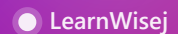


Hosting-Modelle jenseits des Browsers

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Browser, Progressive Web App, Desktop-Shell und mobile Shell im Vergleich: Architektur, Fähigkeiten, Update-Modell und Kosten. 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**

Browser, Progressive Web App, Desktop-Shell und mobile Shell im Vergleich: Architektur, Fähigkeiten, Update-Modell und Kosten.

1. Das Problem im Produktivbetrieb

FieldOps ist eine Außendienstanwendung: Techniker erhalten Arbeitsaufträge, nehmen Fotos und Unterschriften auf, navigieren zu Einsatzorten und arbeiten mit lückenhafter Verbindung. Als gewöhnliche Wisej.NET-Anwendung läuft sie bereits in jedem Browser – das genügt den Disponenten im Büro, aber nicht dem Techniker, der mit einem Balken Empfang und dem Handy im Handschuh im Technikraum steht. Die Frage dieses Moduls lautet nicht „Können wir das bauen?“, sondern „Wie erreicht die App die Menschen, die sie nutzen?“ – und die Antwort darauf ist eine Architekturentscheidung, kein nachträglicher Packaging-Schritt.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Bereitstellungsmodell	Der Weg, auf dem die Anwendung zum Benutzer gelangt: ein Browser-Tab, eine installierte PWA, ein Desktop-Fenster oder eine über den Store verteilte mobile App. Es verändert Verteilung, Updates und Gerätezugriff – niemals den C#-Code, der den Bildschirm aufbaut.
Browser-Engine des Hosts	Jedes Modell rendert dieselbe servergesteuerte Oberfläche in einer Browser-Engine: im eigenen Browser des Benutzers oder in einer eingebetteten Engine – WebView2 unter Windows, WebKit unter macOS und iOS, die System-WebView unter Android.
Wisej.NET Hybrid	Die auf .NET MAUI basierende Shell des Frameworks, die Ihre bestehende Wisej.NET-Anwendung unter Android, iOS und auf dem Desktop hostet und eine Bridge zu Gerätefunktionen wie Kamera, Geolocation, Dateien und Benachrichtigungen ergänzt.
Geräte-Bridge	Der bidirektionale Kanal zwischen Ihrem serverseitigen C#-Code und der nativen Shell: Der Server fordert eine Geräteaktion an, die Shell führt sie aus, und das Ergebnis kommt als Event in Ihren Code zurück, nicht als Rückgabewert.
Capability-Check	Eine Laufzeitprüfung dessen, was die aktuelle Shell tatsächlich kann. Damit wird eine Funktion angezeigt, ausgeblendet oder ersetzt, sodass sich ein Bildschirm in jedem Bereitstellungsmodell korrekt verhält.
Cache der Anwendungs-Shell	Das statische HTML, CSS, die Skripte und Icons, die ein PWA-Service-Worker lokal vorhalten kann. Das ist der Teil, der auch ohne Netzwerk noch lädt – der interaktive, serverseitig gerenderte Teil der App gehört nicht dazu.
Update-Asymmetrie	Der Server kann jederzeit neu bereitgestellt werden, und jede Session erhält beim Neuladen den neuen Build – ein Desktop-Installer oder ein Store-Paket dagegen braucht zuerst ein Release und eine Installation.

3. Der rote Faden der Lektion

Eine Anwendung, vier Eingangstüren

Eine Wisej.NET-Anwendung ist ein Serverprozess. Ihre `Window` - und `Page` -Objekte, deren Controls und Event-Handler leben im Arbeitsspeicher des Servers, und der Browser hält ein gerendertes Abbild, das Benutzeraktionen über eine HTTP- und WebSocket-Verbindung zurücksendet. Nichts an dieser Anordnung interessiert sich dafür, welche Browser-Engine am anderen Ende sitzt. Genau diese eine Tatsache macht vier Bereitstellungsmodelle aus einem einzigen Deployment möglich: dieselbe URL, dieselbe Session, derselbe `Application.Session` -Zustand.

