# 3. Los principales controles

El framework .NET provee, de manera estándar, numerosos controles WPF que cubren la mayoría de necesidades relativas a la creación de una interfaz gráfica. Están definidos en la librería **PresentationFramework.dll**, en el espacio de nombres **System.Windows.Controls**.

La mayoría de estos controles poseen un juego de **propiedades comunes** que permiten gestionar su representación visual en la aplicación, así como su aspecto. A continuación, se enumeran las principales:

|  |  |
| --- | --- |
| **PROPIEDAD** | **CARACTERÍSTICAS** |
| Height y Width | Definen la altura y anchura del control. |
| HorizontalAlignment y VerticalAlignment | Definen cómo debe situarse el control en el espacio que tiene disponible horizontal y verticalmente. Los valores pueden ser Left (alineación a la izquierda), Center (centrado), Right (alineación a la derecha) y Stretch (ocupar todo el espacio disponible) para la alineación horizontal, y Top (alineación arriba), Bottom (alineación abajo), Center (alineación centrada) y Stretch para la alineación vertical. |
| Margin | Define el espacio que debe quedar libre alrededor de los límites del control. |
| Padding | Define el espacio que debe quedar libre entre el borde del control y su contenido. |
| BorderBrush | Define el color del borde del control. |
| BorderThickness | Define el grosor del borde del control. |
| Visibility | Define el estado de visibilidad del control. Existen tres valores disponibles: Visible, Hidden (el control se oculta, pero la zona que debería ocupar está disponible), Collapsed (el control se oculta, pero la zona que debería ocupar se libera). |

Los controles WPF pueden agruparse en varias categorías definidas en función de su objetivo.

## 3.1. Controles de ventanas

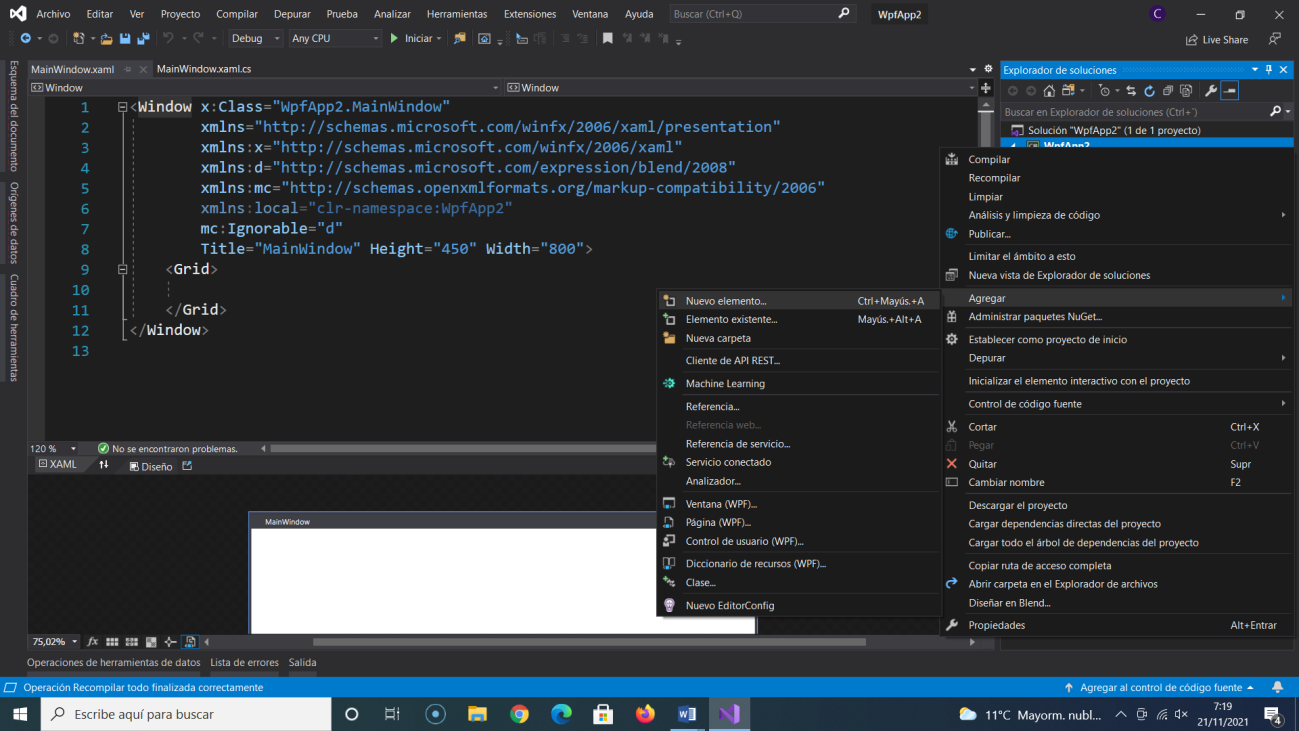
Los controles de ventanas son esenciales en una aplicación WPF. Son ellos los que contienen la interfaz gráfica que se va a crear.

