

Laboratório

Olá Xamarin!

Versão: 1.0.0

Abril de 2017

Alexandre Zollinger Chohfi

@alexandrechohfi



CONTEÚDO**INTRODUÇÃO****EXERCÍCIO: PADRÃO MVVM**

Tarefa 1. Criar o projeto Xamarin.Forms.

Tarefa 2. Criar a organização de pastas do MVVM e criar uma classe base das ViewModels.

Tarefa 3. Criar uma ViewModel para a página principal e associar ela com a ViewModel.

Tarefa 4. Criar a View para a tela e criar os Bindings.

SUMÁRIO

Introdução

Agora que já entendemos como o Xamarin nos ajuda de forma simples a desenvolver nossos aplicativos mobile, vamos começar a organizar o nosso projeto na arquitetura MVVM.

Objetivos

Após a conclusão deste laboratório, os participantes serão capazes de:

- Criar uma aplicação simples Xamarin.Forms.
- Utilizar o padrão MVVM em seus aplicativos, com a separação lógica entre View, ViewModel e Model.

Requisitos

Para a realização deste laboratório é necessário o seguinte:

- Uma equipe de desenvolvimento com o sistema operacional Windows 10 e Visual Studio 2015 o 2017 Community, Professional ou Enterprise.

Tempo estimado para completar este laboratório: **60 minutos**.

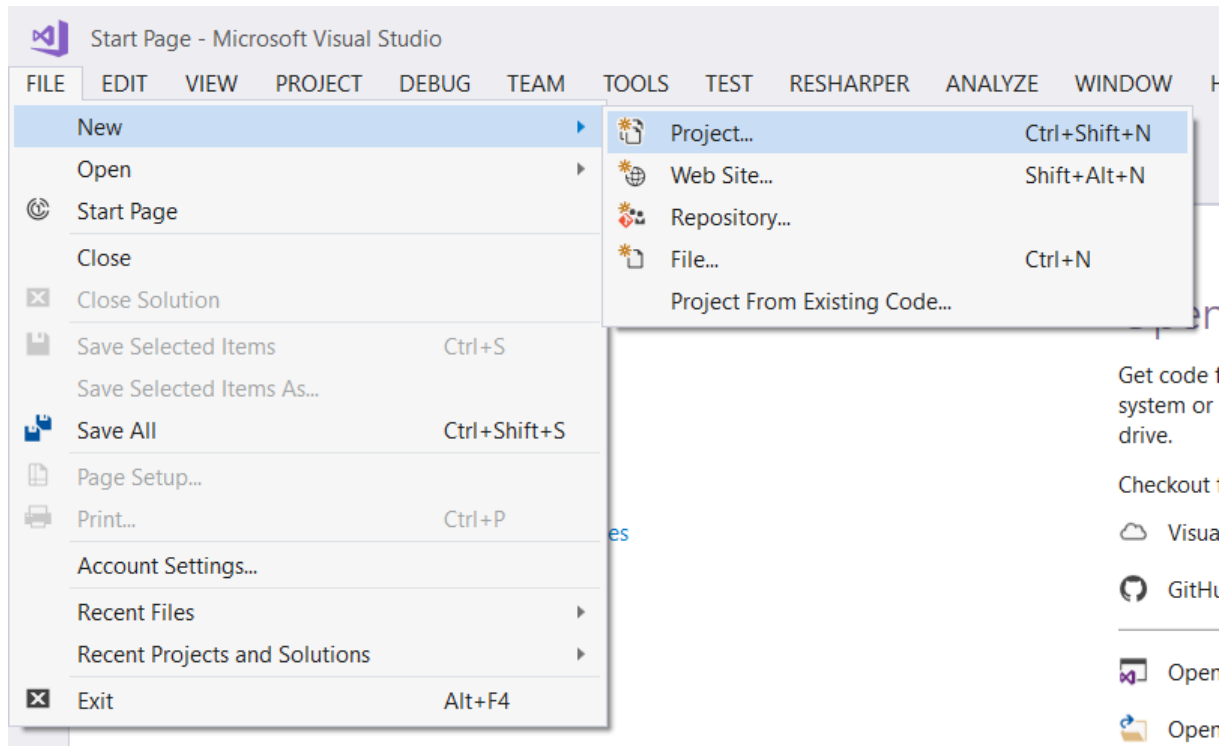
Exercício 1: Criando o projeto no Visual Studio 2017

Para criar um projeto no Visual Studio que dê suporte ao MVVM para Android, iOS e Window, existe um template perfeito para você!

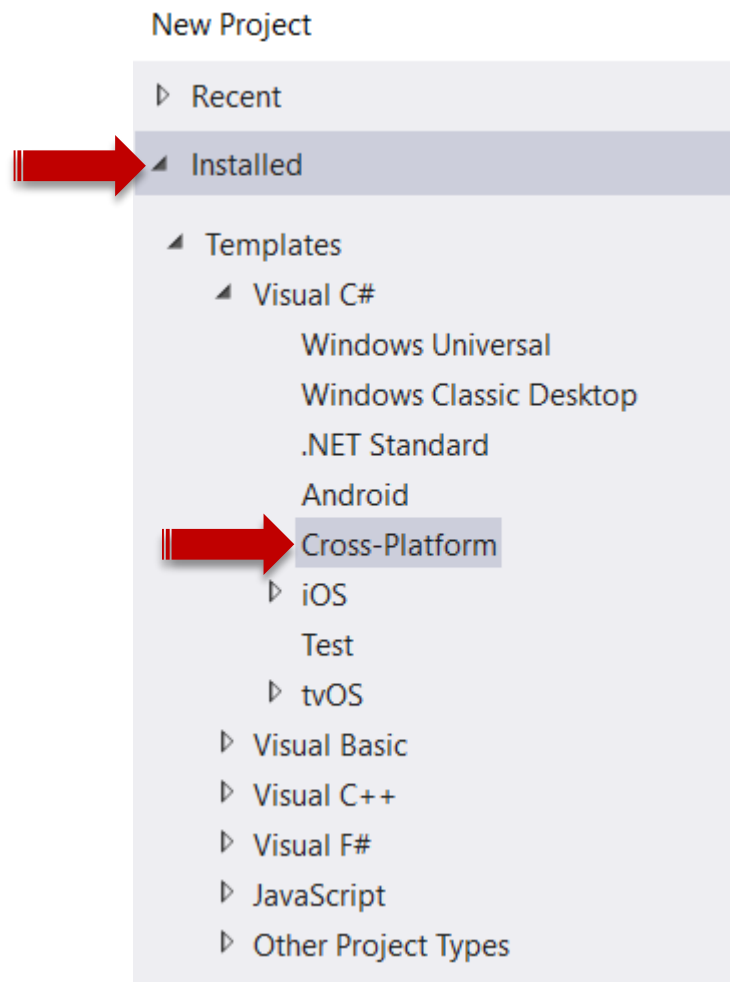
Tarefa 1. Criar o projeto Xamarin.Forms.

