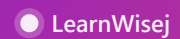


KI-Coding-Übung

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 7 gerade vermittelt hat.

Worum Sie bitten

Liefern Sie das WisejPerfLab-Abschlussprojekt ab. Leiten Sie aus Ihren eigenen Messungen ein Kapazitätsmodell ab: gehaltener Speicher pro inaktiver Session, CPU pro häufigem Szenario, erwartete Spitzenlast an gleichzeitigen Sessions und die Reserve, die Sie einplanen. Schreiben Sie eine HealthCheck.json, deren maxSessions, maxMemory und maxCPU aus diesen Zahlen stammen, begründen Sie jeden Schwellenwert und geben Sie 503 mit retry-after zurück, damit ein Load Balancer neue Benutzer anderswohin leitet, während bestehende Sessions weiterlaufen. Schreiben Sie die Deployment-Notizen für zwei Instanzen hinter einem Load Balancer mit Session-Affinität und WebSocket-Unterstützung und stellen Sie den Abschlussbericht zusammen: Szenarien, Umgebung, Baseline-Belege, Ursachen, Änderungen, Belege nachher, verbleibende Risiken und die Produktivzähler, die zeigen, falls eine Regression zurückkehrt.

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

- ✓ Kapazitätsmodell, das die Sessions pro Server aus dem gemessenen Speicher pro Session und der CPU pro Szenario ableitet
- ✓ HealthCheck.json mit maxSessions, maxMemory, maxCPU, Rückgabecode und retry-after, jeder Schwellenwert begründet
- ✓ Deployment-Notizen zu Session-Affinität, WebSocket-Unterstützung und dem Polling-Fallback
- ✓ Abschließender Performance-Bericht mit Baseline, Ursache, Änderung und Belegen nachher für jeden Fix aus den Modulen
- ✓ Monitoring-Plan, der die Zähler, Schwellenwerte und Alarmer nennt, die jede Regression erkennen würden

Warum es so aufgebaut ist

Sechs Module Profiling haben ein schnelleres WisejPerfLab und einen Ordner voller Traces hervorgebracht. Nichts davon beantwortet die Frage, die das Betriebsteam am Tag des Rollouts stellen wird: Wie viele Benutzer passen auf einen Server, und was passiert, wenn einer voll ist? Profiling auf einem Entwicklerrechner verrät Ihnen die Kosten eines Szenarios; der Produktivbetrieb braucht diese Kosten übersetzt in ein Kapazitätsmodell, in Schwellenwerte, die neue Sessions abweisen, bevor der Prozess in Schwierigkeiten gerät, in eine Deployment-Form, die den serverseitigen Session-Zustand respektiert, und in ein Monitoring, das bemerkt, wenn einer der sechs Fixes eine Regression erleidet. Diese Übersetzung ist Gegenstand dieses Moduls, und der Bericht zum Abschlussprojekt macht die Arbeit für jemanden prüfbar, der Visual Studio nie geöffnet hat.

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
Kapazitätsmodell	Gehaltener Speicher pro inaktiver Session und CPU pro häufigem Szenario, multipliziert mit der erwarteten Spitzenlast an gleichzeitigen Sessions und den Ressourcen der Instanz gegenübergestellt – mit bewusst eingeplanter Reserve.
HealthCheck.json	Konfiguration, über die sich ein Wisej.NET-Server anhand von Session-Anzahl, Speicher und CPU als nicht verfügbar melden kann, damit ein Load Balancer ihm keine neuen Benutzer mehr schickt.
503 mit retry-after	Die Antwort, die eine überlastete Instanz auf einen ersten Request zurückgibt; bestehende Sessions laufen auf ihrem Knoten weiter, während neue Benutzer anderswohin geleitet werden.
Session-Affinität	Konsistentes Routing eines Clients zu der Instanz, die seinen Session-Zustand hält, meist über Sticky Cookies umgesetzt; ohne sie übersteht eine Wisej.NET-Session keinen zweiten Request.
WebSocket-fähiges Load Balancing	Eine Load-Balancer-Konfiguration, die die hochgestufte Verbindung durchlässt; ist WebSocket nicht verfügbar, fällt die App auf Polling zurück, und der Echtzeit-Push geht verloren.
Produktivzähler	Schlanke, laufend erhobene Metriken – CPU, GC-Heap, Allokationsrate, Exception-Rate, Zustand des Thread-Pools –, die eine Regression ohne angehängten Profiler sichtbar machen.
abschließender Performance-Bericht	Szenarien, Umgebung, Baseline-Belege, Ursache, Änderung, Belege nachher, verbleibende Risiken, Schwellenwerte und Monitoring-Plan in einem prüfbaren Dokument.

