

programação cross-platform com

# xamarin

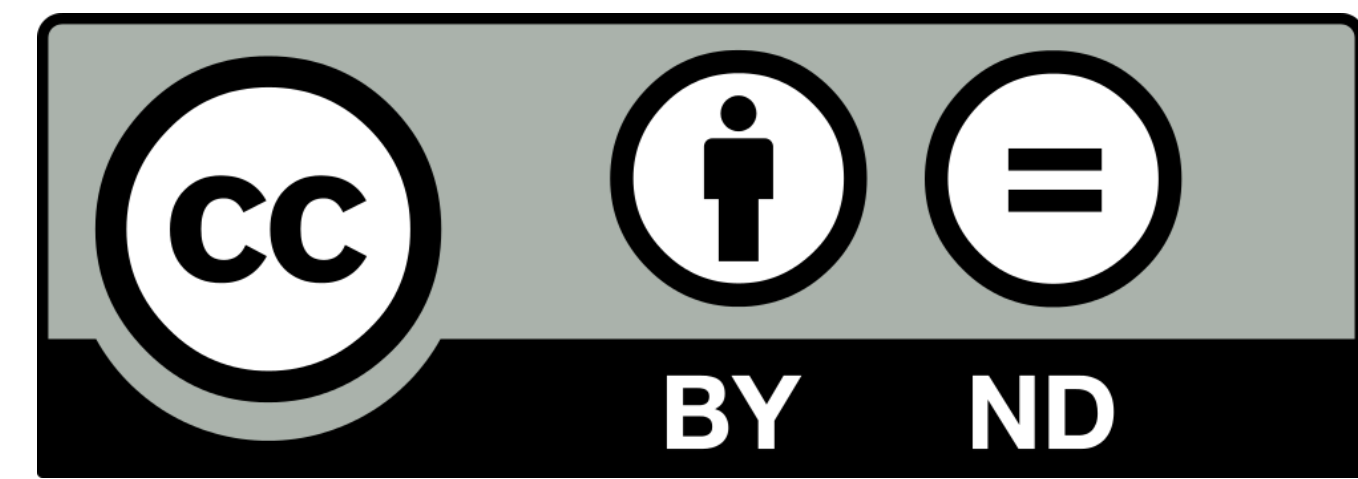


daniel ferreira  
@dfsdaniel

# Licença



- **Compartilhar:**
  - Você tem o direito de copiar e redistribuir o material em qualquer suporte ou formato para qualquer fim, mesmo que comercial.
- **Atribuição:**
  - Você deve dar o crédito apropriado, prover um link para a licença e indicar se mudanças foram feitas. Você deve fazê-lo em qualquer circunstância razoável, mas de maneira alguma que sugira ao licenciante a apoiar você ou o seu uso.
- **Sem derivações:**
  - Se você remixar, transformar ou criar a partir do material, você não pode distribuir o material modificado.
- **Licença CC BY-ND 4.0:**
  - [https://creativecommons.org/licenses/by-nd/4.0/deed.pt\\_BR](https://creativecommons.org/licenses/by-nd/4.0/deed.pt_BR)





# Platform Specific API

# DependencyService



- Permite utilizar a abordagem de “**classes paralelas**” em um projeto do tipo **Portable Class Library (PLC)**.
- Esta classe utiliza .NET Reflection para descobrir a classe específica em tempo de execução.
  - O único requisito é que a classe desejada precisa ser **pública**.
- Também é importante definir uma interface para que as classes nos projetos específicos implementem.
- O namespace e o nome da classe **não precisam ser os mesmos** nos projetos específicos.

# DependencyService



iOS

```
using System;
using UIKit;

[assembly: Dependency(typeof(PlatInfoSap2.iOS.PlatformInfo))]

namespace PlatInfoSap
{
    public class PlatformInfo : IPlatformInfo
    {
        UIDevice device = new UIDevice();

        public string GetModel()
        {
            return device.Model;
        }

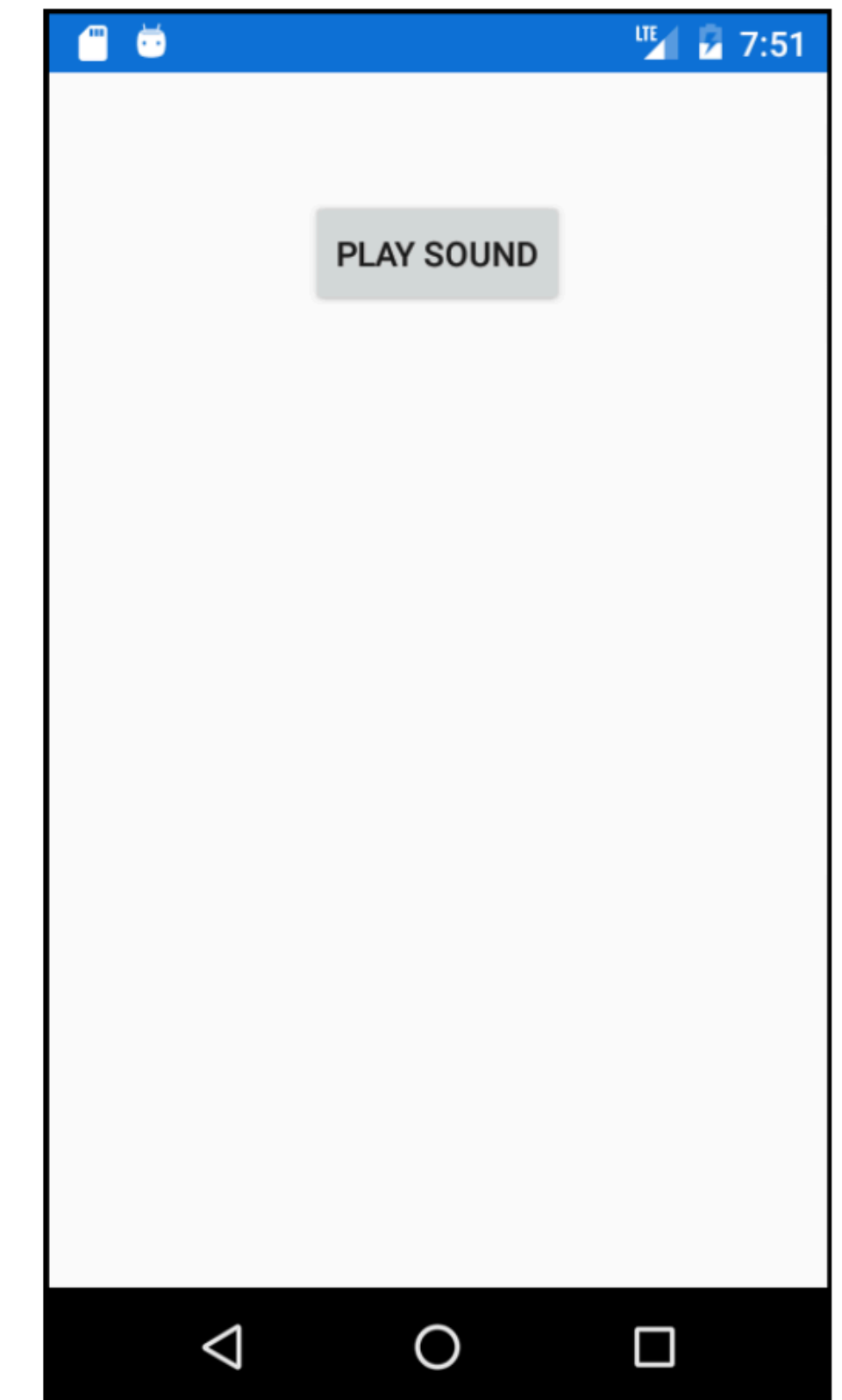
        public string GetVersion()
        {
            return String.Format("{0} {1}",
                                  device.SystemName,
                                  device.SystemVersion);
        }
    }
}
```