Execute os seguintes passos para criar um aplicativo Xamarin.Forms a partir do Visual Studio.

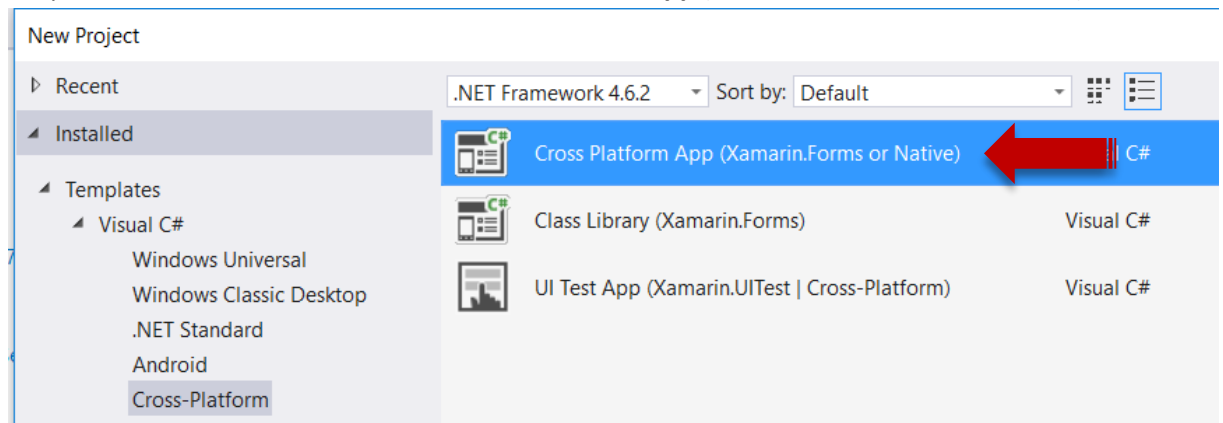
1. Selecione **File > New > Project** no Visual Studio.



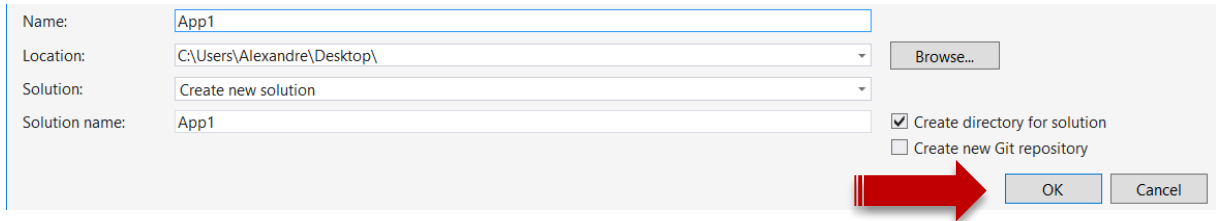
2. No painel esquerdo da janela **New Project** selecione **Visual C# > Cross-Platform** para indicar que deseja criar um aplicativo para multi-plataforma.



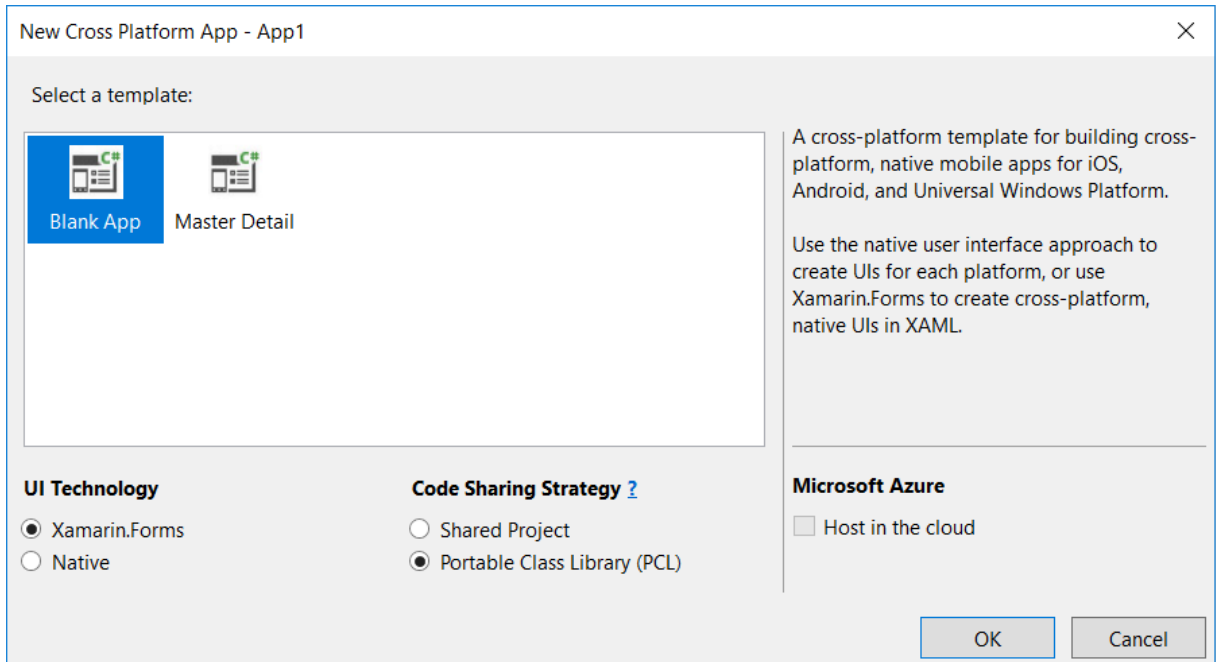
3. No painel direito, selecione o modelo **Cross Platform App (Xamarin.Forms or Native2019)**.



