Introduction to Xamarin.Forms

Enterprise & Mobile .Net

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Agenda

- Introduction to Xamarin Forms
  - Project structure
- UI Building blocks
  - Pages
  - Layouts
  - Views
  - Cells
- Navigation
- Shell
- Platform features

PXL IT

PXL IT

Introduction to Xamarin Forms

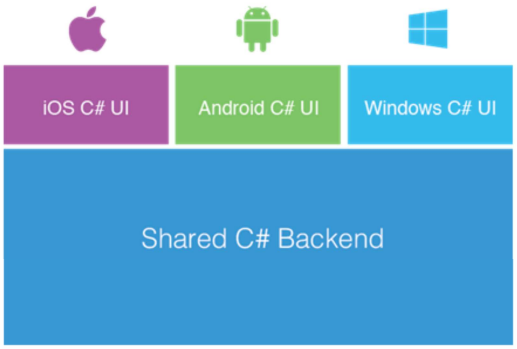
DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Xamarin Forms is a new **set of APIs** allowing you to quickly and easily **write shared User Interface code** that is still **rendered natively** on each platform, while still providing direct access to the underlying SDKs if you need it.

So far with regular Xamarin

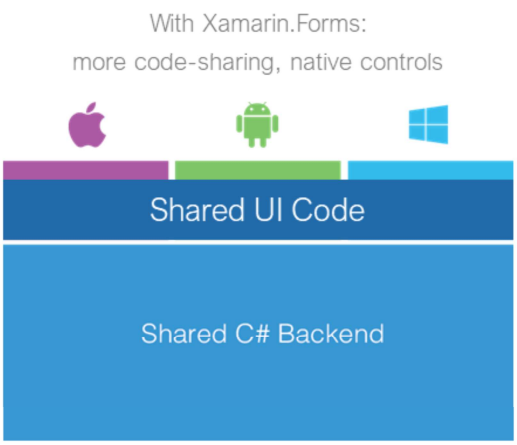
- UI is platform specific, tuned to the platform
- App logic is shared
  - .NET Standard libs
  - Shared Projects
- 70-90% shared code on average



PXL IT

With Xamarin.Forms

- With Xamarin.Forms, we get an API which allows to build sharable user interface code
- Still renders native UI on IOS, Android and Windows Phone
- UI + logic written in C#, allowing 90-99% code sharing



PXL IT

So what is Xamarin.Forms really?

- *Xamarin.Forms is a cross-platform natively backed UI toolkit abstraction that allows developers to easily create user interfaces that can be shared across Android, iOS, UWP and more*
  - Allows rapid development of cross-platform Uis
  - Abstraction on top of UI elements
  - More code sharing than ever before
  - Still native apps
  - Still native performance and look and feel

Xamarin.Forms is a cross-platform natively backed UI toolkit abstraction that allows developers to easily **create user interfaces** that can be **shared across Android, iOS, UWP**.

The user interfaces are **rendered using** the **native controls** of the target platform, allowing Xamarin.Forms applications to retain the appropriate look and feel for each platform.

This guide will provide a quick introduction to Xamarin.Forms and how to get started writing applications with it.

Xamarin.Forms is a framework that allows developers to **rapidly create cross platform user interfaces**. It provides it's **own abstraction for the user interface** that will be rendered **using native controls** on iOS, Android or UWP.

This means that applications can **share a large portion of their user interface code** and still **retain the native look and feel** of the target platform.

So what is Xamarin.Forms really?

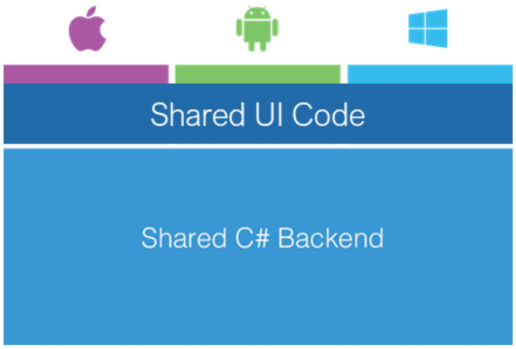
- All Forms apps are written in C#
- RAD but can evolve to more complex apps
  - API they can access is the same as “plain Xamarin”
- Commonly built on top of .NET Standard or shared project layers
  - Consumed by Xamarin.Forms code
- UIs can be built with XAML (with code behind) or fully in code

PXL IT

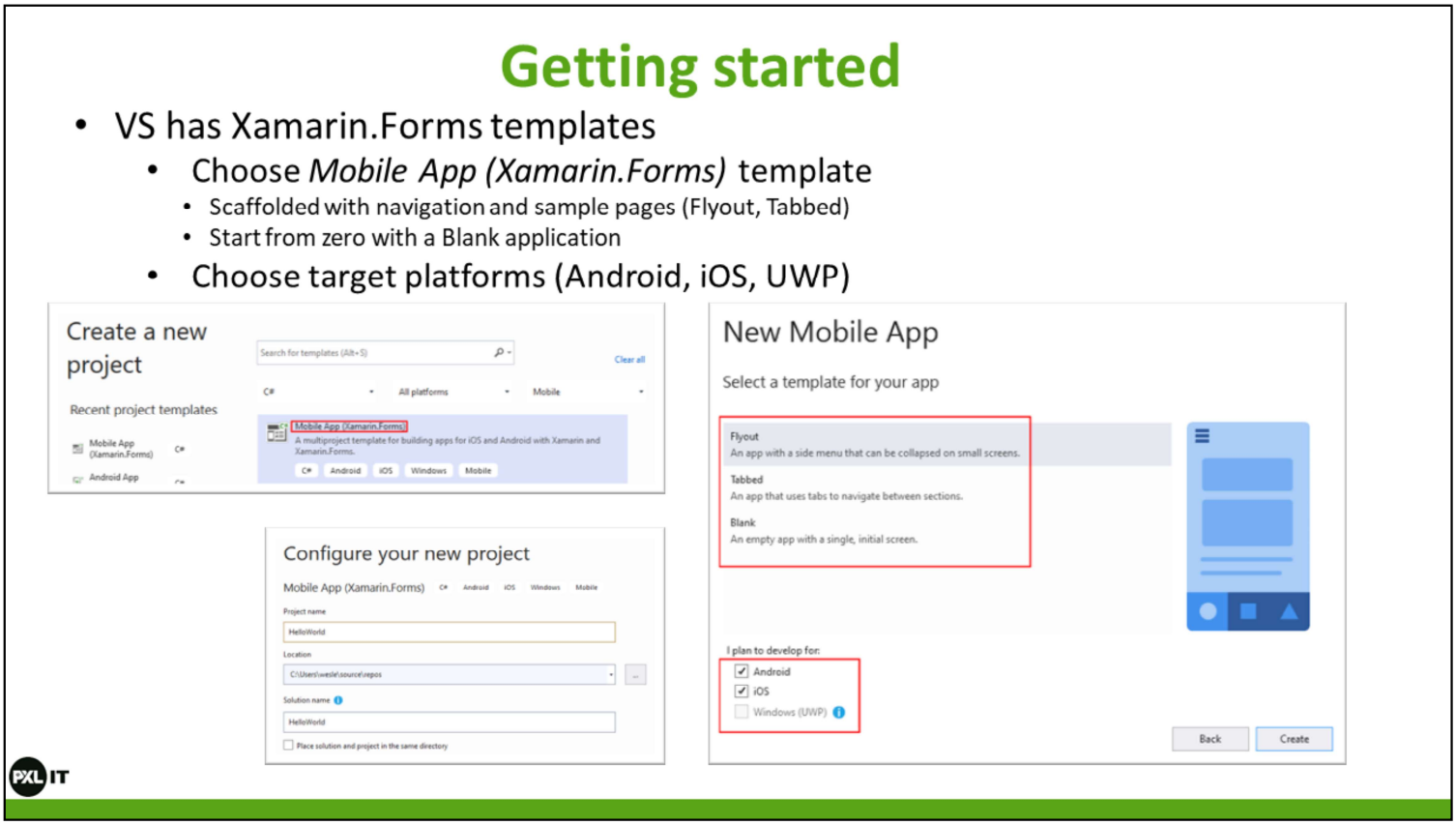
RAD = Rapid Application Development

Main features

- RAD, forms-based app creation
- 40+ Pages, Layouts, and Controls
  - Build from code behind or XAML
- Two-way Data Binding
- Navigation
- Animation API
- Dependency Service
- Messaging Center
- Support for
  - Android 4.0+
  - iOS 6.1+
  - UWP



PXL IT



Xamarin.Forms is implemented as a .NET Standard Library, which makes it very easy to share the Xamarin.Forms API's across a variety of platforms.

The first step to getting started is to create a solution for the various projects that will make up the application.

After choosing the *Mobile App (Xamarin.Forms)* project template and specifying the solution and project name, you can choose between 3 templates:

- Flyout: the solution will be scaffolded with an app that has a side menu that can be collapsed on small screens
- Tabbed: the solution will be scaffolded with an app that uses tabs to navigate between pages
- Blank: the solution will be scaffolded with a blank app without navigation or pages

You must also select the platforms you wish to target.