Im Browser-Tab tippt der Benutzer eine Adresse ein oder klickt auf einen Link und erhält die App. Als Progressive Web App wird dieselbe URL aus dem Browser heraus installiert: Ein Manifest gibt ihr einen Namen, ein Icon und ein eigenständiges Fenster ohne Adressleiste, und ein Service Worker cacht die statische Shell, sodass sich der Start unmittelbar anfühlt. In einer Desktop-Shell öffnet ein kleines natives Programm ein Fenster mit einer eingebetteten Browser-Engine – WebView2 unter Windows, WebKit unter

macOS – und richtet sie auf Ihren Server; der Benutzer sieht einen Eintrag in der Taskleiste und eine Titelleiste, und die Anwendung darin ist dieselbe, die auch die Disponenten verwenden. In einer mobilen Shell hostet Wisej.NET Hybrid die Anwendung in einer .NET-MAUI-App für Android und iOS, die aus einem Store oder über einen Unternehmenskanal installiert wird und auf das Gerät zugreifen kann.

Für FieldOps bedeutet das: Die Liste der Arbeitsaufträge wird ein einziges Mal geschrieben und nie portiert oder geforkt. Die Shells sind Container – sie unterscheiden sich darin, was die App umgibt, nicht darin, was in ihr läuft.

Der Vergleich, der tatsächlich entscheidet

Fünf Spalten entscheiden über das Modell, und die erste ist die ehrliche. **Offline-Verhalten:** Eine serverseitige Oberfläche braucht ihren Server. Bricht die Verbindung ab, kann keine Shell Ihre Event-Handler weiter ausführen, denn die Handler befinden sich nicht auf dem Gerät. Eine PWA oder eine native Shell kann die Shell cachet und einen gestalteten Offline-Bildschirm mit der Zusammenfassung der zuletzt synchronisierten Arbeitsaufträge anzeigen, und sie kann eine Aktion des Technikers in eine Warteschlange stellen, damit sie gesendet wird, sobald das Signal zurückkehrt – aber sie kann keinen Arbeitsauftrag öffnen, den das Gerät noch nie gesehen hat. Mehr zu versprechen ist genau der Weg, auf dem Außendienst-Apps schon in der ersten Woche das Vertrauen ihrer Benutzer verlieren.

Beim **Gerätezugriff** trennen sich die Shells. Ein Browser und eine PWA erhalten, was die Webplattform nach einer Berechtigungsabfrage gewährt: Dateiauswahl, ein Bild über ein Upload-Control, grobe Geolocation, Web-Benachrichtigungen, wo sie unterstützt werden. Eine native Shell ergänzt die Bridge – eine echte Kamerasitzung, Standortbestimmung im Hintergrund, das Dateisystem, Push-Benachrichtigungen, Biometrie. Die **Verteilung** reicht von einer URL, die Sie per E-Mail verschicken, über einen Installer bis zu einer Store-Prüfung, die Tage dauern kann. **Updates** folgen derselben Reihenfolge: Der Server ist sofort aktuell, ein Desktop-Paket braucht einen Update-Kanal, ein Store-Build eine Einreichung. Die **Kosten** sind die Summe aus Zertifikaten, Entwicklerkonten, Testgeräten und den Prüfzyklen, für die Sie von nun an dauerhaft verantwortlich sind.

- Disponenten am Schreibtisch: Browser, sonst nichts.
- Techniker, die Kamera, Standort und Benachrichtigungen brauchen: mobile Shell.
- Lager-Kiosksysteme, die einen wackligen WLAN-Ausfall überstehen müssen: PWA mit einer ehrlichen Offline-Seite.

