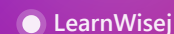


Modèles d'hébergement au-delà du navigateur

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



Guide de la leçon

Navigateur, progressive web app, shell desktop et shell mobile comparés : architecture, capacités, modèle de mise à jour et coût. 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

Navigateur, progressive web app, shell desktop et shell mobile comparés : architecture, capacités, modèle de mise à jour et coût.

1. Le problème en production

FieldOps est une application d'intervention sur le terrain : les techniciens reçoivent des ordres de travail, capturent des photos et des signatures, se rendent sur les sites et travaillent avec une connectivité intermittente. Construite comme une application Wisej.NET ordinaire, elle fonctionne déjà dans n'importe quel navigateur, ce qui suffit aux répartiteurs du bureau, mais pas à un technicien debout dans une salle des machines avec une seule barre de réseau et un téléphone dans une main gantée. La question à laquelle répond ce module n'est pas « pouvons-nous la construire » mais « comment atteint-elle les personnes qui l'utilisent » — et cette réponse est une décision d'architecture, pas une considération de packaging après coup.

2. Concepts clés

Concept	Pourquoi c'est important
modèle de distribution	Le chemin que prend l'application jusqu'à l'utilisateur : un onglet de navigateur, une PWA installée, une fenêtre desktop ou une application mobile distribuée via un store. Il change la distribution, les mises à jour et l'accès à l'appareil — jamais le C# qui construit l'écran.
moteur de navigateur hôte	Chaque modèle affiche la même UI pilotée par le serveur dans un moteur de navigateur : le navigateur de l'utilisateur ou un moteur embarqué — WebView2 sous Windows, WebKit sous macOS et iOS, la WebView système sous Android.
Wisej.NET Hybrid	Le shell du framework basé sur .NET MAUI, qui héberge votre application Wisej.NET existante sur Android, iOS et desktop et ajoute un bridge vers les fonctionnalités de l'appareil comme la caméra, la géolocalisation, les fichiers et les notifications.
bridge vers l'appareil	Le canal bidirectionnel entre votre C# côté serveur et le shell natif : le serveur demande une action sur l'appareil, le shell l'exécute et le résultat revient à votre code sous forme d'événement, pas de valeur de retour.
vérification de capacités	Un test à l'exécution de ce que le shell courant sait réellement faire, utilisé pour afficher, masquer ou remplacer une fonctionnalité afin qu'un même écran se comporte correctement dans chaque modèle de distribution.
cache du shell de l'application	Le HTML, le CSS, les scripts et les icônes statiques qu'un service worker de PWA peut conserver localement. C'est ce qui se charge encore sans réseau — la partie interactive de l'application, rendue par le serveur, n'en fait pas partie.
asymétrie des mises à jour	Le serveur peut être redéployé à tout moment et chaque session obtient le nouveau build au rechargement, alors qu'un installateur desktop ou un package store exige d'abord une release puis une installation.

3. Déroulé de la leçon

Une application, quatre portes d'entrée

Une application Wisej.NET est un processus serveur. Vos objets `Window` et `Page`, leurs contrôles et leurs gestionnaires d'événements vivent dans la mémoire du serveur, et le navigateur en détient un reflet affiché qui renvoie les actions de l'utilisateur via une connexion HTTP et WebSocket. Rien dans cet agencement ne se soucie du moteur de navigateur qui se trouve à l'autre bout. Ce seul fait rend possibles quatre modèles de distribution à partir d'un unique déploiement : la même URL, la même session, le même état `Application.Session`.

Dans un onglet de navigateur, l'utilisateur saisit une adresse ou clique sur un lien et obtient l'application. En tant que progressive web app, la même URL est installée depuis le navigateur : un manifeste lui donne un nom, une icône et une fenêtre standalone sans barre d'adresse, et un service worker met en cache le shell statique pour que le lancement paraisse immédiat. Dans un shell desktop, un petit programme natif ouvre une fenêtre contenant un moteur de navigateur embarqué — WebView2 sous Windows, WebKit sous macOS — et le pointe vers votre serveur ; l'utilisateur voit une entrée dans la barre des tâches et une

barre de titre, et l'application à l'intérieur est celle qu'utilisent les répartiteurs. Dans un shell mobile, Wisej.NET Hybrid héberge l'application dans une app .NET MAUI pour Android et iOS, installée depuis un store ou un canal d'entreprise, et capable d'accéder à l'appareil.

