

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.

● LearnWisej



Questo progetto non lo scriverai a mano. Lo dirigerai.

Quasi nessuno scrive più codice Wisej.NET da un file vuoto: un assistente scrive la prima bozza e lo sviluppatore decide se è giusta. Funziona però solo se conosci la materia, ed è quello che ti ha dato il resto di questo modulo. Qui affidi quelle conoscenze a ChatGPT o a Claude, ti fai restituire il progetto del modulo come soluzione Visual Studio funzionante, e lo rivedi come rivedresti la pull request di un collega.

Che cosa stai chiedendo

Crea la soluzione WisejPerfLab con una dashboard di supporto, una griglia dei ticket e un albero dei clienti alimentati da dati di seed, poi rendila misurabile. Aggiungi un servizio ScenarioProbe che racchiude uno scenario in uno scope con Stopwatch e scrive log PERF strutturati di inizio/fine con il nome dello scenario, l'azione dell'utente, i millisecondi trascorsi e il numero di righe, registralo nell'host builder e racchiudici tre scenari: aggiornamento della dashboard, ricerca dei ticket ed espansione dell'albero dei clienti. Esegui ogni scenario una volta di riscaldamento, poi registra la baseline e scrivi un budget di prestazioni che indica ambiente, configurazione di build, dimensione del dataset e una soglia di accettazione per ogni scenario.

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

- ✓ Soluzione WisejPerfLab con dashboard, griglia dei ticket e albero dei clienti su dati di seed
- ✓ Servizio ScenarioProbe che scrive log PERF strutturati di inizio/fine con millisecondi trascorsi e numero di righe
- ✓ Tre scenari strumentati: aggiornamento della dashboard, ricerca dei ticket ed espansione dell'albero dei clienti
- ✓ Nota di baseline che registra ambiente, configurazione di build, dimensione del dataset e procedura di riscaldamento
- ✓ Budget di prestazioni con una soglia di accettazione per scenario e lo strumento che dimostrerebbe ciascuna

Perché è fatto così

Un team di supporto segnala che WisejPerfLab "è diventato lento". Quella frase non contiene nessuna misura, nessuno scenario e nessun livello. La dashboard potrebbe essere in attesa di SQL, il gestore dell'aggiornamento potrebbe formattare cinquantamila celle, il browser potrebbe ridisegnare una griglia che non avrebbe mai dovuto essere resa per intero, oppure il server potrebbe semplicemente trattenere i form di duecento sessioni chiusi un'ora fa. Ognuno di questi casi ha una correzione diversa e uno

strumento di Visual Studio diverso, e lo sviluppatore che apre il profiler prima di decidere di quale si tratta raccoglierà una traccia che non può rispondere alla domanda. Questo modulo costruisce prima il modello mentale: che cosa succede davvero tra un clic nel browser e una schermata ridisegnata, quanto costa ciascuno di quei passi e come mettere per iscritto uno scenario e un budget, così che i sei moduli successivi abbiano qualcosa con cui confrontarsi.

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
UI lato server	I controlli, le loro proprietà e il loro stato vivono nel processo del server e il browser rende ciò che il server descrive, quindi ogni oggetto di UI che crei è memoria del server e ogni modifica alla UI è CPU del server più un payload.
round trip di un evento	Il percorso completo di un'azione dell'utente: evento nel browser, trasporto al server, il tuo gestore, accesso ai dati, modifiche ai controlli, l'aggiornamento serializzato di ritorno, il ridisegno nel browser — l'unità che l'utente percepisce davvero.
la sessione come unità di capacità	Una sessione possiede pagine, form, controlli, timer, selezioni, binding e dati in cache; alle domande sulla capacità si risponde per sessione, non per richiesta, perché quella memoria resta residente tra un clic e l'altro.
push via WebSocket	Il canale attraverso cui il server invia aggiornamenti che l'utente non ha chiesto; rende economico consegnare dati in tempo reale e rende costoso un task in background senza limitazione, perché ogni stato intermedio diventa un frame.
i cinque contenitori di costo	CPU nel tuo codice, blocchi e attese, allocazione e ritenzione, accesso ai dati e I/O, e payload degli aggiornamenti — la classificazione che decide quale strumento del profiler può anche solo vedere il tuo problema.
latenza e throughput	La latenza è quanto aspetta un singolo utente; il throughput è quante sessioni il server riesce a sostenere. Una correzione che elimina un'attesa aiuta la latenza; una correzione che elimina lavoro o memoria aiuta entrambi, e alcune correzioni della latenza costano throughput.
scenario e baseline	Una sequenza di azioni dell'utente con un nome e ripetibile, più l'ambiente, la configurazione di build, la dimensione del dataset e i numeri misurati che ha prodotto — senza di essa una traccia è rumore e un confronto successivo non ha significato.

