Windows Store – Creación de Aplicaciones

Aplicación de Consumo

Tabla de contenido

Información general

Descripción

Crearemos una aplicación que permita almacenar, modificar, eliminar y listar información registrada por el usuario utilizando la interface de Windows Store y aprendiendo a manejar los parámetros que VisualStudio 2012 ofrece para este tipo de desarrollos.

Expectativas y objetivos

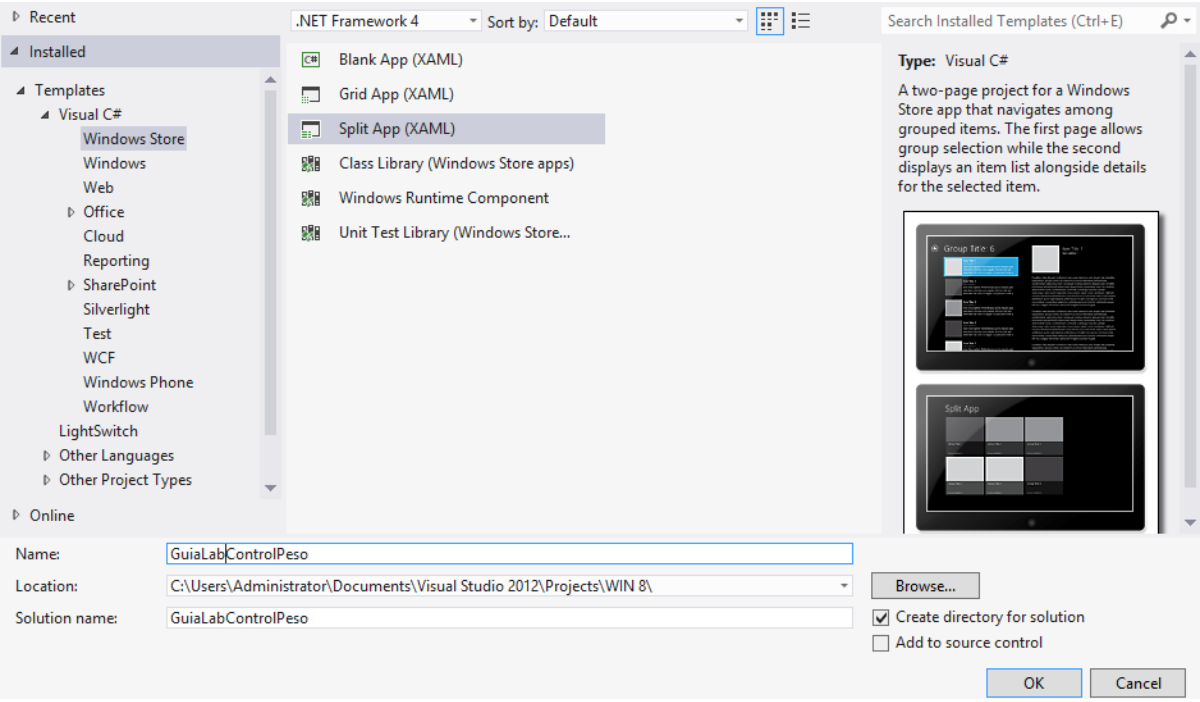
Al finalizar el estudiante tendrá la capacidad de crear su propia aplicación respetando los parámetros que se establecen para el Windows Strore, así como se comprenderá la noción de implementar y utilizar el motor de datos SQLite para aplicaciones móviles.

Material necesario

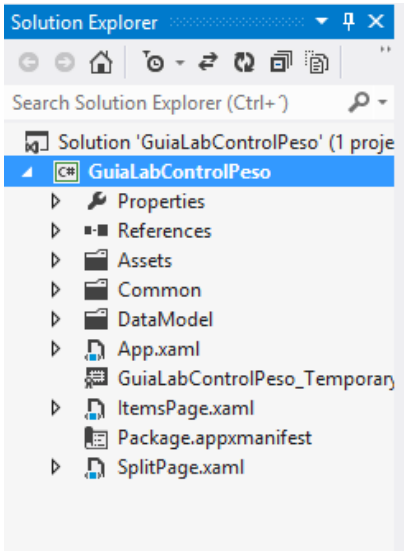
- ✓ Equipo con sistema operativo Windows 8.
- ✓ VisualStudio 2012 en cualquiera de sus versiones que permita crear aplicaciones del tipo Windows Store.
- ✓ Instalar SQLite para utilizarla en la Aplicación.
- ✓ Imágenes con las especificaciones que se darán más adelante para la personalización de la aplicación.
- ✓ Conocimientos básicos en XAML y C#.

Creación de aplicación Windows Store

Para iniciar crearemos una aplicación en Visual Studio 2012 utilizando la plantilla de **Windows Store -> Split App (XAML)** con C#.



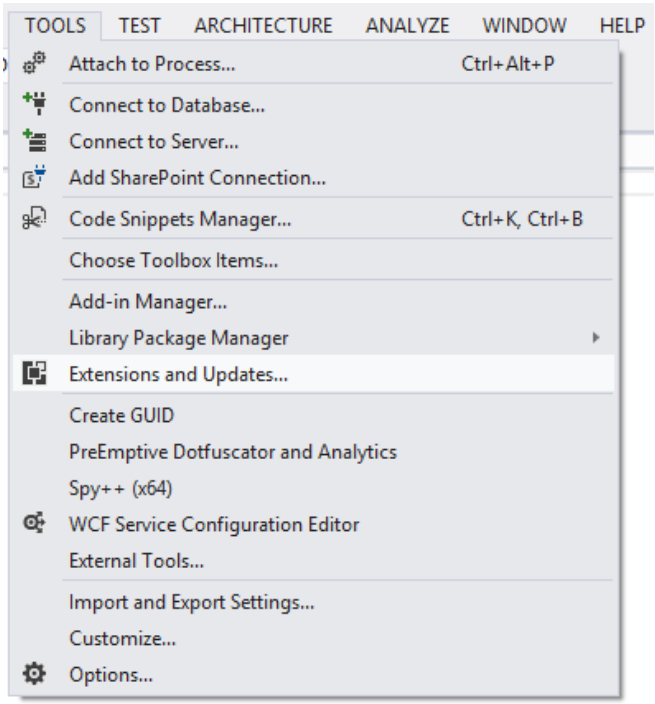
Esto nos creará una plantilla en la que se cargaran por defecto diferentes archivos que nos ayudaran para el desarrollo de nuestra aplicación.



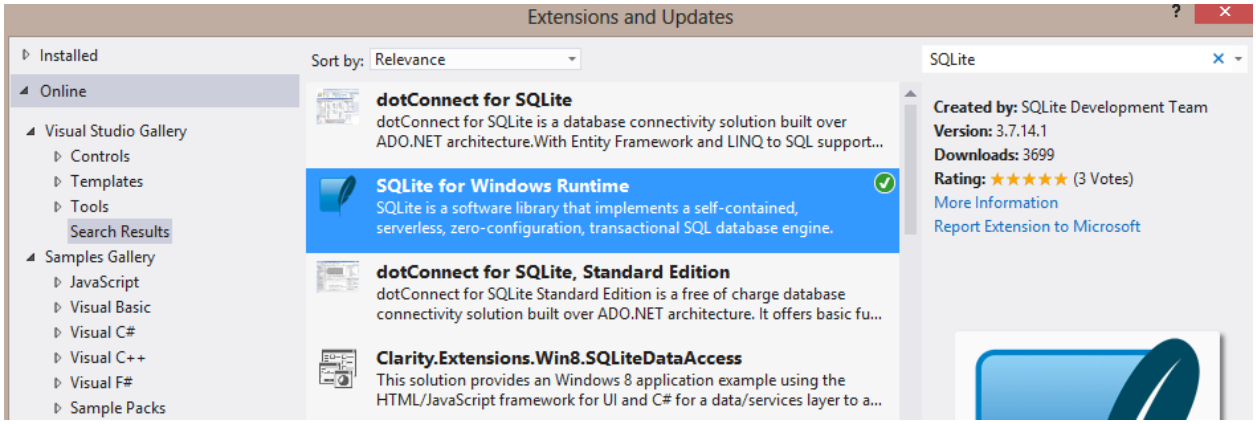
Implementación SQLite

Este motor de base de datos permitirá almacenar la información en el Application Storage local del dispositivo donde se instalará la aplicación, para esto lo primero que debemos hacer es instalarla en nuestro VisualStudio (Esto solo se hace una vez) y luego habilitarla para nuestra aplicación (Esto se hace por cada aplicación en la que se desee implementar SQLite).