Ein Server, eine Codebasis, viele Shells

Die Architektur, die das kostengünstig hält, ist bewusst unspektakulär: ein Wisej.NET-Server-Deployment, eine Projektmappe und Shells, die überhaupt keine Geschäftslogik enthalten. Die Liste der Arbeitsaufträge, die Detailseite und der Ablauf zur Statusaktualisierung sind gewöhnliche Wisej.NET-Bildschirme, die einen **WorkOrderService** aufrufen. Die Shell-Projekte, die in späteren Modulen hinzukommen, tun genau zwei Dinge: Sie hosten die auf Ihren Server gerichtete Browser-Engine und stellen ihr Gerätefunktionen zur Verfügung.

Das Bindeglied zwischen beiden ist der Capability-Check. Statt „Laufe ich unter Android?“ zu prüfen, prüfen Sie, was Sie brauchen: Kann diese Shell ein Foto aufnehmen, einen Standort lesen, eine Benachrichtigung auslösen? Ermitteln Sie diese Fähigkeiten einmal beim Start der Session, legen Sie sie im Session-Zustand ab und lassen Sie jeden Bildschirm sie auslesen. Ein Bildschirm passt sich dann an, statt pro Plattform zu

verzweigen, und eine neue Shell, die nächstes Jahr hinzukommt, erfordert keine Änderung an den Bildschirmen.

```
public sealed class ShellCapabilities
{
    public bool HasCamera { get; set; }
    public bool HasLocation { get; set; }
    public bool CanNotify { get; set; }

    public static ShellCapabilities Current =>
        Application.Session["Shell"] as ShellCapabilities
        ?? new ShellCapabilities(); // einfacher Browser: alles aus
}

// Auf der Detailseite des Arbeitsauftrags:
btnTakePhoto.Visible = ShellCapabilities.Current.HasCamera;
uploadPhoto.Visible = !ShellCapabilities.Current.HasCamera;
```

Beachten Sie den Fallback: kein deaktivierter Button, sondern ein **Upload**-Control, das trotzdem ein Foto zum Arbeitsauftrag bringt. Jede Fähigkeit braucht diese Antwort, bevor die Funktion entworfen wird – deshalb ist der Check ein Ergebnis von Modul 1, die Gerätefunktionen dagegen nicht.

Das Entscheidungsprotokoll zur Bereitstellung schreiben

Das letzte Ergebnis ist ein kurzes Dokument – und dasjenige, das Ihr Team tatsächlich wieder lesen wird. Es benennt die Zielplattformen, die Reihenfolge, in der Sie sie umsetzen, und die Abschlusskriterien, die festlegen, wann ein Ziel erreicht ist. Das Protokoll von FieldOps könnte so lauten: zuerst der Browser, weil die Disponenten ihn brauchen und er die App beweist; dann die Android-Shell, weil die Techniker Android-Geräte bei sich tragen; dann die PWA für die Lager-Kiosksysteme; iOS nur, falls die Tablets der Vorgesetzten freigegeben werden.

Abschlusskriterien verhindern, dass ein Ziel ausufert. „Die Android-Shell ist fertig, wenn ein Techniker auf einem gestellten Gerät einen Arbeitsauftrag vollständig abschließt, die Session das Sperren des Bildschirms übersteht und ein signierter Build über den Unternehmenskanal installiert wird“ ist überprüfbar; „mobile Unterstützung“ ist es nicht. Schreiben Sie das Protokoll, bevor es irgendein Shell-Projekt gibt – denn sobald der erste Installer gebaut ist, wird jede Diskussion über den Umfang zu einer Diskussion über bereits investierten Aufwand.



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:

- ✓ erklären, wie eine Wisej.NET-App unverändert in einem Browser, einer PWA, einer Desktop-WebView-Shell und einer mobilen Shell läuft.
- ✓ die vier Bereitstellungsmodelle hinsichtlich Offline-Verhalten, Gerätezugriff, Verteilung, Updates und Kosten vergleichen.
- ✓ die Architektur von FieldOps so entwerfen, dass ein Server und eine Codebasis jede Shell bedienen.
- ✓ ein Entscheidungsprotokoll zur Bereitstellung schreiben, das die Zielplattformen, die Reihenfolge ihrer Umsetzung und die Abschlusskriterien benennt.

