Tema 1.4.4. Diseño de controles

DISEÑO DE CONTROLES

1. Controles básicos 1
2. Controles de lista 3
3. Listview 4
4. DataGrid..... 5

1. CONTROLES BÁSICOS

Algunos de los controles más habituales son:

- Button
- CheckBox
- Label
- RadioButton
- TextBlock
- TextBox

Button

Los botones en WPF pueden tener un diseño muy complejo, se puede formatear su texto, imágenes, etc.

Permite emplear propiedades como: Padding, Background, Foreground, FontWeight, etc. Recordemos que si queremos dar estilos a los botones de una página, aplicación, etc.; podemos usar <Window.Resources>, app.xaml, etc.

Además, se pueden añadir controles de panel y otros controles para crear un botón multicapa, etc.

```
<!-- Button -->
<Button Click="btnSimple_Click">Simple</Button>
<Button Background="Black" Foreground="White" FontWeight="Bold">Con formato</Button>
<Button>
    <StackPanel Orientation="Horizontal">
        <TextBlock>Botón</TextBlock>
        <TextBlock Foreground="Blue" FontWeight="Bold" Margin="2,0">Compuesto</TextBlock>
    </StackPanel>
</Button>
```

CheckBox

El control CheckBox hereda de la clase ContentControl, por lo que puede tomar personalizarse utilizando otros controles como “hijos”.

Además de la propiedad IsChecked (true/false), tiene otra IsThreeState que introduce la opción de null.

```
<!-- CheckBox -->
<CheckBox IsChecked="True" Name="cbTresEstados" >Habilitarlos todos con un click</CheckBox>
<StackPanel Margin="20,5">
    <CheckBox Name="cbFeatureAbc" IsThreeState="True"> Tres estados</CheckBox>
    <CheckBox Name="cbFeatureXyz" >Otra opción</CheckBox>
</StackPanel>
```

Label

Similar a TextBlock, Label utiliza su propiedad **Content** para alojar cualquier tipo de dato, como por ejemplo, una cadena. Además, Label permite:

- Especificar un borde.
- **Renderizar** otros controles, por ejemplo, una imagen.
- Usar **templates** a través de la propiedad **ContentTemplate**. Usar teclas de acceso para posicionar el **foco**.

Con la propiedad Target se puede especificar un control con el que se relaciona la etiqueta. Por ejemplo:

```
<Label Content="Nombre:" Target="{Binding ElementName=txtNombre}" />
```

RadioButton

Permite trabajar con una lista de opciones posibles y solo una de ellas se puede seleccionar al mismo tiempo.

Estos controles se pueden agrupar dentro de grupos, mediante la propiedad **GroupName**.

```
<!-- RadioButton -->
<Label FontWeight="Bold">¿Qué opción eliges</Label>
<RadioButton GroupName="opc">Primera</RadioButton>
<RadioButton GroupName="opc">Segunda</RadioButton>
<RadioButton GroupName="opc" IsChecked="True">Tercera</RadioButton>
```

TextBlock

Similar al control Label, TextBlock se utiliza con cadenas de texto de varias líneas. Solo puede contener texto.

Tiene **propiedades** interesantes como:

- **TextTrimming**: cuando el texto es demasiado largo, pondrá …
- **TextWrapping**: el texto saltará a la siguiente línea cuando no se pueda ajustar más dentro de la línea actual.

```
<!-- TextBlock-->
<TextBlock Margin="10" TextWrapping="Wrap">
    TextBlock con <Bold>texto negrita</Bold>, <Italic>italic</Italic> y <Underline>subrayado</Underline>.
    <Span Background="Silver" Foreground="Green">dentro de la etiqueta Span</Span>
    Además, añade un hypervínculo a <Hyperlink RequestNavigate="Hyperlink_RequestNavigate"
                                               NavigateUri="https://www.google.com">Google</Hyperlink>.
</TextBlock>
```

Similar a HTML, permite utilizar **** para dar formato al texto entre las etiquetas, puede dar formato a un **hyperlink**, texto en negrita con **<bold>,** etc.

TextBox

Permite escribir texto plano en una sola línea o múltiples líneas como en un editor de texto.

```
<!--TextBox-->
<TextBox Text="Texto en una línea" />
<TextBox AcceptsReturn="True" TextWrapping="Wrap"  Text="Texto en múltiples líneas" />
```

2. CONTROLES DE LISTA

- ComboBox
- ListBox

Aunque existe un tercer componente (ItemsControl), nosotros vamos a trabajar con ComboBox y ListBox.

DataTemplate se empleará para reemplazar la apariencia visual de un elemento de un control (ListBox, ComboBox, ListView, etc.). Si no especifica una plantilla de datos, WPF toma la plantilla predeterminada que es solo un TextBlock.

Combobox

El control ComboBox es parecido al control ListBox, pero usa menos espacio, ya que la lista de ítems se encuentra oculta cuando no se necesita.