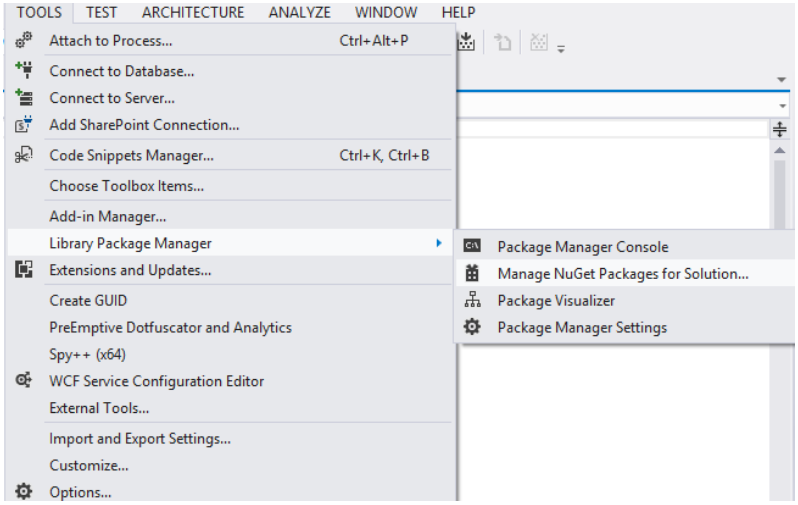
Para esto nos vamos a **Herramientas – Extensiones y Actualizaciones** (**Tools – Extensions and Updates**) en nuestro VisualStudio.



Acá se abrirá la ventana para descargar e instalar el **SQLite for Windows Runtime** en la ruta **Online – Tools** y en la casilla de búsqueda podemos escribir la extensión mencionada, la seleccionamos y hacemos clic sobre el botón **install** en la parte inferior de la ventana.

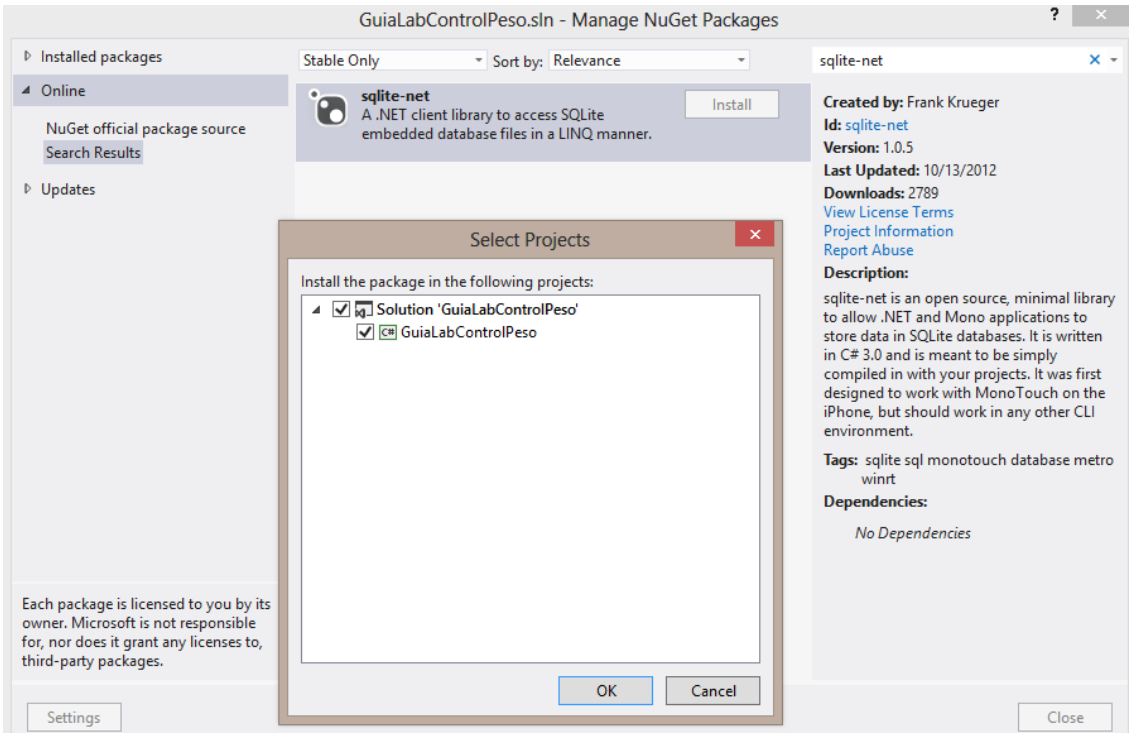


El siguiente paso consiste en habilitar el SQLite en nuestra aplicación, para esto abrimos el panel de administración de paquetes NuGet en **Tools – Library Package Manager – Manage NuGet Packages for Solutions**.

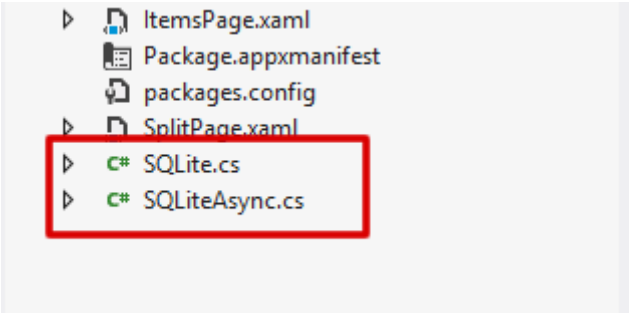


Esto nos abrirá el administrador de paquetes instalados, acá seleccionamos la opción **online** y **en el buscador escribimos sqlite-net**, al hacer clic sobre el botón **install** se abrirá una ventana

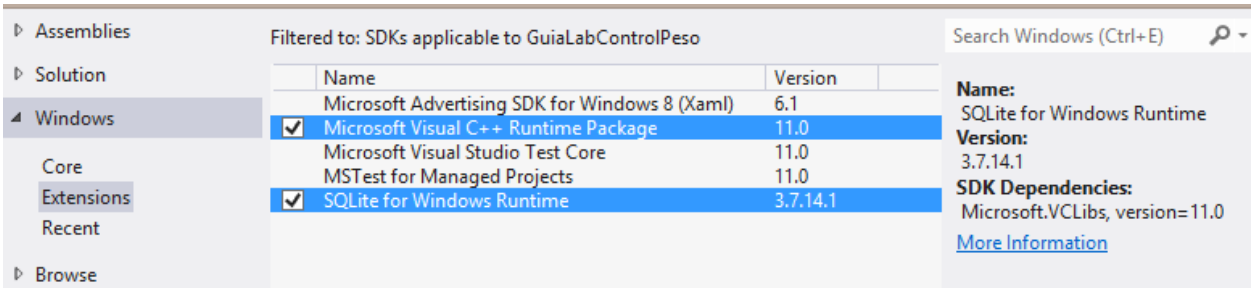
en la que debe hacer check sobre el nombre de nuestra aplicación y luego hacemos clic sobre el botón **OK**.



Al finalizar aparecerán dos nuevas clases que corresponden a la implementación del SQLite que hicimos



Para finalizar agregamos dos referencias para que el SQLite corra en nuestra aplicación sin inconvenientes:



Creación de estructura de Almacenamiento de Información

Para estructurar el esquema de datos vamos a crear dentro de la **carpeta DataModel** dos clases una la llamaremos con el nombre de la tabla que queremos crear (si son varias tablas generamos una clase por tabla), para nuestro ejemplo se creó la clase **RegistroPeso**. Que tiene el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GuiaLabControlPeso.DataModel
{
    class RegistroPeso
    {
        [SQLite.PrimaryKey, SQLite.AutoIncrement]
        public Int32 Id { get; set; }

        public Decimal Peso { get; set; }
        public DateTime Fecha { get; set; }
        public String Detalles { get; set; }
    }
}
```

La línea de **`[SQLite.PrimaryKey, SQLite.AutoIncrement]`** indica que la propiedad **`Int32`** **Id** será la llave primaria y que será un entero aoutoincremental dentro de nuestra base de datos.

También crearemos una clase que en este ejemplo llamaremos **Contexto** la cual manejará la información y las consultas efectuadas a nuestra base de datos; estas consultas serán realizadas mediante LinQ. El código base es el siguiente:

```
using SQLite;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GuiaLabControlPeso.DataModel
{
    class Contexto : SQLiteConnection
    {
```

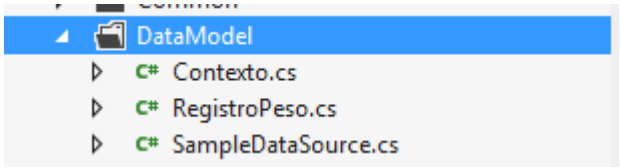
```
        //Sobreescribimos el constructor agregando la ruta donde estará
almacenado el archivo
        public Contexto(String path)
            : base(path)
        {
            //Se crea la tabla RegistroPeso en SQLite
            CreateTable<RegistroPeso>();
        }

        //Agregar un nuevo registro
        public Int32 AddRegistro(RegistroPeso registro)
        {
            return Insert(registro);
        }

        //Editar un registro
        public Int32 EditRegistro(RegistroPeso registro)
        {
            return Update(registro);
        }

