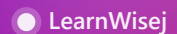


ClientProfile und responsive Eigenschaften

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Wie ClientProfiles.json von oben nach unten abgeglichen wird und so Application.ActiveProfile ergibt, im Designer pro Profil zugewiesene Werte responsiver Eigenschaften, ResponsiveProfileChanged für Verhalten, das responsive Eigenschaften nicht ausdrücken können, sowie Regeln für Entwurf und Test von Profilen. Was das für Wisej.NET-Entwickler bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Wie ClientProfiles.json von oben nach unten abgeglichen wird und so Application.ActiveProfile ergibt, im Designer pro Profil zugewiesene Werte responsiver Eigenschaften, ResponsiveProfileChanged für Verhalten, das responsive Eigenschaften nicht ausdrücken können, sowie Regeln für Entwurf und Test von Profilen.

1. Das Problem im Produktivbetrieb

Die Adaptive Operations Console hat jetzt eine gethemte Shell aus Flow-, Table- und Flex-Containern, die eine Größenänderung ohne Resize-Handler übersteht, zeigt aber allen denselben Bildschirm: Auf einem 390 Pixel breiten Smartphone konkurrieren Navigationsleiste, Ticket-Grid und Details-Editor um dieselbe Breite. CSS-Media-Queries können das nicht lösen, weil die Controls auf dem Server leben: Ein Panel auszublenden, einen Button auf reine Symbolanzeige umzustellen oder einen Bereich neu zu docken, ist eine serverseitige Eigenschaftsänderung, und ein Smartphone-Benutzer, der das Details-Panel verliert, braucht einen anderen Weg, ein Ticket zu bearbeiten, keine verkleinerte Kopie des Desktops. Dieses Modul fügt den Mechanismus hinzu, den Wisej.NET dafür vorsieht: Client-Profile, responsive Eigenschaften und das Event [ResponsiveProfileChanged](#).

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
ClientProfile	Ein benannter Satz gerätebezogener Abgleichregeln (Browserbreite, Bildschirmbreite, Gerätetyp, User-Agent, Querformat), der beschreibt, wie eine Klasse von Clients genutzt wird – kein Marketingname eines Geräts.
ClientProfiles.json	Die Datei, die Wisej.NET liest; Profile werden von oben nach unten ausgewertet, und das erste passende gewinnt. Die Reihenfolge der Einträge ist also Teil der Konfiguration.
Application.ActiveProfile	Das aktuell für diese Session ausgewählte Profil; sein <code>Name</code> ist das, was das Status-Label anzeigt und worauf der Profil-Handler verzweigt.
responsive Eigenschaft	Eine normale .NET-Eigenschaft, markiert mit <code>ResponsivePropertyAttribute</code> , die einen Wert pro Client-Profil hält und den passenden Wert anwendet, sobald sich das aktive Profil ändert.
Designer-first-Anpassung	Responsive Werte werden ausschließlich im Visual-Studio-Designer zugewiesen, was sie serialisiert, sichtbar und deterministisch hält; Code ist für Anpassungen da, die von Daten oder vom Laufzeitzustand abhängen.
ResponsiveProfileChanged	Das Event, das bei einer Änderung des aktiven Profils auf <code>Application</code> , auf Containern wie Form, Page, Desktop und UserControl sowie auf DataGridView und ListView ausgelöst wird; <code>ResponsiveProfileChangedEventArgs.CurrentProfile</code> enthält das neue Profil.
responsive QA-Matrix	Der Testplan mit Breite, Höhe, Ausrichtung, Gerätetyp, Touch, Scrollbars, Schriftskalierung und dem Profilnamen, den Sie bei jeder Größe erwarten.

3. Der rote Faden der Lektion

Wie ein Profil abgeglichen wird

Jeder Eintrag in `ClientProfiles.json` hat einen `name` und eine beliebige Kombination aus `device` (Mobile, Tablet oder Desktop), `landscape`, `minWidth` und `maxWidth` für den Browser-Viewport, `minScreenWidth` und `maxScreenWidth` für den physischen Bildschirm sowie einem Abgleich über `userAgent`. Wisej.NET liest die Datei, wertet die Profile von oben nach unten aus und wählt den ersten Treffer als `Application.ActiveProfile`. Nichts wird zusammengeführt: Ein Client passt zu genau einem Profil oder zu keinem, weshalb das Muster unten auf „Default“ zurückfällt, wenn `ActiveProfile` null ist.

