

Laboratório

Olá Xamarin!

Versão: 1.0.0

Abril de 2017

Angelo Belchior

@angelobelchior



CONTEÚDO

INTRODUÇÃO

EXERCÍCIO: INTERFACE VISUAL COM XAML

Tarefa 1. Criar o projeto Xamarin.Forms.

Tarefa 2. Criar controles e aplicar estilos globais.

Tarefa 3. Criar Layouts usando o StackLayout e ScrollView.

Tarefa 4. Consumir um serviço do Azure para preencher uma ListView.

Tarefa 5. Criar uma ListView.

SUMÁRIO

Introdução

Já aprendemos o que é MVVM e como ele é importante no desenvolvimento de um app em Xamarin.Forms. Agora vamos tratar da letra V, a View! Vamos colocar em prática o que aprendemos no módulo 3!

Objetivos

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

- Criar uma aplicação simples Xamarin.Forms.
- Criar telas reaproveitando estilos.
- Organizar controles na Tela - Layouts
- Customizar células de uma ListView

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**.

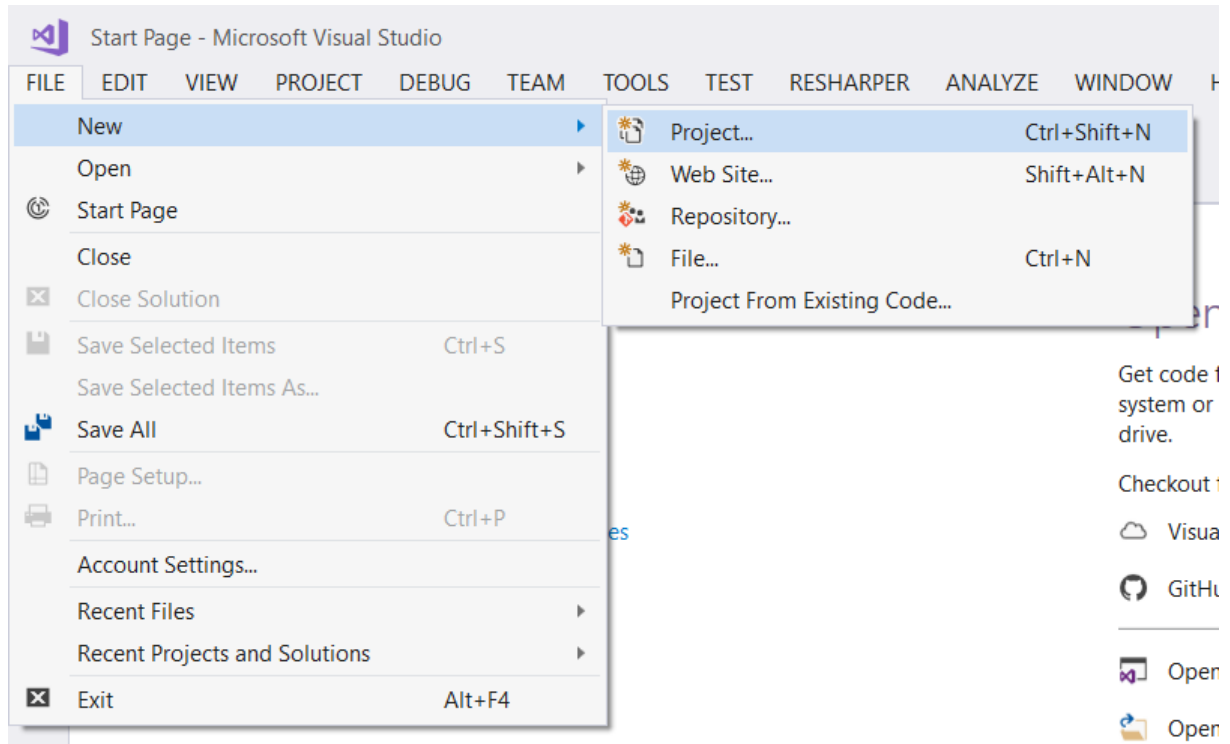
Exercício 1: Criando o projeto no Visual Studio 2017

Para criar um projeto Xamarin.Forms no Visual Studio vamos usar um dos seus Templates.

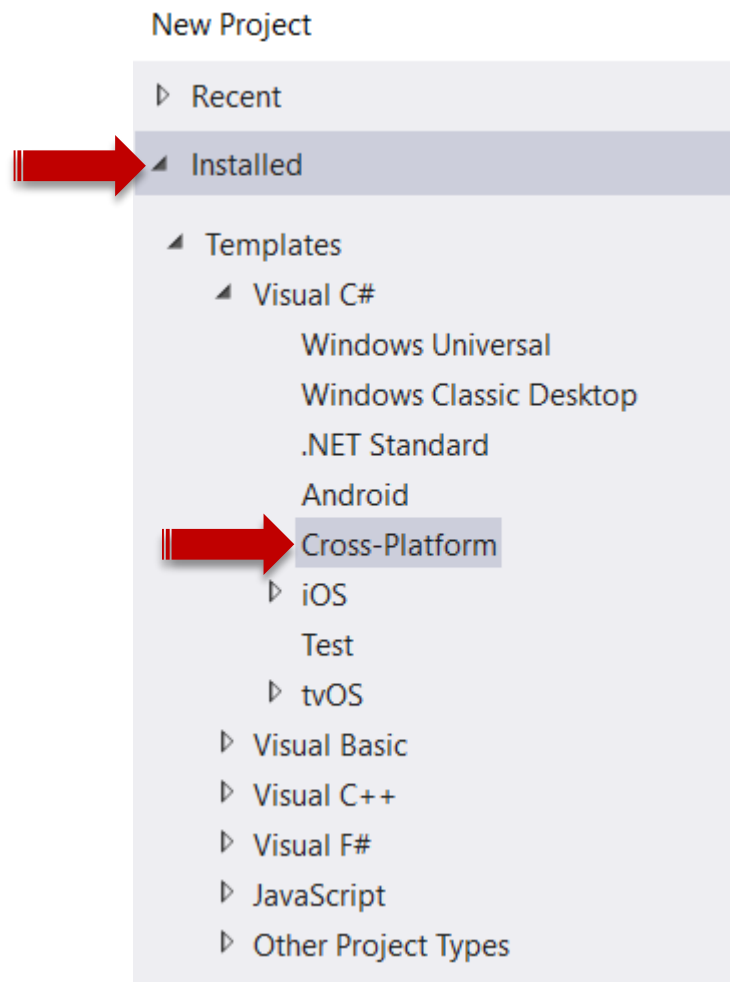
Tarefa 1. Criar o projeto Xamarin.Forms.

