



Xamarin.Forms

Rodrigo Díaz Concha
@rdiazconcha

Rodrigo Díaz Concha

rodrigo@rdiazconcha.com

Microsoft Regional
Director para México



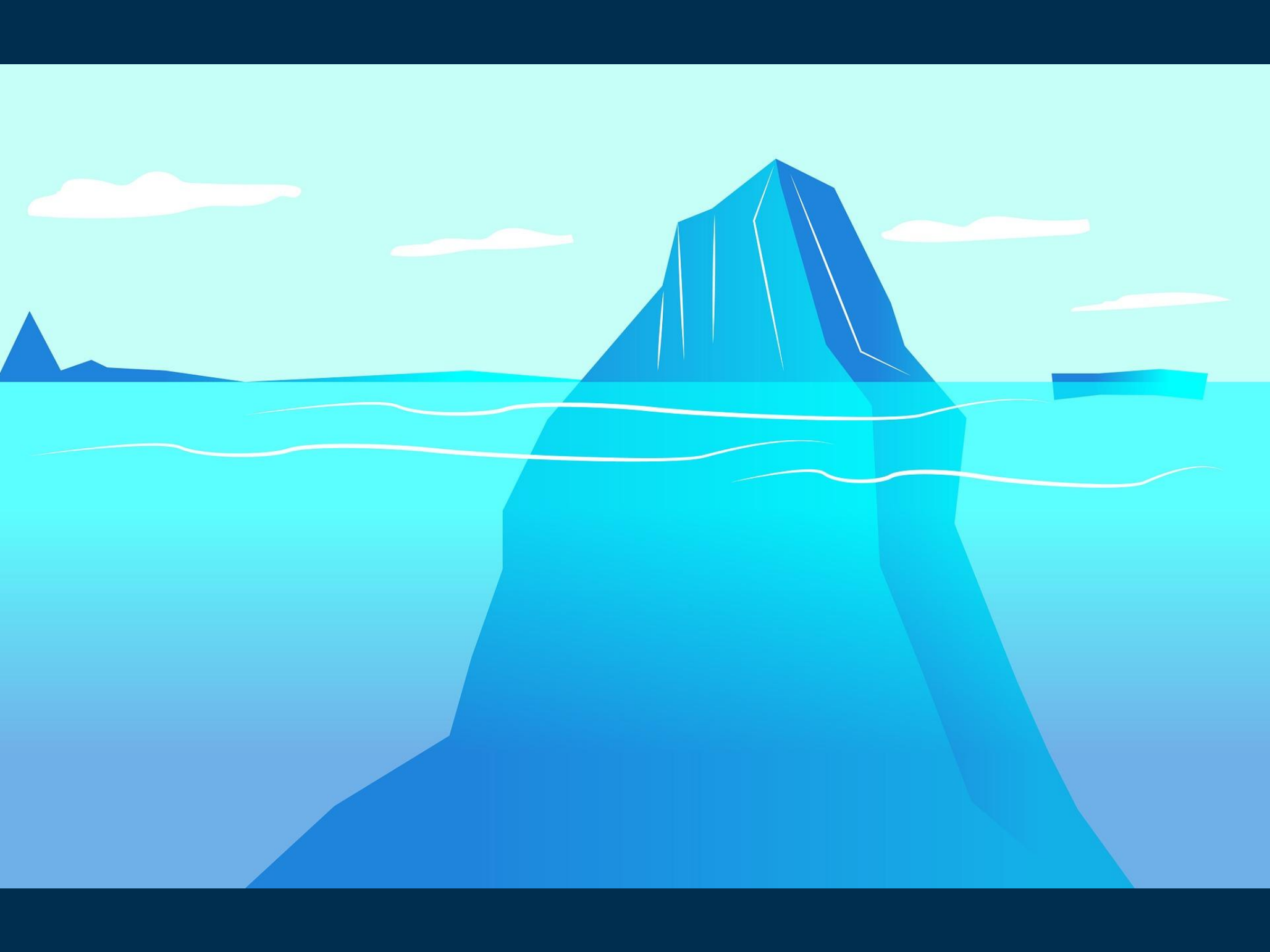
Carnegie Mellon SEI
Software Architecture
Professional

10 veces reconocido como
Microsoft Most Valuable
Professional (MVP)









Introducción a Xamarin

¿Qué es Xamarin?



¿Qué es Xamarin?

Xamarin permite a los desarrolladores de C# alcanzar todas las plataformas móviles más importantes

¿Qué es Xamarin?

Interfaz de Usuario Nativa

¿Qué es Xamarin?

Performance Nativo

¿Qué es Xamarin?

Código compartido entre
todas las plataformas

¿Qué es Xamarin?

C# y el .NET Framework

¿Qué es Xamarin?

Incluido en Visual Studio .NET
Community o superior

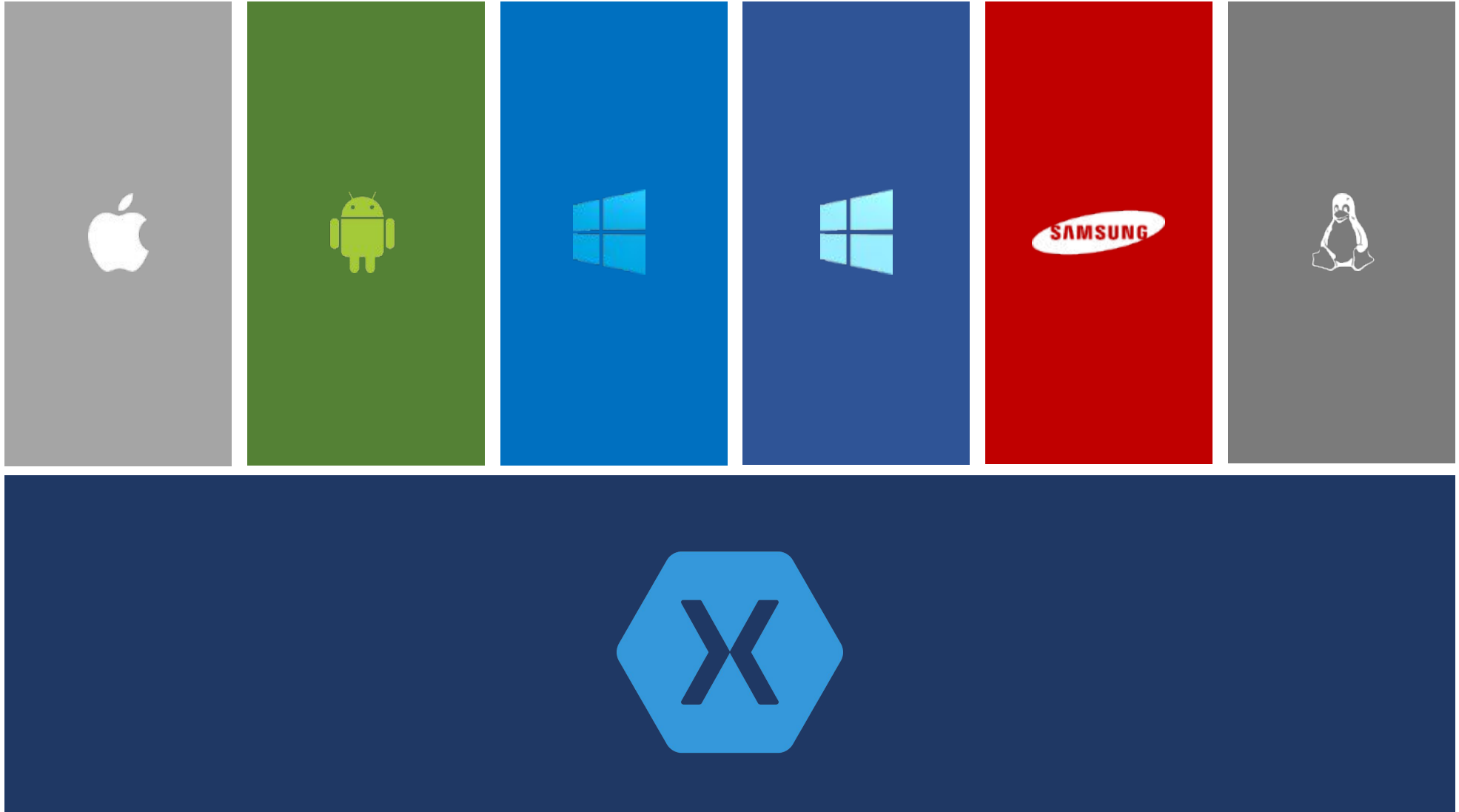
¿Qué es Xamarin?

Adquirido por Microsoft en
febrero 2016

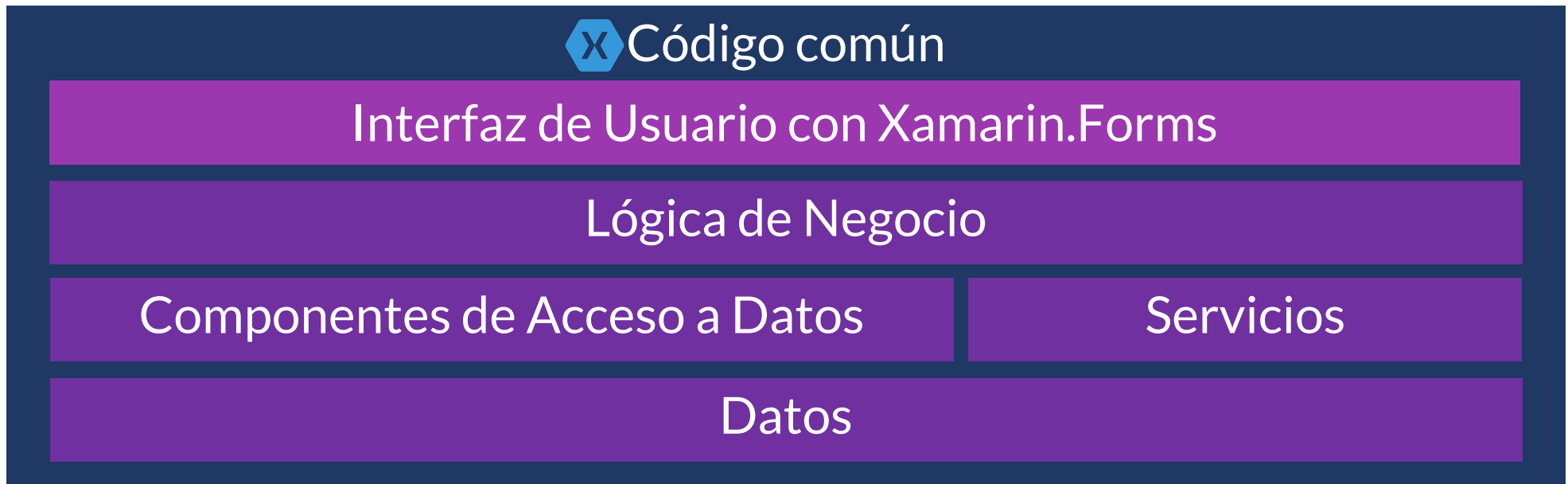
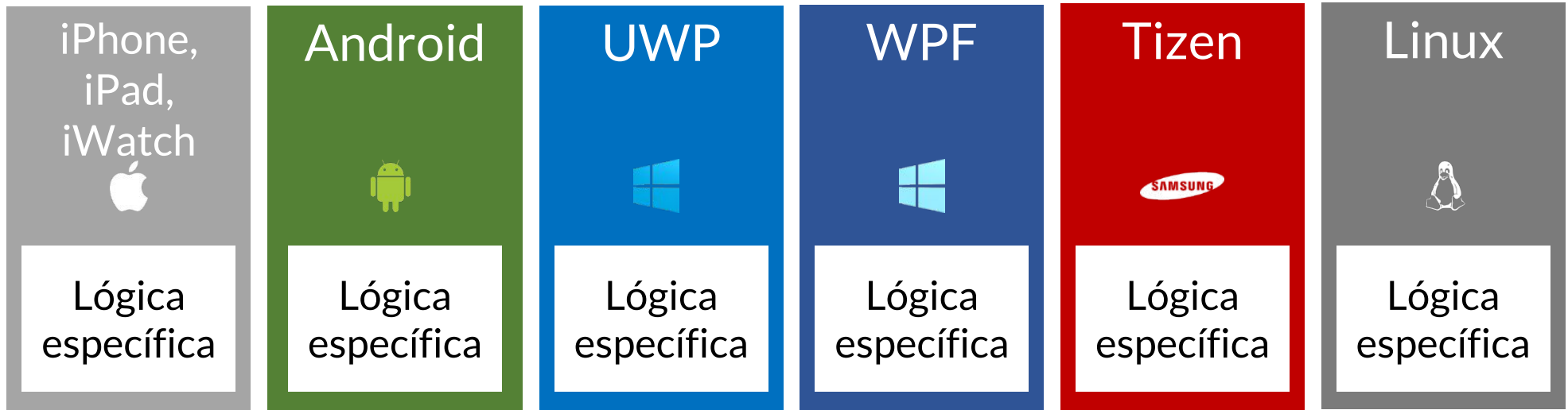
¿Qué es Xamarin?

Open Source y en constante
innovación

¿Qué es Xamarin?



¿Qué es Xamarin?