        //Cargar la información de todos los registros almacenados
        public IEnumerable<RegistroPeso> GetRegistros()
        {
            return from c in Table<RegistroPeso>()
                   select c;
        }

        //Carga un registro específico pasando el ID y se extrae a través de
un objeto tipo RegistroPeso
        public RegistroPeso GetRegistro(Int32 idRegistro)
        {
            RegistroPeso oRegistro = new RegistroPeso();

            var registro = from r in Table<RegistroPeso>()
                           where r.Id == idRegistro
                           select r;

            foreach (var r in registro)
            {
                oRegistro.Peso = r.Peso;
                oRegistro.Fecha = r.Fecha;
                oRegistro.Detalles = r.Detalles;
            }

            return oRegistro;
        }

        //Borra un registro específico
        public Int32 DeleteRegistro(Int32 idRegistro)
```

```
        {
            return Delete<RegistroPeso>(idRegistro);
        }

        //Conteo de total de registros
        public Int32 CountRegistros()
        {
            var registros = from c in Table<RegistroPeso>()
                            select c;

            return registros.Count();
        }
    }
}
```

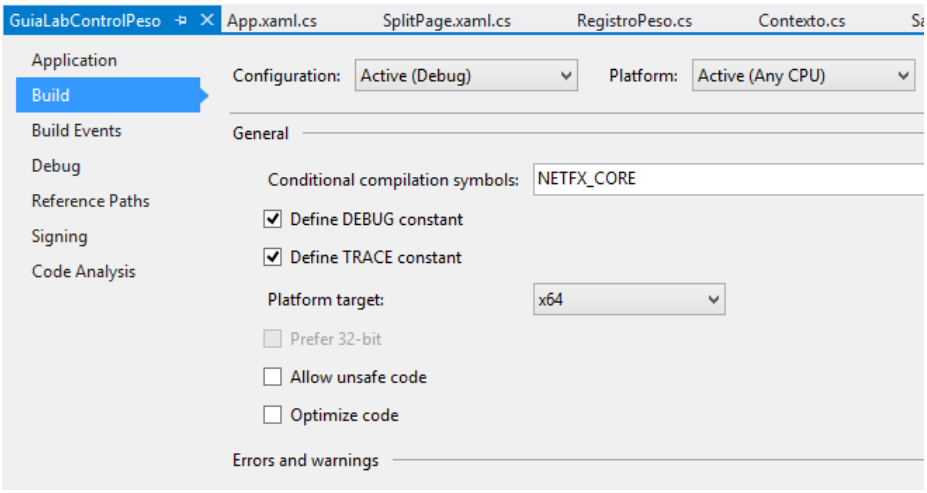
Si se requiere ingresar otra tabla, se crea la clase correspondiente a la tabla (por ejemplo **OtraTabla**) con las propiedades que se requieran y en la clase **Contexto** se podrá agregar la nueva tabla de la siguiente manera:

```
public Contexto(String path)
            : base(path)
        {
            //Se crea la tabla RegistroPeso en SQLite
            CreateTable<RegistroPeso>();
            //Se crea la tabla OtraTabla en SQLite
            CreateTable<OtraTabla>();
        }
```

SQLite nos crea cada tabla si esta no existe, si existe no ejecutará esta parte, esa es una de las ventajas de utilizar este motor de datos para nuestras App móviles. Cabe aclarar que es necesario crear los métodos correspondientes a las nuevas tablas creadas, todo utilizando sentencias LinQ como hemos mencionado antes. Al finalizar se verá así:



Para evitar inconvenientes se recomienda cambiar la plataforma seleccionada a X64 en las propiedades de nuestra aplicación en la opción Build.


GuiaLabControlPeso
App.xaml.cs
SplitPage.xaml.cs
RegistroPeso.cs
Contexto.cs
Application
Build
Build Events
Debug
Reference Paths
Signing
Code Analysis
Configuration: Active (Debug)
Platform: Active (Any CPU)
General
Conditional compilation symbols: NETFX_CORE
Define DEBUG constant
Define TRACE constant
Platform target: x64
Prefer 32-bit
Allow unsafe code
Optimize code
Errors and warnings

Configuración del App.xaml

Acá en este punto debemos iniciar a construir nuestra aplicación y para esto debemos abrir el archivo **App.xaml** y modificamos el nombre que verán los usuarios de nuestra aplicación en el tag con el x:Key=”**AppName”**:

```
<x:String x:Key="AppName">Laboratorio Guía Control de Peso</x:String>
```

Para nuestro ejemplo solamente utilizaremos la página **SplitPage.xaml**, por esta razón haremos una pequeña modificación en **App.xaml.cs**, (Aproximadamente en la línea 80 de nuestro código)

Modificaremos la página de inicio de nuestra aplicación cambiando en la línea de código: **ItemPage**

```
if (!rootFrame.Navigate(typeof(ItemsPage), "AllGroups"))
```

Por **SplitPage**

```
if (!rootFrame.Navigate(typeof(SplitPage), "AllGroups"))
```

Página Principal

Ahora abrimos el archivo SplitPage.xaml que se convierte en nuestra página principal, aca iniciaremos unos ajustes tanto en la interface de usuario en XAML, como en el código que vamos a utilizar.

Interface de Usuario XAML

Lo primero que haremos será buscar el Grid denominado "titlePanel" `<Grid x:Name="titlePanel">` y dentro de este Grid haremos una modificación al TextBlock "pageTitle", con el fin de lograr que el título que aparezca es el que agregamos en la página App.xaml (AppName), el código del TextBlock quedaría de la siguiente manera:

```
<TextBlock x:Name="pageTitle" Grid.Column="1" Text="{StaticResource AppName}" Style="{StaticResource PageHeaderTextStyle}"/>
```

Ahora podremos ver que existe un control del tipo **ListView** llamado "**itemListView**" que es el que desplegará una lista de acuerdo a los datos que hayamos ingresado en nuestra base de datos de acuerdo a los parámetros que establezcamos.

Para poder ver la información como necesitamos vamos a realizar un ajuste en dos pasos, el primero es cambiar el parámetro **ItemTemplate** del ListView por el valor **SplitListItemTemplate**, nuestro código deberá quedar así:

```
ItemTemplate="{StaticResource SplitListItemTemplate}"
```

Acá es donde se complica un poco el asunto ya que lo que hemos hecho es agregarle un nombre de algo que no existe, por eso para completar este paso deberemos crear algo denominado **DataTemplate** que nos permitirá llenar la información de acuerdo a los parámetros que necesitamos. Este **DataTemplate** lo podremos agregar en dentro del área **<Page.Resources>** de la página o en el **App.xaml** (Recomendado) debajo del **AppName** que agregamos al inicio.

El código completo sería el siguiente:

```
<DataTemplate x:Key="SplitListItemTemplate">
                <Grid Width="450" >
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition Width="1*"/>
                    </Grid.ColumnDefinitions>
                    <Grid.Background>
                        <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                            <GradientStop Color="#FFA83609"/>
                            <GradientStop Color="#FF7D2505" Offset="1"/>
                        </LinearGradientBrush>
                    </Grid.Background>
```

```
                    <StackPanel Grid.Column="1" HorizontalAlignment="Left"
Margin="12,0,0,0">
                        <TextBlock Text="{Binding Fecha}" MaxHeight="56"
TextWrapping="Wrap" FontSize="20"/>
                    </StackPanel>
                </Grid>
            </DataTemplate>
```

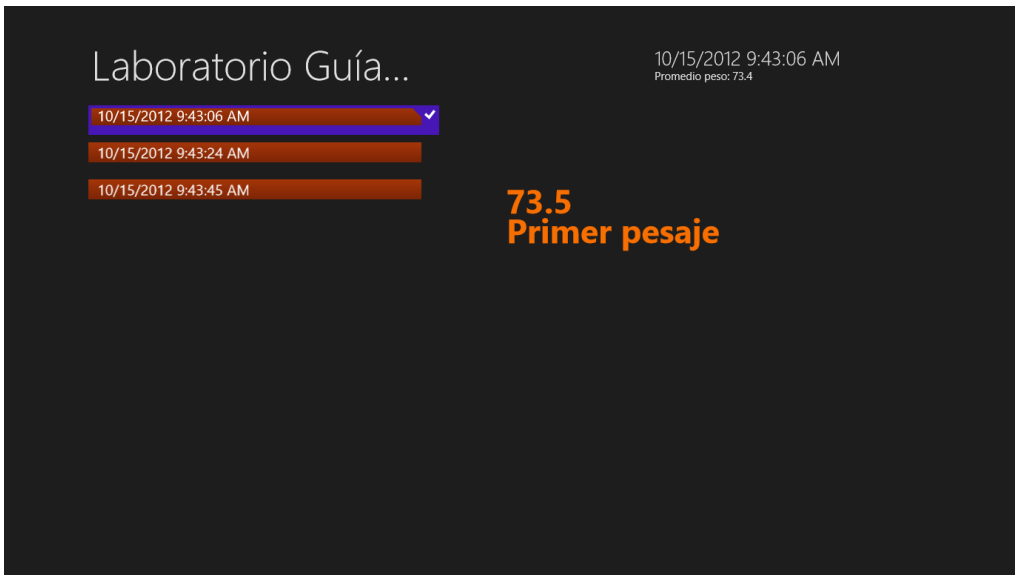
Regresando a nuestra SplitPage, ahora ajustaremos un poco el espacio en donde se verán los datos de acuerdo al ítem seleccionado Estos ajustes los realizaremos en el Grid "itemDetailGrid" que por defecto trae una imagen, un título y un subtítulo del elemento que se ha seleccionado de una lista que será desplegada gracias a un control que se encuentra un poco más arriba llamado "itemListView".

Para que muestre los datos que queremos para esta aplicación se hicieron unos ajustes, primero se comentariza el espacio en donde se muestra una imagen (no se utilizará en este momento), por otra parte se quitará la propiedad Text del itemSubtitle ya que le pasaremos la información de manera dinámica por código. Por último agregamos espacios en donde mostraremos la información necesaria y le pasaremos unos estilos creados por nosotros y que serán agregados dentro del App.xaml (estos estilos aparecen listados en la parte final de ésta guía). El código completo del itemDetailGrid sería el siguiente:

