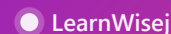


Esercizio di coding con l'IA

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 2 ti ha appena insegnato.

Che cosa stai chiedendo

Esegui correttamente il profiling del lento aggiornamento della dashboard di WisejPerfLab. Passa a Release, riscalda l'app una volta, poi raccogli con Alt+F2 una traccia CPU Usage di quell'unico scenario, restringi la timeline di riepilogo al solo clic e usa Show Hot Path per trovare il percorso principale sotto i frame di dispatch di Wisej.NET. Raccogli una seconda traccia Instrumentation dello stesso scenario per ottenere numero di chiamate e tempo reale, stabilisci dalle due tracce se il costo è CPU dentro il tuo codice o tempo passato in attesa, e salva entrambi i file .diagsession con un nome che riporta modulo, scenario, build, dataset e timestamp.

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

- ✓ Traccia CPU Usage in Release dello scenario di aggiornamento della dashboard, con la timeline ristretta al clic
- ✓ Screenshot del percorso critico che indica la principale funzione applicativa e la sua CPU propria rispetto a quella totale
- ✓ Traccia Instrumentation dello stesso scenario con numero di chiamate e tempo reale
- ✓ Nota della traccia che indica corso, modulo, scenario, build, hosting, strumento, browser e budget atteso
- ✓ Un paragrafo sulla causa radice sospetta che dice verso quale dei cinque contenitori puntano le prove

Perché è fatto così

Il Modulo 1 ha lasciato WisejPerfLab con un numero e nessuna spiegazione: l'aggiornamento della dashboard impiega 1.840 ms. Il passo successivo ovvio è farne il profiling, ed è proprio qui che la maggior parte delle indagini va storta. Una traccia raccolta sotto il debugger misura il debugger. Una traccia avviata prima che l'app sia riscaldata misura il JIT. Una traccia che copre due minuti di clic misura tutto e non

dimostra nulla. Una traccia raccolta con quattro strumenti attivi contemporaneamente misura l'overhead del profiler stesso. Questo modulo è la disciplina meccanica che rende una traccia una prova ammissibile: la build giusta, lo strumento giusto per il contenitore che sospetti, il processo giusto quando l'app non viene avviata da Visual Studio, uno scenario per raccolta e un report che sai leggere.

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
Performance Profiler (Alt+F2)	Il percorso di raccolta post-mortem: può avviare il progetto senza debugger, raccogliere un intero scenario e produrre un report salvato, ed è da lì che vengono le prove vere.
finestra Diagnostic Tools	La vista dal vivo, con il debugger agganciato; utile per un'occhiata veloce mentre esegui il codice passo passo, fuorviante come fonte di tempi perché il debugger cambia ciò che fa il runtime.
matrice di scelta degli strumenti	CPU Usage per il calcolo, Instrumentation per numero di chiamate e tempo reale, Object Allocation per il ricambio di oggetti, Memory Usage per la ritenzione, Database per le query, File I/O per i file,.NET Async per le catene di await,.NET Counters per lo stato di salute dal vivo.
.diagsession	Il file di traccia salvato; dargli un nome con modulo, scenario, build, dataset e timestamp è ciò che rende possibile un confronto prima/dopo settimane più tardi.
timeline di riepilogo	Il grafico di CPU o memoria sull'intera raccolta; restringere la selezione all'esatta azione dell'utente è il passo che elimina avvio e riscaldamento dall'albero delle chiamate.
percorso critico	La singola catena di chiamate più costosa nell'intervallo selezionato; Show Hot Path la espande così puoi trovare il tuo codice sotto i frame del framework.
CPU propria e CPU totale	La totale include i figli, la propria è il solo corpo della funzione: una CPU propria alta significa che il metodo in sé è costoso, una totale alta con una propria bassa significa che il costo sta in ciò che chiama.

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 2. Puoi aprire tu stesso [module-02.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



Usa Claude

Apri [claude.ai](#) con lo stesso prompt già caricato. Chi usa Claude Code può invece

Codex; qualsiasi piano che possa navigare il web recupererà la specifica.

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 2 - "Flusso di profiling in Visual Studio per Wisej.NET" - 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-02.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: Esegui correttamente il profiling del lento aggiornamento della dashboard di WisejPerfLab. Passa a Release, riscalda l'app una volta, poi raccogli con `Alt+F2` una traccia CPU Usage di quell'unico scenario, restringi la timeline di riepilogo al solo clic e usa Show Hot Path per trovare il percorso principale sotto i frame di dispatch di Wisej.NET. Raccogli una seconda traccia Instrumentation dello stesso scenario per ottenere numero di chiamate e tempo reale, stabilisci dalle due tracce se il costo è CPU dentro il tuo codice o tempo passato in attesa, e salva entrambi i file `.diagsession` con un nome che riporta modulo, scenario, build, dataset e timestamp.
3. Produci questi deliverable e dimmi dove si trova ciascuno nella soluzione:
 - Traccia CPU Usage in Release dello scenario di aggiornamento della dashboard, con la timeline ristretta al clic
 - Screenshot del percorso critico che indica la principale funzione applicativa e la sua CPU propria rispetto a quella totale
 - Traccia Instrumentation dello stesso scenario con numero di chiamate e tempo reale
 - Nota della traccia che indica corso, modulo, scenario, build, hosting, strumento, browser e budget atteso
 - Un paragrafo sulla causa radice sospetta che dice verso quale dei cinque contenitori puntano le prove
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:

- La tua traccia CPU Usage spiega solo 300 ms dei 1.840 ms misurati dalla sonda. Che cosa significa quella differenza, e quale strumento useresti per la raccolta successiva?
- Un metodo del framework sta in cima al percorso critico con una CPU totale molto alta e quasi nessuna CPU propria. Spiega che cosa significa e che cosa guarderesti invece di aprire un bug sul framework.
- Perché una seconda persona ha bisogno della nota della traccia per riprodurre il tuo risultato, quando `il.diagsession` contiene già l'intera traccia?

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:

- ✓ La tua traccia CPU Usage spiega solo 300 ms dei 1.840 ms misurati dalla sonda. Che cosa significa quella differenza, e quale strumento useresti per la raccolta successiva?
- ✓ Un metodo del framework sta in cima al percorso critico con una CPU totale molto alta e quasi nessuna CPU propria. Spiega che cosa significa e che cosa guarderesti invece di aprire un bug sul framework.
- ✓ Perché una seconda persona ha bisogno della nota della traccia per riprodurre il tuo risultato, quando il `diagsession` contiene già l'intera traccia?

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 2 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 2 su GitHub](#)

Oppure clona tutto con `git clone https://github.com/iceteagroup/LearnWisej-Samples` e apri `Performance and Profiling Course/Module 2`.



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-02.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.