Mono

- Xamarin está basado en Mono
 - Implementación multiplataforma, libre y abierta del .NET Framework
 - [Compilador de C#](#)
 - [CLR \(Mono Runtime\)](#)
 - [Biblioteca de Clases Base](#)
 - [Biblioteca de Clases Mono](#)
 - Basado en estándares de ECMA
 - Open Source

Ecma-334

<http://www.ecma-international.org/publications/standards/Ecma-334.htm>

Ecma-335

<http://www.ecma-international.org/publications/standards/Ecma-335.htm>

mono-project.com

Código fuente

- Repositorio raíz de Xamarin

github.com/xamarin

- Repositorio de Xamarin.Forms

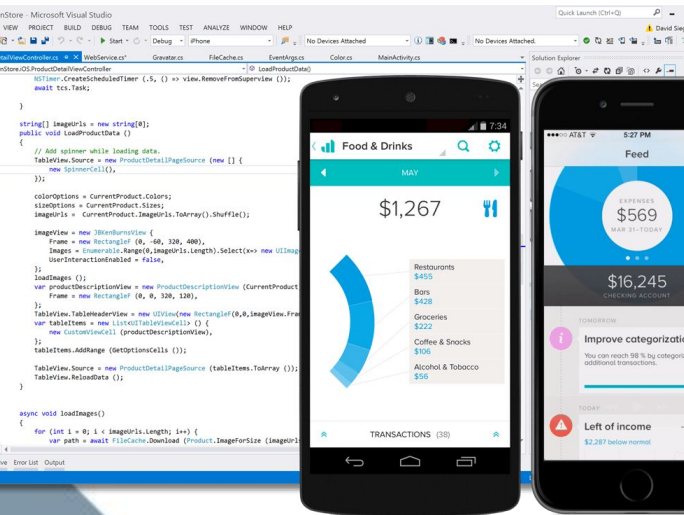
github.com/xamarin/Xamarin.Forms

Instalación y configuración

Ambientes de Desarrollo

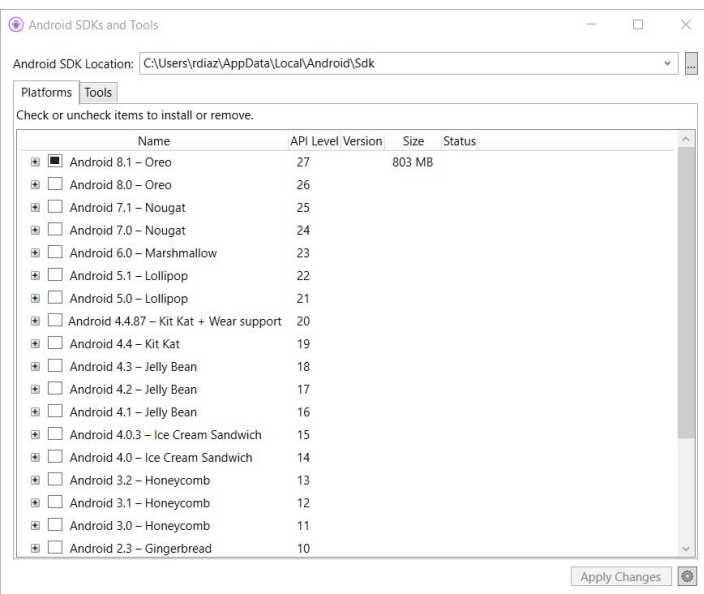
	macOS	Windows
	Visual Studio .NET	Visual Studio .NET
Xamarin.iOS	Sí	Sí (con una Mac)
Xamarin.Android	Sí	Sí
Xamarin.Forms	Solo iOS y Android	Android, UWP (con una Mac: iOS)

Herramientas



- Visual Studio .NET
 - Entorno de desarrollo para aplicaciones Android, iOS, UWP, etc.
 - Windows y macOS
 - Las apps para iOS requieren acceder a una Mac para compilar

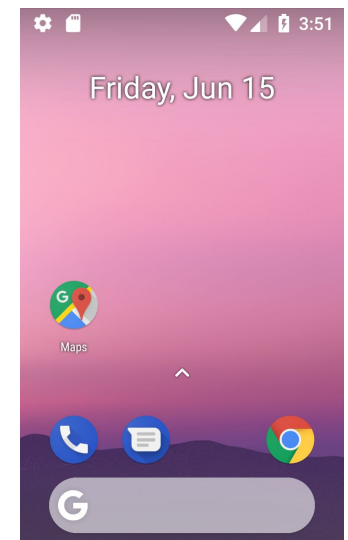
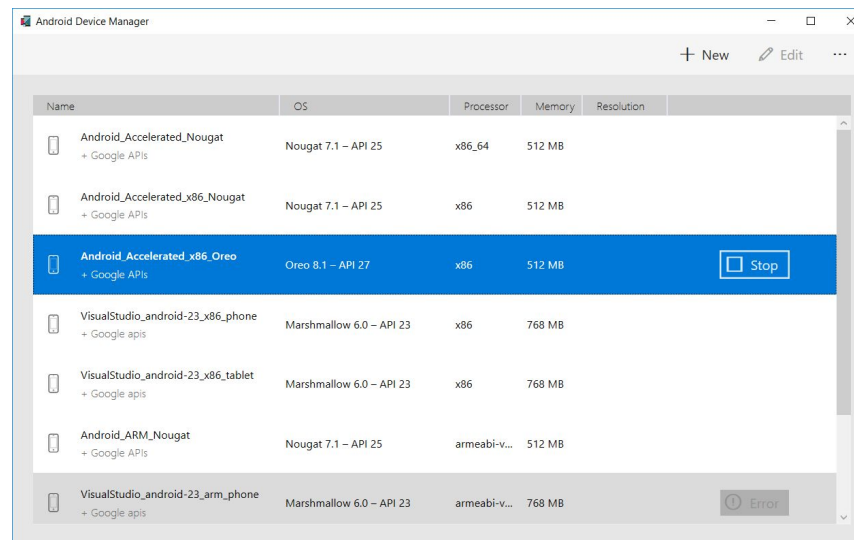
Herramientas



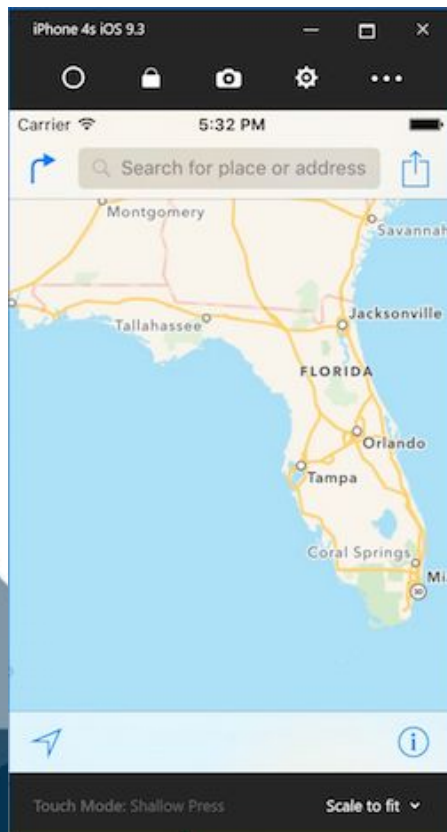
- Android SDK Manager
 - Nuevo administrador de SDKs y componentes relacionados
 - Resuelve las dependencias automáticamente

Herramientas

- Android Device Manager
 - Nueva herramienta para crear, editar y controlar dispositivos virtuales Android
 - Requiere Android SDK Tools 26 o superior

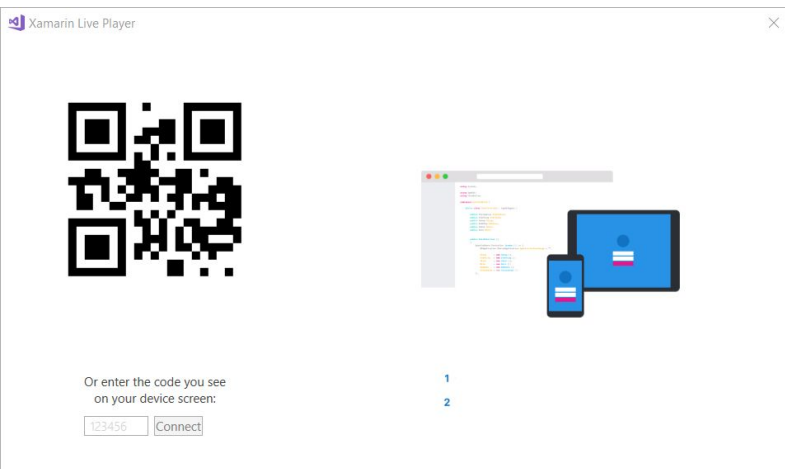


Herramientas



- iOS Simulator for Windows
 - Permite probar y depurar aplicaciones iOS completamente en Visual Studio .NET
 - Soporte a pantallas táctiles
 - Contar con una Mac sigue siendo requerido

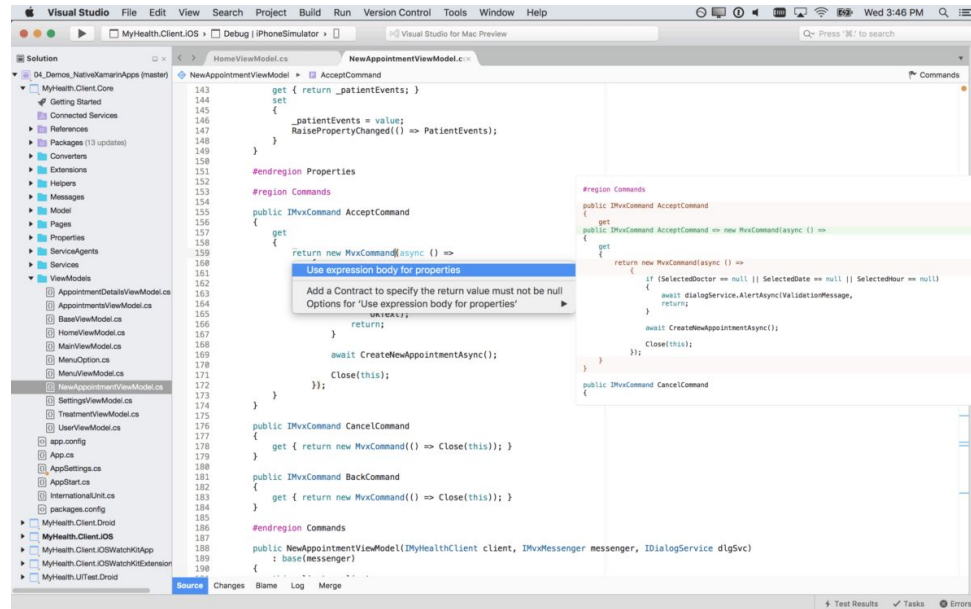
Herramientas



- **Xamarin Live Player**
 - Despliega y ejecuta aplicaciones de una manera rápida y sencilla
 - Requiere tener instalada la app Xamarin Live Player en tu dispositivo
 - Tiene algunas limitantes*

Herramientas

- Visual Studio for Mac
 - Nuevo entorno de desarrollo para aplicaciones Android, iOS, Mac, ASP.NET Core y juegos con Unity



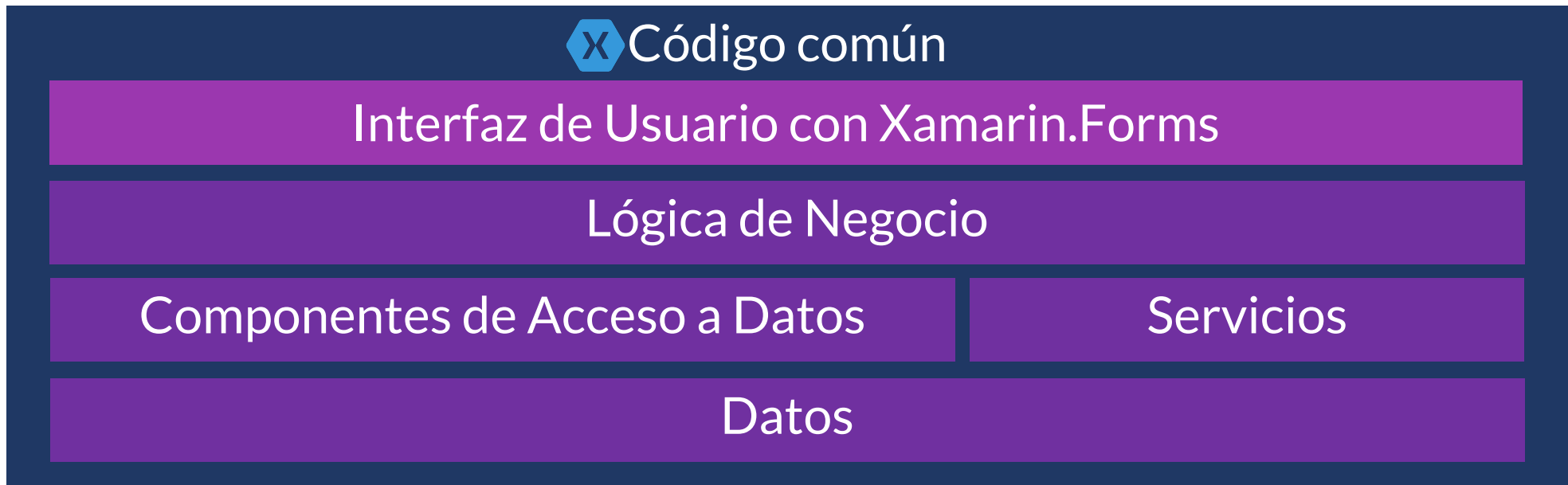
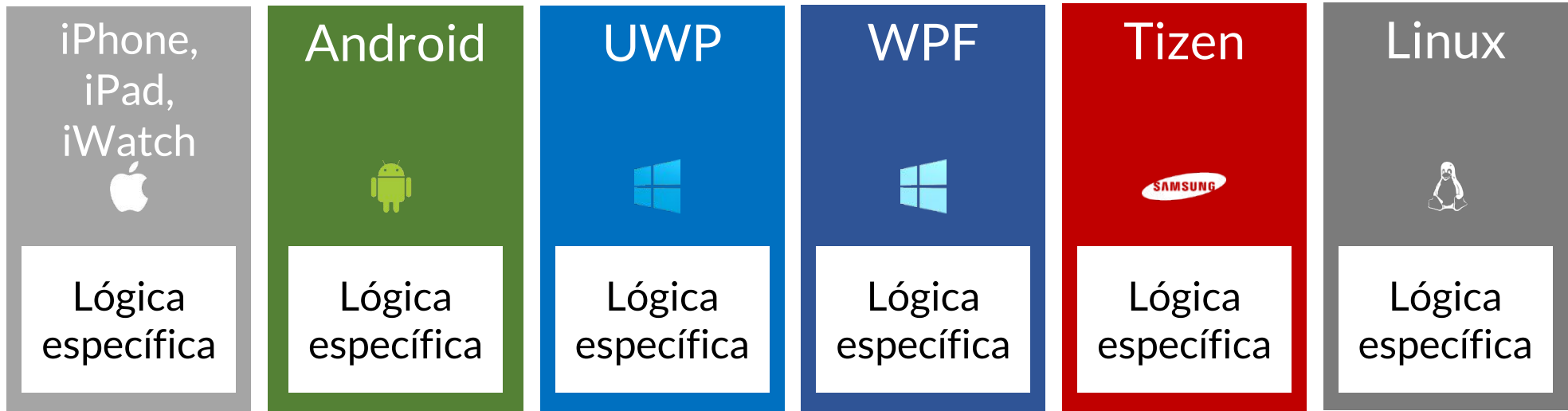
Configuración de Xamarin.iOS

Requiere comunicarse a la Mac
Recuerda configurar el permiso
de “Remote Login”

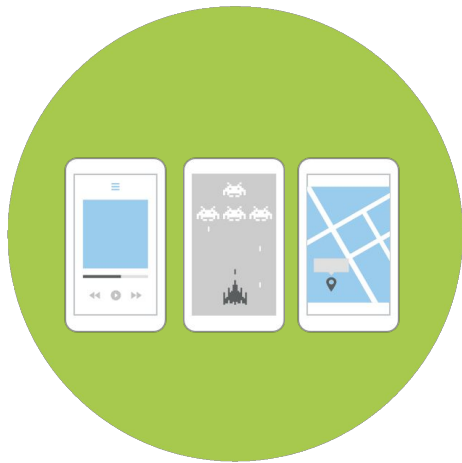
Reto 1

Fundamentos de Xamarin.Forms

¿Qué es Xamarin?



¿Xamarin o Xamarin.Forms?



