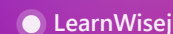


Integración rápida con Wisej.Web.Widget

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Usa Wisej.Web.Widget para integraciones puntuales, prototipos y la validación rápida de bibliotecas de proveedores. Qué significa esto para un desarrollador de integraciones con Wisej.NET — y cómo leer este módulo.



Enfoque de producción

Usa Wisej.Web.Widget para integraciones puntuales, prototipos y la validación rápida de bibliotecas de proveedores.

1. El problema de integración

Un equipo necesita demostrar que un widget de JavaScript puede alojarse en Wisej.NET antes de invertir en un wrapper de control reutilizable.

2. Conceptos clave

Concepto	Cómo usarlo
Packages	Registra los recursos JavaScript y CSS que necesita un widget puntual.
InitScript	Crea el widget del proveedor usando las opciones renderizadas desde .NET.

3. Lección de producción

`Wisej.Web.Widget` es la superficie de integración más rápida. Colocas el widget en una página, adjuntas scripts y estilos mediante Packages y escribes un InitScript que crea o actualiza el objeto de terceros. Este enfoque es ideal cuando el widget se usa una sola vez, cuando todavía se está explorando la API o cuando hay que validar un prototipo antes de construir un wrapper completo. El InitScript debe crear el objeto del proveedor a partir del elemento correcto y guardar la instancia en un lugar predecible, normalmente en `this`. La función update es opcional, pero un prototipo con calidad de producción debería incluirla siempre que el lado .NET pueda cambiar las opciones después del primer renderizado. Algunos widgets de proveedores exponen un método `setOptions` u `option` limpio. Otros requieren destruirlos y volver a crearlos. El código de integración debe dejar ese comportamiento explícito. El objeto Options es cómodo y flexible, pero también es fácil usarlo mal. En los primeros prototipos, las opciones con forma de JSON son aceptables porque ponen un widget en pantalla rápidamente. En cuanto el wrapper se reutiliza, traslada los ajustes importantes a propiedades .NET con nombre para que puedan validarse, versionarse y entenderse sin leer la documentación del proveedor.

Packages para un único widget

Un `Wisej.Web.Widget` declara sus dependencias mediante su colección `Packages`. Cada paquete indica un origen, que puede ser una URL, una ruta de la aplicación o un recurso incrustado, y Wisej.NET lo carga una vez por sesión del navegador, en el orden de la lista, antes de que se ejecute el `InitScript`. Tanto los scripts como las hojas de estilos se declaran así, de modo que el CSS del proveedor viaja junto con su JavaScript y no se olvida en la siguiente pantalla.

Para el dashboard del laboratorio, el indicador necesita un script y una hoja de estilos, mientras que el mando giratorio al estilo jQuery necesita jQuery, el plugin del mando y su hoja de estilos. Dos widgets de la misma página pueden incluir el mismo paquete, y el framework lo carga una sola vez. El orden entre widgets no está garantizado, así que cada widget debe incluir todo lo que necesita en lugar de confiar en que otro widget lo haya cargado antes.

Guarda los archivos del proveedor en la ruta propia de la aplicación, o incrustalos, en lugar de apuntar a una CDN pública. Una caída o un cambio de versión de la CDN no debería poder romper un dashboard de producción.

init crea, update vuelve a sincronizar

El `InitScript` define dos funciones en el widget. `init(options)` se ejecuta una vez, cuando se han cargado los paquetes y existe el contenedor, y crea el objeto del proveedor a partir de las `Options` serializadas.

`update(options, old)` se ejecuta cada vez que el servidor cambia `Options` después de ese primer renderizado y recibe tanto los valores nuevos como los anteriores. Sin una función `update`, cambiar una opción obligaría a destruir y volver a crear el widget entero.

```
init: function (options) {
    this.gauge = new VendorGauge(this.container, options);
},
update: function (options, old) {
    if (options.value !== old.value)
        this.gauge.setValue(options.value);
    if (options.max !== old.max)
        this.gauge.setOptions({ max: options.max });
}
```

La comparación con `old` es lo que hace que `update` sea barato. Solo los valores modificados se envían al objeto del proveedor, una animación no se reinicia por una opción que no tiene nada que ver, y un proveedor que necesita recrearse para una opción concreta puede hacerlo solo cuando esa opción cambió. Escribe la función `update` incluso en un prototipo, porque el botón de servidor del laboratorio que cambia opciones es exactamente el caso que resuelve.

Serializar las opciones desde .NET

La propiedad `Options` es un objeto dinámico en el servidor. Cualquier valor que se le asigne, incluidos objetos anidados, arrays y tipos anónimos, Wisej.NET lo serializa a JSON y llega a JavaScript como el argumento `options`. Una configuración de indicador con un valor, un rango, una lista de bandas de colores y un objeto de etiqueta puede construirse en C# como objetos normales y leerse en el `InitScript` como JavaScript normal.

Dos reglas lo mantienen predecible. Los nombres de las propiedades se convierten a camelCase durante la serialización, así que un `MaxValue` de C# llega como `maxValue`, y el `InitScript` debe usar la grafía de JavaScript. Solo debe asignarse lo que necesita el proveedor; una entidad de base de datos o un objeto de dominio colocado en `Options` se serializa entero, lo que filtra datos al navegador e infla cada renderizado. Construye un objeto pequeño para el widget y asigna ese.

Los objetos anidados se comparan como valores completos durante `update`, así que sustituir el array de bandas provoca una resincronización de las bandas aunque solo haya cambiado un color. Estructura las opciones de modo que los valores que cambian con frecuencia, como la lectura actual, estén en el nivel superior.

Llamar al cliente con `Call`

Parte del comportamiento del proveedor es un método y no una opción: reiniciar una animación, refrescar un diseño o desplazarse hasta un punto. El servidor llega a ellos mediante `Call`, que nombra una función del widget de cliente y pasa argumentos que se serializan igual que `Options`. El `InitScript` define esa función junto a `init` y `update`, y la reenvía a la instancia del proveedor guardada en el widget.

`Call` no se ejecuta de inmediato. Wisej.NET la pone en cola y la envía al navegador con la siguiente respuesta, junto con cualquier cambio de propiedades hecho en la misma petición. Por eso un manejador de botón puede establecer tres opciones y llamar a dos métodos, y el navegador recibe un único lote coherente. También significa que el servidor no puede leer un resultado de `Call`; las variantes que devuelven valores son el tema del Módulo 6.

Ordena el trabajo para que el estado sea correcto antes de que se ejecute el método. Establece primero las opciones y emite la llamada después, porque el lote se aplica en orden y el método del proveedor verá las opciones actualizadas.



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:

- ✓ Configurar paquetes JavaScript y CSS para una instancia de Widget.
- ✓ Escribir funciones ``init(options)`` y, opcionalmente, ``update(options, old)``.
- ✓ Pasar objetos de opciones complejos de .NET a JavaScript mediante la serialización de Wisej.NET.
- ✓ Invocar métodos de cliente desde el servidor con ``Call`` y entender cuándo se pone en cola la llamada.

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

Crea una página de dashboard con dos widgets puntuales: un indicador sencillo y un mando giratorio al estilo jQuery. Ambos deben actualizarse cuando cambien las propiedades del servidor.

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

- ✓ Paquetes del widget configurados
- ✓ InitScript e implementación de update
- ✓ Botón de servidor que cambia las opciones del widget y llama al cliente

Autoevaluación antes de entregar

- ✓ ¿Cuándo se llama a `update`?
- ✓ ¿Qué hace que Options sea flexible pero propenso a errores?
- ✓ ¿Por qué podría ser necesario destruir y volver a crear un widget?