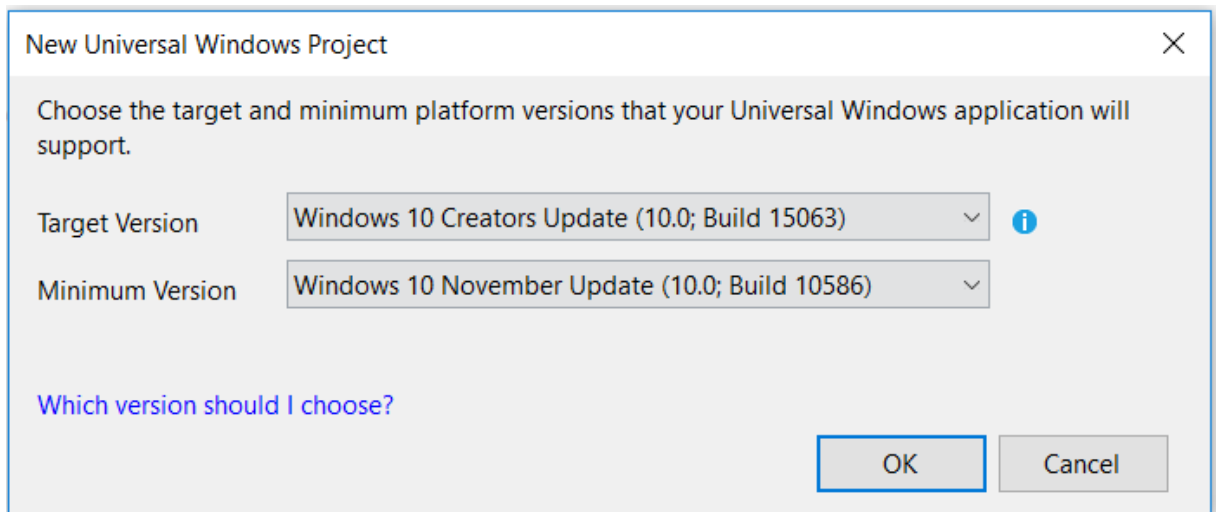
4. Forneça o nome, local e clique em **OK** para criar o projeto.



5. Selecione o tipo de **UI Technology** como **Xamarin.Forms** e a estratégia de compartilhamento de código(**Code Sharing Strategy**) como **Portable Class Library (PCL)**:



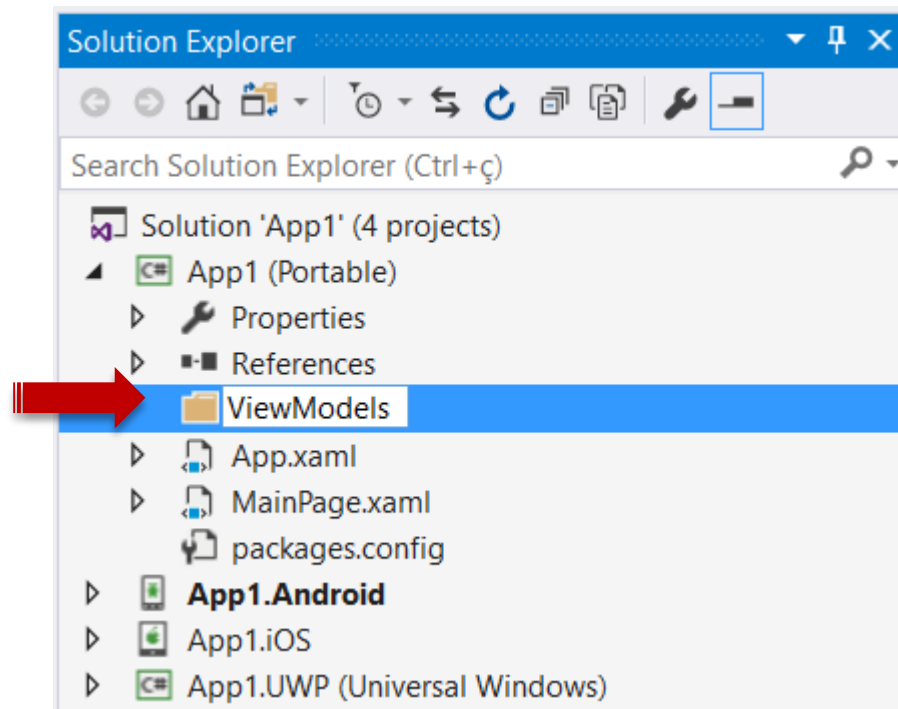
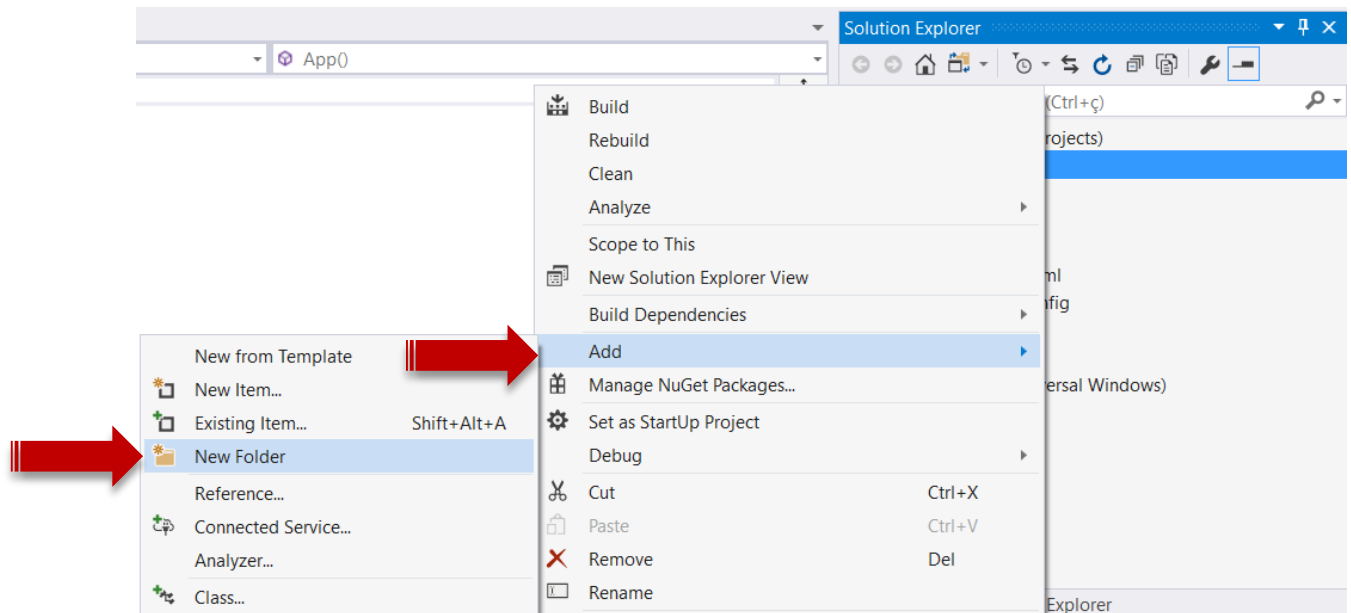
6. Se o seu projeto pedir uma versão do UWP, apenas confirme com a já pré-selecionada



