Programación Avanzada de entornos de escritorio

2021-2022

Escuela Politécnica Superior

Universidad de Alicante

Plataforma universal de Windows (UWP)

https://docs.microsoft.com/es-es/windows/uwp/get-started/universal-application-platform-guide

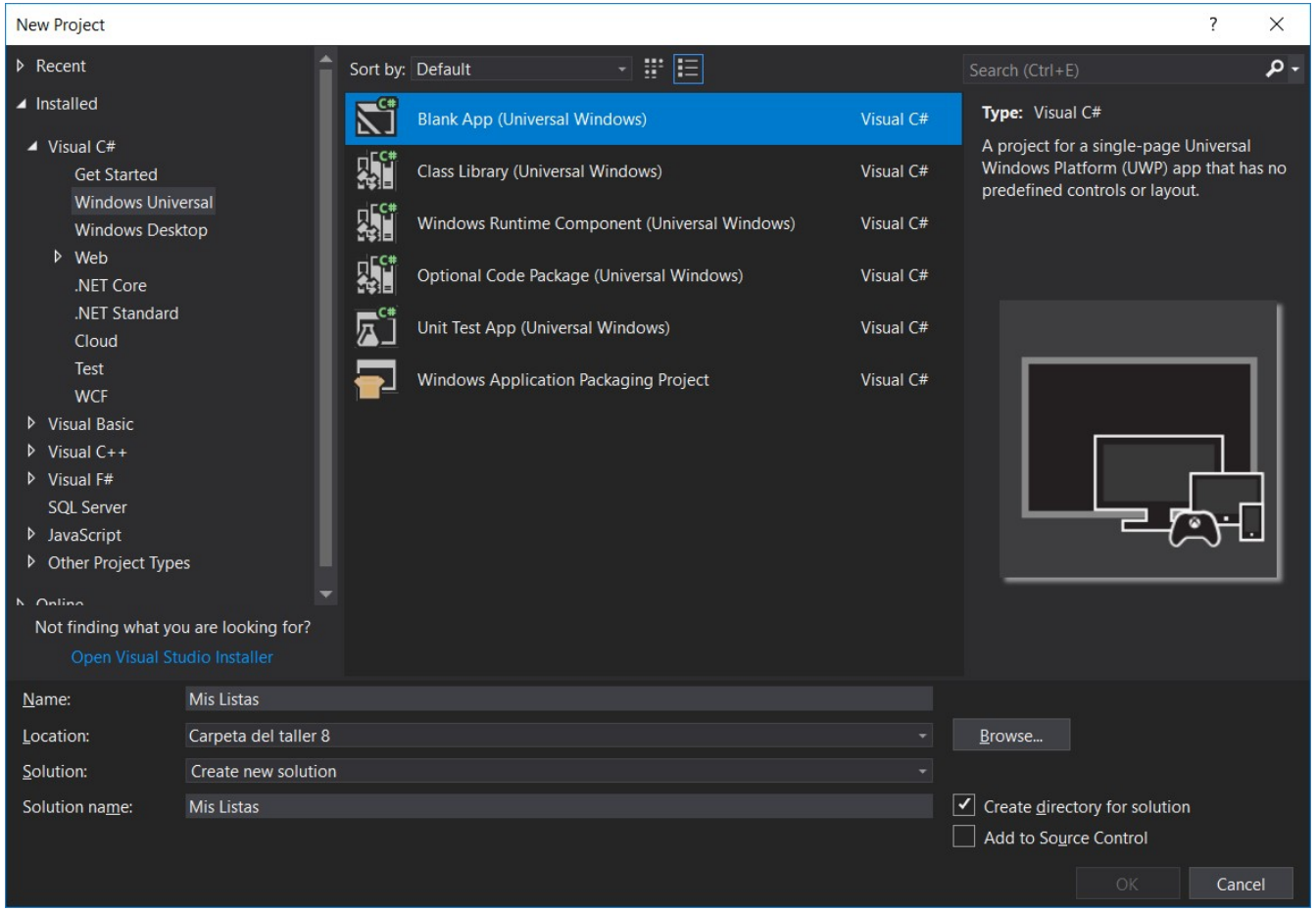
Contenidos

- Crear y configurar el proyecto.
- Personalizar la ventana principal.
  - Captura evento de cierre del programa.
- Añadir controles y manejadores.
- Almacenamiento de datos y acceso a datos locales.
- Otras técnicas/patrones con UWP
- Ampliaciones
- Ejercicios
- Referencias y otras fuentes de información

Captura aplicación del taller finalizada

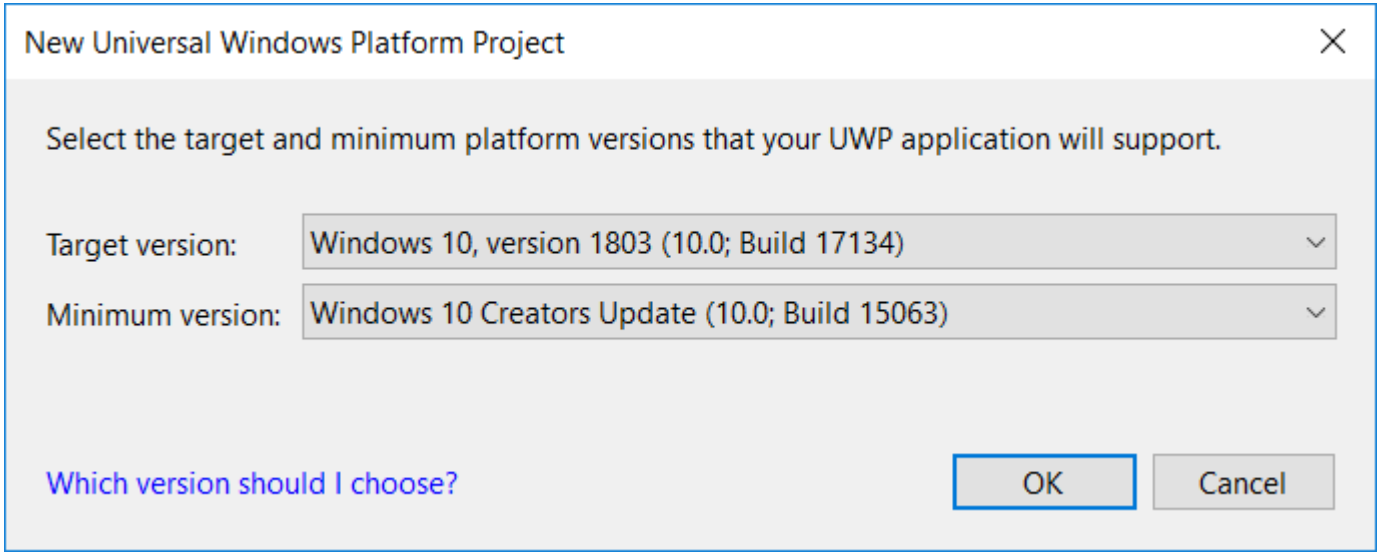
Crear y configurar el proyecto

- Vamos a crear un nuevo proyecto de tipo 'Windows universal', llamado 'Mis Listas'.



Crear y configurar el proyecto

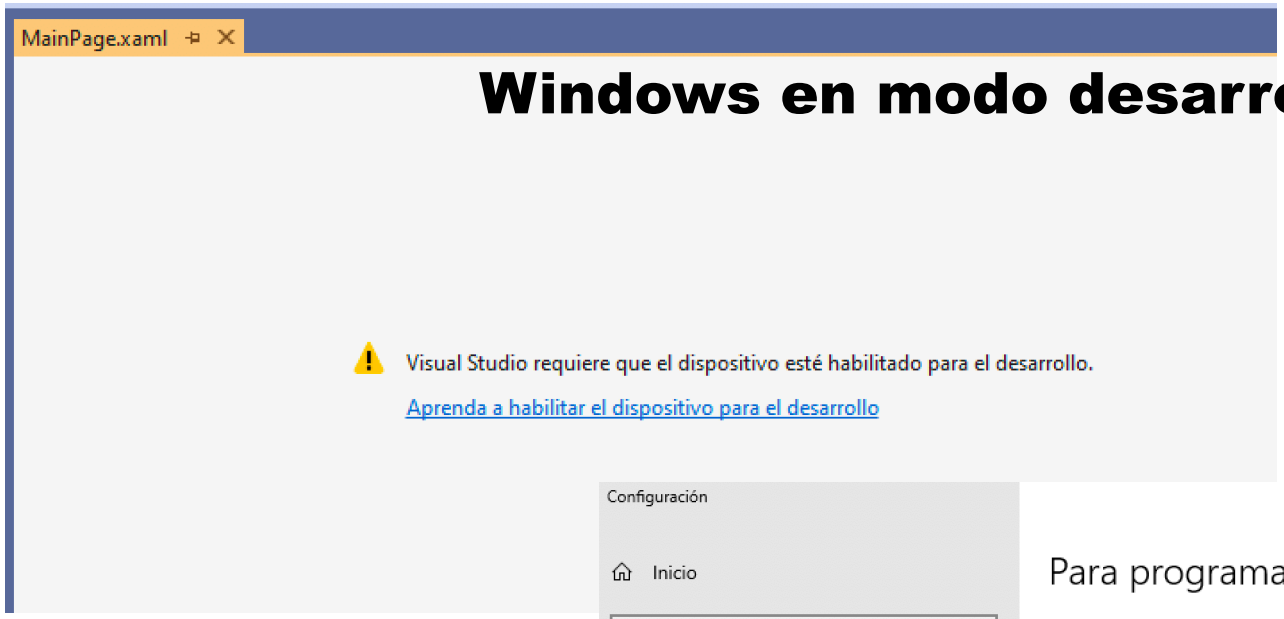
- Seleccionamos la plataforma de destino y la plataforma mínima que de soporte la compilación:

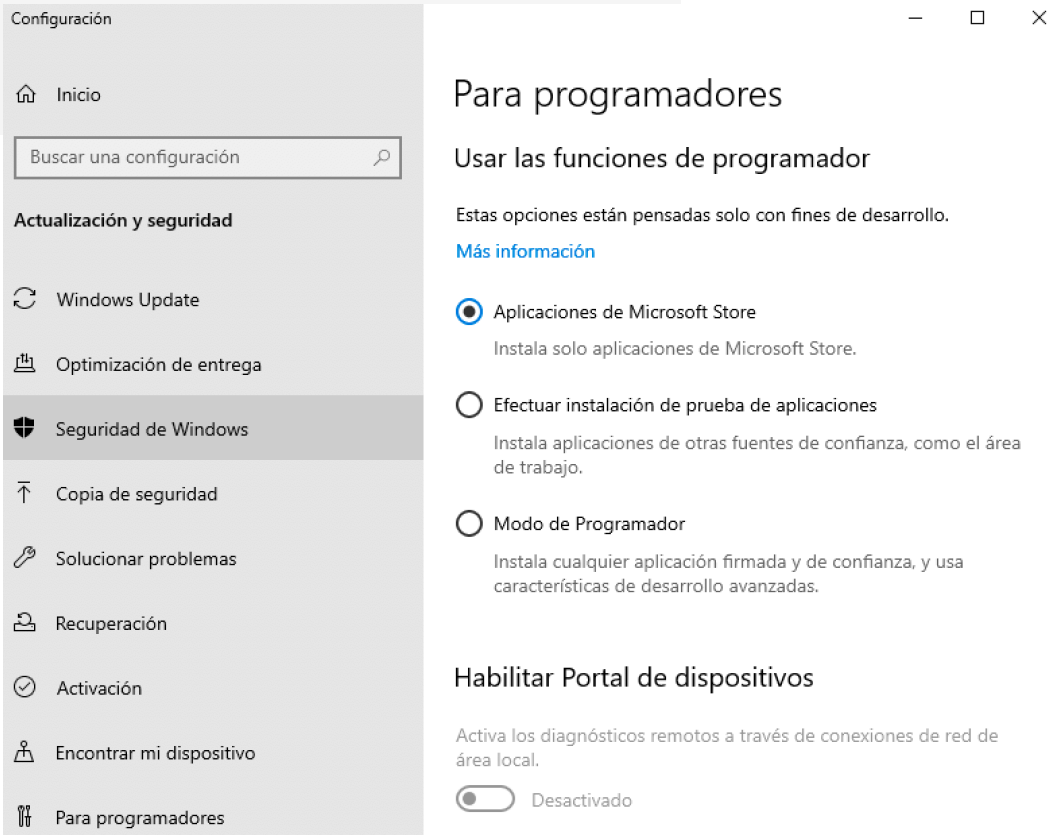


- ¿Qué plataforma escoger? Dependiendo de la plataforma, y versión del Windows dispondremos de unas prestaciones e incompatibilidades diferentes.

