

XAMARIN.FORMS



VISÃO GERAL

Visão geral

Lançado em 28/05/14.

Código fonte aberto, disponível no GitHub.



Destinado a ser uma abstração para escrita da UI dos apps, tal como o .NET Framework para compartilhar código entre as plataformas.

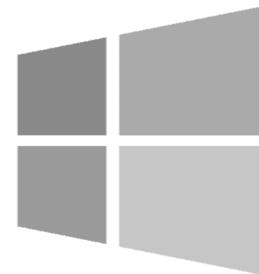
Visão geral



iPhone



Smartphones



iPad



Tablets



Visão geral

Oferece um conjunto de componentes visuais padronizados:

Layouts

Views e Controles

Sistema de Navegação

Efeitos visuais

Etc.

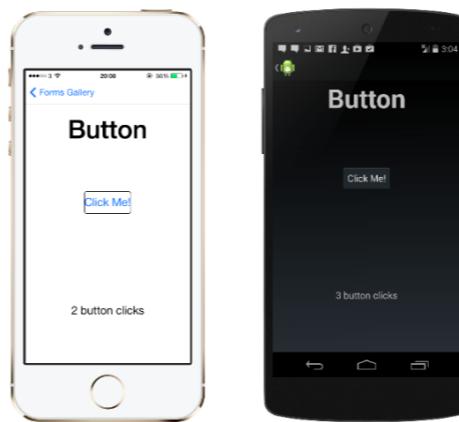


Visão geral

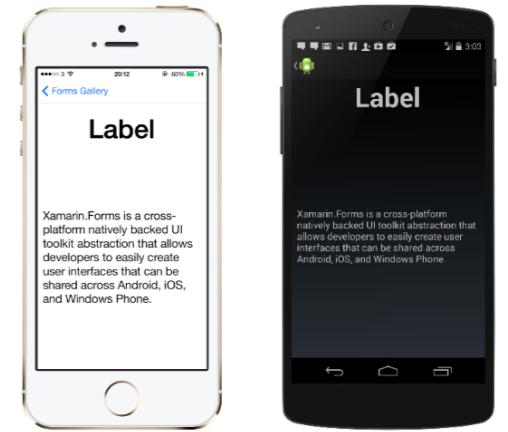
Esses componentes são abstrações, que são processados de maneira específica em cada plataforma, podendo:

1. Ser diretamente transcritos para os componentes nativos de cada plataforma.

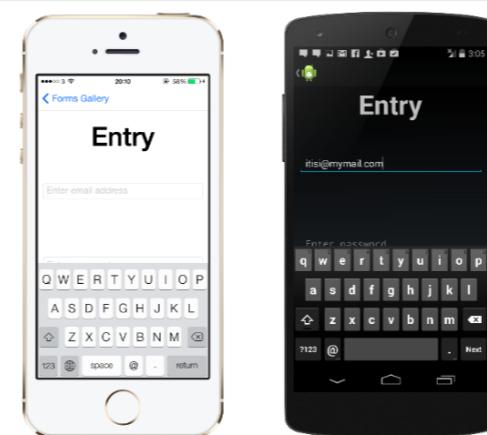
Componente Xamarin iOS / Android



Button
UIButton / Button



Label
UILabel / TextView



Entry
UITextField / EditText



Visão geral

Esses componentes são abstrações, que são processados de maneira específica em cada plataforma, podendo:

2. Ser implementados nativamente

Componente Xamarin - iOS / Android



Stepper

Componente para
seleção numérica



Essa arquitetura nos permite um código único para descrever uma interface, sem perda das características próprias de cada plataforma nem de performance.

VANTAGENS DO XAMARIN.FORMS

Vantagens do Xamarin.Forms

Base de código unificada para UI.

Suporte ao XAML e Databinding.

Acesso total a recursos nativos das plataformas e dispositivos.

Suporte a estilos e internacionalização de maneira unificada.



Vantagens do Xamarin.Forms

Suporte a personalizações granulares:

Nível de controle ou propriedade

Nível de tipo de dispositivo (Idiom)

Nível de plataforma

Modelo de extensibilidade flexível.

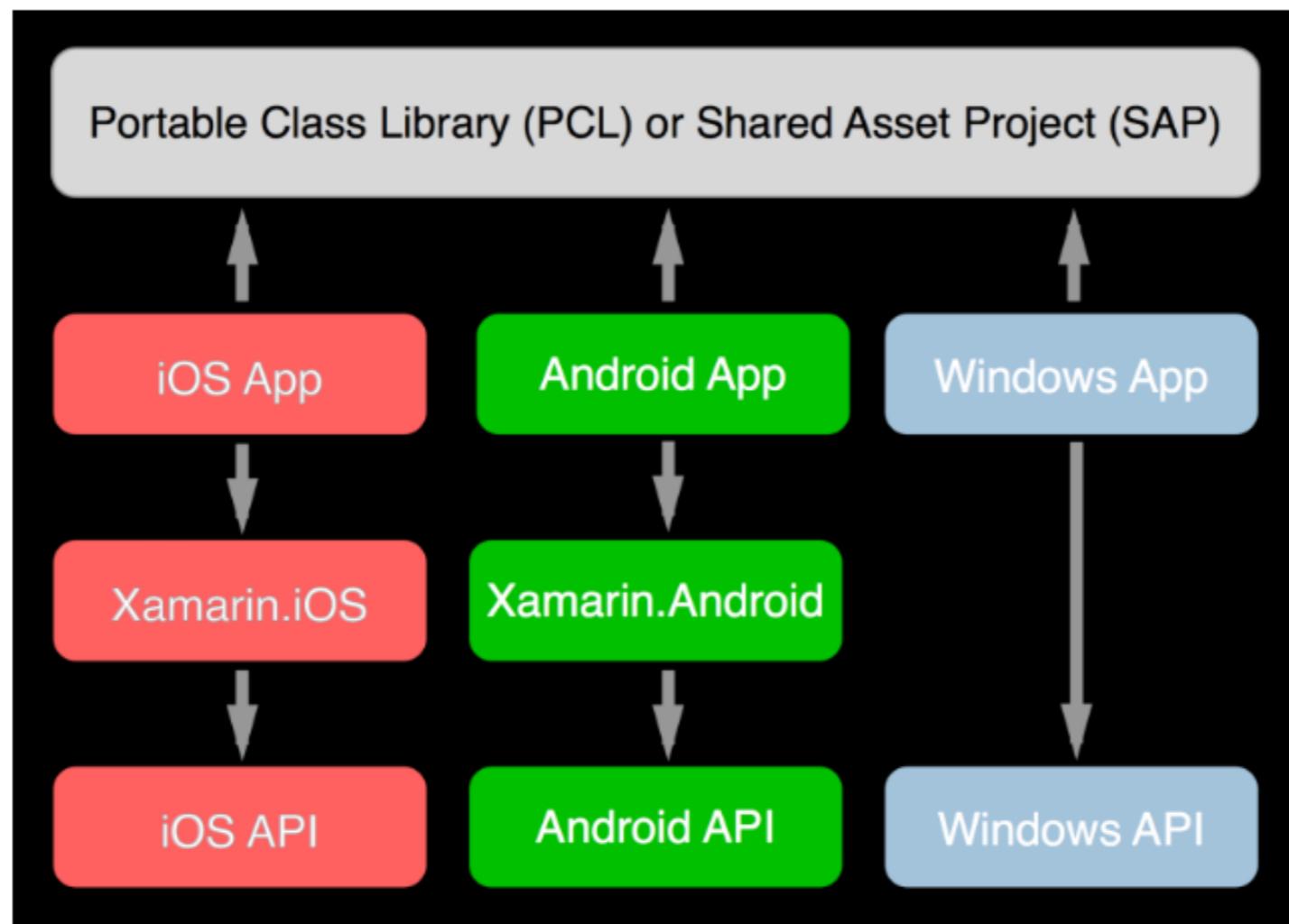
Plugins e suporte da comunidade.