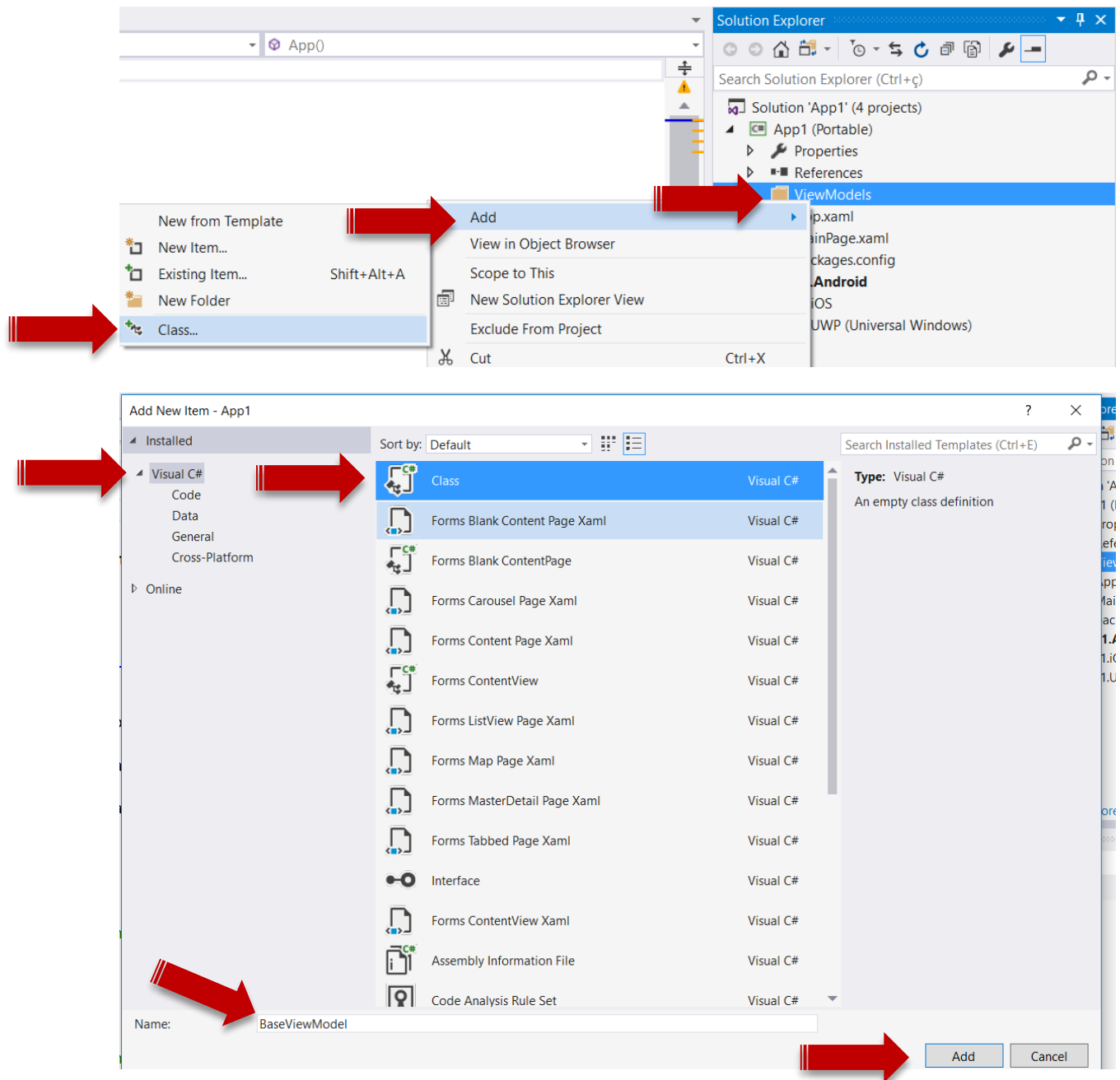
7. Aguarde a criação do projeto, ignorando qualquer instrução de conectar com o Mac.

Tarefa 2. Criar a organização de pastas do MVVM e criar uma classe base das ViewModels.

