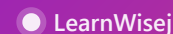


Esercizio di coding con l'IA

Costruisci il Sensor Monitor di questo modulo con ChatGPT o Claude partendo da un prompt già pronto che indirizza il modello alla guida del modulo, poi rivedi, esegui ed estendi quello che ti restituisce — confrontandolo anche con la nostra build di riferimento su GitHub.



Questo esempio non lo scriverai a mano. Lo dirigerai.

Quasi nessuno scrive più codice di integrazione partendo da un file vuoto: un assistente scrive la prima bozza e lo sviluppatore decide se è giusta. Funziona però solo se conosci il vocabolario, il contratto e le regole del framework, che è esattamente ciò che ti ha insegnato il resto di questo modulo. In questo esercizio affidi quelle conoscenze a ChatGPT o a Claude, ti fai restituire l'esempio del Modulo 1 come soluzione Visual Studio funzionante, e poi lo rivedi come rivedresti la pull request di un collega.

Che cosa ne ricavi

L'esempio del Modulo 1 è un **Sensor Monitor**: un `TemperatureGauge : Widget` che ospita una libreria di gauge di terze parti dentro un contenitore widget di Wisej.NET, con `Value` come proprietà validata sul server e `ThresholdExceeded` come normale evento .NET. Accanto al gauge c'è una **traccia Server ⇌ Client** in tempo reale, così ogni messaggio che attraversa la rete si può confrontare con il contratto che hai scritto.

Costruirlo con un prompt richiede un quarto d'ora. Il valore non sta nella digitazione che risparmi, ma nelle tre cose che devi saper fare dopo:

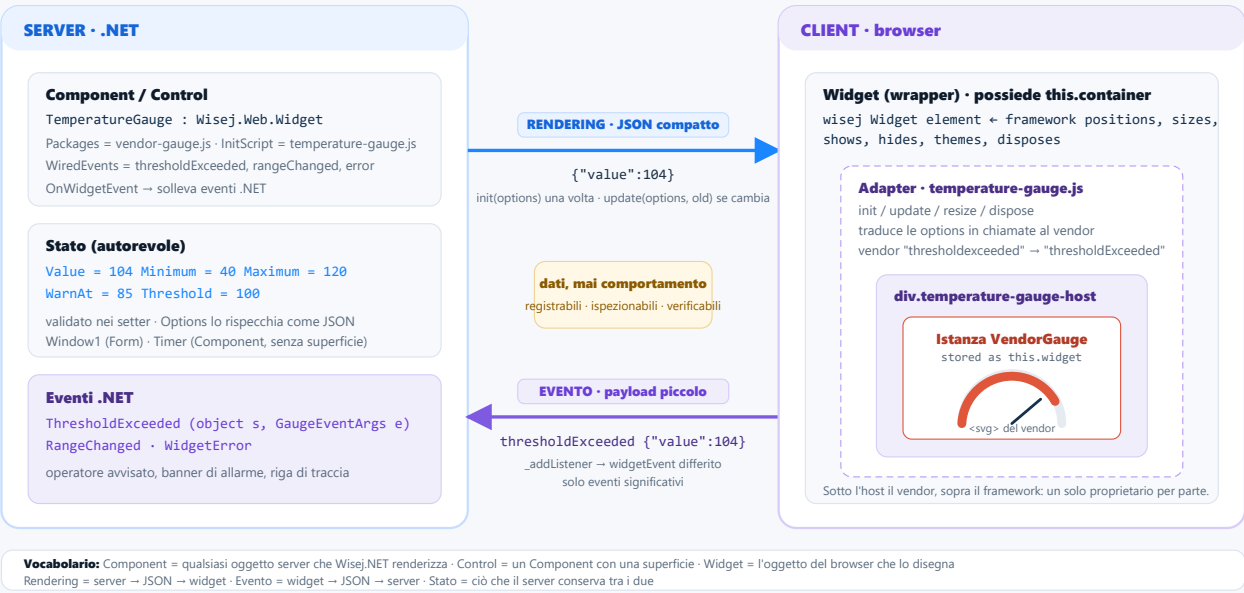
- ✓ **Istruire** l'assistente con il contratto e le regole del framework, così la prima bozza usa già `Wisej.Web.Timer` e ospita la libreria del fornitore dentro un elemento figlio di `this.container`.
- ✓ **Verificare** ciò che ti torna indietro. Codice che compila non è ancora codice che regge un vero caricamento di pagina.
- ✓ **Estendere** codice che non hai scritto tu, che è come sarà d'ora in poi il resto della tua carriera.