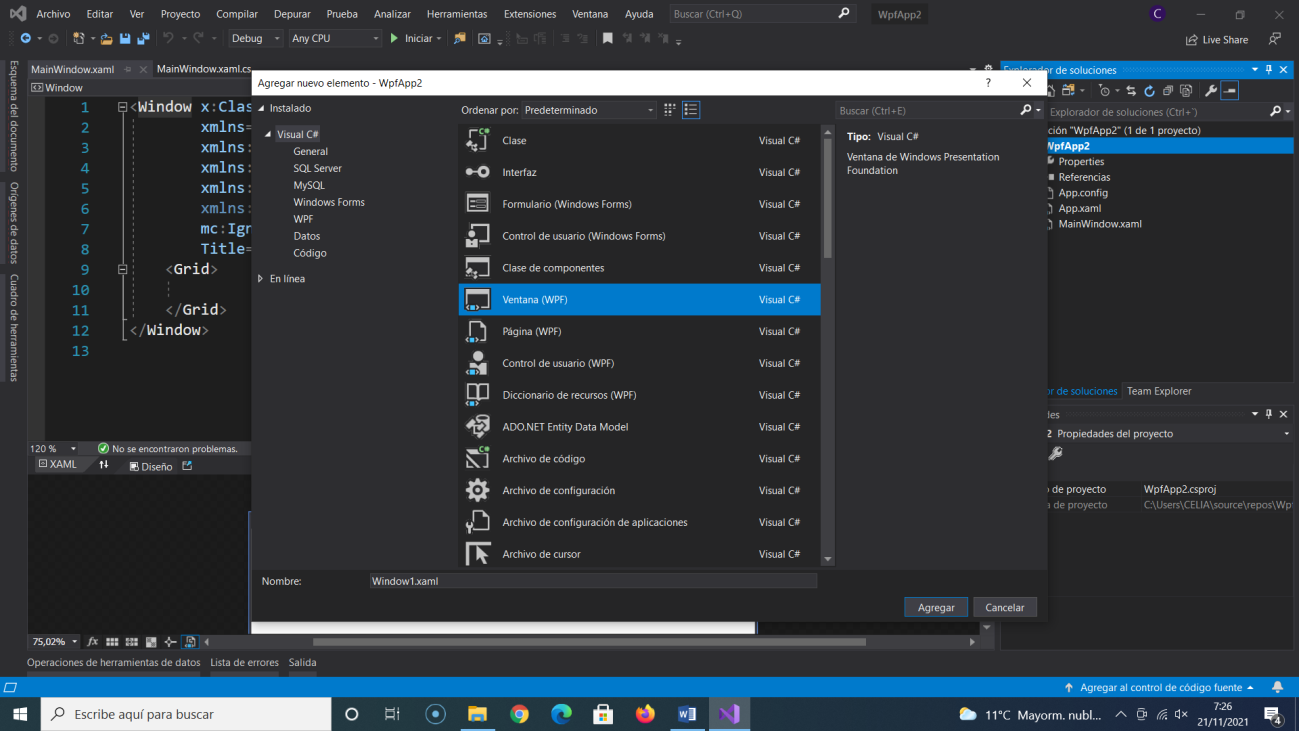
Existen dos controles estándar que pertenecen a esta categoría y se corresponden con modos de navegación distintos.

