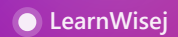


Modelo mental de la biblioteca de controles

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

El árbol de controles del servidor y el modelo de widgets del navegador, las familias de controles, las propiedades comunes y las convenciones de nomenclatura, y el shell de Operations Console que crece a lo largo del curso. Qué significa esto para un desarrollador de Wisej.NET y cómo leer este módulo.



Enfoque de producción

El árbol de controles del servidor y el modelo de widgets del navegador, las familias de controles, las propiedades comunes y las convenciones de nomenclatura, y el shell de Operations Console que crece a lo largo del curso.

1. El problema en producción

Wisej.NET incluye más de cien controles, y un equipo sin un modelo mental compartido de ellos acaba con veinte pantallas de búsqueda distintas, cinco formas de informar de un error de validación y JavaScript personalizado donde bastaría un control nativo. Este módulo fija el vocabulario, las reglas de decisión y las convenciones sobre las que se construyen todos los módulos posteriores, y pone en marcha el shell de Operations Console.

2. Conceptos clave

Concepto	Por qué importa
árbol de controles del servidor	La jerarquía de objetos de control .NET que el servidor mantiene para cada sesión; tu código establece propiedades y gestiona eventos aquí, nunca en markup.
widget del navegador	El objeto JavaScript que Wisej.NET crea para renderizar un control del servidor, sincronizado mediante cambios de estado serializados, no mediante recargas de página.
familia de controles	Un grupo de controles con un mismo propósito (editores, contenedores, listas, cuadrículas, contenido, menús, notificaciones), donde se toman las decisiones.
extender	Un componente que añade propiedades o comportamiento a controles existentes en tiempo de diseño sin crear subclases de ellos.
Widget y extensión	El control <code>Widget</code> aloja un widget JavaScript personalizado o de terceros dentro del modelo de eventos de Wisej.NET; una extensión es una integración empaquetada, normalmente desde NuGet.
UserControl	Una composición reutilizable de controles en el servidor, con significado de negocio y una superficie pública pequeña.
guía de selección de controles	Un documento vivo del equipo que fija, por patrón de pantalla, los controles estándar, el docking, el estilo de validación y de notificación y las convenciones de cuadrícula.

3. Desarrollo de la lección

Primero los objetos del servidor, después los widgets del navegador

El modelo mental del que depende todo lo demás: el código de la aplicación manipula objetos .NET en el servidor, y Wisej.NET serializa los cambios de estado que esos objetos necesitan hacia los widgets JavaScript del navegador. Cuando Operations Console establece `statusLabel.Text`, no se escribe ningún markup; el runtime envía al widget solo el estado modificado. Un `Button` no es una cadena de HTML, sino un objeto de servidor con propiedades, eventos, ciclo de vida, soporte en tiempo de diseño, temas y una representación en el cliente, y cada sesión del navegador posee su propio árbol de estos objetos.

El primer error común se deriva directamente de ahí: tratar los controles como fragmentos HTML y recurrir a markup y scripts de cliente cuando ya existe una propiedad, un evento o un ajuste del tema. El navegador es la segunda frontera: el acceso al sistema de archivos, la impresión, las subidas, las descargas, los iframes y las librerías JavaScript se comportan de forma distinta a los controles de escritorio; por eso existen `Application.Download` y el control `Upload` en lugar de un diálogo de archivos.

Los controles participan en el tema: la documentación de cada control indica la clase de widget y las claves de appearance del tema que utiliza, algo que importa en cuanto el estilo necesita más que `BackColor`, `ForeColor` o `Font`. Aprende cuándo basta con una propiedad sencilla, cuándo usar el tema, cuándo sirve un extender y cuándo está justificado un `UserControl` reutilizable o un `Widget` personalizado. Ignorar las claves de appearance hasta muy avanzado el proyecto es un error común que termina en decenas de sobrescrituras locales de color.

Las familias de controles y las reglas de decisión

La biblioteca se divide en familias, y la familia es donde se toman las decisiones. Los editores (`TextBox` , `TypedTextBox` , `MaskedTextBox` , `DateTimePicker` , `NumericUpDown` , `ComboBox`) recogen entradas con tipo. Los botones y menús (`Button` , `MenuBar` , `ToolBar` , `ContextMenu`) inician comandos. Los contenedores (`Panel` , `SplitContainer` , `TabControl` , `Accordion` , paneles de diseño) crean la estructura de la pantalla. Las listas, árboles, repetidores y cuadrículas (`ListView` , `TreeView` , `DataRepeater` , `DataGridView`) muestran datos. Los controles de contenido muestran texto, imágenes, HTML, PDF, archivos y contenido alojado en el navegador. Las notificaciones (`Toast` , `AlertBox` , `MessageBox`) comunican el estado. Los extenders añaden comportamiento a controles existentes sin crear subclases, y las extensiones y los widgets incorporan capacidades web externas al modelo de componentes y de eventos.

