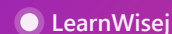


Esercizio di coding con l'IA

Costruisci il progetto TicketOps 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 TicketOps; 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

Rifattorizza l'applicazione per usare servizi iniettati per ticket, utenti, permessi, notifiche e log di audit. Aggiungi un profilo di registrazione dei servizi fittizi e uno di produzione.

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

- ✓ Deliverable: metodo di registrazione dei servizi
- ✓ cinque interfacce con implementazioni fittizie
- ✓ MainPage/Form con iniezione
- ✓ presenter o servizio di workflow per una schermata
- ✓ tabella delle durate dei servizi

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
Dependency injection (DI)	una classe dichiara di che cosa ha bisogno tramite interfaccia e riceve un'implementazione dall'esterno, invece di crearne una da sé.
Container di servizi / IServiceProvider	il registro che associa un contratto alla sua implementazione e al suo lifetime, e risolve le istanze su richiesta.
Lifetime del servizio	quanto a lungo vive un'istanza risolta: applicazione/singleton (condivisa), con scope di sessione (per utente) o transient (per risoluzione).
Servizio con scope di sessione	un'istanza per sessione utente; la collocazione corretta del contesto per utente (Modulo 2).
Constructor injection	dipendenze fornite come parametri del costruttore; si usa per le classi create da te o dal container.
Property injection	una dipendenza assegnata a una proprietà contrassegnata dopo la costruzione; si usa per Form/Page creati dal framework, che non costruisci tu.
Presenter / application service	una classe semplice, senza browser, che contiene la logica decisionale a cui un Form delega, rendendo quella logica testabile con unit test.
Fake / test double	un'implementazione sostitutiva di un'interfaccia usata in un test al posto del servizio reale.

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 8. Puoi aprire tu stesso [module-08.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 TicketOps 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 8 - "Servizi, dependency injection e pattern di UI testabili" - del corso LearnWisej "Architettura di produzione in profondità", 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/intermediate/specs/module-08.md>

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

1. Progetto "TicketOps", 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: Rifattorizza l'applicazione per usare servizi iniettati per ticket, utenti, permessi, notifiche e log di audit. Aggiungi un profilo di registrazione dei servizi fittizi e uno di produzione.
3. Produci questi deliverable e dimmi dove si trova ciascuno nella soluzione:
 - Deliverable: metodo di registrazione dei servizi
 - cinque interfacce con implementazioni fittizie
 - MainPage/Form con iniezione
 - presenter o servizio di workflow per una schermata
 - tabella delle durate dei servizi
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.

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.

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 8 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 8 su GitHub](#)

Oppure clona tutto con `git clone https://github.com/iceteagroup/LearnWisej-Samples` e apri `Production Architecture Deep Dive/Module 8`.



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.

- ✓ **Fallo sopravvivere a una brutta giornata.** *"Esamina questo progetto TicketOps 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 TicketOps 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/intermediate/specs/module-08.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."*



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.