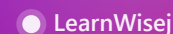


Il sistema visivo di Wisej.NET

Guida della lezione · Guida al lab / esame



Guida della lezione

Come controlli lato server, widget lato client, appearance del tema, stati, classi CSS e motori di layout producono insieme ciò che l'utente vede, quale dei tre livelli di personalizzazione possiede ogni decisione, e lo shell di Adaptive Operations Console che cresce lungo tutto il corso. Che cosa significa per uno sviluppatore Wisej.NET — e come affrontare questo modulo.



Focus di produzione

Come controlli lato server, widget lato client, appearance del tema, stati, classi CSS e motori di layout producono insieme ciò che l'utente vede, quale dei tre livelli di personalizzazione possiede ogni decisione, e lo shell di Adaptive Operations Console che cresce lungo tutto il corso.

1. Il problema in produzione

Un team che tratta Wisej.NET come un framework in cui il CSS viene prima di tutto finisce con `BackColor` impostato su duecento controlli, selettori che si rompono al prossimo aggiornamento e gestori Resize che ricalcolano `Bounds` a mano; quando viene chiesta una versione per il telefono, l'unica opzione rimasta è riscrivere tutto. Questo modulo chiarisce da dove vengono davvero l'aspetto e il layout di un controllo, dà un nome ai tre livelli di personalizzazione e a quale possiede ogni decisione, e avvia lo shell di Adaptive Operations Console che ogni modulo successivo estende.

2. Concetti chiave

Concetto	Perché è importante
controllo sul server e widget client	Scrivi controlli .NET sul server; il browser visualizza widget JavaScript che Wisej.NET mantiene sincronizzati, quindi una decisione visiva può stare da una parte o dall'altra di quel confine.
tema	Un unico file JSON di colori, immagini, font, appearance, stati, stili, proprietà e impostazioni che dà a ogni controllo standard la sua identità; il contratto tra i due mondi.
appearance e <code>AppearanceKey</code>	Ogni controllo espone una chiave di appearance; il tema attivo contiene un'appearance per quella chiave, e una chiave semantica come <code>action-button</code> seleziona una variante.
stato	Una condizione di interazione con nome dentro un'appearance (default, hovered, pressed, focused, invalid, selected) i cui stili e proprietà si applicano mentre è attiva.
<code>CssClass</code> e <code>CssStyle</code>	Stile applicato direttamente nel browser: una classe da un foglio di stile applicativo circoscritto, oppure una stringa di stile grezza sull'elemento del widget per un raro valore una tantum.
motore di layout e <code>Dock</code>	Posizione e dimensione vengono dalla gerarchia dei contenitori, non dal tema; il docking impila le regioni contro i bordi del genitore e riempie ciò che resta.
inventario visivo	L'elenco scritto di controlli che hanno bisogno di appearance, token di colori e font, regioni del layout e breakpoint, da cui si pianifica il lavoro su tema, layout e profili.

3. Il filo della lezione

Il tema è il contratto tra i controlli sul server e i widget nel browser

In Wisej.NET lo sviluppatore scrive controlli .NET sul server, ma ciò che l'utente vede è un widget lato client visualizzato dal browser. Un `Button`, una `TextBox`, un `TabControl`, una `DataGridView`, un `Panel` o un widget personalizzato espone una chiave di appearance, il tema attivo contiene un'appearance per quella chiave, e ogni appearance contiene stati, stili, proprietà e componenti figli. È per questo che una modifica visiva professionale si può fare una volta sola, in modo centralizzato, invece di disperderla tra i singoli controlli: cambia l'appearance del pulsante e ogni pulsante di ogni sessione si adegua.

È anche per questo che il theming di Wisej.NET è più del normale CSS. Il tema arriva dentro i controlli stessi, nelle metriche dei widget da cui dipende il layout, nei font e nelle dimensioni con cui il designer di Visual Studio visualizza le schermate, e in colori, immagini e appearance che il solo CSS non può esprimere. Il motore dei temi può generare classi CSS a partire da appearance e stato, e le proprietà del tema possono scavalcare gli stili, quindi una regola scritta a mano contro il DOM interno del widget combatte il motore invece di usarlo. Tratta il file del tema come un asset di design del prodotto: i colori con nome e i font condivisi sono token di design, le appearance modellano le varianti dei componenti, gli stati esprimono l'interazione, e un pulsante, una griglia, un editor, un pannello, una toolbar e un form sembrano coerenti anche quando li hanno creati sviluppatori diversi.

