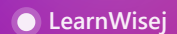


Desktop-Paketierung mit einer WebView-Shell

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Die App in einem nativen Desktop-Fenster hosten, WebView2 unter Windows, Fensterrahmen, Dateisystem- und Betriebssystemintegration sowie Installer. 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

Die App in einem nativen Desktop-Fenster hosten, WebView2 unter Windows, Fensterrahmen, Dateisystem- und Betriebssystemintegration sowie Installer.

1. Das Problem im Produktivbetrieb

Die Techniker von FieldOps verbringen ihre gesamte Schicht in der App: ein Laptop im Transporter, eine Dockingstation im Depot. Der Browser-Build aus den ersten beiden Modulen erledigt die Arbeit, aber er hat kein Icon im Startmenü, keine Präsenz im Infobereich, wenn ein Auftrag zugewiesen wird, keine Möglichkeit, einen exportierten Bericht in einem vom Techniker gewählten Ordner abzulegen, und nichts, was die IT über ihren üblichen Softwarekanal verteilen könnte. Dieses Modul verpackt dieselbe Anwendung unverändert in eine Windows-Shell, die genau diese vier Dinge liefert.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
WebView2-Control	Die Chromium-Engine, die Microsoft als weiterverteilbare Evergreen Runtime ausliefert, eingebettet in ein WinForms- oder WPF-Fenster, sodass Ihre Seite genau wie in Edge gerendert wird.
Desktop-Shell	Eine kleine native ausführbare Datei, die das Fenster, das Tray-Icon und die Betriebssystemaufrufe besitzt und ein WebView2-Control auf Ihre Wisej.NET-Anwendung richtet.
Fensterrahmen	Titel, Icon, Mindestgröße, gespeicherte Position und Einzelinstanz-Verhalten – all das, was die App installiert statt eingebettet wirken lässt.
Nachrichten-Bridge	Der bidirektionale Kanal zwischen Seite und Shell: <code>PostWebMessageAsJson</code> nach unten, <code>WebMessageReceived</code> nach oben, oder ein mit <code>AddHostObjectToScript</code> veröffentlichtes Host-Objekt.
JavaScript-Interop	Wie serverseitiger C#-Code diesen Kanal erreicht: <code>Application.Eval</code> und <code>Application.Call</code> führen Skript im Browserkontext der Session aus, und die Antwort kommt als Client-Event zurück.
Server-Topologie	Ob die Shell zu einem entfernten Unternehmensserver navigiert oder einen selbst gehosteten Wisej.NET-Prozess auf Loopback startet – beim Start entschieden und protokolliert, niemals geraten.
Update-Kanal	Ein Versions-Endpunkt, den die Shell beim Start abfragt, plus der Installer, den sie herunterlädt, wenn die Antwort neuer ist als die laufende Version.

3. Der rote Faden der Lektion

Ein natives Fenster um dieselbe Seite

Die Shell ist kleiner, als Entwickler erwarten: eine WinForms- oder WPF-Anwendung mit einem einzigen `WebView2`-Control, das den Clientbereich ausfüllt, asynchron initialisiert und dann auf Ihre Anwendungsadresse navigiert wird. Genauso gut können Sie mit dem Desktop-Ziel von Wisej.NET Hybrid beginnen, das dieselbe Anordnung für Sie paketierte. Die Seite darin ist die gewöhnliche FieldOps-Session – dieselben serverseitigen `Wisej.Web`-Controls, dasselbe Theme, dieselbe Layout-Logik. Nichts in der App weiß, dass sie eingerahmt wird.

Was die Shell ergänzt, ist alles außerhalb der Seite. Sie setzt den Fenstertitel auf den Namen des Technikers, sobald die Session meldet, wer sich angemeldet hat, gibt dem Fenster ein Icon und eine Mindestgröße, stellt die letzte Position wieder her und verweigert einen zweiten Start – ein zweiter Aufruf aktiviert das laufende Fenster. Ein `NotifyIcon` platziert FieldOps im Infobereich mit einem kurzen Menü: Öffnen, Jetzt synchronisieren, Beenden. Beim Schließen wird das Fenster in den Tray minimiert, statt den Prozess zu beenden, sodass die Toast-Benachrichtigung weiterhin ankommt und die Session dort bleibt, wo der Techniker sie verlassen hat.

Zwei Dinge dürfen nicht verloren gehen. WebView2 benötigt die Evergreen Runtime, daher erkennt der Installer sie und installiert sie bei Bedarf nach, statt beim ersten Start zu scheitern. Und jede Fähigkeit, die die Shell hinzufügt, ist eine Ergänzung, nie ein Ersatz: Derselbe Build muss weiterhin in einem einfachen Browser-Tab laufen – genau dafür ist der Capability-Check im Lab da.

Eine Bridge, zwei Richtungen

WebView2 stellt Ihnen den Kanal bereit. Von der Shell zur Seite liefert

`CoreWebView2.PostWebMessageAsJson` ein JSON-Objekt, das als Message-Event ankommt. In der Gegenrichtung ruft das Skript `window.chrome.webview.postMessage` auf, und die Shell behandelt `WebMessageReceived` und liest `WebMessageAsJson`. Eine dritte Option, `AddHostObjectToScript`, veröffentlicht ein COM-sichtbares Objekt, das Skript wie eine lokale API aufruft – bequem, aber mit einer deutlich größeren Angriffsfläche, denn alles, was in diese Seite geladen wird, kann es aufrufen.