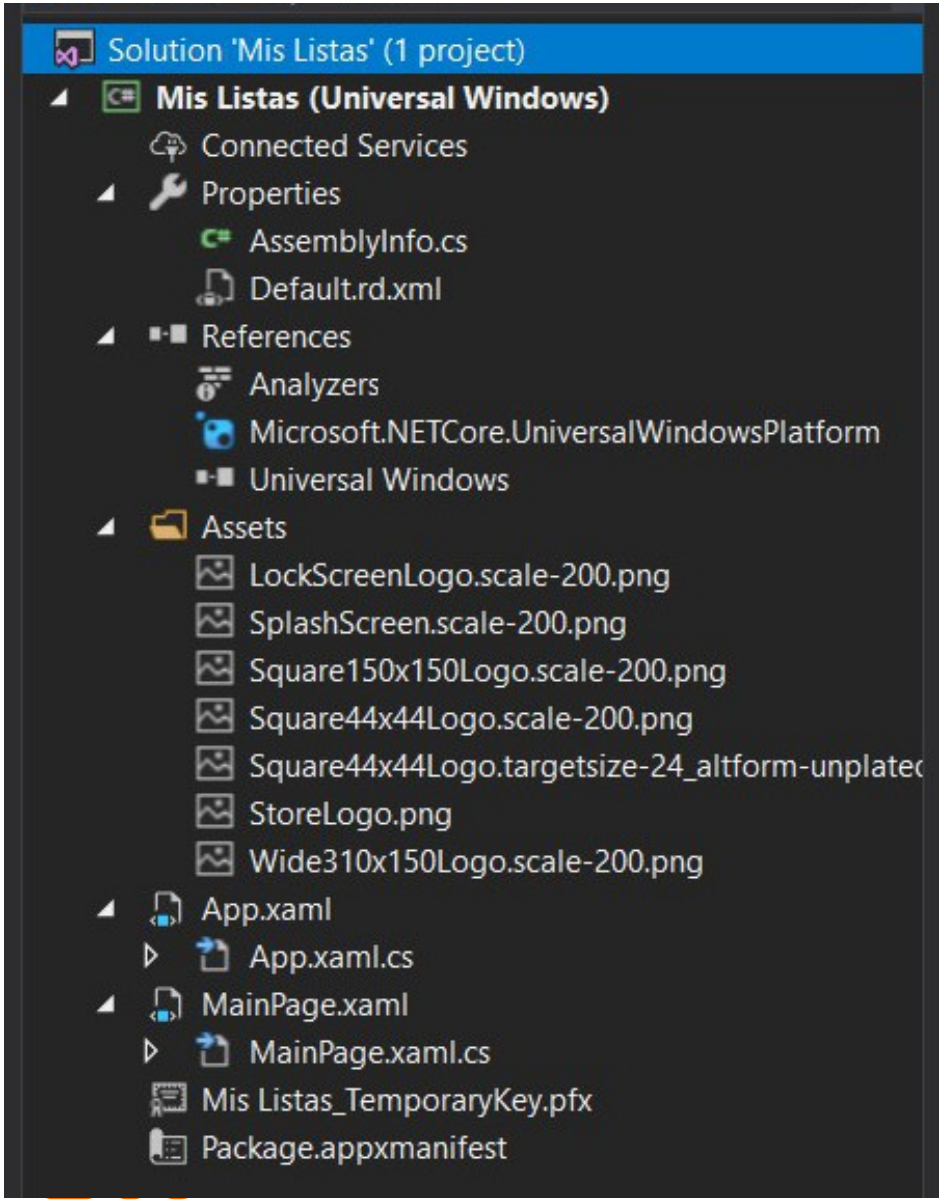
Es conveniente mirar primero la información de las diferentes versiones.

Windows en modo desarrollo



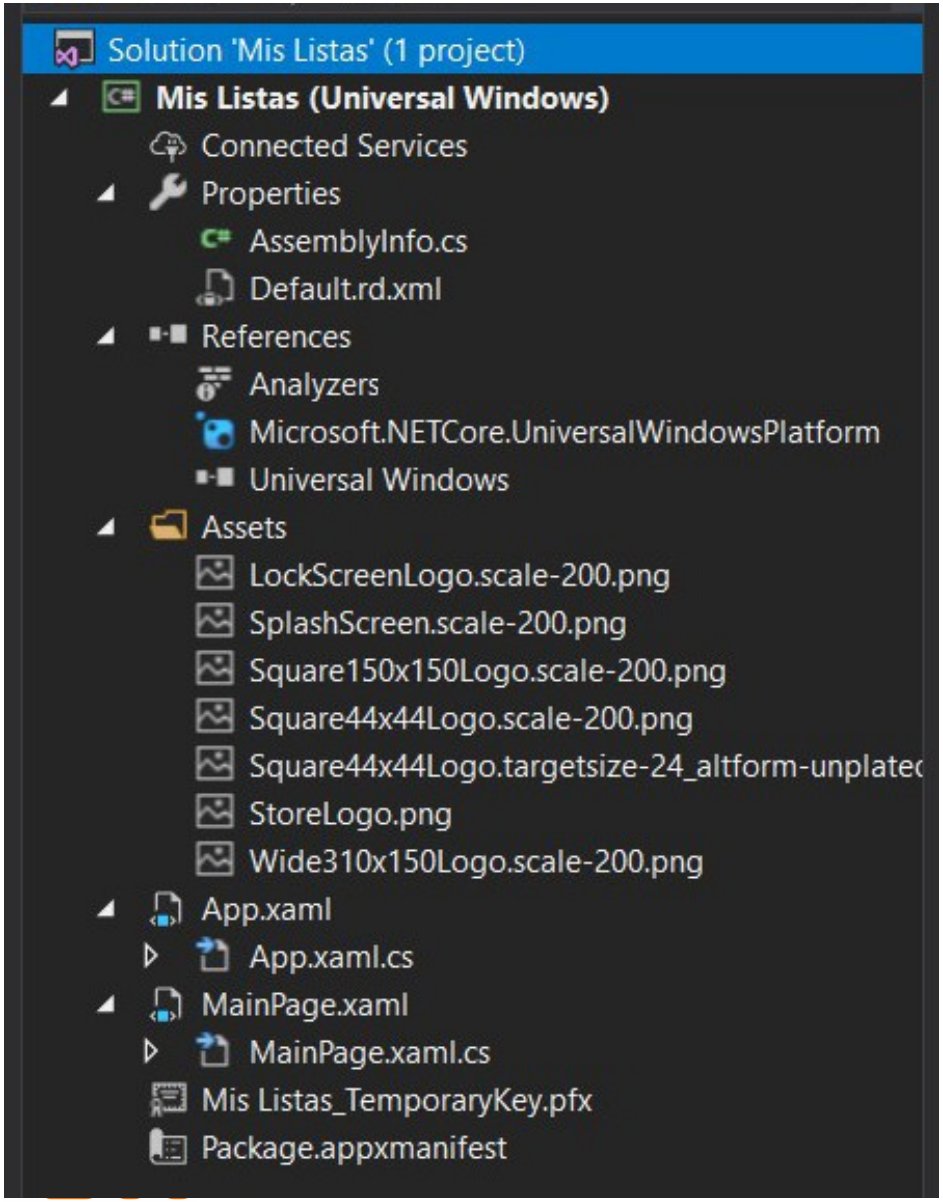


Crear y configurar el proyecto



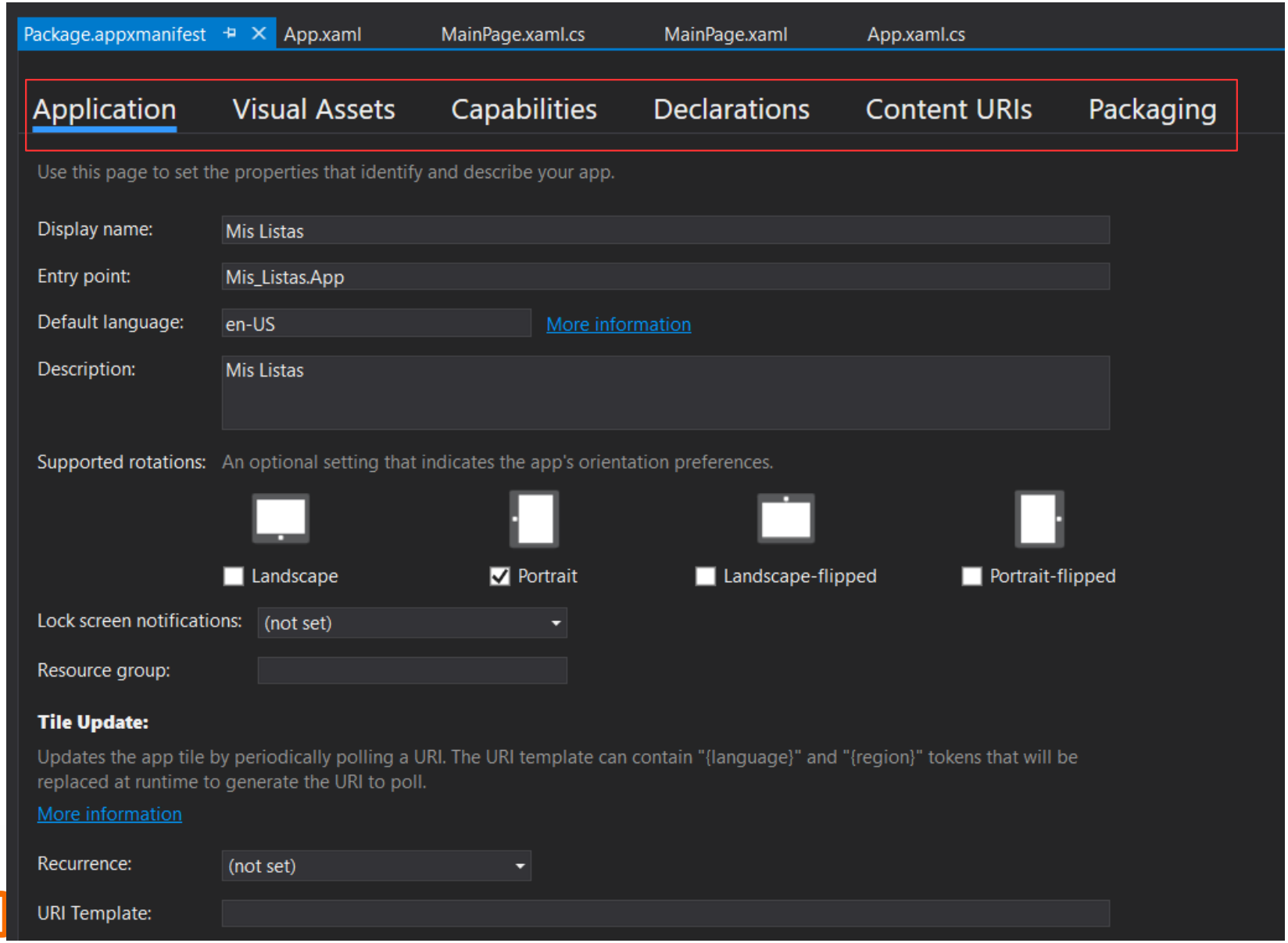
- En un proyecto de windows universal vacío tenemos:
  - AssemblyInfo.cs
    - Información del ensamblado del proyecto.
  - Referencias
  - Assets
    - Directorio donde se almacenaran los recursos estáticos (imágenes, documentos, etc.)
  - App.xaml y code behind
    - Fichero que contiene la clase que representa nuestra aplicación (hereda de Application)

Crear y configurar el proyecto



- En un proyecto de windows universal vacío tenemos:
  - MainPage.xaml y code behind
    - Contiene la página o pantalla principal del aplicativo con su código que hereda del control 'Page'.
  - Fichero .pfx
    - Fichero de intercambio de información personal. Contiene un certificado para firmar el empaquetado de la aplicación en caso de querer distribuirla o subirla al Windows Store.
  - Package.appxmanifest
    - Fichero con información necesaria para la compilación, distribución, visualización y actualización del proyecto.

Package.appxmanifest



Package.appxmanifest

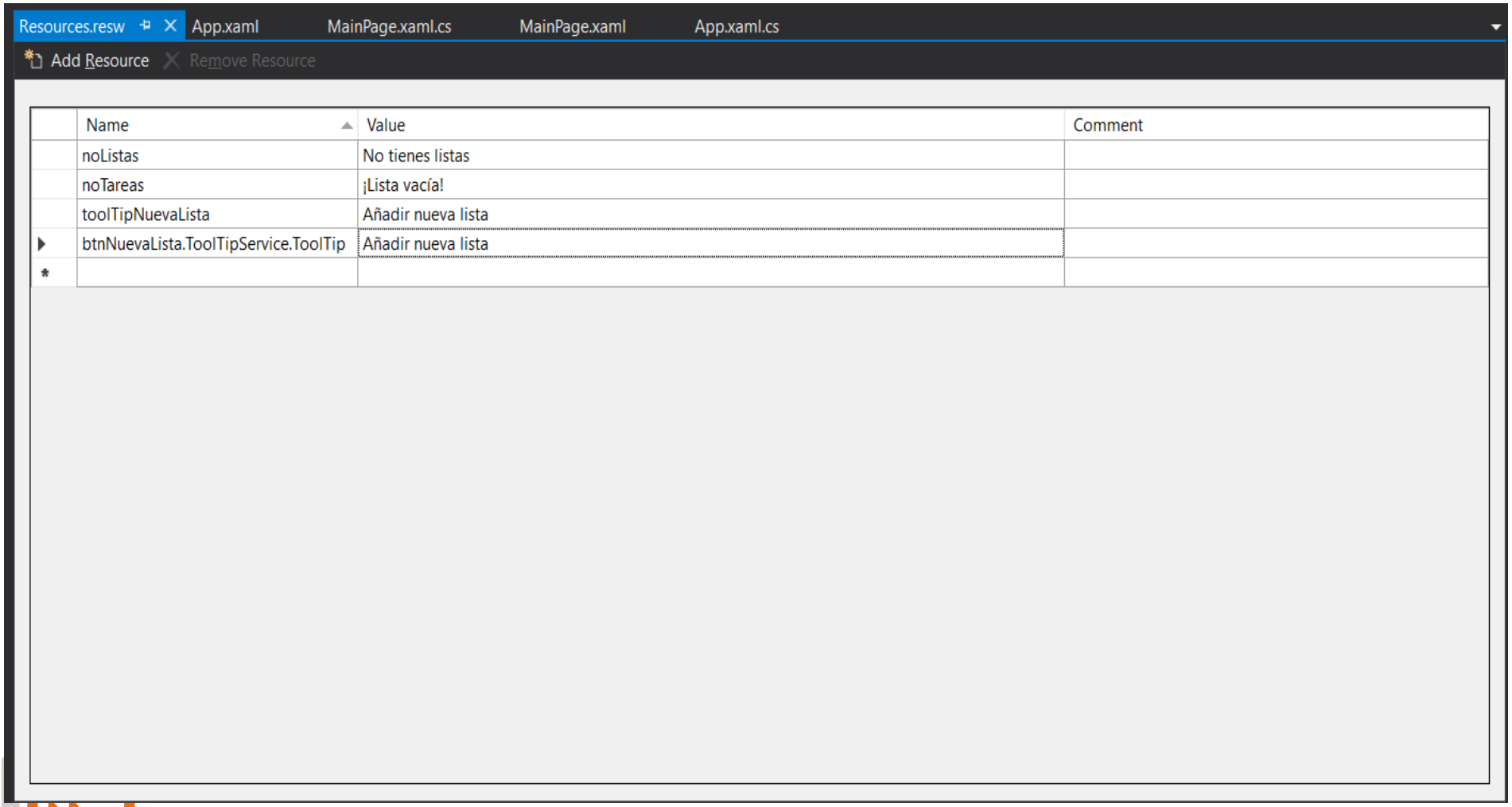
```
<?xml version="1.0" encoding="utf-8"?>
<Package
  xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10"
  xmlns:mp="http://schemas.microsoft.com/appx/2014/phone/manifest"
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
  IgnorableNamespaces="uap mp">
  <Identity Name="3f98a9cd-0fe5-478d-b697-be1704e987ac" Publisher="CN=jaume" Version="1.0.0.0" />
  <mp:PhoneIdentity PhoneProductId="3f98a9cd-0fe5-478d-b697-be1704e987ac" PhonePublisherId="00000000-0
  <Properties>
    <DisplayName>Mis Listas</DisplayName>
    <PublisherDisplayName>jaume</PublisherDisplayName>
    <Logo>Assets\StoreLogo.png</Logo>
  </Properties>
  <Dependencies>
    <TargetDeviceFamily Name="Windows.Universal" MinVersion="10.0.0.0" MaxVersionTested="10.0.0.0" />
  </Dependencies>
  <Resources>
    <Resource Language="x-generate" />
  </Resources>
  <Applications>
    <Application Id="App" Executable="$targetnametoken$.exe" EntryPoint="Mis_Listas.App">
      <uap:VisualElements DisplayName="Mis Listas" Square150x150Logo="Assets\Square150x150Logo.png" Sq
        <uap:DefaultTile Wide310x150Logo="Assets\Wide310x150Logo.png">
        </uap:DefaultTile>
        <uap:SplashScreen Image="Assets\SplashScreen.png" />
        <uap:InitialRotationPreference>
          <uap:Rotation Preference="portrait" />
        </uap:InitialRotationPreference>
      </uap:VisualElements>
    </Application>
  </Applications>
  <Capabilities>
    <Capability Name="internetClient" />
  </Capabilities>
</Package>
```