Lab- und Prüfungsleitfaden

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



Praxis-Lab

Erstellen Sie die FieldOps-Projektmappe mit einer Liste der Arbeitsaufträge, einer Detailseite für einen Arbeitsauftrag und einem Ablauf zur Statusaktualisierung, die in einem einfachen Browser funktionieren. Schreiben Sie das Entscheidungsprotokoll zur Bereitstellung: Vergleichen Sie Browser, PWA, Desktop- und mobile Shell anhand der Anforderungen der Techniker, wählen Sie die Zielplattformen und ihre Reihenfolge, und definieren Sie die Capability-Checks, mit denen die App Gerätefunktionen aktiviert.

Ausgangspunkt

Sie beginnen mit einer neuen, leeren Wisej.NET-Webanwendung namens FieldOps, die nichts enthält außer dem generierten Startfenster und den Standard-Konfigurationsdateien. Alles in diesem Lab erstellen Sie selbst – das Modell und den Service für Arbeitsaufträge, die drei Bildschirme, die Capability-Abstraktion und das schriftliche Entscheidungsprotokoll –, und jede Shell, die in den Modulen 2 bis 6 hinzukommt, umschließt den Server, den Sie hier fertigstellen, ohne ihn zu verändern.

Vorgeschlagenes Vorgehen

- ✓ Modellieren Sie zuerst die Domäne: einen WorkOrder mit Nummer, Kunde, Adresse des Einsatzorts, Priorität, Status und zugewiesenem Techniker, hinter einem WorkOrderService, der die offenen Aufträge des Technikers liefert und eine Statusänderung anwendet.
- ✓ Bauen Sie die Liste der Arbeitsaufträge als an den Service gebundenes Grid mit einem Filter für den angemeldeten Techniker, und machen Sie sie bei schmaler Fensterbreite bedienbar, damit derselbe Bildschirm später auch einen Viewport in Smartphone-Größe übersteht.
- ✓ Bauen Sie die Detailseite für einen Arbeitsauftrag und einen Ablauf zur Statusaktualisierung – Accepted, On site, Completed –, der den Übergang im Service validiert, ihn einmal schreibt und die Liste aktualisiert.
- ✓ Fügen Sie ein ShellCapabilities-Objekt hinzu, das beim Start der Session ermittelt und in Application.Session abgelegt wird, mit Flags für Kamera, Standort und Benachrichtigungen, die in einem einfachen Browser alle false sind, und lassen Sie die Detailseite es auslesen, um zu entscheiden, was sie anzeigt.
- ✓ Schreiben Sie die Vergleichstabelle und das Entscheidungsprotokoll zur Bereitstellung, und zeichnen Sie dann das Architekturdiagramm mit einem Server, einer Codebasis und den Shells, die auf ihn zeigen werden.

Geforderte Ergebnisse

- ✓ FieldOps-Projektmappe mit Liste, Detailseite und Statusaktualisierung
- ✓ Vergleichstabelle der Bereitstellungsmodelle
- ✓ Entscheidungsprotokoll zur Bereitstellung mit Reihenfolge der Zielplattformen
- ✓ Entwurf des Capability-Checks

- ✓ Architekturdiagramm mit einem Server und mehreren Shells

Wiederholungsfragen

- ✓ Ein Techniker verliert mitten in der Aktualisierung eines Arbeitsauftrags den Empfang. Beschreiben Sie genau, welche Teile von FieldOps noch aus einer gecachten Shell angezeigt werden können, welche nicht, und was Ihr Bildschirm in dem Moment tun sollte, in dem die Verbindung abbricht.
- ✓ Ihr Entscheidungsprotokoll stellt die Android-Shell vor die PWA. Was müsste für die Geräte und das Netz der Techniker gelten, damit diese Reihenfolge falsch wäre?
- ✓ Die Detailseite braucht ein Foto. Erklären Sie, warum ein Capability-Check mit einem Upload-Fallback ein besseres Design ist als ein Build-Schalter, der eine reine Mobilversion des Bildschirms erzeugt.



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.