Note: if you want to target UWP you must start from a blank app. This is because the *Flyout* and *Tabbed* templates make use of the *Shell* component which is not supported in UWP (yet).

A Cross Platform app (Xamarin) will be created in Visual Studio and will typically contain the following projects:

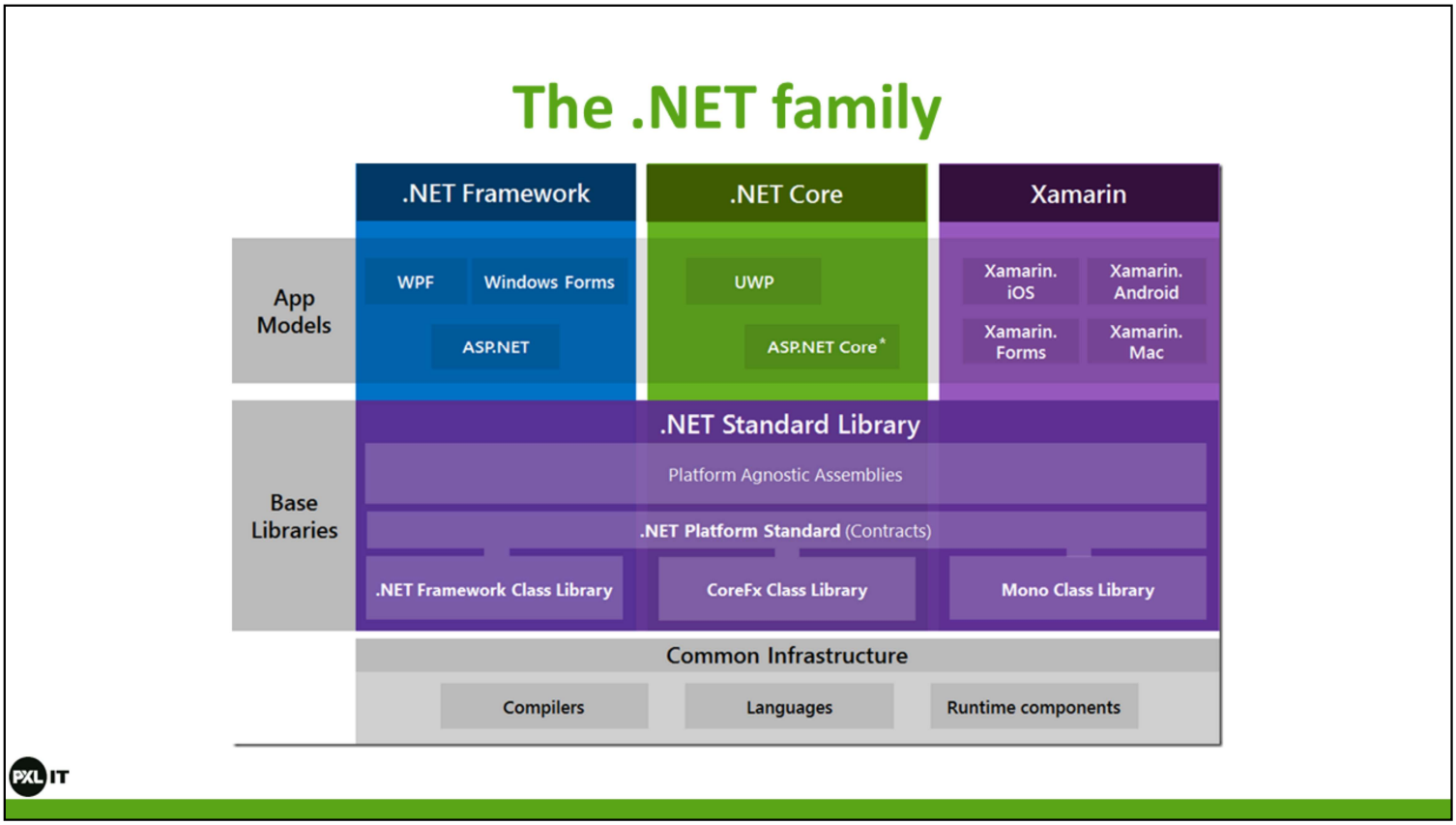
- **.NET Standard project** - This project is the cross platform application library that holds all of the shared code and shared UI.
- A separate project for each targeted platform:
    - **Xamarin.Android Application** - This project holds Android specific code and is the entry point for Android applications.
    - **Xamarin.iOS Application** - This project holds iOS specific code and is the entry point for iOS applications.
    - **UWP Application** - This project holds the UWP specific code and is the entry point for Windows applications.

Getting started

- Resulting solution:

- HelloWorld Project holds shared code, used by all other projects (.NET Standard)
- iOS, Android and UWP projects contain platform-specific UI code

PXL IT



Traditionally, there was the ".NET Framework":

- You build desktop apps (WPF or Winforms) and web apps (ASP.NET)
- On Windows machines targeting windows machines
- Using the BCL (Base Class Library)

Then there was Mono, an open source implementation of .NET running on Linux and Mac.

From this foundation, Xamarin was build. The "Mono Class Library" is a port of the BCL to the mono runtime. Not everything exists there (like WPF and Winforms).

Then ".NET Core" was released, yet again a new beginning:

- A .NET runtime running on all major platforms
- Another framework library implementation
- A set of tools (compilers and application host)

.NET Standard

- Formal specification of .NET API's intented to be be available on all .NET implementations
- (.NET Standard - .NET implementation) ~ (interface – class)
- https://docs.microsoft.com/en-us/dotnet/standard/net-standard

.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0	2.1
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	3.0
.NET Framework [1]	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1 [2]	4.6.1 [2]	4.6.1 [2]	N/A [3]
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4	6.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14	12.16
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8	5.16
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0	10.0
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299	TBD

PXL IT

The .NET Standard is a formal specification of .NET APIs that are intended to be available on all .NET implementations.

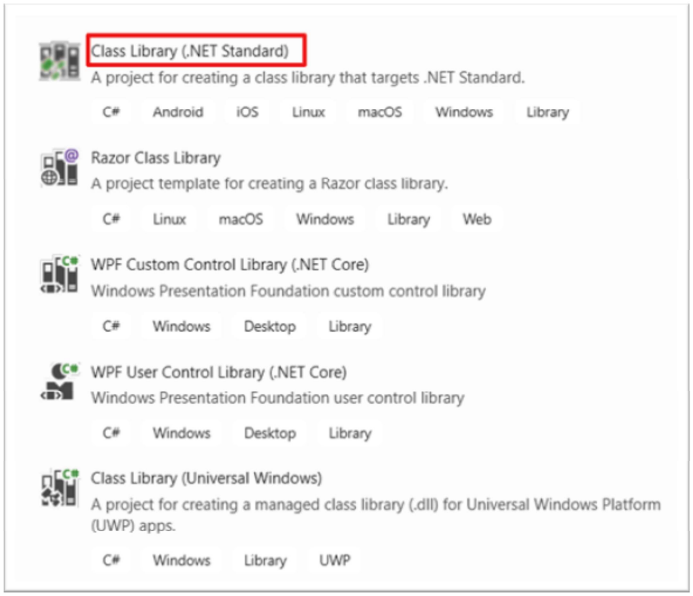
The various .NET implementations target specific versions of .NET Standard.

Each .NET implementation version advertises the highest .NET Standard version it supports, a statement that means it also supports previous versions.

One library to rule them all, where you don't have to worry about platform incompatibilities!

https://docs.microsoft.com/en-us/dotnet/standard/net-standard

Working with class libraries

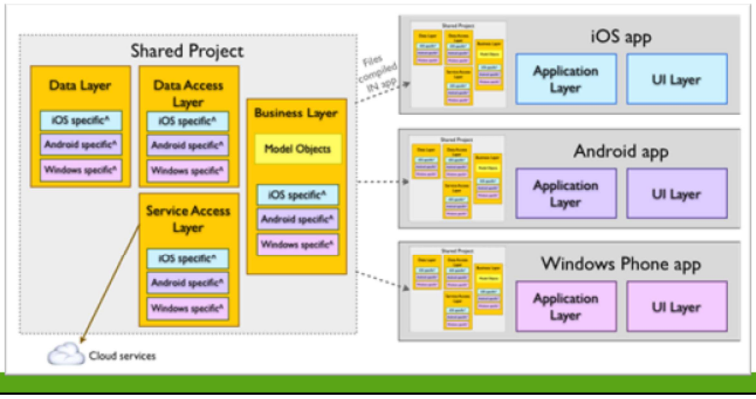


PXL IT

If you work with extra class libraries, choose .NET Standard. This will provide maximum portability across all platforms.

Shared Project

- A project with common code that is referenced by a number of different application projects
- The code is compiled as part of each referencing project
- Conditional compilation makes is possible to differentiate between platforms
  - **Ugly code**
  - **Not recommended**



https://docs.microsoft.com/nl-nl/xamarin/cross-platform/app-fundamentals/shared-projects

Creating our first Xamarin.Forms app

DEMO

Create new app, and run => HelloXamarin

Depending on the IDE (Visual Studio) and the development OS (Windows or Mac) there will be the possibility to create Android, iOS and/or UWP projects.

