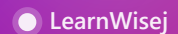


Empaquetado de escritorio con un shell WebView

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Alojar la aplicación en una ventana nativa de escritorio, WebView2 en Windows, el marco de la ventana, el sistema de archivos y la integración con el sistema operativo, y los instaladores. Qué significa esto para un equipo que lleva una aplicación Wisej.NET más allá de la pestaña del navegador, y cómo leer este módulo.



Enfoque de producción

Alojar la aplicación en una ventana nativa de escritorio, WebView2 en Windows, el marco de la ventana, el sistema de archivos y la integración con el sistema operativo, y los instaladores.

1. El problema en producción

Los técnicos de FieldOps pasan un turno entero dentro de la aplicación: un portátil en la furgoneta, una estación de acoplamiento en el depósito. La versión para navegador de los dos primeros módulos hace el trabajo, pero no tiene icono en el menú Inicio, ni presencia en el área de notificaciones cuando se asigna un trabajo, ni forma de dejar un informe exportado en una carpeta elegida por el técnico, ni nada que el departamento de TI pueda distribuir por su canal de software habitual. Este módulo envuelve esa misma aplicación, sin cambios, en un shell de Windows que aporta exactamente esas cuatro cosas.

2. Conceptos clave

Concepto	Por qué importa
control WebView2	El motor Chromium que Microsoft distribuye como Evergreen Runtime redistribuible, integrado en una ventana WinForms o WPF para que tu página se renderice exactamente como en Edge.
shell de escritorio	Un pequeño ejecutable nativo que controla la ventana, el icono de la bandeja y las llamadas al sistema operativo, y apunta un control WebView2 a tu aplicación Wisej.NET.
marco de la ventana	El título, el icono, el tamaño mínimo, la posición recordada y el comportamiento de instancia única que hacen que la aplicación parezca instalada y no incrustada.
punto de mensajes	El canal bidireccional entre la página y el shell: <code>PostWebMessageAsJson</code> hacia la página, <code>WebMessageReceived</code> hacia el shell, o un objeto host publicado con <code>AddHostObjectToScript</code> .
interoperabilidad con JavaScript	Cómo llega a ese canal el código C# del servidor: <code>Application.Eval</code> y <code>Application.Call</code> ejecutan script en el contexto del navegador de la sesión, y la respuesta vuelve como un evento de cliente.
topología del servidor	Si el shell navega a un servidor remoto de la empresa o inicia un proceso Wisej.NET autoalojado en loopback; se decide al iniciar y se registra, nunca se adivina.
canal de actualización	Un endpoint de versión que el shell consulta al iniciar, más el instalador que descarga cuando la respuesta es más reciente que la versión que está ejecutando.

3. Desarrollo de la lección

Una ventana nativa alrededor de la misma página

El shell es más pequeño de lo que esperan los desarrolladores: un ejecutable WinForms o WPF que contiene un único control `WebView2` acoplado para llenar el área de cliente, inicializado de forma asíncrona y después dirigido a la dirección de tu aplicación. También puedes partir del destino de escritorio de Wisej.NET Hybrid, que empaqueta esta misma disposición por ti. La página de dentro es la sesión normal de FieldOps: los mismos controles `Wisej.Web` del servidor, el mismo tema y la misma lógica de diseño. Nada en la aplicación sabe que está enmarcada.

Lo que añade el shell es todo lo que está fuera de la página. Pone como título de la ventana el nombre del técnico en cuanto la sesión indica quién ha iniciado sesión, da a la ventana un icono y un tamaño mínimo, recupera su última posición y se niega a abrirse dos veces: un segundo inicio activa la ventana que ya está en marcha. Un `NotifyIcon` coloca FieldOps en el área de notificaciones con un menú breve: Open, Sync now, Exit. Al cerrar, la ventana se oculta en la bandeja en lugar de terminar el proceso, así que la notificación emergente sigue llegando y la sesión se queda donde la dejó el técnico.

Hay dos cosas que no deben perderse. WebView2 requiere el Evergreen Runtime, así que el instalador lo detecta y lo instala con el bootstrapper en lugar de fallar en el primer inicio. Y cada capacidad que añade

el shell es un añadido, nunca un sustituto: la misma compilación debe seguir funcionando en una pestaña normal del navegador, que es para lo que sirve la comprobación de capacidades del laboratorio.

