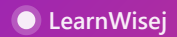


Esercizio di coding con l'IA

Costruisci il progetto ValidationClinic 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

Trasforma il form di contatto di ValidationClinic in un form con validazione programmatica. Crea un solo ErrorProvider per il form con ContainerControl impostato, NeverBlink e l'icona allineata a destra, scrivi un record FieldValidation più gli helper ValidateName, ValidateEmail e ValidatePhone che restituiscono un messaggio invece di toccare la UI, e un metodo Apply che chiama SetError per ogni risultato e compila una validationSummaryLabel visibile solo finché esistono errori. Usa InvalidMessage sul campo del codice cliente per confrontare i due stili di feedback, e assicurati che correggere un campo rimuova solo l'errore di quel campo.

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

- ✓ ErrorProvider a livello di form con ContainerControl, BlinkStyle e allineamento dell'icona configurati
- ✓ Record FieldValidation e metodi helper ValidateName, ValidateEmail e ValidatePhone
- ✓ Metodo Apply che imposta o rimuove l'errore di ogni controllo e restituisce se tutti i campi sono validi
- ✓ validationSummaryLabel che elenca ogni problema attuale ed è nascosta quando non ce ne sono
- ✓ Un campo che usa InvalidMessage invece di ErrorProvider, con una nota che confronta i due approcci

Perché è fatto così

Il Modulo 1 ha dato al form di accettazione clienti di ValidationClinic due gestori Validating, e per due campi funziona. Un vero form di contatto ha un nome, un'email, un numero di telefono, un codice cliente e altro ancora, e se ogni campo riceve un proprio gestore scritto a mano con un proprio modo di mostrare i messaggi, il form diventa presto incoerente: un campo mostra un'icona, un altro non mostra nulla, un terzo

lascia a video un vecchio errore anche dopo che l'utente l'ha corretto. Questo modulo sostituisce questa deriva con un unico schema: piccoli metodi helper decidono che cosa non va, un solo metodo mostra il risultato e un riepilogo visibile dice a ogni utente che cosa correggere senza costringerlo a passare il mouse su qualcosa.

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
ErrorProvider	Un componente extender che disegna un'icona di errore accanto a qualsiasi controllo a cui assegni un messaggio e mostra quel messaggio come tooltip; una sola istanza serve l'intero form.
ContainerControl	Il form o lo user control a cui appartiene il provider; impostarlo su this indica al provider su quale superficie disegnare le sue icone.
SetError(control, message)	Mostra message accanto a control; chiamarlo con una stringa vuota o con null rimuove l'errore solo per quel controllo.
GetError, HasErrors, Clear()	Leggono il messaggio attuale di un controllo, chiedono se sul form resta qualche errore, oppure rimuovono tutti gli errori in un colpo solo quando l'intero form viene reimpostato.
InvalidMessage	Una proprietà dei controlli editor che supportano l'interfaccia IValidation; è il controllo stesso a mostrare il messaggio come tooltip di invalidità, senza alcun provider separato.
Record FieldValidation	Un minuscolo tipo di risultato che contiene il controllo, il nome del campo e un messaggio, dove un messaggio vuoto significa valido; permette alle regole di restituire risposte invece di modificare lo schermo.
validationSummaryLabel	Una semplice Label che elenca in testo ogni problema attuale, visibile solo finché esistono errori, così che il feedback non dipenda mai dal vedere un'icona o dal passarci sopra il mouse.

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 ValidationClinic 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 - "Validazione programmatica e feedback con ErrorProvider" - del corso LearnWisej "Validazione 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/validation-in-wisej-net/specs/module-02.md>

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

1. Progetto "ValidationClinic", 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: Trasforma il form di contatto di ValidationClinic in un form con validazione programmatica. Crea un solo ErrorProvider per il form con ContainerControl impostato, NeverBlink e l'icona allineata a destra, scrivi un record FieldValidation più gli helper ValidateName, ValidateEmail e ValidatePhone che restituiscono un messaggio invece di toccare la UI, e un metodo Apply che chiama SetError per ogni risultato e compila una validationSummaryLabel visibile solo finché esistono errori. Usa InvalidMessage sul campo del codice cliente per confrontare i due stili di feedback, e assicurati che correggere un campo rimuova solo l'errore di quel campo.
3. Produci questi deliverable e dimmi dove si trova ciascuno nella soluzione:
 - ErrorProvider a livello di form con ContainerControl, BlinkStyle e allineamento dell'icona configurati
 - Record FieldValidation e metodi helper ValidateName, ValidateEmail e ValidatePhone
 - Metodo Apply che imposta o rimuove l'errore di ogni controllo e restituisce se tutti i campi sono validi
 - validationSummaryLabel che elenca ogni problema attuale ed è nascosta quando non ce ne sono
 - Un campo che usa InvalidMessage invece di ErrorProvider, con una nota che confronta i due approcci
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:

- Il gestore txtName_Validating di un collega chiama errorProvider.Clear() non appena il nome diventa valido. Che cosa va storto per un utente che ha anche digitato un'email non valida, e che cosa dovrebbe chiamare invece il gestore?
- Il campo del codice cliente mostra il suo messaggio tramite InvalidMessage. Che cosa dovrebbe sapere la label di riepilogo su quel campo per elencarlo anch'esso, e che cosa ti dice questo su quando un provider condiviso è la scelta migliore?
- Un utente digita un'email ben formata che appartiene già a un altro contatto. È feedback sul campo o feedback sulle regole di business, quando dovrebbe essere eseguita la verifica e

dove dovrebbe comparire il messaggio?

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:

- ✓ Il gestore txtName_Validating di un collega chiama errorProvider.Clear() non appena il nome diventa valido. Che cosa va storto per un utente che ha anche digitato un'email non valida, e che cosa dovrebbe chiamare invece il gestore?
- ✓ Il campo del codice cliente mostra il suo messaggio tramite InvalidMessage. Che cosa dovrebbe sapere la label di riepilogo su quel campo per elencarlo anch'esso, e che cosa ti dice questo su quando un provider condiviso è la scelta migliore?
- ✓ Un utente digita un'email ben formata che appartiene già a un altro contatto. È feedback sul campo o feedback sulle regole di business, quando dovrebbe essere eseguita la verifica e dove dovrebbe comparire il messaggio?

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 `Validation in Wisej.NET 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.

- ✓ **Fallo sopravvivere a una brutta giornata.** *"Esamina questo progetto ValidationClinic 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 ValidationClinic 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/validation-in-wisej-net/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."*



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.