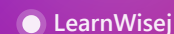


Packaging desktop avec un shell WebView

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



Guide de la leçon

Héberger l'application dans une fenêtre desktop native, WebView2 sous Windows, chrome de fenêtre, système de fichiers et intégration au système d'exploitation, et installateurs. 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

Héberger l'application dans une fenêtre desktop native, WebView2 sous Windows, chrome de fenêtre, système de fichiers et intégration au système d'exploitation, et installateurs.

1. Le problème en production

Les techniciens de FieldOps passent toute leur journée dans l'application : un portable dans la camionnette, une station d'accueil au dépôt. Le build navigateur des deux premiers modules fait le travail, mais il n'a pas d'icône dans le menu Démarrer, aucune présence dans la zone de notification quand une intervention est attribuée, aucun moyen de déposer un rapport exporté dans un dossier choisi par le technicien, et rien que le service informatique puisse déployer par son canal logiciel habituel. Ce module enveloppe cette même application, sans la modifier, dans un shell Windows qui apporte exactement ces quatre éléments.

2. Concepts clés

Concept	Pourquoi c'est important
contrôle WebView2	Le moteur Chromium que Microsoft distribue sous la forme de l'Evergreen Runtime redistribuable, intégré dans une fenêtre WinForms ou WPF pour que votre page s'affiche exactement comme dans Edge.
shell desktop	Un petit exécutable natif qui possède la fenêtre, l'icône de notification et les appels au système, et qui pointe un contrôle WebView2 vers votre application Wisej.NET.
chrome de fenêtre	Le titre, l'icône, la taille minimale, la position mémorisée et le comportement en instance unique qui donnent à l'application l'allure d'un logiciel installé plutôt qu'embarqué.
bridge de messages	Le canal bidirectionnel entre la page et le shell : <code>PostWebMessageAsJson</code> vers la page, <code>WebMessageReceived</code> vers le shell, ou un objet hôte publié avec <code>AddHostObjectToScript</code> .
interopérabilité JavaScript	La façon dont le C# côté serveur atteint ce canal : <code>Application.Eval</code> et <code>Application.Call</code> exécutent du script dans le contexte du navigateur de la session, et la réponse revient sous forme d'événement client.
topologie du serveur	Le fait que le shell navigue vers un serveur distant de l'entreprise ou démarre un processus Wisej.NET auto-hébergé sur le loopback — décidé au lancement et journalisé, jamais deviné.
canal de mise à jour	Un endpoint de version que le shell interroge au démarrage, plus l'installateur qu'il récupère quand la réponse est plus récente que la version qu'il exécute.

3. Déroulé de la leçon

Une fenêtre native autour de la même page

Le shell est plus petit que ce qu'imaginent les développeurs : un exécutable WinForms ou WPF contenant un seul contrôle `WebView2` ancré pour remplir la zone cliente, initialisé de façon asynchrone, puis dirigé vers l'adresse de votre application. Vous pouvez tout aussi bien partir de la cible desktop de Wisej.NET Hybrid, qui package le même agencement pour vous. La page à l'intérieur est la session FieldOps ordinaire — les mêmes contrôles `Wisej.Web` côté serveur, le même thème et la même logique de mise en page. Rien dans l'application ne sait qu'elle est encadrée.

Ce que le shell ajoute, c'est tout ce qui se trouve hors de la page. Il définit le titre de la fenêtre avec le nom du technicien dès que la session indique qui s'est connecté, donne à la fenêtre une icône et une taille minimale, restaure sa dernière position et refuse de s'ouvrir deux fois — un second lancement active la fenêtre déjà ouverte. Un `NotifyIcon` place FieldOps dans la zone de notification avec un menu court : Open, Sync now, Exit. La fermeture masque la fenêtre dans la zone de notification au lieu de terminer le processus, si bien que la notification toast arrive toujours et que la session reste là où le technicien l'a laissée.

Deux choses ne doivent pas se perdre. WebView2 requiert l'Evergreen Runtime ; l'installateur le détecte donc et l'amorce au lieu d'échouer au premier lancement. Et chaque capacité ajoutée par le shell est un ajout,

jamais un remplacement : le même build doit toujours fonctionner dans un simple onglet de navigateur, et c'est à cela que sert la vérification de capacités du lab.

Un bridge, deux directions

WebView2 vous fournit le canal. Du shell vers la page, `CoreWebView2.PostWebMessageAsJson` transmet un objet JSON qui arrive sous forme d'événement message. De la page vers le shell, le script appelle `window.chrome.webview.postMessage` et le shell traite `WebMessageReceived` en lisant `WebMessageAsJson`. Une troisième option, `AddHostObjectToScript`, publie un objet visible par COM que le script appelle comme une API locale — pratique, mais avec une surface bien plus large, car tout ce qui est chargé dans cette page peut l'appeler.

