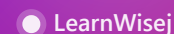


Modello mentale della libreria di controlli

Guida della lezione · Guida al lab / esame



Guida della lezione

L'albero dei controlli sul server e il modello dei widget nel browser, le famiglie di controlli, le proprietà comuni e le convenzioni di denominazione, e lo shell di Operations Console che cresce lungo tutto il corso. Che cosa significa per uno sviluppatore Wisej.NET — e come affrontare questo modulo.



Focus di produzione

L'albero dei controlli sul server e il modello dei widget nel browser, le famiglie di controlli, le proprietà comuni e le convenzioni di denominazione, e lo shell di Operations Console che cresce lungo tutto il corso.

1. Il problema in produzione

Wisej.NET include più di cento controlli e un team che non condivide un modello mentale su di essi finisce con venti schermate di ricerca diverse, cinque modi di segnalare un errore di validazione e JavaScript personalizzato dove sarebbe bastato un controllo nativo. Questo modulo fissa il vocabolario, le regole di scelta e le convenzioni su cui si appoggiano tutti i moduli successivi, e avvia lo shell di Operations Console.

2. Concetti chiave

Concetto	Perché è importante
albero dei controlli sul server	La gerarchia di oggetti controllo .NET che il server mantiene per ogni sessione; il tuo codice imposta proprietà e gestisce eventi qui, mai nel markup.
widget del browser	L'oggetto JavaScript che Wisej.NET crea per rappresentare un controllo lato server, tenuto allineato da cambi di stato serializzati, non da ricaricamenti di pagina.
famiglia di controlli	Un gruppo di controlli che condividono uno scopo (editor, contenitori, liste, griglie, contenuti, menu, notifiche): è qui che si prendono le decisioni.
extender	Un componente che aggiunge proprietà o comportamenti ai controlli esistenti a design time, senza doverli derivare.
Widget ed estensione	Il controllo <code>Widget</code> ospita un widget JavaScript personalizzato o di terze parti dentro il modello a eventi di Wisej.NET; un'estensione è un'integrazione già confezionata, di solito da NuGet.
UserControl	Una composizione riutilizzabile di controlli lato server con un significato di business e una superficie pubblica ridotta.
guida alla scelta dei controlli	Un documento vivo del team che fissa controlli standard, docking, stile di validazione e di notifica e convenzioni sulle griglie per ogni pattern di schermata.

3. Il filo della lezione

Prima gli oggetti sul server, poi i widget nel browser

È il modello mentale da cui dipende tutto il resto: il codice applicativo manipola oggetti .NET sul server e Wisej.NET serializza verso i widget JavaScript del browser solo i cambi di stato di cui hanno bisogno.

Quando Operations Console imposta `statusLabel.Text` non viene scritto alcun markup: il runtime invia al widget soltanto lo stato cambiato. Un `Button` non è una stringa di HTML ma un oggetto sul server con proprietà, eventi, un ciclo di vita, supporto a design time, temi e una rappresentazione lato client — e ogni sessione del browser possiede il proprio albero di questi oggetti.

Il primo errore tipico segue direttamente da qui: trattare i controlli come frammenti di HTML, ricorrendo a markup e script lato client dove esistono già una proprietà, un evento o un'impostazione del tema. Il browser è il secondo confine: accesso al file system, stampa, upload, download, iframe e librerie JavaScript si comportano diversamente dai controlli desktop, ed è per questo che esistono `Application.Download` e il controllo `Upload` invece di una finestra di dialogo dei file.

I controlli partecipano al tema: la documentazione di ciascun controllo indica la classe del widget e le appearance key del tema che usa, cosa che diventa determinante appena lo stile richiede più di `BackColor`, `ForeColor` o `Font`. Devi sapere quando basta una semplice proprietà, quando conviene passare dal tema, quando è sufficiente un extender e quando è giustificato uno `UserControl` riutilizzabile o un `Widget` personalizzato. Ignorare le appearance key fino a fine progetto è un errore che si paga con decine di override di colore locali.

Le famiglie di controlli e le regole di scelta

La libreria si divide in famiglie, ed è nella famiglia che si prendono le decisioni. Gli editor (`TextBox` , `TypedTextBox` , `MaskedTextBox` , `DateTimePicker` , `NumericUpDown` , `ComboBox`) raccolgono input tipizzato. Pulsanti e menu (`Button` , `MenuBar` , `ToolBar` , `ContextMenu`) avviano comandi. I contenitori (`Panel` , `SplitContainer` , `TabControl` , `Accordion` , i pannelli di layout) danno struttura alla schermata. Liste, alberi, repeater e griglie (`ListView` , `TreeView` , `DataRepeater` , `DataGridView`) mostrano dati. I controlli di contenuto rendono testo, immagini, HTML, PDF, file e contenuti ospitati dal browser. Le notifiche (`Toast` , `AlertBox` , `MessageBox`) comunicano lo stato. Gli extender aggiungono comportamento ai controlli esistenti senza derivarli, mentre estensioni e widget portano capacità web esterne dentro il modello a componenti ed eventi.