L'architettura che stai chiedendo

Prima di scrivere il prompt, devi saperlo dire tutto d'un fiato: *il server possiede lo stato dell'applicazione, il browser possiede il DOM e l'istanza del fornitore, e tra i due passano solo dati* — JSON compatto in uscita durante il rendering, payload piccoli in ritorno con gli eventi. Il valore del gauge, i suoi limiti e la sua soglia sono stato del server perché devono essere verificabili. L'animazione della lancetta no. Quando l'assistente ti consegna qualcosa che tiene il valore autorevole in JavaScript, sai già che è sbagliato.

IntegrationLab · Architettura dell'integrazione (Modulo 1)

Il server possiede lo stato. Il browser possiede il DOM e l'istanza del vendor. In rete passano solo dati.



Termine della lezione	Che cosa deve produrre l'assistente
Component / Control	TemperatureGauge sul server, proprietario di Value , Minimum , Maximum , WarnAt , Threshold . Il Wisej.Web.Timer che invia le letture in streaming è un Component senza superficie.
Widget	wwwroot/temperature-gauge.js — l'adapter del browser che crea l'istanza del fornitore in un elemento figlio di this.container e la conserva in this.widget .
Package	wwwroot/vendor-gauge.js , la libreria di terze parti fittizia, caricata una sola volta per pagina.
Rendering	Options → JSON compatto → init(options) al primo rendering, update(options, old) per i campi di primo livello modificati.
Evento	_addListener / _getEventData → OnWidgetEvent → ThresholdExceeded , RangeChanged , WidgetError .

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. Leggi il prompt prima di inviarlo: è la parte di questo esercizio che vale la pena imparare.



Usa ChatGPT



Usa Claude

Apri chatgpt.com con il prompt del Modulo 1 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 [IntegrationLab.slnx](#) in Visual Studio, premi F5 e il Sensor Monitor si apre su [localhost](#). Ti servono l'SDK .NET e il pacchetto NuGet [Wisej-4](#); non si scarica nient'altro, perché il prompt fa scrivere all'assistente anche la libreria di gauge del "vendor".

Sto seguendo il Modulo 1 - "Architettura e vocabolario dell'integrazione" - del "Corso di integrazione applicativa" di LearnWisej, e voglio costruire con te l'esempio 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/app-integration/specs/module-01.md>

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

1. Progetto "IntegrationLab", C#, net8.0 o successivo (la build di riferimento usa net10.0), pacchetto NuGet Wisej-4 4.1.x, host Kestrel che chiama app.UseWisej().
2. Un TemperatureGauge : Wisej.Web.Widget che ospita una libreria di gauge di terze parti. Scrivi tu stesso la libreria "vendor" come wwwroot/vendor-gauge.js e registrala come Package, così non serve scaricare nulla.
3. Lo stato appartiene al server: Value, Minimum, Maximum, WarnAt, Threshold. Valida nei setter delle proprietà e rispecchia ciascuna in Options come campo di primo livello, così Wisej.NET renderizza solo ciò che è cambiato.
4. Un adapter client wwwroot/temperature-gauge.js, incorporato nell'assembly e assegnato a InitScript. Deve creare l'istanza del fornitore dentro un elemento FIGLIO di this.container, conservarla in this.widget e collegare gli eventi tramite _addListener / _getEventData - non chiamare mai fireEvent in modo sincrono dall'interno di update(), e non definire mai un membro che nasconda un metodo qooxdoo come getWidth.
5. WiredEvents = thresholdExceeded, rangeChanged, error, esposti sul server come normali eventi .NET con EventArgs tipizzati che trasportano solo dati.
6. Una Window1 "Sensor Monitor" con il gauge a sinistra e una traccia Server-to-Client in tempo reale a destra, più pulsanti che esercitano cinque percorsi: una lettura valida, una serie in streaming guidata da un Wisej.Web.Timer, un valore fuori intervallo che viene rifiutato, un payload volutamente corrotto che l'adapter deve intercettare e segnalare come un unico evento error, e una risincronizzazione che renderizza di nuovo lo stato autorevole del server.
7. Una cartella docs/ con ClientServerContract.md, WidgetTriage.md, IntegrationDecisionRecord.md e un diagramma dell'architettura.

