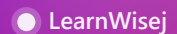


# Contenedores, diseños, navegación y reutilización

Guía de la lección · Guía del laboratorio / examen



## Guía de la lección

Orden de docking, anclaje, diseños de flujo y de tabla, propiedades responsive por perfil de cliente, superficies de navegación y de comandos, y los UserControls como unidad de reutilización. Qué significa esto para un desarrollador de Wisej.NET y cómo leer este módulo.



### Enfoque de producción

Orden de docking, anclaje, diseños de flujo y de tabla, propiedades responsive por perfil de cliente, superficies de navegación y de comandos, y los UserControls como unidad de reutilización.

## 1. El problema en producción

Los contenedores son la arquitectura de la interfaz de usuario: deciden cómo se agrupan, redimensionan, navegan y reutilizan los controles. La Operations Console del Módulo 1 todavía tiene paneles provisionales, y una pantalla en la que cada control se posiciona a mano se vuelve frágil en cuanto se redimensiona el navegador. Los problemas de diseño suelen ser problemas de concepción, no de píxeles, y este módulo sustituye el ajuste píxel a píxel por un plan: regiones acopladas, un SplitContainer, un TabControl, un modelo de comandos real y UserControls que ocultan sus detalles internos.

## 2. Conceptos clave

| Concepto                     | Por qué importa                                                                                                                                                                                                             |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| orden de docking             | El docking fija un hijo a un lado de su contenedor o le hace rellenar lo que queda, y se aplica en el orden de los hijos, así que los mismos cuatro controles producen shells distintos según el orden en que se añadieron. |
| anclaje                      | El anclaje mantiene los bordes de un control ligados a los bordes de su contenedor; anclado a izquierda y derecha se estira con la anchura, y sin anclaje en ningún eje flota manteniendo su posición relativa.             |
| SplitContainer               | Dos regiones adyacentes que el usuario puede redimensionar; la elección adecuada siempre que sea la persona, y no el desarrollador, quien decida cuánto espacio recibe el árbol de navegación o el panel de detalle.        |
| paneles de diseño            | FlowLayoutPanel coloca los hijos uno junto a otro y los ajusta a la línea siguiente, TableLayoutPanel alinea filas y columnas, y FlexLayoutPanel reparte el espacio y ajusta líneas como un diseño responsive moderno.      |
| propiedades responsive       | Una propiedad que guarda un valor distinto por perfil de cliente, fijado en tiempo de diseño, para que un diseño de escritorio pueda cambiar unas pocas propiedades en pantallas estrechas sin una segunda pantalla.        |
| modelo de comandos           | Una regla coherente sobre dónde vive cada comando: MenuBar, ToolBar, RibbonBar, ContextMenu o StatusBar, cada uno con su función, y todos invocando comandos de la aplicación en lugar de contener lógica de negocio.       |
| superficie de un UserControl | La unidad de reutilización: un control compuesto que expone unas pocas propiedades, eventos y métodos significativos en lugar de cada control hijo, para que quienes lo usan nunca se acoplen a su implementación.          |

## 3. Desarrollo de la lección

### Elige el contenedor adecuado para cada tarea

Wisej.NET te ofrece posicionamiento absoluto, docking, anclaje y paneles de diseño más completos, y la primera decisión es qué superficie aloja la pantalla. Una **Page** es la superficie de entrada natural para una aplicación de estilo web como Operations Console: ocupa todo el navegador y no tiene marco de ventana. Un **Form** conviene para ventanas tipo diálogo, flujos modales y patrones de estilo MDI, y un **Desktop** puede alojar varios formularios cuando la aplicación mantiene a propósito una metáfora de escritorio. Dentro de esa superficie, **Panel** y **GroupBox** agrupan controles relacionados; la única diferencia es si el grupo merece un título visible.

Tres contenedores hacen la mayor parte del trabajo estructural. Un **SplitContainer** es ideal cuando el usuario puede redimensionar dos regiones adyacentes, como un árbol de navegación junto a un panel de detalle. Un **TabControl** organiza páginas del mismo nivel para que el usuario pase de una sección a otra sin abrir ventanas nuevas. Un **Accordion** apila secciones plegables para navegación, ajustes y áreas de trabajo plegables. En el laboratorio, la región izquierda de la consola se convierte en un panel de

`SplitContainer` , la región derecha en un `TabControl` , y los marcadores provisionales del Módulo 1 en sus pestañas.

## Orden de docking, anclaje y paneles de diseño

El docking fija un control hijo a un lado de su contenedor ( `DockStyle.Top` , `Bottom` , `Left` , `Right` ) o le hace ocupar el espacio restante ( `DockStyle.Fill` ). Como el docking se aplica en el orden de los hijos, el orden en que se añaden los controles importa: la `ToolBar` y la `StatusBar` deben ocupar sus bordes antes de que el `SplitContainer` rellene el resto; de lo contrario, el control `Fill` se coloca primero y las barras se superponen a él. Cuando un shell se ve mal en el diseñador, revisa el orden de los hijos antes de tocar un solo tamaño.