```
<Grid x:Name="itemDetailGrid" Margin="0,60,0,50">
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition Height="*"/>
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto"/>
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>

                <!--<Image Grid.Row="1" Margin="0,0,20,0" Width="180"
Height="180" Source="{Binding Image}" Stretch="UniformToFill"
AutomationProperties.Name="{Binding Title}"/>-->
                <StackPanel x:Name="itemDetailTitlePanel" Grid.Row="1"
Grid.Column="1">
                    <TextBlock x:Name="itemTitle" Margin="0,-10,0,0"
Text="{Binding Fecha}" Style="{StaticResource SubheaderTextStyle}"/>
                    <TextBlock x:Name="itemSubtitle" Margin="0,0,0,20"
Style="{StaticResource SubtitleTextStyle}"/>
                </StackPanel>
                <TextBlock Grid.Row="2" Grid.ColumnSpan="2" Margin="0,20,0,0"
Text="{Binding Peso}" Style="{StaticResource ThisTextStyle}"/>
```

```
                <TextBlock Grid.Row="3" Grid.ColumnSpan="2" Margin="0,20,0,0"
Text="{Binding Detalles}" Style="{StaticResource ThisTextStyle}"/>
 </Grid>
```

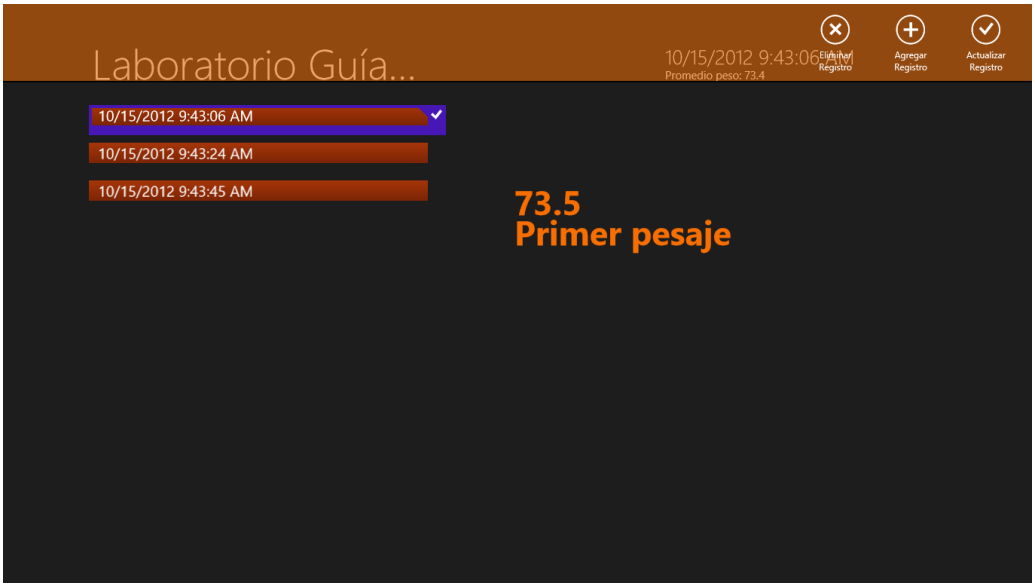
Cuando agreguemos el código en C# nuestra aplicación luciría algo así como:



Ahora agregaremos un elemento Page.TopAppBar que nos permitirá navegar en los diferentes formularios que crearemos para el manejo de nuestros datos, el siguiente código deberá agregarse inmediatamente después del tag de cierre de Page.Resources:

```
<Page.TopAppBar>
        <AppBar Padding="10,0,10,0" Background="#88F76E01">
            <Grid>
                <StackPanel Orientation="Horizontal"
HorizontalAlignment="Right">
                    <Button Click="DelRegistro_Click"
HorizontalAlignment="Right" Style="{StaticResource
WebViewAppBarDeleteStyle}"/>
                    <Button Click="AddRegistro_Click"
HorizontalAlignment="Right" Style="{StaticResource WebViewAppBarAddStyle}"/>
                    <Button Click="UpdRegistro_Click"
HorizontalAlignment="Right" Style="{StaticResource
WebViewAppBarModifyStyle}"/>
                </StackPanel>
            </Grid>
        </AppBar>
    </Page.TopAppBar>
```

Después de agregar el código correspondiente al evento clic de cada uno de los tres botones, nuestra aplicación debería verse de la siguiente manera al hacer clic derecho del mouse en nuestra primera página:



Nota: para que quede completo el esquema reemplace en el área Snapped el valor `Standard80ItemTemplate` por `NarrowListSplitTemplate` que corresponde a uno de los DataTemplate que se agregaran a nuestra aplicación.

Código C#

Para que el ListView muestre la información de los registros guardados es necesario modificar el evento LoadState de nuestra SplitPage, modificando tanto la información como la manera de mostrarla en la página. Se cambiará el código:

```
var group = SampleDataSource.GetGroup((String)navigationParameter);
this.DefaultViewModel["Group"] = group;
this.DefaultViewModel["Items"] = group.Items;
```

por este otro:

```
var rutaBD =
Path.Combine(Windows.Storage.ApplicationData.Current.LocalFolder.Path,
"ControlPeso.sqlite");
Contexto oContext = new Contexto(rutaBD);

itemSubtitle.Text = "Promedio peso: " + oContext.PromedioPeso().ToString();

if (oContext != null)
```

```
{
//this.DefaultViewModel["Group"] = oContext.GetRegistros();
this.DefaultViewModel["Items"] = oContext.GetRegistros();
}
```

En donde nos estamos comunicando con la base de datos creando la ruta que nos pide la clase Contexto.cs que creamos y le asignamos el listado de información al modelo de datos por defecto que toma la página (la plantilla ya maneja este esquema), esto hará que los datos se muestren inmediatamente si ejecutamos la aplicación en este momento.

En este mismo evento LoadState comentarizamos el código final para evitar inconvenientes de cargue de datos cuando naveguemos de una página a esta (para la presente guía no utilizaremos este fragmento de código). El código que comentarizaremos corresponde al último if…else dentro del evento y debería quedar de la siguiente manera:

```
if (pageState == null)
{
    this.itemListView.SelectedItem = null;
    // When this is a new page, select the first item automatically unless
logical page
    // navigation is being used (see the logical page navigation #region
below.)
        if (!this.UsingLogicalPageNavigation() && this.itemsViewSource.View
!= null)
        {
                    this.itemsViewSource.View.MoveCurrentToFirst();
        }
    }
            else
    {
   // Restore the previously saved state associated with this page
   //if (pageState.ContainsKey("SelectedItem") && this.itemsViewSource.View
!= null)
  //{
      //    var selectedItem =
      RegistroPeso.GetItem((String)pageState["SelectedItem"]);
  //    this.itemsViewSource.View.MoveCurrentTo(selectedItem);
  //}
}
```

En el evento SaveState debemos modificar el tipo de valor que carga la variable selectedItem, ya que viene con un valor de ejemplo, la línea de código modificaca debería quedar de la siguiente manera:

```
var selectedItem = (RegistroPeso)this.itemsViewSource.View.CurrentItem;
```

Ahora agregamos 3 eventos clic que corresponden a los tres botones creados en Page.TopAppBar, para agregar, editar o eliminar un registro:

Agregar un registro:

```
private void AddRegistro_Click(object sender, RoutedEventArgs e)
{
       this.Frame.Navigate(typeof(AddRegistro));
}
```

En donde navegaremos a la página que crearemos para agregar un registro.

Editar un registro:

```
private void UpdRegistro_Click(object sender, RoutedEventArgs e)
{
       RegistroPeso registro = (RegistroPeso)this.itemListView.SelectedItem;
       this.Frame.Navigate(typeof(EditarRegistro), registro.Id);
}
```

En donde tomamos el id del registro y lo pasamos a la página EditarRegistro que debemos crear para actualizar datos de nuestros registros.

Para eliminar un registro:

```
private void DelRegistro_Click(object sender, RoutedEventArgs e)
{
      //Se crea un objeto
      RegistroPeso registro = (RegistroPeso)this.itemListView.SelectedItem;

      //Carga la ruta y crear el objeto contexto
      var rutaBD =
Path.Combine(Windows.Storage.ApplicationData.Current.LocalFolder.Path,
"ControlPeso.sqlite");
      Contexto oContext = new Contexto(rutaBD);

      //Borra y devuelve el valor de las líneas afectadas
      Int32 borrado = oContext.DeleteRegistro(registro.Id);

      //si afectó una linea actualiza los datos que muestra el ListView
      if (borrado > 0)
      {
            this.itemListView.InitializeViewChange();

            if (oContext != null)
            {
                  //this.DefaultViewModel["Group"] = oContext.GetRegistros();
```

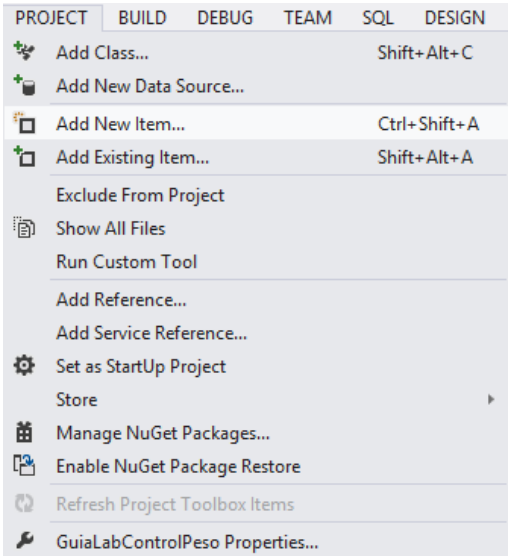
```
                this.DefaultViewModel["Items"] = oContext.GetRegistros();
            }
        }
        //Actualiza la información del promedio de peso
        itemSubtitle.Text = "Promedio peso: " +
oContext.PromedioPeso().ToString();
}
```

El código anterior elimina el registro seleccionado en el ListView y al mismo tiempo actualiza la información que se ve en el control.

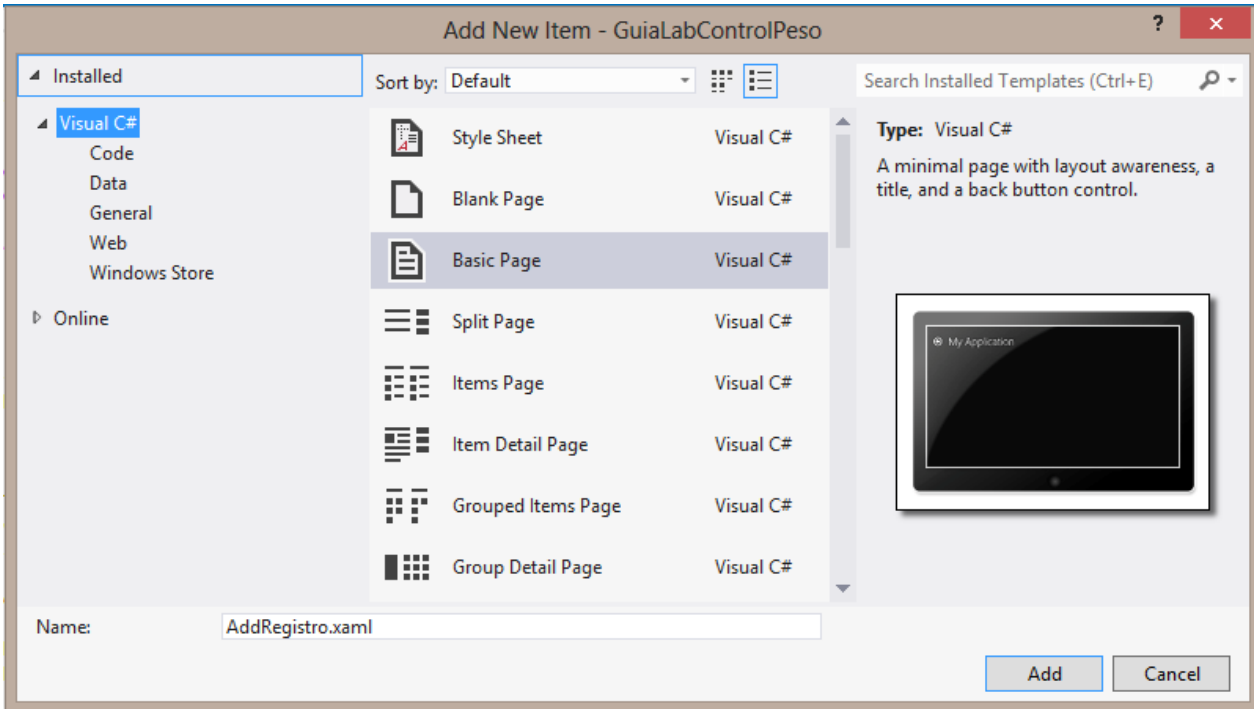
Ya teniendo la base de código se pasaremos a crear los formularios de ingreso de registros y de edición de registros lo que nos permitirá terminar con los procedimientos base de nuestra guía.

Formulario de ingreso de datos

Para crear nuestro formulario de ingreso de registros podemos hacerlo haciendo clic sobre la opción proyectos en la parte superior y seleccionamos la opción correspondiente para agregar un nuevo ítem: Poject – Add New Item…



En la ventana que se abre seleccionamos la opción de página básica (BasicPage), ya que es una plantilla que solo requiere unas pequeñas modificaciones para poder integrarla a nuestra aplicación. La llamaremos AddRegistro.xaml:



Interface de Usuario XAML

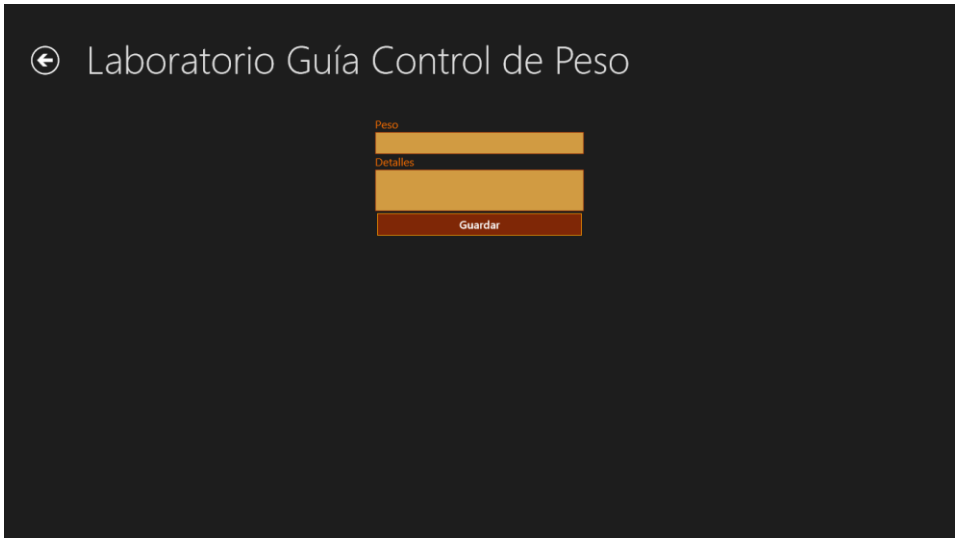
En el área <Page.Resources> quitamos la referencia AppName ya que el nombre de la aplicación la estamos cargando desde la App.xaml.

Agregamos un control StackPanel después del Grid correspondiente al título, dentro de este StackPanel agregaremos el formulario de captura de información, que tiene incluido un botón con el evento correspondiente. A este formulario le hemos agregado algunos estilos que deberán ser agregados el App.xaml y que listamos al final de la presente guía.

El código de este StackPanel es el siguiente:

```
<StackPanel Width="300" Grid.Row="1" Orientation="Vertical">
                <TextBlock x:Name="txtRespuesta" Style="{StaticResource
FormsTextStyle}" Foreground="White" />
                <TextBlock TextWrapping="Wrap"
                           Text="Peso"
                           Style="{StaticResource FormsTextStyle}"/>
                <TextBox x:Name="txtPeso"
                         TextWrapping="Wrap"
                         Style="{StaticResource FormsTextBoxStyle}"/>
                <TextBlock TextWrapping="Wrap"
                           Text="Detalles"
                           Style="{StaticResource FormsTextStyle}"
                           />
                <TextBox x:Name="txtDetalles"
                         TextWrapping="Wrap"
                         AllowDrop="True"
                         Height="60"
                         AcceptsReturn="True"
                         ScrollViewer.VerticalScrollBarVisibility="Auto"
                         Style="{StaticResource FormsTextBoxStyle}"/>

