Laboratório

Aderindo a serviços REST com Xamarin.Forms

Versão: 1.0.0
Dezembro de 2016

Miguel Muñoz Serafín
@msmdotnet

Suporte: mmunoz@mail.ticapacitacion.com

CONTEÚDO

Introdução

Integrar um serviço Web dentro de uma aplicação é um cenário comum. Neste laboratório, será demonstrada a forma de consumir um serviço Web RESTful a partir de uma aplicação Xamarin.Forms. A aplicação Xamarin.Forms será desenvolvida para implementar o padrão MVVM.

Objetivos

Ao finalizar este laboratório, os participantes serão capazes de:

- Implementar o padrão MVVM em uma aplicação Xamarin.Forms.
- Utilizar a classe **HttpClient** para consumir um serviço Web RESTful a partir de uma aplicação Xamarin.Forms.
- Utilizar a classe **JsonConvert** para de serializar dados JSON a objetos .NET.

Requisitos

Para a realização deste laboratório é necessário contar com o seguinte:

- Uma equipe de desenvolvimento com sistema operativo Windows 10 e Visual Studio 2015 Community, Professional ou Enterprise com a plataforma Xamarin.
- Uma equipe Mac com a plataforma Xamarin.
- Uma conexão à Internet.

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

Exercício 1: Consumindo um serviço Web RESTful.

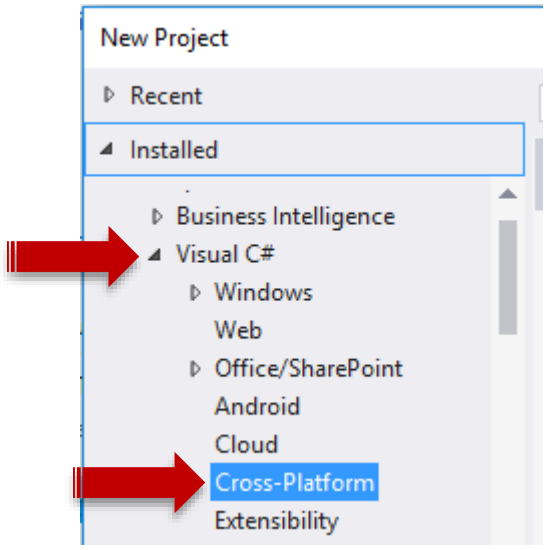
Neste exercício será criada uma aplicação Xamarin.Forms que consumirá um serviço Web RESTful. O serviço RESTful preparado para este laboratório foi desenvolvido com ASP.NET Web API e expõe informação em formato JSON relacionada com o preço dos gatos mais caros do mundo.

A aplicação será desenvolvida implementando o padrão MVVM.

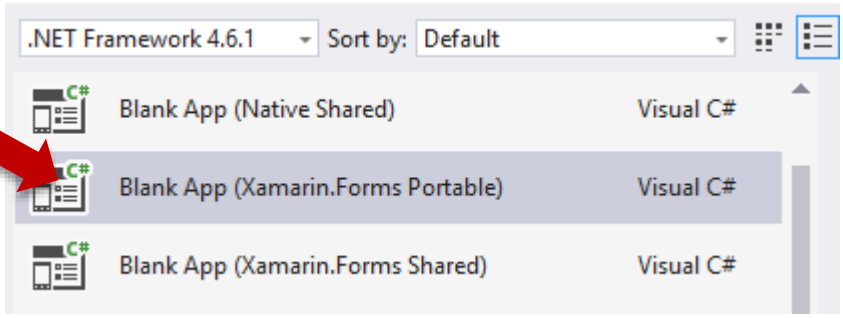
Tarefa 1. Criar uma aplicação Xamarin.Forms.

Nesta tarefa será criada uma aplicação Xamarin.Forms utilizando Microsoft Visual Studio e o modelo **Blank App (Xamarin.Forms Portable)**.

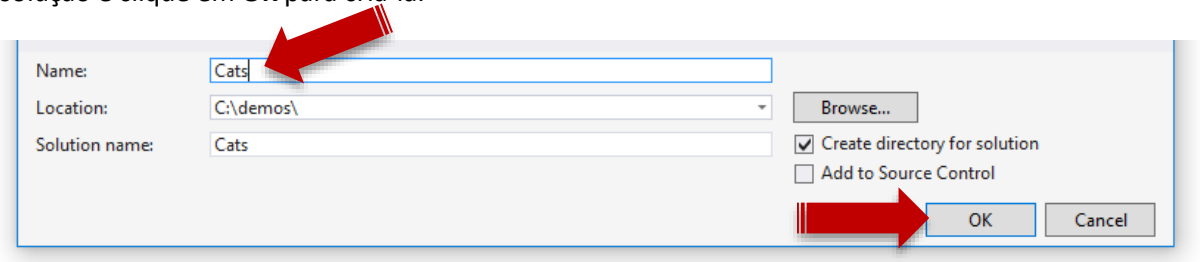
1. Selecione a opção **File > New > Project** no Visual Studio.
2. No painel esquerdo da janela **New Project** selecione **Visual C# > Cross-Platform**.