Tre livelli, e quale possiede ogni decisione

Il livello uno è il tema: identità visiva globale, comportamento degli stati, metriche dei widget, colori con nome, font e appearance dei componenti standard. Il livello due è il modello dei controlli lato server:

`BackColor`, `ForeColor`, `Font`, `Padding`, `Margin`, `Dock`, `Anchor`, `Size`, `MinimumSize`, `MaximumSize`, `AppearanceKey` e le proprietà responsive. Il livello tre è lo stile applicato direttamente nel browser: `CssClass`, `CssStyle`, l'extender `StyleSheet` nel codice più vecchio, file CSS personalizzati, `InitScript` e la personalizzazione dei widget JavaScript. Uno sviluppatore di livello intermedio deve sapere quale livello possiede ogni decisione prima di toccarne uno qualsiasi.

La regola di decisione che tutto il corso applica è breve. Prima il tema, per l'identità dei controlli standard. `AppearanceKey` per una variante semantica come un pulsante di azione o un pulsante distruttivo, lasciando che il tema ne definisca gli stati default, hovered, pressed, focused e disabled. `CssClass` per un aggancio circoscritto e specifico dell'applicazione: una card delle metriche, HTML incorporato, un widget di terze parti. `CssStyle` solo per un piccolo valore una tantum o dinamico, raro e locale. I contenitori di layout per il posizionamento, le proprietà responsive per i valori che il designer assegna per profilo, e `ResponsiveProfileChanged` per il comportamento che il valore di una proprietà non può esprimere.

La regola in negativo conta altrettanto: non mettere pezze in CSS diretto su ogni controllo standard. Le appearance del tema sono migliori per i controlli integrati perché partecipano agli stati, al rendering del designer e ai token condivisi; il CSS è per i contenuti specifici dell'applicazione, i widget di terze parti e l'HTML incorporato. Quando un tema "non si applica", i primi sospettati sono un `BackColor`, `ForeColor`, `Font`, `CssStyle` o una `CssClass` personalizzata diretti che lo scavalcano, una chiave di appearance sbagliata, oppure un tema impostato nel designer ma non selezionato per l'applicazione in esecuzione.

L'inventario visivo: token, regioni, breakpoint

Prima che qualcuno apra il Theme Builder, il lab chiede un inventario dell'applicazione come sarà. Percorri Adaptive Operations Console ed elenca i controlli che hanno bisogno di un'appearance personalizzata: i pulsanti di comando, i pannelli che diventano card delle metriche, i tab, l'intestazione della griglia, lo stato non valido dell'editor dei ticket, i tooltip. Elenca i token di colore che il tema definirà una volta e riutilizzerà ovunque: `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` e `focusFrame`, più un font default, uno heading e uno mono. Elenca le regioni del layout: toolbar, barra di navigazione, area di lavoro con card delle metriche e griglia dei ticket, pannello dei dettagli, barra di stato. Infine elenca i breakpoint: desktop a 1440, small desktop a 1024, tablet a 768 in verticale e 1024 in orizzontale, telefono a 390 in verticale e 844 in orizzontale.

Scrivilo come `VisualInventory.md` dentro il progetto e dai a ogni riga una quarta colonna: il livello che la possiede. Gli angoli arrotondati di una card delle metriche appartengono a una `CssClass` circoscritta; il colore di hover del pulsante appartiene al tema; la larghezza del pannello dei dettagli appartiene a `Dock`; la visibilità della barra di navigazione su un telefono appartiene a una proprietà responsive. Annota nello stesso file la decisione sul tema di base: il Modulo 2 deriva il tema AdaptiveOps da uno dei temi integrati e ne eredita appearance, stati, immagini e metriche, quindi annota perché l'hai scelto, non solo quale.

