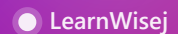


Das visuelle System von Wisej.NET

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Wie serverseitige Controls, clientseitige Widgets, Theme-Appearances, Zustände, CSS-Klassen und Layout-Engines gemeinsam erzeugen, was der Benutzer sieht, welche der drei Anpassungsebenen für welche Entscheidung zuständig ist, und die Shell der Adaptive Operations Console, die im Verlauf des Kurses wächst. Was das für Wisej.NET-Entwickler bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Wie serverseitige Controls, clientseitige Widgets, Theme-Appearances, Zustände, CSS-Klassen und Layout-Engines gemeinsam erzeugen, was der Benutzer sieht, welche der drei Anpassungsebenen für welche Entscheidung zuständig ist, und die Shell der Adaptive Operations Console, die im Verlauf des Kurses wächst.

1. Das Problem im Produktivbetrieb

Ein Team, das Wisej.NET wie ein CSS-first-Framework behandelt, landet bei `BackColor` auf zweihundert Controls, bei Selektoren, die beim nächsten Upgrade brechen, und bei Resize-Handlern, die `Bounds` von Hand neu berechnen; wird dann eine Smartphone-Version verlangt, bleibt nur noch eine Neuentwicklung. Dieses Modul klärt, woher Aussehen und Layout eines Controls wirklich kommen, benennt die drei Anpassungsebenen und ihre Zuständigkeiten und beginnt die Shell der Adaptive Operations Console, die jedes spätere Modul erweitert.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Server-Control und Client-Widget	Sie schreiben .NET-Controls auf dem Server; der Browser zeigt JavaScript-Widgets an, die Wisej.NET synchron hält. Eine visuelle Entscheidung kann also auf beiden Seiten dieser Grenze liegen.
Theme	Eine einzige JSON-Datei mit Farben, Bildern, Schriften, Appearances, Zuständen, Styles, Eigenschaften und Einstellungen, die jedem Standard-Control seine Identität gibt; der Vertrag zwischen beiden Welten.
Appearance und <code>AppearanceKey</code>	Jedes Control stellt einen Appearance-Schlüssel bereit; das aktive Theme enthält eine Appearance für diesen Schlüssel, und ein semantischer Schlüssel wie <code>action-button</code> wählt eine Variante aus.
Zustand	Ein benannter Interaktionszustand innerhalb einer Appearance (default, hovered, pressed, focused, invalid, selected), dessen Styles und Eigenschaften gelten, solange er aktiv ist.
<code>CssClass</code> und <code>CssStyle</code>	Direktes Browser-Styling: eine Klasse aus einem bereichsbezogenen Anwendungs-Stylesheet oder ein roher Style-String am Widget-Element für einen seltenen Einzelwert.
Layout-Engine und <code>Dock</code>	Position und Größe ergeben sich aus der Container-Hierarchie, nicht aus dem Theme; Docking stapelt Bereiche an den Rändern des Parents und füllt den verbleibenden Platz.
visuelle Bestandsaufnahme	Die schriftliche Liste der Controls, die Appearances benötigen, der Farb- und Schrift-Tokens, der Layout-Bereiche und der Breakpoints, auf deren Grundlage Theme, Layout und Profile geplant werden.

3. Der rote Faden der Lektion

Das Theme ist der Vertrag zwischen Server-Controls und Browser-Widgets

In Wisej.NET schreibt der Entwickler .NET-Controls auf dem Server, doch was der Benutzer sieht, ist ein clientseitiges Widget, das der Browser rendert. Ein `Button`, `TextBox`, `TabControl`, `DataGridView`, `Panel` oder ein eigenes Widget stellt einen Appearance-Schlüssel bereit, das aktive Theme enthält eine Appearance für diesen Schlüssel, und jede Appearance enthält Zustände, Styles, Eigenschaften und Child-Komponenten. Deshalb lässt sich eine professionelle visuelle Änderung einmal zentral vornehmen, statt sie über einzelne Controls zu verstreuen: Ändern Sie die Button-Appearance, und jeder Button in jeder Session folgt.

