Escuela Politécnica Superior de Alicante

OFICIAL Máster Web

Programación Avanzada de entornos de escritorio

Taller 6: Aplicaciónes WPF: Diseño y patrones básicos.

Índice

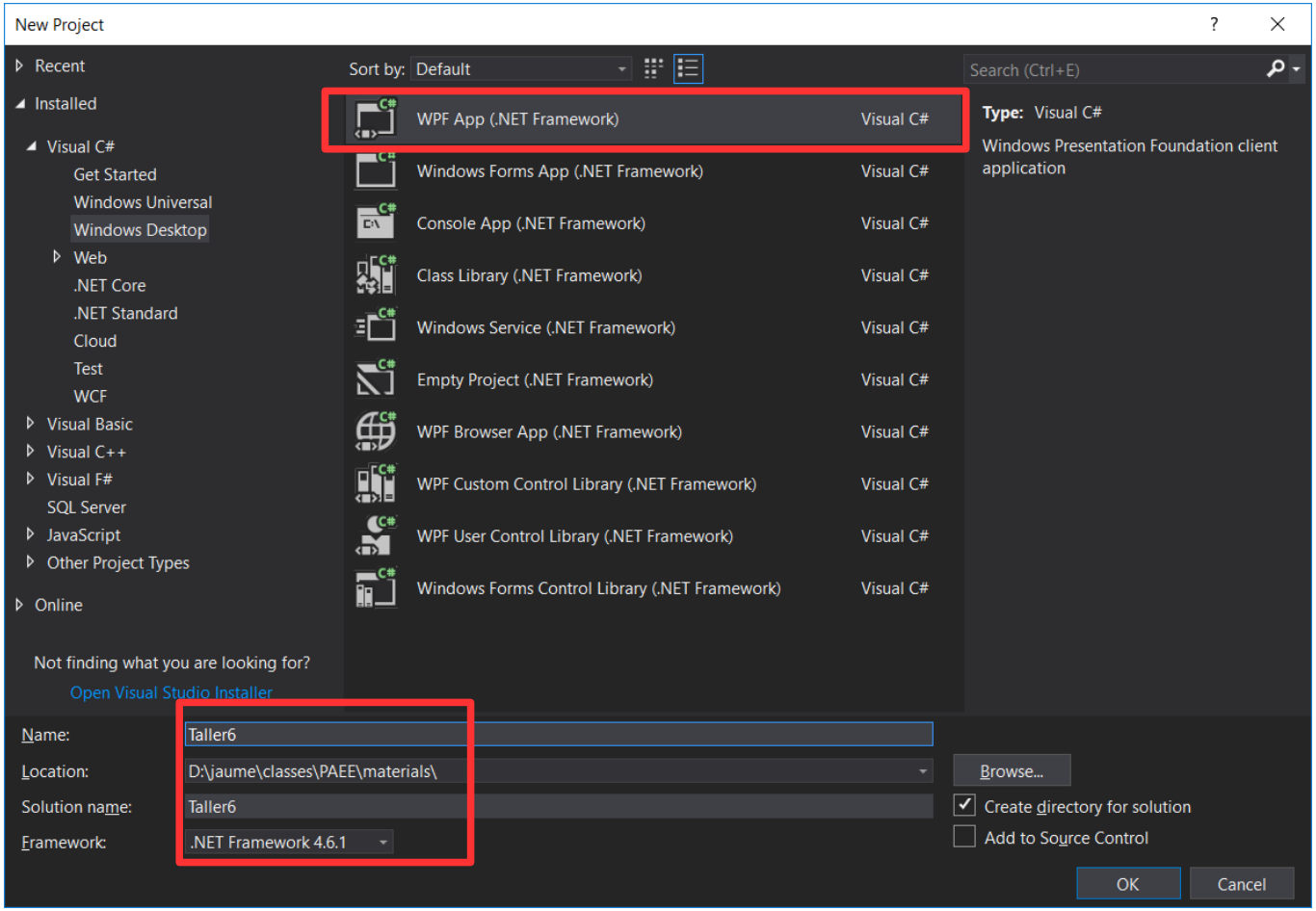
- Proyecto WPF
- Páginas y ventanas
- Acceso a datos por medio de un modelo .cs
- Listados básicos
  - Generar controles en tiempo de ejecución
- User Controls
- Patrones de listados y tablas
- Ejercicios de ampliación

