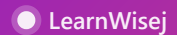


Temas, recursos, localización y modernización de la interfaz

Moderniza las aplicaciones Wisej.NET con temas, CSSStyle, recursos, iconos, cambio de tema en tiempo de ejecución, localización y reglas de coherencia visual.



Objetivo de la lección

Modernizar aplicaciones Wisej.NET usando temas, CSSStyle, recursos, iconos, cambio de tema en tiempo de ejecución, localización y reglas de coherencia visual, separando el *aspecto* (tema/CSS) del *comportamiento* (C#) para que un rediseño o un nuevo idioma nunca obliguen a editar todas las pantallas.

Todo este módulo se apoya en una disciplina: **la apariencia y la lógica son trabajos distintos y van en lugares distintos**. Tu C# decide *qué hace la aplicación*; un tema decide *qué aspecto tiene*; los recursos contienen los *textos e imágenes* que muestra. Cuando esas tres cosas se mantienen separadas, puedes cambiar el estilo del producto, añadir un modo oscuro o publicar una versión en italiano sin tocar un solo manejador de eventos. Cuando se mezclan —un color hexadecimal escrito en un control aquí, un texto fijo en el código allá—, cada cambio se convierte en una búsqueda por todo el código. Esta lección trata de mantenerlas separadas a propósito.

1. Separar la apariencia del comportamiento

En Wisej.NET, el *aspecto* de una pantalla son datos, no código. Un **tema** es un documento JSON que define colores, fuentes, espaciado, bordes y la apariencia de cada tipo de control: botones, cuadrículas, paneles, pestañas. Tu C# nunca menciona `#0d6efd` ni `font-size: 14px`; simplemente crea un `Button`, y el tema activo decide cómo se pinta ese botón. Cambia el tema y toda la aplicación cambia de aspecto, porque los controles nunca llevaron sus propios colores.

Capa	De qué se encarga	Dónde vive
Tema (JSON)	El aspecto global: la paleta de colores, las fuentes, la escala de espaciado y la apariencia por defecto de cada tipo de control.	Un archivo <code>.theme</code> / JSON de tema, o Theme Builder.
CSSStyle	Una modificación visual puntual aplicada a un control o a una pequeña familia de controles.	La propiedad <code>CssStyle</code> del control (en C# o en el diseñador).
Comportamiento (C#)	Lo que la aplicación <i>hace</i> : validación, flujos de trabajo, datos, navegación.	Tus manejadores de eventos y servicios.
Recursos	Los textos e imágenes que muestra la aplicación: rótulos, mensajes, iconos, logotipos.	Archivos de recursos / la carpeta Resources.

La ventaja de centralizar colores y espaciado en un tema en lugar de dar estilo a cada control es el efecto palanca. Supón que el color de tu marca pasa de azul a verde azulado. Con un tema, editas un valor y todos los botones, enlaces y resaltados se actualizan a la vez. Con estilos por control, ese mismo cambio es un buscar y reemplazar en docenas de archivos, y *seguro* que se te escapa alguno. La centralización también impone una coherencia en la que no tuviste que pensar: como cada control lee su espaciado de la misma escala, las pantallas quedan alineadas sin que nadie tenga que ajustar píxeles a ojo.



Regla en palabras sencillas

Si un valor visual describe *toda tu aplicación* (el azul de la marca, el margen estándar), pertenece al tema. Si describe *un control inusual*, pertenece al CSSStyle de ese control. Casi nunca debería vivir como un literal en C#.

2. Temas y Theme Builder

Wisej.NET incluye un conjunto de temas integrados y ofrece **Theme Builder**, un editor visual para crearlos o retocarlos. Partes de un tema base —por ejemplo, uno claro por defecto o una variante oscura como **BootstrapDark-4**—, ajustas la paleta, la tipografía y la apariencia de los controles, y guardas el resultado como tema de tu aplicación. Como Theme Builder edita el mismo JSON que lee el runtime, no hay un paso de «exportación» aparte en el que equivocarse: lo que diseñas es lo que carga la aplicación.

Merece la pena nombrar algunos conceptos. Un tema define **colores con nombre** (una paleta) para que las apariencias hagan referencia a `"highlight"` o `"error"` en lugar de a un hexadecimal en bruto, de modo que una sola edición de la paleta se propaga a todas partes. Las **apariencias** son las definiciones de estilo por tipo de control. Y los **mixins** te permiten superponer un pequeño conjunto de modificaciones sobre un tema base en lugar de duplicarlo entero: la herramienta adecuada cuando quieres «el tema estándar, pero con nuestro color de marca y nuestro logotipo».