El anclaje mantiene los bordes de un control relacionados con los bordes de su contenedor. Un control anclado a izquierda y derecha cambia de anchura cuando cambia la anchura del contenedor; un control sin anclaje horizontal ni vertical flota manteniendo su posición relativa, algo adecuado para una tarjeta de inicio de sesión centrada e inadecuado para una cuadrícula. Usa el docking para las regiones de la aplicación (navegación, cabecera, pie o estado, contenido) y el anclaje dentro de esas regiones para editores y botones.

Para colecciones de controles, recurre a un panel de diseño en lugar de a coordenadas. `FlowLayoutPanel` coloca los hijos uno junto a otro y puede ajustarlos a la línea siguiente, así que una fila de chips de filtro sobrevive a un navegador estrecho. `TableLayoutPanel` organiza el contenido en filas y columnas, que es lo que necesita un formulario con etiquetas y editores alineados. `FlexLayoutPanel` reparte el espacio y ajusta líneas como los diseños responsive modernos, la mejor opción cuando la interfaz necesita un reparto flexible en lugar de una cuadrícula fija.

El diseño responsive en Wisej.NET no se limita a las media queries CSS. Las propiedades responsive guardan varios valores para distintos perfiles de cliente y se establecen en tiempo de diseño en la cuadrícula de propiedades; los perfiles de cliente identifican situaciones de dispositivo o navegador, como un teléfono de hasta 600 píxeles de ancho, una tableta de hasta 1024 y un escritorio por encima de eso. La regla práctica es construir primero un buen diseño de escritorio y después definir el puñado de propiedades que deben cambiar en tabletas o pantallas estrechas: visibilidad, tamaño, docking, columnas visibles, comportamiento del panel lateral y posición de los comandos. No crees una pantalla completamente distinta salvo que el flujo de trabajo cambie de verdad. Cuando una propiedad no basta, la lógica según el perfil en el código hace el mismo trabajo, como muestra el método de perfil estrecho del laboratorio:

```
// OperationsConsole shell: add the regions in this order
this.toolBar.Dock = DockStyle.Top;
this.statusBar.Dock = DockStyle.Bottom;
this.splitMain.Dock = DockStyle.Fill; // navigation | TabControl detail

private void ApplyNarrowProfile(bool narrow)
{
    // narrow screens: hide the navigation panel, show the menu button
    this.splitMain.Panel1Collapsed = narrow;
    this.btnMobileMenu.Visible = narrow;
    this.pnlProfile.Text = narrow ? "Narrow profile" : "Desktop profile"; // a StatusBar
panel
}
}
```

## Superficies de navegación y de comandos

Los controles de navegación necesitan un modelo de comandos coherente, y ese modelo trata de dónde va cada comando, no de su aspecto. Un `MenuBar` es adecuado para los menús tradicionales de una aplicación. Una `ToolBar` contiene las acciones de uso frecuente, y por eso Operations Console coloca ahí New, Refresh y Save. Un `RibbonBar` agrupa los comandos por área de tarea cuando la aplicación tiene muchos comandos, algo excesivo para una consola con una docena. Un `ContextMenu` es para comandos ligados a un objeto concreto: una fila, un nodo o la selección actual. En todos los casos el control lanza un evento y un manejador mínimo invoca un comando de la aplicación; los controles de navegación que contienen lógica de negocio son uno de los errores comunes del módulo.

La `StatusBar` no es un cajón de sastre. Debe mostrar estado relevante, como el número de registros seleccionados, la hora de la última actualización, el estado de validación o el perfil de cliente activo, y el laboratorio pide estado real en lugar de la palabra «Ready». Un `TabControl` merece la misma disciplina. Las pestañas no sustituyen a un flujo de trabajo: úsalas para secciones del mismo nivel como Summary, Details, History, Documents y Notes, nunca como pasos obligatorios de un asistente. No escondas campos obligatorios en pestañas sin una navegación clara hacia los errores de validación y, si las pestañas se pueden mover, decide si el nuevo orden se guarda y si afecta a una sesión o al perfil del usuario.

## El UserControl es la unidad de reutilización

Cuando el mismo grupo de controles aparece dos veces, la tentación es copiarlo, y el coste llega con la tercera copia y el primer cambio. Un `UserControl` es la principal unidad de reutilización. Un control reutilizable debe tener una superficie pública pequeña: propiedades como `CustomerId`, `ReadOnly` y `SelectedDate`, eventos como `Saved` o `SelectionChanged`, y métodos como `LoadData()` o `ValidateContent()`. Evita exponer todos los controles hijos internos salvo que se trate de un control avanzado intencionado; de lo contrario, las pantallas que lo usan quedan acopladas a la implementación, y renombrar un TextBox rompe tres páginas. El laboratorio mueve la barra de estado y la cabecera de registro repetidas de la consola a UserControls; la pregunta de repaso es si alguien podría usarlos sin saber qué hay dentro.

