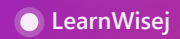


Widgets personalizados, extensiones, temas y proyecto final

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Controles nativos frente a widgets, componentes, extender providers y paquetes de iconos; los packages del control Widget, InitScript, WidgetEvent y la pasarela CallAsync; temas y documentación; el proyecto final de Operations Console. Qué significa esto para un desarrollador de Wisej.NET y cómo leer este módulo.



Enfoque de producción

Controles nativos frente a widgets, componentes, extender providers y paquetes de iconos; los packages del control Widget, InitScript, WidgetEvent y la pasarela CallAsync; temas y documentación; el proyecto final de Operations Console.

1. El problema en producción

Tarde o temprano un producto necesita algo que la biblioteca nativa no incluye: una capacidad especializada del navegador, una librería JavaScript de terceros, una visualización personalizada o una integración reutilizable. La tentación es lanzarse a JavaScript y perder el modelo de Wisej.NET: el estado en el servidor, los temas, el diseñador y un flujo de eventos que otro desarrollador pueda seguir. Este módulo integra un widget de cliente sin renunciar a nada de eso y después completa el proyecto final de Operations Console.

2. Conceptos clave

Concepto	Por qué importa
control nativo	Un control que implementa tanto el lado del servidor como su widget del navegador, así que ya tiene soporte en tiempo de diseño, temas, eventos y estado en el servidor; siempre la primera opción.
tipos de extensión	Control, Widget, Component, Extender Provider e Icon Pack: nombrar el tipo evita que «reutilizable» se confunda con «nuevo control nativo».
control Widget	Un contenedor de cliente de uso general que carga packages JavaScript y CSS, ejecuta un script de inicialización e intercambia eventos y datos con el código del servidor.
Packages e InitScript	La lista de packages declara los archivos de cliente de los que depende el widget; el InitScript crea el objeto JavaScript, conecta sus eventos de cliente y expone solo las funciones que necesita el servidor.
WidgetEvent	El evento del servidor que se lanza cuando el código de cliente llama a <code>fireWidgetEvent</code> ; su tipo y su payload de datos llegan desde el navegador y deben validarse como cualquier otra entrada del usuario.
CallAsync / Eval	La mitad de la pasarela que va del servidor al cliente: llamar a una función expuesta por el widget, o evaluar un script, sin salir de C#.
contrato del widget	Las propiedades, los eventos, el formato del payload, las dependencias de packages y las reglas de temas documentados que permiten a otro desarrollador, o a un asistente de IA con el contexto adecuado, mantener el widget.

3. Desarrollo de la lección

Primero lo nativo, después cinco tipos de extensión

La biblioteca de controles es siempre la primera opción. Un control nativo implementa a la vez el lado del servidor y el del cliente, y por eso participa sin coste adicional en el diseñador de Visual Studio, en `Application.Theme`, en el modelo de eventos y en el estado de servidor por sesión. El código personalizado se reserva para lo que la biblioteca no puede cubrir: una capacidad del navegador sin equivalente nativo, una librería JavaScript de terceros, una visualización a medida o una integración que quieres reutilizar.

Cuando el código personalizado está justificado, nombra el tipo de extensión que estás construyendo. Un **Control** es un control nativo completo con una clase de servidor y un widget de cliente. Un **Widget** es el contenedor de cliente de uso general para un widget JavaScript de terceros o personalizado. Un **Component** puede vivir solo en el servidor o también en el cliente. Un **Extender Provider** añade propiedades y comportamiento a controles existentes, como hacen `ToolTip` o `ErrorProvider`. Un **Icon Pack** empaqueta recursos visuales para el proyecto y el diseñador. La taxonomía evita la confusión de diseño: no todo lo reutilizable es un `UserControl`, y no toda integración JavaScript necesita un nuevo control nativo.

Para Operations Console, las reglas de decisión son breves. Usa primero los controles nativos. Usa un `UserControl` para una composición reutilizable del lado del servidor, como el editor de clientes del Módulo 2. Usa `Widget` para interfaces JavaScript de terceros que puedan vivir dentro de un contenedor de Wisej.NET. Construye una extensión completa solo cuando la integración deba reutilizarse entre aplicaciones y aparecer como un componente de primera clase.

