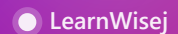


JavaScript esencial para integraciones fiables

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Domina los patrones de JavaScript que aparecen una y otra vez en las integraciones de terceros. Qué significa esto para un desarrollador de integraciones con Wisej.NET — y cómo leer este módulo.



Enfoque de producción

Domina los patrones de JavaScript que aparecen una y otra vez en las integraciones de terceros.

1. El problema de integración

La mayoría de los fallos de integración son fallos de JavaScript sin más: contexto perdido, elemento contenedor equivocado, orden de carga de scripts, sincronización asíncrona o una llamada a la API del proveedor antes de que exista el widget.

2. Conceptos clave

Concepto	Cómo usarlo
Contexto	Conserva el objeto correcto en los callbacks y en el código asíncrono.
Orden de carga	Confirma que los scripts, el CSS y las bibliotecas del proveedor se cargan antes de usarlos.

3. Lección de producción

Un desarrollador de integraciones con Wisej.NET necesita suficiente soltura con JavaScript para leer los ejemplos del proveedor y traducirlos a un adaptador. Eso no exige convertir a un desarrollador de C# en especialista en frameworks de frontend. Sí exige tratar con cuidado el contexto, los callbacks, los elementos del DOM y la ejecución asíncrona. El error más habitual es suponer que `this` dentro de un callback del proveedor sigue refiriéndose al widget de Wisej.NET. Las bibliotecas de terceros rara vez conservan el contexto de tu objeto. Guarda la referencia en una closure, usa `bind` o encamina el callback a través de un método que tenga el contexto correcto. Los widgets de Wisej.NET conservan el contexto en sus propios listeners, pero las bibliotecas de proveedores quedan fuera de ese contrato. El segundo error habitual es usar el elemento equivocado. Algunos widgets quieren un `div` genérico. Otros quieren un `input`, un `textarea`, un `canvas`, una raíz SVG o un elemento concreto identificado por su ID. El `this.container` de Wisej.NET es el lugar seguro para alojar el objeto del proveedor.`

Conservar el this correcto

Dentro de las funciones de un `InitScript`, `this` es el widget de cliente de Wisej.NET, así que this.container` y this.fireWidgetEvent` están disponibles. Dentro de un callback del proveedor,`

`this` es lo que haya decidido el proveedor, normalmente su propia instancia o el elemento DOM que lanzó el evento. El código escrito como si el widget siguiera en el ámbito compila, se ejecuta y falla en el primer callback con un error de método no definido.

```
init: function (options) {  
    var me = this;  
    me.gauge = new VendorGauge(me.container, options);  
    me.gauge.on('change', function (v) {  
        me.fireWidgetEvent('valueChanged', { value: v });  
    });  
}
```

Capturar el widget en una variable local antes del callback es la solución más sencilla y la más fácil de leer en una revisión. Llamar a `bind(this)` sobre el callback consigue lo mismo cuando la función se declara en otro lugar. En ambos casos, la instancia del proveedor se guarda en el widget, aquí como `me.gauge`, para que `update` y la eliminación puedan encontrarla después. El código asíncrono sigue la misma regla: un temporizador, una promesa o un callback de fetch se ejecutan después de que la llamada original haya terminado, y solo una referencia capturada sobrevive hasta ese momento.

El elemento correcto, y solo ese elemento

Cada biblioteca de proveedor documenta qué espera recibir. Algunas aceptan cualquier elemento de bloque y crean sus propios hijos. Otras requieren un `input` para poder mejorarlo, un `canvas` para poder dibujar, una raíz `svg` o un elemento con un ID concreto al que apunta su CSS. Pasarle un `div` a una biblioteca de gráficos que esperaba un `canvas` produce un recuadro vacío y ningún error, y por eso el laboratorio empieza con una prueba en HTML puro antes de que exista código de Wisej.NET.

El alojamiento seguro es siempre `this.container`, el elemento que el widget de Wisej.NET reserva para el contenido. Si el proveedor necesita un elemento hijo concreto, créalo dentro del contenedor, por ejemplo asignando al HTML interno del contenedor un `canvas` o un `input` vacío y pasando después ese hijo al proveedor. Lo que nunca debe ocurrir es modificar el elemento raíz del propio widget, cambiar sus clases o sustituirlo, porque el framework depende de ese elemento para el diseño, la visibilidad y la eliminación.

Una prueba rápida de revisión: busca en el InitScript cualquier elemento distinto de `this.container` que se pase al proveedor. Cualquier otra referencia a un elemento necesita una justificación por escrito.

