

Architettura di disegno in produzione, prestazioni e capstone

Guida della lezione · Guida al lab / esame



Guida della lezione

Rendere il codice di disegno pronto per la produzione: un unico modello visuale con un livello di geometria puro e renderer separati per ciascuna superficie, la strumentazione per tempo di rendering e dimensione del payload, il profiling di un percorso disegnato contro uno off-screen, il comportamento su ridimensionamento, cambio di tema, dati vuoti e font mancante, un equivalente testuale accessibile e il capstone Visual Operations Studio. Che cosa significa per uno sviluppatore .NET esperto che già costruisce schermate Wisej.NET e ora ha bisogno di grafica personalizzata — e come affrontare questo modulo.



Focus di produzione

Rendere il codice di disegno pronto per la produzione: un unico modello visuale con un livello di geometria puro e renderer separati per ciascuna superficie, la strumentazione per tempo di rendering e dimensione del payload, il profiling di un percorso disegnato contro uno off-screen, il comportamento su ridimensionamento, cambio di tema, dati vuoti e font mancante, un equivalente testuale accessibile e il capstone Visual Operations Studio.

1. Il problema in produzione

VisualOperationsStudio possiede ormai quattro superfici di rendering, e ciascuna è stata costruita nel proprio modulo con la propria idea privata di che cosa significhi la salute di una macchina. È così che un sottosistema di disegno marcisce: qualcuno sposta una soglia di warning, tre superfici sono d'accordo e il PNG esportato contraddice in silenzio la schermata da cui proviene. Questo modulo fa collassare quelle quattro interpretazioni in un unico modello dietro un unico livello di geometria, e poi dimostra con misure, percorsi di errore deliberati e un equivalente semantico che il risultato regge dati veri, piattaforme vere e utenti veri.

2. Concetti chiave

Concetto	Perché è importante
Modello visuale	Solo stato semantico: letture, didascalie, nodi e archi, serie di andamento, gravità come scelta di dominio e selezione condivisa, così che nessuna superficie abbia bisogno di una copia.
Livello di geometria	Matematica quasi pura che trasforma lo stato del modello più la dimensione corrente della superficie in rettangoli, angoli di apertura, posizioni delle etichette e regioni di hit, testabile senza alcun controllo presente.
Adapter di superficie	Un renderer per superficie invece di un'interfaccia comune: <code>System.Drawing.Graphics</code> e <code>Wisej.Web.Canvas</code> mantengono le proprie primitive sopra una geometria condivisa.
Budget di prestazioni	Un numero concordato prima di mettere mano alle ottimizzazioni, per esempio un controllo di stato disegnato sotto i 5 ms e un redraw completo del Canvas entro circa 16-50 ms.
Strumentazione del rendering	Tempo trascorso registrato, dimensioni del bitmap, numero di oggetti disegnati e crescita del payload per ciascuna superficie, così che un'ottimizzazione si argomenti con un prima e un dopo.
Test a immagine di riferimento	Un rendering off-screen a input fisso confrontato con un riferimento memorizzato usando una tolleranza, perché antialiasing ed encoder differiscono fra piattaforme.
Equivalente semantico	L'etichetta, l' <code>AccessibleDescription</code> , il tooltip o la lista di accompagnamento che porta ogni valore essenziale mostrato dalla grafica, così che nulla di importante esista solo come pixel.

3. Il filo della lezione

Un modello, un livello di geometria, un renderer per superficie

Il progetto riutilizzabile ha tre livelli. Il modello visuale contiene stato semantico e nient'altro. Il livello di geometria converte quello stato e le dimensioni correnti del viewport in rettangoli, punti, path, posizioni delle etichette e regioni di hit. Il renderer converte la geometria in una superficie, e solo lì qualcosa si sta parlando con un oggetto `Graphics`, con un `Wisej.Web.Canvas` o con un frammento HTML. Un unico modello di topologia può allora essere mostrato in modo interattivo sul Canvas, esportato in PNG tramite `System.Drawing.Managed` ed elencato per l'accessibilità, senza che le regole di dominio esistano tre volte.

Resisti alla tentazione di unificare i renderer stessi: la loro semantica è davvero diversa, e un'interfaccia limitata a ciò che entrambi sanno fare costa più di quanto faccia risparmiare, a meno che l'elemento visivo non sia banale. Condividi invece il modello e la geometria. Per VisualOperationsStudio la ristrutturazione è per lo più uno spostamento: i calcoli dell'indicatore del Modulo 2 e le conversioni in coordinate mondo del Modulo 5 passano in `Geometry`, mentre l'handler di `Paint`, l'handler di `CellPaint`, il metodo `Redraw` del Canvas e il renderer off-screen diventano adapter sottili. Anche la selezione passa sul modello, ed è ciò che permette a un clic su una riga della griglia di ripuntare l'indicatore e di selezionare lo stesso nodo del Canvas.

