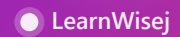


CSS, CssClass, CssStyle, Zustände und Theme-Wechsel zur Laufzeit

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Gezieltes CSS sicher mit `CssClass` und einem bereichsbezogenen Anwendungs-Stylesheet hinzufügen, `CssStyle` für einmalige dynamische Werte reservieren, `AppearanceKey` und eigene Zustände für semantische Varianten nutzen und Themes beim Start oder zur Laufzeit für eine einzelne Session wechseln. Was das für Wisej.NET-Entwickler bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Gezieltes CSS sicher mit `CssClass` und einem bereichsbezogenen Anwendungs-Stylesheet hinzufügen, `CssStyle` für einmalige dynamische Werte reservieren, `AppearanceKey` und eigene Zustände für semantische Varianten nutzen und Themes beim Start oder zur Laufzeit für eine einzelne Session wechseln.

1. Das Problem im Produktivbetrieb

Das AdaptiveOps-Theme aus Modul 2 bestimmt jetzt die Identität jedes Standard-Controls in der Adaptive Operations Console, und die nächsten Anforderungen sind alle klein: Eine Kennzahlenkarte braucht eine abgerundete, erhöhte Fläche, ein SLA-Badge soll kompakt sein, ein Fortschrittsbalken braucht eine aus Live-Daten berechnete Breite, der Befehl Assign soll wie die primäre Aktion aussehen, und die Nachtschicht wünscht sich eine dunkle Variante. Für jede Anforderung gibt es eine verlockende Abkürzung – hier ein `CssStyle`-String, dort ein `BackColor`, das gemeinsame Theme direkt bearbeitet –, und jede Abkürzung verschiebt eine visuelle Entscheidung stillschweigend auf eine Ebene, die nicht dafür zuständig ist. Sechs Monate später kann niemand mehr sagen, was das Theme rendert, was das CSS überschreibt und was der Code flickt. Dieses Modul ordnet jede dieser fünf Anforderungen dem Mechanismus zu, der dafür zuständig ist.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
CssClass	Ein oder mehrere Klassennamen, die im Browser dem Hauptelement des Widgets hinzugefügt werden: der wiederverwendbare Anknüpfungspunkt, den ein bereichsbezogenes Anwendungs-Stylesheet ansprechen kann, ohne das Theme anzufassen.
CssStyle	Ein CSS-String, der direkt als style-Attribut des Widget-Hauptelements gerendert wird; richtig für einen einzelnen berechneten Wert, falsch als allgemeines Styling-Werkzeug, weil Inline-Styles schwer zu untersuchen, zu themen und an einen Designer zu übergeben sind.
bereichsbezogenes Anwendungs-Stylesheet	Eine CSS-Datei, die neben dem Theme geladen wird und deren Selektoren bei eigenen Klassennamen wie <code>.metric-card</code> beginnen und nie in das interne DOM eines Standard-Widgets von Wisej greifen.
Legacy-StyleSheet-Extender	Eine ältere Komponente, die geeigneten Controls CSS-Klassen hinzufügt und Regeln in ihrer Eigenschaft <code>StyleSheetSource</code> hält; eine Migrationshilfe, in neuem Code überflüssig, weil heute jedes Control <code>CssClass</code> und <code>CssStyle</code> besitzt.
AppearanceKey	Der Name einer Appearance im aktiven Theme, mit der ein Control gerendert wird; so drückt C# die Absicht aus (<code>action-button</code>), und das Theme bestimmt Farben, Schriften und Metriken jedes Zustands.
eigene Zustände	Namen in der <code>States</code> -Collection eines Controls, die das Theme wie hovered oder focused gestalten kann – für vorübergehende UI-Bedingungen wie stale, warning oder dirty, die ein Designer sehen und neu gestalten können soll.
Start-Theme und Application.Theme	Das anwendungsweite Theme wird vor dem Start der Anwendung in <code>Web.config</code> oder <code>Default.json</code> gewählt; mit <code>Application.Theme</code> kann eine Session zur Laufzeit mit einem Theme-Objekt arbeiten, ohne zu ändern, was andere Sessions sehen.

