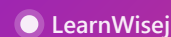


Esercizio di coding con l'IA

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.



Scrivere il codice è ormai la metà facile.

Ottenere codice funzionante per "Memoria, allocazioni e perdite di sessione" da un assistente moderno richiede un buon prompt. Giudicarlo, correggerlo ed estenderlo richiede tutto quello che questo modulo ha trattato. È su questa divisione che è costruito l'esercizio: tu metti la specifica e la revisione, l'assistente mette la digitazione.

Che cosa stai chiedendo

Affronta entrambe le metà del problema di memoria in WisejPerfLab. Esegui lo scenario di ricerca dei ticket sotto .NET Object Allocation e trova i modelli di riga e le stringhe allocati a ogni ricerca, poi riducili proiettando una volta sola invece di ricostruire gli oggetti a ogni ridisegno. Poi cattura uno snapshot di Memory Usage dopo il riscaldamento, apri e chiudi il form di dettaglio del ticket cinquanta volte, cattura un secondo snapshot e confrontali: i form trattenuti hanno come radice una sottoscrizione statica a GlobalTicketBus.TicketChanged e un timer di aggiornamento mai fermato. Annulla la sottoscrizione e ferma entrambi in un override di Dispose, azzera la sorgente dati del BindingSource, poi riesegui lo scenario di apertura/chiusura e mostra che l'heap trattenuto torna vicino allo snapshot A.

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

- ✓ Traccia delle allocazioni che indica il tipo che alloca di più e il percorso di chiamata per lo scenario di ricerca, prima e dopo
- ✓ Confronto tra gli snapshot A/B che mostra il numero di istanze trattenute del form di dettaglio prima della correzione
- ✓ Screenshot del percorso fino alla radice che identifica la sottoscrizione all'evento statico e il timer ancora attivo
- ✓ Override di Dispose che annulla la sottoscrizione all'evento statico, ferma e rilascia il timer e azzera il binding
- ✓ Budget di memoria per sessione e una checklist di rilascio, con lo snapshot finale che dimostra che il percorso di ritenzione è sparito

Perché è fatto così

La dashboard di WisejPerfLab ora rispetta il suo budget di CPU, eppure l'app cede dopo quattro ore in produzione. I problemi di memoria si comportano diversamente da quelli di CPU: sono invisibili in una demo di due minuti, non si presentano mai come un singolo clic lento, e sono il motivo per cui un server

che la mattina regge comodamente ottanta sessioni a metà pomeriggio rifiuta le connessioni. Nel lab ce ne sono due, e non sono lo stesso problema. La ricerca dei ticket alloca molto più del necessario, il che costa tempo di GC e CPU mentre è in esecuzione. Il form di dettaglio del ticket non viene mai davvero rilasciato dopo che l'utente lo chiude, il che costa memoria per sempre. Il primo si trova con una traccia delle allocazioni, il secondo solo con due snapshot e un percorso fino alla radice.

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
pressione sulle allocazioni	Quanta memoria crea uno scenario mentre è in esecuzione; una pressione alta aumenta la frequenza del GC e ruba CPU anche quando non c'è nessuna perdita.
memoria trattenuta	Ciò che è ancora raggiungibile quando lo scenario dovrebbe essere finito; è questa che si moltiplica per il numero di sessioni e decide la capacità.
.NET Object Allocation	Lo strumento che registra ogni allocazione gestita durante uno scenario e la raggruppa per tipo e per percorso di chiamata che alloca.
snapshot di Memory Usage	Catture dell'heap prese in momenti scelti; confrontare lo snapshot A e lo snapshot B mostra quali tipi hanno guadagnato istanze mai rilasciate.
percorso fino alla radice	La catena di riferimenti che tiene vivo un oggetto; è la vera prova di una perdita, e l'unico modo per dimostrare che la perdita è sparita.
radice di ritenzione	Ciò che sta in cima a quella catena — un campo statico, un evento statico, un timer in esecuzione, un task in background o una cache a lunga vita.
budget di memoria per sessione	Un numero, in megabyte trattenuti per sessione inattiva, che trasforma "l'app usa molta memoria" in un superato o non superato e più avanti in una soglia di health check.

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 4. Puoi aprire tu stesso [module-04.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 4 - "Memoria, allocazioni e perdite di sessione" - 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-04.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: Affronta entrambe le metà del problema di memoria in `WisejPerfLab`. Esegui lo scenario di ricerca dei ticket sotto `.NET Object Allocation` e trova i modelli di riga e le stringhe allocati a ogni ricerca, poi riducili proiettando una volta sola invece di ricostruire gli oggetti a ogni ridisegno. Poi cattura uno snapshot di Memory Usage dopo il riscaldamento, apri e chiudi il form di dettaglio del ticket cinquanta volte, cattura un secondo snapshot e confrontali: i form trattenuti hanno come radice una sottoscrizione statica a `GlobalTicketBus.TicketChanged` e un timer di aggiornamento mai fermato. Annulla la sottoscrizione e ferma entrambi in un override di `Dispose`, azzera la sorgente dati del `BindingSource`, poi riesegui lo scenario di apertura/chiusura e mostra che l'heap trattenuto torna vicino allo snapshot A.
3. Produci questi deliverable e dimmi dove si trova ciascuno nella soluzione:
 - Traccia delle allocazioni che indica il tipo che alloca di più e il percorso di chiamata per lo scenario di ricerca, prima e dopo
 - Confronto tra gli snapshot A/B che mostra il numero di istanze trattenute del form di dettaglio prima della correzione
 - Screenshot del percorso fino alla radice che identifica la sottoscrizione all'evento statico e il timer ancora attivo
 - Override di `Dispose` che annulla la sottoscrizione all'evento statico, ferma e rilascia il timer e azzera il binding
 - Budget di memoria per sessione e una checklist di rilascio, con lo snapshot finale che dimostra che il percorso di ritenzione è sparito
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 delle allocazioni è migliorata in modo drastico ma il confronto tra snapshot è invariato. Che tipo di problema hai corretto, quale no, e come si accorgerebbe della differenza un utente?
- Perché il lab apre e chiude il form cinquanta volte invece di una, e che cosa concluderesti da una singola istanza che compare nello snapshot B?

- La callback del timer è lunga una riga e imposta solo una label. Spiega con precisione come tiene in vita un intero form, e che cosa risolve e che cosa non risolve un controllo IsDisposed.

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 delle allocazioni è migliorata in modo drastico ma il confronto tra snapshot è invariato. Che tipo di problema hai corretto, quale no, e come si accorgerebbe della differenza un utente?
- ✓ Perché il lab apre e chiude il form cinquanta volte invece di una, e che cosa concluderesti da una singola istanza che compare nello snapshot B?
- ✓ La callback del timer è lunga una riga e imposta solo una label. Spiega con precisione come tiene in vita un intero form, e che cosa risolve e che cosa non risolve un controllo IsDisposed.

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 4 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 4 su GitHub](#)

Oppure clona tutto con `git clone https://github.com/iceteagroup/LearnWisej-Samples` e apri `Performance and Profiling Course/Module 4`.



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.

- ✓ **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."*
- ✓ **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."*



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.