Deshalb ist Theming in Wisej.NET auch mehr als normales CSS. Das Theme reicht bis in die Controls selbst, in die Widget-Metriken, von denen das Layout abhängt, in die Schriften und Größen, mit denen der Visual-Studio-Designer rendert, und in Farben, Bilder und Appearances, die CSS allein nicht ausdrücken kann. Die Theme-Engine kann aus Appearance und Zustand CSS-Klassen erzeugen, und Theme-Eigenschaften können Styles überschreiben. Eine handgeschriebene Regel gegen das interne Widget-DOM arbeitet also gegen die Engine, statt sie zu nutzen. Behandeln Sie die Theme-Datei als Design-Asset des Produkts: Benannte Farben und gemeinsame Schriften sind Design-Tokens, Appearances bilden

Komponentenvarianten ab, Zustände drücken Interaktion aus, und Button, Grid, Editor, Panel, Toolbar und Formular wirken zusammengehörig, auch wenn verschiedene Entwickler sie erstellt haben.

Drei Ebenen – und welche für welche Entscheidung zuständig ist

Ebene eins ist das Theme: globale visuelle Identität, Zustandsverhalten, Widget-Metriken, benannte Farben, Schriften und die Appearances der Standardkomponenten. Ebene zwei ist das serverseitige Control-Modell: `BackColor`, `ForeColor`, `Font`, `Padding`, `Margin`, `Dock`, `Anchor`, `Size`, `MinimumSize`, `MaximumSize`, `AppearanceKey` und die responsiven Eigenschaften. Ebene drei ist direktes Browser-Styling: `CssClass`, `CssStyle`, der StyleSheet-Extender in Legacy-Code, eigene CSS-Dateien, `InitScript` und die Anpassung von JavaScript-Widgets. Fortgeschrittene Entwickler müssen wissen, welche Ebene für welche Entscheidung zuständig ist, bevor sie eine davon anfassen.

Die Entscheidungsregel, die der ganze Kurs anwendet, ist kurz. Zuerst das Theme für die Identität der Standard-Controls. `AppearanceKey` für eine semantische Variante wie einen Aktions- oder einen destruktiven Button, wobei das Theme die Zustände default, hovered, pressed, focused und disabled definiert. `CssClass` für einen eng begrenzten, anwendungsspezifischen Anknüpfungspunkt: eine Kennzahlenkarte, eingebettetes HTML, ein Widget eines Drittanbieters. `CssStyle` nur für einen kleinen Einzel- oder dynamischen Wert, selten und lokal. Layout-Container für die Positionierung, responsive Eigenschaften für Werte, die der Designer pro Profil zuweist, und `ResponsiveProfileChanged` für Verhalten, das sich nicht als Eigenschaftswert ausdrücken lässt.

Die negative Regel ist ebenso wichtig: Flicken Sie nicht jedes Standard-Control mit direktem CSS. Theme-Appearances sind für integrierte Controls besser, weil sie an Zuständen, am Rendering im Designer und an den gemeinsamen Tokens teilhaben; CSS ist für anwendungsspezifische Inhalte, Widgets von Drittanbietern und eingebettetes HTML gedacht. Wenn ein Theme „nicht greift“, sind die ersten Verdächtigen ein direkt gesetztes `BackColor`, `ForeColor`, `Font`, `CssStyle` oder eine eigene `CssClass`, die es überschreibt, ein falscher Appearance-Schlüssel oder ein Theme, das im Designer eingestellt, aber für die laufende Anwendung nicht ausgewählt ist.

Die visuelle Bestandsaufnahme: Tokens, Bereiche, Breakpoints

Bevor jemand den Theme Builder öffnet, verlangt das Lab eine Bestandsaufnahme der Anwendung, wie sie werden soll. Gehen Sie die Adaptive Operations Console durch und listen Sie die Controls auf, die eine eigene Appearance brauchen: die Befehls-Buttons, die Panels, die zu Kennzahlenkarten werden, die Tabs, den Grid-Header, den ungültigen Zustand des Ticket-Editors, die Tooltips. Listen Sie die Farb-Tokens auf, die das Theme einmal definiert und überall wiederverwendet: `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` und `focusFrame`, dazu eine Standard-, eine Überschriften- und eine Monospace-Schrift. Listen Sie die Layout-Bereiche auf: Toolbar, Navigationsleiste, Arbeitsbereich mit Kennzahlenkarten und Ticket-Grid, Details-Panel, Statusleiste. Listen Sie schließlich die Breakpoints auf: Desktop bei 1440, Small Desktop bei 1024, Tablet bei 768 im Hochformat und 1024 im Querformat, Smartphone bei 390 im Hochformat und 844 im Querformat.