### 3.1.1. Window

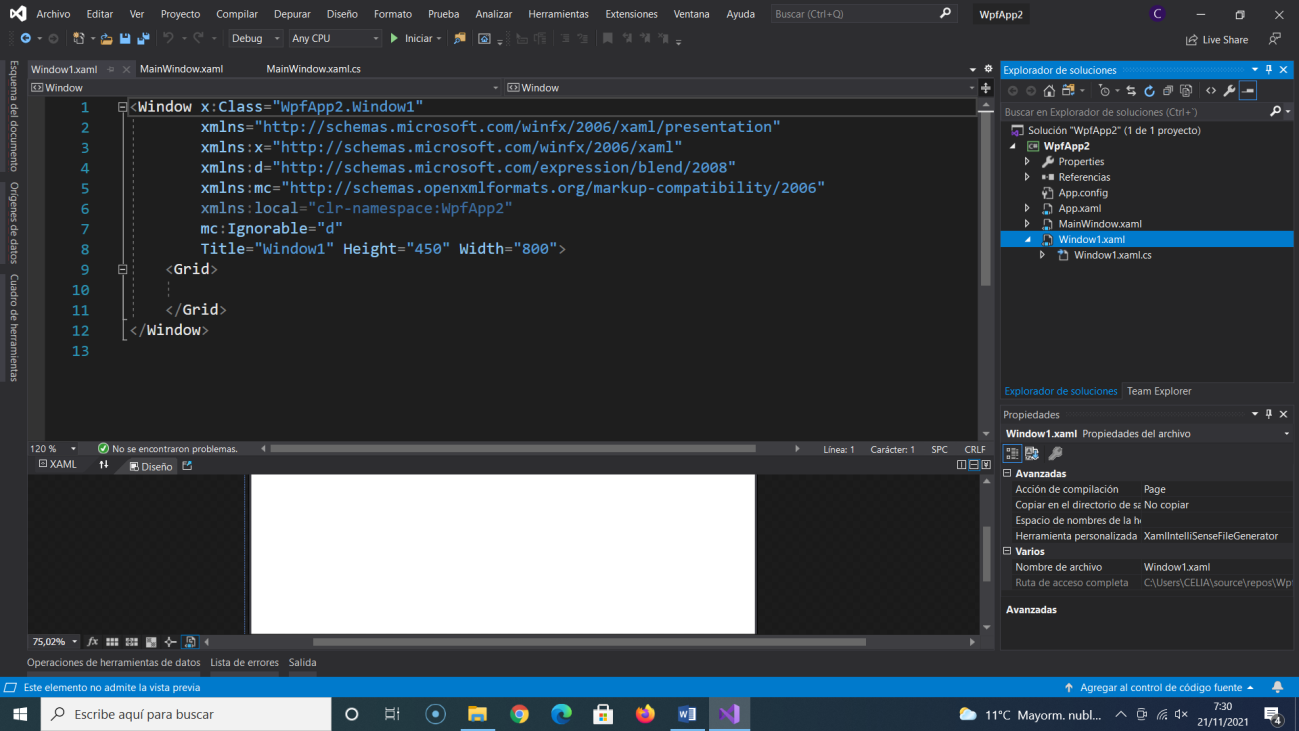
El control **Window** permite definir una ventana de aplicación. Este control de ventanas es el que se utiliza con mayor frecuencia.