1. Crie uma pasta chamada ViewModels no seu projeto Portable.



2. Crie uma nova classe chamada **BaseViewModel** na nova pasta **ViewModels**.




```
BaseViewModel.cs  App.xaml.cs
C# App1
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace App1.ViewModels
8  {
9      class BaseViewModel
10     {
11     }
12 }
13
```

3. Deixe esta classe pública (**public**) e herdando de **INotifyPropertyChanged**.

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace App1.ViewModels
9  {
10     public class BaseViewModel : INotifyPropertyChanged
11     {
12         public event PropertyChangedEventHandler PropertyChanged;
13     }
14 }
15
```

Não se esqueça da linha 3, onde colocamos a diretiva **using** para o namespace **System.ComponentModel**.

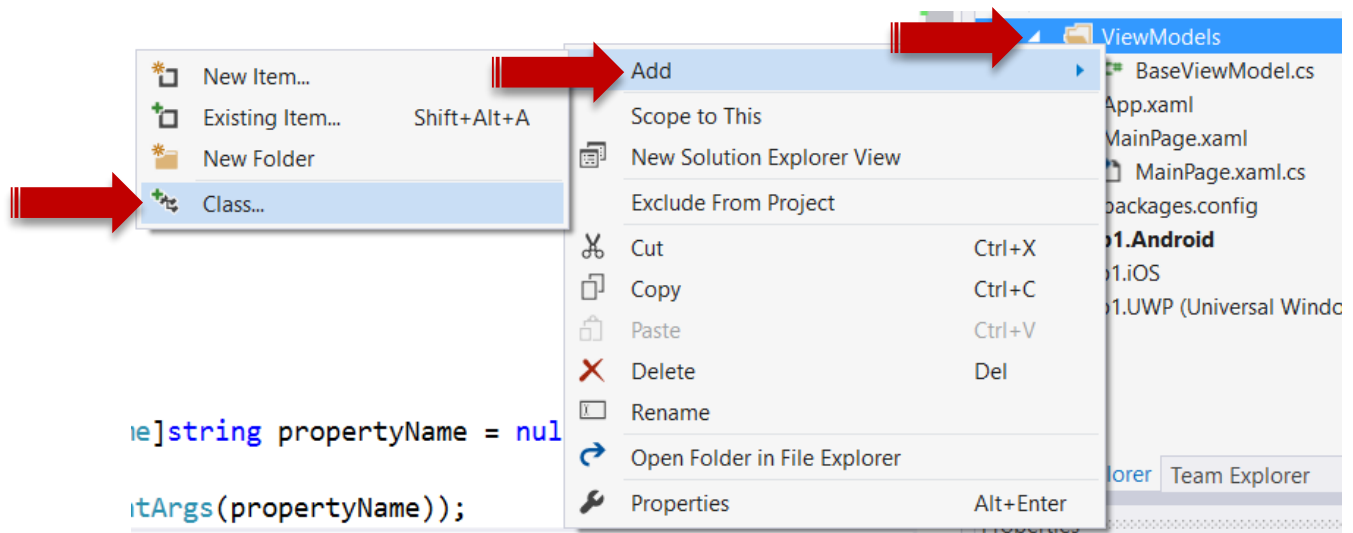
4. Adicione o método **OnPropertyChanged** dentro da classe **BaseViewModel**.

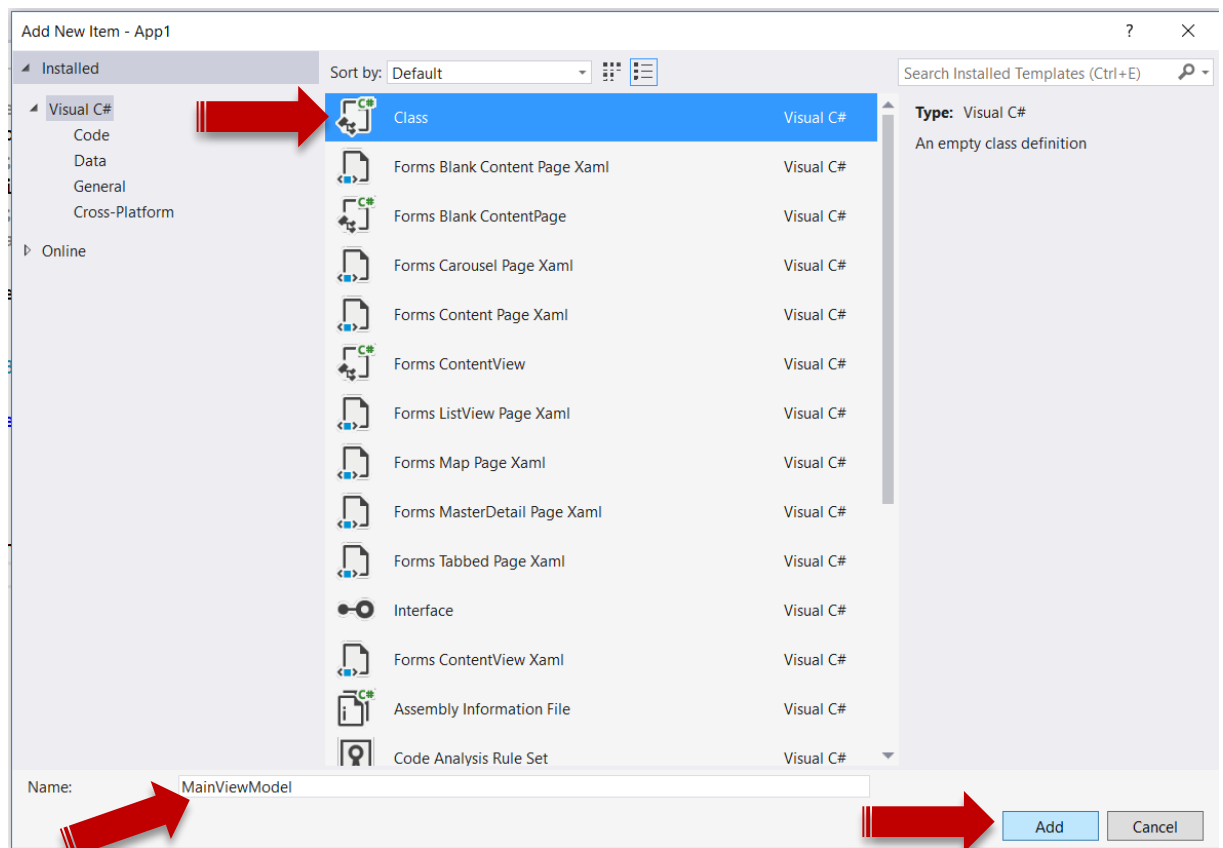
```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Linq;
5 using System.Runtime.CompilerServices;
6 using System.Text;
7 using System.Threading.Tasks;
8
9 namespace App1.ViewModels
10 {
11     0 references
12     public class BaseViewModel : INotifyPropertyChanged
13     {
14         public event PropertyChangedEventHandler PropertyChanged;
15
16         0 references
17         protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
18         {
19             PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
20         }
21     }
22 }
```

Novamente, não se esqueça de adicionar a diretiva **using**, como na linha 5, para o namespace **System.Runtime.CompilerServices**.

Tarefa 3. Criar uma **ViewModel** para a página principal e associar ela com a **ViewModel**.

1. Crie uma nova classe, também dentro da pasta **ViewModels**, chamada **MainViewModel**. Ela será a **ViewModel** da nossa página principal.