Ihr C#-Code lebt auf dem Server und erreicht den Kanal daher über JavaScript-Interop:

`Application.Eval` führt ein Fragment im Browserkontext der Session aus, und `Application.Call` ruft eine Funktion auf, die Sie auf dem Client registriert haben. Die Antwort der Shell kommt als Client-Event zurück, das einen serverseitigen Handler auslöst, der die Controls aktualisiert und Wisej.NET sie neu zeichnen lässt. Weil der Hin- und Rückweg asynchron ist, trägt jede Anfrage eine Kennung – sonst vertauschen zwei kurz hintereinander gestartete Exporte ihre Ergebnisse.

FieldOps nutzt diese Bridge zweimal. Der Export sendet eine `pickFolder`-Anfrage; die Shell öffnet einen nativen Ordnerdialog, gibt den gewählten Pfad zurück, und der Server schreibt den Bericht dorthin. Die Zuweisung funktioniert umgekehrt: Der Server pusht den neuen Auftrag in die Session, die Seite sendet eine `notify`-Nachricht, und die Shell löst über das Tray-Icon eine Toast-Benachrichtigung aus, die das Fenster auf diesem Auftrag aktiviert. In einem einfachen Browser hat keine der beiden Anfragen einen Empfänger, daher blendet der Capability-Check den nativen Weg aus, und der Export fällt auf `Application.Download` zurück.

```
// Shell (WinForms): eine Anfrage der Seite beantworten.
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 }));
};
```

Wo der Server tatsächlich läuft

Die Shell ist nicht die Anwendung, Sie müssen also weiterhin angeben, wo die Anwendung ist. Die Remote-Topologie richtet das WebView2-Control auf den Unternehmensserver: ein Deployment, eine Datenbank, ein Update, das jeden Techniker in dem Moment erreicht, in dem er sich wieder verbindet. Die lokale Topologie bündelt einen selbst gehosteten Wisej.NET-Prozess, den die Shell auf einem Loopback-Port

startet; das funktioniert ganz ohne Empfang, aber dann trägt jeder Transporter eine Kopie der App und einen Datenspeicher mit sich, den jemand abgleichen muss.

FieldOps konfiguriert beides. Eine kleine `shell.config.json` neben der ausführbaren Datei nennt den primären Server, das Timeout für die Prüfung und ob der lokale Fallback erlaubt ist. Beim Start ruft die Shell einen Health-Endpunkt auf; antwortet dieser nicht innerhalb weniger Sekunden, startet sie den mitgelieferten Server und zeigt dauerhaft einen Streifen mit dem Text **Offline-Betrieb – lokaler Server**. Der Modus ist im Fenster sichtbar und wird ins Shell-Log geschrieben, sodass ein Supportanruf nie mit dem Rätselraten beginnt, mit welchem Server der Techniker verbunden war.

Jeder Modus hat Folgen. Cookies des Remote-Servers bedeuten dem lokalen nichts, daher meldet sich der lokale Modus gegen seinen eigenen Speicher an und markiert, was er erfasst, für die spätere Synchronisierung. Der Loopback-Port wird zur Laufzeit gewählt – ein fester Port ist genau der, den ein anderes Produkt schon belegt hat –, und die Shell beendet den Kindprozess beim Beenden, auch nach einem Absturz.

Installieren und aktuell halten

Zwei Paketierungswege sind sinnvoll. MSIX bietet eine benutzerbezogene Installation, eine saubere Deinstallation, Versionsverwaltung durch Windows selbst und Verteilung über Intune – um den Preis der Signierung und eines eingeschränkten Spielraums bei der Installation. Ein klassischer Installer mit WiX Toolset oder Inno Setup gibt volle Kontrolle: Verknüpfungen, den Bootstrapper für die WebView2-Runtime, Registrierungswerte, einen Dienst. Entscheiden Sie danach, wie die Software verteilt wird – FieldOps liefert ein Inno-Setup-Paket aus, weil die Transporter-Rechner neu aufgesetzt werden und das Nachinstallieren der Runtime Teil der Installation sein muss.

Der Update-Kanal ist wenig Arbeit mit großer Wirkung. Beim Start, noch vor der Navigation, liest die Shell ihre eigene Assembly-Version und fragt einen Versions-Endpunkt nach der aktuellen. Ist die Antwort neuer, zeigt sie einen nicht blockierenden Hinweis mit den Änderungen, lädt den Installer herunter, prüft ihn, führt ihn aus und beendet sich. Sie unterbricht einen Techniker nie mitten im Auftrag, und eine unbeantwortete Prüfung ist eine protokollierte Warnung, kein fehlgeschlagener Start.

