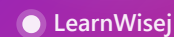


Signature, distribution, mises à jour et projet de synthèse FieldOps

Guide de la leçon · Guide du lab / de l'examen



Guide de la leçon

Signature de code, soumission aux stores, distribution en entreprise, gestion des versions, stratégies de mise à jour selon les shells, télémétrie et livraison du projet de synthèse. Ce que cela signifie pour une équipe qui livre une application Wisej.NET au-delà de l'onglet du navigateur — et comment aborder ce module.



Enjeu en production

Signature de code, soumission aux stores, distribution en entreprise, gestion des versions, stratégies de mise à jour selon les shells, télémétrie et livraison du projet de synthèse.

1. Le problème en production

À la fin de ce cours, FieldOps est un serveur et quatre façons de l'atteindre : un onglet de navigateur, une PWA installée, un shell desktop Windows et un shell mobile sur Android et iOS. Les trois versions packagées sont des fichiers qu'un système d'exploitation refuse d'installer tant qu'une identité en laquelle il a confiance ne les a pas signés, et chacune emprunte un canal différent avec sa propre file de revue et sa propre vitesse de déploiement. Vous pouvez redéployer le serveur avant le déjeuner ; le build iOS qu'a en main un technicien peut avoir trois semaines de retard, et il doit quand même fonctionner.

2. Concepts clés

Concept	Pourquoi c'est important
signature Authenticode	La signature qu'un certificat de signature de code applique à un installateur Windows ou à un MSIX, prouvant que le fichier n'a pas été modifié après la signature et nommant un éditeur vérifié.
horodatage par contre-signature	Un horodatage de confiance enregistré au moment de la signature, pour que l'installateur continue d'être validé après l'expiration du certificat de signature lui-même.
keystore Android et clé d'import	La clé qui signe un build Android ; avec Play App Signing, vous signez avec une clé d'import et Google re-signe avec la clé de signature de l'application qu'il détient pour ce package.
certificat de distribution et profil de provisionnement Apple	Le certificat identifie votre équipe ; le profil relie un App ID, ce certificat, les entitlements et, en dehors de l'App Store, les appareils autorisés.
distribution en entreprise	Livrer un build privé aux appareils d'une organisation — Google Play géré, applications personnalisées Apple Business Manager, un MSIX poussé par Intune — au lieu de le publier publiquement.
version minimale du shell	Le plus ancien build de shell, par plateforme, que le serveur actuel prend encore en charge ; tout ce qui est en dessous est arrêté à la connexion avant d'atteindre la moindre donnée.
événement d'usage d'une fonctionnalité de l'appareil	Un enregistrement écrit chaque fois qu'une capacité du shell s'exécute — caméra, signature, localisation, push, vidage de la file hors ligne — portant le résultat ainsi que la plateforme et la version du shell.

3. Déroulé de la leçon

Ce que prouve une signature, et les trois besoins de FieldOps

Une signature de code affirme exactement deux choses : les octets sont ceux qui ont été signés, et une identité nommée les a signés. Elle ne dit rien sur le bon comportement du code, et toute la chaîne de distribution repose sur ces deux affirmations et rien d'autre. Signez le package Windows avec un certificat de signature de code délivré à votre entreprise et ajoutez toujours un horodatage de confiance ; sans lui, l'installateur cesse d'être validé le jour où le certificat expire. Si vous livrez le shell desktop en MSIX, le sujet du certificat doit correspondre au nom d'éditeur du manifeste du package, sinon Windows le rejette purement et simplement.

Android suit un autre modèle : le build est signé avec une clé conservée dans un keystore, et l'identité qui compte pour l'appareil est cette clé, pas une autorité de certification. Inscrivez FieldOps à Play App Signing : vous signez chaque release avec une clé d'import, Google la vérifie, la retire et re-signe avec la clé de signature de l'application qu'il détient. Cela supprime le pire échec de la distribution mobile, une clé

perdue qui rend l'application définitivement impossible à mettre à jour, car seule la clé d'import est à votre charge et elle peut être réinitialisée.

Apple exige deux éléments. Le certificat de distribution identifie votre équipe ; le profil de provisionnement indique que cet App ID, signé par ce certificat, peut utiliser ces entitlements et — en dehors de l'App Store — s'exécuter sur ces appareils. Les notifications push, que FieldOps utilise pour annoncer un nouvel ordre de travail, sont un entitlement ; un profil antérieur à cette capacité produit donc un build qui s'installe puis ne reçoit silencieusement rien. Le projet de synthèse demande les trois artefacts signés, plus une note par plateforme indiquant qui détient la clé, où elle se trouve et quand elle expire.

