓ Instrumentation-Trace desselben Szenarios mit Aufrufanzahlen und Wall-Clock-Zeit
- ✓ Trace-Notiz mit Kurs, Modul, Szenario, Build, Hosting, Tool, Browser und erwartetem Budget
- ✓ Ein Absatz zur vermuteten Ursache, der angibt, auf welche der fünf Kategorien die Belege hinweisen

Warum es so aufgebaut ist

Modul 1 hat WisejPerfLab mit einer Zahl, aber ohne Erklärung zurückgelassen: Die Dashboard-Aktualisierung dauert 1.840 ms. Der naheliegende nächste Schritt ist, sie zu profilieren – und genau an diesem Schritt scheitern die meisten Untersuchungen. Ein Trace, der unter dem Debugger erfasst wurde,

misst den Debugger. Ein Trace, der gestartet wird, bevor die App aufgewärmt ist, misst den JIT. Ein Trace über zwei Minuten Klickerei misst alles und beweist nichts. Ein Trace mit vier gleichzeitig aktivierten Tools misst den Overhead des Profilers selbst. Dieses Modul vermittelt die handwerkliche Disziplin, die einen Trace zu einem verwertbaren Beleg macht: der richtige Build, das richtige Tool für die vermutete Kategorie, der richtige Prozess, wenn die App nicht von Visual Studio gestartet wird, ein Szenario pro Erfassung und ein Bericht, den Sie zu lesen wissen.

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
Performance Profiler (Alt+F2)	Der Weg der nachträglichen Auswertung: Er kann das Projekt ohne Debugger starten, ein ganzes Szenario erfassen und einen gespeicherten Bericht erzeugen – daher stammen echte Belege.
Fenster „Diagnosetools“	Die Live-Ansicht mit angehängtem Debugger; nützlich für einen schnellen Blick beim schrittweisen Durchlaufen des Codes, aber irreführend als Quelle für Zeitmessungen, weil der Debugger das Verhalten der Runtime verändert.
Matrix zur Tool-Auswahl	CPU Usage für Berechnungen, Instrumentation für Aufrufanzahlen und Wall-Clock-Zeit, Object Allocation für Speicherumschlag, Memory Usage für Retention, Database für Abfragen, File I/O für Dateien, .NET Async für await-Ketten, .NET Counters für den Live-Zustand.
.diagsession	Die gespeicherte Trace-Datei; erst ein Name mit Modul, Szenario, Build, Datenmenge und Zeitstempel macht einen Vorher-nachher-Vergleich auch Wochen später möglich.
Zusammenfassungs-Timeline	Der CPU- oder Speichergraph über die gesamte Erfassung; die Auswahl auf genau die Benutzeraktion einzugrenzen ist der Schritt, der Start und Aufwärmphase aus der Aufrufstruktur entfernt.
Hot Path	Die teuerste Aufrufkette im ausgewählten Bereich; Show Hot Path klappt sie auf, damit Sie Ihren eigenen Code unter den Framework-Frames finden.
eigene vs. gesamte CPU-Zeit	Die gesamte Zeit schließt aufgerufene Funktionen ein, die eigene nur den Funktionsrumpf: Hohe eigene Zeit bedeutet, dass die Methode selbst teuer ist; hohe Gesamtzeit bei niedriger eigener Zeit bedeutet, dass die Kosten in dem liegen, was sie aufruft.

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 2 geht. Sie können [module-02.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 2 – „Profiling-Workflow in Visual Studio für 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-02.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 langsame Dashboard-Aktualisierung in `WisejPerfLab` richtig. Wechseln Sie zu Release, wärmen Sie die App einmal auf, erfassen Sie dann mit `Alt+F2` einen CPU Usage-Trace genau dieses einen Szenarios, grenzen Sie die Zusammenfassungs-Timeline auf den Klick selbst ein und finden Sie mit `Show Hot Path` den obersten Pfad unterhalb der Dispatch-Frames von Wisej.NET. Erfassen Sie einen zweiten Instrumentation-Trace desselben Szenarios, um Aufrufanzahlen und Wall-Clock-Zeit zu erhalten, entscheiden Sie anhand der beiden Traces, ob die Kosten CPU-Zeit in Ihrem Code oder Wartezeit sind, und speichern Sie beide `.diagsession`-Dateien unter einem Namen, der Modul, Szenario, Build, Datenmenge und Zeitstempel enthält.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - Release-CPU Usage-Trace des Szenarios Dashboard-Aktualisierung, Timeline auf den Klick eingegrenzt
 - Hot-Path-Screenshot, der die oberste Anwendungsfunktion und ihre eigene gegenüber der gesamten CPU-Zeit nennt
 - Instrumentation-Trace desselben Szenarios mit Aufrufanzahlen und Wall-Clock-Zeit
 - Trace-Notiz mit Kurs, Modul, Szenario, Build, Hosting, Tool, Browser und erwartetem Budget
 - Ein Absatz zur vermuteten Ursache, der angibt, auf welche der fünf Kategorien die Belege hinweisen
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 CPU Usage-Trace erklärt nur 300 ms der 1.840 ms, die die Probe gemessen hat. Was bedeutet diese Lücke, und welches Tool würden Sie als Nächstes einsetzen?
- Eine Framework-Methode steht ganz oben im Hot Path, mit sehr hoher gesamter und fast keiner eigenen CPU-Zeit. Erklären Sie, was das bedeutet und was Sie sich ansehen würden,

statt einen Framework-Bug zu melden.

– Warum braucht eine zweite Person die Trace-Notiz, um Ihr Ergebnis zu reproduzieren, obwohl die `diagsession` bereits den gesamten Trace enthält?

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 CPU Usage-Trace erklärt nur 300 ms der 1.840 ms, die die Probe gemessen hat. Was bedeutet diese Lücke, und welches Tool würden Sie als Nächstes einsetzen?
- ✓ Eine Framework-Methode steht ganz oben im Hot Path, mit sehr hoher gesamter und fast keiner eigenen CPU-Zeit. Erklären Sie, was das bedeutet und was Sie sich ansehen würden, statt einen Framework-Bug zu melden.
- ✓ Warum braucht eine zweite Person die Trace-Notiz, um Ihr Ergebnis zu reproduzieren, obwohl die `diagsession` bereits den gesamten Trace enthält?

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 2 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 2 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 2](#).



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.

- ✓ **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-02.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."*
- ✓ **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."*



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.