PLC

```
IPlatformInfo platformInfo = DependencyService.Get<IPlatformInfo>();
modelLabel.Text = platformInfo.GetModel();
versionLabel.Text = platformInfo.GetVersion();
```

# Exercício

- Utilizando o conceito de **DependencyService**, crie uma aplicação que possua um botão **“Play Sound”** e que ao tocar no botão, a música comece a ser executada.
- **Dica 1:** No Android, utilize a classe **MediaPlayer**.
- **Dica 2:** no iOS, utilize a classe **AVAudioPlayer**.





# Navegação



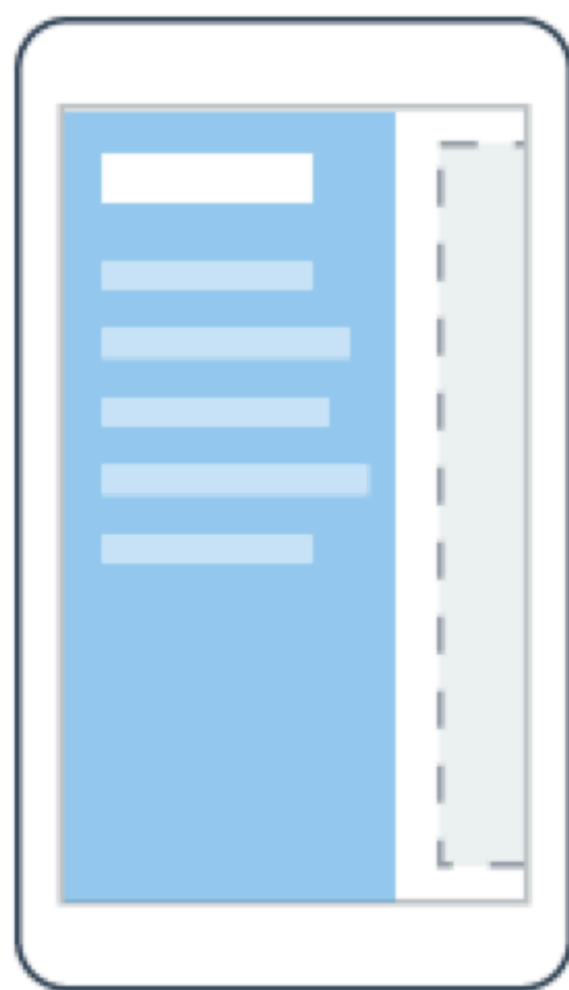
# Navegação



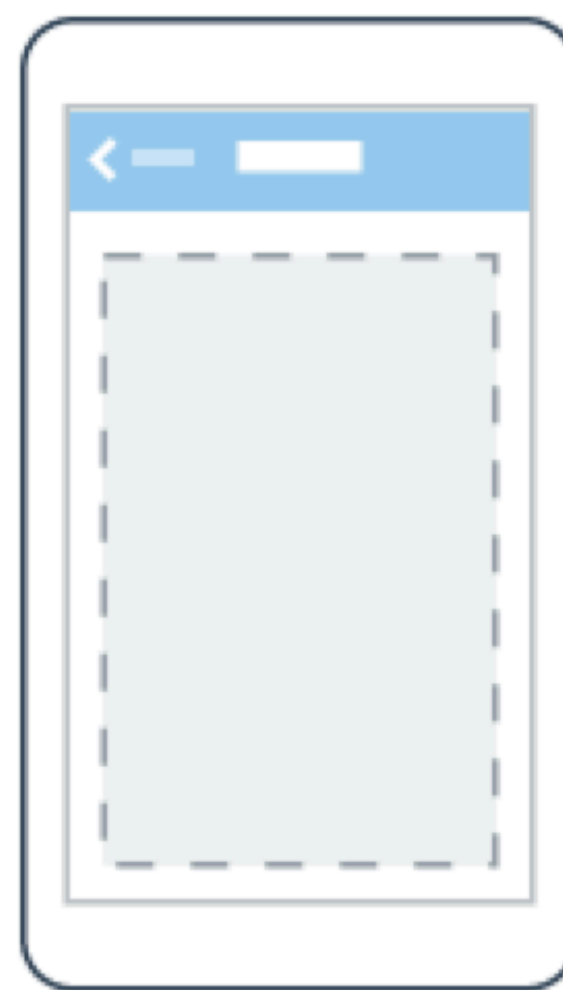
- É possível utilizar diversos modos de navegação para o usuário.
- Cada tipo de navegação utiliza um tipo de página diferente:



ContentPage



MasterDetailPage



NavigationPage



TabbedPage



CarouselPage



# Navegação



- **ContentPage:** Representa uma página (tela) da aplicação e é utilizada em conjunto com os outros tipos.
- **NavigationPage:** Permite realizar uma navegação hierárquica entre as páginas, onde o usuário pode avançar ou voltar nas páginas através de um mecanismo de FILO (First In / Last Out)
- **TabbedPage:** Página única com uma lista de tabs e uma área que exhibe o conteúdo de cada uma.
- **MasterDetailPage:** Gerencia 2 páginas com informações relacionadas. A "**master**" representa os itens e a "**detail**" representa os detalhes de cada item.
- **CarouselPage:** Usuário pode navegar através de várias páginas através do gesto "**swipe**", como se fosse uma galeria.
- **Modal Pages:** Suporte para que as páginas sejam abertas e esperam uma ação do usuário para ser completada. Só é possível voltar para a anterior quando a ação for completa.