Der Abgleich kann beim Laden der Seite oder bei einer Größenänderung des Browsers stattfinden, je nachdem, welche Eigenschaft eine Regel verwendet: Eine Regel zur Viewport-Breite wird neu ausgewertet, während der Browser gezogen wird, während Bildschirmgröße, Gerätetyp und User-Agent den Client selbst beschreiben. Immer wenn sich das ausgewählte Profil ändert, löst Wisej.NET `ResponsiveProfileChanged` aus. Für das Lab schreiben Sie sechs Profile in First-Match-Reihenfolge: Phone, Phone Landscape, Tablet, Tablet Landscape, Small Desktop und Desktop. Die beiden Phone-Einträge teilen `device` Mobile und unterscheiden sich nur in `landscape`; Small Desktop ist ein Desktop-Gerät mit `maxWidth` 1024; Desktop

ist die breite Regel ganz unten. Setzen Sie sie nach oben, verdeckt sie jedes Profil darunter, und nichts warnt Sie.

Responsive Eigenschaften gehören in den Designer

Responsive Eigenschaften sind gewöhnliche Eigenschaften, markiert mit `ResponsivePropertyAttribute` : `Visible` , `Display` , `Size` , `Location` , `Dock` , `Anchor` , `Font` und layoutbezogene Werte, sofern das Control sie unterstützt. Im Designer hält eine solche Eigenschaft für jedes Profil in `ClientProfiles.json` einen anderen Wert, und wenn sich das aktive Profil ändert, wendet das Control den zugehörigen Wert an. Der Designer ist der einzige unterstützte Weg, diese Werte zuzuweisen – eine Regel, keine Vorliebe: Dort zugewiesene Werte werden mit dem Formular serialisiert, sind für den nächsten Entwickler sichtbar und zur Laufzeit deterministisch, während im Code gesetzte Werte zur Entwurfszeit unsichtbar sind und vom Bild abweichen, das der Designer vom Bildschirm zeigt.

Im Lab ist jeder deterministische Unterschied in der Konsole Designer-Arbeit. In beiden Phone-Profilen erhalten das UserControl der Navigationsleiste und das Details-Panel `Visible` false. Die Toolbar-Buttons erhalten auf Phone und Tablet eine reine Symbol- `Display` , damit die Befehlsleiste in eine Zeile passt. In den Tablet-Profilen wechselt das `Dock` des Detailbereichs von Right auf Bottom, sodass er unter dem Ticket-Grid statt daneben sitzt. Keine dieser Änderungen hängt von Daten ab, also gehört keine in den Code; Code ist für Anpassungen da, die von Daten, Berechtigungen, einer Laufzeitberechnung oder dem Profil-Event selbst abhängen.

ResponsiveProfileChanged für Verhalten

Manche Anpassungen sind keine Werte. Auf einem Smartphone kann der Details-Editor überhaupt kein Seitenpanel sein; er muss als modale `Form` über `ShowDialog` geöffnet werden, und das Status-Label soll den Profilnamen anzeigen und protokollieren, damit der Support weiß, welches Layout ein Benutzer sieht. Dafür ist `ResponsiveProfileChanged` da. Es wird auf `Application` ausgelöst, auf ContainerControl-Typen wie `Form`, `Page`, `Desktop` und `UserControl` sowie auf ausgewählten komplexen Controls wie `DataGridView` und `ListView`. Behandeln Sie es auf `Application` für anwendungsweites Verhalten und auf einem zentralen Container, wenn die Logik lokal zu diesem Bereich gehört; die Event-Argumente stellen `CurrentProfile` bereit.

Das Muster unten ist die Form, die das Lab erwartet: einmal abonnieren, das Profil beim Start einmal aus `Application.ActiveProfile` anwenden, damit das anfängliche Layout stimmt, das Event über dieselbe Methode `ApplyProfile` leiten und das Abonnement in `Dispose` beenden, weil `Application` die Seite überlebt. Halten Sie `ApplyProfile` günstig und idempotent: Ein Smartphone, das zwischen Phone und Phone Landscape gedreht wird, löst sie wiederholt aus, und ein bereits geöffneter Editor muss wiederverwendet, nie erneut geöffnet werden. Typische Arbeit im Event: weniger Grid-Spalten laden, das Debounce-Intervall einer Suche ändern, ein Details-Panel in ein modales Formular umwandeln, eine andere Datenvorlage anwenden.

```
// MainPage.cs – eine Methode wendet das Profil beim Start und bei jeder Änderung an
Application.ResponsiveProfileChanged += Application_ResponsiveProfileChanged;
ApplyProfile(Application.ActiveProfile?.Name);

private void Application_ResponsiveProfileChanged(object sender,
ResponsiveProfileChangedEventArgs e)
{
    ApplyProfile(e.CurrentProfile?.Name);
}

private void ApplyProfile(string profileName)
{
    var phone = profileName != null && profileName.StartsWith("Phone");
    this.lblProfile.Text = profileName ?? "Default";           // angezeigt und
    protokolliert
    if (phone && this.ticketEditor == null) OpenDetailsAsModal(); // Seitenpanel → modal,
    nur einmal
}
```

