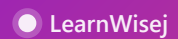


Mobilitaugliches Abschlussprojekt, Barrierefreiheit, Performance und Governance im Produktivbetrieb

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Die Adaptive Operations Console als produktivreifen Satz von Desktop-, Tablet- und Smartphone-Ansichten fertigstellen, Fokus, Kontrast, ungültige Zustände und Touch-Ziele prüfen, Profil-Handler günstig und idempotent halten sowie Theme, CSS und ClientProfiles.json wie Code verwalten. Was das für Wisej.NET-Entwickler bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Die Adaptive Operations Console als produktivreifen Satz von Desktop-, Tablet- und Smartphone-Ansichten fertigstellen, Fokus, Kontrast, ungültige Zustände und Touch-Ziele prüfen, Profil-Handler günstig und idempotent halten sowie Theme, CSS und ClientProfiles.json wie Code verwalten.

1. Das Problem im Produktivbetrieb

Die Arbeit aus sechs Modulen steckt jetzt in einer Solution: ein von einem Basis-Theme abgeleitetes Theme, bereichsbezogenes CSS, eine Shell aus verschachtelten Containern, Flow-, Table- und Flex-Bereiche, eine ClientProfiles.json und ein Profil-Handler. Was daraus ein Produkt macht, ist kein weiteres Feature. Es ist der Nachweis, dass die Konsole in jedem Profil benutzbar ist, dass ein Tastatur- oder Touch-Benutzer sie bedienen kann, dass eine Größenänderung keine teuren Neuaufbauten auslöst und dass der nächste Entwickler oder das nächste Wisej.NET-Upgrade das visuelle System nicht stillschweigend brechen kann. Dieses Modul ist dieser Nachweis, dazu das Grounding-Paket, mit dem ein KI-Assistent Fragen zu Ihren Entscheidungen beantworten kann.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Ansichten des Abschlussprojekts	Die fertige Adaptive Operations Console: eigenes Theme, semantische Appearances, bereichsbezogenes CSS, eine Desktop-Shell, eine adaptive Tablet-Shell, eine Smartphone-Shell, ein Grid, ein Editor, Validierungs-Feedback, Navigation und eine im Testmodus sichtbare Profilanzeige.
visuelles Barrierefreiheits-Review	Ein Durchgang über Fokusrahmen, deaktivierte und ungültige Zustände, Kontrast, Touch-Abstände, Tooltips an reinen Symbol-Buttons und die Tab-Reihenfolge der Tastatur – durch Bedienen des Bildschirms, nicht durch Lesen des Themes.
idempotenter Profil-Handler	Ein <code>ResponsiveProfileChanged</code> -Handler, der die endgültigen Werte für das aktuelle Profil zuweist, sodass ein zweiter Durchlauf oder ein Durchlauf nach einem Neuladen exakt dieselbe UI ergibt.
Layout-Stabilität	<code>MinimumSize</code> und <code>MaximumSize</code> verhindern, dass Bereiche zusammenfallen, und <code>AutoSize</code> bleibt von großen Container-Hierarchien fern, sodass Größenänderungen vorhersehbar und günstig sind.
Theme-Governance	Theme-JSON, bereichsbezogenes Stylesheet und ClientProfiles.json stehen unter Versionskontrolle, werden als Diffs geprüft, als Tokens dokumentiert und nie innerhalb eines integrierten Themes bearbeitet.
upgradesichere Anpassung	Dokumentierte Appearances, Theme-Builder-Arbeitsabläufe und serverseitige Eigenschaften überstehen Upgrades; CSS-Selektoren, die vom internen DOM eines Widgets abhängen, möglicherweise nicht – daher werden sie eng begrenzt, dokumentiert oder vermieden.
responsive QA-Matrix und Grounding-Paket	Die Matrix hält pro Szenario Breiten, Ausrichtungen, erwartetes Profil und Prüfungen fest; das Grounding-Paket hält den Entscheidungsbaum, die Antwortregeln und die Quellenübersicht fest, aus denen ein Assistent antwortet.