Recursos para I18n

- Antes de comenzar, vamos a crear la infraestructura necesaria para tener los recursos de texto centralizados y, de esta forma facilitar la localización a otros idiomas.
- Primero, en el fichero '.appxmanifest', modificaremos el idioma por defecto, cambiándolo a 'es-ES'.
- Después añadiremos un fichero de recursos, para ello:
  - Creamos un directorio 'Strings' en el proyecto.
  - Creamos dentro de 'Strings', un directorio 'es-ES'.
  - Dentro de 'es-ES', añadimos un fichero de recursos 'Resource File (.resw)', lo llamaremos por su nombre por defecto 'Resources.resw'.

Recursos para I18n

- Añadiremos los siguientes recursos, para comenzar:



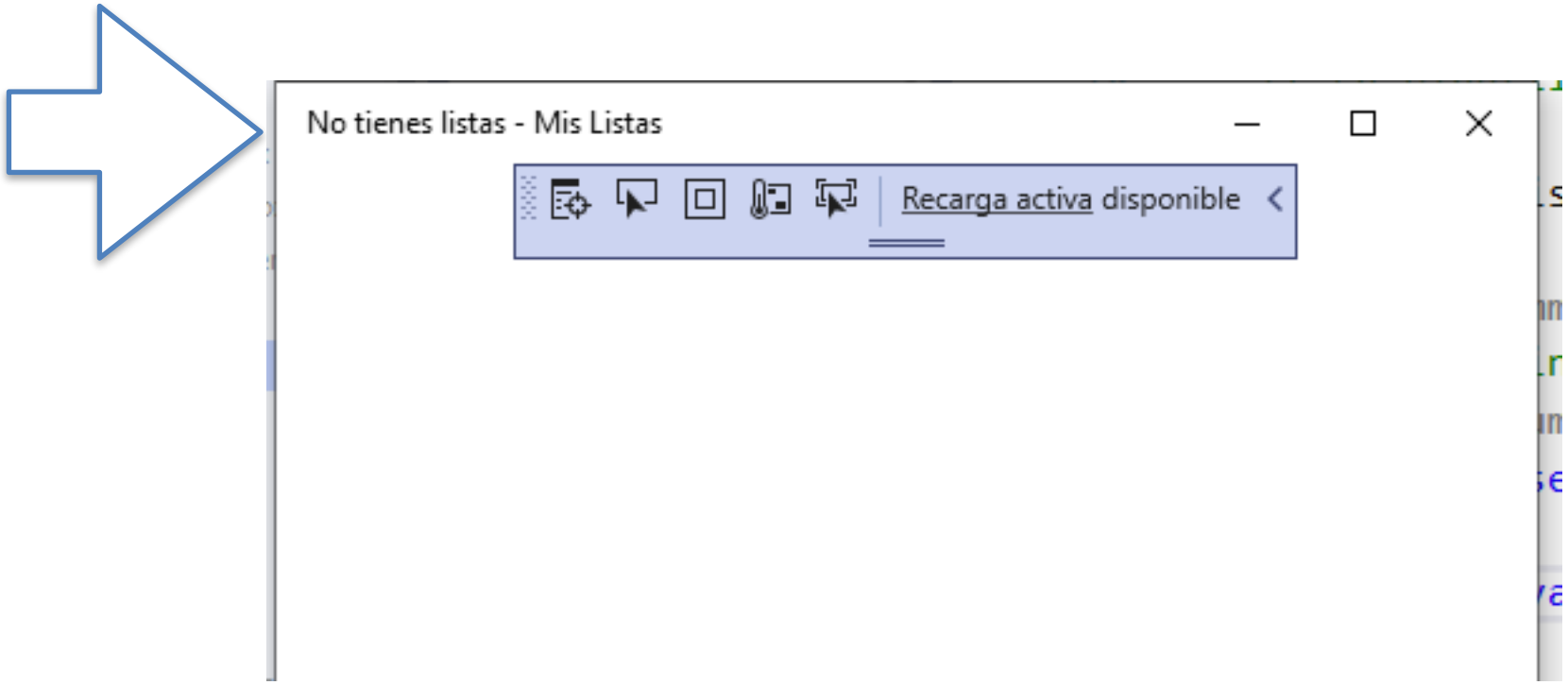
Personalizar la ventana principal

- Vamos a personalizar la ventana principal de nuestra aplicación con un título, color de fondo y dimensiones personalizadas.
- Para añadir un título a la ventana principal tenemos que obtener el objeto ApplicationView, en el constructor de la clase 'MainPage' (esto es similar a lo que hicimos en el taller de i18n):

```
this.textos = new Windows.ApplicationModel.Resources.ResourceLoader();
ApplicationView appView = ApplicationView.GetForCurrentView();

appView.Title = textos.GetString("noListas");
```

  - Este código necesita: `using Windows.UI.ViewManagement;`
- Ademas, en el XAML añadiremos los siguientes atributos al elemento <Page...>:

```
FontFamily="Verdana" Foreground="White" FontSize="14"
Width="600" Height="750" HorizontalAlignment="Stretch"
Margin="0,0,0,0" VerticalAlignment="Stretch"
```



Personalizar la ventana principal

- El siguiente paso consistirá en personalizar el Grid principal de la página:

```
<Grid x:Name="MainGrid" Background="Orange" Width="600"
Height="750" VerticalAlignment="Center" Margin="0,0,0,0">
...</Grid>
```

- Y, en el constructor de 'MainPage' añadiremos:

```
ApplicationView.PreferredLaunchViewSize = new Size(600, 750);
ApplicationView.PreferredLaunchWindowingMode =
ApplicationViewWindowingMode.PreferredLaunchViewSize;
```

Personalizar la ventana principal

- Finalmente, vamos a interceptar el evento de cierre de la aplicación para preguntar al usuario antes (lo hicimos en el taller de personalización).
- Para ello antes tenemos que modificar el fichero 'Package.appxmanifest' (botón derecho > ver código)
  - Añadiremos un nuevo namespace: rescap y lo haremos 'ignorable':

```
<Package

. . .
xmlns:rescap="http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities"
IgnorableNamespaces="uap mp rescap">...</Package>
```

rescap = restricted capability

Personalizar la ventana principal

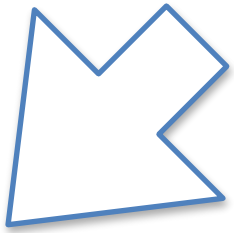
- Modificación del fichero 'Package.appxmanifest'.
  - Ahora añadimos una nueva capacidad al proyecto

```
...<Capabilities>
    <Capability Name="internetClient" />
    <rescap:Capability Name="confirmAppClose"/>
  </Capabilities>
</Package>
```

Personalizar la ventana principal

- Y, en el constructor de 'MainPage':

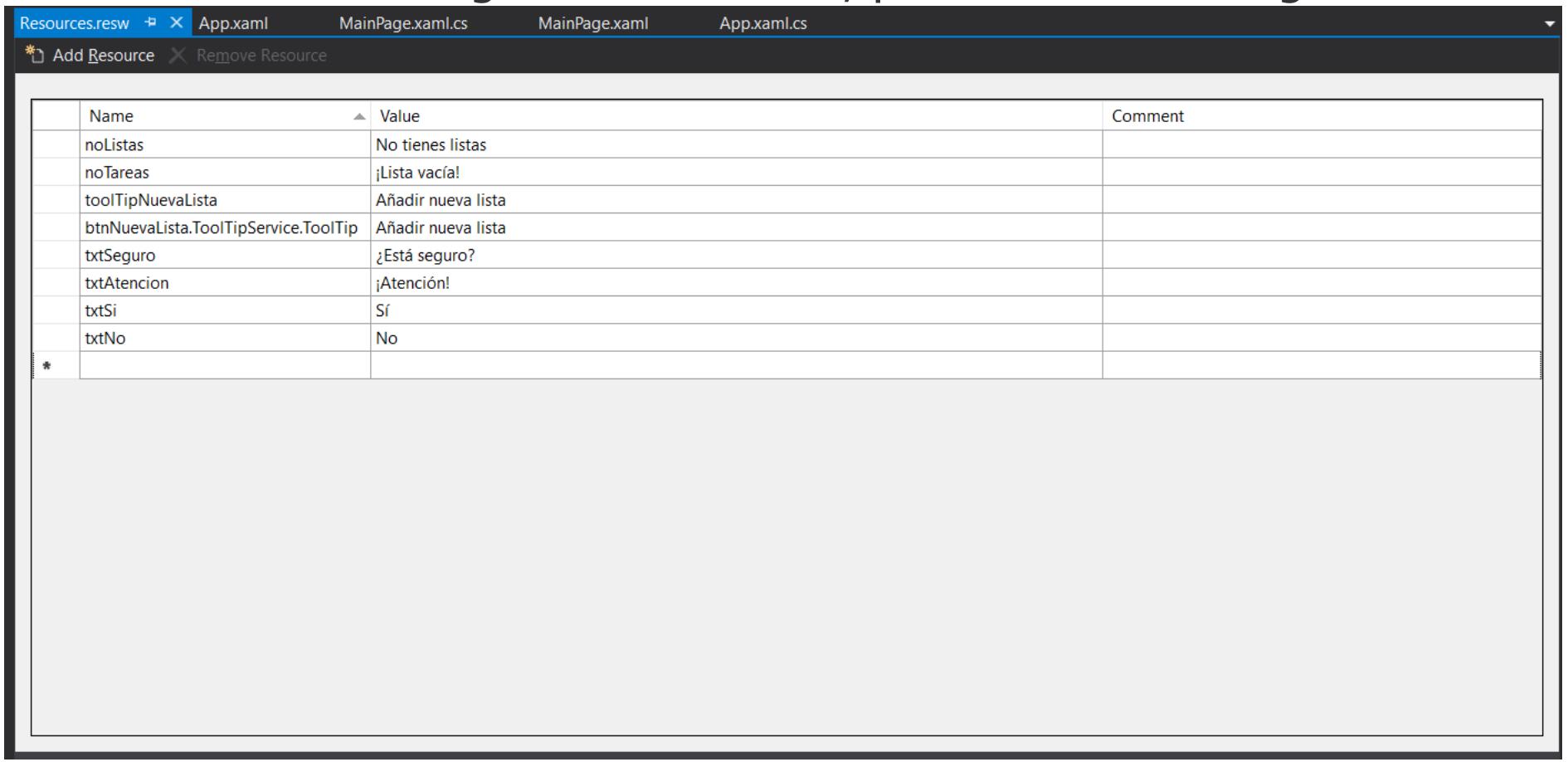
```
private ResourceLoader textos;
public MainPage()
{
    this.InitializeComponent();

    SystemNavigationManagerPreview.GetForCurrentView().CloseRequested += this.OnCloseRequest;

    this.textos = new Windows.ApplicationModel.Resources.ResourceLoader();
```



- Añadiremos ademas:
  - using Windows.UI.Popups;
  - using Windows.ApplicationModel.Resources;

Personalizar la ventana principal

- Añadiremos los siguientes recursos, para el nuevo diálogo:



Name	Value	Comment
noListas	No tienes listas	
noTareas	¡Lista vacía!	
toolTipNuevaLista	Añadir nueva lista	
btnNuevaLista.ToolTipService.ToolTip	Añadir nueva lista	
txtSeguro	¿Está seguro?	
txtAtencion	¡Atención!	
txtSi	Sí	
txtNo	No	
*		

Personalizar la ventana principal

- Y ahora, el evento **OnCloseRequest**, será:

```
private async void OnCloseRequest(object sender, SystemNavigationCloseRequestedPreviewEventArgs e)
{
    e.Handled = true;
    ContentDialog msgBox = new ContentDialog
    {
        Title = textos.GetString("txtAtencion"),
        Content = textos.GetString("txtSeguro"),
        PrimaryButtonText = textos.GetString("txtSi"),
        CloseButtonText = textos.GetString("txtNo")
    };
    var res = await msgBox.ShowAsync();
    if (res == ContentDialogResult.Primary)
    {
        Application.Current.Exit();
    }
}
```

Controles del interfaz

- Ahora incluiremos unas imágenes en el proyecto
  - Para permitir añadir listas: addList.png
  - Para poder borrarlas: deleteList.png
  - Para añadir tareas: addTask.png
  - Y para poder borrar tareas: deleteTask.png
- Para ello, dentro de la carpeta de 'Assets', añadimos un directorio nuevo llamado: 'images layout' y en ella incluiremos las imágenes seleccionadas.