Varianten entwerfen, die Verhalten ändern

Profile sollten Verhalten beschreiben, nicht Geräte. Phone, Tablet Landscape, Small Desktop, Kiosk und Touch Workstation benennen jeweils eine Art, die Konsole zu nutzen, und jede Variante sollte ändern, was die Konsole tut: Das Smartphone erhält einen modalen Editor und einen Menü-Button, das Tablet stapelt den Detailbereich unter dem Grid, der Small Desktop behält die Shell mit einer reinen Symbol-Toolbar, der Desktop zeigt alles. Verwenden Sie Breitenregeln für das Layout, Gerätereignisse für Annahmen zur Eingabe wie Touch-Ziele und ausgeblendete Scrollbars, `userAgent` nur wenn nötig und Querformatregeln nur, wenn die Ausrichtung die UI wirklich verändert. Ein Profil, das nichts außer einer CSS-Abmessung ändert, hätte es nicht gebraucht.

Testen Sie auf beiden Wegen. Echtes Verkleinern des Browsers prüft die Breitenregeln; die Geräteemulation prüft die Regeln zu Gerät, Bildschirmbreite und User-Agent, und nach einem Gerätewechsel im mobilen Emulator von Chrome müssen Sie die Seite neu laden, damit diese Profile neu ausgewertet werden. Die responsive QA-Matrix hat eine Zeile pro Szenario, von Desktop bei 1440 mal 900 bis hinunter zu Phone bei 390 mal 844 und Phone Landscape bei 844 mal 390, jeweils mit dem erwarteten Profilnamen und den Prüfungen, die das Verhalten belegen: Navigation sichtbar, Details gestapelt, modaler Editor erreichbar, kein horizontaler Überlauf.

- Schmale oder spezifische Profile vor breiten; der erste Treffer gewinnt, und nichts wird zusammengeführt.
- Responsive Werte im Designer; `ResponsiveProfileChanged` nur, wenn ein Wert nicht ausreicht, und dann günstig und idempotent.
- In Test-Builds den Profilnamen anzeigen und jedes ausgeblendete Control mit seiner mobilen Alternative dokumentieren.



Typische Fehler

Das breite Desktop-Profil zuerst aufführen, sodass Phone- und Tablet-Profile nie getroffen werden. Responsive Werte im Code statt im Designer zuweisen. Jede Anpassung in den Event-Handler packen, auch die, die ein `Visible` - oder `Dock` -Wert ausgedrückt hätte. Den modalen Editor bei jeder Drehung erneut öffnen, weil der Handler nicht idempotent ist. Ein Panel auf dem Smartphone ausblenden, ohne eine mobile Alternative für seinen Inhalt.



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, wie ClientProfiles.json gelesen und abgeglichen wird und wie Application.ActiveProfile ausgewählt wird.
- ✓ responsive Eigenschaften im Designer für Eigenschaften wie Visible, Display, Size, Location, Dock, Anchor, Font und, sofern verfügbar, layoutbezogene Werte verwenden.
- ✓ ResponsiveProfileChanged auf Application und zentralen Containern behandeln, wenn profilspezifische Logik nötig ist.
- ✓ Varianten für Smartphone, Tablet, Small Desktop und Desktop entwerfen, die das Verhalten der UI ändern, nicht nur CSS-Abmessungen.

Lab- und Prüfungsleitfaden

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



Praktisches Lab

Fügen Sie der Adaptive Operations Console eine ClientProfiles.json mit den Profilen Phone, Phone Landscape, Tablet, Tablet Landscape, Small Desktop und Desktop hinzu, von schmal nach breit geordnet. Verwenden Sie im Designer responsive Eigenschaften, um Navigationsleiste und Details-Panel auf dem Smartphone auszublenden, die Toolbar-Buttons auf eine reine Symbol-Display umzustellen und den Detailbereich auf dem Tablet unter dem Grid neu zu docken. Behandeln Sie ResponsiveProfileChanged, um den Namen des aktiven Profils im Status-Label anzuzeigen, jeden Wechsel zu protokollieren und den Details-Editor auf dem Smartphone als modales Formular statt als Seitenpanel zu öffnen.