3. Der rote Faden der Lektion

Vier Werkzeuge, je eine Regel

Jedes Wisej.NET-Control bietet zwei clientseitige Anknüpfungspunkte. `CssClass` nimmt einen oder mehrere Klassennamen auf, die dem Hauptelement des Widgets hinzugefügt werden; `CssStyle` nimmt einen CSS-String auf, der direkt als style-Attribut dieses Elements gerendert wird. Im Eigenschaftensraster stehen beide nebeneinander, in einer Codebasis verhalten sie sich jedoch sehr unterschiedlich. Eine Klasse ist ein Name, nach dem Sie suchen, den Sie an zwanzig Controls wiederverwenden und in einer Datei umgestalten können; ein Inline-Style ist eine anonyme Überschreibung, die sich gegen das meiste durchsetzt, was das Theme erzeugt, und unsichtbar bleibt, bis jemand den Inspektor des Browsers öffnet.

Die Regel ist kurz. Verwenden Sie `CssClass`, wann immer Sie einen wiederverwendbaren Styling-Anknüpfungspunkt auf Basis einer Anwendungs-CSS-Datei brauchen. Verwenden Sie `CssStyle` nur für sehr kleine Einzel-Styles oder dynamisch berechnete Werte: eine Breite, die einem Prozentwert folgt, eine

Farbe, die mit den Daten kommt. Übermäßiger Gebrauch von `CssStyle` macht die Konsole schwer zu untersuchen, schwer zu themen und für einen Designer schwer zu pflegen. Deshalb erlaubt das Lab genau einen berechneten `CssStyle`-Wert und verlangt, dass Sie aufschreiben, warum er keine Klasse sein konnte.

Zwei ältere Mechanismen tauchen in echten Projekten noch auf. Der StyleSheet-Extender fügt geeigneten Controls CSS-Klassen hinzu und hält Regeln in seiner Eigenschaft `StyleSheetSource`; da jedes moderne Control bereits `CssClass` und `CssStyle` besitzt, ist er eine Migrationshilfe und nichts, was man in neuen Code einführt. Der Stylesheet-Abschnitt des Theme Builders liefert rohe CSS-Regeln innerhalb des Themes aus; er existiert für Widgets von Drittanbietern, eigenes HTML und Integrationsflächen ohne normale Wisej-Appearance, und bei Standard-Widgets können die aus Appearances und Zuständen erzeugten Regeln ihn überschreiben.

Bereichsbezogene Klassen für die eigene UI der Konsole

Das Lab fügt der Adaptive Operations Console `AdaptiveOps.css` hinzu, ein Stylesheet, das die Anwendung neben dem Theme lädt. Seine Aufgabe ist eng begrenzt: die Teile der Konsole gestalten, die Ihnen gehören, nicht Wisej. Eine Klasse `.metric-card` gibt den KPI-Panels Radius, Innenabstand und Erhöhung, `.metric-card .metric-title` setzt den kleinen Titel in Großbuchstaben, und `.compact-badge` macht aus einem Label eine Pille für SLA- und Prioritätsmarker. Jeder Selektor beginnt bei einer Klasse, die Ihnen gehört, sodass das Stylesheet nicht versehentlich anderswo einen Button oder einen Grid-Header umgestalten kann.

Was nicht in diese Datei gehört, ist ebenso wichtig wie das, was hineinkommt. Das Styling von Standard-Controls – Hover- und Pressed-Farben von Buttons, Schriften im Grid-Header, der Rahmen des ungültigen Editors – gehört zu den Theme-Appearances aus Modul 2. Sie per CSS zu flicken, bedeutet fragile Selektoren, die vom internen DOM eines Widgets abhängen und beim nächsten Framework- oder Theme-Update stillschweigend brechen. CSS für Widgets von Drittanbietern folgt derselben Disziplin: auf den Container des Widgets begrenzt und dokumentiert.

