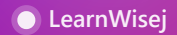


Einrichtung einer Progressive Web App

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Web-App-Manifest, Icons und Splash-Screens, der Service Worker, Installierbarkeit, Offline-Shell und der Update-Ablauf einer PWA. 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

Web-App-Manifest, Icons und Splash-Screens, der Service Worker, Installierbarkeit, Offline-Shell und der Update-Ablauf einer PWA.

1. Das Problem im Produktivbetrieb

Die Techniker von FieldOps öffnen keinen Browser und tippen keine URL ein. Sie tippen auf ein Icon auf einem Smartphone, das den ganzen Vormittag im Transporter lag, in einem Technikraum, in dem das Signal mitten im Einsatz auf null fällt. Eine Progressive Web App gibt ihnen dieses Icon, ein Fenster ohne Browser-Bedienelemente und eine Seite, die noch etwas Nützliches sagt, wenn das Netz verschwindet – aber FieldOps ist eine Wisej.NET-App, ihre Benutzeroberfläche lebt also auf dem Server, und wer so tut, als wäre es anders, macht aus der PWA ein Support-Ticket.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Web-App-Manifest	Eine kleine JSON-Datei, auf die die Startseite verlinkt und die dem Browser Namen, Icons, Farben, Anzeigemodus und Start-URL der App mitteilt – ohne sie hat der Browser nichts zu installieren.
Anzeigemodus	Das <code>display</code> -Feld des Manifests. <code>standalone</code> startet FieldOps in einem eigenen Fenster ohne Adressleiste – erst dadurch fühlt sich eine installierte App nicht mehr wie ein Lesezeichen an.
Maskierbare Icons	Icons, die mit <code>purpose: "maskable"</code> und ausreichendem Innenabstand deklariert sind, damit Android sie in seine eigene Form zuschneiden kann, ohne die Grafik abzuschneiden.
Service Worker	Ein Skript, das der Browser neben der Seite ausführt, von der eigenen Origin der App über HTTPS ausgeliefert, und das Anfragen der Seite aus einem Cache beantworten kann, wenn die Verbindung langsam oder weg ist.
Anwendungs-Shell	Die statischen Dateien, die eine Session braucht, bevor der Server überhaupt etwas sagen kann: die Startseite, die Client-Skripte, das Theme und die Icons. Nur sie darf der Worker cachen.
Cache-First vs. Network-Only	Die Routing-Regel im Fetch-Handler des Workers: Statische Shell-Dateien kommen zuerst aus dem Cache, während alles, was die Wisej.NET-Session mit dem Server austauscht, direkt ins Netzwerk gehen muss – nie in eine gecachte Kopie.
Update-Lebenszyklus	Ein neuer Worker installiert sich, während der alte die Seite noch kontrolliert, und wartet dann. <code>skipWaiting</code> , <code>clients.claim</code> und ein kontrolliertes Neuladen machen aus diesem wartenden Worker die Version, die der Techniker tatsächlich ausführt.

3. Der rote Faden der Lektion

Das Manifest ist der Installationsvertrag

Installierbarkeit ist kein Schalter in Wisej.NET, sondern eine Reihe von Bedingungen, die der Browser prüft. FieldOps muss über HTTPS ausgeliefert werden, die Startseite muss auf ein gültiges Web-App-Manifest verlinken, und für den Scope der Seite muss ein Service Worker registriert sein. Das Manifest liegt bei den übrigen statischen Dateien der Anwendung – im selben Ordner wie Startseite, Theme und Icons – und wird aus `Default.html` referenziert, dem Startdokument von Wisej.NET und der einzigen Stelle im Projekt, an der Sie das HTML des Browsers selbst in der Hand haben.