                <Button x:Name="btnGuardar" Content="Guardar"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
Click="btnGuardar_Click" Style="{StaticResource FormsButtonStyle}"/>

</StackPanel>
```

Lo que se verá después de aplicar los estilos y el código en C# será:



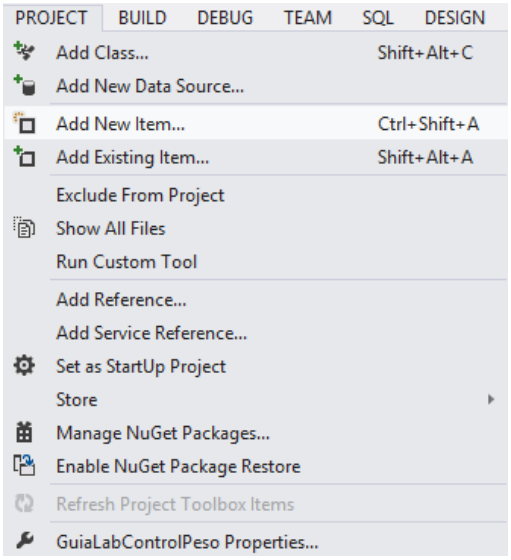
Código C#

El código que agregaremos será el correspondiente al evento del botón:

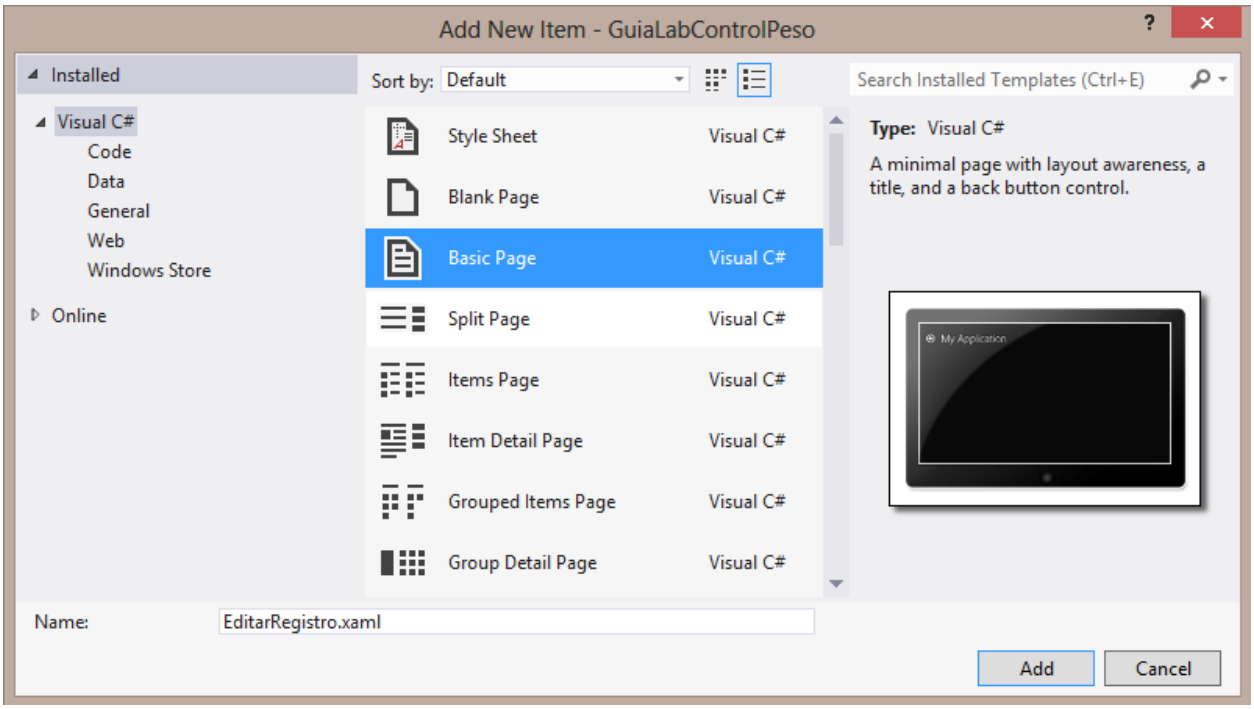
```
private void btnGuardar_Click(object sender, RoutedEventArgs e)
{
      //Objeto registro
      RegistroPeso oRegistro = new RegistroPeso();

      //Se asignan los valores a la instancia de la clase
      oRegistro.Peso = Decimal.Parse(this.txtPeso.Text);
      oRegistro.Fecha = DateTime.Now;
      oRegistro.Detalles = this.txtDetalles.Text;

      //cargamos la ruta de la base de datos y se crea el objeto contexto
      var rutaBD =
Path.Combine(Windows.Storage.ApplicationData.Current.LocalFolder.Path,
"ControlPeso.sqlite");
      Contexto oContext = new Contexto(rutaBD);

      //Se ejecuta el método de agregar registro y se recupera el número de
lineas afectadas
      Int32 registro = oContext.AddRegistro(oRegistro);

      //Mensaje si se pudo guardar el dato
      if (registro > 0)
      {
            txtRespuesta.Text = "Registro Guardado";
      }
}
```

Formulario actualización de datos

Para crear nuestro formulario de ingreso de registros podemos hacerlo haciendo clic sobre la opción proyectos en la parte superior y seleccionamos la opción correspondiente para agregar un nuevo ítem: Poject – Add New Item…



En la ventana que se abre seleccionamos la opción de página básica (BasicPage), ya que es una plantilla que solo requiere unas pequeñas modificaciones para poder integrarla a nuestra aplicación. La llamaremos EditarRegistro.xaml:



Interface de Usuario XAML

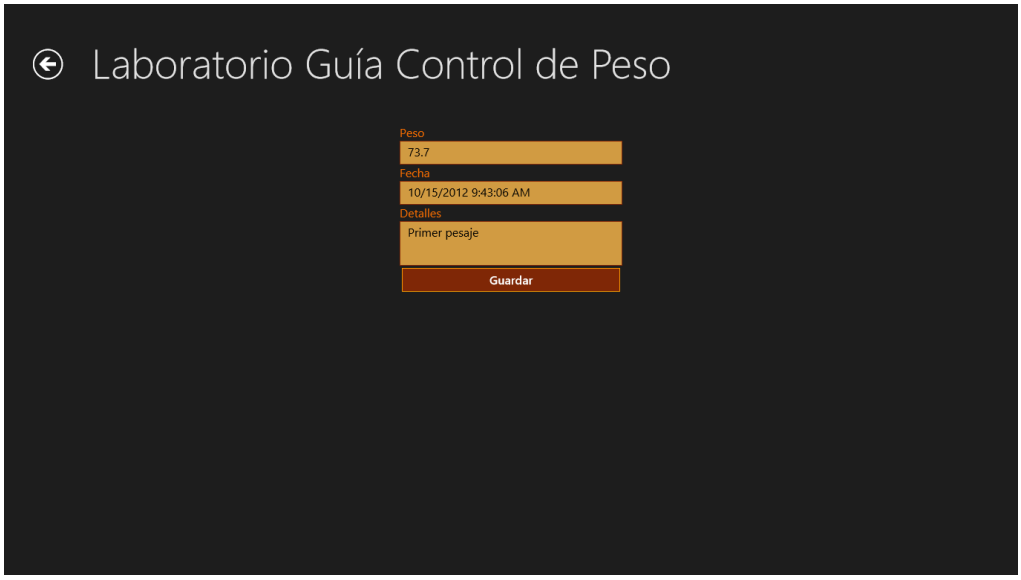
En el área <Page.Resources> quitamos la referencia AppName ya que el nombre de la aplicación la estamos cargando desde la App.xaml.

Agregamos un control StackPanel después del Grid correspondiente al título, dentro de este StackPanel agregaremos el formulario de captura de información, que tiene incluido un botón con el evento correspondiente. A este formulario le hemos agregado algunos estilos que deberán ser agregados el App.xaml y que listamos al final de la presente guía.

El código de este StackPanel es el siguiente:

```
<StackPanel Width="300" Grid.Row="1" Orientation="Vertical">
            <TextBlock x:Name="txtRespuesta" Style="{StaticResource
FormsTextStyle}" Foreground="White" />
            <TextBlock TextWrapping="Wrap"
                       Text="Peso"
                       Style="{StaticResource FormsTextStyle}"/>
            <TextBox x:Name="txtPeso"
                     TextWrapping="Wrap"
                     Style="{StaticResource FormsTextBoxStyle}"/>
            <TextBlock TextWrapping="Wrap"
                       Text="Fecha"
                       Style="{StaticResource FormsTextStyle}"/>
            <TextBox TextWrapping="Wrap"
                     x:Name="txtFecha"
                     Style="{StaticResource FormsTextBoxStyle}"/>
            <TextBlock TextWrapping="Wrap"
                       Text="Detalles"
                       Style="{StaticResource FormsTextStyle}"
                       />
            <TextBox x:Name="txtDetalles"
                     TextWrapping="Wrap"
                     AllowDrop="True"
                     Height="60"
                     AcceptsReturn="True"
                     ScrollViewer.VerticalScrollBarVisibility="Auto"
                     Style="{StaticResource FormsTextBoxStyle}"/>