Un puente, dos direcciones

WebView2 te da el canal. Del shell a la página, `CoreWebView2.PostWebMessageAsJson` entrega un objeto JSON que llega como un evento message. De la página al shell, el script llama a `window.chrome.webview.postMessage` y el shell gestiona `WebMessageReceived` leyendo `WebMessageAsJson`. Una tercera opción, `AddHostObjectToScript`, publica un objeto visible para COM al que el script llama como a una API local: cómodo, y con una superficie mucho mayor, porque cualquier cosa cargada en esa página puede llamarlo.

Tu código C# vive en el servidor, así que llega a ese canal mediante la interoperabilidad con JavaScript: `Application.Eval` ejecuta un fragmento en el contexto del navegador de la sesión y `Application.Call` invoca una función que has registrado en el cliente. La respuesta del shell vuelve como un evento de cliente que dispara un controlador en el servidor, el cual actualiza los controles y deja que Wisej.NET los vuelva a pintar. Como el viaje de ida y vuelta es asíncrono, cada solicitud lleva un identificador; si no, dos exportaciones iniciadas seguidas se intercambiarán los resultados.

FieldOps usa ese puente dos veces. Export envía una solicitud `pickFolder`; el shell abre un cuadro de diálogo de carpetas nativo, devuelve la ruta elegida y el servidor escribe el informe en ella. La asignación funciona al revés: el servidor envía la nueva orden a la sesión, la página publica un mensaje `notify` y el shell genera desde el icono de la bandeja una notificación emergente que activa la ventana en esa orden. En un navegador normal ninguna de las dos solicitudes tiene quien la escuche, así que la comprobación de capacidades oculta la ruta nativa y la exportación recurre a `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 }));
};
```

Dónde se ejecuta realmente el servidor

El shell no es la aplicación, así que sigues teniendo que decir dónde está la aplicación. La topología remota apunta el control WebView2 al servidor de la empresa: un despliegue, una base de datos y una actualización que llega a cada técnico en cuanto se vuelve a conectar. La topología local incluye un proceso Wisej.NET autoalojado que el shell inicia en un puerto de loopback; funciona sin ninguna señal, pero entonces cada furgoneta lleva una copia de la aplicación y un almacén de datos que alguien tiene que reconciliar.

FieldOps configura ambas. Un pequeño `shell.config.json` junto al ejecutable indica el servidor principal, el tiempo de espera del sondeo y si se permite el servidor local de respaldo. Al iniciar, el shell llama a un endpoint de estado; si no responde en unos segundos, inicia el servidor incluido y muestra una franja persistente con el texto **Working offline — local server**. El modo es visible en la ventana y se escribe en el registro del shell, así que una llamada de soporte nunca empieza adivinando con qué servidor estaba hablando el técnico.

Cada modo tiene consecuencias. Las cookies del servidor remoto no significan nada para el local, así que el modo local inicia sesión contra su propio almacén y marca lo que captura para sincronizarlo más tarde. El puerto de loopback se elige en tiempo de ejecución (un puerto fijo es justo el que otro producto ya ha ocupado) y el shell detiene el proceso hijo al salir, también después de un fallo.

Instalarlo y mantenerlo al día

Hay dos vías de empaquetado razonables. MSIX ofrece una instalación por usuario, una desinstalación limpia, gestión de versiones por parte del propio Windows y despliegue mediante Intune, a cambio de la firma y de unas posibilidades limitadas durante la instalación. Un instalador clásico con WiX Toolset o Inno Setup da control total: accesos directos, el bootstrapper del runtime de WebView2, valores del registro, un servicio. Elige según cómo se distribuye el software: FieldOps publica un paquete Inno Setup porque las imágenes de los portátiles de las furgonetas se vuelven a crear y la instalación del runtime debe formar parte de la instalación.

El canal de actualización es poco trabajo con un gran beneficio. Al iniciar, antes de navegar, el shell lee la versión de su propio ensamblado y pide la versión actual a un endpoint de versión. Si la respuesta es más reciente, muestra un aviso no bloqueante con los cambios, descarga el instalador, lo verifica, lo ejecuta y se cierra. Nunca interrumpe a un técnico en mitad de un trabajo, y una comprobación sin respuesta es una advertencia registrada, no un inicio fallido.

Ahora existen dos números de versión. El servidor debería avanzar más rápido que los shells instalados, así que el endpoint publica también la versión mínima del shell que sigue admitiendo, lo que permite al servidor avisar (o rechazar) cuando una furgoneta lleva dos meses sin conexión. La firma y la distribución corresponden al último módulo; aquí el canal solo tiene que existir y ser honesto.