Previewing of the XAML UI is possible:
https://docs.microsoft.com/xamarin/xamarin-forms/xaml/xaml-previewer

Hot reloading while debugging is possible:
https://docs.microsoft.com/nl-nl/xamarin/xamarin-forms/xaml/hot-reload

Sometimes rebuilding takes forever => delete all bin/obj folders and rebuild solution

Final gotcha: watch out for long pathnames (I put my projects in C:\demos and NOT C:\users\johndoe\projects)

App plays an important role

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace HelloWorldForms
{
    7 references
    public partial class App : Application
    {
        3 references
        public App()
        {
            InitializeComponent();

            MainPage = new MainPage();
        }
```

The App class is the starting point.

For a *Blank Template* it sets its *MainPage* property to an instance of a *ContentPage* (→ MainPage class)

Android - MainActivity

- Create an activity with MainLauncher to true
- Activity must inherit from Xamarin.Forms.Platform.Android.FormsAppCompatActivity
- Must initialize Xamarin.Forms
- Display Page in OnCreate

```
[Activity(Label = "HelloWorldXamarinForms", Icon = "@mipmap/icon", Theme = "@style/MainTheme",
    MainLauncher = true,
    ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsAppCompatActivity
{
    protected override void OnCreate(Bundle savedInstanceState)
    {
        TabLayoutResource = Resource.Layout.Tabbar;
        ToolbarResource = Resource.Layout.Toolbar;

        base.OnCreate(savedInstanceState);
        global::Xamarin.Forms.Forms.Init(this, savedInstanceState);
        LoadApplication(new App());
    }
}
```

PXL IT

The MainActivity in Androids initializes the Xamarin.Forms subsystem and loads the App object.

iOS - AppDelegate

- AppDelegate must initialize Xamarin.Forms
  - Call LoadApplication, passing in App instance
  - All in FinishedLaunching

```
[Register(name: "AppDelegate")]
0 references
public partial class AppDelegate : global::Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
{
    0 references
    public override bool FinishedLaunching(UIApplication app, NSDictionary options)
    {
        global::Xamarin.Forms.Forms.Init();
        LoadApplication(new App());

