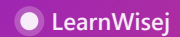


Esercizio di coding con l'IA

Costruisci il progetto GlobalDesk 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.



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 7 ti ha appena insegnato.

Che cosa stai chiedendo

Porta a termine il capstone di GlobalDesk. Racchiudi ogni ricerca di risorse in un LocalizationService con metodi Text, Date, Number e Currency, dagli una politica per le chiavi mancanti che mostra la chiave tra parentesi quadre in sviluppo e la registra nei log in produzione, e fai di quel servizio l'unico punto in cui l'interfaccia tocca ResourceManager. Tieni il dominio fuori da tutto questo: il modello del ticket espone un valore TicketStatus e l'interfaccia lo mappa in un'etichetta localizzata. Poi esegui la matrice di test di localizzazione sull'applicazione finita, coprendo la cultura neutra, il tedesco, una cultura lingua-regione che ricade sul fallback, una chiave mancante di proposito, date e importi formattati, un cambio a runtime, un passaggio con stringhe lunghe e la cultura RTL, usando un set di risorse pseudo-localizzato per portare a galla tutto ciò che è ancora scritto nel codice. Verifica che le cartelle di cultura sopravvivano a una pubblicazione e consegna la checklist di QA come deliverable del capstone.

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

- ✓ LocalizationService con metodi Text, Date, Number e Currency e una politica documentata per le chiavi mancanti
- ✓ Codice di dominio che restituisce un valore TicketStatus che l'interfaccia mappa in un'etichetta localizzata
- ✓ Set di risorse pseudo-localizzato usato per trovare ogni stringa ancora scritta nel codice
- ✓ Matrice di test di localizzazione completata, che copre fallback, formattazione, cambio a runtime e RTL
- ✓ Output pubblicato verificato nelle sue cartelle di cultura, più la checklist di QA della localizzazione di GlobalDesk

Perché è fatto così

Dopo sei moduli GlobalDesk parla due lingue, ma la macchina è sparsa: le risorse del designer sull'editor dei clienti, un helper sopra i file.resx condivisi, un selettore della lingua, un gestore di CultureChanged e

una modalità RTL. Niente dichiara ancora che cosa succede quando una chiave non ha traduzione, e nessuno ha dimostrato che la build tedesca stia ancora dentro una finestra stretta. Un sistema di localizzazione di produzione è prevedibile: sa da dove viene ogni stringa, quale cultura è attiva in questa sessione, che aspetto ha una traduzione mancante e che cosa fa un cambio di lingua alle schermate aperte. Questo modulo trasforma quelle quattro risposte in un piccolo servizio, una matrice di test scritta e il capstone finito.

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
LocalizationService	Una piccola classe che centralizza le scelte della tua applicazione intorno alla localizzazione.NET invece di sostituirla: un solo nome base delle risorse, una sola politica per le chiavi mancanti, un solo insieme di metodi di formattazione.
Politica per le chiavi mancanti	Il comportamento scelto e testato quando GetString non restituisce nulla: un marcatore [Key] visibile mentre sviluppi, una chiave registrata nei log più il testo neutro di fallback in produzione.
Testo di prodotto, di sistema e diagnostico	Tre famiglie con tre proprietari: le tue stringhe.resx, le etichette di Wisej.NET attraverso Resources-[LANG].resx, e il testo dei log o delle eccezioni che resta in un'unica lingua per gli sviluppatori.
Dominio libero dalla cultura	Il modello espone un valore come TicketStatus.Closed e l'interfaccia lo mappa in un'etichetta localizzata, così la logica di business e i suoi test non dipendono mai da una lingua.
Matrice di test di localizzazione	Un elenco scritto di esecuzioni — cultura neutra, cultura tradotta, regione che ricade sul fallback, chiave mancante, valori formattati, cambio a runtime, stringhe lunghe, RTL, etichette di sistema — così la copertura è una checklist e non un ricordo.
Pseudo-localizzazione	Un set di risorse di test i cui valori sono visibilmente alterati e allungati, come [!!! Save customer record!!!], costruito per portare a galla i troncamenti e le stringhe scritte direttamente nel codice, non per tradurre qualcosa.
Risorse satellite	.NET compila le risorse specifiche per cultura in assembly satellite, quindi le cartelle di cultura prodotte dalla build devono viaggiare con l'applicazione quando la pubblichi o la impacchetti.

