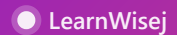


Theming, Ressourcen, Lokalisierung und UI-Modernisierung

Modernisieren Sie Wisej.NET-Anwendungen mit Themes, CSSStyle, Ressourcen, Symbolen, Theme-Wechsel zur Laufzeit, Lokalisierung und Regeln für visuelle Konsistenz.



Lernziel

Modernisieren Sie Wisej.NET-Anwendungen mit Themes, CSSStyle, Ressourcen, Icons, Theme-Wechsel zur Laufzeit, Lokalisierung und Regeln für visuelle Konsistenz – und trennen Sie dabei das *Aussehen* (Theme/CSS) vom *Verhalten* (C#), damit ein Redesign oder eine neue Sprache nie bedeutet, jeden Bildschirm zu bearbeiten.

Alles in diesem Modul beruht auf einer Disziplin: **Darstellung und Logik sind verschiedene Aufgaben und gehören an verschiedene Orte.** Ihr C#-Code entscheidet, *was die App tut*; ein Theme entscheidet, *wie die App aussieht*; Ressourcen enthalten die *Texte und Bilder*, die die App anzeigt. Bleiben diese drei getrennt, können Sie das Produkt neu gestalten, einen Dark Mode ergänzen oder einen italienischen Build ausliefern, ohne einen einzigen Event-Handler anzufassen. Vermischen sie sich – hier ein Hex-Farbwert direkt in einem Control, dort eine fest codierte Beschriftung –, wird jede Änderung zur Suche quer durch die gesamte Codebasis. In dieser Lektion geht es darum, sie ganz bewusst auseinanderzuhalten.

1. Darstellung vom Verhalten trennen

In Wisej.NET ist das *Aussehen* einer Maske Daten, kein Code. Ein **Theme** ist ein JSON-Dokument, das Farben, Schriften, Abstände, Rahmen und das Erscheinungsbild jedes Control-Typs festlegt – Buttons, Grids, Panels, Tabs. Ihr C#-Code erwähnt nie `#0d6efd` oder `font-size: 14px`; er erzeugt einfach einen `Button`, und das aktive Theme entscheidet, wie dieser Button gezeichnet wird. Wechseln Sie das Theme, erhält die ganze Anwendung ein neues Erscheinungsbild, weil die Controls ihre Farben nie selbst getragen haben.

Schicht	Was sie verantwortet	Liegt in
Theme (JSON)	Das globale Aussehen – Farbpalette, Schriften, Abstandsskala und das Standard-Erscheinungsbild jedes Control-Typs.	Einer <code>.theme</code> - bzw. Theme-JSON-Datei oder im Theme Builder.
CSSStyle	Eine gezielte visuelle Überschreibung für ein Control oder eine kleine Gruppe von Controls.	Der Eigenschaft <code>CssStyle</code> des Controls (C# oder Designer).
Verhalten (C#)	Was die App <i>tut</i> – Validierung, Workflows, Daten, Navigation.	Ihren Event-Handleern und Services.
Ressourcen	Die Texte und Bilder, die die App anzeigt – Beschriftungen, Meldungen, Icons, Logos.	Ressourcendateien / dem Ordner Resources.

Der Gewinn, Farben und Abstände zentral im Theme festzulegen, statt jedes Control einzeln zu gestalten, ist Hebelwirkung. Angenommen, Ihre Markenfarbe wechselt von Blau zu Petrol. Mit einem Theme ändern Sie einen Wert, und alle Buttons, Links und Hervorhebungen passen sich gemeinsam an. Mit Styling pro Control ist dieselbe Änderung ein Suchen-und-Ersetzen über Dutzende Dateien – und Sie *werden* eine übersehen. Zentralisierung erzwingt außerdem Konsistenz, über die Sie nicht nachdenken müssen: Weil jedes Control seine Abstände aus derselben Skala liest, sind Masken sauber ausgerichtet, ohne dass jemand Pixel nach Augenmaß schieben muss.



Die Regel in einfachen Worten

Beschreibt ein visueller Wert Ihre *gesamte App* (das Markenblau, den Standardabstand), gehört er ins Theme. Beschreibt er *ein ungewöhnliches Control*, gehört er in den CSSStyle dieses Controls. Als Literal in C# sollte er so gut wie nie vorkommen.