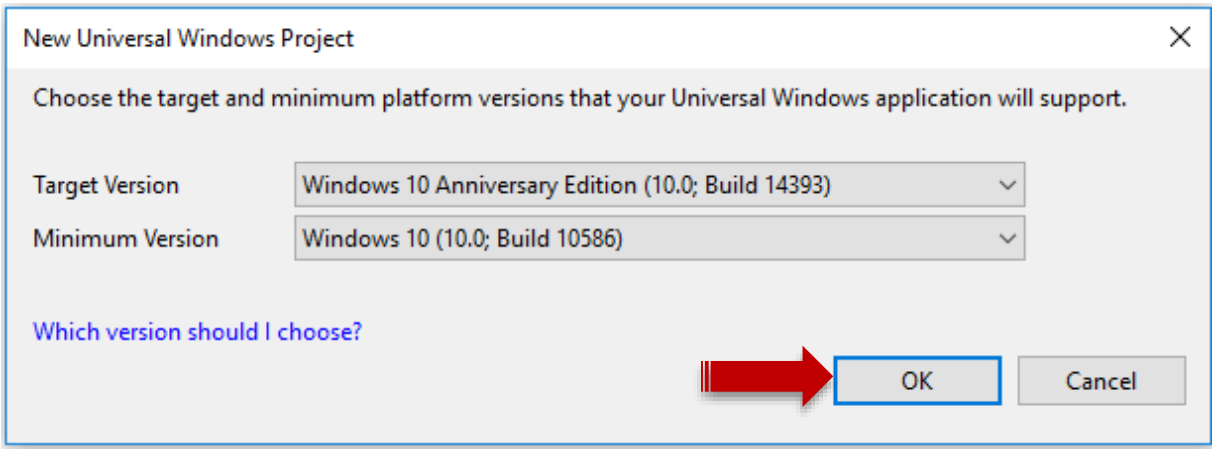
3. Selecione o modelo **Blank App (Xamarin.Forms Portable)**.



4. Forneça **Cats** como nome da solução. Forneça também a localização onde deseja criar a solução e clique em **OK** para criá-la.

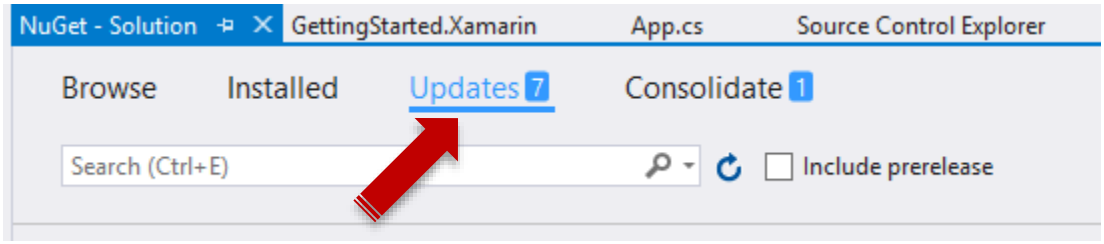


5. Clique no botão **OK** do quadro de diálogo **New Universal Windows Project** para aceitar as versões sugeridas para a aplicação UWP que será criada.



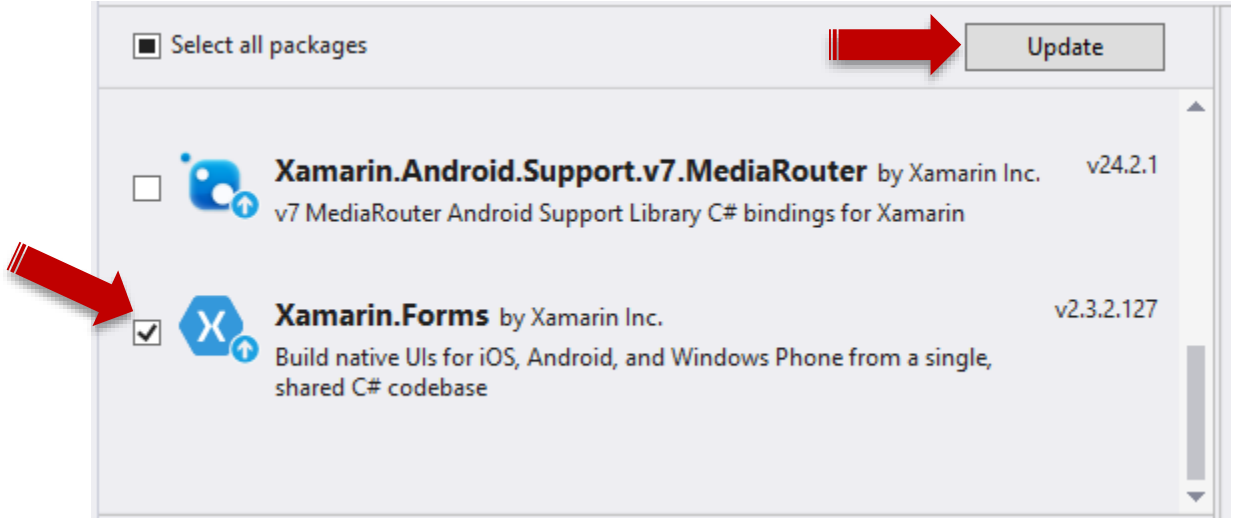
6. Depois que a solução com seus projetos for criada, selecione a opção **Manage NuGet Packages for Solution…** no menu de contexto do nome da solução.