Aplicaciones WPF

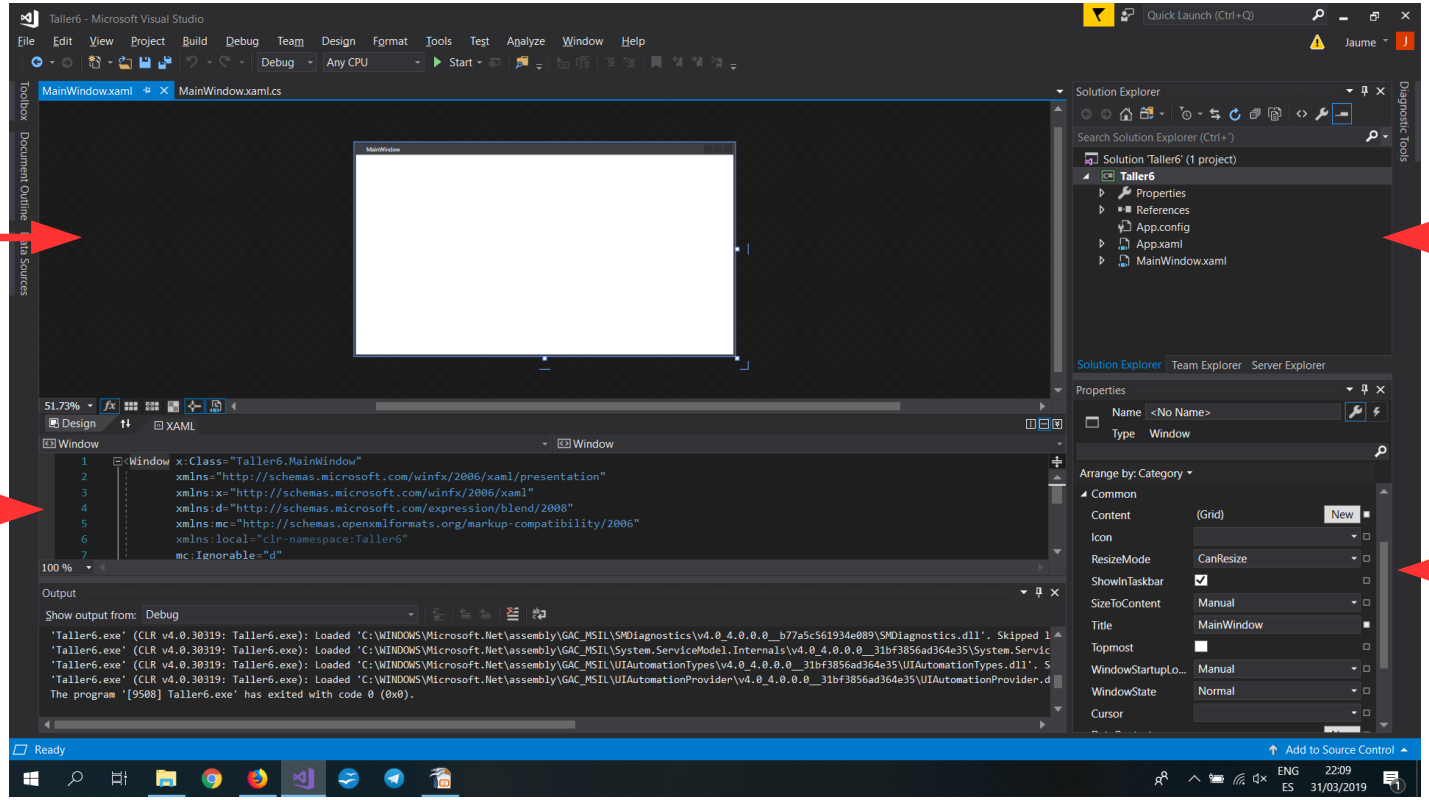
- WPF: Windows Presentation Foundation.
- Es un sistema gráfico de Microsoft diseñado para representar aplicaciones para Windows.
- Usa directX como librería gráfica.
- Utiliza el lenguaje de representación gráfica XAML (Extensible Application Marckup Language)
- Permite gestionar de forma uniforme y centralizada gran cantidad de objetos gráficos: elementos gráficos de windows, render 2D y 3D, tipografía, animaciones e incluso gráficos vectoriales.

Creación del proyecto

- Creamos un nuevo proyecto llamado 'Taller6'



Entorno proyecto WPF



Diseñador

Vista XAML

Explorador

Propie-dades

Creación del proyecto

- Personalizamos el proyecto:
- Renombramos su ventana principal como 'principal', poniendo 'Taller 6' como título.
- Crearemos un botón para salir de la aplicación:

```
<Button x:Name="btnSalir" Content="Salir"
HorizontalAlignment="Left" Margin="677,372,0,0"
VerticalAlignment="Top" Width="97" Height="38"
Background="Red" Foreground="#FF000033"
FontWeight="Bold"/>
```

Creación del proyecto

- El evento de click del botón tendrá el siguiente código:

```
private void btnSalir_Click (object sender,
RoutedEventArgs e) {

   MessageBoxResult res = MessageBox.Show("¿Estás
seguro que deseas salir?", "Atención",

   MessageBoxButton.YesNo, MessageBoxImage.Warning);

   if (res==MessageBoxResult.Yes) {

       Application.Current.Shutdown();

   }

}
```

Accediendo a datos: listado simple

- Ubicaremos dentro de él un botón con las siguientes características:

```
<Button x:Name="btnListado1" Content="Listado simple"
HorizontalAlignment="Left" Margin="75,175,0,0"
VerticalAlignment="Top" Width="150" Height="150"
Background="#00FF00" FontSize="24" FontWeight="Bold">
```

  - Al hacer click sobre él llamaremos a una segunda ventana: winListado1, título: 'Listado simple'.

```
winListado1 listado1 = new winListado1();

listado1.Show();
```

Accediendo a datos: listado simple

- Con la nueva ventana winListado1:
  - Añadiremos una etiqueta 'Listado Simple', arriba.

```
<Label x:Name="lblTitulo" Content="Listado simple"
HorizontalAlignment="Left" Margin="21,10,0,0"
VerticalAlignment="Top" FontSize="24"/>
```