El curso trata la biblioteca como un sistema de diseño, no como una lista que memorizar: lo que importa es qué familia encaja con la tarea, cómo se rellena, cómo se comporta con grandes volúmenes de datos, con el diseño y con los temas, cómo se comunica con el navegador y cómo se mantiene en una aplicación de servidor multiusuario. Una buena decisión empieza por la tarea del usuario: un editor para editar un valor, un `ListView` para elegir de una colección con presentación de iconos, mosaicos o detalles, un `TreeView` para datos jerárquicos con expansión y contracción, un `DataGridView` para registros tabulares con ordenación, edición, columnas personalizadas o grandes volúmenes de datos, y un `UserControl` cuando un grupo de controles tiene un significado de negocio reutilizable.

Tres reglas mantienen honesta la decisión. Elige el control más sencillo que exprese la tarea sin código personalizado, prefiriendo los controles estándar y los extenders antes que cualquier widget JavaScript personalizado; construir un widget antes de agotar la biblioteca nativa es el error más caro del curso. Considera el volumen de datos antes de elegir una estrategia de carga: rellenar `Rows` e `Items` a mano está bien para conjuntos pequeños, los grandes necesitan enlace de datos, modo virtual, caché o carga diferida. Usa las convenciones de tema y appearance para el estilo de toda la aplicación y las propiedades individuales solo para excepciones locales.

Propiedades comunes y convenciones que mantienen unidos todos los módulos

Los nombres deben identificar la función de negocio, no solo el tipo visual: `customerGrid` dice más que `dataGridView1` , y `StatusPanel` , `SearchTextBox` y `SaveButton` hacen legibles los manejadores de eventos sin abrir el diseñador. Conservar los nombres por defecto del diseñador convierte `button3_Click` en arqueología, y los instructores lo penalizan mucho.

El docking y el anclaje deben ser deliberados: `Dock` rellena y apila, `Anchor` sigue los bordes cuando se redimensiona el navegador, y el posicionamiento absoluto solo tiene cabida donde nada más encaja; mezclar estrategias de diseño al azar en una misma pantalla es un error común que se rompe al primer redimensionado. Los controles del flujo de teclado necesitan un `TabIndex` deliberado, los errores de validación necesitan mensajes claros, las imágenes deben seguir una convención de `ImageList` o de recursos, las operaciones costosas necesitan feedback visible y las etiquetas de accesibilidad deben corresponder a la función de negocio.

Una aplicación de producción recoge todo esto en una guía de selección de controles: un documento vivo que define, por patrón de pantalla, los controles estándar, el docking por defecto, el estilo de validación y de notificación, la posición de los menús, el paquete de iconos, las convenciones de cuadrícula y las

expectativas de accesibilidad. No es burocracia; es la forma en que un equipo se mantiene coherente. El `ControlSelection.md` de tu laboratorio es la primera página de esa guía y responde a estas preguntas para cada área:

- ¿El control elegido corresponde a la tarea del usuario, y existe un control nativo más sencillo?
- ¿Los datos son planos, jerárquicos, tabulares, repetidos o visuales, y hasta qué tamaño crecerán?
- ¿La pantalla necesita una frontera de `UserControl`, y está realmente justificado un `Widget` personalizado?

El shell de Operations Console

El proyecto del curso empieza como un shell, y cada módulo añade una sección hasta que tienes una aplicación de referencia funcional en lugar de ejemplos aislados. El shell es una `Page` principal (o un `Form`) con cuatro áreas: la navegación a la izquierda, un área de contenido que ocupa el resto, un área de comandos para las acciones de la sección actual y un área de estado que informa de lo que está haciendo la consola.