2. Themes und Theme Builder

Wisej.NET wird mit einer Reihe integrierter Themes ausgeliefert und stellt den **Theme Builder** bereit, einen visuellen Editor zum Erstellen oder Anpassen von Themes. Sie beginnen mit einem Basis-Theme – zum Beispiel einem hellen Standard-Theme oder einer dunklen Variante wie **BootstrapDark-4** –, passen Palette, Typografie und das Erscheinungsbild der Controls an und speichern das Ergebnis als Theme Ihrer Anwendung. Weil der Theme Builder dasselbe JSON bearbeitet, das die Laufzeit liest, gibt es keinen separaten „Export“-Schritt, bei dem etwas schiefgehen kann: Was Sie gestalten, lädt die App.

Einige Konzepte verdienen einen Namen. Ein Theme definiert **benannte Farben** (eine Palette), sodass Erscheinungsbilder auf `"highlight"` oder `"error"` statt auf rohe Hex-Werte verweisen – eine Änderung an der Palette wirkt sich also überall aus. **Appearances** sind die Stildefinitionen pro Control-Typ. Und **Mixins** erlauben es, eine kleine Menge von Überschreibungen über ein Basis-Theme zu legen, statt das ganze Theme zu forken – das richtige Werkzeug, wenn Sie „das Standard-Theme, aber mit unserer Markenfarbe und unserem Logo“ möchten.

3. Das Theme zur Laufzeit wechseln

Weil das Theme nur das aktive Aussehen ist, können Sie es ändern, während die App läuft – für einen Dark-Mode-Umschalter pro Benutzer, einen barrierefreien Hochkontrastmodus oder ein mandantenspezifisches Erscheinungsbild. Sie setzen das aktive Theme der Anwendung, und Wisej.NET rendert die offenen Masken mit den neuen Appearances neu. Sie bauen keine Controls neu auf und laden keine Daten neu; nur die Darstellung ändert sich.

Ein Dark-Mode-Umschalter, der das aktive Theme zur Laufzeit wechselt

```
private void btnDarkMode_Click(object sender, EventArgs e)
{
    // Das Theme über seinen Namen wählen; die Laufzeit gestaltet offene Masken neu.
    // Das Verhalten ändert sich nicht – nur das aktive Aussehen.
    var next = _darkMode ? "Default-4" : "BootstrapDark-4";
    Application.Theme = next;           // das Aussehen der ganzen App umschalten
    _darkMode = !_darkMode;

    // Die Wahl speichern, damit sie die nächste Anmeldung übersteht.
    _userPrefs.Save("theme", next);
}
```

Das Theme beim Start wählen, bevor die erste Maske gezeichnet wird

```
protected override void OnSessionStart(EventArgs e)
{
    // Die gespeicherte Einstellung lesen und das Aussehen einmal vorab setzen,
    // damit der Benutzer nie kurz das falsche Theme aufblitzen sieht.
    var saved = _userPrefs.Load("theme") ?? "Default-4";
    Application.Theme = saved;
    base.OnSessionStart(e);
}
```

Achten Sie darauf, was der Handler *nicht* tut: keine Farbliterale, kein Umgestalten einzelner Controls, kein Neuladen von Daten. Das Aussehen ist benannte Daten; es zu wechseln, ist eine Änderung in einer Zeile. (Die genaue Eigenschaft bzw. API zum Setzen des aktiven Themes kann je nach Wisej.NET-Version abweichen – der Mechanismus bleibt derselbe: Weisen Sie das aktive Theme der Anwendung über seinen Namen zu, und die Laufzeit gestaltet die UI neu.)

