Desarrolla tus primeras apps móviles con Xamarin.Forms

ISC Luis Beltrán

Microsoft Student Partner
Xamarin Student Partner

@darkicebeam

http://icebeamwp.blogspot.com

beltran_prieto@fai.utb.cz

https://github.com/icebeam7/

Celaya Mobile .NET Developers Group

https://www.meetup.com/CelayaMobileDevelopers/

29 Julio 2016

Descarga los slides: http://bit.ly/XamarinCelaya

Agenda

1. Estructura de un proyecto Xamarin.Forms
2. XAML
3. C#
4. Primer aplicación: Lista de Contactos
5. Segunda aplicación: El juego del gato

Tipos de Proyecto en Xamarin

- Xamarin.Android (aplicación nativa)
- Xamarin.iOS (aplicación nativa)

- Xamarin.Forms Shared Project (aplicación multiplataforma)
- Xamarin.Forms PCL (aplicación multiplataforma)

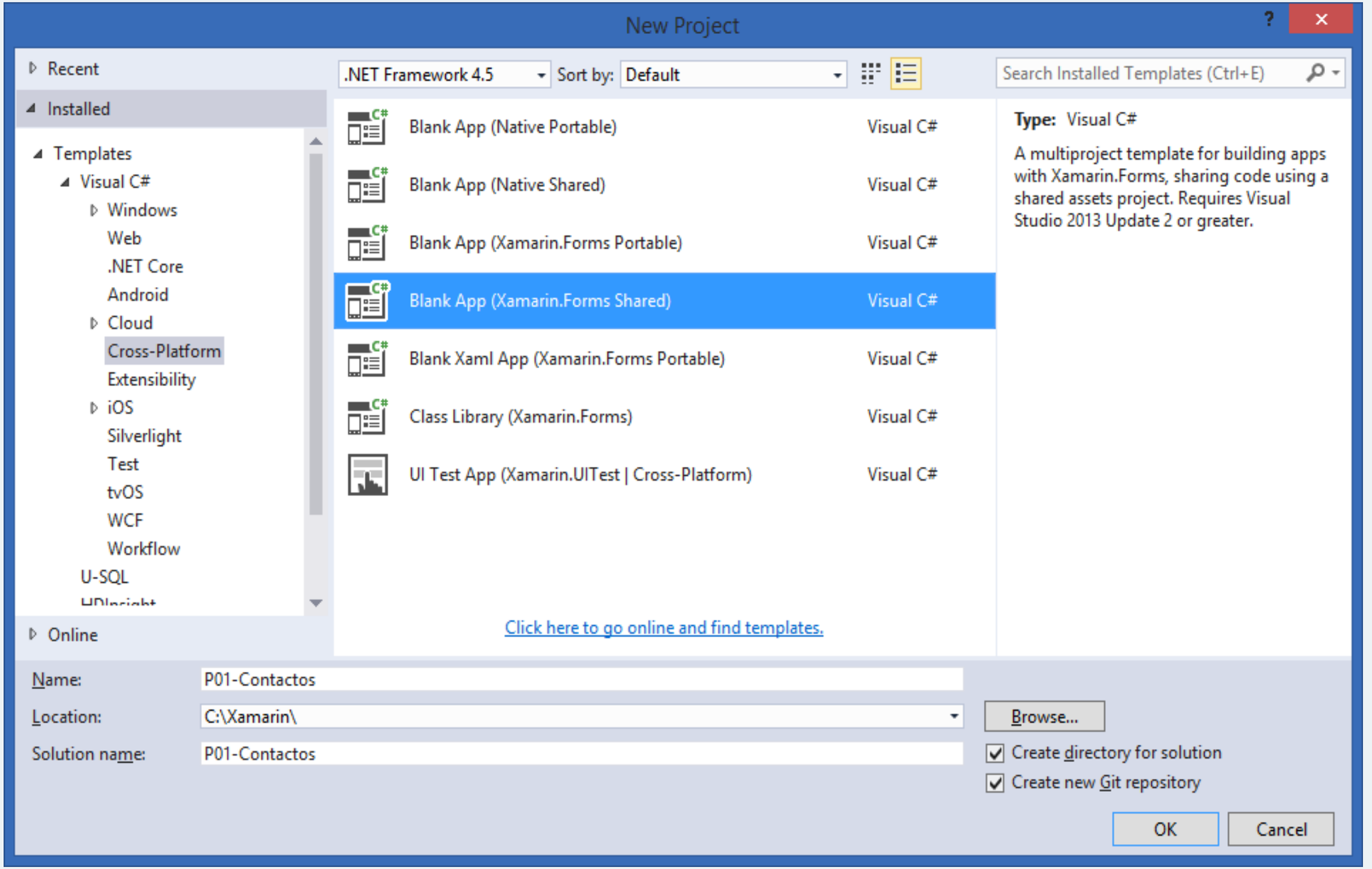
Links de interés:

https://xamarinhelp.com/portable-class-library-pcl-vs-shared-projects/

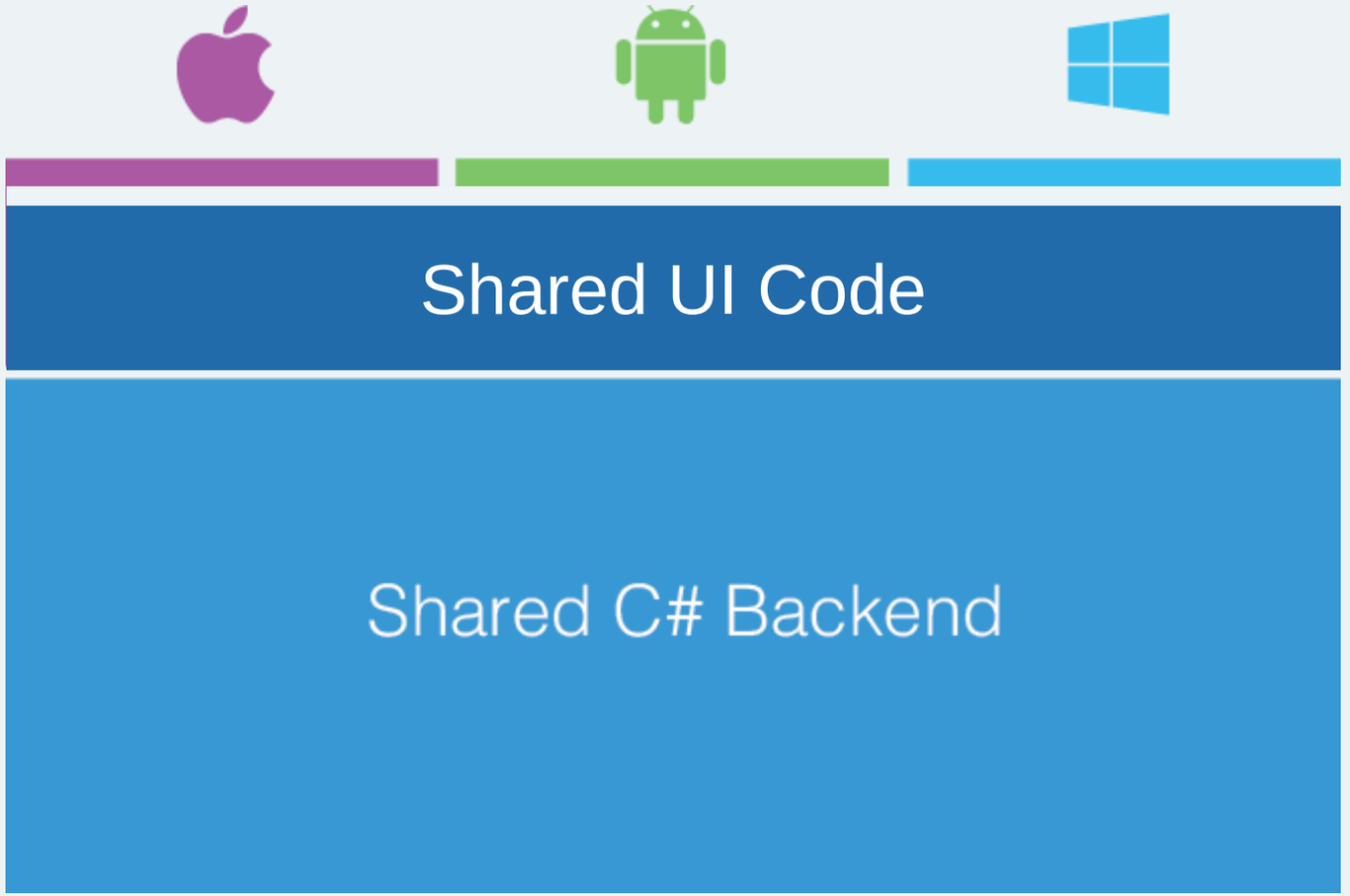
http://forums.xamarin.com/discussion/59381/why-shared-projects

http://tirania.org/blog/archive/2016/Jan-22.html

http://xfcomplete.net/general/2016/01/19/pcl-or-shared-project/


New Project
Recent
Installed
Templates
Visual C#
Windows
Web
.NET Core
Android
Cloud
Cross-Platform
Extensibility
iOS
Silverlight
Test
tvOS
WCF
Workflow
U-SQL
Online
.NET Framework 4.5
Sort by:
Default
Search Installed Templates (Ctrl+E)
Blank App (Native Portable) Visual C#
Blank App (Native Shared) Visual C#
Blank App (Xamarin.Forms Portable) Visual C#
Blank App (Xamarin.Forms Shared) Visual C#
Blank Xaml App (Xamarin.Forms Portable) Visual C#
Class Library (Xamarin.Forms) Visual C#
UI Test App (Xamarin.UITest | Cross-Platform) Visual C#
Click here to go online and find templates.
Type: Visual C#
A multiproject template for building apps with Xamarin.Forms, sharing code using a shared assets project. Requires Visual Studio 2013 Update 2 or greater.
Name: P01-Contactos
Location: C:\Xamarin\
Solution name: P01-Contactos
Browse...
Create directory for solution
Create new Git repository
OK
Cancel

