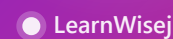


Preparazione alla produzione, risoluzione dei problemi e capstone

Guida della lezione · Guida al lab / esame



Guida della lezione

Preparare le integrazioni per applicazioni reali: prestazioni, diagnostica, aggiornamenti in background, packaging, versioning e manutenibilità. Che cosa significa per uno sviluppatore che integra con Wisej.NET — e come affrontare questo modulo.



Focus di produzione

Preparare le integrazioni per applicazioni reali: prestazioni, diagnostica, aggiornamenti in background, packaging, versioning e manutenibilità.

1. Il problema di integrazione

Un widget può funzionare in una demo e fallire comunque in produzione a causa di race condition, percorsi delle risorse errati, filtri sugli eventi inadeguati, pulizia mancante, crash a design time o volumi di dati mai testati.

2. Concetti chiave

Concetto	Come usarlo
Diagnostica	Rendi visibili gli errori tramite log, strumenti e controlli ripetibili.
Distruzione	Ripulisci le risorse client e i gestori degli eventi quando i widget vengono distrutti.

3. La lezione di produzione

Essere pronti per la produzione è soprattutto una questione di disciplina. L'integrazione deve caricare le risorse in modo deterministico, sopravvivere a creazioni e distruzioni ripetute, aggiornarsi senza perdite di memoria, gestire dati mancanti o malformati, evitare viaggi di andata e ritorno eccessivi verso il server e fallire in modi che gli sviluppatori possano diagnosticare. Deve anche funzionare nel designer di Visual Studio quando è un requisito dichiarato. La risoluzione dei problemi inizia isolando il problema. Se il widget non funziona in JavaScript puro, non è ancora un problema di Wisej.NET. Se funziona in JavaScript puro ma fallisce dentro Wisej.NET, controlla il contenitore, l'ordine degli script, le tempistiche, il contesto e il flusso delle chiamate al server. Usa `debugger`, `sourceURL`, la console del browser e il modello a oggetti del client. Sul server usa il normale debug di Visual Studio e ricorda che gli assembly caricati dal designer possono restare in cache. Il capstone chiede agli studenti di integrare una nuova famiglia di widget JavaScript che non sia Kendo né DevExtreme. Kendo e DevExtreme sono stati gli esempi svolti lungo tutto

il corso. Il capstone dimostra che gli stessi pattern di ciclo di vita, eventi e caricamento dei dati si trasferiscono a una libreria sconosciuta, che è la vera prova che l'approccio all'integrazione è stato capito e non imparato a memoria.

Risolvere i problemi lato client

I problemi lato client si diagnosticano restringendo il punto in cui avvengono. Il primo passo è la demo del fornitore: se il widget fallisce lì con le stesse opzioni, il problema è del fornitore e non c'entra alcun codice Wisej.NET. Se la demo funziona, il passo successivo è l'InitScript iniettato, e un commento `sourceURL` gli dà un nome negli strumenti per sviluppatori così da poter impostare breakpoint su righe reali. Un breakpoint in `init` mostra se il contenitore esiste, se i package sono stati caricati e che cosa contengono le opzioni.

La console dice il resto. Un errore di simbolo non definito al caricamento indica un problema di ordine dei package. Un errore di metodo non trovato dentro una callback indica un `this` perso. Una voce di rete con 404 indica un percorso di risorsa che funzionava in sviluppo ma non sul server. Una clausola di guardia che lancia un messaggio chiaro quando manca un'opzione obbligatoria trasforma un widget vuoto e silenzioso in una diagnosi di una riga.

Registra ogni diagnosi nelle note di risoluzione dei problemi con il sintomo, la causa e la correzione. Le note del capstone sono l'inizio di un manuale operativo del team.

Risolvere i problemi lato server

I problemi del server si affrontano con i normali strumenti di Visual Studio, più due abitudini specifiche di Wisej.NET. Primo, imposta le impostazioni delle eccezioni in modo da interrompere l'esecuzione quando vengono lanciate le eccezioni sollevate dall'integrazione, perché un'eccezione dentro un gestore `WebRequest` o `WidgetEvent` potrebbe altrimenti essere intercettata dal framework e manifestarsi solo come un risultato mancante nel browser. Secondo, ricorda che il Designer esegue il codice server del controllo in modalità design, quindi un crash lì è un crash del server e si può fare il debug agganciandosi al processo del Designer.