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

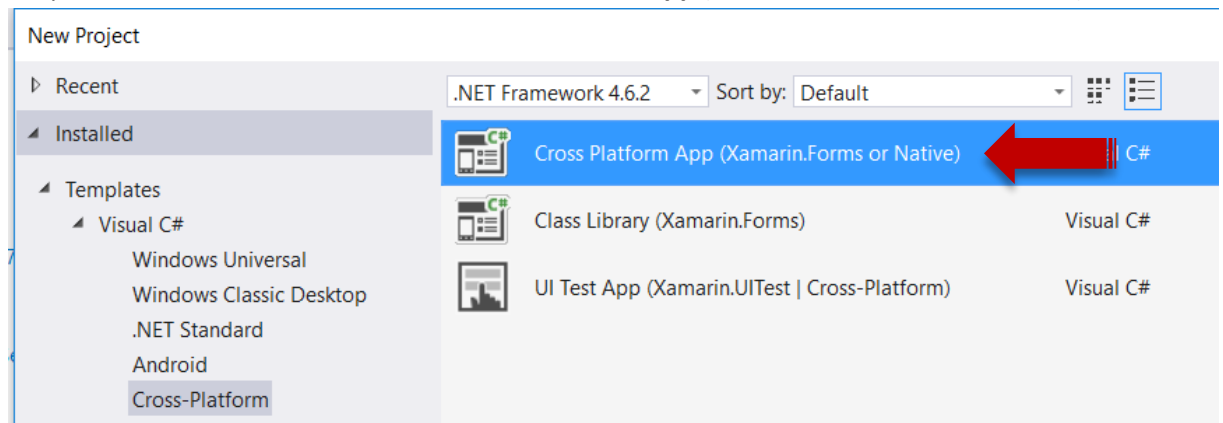
1. Selecione **File > New > Project** no Visual Studio.



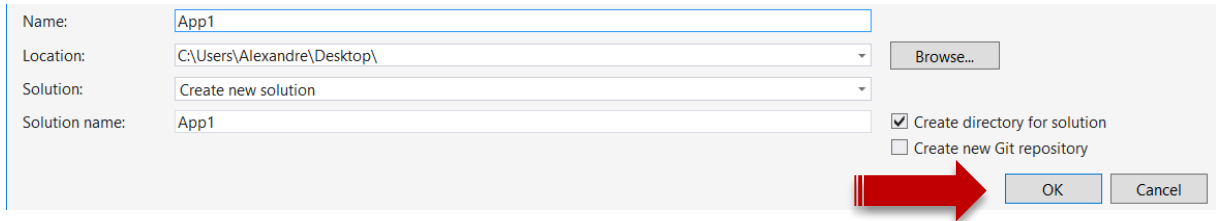
2. No painel esquerdo da janela **New Project** selecione **Visual C# > Cross-Platform** para indicar que deseja criar um aplicativo para multi-plataforma.



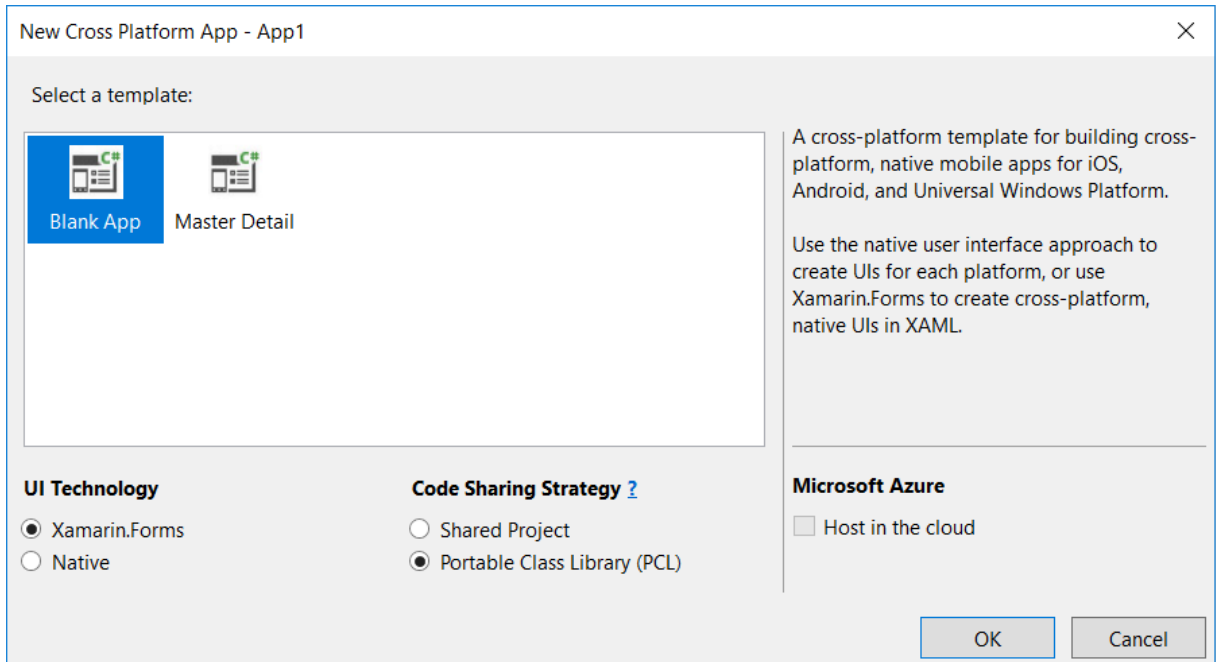
3. No painel direito, selecione o modelo **Cross Platform App (Xamarin.Forms or Native2019)**.



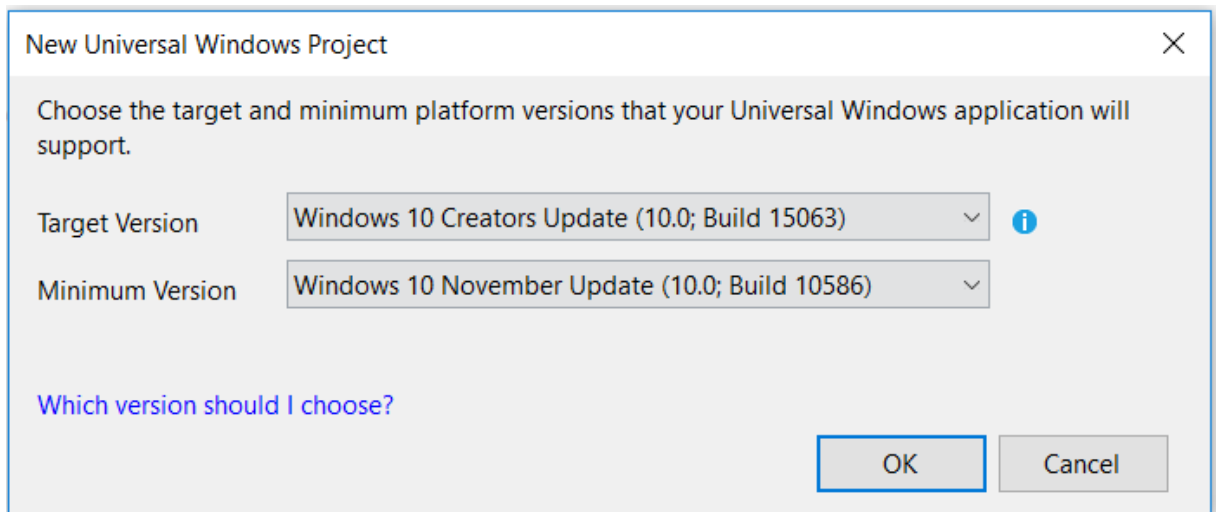
4. Forneça o nome, local e clique em **OK** para criar o projeto.