Die relevanten Felder sind überschaubar. `name` und `short_name` bestimmen, was unter dem Icon und im App-Umschalter erscheint. `icons` braucht ein PNG mit 192 und eines mit 512 Pixeln sowie maskierbare Varianten mit Innenabstand, und iOS verlangt zusätzlich einen `apple-touch-icon`-Link auf der Seite selbst. `theme_color` färbt die Systemleisten um das eigenständige Fenster, während `background_color` den Splash-Screen färbt, den der Browser beim Start erzeugt – beide sollten also zur App-Leiste von FieldOps passen. `display` ist `standalone`, und `start_url` zeigt zusammen mit `scope` auf das

Anwendungstammverzeichnis, damit ein Start auf der Liste der Arbeitsaufträge landet und jede URL unterhalb der App im installierten Fenster bleibt.

Ist der Browser zufrieden, löst er auf der Seite `beforeinstallprompt` aus, statt ein eigenes Banner anzuzeigen. Der Handler in `Default.html` unterdrückt dieses Standardverhalten, bewahrt das Event auf und stellt eine kleine Funktion bereit, die dessen `prompt()`-Methode aufruft. Daraus folgen zwei Regeln. Der Dialog öffnet sich nur nach einem echten Klick im Browser, der auslösende Button gehört also in die Seite, nicht auf den Server. Und der Server entscheidet, *wann* sich das Angebot lohnt – das Lab verlangt es nach der ersten erfolgreichen Session – und blendet das Bedienelement über `Application.Eval` oder das entsprechende JavaScript-Interop ein:

```
private void OnFirstSessionCompleted()
{
    // Default.html hat das beforeinstallprompt-Event aufbewahrt; der Server
    // entscheidet nur, wann das Angebot sichtbar wird. prompt() läuft weiterhin
    // im Klick-Handler der Seite, innerhalb einer echten Benutzergeste.
    Application.Session["InstallOffered"] = true;
    Application.Eval("window.fieldOps.showInstallButton()");
}
```

Ein Worker, der die Shell cacht, niemals die Session

Registrieren Sie den Worker aus `Default.html` mit `navigator.serviceWorker.register` und verweisen Sie auf eine `sw.js` im Anwendungstammverzeichnis, damit ihr Scope die gesamte App abdeckt. Bei `install` öffnet der Worker einen benannten Cache und legt die Shell hinein: die Startseite, die Client-Skripte und Theme-Dateien, die die Seite lädt, bevor die Session existiert, das Manifest, die Icons und die Offline-Seite. Bei `activate` löscht er jeden Cache, dessen Name nicht dem aktuellen entspricht.

Im `fetch`-Handler entscheidet sich, ob eine PWA über einer servergesteuerten Oberfläche gelingt oder scheitert. FieldOps ist eine Wisej.NET-Anwendung: Die Controls sind C#-Objekte auf dem Server, und was während einer Session über die Leitung geht, ist ein Strom von Anfragen und Antworten, die diesen Serverzustand widerspiegeln, plus ein WebSocket für gepushte Aktualisierungen. Nichts davon ist in irgendeinem sinnvollen Sinn cachebar – eine wiedergegebene Antwort beschreibt eine Session, die es nicht mehr gibt. Der Handler unterscheidet deshalb nach der Anfrage: Ein `GET` auf eine bekannte Shell-Datei wird Cache-First beantwortet, Anfragen, die kein `GET` sind, werden nie angefasst, und alles andere geht unverändert ins Netzwerk. Scheitert eine Navigation, weil der Server nicht erreichbar ist, antwortet der Handler mit der gecachten Offline-Seite statt mit der Fehlerseite des Browsers.

Ehrliches Offline-Verhalten für eine servergesteuerte Oberfläche

Sagen Sie klar, was offline hier bedeutet: Die Shell lädt, die App läuft nicht. Ohne Server gibt es keine Session, keine Daten zu Arbeitsaufträgen und keine Möglichkeit, einen Status zu ändern, und kein noch so ausgefeiltes Caching verlagert eine serverseitige Oberfläche in den Browser. Die Designarbeit besteht darin, diesen Zustand nützlich statt kaputt zu machen. FieldOps schreibt nach jedem erfolgreichen Laden eine kleine Zusammenfassung – Anzahl offener Arbeitsaufträge, nächster Einsatzort und Zeitpunkt der Synchronisierung – in den Browser-Speicher, und die Offline-Seite zeigt sie hinter einem deutlichen Banner „Nicht verbunden“ und einer Schaltfläche zum erneuten Versuch an.