# NavigationPage

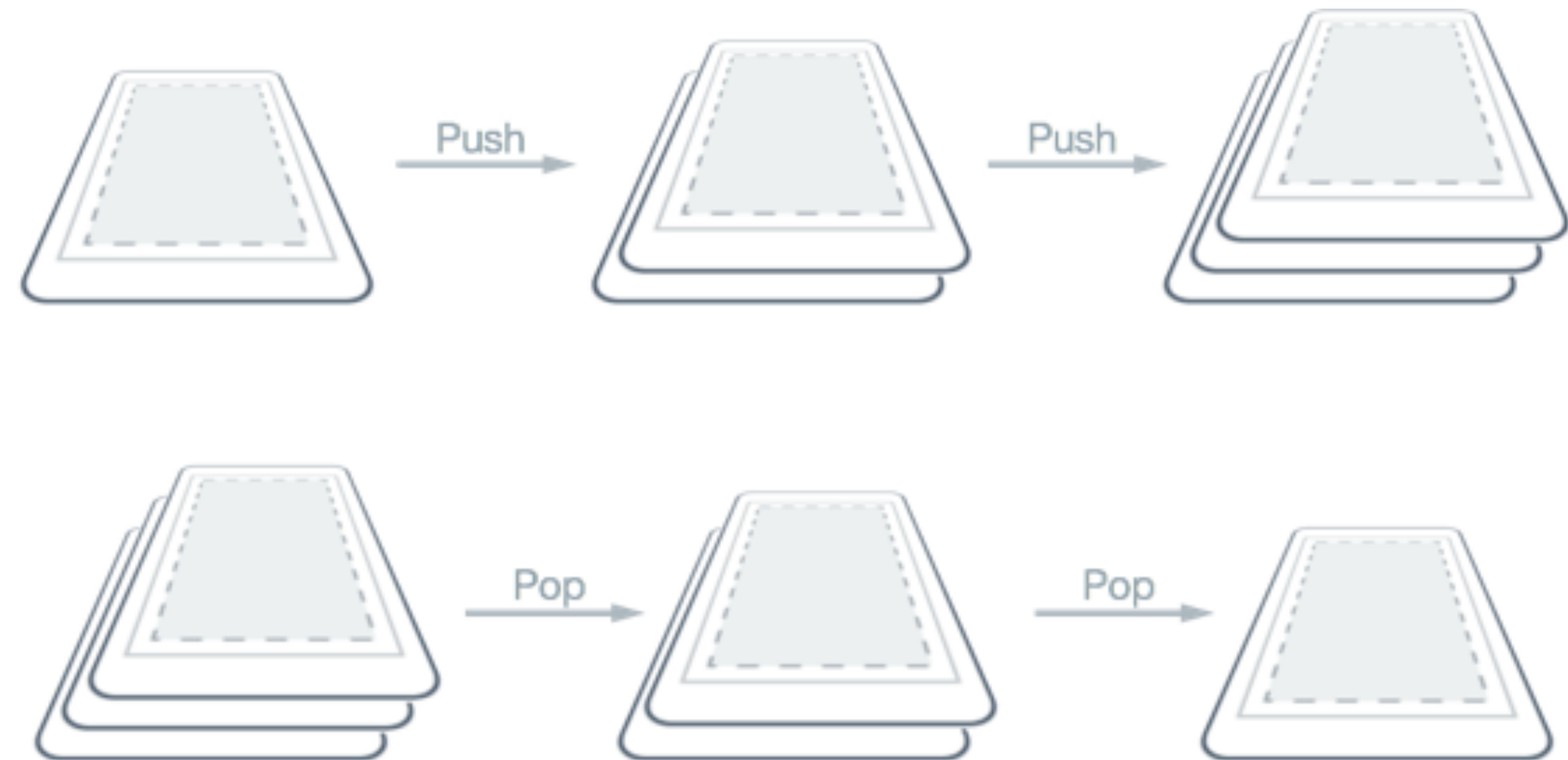


- Também conhecida como navegação hierárquica.
- As páginas são colocadas numa pilha de navegação.
- Recomendável utilizar apenas páginas do tipo **ContentPage**.
- Prove botões de navegação para o usuário (de acordo com a plataforma) e utiliza a propriedade **Title** da **ContentPage**.
- **Importante:** Precisa de uma página root.

```
public App()  
{  
    MainPage = new NavigationPage(new RootPage());  
}
```

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

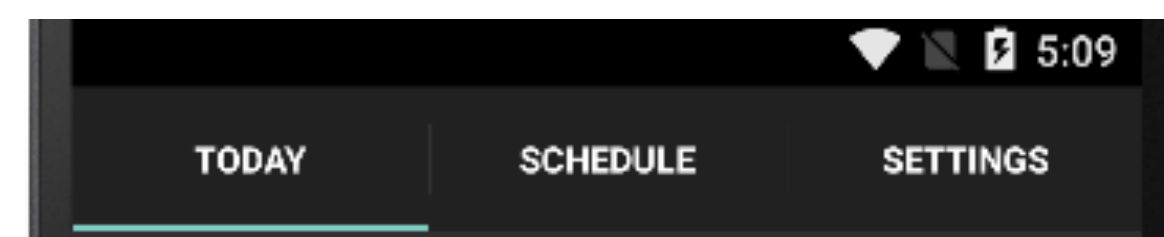
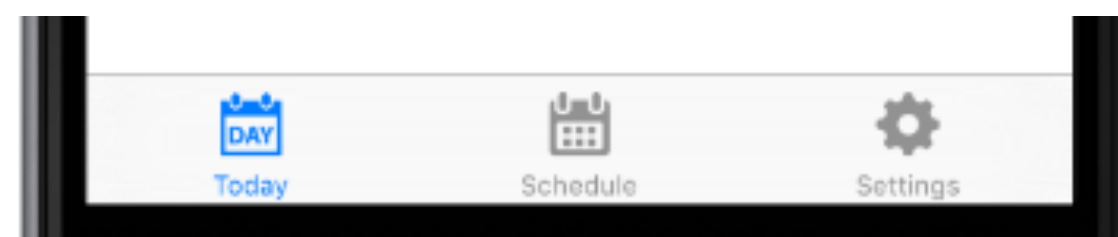
```
Navigation.PopAsync();
```



# TabPage



- Consiste numa lista de “**tabs**” e uma grande área de “**detail**”.
- Recomendável utilizar apenas **ContentPage** ou **NavigationPage**.
- O layout que será exibido depende de cada plataforma.
  - iOS exibe a lista de tabs na parte inferior com imagens transparentes 30x30 (normal), 60x60 (high) e 90x90 (plus). Exibe apenas 5 tabs e uma opção de “mais” caso possua.
  - Android exibe a lista de tabs na parte superior, exibe um scroll caso a quantidade de tabs não caiba na tela e os textos dos itens são automaticamente capitalizados.
  - Windows é semelhante ao Android, porém com os títulos convertidos para minúsculos.