3. Cambiar de tema en tiempo de ejecución

Como el tema no es más que el aspecto activo, puedes cambiarlo mientras la aplicación se ejecuta: para un selector de modo oscuro por usuario, un modo de alto contraste por accesibilidad o una apariencia

específica para cada cliente. Asignas el tema activo de la aplicación y Wisej.NET vuelve a renderizar las pantallas abiertas con las nuevas apariencias. No reconstruyes controles ni recargas datos; solo cambia la pintura.

Un selector de modo oscuro que cambia el tema activo en tiempo de ejecución

```
private void btnDarkMode_Click(object sender, EventArgs e)
{
    // Pick the theme by name; the runtime re-skins open screens.
    // Behavior does not change – only the active look does.
    var next = _darkMode ? "Default-4" : "BootstrapDark-4";
    Application.Theme = next;           // switch the whole app's look
    _darkMode = !_darkMode;

    // Persist the choice so it survives the next login.
    _userPrefs.Save("theme", next);
}
```

Elegir el tema al arrancar, antes de que se pinte la primera pantalla

```
protected override void OnSessionStart(EventArgs e)
{
    // Read the saved preference and set the look once, up front,
    // so the user never sees a flash of the wrong theme.
    var saved = _userPrefs.Load("theme") ?? "Default-4";
    Application.Theme = saved;
    base.OnSessionStart(e);
}
```

Fíjate en lo que el manejador *no* hace: ni colores literales, ni cambios de estilo por control, ni recarga de datos. El aspecto son datos con nombre; cambiarlo es una modificación de una línea. (La propiedad o API exacta para establecer el tema activo puede variar según la versión de Wisej.NET; el mecanismo es el mismo: asignas por nombre el tema activo de la aplicación y el runtime cambia el aspecto de la interfaz).

4. CSSStyle: la excepción puntual

A veces un control necesita de verdad un aspecto distinto al de sus hermanos, de una forma que el tema no debería codificar globalmente: un aviso con un fondo diferenciado, una tarjeta con una sombra suave. Para eso existe la propiedad **CssStyle** del control: asocia CSS a ese widget concreto sin ensuciar el tema ni tener que meterte tú en el DOM del navegador.

Una modificación con CSSStyle en un único panel de estado

```
pnlBanner.CssStyle =
    "background: var(--wisej-warning, #fff3cd);" + // prefer a theme var over a raw hex
    "border-radius: 8px;" +
    "padding: 12px 16px;";

lblBanner.Text = "Your session expires in 5 minutes.";
```

Usa CSSStyle con moderación y de forma deliberada. Cada cadena de CSSStyle es un pequeño fragmento de apariencia que vive *fuera* del tema, así que no seguirá automáticamente un rediseño. El hábito que debes construir: recurre primero al tema y baja a CSSStyle solo cuando un control sea realmente un caso único. Si te encuentras pegando el mismo CSSStyle en muchos controles, es señal de que debería estar en el tema.

Tema frente a CSSStyle frente a propiedad en línea: cuándo usar cada uno

Herramienta	Alcance	Úsala cuando...
Tema (JSON / Theme Builder)	Toda la aplicación, todos los controles de un tipo.	El valor forma parte de la identidad de tu producto: paleta, fuentes, escala de espaciado, aspecto por defecto de los controles. Primera opción para todo lo compartido.
Propiedad CSSStyle	Un control, o una pequeña familia elegida a propósito.	Una excepción visual real que el tema no debería llevar: un aviso especial, una tarjeta resaltada.
Propiedad en línea (BackColor, Font...)	Un control, una propiedad.	Rara vez: un ajuste realmente local y dirigido por los datos (p. ej., una fila se vuelve roja porque un valor está vencido). Nunca para colores de toda la marca.

5. Recursos: imágenes, iconos y textos

La otra mitad de «separar el aspecto de la lógica» es tratar el *contenido* que muestra la aplicación —sus rótulos, mensajes, logotipos e iconos— como **recursos** en lugar de literales repartidos por el código. Las imágenes y los iconos van en la carpeta Resources y se referencian por nombre, para que un diseñador pueda cambiar el logotipo sin recompilar tus manejadores. Los textos visibles para el usuario van en archivos de recursos identificados por clave, que es justo lo que hace posible la localización: la misma clave se resuelve en un texto distinto para cada referencia cultural.