Puede crearse mediante el cuadro de diálogo **Agregar** nuevo elemento (haciendo clic con el **botón derecho** sobre el **nombre del proyecto** en el **Explorador de soluciones**, y seleccionando, a continuación, Agregar – Nuevo elemento, o mediante el atajo de teclado [Ctrl][Mayús]A):





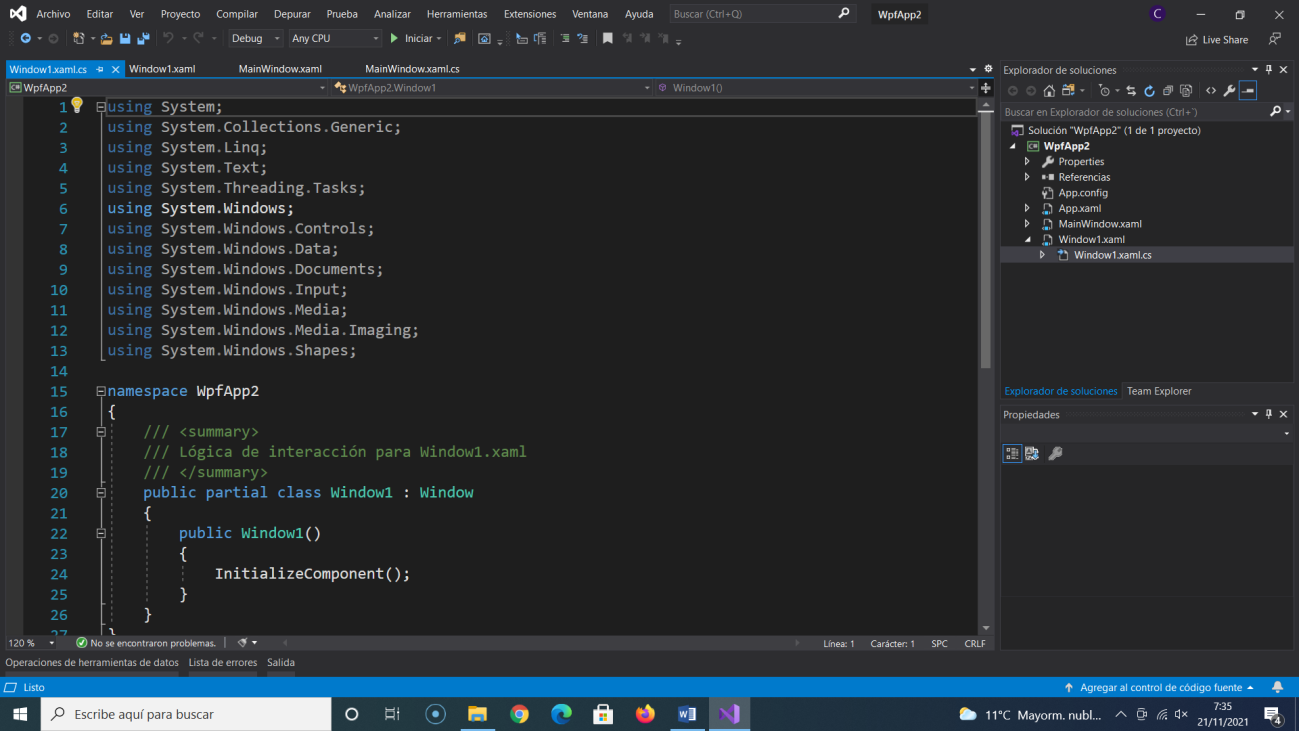
Al validar la opción Ventana (WPF) haciendo clic en el botón Agregar se crean un archivo .xaml y un archivo .xaml.cs que se integran en el proyecto.



El contenido del archivo Window1.xaml es similar al de MainWindow.xaml.

El objeto Window es la raíz del árbol XAML.