```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace App1.ViewModels
8  {
9      0 references
10     class MainViewModel
11     {
12     }
13 }
```


2. Faça esta nova classe herdar da classe que criamos antes, a `BaseViewModel`. Também deixe esta classe pública.

```
1  using System;
2      using System.Collections.Generic;
3      using System.Linq;
4      using System.Text;
5      using System.Threading.Tasks;
6
7  namespace App1.ViewModels
8  {
9      public class MainViewModel : BaseViewModel
10     {
11     }
12 }
13
```

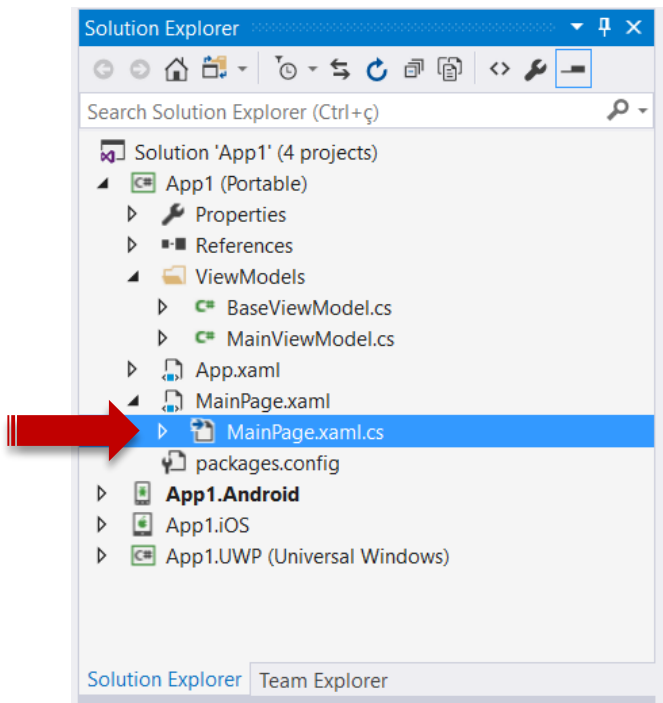


3. Agora a **MainViewModel** está pronta para criarmos nossas propriedades notificáveis para serem consumidas na nossa View. Vamos criar uma propriedade completa do tipo texto (string).

```
1  using System;
2      using System.Collections.Generic;
3      using System.Linq;
4      using System.Text;
5      using System.Threading.Tasks;
6
7  namespace App1.ViewModels
8  {
9      public class MainViewModel : BaseViewModel
10     {
11         private string _propriedadeTexto = "Texto inicial";
12         public string PropriedadeTexto
13         {
14             get { return _propriedadeTexto; }
15             set
16             {
17                 _propriedadeTexto = value;
18                 OnPropertyChanged();
19             }
20         }
21     }
22 }
23
```

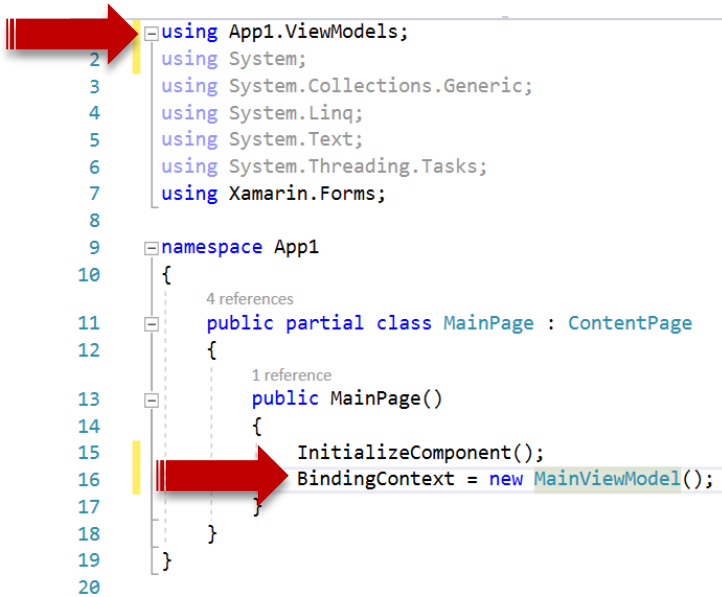


4. Agora vamos associar a ViewModel com a nossa View. Abra o arquivo **MainPage.xaml.cs**.