Deja las decisiones por escrito. El `LayoutNotes.md` del laboratorio es un archivo breve que guardas en el proyecto y que registra qué regiones están acopladas y en qué orden, qué controles están anclados y por

qué, y qué propiedades cambian en el perfil estrecho. Es lo que lee el siguiente desarrollador antes de abrir el diseñador, y debe reflejar estas reglas de decisión de diseño:

- Usa el docking para las regiones de la aplicación y un `SplitContainer` cuando el usuario deba controlar el reparto del espacio.
- Usa `TabControl` para secciones del mismo nivel, `FlowLayoutPanel` para colecciones que se ajustan de línea y `TableLayoutPanel` para formularios con etiquetas y editores alineados.
- Usa propiedades responsive para los cambios específicos de cada perfil y encapsula los diseños repetidos como `UserControls` en lugar de copiar grupos de controles.



### Errores comunes

Mezclar posicionamiento absoluto, docking y flujo sin un plan. Activar `AutoSize` en contenedores o cuadrículas grandes y forzar sin querer cálculos de diseño enormes. Crear pestañas que esconden campos obligatorios y después mostrar errores de validación imprecisos. Exponer públicamente todos los controles hijos de un `UserControl`. Hacer que los controles de navegación se encarguen de la lógica de negocio en lugar de invocar comandos de la aplicación.



### 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:

- ✓ Usar correctamente `Form`, `Page`, `Desktop`, `Panel`, `GroupBox`, `SplitContainer`, `TabControl`, `Accordion`, `FlowLayoutPanel`, `TableLayoutPanel` y `UserControl`.
- ✓ Combinar docking, anclaje, flujo, tabla y propiedades responsive sin pelearse con el motor de diseño.
- ✓ Diseñar superficies de navegación y de comandos con `MenuBar`, `ToolBar`, `RibbonBar`, `StatusBar`, `ContextMenu` y `TabControl`.
- ✓ Empaquetar patrones de interfaz repetidos en `UserControls` con propiedades y eventos públicos limpios.

## Guía del laboratorio / examen

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



### Laboratorio práctico

Sustituye los marcadores provisionales del Módulo 1 por el shell real de Operations Console: un SplitContainer con la navegación a la izquierda, una superficie de detalle TabControl a la derecha, una ToolBar para los comandos frecuentes y una StatusBar que muestre el estado real. Mueve la interfaz repetida a UserControls que expongan solo propiedades y eventos públicos significativos, implementa un diseño para perfil estrecho con propiedades responsive o lógica según el perfil, y documenta tus decisiones de docking y anclaje en LayoutNotes.md.

## Punto de partida

Abre la solución `OperationsConsole` del Módulo 1 y su `Page` principal: todavía tiene los paneles provisionales de navegación, contenido, comandos y estado, las páginas provisionales para Editors, Layouts, Lists and Trees, DataGridView, Dashboard y Widgets, y el UserControl `CustomerEditor` del Módulo 2. Mantén el trabajo de este módulo en su propia carpeta o namespace para que el proyecto final pueda reutilizarlo.

## Enfoque sugerido

- ✓ Esboza el shell antes de abrir el diseñador: ToolBar acoplada con Top, StatusBar acoplada con Bottom y un SplitContainer acoplado con Fill, y escribe primero ese orden de hijos en `LayoutNotes.md`.
- ✓ Construye el shell en ese orden, coloca el control de navegación en el panel izquierdo del SplitContainer y un TabControl en el panel derecho, y convierte cada página provisional del Módulo 1 en una pestaña para una sección del mismo nivel.
- ✓ Añade a la ToolBar los botones New, Refresh y Save y un ContextMenu en la lista de navegación; cada manejador se mantiene mínimo y llama a un método de comando con nombre de la página.
- ✓ Extrae la cabecera de registro y la barra de estado a los UserControls `RecordHeader` y `StatusStrip`, que expongan unas pocas propiedades (por ejemplo `Title`, `RecordCount`) y un evento (por ejemplo `RefreshRequested`), nunca sus controles hijos.
- ✓ Define un perfil de cliente estrecho, da a las propiedades responsive (visibilidad de la navegación, plegado del panel del separador, posición de la ToolBar) un valor para el perfil estrecho o implementa `ApplyNarrowProfile`, actualiza la StatusBar con el perfil activo y haz capturas en anchos de escritorio y estrechos.

## Entregables requeridos

- ✓ Shell de navegación construido con SplitContainer, TabControl, ToolBar y StatusBar
- ✓ Interfaz repetida movida a UserControls con una superficie pública pequeña
- ✓ Diseño para perfil estrecho con propiedades responsive o lógica según el perfil

- ✓ LayoutNotes.md que documenta las decisiones de docking y anclaje
- ✓ Capturas de pantalla en anchos de escritorio y estrechos

## Preguntas de repaso

- ✓ Si el SplitContainer se hubiera añadido a la página antes que la ToolBar y la StatusBar, ¿qué vería el usuario y por qué importa el orden?
- ✓ ¿Qué miembros de tu UserControl podrías renombrar sin tocar ninguna página que lo use, y cuáles no? ¿Qué te dice eso sobre su superficie pública?
- ✓ ¿Qué propiedades cambiaron en el perfil estrecho y por qué no construiste una pantalla estrecha aparte?



### 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.