COMPARTILHAMENTO DE CÓDIGO

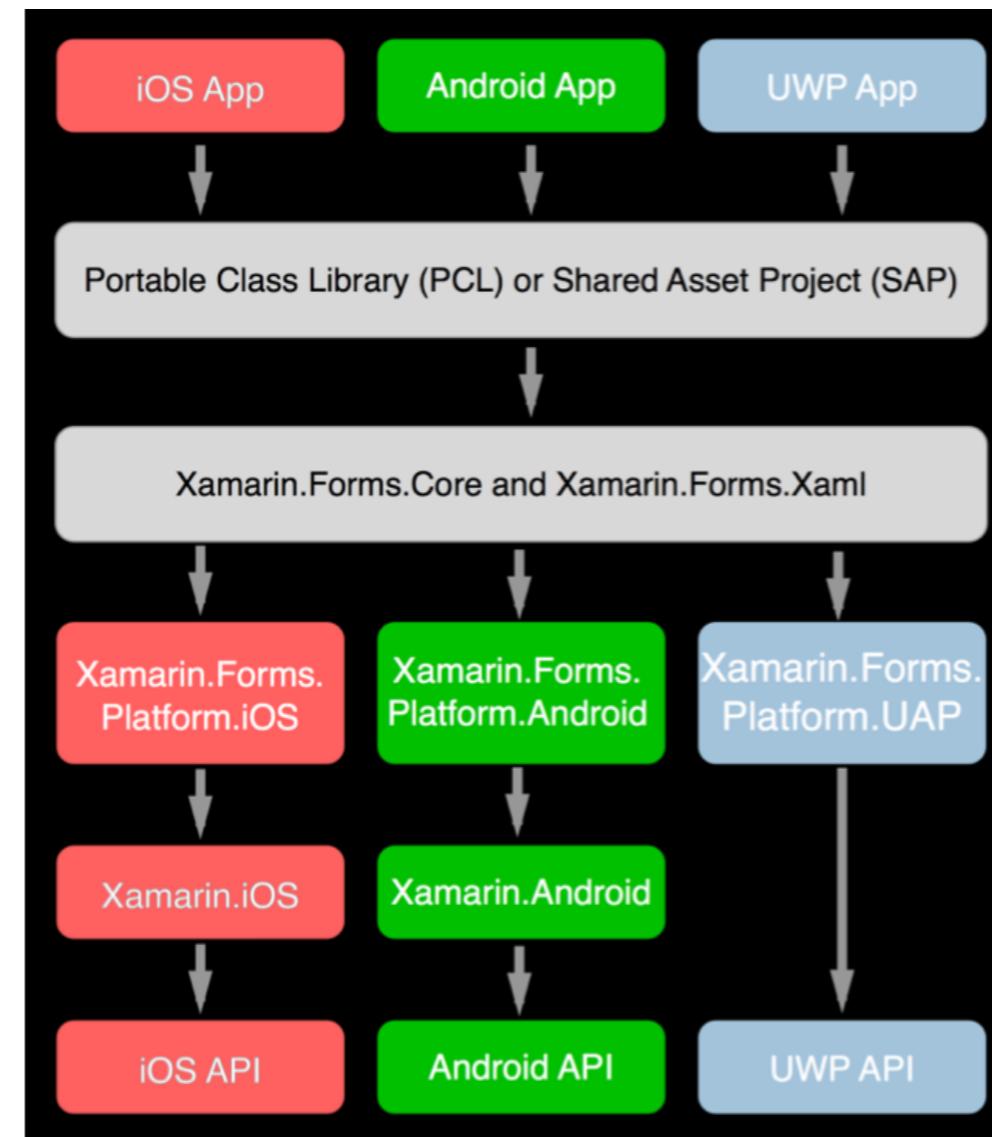
Modelo tradicional

O compartilhamento de código com o Xamarin funciona da seguinte forma:



Usando o Xamarin.Forms

Incluímos uma nova camada de abstração para criação do código de UI.



Camadas

Xamarin.Forms.Core / Xamarin.Forms.Xaml

Camada de abstração

Contém os componentes de alto nível do Xamarin.Forms (Botões, Páginas, Views, etc)

Inclui o **parser** do XAML



Camadas

Xamarin.Forms.Platform.iOS

Implementa nativamente os componentes de alto nível do Forms.

