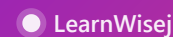


Temi, risorse, localizzazione e modernizzazione della UI

Modernizza le applicazioni Wisej.NET usando temi, CSSStyle, risorse, icone, il cambio di tema a runtime, la localizzazione e regole di coerenza visiva.



Obiettivo della lezione

Modernizzare le applicazioni Wisej.NET usando temi, CSSStyle, risorse, icone, cambio del tema a runtime, localizzazione e regole di coerenza visiva — separando l'*aspetto* (tema/CSS) dal *comportamento* (C#), così che un redesign o una nuova lingua non significhino mai modificare ogni schermata.

Tutto in questo modulo poggia su una disciplina: **aspetto e logica sono lavori diversi e vanno in posti diversi**. Il tuo C# decide *cosa fa l'app*; un tema decide *che aspetto ha l'app*; le risorse contengono le *stringhe e le immagini* che l'app mostra. Quando queste tre cose restano separate, puoi ridisegnare lo stile del prodotto, aggiungere una modalità scura o rilasciare una build in italiano senza toccare un solo gestore di eventi. Quando si mescolano — un colore esadecimale digitato in un controllo qui, una didascalia scritta nel codice là — ogni modifica diventa una caccia in tutto il codice. Questa lezione riguarda il tenerle separate di proposito.

1. Separare l'aspetto dal comportamento

In Wisej.NET l'*aspetto* di una schermata è un dato, non codice. Un **tema** è un documento JSON che definisce colori, font, spaziature, bordi e l'aspetto di ogni tipo di controllo — pulsanti, griglie, pannelli, schede. Il tuo C# non nomina mai `#0d6efd` o `font-size: 14px`; crea semplicemente un `Button`, e il tema attivo decide come disegnarlo. Cambia il tema e l'intera applicazione cambia veste, perché i controlli non hanno mai portato con sé i propri colori.

Livello	Di cosa è responsabile	Dove vive
Tema (JSON)	L'aspetto globale — la palette dei colori, i font, la scala delle spaziature e l'aspetto predefinito di ogni tipo di controllo.	Un file <code>.theme</code> / JSON del tema, oppure Theme Builder.
CSSStyle	Un override visivo mirato applicato a un controllo o a una piccola famiglia di controlli.	La proprietà <code>CssStyle</code> del controllo (C# o Designer).
Comportamento (C#)	Ciò che l'app <i>fa</i> — validazione, flussi di lavoro, dati, navigazione.	I tuoi gestori di eventi e servizi.
Risorse	Le stringhe e le immagini che l'app mostra — didascalie, messaggi, icone, loghi.	File di risorse / la cartella Resources.

Il vantaggio di centralizzare colori e spaziature in un tema invece di dare stile a ogni controllo è la leva. Supponi che il colore del tuo brand passi dal blu al verde acqua. Con un tema modifichi un valore e ogni pulsante, link ed evidenziazione si aggiorna insieme. Con lo stile per controllo, la stessa modifica diventa un cerca-e-sostituisci su decine di file — e *ne* dimenticherai uno. La centralizzazione impone anche una coerenza a cui non hai dovuto pensare: poiché ogni controllo legge le spaziature dalla stessa scala, le schermate si allineano senza che nessuno controlli i pixel a occhio.



La regola in parole semplici

Se un valore visivo descrive *l'intera app* (il blu del brand, il margine standard), appartiene al tema. Se descrive *un controllo insolito*, appartiene al CSSStyle di quel controllo. Non dovrebbe quasi mai vivere come letterale in C#.

2. Temi e Theme Builder

Wisej.NET include una serie di temi integrati e fornisce **Theme Builder**, un editor visuale per crearli o ritocarli. Parti da un tema base — per esempio un default chiaro o una variante scura come **BootstrapDark-4** — e regoli palette, tipografia e aspetto dei controlli, poi salvi il risultato come tema della tua applicazione. Poiché Theme Builder modifica lo stesso JSON che legge il runtime, non c'è un passaggio di "esportazione" separato in cui sbagliare: ciò che progetti è ciò che l'app carica.

Vale la pena dare un nome ad alcuni concetti. Un tema definisce **colori con nome** (una palette), così gli aspetti fanno riferimento a `"highlight"` o `"error"` invece che a valori esadecimali grezzi, e una modifica alla palette si propaga ovunque. Gli **appearance** sono le definizioni di stile per tipo di controllo. E i **mixin** ti permettono di sovrapporre un piccolo insieme di override a un tema base invece di duplicarlo per intero — lo strumento giusto quando vuoi "il tema standard, ma con il nostro colore e il nostro logo".