  - Abajo, ubicaremos un boton 'btnCerrar' que cierre la ventana.

```
<Button x:Name="btnCerrar" Content="Cerrar"
HorizontalAlignment="Left" Margin="677,372,0,0"
VerticalAlignment="Top" Width="97" Height="38"
Background="Red" Foreground="#FF000033" FontWeight="Bold"
Click="btnCerrar_Click"/>
```

  - Y en el centro de la ventana añadiremos un panel de tipo 'Canvas', que llamaremos 'pnlDatos'.

```
<Canvas x:Name="pnlDatos" HorizontalAlignment="Left"
Height="315" Margin="5,52,0,0" VerticalAlignment="Top"
Width="779"/>
```

Accediendo a datos: listado simple

- Vamos a mostrar un listado sencillo de provincias a partir de una fuente de datos local en formato Json.
- El listado está en:
  - https://raw.githubusercontent.com/IagoLast/ pselect/master/data/provincias.json
- Descargamos el contenido y lo guardamos en un fichero en la carpeta del proyecto: provincias.json.
- Para acceder a los datos, vamos a implementar un Interface que contenga la estructura básica de acceso a este tipo de ficheros.
- Heredando de este interface, después crearemos una clase que acceda a un fichero Json y almacene su contenido en una colección o lista para su uso posterior.
- Ubicaremos el Interface y clase con el driver en una nueva carpeta que llamaremos 'Models'.

Accediendo a datos: listado simple

- El interface 'IDataDriver' contendrá:

```
interface IDataDriver {
   int getTotal();  // obtener el total de datos
   Int getTotalKeys(); // obtener total de columnas
   string getKey(int indice);  // obtener un nombre de campo
   dynamic getDato(int indice); // leer un dato
   bool setDato(int indice, dynamic dato); // escribir
   bool loadData(); // cargar y almacenar datos del fichero
   bool saveData();
   bool hayError(); // gestión de errores
   string getError();
}
```

Driver de acceso a datos

- Ahora crearemos una clase que hará las funciones de un driver de acceso a fuentes de datos en formato Json, ya sea en ficheros locales como en URLs remotas.
- Esta clase implementará el interface 'IDataDriver' y por tanto, todos sus métodos.
- En una primera versión solo realizaremos la carga de datos y las operaciones de lectura.
- El origen de datos podrá ser tanto un fichero local como un recurso en la web (url). Crearemos una variable para almacenar este dato: 'local'.
- IMPORTANTE: es necesario instalar 'Newtonsoft.Json' del administrador de paquetes Nuget.

Driver de acceso a datos

- La clase, atributos y constructor:

```
class drvJSON: IDataDriver
{

    public string origen { get; set; }

    private List<dynamic> datos;

    private List<string> keys;

    private bool error;

    private string errorMsg;

    public bool esLocal { get; set;}

    public drvJSON()
    {
        origen = "";
        datos = new List<dynamic>();
        keys = new List<string>();
        error = false;
        errorMsg = "";
        esLocal = true;
    }
```

Driver de acceso a datos

- Métodos básicos y que quedan sin hacer en esta versión:

```
public bool hayError() { return error; }

public string getError() { return errorMsg; }

public int getTotal() { return datos.Count(); }

public int getTotalKeys() { return keys.Count(); }

public dynamic getDato(int indice)
{
    return datos[indice];
}

public string getKey(int indice)
{
    return keys[indice];
}

public bool setDato(int indice, dynamic dato) {
    bool res = true;
    //TODO...
    return res;
}

public bool saveData()
{
    bool res = true;
    //TODO...
    return res;
}
```

Driver de acceso a datos

- Método 'loadData':

```
public bool loadData()
{
    bool res = true;
    string cadena = "";
    if (origen != "") {
        if (origen.StartsWith("http")) {
            esLocal = false;
            using (WebClient wc = new WebClient()) {
                cadena = wc.DownloadString(origen);
            }
        } else {
            esLocal = true;
            cadena = File.ReadAllText(origen);
        }

        dynamic data = JsonConvert.DeserializeObject(cadena);
        foreach (dynamic item in data) {
            Newtonsoft.Json.Linq.JObject aux = (Newtonsoft.Json.Linq.JObject)item;
            foreach (Newtonsoft.Json.Linq.JProperty auxItem in aux.Children()) {
                if (!keys.Exists(k => k.Equals(auxItem.Name))) {
                    keys.Add(auxItem.Name);
                }
            }
            datos.Add(item);
        }
    } else {
        res = false;
    }
    return res;
}
```

Accediendo a datos: listado simple

- Ahora, al construir la ventana winListado1, vamos a acceder a los datos y generar los elementos necesarios para mostrarlos.

```
public winListado1()
{
    InitializeComponent();

    drvJSON provs = new drvJSON();
    provs.origen = "ruta a los datos\\provincias.json";
    provs.loadData();
    for (int i = 0; i < provs.getTotal(); i++)
    {
        Label label = new Label
        {
            Name = "label_" + (i + 1).ToString(),
            Width = Double.NaN, //label.Autosize = true;
            Height = 26,
            Content = provs.getDato(i)[provs.getKey(0)] + " - " + provs.getDato(i)[provs.getKey(1)],
            Foreground = new SolidColorBrush(Color.FromRgb(0, 0, 0x33)),
            Background = new SolidColorBrush(Colors.White),
            Margin = new Thickness(50, i * 26, 0, 0)
        };
        pnlDatos.Height += 26;
        pnlDatos.Children.Add(label);
    }
}
```