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 1. Puoi aprire tu stesso [module-01.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 1 - "Il modello delle prestazioni di 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-01.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: Crea la soluzione WisejPerfLab con una dashboard di supporto, una griglia dei ticket e un albero dei clienti alimentati da dati di seed, poi rendila misurabile. Aggiungi un servizio ScenarioProbe che racchiude uno scenario in uno scope con Stopwatch e scrive log PERF strutturati di inizio/fine con il nome dello scenario, l'azione dell'utente, i millisecondi trascorsi e il numero di righe, registralo nell'host builder e racchiudici tre scenari: aggiornamento della dashboard, ricerca dei ticket ed espansione dell'albero dei clienti. Esegui ogni scenario una volta di riscaldamento, poi registra la baseline e scrivi un budget di prestazioni che indica ambiente, configurazione di build, dimensione del dataset e una soglia di accettazione per ogni scenario.
3. Produci questi deliverable e dimmi dove si trova ciascuno nella soluzione:
 - Soluzione WisejPerfLab con dashboard, griglia dei ticket e albero dei clienti su dati di seed
 - Servizio ScenarioProbe che scrive log PERF strutturati di inizio/fine con millisecondi trascorsi e numero di righe
 - Tre scenari strumentati: aggiornamento della dashboard, ricerca dei ticket ed espansione dell'albero dei clienti
 - Nota di baseline che registra ambiente, configurazione di build, dimensione del dataset e procedura di riscaldamento
 - Budget di prestazioni con una soglia di accettazione per scenario e lo strumento che dimostrerebbe ciascuna
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:

- L'aggiornamento della dashboard registra 1.900 ms ma la CPU del server si muove appena durante lo scenario. Quale dei cinque contenitori esclude questo dato, verso quale punta, e qual è la misura successiva più economica?
- La sonda misura il gestore. Indica due costi che l'utente subisce sullo stesso clic e che il tuo numero `elapsedMs` non include, e spiega come misureresti ciascuno.

– Perché la nota di baseline deve registrare la dimensione del dataset e la configurazione di build, quando la parte interessante sembrano essere il nome dello scenario e il tempo trascorso?

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:

- ✓ L'aggiornamento della dashboard registra 1.900 ms ma la CPU del server si muove appena durante lo scenario. Quale dei cinque contenitori esclude questo dato, verso quale punta, e qual è la misura successiva più economica?
- ✓ La sonda misura il gestore. Indica due costi che l'utente subisce sullo stesso clic e che il tuo numero elapsedMs non include, e spiega come misureresti ciascuno.
- ✓ Perché la nota di baseline deve registrare la dimensione del dataset e la configurazione di build, quando la parte interessante sembrano essere il nome dello scenario e il tempo trascorso?

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 1 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 1 su GitHub](#)

Oppure clona tutto con `git clone https://github.com/iceteagroup/LearnWisej-Samples` e apri `Performance and Profiling Course/Module 1`.



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.

- ✓ **Aggiungi il secondo caso.** *"Estendi questo progetto OpsMonitor con un secondo caso che segue lo stesso schema del primo — un'altra entità, schermata o controllo della stessa forma, collegato allo stesso modo. Estrai in una classe base o in un servizio ciò che è davvero condiviso, ma solo dove se lo merita; due cose simili non chiedono sempre un'astrazione. Mostrami il diff e giustifica ogni cosa che hai estratto."*
- ✓ **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-01.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."*



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.