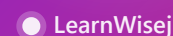


Object model JavaScript, API del browser e contratti di interop sicuri

Guida della lezione · Guida al lab / esame



Guida della lezione

Interop JS avanzata, eventi client, funzionalità del browser, validazione dei contratti. Che cosa significa per un team Wisej.NET avanzato — e come affrontare questo modulo.



Focus di produzione

Interop JS avanzata, eventi client, funzionalità del browser, validazione dei contratti.

1. Il problema in produzione

Wisej.NET permette agli sviluppatori di aggiungere JavaScript dove porta valore, ma l'interop avanzata può diventare fragile se ogni confine non viene trattato come una piccola API.

2. Concetti chiave

Concetto	Perché è importante
modello a oggetti App	La rappresentazione lato client dei componenti Wisej.NET che JavaScript può raggiungere. Usala deliberatamente, tramite punti di ingresso documentati, non frugando tra i dettagli interni.
evento client	Un evento lato browser che viene inoltrato al server. Solo gli eventi significativi dovrebbero attraversare la rete, con payload compatti.
metodo remoto	Un metodo del server che il browser può chiamare direttamente, come un WebMethod, e che restituisce un risultato a JavaScript. Richiede la stessa validazione di qualsiasi endpoint.
rilevamento delle funzionalità del browser	Verificare che cosa supporta il browser prima di usare un'API. Usalo solo per ciò che ha valore lato browser, mentre le regole di business restano sul server.
contratto di interop	Una descrizione documentata e con un nome di ciò che JavaScript invia al server e di ciò che riceve in risposta, così entrambi i lati possono essere testati e versionati.
tempistiche del ciclo di vita	Sapere quando un widget client viene creato, visualizzato, aggiornato e distrutto, così gli script vengono eseguiti al momento giusto e fanno pulizia dopo di sé.
validazione dell'input	Trattare ogni valore che arriva dal browser come non attendibile e validarlo sul server, perché il codice client può essere modificato da chiunque.

3. Il filo della lezione

JavaScript è uno strumento, non l'architettura

Le applicazioni Wisej.NET avanzate possono usare le API del browser, la gestione della tastiera, gli appunti, la fotocamera, gli scanner e widget specializzati. Il server resta l'autorità per flussi di lavoro, permessi e persistenza.

Dai un nome al contratto

Ogni metodo di interop dovrebbe avere un nome, uno scopo, uno schema di input, uno schema di output, una regola di validazione e un comportamento in caso di errore. Trattalo come una minuscola API pubblica tra il browser e il server.

Funzionalità del browser senza regole di business nel browser

JavaScript si guadagna il suo posto in un'applicazione Wisej.NET enterprise quando il browser può fare qualcosa che il server non può: reagire a un tasto prima di un round trip, leggere gli appunti, comunicare con uno scanner o riportare le dimensioni dello schermo e lo stato della rete. La command palette del lab è un buon esempio. Intercettare Ctrl+K, filtrare un elenco mentre l'utente digita ed evidenziare una corrispondenza sono aspetti puramente del browser, quindi appartengono a un piccolo script client.

Decidere che cosa fa un comando è un'altra questione. Quando l'utente preme Invio su `approve`, lo script invia al server una richiesta con un nome, e il server decide se l'utente corrente, nel tenant corrente, può approvare questo ordine di lavoro. Il controllo dei permessi, la transizione di stato, la voce di audit e la persistenza vengono tutti eseguiti in un servizio sul server. Se allo script fosse permesso di modificare direttamente un campo di stato, ogni regola di quel servizio potrebbe essere aggirata dagli strumenti per sviluppatori.

Un test utile in code review è chiedersi che cosa si romperebbe se il browser mentisse. Se la risposta è solo un difetto estetico, il codice è nel posto giusto. Se la risposta è un'approvazione sbagliata o un record trapelato, quella logica deve spostarsi sul server.

Contratti piccoli con input e output con un nome

Un contratto di interop è un accordo scritto su un singolo attraversamento del confine. Il lab lo registra in un documento e nel codice come un tipo di richiesta e un tipo di risultato, come una `ClientCommandRequest` che trasporta un nome di comando, un ID di destinazione e un argomento opzionale, e un metodo server come `RunClientCommand` che restituisce un risultato tipizzato. Ogni campo ha un tipo, un intervallo consentito e un valore predefinito. Nient'altro viene accettato.

Mantenere piccolo il contratto è ciò che lo rende sicuro. Un payload con tre campi con un nome si valida in poche righe, si registra nei log con il suo correlation ID e si versiona quando si aggiunge un campo. Un payload che inoltra un oggetto JavaScript arbitrario non si può validare affatto, perché il server non sa che cosa sta guardando. La validazione viene eseguita prima di chiamare qualsiasi servizio, e un fallimento restituisce un errore con un nome come `PERMISSION_DENIED` o `INVALID_TARGET` anziché il messaggio di un'eccezione.