Accediendo a datos: listado simple

- Para habilitar el scroll y poder acceder a todos los datos del listado, debemos cambiar la vista añadiendo un scrollViewer:

```
<ScrollViewer
ScrollViewer.HorizontalScrollBarVisibility="Auto"
ScrollViewer.VerticalScrollBarVisibility="Auto"
Margin="10,52,9.6,53">

   <Canvas x:Name="pnlDatos" HorizontalAlignment="Left"

   Height="26" VerticalAlignment="Top" Width="707"/>

</ScrollViewer>
```

Accediendo a datos: user control

- Vamos a mejorar el listado creando un control de usuario con las siguientes características:
  - Un título del listado
  - Para cada dato:
    - una etiqueta con datos resumidos
    - y un botón para acceder al dato (vista detalle)
- El primer paso será:
  - Añadir un nuevo botón en la ventana principal: 'Segundo listado' que permita acceder a
  - Una nueva ventana 'winListado2' con un canvas y el botón de 'volver'.

XAML: Estilos

- Antes de crear el nuevo botón, vamos a crear un estilo visual con XAML y aplicaremos este estilo a los botones de la ventana principal.
- Primero debemos crear un nuevo fichero en el proyecto de tipo 'Resource Dictionary' que llamaremos 'Estilos.xaml'.
- En el insertaremos el estilo que deseamos para los botones de la pantalla principal.

XAML: Estilos

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:local="clr-namespace:Taller6">
<Style x:Key="BotonPrincipal" TargetType="Button">
    <Setter Property="Margin" Value="2"/>
    <Setter Property="FontFamily" Value="Verdana"/>
    <Setter Property="FontSize" Value="24px"/>
    <Setter Property="FontWeight" Value="Bold"/>
    <Setter Property="Background" Value="#00FF00" />
    <Setter Property="BorderBrush" Value="#00FF00" />
    <Setter Property="Width" Value="150" />
    <Setter Property="Height" Value="150" />
</Style>
</ResourceDictionary>
```

XAML: Estilos

- A continuación, modificaremos el fichero 'App.xaml' para incluir una referencia al nuevo diccionario de estilos.

```
<Application x:Class="Taller6.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:local="clr-namespace:Taller6"
             StartupUri="MainWindow.xaml">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Estilos.xaml"/>
            </ResourceDictionary.MergedDictionaries>
            <!--Put all previous resources in the App.xaml here-->
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

XAML: Estilos

- Finalmente, añadimos el nuevo botón 'btnListado2' y aplicamos el nuevo estilo a ambos botones, quedando el código XAML como sigue:

```
<Button x:Name="btnListado1" HorizontalAlignment="Left"
Margin="75,175,0,0" VerticalAlignment="Top"
Style="{StaticResource BotonPrincipal}"
Click="btnListado1_Click" Content="Listado simple" />


<Button x:Name="btnListado2" HorizontalAlignment="Left"
Margin="254,175,0,0" VerticalAlignment="Top"
Style="{StaticResource BotonPrincipal}"
Content="Listado con detalle" />
```

