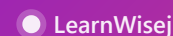


Esercizio di coding con l'IA

Costruisci il progetto OpsMonitor di questo modulo con ChatGPT o Claude partendo da un prompt già pronto che indirizza il modello alla specifica del modulo, poi rivedi, esegui ed estendi quello che ti restituisce — confrontandolo anche con la nostra build di riferimento su GitHub.



L'assistente è la coppia di mani veloci. L'architetto resti tu.

Un assistente può produrre il progetto di questo modulo in pochi minuti. Quello che non può fare è decidere se il risultato è il progetto giusto per la tua applicazione: quella scelta resta tua. Il giudizio lo metti tu: istruiscilo per bene, poi verifica quello che ti torna indietro rispetto a ciò che il Modulo 7 ti ha appena insegnato.

Che cosa stai chiedendo

Consegna il capstone di WisejPerfLab. Ricava un modello di capacità dalle tue misure: memoria trattenuta per sessione inattiva, CPU per scenario comune, concorrenza di picco prevista e il margine che tieni. Scrivi un HealthCheck.json i cui maxSessions, maxMemory e maxCPU derivano da quei numeri e spiega ogni soglia, restituendo 503 con un retry-after così che un load balancer indirizzi altrove i nuovi utenti mentre le sessioni esistenti continuano a funzionare. Scrivi le note di deployment per due istanze dietro un load balancer con affinità di sessione e supporto WebSocket, e componi il report finale: scenari, ambiente, prove di baseline, cause radice, modifiche, prove successive, rischi residui e i contatori di produzione che mostreranno se una regressione ritorna.

Il modulo chiede questi deliverable, e il prompt qui sotto li chiede all'assistente esattamente così:

- ✓ Modello di capacità che ricava le sessioni per server dalla memoria per sessione e dalla CPU per scenario misurate
- ✓ HealthCheck.json con maxSessions, maxMemory, maxCPU, codice di ritorno e retry-after, ogni soglia motivata
- ✓ Note di deployment che coprono affinità di sessione, supporto WebSocket e fallback con polling
- ✓ Report finale sulle prestazioni con baseline, causa radice, modifica e prove successive per ogni correzione dei moduli
- ✓ Piano di monitoraggio che indica i contatori, le soglie e gli alert che intercetterebbero ogni regressione

Perché è fatto così

Sei moduli di profiling hanno prodotto un WisejPerfLab più veloce e una cartella di tracce. Niente di tutto questo risponde alla domanda che il team operativo farà il giorno del rilascio: quanti utenti stanno su un server, e che cosa succede quando uno è pieno? Il profiling sulla macchina di uno sviluppatore ti dice il

costo di uno scenario; la produzione ha bisogno che quel costo venga convertito in un modello di capacità, in soglie che rifiutano nuove sessioni prima che il processo sia in difficoltà, in una forma di deployment che rispetti lo stato di sessione lato server e in un monitoraggio che si accorga se una qualsiasi delle sei correzioni regredisce. Questa traduzione è il contenuto di questo modulo, e il report del capstone è ciò che rende il lavoro verificabile da qualcuno che non ha mai aperto Visual Studio.

Questi sono i termini che l'assistente deve azzeccare. Se quello che ti restituisce li usa con leggerezza, non ha capito il modulo:

Termine	Che cosa significa qui
modello di capacità	Memoria trattenuta per sessione inattiva e CPU per scenario comune, moltiplicate per la concorrenza di picco prevista e rapportate alle risorse dell'istanza, con un margine tenuto deliberatamente.
HealthCheck.json	La configurazione con cui un server Wisej.NET può dichiararsi non disponibile in base al numero di sessioni, alla memoria e alla CPU, così che un load balancer smetta di inviargli nuovi utenti.
503 con retry-after	La risposta che un'istanza non in salute restituisce a una prima richiesta; le sessioni esistenti continuano a girare sul loro nodo mentre i nuovi utenti vengono indirizzati altrove.
affinità di sessione	L'instradamento coerente di un client verso l'istanza che contiene il suo stato di sessione, di solito implementato con cookie sticky; senza di essa una sessione Wisej.NET non sopravvive a una seconda richiesta.
bilanciamento compatibile con WebSocket	Una configurazione del load balancer che lascia passare la connessione aggiornata a WebSocket; quando WebSocket non è disponibile l'app ripiega sul polling e il push in tempo reale si perde.
contatori di produzione	Metriche continue e leggere — CPU, heap del GC, frequenza di allocazione, frequenza delle eccezioni, stato del thread pool — che mostrano una regressione senza un profiler agganciato.
report finale sulle prestazioni	Scenari, ambiente, prove di baseline, causa radice, modifica, prove successive, rischi residui, soglie e piano di monitoraggio, in un unico documento verificabile.