La pasarela Widget en ambas direcciones

El control `Widget` hace tres cosas. Su colección `Packages` carga, en orden, los archivos JavaScript y CSS que necesita la librería de cliente. Su `InitScript` se ejecuta cuando los packages están presentes: `this.init` crea el objeto de cliente dentro de `this.container`, conecta los eventos propios de la librería y expone las pocas funciones que el servidor puede llamar. Por último, transporta mensajes: el código de cliente llama a `this.fireWidgetEvent("ratingChanged", { value: 4 })` y el servidor recibe un `WidgetEvent` cuyos `WidgetEventArgs` exponen el `Type` y el payload `Data`; el código del servidor responde con `CallAsync` para invocar una función del cliente, o con `Eval` para ejecutar un script.

El widget de valoración del laboratorio muestra el viaje de ida y vuelta: un clic en una estrella dispara el evento, y el manejador valida el valor, lo guarda a través de un servicio y lo confirma de vuelta al widget:

```
private async void ratingWidget_WidgetEvent(object sender, WidgetEventArgs e)
{
    if (e.Type != "ratingChanged") return;

    // The payload came from the browser: validate before trusting it.
    int value = Convert.ToInt32(e.Data.value);
    if (value < 1 || value > 5) return;

    await _ratings.SaveAsync(_customerId, value);
    await this.ratingWidget.CallAsync("setSaved", value);
    MessageBox.Show($"Rating {value} saved.");
}
```

Usa el puente con cuidado. El JavaScript se centra en el trabajo exclusivo del navegador: renderizado, gestos, animación. Las reglas de negocio se quedan en el servidor, donde están protegidas, probadas y compartidas con todas las pantallas; duplica alguna en JavaScript solo como optimización deliberada en el cliente, nunca como única copia. Un error clásico: dentro de un callback de la librería, `this` es el objeto de la librería, no el widget, así que captura `var me = this` en `init` y llama a `me.fireWidgetEvent`.

Un contrato que puedas depurar, tematizar y documentar

Un wrapper robusto tiene un contrato. Las propiedades públicas del lado del servidor corresponden a configuración significativa, no a cada opción que expone la librería. Los eventos representan acciones del usuario relevantes para el negocio: `ratingChanged`, no `mouseup`. Si los payloads contienen más de uno o dos valores, define un DTO o una forma JSON clara; el manejador valida y normaliza los datos antes de usarlos, porque cualquiera con las herramientas de desarrollo puede modificar un payload del navegador.

La depuración de widgets empieza en la consola del navegador, y la documentación de Wisej.NET lo dice explícitamente. Cuando un widget aparece en blanco o un evento no llega nunca, revisa primero la consola, después el orden de carga de los packages, las rutas de recursos y el momento de inicialización, y

si el widget se vuelve a crear tras una actualización o un cambio de diseño, lo que significa que `init` debe poder ejecutarse de nuevo sin problemas. Los scripts colocados en el HTML de arranque pueden ejecutarse demasiado pronto para los widgets de Wisej.NET; los recursos incrustados, `Packages`, `InitScript` y las llamadas a nivel de control son patrones más seguros.

Los temas y el comportamiento responsive son el acabado final. Si el widget debe encajar con la consola, decide a propósito: expón propiedades de estilo, asócialas a appearances del tema, usa los extenders `StyleSheet` o `Animation`, o incluye CSS en el package del widget. No fijas los colores de la aplicación en el JavaScript salvo que el widget sea independiente a propósito. Aplica propiedades responsive por perfil de cliente como hiciste con el shell, proporciona etiquetas accesibles y comportamiento de teclado donde sea práctico, y escribe la nota de contexto para la IA: para qué sirve el widget, por qué ningún control nativo encajaba y dónde está el contrato.

- Las dependencias de packages y el orden de carga están documentados.
- Las propiedades, los eventos y el formato del payload están escritos junto al código.
- Las reglas de negocio se aplican en el servidor y la consola del navegador está limpia.