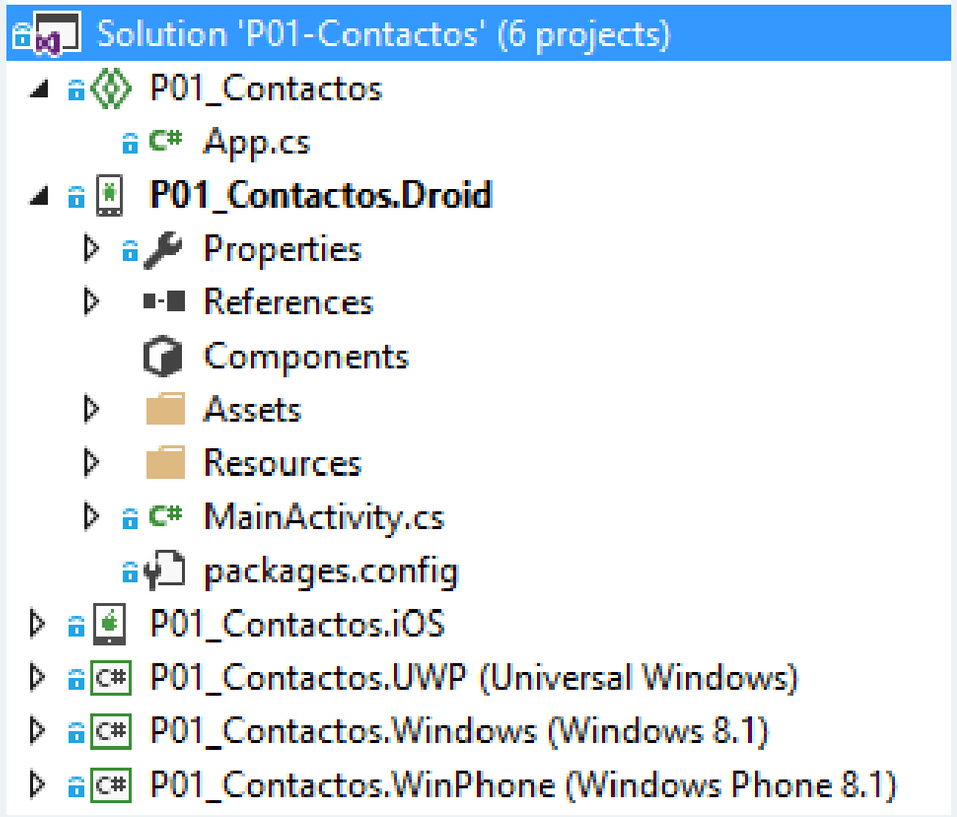
1. Estructura de un proyecto Xamarin.Forms



Estructura de un Proyecto Xamarin.Forms

Una solución que incluya Xamarin.Forms consta de 2 tipos de elemento:

- El **proyecto compartido** (Shared o PCL)
- Los **proyectos de plataforma específica (*):**
  - Android
  - iOS
  - UWP
  - Windows
  - Windows Phone



* Depende de los SDK instalados

Estructura de un Proyecto Xamarin.Forms

Consideraciones:

- Idealmente, un 80-90% de tu código lo implementarás en el Shared Project.
- El resto será código específico de plataforma debido a las diferencias entre plataformas, por ejemplo:
  - Imágenes
  - Manejo de tamaños de pantalla
  - Navegación (Hardware back button)
  - Toques y gestos
  - Notificaciones Push
  - Cámara
  - Mapas y servicios de geolocalización
  - Sensores: acelerómetro, giroscopio, compás
  - Twitter y Facebook
  - NFC

Estructura de un Proyecto Xamarin.Forms

- La clase App es el punto de entrada de toda aplicación en Xamarin.Forms.