Lo shell di Adaptive Operations Console

Il progetto del corso è una console operativa con un tema brandizzato e un layout desktop, tablet e telefono. La versione desktop ha una barra di navigazione a sinistra, una toolbar, card delle metriche, una

griglia dei ticket e un pannello dei dettagli. La versione mobile comprime la navigazione, impila griglia e dettagli, nasconde il testo a bassa priorità, usa la visualizzazione solo icona dove è appropriato e mantiene comodo lo scorrimento al tocco. Il Modulo 1 costruisce solo lo shell desktop, con `Dock` e `Panel` annidati e nient'altro: toolbar in alto, stato in basso, navigazione a sinistra, dettagli a destra, e un'area di lavoro in dock su Fill con un `MinimumSize` così che non collassi mai.

```
private void BuildShell()
{
    toolbarPanel.Dock = DockStyle.Top;        toolbarPanel.Height = 56;
    statusPanel.Dock = DockStyle.Bottom;      statusPanel.Height = 28;
    navigationPanel.Dock = DockStyle.Left;    navigationPanel.Width = 240;
    detailsPanel.Dock = DockStyle.Right;      detailsPanel.Width = 360;
    workspacePanel.Dock = DockStyle.Fill;
    workspacePanel.MinimumSize = new Size(320, 240);
}

private void MainPage_Resize(object sender, EventArgs e)
{
    widthLabel.Text = $"Width: {this.Width} px";
}
```

L'unico comportamento dello shell è la label di stato che riporta la larghezza corrente del browser. La `Page` principale riempie sempre il viewport del browser, quindi il suo evento `Resize` scatta ogni volta che l'utente ridimensiona la finestra; scrivere `this.Width` nella label dà ai moduli successivi un numero da confrontare con i breakpoint di `ClientProfiles.json` e con il nome in `Application.ActiveProfile`. Racchiudi l'aggiornamento in un try / catch che lascia un testo di ripiego onesto invece di un valore vecchio. Ciò che lo shell non deve contenere è importante quanto ciò che contiene:

- Nessun override di `BackColor`, `ForeColor` o `Font`: l'aspetto lo possiede il tema di base fino al Modulo 2.
- Nessuna stringa `CssStyle` e ancora nessun CSS personalizzato: niente nello shell è abbastanza specifico dell'applicazione da averne bisogno.
- Nessun codice `Resize` che imposta `Bounds`: il docking risponde già a ogni larghezza desktop.

Cattura lo screenshot desktop quando lo shell è in esecuzione e scrivi la nota che spiega, regione per regione, quale livello possiede ogni decisione visiva; è il seme della nota di architettura che il capstone richiede.



La regola del sistema visivo

Usa il meccanismo Wisej.NET che possiede il problema: il tema per l'identità dei controlli standard, `AppearanceKey` per le varianti semantiche, `CssClass` per il CSS circoscritto e specifico dell'applicazione, `CssStyle` solo per valori dinamici una tantum, i contenitori di layout invece del codice di ridimensionamento, le proprietà responsive per i valori di profilo impostati nel designer, e `ResponsiveProfileChanged` per il comportamento che non è semplicemente visivo.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Spiegare perché il theming di Wisej.NET è più del normale CSS e come arriva dentro controlli, widget, metriche del designer, font, colori, immagini e appearance.
- ✓ Riconoscere quando usare un file di tema, un tema integrato, una proprietà del controllo, CssClass, CssStyle o un contenitore di layout.
- ✓ Creare un inventario visivo per un'applicazione esistente e definire token del tema, regioni del layout e breakpoint responsive.
- ✓ Avviare il progetto del corso, Adaptive Operations Console, con uno shell desktop di base e il comportamento mobile di destinazione.

Guida al lab / esame

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



Lab pratico

Crea la soluzione Adaptive Operations Console e il suo primo shell desktop: una Page principale con un pannello toolbar in alto, un pannello di navigazione a sinistra, un'area di lavoro su Fill, un pannello dei dettagli a destra e un pannello di stato in basso, costruiti con Dock e Panel annidati. Scegli un tema di base integrato e annota la scelta, scrivi un VisualInventory.md che elenca i controlli che hanno bisogno di appearance personalizzate, i token con nome per colori e font, le regioni del layout e i breakpoint desktop, tablet e telefono, e aggiungi una label di stato che riporta la larghezza corrente del browser così che i moduli successivi possano verificare i profili.