Votre C# vit sur le serveur ; il atteint donc ce canal par l'interopérabilité JavaScript : `Application.Eval` exécute un fragment dans le contexte du navigateur de la session et `Application.Call` invoque une fonction que vous avez enregistrée côté client. La réponse du shell revient sous forme d'événement client qui déclenche un gestionnaire côté serveur, lequel met à jour les contrôles et laisse Wisej.NET les redessiner. L'aller-retour étant asynchrone, chaque requête porte un identifiant — sinon deux exports lancés coup sur coup échangeront leurs résultats.

FieldOps utilise ce bridge à deux reprises. L'export envoie une requête `pickFolder` ; le shell ouvre une boîte de dialogue native de sélection de dossier, renvoie le chemin choisi, et le serveur y écrit le rapport. L'attribution fonctionne en sens inverse : le serveur pousse le nouvel ordre dans la session, la page poste un message `notify`, et le shell affiche depuis l'icône de notification un toast qui active la fenêtre sur cet ordre. Dans un simple navigateur, aucune de ces requêtes n'a d'écouteur ; la vérification de capacités masque donc le chemin natif et l'export se replie sur `Application.Download`.

```
// Shell (WinForms): answer one request from the page.
webView.CoreWebView2.WebMessageReceived += (s, e) =>
{
    var req = JsonSerializer.Deserialize<ShellRequest>(e.WebMessageAsJson);
    if (req.Action != "pickFolder") return;

    using var dlg = new FolderBrowserDialog();
    var path = dlg.ShowDialog() == DialogResult.OK ? dlg.SelectedPath : null;

    webView.CoreWebView2.PostWebMessageAsJson(
        JsonSerializer.Serialize(new { id = req.Id, path }));
};
```

Où s'exécute réellement le serveur

Le shell n'est pas l'application ; il faut donc encore dire où se trouve l'application. La topologie distante pointe le contrôle WebView2 vers le serveur de l'entreprise : un déploiement, une base de données, une mise à jour qui atteint chaque technicien dès qu'il se reconnecte. La topologie locale embarque un processus Wisej.NET auto-hébergé que le shell démarre sur un port loopback ; elle fonctionne sans aucun signal, mais chaque camionnette transporte alors une copie de l'application et un stockage que quelqu'un devra réconcilier.

FieldOps configure les deux. Un petit `shell.config.json` à côté de l'exécutable indique le serveur principal, le délai de la sonde et si le repli local est autorisé. Au lancement, le shell appelle un endpoint de santé ; s'il ne répond pas en quelques secondes, il démarre le serveur embarqué et affiche un bandeau permanent indiquant **Working offline — local server**. Le mode est visible dans la fenêtre et écrit dans le journal du shell, si bien qu'un appel au support ne commence jamais par deviner à quel serveur le technicien parlait.

Chaque mode a ses conséquences. Les cookies du serveur distant n'ont aucune valeur pour le serveur local ; le mode local authentifie donc sur son propre stockage et marque ce qu'il capture pour une synchronisation ultérieure. Le port loopback est choisi à l'exécution — un port fixe est justement celui qu'un autre produit a déjà pris — et le shell arrête le processus enfant à la sortie, y compris après un plantage.

L'installer, et le maintenir à jour

Deux voies de packaging sont raisonnables. MSIX offre une installation par utilisateur, une désinstallation propre, une gestion des versions par Windows lui-même et un déploiement via Intune, au prix de la signature et de possibilités limitées au moment de l'installation. Un installateur classique WiX Toolset ou Inno Setup donne un contrôle total : raccourcis, bootstrapper du runtime WebView2, valeurs de registre, service. Choisissez selon la façon dont le logiciel est distribué — FieldOps livre un package Inno Setup parce que les camionnettes sont réimagées et que l'amorçage du runtime doit faire partie de l'installation.

Le canal de mise à jour demande peu de travail pour un gain important. Au lancement, avant de naviguer, le shell lit la version de son propre assembly et demande la version actuelle à un endpoint de version. Si la réponse est plus récente, il affiche une invite non bloquante décrivant les changements, télécharge l'installateur, le vérifie, l'exécute et se ferme. Il n'interrompt jamais un technicien en pleine intervention, et une vérification restée sans réponse est un avertissement journalisé plutôt qu'un échec au démarrage.