# CarouselPage



- Usuários podem navegar entre o conteúdo de um lado pro outro, como uma galeria.
- A aparência é exatamente a mesma em qualquer plataforma.
- Aceita apenas conteúdo do tipo **ContentPage**.
- Como este component não utiliza **UI Virtualization**, é recomendável que cada página não possua muitos filhos internamente.

# MasterDetailPage

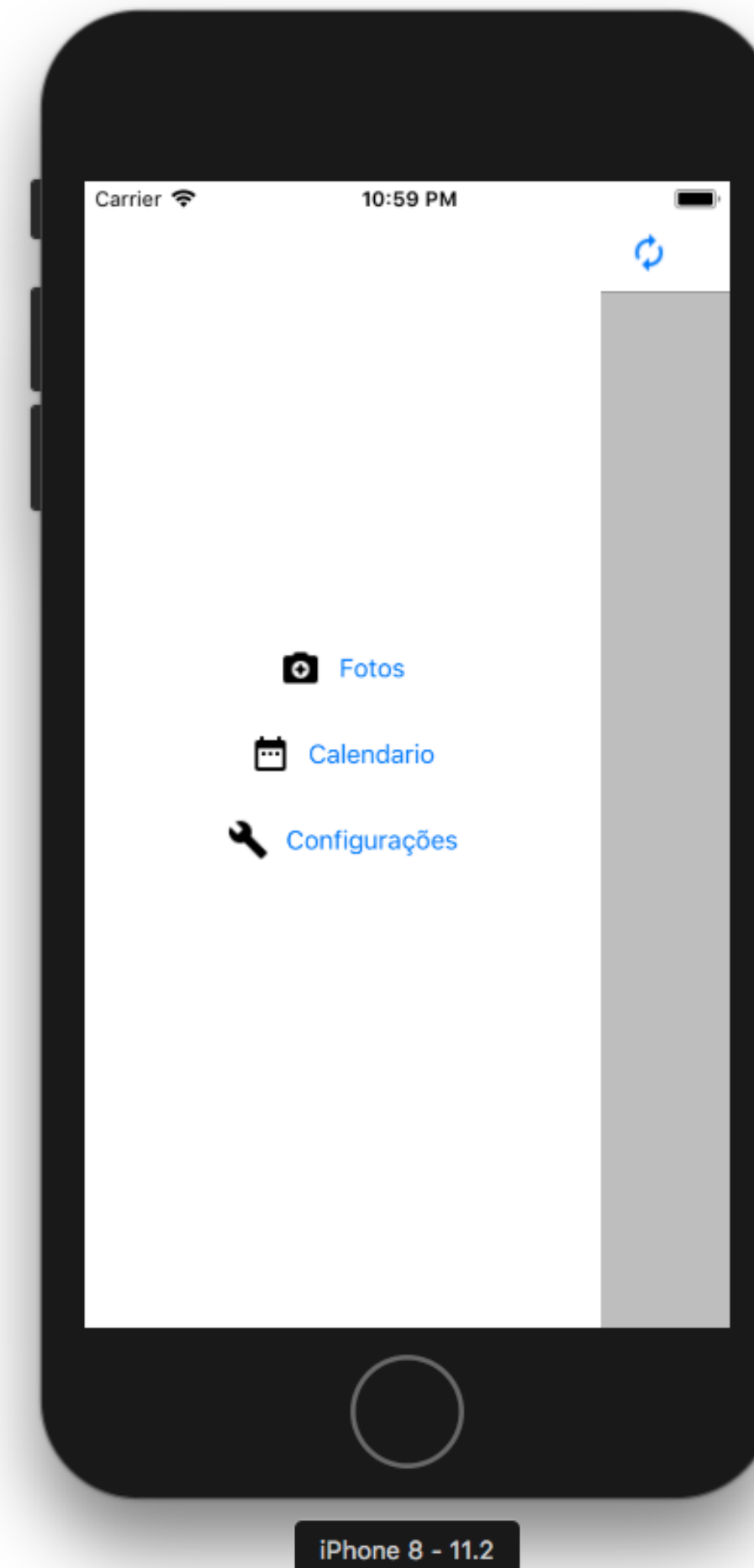


- Uma página que gerencia informações relacionadas entre **2 páginas**:
  - **Master**: Recomendável que seja **ContentPage**.
  - **Detail**: Pode ser **TabbedPage**, **NavigationPage** ou **ContentPage**.
- Inicialmente a página **detail** é exibida.
- Controle da exibição da master através da propriedade **IsPresented**.
  - **IsPresented**: true (exibe a master)
  - **IsPresented**: false (exibe a detail)
- Forma de exibição da **master** definida pela propriedade **MasterBehavior**:
  - **Default**: Usa o padrão da plataforma;
  - **Popover**: A página **detail** cobre parcialmente a **master**;
  - **Split**: A **master** é exibida na esquerda e a **detail** na direita.
  - **SplitOnLandscape**: A tela é dividida apenas quando está em modo **landscape**;
  - **SplitOnPortrait**: A tela é dividida apenas quando está em modo **portrait**.



# Exercício

- Crie uma aplicação do tipo **MasterDetailPage** que exibe 3 opções na “**master**” e que cada opção direcione para uma página diferente.
- Utilize **StackLayout** para exibir as 3 opções.





# Styles



# Styles



- Útil quando vários elementos possuem **o mesmo conjunto de propriedades**.
- **Style** é um único objeto com o um conjunto consolidado (do mesmo tipo de objeto) de propriedades.
- Evita **repetições**.
- Melhora **legibilidade** e **manutenabilidade** do código.
- Possuem esquema de **herança**.
- Podem ser utilizados no XAML e no code-behind.
- Geralmente são armazenados no **ResourceDictionary** (da página ou da aplicação).

