

Controles personalizados, extensiones, wrappers de widgets y componentes reutilizables

Guía de la lección · Guía del laboratorio / examen

● LearnWisej

Guía de la lección

Controles reutilizables, envoltorios de widgets de terceros, propiedades y eventos personalizados, empaquetado de recursos. Lo que esto significa para un equipo avanzado de Wisej.NET — y cómo leer este módulo.



Enfoque de producción

Controles reutilizables, envoltorios de widgets de terceros, propiedades y eventos personalizados, empaquetado de recursos.

1. El problema en producción

Los equipos avanzados suelen necesitar componentes empresariales reutilizables o widgets JavaScript de terceros. El reto es exponerlos como controles de Wisej.NET mantenibles sin perder la productividad del diseñador ni la seguridad del lado del servidor.

2. Conceptos clave

Concepto	Por qué importa
UserControl personalizado	Un control compuesto construido a partir de controles existentes de Wisej.NET que empaqueta el diseño y el comportamiento para reutilizarlos en varias pantallas.
control heredado	Una subclase de un control integrado que añade valores predeterminados, validación o comportamiento sin perder el soporte completo del diseñador y de los temas.
envoltorio de Widget	Un control que aloja una biblioteca JavaScript de terceros dentro de Wisej.NET. Solo debería exponer valores del lado del navegador, mientras las reglas de negocio permanecen en el servidor.
recursos incrustados	Scripts, estilos e imágenes compilados en el ensamblado del control para que el componente se distribuya como un único paquete sin archivos sueltos que desplegar.
contrato de eventos cliente/servidor	El conjunto acordado de eventos y formas de carga útil que se intercambian el widget del navegador y su control de servidor, documentado para que ambos lados puedan evolucionar con seguridad.
valores predeterminados del diseñador	Valores de propiedades y atributos predeterminados razonables para que un control colocado en un formulario funcione de inmediato y tenga un aspecto coherente con el resto de la aplicación.
API del componente	Las propiedades, los métodos y los eventos públicos que expone un control reutilizable. Una API pequeña e intencionada es más fácil de mantener que una que filtra detalles de implementación.

3. Desarrollo de la lección

La reutilización necesita una API

Un control reutilizable debería ser más fácil de usar bien que mal. Dale propiedades, eventos, valores predeterminados y documentación claros. No obligues a cada desarrollador de pantallas a conocer su diseño interno ni sus detalles de JavaScript.

Envuelve el comportamiento, no solo el aspecto

Un envoltorio de un widget de terceros debería traducir llamadas de C# expresadas en términos de negocio en comportamiento del cliente, y traducir los eventos del cliente en eventos de servidor con nombre. Mantén la API pequeña, estable y explícita.

UserControls y controles heredados

Wisej.NET ofrece dos formas de construir un componente reutilizable a partir de controles existentes. Un `UserControl` compone varios controles en una única superficie diseñable con su propio diseño, lo que encaja con un componente como el `StatusTimeline` del laboratorio: una cabecera, una lista de entradas de estado y un pie que, juntos, representan el historial de una orden de trabajo. Un control heredado extiende un único control integrado, como una cuadrícula que siempre muestra las columnas estándar del

tenant, y conserva todas las funciones de su base mientras añade valores predeterminados y comportamiento.

Ambos se colocan desde el Toolbox y ambos aparecen en el Designer, y de eso se trata. Un desarrollador de pantallas arrastra un `StatusTimeline` a un formulario, establece sus propiedades y gestiona sus eventos sin saber cómo se dibuja. La primera pregunta de repaso del laboratorio, si otro desarrollador puede usar el componente sin leer su funcionamiento interno, decide si el componente está terminado.

Elige la composición cuando el componente es algo nuevo hecho de varias partes, y la herencia cuando es una versión especializada de un control existente. Mezclar ambas produce componentes difíciles de tematizar y difíciles de explicar.

Una API de C# estable alrededor de un widget de cliente

El gráfico del laboratorio es un widget JavaScript de terceros, y la aplicación nunca se comunica con él directamente. Una clase `WorkOrderChartWidget` lo envuelve y expone un método como `SetItems`, que acepta una lista de puntos tipados, y un evento de servidor como `SegmentClicked`, que transporta un identificador de segmento. Dentro del envoltorio, el método convierte la lista en la forma de datos del proveedor, y el evento del cliente se traduce desde la callback del proveedor al evento de servidor con nombre.

