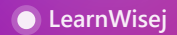


Modelos de alojamiento más allá del navegador

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Navegador, progressive web app, shell de escritorio y shell móvil comparados: arquitectura, capacidades, modelo de actualización y coste. 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

Navegador, progressive web app, shell de escritorio y shell móvil comparados: arquitectura, capacidades, modelo de actualización y coste.

1. El problema en producción

FieldOps es una aplicación de servicio técnico de campo: los técnicos reciben órdenes de trabajo, capturan fotos y firmas, se desplazan hasta los emplazamientos y trabajan con una conectividad irregular.

Construida como una aplicación Wisej.NET normal ya funciona en cualquier navegador, lo que basta para los operadores de la oficina y no basta para un técnico que está en una sala de máquinas con una sola barra de cobertura y el teléfono en una mano con guante. La pregunta que responde este módulo no es «¿podemos construirla?», sino «¿cómo llega a las personas que la usan?», y esa respuesta es una decisión de arquitectura, no un detalle de empaquetado para el final.

2. Conceptos clave

Concepto	Por qué importa
modelo de distribución	La vía por la que la aplicación llega a un usuario: una pestaña del navegador, una PWA instalada, una ventana de escritorio o una aplicación móvil distribuida por una tienda. Cambia la distribución, las actualizaciones y el acceso al dispositivo, nunca el código C# que construye la pantalla.
motor de navegador anfitrión	Todos los modelos renderizan la misma interfaz controlada por el servidor en un motor de navegador: el navegador del propio usuario o uno integrado (WebView2 en Windows, WebKit en macOS e iOS, el WebView del sistema en Android).
Wisej.NET Hybrid	El shell del framework basado en .NET MAUI que aloja tu aplicación Wisej.NET existente en Android, iOS y escritorio, y añade un puente hacia funciones del dispositivo como la cámara, la geolocalización, los archivos y las notificaciones.
puente con el dispositivo	El canal bidireccional entre tu código C# del servidor y el shell nativo: el servidor pide una acción del dispositivo, el shell la realiza y el resultado vuelve a tu código como un evento, no como un valor de retorno.
comprobación de capacidades	Una prueba en tiempo de ejecución de lo que el shell actual puede hacer realmente, que se usa para mostrar, ocultar o sustituir una función para que una misma pantalla se comporte correctamente en todos los modelos de distribución.
caché del shell de la aplicación	El HTML, el CSS, los scripts y los iconos estáticos que el service worker de una PWA puede conservar localmente. Es lo que sigue cargándose sin red; la parte interactiva de la aplicación, renderizada por el servidor, no está en ella.
asimetría de actualización	El servidor puede volver a desplegarse en cualquier momento y todas las sesiones reciben la nueva compilación al recargar, mientras que un instalador de escritorio o un paquete de tienda necesitan antes una publicación y una instalación.

3. Desarrollo de la lección

Una aplicación, cuatro puertas de entrada

Una aplicación Wisej.NET es un proceso del servidor. Tus objetos `Window` y `Page`, sus controles y sus controladores de eventos viven en la memoria del servidor, y el navegador contiene un reflejo renderizado que devuelve las acciones del usuario a través de una conexión HTTP y WebSocket. Nada en esa disposición depende de qué motor de navegador haya al otro lado. Ese único hecho es lo que hace posibles cuatro modelos de distribución a partir de un solo despliegue: la misma URL, la misma sesión, el mismo estado de `Application.Session`.

En una pestaña del navegador, el usuario escribe la dirección o hace clic en un enlace y obtiene la aplicación. Como progressive web app, la misma URL se instala desde el navegador: un manifiesto le da un nombre, un icono y una ventana standalone sin barra de direcciones, y un service worker almacena en caché el shell estático para que el inicio parezca inmediato. En un shell de escritorio, un pequeño programa nativo abre una ventana que contiene un motor de navegador integrado (WebView2 en

Windows, WebKit en macOS) y lo apunta a tu servidor; el usuario ve una entrada en la barra de tareas y una barra de título, y la aplicación que hay dentro es la misma que usan los operadores. En un shell móvil, Wisej.NET Hybrid aloja la aplicación dentro de una app .NET MAUI para Android e iOS, que se instala desde una tienda o un canal empresarial y puede acceder al dispositivo.

