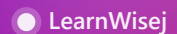


El sistema visual de Wisej.NET

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Cómo los controles del lado del servidor, los widgets del lado del cliente, las appearances del tema, los estados, las clases CSS y los motores de diseño producen juntos lo que ve el usuario, cuál de las tres capas de personalización es responsable de cada decisión, y el shell de Adaptive 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

Cómo los controles del lado del servidor, los widgets del lado del cliente, las appearances del tema, los estados, las clases CSS y los motores de diseño producen juntos lo que ve el usuario, cuál de las tres capas de personalización es responsable de cada decisión, y el shell de Adaptive Operations Console que crece a lo largo del curso.

1. El problema en producción

Un equipo que trata Wisej.NET como un framework basado ante todo en CSS acaba con `BackColor` asignado en doscientos controles, selectores que se rompen en la siguiente actualización y manejadores `Resize` que recalculan `Bounds` a mano; cuando se pide una versión para teléfono, la única opción que queda es reescribirlo todo. Este módulo aclara de dónde vienen realmente el aspecto y el diseño de un control, nombra las tres capas de personalización y cuál es responsable de cada decisión, e inicia el shell de Adaptive Operations Console que amplían todos los módulos siguientes.

2. Conceptos clave

Concepto	Por qué importa
control del servidor y widget del cliente	Escribes controles .NET en el servidor; el navegador muestra widgets JavaScript que Wisej.NET mantiene sincronizados, así que una decisión visual puede estar a cualquiera de los dos lados de esa línea.
tema	Un único archivo JSON de colores, imágenes, fuentes, appearances, estados, estilos, propiedades y ajustes que da a cada control estándar su identidad; el contrato entre ambos mundos.
appearance y <code>AppearanceKey</code>	Cada control expone una clave de appearance; el tema activo contiene una appearance para esa clave, y una clave semántica como <code>action-button</code> selecciona una variante.
estado	Una condición de interacción con nombre dentro de una appearance (default, hovered, pressed, focused, invalid, selected) cuyos estilos y propiedades se aplican mientras está activa.
<code>CssClass</code> y <code>CssStyle</code>	Estilo directo en el navegador: una clase de una hoja de estilos de la aplicación con ámbito, o una cadena de estilo sin procesar en el elemento del widget para un valor puntual poco frecuente.
motor de diseño y <code>Dock</code>	La posición y el tamaño vienen de la jerarquía de contenedores, no del tema; el acoplamiento apila las regiones contra los bordes del padre y rellena lo que queda.
inventario visual	La lista escrita de controles que necesitan appearances, tokens de color y fuente, regiones del diseño y puntos de corte a partir de la cual se planifica el trabajo de tema, diseño y perfiles.

3. Desarrollo de la lección

El tema es el contrato entre los controles del servidor y los widgets del navegador

En Wisej.NET el desarrollador escribe controles .NET en el servidor, pero lo que ve el usuario es un widget del lado del cliente renderizado por el navegador. Un `Button`, `TextBox`, `TabControl`, `DataGridView`, `Panel` o widget personalizado expone una clave de appearance, el tema activo contiene una appearance para esa clave, y cada appearance contiene estados, estilos, propiedades y componentes hijos. Por eso un cambio visual profesional puede hacerse una sola vez, de forma centralizada, en lugar de dispersarse por controles individuales: cambia la appearance del botón y todos los botones de todas las sesiones la siguen.

Por eso también el tematizado de Wisej.NET es más que CSS normal. El tema llega a los propios controles, a las métricas de los widgets de las que depende el diseño, a las fuentes y tamaños con los que renderiza el diseñador de Visual Studio, y a colores, imágenes y appearances que el CSS por sí solo no puede expresar. El motor de temas puede generar clases CSS a partir de la appearance y el estado, y las propiedades del tema pueden sobrescribir los estilos, así que una regla escrita a mano contra el DOM interno del widget lucha contra el motor en lugar de aprovecharlo. Trata el archivo del tema como un

recurso de diseño de producto: los colores con nombre y las fuentes compartidas son tokens de diseño, las appearances modelan variantes de componentes, los estados expresan la interacción, y un botón, una cuadrícula, un editor, un panel, una barra de herramientas y un formulario parecen relacionados aunque los hayan creado desarrolladores distintos.