- Esta clase es llamada en los diversos puntos de entrada de cada plataforma:
  - MainActivity (Android)
  - Main (iOS)
  - App (UWP, Windows, Windows Phone)

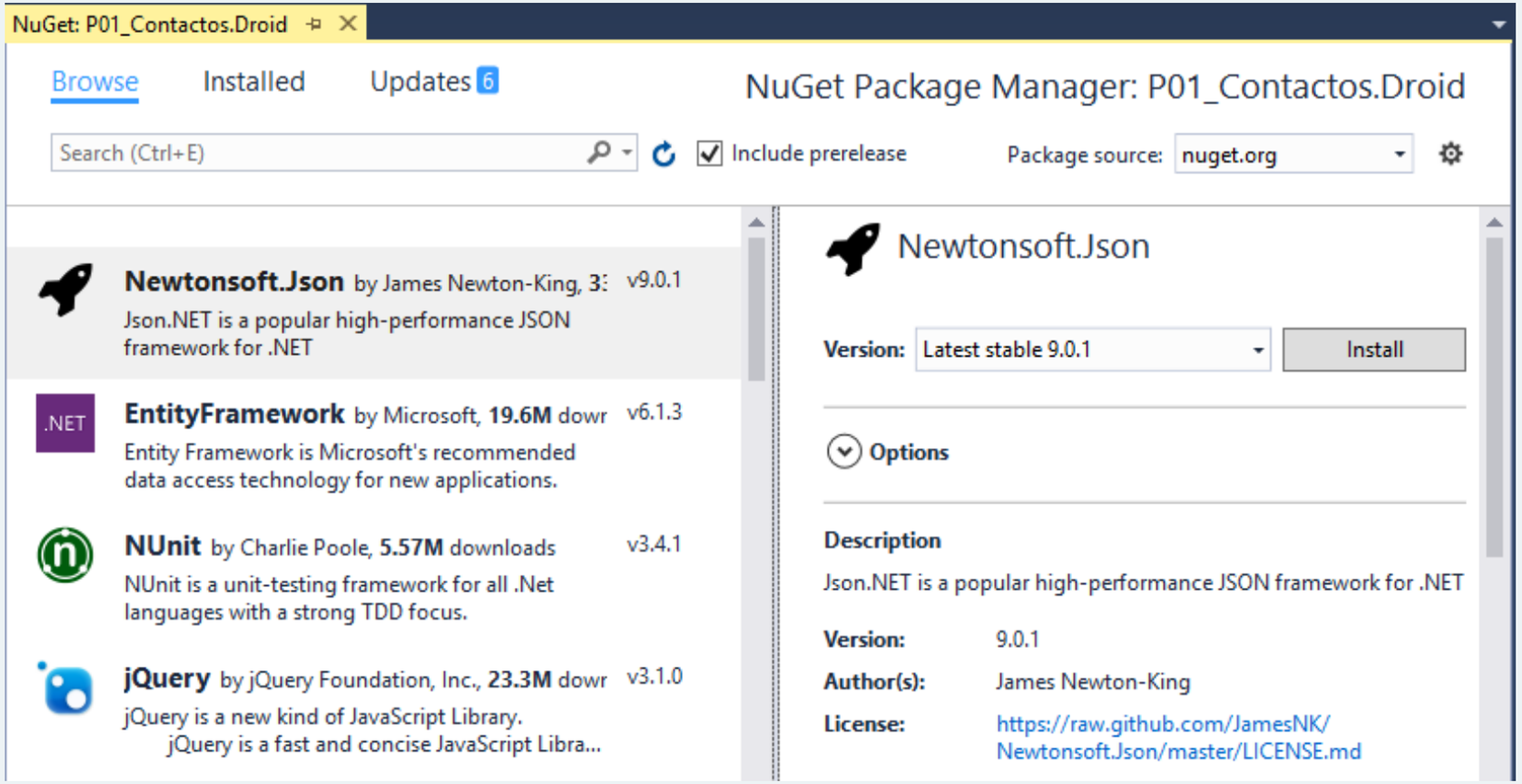
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Xamarin.Forms;

namespace P01_Contactos
{
    public class App : Application
    {
        public App ()
        {
            // The root page of your application
            MainPage = new ContentPage {
                Content = new StackLayout {
                    VerticalOptions = LayoutOptions.Center,
                    Children = {
                        new Label {
                            HorizontalTextAlignment = TextAlignment.Center,
                            Text = "Welcome to Xamarin Forms!"
                        }
                    }
                }
            };
        }
```

Estructura de un Proyecto Xamarin.Forms

- Mediante los NugetPackages es posible agregar referencias a librerías externas para extender la funcionalidad de nuestras aplicaciones.
- Se instalan por plataforma específica.



2. XAML

XAML

- eXtensible Application Markup Language basado en XML.
- Creado por Microsoft como una alternativa de programación para instanciar e inicializar objetos, organizándolos en jerarquías padre-hijo.
- Permite a los desarrolladores definir la IU utilizando un lenguaje de marcado (etiquetas) en lugar de código.
- No es obligatorio definir la UI en XAML, pero el lenguaje es más coherente visualmente hablando.