Store, entreprise ou direct : une décision par plateforme

Trois canaux existent et chaque plateforme en choisit un. Une fiche publique en store convient à une application que tout le monde peut installer : Google Play et l'App Store vous offrent l'hébergement, les mises à jour différentielles, les déploiements progressifs et une file de revue qui se mesure en heures ou en jours. La distribution en entreprise convient mieux à FieldOps, car les techniciens sont des salariés équipés d'appareils gérés : Google Play géré publie en privé pour une organisation et attribue via son EMM, Apple Business Manager distribue une application personnalisée à des organisations nommées qui la poussent avec leur MDM, et sous Windows le MSIX signé est déployé via Intune. Le téléchargement direct — votre propre serveur hébergeant l'installateur signé et un flux de versions — reste utile pour le shell desktop, et il est pratiquement inutilisable sur iOS.

Le navigateur et la PWA font exception, et c'est tout l'intérêt du cours : pas de canal, pas de certificat, pas de revue. Vous déployez le serveur et le chargement de page suivant correspond à la nouvelle version. Chaque heure passée à garder FieldOps pleinement utilisable dans un simple navigateur est une assurance contre la file d'attente d'un store.

Préparer une soumission est un vrai travail ; le lab demande donc un brouillon de fiche store même lorsque vous distribuez en privé : le nom et la description rédigés pour un technicien plutôt que pour un acheteur, des captures d'écran à chaque taille d'appareil exigée, une déclaration de confidentialité et de sécurité des données qui liste honnêtement les données de caméra, de photo et de localisation que FieldOps collecte, et un contact d'assistance. La configuration en entreprise en est le reflet : groupes d'attribution, configuration gérée comme l'adresse du serveur, et la personne de l'équipe informatique du client qui approuve une nouvelle version.

Le serveur sprinte, les shells se traînent

Comme les shells ne peuvent pas suivre le rythme du serveur, la règle de compatibilité doit être explicite. Donnez au serveur sa propre version et une version minimale de shell par plateforme — le plus ancien build avec lequel il accepte encore de dialoguer. Le shell envoie sa plateforme et sa version à la connexion, sous forme de paramètres de l'adresse qu'il charge ou via la poignée de main du bridge, et le serveur les enregistre toutes deux dans `Application.Session` pour que chaque ligne de journal et chaque événement de télémétrie ultérieurs puissent les citer. En dessous du minimum, le technicien reçoit un écran de mise à jour bloquant qui pointe vers la bonne destination pour ce canal, et aucune donnée d'ordre de travail n'est chargée. Au niveau du minimum ou au-dessus mais en retard sur la dernière version, affichez un avis que l'on peut fermer et laissez-le continuer à travailler.

```

public ShellGate AdmitShell(string platform, Version shellVersion)
{
    var minimum = _config.MinimumShellVersion(platform);    // e.g. 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);
}

```

Relever le minimum est un acte délibéré, pas un effet de bord d'une release. Ne le relevez que lorsque le serveur commence à dépendre de quelque chose que les anciens shells ne savent pas faire : une nouvelle méthode du bridge, une structure de données modifiée, une autorisation que l'ancien build n'a jamais demandée. Tout le reste, y compris les nouveaux écrans, doit se dégrader proprement sur les shells déjà sur le terrain. La discipline vaut plus que la vérification elle-même, car elle impose la question avant chaque déploiement — que fera de ceci le build Android vieux de deux mois ?

Une télémétrie qui nomme le shell, et la checklist de publication

Avec quatre distributions en production, un rapport de bug qui dit « ça a planté » est presque inutile. Intégrez un SDK de signalement des plantages dans chaque shell, pour que les plantages natifs et les exceptions non gérées du processus du shell arrivent avec une pile d'appels, le modèle de l'appareil et le numéro de build, même s'ils se sont produits en dehors de votre code d'UI C#. Sur le serveur, journalisez les exceptions non gérées avec la plateforme et la version du shell de la session, pour qu'un même incident puisse être lu des deux côtés. Ajoutez ensuite un événement d'usage par fonctionnalité de l'appareil : photo capturée, signature enregistrée, position résolue, notification push reçue, envoi en file d'attente vidé, chacun avec son résultat et le shell dont il provient. Deux semaines de ces événements répondent à ce que les débats ne tranchent pas, par exemple si le repli caméra se déclenche parce qu'une autorisation a été refusée ou parce que le shell est trop ancien pour posséder cette capacité.

La télémétrie est une donnée sur le travail des personnes ; déclarez-la donc. Pas de contenu de photo, pas d'images de signature, pas de noms de clients, pas de coordonnées brutes là où un identifiant d'ordre de travail suffit — et tout ce que vous collectez doit figurer dans la déclaration de confidentialité que vous avez déposée. Le projet de synthèse se termine ensuite par une checklist de publication, car quatre distributions ont quatre modes de défaillance différents, et la checklist est ce qui rend une publication reproductible par quelqu'un qui n'était pas là lors de sa conception.