Tres capas, y cuál es responsable de cada decisión

La capa uno es el tema: identidad visual global, comportamiento de los estados, métricas de los widgets, colores con nombre, fuentes y las appearances de los componentes estándar. La capa dos es el modelo de controles del lado del servidor: `BackColor`, `ForeColor`, `Font`, `Padding`, `Margin`, `Dock`, `Anchor`, `Size`, `MinimumSize`, `MaximumSize`, `AppearanceKey` y las propiedades responsive. La capa tres es el estilo directo en el navegador: `CssClass`, `CssStyle`, el extender `StyleSheet` en código heredado, archivos CSS personalizados, `InitScript` y la personalización de widgets JavaScript. Un desarrollador de nivel intermedio debe saber qué capa es responsable de cada decisión antes de tocar ninguna.

La regla de decisión que aplica todo el curso es breve. Primero el tema para la identidad de los controles estándar. `AppearanceKey` para una variante semántica, como un botón de acción o un botón destructivo, dejando que el tema defina sus estados default, hovered, pressed, focused y disabled. `CssClass` para un punto de enganche específico de la aplicación y de ámbito reducido: una tarjeta de métricas, HTML incrustado, un widget de terceros. `CssStyle` solo para un valor pequeño, puntual o dinámico, usado con moderación y de forma local. Contenedores de diseño para el posicionamiento, propiedades responsive para los valores que el diseñador asigna por perfil, y `ResponsiveProfileChanged` para el comportamiento que un valor de propiedad no puede expresar.

La regla negativa importa igual: no parchees cada control estándar con CSS directo. Las appearances del tema son mejores para los controles integrados porque participan en los estados, en el renderizado del diseñador y en los tokens compartidos; el CSS es para contenido específico de la aplicación, widgets de terceros y HTML incrustado. Cuando un tema «no se aplica», los primeros sospechosos son un `BackColor`, `ForeColor`, `Font`, `CssStyle` o `CssClass` personalizado que lo sobrescribe, una clave de appearance incorrecta, o un tema definido en el diseñador pero no seleccionado para la aplicación en ejecución.

El inventario visual: tokens, regiones y puntos de corte

Antes de que nadie abra el Theme Builder, el laboratorio pide un inventario de la aplicación tal como será. Recorre Adaptive Operations Console y enumera los controles que necesitan una appearance personalizada: los botones de comando, los paneles que se convierten en tarjetas de métricas, las pestañas, el encabezado de la cuadrícula, el estado invalid del editor de tickets, los tooltips. Enumera los tokens de color que el tema definirá una vez y reutilizará en todas partes: `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` y `focusFrame`, además de una fuente default, una heading y una mono. Enumera las regiones del diseño: barra de herramientas, carril de navegación, área de trabajo con tarjetas de métricas y cuadrícula de tickets, panel de detalles y barra de estado. Por último, enumera los puntos de corte: escritorio a 1440, escritorio pequeño a 1024, tableta a 768 en vertical y 1024 en horizontal, teléfono a 390 en vertical y 844 en horizontal.

Escríbelo como `VisualInventory.md` dentro del proyecto y añade a cada fila una cuarta columna: la capa responsable. Las esquinas redondeadas de una tarjeta de métricas corresponden a una `CssClass` con

ámbito; el color de hover del botón, al tema; el ancho del panel de detalles, a `Dock` ; si el carril es visible en un teléfono, a una propiedad responsive. Anota en el mismo archivo la decisión sobre el tema base: el Módulo 2 deriva el tema `AdaptiveOps` de uno de los temas integrados y hereda sus appearances, estados, imágenes y métricas, así que anota por qué lo elegiste, no solo cuál.