Prima i budget, poi la strumentazione

Decidi i numeri prima di ottimizzare. Budget iniziali ragionevoli: un controllo di stato disegnato sotto i 5 ms di rendering sul server a dimensione normale, una colonna di griglia disegnata a mano senza scatti visibili nello scorrimento con numeri di righe rappresentativi, un redraw completo del Canvas entro circa 16-50 ms, e un export che può richiedere più tempo perché è un comando esplicito ma non deve bloccare sessioni estranee. Ciò che registri cambia da superficie a superficie:

- **Paint** e **CellPaint** sul server: tempo trascorso nell'handler, tasso di allocazione, dimensione dell'immagine temporanea, crescita del payload di risposta e ridisegni per flusso di lavoro.
- Canvas: comandi di disegno per redraw, dimensione del payload della scena, frequenza dei redraw, latenza di round trip del puntatore e il costo che il browser stesso paga su una scena grande.
- System.Drawing.Managed off-screen: dimensioni del bitmap e memoria dei pixel, tempo di encoding, comportamento nel caricamento dei font, dimensione dell'output e differenze visive per piattaforma.

```
var watch = System.Diagnostics.Stopwatch.StartNew();
using (var bitmap = new Bitmap(width, height))
using (var g = Graphics.FromImage(bitmap))
{
    _topologyRenderer.Render(g, scene, layout);
    watch.Stop();
    _metrics.Record("topology.export", watch.Elapsed,
        bitmap.Width, bitmap.Height, layout.VisibleNodes.Count);
}
```

Con la registrazione attiva, fai il profiling di un percorso disegnato sul server contro uno off-screen usando gli strumenti CPU Usage e di allocazione, più microbenchmark attorno alla geometria pura. Un handler veloce per una cella non è automaticamente veloce per mille, i grafi di oggetti creati per cella sono il colpevole abituale, e le sessioni concorrenti vanno dentro la misura, non dopo.

Due matrici di test: piattaforme e browser

Il rendering off-screen va verificato su ogni piattaforma che prometti: un server .NET moderno su Windows, un container Linux, macOS per sviluppo e test dove supportato, e iOS o Android Hybrid se li distribuisce. Automatizza metriche del testo, font richiesti, immagini trasparenti, gradienti, antialiasing e salvataggio delle immagini, appoggiandoti a output di riferimento confrontati con una tolleranza invece che con un'uguaglianza byte per byte, perché un encoder diverso o mezzo pixel di antialiasing sono una differenza di piattaforma e non una regressione.

Il Canvas ha bisogno di una seconda matrice, perché è il browser a fare il rendering: verifica le versioni correnti di Chromium, Firefox e Safari, tenendo d'occhio font, compositing, ombre e gesture touch. Ridimensiona tutto mentre testi, dato che **Redraw** deve ricostruire l'intera scena dal modello e un handler di **Paint** deve ricalcolare la geometria da larghezza e altezza correnti. Struttura la suite a livelli, come consiglia il pacchetto: unit test per normalizzazione, mappatura valore-angolo, hit testing, trasformazioni di zoom e culling; test a immagine di riferimento per il renderer off-screen; test di integrazione per il

rendering della pagina, il redraw dopo un ridimensionamento, lo scorrimento della griglia e i cambi di tema; passaggi manuali per touch, tastiera, screen reader, zoom del browser e alta densità di pixel.

Degrada invece di sollevare eccezioni, e non consegnare mai solo pixel

Disegna apposta gli stati sbagliati. Un'immagine mancante diventa un glifo segnaposto più una voce di log, un font mancante si risolve attraverso una catena di fallback definita, i dati vuoti diventano un elemento visivo neutro «Nessun dato», un numero non valido viene limitato o rifiutato prima del rendering, e un errore di export emerge come un messaggio su cui l'utente può agire invece che come un'immagine vuota. I temi meritano la stessa attenzione: evita di fissare nel codice uno sfondo bianco e un testo nero, accetta i colori come proprietà pubbliche o risolvi dal tema dell'applicazione, e testa un tema scuro oltre a uno chiaro.

Ogni valore disegnato essenziale ha bisogno anche di un gemello semantico. Un indicatore riceve un'etichetta visibile come «CPU 72%» insieme a un `AccessibleDescription`. Una sparkline riceve un testo accessibile del tipo «Andamento: +12% in 7 giorni». Una topologia su Canvas riceve una `TreeView` o una `ListView` di accompagnamento con selezione da tastiera. La gravità espressa come colore ha bisogno di una forma o di una parola accanto, perché il rosso e il verde da soli non arrivano a tutti.