Pour FieldOps, cela signifie que la liste des ordres de travail est écrite une seule fois et n'est jamais portée ni dupliquée. Les shells sont des conteneurs : ce qui les distingue, c'est ce qui entoure l'application, pas ce qui s'exécute à l'intérieur.

La comparaison qui tranche vraiment

Cinq colonnes déterminent le modèle, et la première est celle de l'honnêteté. **Comportement hors ligne** : une UI côté serveur a besoin de son serveur. Quand la connexion tombe, aucun shell ne peut continuer à exécuter vos gestionnaires d'événements, parce que ces gestionnaires ne sont pas sur l'appareil. Une PWA ou un shell natif peut mettre le shell en cache et afficher un écran hors ligne conçu à cet effet avec le dernier récapitulatif synchronisé des ordres de travail, et mettre en file d'attente une action du technicien pour l'envoyer au retour du signal — mais il ne peut pas ouvrir un ordre de travail que l'appareil n'a jamais vu. Promettre davantage, c'est ainsi que les applications de terrain perdent la confiance de leurs utilisateurs dès la première semaine.

L'accès à l'appareil est ce qui distingue les shells. Un navigateur et une PWA obtiennent ce que la plateforme web accorde après une demande d'autorisation : la sélection de fichiers, une image via un contrôle d'envoi, une géolocalisation approximative, les notifications web lorsqu'elles sont prises en charge. Un shell natif ajoute le bridge — une vraie session de caméra, la localisation en arrière-plan, le système de fichiers, les notifications push, la biométrie. **La distribution** va d'une URL envoyée par e-mail à une revue en store qui peut prendre plusieurs jours, en passant par un installateur. **Les mises à jour** suivent le même ordre : le serveur est instantané, un package desktop a besoin d'un canal de mise à jour, un build store a besoin d'une soumission. **Le coût** est la somme des certificats, des comptes développeur, des appareils de test et des cycles de revue dont vous êtes désormais responsable pour toujours.

- Répartiteurs à leur bureau : le navigateur, et rien de plus.
- Techniciens qui ont besoin de la caméra, de la localisation et des notifications : un shell mobile.
- Bornes d'entrepôt qui doivent résister à une coupure du Wi-Fi : une PWA avec une page hors ligne honnête.

Un serveur, une base de code, plusieurs shells

L'architecture qui garde tout cela peu coûteux est volontairement banale : un déploiement unique du serveur Wisej.NET, une seule solution, et des shells qui ne contiennent aucune logique métier. La liste des ordres de travail, la page de détail et le flux de mise à jour du statut sont des écrans Wisej.NET ordinaires qui appellent un `WorkOrderService`. Les projets de shell ajoutés dans les modules suivants font exactement deux choses — héberger le moteur de navigateur pointé vers votre serveur, et lui exposer les fonctionnalités de l'appareil.

La jointure entre les deux est la vérification de capacités. Plutôt que de tester « suis-je sur Android », testez ce dont vous avez besoin : ce shell peut-il prendre une photo, lire une position, émettre une notification. Résolvez ces capacités une fois au démarrage de la session, conservez-les dans l'état de session et laissez chaque écran les lire. Un écran s'adapte alors au lieu de se ramifier par plateforme, et un nouveau shell ajouté l'an prochain ne demande aucune modification des écrans.

```
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(); // plain browser: everything off
}

// In the work-order detail page:
btnTakePhoto.Visible = ShellCapabilities.Current.HasCamera;
uploadPhoto.Visible = !ShellCapabilities.Current.HasCamera;
```

Remarquez le repli : pas un bouton désactivé, mais un contrôle **Upload** qui permet quand même d'ajouter une photo à l'ordre de travail. Chaque capacité a besoin de cette réponse avant que la fonctionnalité ne soit conçue, c'est pourquoi la vérification est un livrable du Module 1, et les fonctionnalités de l'appareil ne le sont pas.

Rédiger le dossier de décision de distribution

Le dernier livrable est un document court, et c'est celui que votre équipe relira vraiment. Il nomme les cibles, l'ordre dans lequel vous les construisez et les critères de sortie qui indiquent qu'une cible est terminée. Le dossier de FieldOps pourrait dire : d'abord le navigateur, parce que les répartiteurs en ont besoin et qu'il valide l'application ; puis le shell Android, parce que c'est ce que les techniciens ont en poche ; puis la PWA pour les bornes d'entrepôt ; iOS uniquement si les tablettes des superviseurs sont approuvées.

