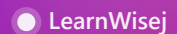


Mise en place d'une Progressive Web App

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



Guide de la leçon

Manifeste d'application web, icônes et écrans de démarrage, service worker, installabilité, shell hors ligne et flux de mise à jour d'une PWA. 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

Manifeste d'application web, icônes et écrans de démarrage, service worker, installabilité, shell hors ligne et flux de mise à jour d'une PWA.

1. Le problème en production

Les techniciens de FieldOps n'ouvrent pas un navigateur pour taper une URL. Ils touchent une icône sur un téléphone resté toute la matinée dans une camionnette, dans une salle des machines où le signal tombe à zéro au milieu d'une intervention. Une progressive web app leur donne cette icône, une fenêtre sans chrome de navigateur et une page qui dit encore quelque chose d'utile quand le réseau disparaît — mais FieldOps est une application Wisej.NET, donc son interface utilisateur vit sur le serveur, et faire comme si ce n'était pas le cas, c'est ainsi qu'une PWA se transforme en ticket de support.

2. Concepts clés

Concept	Pourquoi c'est important
manifeste d'application web	Un petit fichier JSON, référencé depuis la page de démarrage, qui indique au navigateur le nom de l'application, ses icônes, ses couleurs, son mode d'affichage et son URL de démarrage — sans lui, le navigateur n'a rien à installer.
mode d'affichage	Le membre <code>display</code> du manifeste. <code>standalone</code> lance FieldOps dans sa propre fenêtre sans barre d'adresse, ce qui fait qu'une application installée ne ressemble plus à un simple favori.
icônes maskable	Des icônes déclarées avec <code>purpose: "maskable"</code> et une marge de sécurité, pour qu'Android puisse les découper selon sa propre forme sans rogner le visuel.
service worker	Un script que le navigateur exécute à côté de la page, servi depuis l'origine même de l'application en HTTPS, et qui peut répondre aux requêtes de la page depuis un cache quand la connexion est lente ou absente.
shell de l'application	Les fichiers statiques dont une session a besoin avant que le serveur puisse dire quoi que ce soit : la page de démarrage, les scripts client, le thème et les icônes. C'est ce que le worker est autorisé à mettre en cache.
cache d'abord ou réseau uniquement	La règle de routage du gestionnaire fetch du worker : les fichiers statiques du shell viennent d'abord du cache, tandis que tout ce que la session Wisej.NET échange avec le serveur doit aller directement au réseau, jamais à une copie en cache.
cycle de vie des mises à jour	Un nouveau worker s'installe pendant que l'ancien contrôle encore la page, puis il attend. <code>skipWaiting</code> , <code>clients.claim</code> et un rechargement maîtrisé sont ce qui transforme ce worker en attente en version réellement utilisée par le technicien.

3. Déroulé de la leçon

Le manifeste est le contrat d'installation

L'installabilité n'est pas un interrupteur dans Wisej.NET ; c'est un ensemble de conditions que vérifie le navigateur. FieldOps doit être servi en HTTPS, sa page de démarrage doit référencer un manifeste d'application web valide, et un service worker doit être enregistré pour la portée de la page. Le manifeste se trouve avec les autres fichiers statiques de l'application — dans le même dossier que la page de démarrage, le thème et les icônes — et il est référencé depuis `Default.html`, le document de démarrage de Wisej.NET et le seul endroit du projet où vous maîtrisez le HTML du navigateur.

Les membres importants sont peu nombreux. `name` et `short_name` déterminent ce qui apparaît sous l'icône et dans le sélecteur de tâches. `icons` exige un PNG de 192 pixels et un de 512 pixels, plus des variantes maskable avec un visuel entouré de marges, et iOS veut en outre un lien `apple-touch-icon` dans la page elle-même. `theme_color` teinte les barres système autour de la fenêtre standalone, tandis que `background_color` colore l'écran de démarrage que le navigateur génère au lancement ; les deux doivent donc correspondre à la barre d'application de FieldOps. `display` vaut `standalone`, et

`start_url` ainsi que `scope` pointent vers la racine de l'application, pour qu'un lancement arrive sur la liste des ordres de travail et que chaque URL de l'application reste dans la fenêtre installée.