# Criando Styles



- Como geralmente ficam em algum **ResourceDictionary**, necessário utilizar **x:Key** para identificar o style.
- Necessário informar o **TargetType**.
- Sua **ContentProperty** é a propriedade **Setters**, que recebe uma lista de objetos do tipo **Setter**.
  - Setter:
    - Property (do tipo BindableProperty)
    - Value (do tipo object)

```
<ContentPage.Resources>
  <ResourceDictionary>
    <Style x:Key="buttonStyle" TargetType="Button">
      <Setter Property="HorizontalOptions" Value="Center" />
      <Setter Property="VerticalOptions" Value="CenterAndExpand" />
      <Setter Property="BorderWidth" Value="3" />
      <Setter Property="TextColor" Value="Red" />
      <Setter Property="FontSize" Value="Large" />
    </Style>
  </ResourceDictionary>
</ContentPage.Resources>
```

# Styles com OnPlatform



- Também é possível utilizar **OnPlatform** dentro do Setter.
- Funciona apenas com a **sintaxe de elemento**.

```
<Setter Property="BackgroundColor">
  <Setter.Value>
    <OnPlatform x:TypeArguments="Color"
      Android="#404040" />
  </Setter.Value>
</Setter>
<Setter Property="BorderColor">
  <Setter.Value>
    <OnPlatform x:TypeArguments="Color"
      Android="White"
      WinPhone="Black" />
  </Setter.Value>
</Setter>
```

# Styles com Resources



- Se um valor de um **Setter** for um objeto complexo e/ou precisa ser reutilizado, é possível que esse objeto complexo também seja um resource dentro do **ResourceDictionary**.
- A propriedade **Value** do **Setter** irá utilizar este recurso compartilhado.

```
<ResourceDictionary>
  <Color x:Key="btnTextColor"
    x:FactoryMethod="FromHsla">
    <x:Arguments>
      <x:Double>0.83</x:Double>
      <x:Double>1</x:Double>
      <x:Double>0.75</x:Double>
      <x:Double>1</x:Double>
    </x:Arguments>
  </Color>

  <Style x:Key="buttonStyle" TargetType="Button">
    ...
    <Setter Property="TextColor" Value="{StaticResource btnTextColor}" />
    ...
  </Style>
</ResourceDictionary>
```



# Utilizando os Styles



- Para utilizar os styles criados, basta utilizar o markup extension **StaticResource**, como já visto anteriormente:

```
<Button Text="Carpe diem"  
        Style="{StaticResource buttonStyle}" />
```

- É possível **sobrescrever** o valor de uma propriedade utilizada no style. Uma propriedade com valor atribuído diretamente no elemento é chamada de **local setting** ou **manual setting**.
- O valor de uma **local setting** sempre terá precedência sobre os **styles**.

# Observações



- As propriedades e valores utilizados no **Setter** precisam ser do tipo **BindableProperty**.
- Isso significa que:
  - **Eventos** não podem ser utilizados com **Setter**
  - Propriedades **Content** ou **Children** não podem ser utilizados com **Setter**.

```
<Style x:Key="frameStyle" TargetType="Frame">
  <Setter Property="OutlineColor" Value="Accent" />
  <Setter Property="Content">
    <Setter.Value>
      <Label Text="Text in a Frame" />
    </Setter.Value>
  </Setter>
</Style>
```





# Herança e Styles



- A herança entre os tipos dos objetos podem ser utilizados nos styles.
- Ou seja, se desejar ter um style para a grande maioria dos elementos, é possível utilizar o **TargetType="View"**.
- Este style pode ser utilizado num **Button** ou num **Label** sem problemas!

```
<ContentPage.Resources>
  <ResourceDictionary>
    <Style x:Key="viewStyle" TargetType="View">
      <Setter Property="HorizontalOptions" Value="Center" />
      <Setter Property="VerticalOptions" Value="CenterAndExpand" />
      <Setter Property="BackgroundColor" Value="Pink" />
    </Style>
  </ResourceDictionary>
</ContentPage.Resources>
```

```
<StackLayout>
  <Button Text="Carpe diem"
    Style="{StaticResource viewStyle}" />

  <Label Text="A bit of text"
    Style="{StaticResource viewStyle}" />


  <Button Text="Sapere aude"
    Style="{StaticResource viewStyle}" />
</StackLayout>
```





# Herança e Styles




- Também é possível criar um **Style** baseado em outro **Style** existente.
- Utilizamos a propriedade **BasedOn**.
  - Utilize **StaticResource** para referenciar o style base.
- Só é possível se os styles tiverem **TargetType** compatíveis.



```
<Style x:Key="visualStyle" TargetType="VisualElement">
  <Setter Property="BackgroundColor"
    Value="{toolkit:HslColor H=0, S=1, L=0.8}" />
</Style>
```



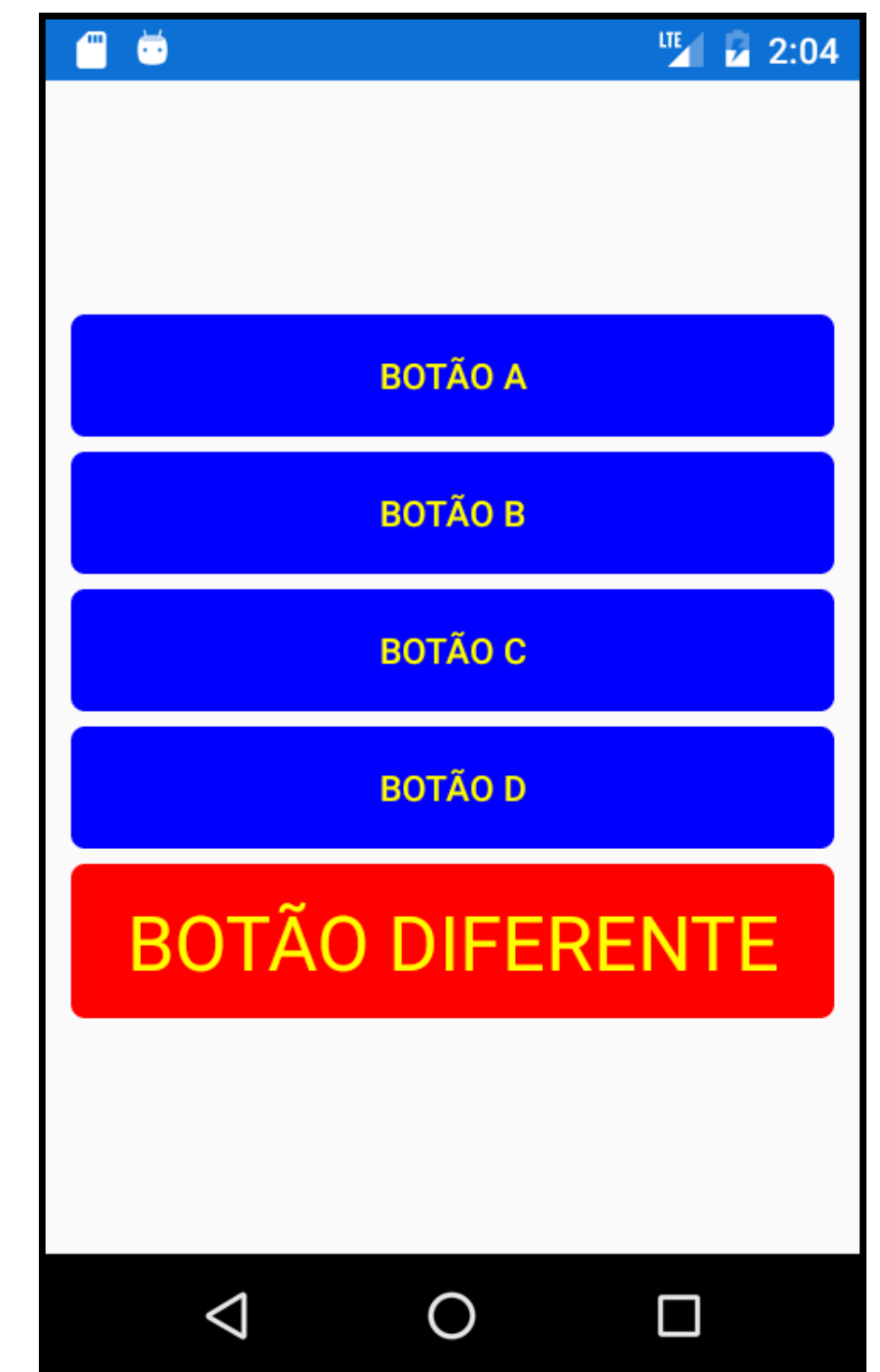
```
<Style x:Key="baseStyle" TargetType="View"
  BasedOn="{StaticResource visualStyle}">
  <Setter Property="HorizontalOptions" Value="Center" />
  <Setter Property="VerticalOptions" Value="CenterAndExpand" />
</Style>
```



```
<Style x:Key="labelStyle" TargetType="toolkit:AltLabel"
  BasedOn="{StaticResource baseStyle}">
  <Setter Property="TextColor" Value="Black" />
  <Setter Property="PointSize" Value="12" />
</Style>
```