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)**:



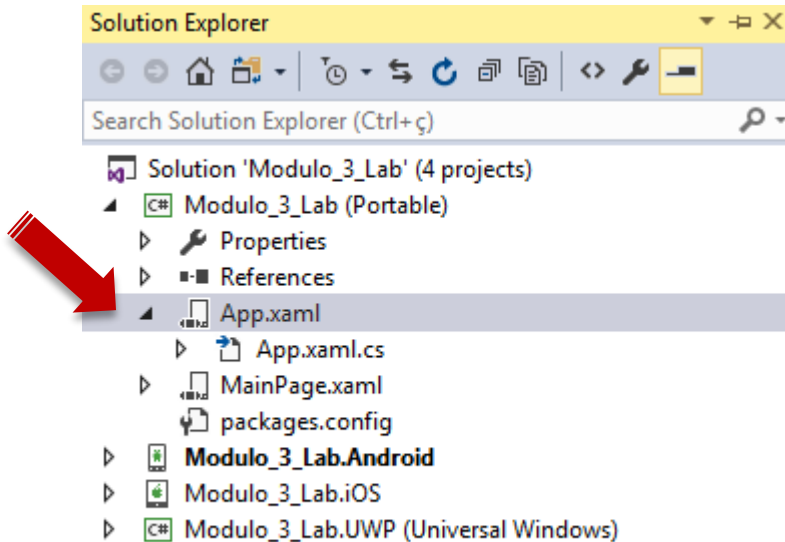
6. Se o seu projeto pedir uma versão do UWP, apenas confirme com a já pré-selecionada



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

Tarefa 2. Criar controles e aplicar estilos globais.

Selecione o arquivo App.xaml do projeto Portable. É nesse arquivo que vamos criar nossos estilos.



Dentro dele, vamos criar três variáveis de cor: “danger”, “warning”, “info”, “success”

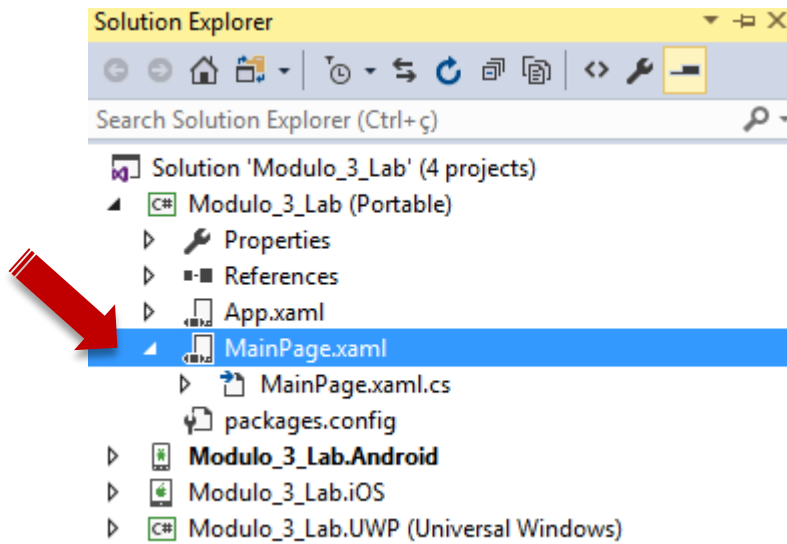
```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Modulo_3_Lab.App">
  <Application.Resources>

    <ResourceDictionary>
      <Color x:Key="danger">Red</Color>
      <Color x:Key="warning">Yellow</Color>
      <Color x:Key="info">Blue</Color>
      <Color x:Key="success">Green</Color>
    </ResourceDictionary>

  </Application.Resources>
</Application>
```

Todos recursos criados na class App.xaml estarão disponíveis para serem consumidos em qualquer parte do App!

Agora vamos aplicar essa variável na cor de texto de um label. Para isso selecione o arquivo MainPage.xaml:



Por padrão, o template do Xamarin.Forms já cria um label. Vamos muda-lo para receber uma cor de fonte vinda de um ResourceDictionary:

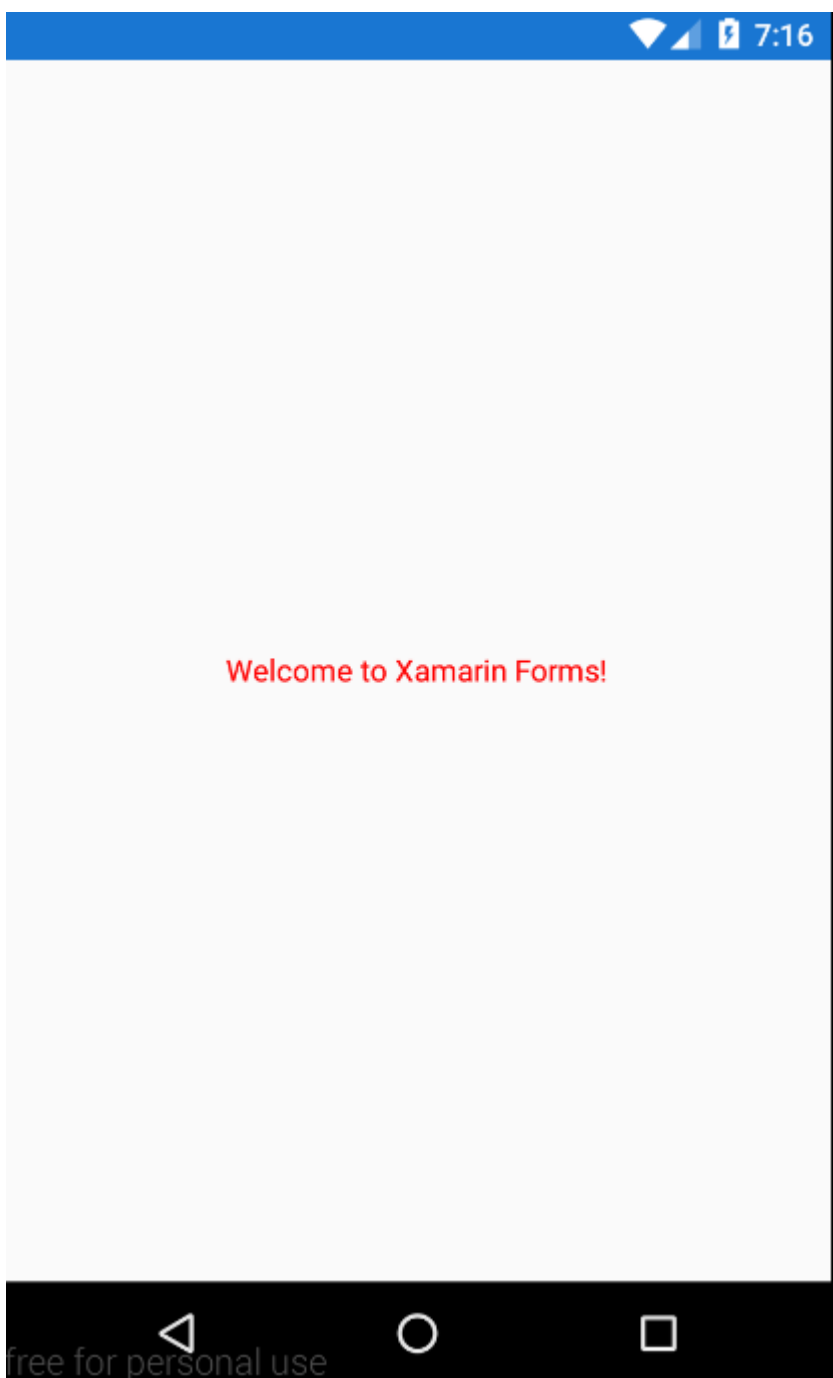
```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:Modulo_3_Lab"
             x:Class="Modulo_3_Lab.MainPage">

    <Label Text="Welcome to Xamarin Forms!"
           VerticalOptions="Center"
           HorizontalOptions="Center"
           TextColor="{StaticResource danger}"
           />

</ContentPage>
```

Execute o app!

Resultado:



Note que o ResourceDictionary “danger” é uma cor, sendo assim, podemos fazer um Binding para qualquer propriedade que aceite um tipo Color!

Tarefa 3. Criar Layouts usando o StatckLayout e ScrollView.

Vamos adicionar mais elementos na nossa MainPage.xaml e testar o uso das cores?

Para isso vamos adicionar um StackLayout, um Layout que empilha elementos horizontalmente e verticalmente. A opção vertical é a default.

Esse conteúdo ficará dentro de `<ContentPage ...></ContentPage>`

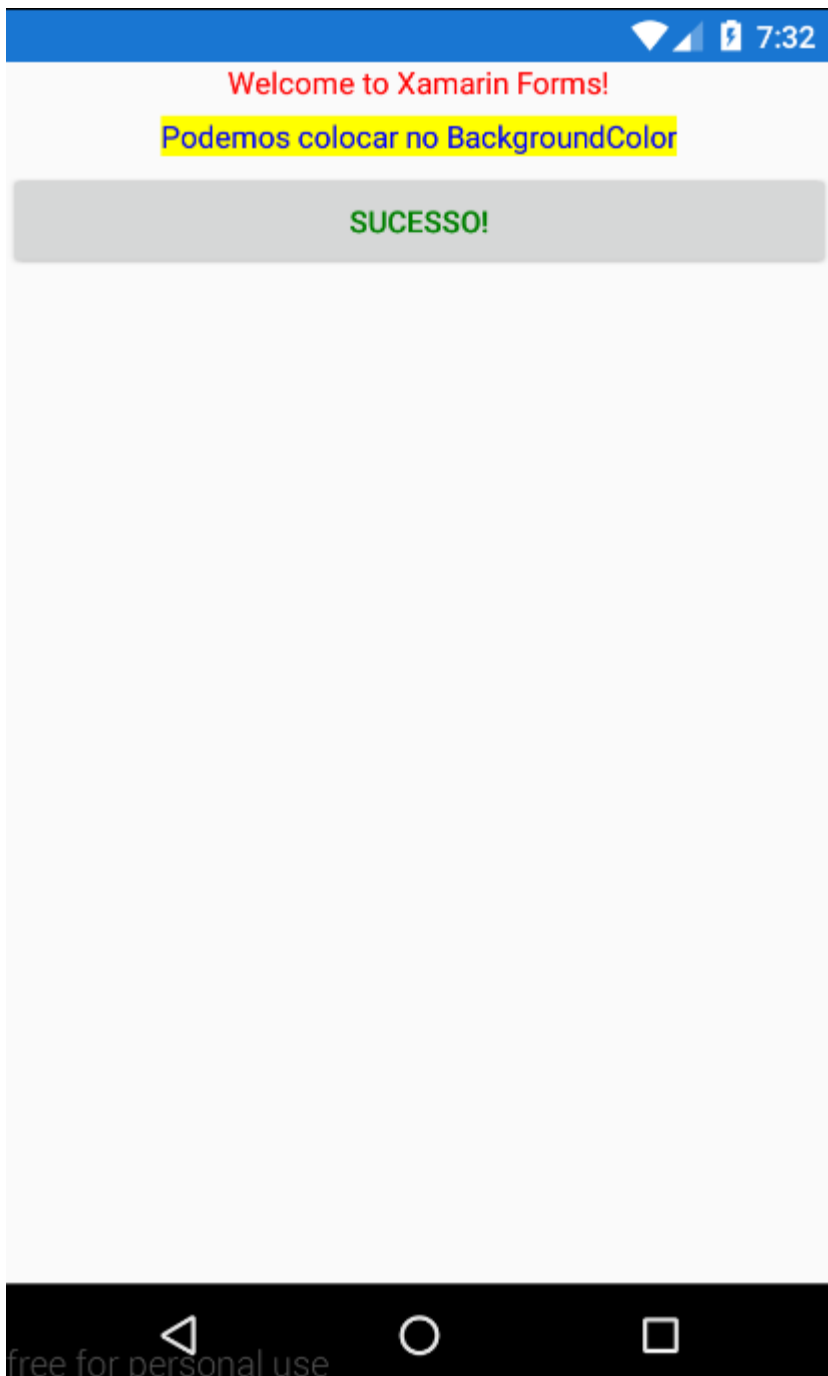
```
<StackLayout>
  <Label Text="Welcome to Xamarin Forms!"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    TextColor="{StaticResource danger}"
  />

  <Label Text="Podemos colocar no BackgroundColor"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    BackgroundColor="{StaticResource warning}"
    TextColor="{StaticResource info}"
  />

  <Button TextColor="{StaticResource success}" Text="Sucesso!" />
</StackLayout>
```

Execute o app!