En el laboratorio, el área de navegación contiene un `Button` con un nombre claro por cada página provisional (Editors, Layouts, Lists and Trees, `DataGridView`, Dashboard, Widgets), el área de contenido es un `Panel` acoplado con `Fill` y el área de estado es un panel acoplado con un `Label`. El único método que merece la pena escribir cambia el contenido, acopla la nueva página con `Fill` y actualiza la etiqueta de estado, de modo que cada cambio de navegación se notifica desde un solo lugar y una página que no se carga no se queda en blanco.

```
private void Navigate(Control content, string title)
{
    try
    {
        contentPanel.Controls.Clear();
        content.Dock = DockStyle.Fill;
        contentPanel.Controls.Add(content);
        statusLabel.Text = $"Showing {title}";
    }
    catch (Exception ex)
    {
        statusLabel.Text = $"Could not open {title}";
        MessageBox.Show(ex.Message);
    }
}
```

Cada página provisional es su propia clase, más adelante un `UserControl`, en su propia carpeta para que el proyecto final pueda reutilizarla, y los nombres reflejan la función en todo momento: `navigationPanel`, `contentPanel`, `commandPanel`, `statusLabel`, `editorsButton`. La ampliación opcional es un pequeño panel de diagnóstico que muestra el nombre del control seleccionado, el ID del registro, el perfil de cliente activo y la hora de la última actualización.



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 relación entre el árbol de controles del servidor de Wisej.NET y los widgets del navegador.
- ✓ Clasificar la biblioteca de controles por propósito: entradas, contenedores, listas, cuadrículas de datos, contenido, menús, notificaciones, extenders y extensiones.
- ✓ Aplicar de forma coherente las propiedades comunes, los eventos, la nomenclatura, el anclaje, el docking, las imágenes, el orden de tabulación, las etiquetas de accesibilidad y las convenciones del diseñador.
- ✓ Crear un shell de proyecto del curso llamado Operations Console que crecerá a lo largo del curso.

Guía del laboratorio / examen

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



Laboratorio práctico

Crea el shell del catálogo de controles de Operations Console: una Page principal con un área de navegación, un área de contenido, un área de comandos y un área de estado; páginas provisionales para Editors, Layouts, Lists and Trees, DataGridView, Dashboard y Widgets; una nota ControlSelection.md dentro del proyecto que indique qué familia de controles usarás para cada área; nombres de controles claros en todo el proyecto; y un área de estado que se actualice cada vez que cambie la navegación.

Punto de partida

Abre Visual Studio y crea una nueva aplicación web Wisej.NET llamada `OperationsConsole`; es la solución que amplían todos los laboratorios posteriores, así que mantén el trabajo de cada módulo en su propia carpeta o namespace. La plantilla te da una `Page` principal vacía y un `Program.cs` que la abre, y todavía no existe nada más.

Enfoque sugerido

- ✓ Esboza las cuatro áreas antes de tocar el diseñador y después acóplalas en orden: navegación a la izquierda, comandos arriba, estado abajo y el panel de contenido añadido en último lugar con Dock en Fill.
- ✓ Cambia el nombre de cada control según su función de negocio en cuanto lo coloques (`navigationPanel` , `contentPanel` , `commandPanel` , `statusLabel` , `editorsButton`) y asigna un `TabIndex` deliberado a los botones de navegación.
- ✓ Crea un UserControl provisional por sección (Editors, Layouts, Lists and Trees, DataGridView, Dashboard, Widgets), cada uno con una etiqueta de título, en su propia carpeta para que el proyecto final pueda reutilizarlo.
- ✓ Escribe un único método `Navigate(Control content, string title)` que vacíe el panel de contenido, acople la página con Fill, la añada y actualice la etiqueta de estado dentro de un try / catch; haz que todos los botones de navegación pasen por él.
- ✓ Añade `ControlSelection.md` al proyecto y registra, para cada área y página provisional, la familia de controles que elegiste y la alternativa nativa más sencilla que descartaste.

Entregables requeridos

- ✓ Shell principal con áreas de navegación, contenido, comandos y estado
- ✓ Páginas provisionales para Editors, Layouts, Lists and Trees, DataGridView, Dashboard y Widgets
- ✓ ControlSelection.md con la familia de controles elegida para cada área
- ✓ Nombres de controles claros y un área de estado que se actualiza al navegar
- ✓ Captura de pantalla y una nota breve que explique las decisiones sobre los controles

Preguntas de repaso

- ✓ Cuando estableces `statusLabel.Text` en el método de navegación, ¿qué viaja realmente al navegador y por qué es distinto de escribir HTML?
- ✓ ¿Qué área del shell convertirías primero en un UserControl y cuál sería su superficie pública?
- ✓ Si un módulo posterior necesita un área de navegación plegable, ¿qué contenedor nativo sustituye al panel acoplado y por qué es preferible a un widget personalizado?



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.