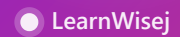


Signierung, Verteilung, Updates und das Abschlussprojekt FieldOps

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Codesignierung, Store-Einreichung, Unternehmensverteilung, Versionierung, Update-Strategien über alle Shells, Telemetrie und die Auslieferung des Abschlussprojekts. 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

Codesignierung, Store-Einreichung, Unternehmensverteilung, Versionierung, Update-Strategien über alle Shells, Telemetrie und die Auslieferung des Abschlussprojekts.

1. Das Problem im Produktivbetrieb

FieldOps beendet diesen Kurs als ein Server mit vier Zugangswegen: ein Browser-Tab, eine installierte PWA, eine Windows-Desktop-Shell und eine mobile Shell unter Android und iOS. Die drei paketierte Varianten sind Dateien, deren Installation ein Betriebssystem verweigert, sofern sie nicht von einer vertrauenswürdigen Identität signiert wurden, und jede nimmt einen anderen Kanal mit eigener Prüfwarteschlange und eigenem Rollout-Tempo. Den Server können Sie noch vor der Mittagspause neu deployen; der iOS-Build, den ein Techniker in der Hand hält, liegt womöglich drei Wochen zurück – und muss trotzdem funktionieren.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Authenticode-Signatur	Die Signatur, die ein Codesignaturzertifikat auf einen Windows-Installer oder ein MSIX anbringt; sie belegt, dass die Datei nach dem Signieren nicht verändert wurde, und nennt einen geprüften Herausgeber.
Zeitstempel der Gegensignatur	Ein vertrauenswürdiger Zeitstempel, der beim Signieren festgehalten wird, damit der Installer auch nach Ablauf des Signaturzertifikats gültig bleibt.
Android-Keystore und Upload-Schlüssel	Der Schlüssel, der einen Android-Build signiert; mit Play App Signing signieren Sie mit einem Upload-Schlüssel, und Google signiert mit dem App-Signaturschlüssel, den es für dieses Paket verwahrt, neu.
Apple-Distributionszertifikat und Provisioning-Profil	Das Zertifikat identifiziert Ihr Team; das Profil verknüpft eine App-ID, dieses Zertifikat, die Entitlements und – außerhalb des App Store – die zugelassenen Geräte.
Unternehmensverteilung	Einen privaten Build an die Geräte einer einzelnen Organisation ausliefern – Managed Google Play, benutzerdefinierte Apps über Apple Business Manager, ein per Intune verteiltes MSIX –, statt ihn öffentlich zu veröffentlichen.
Minimale Shell-Version	Der älteste Shell-Build pro Plattform, den der aktuelle Server noch unterstützt; alles darunter wird beim Verbinden gestoppt, bevor es an irgendwelche Daten gelangt.
Nutzungs-Event für Gerätefunktionen	Ein Datensatz, der bei jeder Ausführung einer Shell-Fähigkeit geschrieben wird – Kamera, Unterschrift, Standort, Push, Leeren der Offline-Warteschlange – mit dem Ergebnis sowie Plattform und Version der Shell.

3. Der rote Faden der Lektion

Was eine Signatur beweist – und die drei, die FieldOps braucht

Eine Codesignatur trifft genau zwei Aussagen: Die Bytes sind die Bytes, die signiert wurden, und eine namentlich bekannte Identität hat sie signiert. Darüber, ob sich der Code gut verhält, sagt sie nichts, und die gesamte Verteilungskette ruht auf diesen beiden Aussagen und auf nichts anderem. Signieren Sie das Windows-Paket mit einem auf Ihr Unternehmen ausgestellten Codesignaturzertifikat und fügen Sie immer einen vertrauenswürdigen Zeitstempel hinzu; ohne ihn ist der Installer ab dem Tag, an dem das Zertifikat abläuft, nicht mehr gültig. Liefern Sie die Desktop-Shell als MSIX aus, muss der Antragsteller des Zertifikats mit dem Herausgebernamen im Paketmanifest übereinstimmen, sonst lehnt Windows das Paket sofort ab.