Bevor eine Klasse ausgeliefert wird, prüfen Sie, was sie aushebeln kann. Ein Control, bei dem `BackColor`, `ForeColor` oder `Font` direkt im Designer gesetzt ist, überschreibt sowohl das Theme als auch Ihre Klasse; ein übrig gebliebener `CssStyle` überschreibt alles. Wenn eine Klasse „nicht funktioniert“, lautet die Checkliste: Sitzt die Klasse am Element, ist am Control eine Eigenschaft gesetzt, ist ein Inline-Style gesetzt, und ist die Regel auf den richtigen Container begrenzt?

Semantische Varianten: AppearanceKey und eigene Zustände

Setzt man `BackColor`, `ForeColor` und `Font` am Save-Button, sieht er heute primär aus und ist morgen unantastbar. `AppearanceKey` ist die Alternative: Das Control wird mit einer benannten Appearance aus dem aktiven Theme gerendert, und das Theme bestimmt die visuellen Details für default, hovered, pressed, disabled, focused und invalid. Die `action-button`-Appearance aus Modul 2 erbt von `button` und überschreibt nur drei Zustände; weist man sie dem primären Befehl zu, drückt C# die fachliche Absicht aus und überlässt die Pixel dem Theme. Dasselbe Muster skaliert auf `destructive-button`, `quiet-panel`, `kpi-card` oder `sidebar-button`, wann immer das fachliche Konzept wiederverwendbar ist.

Eigene Zustände decken die vorübergehenden Bedingungen ab, die ein Appearance-Schlüssel nicht abdecken kann. Jedes Control hat eine `States`-Collection; fügt man einen Namen wie `stale`, `warning`,

`dirty` oder `compact` hinzu, kann das Theme diesen Zustand genauso gestalten wie `hovered` oder `focused`. Das Lab fügt den Zeilen des Ticket-Grids oder dem Status-Panel einen Zustand `stale` hinzu, wenn ein Ticket nicht rechtzeitig aktualisiert wurde, und das Theme entscheidet, wie `stale` aussieht. Ein Zustand ist eine UI-Bedingung, die das Theme visuell darstellen kann, kein Ort für Anwendungsdaten, und Code sollte nie einen Zustand zurücklesen, um eine Entscheidung zu treffen.

```
// MainPage.cs – jede Ebene trifft ihre eigene Entscheidung
this.cardOpen.CssClass          = "metric-card";      // bereichsbezogenes App-CSS
this.lblPriorityBadge.CssClass  = "compact-badge";
this.btnSave.AppearanceKey     = "action-button";    // die Zustände gehören dem Theme

// der eine berechnete Wert: clip-path, nicht width – die Layout-Engine überschreibt
// eine Inline-width bei jedem Layout-Durchlauf und würde sie beim nächsten Resize löschen
this.barFill.CssStyle = $"clip-path:inset(0 {100 - slaPercent}% 0 0)";

if (ticket.IsStale)
    row.States.Add("stale");                          // eigener Zustand, im Theme
                                                       gestaltet
else
    row.States.Remove("stale");
```

Start-Themes, Session-Themes und das gemeinsame Theme

Ein Theme wird vor dem Start der Anwendung angewendet, indem man es in `Web.config` oder in `Default.json` angibt. Das im Visual-Studio-Designer angezeigte Theme zu ändern, ändert nicht das Laufzeit-Theme; die Konsole läuft nur deshalb mit `AdaptiveOps`, weil die Konfiguration es so vorgibt. Benannte Theme-Dateien sind der richtige Ort für stabiles Branding: versioniert, geprüft und für jeden Benutzer identisch.