```
<Grid x:Name="LayoutRoot">
    <Button>
        <TextBox
            Text="Hi World"
            Margin="35"
            FontSize="25"
        />
    </Button>
</Grid>
```

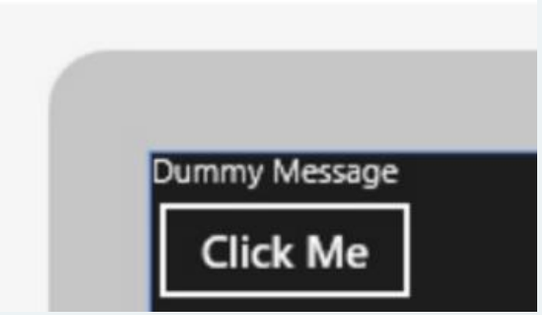
XAML

- Adoptado por varias tecnologías dentro del .NET Framework, encontró su más grande utilidad definiendo el layout de las interfaces de usuario en proyectos tales como Silverlight, WPF, Windows Phone, Windows 8 y Windows RT.
- Se ajusta perfectamente a la arquitectura de aplicaciones MVVM (Model-View-ViewModel).
- XAML define el View que se enlaza al código del ViewModel mediante un enlace de datos (data-binding) basado en XAML.

XAML

- Adoptado por varias tecnologías dentro del .NET Framework, encontró su mayor utilidad definiendo el layout de interfaces de usuario en proyectos Silverlight, WPF, Windows Phone, Windows 8 y Windows RT.
- Se ajusta perfectamente a la arquitectura de aplicaciones MVVM (Model-View-ViewModel).
- XAML define el View que se enlaza al código del ViewModel mediante un enlace de datos (data-binding) basado en XAML.

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel DataContext="{StaticResource Vm}">
        <TextBlock Text="{Binding Message}"/>
        <Button Content="{Binding ButtonText}"/>
    </StackPanel>
</Grid>
```



XAML

- En XAML, cada elemento tiene atributos (propiedades).
- A las propiedades se le asigna un valor.
- Se puede enlazar un manejador de evento en XAML a un método dentro del code-behind (C#).
- No se pueden utilizar ciclos. Pero elementos como ListView definen Templates por ejemplo para diseñar el aspecto de cada elemento contenido.

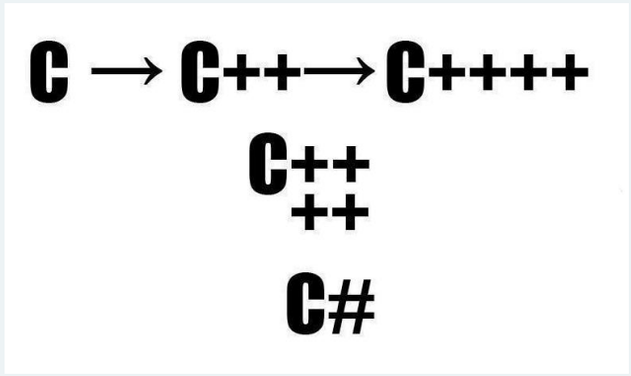
```
<DataTemplate x:Key="belowAverageTemplate">
    <TextBlock Text="{Binding FirstName}" Background="Azure" Foreground="Red"/>
</DataTemplate>

<DataTemplate x:Key="standardTemplate">
    <TextBlock Text="{Binding FirstName}" Background="YellowGreen" Foreground="Green"/>
</DataTemplate>
```

3. C#

El lenguaje C#

- Lenguaje de programación orientado a objetos.
- Permite crear aplicaciones que se ejecutan en .NET Framework.
- Lenguaje de programación adoptado por Microsoft como base de su plataforma .NET.
- Evolución de C++.
- Sencillo, moderno, incluye seguridad de tipos.
- Gracias al Proyecto Mono, se desarrolló un compilador para C# que es independiente de sistema operativo.



El lenguaje C#

Keyword	Class	Range
bool	System.Boolean	true and false
byte	System.Byte	0 to 255
sbyte	System.SByte	-128 to 127
short	System.Int16	-32768 to 32767
ushort	System.Uint16	0 to 65535
int	System.Int32	-2,147,483,648 to 2,147,483,647
uint	System.UInt32	0 to 4,294,967,295
long	System.Int64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	System.UInt64	0 to 18,446,744,073,709,551,615
decimal	System.Decimal	-79,228,162,514,264,337,593,543,950,335 to 79,228,162,514,264,337,593,543,950,335
double	System.Double	-1.79769313486232e308 to 1.79769313486232e308
float	System.Single	-3.402823e38 to 3.402823e38
char	System.Char	0 to 65535

El lenguaje C#

abstract	do	in	protected	true
as	double	int	public	try
base	else	interface	readonly	typeof
bool	enum	internal	ref	uint
break	event	is	return	ulong
byte	explicit	lock	sbyte	unchecked
case	extern	long	sealed	unsafe
catch	false	namespace	short	ushort
char	finally	new	sizeof	using
checked	fixed	null	stackalloc	virtual
class	float	object	static	void
const	for	operator	string	volatile
continue	foreach	out	struct	while
decimal	goto	override	switch	
default	if	params	this	
delegate	implicit	private	throw	

← Palabras reservadas

Palabras reservadas contextuales →

add	dynamic	in	partial	where
ascending	equals	into	remove	yield
async	from	join	select	
await	get	let	set	
by	global	on	value	
descending	group	orderby	var	

¿Una variable puede llamarse igual que una palabra reservada?