Controles del interfaz

- Ahora añadiremos los controles:
- Un comboBox que contendrá las listas existentes y un ComboBoxItem por defecto:

```
<Grid x:Name="MainGrid" Background="Orange" Width="600"
    Height="750" VerticalAlignment="Center" Margin="0,0,0,0">
    <ComboBox x:Name="slctList" HorizontalAlignment="Left"  Height="72"
              VerticalAlignment="Top" Width="495" Foreground="WhiteSmoke"
              FontFamily="Verdana" FontSize="26" FontWeight="Bold"  Margin="5,5,0,0">
        <ComboBoxItem Content="" x:Uid="itemXdef" IsSelected="True"></ComboBoxItem>
    </ComboBox>
</Grid>
```

Controles del interfaz

- Un botón que permita añadir una nueva lista:

```
<Button HorizontalAlignment="Left"  Height="69"  VerticalAlignment="Top"
        Width="85"   Background="Transparent"
        x:Uid="btnNuevaLista"  x:Name="btnNuevaLista"
        ToolTipService.ToolTip=""
        Margin="505,6,0,0">
    <Image  Source="Assets/images layout/addList.png"
            Height="69"  Margin="-10,-6,-10,0"
            VerticalAlignment="Top"  RenderTransformOrigin="0.491,0.248"/>
</Button>
```

Controles del interfaz

- Un grid que contendrá las tareas de la lista seleccionada, que mostrará un mensaje por defecto:

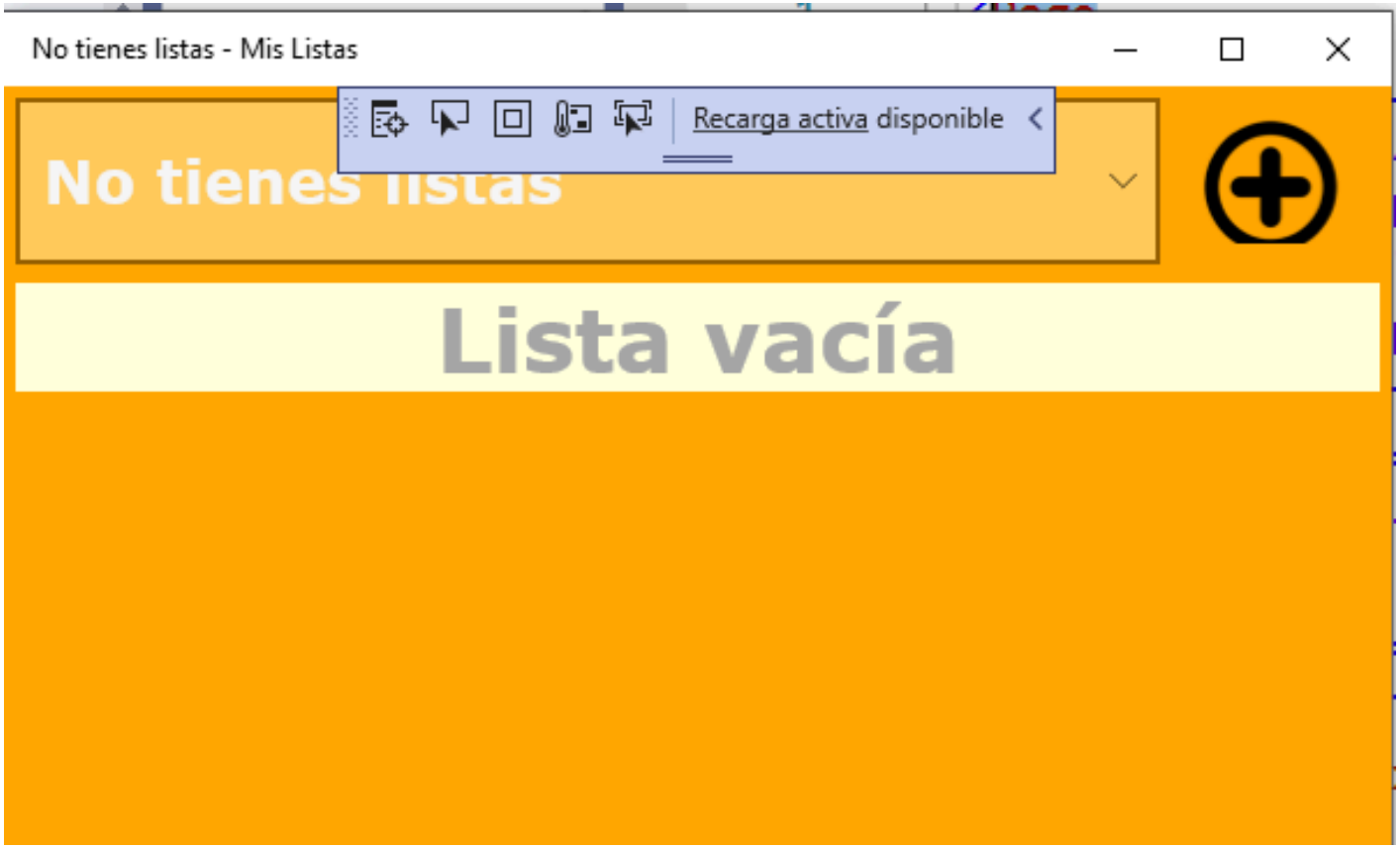
```
<Grid x:Name="ListGrid" Width="590" Height="auto"
      Margin="0,85,0,0" VerticalAlignment="Top">
    <StackPanel Background="LightYellow">
        <TextBlock  x:Name="lblXdef" x:Uid="lblXdef" Text="" Foreground="DarkGray"
                    FontFamily="Verdana" FontSize="38" HorizontalAlignment="Center"
                    FontWeight="Bold" VerticalAlignment="Center">
        </TextBlock>
    </StackPanel>
    <StackPanel x:Name="pnlList" Height="auto"
                ScrollViewer.HorizontalScrollBarVisibility="Auto">
    </StackPanel>
</Grid>
```

Controles del interfaz

- Añadimos los recursos para los controles nuevos que hemos incluido

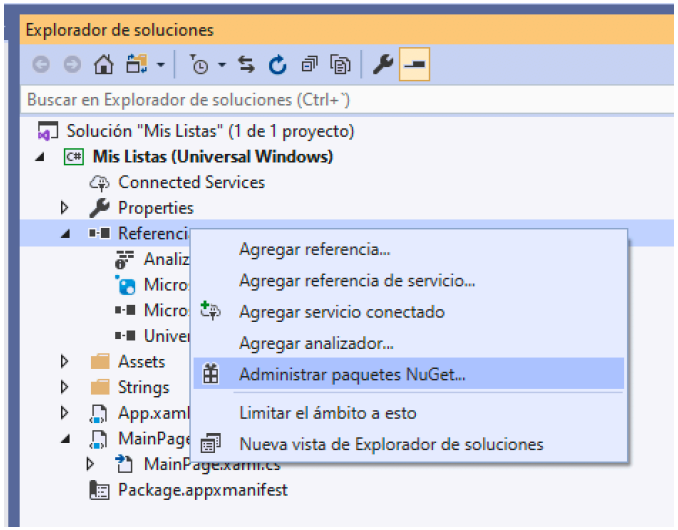
Name	Value
btnNuevaLista.ToolTipService.ToolTip	Añadir nueva lista
noListas	No tienes listas
noTareas	¡Lista vacía!
txtAtencion	¡Atención!
txtNo	No
txtSeguro	¿Está seguro?
txtSi	Sí
toolTipNuevaLista	Añadir nueva lista
itemXdef.Content	No tienes listas
lblXdef.Text	¡Lista vacia!

Estado actual



Almacenamiento de datos

- Antes de comenzar con esta parte deberemos instalar el paquete 'Newtonsoft.Json' desde la ventana de gestión de paquetes NuGet.

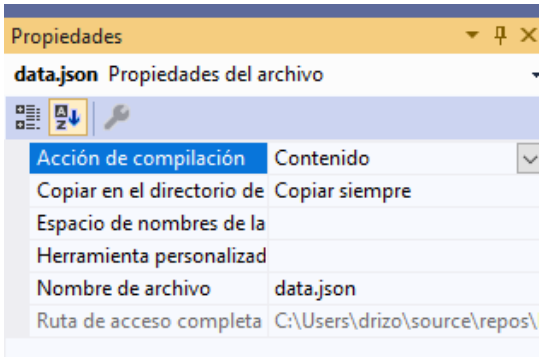


Almacenamiento de datos

- Antes de comenzar con esta parte deberemos instalar el paquete 'Newtonsoft.Json' desde la ventana de gestión de paquetes NuGet.
- Después, crearemos dentro del directorio 'Assets' un nuevo directorio llamado 'Data' y dentro de él crearemos un nuevo fichero llamado 'data.json'. Cuyo contenido será:

```
[
  {
    "name": "Lista de tareas",
    "visible": "true",
    "color": "blue",
    "date": "2019-05-01 0:00:00",
    "tasks": [
      {
        "text": "Tarea de bienvenida",
        "date": "2019-05-01 0:00:00",
        "expires": "2019-12-31 23:59:58",
        "finished": "false",
        "visible": "true"
      }
    ]
  }
]
```

- Debemos modificar las propiedades de este fichero:
  - Compilation action: content
  - Copy to exit directory: Always copy.



Almacenamiento de datos

- Ahora vamos a crear los modelos necesarios, para ello crearemos un directorio llamado 'Models' y dentro de él dos nuevas clases: cList.cs y cTask.cs

```
class cList
{
    public string name { get; set; }
    public DateTime date { get; set; }
    public bool visible { get; set; }
    public string color { get; set; }
    public List<cTask> tasks { get; set; }

    public cList()
    {
        name = "";
        date = DateTime.Now;
        visible = true;
        color = "";
        tasks = new List<cTask>();
    }

}
```

```
class cTask
{
    public string text { get; set; }
    public DateTime date { get; set; }
    public DateTime? expired { get; set; }
    public bool visible { get; set; }
    public bool finished { get; set; }

    public cTask()
    {
        text = "";
        date = DateTime.Now;
        expired = null;
        visible = true;
        finished = false;
    }
```

DateTime? expired permite que este valor sea null

Almacenamiento de datos

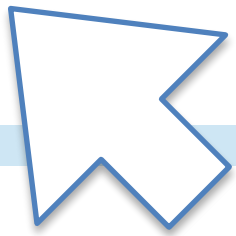
- Ahora accederemos al fichero '.json' y cargaremos los datos que contenga generando los objetos cList y cTask necesarios:
- En el fichero 'MainPage.cs' declaramos la estructura de datos necesaria y en el constructor la inicializaremos e invocaremos a una nueva función:

```
private ResourceLoader textos;
private List<cList> lists;

public MainPage()
{
    //...
    lists = new List<cList>();
    loadData();
}
```

Almacenamiento de datos

- La función loadData, en una primera versión será:

```
async private void loadData()
{
    StorageFolder dataDir =
        await Package.Current.InstalledLocation.GetFolderAsync(@"Assets\Data");
    StorageFile dataFile = await dataDir.GetFileAsync("data.json");
    string cadena = await Windows.Storage.FileIO.ReadTextAsync(dataFile);
    lists = JsonConvert.DeserializeObject<List<cList>>(cadena);
    // si hay listas:
    if (lists.Count >= 1)
    {
        // Mostrar los datos en la interfaz
    }
}
```



Siguiente diapositiva

(Nótese la carga asíncrona de datos con await / async)

Almacenamiento de datos

- Para mostrar los datos en la interfaz tenemos que:
  - Poblar el comboBox 'slctList' con las listas obtenidas y seleccionar la primera lista en él mismo.
  - Mostrar las tareas de la lista seleccionada en el 'ListGrid'