4. CSSStyle – die gezielte Ausnahme

Manchmal muss ein Control wirklich anders aussehen als seine Geschwister, und zwar auf eine Weise, die das Theme nicht global festlegen sollte – ein Warnbanner mit eigenem Hintergrund, eine Karte mit weichem Schatten. Dafür gibt es die Eigenschaft **CSSStyle** des Controls: Sie hängt CSS an genau dieses Widget, ohne das Theme zu verunreinigen oder selbst in das Browser-DOM zu greifen.

```
pnlBanner.CssStyle =
    "background: var(--wisej-warning, #fff3cd);" + // eine Theme-Variable einem rohen Hex-
    Wert vorziehen
    "border-radius: 8px;" +
    "padding: 12px 16px;";

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

Setzen Sie CSSStyle sparsam und bewusst ein. Jeder CSSStyle-String ist ein kleines Stück Darstellung, das *außerhalb* des Themes lebt und einem Redesign daher nicht automatisch folgt. Die Gewohnheit, die Sie entwickeln sollten: zuerst zum Theme greifen und nur dann auf CSSStyle ausweichen, wenn ein Control ein echter Einzelfall ist. Wenn Sie denselben CSSStyle auf viele Controls kopieren, ist das ein Signal, dass er stattdessen ins Theme gehört.

Theme, CSSStyle oder Inline-Eigenschaft – wann Sie was verwenden

Werkzeug	Gültigkeitsbereich	Verwenden, wenn ...
Theme (JSON / Theme Builder)	Die ganze Anwendung, alle Controls eines Typs.	Der Wert zur Identität Ihres Produkts gehört – Palette, Schriften, Abstandsskala, Standard-Aussehen der Controls. Erste Wahl für alles Gemeinsame.
Eigenschaft CSSStyle	Ein Control oder eine kleine, bewusst gewählte Gruppe.	Es sich um eine echte visuelle Ausnahme handelt, die das Theme nicht tragen sollte – ein besonderes Banner, eine hervorgehobene Karte.
Inline-Eigenschaft (BackColor, Font ...)	Ein Control, eine Eigenschaft.	Selten – für eine wirklich lokale, datengetriebene Anpassung (z. B. eine Zeile wird rot, weil ein Wert überfällig ist). Nie für markenweite Farben.

5. Ressourcen – Bilder, Icons und Texte

Die andere Hälfte von „Aussehen und Logik trennen“ besteht darin, die *Inhalte*, die die App anzeigt – Beschriftungen, Meldungen, Logos und Icons –, als **Ressourcen** zu behandeln statt als Literale, die im Code verstreut sind. Bilder und Icons kommen in den Ordner Resources und werden über ihren Namen referenziert, sodass ein Designer das Logo austauschen kann, ohne dass Ihre Handler neu kompiliert werden müssen. Für den Benutzer sichtbare Texte kommen in Ressourcendateien mit benannten Schlüsseln – und genau das ermöglicht Lokalisierung: Derselbe Schlüssel wird je nach Kultur zu einem anderen Text aufgelöst.

Eine lokalisierte Meldung und ein Icon aus Ressourcen laden, statt sie fest zu codieren

```
// Der Text stammt aus einer Ressource mit benanntem Schlüssel – kein Literal.  
// Zur Laufzeit wird er zum Eintrag der aktuellen Kultur aufgelöst.  
lblWelcome.Text = Resources.Strings.Welcome;           // "Welcome" / "Benvenuto" / "Willkommen"  
  
// Das Bild stammt aus einer benannten Ressource, nicht aus einem spontanen Dateipfad.  
btnSave.ImageSource = "resource.wx/App.Resources.SaveIcon.png";  
  
// Ein ganzes Form bzw. eine ganze Page kann als Localizable markiert werden,  
// damit der Designer ihre Beschriftungen automatisch pro Kultur speichert.
```

Die grundlegende Form ist immer dieselbe: ein stabiler **Schlüssel** im Code, der zur Laufzeit abhängig von der aktiven Kultur zu echtem Inhalt aufgelöst wird. Diese Indirektion ist der ganze Trick – der Code referenziert Schlüssel, und die Bedeutung eines Schlüssels kann sich pro Sprache, pro Marke oder pro Saison ändern, ohne dass der Code davon weiß.

6. Lokalisierung – Kultur, Texte und Formatierung

Lokalisierung ist mehr als Übersetzung. Zwei Dinge variieren je nach **Kultur**: die *Texte*, die Sie anzeigen, und die *Formatierung* von Datumswerten, Zahlen und Währungen. Wisej.NET lokalisiert sichtbaren Text, indem es Ressourcenschlüssel für die aktuelle Kultur auflöst (Forms und Pages können als **Localizable** markiert werden, sodass der Designer ihre Beschriftungen pro Sprache verwaltet). Die Formatierung folgt der Kultur des Threads, sodass ein Datum bei einem Benutzer als **7/3/2026** und bei einem anderen als **03/07/2026** erscheint – aus *demselden* C#-Code, weil Sie mit der Kultur statt mit einem festen Muster formatiert haben.

Die Kultur des Benutzers setzen und die Formatierung ihr folgen lassen