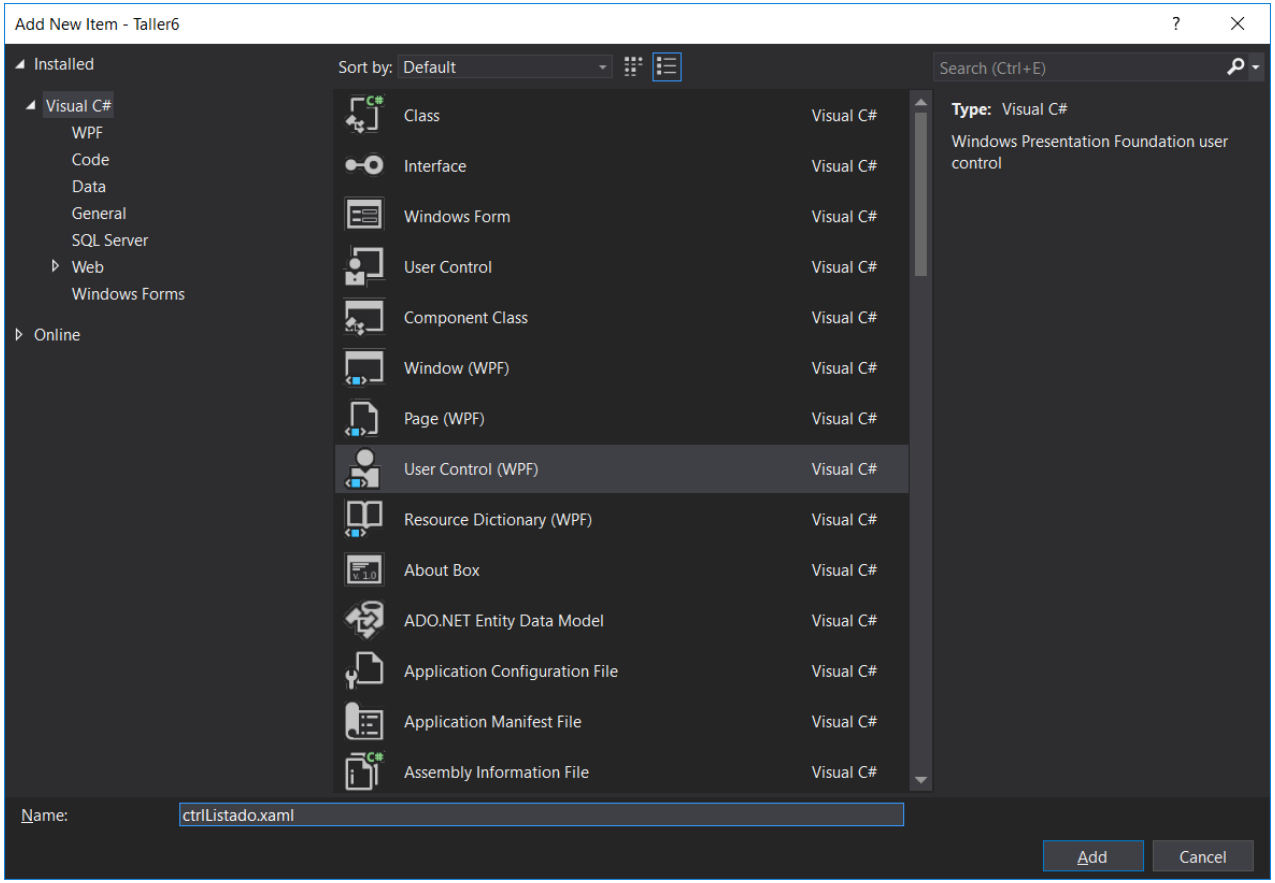
Accediendo a datos: user control

- A continuación, crearemos una nueva ventana 'winListado2', la cual contendrá un canvas 'pnlDatos' que ocupe toda la ventana.
- En la parte inferior derecha, ubicaremos un botón 'btnCerrar' de igual forma que hicimos con la ventana 'winListado1'.

```
...Title="Listado con detalle" Height="450" Width="800">

<Grid>

  <Button x:Name="btnCerrar" Content="Cerrar"
HorizontalAlignment="Left" Margin="677,372,0,0"
VerticalAlignment="Top" Width="97" Height="38" Background="Red"
Foreground="#FF000033" FontWeight="Bold"
Click="btnCerrar_Click"/>

  <Canvas x:Name="pnlListado" HorizontalAlignment="Left"
Height="60" VerticalAlignment="Top" Width="774"
Margin="10,10,0,0"/></Grid>
```

Accediendo a datos: user control

- Ahora vamos a crear un control de usuario 'ctrlListado.xaml':



Definiendo el user control

- Cambiamos las dimensiones y el color de fondo del control.
- Le añadimos una etiqueta para el título.
- Además ponemos un ScrollViewer y un Canvas dentro.

```
<UserControl x:Class="Taller6.ctrlListado"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:Taller6"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="774"
             Background="White">
    <Grid>
        <Label x:Name="lblTitulo" Content="Título" HorizontalAlignment="Left" Margin="21,10,0,0"
               VerticalAlignment="Top" FontSize="24"/>
        <ScrollViewer ScrollViewer.HorizontalScrollBarVisibility="Disabled" ScrollViewer.VerticalScrollBarVisibility="Auto"
                      Margin="10,52,9.6,53">
            <Canvas x:Name="pnlDatos" HorizontalAlignment="Left" Height="25" VerticalAlignment="Top" Width="750"/>
        </ScrollViewer>

    </Grid>
</UserControl>
```

Definiendo el user control

- Definimos dos propiedades: titulo y origen basadas en atributos privados
- Añadimos una variable provada datos que sera un objeto de la clase drvJSON
- En el constructor inicializamos las variables

```
public partial class ctrlListado : UserControl
{
    private string _titulo;
    public string titulo
    {
        get { return _titulo; }
        set { _titulo = value; lblTitulo.Content=_titulo; }
    }

    private string _origen;
    public string origen {
        get { return _origen; }
        set { _origen = value; loadData(); }
    }

    private drvJSON datos { get; set; }

    public ctrlListado()
    {
        InitializeComponent();

        titulo = "Título";
        datos = new drvJSON();
    }
```

Definiendo el user control

- El método loadData realizará la carga de los datos desde el origen
- Y creará los objetos necesarios para mostrar los datos.

```
public void loadData()
{
    if (_origen != "")
    {
        datos.origen = _origen;
        datos.loadData();

        for (int i = 0; i < datos.getTotal(); i++)
        {
            SolidColorBrush bgcolor;
            if (i % 2 == 0)
            {
                bgcolor = new SolidColorBrush(Colors.White);
            }
            else
            {
                bgcolor = new SolidColorBrush(Colors.LightGray);
            }
            Label label = new Label
            {
                Name = "label_" + (i + 1).ToString(),
                Width = Double.NaN, //label.Autosize = true;
                Height = 26,
                Content = datos.getDato(i)[datos.getKey(0)] + " - " + datos.getDato(i)[datos.getKey(1)],
                Foreground = new SolidColorBrush(Color.FromRgb(0, 0, 0x33)),
                Background = bgcolor,
                Margin = new Thickness(50, i * 26, 0, 0)
            };
            pnlDatos.Height += 26;
            pnlDatos.Children.Add(label);
        }
    }
}
```