Quand le navigateur est satisfait, il déclenche `beforeinstallprompt` sur la page au lieu d'afficher sa propre bannière. Le gestionnaire dans `Default.html` bloque ce comportement par défaut, conserve l'événement et expose une petite fonction qui appelle sa méthode `prompt()`. Deux règles en découlent. L'invitation ne s'ouvre qu'à partir d'un vrai clic dans le navigateur, donc le bouton qui la déclenche appartient à la page, pas au serveur. Et c'est le serveur qui décide *quand* l'offre vaut la peine d'être faite — le lab la demande après la première session réussie — en révélant le bouton via `Application.Eval` ou l'interopérabilité JavaScript équivalente :

```
private void OnFirstSessionCompleted()
{
    // Default.html stashed the beforeinstallprompt event; the server only
    // decides when the offer becomes visible. prompt() still runs from the
    // page's own click handler, inside a real user gesture.
    Application.Session["InstallOffered"] = true;
    Application.Eval("window.fieldOps.showInstallButton()");
}
```

Un worker qui met en cache le shell, jamais la session

Enregistrez le worker depuis `Default.html` avec `navigator.serviceWorker.register`, en pointant vers un `sw.js` servi depuis la racine de l'application pour que sa portée couvre toute l'application. Sur `install`, le worker ouvre un cache nommé et y place le shell : la page de démarrage, les scripts client et les fichiers de thème que la page charge avant que la session n'existe, le manifeste, les icônes et la page hors ligne. Sur `activate`, il supprime tous les caches dont le nom n'est pas celui en cours.

Le gestionnaire `fetch` est l'endroit où une PWA au-dessus d'une UI pilotée par le serveur réussit ou échoue. FieldOps est une application Wisej.NET : les contrôles sont des objets C# sur le serveur, et ce qui transite pendant une session est un flux de requêtes et de réponses reflétant cet état serveur, plus un WebSocket pour les mises à jour poussées. Rien de tout cela n'est cachable dans un sens utile — une réponse rejouée décrit une session qui n'existe plus. Le gestionnaire filtre donc selon la requête : un `GET` sur un fichier connu du shell est servi depuis le cache d'abord, les requêtes autres que `GET` ne sont jamais touchées, et tout le reste part vers le réseau sans modification. Quand une navigation échoue parce que le serveur est injoignable, le gestionnaire répond avec la page hors ligne en cache au lieu de l'écran d'erreur du navigateur.

Un hors ligne honnête pour une UI pilotée par le serveur

Dites clairement ce que signifie hors ligne ici : le shell se charge, l'application ne tourne pas. Sans le serveur, il n'y a ni session, ni données d'ordres de travail, ni moyen de changer un statut, et aucune quantité de cache ne déplace une UI côté serveur dans le navigateur. Le travail de conception consiste à rendre cet état utile plutôt que cassé. FieldOps écrit un petit récapitulatif après chaque chargement réussi — nombre d'ordres de travail ouverts, prochain site et heure de la synchronisation — dans le stockage du navigateur, et la page hors ligne l'affiche derrière un bandeau clair « not connected » et un bouton pour réessayer.

- Rien de ce qui est saisi sur la page hors ligne n'est présenté comme enregistré ; les modifications en file d'attente sont une fonctionnalité ultérieure et délibérée, pas un effet de bord d'un cache.
- Chaque surface hors ligne porte l'horodatage de la dernière synchronisation, pour que des données anciennes ne soient jamais prises pour des données actuelles.
- Au retour de la connectivité, l'application se recharge dans une nouvelle session plutôt que de reprendre une session périmée.

La reconnexion mérite la même honnêteté. Une session Wisej.NET peut expirer ou être perdue pendant que l'appareil dort dans une camionnette ; la récupération fiable consiste à recharger la page de démarrage et à laisser le serveur construire une nouvelle session, pas à laisser à l'écran une UI morte qui a l'air vivante.

Livrer une nouvelle version malgré un cache qui fait son travail

Le cache qui rend FieldOps rapide est aussi celui qui laisse un technicien sur le build du mois dernier. Versionnez le nom du cache — `fieldops-shell-v7` — et changez-le dans le même commit que tout fichier du shell. Le nouveau worker s'installe alors dans un nouveau cache, et son gestionnaire `activate` supprime les anciens. Appelez `skipWaiting` pour que le nouveau worker n'attende pas derrière l'ancien, et `clients.claim` pour qu'il contrôle immédiatement la page ouverte.

La page doit encore réagir. Écouter `controllerchange`, ou `updatefound` et `statechange` sur l'enregistrement, lui indique qu'une nouvelle version a pris la main ; FieldOps affiche une petite barre « A new version is ready — Reload » et recharge quand le technicien la touche, pour que personne ne perde un formulaire à moitié rempli à cause d'un rafraîchissement surprise. Deux détails de déploiement complètent le flux : `sw.js` et le manifeste sont servis avec des en-têtes no-cache pour que le navigateur puisse seulement remarquer qu'ils ont changé, et l'application compare son estampille de build avec celle du serveur au démarrage de la session, repérant ainsi un shell qui a survécu à un déploiement au lieu de le laisser dialoguer avec un serveur auquel il ne correspond plus.