```
// Die Kultur für die Session dieses Benutzers setzen (aus einer Einstellung oder einer  
Auswahl).  
var culture = new CultureInfo(_userPrefs.Load("culture") ?? "en-US");  
System.Threading.Thread.CurrentThread.CurrentUICulture = culture; // welche Texte  
System.Threading.Thread.CurrentThread.CurrentCulture = culture; // wie Datumswerte/Zahlen  
formatiert werden  
  
// Jetzt formatiert DERSELBE Code für jeden Benutzer korrekt:  
lblTotal.Text = amount.ToString("C");           // $1,234.50 vs 1.234,50 €  
lblDue.Text = dueDate.ToShortDateString(); // kulturabhängig, kein fest codiertes Muster
```

Eine praktische Warnung: Übersetzter Text ändert seine Länge. Deutsche Beschriftungen sind oft länger als englische; manche Schriftsysteme sind höher. Ein Layout, das auf Englisch perfekt passt, kann in einer anderen Sprache abgeschnitten werden oder unschön umbrechen – **testen Sie Lokalisierung deshalb in echten Bildschirmlayouts**, nicht nur in einer Tabelle mit Texten. Aus diesem Grund behandelt das Modul Lokalisierung als UI-Thema und nicht als Textthema.



Gewohnheit für den Produktivbetrieb

Zuerst das Theme, dann die Ausnahmen. Halten Sie jeden sichtbaren Text und jedes Bild als Ressource, setzen Sie die Kultur einmal pro Session, und formatieren Sie mit der Kultur statt mit einem festen Muster – prüfen Sie das Ergebnis anschließend in der tatsächlichen Maske für jede Sprache, die Sie ausliefern.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ integrierte Themes anwenden und Themes zur Laufzeit oder beim Start wechseln, ohne das Verhalten zu ändern.
- ✓ Theme Builder und CSSStyle nutzen, ohne einmalige Stylings überall zu verstreuen.
- ✓ Farben, Abstände, Texte und Bilder zentralisieren, sodass ein Redesign eine Stelle ändert und nicht jeden Bildschirm.
- ✓ Forms, Pages, Systemressourcen und die Datenformatierung lokalisieren, wo es sinnvoll ist.
- ✓ eine Checkliste zur UI-Modernisierung für Unternehmensanwendungen erstellen.

7. Eine moderne UI in Kürze

Modernisieren heißt nicht nur Theming. Eine moderne Unternehmensmaske verwendet eine einheitliche **Abstandsskala**, eine klare **typografische Hierarchie**, barrierefreien **Kontrast**, wiedererkennbare **Icons** und vorhersagbare **Muster für Status und Fehler**, sodass Benutzer eine visuelle Sprache lernen und sie überall wiederverwenden. Der Grund, warum all das ins Theme gehört, ist derselbe wie zuvor: Konsistenz, die Sie geschenkt bekommen, können Sie nicht versehentlich zerstören, und ein Redesign wird zu einer Datenänderung statt zu einem Projekt.

Typische Fehler

Fehler	Warum es schadet	Besser so
Fest codierte Farben und Abstände, über viele Masken verstreut.	Eine Markenänderung wird zum Suchen- und-Ersetzen über Dutzende Dateien – und Sie werden eine übersehen.	Palette und Abstände im Theme definieren; benannte Werte referenzieren, keine Hex-Werte.
Denselben CSSStyle auf viele Controls kopieren.	Der Stil lebt außerhalb des Themes, folgt also keinem Redesign und gerät aus dem Takt.	Den wiederholten Stil in die Appearance des Themes für diesen Control-Typ übernehmen.
Sichtbare Texte als Literale in C#.	Sie lassen sich nicht übersetzen, und eine Textkorrektur bedeutet, Code neu zu kompilieren.	Texte in Ressourcen mit benannten Schlüsseln halten; pro Kultur auflösen.
Bilder spontan über Dateipfade eingebunden.	Assets verstreuen sich, brechen beim Deployment und lassen sich nicht ohne neuen Build austauschen.	Sie zu den Ressourcen hinzufügen und über ihren Namen referenzieren.