- Navigateur : serveur déployé, endpoint de version mis à jour, test de fumée sur une session neuve.
- PWA : version du cache du service worker incrémentée, flux de mise à jour vérifié depuis le build précédent.
- Desktop : installateur signé et horodaté, signature vérifiée sur une machine vierge, package Intune remplacé.

- Mobile : bonnes clés et bons profils, version minimale du shell revue, déploiement progressif configuré, tableau de bord des plantages surveillé dès le premier jour.



Réflexe de packaging

L'application reste une application web Wisej.NET servie par votre serveur ; le shell est un véhicule de distribution. Gardez chaque fonctionnalité opérationnelle dans un simple navigateur, puis ajoutez les capacités de l'appareil comme améliorations progressives derrière une vérification de capacités.



Objectifs d'apprentissage

À la fin de ce module, vous serez capable de :

- ✓ Signer les packages desktop et mobiles et expliquer ce que prouve chaque certificat.
- ✓ Choisir entre distribution en store, en entreprise et directe pour chaque plateforme, et préparer chaque soumission.
- ✓ Concevoir une stratégie de gestion des versions et de mise à jour dans laquelle le serveur peut évoluer plus vite que les shells.
- ✓ Ajouter une télémétrie des plantages et de l'usage à tous les shells et livrer le projet de synthèse FieldOps.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



Lab pratique

Terminez le projet de synthèse FieldOps : signez l'installateur Windows et les packages Android et iOS, préparez une distribution en entreprise pour les shells mobiles et un brouillon de fiche store, définissez une règle de compatibilité entre les versions du serveur et celles des shells avec une vérification de version minimale au démarrage, ajoutez le signalement des plantages et un événement d'usage pour chaque fonctionnalité de l'appareil, et livrez une checklist de publication couvrant les quatre modèles de distribution.

Point de départ

Vous ouvrez la solution FieldOps telle que le Module 5 l'a laissée : le serveur Wisej.NET avec la liste des ordres de travail, le détail et le flux de statut, le manifeste et le service worker de la PWA, le shell desktop WebView2, le shell .NET MAUI pour Android et iOS, et le bridge qui expose déjà la caméra, la signature, la localisation et les notifications push derrière des vérifications de capacités. Tout fonctionne actuellement à partir de builds de débogage non signés pointés vers un serveur de développement.

Approche suggérée

- ✓ Produisez d'abord les trois identités de signature — un certificat de signature de code pour Windows, un keystore avec une clé d'import inscrite à Play App Signing, et un certificat de distribution Apple avec un profil de provisionnement incluant l'entitlement push — et consignez pour chacune qui détient la clé et quand elle expire.
- ✓ Construisez et signez un package de release par plateforme, puis vérifiez chaque signature comme le fera l'appareil cible : contrôlez la signature Authenticode et son horodatage sur une machine Windows vierge, et vérifiez que les builds mobiles s'installent depuis leur canal prévu plutôt que par un câble de débogage.
- ✓ Ajoutez la poignée de main de version avant tout autre chargement : exposez depuis la configuration la version du serveur et la version minimale du shell par plateforme, faites en sorte que chaque shell communique sa plateforme et sa version à la connexion, et stockez les deux dans la session.
- ✓ Construisez l'écran de mise à jour que le contrôle affiche à un shell obsolète, avec la destination correspondant à son canal, et un avis que l'on peut fermer pour un shell simplement en retard sur le dernier build.
- ✓ Intégrez le signalement des plantages dans chaque shell et un événement d'usage dans chaque appel au bridge, puis rédigez la checklist de publication en déroulant une publication de bout en bout sur le navigateur, la PWA, le desktop et le mobile.

Livrables attendus

- ✓ Packages Windows, Android et iOS signés

- ✓ Configuration de la distribution en entreprise et brouillon de fiche store
- ✓ Vérification de compatibilité des versions du serveur et des shells
- ✓ Télémétrie des plantages et de l'usage
- ✓ Checklist de publication pour tous les modèles de distribution

Questions de révision

- ✓ Le shell Android d'un technicien a six semaines et le serveur a été livré quatre fois depuis. Laquelle de ces quatre modifications du serveur aurait dû relever la version minimale du shell, et comment l'auriez-vous su à ce moment-là ?
- ✓ L'installateur Windows a été signé mais pas horodaté, et le certificat a expiré le mois dernier. Que font les installations existantes, que font les nouveaux téléchargements, et quel est le chemin le plus court pour retrouver un installateur de confiance ?
- ✓ L'événement d'usage de la caméra montre une forte hausse du chemin de repli navigateur, uniquement sur iOS. Nommez trois explications compatibles avec les données, et indiquez quel champ supplémentaire de l'événement permettrait de les départager.



Réflexe de packaging

L'application reste une application web Wisej.NET servie par votre serveur ; le shell est un véhicule de distribution. Gardez chaque fonctionnalité opérationnelle dans un simple navigateur, puis ajoutez les capacités de l'appareil comme améliorations progressives derrière une vérification de capacités.