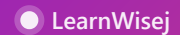


Esercizio di coding con l'IA

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.



Istruiscilo, leggilo, poi mettilo alla prova.

Questo esercizio ha tre parti, e solo la prima riguarda il prompt. Istruisci un assistente con la specifica del modulo e ti fai restituire una soluzione OpsMonitor; leggi quello che ha scritto e decidi se fa davvero ciò che il modulo chiedeva; poi lo spingi più in là di quanto il modulo chiedesse. La seconda e la terza parte sono dove si impara.

Che cosa stai chiedendo

Sistema il pulsante Refresh di WisejPerfLab. Esegui il profiling dello scenario di aggiornamento con CPU Usage, apri il flame graph e Show Hot Path e individua i due colpevoli: un formatter che gira per ogni cella a ogni aggiornamento e una ricostruzione di controlli che ricrea ogni volta il pannello dei KPI. Sostituiscili con un view model DashboardSnapshot le cui stringhe di visualizzazione vengono calcolate una sola volta nel servizio, modifica le label e la sorgente dati del grafico una sola volta dentro uno scope della sonda invece che riga per riga, e tieni il pulsante disabilitato in un try/finally mentre il lavoro è in corso. Riesegui lo scenario identico e cattura il nuovo percorso critico.

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

- ✓ Traccia CPU Usage prima della modifica che indica il formatter ripetuto e la ricostruzione dei controlli con la loro CPU totale e propria
- ✓ View model DashboardSnapshot con le stringhe di visualizzazione precalcolate nel servizio
- ✓ Gestore di aggiornamento a batch che modifica i controlli una sola volta dentro uno scope della sonda, con riabilitazione in try/finally
- ✓ Traccia CPU Usage dopo la modifica dello scenario identico con il nuovo percorso critico
- ✓ Tabella prima/dopo di CPU totale, funzione principale e numero di chiamate, più il rischio residuo

Perché è fatto così

Il Modulo 2 si è chiuso con un nome: TicketFormatter.FormatRow, 68,9% di CPU propria, chiamato cinquantamila volte in un solo clic. È la forma di quasi ogni problema di CPU in una UI lato server. Nessuno ha scritto un metodo lento; qualcuno ha scritto un metodo ragionevole e poi ha fatto in modo che girasse una volta per cella, a ogni aggiornamento, per sempre. Il secondo colpevole nella dashboard di WisejPerfLab è l'immagine speculare: RebuildKpiPanel viene chiamato una volta sola, ma rilascia e ricrea un

pannello pieno di controlli che non avevano bisogno di cambiare. Uno è un problema di numero di chiamate e l'altro un problema di lavoro per chiamata, e questo modulo insegna a riconoscere quale dei due hai davanti e a correggerlo senza andare a tentativi.

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
costo del gestore di eventi	Un clic, un cambio di selezione o un tick di timer gira sul server e blocca l'evento successivo di quella sessione, quindi la quantità di logica innescata dal clic è la quantità di tempo in cui l'utente resta bloccato.
flame graph	Gli stessi dati dell'albero delle chiamate disegnati come barre di larghezza proporzionale, che rendono evidente a colpo d'occhio un singolo frame largo — una funzione che si mangia lo scenario.
CPU propria e numero di chiamate	Una CPU propria alta con poche chiamate significa che il corpo è costoso; un costo per chiamata basso con un numero di chiamate enorme significa che la correzione è chiamarla meno, non micro-ottimizzarla.
view model precalcolato	Un piccolo oggetto che porta stringhe e valori pronti per la visualizzazione, costruito una sola volta da un servizio, così che il livello di UI assegni valori invece di formattarli.
modifica della UI a batch	Impostare le proprietà che l'utente deve vedere in un'unica passata, con valori già preparati, invece di modificare i controlli dentro un ciclo che fa anche del lavoro.
ricostruzione dei controlli	Rilasciare e ricreare i controlli per aggiornarli; riesegue costruzione, layout e applicazione del tema, ed è quasi sempre più costoso che aggiornare i controlli esistenti.
cache di lookup immutabile	Dati di riferimento condivisi da tutta l'applicazione e mai modificati; sicuri da mettere in una cache globale, a differenza dello stato per utente che deve restare nella sessione.