3. Der rote Faden der Lektion

Eine Konsole, drei Shells, ein visuelles System

Das Abschlussprojekt wird nach Konsistenz bewertet, nicht nach Neuheit. Standard-Controls beziehen ihre Identität aus dem AdaptiveOps-Theme, das von einem Basis-Theme abgeleitet ist und benannte Farb-Tokens für Marke, Flächen, Text, danger, warning, success und den Fokusrahmen definiert. Fachliche Varianten sind semantische Appearances, die über `AppearanceKey` zugewiesen werden: `action-button`, `destructive-button`, `kpi-card`, `nav-item`, `compact-editor` oder Entsprechungen, mindestens vier davon. Anwendungsspezifische Inhalte wie Kennzahlenkarten verwenden bereichsbezogene Klassen über `CssClass`, und `CssStyle` taucht nur dort auf, wo ein Wert zur Laufzeit berechnet wird. Greift ein Bildschirm direkt zu `BackColor`, `ForeColor` oder `Font`, fragt das Review, welche dieser Ebenen dafür hätte zuständig sein sollen.

Die Shell behält die Struktur, die Sie in den Modulen 4 und 5 gebaut haben: gedockte Toolbar, Navigationsleiste, Arbeitsbereich, Detailbereich und Statusleiste, mit einem `FlowLayoutPanel` für Filter und Karten, einem `TableLayoutPanel` für den ausgerichteten Ticket-Editor und einem `FlexLayoutPanel` für den proportionalen Arbeitsbereich. Unterschiede zwischen Desktop, Tablet und Smartphone kommen aus `ClientProfiles.json` und im Designer zugewiesenen responsiven Eigenschaften; der eine `ResponsiveProfileChanged` -Handler existiert für Verhalten, das eine Eigenschaft nicht ausdrücken kann, etwa das Öffnen des Details-Editors als modales Formular auf dem Smartphone. Die Profilanzeige bleibt im Statusbereich, im Testmodus sichtbar, sodass jeder Screenshot in der QA-Matrix das Profil nennt, das ihn erzeugt hat.

Barrierefreiheit, die Sie durch Bedienen des Bildschirms prüfen

Ein Theme muss sichtbaren Fokus, klare deaktivierte Zustände, lesbaren Text, ausreichenden Kontrast, Feedback für ungültige Zustände und touchfreundliche Abstände bewahren. Die häufigste Regression ist das Löschen des Fokusrahmens, weil er altmodisch aussieht; die Regel lautet, ihn durch einen besseren aus dem Token `focusFrame` zu ersetzen, nie, ihn zu entfernen. Ungültige und Warnzustände dürfen sich nicht allein auf Farbe verlassen: Kombinieren Sie den danger-Rahmen mit einem Symbol oder einer lesbaren Meldung, damit der Zustand Farbenblindheit und einen Projektor übersteht. Der Kontrast wird an den tatsächlichen Token-Paaren geprüft, die das Theme verwendet, zum Beispiel `textMuted` auf `surfaceAlt`, nicht an der Markenfarbe für sich allein.

Reine Symbol-Buttons in der Toolbar, auf die das Tablet-Profil über den responsiven `Display` -Wert umschaltet, brauchen `ToolTipText` oder ein barrierefreies Label, damit sie auffindbar bleiben. Die Tab-Reihenfolge wird geprüft, indem Sie den Bildschirm mit der Tastatur durchlaufen und sicherstellen, dass `TabStop` und `TabIndex` den primären Befehl dorthin setzen, wo ein Benutzer ihn erwartet. Schließlich braucht ausgeblendeter Inhalt eine Erklärung: Werden Navigationsleiste und Details-Panel auf dem Smartphone ausgeblendet, müssen ihre Funktionen über den mobilen Menü-Button und den modalen Editor weiterhin erreichbar sein, und die Architekturnotiz hält diese Alternative für jedes ausgeblendete Control fest.

Günstige Handler, stabile Layouts, verwaltete Dateien

Responsives Design wird teuer, wenn jede Größenänderung komplexes Layout, neu geladene Daten und Client-Aktualisierungen auslöst. Der Handler, der auf einen Profilwechsel reagiert, muss idempotent und günstig sein: Er weist die endgültige Sichtbarkeit, das Docking und die Größe für das aktuelle Profil zu, statt umzuschalten, baut die UI nicht neu auf, wenn ein paar Eigenschaftsänderungen genügen, und lädt nie Daten neu, nur weil sich die Breite geändert hat. Die Methode `ApplyProfile` des Labs wird einmal im Konstruktor mit `Application.ActiveProfile` und erneut aus dem Handler mit dem aktuellen Profil des Events aufgerufen, und das Abonnement wird entfernt, wenn die Seite freigegeben wird:

```
private void ApplyProfile(string name)
{
    if (name == _lastProfile) return;           // gleiches Profil: nichts zu tun
    _lastProfile = name;
    bool phone = name != null && name.StartsWith("Phone");

    lblProfile.Text = name ?? "Default";         // Anzeige im Testmodus
    navRail.Visible = !phone;                   // endgültige Werte, nie umschalten
    detailsPanel.Visible = !phone;
    btnOpenDetails.Visible = phone;
    ticketGrid.MinimumSize = new Size(phone ? 0 : 320, 240);
}
```