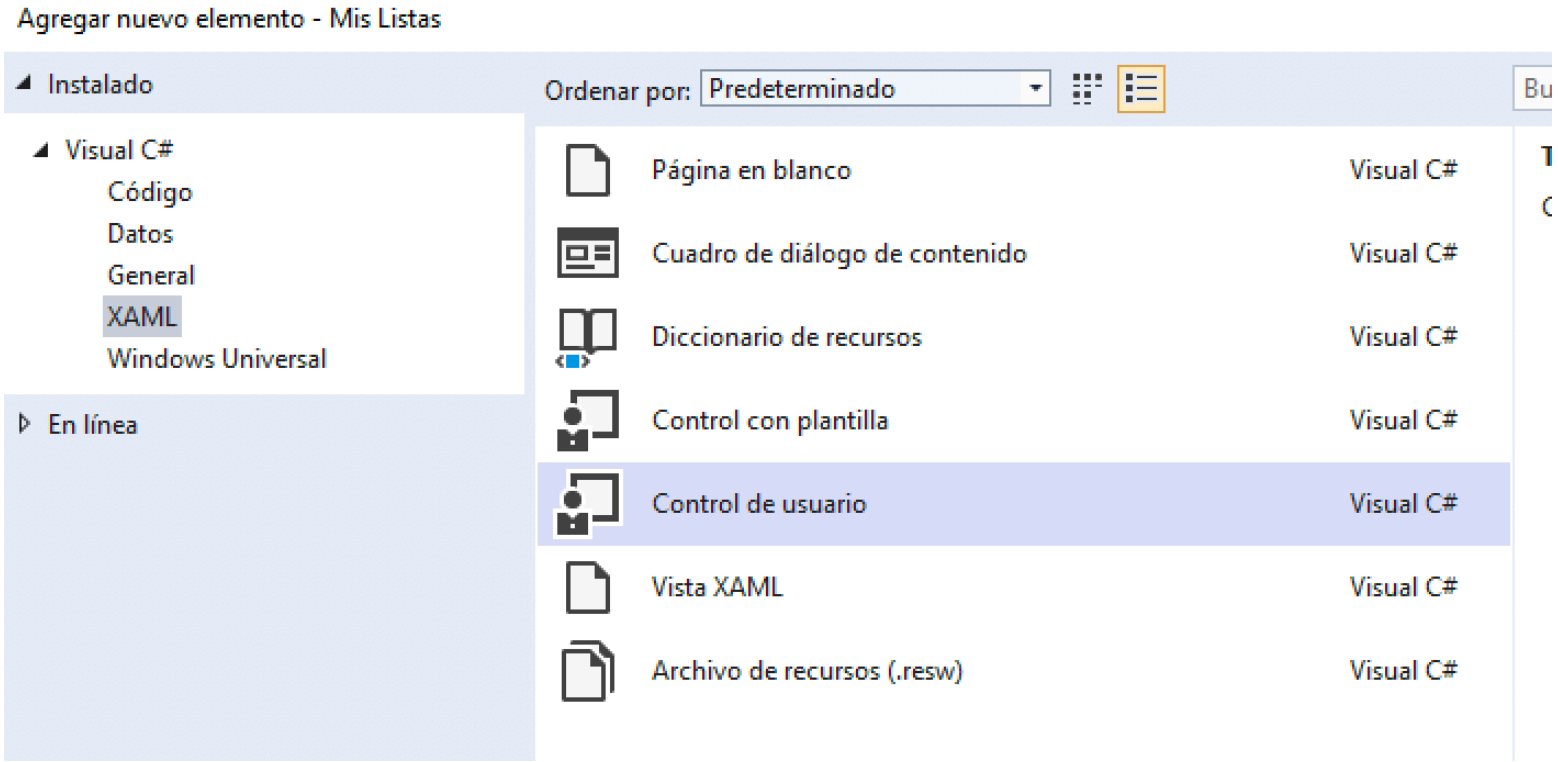
```
// si hay listas:
if (lists.Count >= 1)
{
    //Vamos a poner las listas como  items del comboBox
    slctList.Items.Clear();
    slctList.ItemsSource = lists;
    slctList.DisplayMemberPath = "name";
    slctList.SelectedIndex = 0;
    ApplicationView.GetForCurrentView().Title = lists[0].name;
    // ... y ubicar las tareas en el grid
    LoadTasks(0); // función por definir...

}
```

Estado


[
{
"name": "Lista de tareas",
"visible": "true",
"color": "blue",
"date": "2019-05-0
0:00",
"tasks": [
{
"text": "Tarea de bien
da",
"date": "2019-05-01 0:00
"expires": "2019-12-31 23:5
"finished": "false",
"visible": "true"
}
]
}
]
eas - Mis Listas
Lista de tareas
Lista vacía

Control de usuario

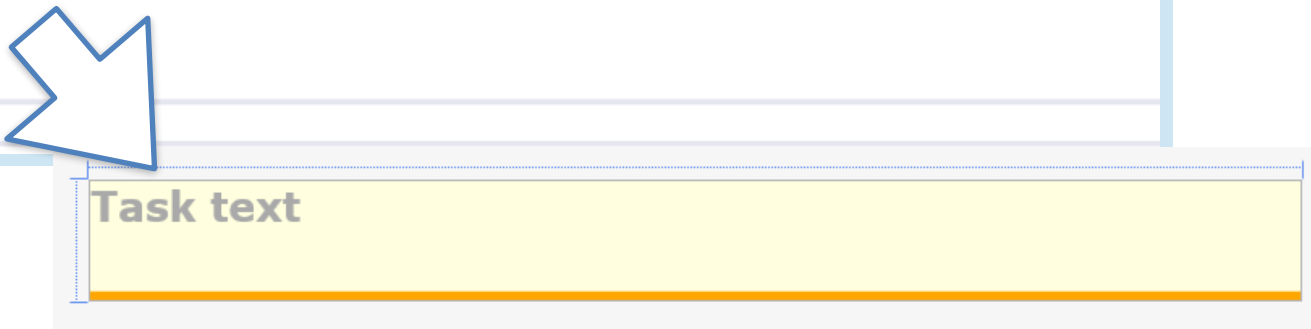
- Antes de mostrar las tareas de la lista seleccionada en el 'ListGrid', vamos a crear un 'User Control' que nos permita encapsular el interfaz y la lógica de una tarea.
- El XAML del nuevo control TareaControl.xaml



Control de usuario

- El XAML del nuevo control:

```
<UserControl
    x:Class="Mis_Listas.TareaControl"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:Mis_Listas"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="60" d:DesignWidth="600">
    <Grid Margin="0,0,0,0" Background="Orange">
        <StackPanel Background="LightYellow" Margin="0,0,0,5">
            <TextBlock x:Name="lblTask" HorizontalAlignment="Left"
                       Margin="1,0,0,0" Text="Task text"
                       TextWrapping="Wrap" VerticalAlignment="Top"
                       Height="auto"  MaxHeight="160" MaxWidth="500"
                       Foreground="DarkGray" FontFamily="Verdana" FontSize="20"
                       FontWeight="Bold"/>
        </StackPanel>
    </Grid>
</UserControl>
```



Control de usuario

- El 'User Control' tendrá el siguiente código en su fichero de 'Code behind':

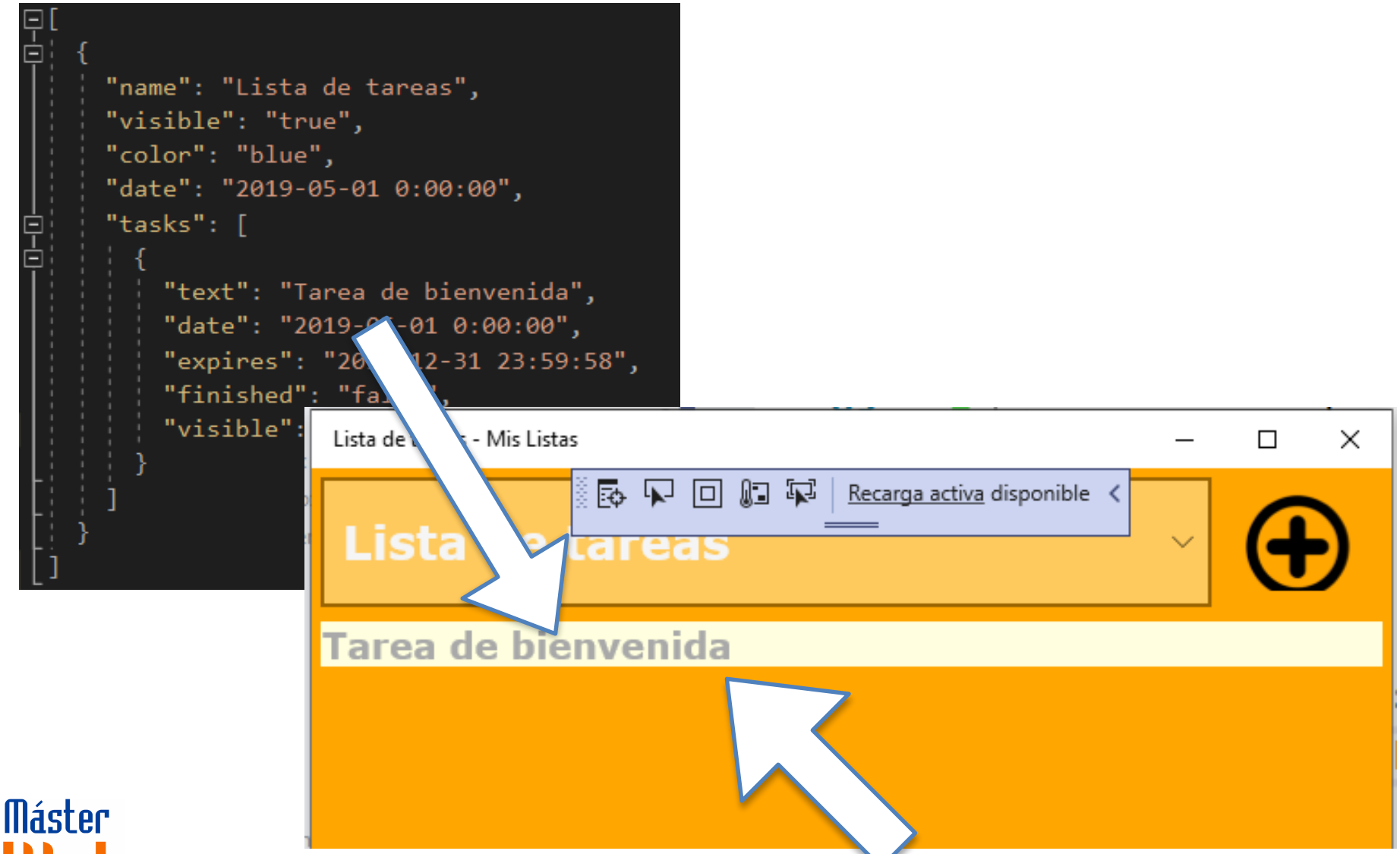
```
public sealed partial class TareaControl : UserControl
{
    public TareaControl(string text = "")
    {
        this.InitializeComponent();
        if (text != "")
        {
            setText(text);
        }
    }

    public void setText(string text)
    {
        lblTask.Text = text;
    }
}
```

Control de usuario

- Ahora debemos mostrar las tareas de la lista seleccionada en el 'ListGrid' rellenando el método que teníamos pendiente en en MainPage.xaml.cs

```
private void LoadTasks(int indice)
{
    cList list = lists.ElementAt(indice);
    pnlList.Children.Clear();
    if (list.tasks.Count > 0)
    {
        lblXdef.Visibility = Visibility.Collapsed;
        foreach (cTask task in list.tasks)
        {
            TareaControl auxControl = new TareaControl(task.text);
            pnlList.Children.Add(auxControl);
        }
    } else
    {
        lblXdef.Visibility = Visibility.Visible;
    }
}
```

Estado actual



Nuevo control TareaControl

Recarga de datos

- Usando la función anterior, vamos a implementar el manejador del evento de cambio en el comboBox slctList.
- Añadimos el evento (haciendo doble clic sobre el comboBox).
- Y su manejador será:

```
private void slctList_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    ComboBox cb = (ComboBox)sender;
    if (cb.Items.Count > 0 && typeof(cList) == cb.Items[cb.SelectedIndex].GetType())
    {
        ApplicationView.GetForCurrentView().Title = lists[cb.SelectedIndex].name;
        LoadTasks(cb.SelectedIndex);
    }
}
```

Añadir una nueva lista

- Usaremos un flyout asociado al botón 'btnNuevaLista'.
- El flyout, que llamaremos 'flyAddList' contendrá
  - un título
  - un separador
  - una etiqueta
  - una caja de texto
  - y dos botones: para aceptar (btnAdd_newList) y para cancelar (btnCancel_newList).

(véase siguiente diapositiva)

```
<Button HorizontalAlignment="Left"  Height="69"  VerticalAlignment="Top"
        Width="85"   Background="Transparent"
        x:Uid="btnNuevaLista"  x:Name="btnNuevaLista"
        ToolTipService.ToolTip=""
        Margin="505,6,0,0">
    <Image  Source="Assets/images layout/addList.png"
            Height="69"  Margin="-10,-6,-10,0"
            VerticalAlignment="Top"  RenderTransformOrigin="0.491,0.248">
    </Image>
    <Button.Flyout>
        <Flyout>
            <StackPanel>
                <TextBlock x:Name="tituloNuevaLista" Text="Título"></TextBlock>
                <Border></Border>
                <TextBlock x:Name="lblNuevaLista" Text="Etiqueta"></TextBlock>
                <Button x:Name="btnAdd_newList" Content="Aceptar"></Button>
                <Button x:Name="btnCancel_newList" Content="Cancelar"></Button>
            </StackPanel>
        </Flyout>
    </Button.Flyout>
</Button>
```

(luego cambiaremos el StackPanel con un Grid)

Añadir una nueva lista

- Usaremos el flyout asociado al botón 'btnNuevaLista'.
- Implementaremos tres manejadores:
  - Al hacer click sobre 'btnNuevaLista'
  - Al aceptar (hacer click sobre btnAdd_newList)
  - Al cancelar (hacer click sobre btnCancel_newList)

Finalmente, añadiremos algunos textos en el fichero de recursos.

```
<Button HorizontalAlignment="Left"  Height="69"  VerticalAlignment="Top"
        Width="85"   Background="Transparent"
        x:Uid="btnNuevaLista"  x:Name="btnNuevaLista"
        ToolTipService.ToolTip=""
        Margin="505,6,0,0"
        Click="btnNuevaLista_Click"
        >
```

etc...

