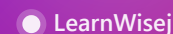


Architettura e vocabolario dell'integrazione

Guida della lezione · Guida al lab / esame



Guida della lezione

Comprendere il modello componente/widget/controllo e la pipeline di rendering ed eventi di Wisej.NET. Che cosa significa per uno sviluppatore che integra con Wisej.NET — e come affrontare questo modulo.



Focus di produzione

Comprendere il modello componente/widget/controllo e la pipeline di rendering ed eventi di Wisej.NET.

1. Il problema di integrazione

Gli sviluppatori trattano spesso i widget JavaScript come oggetti isolati del browser. In Wisej.NET l'integrazione va progettata come un componente posseduto dal server che esegue il rendering dello stato, riceve gli eventi e delega a un widget client il lavoro specifico del browser.

2. Concetti chiave

Concetto	Come usarlo
Component	Oggetto .NET lato server che possiede lo stato e partecipa al rendering.
Widget	Oggetto JavaScript lato browser che crea e aggiorna il DOM e l'istanza del widget.

3. La lezione di produzione

L'integrazione in Wisej.NET parte da una semplice regola architetturale: il server possiede lo stato dell'applicazione, mentre il browser possiede il DOM e l'istanza del widget di terze parti. Un componente Wisej.NET invia a un widget lato client una rappresentazione compatta del proprio stato rilevante. Quel widget lato client crea, configura, aggiorna e collega l'oggetto JavaScript di terze parti. Quando sul client accade qualcosa di importante, il widget Wisej.NET rimanda un payload di evento al componente sul server. È la stessa idea su cui si basano i controlli nativi di Wisej.NET. La differenza è che, nelle integrazioni di terze parti, il widget client diventa di solito un adapter attorno a un'altra libreria JavaScript. L'adapter riceve lo stato da .NET, lo traduce nell'oggetto options o nelle chiamate ai metodi del fornitore, ascolta gli eventi del fornitore e inoltra al server solo le informazioni significative. Una buona integrazione non cerca di nascondere che il controllo di terze parti vive nel browser. Tiene l'oggetto del fornitore dentro l'adapter client, espone solo lo stato e gli eventi di cui il lato .NET ha davvero bisogno e documenta quel contratto, così il server non deve mai indovinare che cosa stia facendo il widget.

Il vocabolario, una parola alla volta

Le discussioni sull'integrazione vanno storte quando due sviluppatori usano la stessa parola per cose diverse. Un Component è qualunque oggetto .NET lato server che Wisej.NET sa rappresentare nel browser; un timer o una sorgente dati sono Component senza superficie visiva. Un Control è un Component che occupa spazio in una pagina e ha una rappresentazione visiva. Un Widget è l'oggetto JavaScript nel browser che disegna quella superficie e reagisce all'utente. Ogni Control è un Component, ma non ogni Component è un Control, e nessuno dei due è il Widget.

Server e Client indicano i due lati della comunicazione. Il rendering è il momento in cui il server serializza lo stato rilevante di un Component e lo invia al suo Widget. Un Event è il momento in cui il Widget rimanda qualcosa indietro. Lo State è tutto ciò che il server conserva tra quei due momenti: valori delle proprietà, selezione, dati e permessi. Nel `TemperatureGauge` del lab, `Value` è stato del server, il gauge del fornitore è il Widget e `ThresholdExceeded` è un Event.

Usa queste parole con precisione nei registri delle decisioni e nelle code review. Se una frase non si riesce a scrivere con queste parole, probabilmente il design non è chiaro.

JSON in uscita, JSON di ritorno

La pipeline di rendering è volutamente semplice. Quando lo stato di un Component cambia, Wisej.NET raccoglie le proprietà modificate, le serializza in un oggetto JSON compatto e consegna quell'oggetto al Widget. Il Widget applica ogni valore: per un controllo nativo significa aggiornare il proprio DOM, per un'integrazione significa tradurre il valore nell'oggetto options del fornitore o in una chiamata a un metodo del fornitore. Il server non tocca mai il DOM e il browser non detiene mai lo stato autorevole.