Xamarin “clásico”

Xamarin.Android
Xamarin.iOS



Xamarin.Forms

Xamarin “clásico”



- Apps que requieran interacción muy especializada
- Apps que usen muchas APIs específicas de la plataforma
- Apps en donde la IU sea más importante que compartir el código

Xamarin.Forms



- Apps de negocio
- Apps de datos
- Apps que requieran poca funcionalidad específica de cada plataforma
- Apps donde compartir el código sea más importante que una IU muy compleja

Estrategias para compartir código



.NET Standard



Proyecto
compartido



.NET Standard

Recomendado

Conjunto de APIs que cada runtime .NET debe implementar

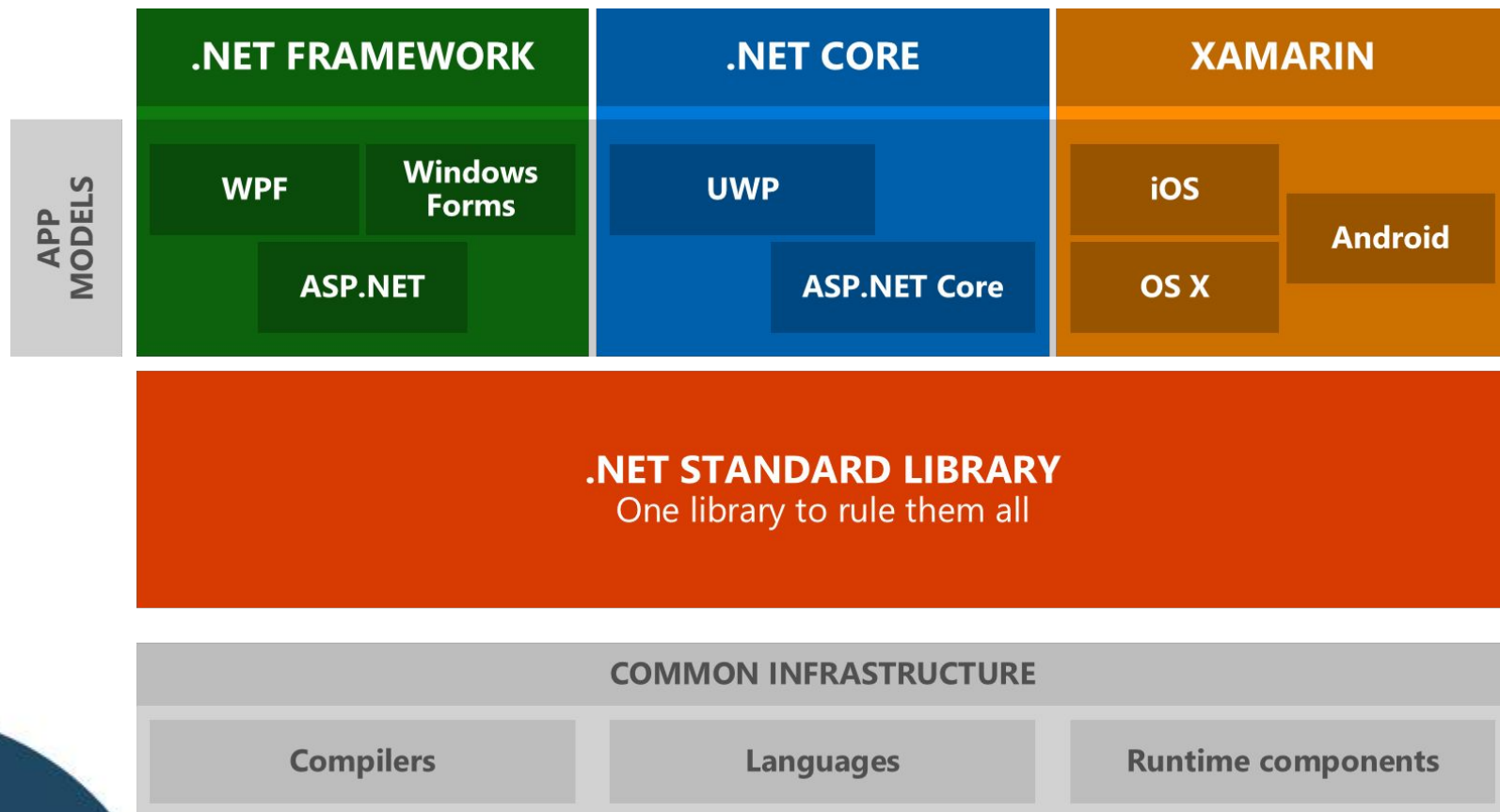


Proyecto compartido

“Hub” para compartir archivos

Requiere directivas `#if` `#endif`

.NET Standard



.NET Standard

.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0
.NET Framework ¹	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1	4.6.1	4.6.1
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299
Windows	8.0	8.0	8.1					
Windows Phone	8.1	8.1	8.1					
Windows Phone Silverlight	8.0							

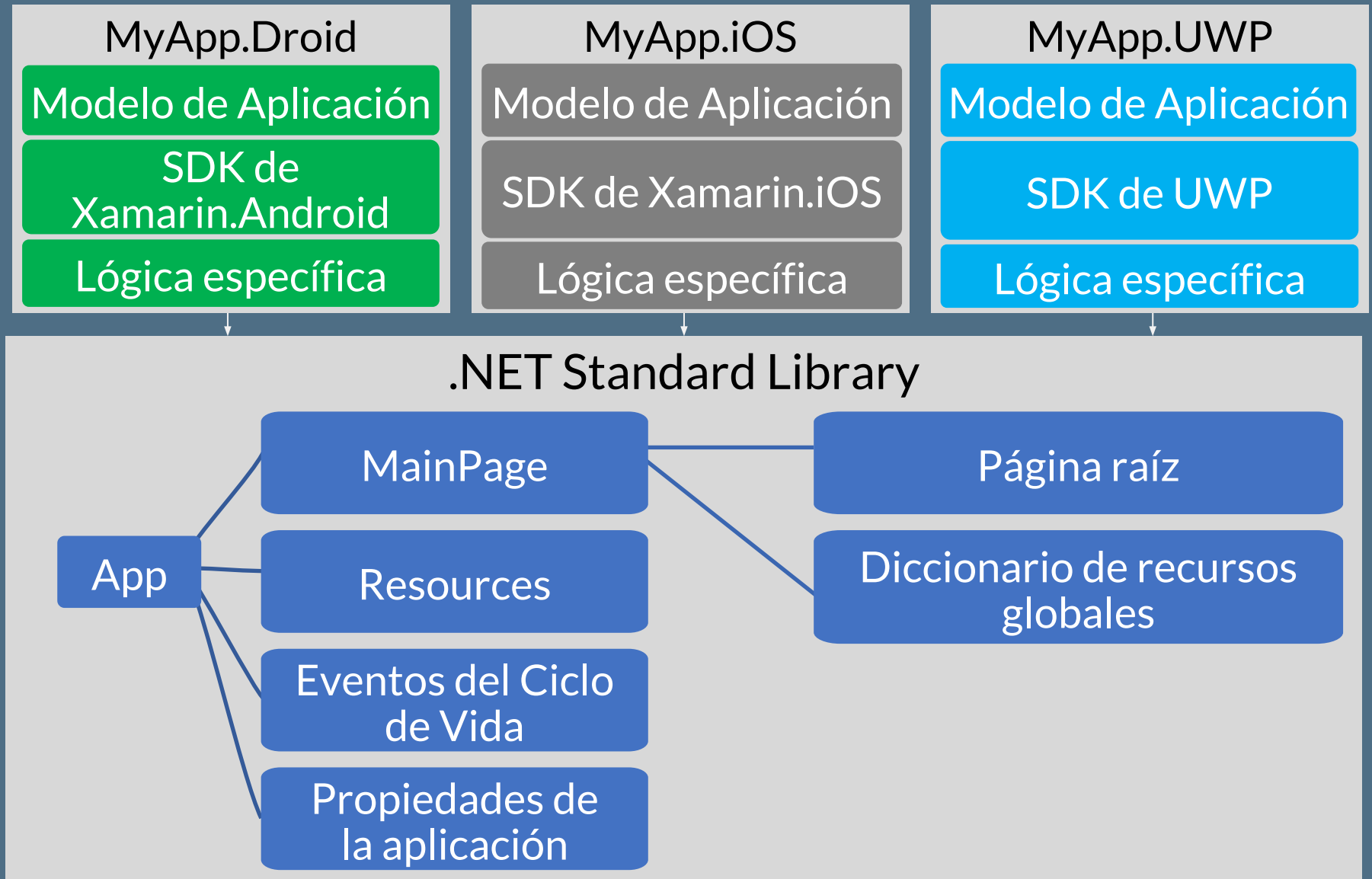
Anatomía de una aplicación

Anatomía de una aplicación

- Visual Studio .NET crea la siguiente estructura en la solución:
 - Proyecto de código común
 - App de Android
 - App de iOS
 - App de UWP

Arquitectura de Xamarin.Forms

Arquitectura de Xamarin.Forms



Ensamblados

- Principales
 - Xamarin.Forms.Core.dll
 - Xamarin.Forms.Platform.dll
 - Xamarin.Forms.Xaml.dll
- Plataforma específica
 - Paquete de Nuget Xamarin.Forms

Jerarquía de clases

Sistema de Propiedades
Enlazables y
Notificaciones

BindableObject

Clase base de los
elementos jerárquicos

Element

Clase base de los
elementos interactivos

VisualElement

Cell

Clase base para todas
las celdas de las listas

Clase base para todos
los controles

View

Page

Clase base para todas las
páginas

Clase base para todos
los elementos de
distribución en pantalla
y paneles contenedores

Layout<T>

Sistema de Propiedades Enlazables

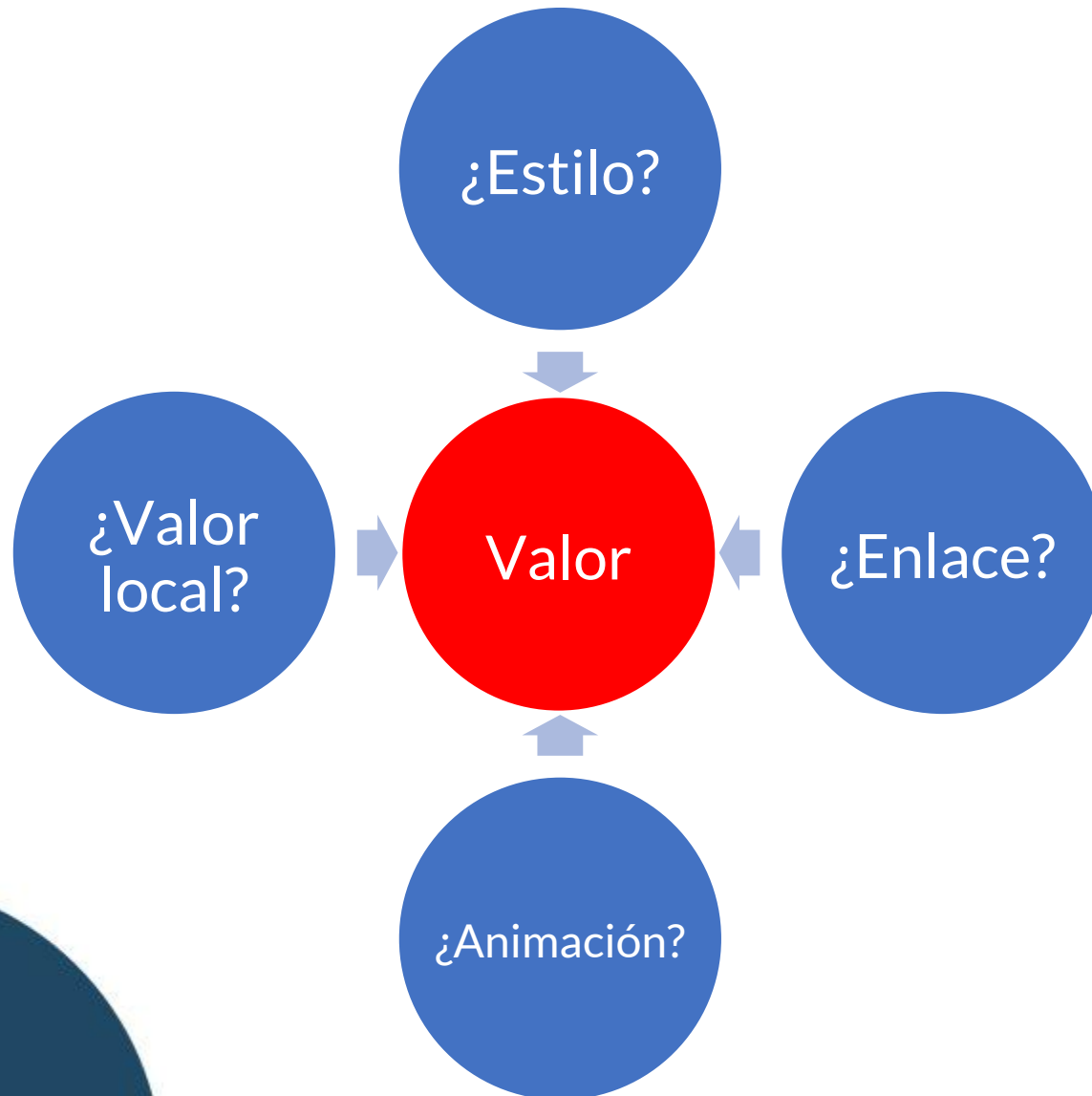
Sistema de Propiedades Enlazables

- El sistema de Propiedades Enlazables permite calcular los valores de las propiedades y notificar cuando un valor ha cambiado
- La clase base BindableObject permite a todas sus clases derivadas usar Propiedades Enlazables

¿Qué es una Propiedad Enlazable?

Son propiedades cuyo valor no lo determinas tú sino que su valor está en función de una gran cantidad de estímulos a su alrededor

¿Qué es una Propiedad Enlazable?



¿Qué es una Propiedad Enlazable?