Il capstone Visual Operations Studio

Il capstone è una sola pagina con quattro viste coordinate: il `TelemetryGauge` disegnato per l'asset selezionato, la griglia di operations con le sue colonne Health e Trend disegnate a mano, la topologia su Canvas con selezione, trascinamento, pan e zoom, e un comando Export che produce un PNG tramite `System.Drawing.Managed`. La selezione è condivisa, e l'export è generato dal modello visuale da un renderer `System.Drawing`, mai provando a fare uno screenshot del browser.

Insieme all'applicazione funzionante consegna le prove: dove viene eseguito ciascun renderer e che cosa attraversa la rete, la dimostrazione che nessuno stato di scena per sessione risieda in uno statico valido per l'intero processo, la tua politica di disporre e di gestione dei font, le misure per due percorsi di rendering con un'ottimizzazione che sai difendere, un piano di verifica su Windows più container Linux con le implicazioni per iOS, Android e macOS, e l'alternativa accessibile. Poi percorri la checklist finale di progettazione e rispondi sì a ogni riga, compresi i punti sul se l'elemento visivo si ricostruisce dallo stato del modello, sul se la superficie è adatta alla frequenza di aggiornamento e sul se l'animazione ad alta frequenza resta sul client.



La regola di scegliere prima la superficie

Scegli la superficie di rendering prima di scrivere una sola chiamata di disegno. Chiediti dove vengono prodotti i pixel e che cosa attraversa la rete: l'handler di Paint di un controllo gira sul server e spedisce un'immagine, `CellPaint` fa lo stesso dentro una singola cella di griglia, il Canvas invia comandi di disegno che il browser esegue e non conserva nulla attraverso un resize, e `System.Drawing.Managed` produce byte di immagine senza alcun controllo. Tieni il modello semantico in oggetti C#, rendi ogni renderer una funzione deterministica di quel modello, invalida solo quando è cambiata una dipendenza visuale e lascia sempre un equivalente testuale di tutto ciò che è importante ed esiste solo come pixel.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Costruire un modello visuale condiviso con renderer System.Drawing e Canvas separati.
- ✓ Fare il profiling di CPU del disegno, allocazioni, dimensioni delle immagini, scorrimento della griglia e comportamento delle sessioni.
- ✓ Testare il rendering multiplatforma e il comportamento del redraw nel browser.
- ✓ Progettare accessibilità, export, temi, layout responsive e gestione degli errori dentro gli elementi visivi personalizzati.
- ✓ Completare il capstone Visual Operations Studio.

Guida al lab / esame

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



Lab pratico

Completa il capstone di VisualOperationsStudio mettendo un unico modello dietro tutte e quattro le superfici. Ristruttura il progetto in cartelle Models, Geometry, Renderers e adapter Wisej, così che l'indicatore disegnato, le celle di griglia disegnate a mano, la topologia su Canvas e l'export PNG leggano tutti lo stesso OperationsModel e condividano il livello di geometria puro, senza stato di business duplicato per renderer. Aggiungi un helper RenderMetrics che registra la durata del rendering, la dimensione del bitmap prodotto e il numero di oggetti della scena disegnati, e usalo per confrontare l'indicatore disegnato sul server con il renderer off-screen prima e dopo un'ottimizzazione che scegli e motivi. Metti alla prova i percorsi di errore — dati vuoti, superficie a dimensione zero, intervallo non valido, font mancante — così che ciascuno degradi in un segnaposto leggibile invece che in un'eccezione, aggiungi una tabella accessibile che elenca ogni valore mostrato dalla grafica e scrivi le note del capstone su architettura, confine browser/server e matrice di test.

Punto di partenza

VisualOperationsStudio come l'hai lasciato dopo il Modulo 6: un controllo `TelemetryGauge` con i propri calcoli dell'indicatore, una griglia di operations con le colonne Health e Trend in `UserPaint`, un editor di topologia su Canvas che tiene una scena per sessione con hit testing, pan e zoom, e un `TopologyImageRenderer` che scrive un PNG tramite `Graphics.FromImage`. Ogni superficie funziona, ma ciascuna conserva la propria copia di soglie, colori e posizioni dei nodi, nulla è misurato, e un font mancante o un insieme di dati vuoto arrivano ancora al codice di disegno come eccezione.