Il corso tratta la libreria come un design system, non come un elenco da memorizzare: quello che conta è quale famiglia è adatta al compito, come la si popola, come si comporta con grandi quantità di dati, layout e temi, come dialoga con il browser e come resta manutenibile in un'applicazione server multi-utente. Una buona decisione parte dal compito dell'utente: un editor per modificare un valore, una `ListView` per scegliere da una collezione con presentazione a icone, tile o dettagli, una `TreeView` per dati gerarchici con espansione e chiusura, una `DataGridView` per record tabellari con ordinamento, modifica, colonne personalizzate o grandi volumi, e uno `UserControl` quando un gruppo di controlli ha un significato di business riutilizzabile.

Tre regole tengono onesta la decisione. Scegli il controllo più semplice che esprime il compito senza codice personalizzato, preferendo i controlli standard e gli extender prima di qualsiasi widget JavaScript su misura: costruire un widget prima di aver esaurito la libreria nativa è l'errore più costoso di tutto il corso. Valuta la dimensione dei dati prima di scegliere la strategia di popolamento: `Rows` e `Items` impostati a mano vanno bene per insiemi piccoli, gli insiemi grandi richiedono binding, modalità virtuale, cache o caricamento lazy. Usa il tema e le convenzioni di appearance per lo stile valido in tutta l'applicazione e le singole proprietà solo per le eccezioni locali.

Proprietà comuni e convenzioni che tengono insieme tutti i moduli

I nomi devono identificare il ruolo di business, non soltanto il tipo visivo: `customerGrid` dice più di `dataGridView1` , e `StatusPanel` , `SearchTextBox` e `SaveButton` rendono leggibili i gestori degli eventi senza aprire il designer. Tenere i nomi predefiniti del designer trasforma `button3_Click` in archeologia, e in sede di valutazione pesa parecchio.

Docking e ancoraggio devono essere intenzionali: `Dock` riempie e impila, `Anchor` segue i bordi quando il browser viene ridimensionato, e il posizionamento assoluto ha senso solo dove nient'altro funziona; mescolare a caso strategie di layout in una stessa schermata è un errore che salta fuori al primo ridimensionamento. I controlli che fanno parte del flusso da tastiera hanno bisogno di un `TabIndex` deliberato, gli errori di validazione di messaggi chiari, le immagini di una convenzione basata su `ImageList` o risorse, le operazioni costose di un feedback visibile, e le etichette di accessibilità devono corrispondere al ruolo di business.

Un'applicazione di produzione mette tutto questo per iscritto in una guida alla scelta dei controlli: un documento vivo che definisce, per ogni pattern di schermata, i controlli standard, il docking predefinito, lo stile di validazione e notifica, la posizione dei menu, il pacchetto di icone, le convenzioni sulle griglie e le aspettative di accessibilità. Non è burocrazia: è il modo in cui un team resta coerente. Il

`ControlSelection.md` del tuo lab è la prima pagina di quella guida e risponde a queste domande per ogni area:

- Il controllo scelto corrisponde al compito dell'utente, e c'è un controllo nativo più semplice?
- I dati sono piatti, gerarchici, tabellari, ripetuti o visuali, e quanto cresceranno?
- La schermata ha bisogno di un confine UserControl, e un Widget personalizzato è davvero giustificato?

Lo shell di Operations Console

Il progetto del corso parte come uno shell e ogni modulo ne aggiunge una sezione, finché non ti ritrovi con un'applicazione di riferimento funzionante invece che con esempi isolati. Lo shell è una `Page` principale (o un `Form`) con quattro aree: la navigazione a sinistra, un'area contenuti che riempie il resto, un'area comandi per le azioni della sezione corrente e un'area di stato che racconta che cosa sta facendo la console.

Nel lab l'area di navigazione contiene un `Button` dal nome chiaro per ogni pagina segnaposto (Editors, Layouts, Lists and Trees, DataGridView, Dashboard, Widgets), l'area contenuti è un `Panel` in dock su Fill e l'area di stato è un pannello in dock con una `Label`. L'unico metodo che vale la pena scrivere sostituisce il contenuto, mette la nuova pagina in dock su Fill e aggiorna la label di stato, così ogni cambio di navigazione viene segnalato da un unico punto e una pagina che non si carica non lascia l'area vuota.

```
private void Navigate(Control content, string title)
{
    try
    {
        contentPanel.Controls.Clear();
        content.Dock = DockStyle.Fill;
        contentPanel.Controls.Add(content);
        statusLabel.Text = $"Sezione attiva: {title}";
    }
    catch (Exception ex)
    {
        statusLabel.Text = $"Impossibile aprire {title}";
        MessageBox.Show(ex.Message);
    }
}
```

Ogni pagina segnaposto è una classe a sé, più avanti uno `UserControl`, nella propria cartella così che il capstone possa riutilizzarla, e i nomi portano il ruolo ovunque: `navigationPanel`, `contentPanel`, `commandPanel`, `statusLabel`, `editorsButton`. L'obiettivo extra è un piccolo pannello diagnostico che mostra il nome del controllo selezionato, l'ID del record, il profilo client attivo e l'orario dell'ultimo aggiornamento.