Resultado:



Podemos notar que os elementos estão empilhados um em cima do outro, e cada um tem uma customização de cor baseada no resource! Agora imagine uma tela onde tenhamos vários elementos, e como eles serão empilhados alguns não serão exibidos pois a tela do device tem um tamanho específico, sendo assim, precisamos criar um ScrollBar para podermos deslizar para cima ou para baixo para acessar os controles ocultos. Para isso temos o ScrollView:

<ScrollView>

```
<StackLayout Spacing="100">
  <Label Text="Welcome to Xamarin Forms!"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    TextColor="{StaticResource danger}"
  />

  <Label Text="Podemos colocar no BackgroundColor"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    BackgroundColor="{StaticResource warning}"
    TextColor="{StaticResource info}"
  />

  <Button TextColor="{StaticResource success}" Text="Sucesso!" />

  <Label Text="Welcome to Xamarin Forms!"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    TextColor="{StaticResource danger}"
  />

  <Label Text="Podemos colocar no BackgroundColor"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    BackgroundColor="{StaticResource warning}"
    TextColor="{StaticResource info}"
  />

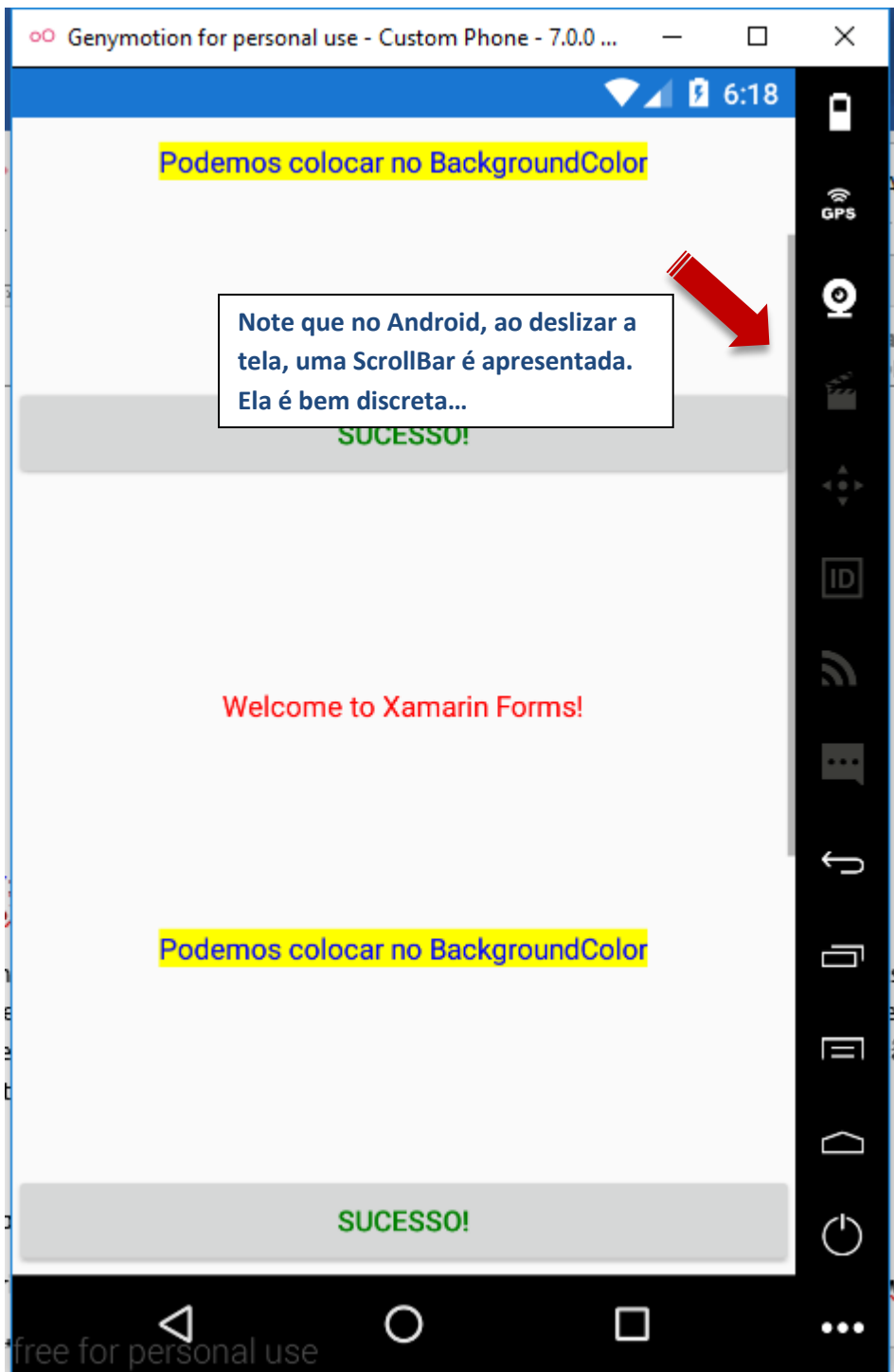
  <Button TextColor="{StaticResource success}" Text="Sucesso!" />

  <Label Text="Welcome to Xamarin Forms!"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    TextColor="{StaticResource danger}"
  />

  <Label Text="Podemos colocar no BackgroundColor"
    VerticalOptions="Center"
    HorizontalOptions="Center"
    BackgroundColor="{StaticResource warning}"
    TextColor="{StaticResource info}"
  />

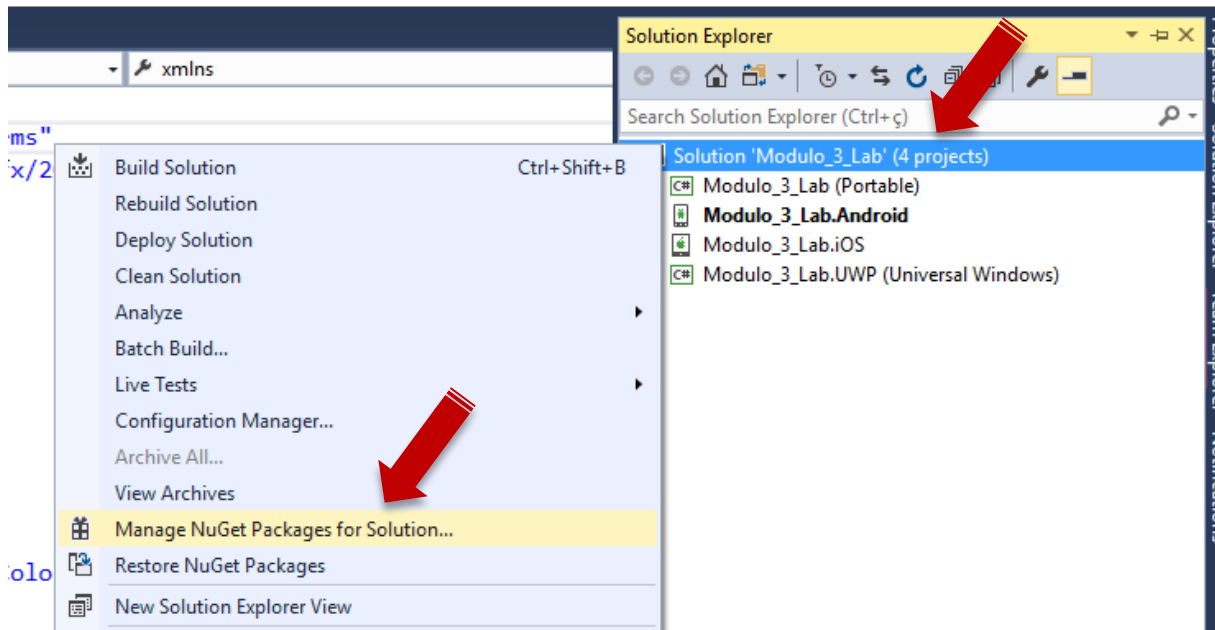
  <Button TextColor="{StaticResource success}" Text="Sucesso!" />
</StackLayout>
</ScrollView>
```