- Nome: a che cosa serve la chiamata, in una frase.
- Schema di input: ogni campo, il suo tipo e la sua regola di validazione.

- Schema di output: la forma del successo e ogni fallimento con un nome.
- Proprietario: quale servizio del server impone le regole.

Eventi client e callback del server

Il traffico tra browser e server scorre in due direzioni. Un evento client porta al server qualcosa che è successo nel browser, come una selezione nella palette o un cambiamento rilevato nelle funzionalità. Una callback del server è il server che chiama un widget client per aggiornarlo, dargli il focus o rileggerne un valore. Entrambe le direzioni attraversano lo stesso confine di fiducia, quindi entrambe richiedono la stessa disciplina.

Per gli eventi client, inoltra solo gli eventi che hanno un significato di business, con un payload compatto. Una palette che sollevasse un evento a ogni tasto inonderebbe la sessione di round trip e non darebbe al server nulla di utile su cui agire. Per le callback del server, aspettati che il widget possa mancare, essere stato rilasciato o non essere ancora inizializzato, e rendi la callback sicura da eseguire due volte. Una callback che aggiunge una riga al DOM dovrebbe verificare se la riga esiste già.

Ogni callback e ogni gestore di evento dovrebbero essere attribuibili. Registra nei log utente, tenant, correlation ID e nome del comando sul lato server di ogni attraversamento, così un tecnico del supporto può ricostruire che cosa ha chiesto il browser e che cosa ha risposto il server.

Tempistiche, ciclo di vita e differenze di funzionalità

Il fallimento di interop più comune è uno script che viene eseguito prima che esista il suo destinatario. Il lab collega lo script della palette in modo che si agganci solo dopo che il widget che lo ospita è stato creato e visualizzato, non al caricamento della pagina. La stessa regola vale per qualsiasi codice che legge un elemento del DOM, si iscrive a un evento di un widget o chiama un'API del fornitore. Se l'elemento viene creato più tardi, lo script deve attendere l'evento di creazione o essere invocato dal widget stesso.

Il ciclo di vita ha una fine oltre che un inizio. Quando un form si chiude o un widget viene rilasciato, i gestori agganciati alla window o al document continuano a girare finché non vengono rimossi. Nel corso di una lunga sessione questi residui si accumulano, trattengono riferimenti a widget morti e producono errori che sembrano non avere a che fare con la schermata che li ha causati. Ogni aggancio richiede lo sgancio corrispondente.

Anche i browser differiscono. Il pannello delle funzionalità del lab esiste proprio perché l'accesso agli appunti, alla fotocamera e il permesso per le notifiche variano per browser, dispositivo e contesto di sicurezza. Rileva la funzionalità, riporta il risultato al server come dato e lascia che il server scelga un fallback, come un upload di file al posto di uno scatto con la fotocamera. Non dedurre mai una funzionalità dalla stringa dello user agent.



La regola di produzione avanzata

Non fermarti quando la schermata sembra corretta. Prima di considerare completo il modulo, spiega a chi appartiene lo stato, le implicazioni di sicurezza, i percorsi di errore e il comportamento in produzione.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Usare JavaScript per le funzionalità del browser senza spostare le regole di business sul client.
- ✓ Definire piccoli contratti di interop con input, output e validazione con un nome.
- ✓ Usare in sicurezza eventi client e callback verso il server.
- ✓ Gestire le differenze di tempi, di ciclo di vita dei widget e di funzionalità del browser.

Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.



Lab pratico

Aggiungi una command palette da tastiera avanzata e un pannello delle funzionalità del browser. Usa JavaScript per raccogliere le funzionalità del browser e attivare comandi lato server, poi imponi i permessi lato server prima di eseguire il comando.

Approccio suggerito

- ✓ Leggi il problema in produzione e individua il rischio architetturale prima di toccare la UI.
- ✓ Crea o aggiorna per primo l'incapsulamento in un servizio, comando, modello o componente coinvolto.
- ✓ Costruisci la schermata o il componente Wisej.NET seguendo gli standard di progetto stabiliti.
- ✓ Aggiungi una dimostrazione del percorso di errore, non solo del percorso felice.
- ✓ Annota che cosa è cambiato nel registro di apprendimento e aggiorna la checklist relativa.

Deliverable richiesti

- ✓ Documento del contratto di interop
- ✓ Script della command palette
- ✓ Metodi di callback lato server
- ✓ Pannello delle funzionalità del browser
- ✓ Note della revisione di sicurezza

Domande di revisione per lo studente

- ✓ Che cosa succede prima che il widget di destinazione esista?
- ✓ Di quali valori provenienti da JavaScript ci si fida?
- ✓ Quali regole di business restano lato server?



La regola di produzione avanzata

Non fermarti quando la schermata sembra corretta. Prima di considerare completo il modulo, spiega a chi appartiene lo stato, le implicazioni di sicurezza, i percorsi di errore e il comportamento in produzione.