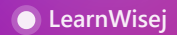


Mobile Shells mit .NET MAUI

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Die App in einer hybriden .NET-MAUI-Shell für Android und iOS hosten, Navigation und Verhalten der Zurück-Taste, App-Lebenszyklus und Plattform-Eigenheiten. Was das für ein Team bedeutet, das eine Wisej.NET-App über den Browser-Tab hinaus ausliefert – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Die App in einer hybriden .NET-MAUI-Shell für Android und iOS hosten, Navigation und Verhalten der Zurück-Taste, App-Lebenszyklus und Plattform-Eigenheiten.

1. Das Problem im Produktivbetrieb

Die Techniker von FieldOps arbeiten mit einem Smartphone im Technikraum, im Keller oder im Transporter, und ein Browser-Tab ist dafür ein schlechter Container: kein Icon für den Startbildschirm, das man verteilen könnte, kein Store- oder Unternehmenskanal und kein Weg zu Kamera und GPS, die Modul 5 brauchen wird. Das mobile Betriebssystem ist zudem ein weit aggressiverer Host – es suspendiert die App, sobald ein Anruf eingeht, zeichnet sie unter einer Notch neu und gibt dem Techniker eine System-Zurück-Geste in die Hand, um die FieldOps nie gebeten hat.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Hybride .NET-MAUI-Shell	Eine kleine native App für Android und iOS, deren einzige Seite Ihre Wisej.NET-Anwendung in einer Web-View hostet – der Techniker sieht also die C#-Bildschirme, die Sie bereits ausliefern.
Nativer Web-View-Host	Die in diese Seite eingebettete Browser-Engine der Plattform – Android System WebView, WKWebView unter iOS –, die denselben Client ausführt wie ein Desktop-Browser; deshalb wird kein Bildschirm für Mobilgeräte neu geschrieben.
Hybrid-Bridge	Der bidirektionale Kanal zwischen der gehosteten Seite und nativem Code, damit serverseitiger C#-Code das Gerät um etwas bitten und die Antwort als Event statt als Seitennavigation erhalten kann.
MAUI-Anwendungslebenszyklus	Die Events, die das Betriebssystem rund um die App auslöst – Fenster erstellt, aktiviert, deaktiviert, gestoppt und fortgesetzt, dazu <code>OnSleep</code> und <code>OnResume</code> –, von denen ein Browser-Tab kein einziges auslöst.
Überlebensfenster der Session	Die Zeitspanne im Hintergrund, in der die Wisej.NET-Session auf dem Server noch lebt, sodass sich die App beim Fortsetzen mit dem vorhandenen Zustand verbinden kann, statt neu zu starten.
Safe-Area-Insets	Der gerätespezifische Innenabstand, der Inhalte von Notch, Statusleiste und Home-Indicator-Streifen fernhält – von der Plattform gemeldet, nicht aus der Bildschirmgröße geraten.
Zuordnung der Zurück-Navigation	Die Regel, die festlegt, was die Zurück-Geste von Android bewirkt: innerhalb der eigenen Navigation der App zurückgehen oder das Verlassen bestätigen lassen – niemals stillschweigend einen halb ausgefüllten Arbeitsauftrag zerstören.

3. Der rote Faden der Lektion

Ein Shell-Projekt, zwei Stores, derselbe Server

Die mobile Shell von FieldOps ist eine .NET-MAUI-Anwendung mit einer einzigen Aufgabe: ein Fenster erstellen, eine Web-View hosten und sie auf den FieldOps-Server richten. Die Hybrid-Unterstützung von Wisej.NET liefert diese Shell als Projekt, das Sie bauen und anpassen, statt es von Grund auf zu schreiben, und die gehostete Seite ist die gewöhnliche FieldOps-Anwendung aus Modul 1. An `Wisej.Web`, dem Visual-Studio-Designer oder `Application.Session` ändert sich nichts; die Controls leben weiterhin auf dem Server, und die Web-View rendert weiterhin ihr Spiegelbild.

