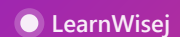


KI-Coding-Übung

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.



Den Code zu schreiben ist heute die leichtere Hälfte.

Funktionierenden Code für „Speicher, Allokationen und Session-Lecks“ bekommen Sie von einem modernen Assistenten mit einem guten Prompt. Ihn zu beurteilen, zu korrigieren und zu erweitern, erfordert alles, was dieses Modul behandelt hat. Genau auf dieser Aufteilung baut die Übung auf: Sie liefern Spezifikation und Review, der Assistent die Tipparbeit.

Worum Sie bitten

Gehen Sie in WisejPerfLab beide Hälften des Speicherproblems an. Führen Sie das Szenario Tickets suchen unter .NET Object Allocation aus, finden Sie die pro Suche allokierten Zeilenmodelle und Strings und reduzieren Sie sie, indem Sie einmal projizieren, statt Objekte bei jedem Neuzeichnen neu aufzubauen. Erstellen Sie danach nach dem Aufwärmen einen Memory Usage-Snapshot, öffnen und schließen Sie das Ticket-Detailformular fünfzigmal, erstellen Sie einen zweiten Snapshot und vergleichen Sie: Die gehaltenen Formulare werden von einem statischen GlobalTicketBus.TicketChanged-Abonnement und einem nie gestoppten Aktualisierungs-Timer verankert. Melden Sie das Abonnement ab und stoppen Sie den Timer in einem überschriebenen Dispose, leeren Sie die Datenquelle der BindingSource, führen Sie dann das Öffnen/Schließen-Szenario erneut aus und zeigen Sie, dass der gehaltene Heap wieder nahe an Snapshot A liegt.

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

- ✓ Allokations-Trace, der den am stärksten allokierenden Typ und Aufrufpfad des Suchszenarios nennt, vorher und nachher
- ✓ Vergleich von Snapshot A/B, der die Anzahl gehaltener Detailformular-Instanzen vor dem Fix zeigt
- ✓ Path-to-Root-Screenshot, der das statische Event-Abonnement und den laufenden Timer identifiziert
- ✓ Dispose-Override, das das statische Event abmeldet, den Timer stoppt und verwirft und das Binding löst
- ✓ Speicherbudget pro Session und eine Dispose-Checkliste, wobei der Snapshot nachher belegt, dass der Retention-Pfad beseitigt ist

Warum es so aufgebaut ist

Das WisejPerfLab-Dashboard hält sein CPU-Budget inzwischen ein, und trotzdem bricht die App nach vier Stunden im Produktivbetrieb zusammen. Speicherprobleme verhalten sich anders als CPU-Probleme: In einer zweiminütigen Demo sind sie unsichtbar, sie zeigen sich nie als einzelner langsamer Klick, und sie sind der Grund, warum ein Server, der morgens bequem achtzig Sessions trägt, am Nachmittag Verbindungen ablehnt. Im Lab gibt es zwei davon, und es ist nicht dasselbe Problem. Die Ticketsuche allokiert weit mehr als nötig, was während der Ausführung GC-Zeit und CPU kostet. Das Ticket-Detailformular wird nach dem Schließen nie wirklich freigegeben, was dauerhaft Speicher kostet. Das eine finden Sie mit einem Allokations-Trace, das andere nur mit zwei Snapshots und einem Path to Root.

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
Allokationsdruck	Wie viel Speicher ein Szenario während der Ausführung erzeugt; hoher Druck erhöht die GC-Häufigkeit und kostet CPU, auch wenn nichts leckt.
gehaltener Speicher	Was noch erreichbar ist, nachdem das Szenario hätte beendet sein sollen; genau das multipliziert sich mit der Anzahl der Sessions und entscheidet über die Kapazität.
.NET Object Allocation	Das Tool, das jede verwaltete Allokation während eines Szenarios aufzeichnet und nach Typ und allozierendem Aufrufpfad gruppiert.
Memory Usage-Snapshots	Heap-Aufnahmen zu gewählten Zeitpunkten; der Vergleich von Snapshot A und Snapshot B zeigt, welche Typen Instanzen hinzugewonnen haben, die nie freigegeben wurden.
Path to Root	Die Kette von Referenzen, die ein Objekt am Leben hält; sie ist der eigentliche Beleg für ein Leck und die einzige Möglichkeit zu beweisen, dass es beseitigt ist.
Retention-Root	Was ganz oben in dieser Kette steht – ein statisches Feld, ein statisches Event, ein laufender Timer, ein Hintergrund-Task oder ein langlebiger Cache.
Speicherbudget pro Session	Eine Zahl in Megabyte gehaltenen Speichers pro inaktiver Session, die aus „die App braucht viel Speicher“ ein Bestanden oder Nicht bestanden und später einen Health-Check-Schwellenwert macht.