El atributo x:Class de este objeto se corresponde con la clase C# asociada a la ventana. Esta clase se encarga de lanzar la inicialización de los controles gráficos mediante su constructor. Es también en esta clase donde se asocian los controladores de eventos con los correspondientes controles de la ventana.



El control posee varias **propiedades** que permiten definir su **aspecto**, su **comportamiento** y su **ubicación**. La siguiente tabla muestra con detalle los principales:

|  |  |
| --- | --- |
| **PROPIEDAD** | **CARACTERÍSTICAS** |
| Title | Define el título de la ventana. |
| ResizeMode | Define los modos de redimensionamiento disponibles para la ventana. |
| ShowInTaskbar | Define si la ventana debe ser visible en la barra de tareas de Windows. |
| WindowState | Define si la ventana debe poder maximizarse, minimizarse o dimensionarse normalmente. |
| WindowStartupLocation | Define si la ventana debe estar centrada respecto a su ventana madre o respecto a la pantalla. |
| Topmost | Define si la ventana debe quedarse en primer plano incluso aunque la aplicación no tenga el foco. |

Para abrir una nueva ventana se instancia su método Show, o bien su método ShowDialog si la ventana debe ser modal.

Window1 window = new Window1();

window.Show();

Este control define también varios elementos, entre ellos Loaded, que permite ejecutar una sección de código cuando el contenido de la ventana se ha cargado, pero todavía no se está mostrando. Los eventos Closing y Closed se producen respectivamente antes y después de cerrar la ventana. El primero permite impedir el cierre de la ventana mediante su controlador de la siguiente manera:

Private void Window\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

e.Cancel = true;

}

La ventana de arranque de una aplicación se define en el archivo App.xaml. una de las propiedades del objeto Application descrita en este archivo es StartupUri. Recibe como valor la ruta relativa que indica qué ventana debe abrirse tras el inicio del programa.

<Application x:Class="WpfApp2.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:WpfApp2"

StartupUri="MainWindow.xaml">

<Application.Resources>

</Application.Resources>

</Application>

### 3.1.2. NavigationWindow

WPF proporciona, además, un modo de multi ventana tradicional, un modo de navegación parecido al de un navegador web. En este modo, el usuario no tiene varias ventanas a su disposición, sino una única ventana, dentro de la cual puede navegar entre varias páginas.