Layout-Stabilität entsteht aus derselben Disziplin: `MinimumSize` und `MaximumSize` an Bereichen mit `FillWeight` und an gedockten Bereichen, damit nichts auf null zusammenfällt, kein `AutoSize` an großen Container-Hierarchien und responsive Datenänderungen, die beabsichtigt sind statt Nebenwirkung einer Größenänderung. Behandeln Sie das Theme dann wie Code. `AdaptiveOps.theme`, `AdaptiveOps.css` und `ClientProfiles.json` liegen mit klaren Namen in der Versionskontrolle; ihre Diffs werden geprüft; die Design-Tokens sind dokumentiert; integrierte Themes werden nie direkt bearbeitet. Das Team hält fest, wann ein Entwickler `BackColor`, `ForeColor` oder `Font` verwenden darf, wann `AppearanceKey`, `CssClass`, `CssStyle` oder eigenes CSS das richtige Werkzeug ist, und pflegt einen Satz Screenshots der wichtigsten Bildschirme in Desktop-, Tablet- und Smartphone-Größe.

Upgradesicherheit folgt aus denselben Regeln. Dokumentierte Theme-Appearences, Theme-Builder-Arbeitsabläufe und serverseitige Eigenschaften sind über Wisej.NET-Versionen hinweg stabil; ein CSS-Selektor, der vom internen DOM eines Widgets abhängt, ist es nicht. Braucht ein Widget eines Drittanbieters oder ein eigenes Widget wirklich direktes CSS, begrenzen Sie den Selektor eng und dokumentieren Sie, warum das Theme-System für dieses Teil nicht zuständig ist. Führen Sie nach jedem Upgrade die responsive Screenshot-Matrix erneut aus und prüfen Sie jeden eigenen Appearance-Schlüssel und jeden widgetspezifischen Zustand.



Typische Fehler

Fokusrahmen entfernen, statt sie neu zu gestalten. Einen ungültigen Zustand nur über Farbe signalisieren. Ein Profil-Handler, der bei jedem Wechsel Zustand umschaltet oder die ganze Seite neu aufbaut. Bereiche mit `FillWeight` ohne `MinimumSize`. Ein integriertes Theme direkt bearbeiten. Globales CSS, das über interne DOM-Selektoren auf Standard-Controls geflickt wird. Ein Screenshot-Satz, der einmal existiert und nach einem Upgrade nie erneut erstellt wird.

Ein Grounding-Paket, aus dem ein Assistent antworten kann

Das letzte Ergebnis ist für zwei Leser geschrieben: den nächsten Entwickler und einen gegroundeten KI-Assistenten. Die Architekturnotiz erklärt jede Theme-, Layout- und Profilentcheidung und welcher Mechanismus dafür zuständig ist. Das Grounding-Paket ergänzt die Antwortregeln, denen der Assistent folgen muss: zuerst aus dem Kurs antworten und danach aus der offiziellen Dokumentation in der Quellenübersicht; vor fragilen internen Selektoren warnen und davor, ein gemeinsames Theme für jede Session zu ändern, wenn nur eine Session gemeint war; nie CSS-Media-Queries als primären responsiven

Mechanismus vorschlagen, wenn responsive Eigenschaften oder ClientProfile passen; und nach Profil und Bildschirmgröße fragen, wenn ein responsives Problem mehrdeutig ist. Die Quellenübersicht listet die Dokumentationsseiten hinter jeder Regel auf, damit der Assistent zitieren kann, statt zu raten.

Im Zentrum des Pakets steht der Entscheidungsbaum des Kurses, der der Reihe nach angewendet wird, bis eine Stufe passt:

- Zuerst das Theme, für die Identität der Standard-Controls.
- `AppearanceKey` für semantische Varianten.
- `CssClass` für bereichsbezogenes Anwendungs-CSS und Widgets von Drittanbietern.
- `CssStyle` nur für einmalige dynamische Werte.
- Layout-Container für adaptive Positionierung.
- Responsive Eigenschaften für im Designer gesetzte Profilwerte.
- `ResponsiveProfileChanged` für Laufzeitverhalten.
- ClientProfiles.json für die Profildefinitionen selbst.
- Governance im Produktivbetrieb für eine wartbare Auslieferung.