Errores comunes

Sustituir controles estándar por JavaScript porque parece moderno. Poner reglas de negocio en el JavaScript del cliente, donde son más difíciles de proteger y probar. No gestionar la reinicialización del widget tras una actualización o un cambio de diseño. Olvidar documentar las dependencias de packages y el contrato de eventos. Construir un widget personalizado que nadie más del equipo pueda mantener.

Completar Operations Console

El proyecto final integra el curso en una sola aplicación, `OperationsConsole`: el shell navegable (Módulos 1 y 3), el editor de clientes validado (Módulo 2), el explorador `TreeView` y `ListView` (Módulo 4), el `DataGridView` Orders (Módulo 5), la pantalla de dashboard y contenido (Módulo 6) y la pantalla con el widget personalizado o la extensión de este módulo. Cada pantalla se evalúa con las preguntas que plantearon los laboratorios: qué control hace el trabajo más importante, dónde vive el estado, qué ocurre con dos sesiones o con diez veces los datos, y qué código corresponde a un `UserControl` o servicio reutilizable.

La rúbrica asigna a la selección de controles y a los controles de datos un 20 % cada uno, y al diseño y la reutilización, a la validación y el feedback, al contrato del widget personalizado y al acabado y la documentación un 15 % cada uno. Prepara una demo de cinco minutos que siga el guion del proyecto final y escribe `CapstoneNotes.md` a partir de la plantilla del cuaderno de trabajo: controles usados y por qué, supuestos sobre el volumen de datos, decisiones de validación y diseño, código reutilizable extraído y una cosa que mejorarías en producción. El código debe ser comprensible para otro desarrollador y explicable por un asistente de IA que se apoye en tus notas.



Regla de la biblioteca de controles

Elige el control nativo más sencillo que exprese la tarea del usuario antes de escribir código personalizado. Los controles nativos ya te dan soporte en tiempo de diseño, temas, eventos y estado en el servidor; recurre después a un extender o a un UserControl, y construye un widget JavaScript solo cuando ningún control nativo encaje.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Explicar la diferencia entre controles nativos, widgets, componentes, extender providers y paquetes de iconos.
- ✓ Usar el control Widget para cargar packages JavaScript/CSS, inicializar un widget de cliente e intercambiar eventos y datos con el código del servidor.
- ✓ Aplicar temas, propiedades responsive, notas de accesibilidad y notas de contexto para la IA a un paquete de control personalizado.
- ✓ Completar y presentar el proyecto final de Operations Console usando controles nativos y al menos un widget personalizado o una extensión.

Guía del laboratorio / examen

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



Laboratorio práctico

Envuelve un pequeño widget JavaScript de valoración o indicador con el control Widget, o configura un paquete de extensión de Wisej.NET: carga sus packages JavaScript y CSS, lanza un evento del lado del servidor desde el cliente con el patrón de eventos de widget, llama a una función del lado del cliente desde el servidor con CallAsync o Eval, aplícale un tema acorde con la consola y, por último, pule el proyecto final completo de Operations Console y prepara una demo de 5 minutos.

Punto de partida

Abre la solución `OperationsConsole` que has ido ampliando en los Módulos 1 a 6: el shell, el editor de clientes, el explorador de documentos, la cuadrícula Orders y el dashboard ya existen, y la pestaña Widgets sigue siendo la página provisional del Módulo 1. Mantén el trabajo de este módulo en su propia carpeta o namespace para que el proyecto final pueda reutilizarlo.

Enfoque sugerido

- ✓ Decide, y déjalo por escrito, por qué ningún control nativo encaja con la valoración o el indicador que vas a añadir; si un control nativo sí encaja, úsalo y elige otro widget como objetivo.
- ✓ Coloca un control `Widget` en la página Widgets, añade los archivos JavaScript y CSS a su colección `Packages` en orden de carga y escribe un `InitScript` cuyo `init` cree el objeto de cliente dentro de `this.container` y pueda ejecutarse de nuevo sin problemas tras una actualización.
- ✓ En el callback del cliente captura `var me = this` y llama a `me.fireWidgetEvent("ratingChanged", { value })`; en el servidor gestiona `WidgetEvent`, comprueba `e.Type`, valida y normaliza `e.Data` y después guarda a través de un servicio.
- ✓ Confirma al navegador con `CallAsync("setSaved", value)` o `Eval`, muestra al usuario un Toast o un `AlertBox` y aplica el tema al widget mediante `appearances` del tema, CSS empaquetado o el extender `StyleSheet` en lugar de colores fijados en JavaScript.
- ✓ Escribe el contrato junto al código (packages, propiedades, eventos, formato del payload, notas de depuración) y después recorre cada pantalla del proyecto final según la rúbrica y ensaya el guion de la demo de cinco minutos.