Inclui os renderers responsáveis pela implementação dessa abstração

Xamarin.Forms.Platform.Android

Mesma coisa que o anterior mas específico para Android



.NET Standard vs SAP

.NET Standard

Código é compilado em uma biblioteca do tipo .NET Standard que é referenciada pelos projetos (dynamic linking)

Método recomendado por ser menos suscetível a problemas de compilação



PCL vs SAP

SAP (Shared Asset Project)

O código é compilado junto com o código de cada projeto
(static linking)

Permite usar compilação condicional para gerar códigos específicos de uma plataforma

Mais simples de compartilhar código, mas pode gerar conflitos entre os projetos e virar uma “colcha de retalhos” para tratar casos específicos



ANATOMIA DE UM APP

Anatomia de um app

O Xamarin Studio e o Visual Studio tem templates para criação de projetos Xamarin.Forms.

Uma solution básica é composta de 3 projetos:

Projeto do Xamarin.Forms

Uma biblioteca PCL ou Shared Project.
Contém o código compartilhado do app em si.

Projeto do app Android

Um projeto Xamarin.Android.
Faz referência ao projeto do Xamarin.Forms.

Projeto do app iOS

Um projeto Xamarin.iOS.
Faz referência ao projeto do Xamarin.Forms.



Anatomia de um app

O Visual Studio também cria por padrão mais 3 projetos referentes a plataforma Windows.

Os projetos específicos de cada plataforma referenciam o projeto do Xamarin.Forms, que contém todo o código da UI e do app.

Contém apenas o código básico para inicialização do “Engine” do Xamarin.Forms.

Geram os binários para distribuição (IPA, APK, etc.).



PRÁTICA: PRIMEIRO APP

CONHECENDO o XAML

Conhecendo o XAML

Sigla de eXtensible Application Markup Language, é derivada do XML como o nome sugere.

Criada pela Microsoft como linguagem para descrever a interface de aplicações.

Conhecendo o XAML

Introduzida com o Framework WPF (Windows Presentation Foundation) no Visual Studio 2008.

O objetivo era separar de maneira efetiva o código de apresentação do código do programa, de forma que o designer pudesse trabalhar a interface de maneira isolada.

Expression Blend era usado pelos designers para criação das interfaces.

Conhecendo o XAML

Foi implementada pela Xamarin para uso com o Xamarin.Forms.

Contém um conjunto rico de recursos para criação de interfaces, sendo mais simples e legível que a codificação direta em C#.

ANATOMIA DE UMA CLASSE XAML

Anatomia de uma classe XAML

Um código XAML é a representação de um gráfico de objetos .NET

No caso do XAML para Xamarin.Forms, cada arquivo representa a estrutura de uma Página ou View.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="HelloForms.Xaml.MainPage">
</ContentPage>
```

Exemplo de código

PARTES

2 Namespaces XML (xmlns)

<http://xamarin.com/schemas/2014/forms>

Tags definidas sem prefixo de namespace

São as classes pertencentes ao Xamarin.Forms

<http://schemas.microsoft.com/winfx/2009/xaml>

Prefixo de namespace x:

Diversos elementos e atributos intrínsecos ao XAML

Supostamente suportamos em todas as implementações do XAML

x:Class

É o nome da classe que representa o objeto raiz.

O FQN de uma classe .NET

Só pode aparecer no elemento raiz.

Usado pelo VS/XS para determinar o "Code-Behind".

CONTEÚDO

Conteúdo

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="XamlSamples.XamlPlusCodePage"
    Title="XAML + Code Page">
    <StackLayout>
        <Slider
            VerticalOptions="CenterAndExpand"
            ValueChanged="OnSliderValueChanged" />
        <Label
            Text="A simple Label"
            Font="Large"
            HorizontalOptions="Center"
            VerticalOptions="CenterAndExpand" />
        <Button
            Text="Click Me!"
            HorizontalOptions="Center"
            VerticalOptions="CenterAndExpand"
            Clicked="OnButtonClicked" />
    </StackLayout>
</ContentPage>
```

O gráfico de objetos daquele componente será definido pelos elementos definidos no interior do elemento raiz (**ContentPage**).

Declara uma página (**ContentPage**) com um **StackLayout**, empilhando verticalmente um **Label** e um **Botão (Button)**.

CODE-BEHIND

Code-Behind

O XAML é uma linguagem declarativa que nos permite apenas declarar os elementos que formarão aquele gráfico de objetos.

O “Code-Behind” define um arquivo de código-fonte C# que complementa o XAML, podendo manipular esse gráfico ou responder a eventos gerados por seus objetos.

Code-Behind

x:Name - Acessando objetos do XAML

Se precisarmos acessar objetos declarados no XML, podemos incluir um atributo **x:Name**

Ao dar um nome a esse objeto, ele se torna acessível através do Code-Behind.

```
<Label  
    x:Name="simpleLabel"  
    Text="A simple Label"  
    Font="Large"  
    HorizontalOptions="Center"  
    VerticalOptions="CenterAndExpand" />
```

Exemplo



Code-Behind

Respondendo a Eventos

Podemos atribuir o nome de um **Event Handler** a objetos .NET que declarem um evento.

Escrevemos esse **Event Handler** no “Code-Behind”.

```
<Button  
    Text="Click Me!"  
    HorizontalOptions="Center"  
    VerticalOptions="CenterAndExpand"  
    Clicked="OnButtonClicked" />
```

Exemplo

PRÁTICA: APPS COM XAML

COMPONENTES DO XAMARIN.FORMS

Componentes do Xamarin.Forms

Os componentes visuais do Xamarin.Forms são geralmente nomeados “Controles”.

Se dividem em 4 categorias principais:

Pages

Layouts

Views

Cells

PAGES

Pages

Representam uma tela do App.

Incluem especializações que ajudam na construção das estruturas mais comuns de apps.

Controles Disponíveis:

ContentPage

MasterDetailPage

NavigationPage

TabbedPage

TemplatePage

CarrousselPage

LAYOUTS

Layouts

Controles usados para argupar e organizar outros controles.