7. Clique na opção **Updates** para ver as atualizações disponíveis.



Visual Studio pode indicar que existem atualizações para o pacote NuGet Xamarin.Forms e todas suas unidades, porém, Xamarin.Forms está configurado para dependências de versões específicas. Portanto, embora o Visual Studio te indique que existem novas versões disponíveis de pacotes Xamarin.Android.Support, Xamarin.Forms não é necessariamente compatível com essas novas versões.

8. Selecione o pacote **Xamarin.Forms** e clique em **Update** para iniciar a atualização.



É provável que você será solicitado para aceitar as alterações e reiniciar o Visual Studio para concluir a instalação.

Tarefa 2. Instalar os pacotes NuGet.

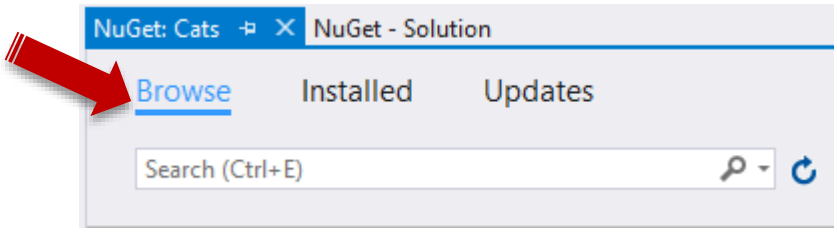
Dois pacotes Nuget são necessários em nossa aplicação:

- **Microsoft.Net.Http**. Este pacote proporciona uma interface de programação para aplicações baseadas em HTTP/REST. O pacote inclui a classe **HttpClient** que é utilizada para enviar

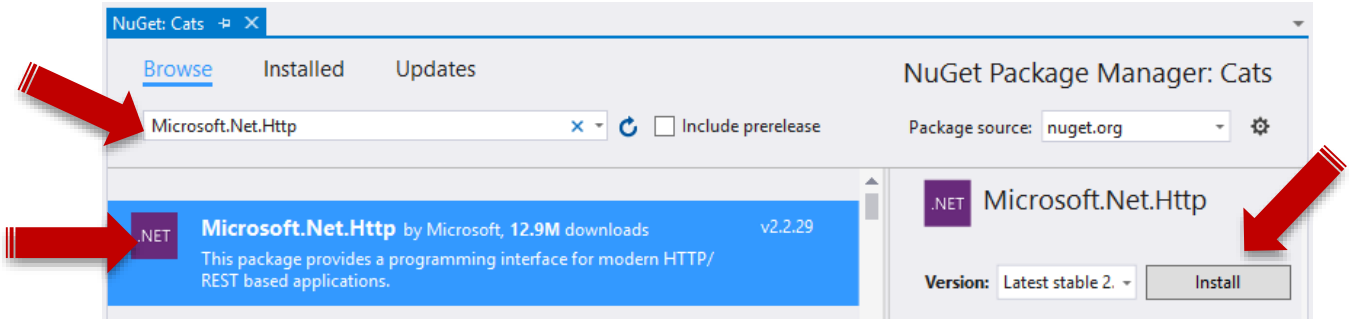
petições sobre HTTP. Inclua também as classes **HttpRequestMessage** e **HttpResponseMessage** para processar as mensagens HTTP.

- **Newtonsoft.Json**. Este pacote é um Framework que permite serializar e deserializar dados em formato JSON. É útil para processar os dados que são enviados ou recebidos à partir do serviço REST.