Il percorso di ritorno ha la stessa forma, al contrario. Quando il Widget decide che qualcosa è rilevante, invia al server un evento con nome e un piccolo payload JSON, e il server solleva un evento .NET sul Component. Nel lab, impostare `Value` a 104 sul server invia al gauge una piccola modifica JSON e, quando il gauge supera la soglia, torna indietro un solo evento con il valore corrente e in C# scatta `ThresholdExceeded`.

Tutto ciò che attraversa la comunicazione sono dati, mai comportamento. È questo che rende l'integrazione verificabile: il JSON in entrambe le direzioni può essere registrato nei log, ispezionato negli strumenti del browser e confrontato con un contratto scritto.

Perché l'oggetto del fornitore vive dentro un contenitore

Un Widget di Wisej.NET possiede un elemento DOM che il framework posiziona, dimensiona, mostra, nasconde, applica al tema e distrugge. Anche una libreria di terze parti vuole possedere un elemento: vi crea dei figli, vi collega gestori e a volte lo sostituisce. Se all'oggetto del fornitore venisse consegnato direttamente l'elemento di Wisej.NET, i due proprietari entrerebbero in conflitto. Le modifiche di layout andrebbero perse, la distruzione lascerebbe gestori orfani e una chiamata del fornitore che ricostruisce l'elemento romperebbe la contabilità interna del framework.

Ospitare l'oggetto del fornitore dentro un elemento contenitore fornito dal Widget evita il conflitto. Il framework tiene il proprio elemento, il fornitore tiene il proprio sottoalbero al di sotto e il Widget fa da mediatore tra i due. Il ridimensionamento diventa una chiamata al metodo `resize` del fornitore, la

distruzione diventa una chiamata al metodo `destroy` del fornitore seguita dallo svuotamento del contenitore e il tema diventa uno stile applicato al contenitore.

L'eccezione è una libreria pensata per potenziare un singolo elemento sul posto, come un campo di input con maschera. Anche in quel caso il potenziamento agisce su un elemento dentro il contenitore, mai sulla radice del framework.

Scegliere la profondità dell'integrazione

Non tutti i widget meritano lo stesso investimento. Un'integrazione *naïve* mette un `Wisej.Web.Widget` su una pagina con uno script e delle opzioni; è la scelta giusta per un prototipo, una dashboard interna o un widget usato una sola volta. Un'integrazione riutilizzabile racchiude lo stesso widget in una classe con proprietà tipizzate, così da poterlo mettere su molte schermate senza copiare la configurazione. Un'integrazione adatta ai temi va oltre e collega l'aspetto del widget al tema dell'applicazione. Un'integrazione dall'aspetto nativo è un controllo personalizzato completo, con una classe server e una classe client, progettato, applicato al tema e guidato dagli eventi come un controllo nativo.

Il lab prende questa decisione per tre candidati. Un piccolo gauge usato su una sola dashboard è una *naïve*. Un editor di testo ricco che compare in diversi form e deve seguire il tema merita un wrapper riutilizzabile e adatto ai temi. Una griglia dati che diventa la superficie standard per le liste dell'applicazione giustifica un controllo dall'aspetto nativo, perché da essa dipendono decine di schermate e anni di manutenzione.

- Quante schermate lo useranno, oggi e fra due anni.
- Se deve seguire il tema e funzionare nel Designer.
- Chi gestisce la versione della libreria del fornitore e le sue modifiche incompatibili.
- Quanto costa, più avanti, aver sbagliato la profondità.



Regola di produzione

Tieni ben distinti la proprietà dello stato in `Wisej.NET` e il comportamento del widget client. Documenta il contratto client/server prima di affidarti all'integrazione in produzione.