La regola da libreria di controlli

Scegli il controllo nativo più semplice che esprime il compito dell'utente prima di scrivere codice personalizzato. I controlli nativi ti danno già supporto a design time, temi, eventi e stato sul server; il passo successivo è un extender o uno UserControl, e solo quando nessun controllo nativo è adatto si costruisce un widget JavaScript.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Spiegare la relazione tra l'albero dei controlli sul server di Wisej.NET e i widget nel browser.
- ✓ Classificare la libreria di controlli per scopo: input, contenitori, liste, griglie dati, contenuti, menu, notifiche, extender ed estensioni.
- ✓ Applicare in modo coerente proprietà ed eventi comuni, denominazione, ancoraggio, docking, immagini, ordine di tabulazione, etichette di accessibilità e convenzioni del designer.
- ✓ Creare lo shell del progetto del corso chiamato Operations Console, che crescerà lungo tutto il percorso.

Guida al lab / esame

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



Lab pratico

Crea lo shell del catalogo di controlli di Operations Console: una Page principale con un'area di navigazione, un'area contenuti, un'area comandi e un'area di stato; pagine segnaposto per Editors, Layouts, Lists and Trees, DataGridView, Dashboard e Widgets; una nota ControlSelection.md dentro il progetto che elenca la famiglia di controlli che userai per ciascuna area; nomi di controlli chiari ovunque; e un'area di stato che si aggiorna a ogni cambio di navigazione.

Punto di partenza

Apri Visual Studio e crea una nuova applicazione web Wisej.NET chiamata `OperationsConsole`; è la soluzione che ogni lab successivo estende, quindi tieni il lavoro di ciascun modulo nella propria cartella o nel proprio namespace. Il template ti dà una `Page` principale vuota e un `Program.cs` che la apre, e nient'altro esiste ancora.

Approccio suggerito

- ✓ Abbozza le quattro aree prima di toccare il designer, poi mettile in dock in quest'ordine: navigazione a sinistra, comandi in alto, stato in basso e il pannello dei contenuti aggiunto per ultimo con Dock impostato su Fill.
- ✓ Rinomina ogni controllo con il suo ruolo di business appena lo inserisci (`navigationPanel` , `contentPanel` , `commandPanel` , `statusLabel` , `editorsButton`) e imposta un `TabIndex` deliberato sui pulsanti di navigazione.
- ✓ Crea uno UserControl segnaposto per sezione (Editors, Layouts, Lists and Trees, DataGridView, Dashboard, Widgets), ciascuno con una label di titolo, nella propria cartella così che il capstone possa riutilizzarlo.
- ✓ Scrivi un unico metodo `Navigate(Control content, string title)` che svuota il pannello dei contenuti, mette la pagina in dock su Fill, la aggiunge e aggiorna la label di stato dentro un try / catch; fai passare da lì ogni pulsante di navigazione.
- ✓ Aggiungi `ControlSelection.md` al progetto e annota, per ogni area e pagina segnaposto, la famiglia di controlli che hai scelto e l'alternativa nativa più semplice che hai scartato.

Deliverable richiesti

- ✓ Shell principale con aree di navigazione, contenuti, comandi e stato
- ✓ Pagine segnaposto per Editors, Layouts, Lists and Trees, DataGridView, Dashboard e Widgets
- ✓ ControlSelection.md con la famiglia di controlli scelta per ogni area
- ✓ Nomi di controlli chiari e un'area di stato che si aggiorna alla navigazione
- ✓ Screenshot e una breve nota che spiega le scelte sui controlli

Domande di autovalutazione

- ✓ Quando imposti `statusLabel.Text` nel metodo di navigazione, che cosa viaggia davvero verso il browser, e perché è diverso dallo scrivere HTML?
- ✓ Quale area dello shell trasformeresti per prima in uno UserControl, e quale sarebbe la sua superficie pubblica?
- ✓ Se un modulo successivo avesse bisogno di un'area di navigazione richiudibile, quale contenitore nativo sostituirebbe il pannello in dock, e perché è preferibile a un widget personalizzato?



La regola da libreria di controlli

Scegli il controllo nativo più semplice che esprime il compito dell'utente prima di scrivere codice personalizzato. I controlli nativi ti danno già supporto a design time, temi, eventi e stato sul server; il passo successivo è un extender o uno UserControl, e solo quando nessun controllo nativo è adatto si costruisce un widget JavaScript.