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 7 geht. Sie können [module-07.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 7 – „Skalierung, Health Checks, Load Balancing und abschließendes Tuning“ – 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-07.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: Liefern Sie das WisejPerfLab-Abschlussprojekt ab. Leiten Sie aus Ihren eigenen Messungen ein Kapazitätsmodell ab: gehaltener Speicher pro inaktiver Session, CPU pro häufigem Szenario, erwartete Spitzenlast an gleichzeitigen Sessions und die Reserve, die Sie einplanen. Schreiben Sie eine `HealthCheck.json`, deren `maxSessions`, `maxMemory` und `maxCPU` aus diesen Zahlen stammen, begründen Sie jeden Schwellenwert und geben Sie 503 mit `retry-after` zurück, damit ein Load Balancer neue Benutzer anderswohin leitet, während bestehende Sessions weiterlaufen. Schreiben Sie die Deployment-Notizen für zwei Instanzen hinter einem Load Balancer mit Session-Affinität und WebSocket-Unterstützung und stellen Sie den Abschlussbericht zusammen: Szenarien, Umgebung, Baseline-Belege, Ursachen, Änderungen, Belege nachher, verbleibende Risiken und die Produktivzähler, die zeigen, falls eine Regression zurückkehrt.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - Kapazitätsmodell, das die Sessions pro Server aus dem gemessenen Speicher pro Session und der CPU pro Szenario ableitet
 - `HealthCheck.json` mit `maxSessions`, `maxMemory`, `maxCPU`, Rückgabecode und `retry-after`, jeder Schwellenwert begründet
 - Deployment-Notizen zu Session-Affinität, WebSocket-Unterstützung und dem Polling-Fallback
 - Abschließender Performance-Bericht mit Baseline, Ursache, Änderung und Belegen nachher für jeden Fix aus den Modulen
 - Monitoring-Plan, der die Zähler, Schwellenwerte und Alarme nennt, die jede Regression erkennen würden
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 Health Check weist ab 150 neue Sessions ab. Ein Kollege fragt, warum nicht 300, da der

Prozess im Test bei 300 nicht abgestürzt ist. Antworten Sie mit Ihren eigenen Zahlen.

- Ein Deployment geht hinter einem Load Balancer ohne Session-Affinität live. Beschreiben Sie genau, was ein Benutzer erlebt und warum das ein Korrektheitsfehler und keine Verlangsamung ist.
- In sechs Monaten ist die Dashboard-Aktualisierung wieder langsam. Welches einzelne Signal aus dem Produktivbetrieb würde es Ihnen zuerst zeigen, und was würden Sie als Nächstes erfassen?

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 Health Check weist ab 150 neue Sessions ab. Ein Kollege fragt, warum nicht 300, da der Prozess im Test bei 300 nicht abgestürzt ist. Antworten Sie mit Ihren eigenen Zahlen.
- ✓ Ein Deployment geht hinter einem Load Balancer ohne Session-Affinität live. Beschreiben Sie genau, was ein Benutzer erlebt und warum das ein Korrektheitsfehler und keine Verlangsamung ist.
- ✓ In sechs Monaten ist die Dashboard-Aktualisierung wieder langsam. Welches einzelne Signal aus dem Produktivbetrieb würde es Ihnen zuerst zeigen, und was würden Sie als Nächstes erfassen?

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 7 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 7 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 7`.



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-07.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.