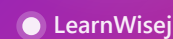


# Llamadas cliente-servidor, serialización y acceso a objetos

Guía de la lección · Guía del laboratorio / examen



## Guía de la lección

Mueve datos y comandos de forma segura entre los componentes .NET y los widgets del navegador. Qué significa esto para un desarrollador de integraciones con Wisej.NET — y cómo leer este módulo.



### Enfoque de producción

Mueve datos y comandos de forma segura entre los componentes .NET y los widgets del navegador.

## 1. El problema de integración

Una integración útil necesita algo más que el renderizado inicial. El servidor debe llamar a métodos del cliente, el cliente puede necesitar llamar a métodos del servidor y ambos lados necesitan contratos de datos predecibles.

## 2. Conceptos clave

| Concepto         | Cómo usarlo                                                                                |
|------------------|--------------------------------------------------------------------------------------------|
| Call / CallAsync | Invoca métodos del widget de cliente desde el código de servidor.                          |
| DTO              | Usa objetos de transferencia de datos pequeños para tener contratos serializados estables. |

## 3. Lección de producción

Wisej.NET ofrece una capa de comunicación que hace que las llamadas de servidor a cliente resulten naturales desde C#. El servidor puede llamar a un método del cliente y continuar, o puede esperar (await) un valor del navegador cuando el flujo de trabajo lo requiera. El cliente también puede llamar a métodos de servidor expuestos con `[WebMethod]`, usando el estilo de callback o el estilo promise/await cuando esté disponible. La serialización es una ventaja, pero también una responsabilidad. Wisej.NET puede serializar objetos anónimos, DTO y valores primitivos, pero eso no significa que deba enviarse al navegador cualquier objeto del servidor. Una buena integración usa objetos pequeños que se ajustan a lo que necesita el widget de JavaScript. Evita filtrar a las opciones del cliente modelos de dominio internos, entidades de base de datos o estado sensible para la seguridad. Los componentes con nombre se reflejan en el modelo de objetos del cliente. Esto es cómodo para demostraciones y para código de aplicación controlado, pero los wrappers de integración deberían evitar depender de rutas globales frágiles.

## Call, CallAsync y EvalAsync

Las tres llamadas de servidor a cliente se diferencian en una sola cosa: si el servidor espera o no. `Call` nombra un método del cliente, pasa argumentos serializados y vuelve inmediatamente; la invocación se pone en cola y se envía con la siguiente respuesta. Es la opción adecuada para comandos de los que el servidor no necesita respuesta, como establecer el valor del indicador o reiniciar una animación.

`CallAsync` también nombra un método del cliente, pero devuelve una Task que se completa con el valor de retorno del método cuando responde el navegador. `EvalAsync` es el mismo mecanismo para una expresión JavaScript arbitraria evaluada en la página.

Esperar un resultado del cliente suspende el manejador de servidor hasta que se completa el viaje de ida y vuelta al navegador. Es aceptable cuando el flujo depende de verdad del valor, como leer el tamaño renderizado antes de elegir un diseño, y es un desperdicio cuando el valor solo se registra. La regla del laboratorio es usar `Call` por defecto y recurrir a una variante con espera solo cuando la siguiente sentencia del servidor no pueda ejecutarse sin la respuesta.

- ¿Se usa el resultado del cliente en el mismo manejador? Si no, usa `Call`.
- Qué ocurre si el navegador nunca responde, por ejemplo porque se cerró la pestaña.
- ¿Se valida el valor esperado antes de confiar en él?

## Serialización y camelCase

Cada argumento que se pasa al cliente y cada valor que este devuelve pasan por el serializador JSON de Wisej.NET. Los valores primitivos, los arrays, los diccionarios, los objetos anónimos y las clases normales se serializan, y los nombres de las propiedades se convierten a camelCase a la salida, de modo que un `SelectedIndex` de C# llega a JavaScript como `selectedIndex`. El código de cliente que lee el nombre en PascalCase obtiene undefined y ningún error.