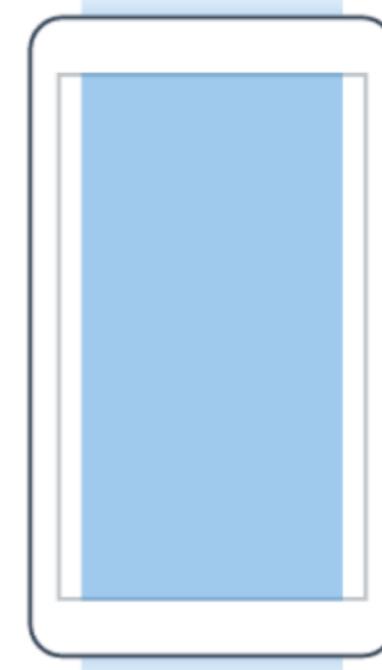
Controles Disponíveis:



ContentPresenter



ContentView

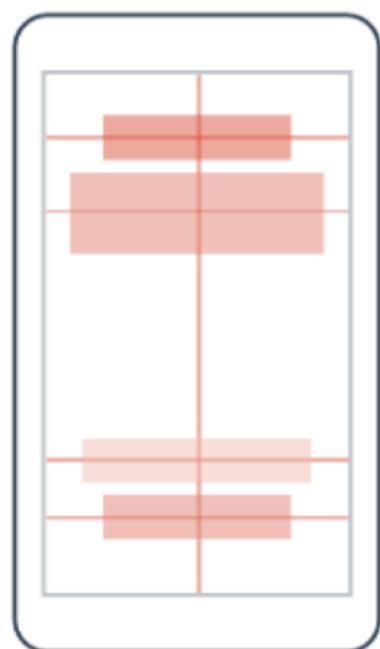


ScrollView

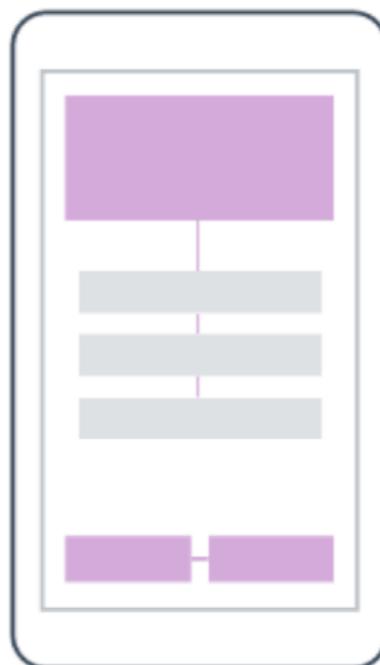
Layouts

Controles usados para argupar e organizar outros controles.

Controles Disponíveis:



AbsoluteLayout



RelativeLayout



Grid

Layouts

Controles usados para argupar e organizar outros controles.

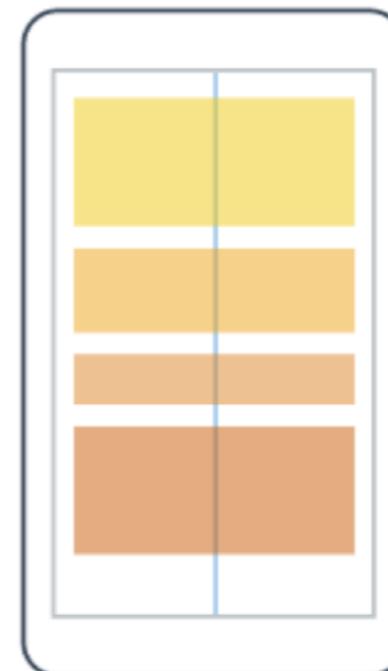
Controles Disponíveis:



Frame



TemplatedView



StackLayout

VIEWS

Views

Controles especializados na apresentação de elementos comuns de interface e interação com o usuário.

Controles Disponíveis:

ActivityIndicator	Editor	Picker	Switch
BoxView	Entry	ProgressBar	TableView
Button	Image	Slider	TimePicker
DatePicker	ListView	Stepper	WebView

CELLS

Cells

Um tipo especial de Controle utilizado em ListViews e TableViews.

Controles Disponíveis:

EntryCell

SwitchCell

TextCell

ImageCell

COMPONENTES BÁSICOS

Componentes básicos

Aprofundamento sobre os controles básicos do **Xamarin.Forms**.

Esses componentes formam o conjunto de elementos fundamentais para construção de interfaces.

Vamos aprofundar mais sobre eles devido a sua importância.

Controles:

ContentPage

StackLayout

Grid

Label

Button

Entry

Switch

ContentPage

Representa uma tela para qual podemos navegar no **Xamarin.Forms**.

Contém uma única View que representa seu conteúdo.

Atribuímos a Propriedade 'Content' (do tipo View) ao seu conteúdo.

Label

Exibição de textos curtos.

Permite diversos tipos de personalização.

Propriedades:

Text

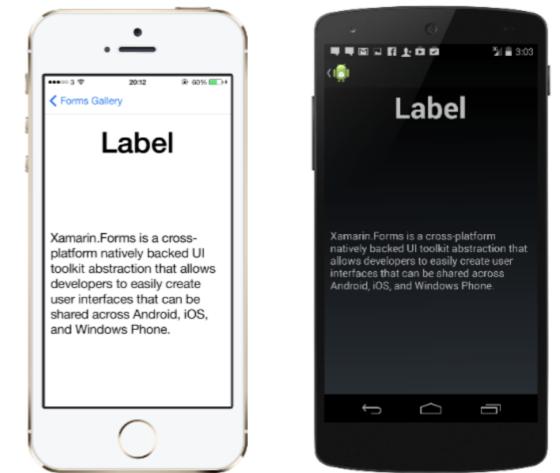
FontSize

FontFamily

TextColor

FontAttributes (Negrito, Itálico)

Horizontal/VerticalTextAlignment



Possível atribuir conteúdo formatado usando “**FormattedString**”.

DEMO: LABEL

StackLayout

Um tipo de layout onde seus elementos são dispostos linearmente (ou ‘empilhados’).