            <Button x:Name="btnGuardar" Content="Guardar"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
Click="btnEditar_Click" Style="{StaticResource FormsButtonStyle}"/>
```

Lo que se verá después de aplicar los estilos y el código en C# será:



Código C#

Para poder cargar la información dentro de nuestro formulario de edición de datos lo primero que haremos es crear una variable de tipo entero que llamaremos idRegistro al inicio de la clase:

```
public sealed partial class EditarRegistro :
GuiaLabControlPeso.Common.LayoutAwarePage
{
     Int32 idRegistro;
```

Lo siguiente será cargar la información de acuerdo al ítem que hayamos seleccionado el ListView de nuestra página principal, esto se hace en el evento LoadState de nuestra EditarRegistro.xaml.cs:

```
//Se asigna el id a la variable idRegistro
idRegistro = (Int32)navigationParameter;

//Se carga la ruta de la BD y se crea el objeto contexto
var rutaBD =
Path.Combine(Windows.Storage.ApplicationData.Current.LocalFolder.Path,
"ControlPeso.sqlite");
Contexto oContext = new Contexto(rutaBD);

//Se cargan los valores a los controles si contexto no es nulo
if (oContext != null)
{
//creamos una instanci de registro con los valores cargados de contexto
RegistroPeso oRegistro = (RegistroPeso)oContext.GetRegistro(idRegistro);
```

```
this.txtPeso.Text = oRegistro.Peso.ToString();
this.txtFecha.Text = oRegistro.Fecha.ToString();
this.txtDetalles.Text = oRegistro.Detalles;
}
```

Para finalizar agregamos el evento clic del botón editar:

```
private void btnEditar_Click(object sender, RoutedEventArgs e)
        {
            //Se crea el objeto registro
            RegistroPeso oRegistro = new RegistroPeso();

            //Se asignan los valores a la instancia creada
            oRegistro.Id = idRegistro;
            oRegistro.Peso = Decimal.Parse(this.txtPeso.Text);
            oRegistro.Fecha = DateTime.Parse(this.txtFecha.Text);
            oRegistro.Detalles = this.txtDetalles.Text;

            //Se carga la ruta de la base de datos y se crea el objeto
contexto
            var rutaBD =
Path.Combine(Windows.Storage.ApplicationData.Current.LocalFolder.Path,
"ControlPeso.sqlite");
            Contexto oContext = new Contexto(rutaBD);

            //Ejecutamos el método de edición y recuperamos el número de
registros afectados
            Int32 registro = oContext.EditRegistro(oRegistro);

            //Mensaje si hubo modificación o no
            if (registro > 0)
            {
                txtRespuesta.Text = "Registro Actualizado";
            }
            else
            {
                txtRespuesta.Text = "El registro no pudo actualizarse, por
favor vuelva a intentarlo";
            }
}
```

Personalización

Los siguientes son los valores correspondientes a los estilos y a los datatemplates que se requieren para personalizar un poco más nuestra aplicación y que deben incluirse dentro de la App.xaml:

```
<SolidColorBrush x:Key="BlockBackgroundBrush" Color="#FFFFFFFF"/>

            <Style x:Key="ThisTextStyle" TargetType="TextBlock" BasedOn="{StaticResource
BodyTextStyle}">
                <Setter Property="FontSize" Value="45"/>
                <Setter Property="FontWeight"  Value="Bold"/>
                <Setter Property="Foreground" Value="#FFF76E01"/>
            </Style>

            <!-- Forms Text Style -->
            <Style x:Key="FormsTextStyle" TargetType="TextBlock" BasedOn="{StaticResource
BasicTextStyle}">
                <Setter Property="FontSize" Value="16"/>
                <Setter Property="Foreground" Value="#FFF76E01"/>
            </Style>

            <!-- Forms TextBox Style -->
            <Style x:Key="FormsTextBoxStyle" TargetType="TextBox" >
                <Setter Property="FontSize" Value="16"/>
                <!--<Setter Property="Foreground" Value="#FFF76E01"/>-->
                <Setter Property="Background" Value="#FFFEBB4B"/>
                <Setter Property="BorderThickness"  Value="1"/>
                <Setter Property="BorderBrush"  Value="#FFA33106"/>
            </Style>

            <!-- Forms Button Style -->
            <Style x:Key="FormsButtonStyle" TargetType="Button" >
                <Setter Property="FontSize" Value="16"/>
                <!--<Setter Property="Foreground" Value="#FFF76E01"/>-->
                <Setter Property="Background" Value="#FF7F2706"/>
                <Setter Property="BorderThickness"  Value="1"/>
                <Setter Property="BorderBrush"  Value="#FFFA9902"/>
            </Style>

            <!-- Botón agregar parametros -->
            <Style x:Key="WebViewAppBarParametroStyle" TargetType="Button"
BasedOn="{StaticResource AppBarButtonStyle}">
                <Setter Property="AutomationProperties.AutomationId"
Value="WebViewAppBarButton"/>
                <Setter Property="AutomationProperties.Name" Value="Ingresar Datos"/>
                <Setter Property="Content" Value="&#xE125;"/>
            </Style>

            <!-- Botón agregar registro -->
            <Style x:Key="WebViewAppBarAddStyle" TargetType="Button"
BasedOn="{StaticResource AppBarButtonStyle}">
                <Setter Property="AutomationProperties.AutomationId"
Value="WebViewAppBarButton"/>
                <Setter Property="AutomationProperties.Name" Value="Agregar Registro"/>
                <Setter Property="Content" Value="&#xE109;"/>
            </Style>
```

```
            <!-- Botón Borrar registro -->
            <Style x:Key="WebViewAppBarDeleteStyle" TargetType="Button"
BasedOn="{StaticResource AppBarButtonStyle}">
                <Setter Property="AutomationProperties.AutomationId"
Value="WebViewAppBarButton"/>
                <Setter Property="AutomationProperties.Name" Value="Eliminar Registro"/>
                <Setter Property="Content" Value="&#xE10A;"/>
            </Style>

            <!-- Botón Editar registro -->
            <Style x:Key="WebViewAppBarModifyStyle" TargetType="Button"
BasedOn="{StaticResource AppBarButtonStyle}">
                <Setter Property="AutomationProperties.AutomationId"
Value="WebViewAppBarButton"/>
                <Setter Property="AutomationProperties.Name" Value="Actualizar
Registro"/>
                <Setter Property="Content" Value="&#xE10B;"/>
            </Style>

            <!-- LisItemView -->
            <DataTemplate x:Key="SplitListItemTemplate">
                <Grid Width="450" >
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition Width="1*"/>
                    </Grid.ColumnDefinitions>
                    <Grid.Background>
                        <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                            <GradientStop Color="#FFA83609"/>
                            <GradientStop Color="#FF7D2505" Offset="1"/>
                        </LinearGradientBrush>
                    </Grid.Background>
                    <StackPanel Grid.Column="1" HorizontalAlignment="Left"
Margin="12,0,0,0">
                        <TextBlock Text="{Binding Fecha}" MaxHeight="56"
TextWrapping="Wrap" FontSize="20"/>
                    </StackPanel>
                </Grid>
            </DataTemplate>

            <!-- Used in Snapped Split view -->
            <DataTemplate x:Key="NarrowListSplitTemplate">
                <Grid Height="80" Width="320">
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition Width="1*"/>
                    </Grid.ColumnDefinitions>
                    <Grid.Background>
                        <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                            <GradientStop Color="#FFA83609"/>
                            <GradientStop Color="#FF7D2505" Offset="1"/>
                        </LinearGradientBrush>
                    </Grid.Background>
                    <StackPanel Grid.Column="1" HorizontalAlignment="Left"
Margin="12,8,0,0">
                        <TextBlock Text="{Binding Fecha}" MaxHeight="56"
TextWrapping="Wrap"/>
```

```
                    </StackPanel>
                    <StackPanel Grid.Column="1" HorizontalAlignment="Left"
Margin="12,45,0,0" Orientation="Horizontal">
                        <TextBlock Text="{Binding Peso}" FontSize="16"
TextWrapping="Wrap"/>
                    </StackPanel>
                </Grid>
            </DataTemplate>
```

Sugerimos modificar los valores de diseño asociados al código XAML anterior para personalizar la aplicación que van a desarrollar.

Settings

Por último, es necesario agregar al menos el setting de privacidad de nuestra aplicación, para esto pueden remitirse directamente al código de la aplicación, crear tres carpetas por organización: Settings, UIHelpers y UIView

En la carpeta Setting agregaremos dos clases, la primera la llamaremos USetings.cs que deberá que heredará de la clase BlindableBase.