3. Cambiare tema a runtime

Poiché il tema è solo l'aspetto attivo, puoi cambiarlo mentre l'app è in esecuzione — per un interruttore della modalità scura per utente, una modalità ad alto contrasto per l'accessibilità o una veste specifica per tenant. Imposti il tema attivo dell'applicazione e Wisej.NET ridisegna le schermate aperte con i nuovi appearance. Non ricostruisci controlli né ricarichi dati; cambia solo la pittura.

Un interruttore della modalità scura che sostituisce il tema attivo a runtime

```
private void btnDarkMode_Click(object sender, EventArgs e)
{
    // Sceglie il tema per nome; il runtime ridisegna le schermate aperte.
    // Il comportamento non cambia - cambia solo l'aspetto attivo.
    var next = _darkMode ? "Default-4" : "BootstrapDark-4";
    Application.Theme = next;           // cambia l'aspetto di tutta l'app
    _darkMode = !_darkMode;

    // Salva la scelta così sopravvive al login successivo.
    _userPrefs.Save("theme", next);
}
```

Scegliere il tema all'avvio, prima che la prima schermata venga disegnata

```
protected override void OnSessionStart(EventArgs e)
{
    // Legge la preferenza salvata e imposta l'aspetto una volta, all'inizio,
    // così l'utente non vede mai un lampo del tema sbagliato.
    var saved = _userPrefs.Load("theme") ?? "Default-4";
    Application.Theme = saved;
    base.OnSessionStart(e);
}
```

Nota cosa il gestore *non* fa: niente colori letterali, niente restyling per controllo, niente ricaricamento dei dati. L'aspetto è un dato con un nome; cambiarlo è una modifica di una riga. (La proprietà/API esatta per impostare il tema attivo può variare con la versione di Wisej.NET — il meccanismo è lo stesso: assegni per nome il tema attivo dell'applicazione e il runtime ridisegna la UI.)

4. CSSStyle — l'eccezione mirata

A volte un controllo ha davvero bisogno di apparire diverso dai suoi simili in un modo che il tema non dovrebbe codificare globalmente — un banner di avviso con uno sfondo distinto, una card con un'ombra morbida. È a questo che serve la proprietà **CSSStyle** del controllo: associa del CSS a quel widget specifico senza inquinare il tema e senza dover mettere tu le mani nel DOM del browser.

Un override CSSStyle su un singolo pannello di stato

```
pnlBanner.CSSStyle =
    "background: var(--wisej-warning, #fff3cd);" + // preferisci una variabile del tema a
    un esadecimale grezzo
    "border-radius: 8px;" +
    "padding: 12px 16px;";

lblBanner.Text = "Your session expires in 5 minutes.";
```

Usa CSSStyle con parsimonia e consapevolezza. Ogni stringa CSSStyle è un pezzetto di aspetto che vive *fuori* dal tema, quindi non seguirà automaticamente un redesign. L'abitudine da costruire: ricorri prima al tema e scendi a CSSStyle solo quando un controllo è davvero un caso unico. Se ti ritrovi a incollare lo stesso CSSStyle su molti controlli, è il segnale che appartiene invece al tema.

Tema, CSSStyle o proprietà inline — quando usare ciascuno

Strumento	Ambito	Usalo quando...
Tema (JSON / Theme Builder)	L'intera applicazione, tutti i controlli di un tipo.	Il valore fa parte dell'identità del prodotto — palette, font, scala delle spaziature, aspetto predefinito dei controlli. Prima scelta per tutto ciò che è condiviso.
Proprietà CSSStyle	Un controllo, o una piccola famiglia voluta.	Una vera eccezione visiva che il tema non dovrebbe portare — un banner speciale, una card evidenziata.
Proprietà inline (BackColor, Font...)	Un controllo, una proprietà.	Raramente — un ritocco davvero locale e guidato dai dati (es. una riga diventa rossa perché un valore è scaduto). Mai per i colori del brand.

5. Risorse — immagini, icone e stringhe

L'altra metà del "separare l'aspetto dalla logica" è trattare il *contenuto* che l'app mostra — didascalie, messaggi, loghi e icone — come **risorse** invece che come letterali sparsi nel codice. Immagini e icone vanno nella cartella Resources e sono referenziate per nome, così un designer può sostituire il logo senza ricompilare i tuoi gestori. Le stringhe visibili all'utente vanno in file di risorse indicizzati per nome, ed è proprio questo a rendere possibile la localizzazione: la stessa chiave si risolve in un testo diverso per ogni cultura.