Wisej.NET stellt außerdem `Application.Theme` bereit, über das eine Session zur Laufzeit mit einem Theme-Objekt arbeiten kann. Dort sitzt der Hell/Dunkel-Umschalter des Labs: Der Schalter in der Toolbar lädt die dunkle Variante, weist sie der aktuellen Session zu und merkt sich die Wahl in `Application.Session`. Die subtile Gefahr: Ein Theme-Objekt lässt sich direkt verändern. Verändert man das von allen Sessions geteilte Theme, weil ein Benutzer den Dunkelmodus wollte, ändert sich die Konsole für die gesamte Nachtschicht; das sichere Muster ist eine separate benannte Theme-Datei oder eine Kopie pro Session, niemals eine Bearbeitung des gemeinsamen Objekts.

Die Entscheidungsregel: benannte Theme-Dateien für stabiles Branding, Theme-Wechsel zur Laufzeit für Personalisierung, Mandanten-Akzente, Demos und kontrollierte Vorschauen. Belegen Sie die Grenze mit zwei Browser-Sessions nebeneinander, einer dunklen und einer hellen, bevor Sie das Ergebnis als fertig betrachten.



Typische Fehler

`CssStyle`-Strings, die auf jeden Button kopiert werden, bis niemand mehr weiß, was das Theme rendert. Selektoren, die in das interne DOM eines Standard-Widgets greifen. Im Designer gesetzte `BackColor`, `ForeColor` oder `Font`, die Theme und Klasse stillschweigend überschreiben. `States` als Datenspeicher verwenden. Ein Dunkelmodus-Schalter, der das gemeinsame Theme für jede Session bearbeitet.



Gewohnheit für das visuelle System

Nutzen Sie den Wisej.NET-Mechanismus, der für das Problem zuständig ist: das Theme für die Identität der Standard-Controls, AppearanceKey für semantische Varianten, CssClass für bereichsbezogenes anwendungsspezifisches CSS, CssStyle nur für einmalige dynamische Werte, Layout-Container statt Resize-Code, responsive Eigenschaften für im Designer gesetzte Profilwerte und ResponsiveProfileChanged für Verhalten, das nicht rein visuell ist.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ CssClass, CssStyle, eigenes CSS und die Muster des Legacy-StyleSheet-Extenders korrekt einsetzen.
- ✓ bereichsbezogene Klassen für anwendungsspezifische UI definieren und das Styling der Standard-Controls in den Themes belassen.
- ✓ AppearanceKey und eigene Zustände nutzen, um semantische Control-Varianten zu erstellen.
- ✓ Start-Themes anwenden und den Unterschied zwischen sessionspezifischen und globalen Theme-Änderungen verstehen.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praktisches Lab

Fügen Sie der Adaptive Operations Console ein bereichsbezogenes Stylesheet `AdaptiveOps.css` mit Klassen für Kennzahlenkarten und kompakte Badges hinzu, wenden Sie diese über `CssClass` an und verwenden Sie `CssStyle` nur für einen einzigen berechneten Wert, etwa eine Fortschrittsbreite. Weisen Sie dem primären Befehl den `AppearanceKey` `action-button` zu, fügen Sie den Zeilen des Ticket-Grids oder einem Status-Panel einen eigenen Zustand `stale` hinzu und implementieren Sie in der Toolbar einen Hell/Dunkel-Umschalter, der das Theme nur für die aktuelle Session wechselt, ohne das gemeinsame Theme zu verändern.

Ausgangspunkt

Öffnen Sie die Adaptive Operations Console aus Modul 2: die gedockte Shell mit Toolbar-, Navigations-, Arbeitsbereichs-, Details- und Status-Panel, das in `Web.config` oder `Default.json` ausgewählte `AdaptiveOps`-Theme, seine benannten Tokens und die `action-button`-Appearance. Der Arbeitsbereich enthält bereits vier Kennzahlen-Panels, einen SLA-Fortschrittsbalken und das Ticket-Grid, die aber noch mit Standard-Apearances oder spontan gesetzten Werten für `BackColor`, `ForeColor` und `CssStyle` gestaltet sind, und die Toolbar hat keinen Theme-Umschalter.