Orden de dependencias de scripts y estilos

Los scripts del navegador se ejecutan en el orden en que se cargan, y un plugin que extiende jQuery lanzará un error si jQuery todavía no es global cuando se ejecuta el plugin. Lo mismo ocurre con una biblioteca de gráficos que espera su núcleo antes que sus módulos de extensión, y con un tema CSS que debe cargarse después de la hoja de estilos base de la biblioteca para que sus reglas prevalezcan. En una página HTML independiente, el desarrollador controla este orden con la secuencia de etiquetas script. En Wisej.NET, la lista Packages del widget cumple ese mismo papel, y las entradas se cargan en el orden en que aparecen.

El orden del laboratorio es el habitual: primero jQuery, después la hoja de estilos del proveedor, luego el script del proveedor y, solo entonces, el InitScript que los usa. Un orden incorrecto falla de forma confusa. El error aparece en la consola durante la carga, a menudo antes de que se ejecute código del widget, y menciona un símbolo no definido de la biblioteca del proveedor en lugar de algo que haya escrito el desarrollador.

- Enumera todos los archivos que incluye el ejemplo del proveedor, en el orden en que los incluye.
- Comprueba si la aplicación ya carga jQuery u otra biblioteca compartida, para no cargar dos copias.
- Fija las versiones en las rutas de los paquetes para que una actualización sea un cambio deliberado.

Depurar scripts inyectados

Un InitScript se inyecta en la página en tiempo de ejecución, así que no aparece como archivo en el panel de código fuente del navegador y no se puede poner un punto de interrupción por su nombre. Añadir un comentario `sourceURL` al final del script le da un nombre, y las herramientas de desarrollo lo muestran entonces como si fuera un archivo real. Una sentencia `debugger` dentro de `init` detiene la ejecución cuando las herramientas están abiertas, que es la forma más rápida de inspeccionar `this`, el objeto de opciones y el contenedor en el momento en que se crea el objeto del proveedor.

La consola es la segunda herramienta. Registrar la instancia del proveedor tras crearla confirma que existe y muestra qué métodos expone. Inspeccionar el widget en el panel de elementos confirma que los hijos del proveedor se crearon dentro del contenedor y no en otro sitio. El panel de red confirma que cada paquete se cargó con estado 200 y en el orden esperado.

Las notas de depuración del laboratorio registran dónde se guarda la instancia del proveedor y cómo se verificó. Esas notas se convierten en la primera página de la guía de resolución de problemas cuando el mismo widget se comporta mal en producción.



Regla de producción

Mantén claras la propiedad del estado en Wisej.NET y el comportamiento del widget de cliente. Documenta el contrato cliente/servidor antes de confiar en la integración en producción.



Error habitual que debes evitar

Copiar JavaScript de demostración en las pantallas sin documentar el ciclo de vida, los recursos, los contratos de eventos, la limpieza ni los límites de seguridad.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Usar closures y bind para conservar el contexto en callbacks asíncronos.
- ✓ Identificar el elemento DOM correcto para un widget y evitar modificar indebidamente la raíz del widget de Wisej.NET.

- ✓ Entender el orden de dependencias de scripts y CSS, sobre todo en bibliotecas basadas en jQuery.
- ✓ Usar con eficacia las herramientas de desarrollo del navegador, sourceURL, la inspección en consola y las sentencias debugger.

Guía del laboratorio / examen

Lo que construirás en el laboratorio práctico, el flujo de trabajo recomendado, los entregables requeridos y la autoevaluación antes de entregar.



Laboratorio práctico

Toma un ejemplo en JavaScript puro de un widget de indicador o de mando giratorio (knob) y haz que funcione primero en una página HTML aislada. Después reescríbelo como InitScript de un Widget de Wisej.NET.

Flujo de trabajo recomendado

- ✓ Abre el proyecto existente del curso de integración de Wisej.NET y ejecútalo en local.
- ✓ Identifica el componente de servidor, el widget de cliente, el archivo de recursos o el endpoint que intervienen en este módulo.
- ✓ Haz un solo cambio pequeño cada vez y pruébalo en el navegador antes de continuar.
- ✓ Mantén el JavaScript específico del proveedor dentro del adaptador o wrapper en lugar de repartirlo por las páginas.
- ✓ Actualiza la interfaz para que el estudiante vea con claridad el éxito, los errores, el progreso o los cambios de comunicación.
- ✓ Toma capturas de pantalla o notas que demuestren que la integración funciona y expliquen la decisión de diseño.

Entregables requeridos

- ✓ Prueba funcional en JavaScript puro
- ✓ InitScript de Wisej.NET seguro respecto al contexto
- ✓ Notas de depuración que muestran dónde se guarda la instancia del widget

Autoevaluación antes de entregar

- ✓ ¿Contra qué te protege ``var me = this;``?
- ✓ ¿Cuándo deberías usar ``this.container.innerHTML``?
- ✓ ¿Por qué es útil una prueba en JS puro antes de la integración con Wisej.NET?