Entregables requeridos

- ✓ Pantalla basada en Widget o en una extensión con sus packages JavaScript y CSS cargados
- ✓ Evento de cliente a servidor lanzado con el patrón de eventos de widget o un evento nativo de la extensión
- ✓ Llamada de servidor a cliente mediante `CallAsync`, `Eval` o la API de la extensión
- ✓ Contrato cliente/servidor documentado: packages, propiedades, eventos y formato del payload

- ✓ Proyecto final de Operations Console pulido con un guion de demo de 5 minutos

Preguntas de repaso

- ✓ ¿Qué pasaría si un usuario modificara el payload de `ratingChanged` en las herramientas de desarrollo del navegador antes de que llegue a tu manejador, y qué línea de tu código lo impide?
- ✓ Tu widget aparece en blanco después de ocultar y volver a mostrar la pestaña Widgets: ¿qué tres cosas comprobabas, en qué orden, y por qué el HTML de arranque no es la solución?
- ✓ ¿Qué regla de tu widget podrías trasladar a JavaScript como optimización y cuál no debe salir nunca del servidor?

Entrega del proyecto final

El laboratorio de este módulo es también la entrega del proyecto final. Operations Console debe incluir seis pantallas: el shell de la aplicación con áreas de navegación, comandos y estado; el editor validado de clientes o contactos; el explorador TreeView/ListView con ID estables y una estrategia diferida o virtual; el DataGridView Orders con columnas explícitas, formato y una estrategia para grandes volúmenes de datos; la pantalla de dashboard y contenido con ChartJS y una vista previa de documentos; y la pantalla con el Widget personalizado o la extensión de este laboratorio. Entrega el proyecto con el código fuente, capturas de cada pantalla, `CapstoneNotes.md` escrito a partir de la plantilla del cuaderno de trabajo y una demo de cinco minutos (en directo o grabada) que siga este guion:

- ✓ Explica la arquitectura de la aplicación basada en la biblioteca de controles y después muestra el editor y su comportamiento de validación.
- ✓ Muestra la navegación por el árbol y la lista con la carga de detalles, y después la cuadrícula y cómo gestiona grandes volúmenes de datos.
- ✓ Muestra el dashboard y los controles de contenido, y después el widget personalizado o la extensión, incluida su comunicación de eventos en ambas direcciones.
- ✓ Explica qué debería responder tu asistente de IA si le preguntan por qué se eligió cada control.

Área de la rúbrica	Peso	Cómo es un resultado excelente
Selección de controles	20%	Controles nativos, extenders, UserControls, widgets y extensiones usados de forma adecuada y con una justificación clara.
Diseño y reutilización	15%	Contenedores, docking, anclaje, separadores, pestañas y UserControls usados con limpieza.
Controles de datos	20%	Patrones de TreeView, ListView y DataGridView con identidad, enlace de datos y estrategia para grandes volúmenes correctos.
Validación y feedback	15%	Propiedades de los editores, ErrorProvider y notificaciones usados con claridad y coherencia.
Widget personalizado o extensión	15%	Un contrato cliente/servidor documentado, eventos gestionados y ninguna lógica de negocio fuera de lugar.
Acabado y documentación	15%	Tema, comportamiento responsive, comentarios y notas de contexto para la IA pensados para producción.



Regla de la biblioteca de controles

Elige el control nativo más sencillo que exprese la tarea del usuario antes de escribir código personalizado. Los controles nativos ya te dan soporte en tiempo de diseño, temas, eventos y estado en el servidor; recurre después a un extender o a un UserControl, y construye un widget JavaScript solo cuando ningún control nativo encaje.