Añadir una nueva lista

- El XAML del flyout contendrá:

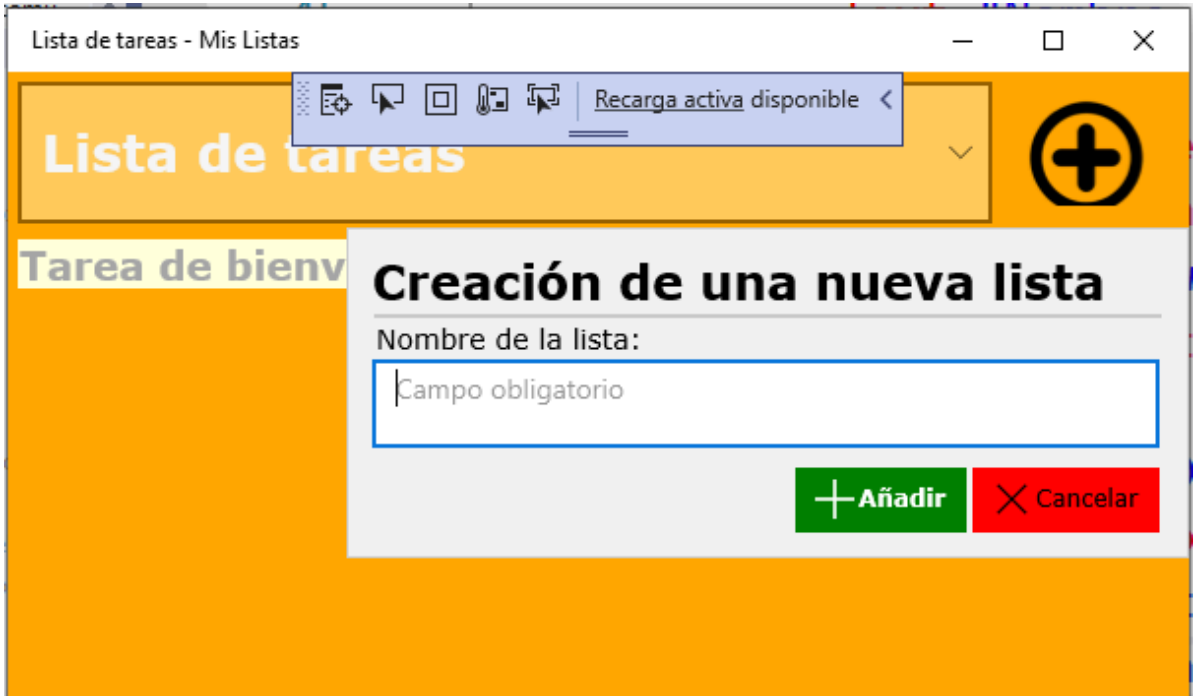
```
<Button.Flyout>
    <Flyout x:Name="flyAddList">
        <Grid>
            <TextBlock x:Name="tituloNuevaLista" Margin="0,0,0,0"
                       FontSize="24" FontWeight="Bold"
                       FontFamily="Verdana"
                       Text="Creación de una nueva lista"/>
            <Border Width="400" VerticalAlignment="Top"
                    Margin="0,32,0,0" Height="2"
                    Background="#CCCCCC"/>
            <TextBlock x:Name="lblNuevaLista" Margin="2,35,0,0"
                       Width="Auto" TextWrapping="Wrap"
                       HorizontalAlignment="Left"
                       FontSize="14" FontWeight="Normal"
                       FontFamily="Verdana"
                       Text="Nombre de la lista: "/>
            <TextBox Margin="0, 55, 2, 0" Width="400"
                     VerticalAlignment="Top"
                     HorizontalAlignment="Right" Height="45"
                     x:Name="txt_newList_name"
                     PlaceholderText="Campo obligatorio">
            </TextBox>
```

Añadir una nueva lista

- Continuación del grid del flyout:

```
                PlaceholderText="Campo obligatorio">
            </TextBox>
            <Button Margin="0,110,100,0" FontFamily="Verdana"
                    FontSize="12" FontWeight="Bold" Background="Green"
                    Foreground="White" HorizontalAlignment="Right"
                    x:Name="btnAdd_newList" Click="btnAdd_newList_Click">
                <StackPanel Orientation="Horizontal">
                    <SymbolIcon Symbol="Add"/>
                    <TextBlock x:Name="txt_btnAdd_newList"
                               Text="Añadir" Margin="2,2,0,0">
                    </TextBlock>
                </StackPanel>
            </Button>
            <Button Margin="0, 110, 2, 0" FontFamily="Verdana"
                    FontSize="12" FontWeight="Normal" Background="Red"
                    HorizontalAlignment="Right"
                    x:Name="btnCancel_newList" Click="btnCancel_newList_Click">
                <StackPanel Orientation="Horizontal">
                    <SymbolIcon Symbol="Cancel"/>
                    <TextBlock x:Name="txt_btnCancel_newList"
                               Text="Cancelar" Margin="2,2,0,0">
                    </TextBlock>
                </StackPanel>
            </Button>
        </Grid>
    </Flyout>
</Button.Flyout>
```

Añadir una nueva lista



Añadir una nueva lista

- Definiremos nuevos textos en el fichero de recursos:

Name	Value
btnNuevaLista.ToolTipService.ToolTip	Añadir nueva lista
itemXdef.Content	No tienes listas
lblXdef.Text	¡Lista vacia!
noListas	No tienes listas
noTareas	¡Lista vacía!
toolTipNuevaLista	Añadir nueva lista
txtAtencion	¡Atención!
txtNo	No
txtSeguro	¿Está seguro?
txtSi	Sí
tituloNuevaLista	Creación de una nueva lista
lblNuevaLista	Nombre de la lista:
ttpNuevaLista	Campo obligatorio
errNuevaLista	Debe escribir un nombre para la lista
btnAdd	Añadir
btnCancel	Cancelar

Añadir una nueva lista

- Ahora, el código del evento que mostrará el flyout de añadir nueva lista:

```
private void btnNuevaLista_Click(object sender, RoutedEventArgs e)
{
    tituloNuevaLista.Text = textos.GetString("tituloNuevaLista");
    lblNuevaLista.Text = textos.GetString("lblNuevaLista");
    txt_btnAdd_newList.Text = textos.GetString("btnAdd");
    txt_btnCancel_newList.Text = textos.GetString("btnCancel");
    txt_newList_name.Text = "";
    txt_newList_name.PlaceholderText = textos.GetString("ttpNuevaLista");
    txt_newList_name.PlaceholderForeground = new SolidColorBrush(Colors.LightGray);
}
```

Añadir una nueva lista

- El evento de cancelar, simplemente cerrará el flyout:

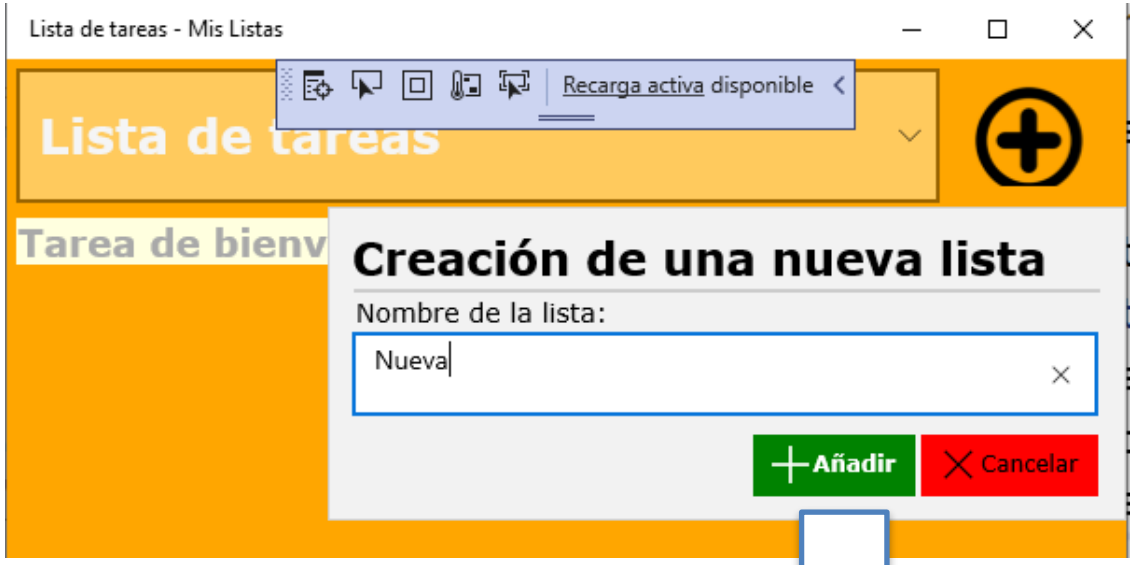
```
private void btnCancel_newList_Click(object sender, RoutedEventArgs e)
{
    flyAddList.Hide();
}
```

Añadir una nueva lista

- Y el evento que añade la nueva lista:

```
private void btnAdd_newList_Click(object sender, RoutedEventArgs e)
{
    if (txt_newList_name.Text != "")
    {
        flyAddList.Hide();
        cList lista = new cList();
        lista.name = txt_newList_name.Text;
        lista.color = "";
        lista.date = DateTime.Now;
        lista.tasks = new List<cTask>();
        lista.visible = true;
        lists.Add(lista);
        slctList.ItemsSource = lists;
        slctList.DisplayMemberPath = "name";
        slctList.UpdateLayout();
    }
    else
    {
        txt_newList_name.PlaceholderText = textos.GetString("errNuevaLista");
        txt_newList_name.PlaceholderForeground = new SolidColorBrush(Colors.Red);
    }
}
```

Estado actual



Eliminar una lista

- Vamos a añadir un nuevo botón para eliminar una lista.
- Le vincularemos la nueva imagen: 'deleteList.png'.
- El código XAML quedará así (solo se muestran el nuevo botón y las propiedades que hayan cambiado:

```
<Grid x:Name="MainGrid" Background="Orange" Width="600"
    Height="750" VerticalAlignment="Center" Margin="0,0,0,0">
    <ComboBox x:Name="slctList" HorizontalAlignment="Left"  Height="72"
              VerticalAlignment="Top" Width="495" Foreground="WhiteSmoke"
              FontFamily="Verdana" FontSize="26" FontWeight="Bold"  Margin="5,5,0,0"
              SelectionChanged="slctList_SelectionChanged">
        <ComboBoxItem Content="" x:Uid="itemXdef" IsSelected="True"></ComboBoxItem>
    </ComboBox>
    <Button HorizontalAlignment="Left" Height="69" VerticalAlignment="Top" Width="60"
            x:Uid="btnBorraLista" x:Name="btnBorraLista" ToolTipService.ToolTip=""
            Margin="445,4,0,0" Background="Transparent" Click="btnBorraLista_Click">
        <Image Source="Assets/images layout/deleteList.png" Height="69"
               Margin="-10.4,0,-9.8,0" VerticalAlignment="Center"
               RenderTransformOrigin="0.491,0.248" HorizontalAlignment="Stretch"/>
    </Button>
```

Eliminar una lista

Eliminar una lista

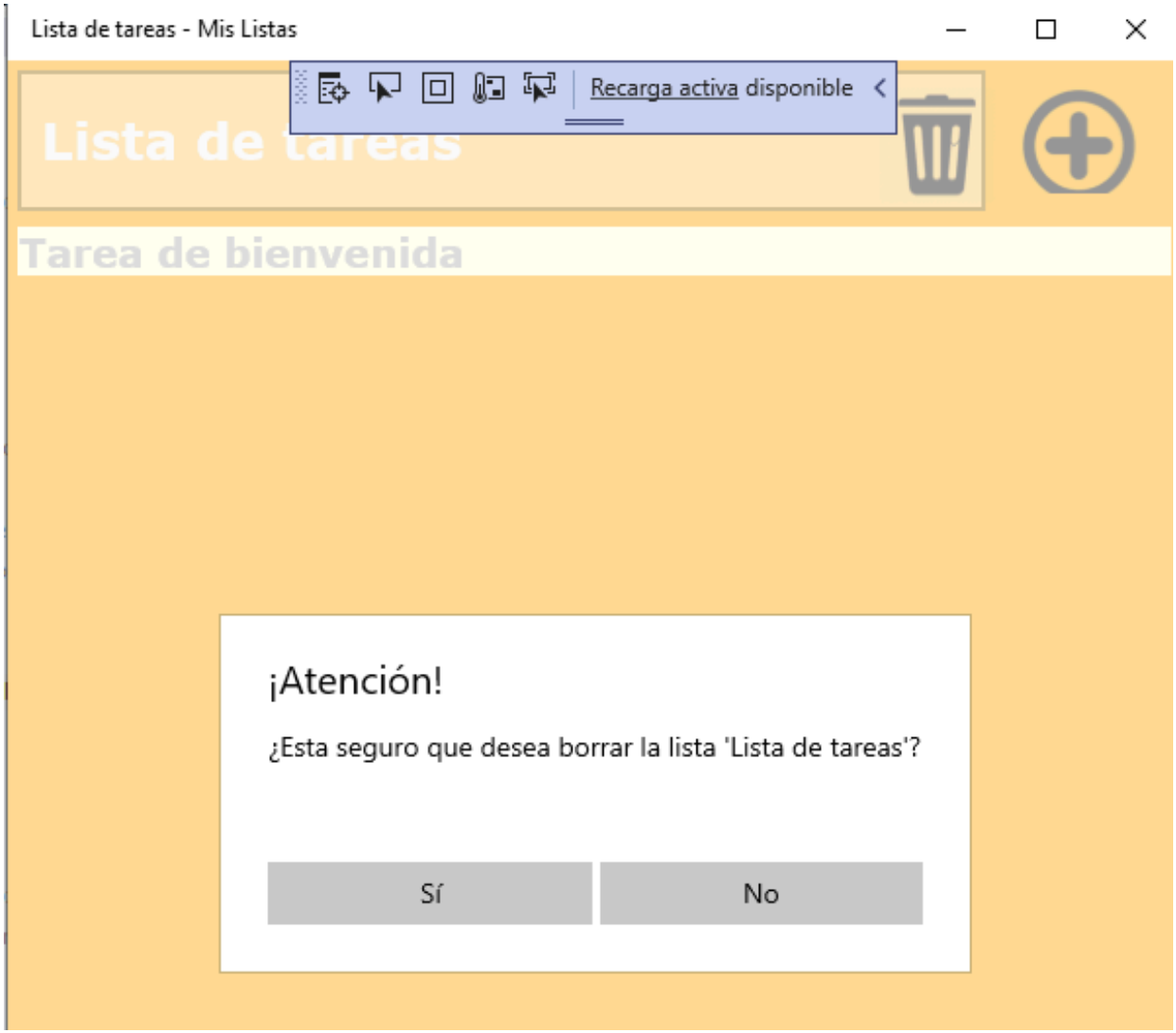
- En el fichero de recursos añadiremos unos nuevos textos:
  - btnBorraLista.ToolTipService.ToolTip: "Elimina la lista actual"
  - txtSeguroBorrar: "¿Esta seguro que desea borrar la lista '{0}'?"
- El evento `btnBorraLista_Click' gestionará el borrado de una lista.
- Este evento mostrará un diálogo pidiendo confirmación del borrado.
- En caso de que sólo quede una lista, no realizará borrado alguno.
  - Una mejora podría ser que el programa avisara que sólo queda una o bien que permitiera el borrado y a continuación, si no quedan listas, generar una lista nueva por defecto.