```
public class USettings : BindableBase
    {
        string _valorTexto;
        int _valorNumero;

        public string ValorTexto
        {
            get
            {
                return _valorTexto;
            }
            set
            {
                SetProperty(ref _valorTexto, value);
            }
        }

        public int ValorNumero
        {
            get
            {
                return _valorNumero;
            }
            set
            {
                SetProperty(ref _valorNumero, value);
            }
```

```
        }

    }
```

La segunda la llamaremos SetingsManager.cs y que nos permitirá manejar las páginas tipo Settings asociadas a nuestra aplicación, el código será el siguiente:

```
public class SettingsManager
 {
     public Settings LoadFromLocalSettings()
     {
         Settings settings = new Settings();

         var rootContainer = ApplicationData.Current.LocalSettings;
         settings.ValorTexto = GetValue<string>("ValorTexto");
         settings.ValorNumero = GetValue<int>("ValorNumero");

         return settings;
     }

     public void SaveToLocalSettings(Settings settings)
     {
         var rootContainer = ApplicationData.Current.LocalSettings;
         rootContainer.Values["ValorTexto"] = settings.ValorTexto;
         rootContainer.Values["ValorNumero"] = settings.ValorNumero;
     }

     private T GetValue<T>(string valuename)
     {
         T value;
         var rootContainer = ApplicationData.Current.LocalSettings;

         try
         { value = (T)rootContainer.Values[valuename]; }
         catch { value = default(T); }

         return value;
     }
 }
```

Dentro de la carpeta UIHelpers agregamos la clase SettingsFlyouts.cs que tiene el siguiente código:

```
class SettingsFlyout
{
    private const int _width = 346;
    private Popup _popup;

    public void ShowFlyout(UserControl control)
    {
```

```
            _popup = new Popup();
            _popup.Closed += OnPopupClosed;
            Window.Current.Activated += OnWindowActivated;
            _popup.IsLightDismissEnabled = true;
            _popup.Width = _width;
            _popup.Height = Window.Current.Bounds.Height;

            control.Width = _width;
            control.Height = Window.Current.Bounds.Height;

            _popup.Child = control;
            _popup.SetValue(Canvas.LeftProperty, Window.Current.Bounds.Width
- _width);
            _popup.SetValue(Canvas.TopProperty, 0);
            _popup.IsOpen = true;
        }

        private void OnWindowActivated(object sender,
Windows.UI.Core.WindowActivatedEventArgs e)
        {
            if (e.WindowActivationState ==
Windows.UI.Core.CoreWindowActivationState.Deactivated)
            {
                _popup.IsOpen = false;
            }
        }

        void OnPopupClosed(object sender, object e)
        {
            Window.Current.Activated -= OnWindowActivated;
        }
    }
```

Esta clase nos permite manejar los valores de las ventanas que se desplegaran con los datos e información de los settings asociados a la aplicación.

Ahora crearemos un control de usuario (UserControl) (Project – Add New Item) al que llamaremos PrivacySetting.xaml en el que quitaremos el Grid existente y agregaremos el siguiente código:

```
<Grid Background="#FF595D81">
        <StackPanel>
            <Button Style="{StaticResource BackButtonStyle}"
Click="OnBackButtonClick" />
            <TextBlock TextWrapping="Wrap" Text="Esta aplicación no comparte
información del usuario con otras aplicaciones ni con terceros.
" Foreground="#FFFFFFFF" FontSize="18" Margin="20,0,10,0" />
        </StackPanel>
</Grid>
```

En la parte correspondiente al código (PrivacySetting.xaml.cs) reemplazamos por el siguiente código en donde implementamos la funcionalidad completa para está área:

```
public sealed partial class PrivacySetting : UserControl
    {
        USettings Settings { get; set; }

        public PrivacySetting(USettings settings)
        {
            Settings = settings;

            this.InitializeComponent();
        }

        private void OnBackButtonClick(object sender,
Windows.UI.Xaml.RoutedEventArgs e)
        {

            if (this.Parent.GetType() == typeof(Popup))
            {
                ((Popup)this.Parent).IsOpen = false;
            }
            SettingsPane.Show();
        }

        private void UserControl_Loaded(object sender,
Windows.UI.Xaml.RoutedEventArgs e)
        {
            this.DataContext = Settings;
        }
    }
```

Ahora en el código de nuestra primera página (SplitPage.xaml.cs), creamos una instancia de la clase USettings al inicio:

```
public sealed partial class SplitPage : GuiaLabControlPeso.Common.LayoutAwarePage
{
   USettings _settings = new USettings();
```

Al final agregamos el evento CommandsRequested :

```
private void MainPage_CommandsRequested(SettingsPane sender,
SettingsPaneCommandsRequestedEventArgs args)
{
args.Request.ApplicationCommands.Clear();
// Add a Preferences command
var preferences = new SettingsCommand("preferences", "Privacidad", (handler)
=>
{
 var settings = new SettingsFlyout();
 settings.ShowFlyout(new PrivacySetting(_settings));
```

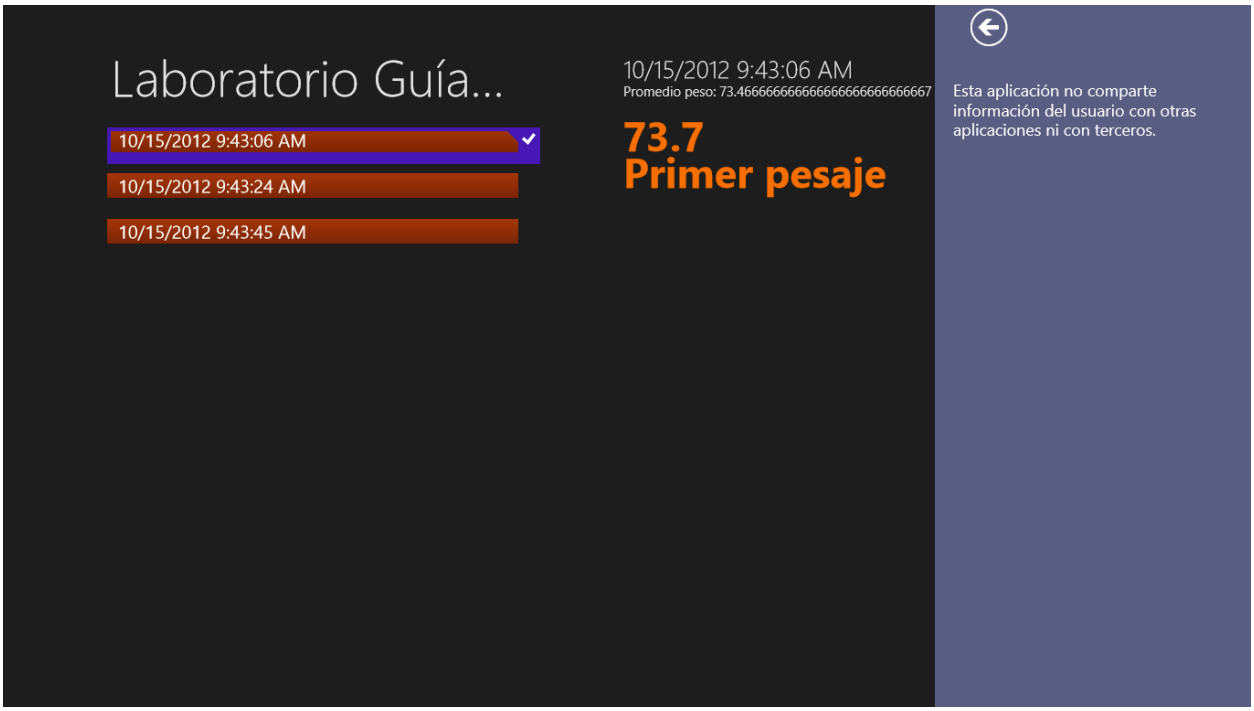
```
});

args.Request.ApplicationCommands.Add(preferences);
}
```

Para finalizar en el LoadState agregamos la siguiente línea de código en la parte inicial de este evento:

```
SettingsPane.GetForCurrentView().CommandsRequested +=
MainPage_CommandsRequested;
```

Para ver el resultado abrimos los settings de nuestra aplicación y seleccionamos la opción privacidad:

Esta abrirá el control creado con la información correspondiente a la privacidad que hemos agregado:



La presente guía es una base para la creación de aplicaciones al estilo WinRT Windows Store, pueden construirse a partir de ésta varias aplicaciones con diferentes grados de complejidad, para mayor información les invitamos a inscribirse en el curso correspondiente en http://www.microsoftvirtualacademy.com y revisar el sitio http://msdn.microsoft.com/es-ES/windows/apps