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 7. Puoi aprire tu stesso [module-07.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 GlobalDesk 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 7 - "Architettura di localizzazione in produzione, test e capstone" - del corso LearnWisej "Localizzazione in Wisej.NET", 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/localization-in-wisej-net/specs/module-07.md>

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

1. Progetto "GlobalDesk", 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: Porta a termine il capstone di GlobalDesk. Racchiudi ogni ricerca di risorse in un `LocalizationService` con metodi `Text`, `Date`, `Number` e `Currency`, dagli una politica per le chiavi mancanti che mostra la chiave tra parentesi quadre in sviluppo e la registra nei log in produzione, e fai di quel servizio l'unico punto in cui l'interfaccia tocca `ResourceManager`. Tieni il dominio fuori da tutto questo: il modello del ticket espone un valore `TicketStatus` e l'interfaccia lo mappa in un'etichetta localizzata. Poi esegui la matrice di test di localizzazione sull'applicazione finita, coprendo la cultura neutra, il tedesco, una cultura lingua-regione che ricade sul fallback, una chiave mancante di proposito, date e importi formattati, un cambio a runtime, un passaggio con stringhe lunghe e la cultura RTL, usando un set di risorse pseudo-localizzato per portare a galla tutto ciò che è ancora scritto nel codice. Verifica che le cartelle di cultura sopravvivano a una pubblicazione e consegna la checklist di QA come deliverable del capstone.
3. Produci questi deliverable e dimmi dove si trova ciascuno nella soluzione:
 - `LocalizationService` con metodi `Text`, `Date`, `Number` e `Currency` e una politica documentata per le chiavi mancanti
 - Codice di dominio che restituisce un valore `TicketStatus` che l'interfaccia mappa in un'etichetta localizzata
 - Set di risorse pseudo-localizzato usato per trovare ogni stringa ancora scritta nel codice
 - Matrice di test di localizzazione completata, che copre fallback, formattazione, cambio a runtime e RTL
 - Output pubblicato verificato nelle sue cartelle di cultura, più la checklist di QA della localizzazione di GlobalDesk
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:

- Un revisore apre GlobalDesk in francese, una cultura che non hai mai tradotto, e vede etichette inglesi con date francesi formattate correttamente. Spiega quale meccanismo ha

prodotto ciascuna metà di quella schermata e di' se qualcosa è rotto.

– La tua esecuzione pseudo-localizzata mostra una label che riporta ancora "Save" in inglese puro. Elenca i tre punti da cui quel testo potrebbe arrivare e descrivi come li distingueresti senza tirare a indovinare.

– Un collega propone di tenere in cache il CultureInfo risolto in un campo statico di LocalizationService "perché cercarlo a ogni chiamata è uno spreco". Descrivi che cosa sperimenterebbe un secondo utente autenticato e che cosa suggeriresti al suo posto.

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:

- ✓ Un revisore apre GlobalDesk in francese, una cultura che non hai mai tradotto, e vede etichette inglesi con date francesi formattate correttamente. Spiega quale meccanismo ha prodotto ciascuna metà di quella schermata e di' se qualcosa è rotto.
- ✓ La tua esecuzione pseudo-localizzata mostra una label che riporta ancora "Save" in inglese puro. Elenca i tre punti da cui quel testo potrebbe arrivare e descrivi come li distingueresti senza tirare a indovinare.
- ✓ Un collega propone di tenere in cache il CultureInfo risolto in un campo statico di LocalizationService "perché cercarlo a ogni chiamata è uno spreco". Descrivi che cosa sperimenterebbe un secondo utente autenticato e che cosa suggeriresti al suo posto.

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.



Per questo corso non c'è ancora un esempio di riferimento — il tuo è il primo

Il [repository degli esempi LearnWisej](#) contiene build già fatte per quasi tutti i corsi, ma non ancora per questo. Significa che non c'è niente con cui confrontarsi e nessuna scorciatoia: la specifica e la tua revisione sono tutta la rete di sicurezza. Tieni quello che costruisci — e se viene bene, ci farebbe piacere vederlo.

Tre prompt per andare oltre

Punta il tuo assistente sul progetto che hai appena costruito 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/localization-in-wisej-net/specs/module-07.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 GlobalDesk 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 GlobalDesk 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.