- Las Propiedades Enlazables nos dan la siguiente funcionalidad:
 - Enlace de Datos
 - Estilización y Plantillas
 - Animación
 - Notificación de cambio de valor

Precedencia de Valores

Animaciones activas

Valor local

Valor proveniente de
la Plantilla

Estilo

Default

- Se consideran también los valores obtenidos a través de {Binding} o {StaticResource}

BindableObject

- Clase base para todos los objetos del sistema de Propiedades Enlazables

BindableObject

- Miembros
 - GetValue()
 - Regresa el valor de una Propiedad Enlazable
 - SetValue()
 - Establece el valor de una Propiedad Enlazable
 - ClearValue()
 - Limpia el valor de una Propiedad Enlazable
 - BindingContext
 - Contexto para Enlace de Datos

BindableProperty

- Representa una Propiedad Enlazable
- Deben cumplir los siguientes requisitos:
 - public
 - static
 - readonly
 - Su nombre debe tener el sufijo “Property”

BindableProperty

- Se acceden a través de una propiedad CLR tradicional
 - La propiedad sólo será una “fachada” para acceder al valor que regrese el Sistema de Propiedades Enlazables
- Opcionalmente, puede indicar un callback para notificar el cambio de valor

```
...public class VisualElement : Element, IAnimatable, IVisualElementController, IElementController, IResourcesProvider
{
    ...public static readonly BindableProperty AnchorXProperty;
    ...public static readonly BindableProperty AnchorYProperty;
    ...public static readonly BindableProperty BackgroundColorProperty;
    ...public static readonly BindableProperty BehaviorsProperty;
    ...public static readonly BindableProperty HeightProperty;
    ...public static readonly BindableProperty HeightRequestProperty;
    ...public static readonly BindableProperty InputTransparentProperty;
    ...public static readonly BindableProperty IsEnabledProperty;
    ...public static readonly BindableProperty IsFocusedProperty;
    ...public static readonly BindableProperty IsVisibleProperty;
    ...public static readonly BindableProperty MinimumHeightRequestProperty;
    ...public static readonly BindableProperty MinimumWidthRequestProperty;
    ...public static readonly BindableProperty NavigationProperty;
    ...public static readonly BindableProperty OpacityProperty;
    ...public static readonly BindableProperty RotationProperty;
    ...public static readonly BindableProperty RotationXProperty;
    ...public static readonly BindableProperty RotationYProperty;
    ...public static readonly BindableProperty ScaleProperty;
    ...public static readonly BindableProperty StyleProperty;
    ...public static readonly BindableProperty TranslationXProperty;
    ...public static readonly BindableProperty TranslationYProperty;
    ...public static readonly BindableProperty TriggersProperty;
    ...public static readonly BindableProperty WidthProperty;
    ...public static readonly BindableProperty WidthRequestProperty;
    ...public static readonly BindableProperty XProperty;
    ...public static readonly BindableProperty YProperty;
```

Usando Propiedades Enlazables

- Aquellas que tienen una propiedad CLR “fachada”, se usan como cualquier otra

```
button1.WidthRequest = 300;  
button1.HeightRequest = 200;  
button1.TextColor = Color.Red;  
  
//etc.
```

Usando Propiedades Enlazables

- Métodos GetValue() y SetValue()

```
if ((Color)boton.GetValue(Button.TextColorProperty) == Color.Black)
{
    boton.SetValue(Button.TextColorProperty, Color.Red);
}
```

- Comúnmente se usan con Propiedades Adjuntas

```
if ((double)boton.GetValue(Grid.ColumnProperty) == 0)
{
    boton.SetValue(Grid.ColumnProperty, 50);
}
```

Creando Propiedades Enlazables

- BindableProperty.Create()
- Ejemplo:

```
public static readonly BindableProperty TextProperty =  
    BindableProperty.Create(nameof(TextProperty),  
        typeof(string), typeof(AnimatedButtonControl),  
        defaultValue: null,  
        propertyChanged: ((bindable, value, newValue) =>  
        {  
            ...  
        }));
```

Propiedades Adjuntas

- Tipo especial de Propiedad Enlazable
- Notifican al elemento padre que se requiere cierto valor
- Si se establece el valor de una propiedad de otro padre, se omitirá

Propiedades Adjuntas

- Sintaxis en XAML:

```
<Grid>  
  <Button Grid.Column="1" />  
</Grid>
```

Propiedades Adjuntas

- Propiedades Adjuntas vía código
 - Puedes utilizar los métodos SetValue() y GetValue()

```
if ((int)control.GetValue(Grid.ColumnProperty) == 0) {  
    control.SetValue(Grid.ColumnProperty, 1);  
}
```

- Puedes usar los métodos específicamente pensados para leer o establecer los valores de las Propiedades Adjuntas
 - Evitan el boxing / unboxing

```
if (Grid.GetColumn(control) == 0) {  
    Grid.SetColumn(control, 1);  
}
```

Manejo de Eventos

Manejo de Eventos

- Podemos manejar eventos de tres maneras diferentes:
 - Desde XAML
 - A través de código usando la sintaxis estándar
 - A través de código usando Expresiones Lambda

Eventos

ModalPushing

ModalPushed

ModalPopping

ModalPopped

Propiedades

- **Application.Properties**
 - Diccionario para guardar datos de cualquier tipo
 - Solo tipos primitivos y serializables
 - Los datos son guardados en el dispositivo automáticamente
- **Application.SavePropertiesAsync()**
 - Proactivamente guarda los datos del diccionario de propiedades

El Lenguaje XAML

¿Qué es XAML?

- Lenguaje de Marcado, acrónimo de eXtensible Application Markup Language
- Expresa, declarativamente, las interfaces de usuario en aplicaciones Xamarin.Forms, WPF, UWP, etc.
- Lenguaje para instanciar objetos

Reglas básicas de XAML

- Mismas reglas que en XML
 - Debe haber un nodo raíz
 - Todo elemento debe cerrar
 - Sensible a mayúsculas y minúsculas
 - Hay elementos y subelementos
 - Hay atributos

Reglas básicas de XAML

- Los elementos instancian un objeto

```
<Entry />
```

- Los atributos cambian las características los objetos

```
<Entry Text="Hola" />
```

- El anidamiento implica pertenencia
 - Es un acceso directo para establecer el contenido

```
<Grid>  
    <Entry />  
</Grid>
```


Reglas básicas de XAML

- A todo elemento le corresponde una clase que la respalda en el API de Xamarin.Forms
 - Ejemplos:

Elemento XAML	Clase
<Label />	Xamarin.Forms.Label
<Entry />	Xamarin.Forms.Entry
<Button />	Xamarin.Forms.Button
<ListView />	Xamarin.Forms.ListView
<ContentPage />	Xamarin.Forms.ContentPage

Espacios de nombres XML

- Usados para organizar lógicamente las clases
 - <http://xamarin.com/schemas/2014/forms>
 - Espacio de nombres base. Abarca varios espacios de nombres CLR
 - <http://schemas.microsoft.com/winfx/2009/xaml>
 - Espacio de nombres del lenguaje XAML

Espacios de nombres XML

- El atributo xmlns importa un Espacio de nombres XML para poner a nuestro alcance todos sus miembros incluidos en él
 - Similar a la sentencia using en C# o Imports en VB
- De manera opcional, podemos asignar un alias a un Espacio de nombres (xmlns:alias)

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"  
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
... />
```

Espacios de nombres XML

- Podemos establecer el atributo xmlns en cualquier nivel
 - Sin embargo, la mejor práctica es colocarlo en la raíz
 - [ContentPage](#)
 - [ContentView](#)
 - Etc.

```
<otro:ContentPage xmlns:otro="http://xamarin.com/schemas/2014/forms"
                  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                  x:Class="App4.MainPage">

    <otro:Button Text="Botón" />

</otro:ContentPage>
```

Espacios de nombres XML

- Miembros del Espacio de nombres del lenguaje XAML

x:Class	Indica el nombre totalmente cualificado de la clase de código relacionada
x:FactoryMethod	Indica el método estático a utilizar para inicializar el objeto en vez del constructor
x:Key	Especifica el identificador único de un objeto dentro de un Diccionario de Recursos
x:Name	Especifica el nombre del objeto para poder referenciarlo en el código
x:Arguments	Especifica los argumentos para un constructor para inicializar un objeto
x:TypeArguments	Indica el tipo de los argumentos para un constructor no predeterminado

Nombres para los elementos

- El atributo x:Name sirve para asignar un nombre único a los elementos
- Solo le debes poner nombre a los elementos que necesitas referenciar en XAML o en C#
 - Poner nombre a todos los elementos es mala práctica

```
<Label x:Name="CountryLabel" />
```

```
CountryLabel.Text = persona.Pais;
```

```
... Text="{Binding Source={x:Reference CountryLabel}}"
```

Sintaxis de Subelementos

- Todas las propiedades se pueden establecer como subelementos
- Necesario únicamente cuando el valor de la propiedad es un objeto complejo y no se cuenta con un Convertidor de Tipo

```
<Button Text="Botón"  
TextColor="Red"  
FontSize="22" />
```



```
<Button>  
  <Button.Text>Botón</Button.Text>  
  <Button.TextColor>Red</Button.TextColor>  
  <Button.FontSize>22</Button.FontSize>  
</Button>
```

Sintaxis de Contenido

- Algunos tipos soportan la sintaxis de contenido
 - Clases decoradas con el atributo [ContentProperty]

```
<Grid>  
  <Grid.Children>  
    <Button />  
  </Grid.Children>  
</Grid>
```



```
<Grid>  
  <Button />  
</Grid>
```

```
<Label Text="Hola" />
```



```
<Label>Hola</Label>
```


Extensiones de Marcado

- Clases que permiten al intérprete de XAML el evaluar de manera diferente sus valores
 - {StaticResource}
 - {DynamicResource}
 - {Binding}
 - {TemplateBinding}
 - {x:Null}
 - {x:Static}
 - {x>Type}
 - {x:Reference}
 - {x:Arguments}

```
<Elemento Propiedad="{Extension [Propiedades]}" />
```

Extensiones de Marcado

- **{StaticResource}**
 - Referencia recursos de manera estática
- **{DynamicResource}**
 - Referencia recursos de manera dinámica
- **{Binding}**
 - Expresa un enlace de datos
- **{TemplateBinding}**
 - Enlaza el valor de una propiedad de una plantilla a un valor concreto de la clase que la implementa

Extensiones de Marcado

- **{x:Null}**
Permite asignar como valor un null
- **{x:Reference}**
Permite enlace entre elementos
- **{x:Static}**
Permite acceder a un campo estático, constant o enumeración
- **{x:FactoryMethod} y {x:Arguments}**
Permiten invocar un método y establecer parámetros

Recursos

Recursos

- Los recursos son un diccionario de elementos
- La clase base VisualElement implementa la propiedad Resources
 - Debe declararse un elemento de tipo ResourceDictionary
 - Permiten guardar cualquier tipo de objeto
 - Se identifican por una clave única (atributo x:Key)

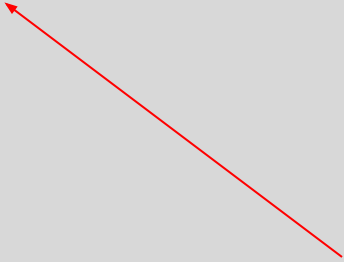
```
<Grid.Resources>  
  <ResourceDictionary>  
    <Color x:Key="MiColor">Red</Color>  
  </ResourceDictionary>  
</Grid.Resources>
```

Recursos

- Se referencian en XAML por medio de {StaticResource} o {DynamicResource}
- Pueden ser usadas con propiedades regulares CLR

```
<ContentPage.Resources>
    <ResourceDictionary>
        <Color x:Key="MiColor">Red</Color>
    </ResourceDictionary>
</ContentPage.Resources>

<Grid>
    <BoxView Color="{StaticResource MiColor}" />
</Grid>
```




Recursos

- Siempre se referenciará al recurso con nivel de visibilidad más cercano

```
<ContentPage.Resources>
  <ResourceDictionary>
    <Color x:Key="MiColor">Red</Color>
  </ResourceDictionary>
</ContentPage.Resources>

<Grid>
  <Grid.Resources>
    <ResourceDictionary>
      <Color x:Key="MiColor">Blue</Color>
    </ResourceDictionary>
  </Grid.Resources>

  <BoxView Color="{StaticResource MiColor}" />
</Grid>
```



Recursos

- Recursos vía código
 - En código usamos la propiedad Resources para leer o agregar objetos al diccionario de recursos
 - La propiedad Resources es de tipo ResourceDictionary el cual es un diccionario de tipo <string, object>

```
private void Button_OnClicked(object sender, EventArgs e) {  
    if (Resources.ContainsKey("MiColor")) {  
        var myColor = (Color)Resources["MiColor"];  
    }  
  
    Resources.Add("pi", 3.1416);  
}
```

Recursos

- **{StaticResource}**
 - Se resuelve durante la construcción de los objetos
 - Es asignado una sola vez y los cambios son ignorados
 - Debe estar declarado en XAML antes de utilizarse
- **{DynamicResource}**
 - Es evaluado y resuelto cada vez que el elemento necesita el recurso

XAML compilado

XAML compilado

- Atributo Xamarin.Forms.Xaml.XamlCompilation
 - Indica si el XAML será compilado
 - Mejora el desempeño de la aplicación
 - Chequeo del código XAML en tiempo de compilación
 - Los archivos .xaml no serán incluidos en el ensamblado final
 - Puede ser usado a nivel de ensamblado o clase

```
//Ensamblado  
[assembly:XamlCompilation(XamlCompilationOptions.Compile)]  
...  
//Clase  
[XamlCompilation(XamlCompilationOptions.Compile)]
```

Contenedores

Contenedores

- Controlan la renderización de elementos
 - Tamaño y dimensiones
 - Posición
 - Acomodo de elementos hijos
- Heredan de `Layout<T>`
 - `StackLayout`
 - `Grid`
 - `FlexLayout`
 - ...

StackLayout

- Contenedor para apilar elementos de manera horizontal o vertical

StackLayout

```
<StackLayout Orientation="Horizontal">  
  <BoxView Color="Red"  
    HorizontalOptions="Center" />  
  <BoxView Color="Green"  
    HorizontalOptions="Center" />  
  <BoxView Color="Blue"  
    HorizontalOptions="Center" />  
</StackLayout>
```



StackLayout

```
<StackLayout Orientation="Horizontal">  
  <BoxView Color="Red"  
    HorizontalOptions="Center" />  
  <BoxView Color="Green"  
    HorizontalOptions="Center" />  
  <BoxView Color="Blue"  
    HorizontalOptions="Center" />  
</StackLayout>
```



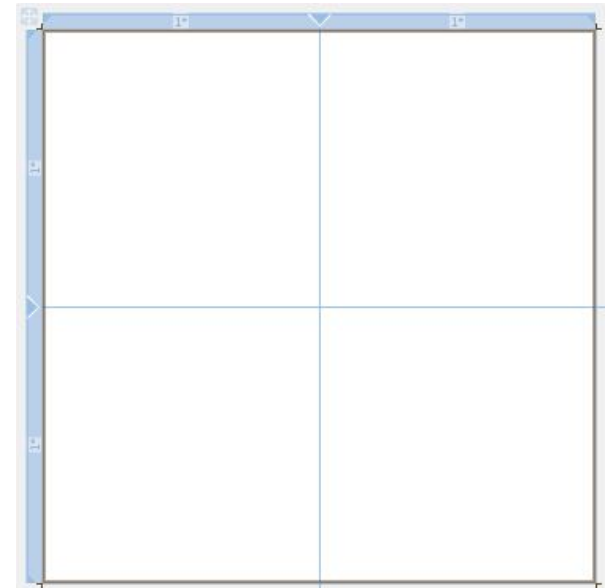
Grid

- Similar a una tabla
- El más flexible de todos los contenedores
 - Predeterminado
- Requiere columnas y filas
- Propiedades Adjuntas:
 - Column
 - Row
 - ColumnSpan
 - RowSpan

Grid

- Propiedades ColumnDefinitions y RowDefinitions
 - Permiten definir las columnas y filas para el Grid
 - Inherentemente tiene una sola celda