Dispõe dos elementos verticalmente (padrão) ou horizontalmente, definido pela propriedade “Orientation”.

Espaçamento entre os elementos é determinado pela propriedade “Spacing”.



StackLayout

É uma das formas mais simples de compor interfaces.

É muito comum aninhar **StackLayouts**.



DEMO: STACK LAYOUT

Grid

Um tipo de layout onde os elementos são dispostos em linhas e colunas.

As linhas e colunas são pré-definidas e podem ter largura e altura variados.

Propriedades “**ColumnSpacing**” e “**RowSpacing**” determinam o espaçamento entre linhas e colunas.



Grid

Propriedades “**RowDefinitions**” e “**ColumnDefinitions**” determinam as linhas e colunas

RowDefinition

Propriedade Height determina a altura da coluna

ColumnDefinition

Propriedade Width determina a largura da coluna



Grid

Determinando largura e altura

Há 3 formas de se determinar largura e altura das células do Grid:

Absolute

Auto

Star

As possibilidades são definidas pela enumeração “**GridUnitType**”

Grid

Em XAML usamos um “Attached Property” para atribuir em qual linha/coluna do Grid aquele elemento se mostra.

DEMO: GRID

ScrollView

Um tipo de Layout que possibilita rolagem vertical e horizontal de seu conteúdo

Usado para interfaces que extrapolam o tamanho disponível na tela

Atribuímos a sua propriedade Content, um layout.

DEMO: SCROLLVIEW

Buttons

Botões.

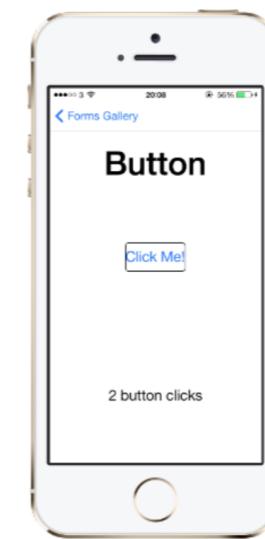
Permite diversos tipos de personalização.

Propriedades:

Text

Image

Font(Atributes/Family/Size)



TextColor

Border(Radius/Color/Width)

Evento Clicked.

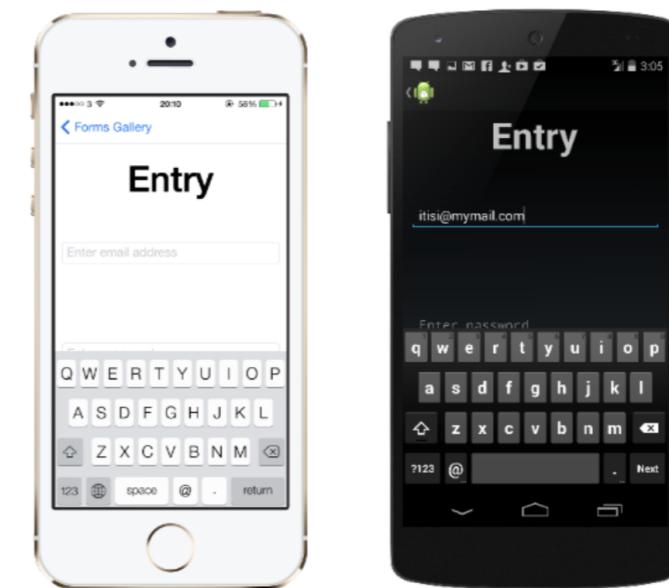
DEMO: BOTÃO

Entry

Entrada básica de texto.

Uma linha de texto.

Pode ser usado para obter senhas.



Propriedades:

Text

IsPassword

Font(Atributes/Family/Size)

Placeholder

HorizontalTextAlignment

TextColor



Entry

Eventos

Completed

TextChanged



DEMO: ENTRY

Switch

Um Checkbox para entrada de um valor verdadeiro/falso.

Marcação é controlada pela propriedade “IsToggled”

Contém o evento Toggled para notificar sobre mudanças.



DEMO: SWITCH

NAVEGAÇÃO COM XAMARIN.FORMS



PÁGINAS

Páginas

Toda navegação em Xamarin.Forms acontece através de Páginas
(Componentes que herdam da Classe “Page”.

Dois tipos de navegação:

Não Modal

Uma navegação
sequencial, como
se fossem páginas
de um site

Modal

Uma página que
requer foco ou ação
específica do usuário

Páginas

A navegação é controlada por 2 pares de métodos:

Não Modal

```
Task PushAsync(Page page)  
Task PopAsync()
```

Modal

```
Task PushModalAsync(Page page)  
Task PopModalAsync()
```

Páginas

Os 4 métodos estão disponíveis na propriedade “Navigation”

Exemplo:

```
await Navigation.PushAsync(new DetailPage());  
await Navigation.PushModalAsync(new DetailPage());
```

ENTENDENDO A NAVEGAÇÃO

Páginas não-modais

Para criar uma navegação não modal, é necessário que a página esteja inserida em um “**NavigationPage**”.

O **NavigationPage** automaticamente inclui uma barra de navegação padrão do sistema.

Páginas modais

Recomendada quando se deseja obter alguma informação ou interação do usuário, antes de retornar a tela anterior;

Páginas modais podem iniciar uma nova sequência de navegação não-modal se a nova página for uma “**NavigationPage**”.

Controlando a navegação

Uma página criada como não modal, com o método “PushAsync(Page)”, deve ser removida usando o método “PopAsync()”.

O mesmo é válido para páginas modais, quando criadas com “PushModalAsync()” devem ser removidas com “PopModalAsync()”.

TIPOS DE PÁGINAS E NAVEGAÇÕES

NavigationPage

Permite criar uma estrutura de navegação sequencial;

As páginas incluem por padrão uma barra de navegação;

A apresentação e exclusão das páginas é muito semelhante a de um browser;



NavigationPage

É possível modificar a aparência da barra de navegação ou remove-la.