Ein MAUI-Projekt zielt gleichzeitig auf Android und iOS, und die Plattformordner darunter enthalten nur, was sich wirklich unterscheidet: die Android-Activity und das Manifest, die iOS- `Info.plist` und Entitlements, die Icons und Splash-Screens. Halten Sie die Serveradresse in einem einzigen Konfigurationswert, den die Shell beim Start liest – so zeigt Debug auf einen Laptop, Test auf Staging und Release in den Produktivbetrieb, ohne dass Code geändert werden muss.

Diese Aufteilung bestimmt auch die Wirtschaftlichkeit der Shell. FieldOps liefert neuen Servercode in dem Moment aus, in dem Sie ihn deployen; die Shell muss nur neu gebaut werden, wenn sich die Shell selbst ändert – ein neues Icon, eine neue Berechtigung, eine neue Bridge-Fähigkeit. Alles, was auf dem Server leben kann, sollte dort leben, denn den Server können Sie noch heute Nachmittag korrigieren, während sich eine Store-Prüfwarteschlange in Tagen bemsist.

Hintergrundbetrieb ist der Normalfall, nicht der Fehlerfall

Auf einem Smartphone wird die App ständig unterbrochen: ein Anruf, eine Teilenummer, die in einer anderen App nachgeschlagen wird, ein gesperrter Bildschirm beim Hochsteigen einer Leiter. Jede dieser Unterbrechungen löst die MAUI-Lebenszyklussequenz aus – Fenster deaktiviert, dann gestoppt, auf dem Rückweg fortgesetzt, wobei parallel die Überschreibungen `OnSleep` und `OnResume` auf Anwendungsebene ausgelöst werden. Behandeln Sie sie explizit, denn das Standardverhalten einer aus dem Speicher verdrängten Web-View ist das Neuladen der Seite, und ein Neuladen bedeutet eine neue Wisej.NET-Session: Der halb eingetippte Zählerstand ist weg.

Das funktionierende Design behandelt einen kurzen Hintergrundaufenthalt als Pause und einen langen als Neustart. Beim Stoppen merken Sie sich die Uhrzeit und lassen die Session ruhen. Beim Fortsetzen vergleichen Sie die verstrichene Zeit mit dem Session-Timeout, das für den FieldOps-Server konfiguriert ist; sollte die Session noch leben, verbinden Sie sich neu und lassen den Server den aktuellen Zustand zurückpushen, statt die Web-View erneut navigieren zu lassen. Ist die Lücke länger als das Timeout oder meldet die Plattform, dass die App wegen Speichermangels beendet wurde, stellen Sie gezielt wieder her: neu laden und dann den Arbeitsauftrag anhand einer Kennung wieder öffnen, die die Shell lokal gespeichert hat. Behandeln Sie eine Warnung bei knappem Speicher genauso wie ein Stoppen.

```
protected override Window CreateWindow(IActivationState state)
{
    var window = base.CreateWindow(state);
    window.Stopped += (s, e) => ShellState.Backgrounded(DateTime.UtcNow);
    window.Resumed += (s, e) => ShellState.Resume(); // neu verbinden, nicht neu laden
    return window;
}

protected override void OnSleep() => ShellState.Backgrounded(DateTime.UtcNow);
protected override void OnResume() => ShellState.Resume();
```

Die Serverseite dieses Vertrags ist genauso wichtig: Halten Sie das Session-Timeout von FieldOps lang genug, um eine realistische Unterbrechung abzudecken, und lassen Sie den Editor für Arbeitsaufträge bei jeder Feldänderung einen Entwurf speichern, nicht erst beim Absenden.

Zurück, Notches und der Streifen am unteren Rand

Die Zurück-Geste von Android ist ein Event auf Systemebene, und wer sie ignoriert, bekommt das Plattform-Standardverhalten: Die Activity wird geschlossen – ein Techniker, der auf der Detailseite eines Arbeitsauftrags zurückwischt, verliert also die App, statt zur Liste zurückzukehren. Fangen Sie die Zurück-Navigation in der Shell ab und leiten Sie sie zuerst an die Web-App weiter – die Detailseite kehrt zur Liste zurück, ein offener Dialog schließt sich – und fallen Sie erst dann auf eine Bestätigungsabfrage zurück, wenn es nichts mehr gibt, wohin man zurückkehren kann. Der Zustand, der die Frage „Gibt es etwas, wohin

man zurückkehren kann?“ beantwortet, lebt auf dem Server, die Shell muss also die App fragen, statt zu raten: Führen Sie in FieldOps selbst einen kleinen, expliziten Navigationsstapel. iOS hat keine Zurück-Taste, das eigene Zurück-Bedienelement der App muss daher in jedem Fall vorhanden sein.