Punto di partenza

Apri Visual Studio e crea una nuova applicazione web Wisej.NET chiamata `AdaptiveOperationsConsole`; è la soluzione che ogni lab successivo estende, quindi tieni il lavoro di ciascun modulo nella propria cartella. Il template ti dà una `Page` principale vuota, un `Program.cs` che la apre e la selezione del tema predefinito; non esistono ancora pannelli, file di tema né inventario.

Approccio suggerito

- ✓ Abbozza prima su carta le cinque regioni, poi inserisci cinque `Panel` nella Page principale e mettili in dock in quest'ordine: `toolbarPanel` in alto, `statusPanel` in basso, `navigationPanel` a sinistra, `detailsPanel` a destra, e `workspacePanel` per ultimo con Dock impostato su Fill e un `MinimumSize` di 320 per 240.
- ✓ Scegli uno dei temi integrati come base, selezionalo per l'applicazione in esecuzione (non solo nel designer) e scrivi negli appunti del lab il motivo della scelta prima di cambiare qualsiasi altra cosa.
- ✓ Aggiungi una `widthLabel` al pannello di stato, gestisci l'evento `Resize` della Page principale e scrivici `$"Width: {this.Width} px"` da un unico metodo `ReportWidth()` racchiuso in un try / catch; chiamalo una volta dal costruttore così che la label non sia mai vuota.
- ✓ Crea nel progetto `VisualInventory.md` con quattro tabelle: controlli che hanno bisogno di appearance personalizzate, token di colori e font (brandPrimary, brandAccent, surface, surfaceAlt, textMain, textMuted, danger, warning, success, focusFrame; default, heading, mono), regioni del layout, e i breakpoint desktop, tablet e telefono; dai a ogni riga una colonna "posseduto da" che indica tema, proprietà del controllo, CssClass, CssStyle o contenitore di layout.
- ✓ Esegui lo shell, ridimensiona il browser dalla larghezza desktop a quella tablet e telefono osservando la label, cattura lo screenshot desktop e scrivi la nota che spiega quale livello possiede ogni decisione visiva nello shell.

Deliverable richiesti

- ✓ Soluzione Adaptive Operations Console con uno shell desktop in dock composto dai pannelli toolbar, navigazione, area di lavoro, dettagli e stato

- ✓ Tema di base scelto e configurato, con il motivo annotato negli appunti del lab
- ✓ VisualInventory.md che elenca i controlli che hanno bisogno di appearance, i token del tema, le regioni del layout e i breakpoint
- ✓ Label di stato che riporta la larghezza corrente del browser
- ✓ Screenshot desktop e una nota che spiega quale livello possiede ogni decisione visiva

Domande di revisione per lo studente

- ✓ Il comando principale della toolbar ha bisogno del colore del brand su ogni schermata. Quale dei tre livelli possiede questa decisione, e che cosa andrebbe storto se invece impostassi `BackColor` sul pulsante?
- ✓ La tua label della larghezza indica 1024 px e il pannello dei dettagli è ancora visibile. Quale meccanismo di un modulo successivo deciderà se deve esserlo, e perché in Wisej.NET una media query CSS è la prima risposta sbagliata?
- ✓ Se domani il tema di base venisse sostituito con un altro tema integrato, quali righe del tuo VisualInventory.md cambierebbero e quali no, e che cosa ti dice questo su dove vivono le decisioni dell'inventario?



La regola del sistema visivo

Usa il meccanismo Wisej.NET che possiede il problema: il tema per l'identità dei controlli standard, AppearanceKey per le varianti semantiche, CssClass per il CSS circoscritto e specifico dell'applicazione, CssStyle solo per valori dinamici una tantum, i contenitori di layout invece del codice di ridimensionamento, le proprietà responsive per i valori di profilo impostati nel designer, e ResponsiveProfileChanged per il comportamento che non è semplicemente visivo.