Ao remover a barra de navegação é importante proporcionar outras formas para o usuário se localizar dentro da estrutura de páginas do aplicativo, e voltar para a página anterior

Propriedades

BarBackgroundColor

BarTextColor

NavigationPage

Métodos

NavigationPage.SetHasBackButton(Page, bool)

NavigationPage.SetBackButtonTitle(Page, string)

NavigationPage.SetHasNavigationBar(Page, bool)

NavigationPageSetTitleIcon(Page, string)

DEMO: NAVIGATION PAGE

TabPage

Um tipo especial de página cujos filhos também são páginas;

Cada filho é apresentado como uma aba;

O sistema de abas respeita as particularidades do sistema:



Usa o ActionBar para apresentar as abas na parte superior.



Usa o UITabViewController apresentando as abas na parte inferior



TabPage

As páginas filhas do Tabbed Page podem ser navigation Pages, criando uma estrutura de navegação própria para cada seção.

Suas páginas podem ser alimentadas por Data-Binding.

O título das seções é definidos pela Propriedade “Title” de cada página filha.

O ícone das seções é definido pela Propriedade “Icon” de cada página filha (apenas iOS).



DEMO: TABBEDPAGE

MasterDetailPage

Um tipo de página com duas páginas filha:

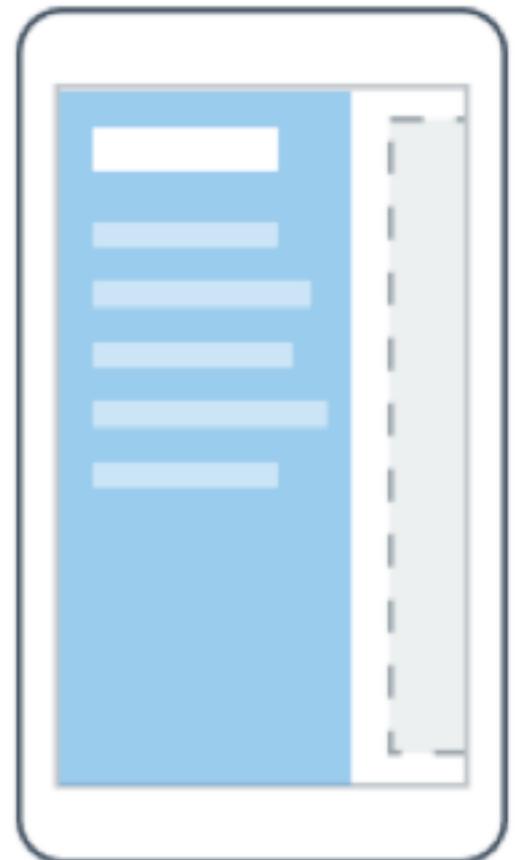
Master

Uma página de navegação;
Geralmente apresenta um resumo de conteúdos para os quais se pode obter mais informações;

Aparece na lateral, como um “Drawer”.

Details

Uma página que complementa a seleção feita na master;
Varia de acordo com a seleção.



MasterDetailPage

O MasterDetailPage proporciona uma maneira facilitada de criar menus laterais em aplicativos

Quando utilizada em templates, possibilita visualizar as duas páginas lado a lado.

MasterDetailPage

Apresentação é controlada pela propriedade "MasterBehavior"

Default

Split

SplitOnLandscape

SplitOnPortrait

Popover

Só tem efeito em Tablets e Desktops, ignorada em Smartphones (Idiom)

MasterDetailPage

Suporta gesto de Swipe para apresentar a Master;

Controlado pela propriedade “**IsGestureEnabled**”.

DEMO: MASTERDETAILPAGE

FLEXLAYOUT



Conhecendo o Flex Layout

StackLayout "anabolizado"