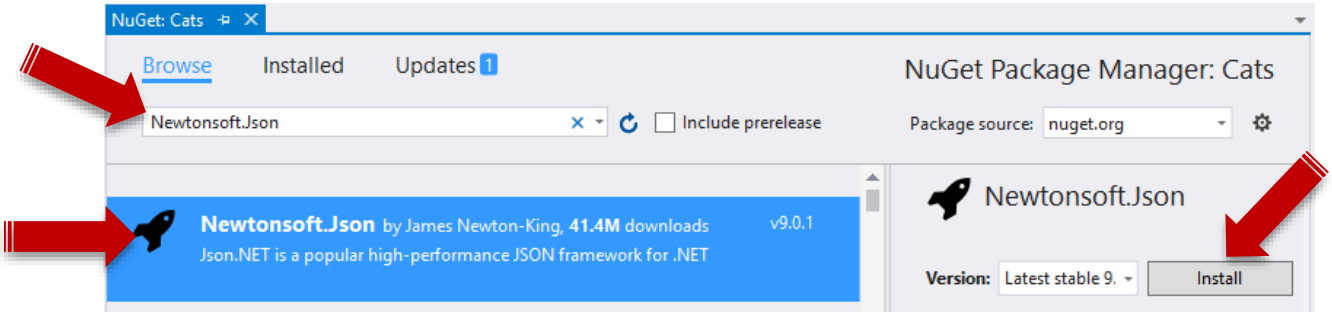
1. Selecione a opção **Manage NuGet Packages…** do menu contextual do projeto PCL.
2. Na janela **NuGet** selecione a aba **Browse**.



3. No quadro de buscas escreva **Microsoft.Net.Http,** selecione o pacote **Microsoft.Net.Http** e clique em **Install** para instalar o pacote NuGet.



4. Confirme as alterações e o acordo de licenças quando for solicitado.
5. Após a instalação do pacote, escreva no quadro de buscas **Newtonsoft.Json**, selecione o pacote **Newtonsoft.Json** e clique em **Install** para instalar o pacote NuGet.

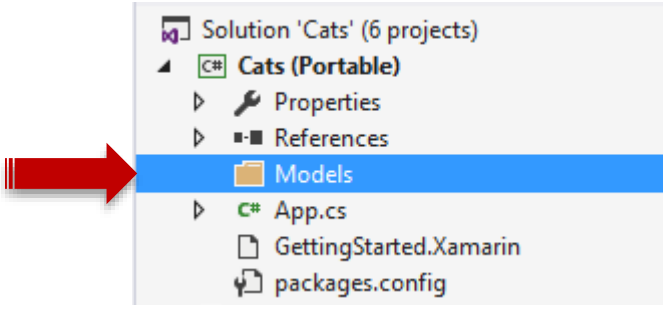


6. Confirme as alterações quando for solicitado.

Tarefa 3. Criar o Modelo.

A aplicação obterá do serviço REST os dados das raças de gatos mais caras do mundo. Nesta tarefa você criará a classe modelo que te permitirá armazenar os dados de uma raça de gato.

1. No projeto PCL acrescente um novo diretório chamado **Models**.



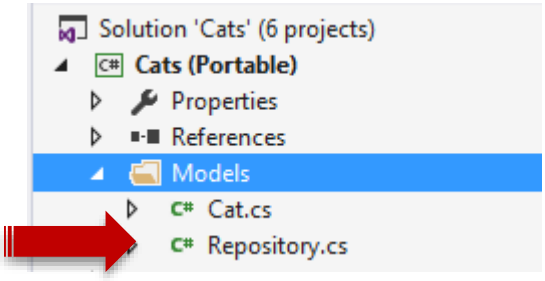
2. Dentro do diretório **Models** acrescente a classe **Cat** com as seguintes propriedades públicas:

Nome	Tipo	Descrição
Id	string	Identificador único de uma raça de gato.
Name	string	Nome da raça do gato.
Description	string	Descrição da raça do gato.
Price	int	Preço de um gato desta raça.
WebSite	string	URL do site na internet onde é possível encontrar mais informações sobre a raça de gato.
Image	string	URL da imagem de um gato desta raça.

O código da classe será similar ao seguinte:

```
public class Cat
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public int Price { get; set; }
    public string WebSite { get; set; }
    public string Image { get; set; }
}
```

3. Dentro do diretório **Models** acrescente uma nova classe pública chamada **Repository**. Esta classe conterá a lógica de acesso aos dados da aplicação.



4. Acrescente o seguinte código à classe **Repository** para definir e implementar o método **GetCats**. Este método irá retornar uma lista de dados obtidos a partir do serviço REST.

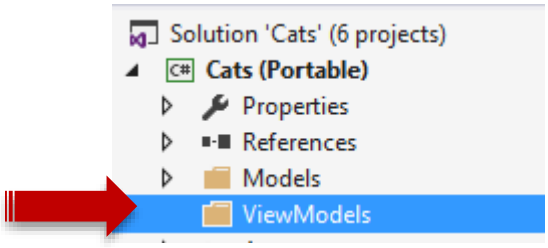
```
public async Task<List<Cat>> GetCats()
{
    List<Cat> Cats;
    var URLWebAPI = "http://demos.ticapacitacion.com/cats";
    using (var Client = new System.Net.Http.HttpClient())
    {
        var JSON = await Client.GetStringAsync(URLWebAPI);
        Cats = Newtonsoft.Json.JsonConvert.DeserializeObject<List<Cat>>(JSON);
    }
    return Cats;
}
```

Note que o método está definido para executar-se de forma assíncrona e que os dados são obtidos a partir do serviço **http://demos.ticapacitacion.com/cats** utilizando a classe **HttpClient**. Os dados JSON obtidos são deserializados utilizando o método **DeserializeObject** do objeto **JsonConvert**.