El lenguaje C#

Category (by precedence)	Operator(s)	Associativity
Primary	x.y f(x) a[x] x++ x-- new typeof default checked	left
Unary	+ - ! ~ ++x --x (T)x	right
Multiplicative	* / %	left
Additive	+ -	left
Shift	<< >>	left
Relational	< > <= >= is as	left
Equality	== !=	right
Logical AND	&	left
Logical XOR	^	left
Logical OR	|	left
Conditional AND	&&	left
Conditional OR	||	left
Null Coalescing	??	left
Ternary	?:	right
Assignment	= *= /= %= += -= <<= >>= &= ^= |= =>	right

C# 6.0

Initializers for auto-properties

You can now add an initializer to an auto-property, just as you can in a field:

```
public class Customer
{
    public string First { get; set; } = "Jane";
    public string Last { get; set; } = "Doe";
}
```

The initializer directly initializes the backing field; it doesn't work through the setter of the auto-property. The initializers are executed in order as written, just as – and along with – field initializers.

Just like field initializers, auto-property initializers cannot reference `this` – after all they are executed before the object is properly initialized.

Expression bodies on method-like members

Methods as well as user-defined operators and conversions can be given an expression body by use of the "lambda arrow":

```
public Point Move(int dx, int dy) => new Point(x + dx, y + dy);
public static Complex operator +(Complex a, Complex b) => a.Add(b);
public static implicit operator string(Person p) => p.First + " " + p.Last;
```

The effect is exactly the same as if the methods had had a block body with a single return statement.

For void-returning methods – and `Task`-returning async methods – the arrow syntax still applies, but the expression following the arrow must be a statement expression (just as is the rule for lambdas):

```
public void Print() => Console.WriteLine(First + " " + Last);
```

C# 6.0

Using static

The feature allows all the accessible static members of a type to be imported, making them available without qualification in subsequent code:

```
using static System.Console;
using static System.Math;
using static System.DayOfWeek;
class Program
{
    static void Main()
    {
        WriteLine(Sqrt(3*3 + 4*4));
        WriteLine(Friday - Monday);
    }
}
```

This is great for when you have a set of functions related to a certain domain that you use all the time. `System.Math` would be a common example of that. It also lets you directly specify the individual named values of an enum type, like the `System.DayOfWeek` members above.

Null-conditional operators

Sometimes code tends to drown a bit in null-checking. The null-conditional operator lets you access members and elements only when the receiver is not-null, providing a null result otherwise:

```
int? length = customers?.Length; // null if customers is null
Customer first = customers?[0];  // null if customers is null
```

The null-conditional operator is conveniently used together with the null coalescing operator `??`:

```
int length = customers?.Length ?? 0; // 0 if customers is null
```

The null-conditional operator exhibits short-circuiting behavior, where an immediately following chain of member accesses, element accesses and invocations will only be executed if the original receiver was not null:

```
int? first = customers?[0].Orders.Count();
```

This example is essentially equivalent to:

```
int? first = (customers != null) ? customers[0].Orders.Count() : null;
```

C# 6.0

String interpolation

`String.Format` and its cousins are very versatile and useful, but their use is a little clunky and error prone. Particularly unfortunate is the use of `{0}` etc. placeholders in the format string, which must line up with arguments supplied separately:

```
var s = String.Format("{0} is {1} year{{s}} old", p.Name, p.Age);
```

String interpolation lets you put the expressions right in their place, by having "holes" directly in the string literal:

```
var s = $"{p.Name} is {p.Age} year{{s}} old";
```

Just as with `String.Format`, optional alignment and format specifiers can be given:

```
var s = $"{p.Name,20} is {p.Age:D3} year{{s}} old";
```

The contents of the holes can be pretty much any expression, including even other strings:

```
var s = $"{p.Name} is {p.Age} year{(p.Age == 1 ? "" : "s")} old";
```

Notice that the conditional expression is parenthesized, so that the `: "s"` doesn't get confused with a format specifier.

Await in catch and finally blocks

In C# 5 we don't allow the `await` keyword in catch and finally blocks, because we'd somehow convinced ourselves that it wasn't possible to implement. Now we've figured it out, so apparently it wasn't impossible after all.

This has actually been a significant limitation, and people have had to employ unsightly workarounds to compensate. That is no longer necessary:

```
Resource res = null;
try
{
    res = await Resource.OpenAsync(…);       // You could do this.
    …
}
catch(ResourceException e)
{
    await Resource.LogAsync(res, e);         // Now you can do this …
}
finally
{
    if (res != null) await res.CloseAsync(); // … and this.
}
```

The implementation is quite complicated, but you don't have to worry about that. That's the whole point of having async in the language.