Accediendo a datos: user control

- Finalmente, modificaremos la ventana winListado2:
- Incluiremos el control de usuario:

```
<Canvas x:Name="pnlListado"
HorizontalAlignment="Left" Height="357"
VerticalAlignment="Top" Width="774"
Margin="10,10,0,0">

            <local:ctrlListado x:Name="listado"
Height="357" Width="774"></local:ctrlListado>

</Canvas>
```

Accediendo a datos: user control

- Finalmente, modificaremos la ventana winListado2:
- El constructor de winListado2 será:

```
public winListado2() {
    InitializeComponent();
    listado.titulo = "Provincias";
    listado.origen = "ruta_a_datos\\provincias.json";
}
```

User control: botón de detalle

- Para añadir un botón que muestre el detalle del dato sobre el que hacemos click:
  - deberemos crear un Button en cada iteración del bucle for de la función 'loadData'
  - Además, el botón lo deberemos añadir como hijo del canvas 'pnlDatos'
  - Vincular el evento del botón a una nueva función que muestra en un messageBox la información del dato en cuestión: btnDetalle_click.

User control: botón de detalle

- Cambios en la función 'loadData'.

```
for (int i = 0; i < datos.getTotal(); i++)
{
    SolidColorBrush bgcolor;
    if (i % 2 == 0)
    {
        bgcolor = new SolidColorBrush(Colors.White);
    }
    else
    {
        bgcolor = new SolidColorBrush(Colors.LightGray);
    }
    Label label = new Label
    {
        Name = "label_" + (i + 1).ToString(),
        Width = 600,
        Height = 26,
        Content = datos.getDato(i)[datos.getKey(0)] + " - " + datos.getDato(i)[datos.getKey(1)],
        Foreground = new SolidColorBrush(Color.FromRgb(0, 0, 0x33)),
        Background = bgcolor,
        Margin = new Thickness(50, i * 26, 0, 0)
    };

    Button boton = new Button {
        Content = "Detalles",
        Name = "btn_" + (i + 1).ToString(),
        Width = 70,
        Height = 22,
        Margin = new Thickness(655, (i*26)+2, 0, 0),
        Background = bgcolor
    };

    int id = i;
    boton.Click += (sender, EventArgs) => { btnDetalle_Click(sender, EventArgs, id); };

    pnlDatos.Height += 26;
    pnlDatos.Children.Add(label);
    pnlDatos.Children.Add(boton);
}
```

User control: botón de detalle

- El manejador para el evento de click sobre los botones de 'detalle'.

```
private void btnDetalle_Click(object sender, EventArgs e, int id)
{

    string info = datos.getKey(0) + ": " + datos.getDato(id)[datos.getKey(0)];
    info += "\n" + datos.getKey(1) + ": " + datos.getDato(id)[datos.getKey(1)];
    MessageBox.Show(info,
        "Detalle del dato: " + datos.getDato(id)[datos.getKey(0)],
        MessageBoxButton.OK,
        MessageBoxImage.Information);
}
```

Accediendo a datos: listado tabular

- Vamos a crear ahora un listado tabular aplicando algunos patrones de listados tabulares.
- Para ello crearemos un nuevo control de usuario que mostrará los datos por medio de una tabla y permitira filtrar y ordenar.
- Para acceder al nuevo control usaremos un tercer botón desde la ventana principal.

Accediendo a datos: listado tabular

- El nuevo user control, que llamaremos ctrlTabla contendrá:
  - Un título
  - Una fila de cabecera con los nombres de las columnas.
  - Una fila por cada elemento encontrado en el origen de datos con una columna por dato.
  - Una columna adicional con el encabezado vacio y servirá para mostrar botones de acciones individuales sobre los datos.
  - Tendrá la misma lógica que el anterior control ctrlListado para gestionar el título, la carga de datos y el acceso al origen de los mismos.

Accediendo a datos: listado tabular

- El control de usuario 'ctrlTabla':

```
<UserControl x:Class="Taller6.ctrlTabla"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:Taller6"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="774"
             Background="White">

    <Grid>
        <Label x:Name="lblTitulo" Content="Título" HorizontalAlignment="Left"
               Margin="21,10,0,0" VerticalAlignment="Top" FontSize="24"/>
        <ScrollViewer ScrollViewer.HorizontalScrollBarVisibility="Disabled"
                      ScrollViewer.VerticalScrollBarVisibility="Auto" Margin="10,52,9.6,53">
            <Canvas x:Name="pnlDatos" HorizontalAlignment="Left" Height="25"
                    VerticalAlignment="Top" Width="750"/>
        </ScrollViewer>

    </Grid>

</UserControl>
```

Accediendo a datos: listado tabular

- El código del control de usuario 'ctrlTabla':

```
public partial class ctrlTabla : UserControl
{
    private string _titulo;
    public string titulo
    {
        get { return _titulo; }
        set { _titulo = value; lblTitulo.Content = _titulo; }
    }

    private string _origen;
    public string origen
    {
        get { return _origen; }
        set { _origen = value; loadData(); }
    }

    private drvJSON datos { get; set; }

    public ctrlTabla()
    {
        InitializeComponent();

        titulo = "Título";
        datos = new drvJSON();
    }
```

Accediendo a datos: listado tabular

- La función 'loadData':
  - Obtendrá los datos del fichero indicado en 'origen'.
  - Mostrará los encabezados de las columnas a partir de un bucle que recorrerá todas las claves encontradas.
  - Añadirá un encabezado adicional sin texto
  - Después, mediante dos bucles anidados recorreremos todas las tuplas y para cada una un segundo bucle recorrerá todas las claves y mostrará un 'label' por dato encontrado.
    - Al final del bucle interior, añadiremos el botón para acceder al detalle del dato.