Das Bewertungsschema gewichtet Theme-Design und Konsistenz mit 25 Prozent, die Korrektheit des responsiven Layouts mit 25, ClientProfile und responsive Eigenschaften mit 20, Wartbarkeit und Governance mit 15 sowie Erklärung und Grounding-Abstimmung mit 15. Diese letzten dreißig Punkte verdienen Sie mit der Notiz und dem Paket: Wenn der Assistent allein anhand dessen, was Sie geschrieben haben, erklären kann, warum der Detailbereich auf dem Smartphone ein modales Formular ist und warum die Kennzahlenkarte eine `CssClass` statt einer Appearance ist, ist die Konsole fertig.



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:

- ✓ eine gebrandete UI für Desktop, Tablet und Smartphone ausliefern, die Theme, Layout-Container und ClientProfile konsistent nutzt.
- ✓ die visuelle Barrierefreiheit prüfen: Fokusindikatoren, ungültige Zustände, Kontrast, Touch-Ziele, Tab-Reihenfolge und das Verhalten ausgeblendeter Inhalte.
- ✓ eine Checkliste für den Produktivbetrieb zu Theme-Dateien, CSS-Dateien, ClientProfiles.json, Screenshots, Abnahmetests und upgradesicheren Anpassungen erstellen.
- ✓ Quellenübersicht und RAG-Materialien so bündeln, dass ein gegroundeter KI-Assistent Implementierungsfragen aus dem Kurs beantworten kann.

Lab- und Prüfungsleitfaden

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



Praktisches Lab

Stellen Sie das Abschlussprojekt Adaptive Operations Console fertig: ein eigenes Theme mit benannten Tokens und mindestens vier semantischen Appearances, bereichsbezogene CssClass-Nutzung, eine Desktop-Shell mit Flow-, Table- und Flex-Bereichen, eine ClientProfiles.json mit Verhalten für Smartphone, Tablet, Small Desktop und Desktop, responsive Eigenschaften plus ein idempotenter ResponsiveProfileChanged-Handler und eine im Testmodus sichtbare Profilanzeige. Führen Sie das Barrierefreiheits-Review für Fokusrahmen, ungültige Zustände, Kontrast, Touch-Ziele und Tab-Reihenfolge durch, füllen Sie die responsive QA-Matrix mit Screenshots für jedes Profil aus und schreiben Sie die Architekturnotiz und das Grounding-Paket, die jede Theme-, Layout- und Profilentcheidung erklären.

Ausgangspunkt

Öffnen Sie die Solution Adaptive Operations Console, die Sie durch die Module 1 bis 6 geführt haben: AdaptiveOps.theme und das bereichsbezogene AdaptiveOps.css existieren, die Shell besteht aus verschachtelten Containern mit Flow-, Table- und Flex-Bereichen, ClientProfiles.json listet sechs Profile auf, und der Handler aus Modul 6 zeigt bereits den Namen des aktiven Profils im Status-Label an. In diesem Lab wird nichts von Grund auf neu gebaut; jeder Schritt härtet, prüft oder dokumentiert, was vorhanden ist.

Vorgeschlagenes Vorgehen

- ✓ Prüfen Sie zuerst die visuellen Ebenen: Bestätigen Sie, dass das Theme von einem Basis-Theme mit benannten Tokens abgeleitet ist, zählen Sie die über `AppearanceKey` zugewiesenen semantischen Appearances (fügen Sie `destructive-button`, `kpi-card` oder `nav-item` hinzu, wenn es weniger als vier sind), und verschieben Sie jede verirrte Überschrift von `BackColor`, `ForeColor` oder `Font` ins Theme oder in eine `CssClass`.
- ✓ Härten Sie das Layout: Setzen Sie `MinimumSize` an jedem Bereich mit `FillWeight` oder `Docking` und `MaximumSize`, wo ein Bereich nicht wachsen darf, entfernen Sie `AutoSize` aus großen Container-Hierarchien und gehen Sie durch alle Breiten, um sicherzustellen, dass nichts zusammenfällt oder horizontal überläuft.
- ✓ Schreiben Sie die Profilogik als eine Methode `ApplyProfile(string name)` neu, die endgültige Werte zuweist und früh zurückkehrt, wenn sich das Profil nicht geändert hat; rufen Sie sie im Konstruktor mit `Application.ActiveProfile` und aus dem `ResponsiveProfileChanged`-Handler auf, beenden Sie das Abonnement in `Dispose` und zeigen Sie `lblProfile` nur, wenn ein Testmodus-Flag gesetzt ist.
- ✓ Führen Sie das Barrierefreiheits-Review durch Bedienen des Bildschirms durch: Gehen Sie mit Tab durch jeden Bereich und korrigieren Sie `TabIndex` und `TabStop`, lösen Sie die Validierung aus und bestätigen Sie, dass der ungültige Zustand neben der Farbe ein Symbol oder eine Meldung hat, prüfen Sie den Kontrast jedes Token-Paars, ergänzen Sie `ToolTipText` an reinen Symbol-Buttons und messen Sie die Touch-Ziele in Smartphone-Breite.