Deux numéros de version coexistent désormais. Le serveur doit évoluer plus vite que les shells installés ; l'endpoint publie donc aussi la version minimale du shell qu'il prend encore en charge, ce qui permet au serveur d'avertir — ou de refuser — lorsqu'une camionnette est restée hors ligne pendant deux mois. La signature et la distribution relèvent du dernier module ; ici, le canal doit seulement exister et être honnête.

- L'Evergreen Runtime est amorcé par l'installateur, pas supposé présent.
- Chaque requête du bridge porte un id, et chaque réponse lui est associée.
- Le mode de serveur est visible dans la fenêtre et écrit dans le journal.



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 :

- ✓ Héberger l'application Wisej.NET dans un shell desktop basé sur WebView2, avec une fenêtre native, une barre de titre et une icône dans la zone de notification.
- ✓ Relier l'application web et le shell pour que l'application puisse appeler des fonctionnalités natives comme les boîtes de dialogue de fichiers, l'impression et les notifications.
- ✓ Décider si le serveur s'exécute localement à côté du shell ou à distance, et configurer chacun des deux cas.
- ✓ Construire un installeur et un canal de mise à jour pour le package desktop.

Guide du lab / de l'examen

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



Lab pratique

Packagez FieldOps pour Windows : un shell WebView2 avec une fenêtre native, un titre personnalisé et une icône dans la zone de notification, un bridge qui permet à l'application d'ouvrir un sélecteur de dossier natif pour exporter les rapports d'ordres de travail et d'afficher une notification native à l'arrivée d'un nouvel ordre, une configuration qui dirige le shell vers le serveur distant avec un repli local, et un installateur avec une vérification des mises à jour au lancement.

Point de départ

Vous ouvrez la solution FieldOps telle qu'elle se présente après le Module 2 : la liste des ordres de travail, la page de détail et le flux de mise à jour du statut fonctionnent tous dans un navigateur, et l'application s'installe déjà comme PWA avec son manifeste et son service worker. Rien ne change dans ce projet web pendant ce lab — vous ajoutez à côté un second projet, un exécutable de shell Windows.

Approche suggérée

- ✓ Ajoutez un projet de shell WinForms ou WPF avec un unique contrôle WebView2 qui remplit la fenêtre, initialisez-le de façon asynchrone et dirigez-le vers l'adresse de FieldOps avant d'écrire la moindre ligne de code du bridge.
- ✓ Donnez à la fenêtre son chrome : l'icône de FieldOps, un titre qui affiche le technicien connecté, une taille et une position mémorisées, l'activation en instance unique, et un NotifyIcon avec Open, Sync now et Exit qui maintient le processus en vie quand la fenêtre est fermée.
- ✓ Construisez le bridge dans les deux sens — traitez WebMessageReceived dans le shell pour une requête pickFolder et une requête notify, répondez avec PostWebMessageAsJson en renvoyant l'id de la requête, et côté serveur atteignez la page avec Application.Eval et recevez la réponse sous forme d'événement client.
- ✓ Écrivez shell.config.json à côté de l'exécutable avec l'adresse du serveur distant, le délai de la sonde et l'option de repli local, puis démarrez le serveur auto-hébergé embarqué et affichez le bandeau hors ligne lorsque la sonde de santé ne répond pas.
- ✓ Produisez l'installateur en y incluant le bootstrapper du runtime WebView2, et ajoutez la vérification de version au lancement qui compare la version de l'assembly du shell avec l'endpoint de version et propose le nouvel installateur sans bloquer le démarrage.

Livrables attendus

- ✓ Shell desktop WebView2 avec fenêtre et icône de notification
- ✓ Bridge natif pour le sélecteur de dossier et les notifications
- ✓ Configuration du serveur distant et local
- ✓ Package d'installation
- ✓ Vérification des mises à jour au lancement

Questions de révision

- ✓ Un technicien lance deux exports à moins d'une seconde d'intervalle et le shell répond aux deux boîtes de dialogue de dossier. Qu'est-ce qui, dans la conception de votre bridge, empêche le second chemin d'être écrit dans le premier rapport, et que se passerait-il sans cela ?
- ✓ Votre bouton Export ouvre désormais une boîte de dialogue native de sélection de dossier. Décrivez précisément ce que fait ce même bouton lorsque la page est ouverte dans Edge plutôt que dans le shell, et à quel endroit de votre code cette différence est décidée.
- ✓ L'endpoint de version est injoignable parce que la camionnette n'a pas de réseau. Argumentez sur ce que le shell doit faire lors de ce lancement, et expliquez ce qu'un technicien perdrait avec chacune des alternatives que vous avez écartées.



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.