Um tipo de painel onde os elementos são dispostos linearmente

Derivado do Flexbox do HTML

Ajuda na composição de layouts planos e responsivos

Substitui o StackLayout e o Grid

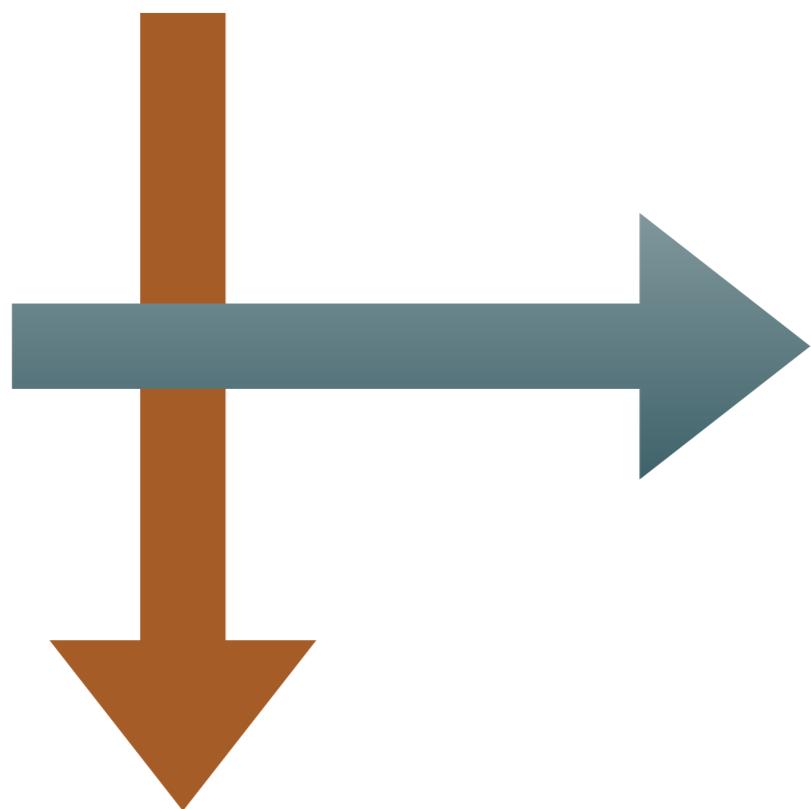
Conhecendo o Flex Layout

Definição do eixo de distribuição

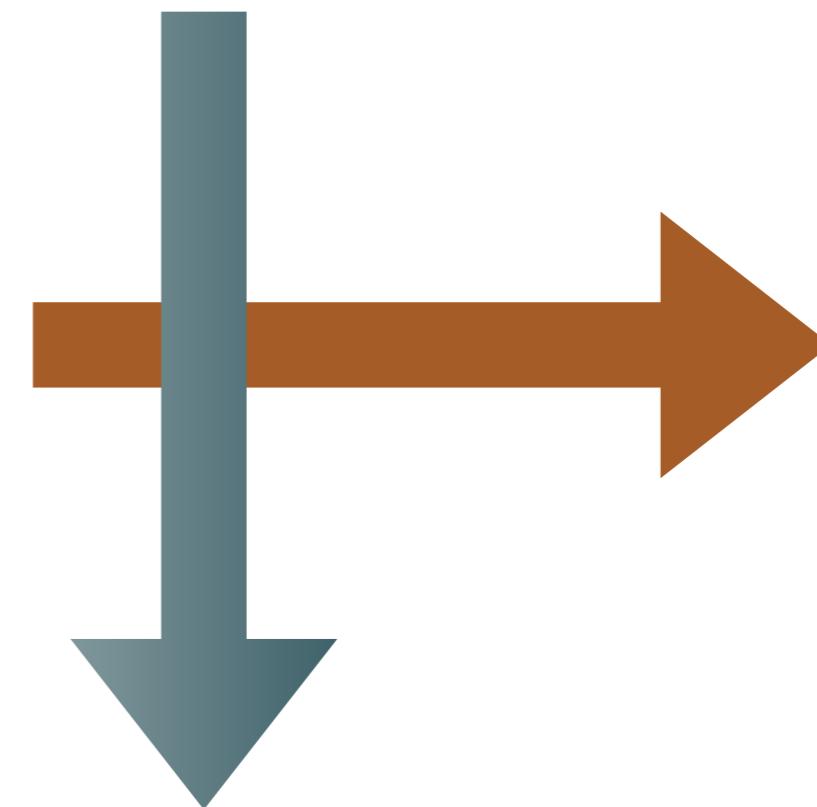
- **Direction:** determina a direção os elementos serão dispostos:
 - **Row:** horizontalmente da esquerda para direita
 - **Column:** verticalmente de cima para baixo
 - **ColumnReverse/RowReverse:** inverte a direção (Row - direita para esquerda, Column - de baixo para cima)

Conhecendo o Flex Layout

Row



Column



Conhecendo o Flex Layout

Principais propriedades

- **Wrap**: configura o comportamento do layout quando seus elementos excedem o espaço das linhas ou colunas
- **NoWrap**: padrão, não há quebra de linha ou coluna, os elementos são “espremidos”
- **Wrap**: uma nova linha ou coluna será criada quando um elemento precisar de mais espaço que o disponível
- **Reverse**: equivalente ao Wrap, porém o elemento entrará na nova linha ou coluna na ordem inversa do Wrap

Conhecendo o Flex Layout

Principais propriedades

- **JustifyContent** (*Start, Center, End, SpaceBetween, SpaceAround, SpaceEvenly*) - espaçamento dos itens no eixo principal
- **AlignItems** (*Stretch, Center, Start, End*) - alinhamento dos itens no eixo cruzado
- **AlignContent** (*Stretch, Center, Start, End, SpaceBetween, SpaceAround, SpaceEvenly*) - alinhamento geral no eixo cruzado

Conhecendo o Flex Layout

Propriedades Anexadas

- **AlignSelf:** sobrescreve o *AlignItems* individualmente
- **Order:** ordem em que o elemento é apresentado
- **Basis:** tamanho do elemento, proporcional (%) ou absoluto
- **Grow:** determina a expansão do elemento
- **Shrink:** inverso do grow

Referências

- Documentação oficial do componente FlexLayout
- Projeto Flexibility no GitHub
- FlexLayout Demos

ESTILOS



Trabalhando com estilos

Há duas formas de aplicar de estilos no **Xamarin.Forms**

- **Estilos Nativos:** derivada do XAML. Personalização completa das propriedades. Algum nível de herança.
- **CSS:** é a mesma linguagem aplicada aos elementos do Forms. Tem algumas limitações com relação ao anterior.

Estilos Nativos

São declarados em XML

Deve especificar o elemento ao qual podem ser aplicadas

Podem ser declarados em linha, ou colocado em bibliotecas de estilos (App.xaml).

Inclui um mecanismo de herança

x:Key determina o nome do estilo. Caso omitido cria um estilo implícito

Estilos Nativos

Declarando um estilo:

```
<ContentPage.Resources>
    <ResourceDictionary>
        <Style x:Key="titulo" x:Type="Label">
            <Setter Property="FontSize" Value="Large" />
            <Setter Property="TextAttributes" Value="Bold" />
            <Setter Property="TextColor" Value="White" />
        </Style>
    </ResourceDictionary>
</ContentPage.Resources>
```

Estilos Nativos

Aplicando um estilo:

```
<Label  
    Style="{StaticResource titulo}"  
    Text="Texto apresentado no Label" />
```

Estilos CSS

Buscam atrair os desenvolvedores Web e simplificar a declaração de estilos (os nativos são muito prolixos).

Podem ser incorporados em um XAML ou declarados em arquivos .css, facilitando o compartilhamento de estilos.

Suporta os seletores mais comuns do CSS.

Estilos CSS - Seletores

- *
- .class (*StyleClass*)
- #id (*StyleId*)
- element
- ^base
- element, element (*múltiplo*)
- element element (*filhos*)
- element>element (*pai*)
- element+element (*depois*)
- element~element (*antes*)

Estilos CSS - Seletores indisponíveis

- [attribute]
- @media
- @supports
- : e ::

Estilos CSS - Propriedades

O CSS suporta apenas um sub-conjunto das propriedades dos elementos do Xamarin.Forms.

Procura mapear as propriedades já existentes

Consulte a [documentação](#) para saber quais propriedades são suportadas.

DATA BINDING



Problema/Dificuldade

Sincronização dados entre Views e Modelos - bidirecional

Solução “Primitiva”

Monitorar os eventos de modificação de dados em ambas as partes.

Atualizar as informações a partir desses eventos.

Problema: muito trabalhoso e propenso a erros!

Importante: continua sendo uma solução válida.

Entra o DataBinding

O que é?