- El Evergreen Runtime lo instala el instalador con el bootstrapper, no se da por supuesto.
- Cada solicitud al puente lleva un id, y cada respuesta se asocia a él.
- El modo de servidor es visible en la ventana y se escribe en el registro.



Regla de empaquetado

La aplicación sigue siendo una aplicación web Wisej.NET servida por tu servidor; el shell es un medio de distribución. Mantén todas las funciones operativas en un navegador normal y añade después las capacidades del dispositivo como mejoras progresivas detrás de una comprobación de capacidades.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Alojar la aplicación Wisej.NET en un shell de escritorio basado en WebView2 con ventana nativa, barra de título y bandeja del sistema.
- ✓ Conectar la aplicación web y el shell para que la aplicación pueda llamar a funciones nativas como los cuadros de diálogo de archivos, la impresión y las notificaciones.
- ✓ Decidir si el servidor se ejecuta en local junto al shell o en remoto, y configurar cada opción.
- ✓ Construir un instalador y un canal de actualización para el paquete de escritorio.

Guía del laboratorio / examen

Qué construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Empaqueta FieldOps para Windows: un shell WebView2 con ventana nativa, título personalizado e icono en la bandeja del sistema, un puente que permita a la aplicación abrir un selector de carpetas nativo para exportar informes de órdenes de trabajo y mostrar una notificación nativa cuando llegue una orden nueva, una configuración que apunte el shell al servidor remoto con un servidor local de respaldo, y un instalador con comprobación de actualizaciones al iniciar.

Punto de partida

Abres la solución FieldOps tal como queda después del Módulo 2: la lista de órdenes de trabajo, la página de detalle de la orden de trabajo y el flujo de actualización del estado funcionan en un navegador, y la aplicación ya se instala como PWA con su manifiesto y su service worker. Nada de ese proyecto web cambia en este laboratorio: añades junto a él un segundo proyecto, un ejecutable de shell para Windows.

Enfoque sugerido

- ✓ Añade un proyecto de shell WinForms o WPF con un único control WebView2 que llene la ventana, inicialízalo de forma asíncrona y haz que navegue a la dirección de FieldOps antes de escribir código del puente.
- ✓ Dale a la ventana su marco: el icono de FieldOps, un título que muestre el técnico que ha iniciado sesión, un tamaño y una posición recordados, activación de instancia única y un NotifyIcon con Open, Sync now y Exit que mantenga vivo el proceso cuando se cierra la ventana.
- ✓ Construye el puente en ambas direcciones: gestiona WebMessageReceived en el shell para una solicitud pickFolder y una notify, responde con PostWebMessageAsJson devolviendo el id de la solicitud, y en el servidor llega a la página con Application.Eval y recibe la respuesta como un evento de cliente.
- ✓ Escribe shell.config.json junto al ejecutable con la dirección del servidor remoto, un tiempo de espera del sondeo y el interruptor del servidor local de respaldo; después inicia el servidor autoalojado incluido y muestra la franja de modo sin conexión cuando el sondeo de estado no responda.
- ✓ Genera el instalador con el bootstrapper del runtime de WebView2 incluido y añade la comprobación de versión al iniciar, que compara la versión del ensamblado del shell con el endpoint de versión y ofrece el nuevo instalador sin bloquear el arranque.

Entregables requeridos

- ✓ Shell de escritorio WebView2 con ventana e icono en la bandeja
- ✓ Puente nativo para el selector de carpetas y las notificaciones
- ✓ Configuración de servidor remoto y local
- ✓ Paquete de instalación

- ✓ Comprobación de actualizaciones al iniciar

Preguntas de repaso

- ✓ Un técnico inicia dos exportaciones con menos de un segundo de diferencia y el shell responde a los dos cuadros de diálogo de carpetas. ¿Qué elemento del diseño de tu puente impide que la segunda ruta se escriba en el primer informe, y qué fallaría sin él?
- ✓ Tu botón Export abre ahora un cuadro de diálogo de carpetas nativo. Describe con precisión qué hace ese mismo botón cuando la página se abre en Edge en lugar de en el shell, y en qué parte de tu código se decide esa diferencia.
- ✓ El endpoint de versión no está disponible porque la furgoneta no tiene señal. Argumenta qué debería hacer el shell en ese inicio y explica qué pierde un técnico con cada una de las alternativas que has descartado.



Regla de empaquetado

La aplicación sigue siendo una aplicación web Wisej.NET servida por tu servidor; el shell es un medio de distribución. Mantén todas las funciones operativas en un navegador normal y añade después las capacidades del dispositivo como mejoras progresivas detrás de una comprobación de capacidades.