```
<Grid>  
  <Grid.ColumnDefinitions>  
    <ColumnDefinition />  
    <ColumnDefinition />  
  </Grid.ColumnDefinitions>  
  
  <Grid.RowDefinitions>  
    <RowDefinition />  
    <RowDefinition />  
  </Grid.RowDefinitions>  
</Grid>
```



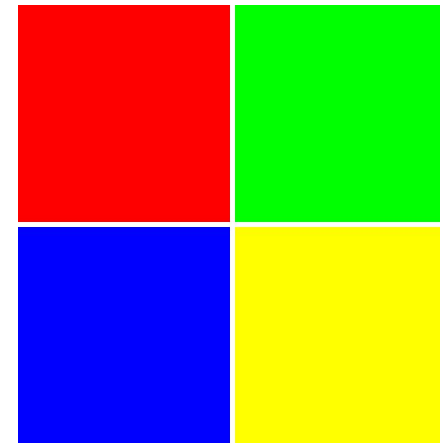
Grid

- Las Propiedades Adjuntas posicionan los elementos

```
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>

  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>

  <BoxView Color="Red" />
  <BoxView Color="Lime" Grid.Column="1" />
  <BoxView Color="Blue" Grid.Row="1" />
  <BoxView Color="Yellow" Grid.Row="1"
    Grid.Column="1" />
</Grid>
```



Grid – Tamaños de Columnas y Filas



Fijo

Valor absoluto



Automático

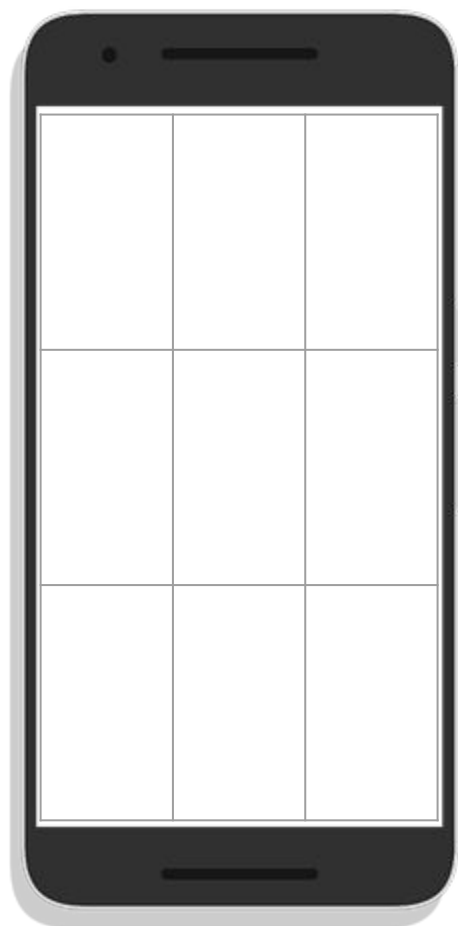
Crece en función
de su contenido

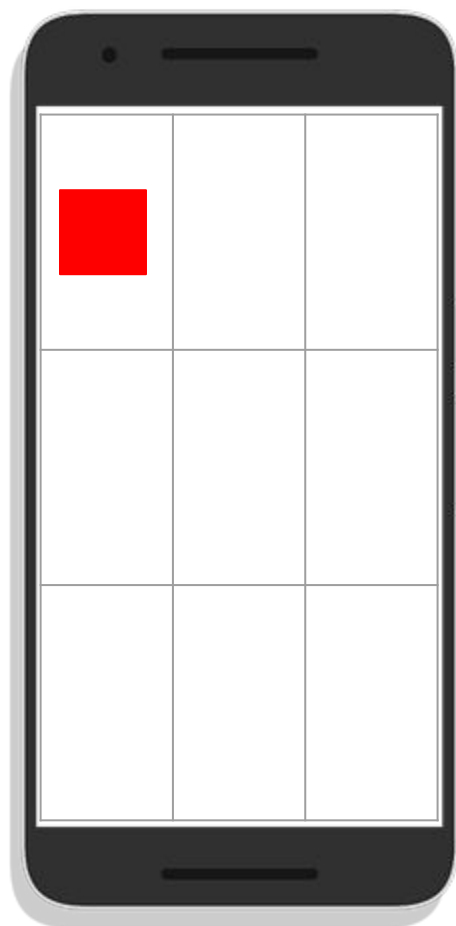


Proporcional

Factor expresado
con un número y *

Reto 2





Controles Personalizados y Renderers

<Entry />



Entry Custom
Renderer

EditText



Entry Custom
Renderer

UIView



Entry Custom
Renderer

TextBox

Estilos

Estilos

- Xamarin.Forms.Style
- Elementos que nos permiten cambiar un grupo de propiedades
- Similares a CSS

```
<Style x:Key="EstiloBotonBase" TargetType="Button">  
    <Setter Property="WidthRequest"  
            Value="200" />  
    <Setter Property="TextColor"  
            Value="Yellow" />  
</Style>
```

Estilos

- Propiedad TargetType
 - Indica el tipo al cual el estilo puede aplicarse

```
<Style x:Key="EstiloBotonBase" TargetType="Button">
```

```
...
```

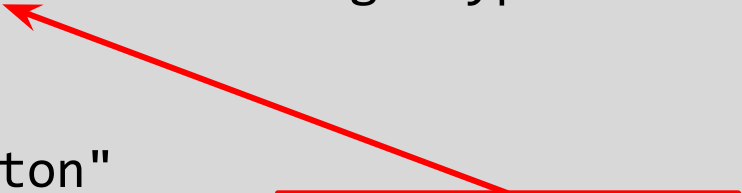
```
</Style>
```

```
<Button Style="{StaticResource EstiloBotonBase}" />
```

Estilos

- Propiedad BasedOn
 - Permite crear estilos en cascada

```
<Style x:Key="EstiloBotonBase" TargetType="Button">
...
</Style>
<Style x:Key="EstiloBoton"
    BasedOn="{StaticResource EstiloBotonBase}"
    TargetType="Button">
    <Setter Property="TextColor"
        Value="Blue" />
    <Setter Property="FontSize"
        Value="20" />
</Style>
```



Estilos

- Estilos Implícitos
 - No utilizan x:Key
 - Se basan en el TargetType

```
<Style TargetType="Label">  
  <Setter Property="FontFamily" Value="Impact" />  
  <Setter Property="FontSize" Value="20" />  
  <Setter Property="TextColor" Value="Yellow" />  
  <Setter Property="BackgroundColor" Value="Black" />  
</Style>
```

```
<Label Text="Texto 1" />  
<Label Text="Texto 2" />
```

Texto 1

Texto 2

Estilos

- ApplyToDerivedTypes

```
public class MyLabel : Label {  
    ...  
}
```

```
<Style TargetType="Label" ApplyToDerivedTypes="True">  
    <Setter Property="FontFamily" Value="Impact" />  
    <Setter Property="FontSize" Value="20" />  
    <Setter Property="TextColor" Value="Yellow" />  
    <Setter Property="BackgroundColor" Value="Black" />  
</Style>
```

```
<Label Text="Texto 1" />
```

Texto 1

```
<Label Text="Texto 2" />
```

Texto 2

```
<local:MyLabel Text="Texto 3" />
```

Texto 3

Estilos

- **Triggers en Estilos**
 - Expresan acciones que modifican la apariencia de los controles según alguna lógica
 - Tipos soportados
 - **Trigger**
 - **DataTrigger**
 - **EventTrigger**
 - **MultiTrigger**
 - Se establecen en la colección Triggers de la clase Style

Estilos

- Triggers en Estilos
 - Ejemplo Trigger: Cambiar el texto de un Entry, solo si tiene el enfoque el control

```
<Style x:Key="MyEntryStyle" TargetType="Entry">
  <Style.Triggers>
    <Trigger TargetType="Entry" Property="IsFocused" Value="True">
      <Setter Property="TextColor" Value="Red" />
    </Trigger>
  </Style.Triggers>
</Style>
```

Estilos

- Triggers en Estilos
 - Ejemplo DataTrigger: Habilitar un botón en función del texto de un Entry

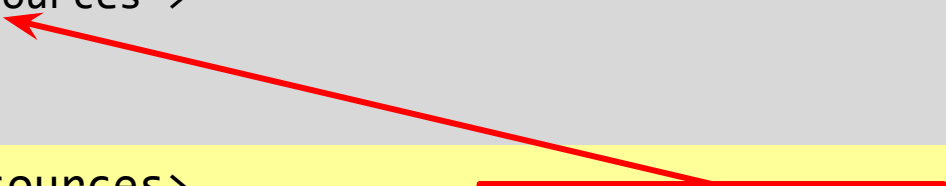
```
<Style TargetType="Button">
  <Style.Triggers>
    <DataTrigger
      TargetType="Button"
      Binding="{Binding Text.Length, Source={x:Reference Nombre}}" Value="0">
      <Setter Property="IsEnabled" Value="False" />
    </DataTrigger>
  </Style.Triggers>
</Style>
```

Estilos

- Diccionarios Mezclados
 - Combinan un Diccionario de Recursos con otro
 - Un ensamblado externo contiene el Diccionario de Recursos

```
<?xml version="1.0" encoding="utf-8" ?>
<ResourceDictionary xmlns="http://xamarin.com/schemas/2014/forms"
                    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                    x:Class="App4.MyResources">
  <Color x:Key="Color1">Aqua</Color>
  ...
</ResourceDictionary>
```

```
<ContentPage.Resources>
  <ResourceDictionary MergedWith="local:MyResources">
    <Style>...
  </ResourceDictionary>
</ContentPage.Resources>
```



Transformaciones y Animaciones

Transformaciones

- Propiedades de la clase base **VisualElement**
 - Rotation
 - RotationX
 - RotationY
 - TranslationX
 - TranslationY
 - Scale

Animaciones

- Xamarin.Forms proporciona dos clases para animación:
 - ViewExtensions
 - AnimationExtensions

ViewExtensions

- Clase con métodos de extensión para la clase `VisualElement`, para iniciar animaciones predefinidas:
 - Desvanecimiento
 - Escala
 - Rotación
 - Traslación

ViewExtensions

```
//Desvanecimiento  
await this.FadeTo(0, 50);
```

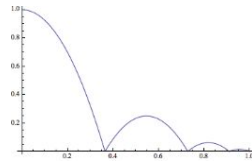
```
//Rotación  
await this.RotateTo(360, 600, Easing.SinIn);
```

```
//Escala  
await this.ScaleTo(0.5, 450, Easing.Linear);
```

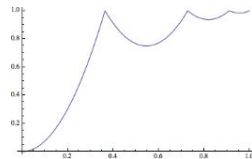
```
//Traslación  
await this.TranslateTo(100, 0,  
easing:Easing.CubicOut);
```


Funciones de suavización

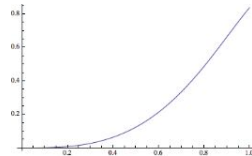
BounceIn



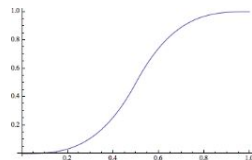
BounceOut



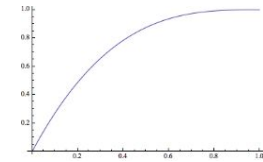
CubicIn



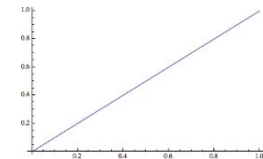
CubicInOut



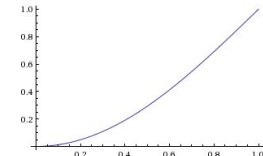
CubicOut



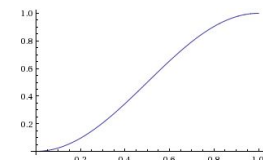
Linear



SinIn

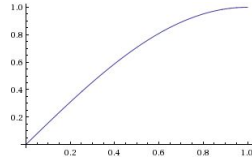


SinInOut

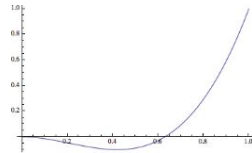


Funciones de suavización

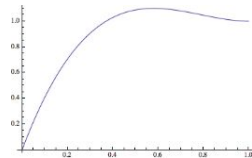
SinOut



SpringIn



SpringOut



Navegación

Navegación

La clase VisualElement incluye la propiedad Navigation

Navegación

- INavigation
 - Miembros para navegación jerárquica
 - `PushAsync()`
 - `PopAsync()`
 - Miembros para navegación modal
 - `PushModalAsync()`
 - `PopModalAsync()`

Navegación jerárquica

- La propiedad MainPage de la aplicación debe ser de tipo NavigationPage
 - Navegación de LIFO (Last In, First Out)
 - Se recomienda que solo sean ContentPage
 - La propiedad Title será usada como título
 - Los métodos OnAppearing y OnDisappearing son invocados cuando se navega hacia o fuera de una página respectivamente

Navegación jerárquica

- Ejemplo:
 - Estableciendo NavigationPage como página principal

```
public class App : Application
{
    public App() {
        // Indica que la página raíz será Page1
        MainPage = new NavigationPage(new Page1());
    }
    ...
}
```

Navegación jerárquica

- Ejemplo:
 - Navegando de Page1 a Page2

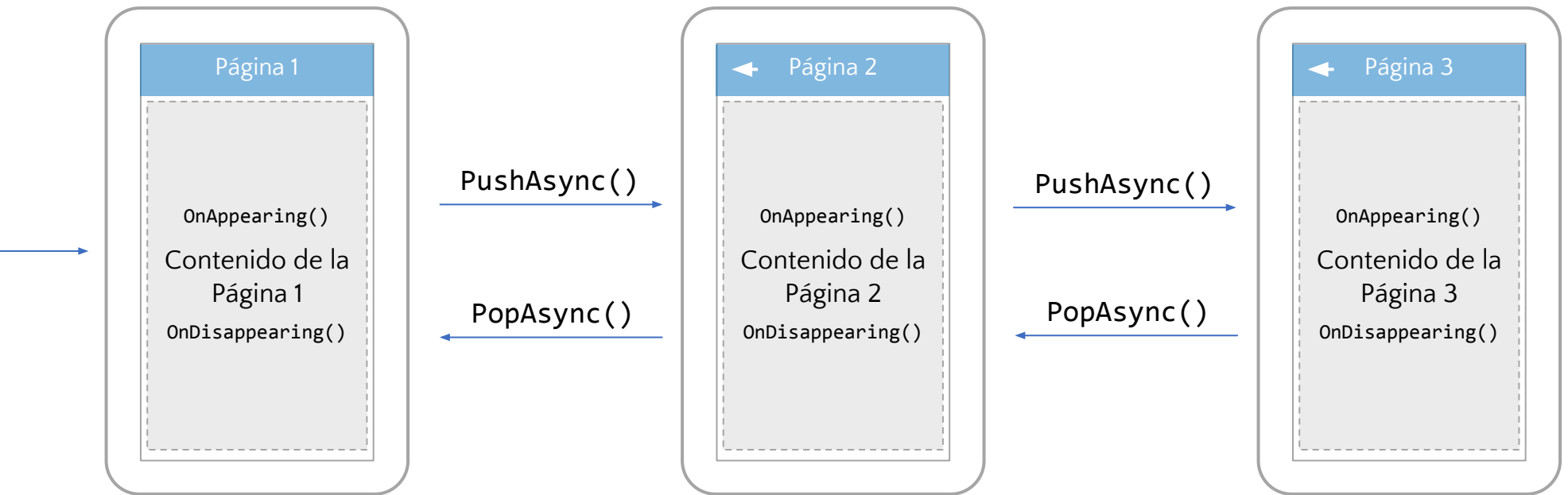
```
await Navigation.PushAsync(new Page2());
```

- Navegando programáticamente de regreso, de Page2 a Page1