Es gibt jetzt zwei Versionsnummern. Der Server sollte sich schneller weiterentwickeln als die installierten Shells, daher veröffentlicht der Endpunkt auch die minimale Shell-Version, die er noch unterstützt – so kann der Server warnen oder den Zugriff verweigern, wenn ein Transporter zwei Monate offline war. Signierung und Verteilung gehören ins letzte Modul; hier muss der Kanal nur existieren und ehrlich sein.

- Die Evergreen Runtime wird vom Installer nachinstalliert, nicht vorausgesetzt.
- Jede Bridge-Anfrage trägt eine ID, und jede Antwort wird ihr zugeordnet.
- Der Servermodus ist im Fenster sichtbar und wird ins Log geschrieben.



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:

- ✓ die Wisej.NET-App in einer auf WebView2 basierenden Desktop-Shell mit nativem Fenster, Titelleiste und Infobereich hosten.
- ✓ Web-App und Shell so verbinden, dass die App native Funktionen wie Dateidialoge, Drucken und Benachrichtigungen aufrufen kann.
- ✓ entscheiden, ob der Server lokal neben der Shell oder remote läuft, und beide Varianten konfigurieren.
- ✓ einen Installer und einen Update-Kanal für das Desktop-Paket erstellen.

Lab- und Prüfungsleitfaden

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



Praxis-Lab

Paketieren Sie FieldOps für Windows: eine WebView2-Shell mit nativem Fenster, eigenem Titel und Tray-Icon, eine Bridge, über die die App einen nativen Ordnerdialog zum Export von Auftragsberichten öffnen und bei Eingang eines neuen Auftrags eine native Benachrichtigung anzeigen kann, eine Konfiguration, die die Shell auf den Remote-Server mit lokalem Fallback richtet, und ein Installer mit Update-Prüfung beim Start.

Ausgangspunkt

Sie öffnen die FieldOps-Projektmappe im Stand nach Modul 2: Die Liste der Arbeitsaufträge, die Detailseite und der Ablauf zur Statusaktualisierung laufen alle im Browser, und die App lässt sich mit Manifest und Service Worker bereits als PWA installieren. An diesem Webprojekt ändert sich in diesem Lab nichts – Sie fügen daneben ein zweites Projekt hinzu, eine ausführbare Windows-Shell.

Vorgeschlagenes Vorgehen

- ✓ Fügen Sie ein WinForms- oder WPF-Shell-Projekt mit einem einzigen WebView2-Control hinzu, das das Fenster ausfüllt, initialisieren Sie es asynchron und navigieren Sie es zur FieldOps-Adresse, bevor Sie irgendeinen Bridge-Code schreiben.
- ✓ Geben Sie dem Fenster seinen Rahmen: das FieldOps-Icon, einen Titel mit dem angemeldeten Techniker, gespeicherte Größe und Position, Aktivierung als Einzelinstanz und ein NotifyIcon mit Öffnen, Jetzt synchronisieren und Beenden, das den Prozess am Leben hält, wenn das Fenster geschlossen wird.
- ✓ Bauen Sie die Bridge in beide Richtungen – behandeln Sie WebMessageReceived in der Shell für eine pickFolder- und eine notify-Anfrage, antworten Sie mit PostWebMessageAsJson inklusive der Anfrage-ID, und erreichen Sie auf dem Server die Seite mit Application.Eval und empfangen Sie die Antwort als Client-Event.
- ✓ Schreiben Sie shell.config.json neben die ausführbare Datei, mit der Adresse des Remote-Servers, einem Timeout für die Prüfung und dem Schalter für den lokalen Fallback; starten Sie dann den mitgelieferten selbst gehosteten Server und zeigen Sie den Offline-Streifen, wenn die Health-Prüfung nicht antwortet.
- ✓ Erstellen Sie den Installer einschließlich des Bootstrappers für die WebView2-Runtime, und ergänzen Sie die Versionsprüfung beim Start, die die Assembly-Version der Shell mit dem Versions-Endpunkt vergleicht und den neuen Installer anbietet, ohne den Start zu blockieren.

Geforderte Ergebnisse

- ✓ WebView2-Desktop-Shell mit Fenster und Tray
- ✓ Native Bridge für Ordnerauswahl und Benachrichtigungen
- ✓ Konfiguration für Remote- und lokalen Server

- ✓ Installer-Paket
- ✓ Update-Prüfung beim Start

Wiederholungsfragen

- ✓ Ein Techniker startet zwei Exporte innerhalb einer Sekunde, und die Shell beantwortet beide Ordnerdialoge. Was in Ihrem Bridge-Design verhindert, dass der zweite Pfad in den ersten Bericht geschrieben wird, und was würde ohne diese Maßnahme schiefgehen?
- ✓ Ihr Export-Button öffnet jetzt einen nativen Ordnerdialog. Beschreiben Sie genau, was derselbe Button tut, wenn die Seite in Edge statt in der Shell geöffnet wird, und wo in Ihrem Code dieser Unterschied entschieden wird.
- ✓ Der Versions-Endpunkt ist nicht erreichbar, weil der Transporter keinen Empfang hat. Begründen Sie, was die Shell bei diesem Start tun sollte, und erklären Sie, was ein Techniker bei jeder der verworfenen Alternativen verliert.



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.