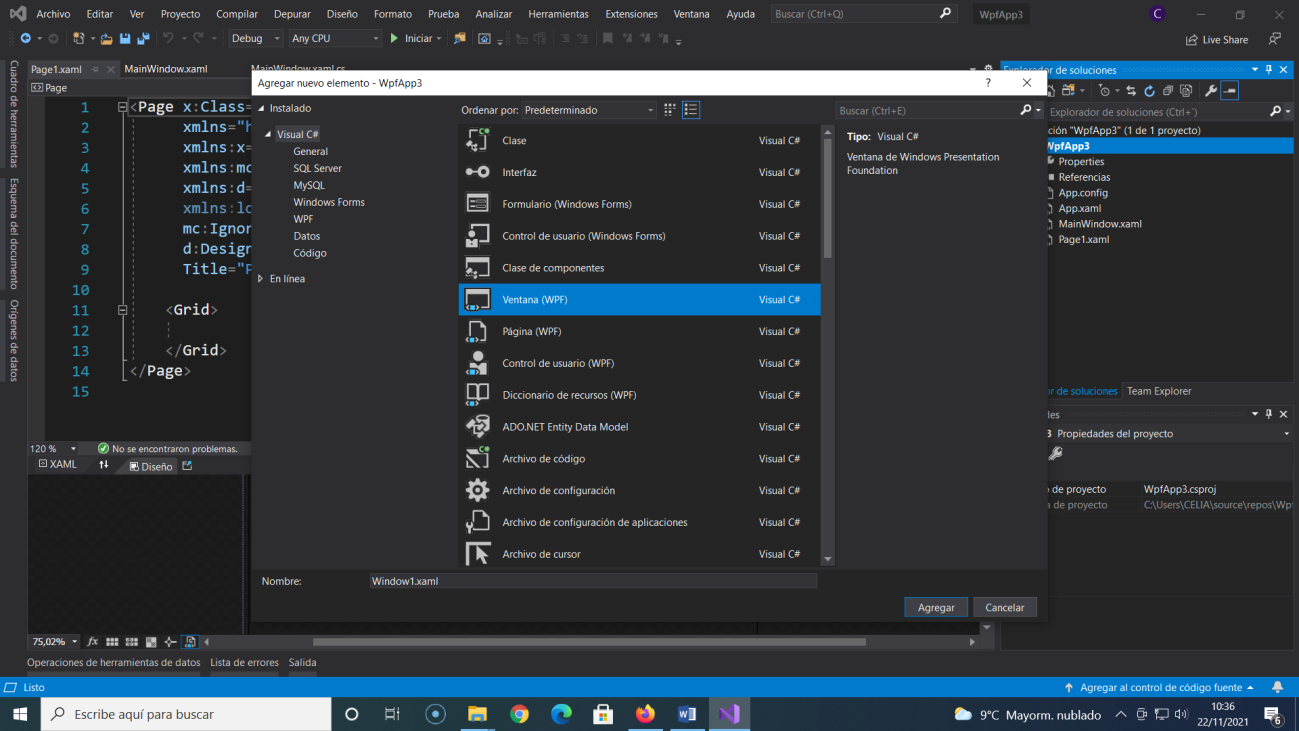
La ventana principal es de tipo NavigationWIndow y los distintos contenidos se sitúan en objetos Page. Como ocurre en un navegador web, existe un objeto NavigationWindow que dispone de dos botones Anterior y Siguiente que permite gestionar un histórico de navegación entre cada objeto Page.

NavigationWindow es una clase derivada de la clase Window que tiene acceso a todas las propiedades que hemos visto para las ventanas. Posee también propiedades propias de su modo de navegación.

|  |  |
| --- | --- |
| PROPIEDAD | CARACTERÍSTICAS |
| Source | Define la URI de la página que se muestra actualmente. |
| BackStack | Contiene el histórico de las páginas anteriores. |
| FordwardStack | Contiene la lista de páginas siguientes. |
| CanGoBack | Define, si es posible, para la ventana, navegar a una página anterior. |
| CanGoForward | Define, si es posible, para la ventana, navegar a una página posterior. |

La creación de una NavigationWindow requiere cierta manipulación, pues no existe ninguna plantilla para este tipo de objeto en Visual Studio.

* Abro la ventana **Agregar un nuevo elemento** (clic con el botón derecho sobre el proyecto y, a continuación, **Agregar – Nuevo elemento** o [Ctrl][Mayús]A) y cree la ventana (WPF).