Caricare un messaggio localizzato e un'icona dalle risorse invece di scriverli nel codice

```
// La stringa arriva da una risorsa indicizzata per nome - non da un letterale.  
// A runtime si risolve nella voce per la cultura corrente.  
lblWelcome.Text = Resources.Strings.Welcome;          // "Welcome" / "Benvenuto" / "Willkommen"  
  
// L'immagine arriva da una risorsa con nome, non da un percorso di file improvvisato.  
btnSave.ImageSource = "resource.wx/App.Resources.SaveIcon.png";  
  
// Un intero Form/Page può essere marcato Localizable così il Designer  
// memorizza automaticamente le sue didascalie per ogni cultura.
```

La forma generale è sempre la stessa: una **chiave** stabile nel codice, risolta a runtime nel contenuto reale in base alla cultura attiva. Tutto il trucco sta in questa indirezione — il codice fa riferimento alle chiavi, e il significato di una chiave può cambiare per lingua, per brand o per stagione senza che il codice lo sappia.

6. Localizzazione — cultura, stringhe e formattazione

Localizzare non è solo tradurre. Due cose variano in base alla **cultura**: le *stringhe* che mostri e la *formattazione* di date, numeri e valute. Wisej.NET localizza il testo visibile all'utente risolvendo le chiavi delle risorse per la cultura corrente (Form e Page possono essere marcati **Localizable** così il Designer gestisce le loro didascalie per ogni lingua). La formattazione segue la cultura del thread, quindi una data appare come **7/3/2026** per un utente e **03/07/2026** per un altro — dallo stesso C#, perché hai formattato con la cultura invece che con un pattern fisso.

Impostare la cultura dell'utente, poi lasciare che la formattazione la segua

```
// Imposta la cultura per la sessione di questo utente (da una preferenza o un selettore).
var culture = new CultureInfo(_userPrefs.Load("culture") ?? "en-US");
System.Threading.Thread.CurrentThread.CurrentUICulture = culture; // quali stringhe
System.Threading.Thread.CurrentThread.CurrentCulture = culture; // come si formattano
date/numeri

// Ora lo STESSO codice formatta correttamente per ogni utente:
lblTotal.Text = amount.ToString("C"); // $1,234.50 vs 1.234,50 €
lblDue.Text = dueDate.ToShortDateString(); // sensibile alla cultura, nessun pattern fisso
```

Un avvertimento pratico: il testo tradotto cambia lunghezza. Le etichette in tedesco sono spesso più lunghe di quelle in inglese; alcuni sistemi di scrittura sono più alti. Un layout che sta perfettamente in inglese può tagliare o andare a capo male in un'altra lingua, quindi **testa la localizzazione nei layout reali delle schermate**, non solo in un foglio di calcolo di stringhe. Per questo il modulo tratta la localizzazione come una questione di UI, non di testo.



Regola di produzione

Prima il tema, poi le eccezioni. Tieni ogni stringa e immagine visibile all'utente come risorsa, imposta la cultura una volta per sessione e formatta con la cultura invece che con un pattern fisso — poi verifica il risultato nella schermata reale per ogni lingua che rilasci.



Obiettivi di apprendimento

Al termine di questo modulo sarai in grado di:

- ✓ Applicare i temi integrati e cambiare tema a runtime o all'avvio senza modificare il comportamento.
- ✓ Usare Theme Builder e CSSStyle senza spargere ovunque stili a tantum.
- ✓ Centralizzare colori, spaziature, stringhe e immagini così che un redesign modifichi un solo punto, non ogni schermata.
- ✓ Localizzare Form, Page, risorse di sistema e formattazione dei dati dove opportuno.
- ✓ Creare una checklist di modernizzazione della UI per le schermate enterprise.

7. Una UI moderna, in breve

Modernizzare non significa solo applicare temi. Una schermata enterprise moderna usa una **scala di spaziature** coerente, una chiara **gerarchia tipografica**, un **contrasto** accessibile, **icone** riconoscibili e **schemi di stato e di errore** prevedibili, così gli utenti imparano un unico linguaggio visivo e lo riutilizzano ovunque. Il motivo per cui tutto questo appartiene al tema è lo stesso di prima: la coerenza che ottieni gratis è coerenza che non puoi rompere per sbaglio, e un redesign diventa una modifica di dati invece di un progetto.