Approccio suggerito

- ✓ Crea prima le cartelle `Models`, `Geometry`, `Renderers` e adapter, poi sposta una superficie alla volta su un unico `OperationsModel` che possiede gli asset, le loro letture, i nodi e gli archi della topologia, le soglie di gravità e la selezione condivisa. Elimina via via la copia privata di ciascun renderer, e tieni lo stato della scena sulla sessione invece che in un campo statico.
- ✓ Solleva i calcoli delle coordinate dai quattro renderer verso il livello di geometria, così che riceva lo stato del modello più una dimensione e restituisca i limiti dell'anello, gli angoli di apertura, i rettangoli delle celle, i rettangoli dei nodi e le posizioni delle etichette. Lì, dove non serve alcun controllo Wisej, scrivi unit test per la mappatura valore-angolo, la normalizzazione di un intervallo invertito, l'hit testing e il culling della regione visibile.
- ✓ Scrivi `RenderMetrics` con un metodo `Record` che riceve il nome della superficie, il tempo trascorso dello `Stopwatch`, larghezza e altezza dell'output e il numero di oggetti disegnati, poi chiamalo dall'adapter di `Paint` dell'indicatore e dal renderer off-screen. Cattura una baseline, applica un'ottimizzazione che sai difendere e registra di nuovo gli stessi numeri.
- ✓ Aggiungi i percorsi degradati: restituisci un elemento visivo neutro «Nessun dato» quando il modello è vuoto, esci immediatamente su una superficie a dimensione zero, limita o rifiuta un intervallo non valido prima di disegnare, e risolvi i font attraverso una catena di fallback così che

una famiglia mancante produca un segnaposto leggibile invece di un'eccezione. Racchiudi l'export in `try / catch` così che un errore raggiunga l'utente come messaggio invece che come immagine vuota.

- ✓ Costruisci la tabella accessibile che elenca ogni valore mostrato dalla grafica, dai all'indicatore una lettura visibile più un `AccessibleDescription`, poi scrivi `CapstoneNotes.md` con il diagramma dei livelli, dove viene eseguito ciascun renderer e che cosa attraversa la rete, la matrice di test per Windows, i container Linux e i target Hybrid, e le misure prima e dopo.

Prima di considerare finito il capstone, esegui lo script della demo dall'inizio alla fine: cambia un valore di telemetria e osserva tutte e quattro le superfici concordare, ridimensiona o ricrea il Canvas e mostra che `Redraw` lo ricostruisce, scorri la griglia e spiega il compromesso fra disegno a mano e HTML, trascina e zooma un nodo della topologia e spiega l'hit testing con le coordinate inverse, esporta un PNG e di' ad alta voce dove sta girando `System.Drawing.Managed`, poi presenta le misure e la soglia di ottimizzazione che hai scelto. La nota sull'architettura deve anche un'alternativa scartata per ciascuna superficie, quindi preparati a dire perché l'indicatore è disegnato invece che a tema, perché la colonna della griglia è disegnata a mano invece che HTML, perché la topologia è su Canvas invece che un'immagine disegnata sul server, e perché l'export è off-screen invece che uno screenshot del browser.

Deliverable richiesti

- ✓ Livelli Models, Geometry, Renderers e adapter con un solo `OperationsModel` che guida tutte e quattro le superfici
- ✓ Helper `RenderMetrics` che registra durata del rendering, dimensione del bitmap e numero di oggetti disegnati per ciascuna superficie
- ✓ Misure prima e dopo per un'ottimizzazione, con la motivazione della scelta
- ✓ Percorsi di errore per dati vuoti, dimensione zero, intervallo non valido e font mancante che degradano invece di sollevare eccezioni
- ✓ `CapstoneNotes.md` con l'architettura, il confine server/browser, la matrice di test e una tabella di dati accessibile

Domande di autovalutazione

- ✓ Il tuo livello di geometria riceve il modello e una dimensione e restituisce forme, ma un membro del team vorrebbe che restituisse un `GraphicsPath` così che il renderer dell'indicatore si accorci. Che cosa costa questo al renderer Canvas e agli unit test, e dove tratteresti invece il confine?
- ✓ Il PNG esportato e il Canvas a schermo differiscono di qualche pixel sull'agente di build Linux ma coincidono su Windows. Quali misure e quale tipo di test ti direbbero se si tratta di un difetto o di una differenza di piattaforma attesa?
- ✓ Chi rivede il tuo lavoro dice che la tabella accessibile è ridondante perché ogni indicatore ha già un tooltip. Spiega che cosa dà la tabella a chi usa solo la tastiera o uno screen reader e che il tooltip non dà, e che cosa implica per i chip di gravità della griglia basati solo sul colore.



La regola di scegliere prima la superficie

Scegli la superficie di rendering prima di scrivere una sola chiamata di disegno. Chiediti dove vengono prodotti i pixel e che cosa attraversa la rete: l'handler di Paint di un controllo gira sul server e spedisce un'immagine, CellPaint fa lo stesso dentro una singola cella di griglia, il Canvas invia comandi di disegno che il browser esegue e non conserva nulla attraverso un resize, e System.Drawing.Managed produce byte di immagine senza alcun controllo. Tieni il modello semantico in oggetti C#, rendi ogni renderer una funzione deterministica di quel modello, invalida solo quando è cambiata una dipendenza visuale e lascia sempre un equivalente testuale di tutto ciò che è importante ed esiste solo come pixel.