Ausgangspunkt

Öffnen Sie die Adaptive Operations Console so, wie Modul 5 sie hinterlassen hat: Die Shell besteht aus verschachtelten Containern mit einem Flow-Kennzahlenkartenbereich, einem Ticket-Editor mit TableLayoutPanel im Details-UserControl und einem Flex-Arbeitsbereich, das AdaptiveOps-Theme und das bereichsbezogene Stylesheet sind geladen, und das Status-Label aus Modul 1 meldet weiterhin die Browserbreite. Eine `ClientProfiles.json` gibt es noch nicht, daher ist `Application.ActiveProfile` für jeden Client null, und jede Breite zeigt die Desktop-Shell.

Vorgeschlagenes Vorgehen

- ✓ Schreiben Sie `ClientProfiles.json` in das Stammverzeichnis der Anwendung, mit sechs Einträgen in First-Match-Reihenfolge (Phone, Phone Landscape, Tablet, Tablet Landscape, Small Desktop, Desktop): Die beiden Phone-Einträge verwenden `device` Mobile und unterscheiden sich über `landscape`, das Tablet-Paar macht es ebenso mit Tablet, Small Desktop ist ein Desktop-Gerät mit `maxWidth` 1024, und Desktop ist die breite Regel am Schluss. Starten Sie einmal und prüfen Sie, dass die Profilnamen im Designer erscheinen.
- ✓ Geben Sie im Designer dem UserControl der Navigationsleiste und dem Details-Panel für beide Phone-Profile den `Visible`-Wert false und prüfen Sie, dass das Grid im Arbeitsbereich weiterhin seine `MinimumSize` hat, damit das Smartphone-Layout nicht auf null zusammenfällt.
- ✓ Setzen Sie, ebenfalls im Designer, die `Display` der Toolbar-Buttons für die Profile Phone und Tablet auf reine Symbolanzeige und ändern Sie das `Dock` des Detailbereichs für beide Tablet-Profile von Right auf Bottom, sodass er unter dem Ticket-Grid sitzt; lassen Sie die Desktop-Werte unverändert.
- ✓ Abonnieren Sie in `MainPage` einmalig `Application.ResponsiveProfileChanged`, schreiben Sie eine Methode `ApplyProfile(string profileName)`, die `lblProfile.Text` setzt und pro Wechsel eine Logzeile schreibt, rufen Sie sie bei der Konstruktion mit `Application.ActiveProfile?.Name` und aus dem Handler mit `e.CurrentProfile?.Name` auf und beenden Sie das Abonnement in `Dispose`.
- ✓ Lassen Sie `ApplyProfile` für die Phone-Profile den Ticket-Editor in einer modalen `Form` über `ShowDialog` öffnen und das Formular wiederverwenden, falls es bereits offen ist; testen Sie

dann durch Ziehen des Browsers und per Geräteemulation mit einem Neuladen nach jedem Gerätewechsel, und halten Sie das Profil-Label bei jeder Größe fest.

Geforderte Ergebnisse

- ✓ ClientProfiles.json mit den Profilen Phone, Phone Landscape, Tablet, Tablet Landscape, Small Desktop und Desktop in First-Match-Reihenfolge
- ✓ Im Designer zugewiesene Werte responsiver Eigenschaften, die Navigationsleiste und Details-Panel auf dem Smartphone ausblenden
- ✓ Reine Symbol-Display der Toolbar und neu gedockter Detailbereich für Tablet, gesetzt über responsive Eigenschaften
- ✓ ResponsiveProfileChanged-Handler, der den Namen des aktiven Profils anzeigt und protokolliert
- ✓ Details-Editor, der auf dem Smartphone über den Profil-Handler als modales Formular geöffnet wird

Wiederholungsfragen

- ✓ Würde Small Desktop an den Anfang von ClientProfiles.json verschoben: Welche der anderen fünf Profile könnten dann noch getroffen werden, und was sähe ein Tablet im Querformat?
- ✓ Das Neu-Docken auf dem Tablet liegt im Designer, der modale Editor auf dem Smartphone im Code. Was entscheidet, welcher Mechanismus für welche Änderung zuständig ist, und was ginge schief, wenn Sie sie vertauschen?
- ✓ Ein Tester hat ein Smartphone viermal gedreht, und Ihr Handler ist viermal gelaufen. Welcher Zustand muss nach dem vierten Durchlauf identisch mit dem nach dem ersten sein, und wie garantiert Ihr Code das?



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.