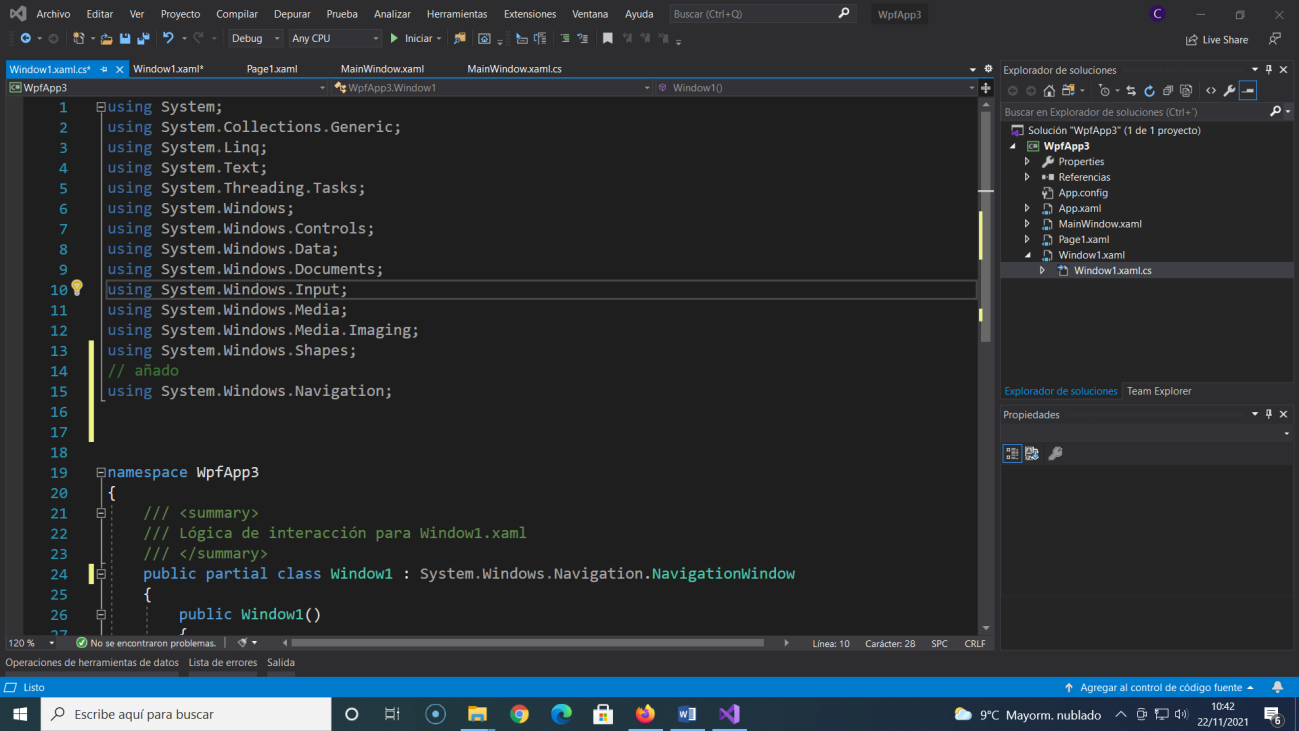


* En el archivo .xaml creado, modifico la declaración del objeto raíz: Window se convierte en NavigationWindow. Elimino las etiquetas <Grid> y </Grid> contenidas en el objeto NavigationWindow. Este componente no admite el contenido directo.
* Modificar también el archivo .xaml.cs asociado de la siguiente forma:

public partial class Window1:Window

Se convierte en:

public partial class Window1 : **NavigationWindow**



Ya es posible utilizar la ventana. Se puede asignar a la propiedad StartupUri del objeto Application definido en App.xaml con la ruta relativa que apunta a la ventana que acabamos de crear.

StartupUri=”Window1.xaml”

La creación de un objeto Page puede realizarse mediante una plantilla.

En la ventana **Agregar nuevo elemento**, elijo **Página (WPF).** Esto agrega un archivo Page1.xaml al proyecto, así como un archivo code-behind asociado.

El código XAML de esta página es relativamente parecido al de una nueva ventana:

<Page x:Class="WpfApp1.Page1"

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:WpfApp1"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800"

Title="Page1">

<Grid>

</Grid>

</Page>

El contenido de la página se mostrará en la NaviagationWindow cuando la propiedad Source de la ventana tenga como valor “Page1.xaml”. es posible modificar este valor asignando la ruta de un archivo o utilizando el método Navigate de la ventana. Este método recibe como parámetro la ruta relativa del archivo .xaml que define una página.

using System.Windows.Navigation;

namespace WpfApp3

{

/// <summary> /// Lógica de interacción para Window1.xaml

/// </summary>

public partial class Window1 : System.Windows.Navigation.NavigationWindow

{

public Window1()

{

InitializeComponent();

NavigationWindow ventana = new NavigationWindow();

ventana.Navigate("Page1.xaml");

}

}

}