Errore comune da evitare

Copiare nelle schermate il JavaScript delle demo senza documentare ciclo di vita, risorse, contratti degli eventi, pulizia o confini di sicurezza.



Obiettivi di apprendimento

Al termine di questo modulo sarai in grado di:

- ✓ Distinguere Component, Control, Widget, Client, Server, Rendering, Event e State.
- ✓ Spiegare come i componenti server inviano lo stato JSON ai widget client e ricevono indietro i dati degli eventi in JSON.

- ✓ Descrivere perché i widget di terze parti vivono di solito dentro un contenitore del widget Wisej.NET invece di sostituire il widget Wisej.NET stesso.
- ✓ Scegliere se un widget debba essere una tantum, riutilizzabile, adatto ai temi o dall'aspetto nativo.

Guida al lab / esame

Che cosa costruirai nel lab pratico, il flusso di lavoro consigliato, i deliverable richiesti e le domande di autovalutazione prima della consegna.



Lab pratico

Crea un registro delle decisioni di integrazione per tre widget candidati: un piccolo gauge, un editor di testo ricco e una griglia dati. Decidi quale tipo di integrazione merita ciascuno e spiega perché.

Flusso di lavoro consigliato

- ✓ Apri il progetto esistente del corso di integrazione Wisej.NET ed esegui in locale.
- ✓ Individua il componente server, il widget client, il file di risorse o l'endpoint coinvolti in questo modulo.
- ✓ Fai una piccola modifica alla volta e provala nel browser prima di andare avanti.
- ✓ Tieni il JavaScript specifico del fornitore dentro l'adapter o il wrapper invece di spargerlo tra le pagine.
- ✓ Aggiorna la UI in modo che si vedano chiaramente successo, errore, avanzamento o cambiamenti nella comunicazione.
- ✓ Raccogli screenshot o note che dimostrano che l'integrazione funziona e spiegano la scelta progettuale.
- ✓ Se usi un assistente di coding basato sull'IA, forniscigli il contratto e le regole di Wisej.NET riportate sotto prima di chiedergli del codice; l'esercizio di coding con l'IA di questo modulo ti guida passo passo dall'inizio alla fine.

Deliverable richiesti

- ✓ Diagramma dell'architettura di integrazione
- ✓ Tabella di valutazione dei widget
- ✓ Primo ADR: approccio di integrazione scelto per ciascun widget candidato

Domande di autovalutazione

- ✓ Perché un Control è sempre un Component, ma non ogni Component è un Control?
- ✓ Perché la maggior parte dei widget di terze parti dovrebbe essere ospitata dentro un elemento contenitore?
- ✓ Quale stato appartiene al server e quale può restare locale nel browser?

Lavorare con un assistente di coding basato sull'IA (Claude, OpenAI)

Claude, ChatGPT e gli assistenti di coding costruiti su di essi (Claude Code, Codex, Copilot) scrivono questa integrazione in fretta, e una richiesta senza contesto ti restituisce una risposta generica: un

`System.Windows.Forms.Timer`, un widget creato direttamente su `this.container`, un evento lanciato con `fireWidgetEvent` da dentro `update()` oppure un'istruzione `return` dentro `EvalAsync`. Tutti

compilano o vengono interpretati senza errori, e tutti falliscono a runtime. Dai all'assistente il contratto e le regole del framework fin dall'inizio e questi problemi spariscono. L'assistente è un paio di mani veloci; l'architetto che possiede il contratto e verifica ogni affermazione resti tu.