Tarefa 4. Criar o ViewModel.

Nesta tarefa você criará o ViewModel pela classe **CatsViewModel**. Esta classe proporcionará toda a funcionalidade que necessita a Vista Xamarin.Forms principal da aplicação para mostrar os dados das raças de gatos. O ViewModel conterá a lista de objetos **Cat** e um método que poderá ser invocado para obter a lista de objetos **Cat** do repositório. Também conterá uma propriedade booleana que indicará se estamos obtendo os dados em uma tarefa de fundo (Background Task).

1. No projeto PCL acrescente um novo diretório chamado **ViewModels**.



2. No diretório ViewModels agregue a classe pública chamada **CatsViewModel**.
3. Uma classe ViewModel deve ser capaz de notificar as mudanças que acontecerem em suas propriedades através da implementação da interface **INotifyPropertyChanged**.

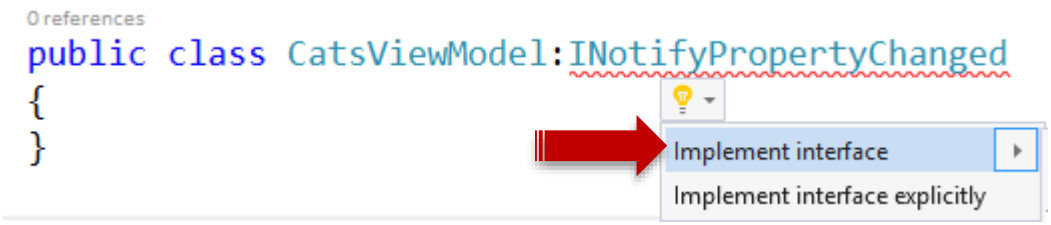
   Acrescente ao início do arquivo **CatsViewModel.cs** o seguinte código para importar o espaço de nomes da interface **INotifyPropertyChanged**.

   ```
   using System.ComponentModel;
   ```

4. Modifique a definição da classe **CatsViewModel** para indicar que implementa a interface **INotifyPropertyChanged**.

   ```
   public class CatsViewModel:INotifyPropertyChanged
   {
   }
   ```

   INotifyPropertyChanged é importante para a relação a dados em Frameworks MVVM. É uma interface que quando é implementada permite a nossa Vista conhecer as alterações do ViewModel.
5. Com a ajuda do intellisense do Visual Studio implemente a interface **INotifyPropertyChanged**.

   Isto agregará a seguinte linha de código:

   ```
   public event PropertyChangedEventHandler PropertyChanged;
   ```

6. Acrescente o seguinte código para definir um método de ajuda chamado **OnPropertyChanged** que lançará o evento **PropertyChanged**. Invocaremos este método de ajuda quando uma propriedade do ViewModel mudar.

   ```
   private void OnPropertyChanged(
       [System.Runtime.CompilerServices.CallerMemberName]
       string propertyName = null) =>
           PropertyChanged?.Invoke(this,
               new PropertyChangedEventArgs(propertyName));
   ```

   Observe que estamos aplicando o atributo **CallerMemberName** ao parâmetro **propertyName.** Este atributo permite obter o nome de membro da classe que invoca este método. Isso evitará que devemos especificar mediante código o nome a propriedade modificado no momento de utilizar este método.

7. O seguinte passo é criar uma propriedade que permita a vista determinar se o ViewModel se encontra ocupado. Desta forma poderemos evitar realizar operações duplicadas, como por exemplo quando o usuário atualiza os dados múltiplas vezes.

   Adicione o seguinte código para criar o campo de respaldo da propriedade.

   ```
   private bool Busy;
   ```

8. Adicione o seguinte código para criar a propriedade **IsBusy**.

   ```
   public bool IsBusy
   {
       get
       {
           return Busy;
       }
       set
       {
           Busy = value;
           OnPropertyChanged();
       }
   }
   ```

   Note que estamos invocando o método **OnPropertyChanged** quando o valor da propriedade muda. A infraestrutura de ligação do Xamarin.Forms se inscreverá para nosso evento **PropertyChanged** para que a interface de usuário seja notificada da mudança.

9. Adicione o seguinte código para definir uma propriedade **Cats** que armazenará a lista de objetos **Cat**.

   ```
   public ObservableCollection<Cat> Cats { get; set; }
   ```

   Note que estamos utilizando **ObservableCollection** pois esta classe tem suporte para eventos **CollectionChanged** que ocorrem quando agregamos ou eliminamos elementos da coleção. Isto é muito útil já que não temos que invocar ao método **OnPropertyChanged** por cada mudança nos elementos da coleção.