Um mecanismo integrado ao Forms que permite a sincronização de elementos visuais e um modelo, **de forma declarativa**.

Pode ser configurado através do C# ou do XAML.

Xamarin.Forms inclui diversas extensões para facilitar a declaração e o uso.

Entra o DataBinding

Sintaxe:

```
<Label Text="{Binding Source=objeto, Path=propriedade}" />
```

Onde:

- objeto é uma referência para o objeto que contém as informações a serem vinculadas
- Propriedade é o nome da propriedade desse objeto que irá alimentar o Text desse label

BindingContext

A propriedade **BindingContext** está disponível em todos os objetos que podem ser vinculados a uma expressão de Binding.

= Qualquer Objeto que herde de **BindableObject**

Importante: VisualElement, classe base de qualquer elemento do Xamarin.Forms (Layouts, Views, Pages, etc) herda de **BindableObject**.

BindingContext

Ao atribuir a propriedade BindingContext de um elemento, esse contexto se torna disponível a todos os elementos filhos desse objeto.

```
thePage.BindingContext = new User();
```

```
<ContentPage x:Name="thePage">
    <StackLayout>
        <Label Text="{Binding Name}" />
        <Label Text="{Binding Email}" />
    </StackLayout>
</ContentPage>
```

INotifyPropertyChanged

Objetos que sirvam como fonte de Binding devem implementar a Interface **INotifyPropertyChanged**.

```
public class User : INotifyPropertyChanged
{
    string _name;
    public string Name {
        get => _name;
        set {
            _name = value;
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(Name)));
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;
}
```

Bindable Properties

As expressões de Binding podem ser associadas a propriedades que sejam declaradas nos elementos visuais como **BindableProperty**, isso é, uma propriedade que pode receber uma expressão de Binding.

As propriedades mais importantes de cada elemento visual são declaradas dessa forma.

É necessário verificar a documentação para saber quais são essas propriedades.

É possível declarar BindableProperties em seus próprios componentes!

Bindable Properties

Exemplo: Label

Properties

[AnchorX](#)

Gets or sets the X component of the center point for any transform, relative to the bounds of the element.

This is a bindable property.

(Inherited from [VisualElement](#))

[AnchorY](#)

Gets or sets the Y component of the center point for any transform, relative to the bounds of the element.

This is a bindable property.

(Inherited from [VisualElement](#))

[AutomationId](#)

Gets or sets a value that allows the automation framework to find and interact with this element.

(Inherited from [Element](#))

[BackgroundColor](#)

Gets or sets the color which will fill the background of a VisualElement.

This is a bindable property.

(Inherited from [VisualElement](#))

x:Reference

É possível realizar Binding entre propriedades dos elementos visuais de sua página, usando a extensão **x:Reference** do XAML.

```
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="BindingSamples.MainPage">
    <StackLayout Padding="10, 0">
        <Label Text="BINDING"
            FontSize="40"
            HorizontalOptions="Center"
            VerticalOptions="CenterAndExpand"
            Scale="{Binding Source={x:Reference slider},
                Path=Value}" />

        <Slider x:Name="slider"
            Minimum="-2"
            Maximum="2"
            VerticalOptions="CenterAndExpand" />
    </StackLayout>
</ContentPage>
```

TÓPICOS COMPLEMENTARES



Custom Renderers

- Mecanismo usado pelo Forms para implementar os controles nas plataformas nativas
- Podemos criar renderers para:
 - Personalizar o comportamento de controles existentes
 - Criar novos controles personalizados
- Documentação

Platform Effects

- Uma alternativa mais simples que os Custom Renderers para personalizar a renderização de um controle
- Um componente pode receber um ou mais efeitos
- O **Effect** participa da fase de renderização do componente, podendo personalizar suas características
- [Documentação](#)

Native Embedding

- O Forms permite que sejam incluídos ou declarados controles os controles próprios de uma plataforma
- Documentação

Native Forms

- Páginas do Form podem ser instanciadas em projetos nativos do Android ou iOS.
- Uma alternativa para desenvolvimento de um aplicativo “híbrido”, escrito parcialmente nativo e parcialmente via Forms.
- Documentação

Xamarin.Essentials

- Biblioteca oferecendo uma interface unificada para acessar recursos comuns as plataformas, como:
 - Acelerometro, Giroscópio e Bússola
 - Geolocalização e Geocodificação
 - Conectividade, Envio de Email e SMS
 - Entre outros
- Documentação

Xamarin.Essentials

- Biblioteca oferecendo uma interface unificada para acessar recursos comuns as plataformas, como:
 - Acelerometro, Giroscópio e Bússola
 - Geolocalização e Geocodificação
 - Conectividade, Envio de Email e SMS
 - Entre outros
- Documentação

NuGets Fundamentais

- Xamarin.Forms.GoogleMaps: Fork do projeto Forms.Maps original, usando o Google Maps no iOS em lugar dos mapas nativos da Apple.
- SkiaSharp: implementação .NET da biblioteca Skia para desenho e manipulação de gráficos 2D com alta performance. Compatível com Forms.
- ACR.UserDialogs: biblioteca Cross-Platform que possibilita chamar caixas de diálogo nativas de cada plataforma. Indispensável para trabalho com Forms.

NuGets Fundamentais

- PropertyChanged.Fody: um plugin da biblioteca Fody que implementa automaticamente a interface *INotifyPropertyChanged*, necessária para o DataBinding, durante o processo de compilação.
- EntityFramework.Core: versão portável da biblioteca Entity Framework da Microsoft, poderosa ferramenta de ORM para plataforma .NET. Permite a criação de modelos com persistência em SQLite.
- Videoaula na Xamarin.University

NuGets Fundamentais

- Flurl.Http: biblioteca extremamente versátil para composição de requests HTTP. Permite a criação de um modelo de comunicação com API's através de um código unificado.
- Polly: uma biblioteca para facilitar o tratamento de código que pode potencialmente falhar, como chamadas de rede ou gravação em disco. Funciona através de Policies, onde é possível facilmente determinar tentativas sucessivas ou Timeout.

QUESTÕES

OBRIGADO! 🙏