Costruiscilo — scegli il tuo assistente

Entrambi i pulsanti aprono una nuova scheda con lo **stesso prompt già pronto**. Il prompt indirizza il modello alla specifica di questo modulo — un unico file Markdown con la lezione, la traccia del lab e i deliverable — così lavora sul materiale vero invece di tirare a indovinare che cosa sia il Modulo 7. Puoi aprire tu stesso [module-07.md](#) per vedere esattamente che cosa leggerà.



Usa ChatGPT



Usa Claude

Apri chatgpt.com con il prompt di questo modulo già caricato. Funziona con GPT-5 o Codex; qualsiasi piano che possa navigare il web recupererà la specifica.

Apri claude.ai con lo stesso prompt già caricato. Chi usa Claude Code può invece incollarlo in un terminale dentro una cartella vuota.



Mandalo così com'è e ottieni un progetto, non un frammento

Il prompt si chiude chiedendo l'intera soluzione in un unico [.zip](#) con i passi per compilarla ed eseguirla. Scaricalo, apri la soluzione in Visual Studio, premi F5 e OpsMonitor si apre su [localhost](#). Ti servono l'SDK .NET e il pacchetto NuGet [Wisej-4](#); tutto il resto il prompt chiede all'assistente di scriverlo da sé.

Sto seguendo il Modulo 7 - "Scalabilità, health check, bilanciamento del carico e ottimizzazione finale" - del corso LearnWisej "Prestazioni e profiling", e voglio costruire con te il progetto del modulo. Rispondimi in italiano.

Comincia leggendo la specifica del modulo. È un unico file Markdown (in inglese) che contiene l'intera lezione, la traccia del lab e i deliverable:

<https://learnwisej.com/assets/courses/performance-and-profiling/specs/module-07.md>

Poi costruisci un progetto Wisej.NET 4 completo che io possa aprire in Visual Studio:

1. Progetto "OpsMonitor", C#, net8.0 o successivo, pacchetto NuGet Wisej-4 4.1.x, host Kestrel che chiama `app.UseWisej()`.
2. Costruisci quello che chiede il lab della specifica: Consegna il capstone di WisejPerflab. Ricava un modello di capacità dalle tue misure: memoria trattenuta per sessione inattiva, CPU per scenario comune, concorrenza di picco prevista e il margine che tieni. Scrivi un `HealthCheck.json` i cui `maxSessions`, `maxMemory` e `maxCPU` derivano da quei numeri e spiega ogni soglia, restituendo 503 con un `retry-after` così che un load balancer indirizzi altrove i nuovi utenti mentre le sessioni esistenti continuano a funzionare. Scrivi le note di deployment per due istanze dietro un load balancer con affinità di sessione e supporto WebSocket, e componi il report finale: scenari, ambiente, prove di baseline, cause radice, modifiche, prove successive, rischi residui e i contatori di produzione che mostreranno se una regressione ritorna.
3. Produci questi deliverable e dimmi dove si trova ciascuno nella soluzione:
 - Modello di capacità che ricava le sessioni per server dalla memoria per sessione e dalla CPU per scenario misurate
 - `HealthCheck.json` con `maxSessions`, `maxMemory`, `maxCPU`, codice di ritorno e `retry-after`, ogni soglia motivata
 - Note di deployment che coprono affinità di sessione, supporto WebSocket e fallback con polling
 - Report finale sulle prestazioni con baseline, causa radice, modifica e prove successive per ogni correzione dei moduli
 - Piano di monitoraggio che indica i contatori, le soglie e gli alert che intercetterebbero ogni regressione
4. Segui la specifica ovunque nomini una classe, una proprietà o un evento di Wisej.NET - usa esattamente quello. Dove lascia una scelta aperta, dimmi quale API hai scelto e perché.
5. Tieni l'autorità sul server: stato e validazione vivono in .NET, il browser disegna e riferisce. Non spostare mai una decisione in JavaScript solo perché lì è più comodo.
6. Rendi il risultato dimostrabile: devo poterlo eseguire e vedere il comportamento di cui parla questo modulo, compreso che cosa succede quando qualcosa va storto.

Prima di scrivere codice, riformula con parole tue che cosa costruirai e dove starà ogni pezzo, così posso correggere il progetto finché costa poco.

Quando il codice è pronto, impacchetta l'intera soluzione in un unico file .zip scaricabile e dammi i passi esatti per compilarla ed eseguirla.

Infine rispondi a queste domande della specifica, usando come prova la tua implementazione:

- Il tuo health check rifiuta nuove sessioni a 150. Un collega chiede perché non 300, visto che nei test il processo non è andato in crash a 300. Rispondi con i tuoi numeri.
- Un deployment va in produzione dietro un load balancer senza affinità di sessione. Descrivi esattamente che cosa sperimenta un utente, e perché è un errore di correttezza e non un rallentamento.