Cargar un mensaje localizado y un icono desde recursos en lugar de dejarlos fijos en el código

```
// String comes from a resource keyed by name – not a literal.  
// At runtime it resolves to the entry for the current culture.  
lblWelcome.Text = Resources.Strings.Welcome;      // "Welcome" / "Benvenuto" / "Willkommen"  
  
// Image comes from a named resource, not an ad-hoc file path.  
btnSave.ImageSource = "resource.wx/App.Resources.SaveIcon.png";  
  
// A whole Form/Page can be marked Localizable so the Designer  
// stores its captions per culture automatically.
```

La forma general siempre es la misma: una **clave** estable en el código, resuelta en contenido real en tiempo de ejecución según la referencia cultural activa. Esa indirección es todo el truco: el código hace referencia a claves, y el significado de una clave puede cambiar por idioma, por marca o por temporada sin que el código lo sepa.

6. Localización: referencia cultural, textos y formato

Localizar es más que traducir. Hay dos cosas que varían según la **referencia cultural** (culture): los *textos* que muestras y el *formato* de fechas, números y moneda. Wisej.NET localiza el texto visible para el usuario resolviendo las claves de recursos para la referencia cultural actual (los Forms y Pages pueden marcarse como **Localizable** para que el diseñador gestione sus rótulos por idioma). El formato sigue la referencia cultural del hilo, así que una fecha se muestra como **7/3/2026** para un usuario y como **03/07/2026** para otro, con el *mismo* C#, porque formateaste con la referencia cultural y no con un patrón fijo.

Establecer la referencia cultural del usuario y dejar que el formato la siga

```
// Set the culture for this user's session (from a preference or a picker).
var culture = new CultureInfo(_userPrefs.Load("culture") ?? "en-US");
System.Threading.Thread.CurrentThread.CurrentUICulture = culture; // which strings
System.Threading.Thread.CurrentThread.CurrentCulture = culture; // how dates/numbers
format

// Now the SAME code formats correctly for every user:
lblTotal.Text = amount.ToString("C"); // $1,234.50 vs 1.234,50 €
lblDue.Text = dueDate.ToShortDateString(); // culture-aware, no hard-coded pattern
```

Una advertencia práctica: el texto traducido cambia de longitud. Las etiquetas en alemán suelen ser más largas que en inglés; algunas escrituras son más altas. Un diseño que encaja perfectamente en inglés puede recortarse o partirse mal en otro idioma, así que **prueba la localización en los diseños reales de las pantallas**, no solo en una hoja de cálculo de textos. Por eso este módulo trata la localización como un asunto de interfaz, no de texto.



Regla de producción

Primero el tema, después las excepciones. Mantén cada texto e imagen visible para el usuario como recurso, establece la referencia cultural una vez por sesión y formatea con ella en lugar de con un patrón fijo; después comprueba el resultado en la pantalla real para cada idioma que publiques.



Objetivos de aprendizaje

Al terminar este módulo serás capaz de:

- ✓ Aplicar temas integrados y cambiar de tema en tiempo de ejecución o al arrancar sin cambiar el comportamiento.
- ✓ Usar Theme Builder y CSSStyle sin repartir estilos puntuales por todas partes.
- ✓ Centralizar colores, espaciado, textos e imágenes para que un rediseño se edite en un solo lugar, no en cada pantalla.
- ✓ Localizar Forms, Pages, recursos del sistema y el formato de los datos cuando corresponda.
- ✓ Crear una lista de comprobación de modernización de la interfaz para pantallas empresariales.

7. Una interfaz moderna, en resumen

Modernizar no es solo aplicar temas. Una pantalla empresarial moderna usa una **escala de espaciado** coherente, una **jerarquía tipográfica** clara, un **contraste** accesible, **iconos** reconocibles y **patrones de estado y de error** predecibles, para que los usuarios aprendan un único lenguaje visual y lo reutilicen en todas partes. La razón por la que todo esto pertenece al tema es la misma de antes: la coherencia que obtienes gratis es coherencia que no puedes romper por accidente, y un rediseño se convierte en una edición de datos en lugar de en un proyecto.