Android folgt einem anderen Modell: Der Build wird mit einem Schlüssel aus einem Keystore signiert, und die Identität, die für das Gerät zählt, ist dieser Schlüssel, nicht eine Zertifizierungsstelle. Melden Sie FieldOps bei Play App Signing an: Dort signieren Sie jedes Release mit einem Upload-Schlüssel, Google prüft ihn, entfernt ihn und signiert mit dem App-Signaturschlüssel neu, den es verwahrt. Damit entfällt der schlimmste Fehler bei der mobilen Verteilung – ein verlorener Schlüssel, der die App dauerhaft von

Updates abschneidet –, denn verlieren können Sie nur den Upload-Schlüssel, und der lässt sich zurücksetzen.

Apple benötigt zwei Artefakte. Das Distributionszertifikat identifiziert Ihr Team; das Provisioning-Profil legt fest, dass diese App-ID, signiert mit jenem Zertifikat, diese Entitlements nutzen und – außerhalb des App Store – auf diesen Geräten laufen darf. Push-Benachrichtigungen, mit denen FieldOps einen neuen Arbeitsauftrag ankündigt, sind ein Entitlement; ein Profil, das älter ist als diese Fähigkeit, erzeugt also einen Build, der sich installieren lässt und dann stillschweigend nichts empfängt. Das Abschlussprojekt verlangt alle drei signierten Artefakte sowie pro Plattform eine Notiz, wer den Schlüssel verwahrt, wo er liegt und wann er abläuft.

Store, Unternehmen oder direkt: eine Entscheidung pro Plattform

Es gibt drei Kanäle, und jede Plattform wählt einen. Ein öffentlicher Store-Eintrag passt zu einer App, die jeder installieren darf: Google Play und der App Store bieten Hosting, Delta-Updates, gestaffelte Rollouts und eine Prüfwarteschlange, die sich in Stunden bis Tagen bemisst. Die Unternehmensverteilung passt besser zu FieldOps, denn die Techniker sind Mitarbeiter mit verwalteten Geräten: Managed Google Play veröffentlicht privat für eine Organisation und weist die App über deren EMM zu, Apple Business Manager verteilt eine benutzerdefinierte App an namentlich genannte Organisationen, die sie per MDM ausrollen, und unter Windows wird das signierte MSIX über Intune verteilt. Der direkte Download – Ihr eigener Server hostet den signierten Installer und einen Versions-Feed – bleibt für die Desktop-Shell nützlich und ist unter iOS praktisch nicht nutzbar.

Browser und PWA fallen aus dem Rahmen, und genau das ist der Kern des ganzen Kurses: kein Kanal, kein Zertifikat, keine Prüfung. Sie deployen den Server, und das nächste Laden der Seite ist die neue Version. Jede Stunde, die Sie investieren, damit FieldOps in einem einfachen Browser voll nutzbar bleibt, ist eine Versicherung gegen eine Store-Warteschlange.

Eine Einreichung vorzubereiten ist echte Arbeit, deshalb verlangt das Lab einen Entwurf für den Store-Eintrag, auch wenn Sie privat verteilen: Name und Beschreibung für einen Techniker statt für einen Kunden im Store geschrieben, Screenshots in jeder geforderten Gerätegröße, eine Datenschutz- und Datensicherheitserklärung, die die von FieldOps erfassten Kamera-, Foto- und Standortdaten ehrlich auflistet, und ein Support-Kontakt. Die Unternehmenseinrichtung ist das Spiegelbild dazu: Zuweisungsgruppen, verwaltete Konfiguration wie die Serveradresse und wer im IT-Team des Kunden eine neue Version freigibt.

Der Server sprintet, die Shells kriechen

Weil die Shells mit dem Server nicht Schritt halten können, muss die Kompatibilitätsregel explizit sein. Geben Sie dem Server eine eigene Version und eine minimale Shell-Version pro Plattform – den ältesten Build, mit dem er noch kommunizieren will. Die Shell sendet beim Verbinden ihre Plattform und Version, als Parameter an der geladenen Adresse oder über den Bridge-Handshake, und der Server hält beides in `Application.Session` fest, damit jede spätere Log-Zeile und jedes Telemetrie-Event sie benennen kann. Unterhalb des Minimums erhält der Techniker einen blockierenden Update-Bildschirm, der auf das richtige Ziel für diesen Kanal verweist, und es werden überhaupt keine Daten zu Arbeitsaufträgen geladen. Auf oder über dem Minimum, aber hinter dem neuesten Build, zeigen Sie einen schließbaren Hinweis und lassen ihn weiterarbeiten.

```

public ShellGate AdmitShell(string platform, Version shellVersion)
{
    var minimum = _config.MinimumShellVersion(platform);    // z. B. 2.4.0
    if (shellVersion < minimum)
        return ShellGate.MustUpdate(minimum, _config.UpdateTarget(platform));

    Application.Session["ShellPlatform"] = platform;
    Application.Session["ShellVersion"] = shellVersion;

    return shellVersion < _config.LatestShellVersion(platform)
        ? ShellGate.UpdateAvailable(_config.ServerVersion)
        : ShellGate.Current(_config.ServerVersion);
}