        return base.FinishedLaunching(app, options);
    }
}
```

PXL IT

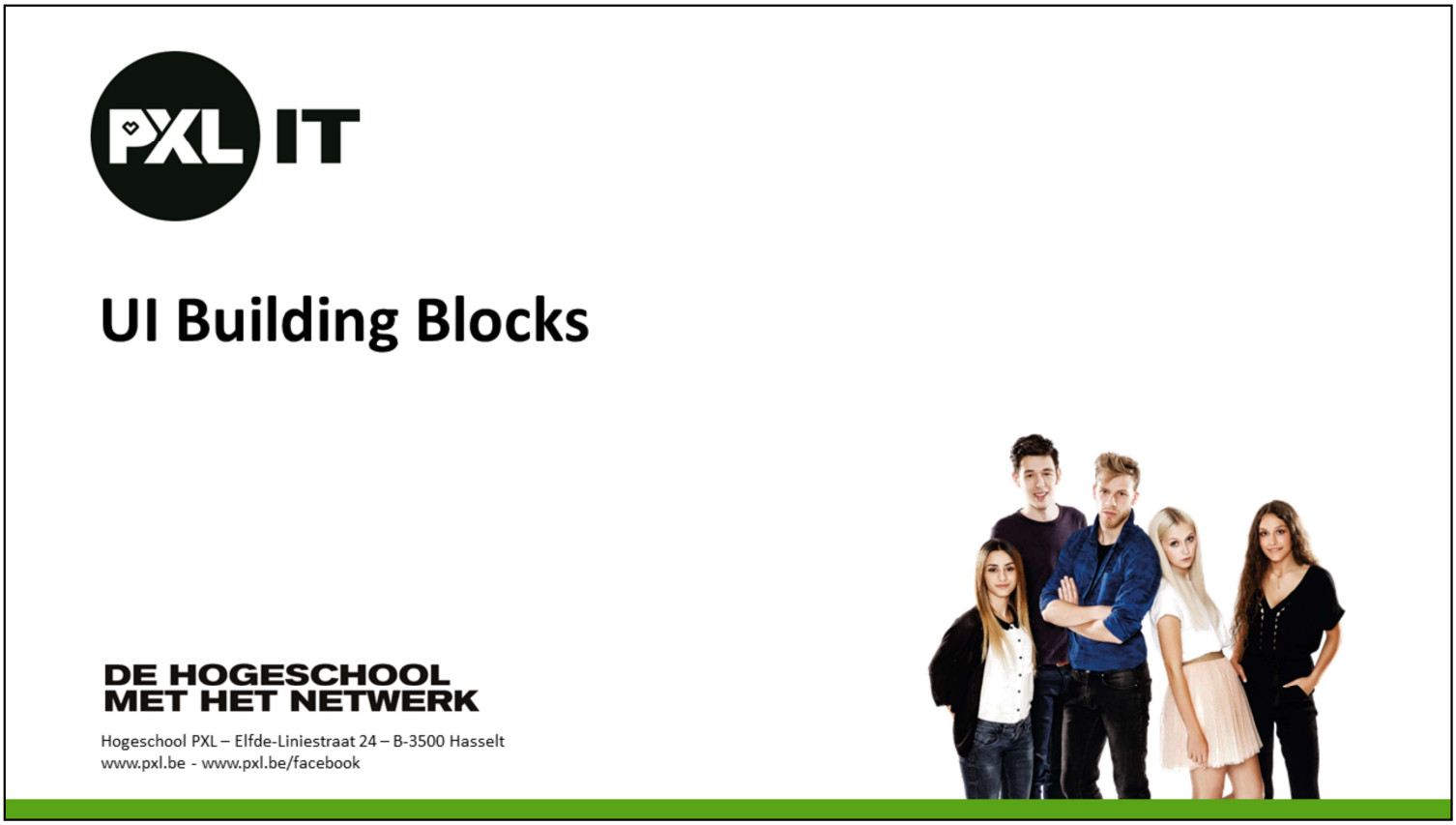
As with the Android application, the first step in the FinishedLaunching event is to initialize the Xamarin.Forms framework with a call to Xamarin.Forms.Forms.Init().

This step causes the iOS specific implementation of Xamarin.Forms to be globally loaded in the application.

And let's see that again…

DO IT YOURSELF

Build this demo yourself:
https://docs.microsoft.com/xamarin/get-started/quickstarts/single-page



Reminder: Views are Controls

Important building blocks in Xamarin.Forms

- 4 main classes are used for composing the Xamarin.Forms UI
  - **Page**
    - Xamarin.Forms represents a single screen in your application
      - Analogous to an Android Activity, an UWP Page, or an iOS View Controller
  - **Layout**
    - Specialized View subtype
      - Meant to act as a container for other Layout or Views
      - Layout subtypes typically contain logic that is specific to organizing the child views in a certain way
  - **View**
    - controls or widgets in other platforms
      - UI elements such as labels, buttons, text fields, etc
  - **Cell**
    - Specialized element that is used for items in a list or a table
    - It describes how each item in a list should be drawn

PXL IT

Reference:

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/controls

PXL IT

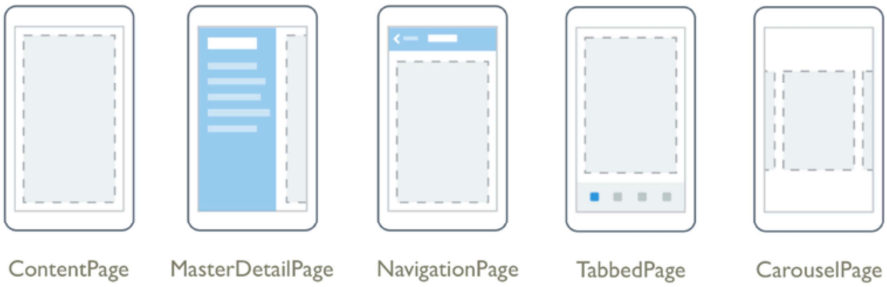
UI Building Blocks: Pages

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

It all starts with a Page

- Page is a visual element that occupies most or all of the screen and contains one or more view objects (layout, view, other pages)
  - Activity in Android
  - View Controller in iOS
  - Page in UWP
- Available pages
  - ContentPage
  - MasterDetailPage
  - NavigationPage
  - TabbedPage
  - CarouselPage

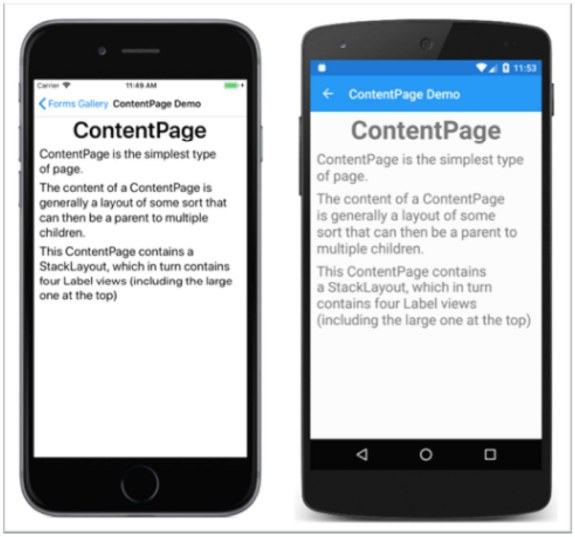


PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/controls/pages

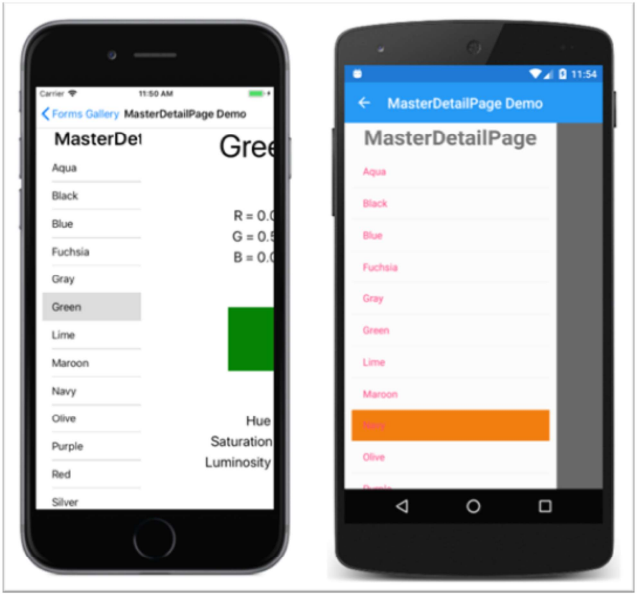
Pages: ContentPage

- A Content Page displays a single View
  - Commonly a StackLayout or a ScrollView



Pages: Master Detail page

- A Page that manages two panes of information



Pages: NavigationPage

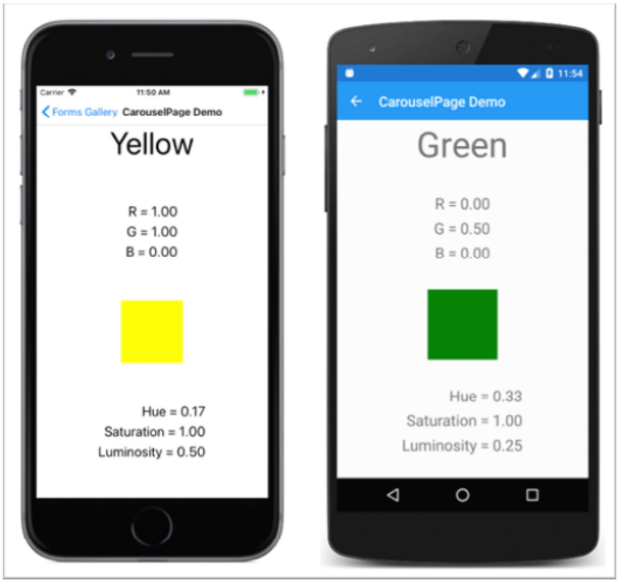
- A Page that manages the navigation and user-experience of a stack of other pages

Pages: TabbedPage

- A Page that allows navigation between children pages, using tabs

Pages: CarouselPage

- A Page allowing swipe gestures between subpages, like a gallery



PXL IT

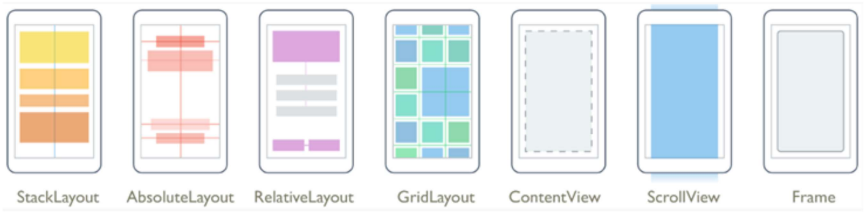
UI Building Blocks: Layouts

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

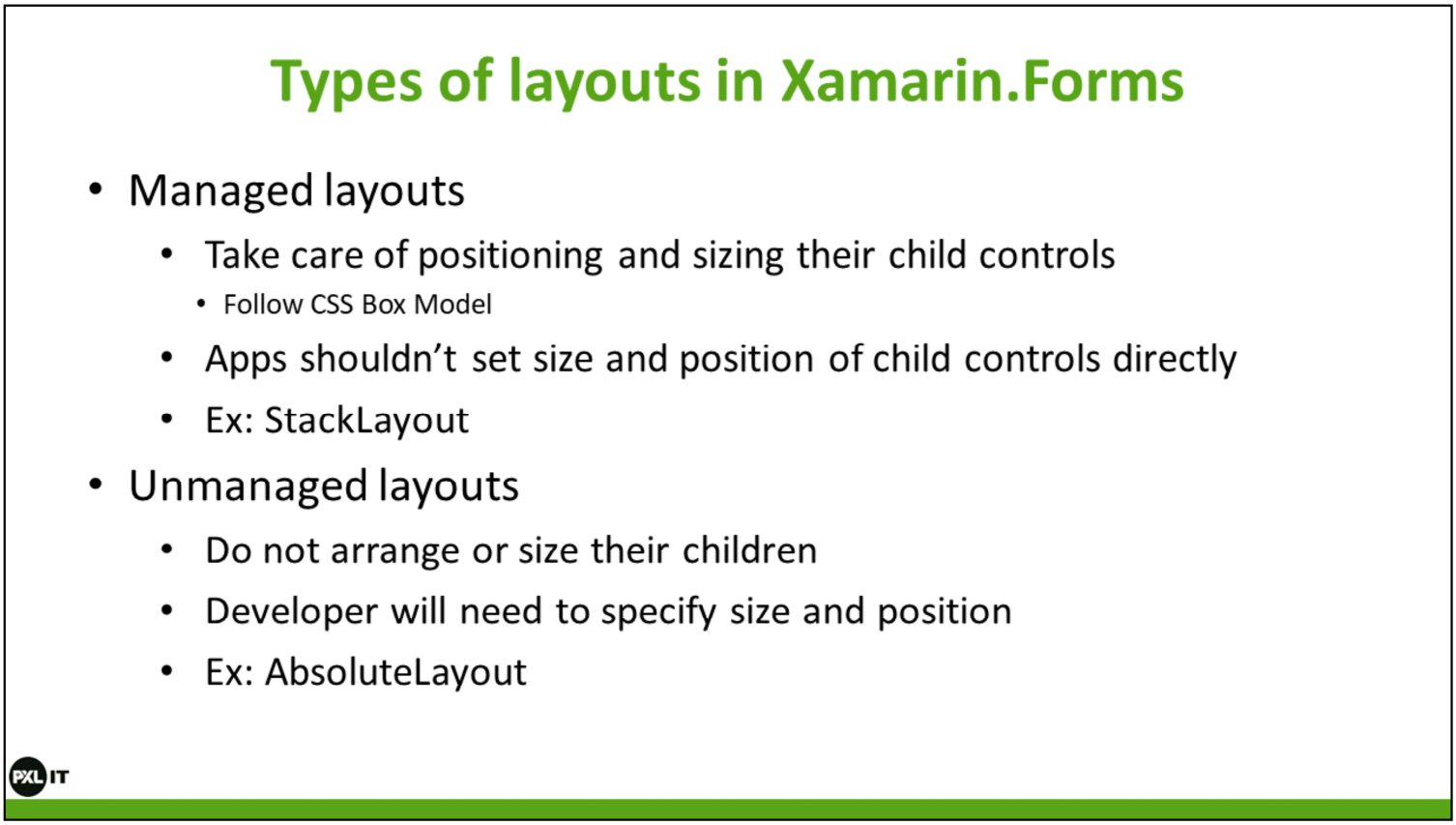
Layouts in Xamarin.Forms

- Layouts are subtypes of views that act as **containers** for views and other layouts
  - Layouts with single content
    - ContentView
    - Frame
    - ScrollView
  - Layouts with multiple children
    - StackLayout
    - Grid
    - AbsoluteLayout
    - RelativeLayout
    - FlexLayout



PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/controls/layouts
https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/layouts



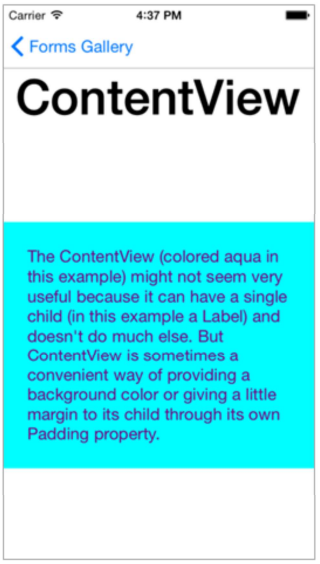
Controls themselves are hosted inside of a layout.

Xamarin.Forms has two different categories of layouts that arrange the controls in very different ways:

- **Managed Layouts** - these are layouts that will take care of positioning and sizing child controls on the screen and follow the CSS Box Model. Applications should not attempt to directly set the size or position of child controls.
  One common example of a managed Xamarin.Forms layout is the *StackLayout*.
  CSS Box Model reference: http://www.w3schools.com/css/css_boxmodel.asp
- **Unmanaged Layouts** - as opposed to managed layouts, unmanaged layouts will not arrange or position their children on the screen.
  Typically, the user will specify the size and location of the child control as it is being added to the layout.
  The *AbsoluteLayout* is an example of an unmanaged layout control.

Layout: ContentView

- An element with a single content
  - ContentView has very little use of its own
    - Base class for user-defined compound views

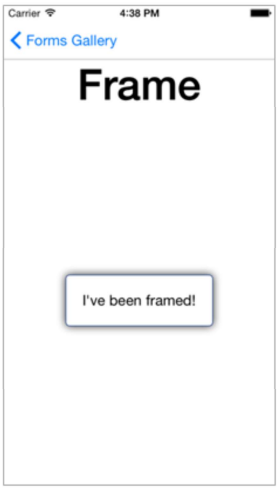


PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/layouts/contentview

Layouts: Frame

- An element containing a single child used to wrap a view with a border (with color, shadow, corner radius)



https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/layouts/frame

Layouts: ScrollView

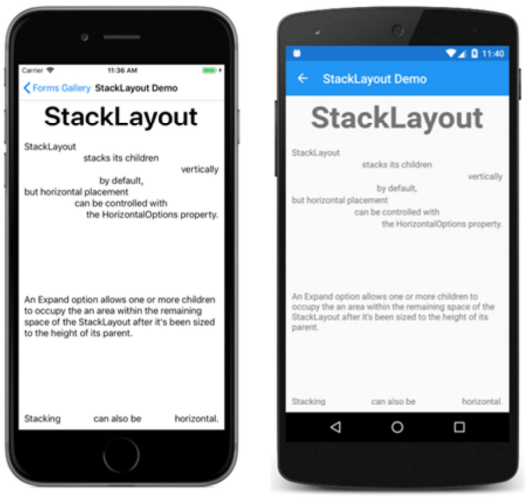
- An element capable of scrolling if its Content is larger

PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/layouts/scroll-view

StackLayout

- Very common managed layout
  - Arranges controls automatically, no matter what the screen size
  - Positioned horizontally or vertically next to each other in order they were added
  - Spacing is determined by HorizontalOptions and VerticalOptions
    - Default is using entire screen



PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/layouts/stack-layout

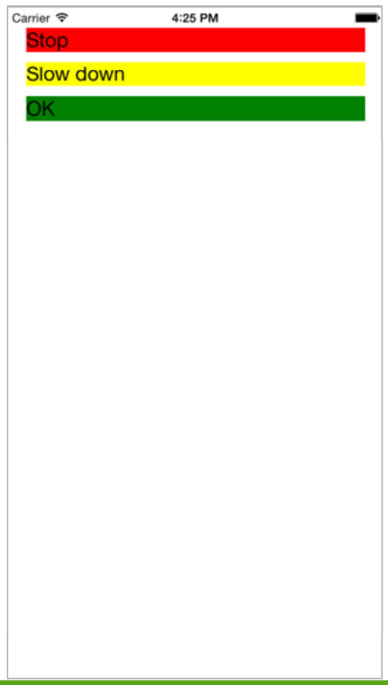
In code