10. Agrega o seguinte código ao início do arquivo para importar o espaço de nomes das classes modelo da aplicação e da classe **ObservableCollection**.

    ```
    using System.Collections.ObjectModel;
    using Cats.Models;
    ```

11. Adicione o seguinte código para definir o construtor do ViewModel. O código inicializará a propriedade **Cats**.

    ```
    public CatsViewModel()
    {
        Cats = new ObservableCollection<Models.Cat>();
    }
    ```

12. Adicione o seguinte código para definir um método assíncrono chamado **GetCats** que obterá os dados das raças de gatos a partir do repositório.

```
async Task GetCats()
{
}
```

13. Dentro do método **GetCats** adicione o seguinte código que permitirá detectar se atualmente o ViewModel se encontra ocupado obtendo os dados.

```
if(!IsBusy)
{
}
return;
```

14. Dentro do bloco **if** adicione o seguinte código **try/catch/finally**.

```
Exception Error = null;
try
{
    IsBusy = true;
}
catch (Exception ex)
{
    Error = ex;
}
finally
{
    IsBusy = false;
}
```

Note que estamos estabelecendo **IsBusy** a **true** e posteriormente a **false** quando iniciamos a recuperação da informação a partir do repositório e quando terminamos de obter os dados.

15. Adicione o seguinte código dentro do bloco **try** para obter os dados do repositório.

```
try
{
    IsBusy = true;
    var Repository = new Repository();
    var Items = await Repository.GetCats();
}
```

16. Dentro do bloco **try,** debaixo do código anterior, adicione o seguinte código para limpar a lista atual de objetos **Cat** e carregá-los a partir da coleção **Items**.

```
    Cats.Clear();
    foreach(var Cat in Items)
    {
        Cats.Add(Cat);
    }
```

Se algo der errado, o bloco **catch** guardará a exceção e depois do bloco **finally** poderemos mostrar uma mensagem emergente.

17. Adicione o seguinte código logo após do bloco **finally** para mostrar uma mensagem em caso de que se tenha gerado uma exceção.

```
if (Error != null)
{
    await Xamarin.Forms.Application.Current.MainPage.DisplayAlert(
        "Error!", Error.Message, "OK");
}
```

O método principal do ViewModel para obter os dados terá sido completado. Ao invés de invocar o método diretamente, o exporemos com um **Command**. Um objeto **Command** tem uma interface que sabe qual método invocar e tem uma forma opcional de descrever se o **Command** está habilitado.

18. Adicione o seguinte código à classe **CatsViewModel** para criar um novo comando chamado **GetCatsCommand**.

```
public Command GetCatsCommand { get; set; }
```

19. Adicione o seguinte código ao início do arquivo para importar o espaço de nomes da classe **Command**.

```
using Xamarin.Forms;
```

20. Dentro do construtor de **CatsViewModel** adicione o seguinte código para inicializar o comando **GetCatsCommand** passando a ele dois métodos: um para se invocar quando o comando for executado e outro para determinar quando o comando estiver habilitado. Ambos os métodos estão implementados como expressões lambda.

```
GetCatsCommand = new Command(
    async () => await GetCats(),
    () => !IsBusy
    );
```

A única modificação que teremos que fazer é para o caso em que o valor da propriedade **IsBusy** mude. Neste caso, teremos que reavaliar a função que determina se o comando está habilitado.

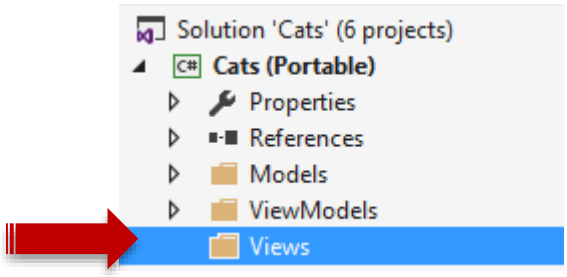
21. Adicione o seguinte código ao final do bloco **set** da propriedade **IsBusy** para invocar o método **ChangeCanExecute** do comando **GetCatsCommand**. Ao executar o método **ChangeCanExecute**, a função que determina se o comando está habilitado será reavaliada.

```
GetCatsCommand.ChangeCanExecute();
```

Tarefa 5. Criar a Vista.

Finalmente, é hora de construir a interface de usuário Xamarin.Forms que constituirá o elemento View de nossa aplicação MVVM.