Kultur ignorieren – feste Muster für Datum und Zahlen.	Datumswerte, Dezimalzahlen und Währungen erscheinen in anderen Regionen falsch.	Die Kultur der Session setzen und damit formatieren (<code>ToString("C")</code> , kurzes Datumsformat).
Übersetzungen nur als Textliste testen.	Längerer Text wird abgeschnitten oder bricht um; Layouts gehen in der echten Maske kaputt.	Jede Sprache vor der Auslieferung im tatsächlichen Layout testen.

Wichtige Begriffe

- **Theme** – das JSON-Dokument, das das Aussehen der ganzen App festlegt: Palette, Schriften, Abstände und Appearances pro Control.
- **Theme Builder** – der visuelle Editor zum Erstellen und Anpassen von Themes.
- **Appearance** – die Stildefinition eines Themes für einen Control-Typ.
- **Mixin** – eine kleine Schicht von Überschreibungen über einem Basis-Theme, statt es zu forken.
- **CSSStyle** – die Eigenschaft eines Controls für eine gezielte CSS-Überschreibung an genau diesem Widget.
- **Ressource** – ein benannter Text oder ein benanntes Bild, das die App statt eines Literals referenziert; ermöglicht Austausch und Übersetzung.
- **Eigenschaft Localizable** – ein Form bzw. eine Page (oder ein Control), so markiert, dass der Designer die Beschriftungen pro Kultur speichert.
- **Kultur** – die Einstellung, die sowohl bestimmt, welche Texte erscheinen, als auch, wie Datumswerte, Zahlen und Währungen formatiert werden.
- **Theme-Wechsel zur Laufzeit** – das aktive Theme ändern, während die App läuft, und offene Masken neu gestalten, ohne sie neu aufzubauen.

Der typische Fehler, den Sie vermeiden sollten



Typischer Fehler

Fest codierte Farben, einmaliges CSS und literale Texte über viele Masken verstreuen. In einer Demo funktioniert das, im Produktivbetrieb verrottet es – eine Umgestaltung oder eine neue Sprache bedeutet dann, jede Datei zu bearbeiten statt eines Themes und eines Ressourcen-Sets. Genau diesen Code-Smell beseitigen Sie im Lab dieses Moduls.

Checkliste für Designer und Code-Review

- ✓ Der Bildschirm bleibt im Visual Studio / Wisej.NET Designer nutzbar.
- ✓ Controls sind so klar benannt, dass ein Teammitglied dem Event-Code folgen kann.
- ✓ Farben und Abstände stammen aus dem Theme, nicht aus Literalen in C# oder wiederholtem CSSStyle.
- ✓ Sichtbare Texte und Bilder sind Ressourcen, keine fest codierten Werte.
- ✓ Die Kultur wird pro Session gesetzt, und die Formatierung folgt ihr.
- ✓ Die Geschäftslogik steckt nicht in visuellen Event-Handlern fest.
- ✓ Fehlerpfade sind sichtbar, werden geloggt und erklärt, ohne interne Details preiszugeben.
- ✓ Das Ergebnis lässt sich reviewen, ohne den ganzen Kurs auszuführen.



Selbstcheck vor dem Lab

Welche Stile gehören auf Theme-Ebene, und welche sind echte Ausnahmen auf Ebene einer Maske? Können Sie das Aussehen der ganzen App in einer Zeile wechseln, ohne einen Handler anzufassen? Ist jeder sichtbare Text und jedes Bild eine Ressource? Was ändert sich bei deutschem oder italienischem Text, und übersteht das Layout es? Kann ein künftiges Redesign vermeiden, jeden Bildschirm zu bearbeiten? Wenn Sie alle Fragen mit Ja beantworten, sind Sie bereit.

Begleitcode

`ThemingLocalizationPatterns.cs` fasst die Muster dieses Moduls in einer kompilierbaren Datei zusammen. Laden Sie sie herunter und lesen Sie sie parallel zum Lab.

↓ [ThemingLocalizationPatterns.cs herunterladen](#)