Scrivi docs/ClientServerContract.md prima di scrivere qualsiasi codice, e mostramelo prima di andare avanti.

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

Gli errori che qui fanno perdere tempo sono quelli che il compilatore non può intercettare: codice che compila senza problemi e poi si comporta male a un vero caricamento di pagina. Questi due ci sono costati una sessione di debug quando abbiamo costruito l'esempio di riferimento, e sono le prime cose da controllare nel tuo:

- ✓ L'adapter chiamava `fireWidgetEvent` in modo sincrono dall'interno di una callback del fornitore. Gli eventi sollevati mentre si sta applicando un aggiornamento dal server vengono scartati: il collegamento va fatto con `_addListener` / `_getEventData`.
- ✓ L'adapter definiva `this.getWidth`, nascondendo un metodo qooxdoo. Il widget veniva renderizzato largo zero pixel finché l'helper non è stato rinominato.

Poi esegui i cinque percorsi chiesti dal prompt — lettura valida, serie in streaming, valore rifiutato, payload corrotto, risincronizzazione — e confronta ogni riga della traccia con `docs/ClientServerContract.md`. Se in rete passa un messaggio che il contratto non descrive, è sbagliato il contratto o il codice, e sta a te decidere quale dei due.



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, con i deliverable in `docs/`, la classe lato server, l'adapter e la finestra della traccia. 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 `Application Integration Course/Module 1/IntegrationLab.slnx`.



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. Il gauge del fornitore è fittizio, non ci sono unit test, il supporto al designer è minimo e nulla è irrobustito per la produzione. 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 portare l'esempio più avanti

Punta il tuo assistente sulla cartella clonata (o incollagli i file che servono) e prova questi. Ognuno è un esercizio di revisione ed estensione, non una riscrittura.

- ✓ **Sostituisci la libreria fittizia con una vera.** *"In questo progetto `IntegrationLab`, sostituisci `wwwroot/vendor-gauge.js` con un vero gauge di terze parti — per esempio un radial bar di `ApexCharts`, o un gauge su canvas caricato come `Package`. Lascia invariato `docs/ClientServerContract.md`: il contratto del server, i campi di `Options` e i tre eventi collegati devono restare esattamente come sono, e può cambiare solo l'adapter. Mostrami il diff e dimmi che cosa la libreria vera ti ha costretto a gestire che quella fittizia non richiedeva."*
- ✓ **Aggiungi un secondo widget riusando lo schema.** *"Aggiungi a questo progetto un `HumidityGauge : Widget` che segua la stessa architettura di `TemperatureGauge`: stato validato sul server e rispecchiato in `Options`, un adapter che ospita il fornitore in un elemento figlio, eventi collegati tramite `_addListener` / `_getEventData`. Metti entrambi i gauge nella finestra `Sensor Monitor` con un unico log di traccia condiviso, ed estrai in una classe base ciò che è davvero comune — ma solo se se lo merita. Aggiorna il documento del contratto perché copra entrambi."*
- ✓ **Fallo sopravvivere a una brutta giornata.** *"Esamina questo progetto cercando le modalità di guasto che non gestisce ancora: lo script del fornitore che non si carica, un evento che arriva dopo che il widget è stato distrutto, due modifiche rapide di `Value` in competizione con il primo rendering, e un browser senza `ResizeObserver`. Per ciascuna dimmi prima se è un rischio reale in `Wisej.NET 4` e perché, e solo dopo scrivi la correzione. Aggiungi una voce a `docs/IntegrationDecisionRecord.md` per tutto ciò che hai deciso di lasciare com'è."*



Portalo dentro il lab

Il lab che chiude questo modulo chiede un diagramma dell'architettura, una tabella di triage dei widget e un integration decision record per tre widget candidati. Qualunque cosa costruisci qui — il tuo esempio o il nostro — contiene già in `docs/` un esempio svolto di tutti e tre. Usalo come forma della tua risposta, non come la risposta.