El shell de Adaptive Operations Console

El proyecto del curso es una consola de operaciones con un tema de marca y un diseño para escritorio, tableta y teléfono. La versión de escritorio tiene un carril de navegación a la izquierda, una barra de herramientas, tarjetas de métricas, una cuadrícula de tickets y un panel de detalles. La versión móvil contrae la navegación, apila la cuadrícula y los detalles, oculta el texto de baja prioridad, usa la visualización solo icono donde conviene y mantiene un desplazamiento táctil cómodo. El Módulo 1 construye solo el shell de escritorio, con `Dock` y `Panel` anidados y nada más: barra de herramientas arriba, estado abajo, navegación a la izquierda, detalles a la derecha, y un área de trabajo acoplada para rellenar con un `MinimumSize` para que nunca se colapse.

```
private void BuildShell()
{
    toolbarPanel.Dock = DockStyle.Top;        toolbarPanel.Height = 56;
    statusPanel.Dock = DockStyle.Bottom;      statusPanel.Height = 28;
    navigationPanel.Dock = DockStyle.Left;    navigationPanel.Width = 240;
    detailsPanel.Dock = DockStyle.Right;      detailsPanel.Width = 360;
    workspacePanel.Dock = DockStyle.Fill;
    workspacePanel.MinimumSize = new Size(320, 240);
}

private void MainPage_Resize(object sender, EventArgs e)
{
    widthLabel.Text = $"Width: {this.Width} px";
}
```

El único comportamiento del shell es la etiqueta de estado que muestra el ancho actual del navegador. La `Page` principal siempre ocupa el viewport del navegador, así que su evento `Resize` se lanza cada vez que el usuario redimensiona la ventana; escribir `this.Width` en la etiqueta da a los módulos siguientes un número que comparar con los puntos de corte de `ClientProfiles.json` y con el nombre de `Application.ActiveProfile` . Envuelve la actualización en un try / catch que deje un texto alternativo honesto en lugar de un valor obsoleto. Lo que el shell no debe contener es tan importante como lo que hace:

- Nada de sobrescrituras de `BackColor` , `ForeColor` o `Font` : el tema base es responsable del aspecto hasta el Módulo 2.
- Nada de cadenas `CssStyle` ni CSS personalizado todavía: nada en el shell es lo bastante específico de la aplicación para necesitarlos.
- Nada de código `Resize` que asigne `Bounds` : el acoplamiento ya resuelve cualquier ancho de escritorio.

Haz la captura de escritorio cuando el shell se ejecute y escribe la nota que explique, región por región, qué capa es responsable de cada decisión visual; es el germen de la nota de arquitectura que pide el

proyecto final.



Regla del sistema visual

Usa el mecanismo de Wisej.NET responsable del problema: el tema para la identidad de los controles estándar, AppearanceKey para las variantes semánticas, CssClass para CSS específico de la aplicación con ámbito, CssStyle solo para valores dinámicos puntuales, contenedores de diseño en lugar de código de redimensionado, propiedades responsive para los valores por perfil definidos en el diseñador, y ResponsiveProfileChanged para el comportamiento que no es simplemente visual.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Explicar por qué el tematizado de Wisej.NET es más que CSS normal y cómo llega a los controles, los widgets, las métricas del diseñador, las fuentes, los colores, las imágenes y las appearances.
- ✓ Identificar cuándo usar un archivo de tema, un tema integrado, una propiedad de control, CssClass, CssStyle o un contenedor de diseño.
- ✓ Crear un inventario visual de una aplicación existente y definir tokens del tema, regiones del diseño y puntos de corte responsive.
- ✓ Iniciar el proyecto del curso, Adaptive Operations Console, con un shell de escritorio de base y el comportamiento móvil objetivo.

Guía del laboratorio / examen

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



Laboratorio práctico

Crea la solución Adaptive Operations Console y su primer shell de escritorio: una Page principal con un panel de barra de herramientas arriba, un panel de navegación a la izquierda, un área de trabajo de relleno, un panel de detalles a la derecha y un panel de estado abajo, contruidos con Dock y Panels anidados. Elige un tema base integrado y documenta la elección, escribe un `VisualInventory.md` que enumere los controles que necesitan appearances personalizadas, los tokens con nombre de color y fuente, las regiones del diseño y los puntos de corte de escritorio, tableta y teléfono, y añade una etiqueta de estado que muestre el ancho actual del navegador para que los módulos siguientes puedan verificar los perfiles.