# Exercício

- Utilizando **Styles**, crie uma aplicação que exiba página **4 botões idênticos**, com as seguintes propriedades:
  - Cor de fundo: azul
  - Cor do texto: amarelo
  - Largura da borda: 5
  - Largura: 300
- Adicione outro botão exatamente igual aos demais, porém com uma propriedade a mais:
  - Tamanho da Fonte: 30
  - Cor de fundo: vermelho





# Data Binding

# Data Binding

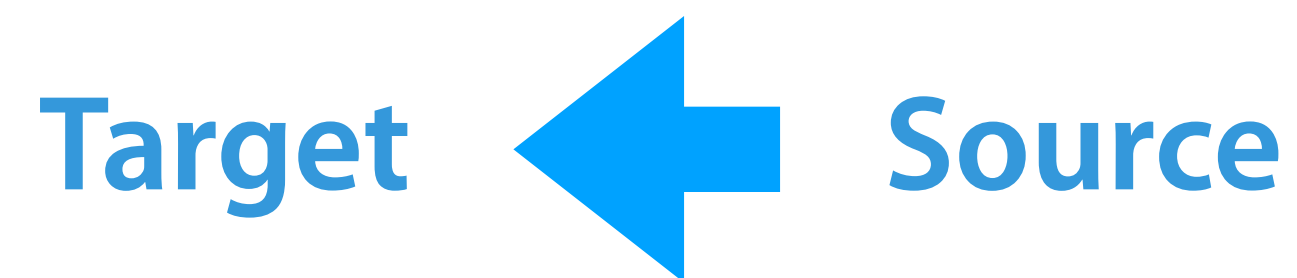


- Forma automatizada de **conectar valores de propriedades** entre objetos.
- Parte crucial do modelo **MVVM**.
- Envolve uma série de métodos, classes e interfaces:
  - **Binding**: Classe principal que define as características do binding.
  - **BindingContext**: Propriedade que recebe um **BindableObject**.
  - **SetBinding**: Método da classe **BindableObject** que realiza o binding.
  - **BindableObjectExtensions**: Classe que define alguns overloads para do método **SetBinding**.
  - **BindingExtension**: Classe que define um markup extension e permite realizar o binding no XAML.
  - **ReferenceExtension**: Classe que define um markup extension e referenciar outro elemento no XAML.
  - **INotifyPropertyChanged**: Interface que exige que uma notificação seja disparada quando alguma propriedade muda no objeto.
  - **IValueConverter**: Interface com métodos que permite a conversão de dados durante o binding.

# Data Binding



- Sempre envolve um **target** e um **source**.
- **Source**: Propriedade de algum objeto que será observada sua mudança de valores. Este objeto precisa emitir algum tipo de notificação quando esta propriedade for alterada (**INotifyPropertyChanged**).
- **Target**: Propriedade de algum objeto que vai receber o valor alterado. Precisa ser do tipo **BindableProperty**.



# Data Binding



```
Label label = new Label
{
    Text = "Opacity Binding Demo",
    HorizontalOptions = LayoutOptions.Center
};

Slider slider = new Slider
{
    VerticalOptions = LayoutOptions.CenterAndExpand
};

label.BindingContext = slider;
label.SetBinding(Label.OpacityProperty, "Value");
```

```
<Label Text="Opacity Binding Demo"
        HorizontalOptions="Center"
        BindingContext="{x:Reference Name=slider}"
        Opacity="{Binding Path=Value}" />

<Slider x:Name="slider"
        VerticalOptions="CenterAndExpand" />
```



slider.Value



# Utilizando apenas Binding



```
Binding binding = new Binding
{
    Source = slider,
    Path = "Value"
};
label.SetBinding(Label.OpacityProperty, binding);
```

A large blue arrow pointing downwards, indicating the transformation from the C# code to the XAML code.A large blue arrow pointing downwards, indicating the transformation from the C# code to the XAML code.

```
<Label Text="Binding Source Demo"
        HorizontalOptions="Center"
        Opacity="{Binding Source={x:Reference Name=slider},
                        Path=Value}" />

<Slider x:Name="slider"
        VerticalOptions="CenterAndExpand" />
```