```

Das Anheben des Minimums ist eine bewusste Handlung, kein Nebeneffekt eines Releases. Heben Sie es nur an, wenn der Server beginnt, sich auf etwas zu verlassen, das ältere Shells nicht können: eine neue Bridge-Methode, eine geänderte Struktur der Nutzdaten, eine Berechtigung, die der alte Build nie angefordert hat. Alles andere, einschließlich neuer Bildschirme, muss auf den bereits im Feld befindlichen Shells kontrolliert zurückfallen. Diese Disziplin ist mehr wert als die Prüfung selbst, denn sie erzwingt vor jedem Deployment die Frage: Was macht der zwei Monate alte Android-Build damit?

Telemetrie, die die Shell benennt, und die Release-Checkliste

Mit vier Auslieferungsformen im Produktivbetrieb ist ein Fehlerbericht, der „es ist abgestürzt“ lautet, nahezu nutzlos. Binden Sie in jede Shell ein SDK für Absturzberichte ein, damit native Abstürze und unbehandelte Exceptions im Shell-Prozess mit Stacktrace, Gerätemodell und Build-Nummer ankommen, obwohl sie außerhalb Ihres C#-UI-Codes aufgetreten sind. Protokollieren Sie auf dem Server unbehandelte Exceptions zusammen mit Plattform und Version der Shell der Session, damit sich ein Vorfall von beiden Seiten lesen lässt. Ergänzen Sie dann ein Nutzungs-Event pro Gerätefunktion: Foto aufgenommen, Unterschrift gespeichert, Standort ermittelt, Push empfangen, zwischengespeicherter Upload gesendet – jeweils mit Ergebnis und der Shell, aus der es stammt. Zwei Wochen solcher Daten beantworten, was Diskussionen nicht klären können, etwa ob der Kamera-Fallback greift, weil eine Berechtigung verweigert wurde, oder weil die Shell zu alt ist, um die Fähigkeit überhaupt zu haben.

Telemetrie sind Daten über die Arbeit von Menschen, also legen Sie sie offen. Keine Fotoinhalte, keine Unterschriftsbilder, keine Kundennamen, keine rohen Koordinaten, wo eine Kennung des Arbeitsauftrags genügt – und was Sie tatsächlich erfassen, gehört in die Datenschutzerklärung, die Sie eingereicht haben. Das Abschlussprojekt endet dann mit einer Release-Checkliste, denn vier Auslieferungsformen haben vier verschiedene Fehlerbilder, und die Checkliste macht ein Release wiederholbar – auch für jemanden, der nicht dabei war, als es entworfen wurde.

- Browser: Server deployt, Versions-Endpunkt aktualisiert, Smoke-Test mit einer frischen Session.
- PWA: Cache-Version des Service Workers erhöht, Update-Ablauf ausgehend vom vorherigen Build geprüft.
- Desktop: Installer signiert und mit Zeitstempel versehen, Signatur auf einem sauberen Rechner geprüft, Intune-Paket ersetzt.

- Mobil: richtige Schlüssel und Profile, minimale Shell-Version überprüft, gestaffelter Rollout eingestellt, Absturz-Dashboard am ersten Tag beobachtet.



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:

- ✓ Desktop- und Mobilpakete signieren und erklären, was jedes Zertifikat beweist.
- ✓ pro Plattform zwischen Store-, Unternehmens- und Direktverteilung wählen und jede Einreichung vorbereiten.
- ✓ eine Versionierungs- und Update-Strategie entwerfen, bei der sich der Server schneller bewegen kann als die Shells.
- ✓ Absturz- und Nutzungstelemetrie über alle Shells hinweg ergänzen und das Abschlussprojekt FieldOps ausliefern.