Errores comunes

Error	Por qué perjudica	Haz esto en su lugar
Colores y espaciado fijos en el código, repartidos por las pantallas.	Un cambio de marca se convierte en un buscar y reemplazar en docenas de archivos, y se te escapará alguno.	Define la paleta y el espaciado en el tema; haz referencia a valores con nombre, no a hexadecimales.
Pegar el mismo CSSStyle en muchos controles.	El estilo vive fuera del tema, así que no seguirá un rediseño y se desincroniza.	Sube el estilo repetido a la apariencia del tema para ese tipo de control.
Textos visibles para el usuario escritos como literales en C#.	No se pueden traducir, y corregir un texto obliga a recompilar el código.	Guarda los textos en recursos identificados por clave; resuélvelos según la referencia cultural.
Imágenes incrustadas sobre la marcha desde rutas de archivo.	Los recursos gráficos se dispersan, fallan al desplegar y no se pueden cambiar sin recompilar.	Añádelas a Resources y haz referencia a ellas por nombre.
Ignorar la referencia cultural: patrones fijos de fecha y número.	Las fechas, los decimales y la moneda se muestran mal en otras regiones.	Establece la referencia cultural de la sesión y formatea con ella (<code>Tostring("C")</code> , fecha corta).
Probar las traducciones solo como lista de textos.	El texto más largo se recorta o se parte; los diseños se rompen en la pantalla real.	Prueba cada idioma en el diseño real antes de publicar.

Vocabulario clave

- **Tema:** el documento JSON que define el aspecto de toda la aplicación: paleta, fuentes, espaciado y apariencias por control.
- **Theme Builder:** el editor visual para crear y retocar temas.
- **Apariencia:** la definición de estilo de un tema para un tipo de control.
- **Mixin:** una pequeña capa de modificaciones superpuesta a un tema base en lugar de duplicarlo.
- **CSSStyle:** la propiedad de un control para aplicar una modificación CSS puntual a ese widget concreto.
- **Recurso:** un texto o imagen con nombre al que la aplicación hace referencia en lugar de a un literal, lo que permite sustituirlo y traducirlo.

- **Propiedad Localizable:** un Form/Page (o control) marcado para que el diseñador guarde sus rótulos por referencia cultural.
- **Referencia cultural (culture):** el ajuste que determina tanto qué textos se muestran como el formato de fechas, números y moneda.
- **Cambio de tema en tiempo de ejecución:** cambiar el tema activo mientras se ejecuta la aplicación, cambiando el aspecto de las pantallas abiertas sin reconstruirlas.

Error habitual que debes evitar



Error habitual

Repartir colores fijos, CSS puntual y textos literales por muchas pantallas. Funciona en una demo y se degrada en producción: un cambio de estilo o un nuevo idioma obliga entonces a editar cada archivo en lugar de un tema y un conjunto de recursos. Es exactamente el mal olor que eliminarás en el laboratorio de este módulo.

Lista de revisión del diseñador y del código

- ✓ La pantalla sigue siendo utilizable en el diseñador de Visual Studio / Wisej.NET.
- ✓ Los controles tienen nombres lo bastante claros para que un compañero pueda seguir el código de los eventos.
- ✓ Los colores y el espaciado vienen del tema, no de literales en C# ni de CSSStyle repetidos.
- ✓ Los textos e imágenes visibles para el usuario son recursos, no valores fijos en el código.
- ✓ La referencia cultural se establece por sesión y el formato la sigue.
- ✓ La lógica de negocio no queda atrapada en los manejadores de eventos visuales.
- ✓ Las rutas de fallo son visibles, se registran y se explican sin exponer detalles internos.
- ✓ El entregable puede revisarse sin ejecutar el curso completo.



Autoevaluación antes del laboratorio

¿Qué estilos son del tema y cuáles son verdaderas excepciones a nivel de pantalla? ¿Puedes cambiar el aspecto de toda la aplicación en una línea sin tocar un manejador? ¿Cada texto e imagen visible para el usuario es un recurso? ¿Qué cambia con un texto en alemán o en italiano, y sobrevive el diseño? ¿Puede un rediseño futuro evitar editar cada pantalla? Si respondes que sí a todo, estás listo.

Código complementario

`ThemingLocalizationPatterns.cs` reúne los patrones de este módulo en un único archivo compilable. Descárgalo y léelo junto con el laboratorio.

↓ [Descargar ThemingLocalizationPatterns.cs](#)