1. Adicione um diretório **Views** na raiz do projeto PCL.



2. Dentro do diretório **Views** adicione um novo elemento **Forms Xaml Page** chamado **CatsPage.xaml**.
   Para esta página adicionaremos controles alinhados verticalmente. Podemos utilizar um controle **StackLayout** para fazer isso.
3. Substitua o elemento **Label** dentro de **ContentPage** pelo seguinte código.

```
<StackLayout Spacing="0">

</StackLayout>
```

   Este será o contentor de onde todos os controles child serão colocados. Note que especificamos que os controles child não terão espaço entre eles.
4. Adicione o seguinte código para criar um botão com uma ligações ao comando **GetCatsCommand** do ViewModel. O comando toma o lugar de um manipulador do evento **Clicked** e será executado quando o usuário tocar o botão.

```
<Button Text="Sincronizar" Command="{Binding GetCatsCommand}"/>
```

   Abaixo do botão podemos mostrar um indicador para informar ao usuário quando a aplicação estiver obtendo os dados do servidor. Para fazer isso, podemos utilizar um controle **ActivityIndicator** e vinculá-lo à propriedade **IsBusy** do ViewModel.
5. Adicione o seguinte código para definir o controle **ActivityIndicator**.

```
<ActivityIndicator IsRunning="{Binding IsBusy}" IsVisible="{Binding IsBusy}"/>
```

   Utilizaremos um **ListView** que se vincule à coleção **Cats** para mostrar todos os elementos. Podemos utilizar uma propriedade especial chamada ***x.Name=””*** para nomear qualquer controle.
6. Adicione o seguinte código para definir o elemento **ListView**.

```
<ListView x:Name="ListViewCats" ItemsSource="{Binding Cats}" >

</ListView>
```

Agora necessitamos descrever a forma em que serão mostrados os elementos da coleção. Para fazer isto podemos utilizar um **ItemTemplate** que tenha um **DataTemplate** com uma vista específica. Xamarin.Forms possui algumas células (Cells) que podemos utilizar. Utilizaremos **ImageCell** que tem uma imagem e duas linhas de texto.

7. Agrega o seguinte código XAML dentro do **ListView** para definir a forma em que serão mostrados os elementos da coleção **Cats**.

```
<ListView.ItemTemplate>
  <DataTemplate>
    <ImageCell Text="{Binding Name}"
               Detail="{Binding Price, StringFormat='{0:c} dólares'}"
               ImageSource="{Binding Image}"/>
  </DataTemplate>
</ListView.ItemTemplate>
```

8. Adicione o seguinte código dentro da etiqueta de elemento **ContentPage** para definir um alias ao espaço de nomes do ViewModel e agregar um título à página.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
     xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
     x:Class="Cats.Views.CatsPage"
     xmlns:VM="clr-namespace:Cats.ViewModels"
     Title="Cats">
```

9. Finalmente, adicione o seguinte código XAML para definir o contexto de ligação especificado nas propriedades dos controles agregados à página.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="Cats.Views.CatsPage"
             xmlns:VM="clr-namespace:Cats.ViewModels"
             Title="Cats">
  <ContentPage.BindingContext>
    <VM:CatsViewModel/>
  </ContentPage.BindingContext>
```

Xamarin.Forms automaticamente carregará e mostrará a imagem a partir do servidor.

Tarefa 6. Testar a aplicação.

Antes de testar a aplicação, teremos que fazer algumas modificações.

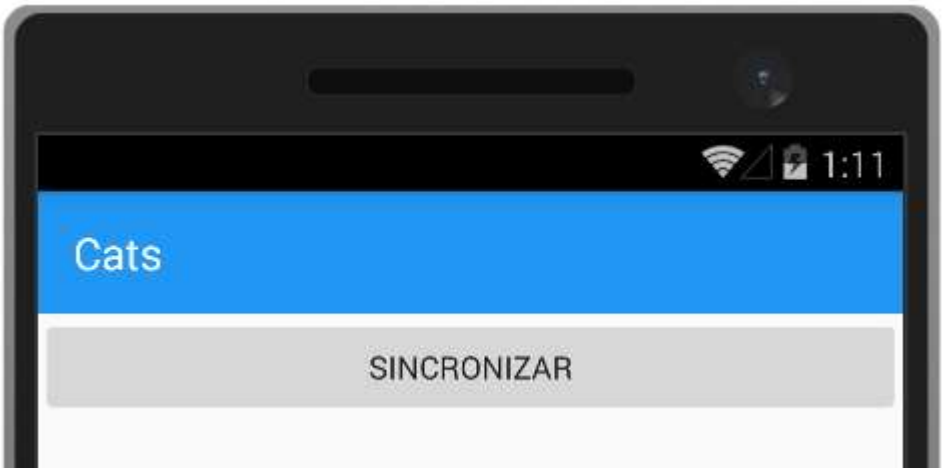
1. Abra o arquivo **App.cs** do projeto PCL.