```
1  using System;
2      using System.Collections.Generic;
3      using System.Linq;
4      using System.Text;
5      using System.Threading.Tasks;
6      using Xamarin.Forms;
7
8  namespace App1
9  {
10     4 references
11     public partial class MainPage : ContentPage
12     {
13         1 reference
14         public MainPage()
15         {
16             InitializeComponent();
17         }
18     }
```

- Logo abaixo da chamada **InitializeComponent()**, instancie a classe **MainViewModel** e atribua esta instância à propriedade **BindingContext**.



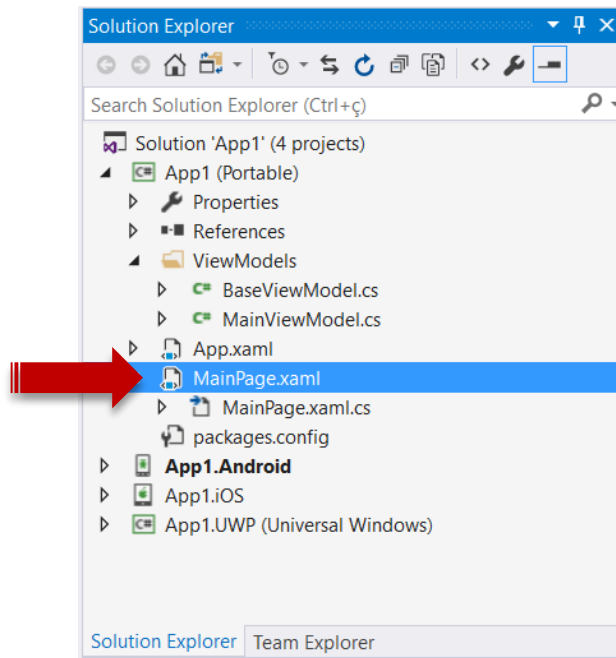
```
1 using App1.ViewModels;
2 using System;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Text;
6 using System.Threading.Tasks;
7 using Xamarin.Forms;
8
9 namespace App1
10 {
11     4 references
12     public partial class MainPage : ContentPage
13     {
14         1 reference
15         public MainPage()
16         {
17             InitializeComponent();
18             BindingContext = new MainViewModel();
19         }
20     }
```

Não se esqueça de adicionar a diretiva **using** para o namespace das suas ViewModels. Ele deve ser o nome do projeto, ponto, e a palavra “**ViewModels**”, ali na linha 1.

Agora a sua View e ViewModel já estão ligadas! Vamos finalmente fazer uma View bem simples que mapeia a nossa propriedade com Binding!

Tarefa 4. Criar a View para a tela e criar os Bindings.

- Abra o arquivo **MainPage.xaml**.

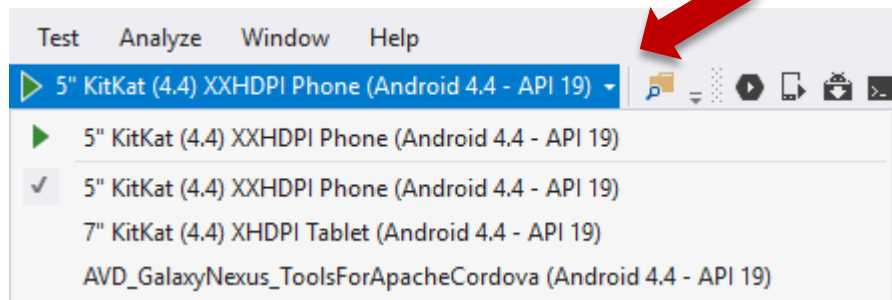


2. Altere o XAML atual para este aqui, que tem apenas dois elementos gráficos, um Label e um Editor:

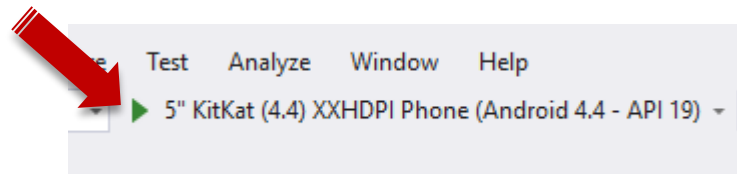