Criamos um ScrollView e colocamos nosso StackLayout dentro dele. Note que adicionamos a propriedade Spacing como valor de 100 para podermos adicionar uma distância entre os elementos e, consequentemente, fazer com que eles extrapolem o tamanho da tela, forçando a criação de uma ScrollBar lateral:

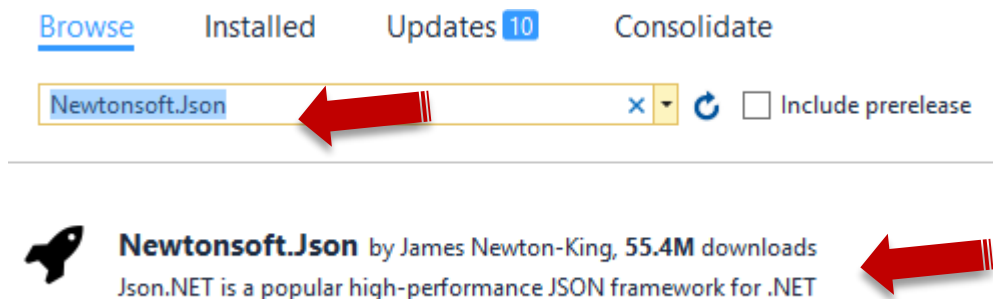


Tarefa 4. Consumir um serviço do Azure para preencher uma ListView.

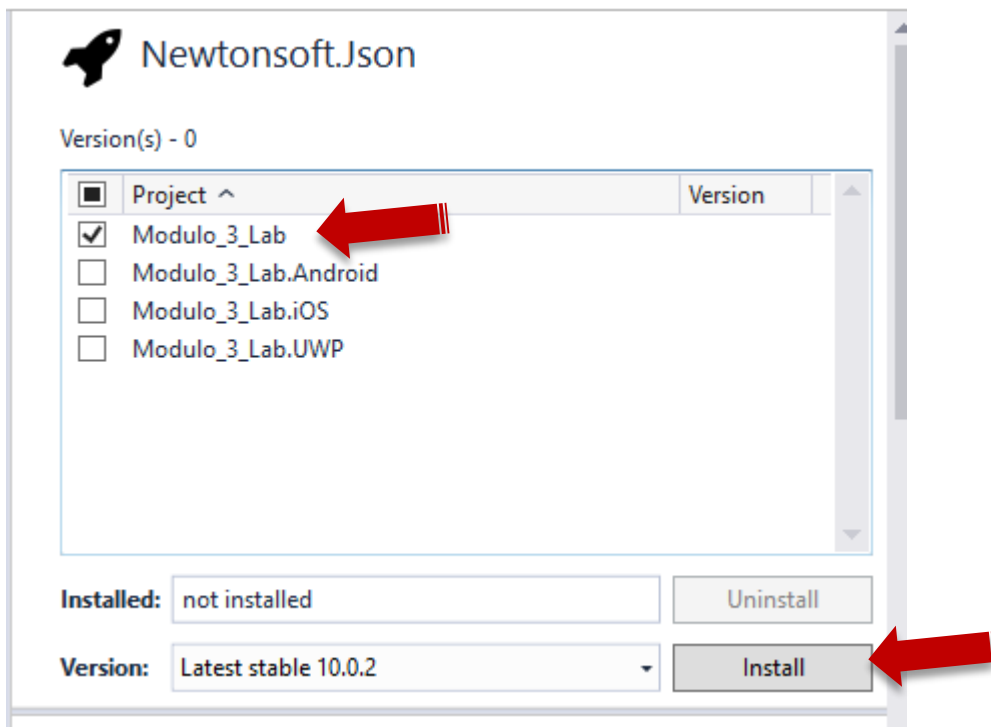
Para consumir um serviço no Azure, precisamos adicionar duas bibliotecas: Newtonsoft.Json, para fazer manipulação do Json (conteúdo retornado da Api do Azure) e Microsoft.Net.Http para fazer requisições Http. Para isso, clique com o botão direito na Solution, e em seguida clique em Manage NuGet Package for Solution...:



Na aba Browse, digite Newtonsoft.Json e dê enter:



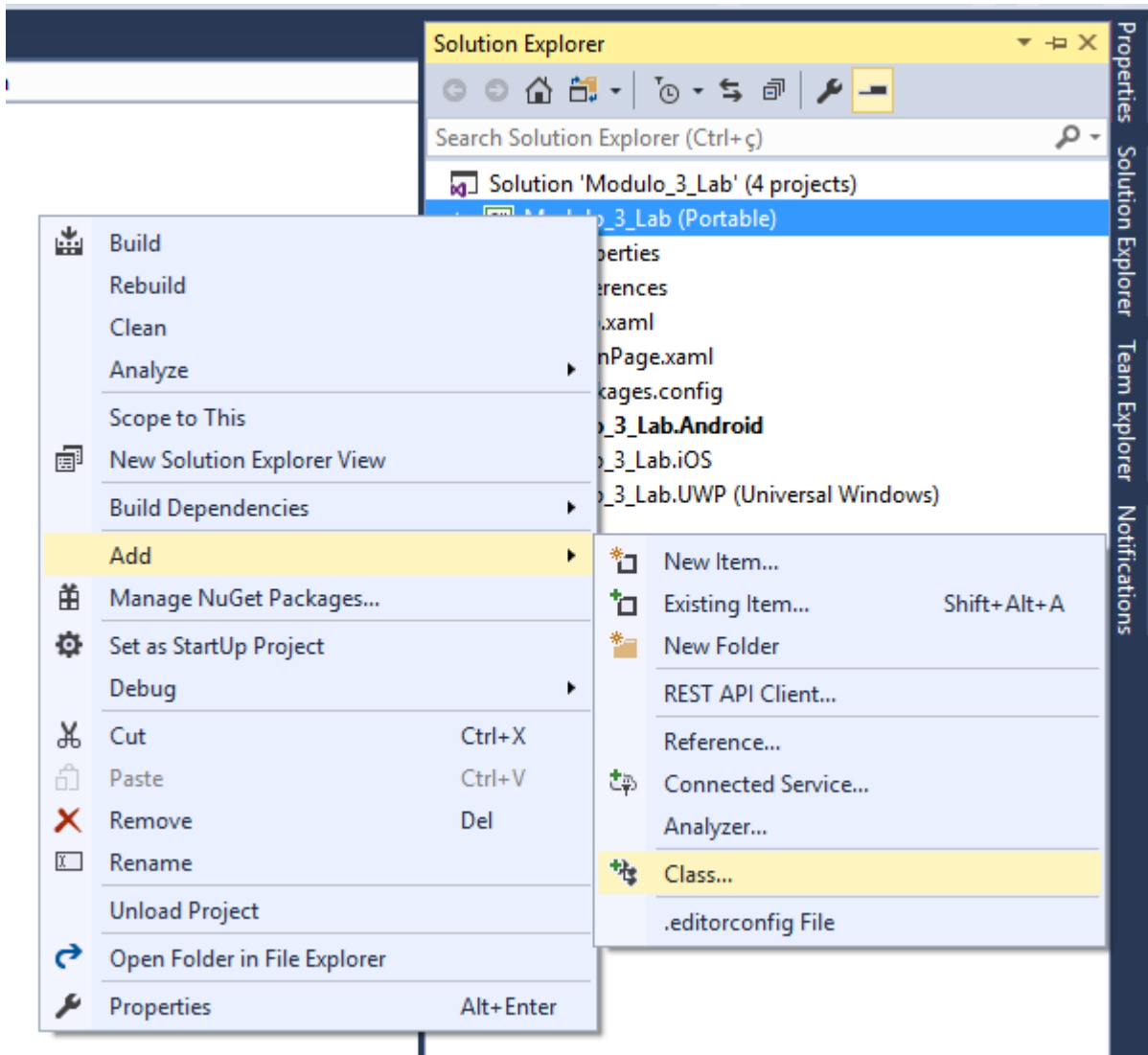
Selecione o Pacote Newtonsoft.Json. No painel ao lado, selecione o projeto Portable e clique em Install.



Faça o mesmo para Microsoft.Net.Http!

Após a instalação, vamos criar nossa classe Model e nosso serviço de acesso à Api!

Primeiro criaremos nossa classe Model chamada Tag. Para isso, clique com o botão direito no projeto Portable, selecione a opção Add e em seguida Class. Dê o nome da classe de Tag e clique em ok!



Nossa classe Tag deverá ter as seguintes propriedades:

```
public class Tag
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Slug { get; set; }
    public string Description { get; set; }
}
```

Essas propriedades serão preenchidas com os valores que virão da Api.

Falando em Api, vamos criar nosso serviço. Adicione uma classe chamada Api. Acima temos imagens e instruções para adição de um arquivo.

Para podermos fazer a requisição Http efetuar um parse do Json transformando-o em um objeto do tipo Tag, adicione os seguintes namespaces.

```
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;
using Newtonsoft.Json;
```

Agora vamos deixar nossa classe estática, já que ela não guardará nenhum estado.

```
public static class Api
```

Por fim vamos adicionar nosso método de acesso a Api do Azure:

```
    public static async Task<List<Tag>> GetTagsAsync()
    {
        const string BaseUrl = https://monkey-hub-api.azurewebsites.net/api/;

        var httpClient = new HttpClient();
        httpClient.DefaultRequestHeaders.Accept.Add(new
        MediaTypeWithQualityHeaderValue("application/json"));

        var response = await
        httpClient.GetAsync($"{BaseUrl}Tags").ConfigureAwait(false);

        if (response.IsSuccessStatusCode)
        {
            using (var responseStream = await
            response.Content.ReadAsStreamAsync().ConfigureAwait(false))
            {
                return JsonConvert.DeserializeObject<List<Tag>>(
                    await new StreamReader(responseStream)
                        .ReadToEndAsync().ConfigureAwait(false));
            }
        }

        return null;
    }
```

Basicamente o que é feito é:

1. Configuramos a requisição para que o tipo de conteúdo vindo do Backend Azure seja "application/json". Tem situações que podemos forçar para que o retorno seja em XML. Isso vai depender de como o seu backend está montado.
2. Fazemos uma requisição GET assíncrona para obter os dados.
3. Verificamos se a requisição foi efetuada com sucesso.
4. Lemos o retorno da Api (Stream) e o convertemos em uma lista de Tags.

Agora que temos nossa classe de serviço que retorna uma lista de tags, já podemos preencher uma ListView 🙌

Tarefa 5. Criar uma ListView.

Abra o arquivo MainPage.xaml. Vamos reformular nosso layout excluindo o conteúdo da ContentPage. Em seguida vamos adicionar um StackLayout com um botão e uma Lista.

No botão, daremos o nome de btnCarregar, e na lista daremos o nome de lvwTags.