Accediendo a datos: listado tabular

- La función 'loadData'.
- Código que muestra la cabecera de la tabla de datos.

```
public void loadData()
{
    if (origen != "")
    {
        datos.origen = origen;
        datos.loadData();
        for (int i = 0; i < datos.getTotalKeys(); i++)
        {
            Label label = new Label
            {
                Name = "lblHeader_" + (i + 1).ToString(),
                Content = datos.getKey(i),
                Margin = new Thickness(i * 100, 0, 0, 0),
                FontFamily=new FontFamily("Arial"),
                FontSize=12.0,
                FontWeight=FontWeights.Bold,
                Background = new SolidColorBrush(Color.FromRgb(0, 0, 0x33)),
                Foreground = new SolidColorBrush(Colors.White),
                Width = 100,
                Height = 24,
                HorizontalContentAlignment=HorizontalAlignment.Center
            };

            pnlDatos.Children.Add(label);
        }

        Label labelOps = new Label
        {
            Name = "lblHeader_Ops",
            Content=" ",
            Width = 100,
            Height = 24,
            Margin = new Thickness(0 + (datos.getTotalKeys() * 95), 0, 0, 0),
            Background = new SolidColorBrush(Color.FromRgb(0, 0, 0x33)),
            Foreground = new SolidColorBrush(Colors.White),
        };

        pnlDatos.Children.Add(labelOps);
```

Accediendo a datos: listado tabular

- La función 'loadData'.
- Código que muestra las filas/columnas con los datos y el botón de la vista detalle.

```
for (int i = 0; i < datos.getTotal(); i++)
{
    for (int j = 0; j < datos.getTotalKeys(); j++)
    {
        SolidColorBrush bgcolor;
        if (i % 2 == 0)
        {
            bgcolor = new SolidColorBrush(Colors.White);
        }
        else
        {
            bgcolor = new SolidColorBrush(Colors.LightGray);
        }
        Label label = new Label {
            Name = "lblData_" + (i + 1).ToString() + "_" + (j + 1).ToString(),
            Width=100,
            Height= 24,
            Content= datos.getDato(i)[datos.getKey(j)],
            Foreground=new SolidColorBrush(Color.FromRgb(0, 0, 0x33)),
            Background=bgcolor,
            Margin = new Thickness(0 + (j * 100), 24 + (i * 24), 0, 0),
            FontFamily = new FontFamily("Arial"),
            FontSize = 12.0,
            BorderThickness = new Thickness(1, 0, 0, 0)
        };
        pnlDatos.Children.Add(label);
    }

    Button boton = new Button {
        Content= "Detalle",
        Name = "btn_" + (i + 1).ToString(),
        Width= 70,
        Height= 22,
        Margin = new Thickness(2 + (datos.getTotalKeys() * 100), 25 + (i * 24), 0, 0),

    };
    pnlDatos.Children.Add(boton);
}
```

Accediendo a datos: listado tabular

- Para permitir el acceso al nuevo listado:
  - En la ventana principal crearemos un nuevo botón 'btnListado3'.

```
<Button x:Name="btnListado3" HorizontalAlignment="Left"
Margin="435,175,0,0" VerticalAlignment="Top"
Style="{StaticResource BotonPrincipal}"
Click="btnListado3_Click" Content="”Listado tabular”/>
```

  - El evento de click:

```
private void btnListado3_Click(object sender,
RoutedEventArgs e) {

   winListado3 listado3 = new winListado3();

   listado3.Show();

}
```

Accediendo a datos: listado tabular

- Creamos la nueva ventana 'winListado3' y la configuramos adecuadamente:

```
<Window x:Class="Taller6.winListado3"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Taller6"
        mc:Ignorable="d"
        Title="Listado tabular" Height="450" Width="800">
    <Grid>
        <Button x:Name="btnCerrar" Content="Cerrar" HorizontalAlignment="Left" Margin="677,372,0,0" VerticalAlignment="Top"
                Width="97" Height="38" Background="Red" Foreground="#FF000033" FontWeight="Bold" Click="btnCerrar_Click"/>
        <Canvas x:Name="pnlListado" HorizontalAlignment="Left" Height="357" VerticalAlignment="Top"
                Width="774" Margin="10,10,0,0">
            <local:ctrlTabla x:Name="tabla" Height="357" Width="774"></local:ctrlTabla>
        </Canvas>
    </Grid>
</Window>
```

Accediendo a datos: listado tabular

- El código de la ventana 'winListado3':

```
public winListado3() {

    InitializeComponent();

    tabla.titulo = "Listado tabular";

    tabla.origen = "ruta a los datos\\provincias.json";

}


private void btnCerrar_Click (object sender,
RoutedEventArgs e) {

    this.Close();

}
```

Accediendo a datos: vista detalle

- La función 'btnDetalle_click':
- La función del botón de detalle, mostrará los datos en un nuevo canvas que contendrá una tabla con dos columnas (en la primera, el nombre del campo y en la segunda el dato).
- Debemos recorrer el vector de claves para mostrar los datos de cada una, así como el nombre del campo correspondiente
- Le añadiremos un título y un botón para cerrarlo y poder volver al listado tabular.

