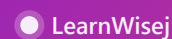


Modelo de objetos de JavaScript, API del navegador y contratos de interoperabilidad seguros

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Interoperabilidad JS avanzada, eventos de cliente, capacidades del navegador, validación de contratos. Lo que esto significa para un equipo avanzado de Wisej.NET — y cómo leer este módulo.



Enfoque de producción

Interoperabilidad avanzada con JS, eventos del cliente, capacidades del navegador, validación de contratos.

1. El problema en producción

Wisej.NET permite a los desarrolladores añadir JavaScript allí donde aporta valor, pero la interoperabilidad avanzada puede volverse frágil a menos que cada límite se trate como una pequeña API.

2. Conceptos clave

Concepto	Por qué importa
modelo de objetos App	La representación del lado del cliente de los componentes de Wisej.NET a la que puede acceder JavaScript. Úsala de forma deliberada, a través de puntos de entrada documentados, sin hurgar en los detalles internos.
evento del cliente	Un evento del lado del navegador que se reenvía al servidor. Solo los eventos significativos deberían viajar por la red, con cargas útiles compactas.
método remoto	Un método de servidor al que el navegador puede llamar directamente, como un WebMethod, que devuelve un resultado a JavaScript. Necesita la misma validación que cualquier endpoint.
detección de funciones del navegador	Comprobar qué admite el navegador antes de usar una API. Úsala solo para aportar valor en el navegador, mientras las reglas de negocio permanecen en el servidor.
contrato de interoperabilidad	Una descripción documentada y con nombre de lo que JavaScript envía al servidor y de lo que recibe de vuelta, para que ambos lados se puedan probar y versionar.
momento del ciclo de vida	Saber cuándo se crea, se renderiza, se actualiza y se destruye un widget de cliente, para que los scripts se ejecuten en el momento adecuado y limpien lo que dejan.
validación de entradas	Tratar cada valor que llega desde el navegador como no confiable y validarlo en el servidor, porque cualquiera puede modificar el código del cliente.

3. Desarrollo de la lección

JavaScript es una herramienta, no la arquitectura

Las aplicaciones avanzadas de Wisej.NET pueden usar las API del navegador, la gestión del teclado, el portapapeles, la cámara, los escáneres y widgets especializados. El servidor sigue siendo la autoridad sobre el flujo de trabajo, los permisos y la persistencia.

Pon nombre al contrato

Cada método de interoperabilidad debería tener un nombre, un propósito, un esquema de entrada, un esquema de salida, una regla de validación y un comportamiento ante fallos. Trátalo como una pequeña API pública entre el navegador y el servidor.

Capacidades del navegador sin reglas de negocio en el navegador

JavaScript se gana su sitio en una aplicación empresarial de Wisej.NET cuando el navegador puede hacer algo que el servidor no puede: reaccionar a una pulsación de tecla antes de un viaje de ida y vuelta, leer el portapapeles, comunicarse con un escáner o informar del tamaño de la pantalla y del estado de la red. La paleta de comandos del laboratorio es un buen ejemplo. Capturar Ctrl+K, filtrar una lista mientras el usuario escribe y resaltar una coincidencia son asuntos puramente del navegador, así que pertenecen a un pequeño script de cliente.

Decidir qué hace un comando es otra cuestión. Cuando el usuario pulsa Enter sobre `approve`, el script envía una solicitud con nombre al servidor, y el servidor decide si el usuario actual, en el tenant actual, puede aprobar esa orden de trabajo. La comprobación de permisos, la transición de estado, la entrada de auditoría y la persistencia se ejecutan en un servicio del servidor. Si se permitiera al script cambiar directamente un campo de estado, cualquier regla de ese servicio se podría eludir desde las herramientas de desarrollo.

Una prueba útil en la revisión de código es preguntarse qué se rompería si el navegador mintiera. Si la respuesta es solo un fallo estético, el código está en el lugar correcto. Si la respuesta es una aprobación indebida o un registro filtrado, esa lógica debe pasar al servidor.

Contratos pequeños con entradas y salidas con nombre

Un contrato de interoperabilidad es un acuerdo escrito sobre un único cruce de límite. El laboratorio lo recoge en un documento y en el código como un tipo de solicitud y un tipo de resultado, por ejemplo un `ClientCommandRequest` que transporta el nombre de un comando, un ID de destino y un argumento opcional, y un método de servidor como `RunClientCommand` que devuelve un resultado tipado. Cada campo tiene un tipo, un rango permitido y un valor predeterminado. No se acepta nada más.

Mantener el contrato pequeño es lo que lo hace seguro. Una carga útil con tres campos con nombre se puede validar en pocas líneas, registrar con su ID de correlación y versionar cuando se añade un campo. Una carga útil que reenvía un objeto JavaScript arbitrario no se puede validar en absoluto, porque el servidor no sabe qué está mirando. La validación se ejecuta antes de llamar a cualquier servicio, y un fallo devuelve un error con nombre como `PERMISSION_DENIED` o `INVALID_TARGET` en lugar de un mensaje de excepción.