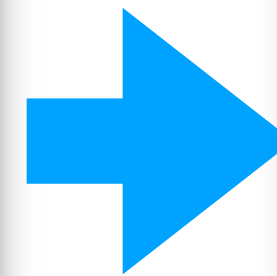


# Simplificando...



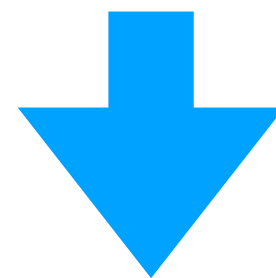
```
<Label Text="Binding Source Demo"
      HorizontalOptions="Center"
      Opacity="{Binding Source={x:Reference Name=slider},
                    Path=Value}" />

<Slider x:Name="slider"
        VerticalOptions="CenterAndExpand" />
```



```
<Label Text="Binding Source Demo"
      HorizontalOptions="Center"
      Opacity="{Binding Source={x:Reference slider},
                    Path=Value}" />

<Slider x:Name="slider"
        VerticalOptions="CenterAndExpand" />
```



```
<Label Text="Binding Source Demo"
      HorizontalOptions="Center"
      Opacity="{Binding Value, Source={x:Reference slider}}}" />

<Slider x:Name="slider"
        VerticalOptions="CenterAndExpand" />
```



# BindingContext



```
<Label Text="Opacity Binding Demo"
      HorizontalOptions="Center"
      BindingContext="{x:Reference Name=slider}"
      Opacity="{Binding Path=Value}" />

<Slider x:Name="slider"
      VerticalOptions="CenterAndExpand" />
```



```
<Label Text="Binding Source Demo"
      HorizontalOptions="Center"
      Opacity="{Binding Source={x:Reference Name=slider},
                        Path=Value}" />

<Slider x:Name="slider"
      VerticalOptions="CenterAndExpand" />
```

- **BindingContext** é visível para:
  - Todas as propriedades do elemento **target**.
  - Todos os elementos filhos do **target**.
- Se desejar fazer binding em diferentes propriedades, recomenda-se utilizar apenas a classe **Binding** (2ª opção).

# BindingContext



```
<Label Text="Binding Source Demo"
      HorizontalOptions="Center"
      BindingContext="{Binding Source={x:Reference Name=slider},
                          Path=Value}"
      Opacity="{Binding}" />
```

# Exercício



- Utilizando **data binding**, crie um aplicativo que funciona como um web browser. A aplicação deve possuir um botão voltar e avançar que devem estar desabilitado/habilitado de acordo com as informações do **WebView**.
- Dica 1:** Utilizar o controle **WebView**.
- Dica 2:** Utilizar **DataBindingContext**
- Dica 3:**

Info.plist

```
<key>NSAppTransportSecurity</key>
<dict>
  <key>NSAllowsArbitraryLoads</key>
  <true/>
</dict>
```





# Consumindo Webservices



# WebRequest



- **Classe abstrata** que permite fazer requisições a serviços web de forma assíncrona.
- Utiliza protocolo **RESTFUL**.
- Namespace: **System.NET**.
  - Classe base da **HttpWebRequest**.
- Permite consumir diversos tipos de resposta:
  - JSON
  - XML
- A requisição inicia numa **thread separada** da execução principal.
  - Necessário utilizar **Device.BeginInvokeOnMainThread()**

```
Uri uri = new Uri("https://developer.xamarin.com/demo/stock.json");  
request = WebRequest.Create(uri);  
request.BeginGetResponse(WebRequestCallback, null);
```

```
void WebRequestCallback(IAsyncResult result)  
{  
    Device.BeginInvokeOnMainThread(() => {  
        // altera elementos da thread principal.  
    });  
}
```

**BeginGetResponse:** Inicia a requisição assíncrona.

**EndGetReponse:** Recupera o resultado retornado.

# Consumindo Json



- Para fazer o parser de resultados **json** para nossas classes, utilizamos:
  - Classe **DataContractJsonSerializer**
  - Anotations **[DataContract]** e **[DataMember]**.

```
{
  photos:[
    "https://developer.xamarin.com/demo/IMG_0074.JPG",
    "https://developer.xamarin.com/demo/IMG_0078.JPG",
    "https://developer.xamarin.com/demo/IMG_0308.JPG",
    "https://developer.xamarin.com/demo/IMG_0437.JPG",
    "https://developer.xamarin.com/demo/IMG_0462.JPG",
    "https://developer.xamarin.com/demo/IMG_0475.JPG"
  ]
}
```

```
[DataContract]
class ImageList
{
    [DataMember(Name = "photos")]
    public List<string> Photos = null;
}
```




# Consumindo Json



```
void WebRequestCallback(IAsyncResult result)
{
    Device.BeginInvokeOnMainThread(() => {
        try {
            Stream stream = request.EndGetResponse(result).GetResponseStream();

            var jsonSerializer = new DataContractJsonSerializer(typeof(ImageList));
            imageList = (ImageList)jsonSerializer.ReadObject(stream);

            if (imageList.Photos.Count > 0)
                FetchPhoto();
        }
        catch (Exception exc) {
            filenameLabel.Text = exc.Message;
        }
    });
}
```