Safe Areas sind die andere sichtbare Plattform-Eigenheit. Ein Smartphone mit Notch reserviert den oberen Streifen, und ein modernes iPhone reserviert am unteren Rand einen Streifen für den Home-Indicator; dort gezeichnete Inhalte sind verdeckt oder nicht antippbar. Lesen Sie die von der Plattform gemeldeten Insets und wenden Sie sie als Innenabstand um den gehosteten Inhalt an, sodass die App-Leiste von FieldOps unter der Statusleiste und die Buttons zur Statusaktualisierung über dem Home-Indicator liegen. Versehen Sie den Host mit Innenabstand, statt feste Pixelwerte in das Wisej.NET-Layout zu schreiben – dann bleiben dieselben Bildschirme korrekt, in einem einfachen Browser, auf einem Gerät ohne Notch und im Querformat.

Emulatoren, Geräte und zwei Build-Befehle

Der Build-Zyklus ist plattformspezifisch und sollte eingerichtet werden, bevor Sie irgendeinen Shell-Code schreiben. Android braucht das SDK und mindestens ein Image für den Android Emulator; ein Debug-Lauf ist `dotnet build -t:Run -f net8.0-android` oder das entsprechende Ziel in Visual Studio. iOS braucht einen Mac im Zyklus – den iOS Simulator und einen Build wie `dotnet build -f net8.0-ios` –, auch wenn Sie unter Windows entwickeln und sich remote mit diesem Mac koppeln. Hot Restart verkürzt den Zyklus, und weil die meisten Änderungen an FieldOps serverseitig sind, brauchen viele überhaupt keinen Rebuild: C#-Code ändern, Server neu starten, in der Shell nach unten ziehen.

Das Debugging erstreckt sich über zwei Laufzeitumgebungen: Breakpoints im C#-Code der Shell decken Lebenszyklus und Bridge-Aufrufe ab, Breakpoints im Servercode die Bildschirme. Kann der Emulator Ihren Entwicklungsserver nicht erreichen, liegt es fast immer an der Loopback-Adresse – `localhost` eines Emulators ist das emulierte Gerät, nicht Ihr Rechner – oder an einem selbstsignierten Zertifikat, das die Plattform ablehnt. Beweisen Sie die Shell anschließend auf echter Hardware: Touch-Ziele, die Tastatur, die ein Feld verdeckt, und die Verdrängung bei Speicherdruck verhalten sich dort anders, und die Screenshots aus beiden Emulatoren aus dem Lab sind die Referenz, mit der Sie diesen Gerätelauflauf vergleichen.



Packaging-Grundsatz

Die App bleibt eine Wisej.NET-Web-App, die von Ihrem Server ausgeliefert wird; die Shell ist nur das Transportmittel. Sorgen Sie dafür, dass jede Funktion in einem einfachen Browser funktioniert, und ergänzen Sie Gerätefähigkeiten dann als progressive Erweiterungen hinter einem Capability-Check.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ die Wisej.NET-App in einer hybriden .NET-MAUI-Shell für Android und iOS hosten.
- ✓ mobile Lebenszyklus-Events – Hintergrund, Fortsetzen, knapper Speicher – behandeln, ohne die Session zu verlieren.
- ✓ Plattformnavigation, Zurück-Taste und Safe Areas auf die Web-App abbilden.
- ✓ Emulatoren, Debugging auf dem Gerät und die Build-Pipeline für beide Plattformen einrichten.