Lab- und Prüfungsleitfaden

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



Praxis-Lab

Schließen Sie das Abschlussprojekt FieldOps ab: Signieren Sie den Windows-Installer sowie die Android- und iOS-Pakete, bereiten Sie eine Unternehmensverteilung für die mobilen Shells und einen Entwurf für den Store-Eintrag vor, definieren Sie eine Kompatibilitätsregel zwischen Server- und Shell-Versionen mit einer Prüfung der Mindestversion beim Start, ergänzen Sie Absturzberichte und ein Nutzungs-Event für jede Gerätefunktion, und liefern Sie eine Release-Checkliste, die alle vier Bereitstellungsmodelle abdeckt.

Ausgangspunkt

Sie öffnen die FieldOps-Projektmappe im Stand nach Modul 5: der Wisej.NET-Server mit Liste, Detailseite und Statusablauf der Arbeitsaufträge, das PWA-Manifest und der Service Worker, die WebView2-Desktop-Shell, die .NET-MAUI-Shell für Android und iOS und die Bridge, die Kamera, Unterschrift, Standort und Push bereits hinter Capability-Checks bereitstellt. Alles läuft derzeit aus unsignierten Debug-Builds, die auf einen Entwicklungsserver zeigen.

Vorgeschlagenes Vorgehen

- ✓ Erstellen Sie zuerst die drei Signaturidentitäten – ein Codesignaturzertifikat für Windows, einen Keystore mit einem bei Play App Signing registrierten Upload-Schlüssel und ein Apple-Distributionszertifikat mit einem Provisioning-Profil, das das Push-Entitlement enthält – und halten Sie für jede fest, wer den Schlüssel verwahrt und wann er abläuft.
- ✓ Bauen und signieren Sie ein Release-Paket pro Plattform und prüfen Sie jede Signatur so, wie es das Zielgerät tun wird: Kontrollieren Sie die Authenticode-Signatur und ihren Zeitstempel auf einem sauberen Windows-Rechner, und stellen Sie sicher, dass sich die mobilen Builds über ihren vorgesehenen Kanal installieren lassen statt über ein Debug-Kabel.
- ✓ Ergänzen Sie den Versions-Handshake, bevor irgendetwas anderes lädt: Stellen Sie die Serverversion und die minimale Shell-Version pro Plattform aus der Konfiguration bereit, lassen Sie jede Shell beim Verbinden ihre Plattform und Version melden, und speichern Sie beides in der Session.
- ✓ Bauen Sie den Update-Bildschirm, den die Prüfung einer veralteten Shell anzeigt, mit dem Ziel, das zu ihrem Kanal passt, sowie einen schließbaren Hinweis für eine Shell, die lediglich hinter dem neuesten Build liegt.
- ✓ Binden Sie Absturzberichte in jede Shell und ein Nutzungs-Event in jeden Bridge-Aufruf ein, und schreiben Sie dann die Release-Checkliste, indem Sie ein Release vollständig über Browser, PWA, Desktop und Mobilgeräte hinweg durchspielen.

Geforderte Ergebnisse

- ✓ Signierte Pakete für Windows, Android und iOS
- ✓ Einrichtung der Unternehmensverteilung und Entwurf des Store-Eintrags

- ✓ Kompatibilitätsprüfung zwischen Server- und Shell-Version
- ✓ Absturz- und Nutzungstelemetrie
- ✓ Release-Checkliste für alle Bereitstellungsmodelle

Wiederholungsfragen

- ✓ Die Android-Shell eines Technikers ist sechs Wochen alt, und der Server wurde seitdem viermal ausgeliefert. Welche dieser vier Serveränderungen hätte die minimale Shell-Version anheben müssen, und woran hätten Sie das zum jeweiligen Zeitpunkt erkannt?
- ✓ Der Windows-Installer wurde signiert, aber ohne Zeitstempel, und das Zertifikat ist letzten Monat abgelaufen. Was passiert mit bestehenden Installationen, was mit neuen Downloads, und was ist der kürzeste Weg zurück zu einem vertrauenswürdigen Installer?
- ✓ Das Nutzungs-Event der Kamera zeigt einen starken Anstieg des Browser-Fallback-Wegs – ausschließlich unter iOS. Nennen Sie drei Erklärungen, die zu den Daten passen, und sagen Sie, welches zusätzliche Feld im Event sie voneinander unterscheiden würde.



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.