# Consumindo XML



```
void WebRequestCallback(IAsyncResult result)
{
    Device.BeginInvokeOnMainThread(() => {
        try {
            Stream stream = request.EndGetResponse(result).GetResponseStream();

            var serializer = new XmlSerializer(typeof(ImageList));
            imageList = (ImageList)serializer.Deserialize(stream);

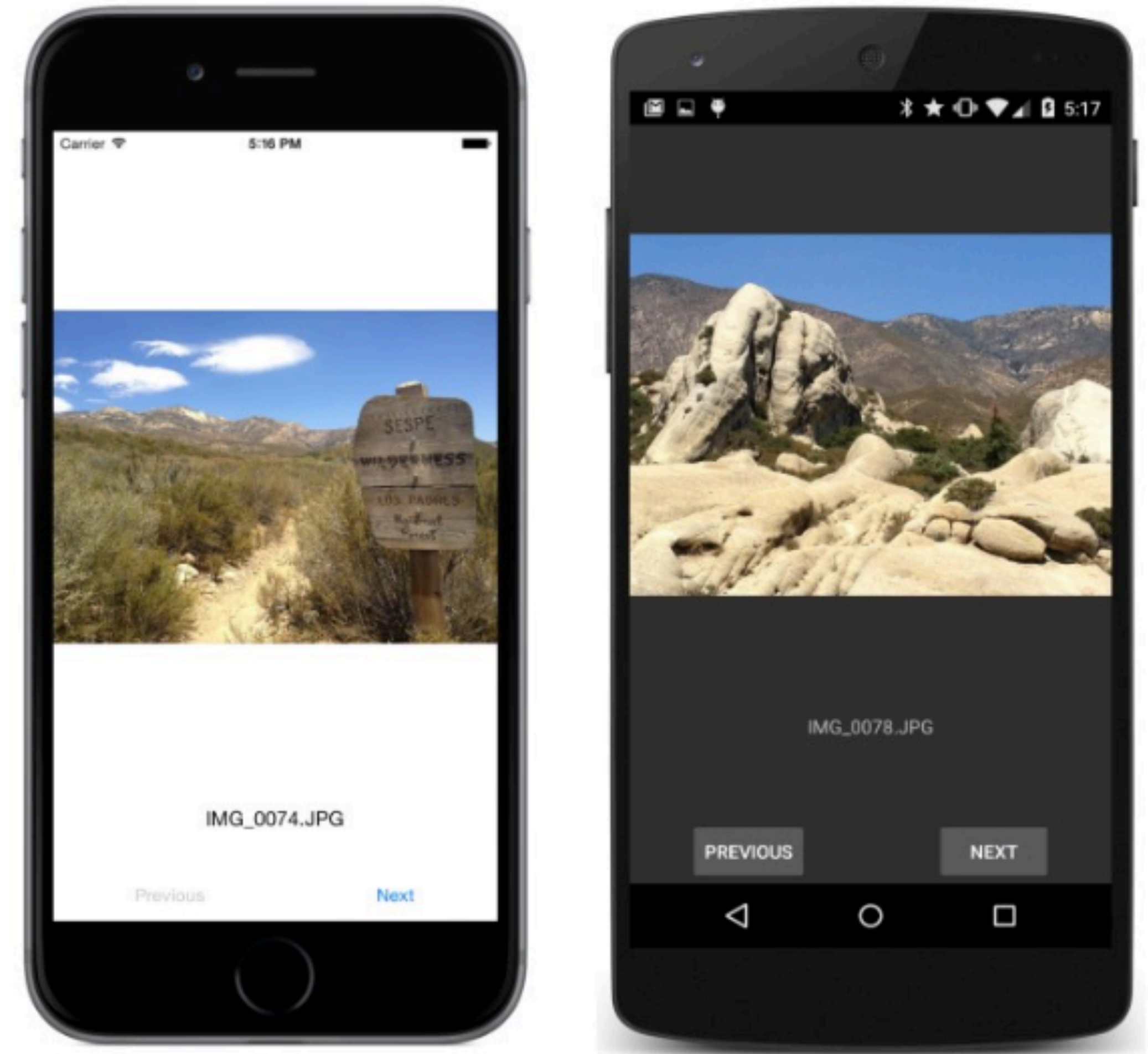
            if (imageList.Photos.Count > 0)
                FetchPhoto();
        }
        catch (Exception exc) {
            filenameLabel.Text = exc.Message;
        }
    });
}
```

```
[XmlRoot(ElementName = "feed")]
class ImageList
{
    [XmlElement(ElementName = "photos")]
    public List<string> Photos = null;
}
```

# Exercício



- Crie uma aplicação que exibe cada imagem retornada no exemplo anterior de forma separada.
- Tenha botões de avançar e voltar para visualizar as outras fotos.
- Os botões devem ser habilitados/desabilitados de acordo com a imagem exibida.
- Dica: Utilize a classe **UriImageSource**.





# Banco de Dados

# Banco de Dados



- O Xamarin disponibiliza acesso a dados em projetos PCL através do uso da biblioteca **SQLite**.
  - Instalação via **NuGet**.
  - Pacote **sqlite-net-pcl** (by Frank A. Krueger)
- O projeto específico de cada plataforma também precisa ter a biblioteca instalada.
- É possível utilizar **DependencyService** para recuperar o caminho correto do arquivo em cada plataforma.
  - Arquivos no formato **.db3**.



# SqlConnection



- Cria uma referência para o arquivo do banco de dados especificado no **path**.
- **Path** precisa ser específico de cada plataforma.
- Caso o arquivo não exista, ele é criado automaticamente.

```
SQLiteConnection database = new SQLiteConnection(dbPath);  
database.CreateTable<TodoItem>();
```



# Path Específico



Android

```
public string GetLocalFilePath(string filename)
{
    string path = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
    return Path.Combine(path, filename);
}
```

iOS

```
public string GetLocalFilePath(string filename)
{
    string docFolder = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
    string libFolder = Path.Combine(docFolder, "..", "Library", "Databases");

    if (!Directory.Exists(libFolder))
    {
        Directory.CreateDirectory(libFolder);
    }

    return Path.Combine(libFolder, filename);
}
```

# Database no App.cs



- Recomendável que a instância do **SqlConnection** fique no **App.cs**.
- Padrão **singleton**.
- Acessível por variável **publica** e **estática**.

```
static TodoItemDatabase database;

public static TodoItemDatabase Database
{
    get
    {
        if (database == null)
        {
            database = new TodoItemDatabase(
                DependencyService.Get<IFileHelper>().GetLocalFilePath("TodoSQLite.db3"));
        }
        return database;
    }
}
```

# Atributos das Tabelas



- **[PrimaryKey]**: Faz com que um **atributo int** seja a chave primária da tabela. Não suporta chaves primárias compostas.
- **[AutoIncrement]**: Faz com que um **atributo int** seja incrementado automaticamente (de 1 em 1).
- **[Column(name)]**: Faz com que a propriedade represente uma coluna na tabela. O parâmetro **name** é opcional.
- **[Table(name)]**: Com com que a classe represente uma tabela no banco de dados. O parâmetro **name** é opcional.
- **[MaxLength(value)]**: Restringe o tamanho de uma **propriedade texto** quando uma operação de **insert** ou **update** é executada.
- **[Ignore]**: Propriedade é ignorada no banco de dados. Útil para tipos que o SQLite não suporta.
- **[Unique]**: Garante que o valor dessa propriedade não se repetirá na tabela.

# Métodos



- **CreateTable<T>**: Cria uma tabela do tipo **T** no banco de dados.
- **Insert**: Adiciona um novo objeto ao banco de dados.
- **Get<T>**: Tenta recuperar um objeto do tipo **T** com a chave primária passada como argumento.
- **Table<T>**: Retorna todos os objetos do tipo **T**.
- **Delete**: Remove um objeto com a chave primária passada como argumento.
- **Query<T>**: Executa uma **query SQL** que retorna uma lista de objetos do tipo **T**.
- **Execute**: Executa comandos que não retornam resultados (como INSERT, UPDATE e DELETE).



# Projeto

# Definição



- Projeto com **tema livre**.
- Aplicação deve funcionar corretamente no **Android ou no iOS**.
  - Informar no e-mail qual escolheu.
- Projeto **individual**.
- Data limite para entrega: **04/11/2018**.
- Mandar para **dfs@cesar.org.br**.
  - Remover diretórios **bin** e **obj** antes de enviar.



# Critérios para nota



- Relevância / criatividade da aplicação;
- Escolher (informar no e-mail):
  - Consumo de web service;
  - Uso de banco de dados;
- Layouts das páginas utilizadas;
- Utilizar data bindings;
- Utilizar persistência de dados com o Application.Properties.
- Utilização de markup extensions;