```
1      <?xml version="1.0" encoding="utf-8" ?>
2      <ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
3                  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
4                  xmlns:local="clr-namespace:App1"
5                  x:Class="App1.MainPage">
6          <StackLayout VerticalOptions="Center">
7              <Editor Text="{Binding PropriedadeTexto}" />
8              <Label Text="{Binding PropriedadeTexto}" />
9          </StackLayout>
10     </ContentPage>
```

Agora você já pode executar o seu projeto! Ele funciona tanto no Windows, quanto no iOS e no Android.

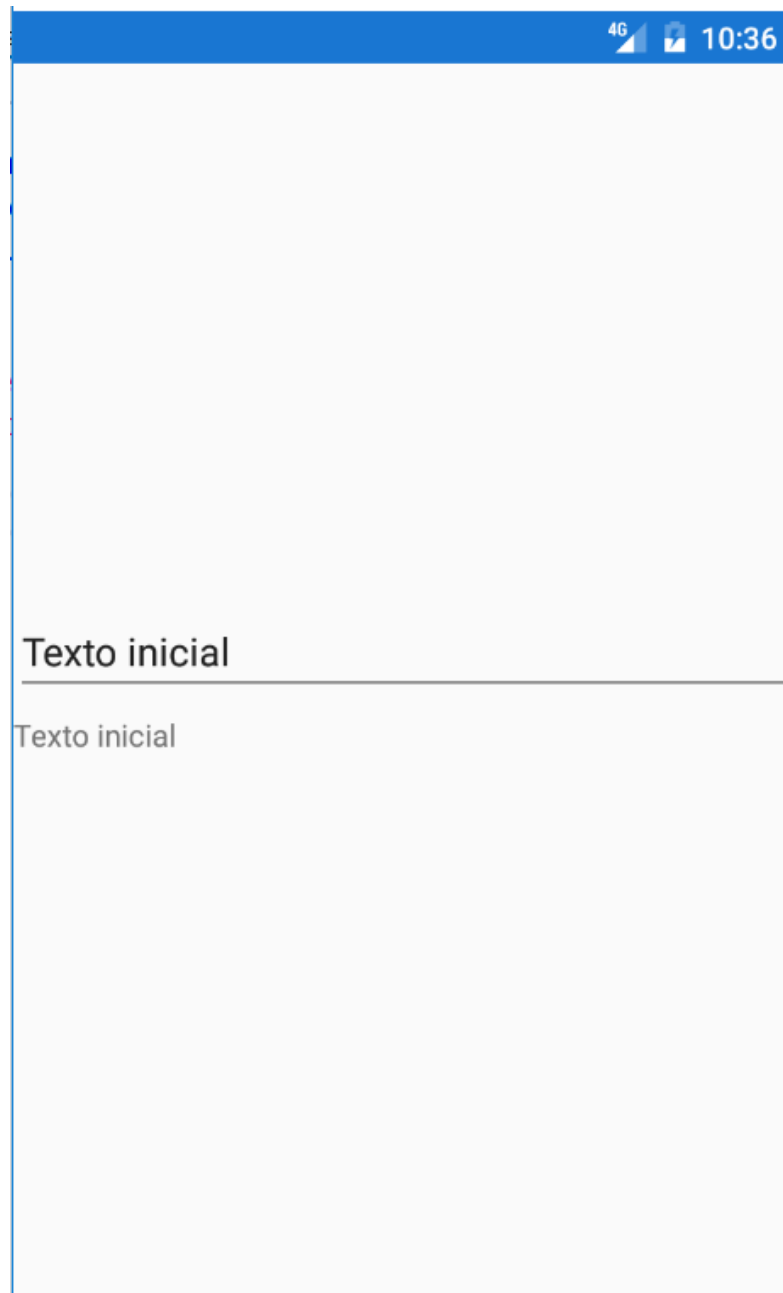
3. Vamos executar no Android, para ver como fica. Na lista de simuladores selecione uma opção onde queira implantar o aplicativo.



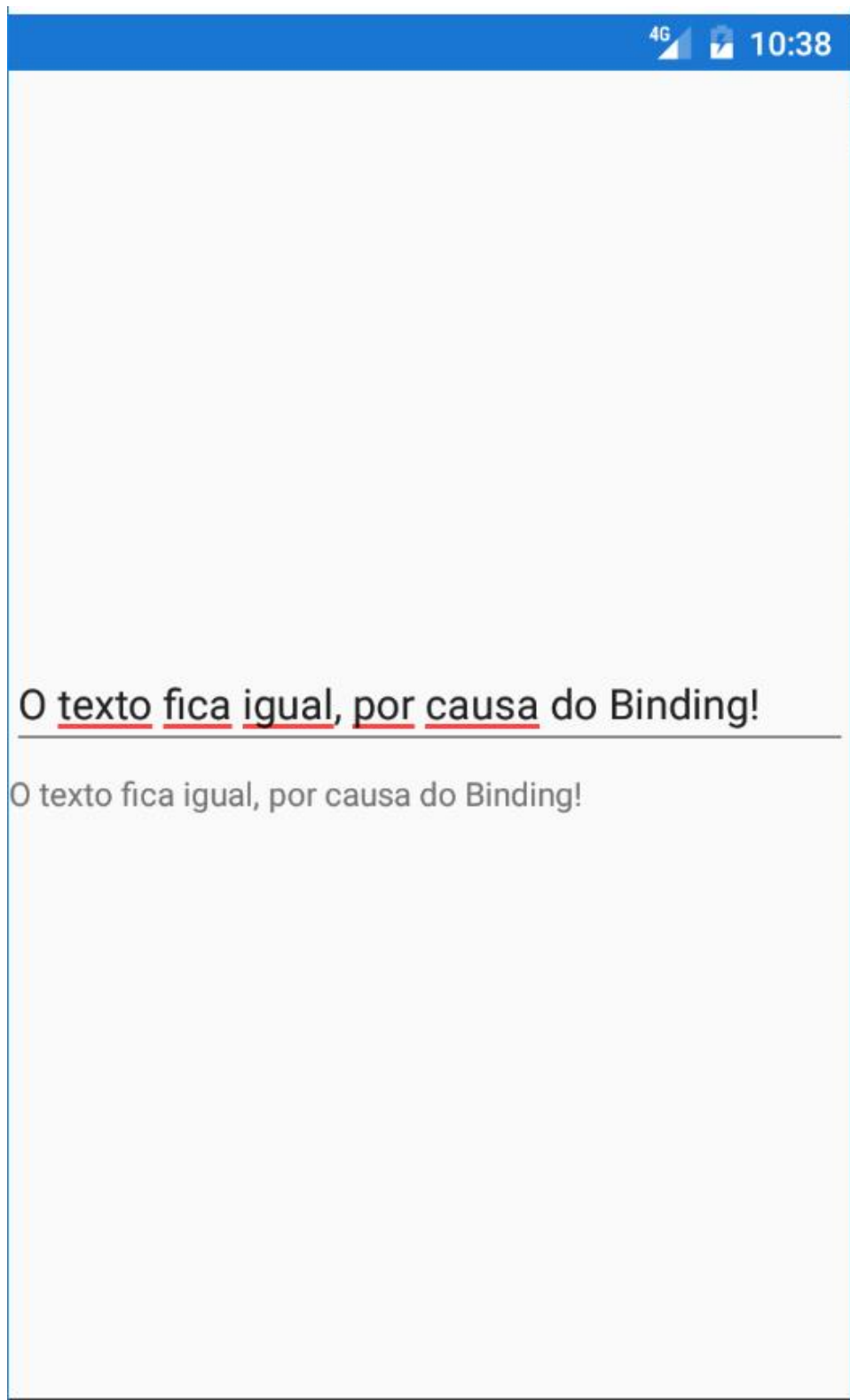
4. Clique em **Start** para implantar o aplicativo no simulador selecionado.



Se tudo funcionar corretamente, o aplicativo será exibido no simulador Android que você escolheu.



5. Altere o texto no campo de cima e veja como ele é alterado no label de baixo.



6. Volte ao Visual Studio e pare a execução.

Resumo

Neste laboratório, você criou algumas classes básicas para o funcionamento do seu app com MVVM.

Você também criou uma tela bem simples e viu como funciona o mecanismo de notificação do MVVM com Xamarin.Forms.

Agora você está entrando no mundo maravilhoso do desenvolvimento de apps com Xamarin.Forms!

Quando tiver terminado este laboratório publique a seguinte mensagem no Twitter e no Facebook:

Eu terminei o #Lab2 da #MaratonaXamarinbr Intermediária e estou entrando no mundo do MVVM!