Accediendo a datos: vista detalle

- En el XAML, añadiremos los siguientes elementos:

```
<ScrollViewer x:Name="boxDetalle"
ScrollViewer.HorizontalScrollBarVisibility="Disabled"
ScrollViewer.VerticalScrollBarVisibility="Auto"
Margin="792,52,-772.6,52.6">

   <Canvas x:Name="pnlDetalle" HorizontalAlignment="Left"
Height="25" VerticalAlignment="Top" Width="750"/>

</ScrollViewer>

<Button x:Name="btnVolver" Content="Volver"
HorizontalAlignment="Left" Margin="666,20,0,0"
VerticalAlignment="Top" Width="75" Height="24"
Background="Red" Foreground="#FF000033" FontWeight="Bold"
Click="btnVolver_Click"/>
```

Accediendo a datos: vista detalle

- En el Code Behind del user control, añadiremos un nuevo atributo para poder reutilizar la etiqueta del título. Y en el constructor, ocultaremos los elementos de la vista detalle:

```
private string _oldTitulo;
public ctrlTabla() {
    InitializeComponent();
    btnVolver.Visibility = Visibility.Hidden;
    boxDetalle.Visibility = Visibility.Hidden;
    titulo = "Título";
    datos = new drvJSON();
}
```

Accediendo a datos: vista detalle

- El código de 'btnDetalle_click':

```
private void btnDetalle_Click(object sender, EventArgs e,
int id) {

    _oldTitulo = titulo;

    titulo = "Detalle de ";

    btnVolver.Visibility = Visibility.Visible;

    ScrollViewer aux = (ScrollViewer)pnlDatos.Parent;

    aux.Visibility = Visibility.Hidden;

    // Código para mostrar el detalle del dato seleccionado.

    boxDetalle.Visibility = Visibility.Visible;

    boxDetalle.Margin = new Thickness(10, 52, 9.6, 53);

}
```

Accediendo a datos: vista detalle

- El código de 'btnVolver_click':

```
private void btnVolver_Click (object sender,
RoutedEventArgs e) {

   titulo = _oldTitulo;

   btnVolver.Visibility = Visibility.Hidden;

   ScrollViewer aux = (ScrollViewer)pnlDatos.Parent;

   aux.Visibility = Visibility.Visible;

   boxDetalle.Visibility = Visibility.Hidden;

   boxDetalle.Margin = new

      Thickness (792, 52, -772.6, 52.6);

}
```

Accediendo a datos: vista detalle

- Finalmente, el código necesario para mostrar los datos en la vista detalle:

```
titulo = "Detalle de " + datos.getDato(id)[datos.getKey(1)];
btnVolver.Visibility = Visibility.Visible;
ScrollViewer aux = (ScrollViewer)pnlDatos.Parent;
aux.Visibility = Visibility.Hidden;

pnlDetalle.Children.Clear();

for (int i = 0; i < datos.getTotalKeys(); i++)
{
    Label auxKey = new Label {
        Content = datos.getKey(i),
        FontFamily = new FontFamily("Arial"),
        FontSize = 12,
        FontWeight= FontWeights.Bold,
        Margin = new Thickness(10, (i * 25), 0, 0),
        Width = 200,
        Height=25,
        HorizontalContentAlignment = HorizontalAlignment.Right,
        Background = new SolidColorBrush(Color.FromRgb(0, 0, 0x33)),
        Foreground = new SolidColorBrush(Colors.White),
    };
    pnlDetalle.Children.Add(auxKey);
    Label auxDato = new Label {
        Content = datos.getDato(id)[datos.getKey(i)],
        FontFamily = new FontFamily("Arial"),
        FontSize = 12,
        FontWeight = FontWeights.Regular,
        Margin = new Thickness(210, (i * 25), 0, 0),
        Width = 200,
        Height = 25,
        HorizontalContentAlignment = HorizontalAlignment.Left
    };
    pnlDetalle.Children.Add(auxDato);
}
```

Ejercicios de ampliación

- Modificar el *User control* 'ctrlListado'
  - crea un 'subcontrol de usuario' en el que se encapsule la lógica necesaria para mostrar una fila o tupla individual con sus botones de acción.
- Modificar el *User control* 'ctrlTabular'
  - Implementar el patrón de ordenación de datos al hacer click sobre las cabeceras.
  - Añadir un formulario sencillo de búsqueda encima de la tabla para poder filtrar los datos.

Ejercicios de ampliación

- En el driver de acceso a datos 'drvJSON', añadir código para capturar excepciones y errores usando 'hayError', 'error' y 'errorMsg'.
- En el listado 1 (listado simple), añadir lógica para la gestión de errores.
- En el control de usuario 'ctrlListado', añadir código para tratamiento de errores.