Il Designer tiene anche in cache gli assembly caricati. Dopo aver modificato il codice di un controllo, il Designer può continuare a visualizzare la versione vecchia finché Visual Studio non viene riavviato, e sembra una modifica che non ha avuto effetto. Riavviare fa parte della disciplina, non è un espediente. Il codice in modalità design dovrebbe rilevare la modalità design e mostrare dati di esempio invece di chiamare servizi, così che il Designer non abbia mai bisogno di un database.

Quando il gestore del server è corretto e il browser continua a non mostrare nulla, controlla il corpo della risposta nel pannello di rete prima di toccare di nuovo il server. La maggior parte dei problemi rimanenti è un content type o una forma di serializzazione.

Aggiornamenti in background fatti in sicurezza

I widget che mostrano dati in tempo reale hanno bisogno di aggiornamenti che partono dal server senza una richiesta dell'utente. Il pattern ha tre parti. `Application.StartTask` esegue il lavoro su un thread in background mantenendo il contesto della sessione, così il codice può ancora raggiungere la pagina e il widget. Dentro quel task il server cambia le opzioni o emette una `Call` sul widget, esattamente come

farebbe il gestore di un pulsante. Poi `Application.Update` invia al browser le modifiche in sospeso, perché senza una richiesta in corso nient'altro le invierebbe.

Serve attenzione in tre punti. La frequenza degli aggiornamenti deve essere limitata, perché ogni chiamata è un viaggio di andata e ritorno e un widget aggiornato cento volte al secondo è una pagina lenta. La sessione può terminare mentre il task è in esecuzione, quindi il task deve gestire un widget già distrutto senza lanciare eccezioni. E la chiamata al fornitore deve essere eseguita sul client dopo la modifica di opzione da cui dipende, cosa che il raggruppamento garantisce solo quando entrambe vengono emesse nello stesso aggiornamento.

Raggruppa le modifiche, invia una volta per ogni cambiamento significativo e ferma il task quando la pagina si chiude.

La checklist di produzione

Il capstone viene consegnato con una checklist che un revisore può applicare a qualunque libreria di integrazione, e ogni sezione corrisponde a un punto in cui le demo falliscono in produzione. Risorse: ogni percorso di package è un percorso dell'applicazione o una risorsa incorporata, le versioni sono fissate e non serve alcuna CDN. Versioning: la versione del fornitore, la versione del wrapper e le note sulle modifiche incompatibili sono registrate, e un aggiornamento ha una pagina di test. Sicurezza: ogni endpoint valida e autorizza, nessun oggetto di dominio attraversa la comunicazione e l'HTML grezzo proveniente dal fornitore viene sanificato.

Prestazioni: il primo rendering carica una sola pagina di dati, gli eventi vengono filtrati sul client, gli aggiornamenti in background sono limitati e la distruzione elimina l'istanza del fornitore e scollega ogni gestore, così una sessione lunga non perde memoria. Design time: il controllo mostra dati di esempio nel Designer senza servizi e senza lanciare eccezioni.

- Il wrapper si può creare e distruggere cento volte senza crescita della memoria?
- Si riesce a distinguere un errore dello script del fornitore da un errore dell'infrastruttura di Wisej.NET?
- Un altro sviluppatore ha tutto ciò che gli serve per aggiornare il fornitore senza l'autore originale?



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:

- ✓ Risolvere i problemi del codice client con Chrome DevTools, nomi di script dinamici, breakpoint, ispezione dalla console e demo del fornitore.
- ✓ Risolvere i problemi del codice server con Visual Studio, impostazioni delle eccezioni, debug in modalità design e la disciplina del riavvio del designer.
- ✓ Usare con attenzione gli aggiornamenti in background con `Application.StartTask`, le chiamate al client e `Application.Update`.
- ✓ Preparare checklist di risorse, versioning, sicurezza e prestazioni per una libreria di integrazione.

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

Capstone: integra un widget di terze parti che non conosci come wrapper Wisej.NET riutilizzabile, con chiamate dal server, almeno un evento e caricamento dei dati tramite postback o WebMethod.

Flusso di lavoro consigliato

- ✓ Apri il progetto esistente del corso di integrazione Wisej.NET ed esegilo 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.

Deliverable richiesti

- ✓ Implementazione del capstone
- ✓ ADR dell'integrazione
- ✓ Checklist di risorse e sicurezza
- ✓ Script della demo e note di risoluzione dei problemi

Domande di autovalutazione

- ✓ Come dimostri che un errore dipende dall'uso della libreria del fornitore e non dall'infrastruttura di Wisej.NET?
- ✓ Che cosa va ripulito quando un widget viene distrutto?
- ✓ Che cosa deve contenere una checklist di integrazione per la produzione?