Istruiscilo come istruiresti un nuovo membro del team

- ✓ **Prima il vocabolario.** Specifica che un Component è un oggetto .NET sul server che possiede lo stato, un Control è un Component con una superficie e un Widget è l'adapter nel browser. "Widget" senza questa definizione invita a scrivere codice solo lato client.
- ✓ **Le regole del framework nel primo messaggio.** Incollale alla lettera: la classe server deriva da `Wisej.Web.Widget` ; gli script del fornitore sono `Packages` ; l'adapter è l' `InitScript` caricato con `GetResourceString` da una risorsa incorporata; lo stato si scrive in `((dynamic)this.Options)` e arriva come `init(options)` / `update(options, old)` ; gli eventi client si dichiarano in `WiredEvents` , si collegano sul client tramite `_addListener` / `_getEventData` e arrivano in `OnWidgetEvent` .
- ✓ **Il contratto prima del codice.** Dai all'assistente i campi di stato, i nomi e i payload degli eventi e i percorsi di errore. Chiedigli di riformulare il contratto con parole sue prima di scrivere una sola riga: una riformulazione sbagliata costa meno da correggere del codice sbagliato.
- ✓ **Indica le versioni.** "Wisej.NET 4, NuGet Wisej-4 4.1.0, net10.0, host Kestrel" elimina un'intera classe di risposte pensate per Wisej 2 / .NET Framework.

Chiedi pezzi piccoli e rivedili uno per uno

- ✓ Richiedi prima solo la classe server, rivedila, poi richiedi solo l'adapter client. Un file per turno mantiene la revisione onesta e gli errori circoscritti.
- ✓ Di' che cosa *non* deve cambiare: "mantieni il contratto; non aggiungere eventi; non validare in JavaScript". Gli assistenti allargano volentieri un design mentre correggono un bug.
- ✓ Incolla l'output del compilatore e gli errori della console del browser **alla lettera**. Errori parafrasati portano a correzioni tirate a indovinare.
- ✓ Quando un'API ti sembra sconosciuta, chiedi: "Quale classe e quale membro di Wisej.NET sono? Dove sono documentati?" Verifica la risposta con la documentazione di Wisej.NET o con la documentazione XML del framework prima di accettarla.
- ✓ Chiedi esplicitamente i percorsi di errore: "Che cosa succede quando il fornitore lancia un'eccezione dentro `update()` ? Mostrami il percorso di codice che la trasforma in un unico evento `error` ." Se non lo chiedi, qualunque prima bozza resta sul percorso felice.

Verifica, poi conserva quello che hai imparato

- ✓ Nulla è finito finché non compila e i cinque percorsi non girano nel browser: successo, avanzamento, un valore non valido, un payload corrotto e il ripristino. Il pannello di traccia è la tua prova.
- ✓ Tieni una breve nota delle convenzioni e delle correzioni su cui ti sei assestato e incollala nel primo messaggio della sessione successiva. Quella nota vale più di qualunque singolo prompt.
- ✓ Tieni il JavaScript specifico del fornitore dentro l'adapter anche quando l'assistente si offre di metterlo direttamente nella pagina; la stessa regola vale per il codice scritto da persone e da

un'IA.

- ✓ Non incollare mai in un prompt chiavi di licenza, stringhe di connessione o dati dei clienti. Usa segnaposto e una libreria del fornitore fittizia.



Regole di Wisej.NET da indicare nelle istruzioni

Gli eventi lanciati in modo sincrono da dentro `update()` vengono scartati: collegali tramite `_addListener`. Definire `this.getWidth`, `this.getHeight` o `this.destroy` sul wrapper nasconde un metodo di qooxdoo e può far collassare il widget a 0 px. `EvalAsync` accetta un'espressione, non un'istruzione `return`. Gli oggetti anonimi assegnati a `Options` vengono convertiti in camelCase durante la trasmissione; i valori restituiti dai WebMethod no. Usa `Wisej.Web.Timer`, mai `System.Windows.Forms.Timer`.

La sezione **Esercizio di coding con l'IA** di questo modulo mette in pratica tutto questo: ti consegna un prompt già pronto che indirizza un assistente a questa guida, e il nostro Sensor Monitor — costruito esattamente così — ha richiesto due correzioni dopo la prima esecuzione, entrambe elencate sopra.