Schreiben Sie das als `VisualInventory.md` ins Projekt und geben Sie jeder Zeile eine vierte Spalte: die zuständige Ebene. Die abgerundeten Ecken einer Kennzahlenkarte gehören zu einer bereichsbezogenen `CssClass`; die Hover-Farbe des Buttons gehört zum Theme; die Breite des Details-Panels gehört zu `Dock`; ob die Navigationsleiste auf dem Smartphone sichtbar ist, gehört zu einer responsiven Eigenschaft. Halten

Sie in derselben Datei die Entscheidung für das Basis-Theme fest: Modul 2 leitet das AdaptiveOps-Theme von einem der integrierten Themes ab und erbt dessen Appearances, Zustände, Bilder und Metriken. Dokumentieren Sie also, warum Sie es gewählt haben, nicht nur welches.

Die Shell der Adaptive Operations Console

Das Kursprojekt ist eine Operations-Konsole mit einem Marken-Theme sowie einem Desktop-, einem Tablet- und einem Smartphone-Layout. Die Desktop-Version hat links eine Navigationsleiste, eine Toolbar, Kennzahlenkarten, ein Ticket-Grid und ein Details-Panel. Die mobile Version klappt die Navigation ein, stapelt Grid und Details, blendet nachrangigen Text aus, zeigt wo sinnvoll nur Symbole und hält das Scrollen per Touch angenehm. Modul 1 baut nur die Desktop-Shell, mit `Dock` und verschachtelten `Panel`s und nichts sonst: Toolbar oben, Status unten, Navigation links, Details rechts und ein füllend gedockter Arbeitsbereich mit einer `MinimumSize`, damit er nie zusammenfällt.

```
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";
}
```

Das einzige Verhalten in der Shell ist das Status-Label, das die aktuelle Browserbreite meldet. Die Haupt-`Page` füllt immer den Browser-Viewport, daher wird ihr `Resize`-Event bei jeder Größenänderung des Fensters ausgelöst; schreibt man `this.Width` in das Label, haben spätere Module eine Zahl, die sie mit den Breakpoints in `ClientProfiles.json` und dem Namen in `Application.ActiveProfile` vergleichen können. Umschließen Sie die Aktualisierung mit einem try / catch, das einen ehrlichen Ersatztext statt eines veralteten Werts hinterlässt. Was die Shell nicht enthalten darf, ist ebenso wichtig wie das, was sie tut:

- Keine Überschreibungen von `BackColor`, `ForeColor` oder `Font`: Bis Modul 2 bestimmt das Basis-Theme das Aussehen.
- Noch keine `CssStyle`-Strings und kein eigenes CSS: Nichts in der Shell ist anwendungsspezifisch genug, um sie zu brauchen.
- Kein Resize-Code, der `Bounds` setzt: Docking beantwortet bereits jede Desktop-Breite.

Machen Sie den Desktop-Screenshot, sobald die Shell läuft, und schreiben Sie die Notiz, die Bereich für Bereich erklärt, welche Ebene für welche visuelle Entscheidung zuständig ist; sie ist der Keim der Architekturnotiz, die das Abschlussprojekt verlangt.



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:

- ✓ erklären, warum Theming in Wisej.NET mehr als normales CSS ist und wie es bis in Controls, Widgets, Designer-Metriken, Schriften, Farben, Bilder und Appearances reicht.
- ✓ erkennen, wann eine Theme-Datei, ein integriertes Theme, eine Control-Eigenschaft, CssClass, CssStyle oder ein Layout-Container das richtige Mittel ist.
- ✓ eine visuelle Bestandsaufnahme einer bestehenden Anwendung erstellen und Theme-Tokens, Layout-Bereiche und responsive Breakpoints definieren.
- ✓ das Kursprojekt Adaptive Operations Console mit einer grundlegenden Desktop-Shell und dem angestrebten mobilen Verhalten beginnen.

Lab- und Prüfungsleitfaden

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



Praktisches Lab

Legen Sie die Solution Adaptive Operations Console und ihre erste Desktop-Shell an: eine Haupt-Page mit einem Toolbar-Panel oben, einem Navigations-Panel links, einem füllenden Arbeitsbereich, einem Details-Panel rechts und einem Status-Panel unten, aufgebaut mit Dock und verschachtelten Panels. Wählen Sie ein integriertes Basis-Theme und dokumentieren Sie die Wahl, schreiben Sie eine `VisualInventory.md`, die die Controls mit eigenen Appearances, die benannten Farb- und Schrift-Tokens, die Layout-Bereiche sowie die Breakpoints für Desktop, Tablet und Smartphone auflistet, und fügen Sie ein Status-Label hinzu, das die aktuelle Browserbreite meldet, damit spätere Module die Profile überprüfen können.