Eliminar una lista

- El evento 'btnBorraLista_Click' tendrá el siguiente código:

```
async private void btnBorraLista_ClickAsync(object sender, RoutedEventArgs e)
{
    cList lista = lists.ElementAt(slctList.SelectedIndex);
    ContentDialog msgBox = new ContentDialog
    {
        Title = textos.GetString("txtAtencion"),
        Content = String.Format(textos.GetString("txtSeguroBorrar"), lista.name)
        PrimaryButtonText = textos.GetString("txtSi"),
        CloseButtonText = textos.GetString("txtNo")
    };
    var res = await msgBox.ShowAsync();
    if (res == ContentDialogResult.Primary)
    {
        if (lists.Count > 1)
        {
            lists.RemoveAt(slctList.SelectedIndex);
            slctList.ItemsSource = lists;
            slctList.DisplayMemberPath = "name";
            slctList.UpdateLayout();
            slctList.SelectedIndex = 0;
        }
    }
}
```

Estado

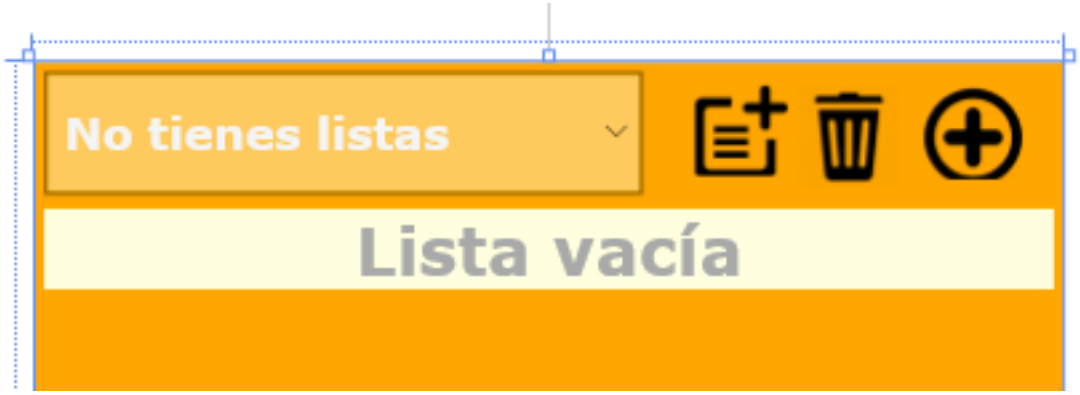


Añadir nueva tarea

- Primero haremos espacio para un nuevo botón, modificando la posición del grid 'ListGrid'.
- El nuevo botón usará la imagen 'addTask.png' que cargamos anteriormente.
- El XAML quedará como sigue:

```
</ComboBox>
<Button x:Name="btnNuevaTarea" Height="69" Width="69"
        Margin="378,4,0,0" VerticalAlignment="Top"
        Background="Transparent" Click="btnNuevaTarea_Click">
    <Image Source="Assets/images layout/addTask.png"
           Height="69" Margin="-10,-4,-10,0"
           VerticalAlignment="Top"
           RenderTransformOrigin="0.491,0.248"/>
</Button>
<Button HorizontalAlignment="Left" Height="69" VerticalAlignment=
```

Estado actual



Añadir nueva tarea

- Despues añadiremos unos nuevos resursos de texto en el fichero '.resw':
  - errNuevaTarea: "Debe escribir una descripción"
  - lblNuevaTarea: "Descripción:"
  - tituloNuevaTarea: "Añadir una nueva tarea"
  - ttpNuevaTarea: "Dato obligatorio "

Añadir nueva tarea

- El XAML del flyout del botón de añadir tarea será:

```
</ComboBox>
<Button x:Name="btnNuevaTarea" Height="69" Width="69"
        Margin="378,4,0,0" VerticalAlignment="Top"
        Background="Transparent" Click="btnNuevaTarea_Click">
    <Image Source="Assets/images layout/addTask.png"
           Height="69" Margin="-10,-4,-10,0"
           VerticalAlignment="Top"
           RenderTransformOrigin="0.491,0.248"/>
    <Button.Flyout>
        <Flyout x:Name="flyAddTask">
            <Grid Background="Orange" BorderBrush="LightGray"
                  BorderThickness="1" Padding="2,2,2,2">
                <TextBlock x:Name="tituloNuevaTarea" Margin="0,0,0,0"
                           FontSize="24" FontWeight="Bold" FontFamily="Verdana"
                           Text="Creación de  una nueva tarea" />
                <Border Width="400" VerticalAlignment="Top" Margin="0, 32, 0, 0"
                        Height="2" Background="#CCCCCC" />
```

Añadir nueva tarea

- El XAML del flyout, continuación:

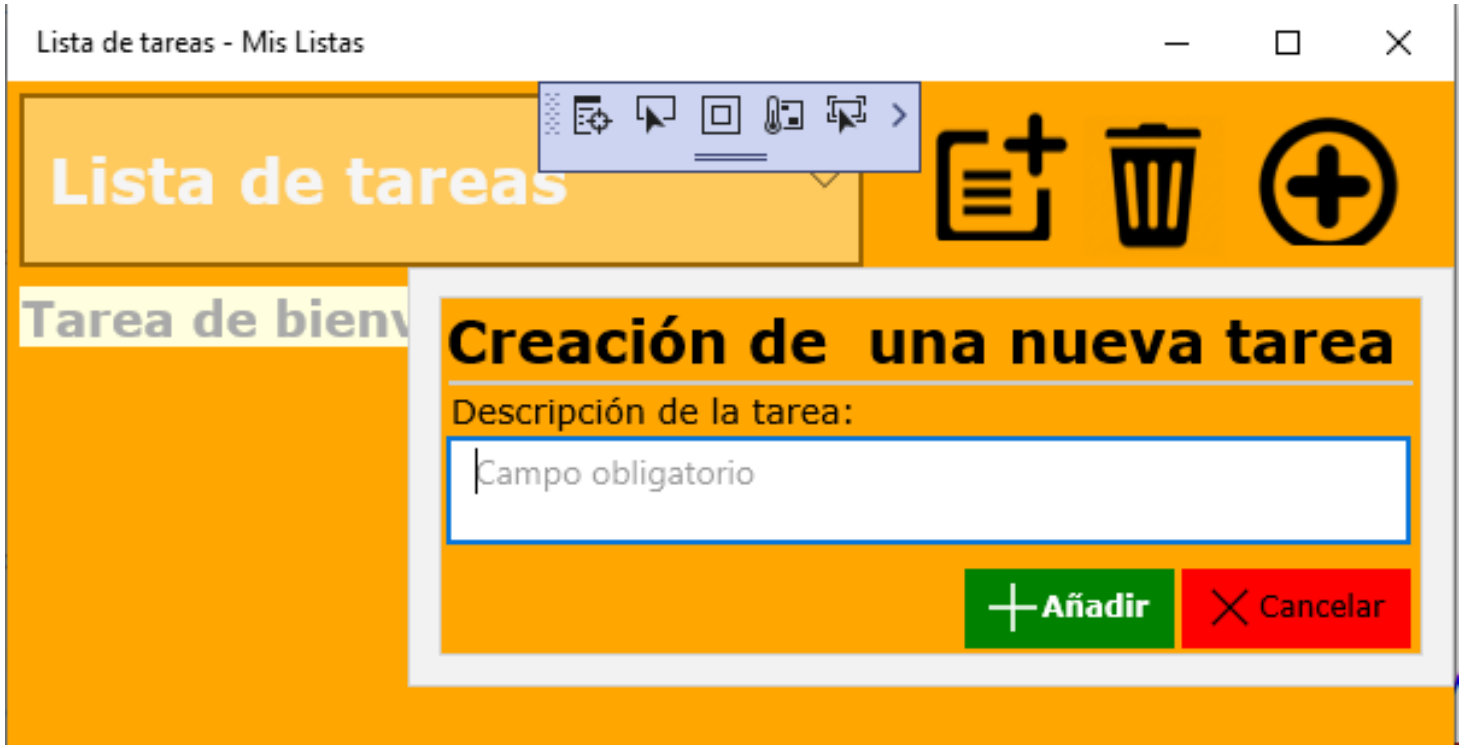
```
        Height="2" Background="#CCCCCC" />
<TextBlock x:Name="lblNuevaTarea" Margin="2, 35, 0, 0"
           Width="Auto" TextWrapping="Wrap"
           HorizontalAlignment="Left" FontSize="14"
           FontWeight="Normal" FontFamily="Verdana"
           Text="Descripción de la tarea: " />
<TextBox Margin="0, 55, 2, 0" Width="400"
         VerticalAlignment="Top" HorizontalAlignment="Right"
         Height="45" x:Name="txt_newTask_name"
         PlaceholderText="Campo obligatorio"></TextBox>
<Button Margin="0, 110, 100, 0" FontFamily="Verdana"
        FontSize="12" FontWeight="Bold" Background="Green"
        HorizontalAlignment="Right" Foreground="White"
        x:Name="btnAdd_newTask" Click="btnAdd_newTask_Click">
    <StackPanel Orientation="Horizontal">
        <SymbolIcon Symbol="Add" />
        <TextBlock x:Name="txt_btnAdd_newTask" Text="Añadir"
                   Margin="2,2,0,0"></TextBlock>
    </StackPanel>
</Button>
```

Añadir nueva tarea

- El XAML del flyout, continuación:

```
                    </StackPanel>
                </Button>
                <Button Margin="0, 110, 2, 0" FontFamily="Verdana" FontSize="12"
                        FontWeight="Normal" Background="Red"
                        HorizontalAlignment="Right" x:Name="btnCancel_newTask"
                        Click="btnCancel_newTask_Click">
                    <StackPanel Orientation="Horizontal">
                        <SymbolIcon Symbol="Cancel" />
                        <TextBlock x:Name="txt_btnCancel_newTask"
                                   Text="Cancelar" Margin="2,2,0,0">
                        </TextBlock>
                    </StackPanel>
                </Button>
            </Grid>
        </Flyout>
    </Button.Flyout>
</Button>
```

Estado actual



Añadir nueva tarea

- El botón de añadir nueva tarea, antes de mostrar el flyout, lo inicializará de esta forma:

```
private void btnNuevaTarea_Click(object sender, RoutedEventArgs e)
{
    tituloNuevaTarea.Text = textos.GetString("tituloNuevaTarea");
    lblNuevaTarea.Text = textos.GetString("lblNuevaTarea");
    txt_btnAdd_newTask.Text = textos.GetString("btnAdd");
    txt_btnCancel_newTask.Text = textos.GetString("btnCancel");
    txt_newTask_name.Text = "";
    txt_newTask_name.PlaceholderText = textos.GetString("ttpNuevaTarea");
    txt_newTask_name.PlaceholderForeground = new SolidColorBrush(Colors.LightGray);
}
```

Añadir nueva tarea

- El botón de cancelar la acción de añadir nueva tarea:

```
private void btnCancel_newTask_Click(object sender, RoutedEventArgs e)
{
    flyAddTask.Hide();

}
```

Añadir nueva tarea

- El botón de 'Aceptar', que añade efectivamente la nueva tarea:

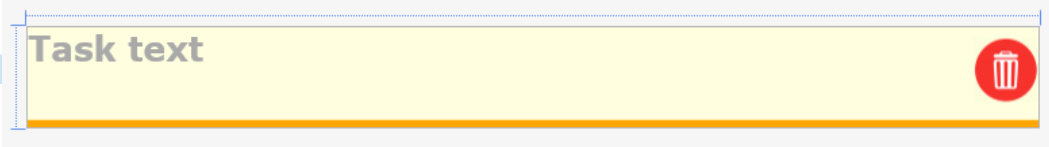
```
private void btnAdd_newTask_Click(object sender, RoutedEventArgs e)
{
    if (txt_newTask_name.Text != "")
    {
        flyAddTask.Hide();
        cTask tarea = new cTask();
        tarea.text = txt_newTask_name.Text;
        tarea.date = DateTime.Now;
        tarea.visible = true;
        tarea.expired = null;
        tarea.finished = false;
        lists[slctList.SelectedIndex].tasks.Add(tarea);
        LoadTasks(slctList.SelectedIndex);
    }
    else
    {
        txt_newTask_name.PlaceholderText = textos.GetString("errNuevaTarea");
        txt_newTask_name.PlaceholderForeground = new SolidColorBrush(Colors.Red);
    }
}
```

Estado actual

Operaciones sobre tareas: eliminar

- Primero, debemos modificar el control de usuario 'TareaControl' para añadirle un nuevo botón, el cual usará una nueva imagen: 'deleteTask.png'. El XAML del control quedará así:

```
<Grid Margin="0,0,0,0" Background="Orange">
    <StackPanel Background="LightYellow" Margin="0,0,0,5">
        <TextBlock x:Name="lblTask" HorizontalAlignment="Left"
                   Margin="1,0,0,0" Text="Task text"
                   TextWrapping="Wrap" VerticalAlignment="Top"
                   Height="auto"  MaxHeight="160" MaxWidth="500"
                   Foreground="DarkGray" FontFamily="Verdana" FontSize="20"
                   FontWeight="Bold"/>
        <Button HorizontalAlignment="Right" Height="48"
                VerticalAlignment="Center" Margin="0,0,0,0" Width="40"
                Background="Transparent" x:Name="btnDeleteTask"
                Click="btnDeleteTask_Click">
            <Image Source="Assets/images layout/deleteTask.png" Height="40"
                   Margin="-10,-50,-10,-4" VerticalAlignment="Center" />
        </Button>
    </StackPanel>
</Grid>
```



Operaciones sobre tareas: eliminar

- Y, debemos definir la clase deleteTaskEventArgs:

```
public class deleteTaskEventArgs
{
    public int list {get; set;}
    public int task { get; set; }
    public deleteTaskEventArgs(int l, int t): base()
    {
        list = l;
        task = t;
    }
}
```

Operaciones sobre tareas: eliminar

- Añadimos un nuevo recurso de texto:
  - txtSeguroBorrarTarea: "¿Esta seguro que desea borrar la tarea '{0}'?"
- A continuación, vamos a modificar el 'code behind' del control de usuario para que permita realizar esta nueva operación:
  - Añadiremos dos nuevas propiedades y un evento nuevo.

```
public int list { get; set; }

public int task { get; set; }

public event EventHandler<deleteTaskEventArgs> deleteTask;
```

  - Este evento lo usaremos para poder invocar desde los clicks sobre los botones de borrar con una función de borrar tarea de la página 'MainPage'.