```
await Navigation.PopAsync();
```

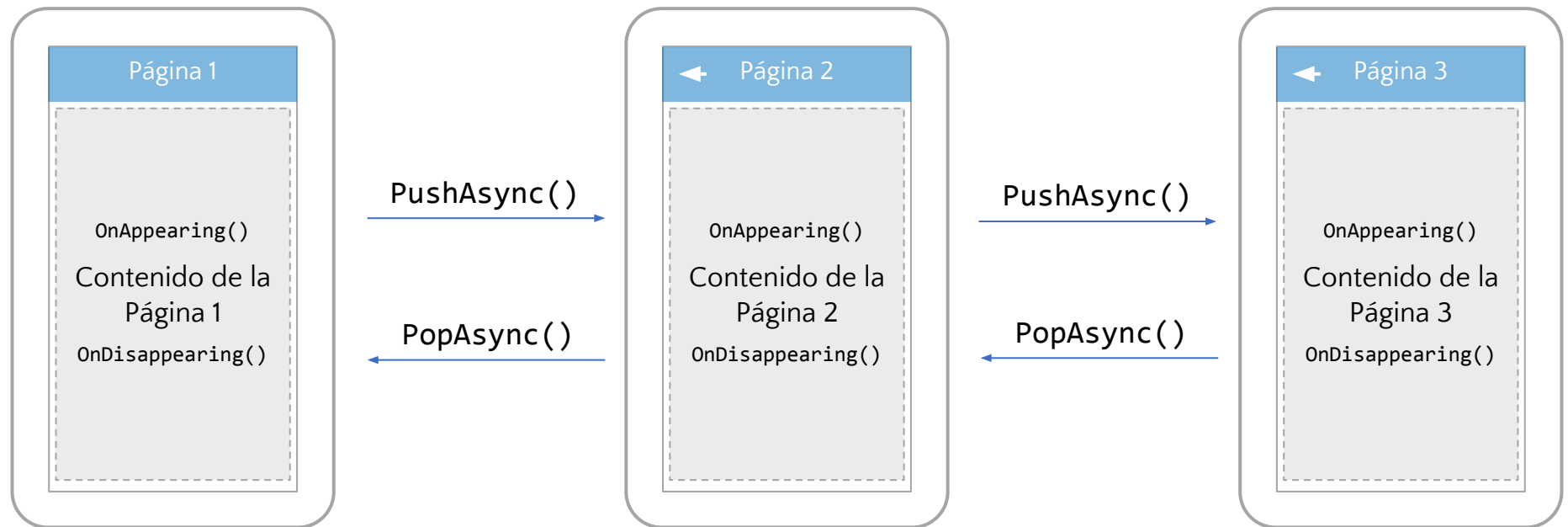

Navegación jerárquica

- Android



Navegación jerárquica

- iOS y UWP



Navegación modal

Presenta las páginas de manera “modal”

No requiere una instancia de `NavigationPage`

```
await Navigation.PushModalAsync(new Pagina3());
```

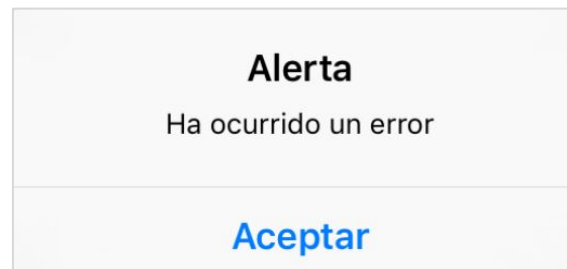
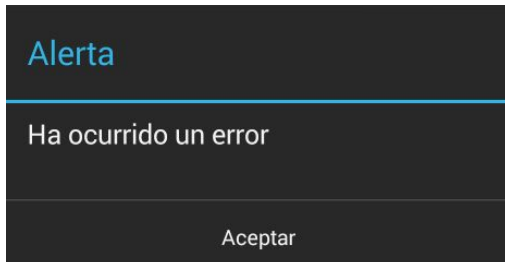
Cajas de diálogo y mensajes

DisplayAlert

- Clase Page
- Muestra una caja de diálogo con uno o dos botones

```
//Un solo botón
```

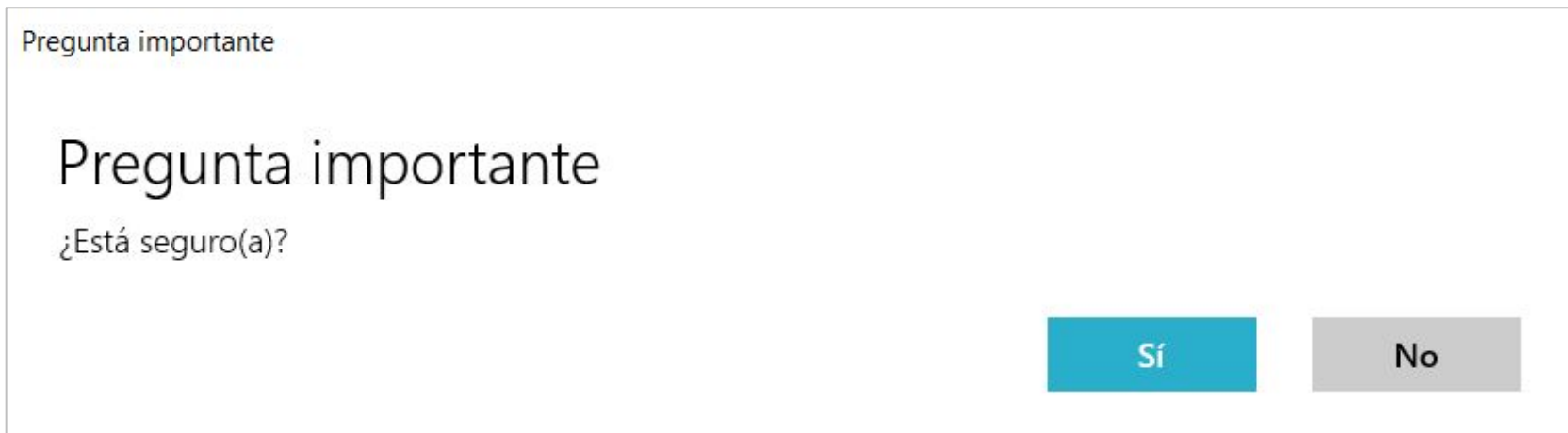
```
await DisplayAlert("Alerta", "Ha ocurrido un error",  
"Aceptar");
```



DisplayAlert

- Uso como función

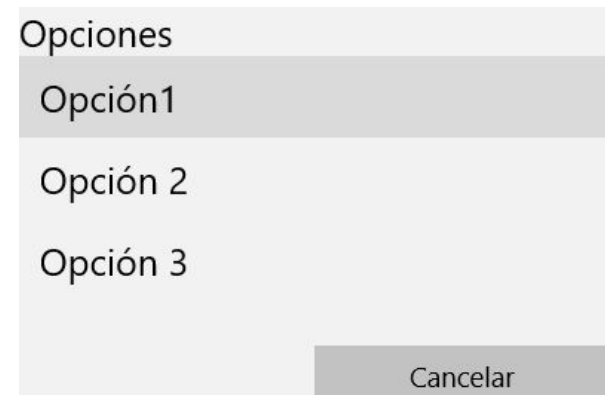
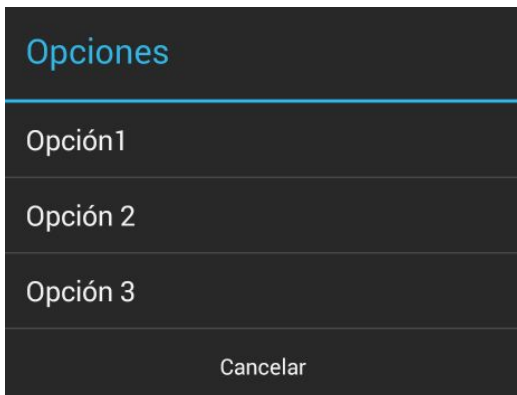
```
var result =  
await DisplayAlert("Pregunta importante", "¿Está  
seguro(a)?", "Sí", "No");
```



DisplayActionSheet

- Clase Page
- Muestra una caja de diálogo con una lista de opciones

```
var resultado =  
    await DisplayActionSheet("Opciones", "Cancelar", null,  
        "Opción1", "Opción 2", "Opción 3");
```



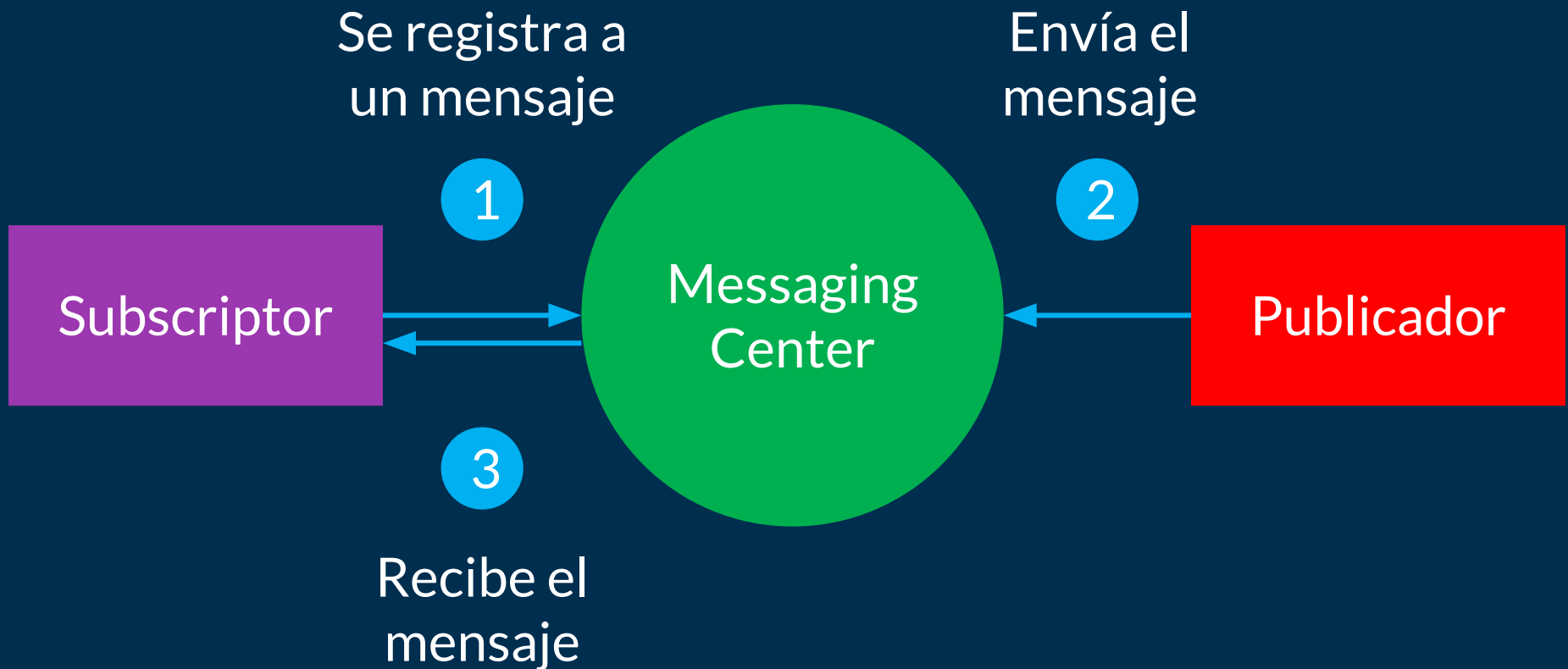
MessagingCenter

- Servicio para enviar y recibir mensajes entre diferentes objetos
 - Ejemplo: entre páginas
 - Útil en los ViewModel de MVVM
- Permite reducir el acoplamiento

MessagingCenter

- Basado en el Patrón de Diseño Pub/Sub
- Miembros
 - Send()
 - Subscribe()
 - Unsubscribe()

MessagingCenter



MessagingCenter

- Envío de mensajes
 - Envío simple

```
MessagingCenter.Send(this, "Mensaje");
```

- Envío de un mensaje con argumentos

```
MessagingCenter.Send(this, "Mensaje", "arg");
```

MessagingCenter

- Subscripción a mensajes
 - Subscripción simple

```
MessagingCenter.Subscribe<Page2>(this, "Mensaje",  
    async (s) => {  
        await DisplayAlert("Mensaje", "Hola desde Página 1", "Aceptar");  
    });
```

MessagingCenter

- Suscripción a un mensaje con argumentos