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 4 geht. Sie können [module-04.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 4 – „Speicher, Allokationen und Session-Lecks“ – 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-04.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: Gehen Sie in WisejPerfLab beide Hälften des Speicherproblems an. Führen Sie das Szenario Tickets suchen unter .NET Object Allocation aus, finden Sie die pro Suche allokierten Zeilenmodelle und Strings und reduzieren Sie sie, indem Sie einmal projizieren, statt Objekte bei jedem Neuzeichnen neu aufzubauen. Erstellen Sie danach nach dem Aufwärmen einen Memory Usage-Snapshot, öffnen und schließen Sie das Ticket-Detailformular fünfzigmal, erstellen Sie einen zweiten Snapshot und vergleichen Sie: Die gehaltenen Formulare werden von einem statischen `GlobalTicketBus.TicketChanged`-Abonnement und einem nie gestoppten Aktualisierungs-Timer verankert. Melden Sie das Abonnement ab und stoppen Sie den Timer in einem überschriebenen `Dispose`, leeren Sie die Datenquelle der `BindingSource`, führen Sie dann das Öffnen/Schließen-Szenario erneut aus und zeigen Sie, dass der gehaltene Heap wieder nahe an Snapshot A liegt.
3. Liefere diese Ergebnisse und sag mir, wo sich jedes davon in der Projektmappe befindet:
 - Allokations-Trace, der den am stärksten allozierenden Typ und Aufrufpfad des Suchszenarios nennt, vorher und nachher
 - Vergleich von Snapshot A/B, der die Anzahl gehaltener Detailformular-Instanzen vor dem Fix zeigt
 - Path-to-Root-Screenshot, der das statische Event-Abonnement und den laufenden Timer identifiziert
 - `Dispose-Override`, das das statische Event abmeldet, den Timer stoppt und verwirft und das Binding löst
 - Speicherbudget pro Session und eine `Dispose-Checkliste`, wobei der Snapshot nachher belegt, dass der Retention-Pfad beseitigt ist
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 Allokations-Trace hat sich drastisch verbessert, aber der Snapshot-Vergleich ist

unverändert. Welche Art von Problem haben Sie behoben, welche nicht, und wie würde ein Benutzer den Unterschied bemerken?

- Warum öffnet und schließt das Lab das Formular fünfzigmal statt einmal, und was würden Sie aus einer einzelnen Instanz in Snapshot B schließen?
- Der Timer-Callback ist eine Zeile lang und setzt nur ein Label. Erklären Sie genau, wie er ein ganzes Formular am Leben hält und was eine IsDisposed -Prüfung löst und was nicht.

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 Allokations-Trace hat sich drastisch verbessert, aber der Snapshot-Vergleich ist unverändert. Welche Art von Problem haben Sie behoben, welche nicht, und wie würde ein Benutzer den Unterschied bemerken?
- ✓ Warum öffnet und schließt das Lab das Formular fünfzigmal statt einmal, und was würden Sie aus einer einzelnen Instanz in Snapshot B schließen?
- ✓ Der Timer-Callback ist eine Zeile lang und setzt nur ein Label. Erklären Sie genau, wie er ein ganzes Formular am Leben hält und was eine IsDisposed -Prüfung löst und was nicht.

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 4 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 4 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 4`.



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.

- ✓ **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."*
- ✓ **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."*



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.