Para FieldOps eso significa que la lista de órdenes de trabajo se escribe una sola vez y nunca se adapta ni se bifurca. Los shells son contenedores: lo que los diferencia es lo que rodea a la aplicación, no lo que se ejecuta dentro.

La comparación que realmente decide

Cinco columnas deciden el modelo, y la primera es la honesta. **Funcionamiento sin conexión**: una interfaz del lado del servidor necesita su servidor. Cuando se cae la conexión, ningún shell puede seguir ejecutando tus controladores de eventos, porque los controladores no están en el dispositivo. Una PWA o un shell nativo pueden almacenar en caché el shell y mostrar una pantalla sin conexión diseñada con el último resumen sincronizado de las órdenes de trabajo, y pueden poner en cola una acción del técnico para enviarla cuando vuelva la señal, pero no pueden abrir una orden de trabajo que el dispositivo nunca ha visto. Prometer más que eso es la forma en que las aplicaciones de campo pierden la confianza de sus usuarios en la primera semana.

El **acceso al dispositivo** es donde los shells se separan. Un navegador y una PWA obtienen lo que concede la plataforma web tras una solicitud de permiso: selección de archivos, una imagen mediante un control de subida, geolocalización aproximada y notificaciones web donde estén disponibles. Un shell nativo añade el puente: una sesión de cámara real, ubicación en segundo plano, el sistema de archivos, notificaciones push y biometría. La **distribución** va desde una URL que envías por correo, pasando por un instalador, hasta la revisión de una tienda que puede tardar días. Las **actualizaciones** siguen el mismo orden: el servidor es inmediato, un paquete de escritorio necesita un canal de actualización y una compilación para tienda necesita un envío. El **coste** es la suma de certificados, cuentas de desarrollador, dispositivos de prueba y los ciclos de revisión de los que ahora eres responsable para siempre.

- Operadores en un escritorio: navegador, y nada más.
- Técnicos que necesitan cámara, ubicación y notificaciones: shell móvil.
- Quioscos de almacén que deben sobrevivir a una caída de una wifi inestable: PWA con una página sin conexión honesta.

Un servidor, una base de código, muchos shells

La arquitectura que mantiene esto barato es deliberadamente aburrida: un único despliegue del servidor Wisej.NET, una solución y shells que no contienen ninguna lógica de negocio. La lista de órdenes de trabajo, la página de detalle y el flujo de actualización del estado son pantallas Wisej.NET normales que llaman a un `WorkOrderService`. Los proyectos de shell que se añaden en módulos posteriores hacen exactamente dos cosas: alojar el motor del navegador apuntado a tu servidor y exponerle las funciones del dispositivo.

La unión entre ambos es la comprobación de capacidades. En lugar de comprobar «¿estoy en Android?», comprueba lo que necesitas: ¿puede este shell hacer una foto, leer una ubicación, generar una notificación? Resuelve esas capacidades una vez al iniciar la sesión, guárdalas en el estado de la sesión y

deja que cada pantalla las lea. Así una pantalla se adapta en lugar de ramificarse por plataforma, y un shell nuevo añadido el año que viene no obliga a modificar las pantallas.

```
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;
```

Fíjate en la alternativa: no es un botón desactivado, sino un control **Upload** que sigue permitiendo añadir una foto a la orden de trabajo. Cada capacidad necesita esa respuesta antes de diseñar la función, y por eso la comprobación es un entregable del Módulo 1 y las funciones del dispositivo no lo son.

Redactar el registro de decisión de distribución

El último entregable es un documento breve, y es el que tu equipo volverá a leer de verdad. Nombra los destinos, el orden en que los construyes y los criterios de salida que indican cuándo un destino está terminado. El registro de FieldOps podría decir: primero el navegador, porque los operadores lo necesitan y demuestra que la aplicación funciona; después el shell de Android, porque es lo que llevan los técnicos; luego la PWA para los quioscos del almacén; iOS solo si se aprueban las tabletas de los supervisores.