```
public class StackLayoutExample: ContentPage
{
    public StackLayoutExample()
    {
        Padding = new Thickness(20);
        var red = new Label
        {
            Text = "Stop",
            BackgroundColor = Color.Red,
            Font = Font.SystemFontOfSize (20)
        };
        var yellow = new Label
        {
            Text = "Slow down",
            BackgroundColor = Color.Yellow,
            Font = Font.SystemFontOfSize (20)
        };

        Content = new StackLayout
        {
            Spacing = 10,
            Children = { red, yellow}
        };
    }
}
```

In XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             x:Class="HelloXamarinFormsWorldXaml.StackLayoutExample1"
             Padding="20">

  <StackLayout Spacing="10">

    <Label Text="Stop"
           BackgroundColor="Red"
           Font="20" />

    <Label Text="Slow down"
           BackgroundColor="Yellow"
           Font="20" />

    <Label Text="Go"
           BackgroundColor="Green"
           Font="20" />

  </StackLayout>
</ContentPage>
```

The result

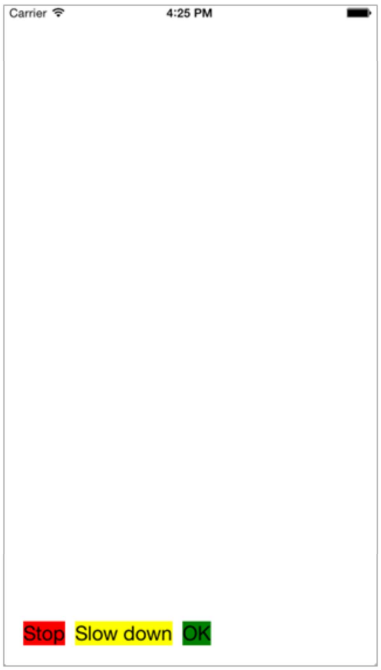


Different orientation

```
public class StackLayoutExample: ContentPage
{
    public StackLayoutExample()
    {
        // Code that creates labels removed for clarity

        Content = new StackLayout
        {
            Spacing = 10,
            VerticalOptions = LayoutOptions.End,
            Orientation = StackOrientation.Horizontal,
            HorizontalOptions = LayoutOptions.Start,
            Children = { red, yellow, green }
        };
    }
}
```

PXL IT

The result



Setting width and height

- Setting size isn't possible, however, we can specify WidthRequest and HeightRequest

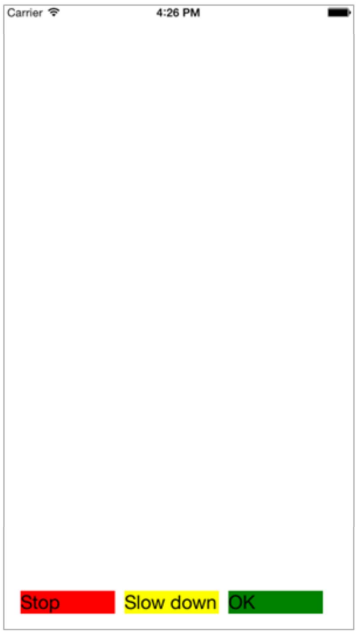
```
var red = new Label
{
    Text = "Stop",
    BackgroundColor = Color.Red,
    Font = Font.SystemFontOfSize (20),
    WidthRequest = 100
};
var yellow = new Label
{
    Text = "Slow down",
    BackgroundColor = Color.Yellow,
    Font = Font.SystemFontOfSize (20),
    WidthRequest = 100
};
var green = new Label
{
    Text = "Go",
    BackgroundColor = Color.Green,
    Font = Font.SystemFontOfSize (20),
    WidthRequest = 100
};
```

PXL IT

https://docs.microsoft.com/dotnet/api/xamarin.forms.visualelement.widthrequest

https://docs.microsoft.com/dotnet/api/Xamarin.Forms.VisualElement.HeightRequest

The result



Managing spacing

- VerticalOptions/HorizontalOptions
  - Sets how the child content is stretched or positioned
  - Used mostly on containers
- Spacing
  - Spacing added between child elements
  - Used on StackLayout
- Padding
  - Padding around the element

PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/layouts/layout-options

https://docs.microsoft.com/dotnet/api/Xamarin.Forms.StackLayout.Spacing

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/layouts/margin-and-padding

Managing width and height

- WidthRequest/HeightRequest
  - Request a specific width/height
  - Overrides the measured width and height
- MinimumWidthRequest/MinimumHeightRequest
  - Absolute minimum for the control
- Width/Height
  - Read-only
  - Final calculated width and height
- Bounds
  - Read-only
  - Position and size of the frame (of the control) relative to its parent

PXL IT

https://docs.microsoft.com/dotnet/api/xamarin.forms.visualelement.widthrequest

https://docs.microsoft.com/dotnet/api/Xamarin.Forms.VisualElement.HeightRequest

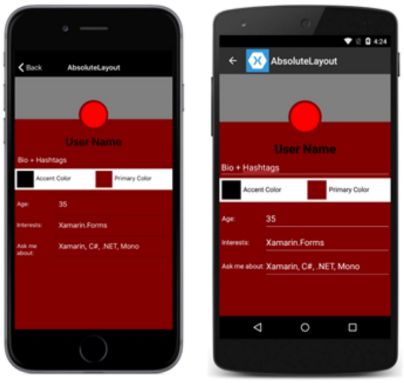
https://docs.microsoft.com/dotnet/api/xamarin.forms.visualelement.width

https://docs.microsoft.com/dotnet/api/xamarin.forms.visualelement.height

https://docs.microsoft.com/dotnet/api/Xamarin.Forms.VisualElement.Bounds

AbsoluteLayout

- Unmanaged container: all controls must be explicitly get a position within the layout
  - Similar to iOS AbsoluteLayout or WinForms
- Very precise, more difficult on multiple resolutions



PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/layouts/absolute-layout

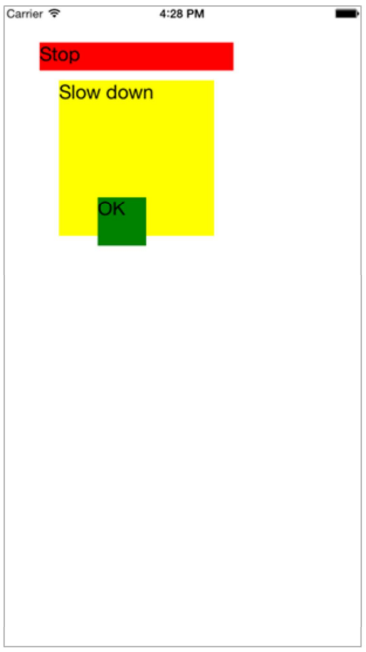
```
public class MyAbsoluteLayoutPage : ContentPage
{
    public MyAbsoluteLayoutPage()
    {
        var red = new Label
        {
            Text = "Stop",
            BackgroundColor = Color.Red,
            Font = Font.SystemFontOfSize (20),
            WidthRequest = 200,
            HeightRequest = 30
        };
        var yellow = new Label
        {
            Text = "Slow down",
            BackgroundColor = Color.Yellow,
            Font = Font.SystemFontOfSize (20),
            WidthRequest = 160,
            HeightRequest = 160
        };
        var absLayout = new AbsoluteLayout();
        absLayout.Children.Add(red, new Point(20,20));
        absLayout.Children.Add(yellow, new Point(40,60));
        absLayout.Children.Add(green, new Point(80,180));

        Content = absLayout;
    }
}
```

In XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             x:Class="HelloXamarinFormsWorldXaml.AbsoluteLayoutExample"
             Padding="20">

  <AbsoluteLayout>

    <Label Text="Stop"
           BackgroundColor="Red"
           Font="20"
           AbsoluteLayout.LayoutBounds="20,20,200,30" />

    <Label Text="Slow down"
           BackgroundColor="Yellow"
           Font="20"
           AbsoluteLayout.LayoutBounds="40,60,160,160" />

    <Label Text="Go"
           BackgroundColor="Green"
           Font="20"
           AbsoluteLayout.LayoutBounds="80,180,50,50" />

  </AbsoluteLayout>

</ContentPage>
```

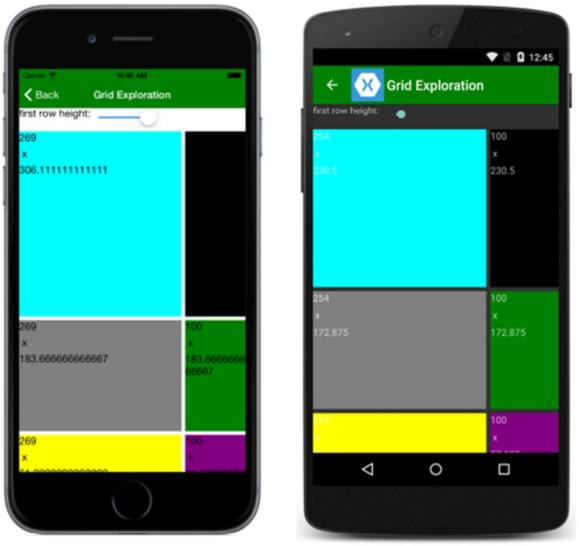
PXL IT

The result



Layouts: Grid

- A layout containing views arranged in rows and columns

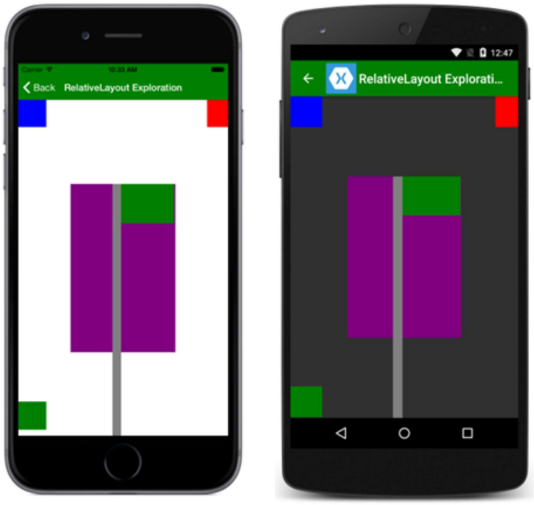


PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/layouts/grid

Layouts: RelativeLayout

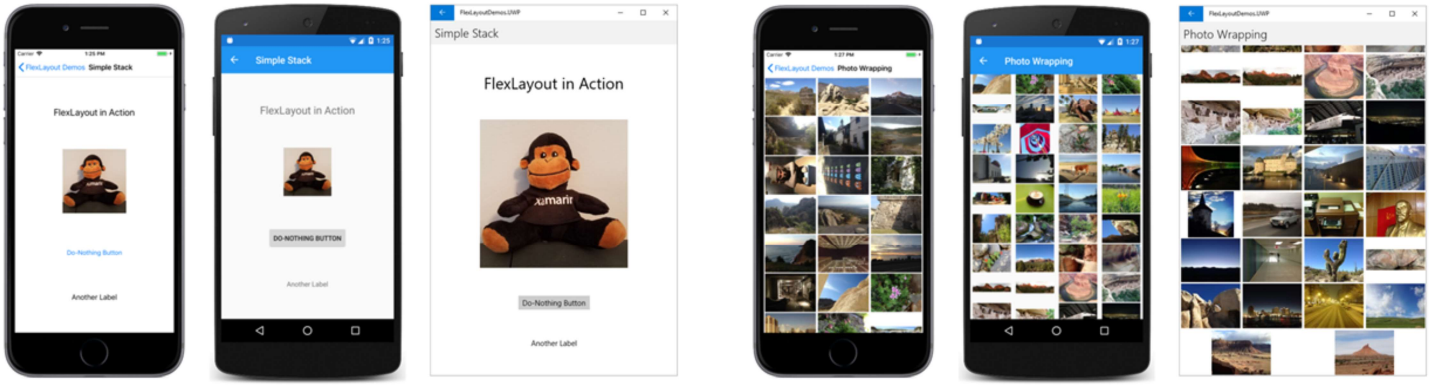
- A Layout that positions views on screen relative to the overall layout or to other views



https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/layouts/relative-layout

Layouts: FlexLayout

- A Layout for stacking or wrapping a collection of child views
  - ~ CSS Flexible Box Layout
  - Can be used for stacking or wrapping items



PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/layouts/flex-layout

PXL IT

UI Building Blocks: Views

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Views

- View is base class for all controls/widgets (buttons, labels…)
  - Coupled with a renderer to generate native visual element

PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/controls/views

Common views

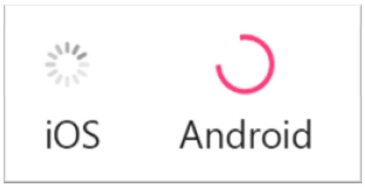
Xamarin.Forms Control	Description
Label	The Label is a read-only text display control
Entry	An Entry is a simple single-line text-input control
Button	Buttons are used to initiate commands
Image	This control is used to display a bitmap
ListView	The ListView presents a scrolling list of items. The items inside a list are known as cells
…. and many more	https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/controls/views

PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/controls/views

Views: ActivityIndicator

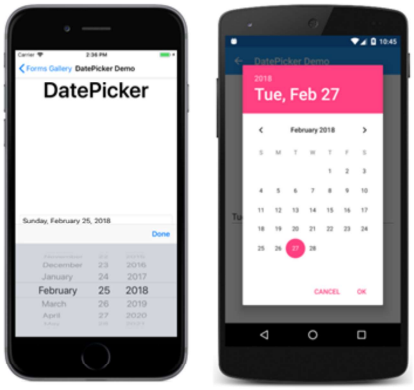
- A visual control used to indicate that something is ongoing
  - Gives a visual clue to the user that something is happening, without information about its progress



https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/activityindicator

Views: DatePicker

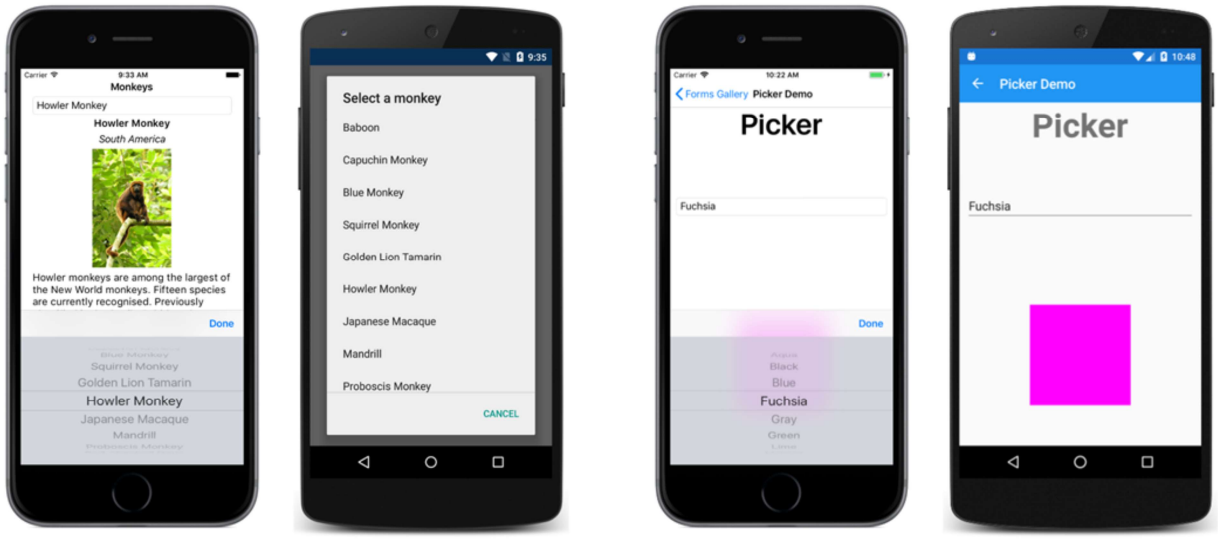
- Allows date picking (duh)
  - Visual representation of a DatePicker is very similar to the one of Entry
    - Except that a special control for picking a date appears in place of a keyboard



https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/datepicker

Views : Picker

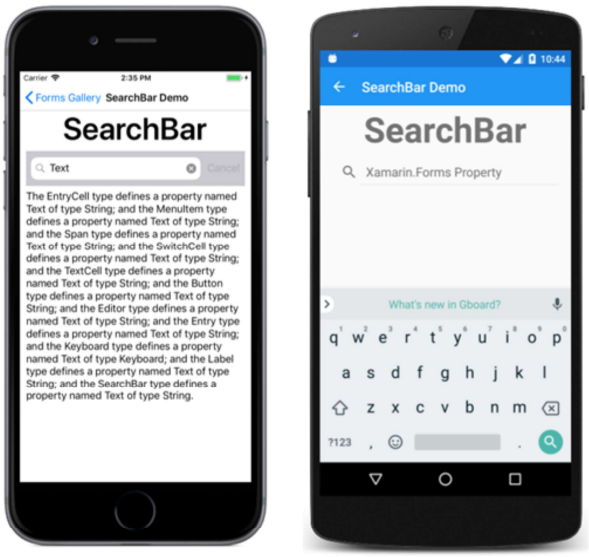
- A View control for picking an element in a list



https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/picker

Views: SearchBar

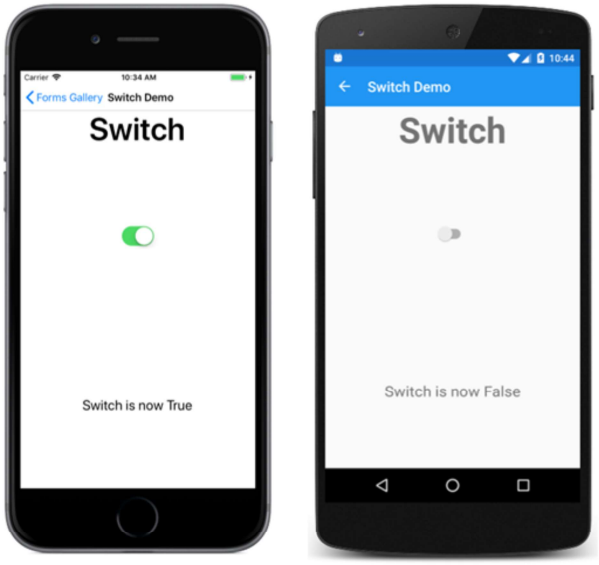
- A View control that provides a search box



https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/searchbar

Views: Switch

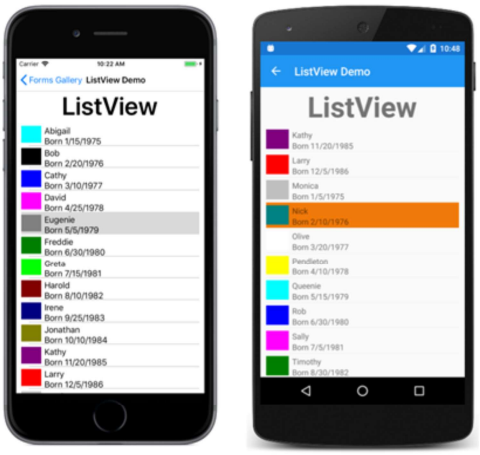
- A View control that provides a toggled value



https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/switch

Views: ListView

- A View control for presenting lists of data
  - Use ItemsSource property or databinding with a template to define the items
  - Each item is contained in a cell
    - By default, TextCell (renders single line of text)
  - Need more flexibility -> CollectionView



PXL IT

ListViews are a very common control in mobile applications and deserve to be covered in a bit more detail.

The **ListView** is responsible **for displaying a collection of items** on the screen; **each item** in the ListView will be **contained in a** single **cell**.

By default, a ListView will use the built-in TextCell template and render a single line of text. The code snippet in the slide is a simple example of using the ListView.

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/listview

Views: ListView - Binding to custom class

- TextCell can be used to show text representations of custom classes