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 3. Puoi aprire tu stesso [module-03.md](#) per vedere esattamente che cosa leggerà.



Usa ChatGPT

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.



Usa Claude

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 3 - "Percorsi critici della CPU, gestori di eventi e lavoro di UI lato server" - 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-03.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: Sistema il pulsante Refresh di `WisejPerflab`. Esegui il profiling dello scenario di aggiornamento con CPU Usage, apri il flame graph e Show Hot Path e individua i due colpevoli: un formatter che gira per ogni cella a ogni aggiornamento e una ricostruzione di controlli che ricrea ogni volta il pannello dei KPI. Sostituiscili con un view model `DashboardSnapshot` le cui stringhe di visualizzazione vengono calcolate una sola volta nel servizio, modifica le label e la sorgente dati del grafico una sola volta dentro uno scope della sonda invece che riga per riga, e tieni il pulsante disabilitato in un `try/finally` mentre il lavoro è in corso. Riesegui lo scenario identico e cattura il nuovo percorso critico.
3. Produci questi deliverable e dimmi dove si trova ciascuno nella soluzione:
 - Traccia CPU Usage prima della modifica che indica il formatter ripetuto e la ricostruzione dei controlli con la loro CPU totale e propria
 - View model `DashboardSnapshot` con le stringhe di visualizzazione precalcolate nel servizio
 - Gestore di aggiornamento a batch che modifica i controlli una sola volta dentro uno scope della sonda, con riabilitazione in `try/finally`
 - Traccia CPU Usage dopo la modifica dello scenario identico con il nuovo percorso critico
 - Tabella prima/dopo di CPU totale, funzione principale e numero di chiamate, più il rischio residuo
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:

- Entrambi i colpevoli erano lenti, ma per ragioni opposte. Spiega quale era un problema di numero di chiamate e quale un problema di lavoro per chiamata, e perché le prove del profiler per ciascuno appaiono diverse.
- Un collega propone di marcare il gestore dell'aggiornamento come `async` per renderlo più veloce. Che cosa cambierebbe davvero, che cosa no, e come lo mostrerebbe la traccia?
- Il tuo snapshot viene costruito una volta per aggiornamento. Quale rischio di dati vecchi

o di correttezza introduce, e che cosa dovresti cambiare se lo stesso snapshot fosse messo in cache tra più sessioni?

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:

- ✓ Entrambi i colpevoli erano lenti, ma per ragioni opposte. Spiega quale era un problema di numero di chiamate e quale un problema di lavoro per chiamata, e perché le prove del profiler per ciascuno appaiono diverse.
- ✓ Un collega propone di marcare il gestore dell'aggiornamento come async per renderlo più veloce. Che cosa cambierebbe davvero, che cosa no, e come lo mostrerebbe la traccia?
- ✓ Il tuo snapshot viene costruito una volta per aggiornamento. Quale rischio di dati vecchi o di correttezza introduce, e che cosa dovresti cambiare se lo stesso snapshot fosse messo in cache tra più sessioni?

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 3 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 3 su GitHub](#)

Oppure clona tutto con `git clone https://github.com/iceteagroup/LearnWisej-Samples` e apri `Performance and Profiling Course/Module 3`.



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.

- ✓ **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."*
- ✓ **Fallo sopravvivere a una brutta giornata.** *"Esamina questo progetto OpsMonitor cercando le modalità di guasto che non gestisce ancora — input sbagliato o mancante, una dipendenza lenta o non disponibile, lavoro che arriva dopo che l'utente ha cambiato pagina, e due azioni che si accavallano. Per ciascuna dimmi prima se è un rischio reale in Wisej.NET 4 e perché, e solo dopo scrivi la correzione. Dove decidi che non serve gestire nulla, dillo e spiega il ragionamento invece di aggiungere codice difensivo."*



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.