El sentido contrario tiene sus propias reglas. Un objeto JavaScript devuelto por `CallAsync` llega al servidor como valor dinámico, y el código C# debe leerlo con los nombres de JavaScript o convertirlo en un DTO tipado. Las fechas, los números que superan el rango de enteros seguros de JavaScript y los enums merecen una revisión en cada contrato, porque son los valores que con más frecuencia se deforman en tránsito.

La serialización es también donde se producen las fugas de datos. Una entidad con una propiedad de navegación hacia un registro de cliente serializa también al cliente. El hábito seguro es no pasar nunca un objeto de dominio por la conexión, solo un objeto construido para ese cruce.

## Acceder al modelo de objetos del cliente

Wisej.NET expone su árbol de componentes a los scripts de la página. `App.MainPage` es la página principal actual, los controles con nombre son accesibles como propiedades de su contenedor y `widget(id)` devuelve cualquier componente por su identificador de cliente. En una demostración, esto facilita que un script obtenga un botón, lea un cuadro de texto o encuentre el indicador recorriendo el árbol.

Un wrapper de integración no debería depender de ese árbol. Una ruta como «el indicador del tercer panel de la página principal» se rompe en cuanto un desarrollador renombra un control o lo mueve a un group box. El wrapper ya tiene una referencia a sí mismo como `this` y a su propio contenedor, y todo lo

que necesita del servidor llega mediante opciones y llamadas. Todos los comandos del indicador del laboratorio se invocan directamente sobre el widget, nunca se buscan por ruta.

Reserva el modelo de objetos del cliente para scripts de página bajo el control de la aplicación, como un manejador de atajos de teclado, y trata cualquier uso dentro de un wrapper como un hallazgo de la revisión de código.

### DTO que se mantienen pequeños y estables

Un objeto de transferencia de datos para la conexión es una clase con unas pocas propiedades primitivas, con nombres según lo que necesita el widget, y nada más. El laboratorio devuelve el estado de cliente seleccionado del indicador como un objeto de este tipo: el valor actual, si está por encima del umbral y el ancho y alto renderizados. Cuatro números y un booleano, serializados en unas pocas decenas de bytes y legibles en una entrada de registro.

Los DTO pequeños son estables porque cambian poco. Añadir una propiedad es seguro, ya que el cliente ignora lo que no conoce. Renombrar o eliminar una es un cambio incompatible que debe versionarse, y un DTO con cuatro propiedades se versiona en minutos. Un DTO que replica una entidad de dominio cambia cada vez que cambia la entidad, y cada cambio es una oportunidad de romper el cliente sin avisar.

Escribe los DTO primero, antes que las llamadas que los transportan. Una vez fijadas las formas que cruzan la conexión, tanto el código de servidor como el InitScript tienen algo concreto contra lo que probarse.



#### 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 ``Call``, ``CallAsync`` y ``EvalAsync`` desde el código de servidor para interactuar con el widget de cliente.
- ✓ Entender la serialización JSON de Wisej.NET, incluida la conversión a camelCase de los nombres de propiedades.
- ✓ Usar de forma adecuada el modelo de objetos del cliente, como ``App.MainPage``, los widgets con nombre y ``widget(id)``.
- ✓ Definir DTO pequeños y estables en lugar de pasar objetos de servidor grandes o accidentales.

## 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

Añade comandos al indicador reutilizable: establecer el valor, leer el tamaño renderizado, reiniciar la animación y devolver al servidor el estado seleccionado del cliente.

## 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

- ✓ Llamadas a métodos de servidor a cliente
- ✓ Un ejemplo de ``CallAsync`` o ``EvalAsync``
- ✓ DTO usado para devolver el estado del cliente

## Autoevaluación antes de entregar

- ✓ ¿Cuándo debería el código de servidor esperar (await) un resultado del cliente?
- ✓ ¿Por qué importa el camel-casing al pasar objetos a JavaScript?
- ✓ ¿Por qué los DTO deberían ser pequeños?