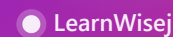


Clases de widget reutilizables y propiedades aptas para el diseñador

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Pasa de instancias puntuales de Widget a clases de servidor reutilizables con propiedades tipadas y una configuración controlada. Qué significa esto para un desarrollador de integraciones con Wisej.NET — y cómo leer este módulo.



Enfoque de producción

Pasa de instancias puntuales de Widget a clases de servidor reutilizables con propiedades tipadas y una configuración controlada.

1. El problema de integración

El prototipo funciona, pero el mismo widget se usará en muchas pantallas. Copiar Packages, Options e InitScript en cada página genera un riesgo de mantenimiento.

2. Conceptos clave

Concepto	Cómo usarlo
Propiedades tipadas	Expone propiedades .NET estables en lugar de las opciones en bruto del proveedor.
Configuración de recursos	Mantén la configuración de paquetes y scripts dentro de la clase wrapper.

3. Lección de producción

Una clase de widget reutilizable es el siguiente paso natural tras un prototipo que funciona. El desarrollador de la aplicación no debería tener que recordar en cada pantalla el orden de los scripts del proveedor, el código exacto de inicialización ni los nombres en bruto de las opciones. La clase wrapper centraliza esas decisiones. La clase de servidor sigue derivando de `Wisej.Web.Widget`, pero su superficie pública se convierte en una API .NET. Por ejemplo, un indicador puede exponer `Value`, `Minimum`, `Maximum`, `Label` y `AnimationEnabled` en lugar de obligar a cada desarrollador a editar un objeto Options dinámico. La clase puede seguir ofreciendo una vía de escape para opciones avanzadas, pero el camino habitual debe ser tipado, fácil de descubrir y apto para el diseñador. Aquí es también donde importan los nombres, los valores predeterminados, las categorías, las descripciones y los convertidores de tipos. El wrapper ya no es un simple host rápido de scripts. Se está convirtiendo en parte del framework de la aplicación que usan otros desarrolladores.

Una clase que es dueña de su configuración

El laboratorio refactoriza el prototipo en `IntegrationLab.Controls.SimpleGauge`, una clase que deriva de `Wisej.Web.Widget`. Su constructor, o su sobrescritura de inicialización, asigna los Packages, el InitScript y las Options predeterminadas que antes se escribían en cada página. Una pantalla que necesita un indicador declara ahora un campo `SimpleGauge`, lo arrastra desde el cuadro de herramientas y nunca ve una línea de JavaScript.

El propio InitScript pasa a un recurso incrustado o a una constante de la clase, de modo que se versiona junto con el código C# que depende de él. Las rutas de los paquetes del proveedor se convierten en constantes en un único lugar. Cuando el proveedor publica una nueva versión, cambia una sola clase y todas las pantallas la recogen en la siguiente compilación.

Una clase wrapper es también el primer lugar donde un registro de decisiones se convierte en código. El nombre de la clase, su espacio de nombres y su comentario de resumen indican para qué sirve el widget y qué biblioteca del proveedor envuelve, que es lo primero que necesitará saber un desarrollador dentro de dos años.

Ocultar las vías de escape

Una vez que la clase es dueña de la configuración, los miembros en bruto se vuelven peligrosos. Un desarrollador que edita Packages en una instancia rompe esa instancia y ninguna otra, que es el tipo de error más difícil de encontrar. Por eso el wrapper oculta o protege Packages, InitScript y el objeto Options en bruto, para que no aparezcan en la ventana Propiedades y no puedan cambiarse a la ligera desde el código de la aplicación.

En la práctica, esto significa sobrescribir los miembros como protegidos, o marcarlos para que el Designer no los muestre ni los serialice. La superficie pública de la clase queda entonces limitada a las propiedades tipadas y los eventos que expone deliberadamente. Los escenarios avanzados pueden seguir cubriéndose con un método protegido que permita a una clase derivada ampliar las opciones, lo que mantiene la vía de escape al alcance de quien entiende las consecuencias.

La autoevaluación del laboratorio pregunta qué miembros deberían ocultarse a los usuarios habituales del wrapper. La respuesta es todo miembro cuyo mal uso pueda romper el widget sin avisar.

Propiedades tipadas que controlan Options

Una propiedad tipada es un pequeño fragmento de C# que lee y escribe en el objeto Options mientras presenta un tipo .NET normal. `Value` es un double con comprobación de rango, `Minimum` y `Maximum` son doubles que se validan entre sí y `Caption` es un string. Cada setter asigna la opción en camelCase, y Wisej.NET renderiza el cambio y llama a `update` en el cliente exactamente igual que en el prototipo.

Los atributos hacen que las propiedades sean aptas para el diseñador. Un atributo de valor predeterminado indica al Designer qué no debe serializar, una categoría agrupa las propiedades en la ventana Propiedades y una descripción explica cada una en el panel de ayuda. Una propiedad costosa de renderizar, como una que obliga al proveedor a reconstruirse, debería indicarlo en su descripción y solo asignar la opción cuando el valor haya cambiado realmente.

- Qué cambios de propiedades deben forzar una actualización en el cliente y cuáles son de solo lectura tras la creación.

- Qué valores se validan en el setter y qué excepción lanza un valor incorrecto.
- Qué valores predeterminados coinciden con los del proveedor, para que una instancia nueva se renderice de forma sensata.

Responsabilidades del wrapper frente a las de la aplicación

El wrapper se encarga del proveedor: cargar sus archivos, crear y eliminar su instancia, traducir las opciones y reenviar los eventos. La aplicación se encarga del significado: qué lectura muestra el indicador, qué umbral importa y qué ocurre cuando se supera. Un wrapper que contiene una llamada a la base de datos o una regla de negocio ha cruzado esa línea, igual que una página que contiene código de inicialización del proveedor.

La página de demostración del laboratorio es la prueba. Coloca un `SimpleGauge`, enlaza `Value` con una lectura de un servicio y gestiona un evento `.NET`. No hay `InitScript` en la página ni ningún nombre del proveedor en su código. Si mañana se sustituyera la biblioteca del proveedor, la página no cambiaría.

Una prueba útil para trazar el límite es imaginar que distribuyes el wrapper como paquete NuGet a otro equipo. Todo lo que ese equipo tuviera que editar dentro del paquete pertenece a la aplicación.



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:

- ✓ Crear una clase que derive de `Wisej.Web.Widget`.
- ✓ Ocultar o proteger `Packages`, `InitScript` y las `Options` en bruto cuando corresponda.
- ✓ Exponer propiedades `.NET` tipadas que actualicen el objeto `Options` de forma segura.
- ✓ Separar la configuración de recursos del proveedor del uso en la aplicación.

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

Refactoriza el prototipo del indicador en `IntegrationLab.Controls.SimpleGauge``, con propiedades tipadas y registro predeterminado de recursos.

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

- ✓ Clase reutilizable derivada de Widget
- ✓ Propiedades tipadas con valores predeterminados
- ✓ Página de demostración que usa la clase sin InitScript personalizado en la página

Autoevaluación antes de entregar

- ✓ ¿Qué miembros deberían ocultarse a los usuarios habituales del wrapper?
- ✓ ¿Qué cambios de propiedades deberían forzar actualizaciones en el cliente?
- ✓ ¿Qué pertenece al wrapper y qué pertenece a la aplicación?