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 :

- ✓ Ajouter un manifeste d'application web avec icônes, couleurs et mode d'affichage pour que l'application puisse être installée depuis le navigateur.
- ✓ Enregistrer un service worker qui met en cache le shell de l'application et le sert lorsque le réseau est lent ou absent.
- ✓ Expliquer ce qui peut et ne peut pas fonctionner hors ligne dans une UI côté serveur, et concevoir l'expérience hors ligne en toute honnêteté.

- ✓ Gérer les mises à jour de la PWA pour que les utilisateurs obtiennent la nouvelle version sans cache périmé.

Guide du lab / de l'examen

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



Lab pratique

Transformez FieldOps en PWA : ajoutez le manifeste avec les icônes, la couleur de thème et l'affichage standalone, enregistrez un service worker qui met en cache le shell et les ressources statiques, affichez une invitation à l'installation après la première session réussie, affichez une page hors ligne avec le dernier récapitulatif synchronisé des ordres de travail lorsque le serveur est injoignable, et implémentez le flux de mise à jour qui recharge l'application quand une nouvelle version est disponible.

Point de départ

Vous ouvrez la solution FieldOps construite au Module 1 : une application web Wisej.NET 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 déjà correctement dans un simple onglet de navigateur. Rien dans l'application n'est encore installable — il n'y a ni manifeste, ni service worker, ni page hors ligne — et les fichiers statiques que vous allez ajouter se placent à côté de `Default.html` et des fichiers de thème et d'icônes existants.

Approche suggérée

- ✓ Servez FieldOps en HTTPS en local, puis écrivez `manifest.webmanifest` avec le nom de l'application, des icônes de 192 et 512 pixels incluant des variantes maskable, les couleurs de thème et d'arrière-plan de FieldOps, `display: standalone` et une URL de démarrage à la racine de l'application, et référencez-le, ainsi qu'une apple-touch-icon, depuis `Default.html`.
- ✓ Ajoutez `sw.js` à la racine de l'application avec un nom de cache versionné, un gestionnaire install qui précharge le shell en cache — page de démarrage, scripts client, thème, icônes et page hors ligne — et un gestionnaire activate qui supprime tous les caches d'une version plus ancienne.
- ✓ Écrivez le gestionnaire fetch de sorte que les requêtes du shell soient servies depuis le cache d'abord, tandis que chaque requête de la session Wisej.NET vers le serveur part vers le réseau sans modification, et qu'une navigation en échec se replie sur la page hors ligne en cache.
- ✓ Interceptez `beforeinstallprompt` dans la page, conservez l'événement, et faites révéler par le serveur le bouton Install de la page via l'interopérabilité JavaScript, uniquement après que le technicien a terminé une session réussie avec la liste des ordres de travail chargée.
- ✓ Écrivez un récapitulatif compact des ordres de travail dans le stockage du navigateur après chaque chargement réussi, pour que la page hors ligne puisse l'afficher avec l'horodatage de la dernière synchronisation, puis incrémentez la version du cache, déployez, et vérifiez que la barre de mise à jour apparaît et recharge dans le nouveau build.

Livrables attendus

- ✓ Manifeste d'application web avec icônes et couleurs
- ✓ Service worker mettant en cache le shell

- ✓ Invitation à l'installation après la première session
- ✓ Page hors ligne avec le dernier récapitulatif synchronisé
- ✓ Flux de mise à jour et vérification de version

Questions de révision

- ✓ Un relecteur demande pourquoi le service worker ne met pas simplement en cache chaque réponse pour que FieldOps continue de fonctionner en sous-sol. Expliquez ce qu'une réponse de session Wisej.NET rejouée ferait réellement à l'écran du technicien, et où se situe la limite honnête.
- ✓ Votre offre d'installation fonctionne sur votre poste mais n'apparaît jamais sur le téléphone du technicien. Nommez trois conditions distinctes qui pourraient en être la cause, et indiquez comment vous les distingueriez depuis l'appareil.
- ✓ Un technicien a l'application installée et ouverte lorsque vous déployez un nouveau build. Décrivez chaque étape entre votre déploiement et l'exécution du nouveau shell sur ce téléphone, et identifiez l'étape qui échouerait silencieusement si le nom du cache ne changeait jamais.



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.