O XAML deve ser algo como:

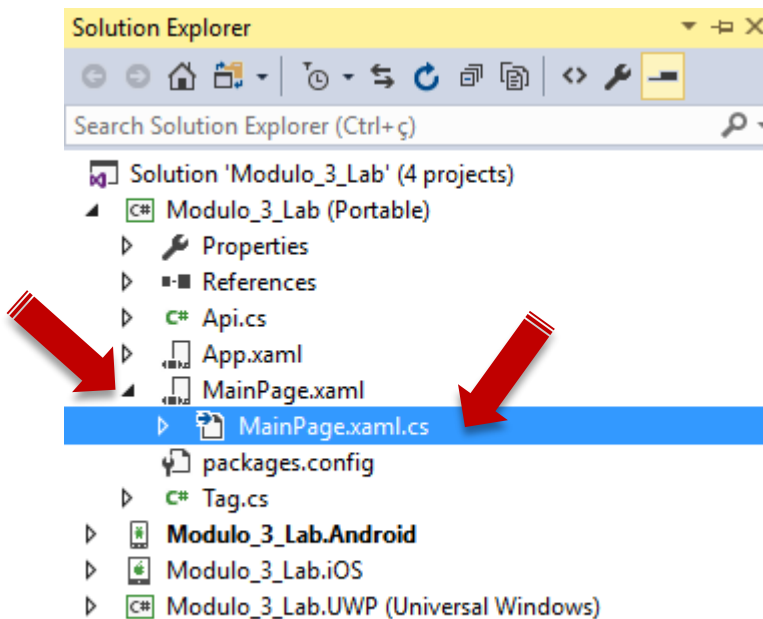
```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:Modulo_3_Lab"
             x:Class="Modulo_3_Lab.MainPage">

    <StackLayout>
        <Button x:Name="btnCarregar"
              Text="Carregar"
              TextColor="{StaticResource success}" />

        <ListView x:Name="lvwTags">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <TextCell Text="{Binding Name}"
                           Detail="{Binding Description}" />
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackLayout>
</ContentPage>
```

Note que adicionamos uma cor de texto no nosso botão.

Nós damos nome aos elementos para podermos gerencia-los no codebehind da Página. Sendo assim, abra o arquivo MainPage.xaml.cs. Para fazer isso clique na seta que fica do lado esquerdo do nome do arquivo. Isso vai abrir mais um nível na árvore do projeto. Logo em seguida dê um duplo clique no arquivo MainPage.xaml.cs:



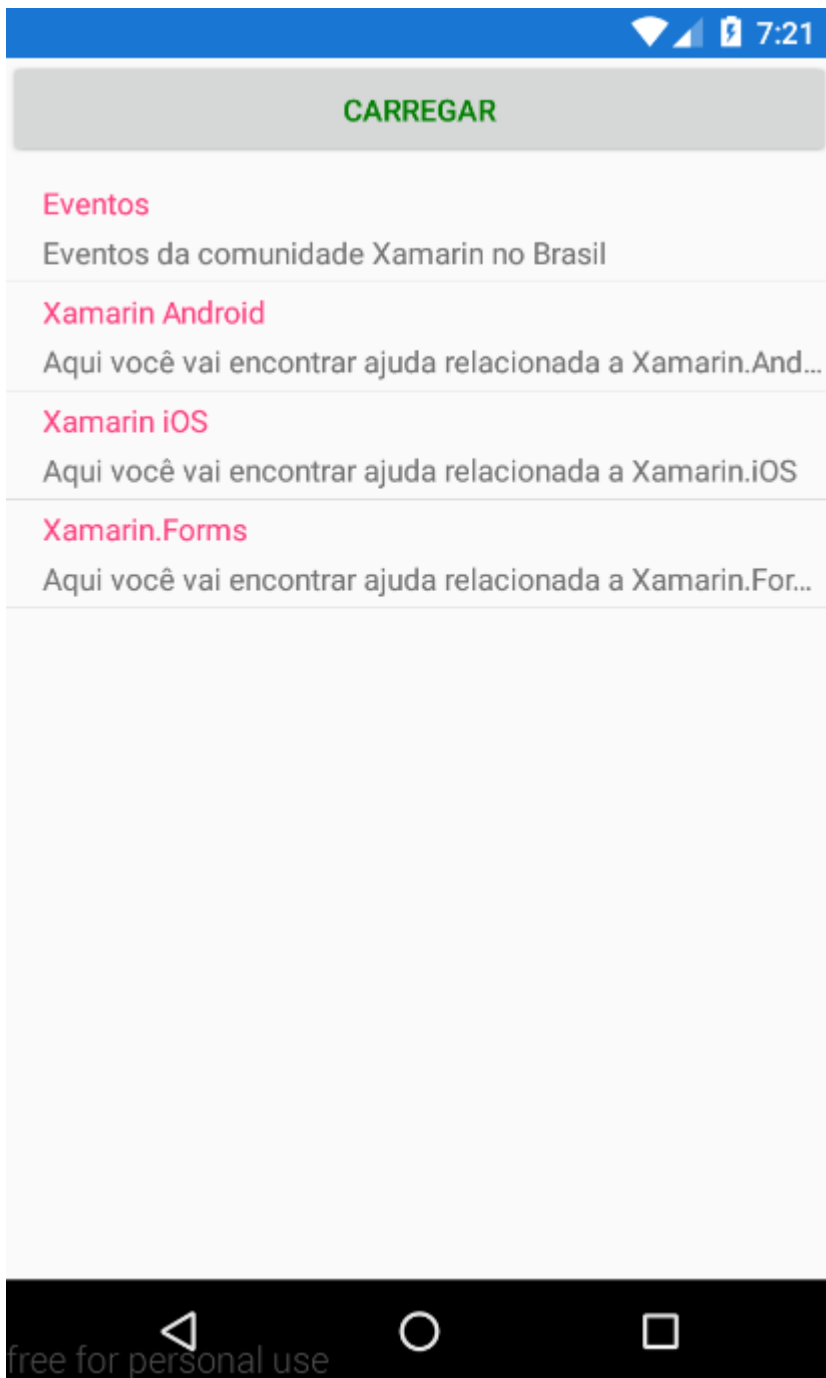
Agora nós vamos efetuar nossa requisição. Para isso vamos adicionar o código que invoca a api no clique do botão. No construtor MainPage, logo abaixo do método `InitializeComponent` adicione o código do clique do botão.

```
public MainPage()
{
    InitializeComponent();

    this.btnCarregar.Clicked += async (sender, e) =>
    {
        var tags = await Api.GetTagsAsync();
        this.lvwTags.ItemsSource = tags;
    };
}
```

Execute o app!

Resultado:



Resumo

Neste laboratório, você criou alguns estilos que podem ser aplicados em controles em qualquer parte do seu app, reaproveitando muito código e facilitando o desenvolvimento.

Você conseguiu organizar controles dentro de um StackLayout e consegui criar uma tela com Scroll usando o ScrollView.

Você consumiu um serviço do Azure de forma simples e rápida!

Você já sabe trabalhar com listas e customizar suas células, e como já aprendeu MVVM já consegue ser muito produtivo!

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

Eu terminei o #Lab2 da #MaratonaXamarin Intermediária e já sei como criar telas e estilos usando XAML!