- Nichts, was auf der Offline-Seite erfasst wird, wird als gespeichert dargestellt; zwischengespeicherte Änderungen sind eine spätere, bewusste Funktion, kein Nebeneffekt eines Caches.
- Jede Offline-Ansicht trägt den Zeitstempel der letzten Synchronisierung, damit alte Daten nie für aktuelle gehalten werden.
- Kehrt die Verbindung zurück, lädt die App in eine frische Session neu, statt eine veraltete fortzusetzen.

Die Wiederverbindung verdient dieselbe Ehrlichkeit. Eine Wisej.NET-Session kann ablaufen oder verloren gehen, während das Gerät im Transporter im Ruhezustand liegt; die verlässliche Wiederherstellung besteht darin, die Startseite neu zu laden und den Server eine neue Session aufbauen zu lassen – nicht darin, eine tote Oberfläche auf dem Bildschirm stehen zu lassen, die lebendig aussieht.

Eine neue Version an einem Cache vorbeibringen, der seine Arbeit tut

Der Cache, der FieldOps schnell macht, ist auch das, was einen Techniker auf dem Build vom letzten Monat festhält. Versionieren Sie den Cache-Namen – `fieldops-shell-v7` – und ändern Sie ihn im selben Commit wie jede Shell-Datei. Der neue Worker installiert sich dann in einen neuen Cache, und sein `activate`-Handler löscht die alten. Rufen Sie `skipWaiting` auf, damit der neue Worker nicht hinter dem alten wartet, und `clients.claim`, damit er die geöffnete Seite sofort kontrolliert.

Die Seite muss darauf trotzdem noch reagieren. Das Lauschen auf `controllerchange` oder auf `updatefound` und `statechange` an der Registrierung teilt ihr mit, dass eine neue Version übernommen hat; FieldOps zeigt eine kleine Leiste „Eine neue Version ist bereit – Neu laden“ und lädt neu, sobald der Techniker darauf tippt, sodass niemand ein halb ausgefülltes Formular durch eine überraschende Aktualisierung verliert. Zwei Deployment-Details vervollständigen den Ablauf: `sw.js` und das Manifest werden mit No-Cache-Headern ausgeliefert, damit der Browser ihre Änderung überhaupt bemerken kann, und die App vergleicht beim Start der Session ihren Build-Stempel mit dem des Servers – so fällt eine Shell auf, die ein Deployment überlebt hat, statt mit einem Server zu sprechen, zu dem sie nicht mehr passt.



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:

- ✓ ein Web-App-Manifest mit Icons, Farben und Anzeigemodus hinzufügen, damit sich die App aus dem Browser installieren lässt.
- ✓ einen Service Worker registrieren, der die Anwendungs-Shell cacht und ausliefert, während das Netzwerk langsam oder nicht vorhanden ist.
- ✓ erklären, was in einer serverseitigen Oberfläche offline funktionieren kann und was nicht, und das Offline-Erlebnis ehrlich gestalten.

- ✓ PWA-Updates so handhaben, dass Benutzer die neue Version ohne veralteten Cache erhalten.

Lab- und Prüfungsleitfaden

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



Praxis-Lab

Machen Sie FieldOps zu einer PWA: Fügen Sie das Manifest mit Icons, Theme-Farbe und Standalone-Anzeige hinzu, registrieren Sie einen Service Worker, der die Shell und statische Assets cacht, zeigen Sie nach der ersten erfolgreichen Session einen Installationsdialog an, blenden Sie eine Offline-Seite mit der Zusammenfassung der zuletzt synchronisierten Arbeitsaufträge ein, wenn der Server nicht erreichbar ist, und implementieren Sie den Update-Ablauf, der neu lädt, sobald eine neue Version verfügbar ist.