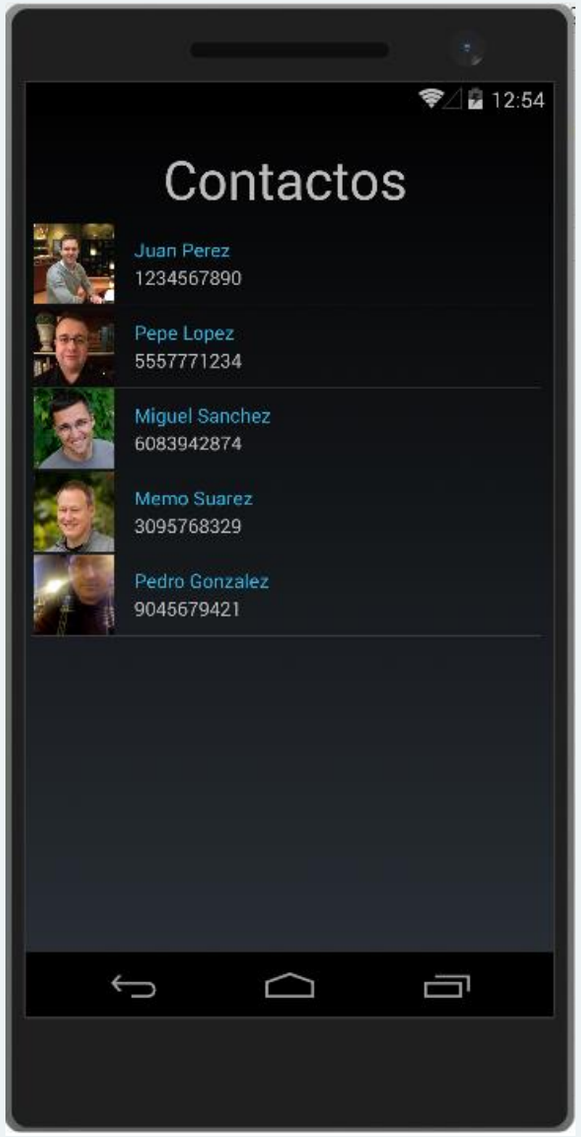
Aprender C#

- Curso de C# http://www.csharpya.com.ar/
- Tutorial Curso C# Codigo Facilito http://bit.ly/1JsnykC
- Fundamentos de C# para Principiantes Absolutos http://bit.ly/28LFTD1
- Jump Start de Programación en C# http://bit.ly/1qfWeQl
- ¿Qué hay de Nuevo en C# 6.0? http://bit.ly/1cgztJb
- 20 preguntas de C# contestadas http://bit.ly/1CLcEtw
- C# conciso http://bit.ly/1gtHEEj
- Edx https://www.edx.org/course/programming-c-microsoft-dev204x-0

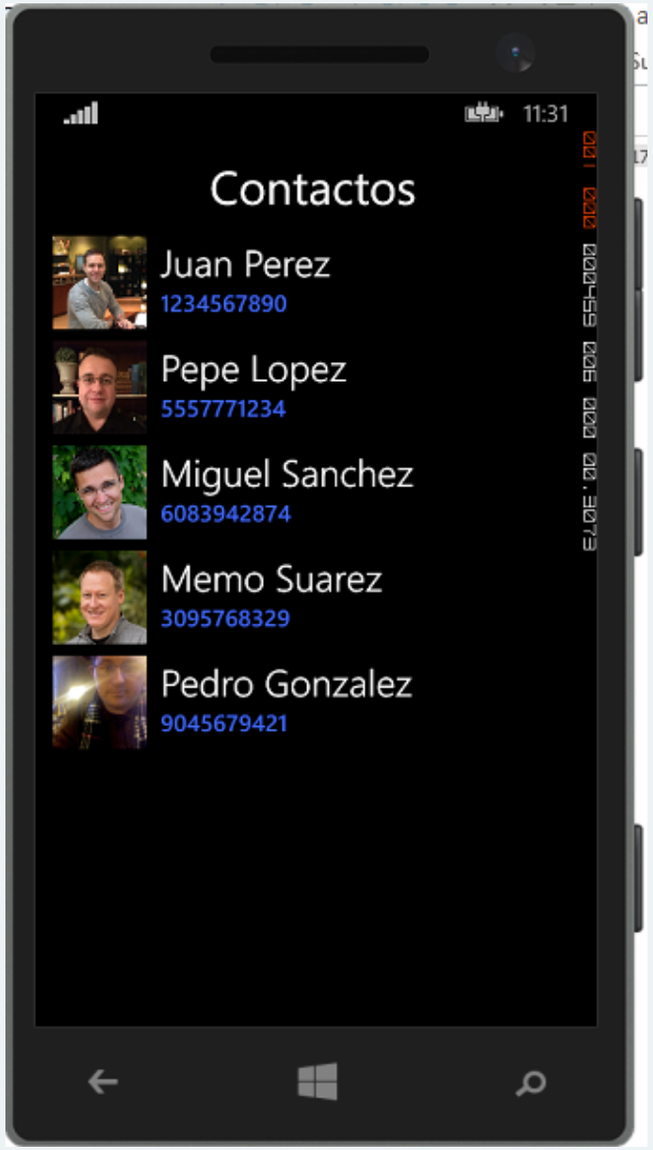
Receso (5 min)