Vorgeschlagenes Vorgehen

- ✓ Legen Sie `AdaptiveOps.css` an, laden Sie es mit der Anwendung neben dem Theme und schreiben Sie nur Selektoren, die bei Klassen beginnen, die Ihnen gehören: `.metric-card`, `.metric-card .metric-title` und `.compact-badge`. Wenden Sie sie über `CssClass` auf die vier Kennzahlen-Panels sowie auf die SLA- und Prioritäts-Labels an.
- ✓ Finden Sie den einzigen Wert, der zur Laufzeit berechnet werden muss, die SLA-Fortschrittsbreite, setzen Sie ihn in einer Methode über `CssStyle` und schreiben Sie einen Kommentar, warum eine Klasse ihn nicht ausdrücken konnte. Entfernen Sie jeden anderen `CssStyle` und jede an diesen Controls verbliebene Überschreibung von `BackColor`, `ForeColor` oder `Font`.
- ✓ Weisen Sie dem primären Befehl (`btnSave`) `AppearanceKey = "action-button"` zu und bestätigen Sie, dass das Aussehen für `hovered`, `pressed` und `disabled` ganz ohne Code aus dem Theme kommt.
- ✓ Fügen Sie im Theme Builder der Appearance der Ticketzeile oder des Status-Panels einen Zustand `stale` hinzu, gestaltet mit dem Token `warning`, und fügen Sie `"stale"` über `States` hinzu bzw. entfernen Sie es, wenn die letzte Aktualisierung eines Tickets den Schwellenwert überschreitet. Den Schwellenwert halten Sie in den Daten, das Aussehen im Theme.
- ✓ Fügen Sie der Toolbar einen Theme-Umschalter hinzu, der eine zweite benannte Theme-Datei, `AdaptiveOps-Dark`, lädt, sie `Application.Theme` für die aktuelle Session zuweist und die Wahl in `Application.Session` speichert. Prüfen Sie mit zwei Browser-Sessions, dass das gemeinsame Theme unangetastet bleibt.

Geforderte Ergebnisse

- ✓ Bereichsbezogene AdaptiveOps.css mit den Klassen metric-card und compact-badge, angewendet über CssClass
- ✓ CssStyle für genau einen berechneten Einzelwert verwendet, mit dokumentierter Begründung
- ✓ Primärer Befehl mit dem AppearanceKey action-button
- ✓ Eigener Zustand stale über States hinzugefügt und im Theme gestaltet
- ✓ Hell/Dunkel-Umschalter nur für die Session, der das gemeinsame Theme unangetastet lässt

Wiederholungsfragen

- ✓ Ein Designer wünscht einen kleineren Eckenradius für jede Kennzahlenkarte. Welche Datei ändert sich, und an wie vielen Stellen hätten Sie ändern müssen, wenn der Radius mit `CssStyle` gesetzt worden wäre?
- ✓ Warum braucht der Button Assign keinen Code für hover oder pressed, und was genau würden Sie verlieren, wenn Sie seinen `AppearanceKey` durch zur Palette passende Werte für `BackColor` und `ForeColor` ersetzen?
- ✓ Zwei Personen sind angemeldet, und eine davon wechselt in den Dunkelmodus. Beschreiben Sie, was jede sieht, und zeigen Sie auf die Zeile Ihres Codes, die das garantiert.



Gewohnheit für das visuelle System

Nutzen Sie den Wisej.NET-Mechanismus, der für das Problem zuständig ist: das Theme für die Identität der Standard-Controls, AppearanceKey für semantische Varianten, CssClass für bereichsbezogenes anwendungsspezifisches CSS, CssStyle nur für einmalige dynamische Werte, Layout-Container statt Resize-Code, responsive Eigenschaften für im Designer gesetzte Profilwerte und ResponsiveProfileChanged für Verhalten, das nicht rein visuell ist.