– Fra sei mesi l'aggiornamento della dashboard è di nuovo lento. Quale singolo segnale di produzione te lo direbbe per primo, e che cosa raccoglieresti dopo?

Chiedimi qualsiasi cosa non ti sia chiara prima di cominciare.

Poi rivedi quello che ti è tornato indietro

Codice che compila non è ancora codice giusto. Le domande interessanti sono domande di progetto: è questo il controllo Wisej.NET che il compito richiede davvero, segue il ciclo di vita che il framework ti offre, lo stato autorevole sta sul server o è stato parcheggiato nel browser perché era comodo? Leggi il risultato da revisore, non da cliente.

Questo modulo ti dà la checklist. Rispondi a ciascuna domanda guardando il codice che hai appena ricevuto:

- ✓ Il tuo health check rifiuta nuove sessioni a 150. Un collega chiede perché non 300, visto che nei test il processo non è andato in crash a 300. Rispondi con i tuoi numeri.
- ✓ Un deployment va in produzione dietro un load balancer senza affinità di sessione. Descrivi esattamente che cosa sperimenta un utente, e perché è un errore di correttezza e non un rallentamento.
- ✓ Fra sei mesi l'aggiornamento della dashboard è di nuovo lento. Quale singolo segnale di produzione te lo direbbe per primo, e che cosa raccoglieresti dopo?

Tutto ciò che hai dovuto correggere vale la pena di scriverlo. Una breve nota con le convenzioni e le decisioni a cui sei arrivato, incollata nel primo messaggio della sessione successiva, vale più di qualsiasi prompt ingegnoso.



Questo l'abbiamo già costruito — confronta il tuo con il nostro

L'implementazione di riferimento del Modulo 7 si trova nel repository pubblico degli esempi. Costruisci prima il tuo, poi apri il nostro: la parte interessante sono le differenze.



[Apri l'esempio del Modulo 7 su GitHub](#)

Oppure clona tutto con `git clone https://github.com/iceteagroup/LearnWisej-Samples` e apri `Performance and Profiling Course/Module 7`.



Il nostro esempio è un punto di partenza, non un prodotto finito

È stato costruito nello stesso modo in cui stai costruendo il tuo — con un prompt, poi corretto — e si ferma deliberatamente dove si ferma il modulo. Aspettati soluzioni provvisorie dove andrebbe codice di produzione, gestione degli errori essenziale e nessun test. Prendilo come un invito: usa quello che hai imparato in questo modulo, più l'assistente, per portarlo più avanti. Migliorare codice che non hai scritto tu è l'abilità che questo corso insegna davvero.

Tre prompt per andare oltre

Punta il tuo assistente sul progetto che hai appena costruito — o sul nostro e prova questi. Ognuno è un esercizio di revisione ed estensione, non una riscrittura.

- ✓ **Dimostra che funziona davvero.** *"Passa in rassegna questo progetto deliverable per deliverable rispetto alla specifica del modulo su <https://learnwisej.com/assets/courses/performance-and-profiling/specs/module-07.md> e mostrami la prova che ciascuno è davvero fatto — il percorso nel codice, su che cosa cliccare e che cosa dovrei vedere. Dove un deliverable è soddisfatto solo in parte, dillo chiaramente invece di arrampicarti sugli specchi. Poi aggiungi quello che manca."*
- ✓ **Rivedilo come faresti con una pull request.** *"Accompagnami attraverso questo codice OpsMonitor come se stessi rivedendo la pull request di un collega e non l'avessi scritto tu. Dove vive lo stato, che cosa succede dall'inizio alla fine su una singola azione dell'utente, che cosa rimanderesti indietro per modifiche, e quali parti non ti sentiresti tranquillo a mettere davanti a un cliente? Sii specifico e cita le righe di cui parli — non riassumermi il progetto."*
- ✓ **Sostituisci un pezzo senza spostare le giunzioni.** *"Scegli la parte di questo progetto OpsMonitor che più evidentemente sta al posto di qualcosa di reale — una sorgente dati finta, un archivio in memoria, un servizio semplificato — e sostituiscila con un'implementazione di forma produttiva. Il contratto ai due lati non deve cambiare: stessi tipi, stesse chiamate, stesso comportamento in caso di errore. Dimmi che cosa l'implementazione reale ti ha costretto a gestire e che la versione provvisoria non gestiva."*



Portalo dentro il lab

Il lab pratico che chiude questo modulo chiede gli stessi deliverable che hai appena fatto produrre all'assistente. Usa quello che hai qui come forma della tua risposta e come prova a sostegno — non come la risposta in sé. Il lab viene valutato sul tuo ragionamento, e quella parte deve essere tua.