- Nombre: para qué sirve la llamada, en una frase.
- Esquema de entrada: cada campo, su tipo y su regla de validación.
- Esquema de salida: la forma del resultado correcto y cada fallo con nombre.
- Propietario: qué servicio del servidor aplica las reglas.

Eventos del cliente y callbacks del servidor

El tráfico entre el navegador y el servidor fluye en dos direcciones. Un evento del cliente lleva al servidor algo que ha ocurrido en el navegador, como una selección en la paleta o un cambio detectado en las capacidades. Una callback del servidor es el servidor llamando a un widget de cliente para actualizarlo, darle el foco o leer un valor. Ambas direcciones cruzan el mismo límite de confianza, así que ambas necesitan la misma disciplina.

En los eventos del cliente, reenvía solo los eventos que tienen un significado de negocio, con una carga útil compacta. Una paleta que lanzara un evento por cada pulsación de tecla inundaría la sesión de viajes de ida y vuelta y no daría al servidor nada útil sobre lo que actuar. En las callbacks del servidor, cuenta con que el widget puede no existir, estar liberado o no haberse inicializado todavía, y haz que la callback se pueda ejecutar dos veces sin problemas. Una callback que añade una fila al DOM debería comprobar si la fila ya existe.

Cada callback y cada manejador de eventos deberían ser atribuibles. Registra el usuario, el tenant, el ID de correlación y el nombre del comando en el lado del servidor de cada cruce, para que un ingeniero de

soporte pueda reconstruir qué pidió el navegador y qué respondió el servidor.

Momento, ciclo de vida y diferencias de capacidades

El fallo de interoperabilidad más habitual es un script que se ejecuta antes de que exista su destino. El laboratorio conecta el script de la paleta para que se enganche solo después de que se haya creado y renderizado el widget anfitrión, no al cargar la página. La misma regla se aplica a cualquier código que lea un elemento del DOM, se suscriba a un evento de un widget o llame a una API de un proveedor. Si el elemento se crea más tarde, el script debe esperar al evento de creación o ser invocado desde el propio widget.

El ciclo de vida tiene un final además de un principio. Cuando se cierra un formulario o se libera un widget, los manejadores enganchados a la ventana o al documento siguen ejecutándose a menos que se eliminen. A lo largo de una sesión larga esos restos se acumulan, mantienen referencias a widgets muertos y producen errores que parecen no tener relación con la pantalla que los causó. Cada enganche necesita su desenganche correspondiente.

Los navegadores también difieren. El panel de capacidades del laboratorio existe precisamente porque el acceso al portapapeles, el acceso a la cámara y el permiso de notificaciones varían según el navegador, el dispositivo y el contexto de seguridad. Detecta la función, informa del resultado al servidor como datos y deja que el servidor elija una alternativa, como una carga de archivo en lugar de una captura de cámara. Nunca supongas una capacidad a partir de la cadena del agente de usuario.



Regla avanzada de producción

No te detengas cuando la pantalla parezca correcta. Explica a quién pertenece el estado, las implicaciones de seguridad, las rutas de fallo y el comportamiento en producción antes de dar el módulo por terminado.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Usar JavaScript para las capacidades del navegador sin trasladar reglas de negocio al cliente.
- ✓ Definir contratos de interoperabilidad pequeños con entradas, salidas y validación con nombre.
- ✓ Usar eventos del cliente y callbacks del servidor de forma segura.
- ✓ Gestionar el momento de ejecución, el ciclo de vida de los widgets y las diferencias de capacidades entre navegadores.

Guía del laboratorio / examen

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



Laboratorio práctico

Añade una paleta de comandos de teclado avanzada y un panel de capacidades del navegador. Usa JavaScript para recopilar las funciones del navegador y lanzar comandos en el servidor, y después aplica los permisos en el servidor antes de ejecutar el comando.

Enfoque sugerido

- ✓ Lee el problema en producción e identifica el riesgo de arquitectura antes de tocar la UI.
- ✓ Crea o actualiza primero el límite correspondiente de servicio, comando, modelo o componente.
- ✓ Construye la pantalla o el componente Wisej.NET siguiendo los estándares de proyecto establecidos.
- ✓ Añade una demostración de la ruta de fallo, no solo del caso ideal.
- ✓ Anota lo que ha cambiado en el diario de aprendizaje y actualiza la lista de comprobación correspondiente.

Entregables requeridos

- ✓ Documento del contrato de interoperabilidad
- ✓ Script de la paleta de comandos
- ✓ Métodos de callback del servidor
- ✓ Panel de capacidades del navegador
- ✓ Notas de la revisión de seguridad

Preguntas de repaso

- ✓ ¿Qué ocurre antes de que exista el widget de destino?
- ✓ ¿En qué valores procedentes de JavaScript se confía?
- ✓ ¿Qué reglas de negocio permanecen en el servidor?



Regla avanzada de producción

No te detengas cuando la pantalla parezca correcta. Explica a quién pertenece el estado, las implicaciones de seguridad, las rutas de fallo y el comportamiento en producción antes de dar el módulo por terminado.