```
MessagingCenter.Subscribe<Page2, string>(this, "Mensaje",  
    async (s, a) => {  
        await DisplayAlert("Mensaje", "Hola desde Página 1", "Aceptar");  
    });
```

MessagingCenter

- Quitar subscripción

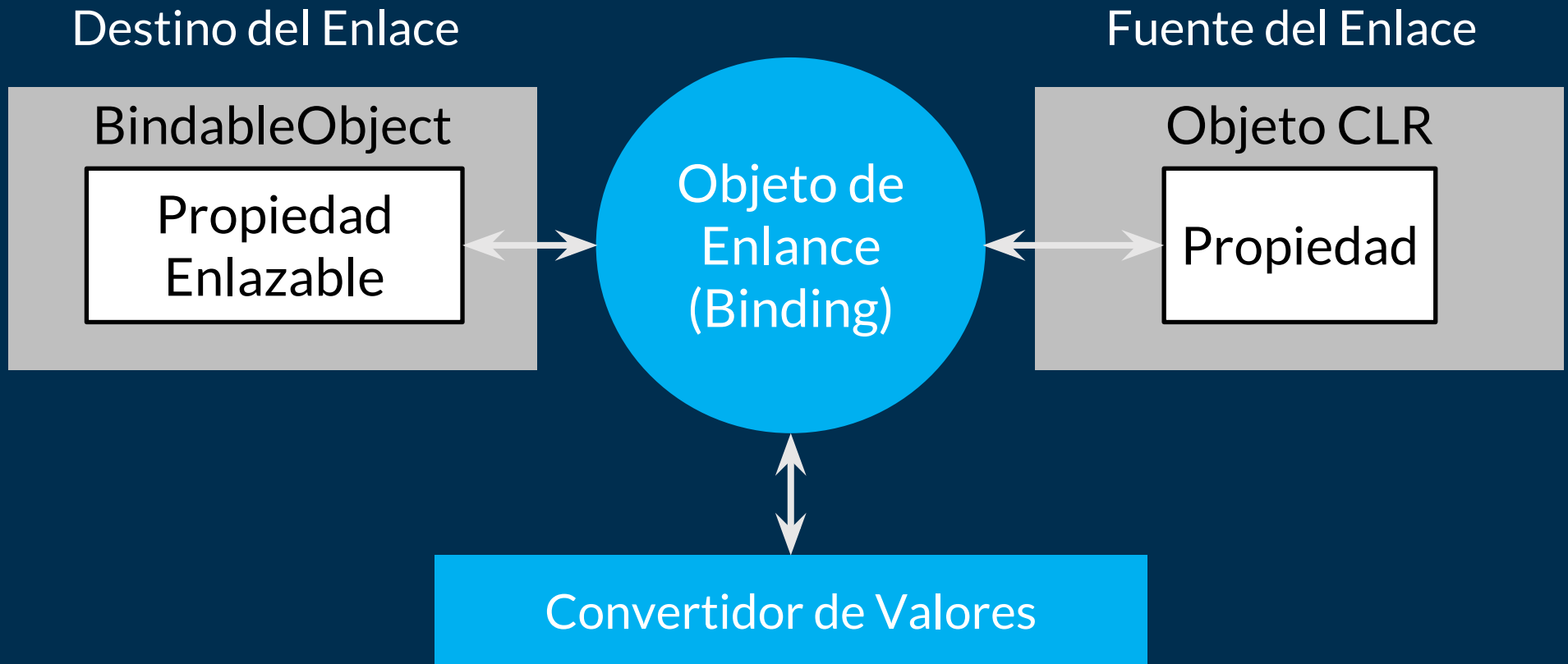
```
MessagingCenter.Unsubscribe<Page2>(this, "Mensaje");
```

Enlace de Datos

Enlace de Datos

Es el proceso que establece la conexión entre la Interfaz de Usuario y la lógica de negocio

Enlace de Datos



Enlace de Datos

- Modelo robusto
 - Objeto Fuente
 - Propiedad en el Objeto Fuente
 - Control Destino
 - Propiedad en el Control Destino
- Se expresan a través de la extensión del Markup {Binding}

```
<Label Text="{Binding Source={StaticResource album1}, Path=Titulo}" />
```

Enlace de Datos

- Propiedades principales de {Binding}
 - Source
 - Indica el Objeto fuente
 - Path
 - Indica una ruta a la Propiedad en el Objeto fuente
 - Parámetro predeterminado
 - Mode
 - Indica el modo para el Enlace

Enlace de Datos

- No se establece la propiedad Path cuando se necesita enlazar todo un objeto

```
<ListView ItemsSource="{Binding}" />
```

Modos de Enlace

- Default
 - Modo predeterminado de la Propiedad Enlazable

Modos de Enlace

- OneTime
 - El enlace se establece, y el Objeto fuente y Control destino no se vuelven a comunicar entre sí

Modos de Enlace

- OneWay
 - El enlace se establece y cualquier cambio que suceda en el Objeto fuente le avisará al Control destino

Modos de Enlace

- OneWayToSource
 - El enlace se establece y cualquier cambio que suceda en el Control destino le avisará al Objeto fuente

Modos de Enlace

- TwoWay
 - El enlace se establece y cualquier cambio que suceda en el Objeto fuente le avisará al Control destino y viceversa (bi-direccional)

Modos de Enlace

- El modo predeterminado puede variar según la clase y Propiedad Enlazable
- Se declaran explícitamente a través del atributo Mode
 - Mejor práctica

```
<Label  
  Text="{Binding Source={StaticResource disco1},  
              Path=Titulo, Mode=OneWay}" />
```

Interfaces de notificación

- Los modos OneWay y TwoWay requieren que el Objeto fuente implemente alguna de las Interfaces **INotifyPropertyChanged** o **INotifyCollectionChanged**
- Implementan un evento el cual es “escuchado” en la infraestructura de Enlace de Datos

Interfaces de notificación

- `INotifyPropertyChanged` se usa en objetos individuales, para notificar que una propiedad ha cambiado
- `INotifyCollectionChanged` se usa en colecciones, para notificar que la colección ha cambiado

Interfaces de notificación

- Generalmente, no es necesario implementar directamente la interfaz `INotifyCollectionChanged`
 - Usa la clase `ObservableCollection<T>` para tus colecciones

ObservableCollection<T>

- Colección genérica recomendada para Enlace de Datos
- Implementa INotifyPropertyChanged y INotifyCollectionChanged
- Automáticamente actualiza el control de lista enlazado
- System.Collections.ObjectModel

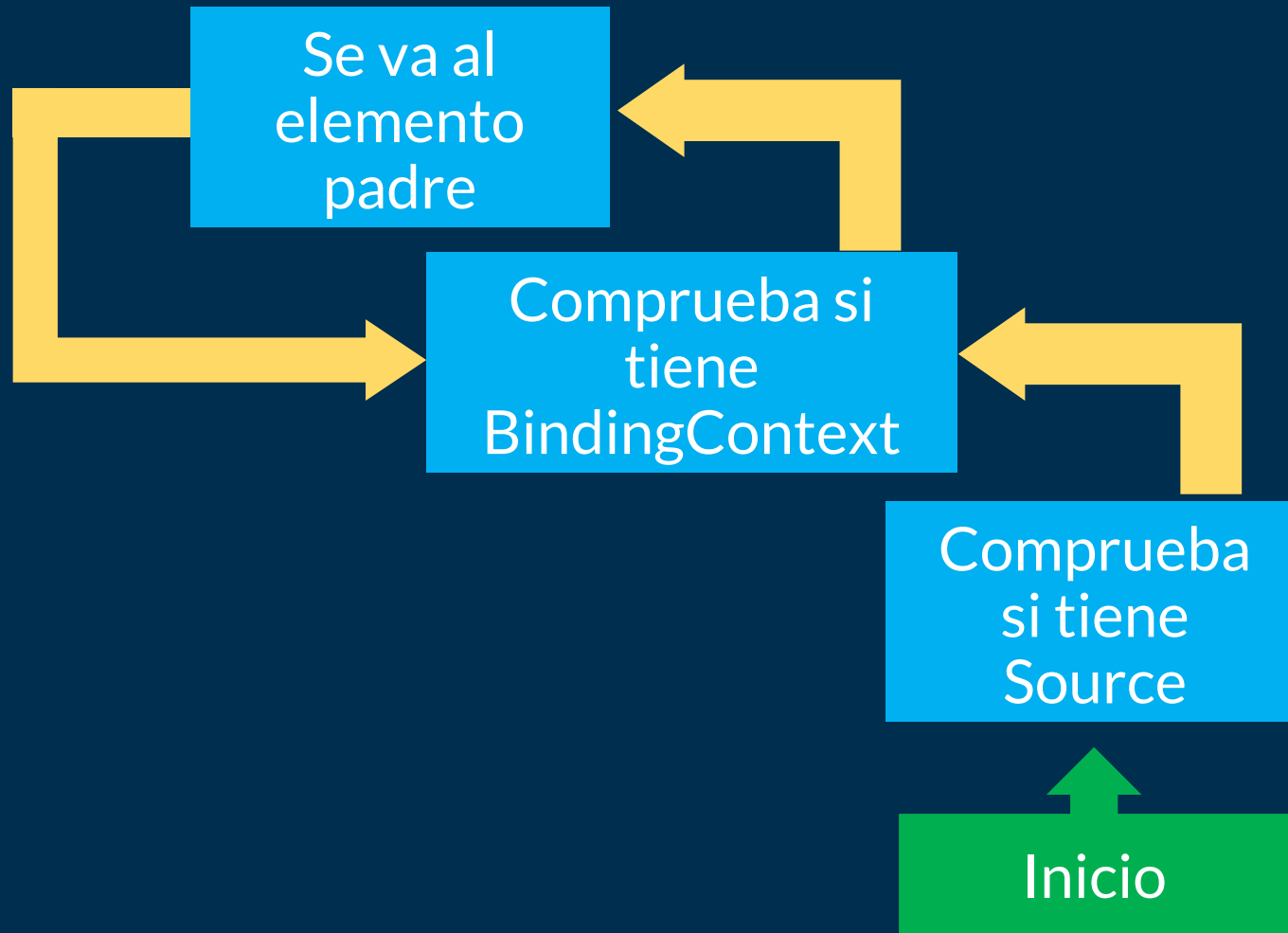
Contexto de Enlace de Datos

- `BindableObject.BindingContext`
 - Incluso en `Application`
- Establece un Objeto como fuente de datos en un control o en un contenedor padre
- Útil cuando estamos enlazando múltiples propiedades de la misma Fuente

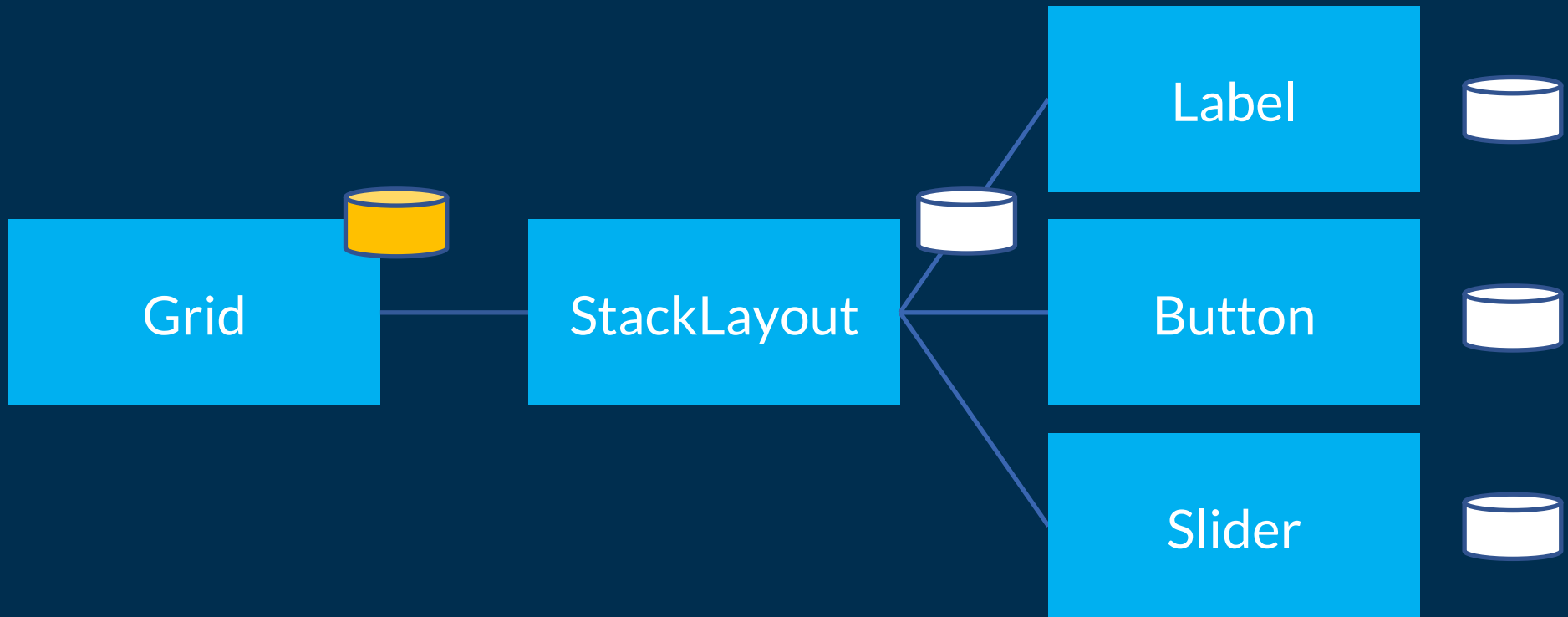
Contexto de Enlace de Datos

```
<Grid BindingContext="{StaticResource album1}">  
    <Label Text="{Binding Path=Titulo, Mode=OneWay}" />  
</Grid>
```

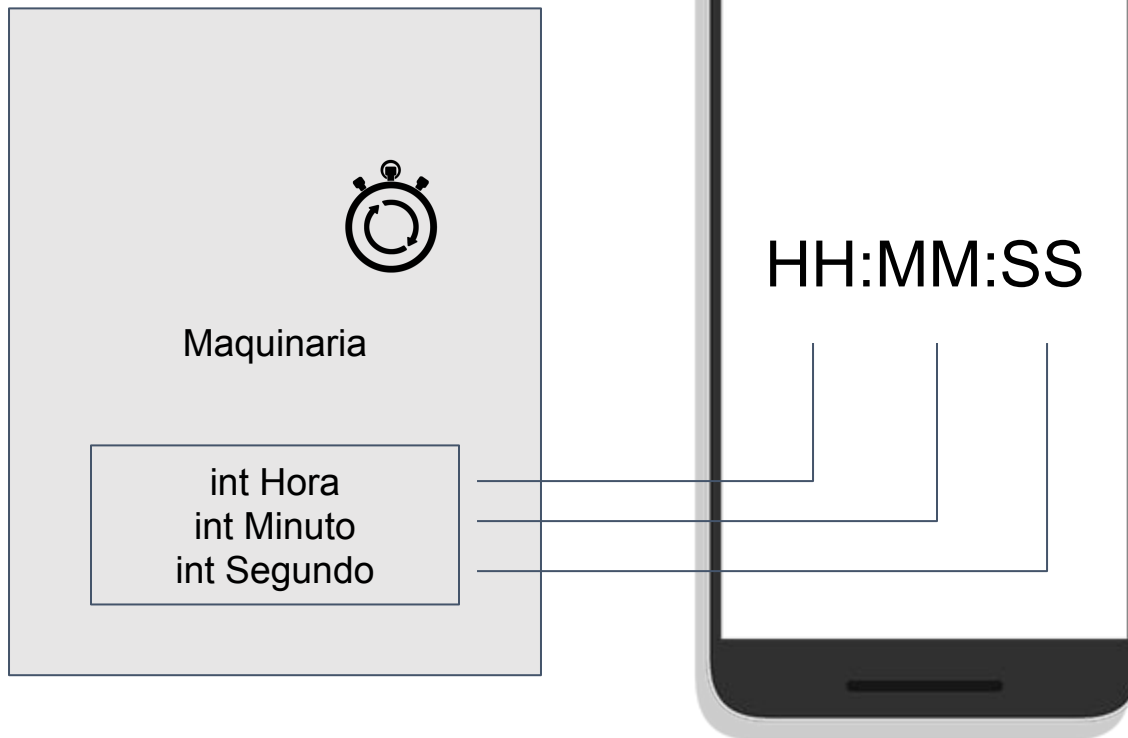

Contexto de Enlace de Datos



Contexto de Enlace de Datos



Reto 3



Enlace entre elementos

Podemos usar la Extensión de Marcado { x:Reference } como la fuente del enlace

Indica el nombre del control al que deseas enlazarte

Enlace entre elementos