Errori comuni

Errore	Perché fa danni	Cosa fare invece
Colori e spaziature scritti nel codice e sparsi tra le schermate.	Un cambio di brand diventa un cerca-e-sostituisci su decine di file — e ne dimenticherai uno.	Definisci palette e spaziature nel tema; fai riferimento a valori con nome, non a esadecimali.
Incollare lo stesso CSSStyle su molti controlli.	Lo stile vive fuori dal tema, quindi non segue un redesign e perde la sincronia.	Promuovi lo stile ripetuto nell'appearance del tema per quel tipo di controllo.
Stringhe visibili all'utente scritte come letterali in C#.	Non si possono tradurre e una correzione del testo richiede di ricompilare il codice.	Tieni le stringhe in risorse indicizzate per nome; risolvi per cultura.
Immagini incorporate alla buona da percorsi di file.	Le risorse si disperdono, si rompono al rilascio e non si possono sostituire senza una rebuild.	Aggiungile a Resources e referenziale per nome.
Ignorare la cultura — pattern fissi per date e numeri.	Date, decimali e valute appaiono sbagliati per le altre aree geografiche.	Imposta la cultura della sessione e formatta con quella (<code>ToString("C")</code> , data breve).
Testare le traduzioni solo come elenco di stringhe.	Il testo più lungo viene tagliato o va a capo; i layout si rompono nella schermata reale.	Testa ogni lingua nel layout reale prima del rilascio.

Vocabolario chiave

- **Tema** — il documento JSON che definisce l'aspetto dell'intera app: palette, font, spaziature e appearance per controllo.
- **Theme Builder** — l'editor visuale per creare e ritoccare i temi.
- **Appearance** — la definizione di stile di un tema per un tipo di controllo.
- **Mixin** — un piccolo strato di override sovrapposto a un tema base invece di duplicarlo.
- **CSSStyle** — la proprietà di un controllo per un override CSS mirato su quel widget specifico.
- **Risorsa** — una stringa o un'immagine con nome a cui l'app fa riferimento al posto di un letterale, così da poterla sostituire e tradurre.
- **Proprietà Localizable** — un Form/Page (o controllo) marcato in modo che il Designer memorizzi le sue didascalie per ogni cultura.
- **Cultura** — l'impostazione che determina sia quali stringhe vengono mostrate sia come si formattano date, numeri e valute.
- **Cambio del tema a runtime** — cambiare il tema attivo mentre l'app è in esecuzione, ridisegnando le schermate aperte senza ricostruirle.

Errore comune da evitare



Errore comune

Spargere colori scritti nel codice, CSS una tantum e stringhe letterali su molte schermate. Funziona in una demo e marcisce in produzione — un restyling o una nuova lingua significano allora modificare ogni file invece di un solo tema e un solo set di risorse. È esattamente il difetto che eliminerai nel lab di questo modulo.

Checklist di revisione del designer e del codice

- ✓ La schermata resta utilizzabile nel Designer di Visual Studio / Wisej.NET.
- ✓ I controlli hanno nomi abbastanza chiari perché un collega possa seguire il codice degli eventi.
- ✓ Colori e spaziature arrivano dal tema, non da letterali in C# o da CSSStyle ripetuti.
- ✓ Le stringhe e le immagini visibili all'utente sono risorse, non valori scritti nel codice.
- ✓ La cultura è impostata per sessione e la formattazione la segue.
- ✓ La logica di business non è intrappolata nei gestori di eventi visivi.
- ✓ I percorsi di errore sono visibili, registrati nei log e spiegati senza esporre dettagli interni.
- ✓ Il deliverable si può revisionare senza eseguire l'intero corso.



Autoverifica prima del lab

Quali stili sono a livello di tema e quali sono vere eccezioni a livello di schermata? Riesci a cambiare l'aspetto dell'intera app con una riga senza toccare un gestore? Ogni stringa e immagine visibile all'utente è una risorsa? Cosa cambia con un testo in tedesco o in italiano, e il layout regge? Un futuro redesign può evitare di modificare ogni schermata? Se la risposta è sì a tutto, sei pronto.

Codice di accompagnamento

`ThemingLocalizationPatterns.cs` raccoglie i pattern di questo modulo in un unico file compilabile. Scaricalo e leggilo insieme al lab.

↓ [Scarica ThemingLocalizationPatterns.cs](#)