```
listView.ItemsSource = new TodoItem [] {
    new TodoItem {Name = "Buy pears"},
    new TodoItem {Name = "Buy oranges", Done=true},
    new TodoItem {Name = "Buy mangos"},
    new TodoItem {Name = "Buy apples", Done=true},
    new TodoItem {Name = "Buy bananas", Done=true}
};
```

```
listView.ItemTemplate = new DataTemplate(typeof(TextCell));
listView.ItemTemplate.SetBinding(TextCell.TextProperty,
"Name");
```

Views: ListView - Item selection

```
listView.ItemSelected += async (sender, e) => {
    await DisplayAlert("Tapped!", e.SelectedItem + " was tapped.", "OK");
};
```

PXL IT

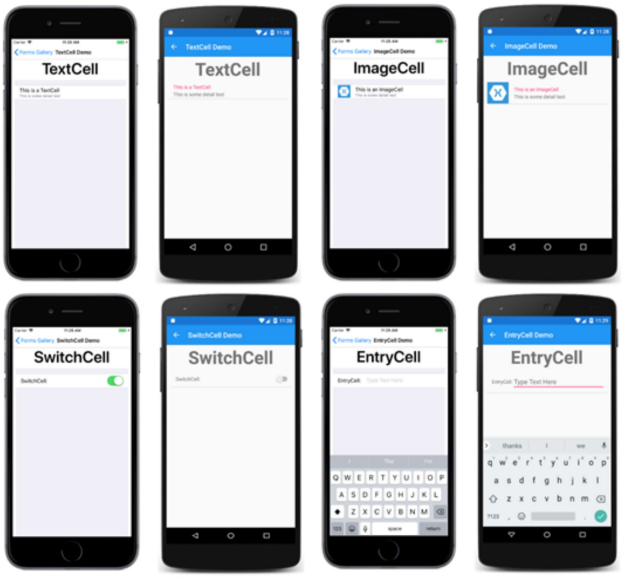
UI Building Blocks: Cells

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Cells

- A cell is a specialized element used for items in a ListView or TableView
- Build-in types:
  - TextCell (Text, detail)
  - ImageCell (Image, text, detail)
  - SwitchCell (Text, on/off switch)
  - EntryCell (Label, editable text)



PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/controls/cells

Custom cells

- We can use a custom cell by subclassing ViewCell

```
class EmployeeCell : ViewCell
{
    public EmployeeCell()
    {
        var image = new Image
                    {
                        HorizontalOptions = LayoutOptions.Start
                    };
        image.SetBinding(Image.SourceProperty, new Binding("ImageUri"));
        image.WidthRequest = image.HeightRequest = 40;

        var nameLayout = CreateNameLayout();

        var viewLayout = new StackLayout()
                         {
                             Orientation = StackOrientation.Horizontal,
                             Children = { image, nameLayout }
                         };
        View = viewLayout;
    }
}
```

EmployeeCell adds an Image and binds it to the ImageUri property of the Employee object (Data binding will be covered in just a moment).

It creates a StackLayout with a vertical orientation to hold the two Labels . The Labels are bound to the DisplayName property and the Twitter property of the Employee object.

It creates another StackLayout that will host the Image and the StackLayout from the previous two steps. It will arrange its children using a horizontal orientation.

https://docs.microsoft.com/xamarin/xamarin-forms/user-interface/listview/customizing-cell-appearance

Custom cells

- Also, set the type of this class as ItemTemplate for the ListView