Ausgangspunkt

Öffnen Sie Visual Studio und legen Sie eine neue Wisej.NET-Webanwendung mit dem Namen `AdaptiveOperationsConsole` an; diese Solution erweitert jedes spätere Lab, halten Sie die Arbeit jedes Moduls also in einem eigenen Ordner. Die Vorlage liefert eine leere Haupt- `Page` , eine `Program.cs` , die sie öffnet, und die Standardauswahl des Themes; Panels, Theme-Datei und Bestandsaufnahme gibt es noch nicht.

Vorgeschlagenes Vorgehen

- ✓ Skizzieren Sie die fünf Bereiche zuerst auf Papier, ziehen Sie dann fünf `Panel` s auf die Haupt-Page und docken Sie sie in dieser Reihenfolge: `toolbarPanel` oben, `statusPanel` unten, `navigationPanel` links, `detailsPanel` rechts und zuletzt `workspacePanel` mit Dock auf Fill und einer `MinimumSize` von 320 mal 240.
- ✓ Wählen Sie eines der integrierten Themes als Basis, wählen Sie es für die laufende Anwendung aus (nicht nur im Designer) und notieren Sie die Begründung in Ihren Lab-Notizen, bevor Sie irgendetwas anderes ändern.
- ✓ Fügen Sie dem Status-Panel ein `widthLabel` hinzu, behandeln Sie das `Resize` -Event der Haupt-Page und schreiben Sie `$"Width: {this.Width} px"` aus einer einzigen, mit try / catch umschlossenen Methode `ReportWidth()` hinein; rufen Sie sie einmal im Konstruktor auf, damit das Label nie leer ist.
- ✓ Legen Sie im Projekt eine `VisualInventory.md` mit vier Tabellen an: Controls, die eigene Appearances brauchen, Farb- und Schrift-Tokens (brandPrimary, brandAccent, surface, surfaceAlt, textMain, textMuted, danger, warning, success, focusFrame; default, heading, mono), Layout-Bereiche sowie die Breakpoints für Desktop, Tablet und Smartphone; geben Sie jeder Zeile eine Spalte „zuständig“, die Theme, Control-Eigenschaft, CssClass, CssStyle oder Layout-Container nennt.
- ✓ Starten Sie die Shell, verkleinern Sie den Browser von Desktop- über Tablet- auf Smartphone-Breite und beobachten Sie dabei das Label, machen Sie den Desktop-Screenshot und schreiben Sie die Notiz, die erklärt, welche Ebene für welche visuelle Entscheidung in der Shell zuständig ist.

Geforderte Ergebnisse

- ✓ Solution Adaptive Operations Console mit gedockter Desktop-Shell aus Toolbar-, Navigations-, Arbeitsbereichs-, Details- und Status-Panel
- ✓ Basis-Theme ausgewählt und konfiguriert, Begründung in den Lab-Notizen festgehalten
- ✓ VisuellInventory.md mit den Controls, die Appearances benötigen, den Theme-Tokens, den Layout-Bereichen und den Breakpoints
- ✓ Status-Label, das die aktuelle Browserbreite meldet
- ✓ Desktop-Screenshot und eine Notiz, welche Ebene für welche visuelle Entscheidung zuständig ist

Wiederholungsfragen

- ✓ Der primäre Befehl der Toolbar braucht auf jedem Bildschirm die Markenfarbe. Welche der drei Ebenen ist für diese Entscheidung zuständig, und was ginge schief, wenn Sie stattdessen `BackColor` am Button setzen?
- ✓ Ihr Breiten-Label zeigt 1024 px, und das Details-Panel ist noch sichtbar. Welcher Mechanismus aus einem späteren Modul entscheidet, ob es das sein sollte, und warum ist eine CSS-Media-Query in Wisej.NET die falsche erste Antwort?
- ✓ Wenn das Basis-Theme morgen gegen ein anderes integriertes Theme getauscht würde: Welche Zeilen Ihrer VisuellInventory.md würden sich ändern und welche nicht, und was sagt Ihnen das darüber, wo die Entscheidungen der Bestandsaufnahme angesiedelt sind?



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.