```
<Label  
Text="{Binding Propiedad,  
    Source={x:Reference miControl} } />
```

Propiedad **StringFormat**

Establece un formato para
presentar correctamente
las cadenas

Propiedad StringFormat

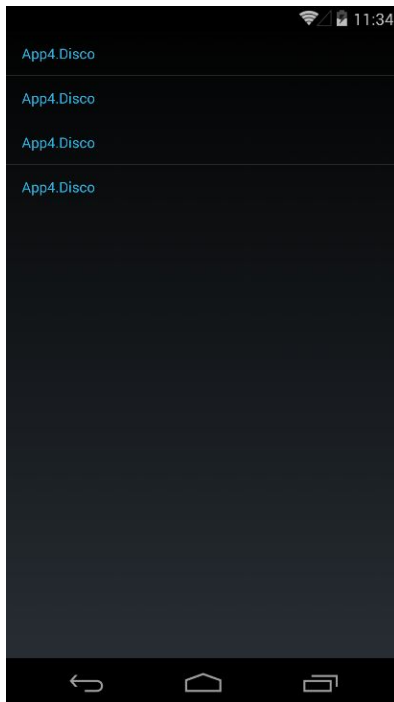
```
<Label  
Text="{Binding Precio, StringFormat='{{0:C}}'}" />
```

```
<Label  
Text="{Binding Fecha,  
StringFormat='{{0:dd/MM/yyyy}}'}" />
```

Plantillas de Datos

Expresan el XAML que representa un elemento en un control de contenido y/o control de lista

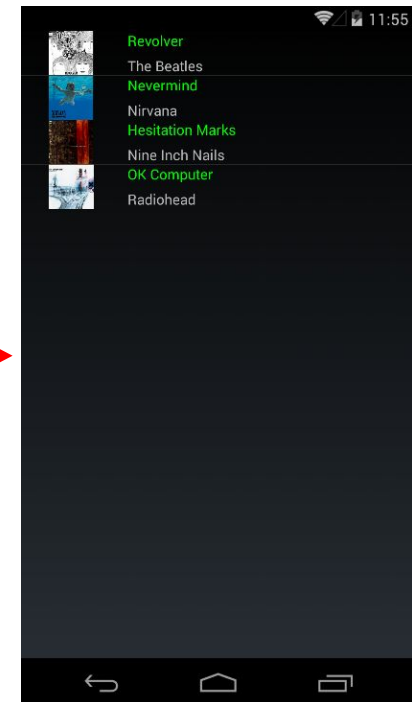
Plantillas de Datos



```
<DataTemplate>
  <ViewCell>
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition />
      </Grid.ColumnDefinitions>

      <Image Source="{Binding Portada}"
        WidthRequest="100" />

      <StackLayout Grid.Column="1">
        <Label Text="{Binding Titulo}"
          TextColor="Lime" />
        <Label Text="{Binding Grupo}" />
      </StackLayout>
    </Grid>
  </ViewCell>
</DataTemplate>
```



Plantillas de Datos

- Se expresan con el elemento DataTemplate
 - En la propiedad ItemTemplate de los controles de lista, en línea o referenciando un Recurso

```
<ListView ItemsSource="{Binding Discos}">
  <ListView.ItemTemplate>
    <DataTemplate>
      <ViewCell>
        ...
      </ViewCell>
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

```
<ListView ItemsSource="{Binding Discos}"
  ItemTemplate="{StaticResource DiscosDataTemplate}"
```

Plantillas de Datos

- La raíz debe ser cualquier tipo de Celda
 - EntryCell
 - ImageCell
 - SwitchCell
 - TextCell
 - ViewCell

Plantillas de Datos

- Si están declaradas en los Recursos, pueden ser reutilizadas entre diferentes controles de lista

```
<ListView ItemTemplate="{StaticResource  
AlbumsDataTemplate}" />
```

```
<Picker ItemTemplate="{StaticResource  
AlbumsDataTemplate}" />
```

Convertidores de Valor

Clases que implementan la interfaz
IValueConverter

Incluye dos métodos a implementar:

Convert

Método que se ejecuta cuando el valor fluye
del Objeto fuente hacia el Control destino

ConvertBack

Método que se ejecuta cuando el valor fluye
de regreso, del Control destino hacia el
Objeto fuente

Convertidores de Valor

- Ejemplo:

```
public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
{
    int entrada = (int)value;
    Color resultado = Color.Green;

    if (entrada < 50)
    {
        resultado = Color.Red;
    }

    return resultado;
}
```

Convertidores de Valor

- Se utilizan declarándolos en un Diccionario de Recursos e invocándolos a través del parámetro Converter

```
<ContentPage.Resources>
  <ResourceDictionary>
    <local:InventarioConverter
x:Key="InventarioConverter" />
  </ResourceDictionary>
</ContentPage.Resources>
```

```
<Label Text="{Binding Banda}"
  TextColor="{Binding Inventario,
    Converter={StaticResource
InventarioConverter}}"/> />
```


Convertidores de Valor

- De manera opcional, pueden recibir parámetros
 - Se establecen a través del atributo ConverterParameter de la clase base Binding

```
<Label Text="{Binding Banda}"  
    TextColor="{Binding Inventario,  
Converter={StaticResource InventarioConverter},  
    ConverterParameter='255,255,0'}" />
```

Comandos

Comandos

Son «funcionalidad enlazable»

Clases que implementan la interfaz
ICommand

Miembros:

Execute: Indica qué acción se realizará

CanExecute: Determina si el Comando puede ejecutar

CanExecuteChanged: Re-evalúa el CanExecute

Comandos

- Están soportados en algunos controles y gestos
 - Button
 - ToolbarItem
 - SearchBar
 - TapGestureRecognizer
 - ...

Comandos

- Utiliza los atributos **Command** y **CommandParameter** para enlazar

```
<Button Text="Consultar..."  
    Command="{Binding GetDiscoCommand}"  
    CommandParameter="{Binding Text,  
        Source={x:Reference entry1}}}" />
```

Comandos

```
public class ConsultaCommand : ICommand {  
    Action action;  
  
    public ConsultaCommand(Action action) {  
        this.action = action;  
    }  
  
    public bool CanExecute(object parameter) {  
        return true;  
    }  
  
    public event EventHandler CanExecuteChanged;  
  
    public void Execute(object parameter) {  
        action();  
    }  
}
```

```
<Button Command="{Binding ConsultaCommand}"... />
```

Comandos

- Implementaciones existentes recomendadas
 - Prism (clase DelegateCommand)
 - Xamarin.Forms (clase Command)
 - MVVM Light (clase RelayCommand)

El Patrón de Diseño Model-View-ViewModel (MVVM)

¿Qué es MVVM?

Patrón de Diseño natural para plataformas de XAML

Aprovecha al máximo el Enlace de Datos

Ventajas

Separación de preocupaciones

Pruebas Unitarias

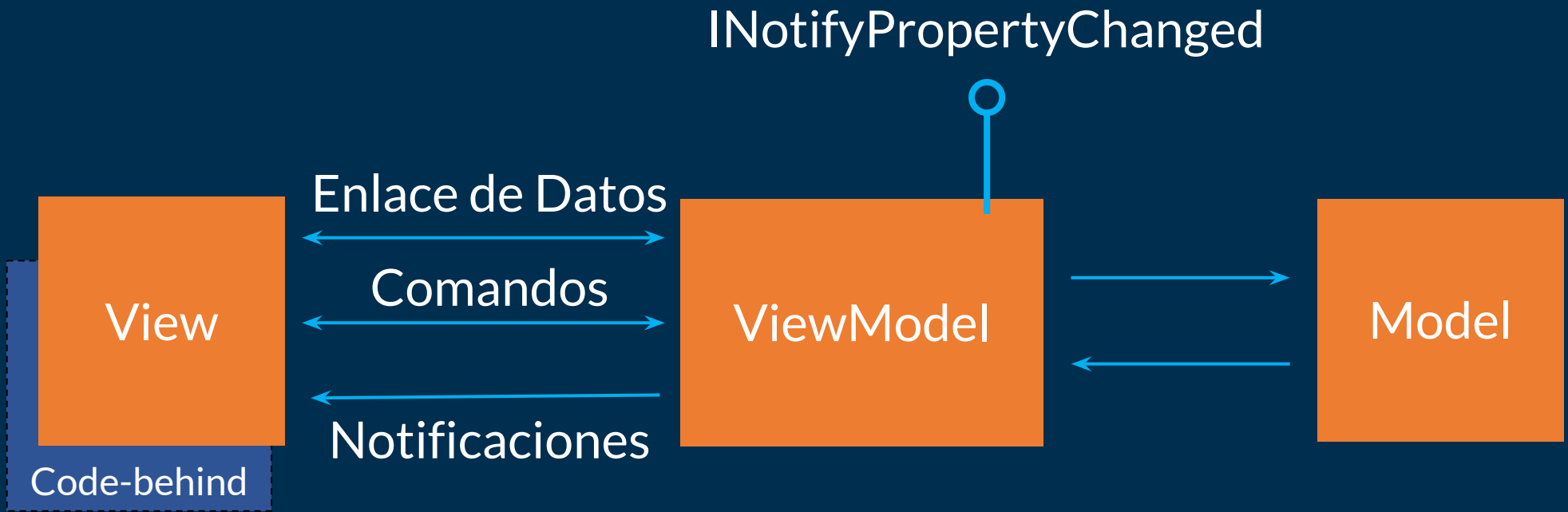
Mantenimiento de código

Consistencia

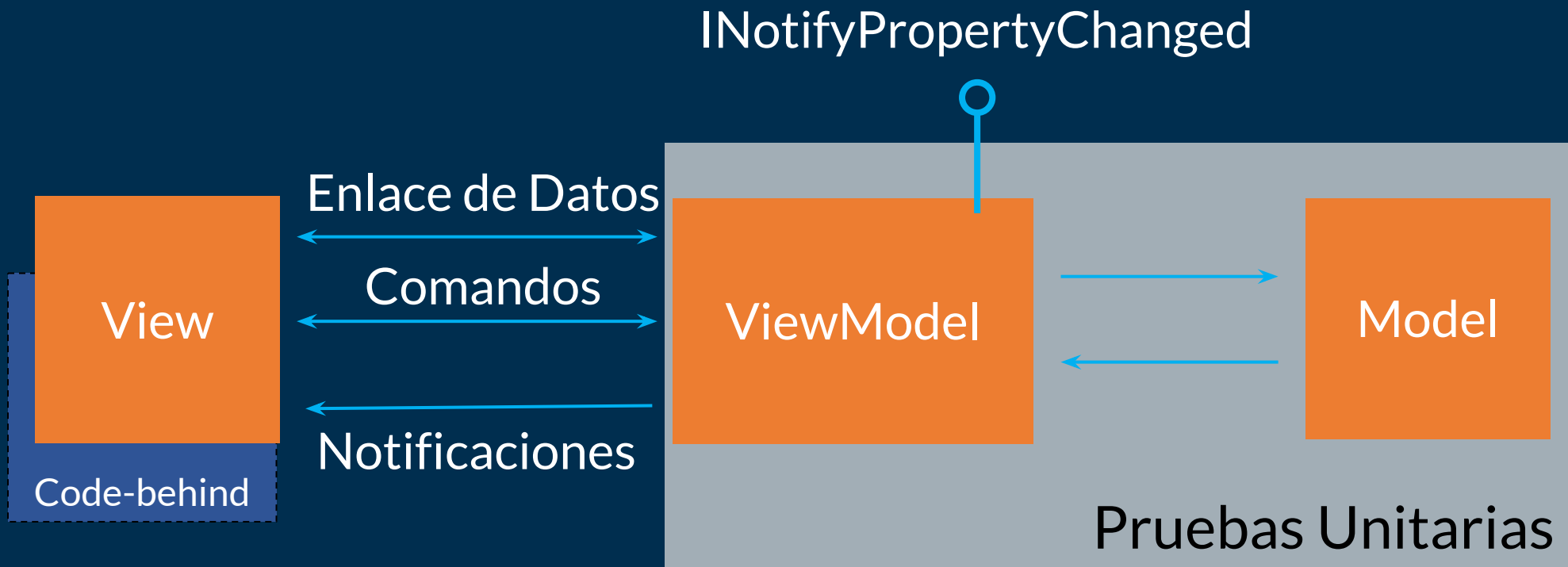
Desacoplamiento

Reutilización de código

¿Qué es MVVM?



¿Qué es MVVM?



View



- Define la Interfaz de Usuario
- Estilos, Recursos
- Páginas, Plantillas de Datos, etc.
- El Contexto de Enlace es el ViewModel
- Poco o nada de code-behind
- Solo código que no necesite Pruebas Unitarias
- Actualizada a través de Enlace de Datos



ViewModel

- Es una abstracción de la Vista
- Implementa la Lógica de Presentación
- Adapta el Modelo a la Vista
- Mantiene el estado
- Expone propiedades a las que se enlaza la Vista (datos y Comandos)



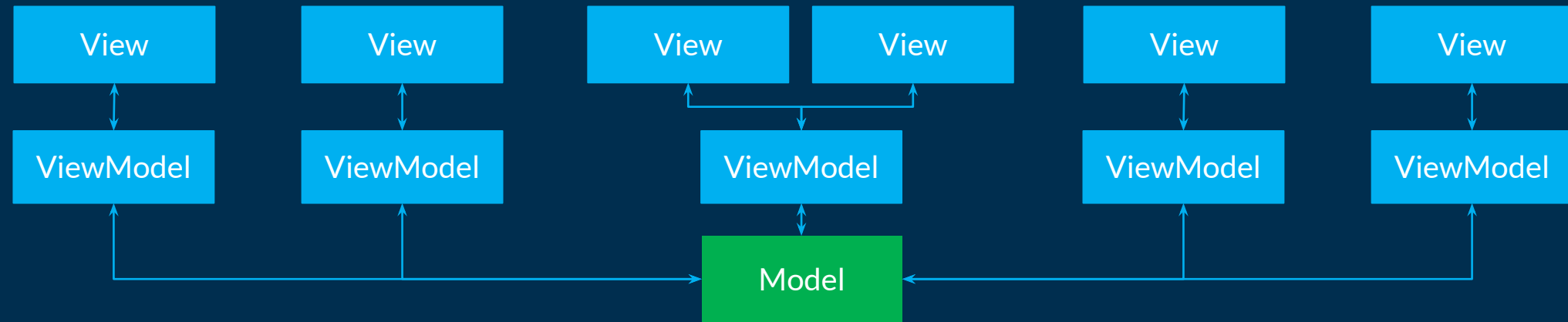
Model

- Tu dominio
- Objetos de datos
 - DTO, POJO
 - Modelo de datos generado
 - Modelo de proxy generado
- Capa de Servicios
 - Repositorios
 - Lógica de negocio relacionada a la consulta y administración de los datos de la aplicación

Cardinalidad

- Generalmente, una Vista tiene un ViewModel
- Un ViewModel puede ser usado en una o más Vistas
 - Aunque generalmente se recomienda 1:1
- Un Model puede ser usado en uno o más ViewModels

Cardinalidad





¡Gracias!