Lab- und Prüfungsleitfaden

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



Praxis-Lab

Bauen Sie die mobile Shell von FieldOps mit .NET MAUI: eine hybride Seite, die die App unter Android und iOS hostet, eine Session, die den Hintergrundbetrieb übersteht und bei der Rückkehr fortgesetzt wird, die Android-Zurück-Taste, abgebildet auf die Navigation in der App, Safe-Area-Innenabstand für Geräte mit Notch und ein Debug-Build, der auf einem Emulator pro Plattform läuft, mit Screenshots der Liste der Arbeitsaufträge.

Ausgangspunkt

Sie öffnen die FieldOps-Projektmappe aus Modul 1 – die Liste der Arbeitsaufträge, die Detailseite und den Ablauf zur Statusaktualisierung, die bereits in einem einfachen Browser laufen und sich seit Modul 2 als PWA installieren lassen. An diesen Bildschirmen ändert sich in diesem Lab nichts; Sie fügen neben dem Webprojekt einen zweiten Kopf hinzu, eine .NET-MAUI-Shell, die dieselbe serverseitig gerenderte App auf einem Smartphone hostet.

Vorgeschlagenes Vorgehen

- ✓ Fügen Sie das MAUI-Shell-Projekt zur FieldOps-Projektmappe hinzu, zielen Sie auf Android und iOS, und lesen Sie die Serveradresse aus der Konfiguration, damit Debug- und Release-Builds auf unterschiedliche Hosts zeigen.
- ✓ Hosten Sie die FieldOps-App in der Web-View der Shell und prüfen Sie, dass die Liste der Arbeitsaufträge, die Detailseite und eine Statusaktualisierung vollständig funktionieren, bevor Sie irgendetwas Plattformspezifisches anfassen.
- ✓ Behandeln Sie die Fenster-Events für Stoppen, Fortsetzen und knappen Speicher: Merken Sie sich den Zeitpunkt, zu dem die App in den Hintergrund ging, und verbinden Sie sich beim Fortsetzen innerhalb des Session-Timeouts neu, statt die Seite neu zu laden.
- ✓ Fangen Sie die Android-Zurück-Navigation ab und leiten Sie sie in den Navigationsstapel von FieldOps, mit einer Bestätigungsabfrage nur dann, wenn es nichts mehr gibt, wohin man zurückkehren kann; wenden Sie die gemeldeten Safe-Area-Insets als Innenabstand um den gehosteten Inhalt an.
- ✓ Führen Sie einen Debug-Build im Android Emulator und im iOS Simulator aus, erfassen Sie jeweils die Liste der Arbeitsaufträge und notieren Sie jeden Unterschied, den Sie pro Plattform berücksichtigen mussten.

Geforderte Ergebnisse

- ✓ Hybrides .NET-MAUI-Shell-Projekt für Android und iOS
- ✓ Lebenszyklus-Behandlung für Hintergrund und Fortsetzen
- ✓ Zuordnung von Zurück-Taste und Safe Areas
- ✓ Emulator-Builds für beide Plattformen

- ✓ Screenshots aus beiden Emulatoren

Wiederholungsfragen

- ✓ Ein Techniker ist mitten in einer Statusaktualisierung, als ein dreiminütiger Anruf eingeht. Was muss auf dem Server und in der Shell gelten, damit die halb fertige Aktualisierung danach noch da ist, und was ändert sich, wenn der Anruf eine Stunde dauert?
- ✓ Die Shell weiß nicht, welcher FieldOps-Bildschirm geöffnet ist. Wie entscheidet eine Zurück-Geste zwischen dem Zurückkehren von der Detailseite, dem Schließen eines Dialogs und dem Verlassen der App, und wo lebt dieser Zustand tatsächlich?
- ✓ Welche Änderungen dieses Moduls würde ein Techniker noch bemerken, wenn er FieldOps im Browser des Smartphones statt in der Shell öffnet, und welche würden verschwinden? Was sagt Ihnen das darüber, wo der Safe-Area-Innenabstand hingehört?



Packaging-Grundsatz

Die App bleibt eine Wisej.NET-Web-App, die von Ihrem Server ausgeliefert wird; die Shell ist nur das Transportmittel. Sorgen Sie dafür, dass jede Funktion in einem einfachen Browser funktioniert, und ergänzen Sie Gerätefähigkeiten dann als progressive Erweiterungen hinter einem Capability-Check.