Operaciones sobre tareas: eliminar

- Por tanto el constructor del user control se modificará:

```
public TareaControl(string text = "", int list = 0, int task = 0)
{
    this.InitializeComponent();
    if (text != "")
    {
        setText(text);
    }
    this.list = list; this.task = task;
}
```

Operaciones sobre tareas: eliminar

- El evento que maneja el click sobre el botón de borrar tarea del user control será:

```
private void btnDeleteTask_Click(object sender, RoutedEventArgs e)
{
    if (deleteTask != null)
    {
        deleteTaskEventArgs ea = new deleteTaskEventArgs(list, task);
        deleteTask(sender, ea);
    }
}
```

Operaciones sobre tareas: eliminar

- Y añadimos la función que elimina las tareas, en el fichero MainPage.xaml.cs:

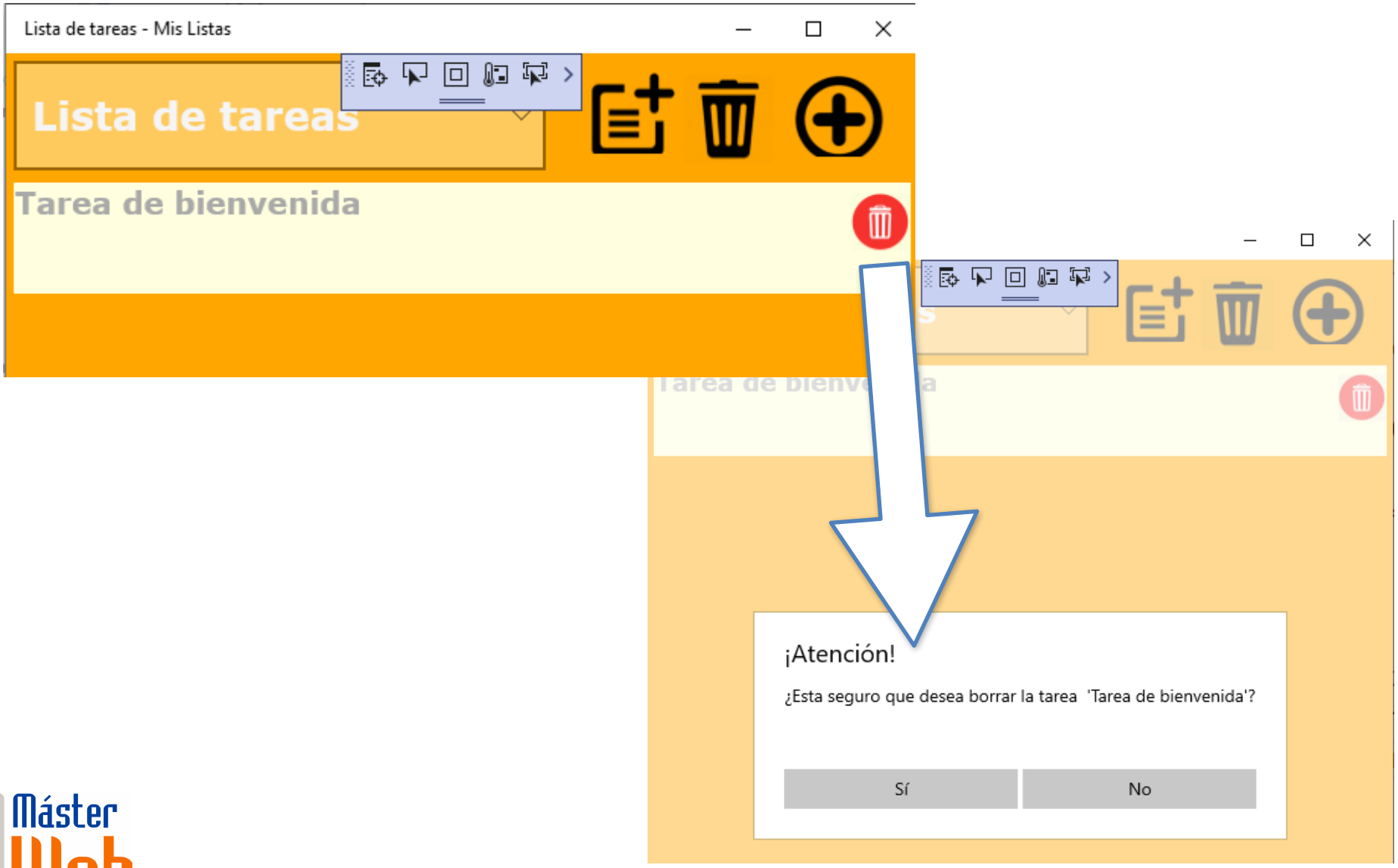
```
async private void deleteTask(object sender, deleteTaskEventArgs e)
{
    cTask tarea = lists.ElementAt(e.list).tasks.ElementAt(e.task);
    ContentDialog msgBox = new ContentDialog
    {
        Title = textos.GetString("txtAtencion"),
        Content = String.Format(textos.GetString("txtSeguroBorrarTarea"), tarea.text),
        PrimaryButtonText = textos.GetString("txtSi"),
        CloseButtonText = textos.GetString("txtNo")
    };

    var res = await msgBox.ShowAsync();
    if (res == ContentDialogResult.Primary)
    {
        lists.ElementAt(e.list).tasks.RemoveAt(e.task);
        LoadTasks(e.list);
    }
}
```

Operaciones sobre tareas: eliminar

- En el fichero MainPage.xaml.cs, la función loadTasks cambiará, pues varia la forma de generar los user controls de las tareas:

```
int numTask = 0;
foreach (cTask task in list.tasks)
{
    TareaControl auxControl = new TareaControl(task.text, slctList.SelectedIndex, numTask);
    auxControl.deleteTask += new EventHandler<deleteTaskEventArgs>(deleteTask);
    pnlList.Children.Add(auxControl);
    numTask++;
}
```

Estado actual

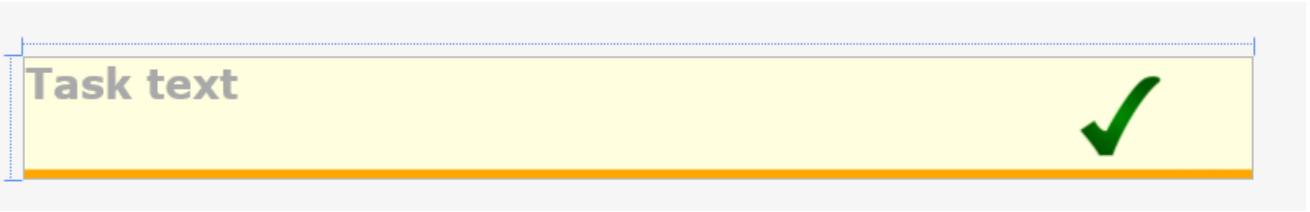


Marcar una tarea como terminada

- Añadimos una nueva imagen: check.png.

- Y un nuevo botón en el control de usuario, a continuación del botón de borrar tarea, dentro del mismo grid:

```
<Button HorizontalAlignment="Right" Height="48"  Width="40"
        VerticalAlignment="Center"  Margin="0,0,45,0"
        Background="Transparent"
        x:Name="btnCheckTask"
        Click="btnCheckTask_Click">
    <Image Source="Assets/images layout/check.png"
           Height="40"  Margin="-10,-39,-10,0" VerticalAlignment="Top"/>
</Button>
```



Marcar una tarea como terminada

- Ahora, en el code behind del control de usuario, añadiremos el nuevo evento, su manejador y la clase de los argumentos del evento:

```
public event EventHandler<checkTaskEventArgs> checkTask;
...
private void btnCheckTask_Click(object sender, RoutedEventArgs e)
    {  if (checkTask != null){
        checkTaskEventArgs ea = new checkTaskEventArgs(list, task);
        checkTask(sender, ea);
    }
}
...
public class checkTaskEventArgs
  {  public int list { get; set; }
  public int task { get; set; }
   public checkTaskEventArgs(int l, int t) : base()
        { list =l;
        task =t;
    }
}
```

Marcar una tarea como terminada

- En el code behind de MainPage, modificamos LoadTasks:

```
if (list.tasks.Count() > 0 &&
list.tasks.Where(t=>t.finished==false).Count()>0) {
  lblXdef.Visibility = Visibility.Collapsed;
  int numTask = 0;
  foreach (cTask task in list.tasks){
    if (!task.finished) {
      TareaControl auxControl = new
TareaControl(task.text, slctList.SelectedIndex,
numTask);
      auxControl.deleteTask += new
EventHandler<deleteTaskEventArgs>(deleteTask);
      auxControl.checkTask += new
EventHandler<checkTaskEventArgs>(checkTask);
      pnlList.Children.Add(auxControl);
    }
    numTask++;
  }
} else {
  lblXdef.Visibility = Visibility.Visible;
}
```

Marcar una tarea como terminada

- Y añadimos la función 'checkTask':

```
private void checkTask(object sender, checkTaskEventArgs e)
{
    lists.ElementAt(e.list).tasks
.ElementAt(e.task).finished=true;
    LoadTasks(e.list);
}
```

Almacenamiento de datos

- Para guardar los cambios, vamos a crear una función 'saveData' que almacenará las listas y tareas en el fichero data.json.
- Vamos a invocarla en el evento 'OnCloseRequest', justo antes de salir.
  - Sería conveniente estudiar en qué momentos deberíamos guardar los datos para evitar pérdidas.
- La función tiene que almacenar los datos en un fichero ubicado en la carpeta 'Assets', la cual, con UWP, **es de sólo lectura**.
- Por tanto, para poder escribir los cambios en el fichero, primero crearemos un fichero con los nuevos datos en el espacio de almacenamiento del aplicativo y después lo moveremos a 'Assets'.
- Cosas a mejorar: verificar errores, hacer un backup de los datos anteriores (.bak), mantener el fichero 'datos.json' original, etc.

Almacenamiento de datos

- La función 'saveData' será:

```
async private void saveData() {
    StorageFolder appDir = ApplicationData.Current.LocalFolder;
    StorageFile tempFile = await
appDir.CreateFileAsync("data.json",
CreationCollisionOption.ReplaceExisting);
    await FileIO.WriteTextAsync(tempFile,
JsonConvert.SerializeObject(lists));
    StorageFolder dataDir = await
Package.Current.InstalledLocation.GetFolderAsync(@"Assets\Data
");
    await tempFile.MoveAsync(dataDir, "data.json",
NameCollisionOption.ReplaceExisting);
}
```

Almacenamiento de datos

- Finalmente, en el evento 'OnCloseRequest', invocaremos a 'saveData':

```
if (res == ContentDialogResult.Primary)
{
    saveData();
    Application.Current.Exit();
}
```

Actividad

Sube una versión personal de esta aplicación u otra (p.ej la calculadora, el monitor del sistema, u otra) usando UWP

Ampliar temas:

- I18n y L9n: https://docs.microsoft.com/en- us/windows/uwp/app-resources/localize-strings-ui-manifest
- Async y await: https://docs.microsoft.com/en- us/dotnet/csharp/programming-guide/concepts/async/
- Acceso a ficheros desde UWP: https://docs.microsoft.com/en-us/windows/uwp/files/file-access-permissions
- Toasts o notificaciones: https://docs.microsoft.com/en- us/windows/uwp/design/shell/tiles-and-notifications/adaptive-interactive-toasts

Ejercicios de ampliación, sencillos

- Ampliar el aplicativo anterior añadiendo las siguientes prestaciones:
  - Control de errores a la hora de leer/escribir en el fichero de datos.
  - Antes de guardar los datos, realizar una copia de seguridad del fichero actual llamada: 'datos.json.old'.
  - Añadir botones y código necesario para ordenar las tareas por nombre o fecha.
  - Añadir un botón que permita ver también las tareas finalizadas: este botón alternará entre ver solo las tareas pendientes o ver todas las tareas activas.
  - Añadir un botón que permita ver las tareas no activas (eliminadas) y poder recuperarlas.

Ejercicios de ampliación, menos sencillos

- Ampliar el aplicativo anterior añadiendo las siguientes prestaciones:
  - Permitir modificar una tarea, al hacer click sobre su texto se hará editable.
  - Al añadir una lista, especificar además del nombre, el color (de un ColorPicker).
    - Por tanto, se deberá cambiar el color de fondo del grid de tareas en consecuencia.
  - Al añadir una tarea, poder especificar la fecha de vencimiento (desde un DatePicker).
    - En consecuencia, en el interfaz, remarcar aquellas tareas que hayan vencido y no estén finalizadas.

Referencias

- Algunos tutoriales o documentos on line que pueden servir para ampliar lo explicado en este taller:
  - https://docs.microsoft.com/en-us/previous-versions/windows/apps/hh965329(v=win.10)
  - https://docs.microsoft.com/en-us/windows/uwp/design/layout/layout-panels
  - https://mikaelkoskinen.net/post/uwp-xaml-responsive-layout-using-grid-and-adaptivetrigger
  - https://docs.microsoft.com/en-us/windows/uwp/files/quickstart-reading-and-writing-files
  - https://docs.microsoft.com/en-us/windows/uwp/design/controls-and-patterns/dialogs-and-flyouts/index