Ausgangspunkt

Sie öffnen die FieldOps-Projektmappe, die Sie in Modul 1 erstellt haben: eine Wisej.NET-Webanwendung mit einer Liste der Arbeitsaufträge, einer Detailseite für einen Arbeitsauftrag und einem Ablauf zur Statusaktualisierung, die bereits in einem einfachen Browser-Tab korrekt laufen. Installierbar ist an der App noch nichts – es gibt kein Manifest, keinen Service Worker und keine Offline-Seite –, und die statischen Dateien, die Sie hinzufügen, liegen neben der vorhandenen `Default.html`, den Theme- und den Icon-Dateien.

Vorgeschlagenes Vorgehen

- ✓ Liefern Sie FieldOps lokal über HTTPS aus und schreiben Sie dann `manifest.webmanifest` mit dem App-Namen, Icons mit 192 und 512 Pixeln einschließlich maskierbarer Varianten, der Theme- und Hintergrundfarbe von FieldOps, `display: standalone` und einer Start-URL im Anwendungsstammverzeichnis, und verlinken Sie es zusammen mit einem apple-touch-icon aus `Default.html`.
- ✓ Fügen Sie `sw.js` im Anwendungsstammverzeichnis hinzu, mit einem versionierten Cache-Namen, einem Install-Handler, der die Shell vorab cacht – Startseite, Client-Skripte, Theme, Icons und die Offline-Seite –, und einem Activate-Handler, der jeden Cache einer älteren Version löscht.
- ✓ Schreiben Sie den Fetch-Handler so, dass Shell-Anfragen Cache-First beantwortet werden, während jede Anfrage der Wisej.NET-Session an den Server unverändert ins Netzwerk geht, und eine fehlgeschlagene Navigation auf die gecachte Offline-Seite zurückfällt.
- ✓ Fangen Sie `beforeinstallprompt` in der Seite ab, bewahren Sie das Event auf, und lassen Sie den Server den Installieren-Button der Seite per JavaScript-Interop erst einblenden, nachdem der Techniker eine erfolgreiche Session mit geladener Liste der Arbeitsaufträge abgeschlossen hat.
- ✓ Schreiben Sie nach jedem erfolgreichen Laden eine kompakte Zusammenfassung der Arbeitsaufträge in den Browser-Speicher, damit die Offline-Seite sie mit dem Zeitstempel der letzten Synchronisierung anzeigen kann; erhöhen Sie dann die Cache-Version, deployen Sie, und prüfen Sie, dass die Update-Leiste erscheint und in den neuen Build neu lädt.

Geforderte Ergebnisse

- ✓ Web-App-Manifest mit Icons und Farben

- ✓ Service Worker, der die Shell cacht
- ✓ Installationsdialog nach der ersten Session
- ✓ Offline-Seite mit der zuletzt synchronisierten Zusammenfassung
- ✓ Update-Ablauf und Versionsprüfung

Wiederholungsfragen

- ✓ Ein Reviewer fragt, warum der Service Worker nicht einfach jede Antwort cacht, damit FieldOps auch unter der Erde weiter funktioniert. Erklären Sie, was eine wiedergegebene Antwort einer Wisej.NET-Session tatsächlich mit dem Bildschirm des Technikers anrichten würde und wo die ehrliche Grenze verläuft.
- ✓ Ihr Installationsangebot funktioniert an Ihrem Schreibtisch, erscheint aber nie auf dem Smartphone des Technikers. Nennen Sie drei unterschiedliche Bedingungen, die dafür verantwortlich sein könnten, und sagen Sie, wie Sie sie am Gerät voneinander unterscheiden würden.
- ✓ Ein Techniker hat die App installiert und geöffnet, während Sie einen neuen Build deployen. Beschreiben Sie jeden Schritt zwischen Ihrem Deployment und dem Moment, in dem auf diesem Smartphone die neue Shell läuft, und benennen Sie den Schritt, der stillschweigend scheitern würde, wenn sich der Cache-Name nie ändert.



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.