- ✓ Füllen Sie die responsive QA-Matrix (Desktop, Small Desktop, Tablet hoch und quer, Smartphone hoch und quer) mit dem erwarteten Profil, einem Screenshot und den Prüfungen für jede Zeile aus; schreiben Sie dann die Architekturnotiz und das Grounding-Paket: Entscheidungsbaum, Antwortregeln, Quellenübersicht und eine Frage, die der Assistent aus Ihrem Material beantworten muss.

Geforderte Ergebnisse

- ✓ Fertige Adaptive Operations Console mit eigenem Theme, semantischen Appearances, bereichsbezogenem CSS sowie Flow-, Table- und Flex-Bereichen
- ✓ ClientProfiles.json, responsive Eigenschaften und ein idempotenter ResponsiveProfileChanged-Handler mit Profilanzeige im Testmodus
- ✓ Barrierefreiheits-Review zu Fokusrahmen, ungültigen Zuständen, Kontrast, Touch-Zielen und Tab-Reihenfolge
- ✓ Responsive QA-Matrix mit Desktop-, Tablet- und Smartphone-Screenshots in beiden Ausrichtungen
- ✓ Architekturnotiz und KI-Grounding-Paket, die jede Theme-, Layout- und Profilentcheidung erklären

Wiederholungsfragen

- ✓ Ihr `ResponsiveProfileChanged` -Handler läuft nach einem Neuladen des Browsers zweimal für das Tablet-Profil. Welche Ihrer Zeilen garantiert, dass der zweite Durchlauf nichts ändert, und was sähe der Benutzer, wenn diese Zeile fehlte?
- ✓ Ein Reviewer bittet Sie, den Fokusrahmen von `action-button` zu entfernen, weil er „altmodisch aussieht“. Was ändern Sie stattdessen, welches Token steuert es, und welche zwei anderen Zustände prüfen Sie danach erneut im Theme Builder?
- ✓ Der Header des Ticket-Grids braucht eine Anpassung, die nur ein Selektor auf das interne DOM des Widgets erreicht. Was versuchen Sie zuerst, was muss zutreffen, bevor der Selektor erlaubt ist, und was schreiben Sie auf, damit das nächste Upgrade das Team nicht überrascht?

Abgabe des Abschlussprojekts

Dieses Lab ist zugleich die Abgabe des Abschlussprojekts für den Kurs. Reichen Sie das Quellprojekt, AdaptiveOps.theme, AdaptiveOps.css, ClientProfiles.json, die ausgefüllte responsive QA-Matrix mit ihren Screenshots, das Barrierefreiheits-Review, die Architekturnotiz und das Grounding-Paket ein. Seien Sie bereit, die Konsole durch jedes Profil zu führen und für jedes Control auf dem Bildschirm zu beantworten, welcher Mechanismus für sein Aussehen zuständig ist und warum.

Bewertungsbereich	Gewichtung	So sieht „exzellent“ aus
Theme-Design und Konsistenz	25%	Von einem Basis-Theme abgeleitetes Theme, Tokens überall wiederverwendet, semantische Appearances für jede fachliche Variante, keine spontanen Farb- oder Schriftüberschreibungen.
Korrektheit des responsiven Layouts	25%	Container bestimmen das Layout, Bereiche haben Mindestgrößen, jede Breite der QA-Matrix ist in beiden Ausrichtungen benutzbar.
ClientProfile und responsive Eigenschaften	20%	Profilreihenfolge nach First Match, im Designer zugewiesene Werte zuerst, ein idempotenter Handler für echte Verhaltensänderungen, Anzeige im Testmodus.
Wartbarkeit und Governance	15%	Dateien unter Versionskontrolle, Diffs geprüft, CSS-Ausnahmen dokumentiert, Screenshot-Matrix und Barrierefreiheits-Review vollständig und nach Upgrades wiederholbar.
Erklärung und KI-Grounding-Abstimmung	15%	Architekturnotiz und Grounding-Paket, aus denen ein Assistent jede Entscheidung erklären und vor fragilen Lösungen warnen kann.



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.