http://bit.ly/xamarin-books

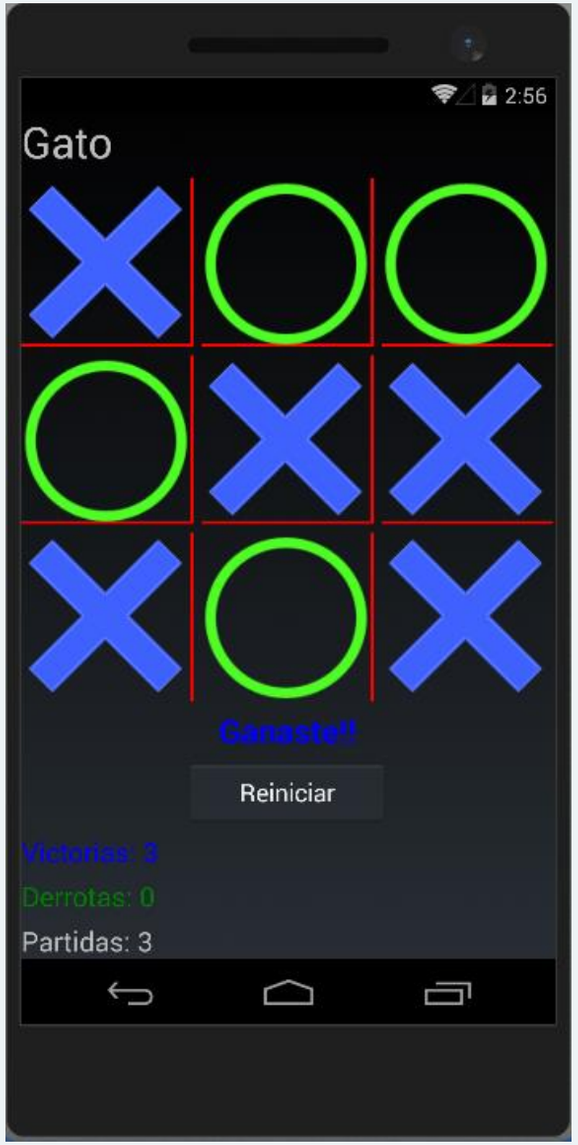
4. Primera aplicación: Lista de contactos



https://github.com/icebeam7/P01-Contactos



5. Segunda aplicación: El juego del gato



https://github.com/icebeam7/P02-JuegoGato

¿Preguntas?

Camino a SG Next: Bots, Servicios Cognitivos y App móviles

EVENT DETAILS

- **START:** 3 AGOSTO 2016 2:00 PM
- **CATEGORIES:** WEBINAR LUNCH & LEARN

Los servicios cognitivos, tales como Visión de Computadora, Emociones, Reconocedor de Voz y Lenguaje Natural, entre otros permiten añadir Inteligencia Artificial a tus aplicaciones.

Un bot es una aplicación que se comporta como un humano, es decir, es inteligente y puede interactur con otros humanos. Puede platicar contigo, tomar decisiones y darte información útil realizando cálculos a gran velocidad o analizando grandes cantidades de información en segundos.

En este webinar aprenderás lo fácil y sencillo que es integrar un bot y servicios cognitivos a una aplicación móvil de recomendaciones (películas, nutrición y/o otros servicios).

Tecnologías a utilizar:
* Microsoft Bot Framework
* Microsoft Cognitive Services
* Microsoft Azure
* Xamarin Platform

--

Acerca del conferencista
Luis Beltrán se desempeña como profesor en el Instituto Tecnológico de Celaya, y actualmente estudia el Doctorado en Ingeniería Informática en la Universidad Tomás Bata en Zlín, República Checa. Es Microsoft Student Partner y Xamarin Student Partner. Sus áreas de interés son el desarrollo de aplicaciones móviles, cloud computing con Azure, C# y Machine Learning.

¡Regístrate!

SG Campus

Registro para la charla

http://bit.ly/sgbots

¡Gracias!

ISC Luis Beltrán

Microsoft Student Partner
Xamarin Student Partner

@darkicebeam

http://icebeamwp.blogspot.com

beltran_prieto@fai.utb.cz

https://github.com/icebeam7/

Celaya Mobile .NET Developers Group

https://www.meetup.com/CelayaMobileDevelopers/

29 Julio 2016

Descarga los slides: http://bit.ly/XamarinCelaya