Les critères de sortie empêchent une cible de dériver. « Le shell Android est terminé lorsqu'un technicien réalise un ordre de travail de bout en bout sur un appareil fourni, que la session survit au verrouillage de l'écran et qu'un build signé s'installe via le canal d'entreprise » est vérifiable ; « support mobile » ne l'est pas. Rédigez le dossier avant qu'un projet de shell n'existe, car une fois le premier installateur construit, chaque discussion sur le périmètre devient une discussion sur l'effort déjà investi.



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 :

- ✓ Expliquer comment une application Wisej.NET s'exécute sans modification dans un navigateur, une PWA, un shell desktop WebView et un shell mobile.

- ✓ Comparer les quatre modèles de distribution sur le comportement hors ligne, l'accès à l'appareil, la distribution, les mises à jour et le coût.
- ✓ Concevoir l'architecture de FieldOps pour qu'un serveur et une base de code servent chaque shell.
- ✓ Rédiger un dossier de décision de distribution qui nomme les cibles, l'ordre dans lequel les construire et les critères de sortie.

Guide du lab / de l'examen

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



Lab pratique

Créez la solution FieldOps avec une liste des ordres de travail, une page de détail d'ordre de travail et un flux de mise à jour du statut qui fonctionnent dans un simple navigateur. Rédigez le dossier de décision de distribution : comparez le navigateur, la PWA, le shell desktop et le shell mobile au regard des besoins des techniciens, choisissez les cibles et leur ordre, et définissez les vérifications de capacités que l'application utilisera pour activer les fonctionnalités de l'appareil.

Point de départ

Vous partez d'une nouvelle Wisej.NET Web Application vide nommée FieldOps, qui ne contient que la fenêtre de démarrage générée et les fichiers de configuration par défaut. Tout dans ce lab est à créer par vous — le modèle et le service des ordres de travail, les trois écrans, l'abstraction des capacités et le dossier de décision rédigé — et chaque shell ajouté dans les Modules 2 à 6 enveloppe le serveur que vous terminez ici sans le modifier.

Approche suggérée

- ✓ Modélisez d'abord le domaine : un WorkOrder avec numéro, client, adresse du site, priorité, statut et technicien assigné, derrière un WorkOrderService qui renvoie les ordres ouverts du technicien et applique un changement de statut.
- ✓ Construisez la liste des ordres de travail sous forme de grille liée au service, avec un filtre sur le technicien connecté, et rendez-la utilisable dans une fenêtre étroite pour que le même écran résiste plus tard à une zone d'affichage de la taille d'un téléphone.
- ✓ Construisez la page de détail d'un ordre de travail et un flux de mise à jour du statut — Accepted, On site, Completed — qui valide la transition dans le service, l'écrit une seule fois et actualise la liste.
- ✓ Ajoutez un objet ShellCapabilities résolu au démarrage de la session et stocké dans Application.Session, avec des indicateurs pour la caméra, la localisation et les notifications, tous à false dans un simple navigateur, et faites en sorte que la page de détail le lise pour décider de ce qu'elle affiche.
- ✓ Rédigez le tableau comparatif et le dossier de décision de distribution, puis dessinez le schéma d'architecture montrant un serveur, une base de code et les shells qui pointeront vers lui.

Livrables attendus

- ✓ Solution FieldOps avec liste, détail et mise à jour du statut
- ✓ Tableau comparatif des modèles de distribution
- ✓ Dossier de décision de distribution avec l'ordre des cibles
- ✓ Conception de la vérification des capacités
- ✓ Schéma d'architecture avec un serveur et plusieurs shells

Questions de révision

- ✓ Un technicien perd le signal au milieu de la mise à jour d'un ordre de travail. Décrivez précisément quelles parties de FieldOps peuvent encore être affichées depuis un shell en cache, lesquelles ne le peuvent pas, et ce que votre écran doit faire au moment où la connexion tombe.
- ✓ Votre dossier de décision place le shell Android avant la PWA. Qu'est-ce qui devrait être vrai des appareils et du réseau des techniciens pour que cet ordre soit mauvais ?
- ✓ La page de détail a besoin d'une photo. Expliquez pourquoi une vérification de capacités associée à un repli Upload est une meilleure conception qu'une option de compilation qui produit une version de l'écran réservée au mobile.



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.