La estabilidad viene de mantener la API pequeña y expresada en términos de la aplicación. Un método con el nombre de lo que la aplicación quiere, y no de la función del proveedor, sobrevive a una actualización o a un cambio de proveedor. El envoltorio es también donde se gestiona el fallo: cuando el script del proveedor está bloqueado o no se carga, el envoltorio renderiza una alternativa sencilla y no propaga errores a la pantalla, que es la alternativa elegante del laboratorio.

La segunda pregunta de repaso, qué datos pasan del servidor al cliente, se responde con los DTO del envoltorio y nada más. Un envoltorio que envía la entidad de la orden de trabajo al gráfico ha filtrado el dominio al navegador.

Empaquetar los recursos de forma coherente

Un componente reutilizable depende de scripts, hojas de estilo, imágenes y, a veces, fuentes, y todos ellos deben llegar con el componente allí donde se use. La forma fiable es incrustarlos como recursos en el ensamblado del componente y referenciarlos mediante el mecanismo de recursos del framework, de modo que añadir el ensamblado a un proyecto añada todo lo que necesita el componente. Los archivos sueltos copiados en cada aplicación se desincronizan en menos de una versión.

El paquete de recursos incrustados del laboratorio sigue una convención: una carpeta por componente, nombres de archivo con versión para las bibliotecas del proveedor y un único lugar que enumera qué se carga y en qué orden. Las imágenes que usa el componente se adaptan al tema siempre que es posible, para que un tema oscuro no muestre un icono claro. La hoja de estilo propia del componente se carga después de la del proveedor para que prevalezcan sus sobrescrituras.

- ¿Se puede añadir el componente a un proyecto nuevo referenciando un único ensamblado?
- ¿Es visible la versión del proveedor en los nombres de los recursos y en las notas de la versión?
- ¿Actualizar el proveedor cambia una sola carpeta y una sola lista?

Propiedades, eventos y valores predeterminados en tiempo de diseño

La superficie pública de un componente es lo que otros desarrolladores ven en la ventana Properties y en IntelliSense. Cada propiedad tiene un tipo que corresponde a su significado, un atributo de valor predeterminado para que el Designer sepa qué no debe serializar, una categoría y una descripción. Cada evento tiene una clase de argumentos tipada que transporta exactamente lo que necesita un manejador, como el segmento en el que se ha hecho clic, y un nombre que dice qué ha ocurrido en lugar de qué callback del proveedor se ha disparado.

El tiempo de diseño es un escenario de primera clase. El Designer crea el componente sin servicios y sin datos, así que el componente debe detectar el modo de diseño y renderizar datos de ejemplo de forma segura. El modo de ejemplo del laboratorio muestra una línea de tiempo y un gráfico verosímiles en el Designer, lo que permite a un desarrollador de pantallas maquetar el formulario sin ejecutar la aplicación, y nunca llama a un servicio ni lanza excepciones. La tercera pregunta de repaso, si el componente se renderiza de forma segura en el Designer, se comprueba colocándolo en un formulario vacío.

Documenta la API en el propio componente, en los comentarios de resumen y en las descripciones, porque es la única documentación que la mayoría de los usuarios del componente leerán jamás.



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:

- ✓ Diseñar UserControls personalizados y controles heredados para su reutilización.
- ✓ Envolver un widget de cliente tras una API de C# estable.
- ✓ Empaquetar scripts, estilos, imágenes y recursos de forma coherente.
- ✓ Exponer propiedades, eventos y valores predeterminados en tiempo de diseño que otros desarrolladores puedan usar con seguridad.

Guía del laboratorio / examen

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



Laboratorio práctico

Construye un control StatusTimeline reutilizable y un Widget de gráfico o visualización envuelto para el historial de las órdenes de trabajo. Expón propiedades de C#, eventos de servidor, recursos incrustados y un modo de ejemplo en tiempo de diseño.

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

- ✓ UserControl StatusTimeline
- ✓ Envoltorio de Widget o control personalizado
- ✓ Paquete de recursos incrustados
- ✓ Documento del contrato de eventos
- ✓ Pantalla de ejemplo de uso

Preguntas de repaso

- ✓ ¿Puede otro desarrollador usar el componente sin leer su funcionamiento interno?
- ✓ ¿Qué datos pasan del servidor al cliente?
- ✓ ¿Se renderiza el componente de forma segura en el diseñador?



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.