Punto de partida

Abre Visual Studio y crea una nueva aplicación web de Wisej.NET llamada `AdaptiveOperationsConsole`; es la solución que amplían todos los laboratorios posteriores, así que guarda el trabajo de cada módulo en su propia carpeta. La plantilla te da una `Page` principal vacía, un `Program.cs` que la abre y la selección de tema predeterminada; todavía no existen paneles, ni archivo de tema, ni inventario.

Enfoque sugerido

- ✓ Dibuja primero las cinco regiones en papel, luego coloca cinco `Panel` en la Page principal y acóplalos en este orden: `toolbarPanel` arriba, `statusPanel` abajo, `navigationPanel` a la izquierda, `detailsPanel` a la derecha y `workspacePanel` el último, con Dock en Fill y un `MinimumSize` de 320 por 240.
- ✓ Elige uno de los temas integrados como base, selecciónalo para la aplicación en ejecución (no solo en el diseñador) y anota el motivo de la elección en tus notas del laboratorio antes de cambiar nada más.
- ✓ Añade una `widthLabel` al panel de estado, gestiona el evento `Resize` de la Page principal y escribe en ella `$"Width: {this.Width} px"` desde un único método `ReportWidth()` envuelto en try / catch; llámalo una vez desde el constructor para que la etiqueta nunca esté vacía.
- ✓ Crea `VisualInventory.md` en el proyecto con cuatro tablas: controles que necesitan appearances personalizadas, tokens de color y fuente (brandPrimary, brandAccent, surface, surfaceAlt, textMain, textMuted, danger, warning, success, focusFrame; default, heading, mono), regiones del diseño, y puntos de corte de escritorio, tableta y teléfono; añade a cada fila una columna «responsable» que indique tema, propiedad del control, CssClass, CssStyle o contenedor de diseño.
- ✓ Ejecuta el shell, redimensiona el navegador del ancho de escritorio al de tableta y al de teléfono mientras observas la etiqueta, haz la captura de escritorio y escribe la nota que explique qué capa es responsable de cada decisión visual del shell.

Entregables requeridos

- ✓ Solución Adaptive Operations Console con un shell de escritorio acoplado formado por los paneles de barra de herramientas, navegación, área de trabajo, detalles y estado
- ✓ Tema base elegido y configurado, con el motivo anotado en las notas del laboratorio
- ✓ VisualInventory.md con los controles que necesitan appearances, los tokens del tema, las regiones del diseño y los puntos de corte
- ✓ Etiqueta de estado que muestra el ancho actual del navegador
- ✓ Captura de pantalla de escritorio y una nota que explique qué capa es responsable de cada decisión visual

Preguntas de repaso

- ✓ El comando principal de la barra de herramientas necesita el color de marca en todas las pantallas. ¿Cuál de las tres capas es responsable de esa decisión y qué fallaría si asignaras `BackColor` al botón?
- ✓ Tu etiqueta de ancho indica 1024 px y el panel de detalles sigue visible. ¿Qué mecanismo de un módulo posterior decidirá si debe estarlo, y por qué una media query CSS es la primera respuesta equivocada en Wisej.NET?
- ✓ Si mañana se cambiara el tema base por otro tema integrado, ¿qué filas de tu VisualInventory.md cambiarían y cuáles no, y qué te dice eso sobre dónde residen las decisiones del inventario?



Regla del sistema visual

Usa el mecanismo de Wisej.NET responsable del problema: el tema para la identidad de los controles estándar, AppearanceKey para las variantes semánticas, CssClass para CSS específico de la aplicación con ámbito, CssStyle solo para valores dinámicos puntuales, contenedores de diseño en lugar de código de redimensionado, propiedades responsive para los valores por perfil definidos en el diseñador, y ResponsiveProfileChanged para el comportamiento que no es simplemente visual.