Los criterios de salida impiden que un destino se desvíe. «El shell de Android está terminado cuando un técnico completa una orden de trabajo de principio a fin en un dispositivo proporcionado, la sesión sobrevive al bloqueo de pantalla y una compilación firmada se instala a través del canal empresarial» es comprobable; «soporte móvil» no lo es. Redacta el registro antes de que exista cualquier proyecto de shell, porque en cuanto se construye el primer instalador, toda discusión sobre el alcance se convierte en una discusión sobre el esfuerzo ya invertido.



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:

- ✓ Explicar cómo una aplicación Wisej.NET se ejecuta sin cambios en un navegador, una PWA, un shell WebView de escritorio y un shell móvil.
- ✓ Comparar los cuatro modelos de distribución en funcionamiento sin conexión, acceso al dispositivo, distribución, actualizaciones y coste.
- ✓ Diseñar la arquitectura de FieldOps para que un servidor y una base de código sirvan a todos los shells.
- ✓ Redactar un registro de decisión de distribución que nombre los destinos, el orden en que se construyen y los criterios de salida.

Guía del laboratorio / examen

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



Laboratorio práctico

Crea la solución FieldOps con una lista de órdenes de trabajo, una página de detalle de la orden de trabajo y un flujo de actualización del estado que funcionen en un navegador normal. Redacta el registro de decisión de distribución: compara el navegador, la PWA, el shell de escritorio y el shell móvil según las necesidades de los técnicos, elige los destinos y su orden, y define las comprobaciones de capacidades que la aplicación usará para activar las funciones del dispositivo.

Punto de partida

Partes de una Wisej.NET Web Application nueva y vacía llamada FieldOps, que solo contiene la ventana de inicio generada y los archivos de configuración predeterminados. Todo lo que hay en este laboratorio lo creas tú (el modelo y el servicio de órdenes de trabajo, las tres pantallas, la abstracción de capacidades y el registro de decisión escrito), y cada shell que se añade en los módulos 2 a 6 envuelve sin cambiarlo el servidor que terminas aquí.

Enfoque sugerido

- ✓ Modela primero el dominio: un WorkOrder con número, cliente, dirección del emplazamiento, prioridad, estado y técnico asignado, detrás de un WorkOrderService que devuelva las órdenes abiertas del técnico y aplique un cambio de estado.
- ✓ Construye la lista de órdenes de trabajo como una cuadrícula enlazada al servicio, con un filtro por el técnico que ha iniciado sesión, y hazla utilizable con un ancho de ventana estrecho para que la misma pantalla resista más adelante una vista del tamaño de un teléfono.
- ✓ Construye la página de detalle de una orden de trabajo y un flujo de actualización del estado (Accepted, On site, Completed) que valide la transición en el servicio, la escriba una sola vez y actualice la lista.
- ✓ Añade un objeto ShellCapabilities que se resuelva al iniciar la sesión y se guarde en Application.Session, con indicadores de cámara, ubicación y notificaciones, todos a false en un navegador normal, y haz que la página de detalle lo lea para decidir qué muestra.
- ✓ Redacta la tabla comparativa y el registro de decisión de distribución, y después dibuja el diagrama de arquitectura con un servidor, una base de código y los shells que apuntarán a él.

Entregables requeridos

- ✓ Solución FieldOps con lista, detalle y actualización del estado
- ✓ Tabla comparativa de los modelos de distribución
- ✓ Registro de decisión de distribución con el orden de los destinos
- ✓ Diseño de la comprobación de capacidades
- ✓ Diagrama de arquitectura con un servidor y varios shells

Preguntas de repaso

- ✓ Un técnico pierde la señal a mitad de la actualización de una orden de trabajo. Describe exactamente qué partes de FieldOps se pueden seguir mostrando desde un shell en caché, cuáles no, y qué debería hacer tu pantalla en el momento en que se cae la conexión.
- ✓ Tu registro de decisión coloca el shell de Android antes que la PWA. ¿Qué tendría que ser cierto sobre los dispositivos y la red de los técnicos para que ese orden fuera incorrecto?
- ✓ La página de detalle necesita una foto. Explica por qué una comprobación de capacidades con una alternativa Upload es un diseño mejor que un interruptor en tiempo de compilación que genere una versión de la pantalla solo para móvil.



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.