```
List<Employee> myListOfEmployeeObjects = GetAListOfAllEmployees();
var listView = new ListView
{
    RowHeight = 40
};
listView.ItemsSource = myListOfEmployeeObjects;
listView.ItemTemplate = new DataTemplate(typeof(EmployeeCell));
```

PXL IT

This code will provide a List<Employee> objects to the ListView. Each cell will be rendered using the EmployeeCell class. The ListView will pass the Employee object to the EmployeeCell as its BindingContext.

Navigation

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Navigation

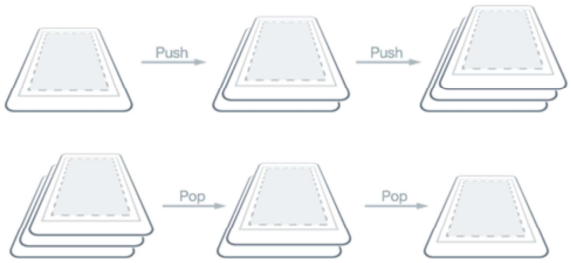
- Pages that group other pages
  - MasterDetailPage
  - TabbedPage
  - CarouselPage
- NavigationPage
  - Maintains a stack of Page objects
  - The Page on top of the stack is visible

PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/app-fundamentals/navigation/

NavigationPage

- Navigation on NavigationPage is last-in, first-out
  - Pages are being pushed onto a stack with forward navigation
  - Pages are retrieved from the stack on backward navigation
- Handled in Xamarin.Forms with the *INavigation* interface
  - Accessible via *Navigation* property of current page




```
public interface INavigation
{
    Task PushAsync(Page page);
    Task<Page> PopAsync();
    Task PopToRootAsync();
    Task PushModalAsync(Page page);
    Task<Page> PopModalAsync();
}
```

PXL IT

Navigation on a NavigationPage can be thought of as a **last-in, first-out stack of Page objects**.

To move from one page to another an application will push a new page onto this stack.

To return back to the previous page the application will pop the current page from the stack.

This navigation in Xamarin.Forms is handled by the *INavigation* interface which provides the methods displayed in the slide.

Navigable elements (pages) have a *Navigation* property of type *INavigation*.

So to navigate to another page you can use the Navigation property of the current page.

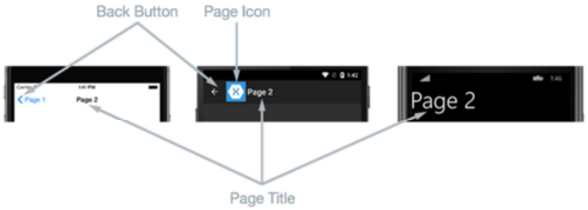
https://docs.microsoft.com/xamarin/xamarin-forms/app-fundamentals/navigation/hierarchical

https://docs.microsoft.com/en-us/dotnet/api/xamarin.forms.navigableelement.navigation?view=xamarin-forms#Xamarin_Forms_NavigableElement_Navigation

https://docs.microsoft.com/dotnet/api/xamarin.forms.inavigation

NavigationPage

- NavigationPage implements the INavigation interface
  - Also adds navigation bar at the top of the page with title and back button



- Typically, a NavigationPage wraps the first page in the app

```
public static Page GetMainPage()
{
    var mainNav = new NavigationPage(new EmployeeListPage());
    return mainNav;
}
```

The **NavigationPage** class that **implements** the **INavigation interface** and will manage the stack of Pages.

The NavigationPage class will also add a navigation bar to the top of the screen that displays a title and will also have a platform appropriate Back button that will return to the previous page.

The code in the slide shows how to wrap a NavigationPage around the first page in an application.

NavigationPage

- To show the next page, we use PushAsync()

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

- To navigate backward, we use PopAsync()

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

PXL IT

Shell

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Shell

- Reduces complexity by providing the fundamental features that most mobile apps require
  - Flyout
    - Root menu accessible through an icon or by swiping from the side of the screen
  - Tabs
    - After a flyout the next level of navigation is a bottom tab bar
    - Tab class implements INavigation
  - Navigation
    - URI-based navigation that uses routes to navigate to any page in the application
  - Search
    - Search box can be added to the top of each page

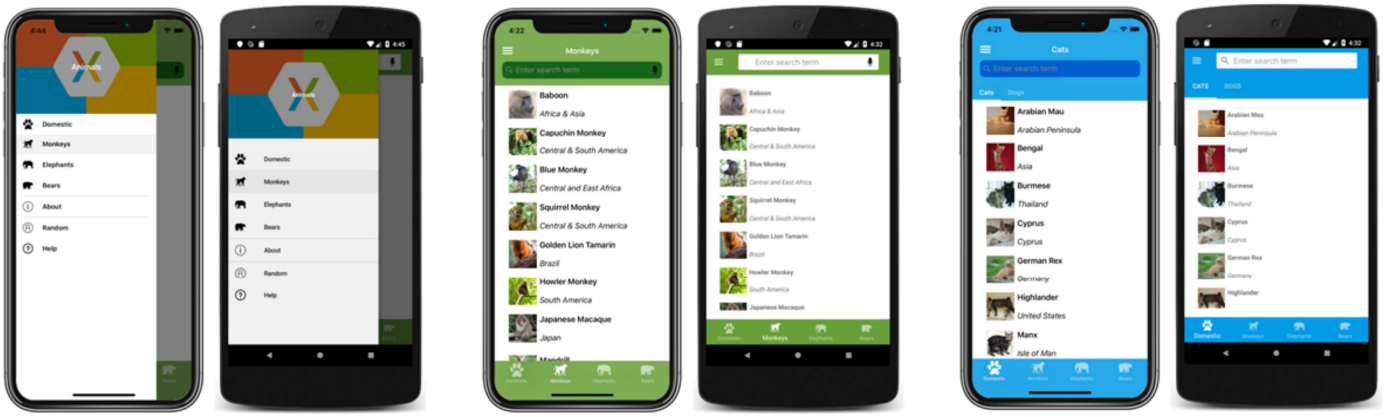
PXL IT

https://docs.microsoft.com/xamarin/xamarin-forms/app-fundamentals/shell/

https://docs.microsoft.com/xamarin/xamarin-forms/app-fundamentals/shell/introduction

Only iOS/Android, no UWP version (yet)

Shell


Domestic
Monkeys
Elephants
Bears
About
Random
Help
Baboon
Africa & Asia
Capuchin Monkey
Central & South America
Blue Monkey
Central and East Africa
Squirrel Monkey
Central & South America
Golden Lion Tamarin
Brazil
Howler Monkey
South America
Japanese Macaque
Japan
Arabian Mau
Arabian Peninsula
Bengal
Asia
Burmese
Thailand
Cyprus
Cyprus
German Rex
Germany
Highlander
United States
Manx
Isle of Man

Platform features

DE HOGESCHOOL MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Platform tweaks

- The Device class allows detecting the used platform in code
  - Small platform tweaks become possible
- Device.Idiom
  - Used to alter layouts or functionality depending on whether the device is a phone or a tablet
    - Enum value: Phone or Tablet

```
if (Device.Idiom == TargetIdiom.Phone) {
    // layout views vertically
} else {
    // layout views horizontally
}
```

PXL IT

Xamarin.Forms provides a **number of methods that allow functionality and layout to be altered for each platform**.

For simpler platform-specific tweaks, such as layout changes, the Device class allows developers to detect the platform in shared code.

References:

https://docs.microsoft.com/xamarin/xamarin-forms/platform

https://docs.microsoft.com/xamarin/xamarin-forms/platform/device

Platform tweaks

- Device.OS can be set to one of the TargetPlatform values
  - iOS, Android, UWP, macOS, Linux, Tizen
  - Allows small changes to be made in the layout

```
if (Device.RuntimePlatform == Device.iOS)
{
    stackLayout.Padding = new Thickness(0, 20, 0, 0);
} else if (Device.RuntimePlatform == Device.Android)
{
    label.FontSize = Device.GetNamedSize(NamedSize.Medium, label);
}
else
{
    label.FontSize = 24;
}
```

PXL IT

Platform tweaks

- Device.OpenUri can be used to open another app based on a protocol

```
Device.OpenUri(new Uri("http://xamarin.com/evolve"));
```

PXL IT

UI Building Blocks

DEMO – FORMS GALLERY

Demo: Forms Gallery contains demos for all possible controls

Hosted here: https://docs.microsoft.com/samples/xamarin/xamarin-forms-samples/formsgallery

CAN YOU MAKE SOMETHING REALLY PRETTY WITH XAMARIN.FORMS?

81

Some examples and pointers

- https://www.syncfusion.com/essential-xamarin-ui-kit
- https://github.com/jsuarezruiz/xamarin-forms-goodlooking-UI
- https://devblogs.microsoft.com/xamarin/snppts-ui-snippets-xamarin-forms/

PXL IT

https://www.syncfusion.com/essential-xamarin-ui-kit
https://github.com/jsuarezruiz/xamarin-forms-goodlooking-UI
https://devblogs.microsoft.com/xamarin/snppts-ui-snippets-xamarin-forms/