Permite personalizar su contenido modificando el control **ComboBoxItem**:

```
<!--ComboBox -->
<ComboBox>
    <ComboBoxItem>
        <StackPanel Orientation="Horizontal">
            <TextBlock Foreground="Red">Rojo</TextBlock>
        </StackPanel>
    </ComboBoxItem>
    <ComboBoxItem>
        <StackPanel Orientation="Horizontal">
            <TextBlock Foreground="Green">Verde</TextBlock>
        </StackPanel>
    </ComboBoxItem>
    <ComboBoxItem>
        <StackPanel Orientation="Horizontal">
            <TextBlock Foreground="Blue">Azul</TextBlock>
        </StackPanel>
    </ComboBoxItem>
</ComboBox>
```

Cuando se quiere mostrar el contenido personalizado de datos que provienen de una BBDD, se puede utilizar una plantilla de datos personalizada o **DataTemplate**. Para ello, primero tenemos que indicar que cada elemento del combo es una plantilla **<ComboBox.ItemTemplate>.**

```
<ComboBox Name="cmbColors">
    <ComboBox.ItemTemplate>
        <DataTemplate>
            <StackPanel Orientation="Horizontal">
                <Image Source="/wifi.PNG" />
                <TextBlock Text="{Binding}" />
            </StackPanel>
        </DataTemplate>
    </ComboBox.ItemTemplate>
</ComboBox>
```

Para controlar la selección de los elementos del combo se utiliza la propiedad **SelectedIndex** y el evento **SelectionChanged** para capturar cuándo se cambia el elemento seleccionado, ya sea por código o por el usuario.

ListBox

Permite la selección de uno o varios elementos de la lista. Para añadir elementos a la lista es necesario trabajar con la propiedad <**ListBoxItem**>.

```
<!-- ListBox -->
<ListBox>
    <ListBoxItem>
        <StackPanel Orientation="Horizontal">
            <TextBlock>ListBox Item #1</TextBlock>
        </StackPanel>
    </ListBoxItem>
    <ListBoxItem>
        <StackPanel Orientation="Horizontal">
            <TextBlock>ListBox Item #2</TextBlock>
        </StackPanel>
    </ListBoxItem>
</ListBox>
```

Al igual que con combobox, también se pueden personalizar los ítem de BBDD mediante plantillas de datos con <**ListBox.ItemTemplate**> y <**DataTemplate**>.

La selección de datos se complica ya que se pueden seleccionar más de un elemento. Si solo se elige un elemento, se puede controlar con **SelectedItem.** Si se quiere permitir la selección múltiple, en este caso es necesario cambiar el valor de la propiedad **SelectionMode** a **Extended**. En este caso será necesario evaluar **SelectedItems**.

3. DATAGRID

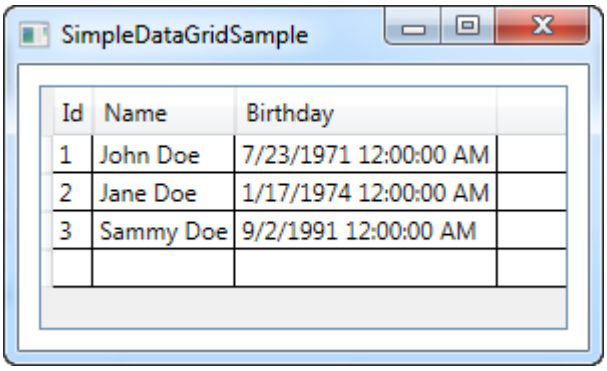
El control DataGrid es parecido al ListView, cuando usamos un GridView, pero ofrece una funcionalidad adicional. Por Ejemplo, el DataGrid puede generar columnas automáticamente, dependiendo de los datos que se le proporciona. El DataGrid también se puede editar de forma predeterminada, lo que permite al usuario final cambiar los valores sobre el origen de los datos.

El uso más común de los DataGrid es en combinación con una base de datos.

Para empezar, vamos a crear un DataGrid sin establecer ninguna propiedad, porque el control está listo para usarse sin necesidad de ninguna configuración previa. En este primer ejemplo, haremos justo eso, y luego asignaremos nuestra propia lista de objetos de Usuario como el origen de los objetos.

```
    Title="SimpleDataGridSample" Height="180" Width="300">
    <Grid Margin="10">
        <DataGrid Name="dgSimple"></DataGrid>
    </Grid>
</Window>
```

```
namespace Nombre de tu proyecto WPF
{
    public partial class SimpleDataGridSample : Window
    {
        public SimpleDataGridSample()
        {
            InitializeComponent();

            List<User> users = new List<User>();
            users.Add(new User() { Id = 1, Name = "John Doe", Birthday = new DateTime(1971, 7, 23) });
            users.Add(new User() { Id = 2, Name = "Jane Doe", Birthday = new DateTime(1974, 1, 17) });
            users.Add(new User() { Id = 3, Name = "Sammy Doe", Birthday = new DateTime(1991, 9, 2) });

            dgSimple.ItemsSource = users;
        }
    }

    public class User
    {
        public int Id { get; set; }

        public string Name { get; set; }

        public DateTime Birthday { get; set; }
    }
}
```



La fuente podría haber sido fácilmente una tabla / vista de base de datos o incluso un archivo XML: DataGrid no es exigente acerca de dónde obtiene sus datos. Si haces clic dentro de una de las celdas, puede ver que tiene permiso para editar cada una de las propiedades de forma predeterminada. También puedes probar haciendo clic en uno de los encabezados de columna, verás que DataGrid admite la ordenación de inmediato.

La última fila vacía te permitirá agregar a la fuente de datos, simplemente completando las celdas.