2. Substitua o código do construtor pelo seguinte.

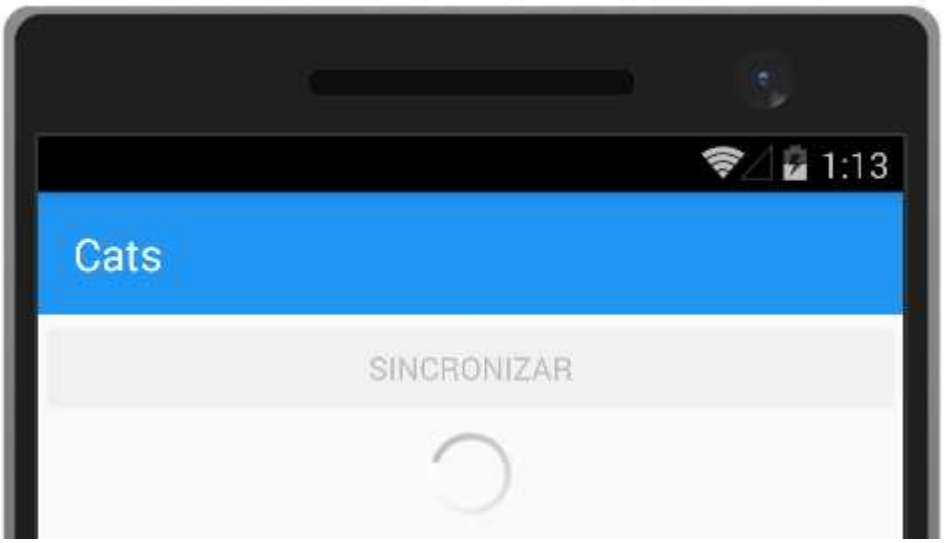
```
public App()
{
    // The root page of your application
    var content = new Views.CatsPage();

    MainPage = new NavigationPage(content);

}
```

Este código é o ponto de entrada da aplicação. O código simplesmente cria uma instância de **CatsPage** e a envolve em uma página de navegação para que seja mostrada ao usuário.

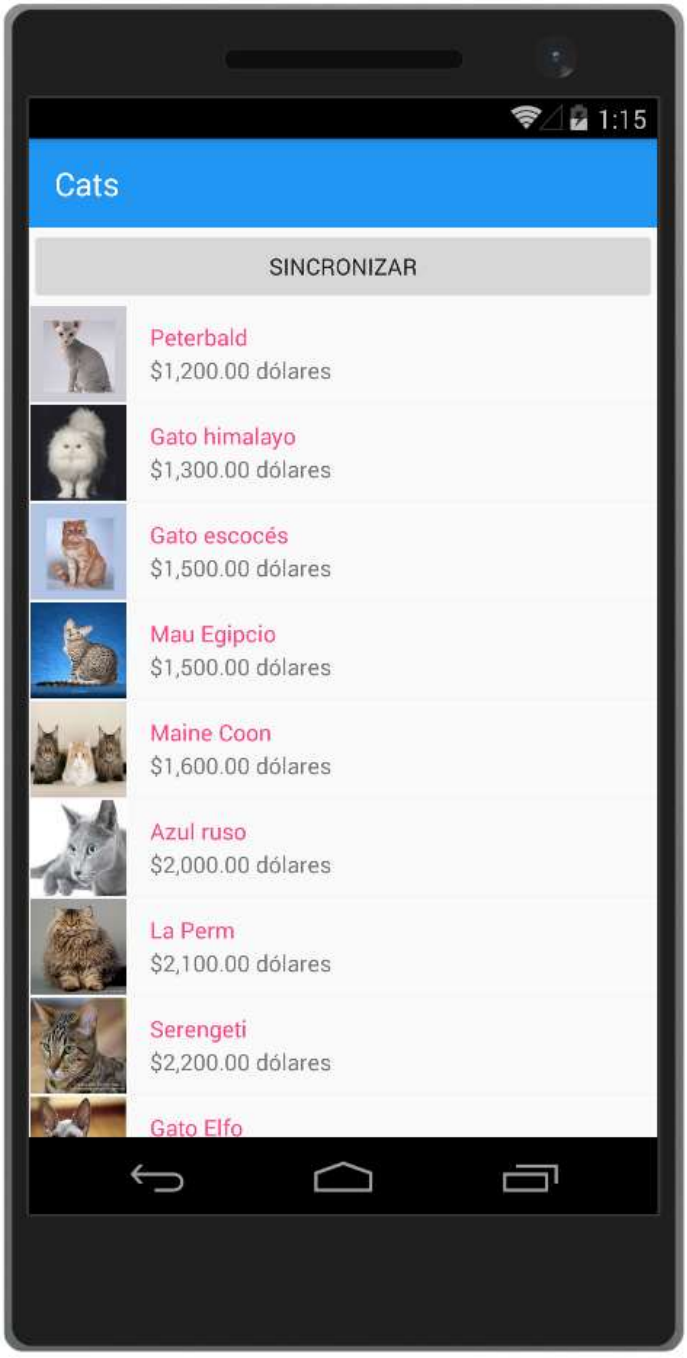
3. Selecione o projeto Android como projeto de início.
4. Execute a aplicação no emulador de sua preferência. Será exibida uma tela similar à seguinte:



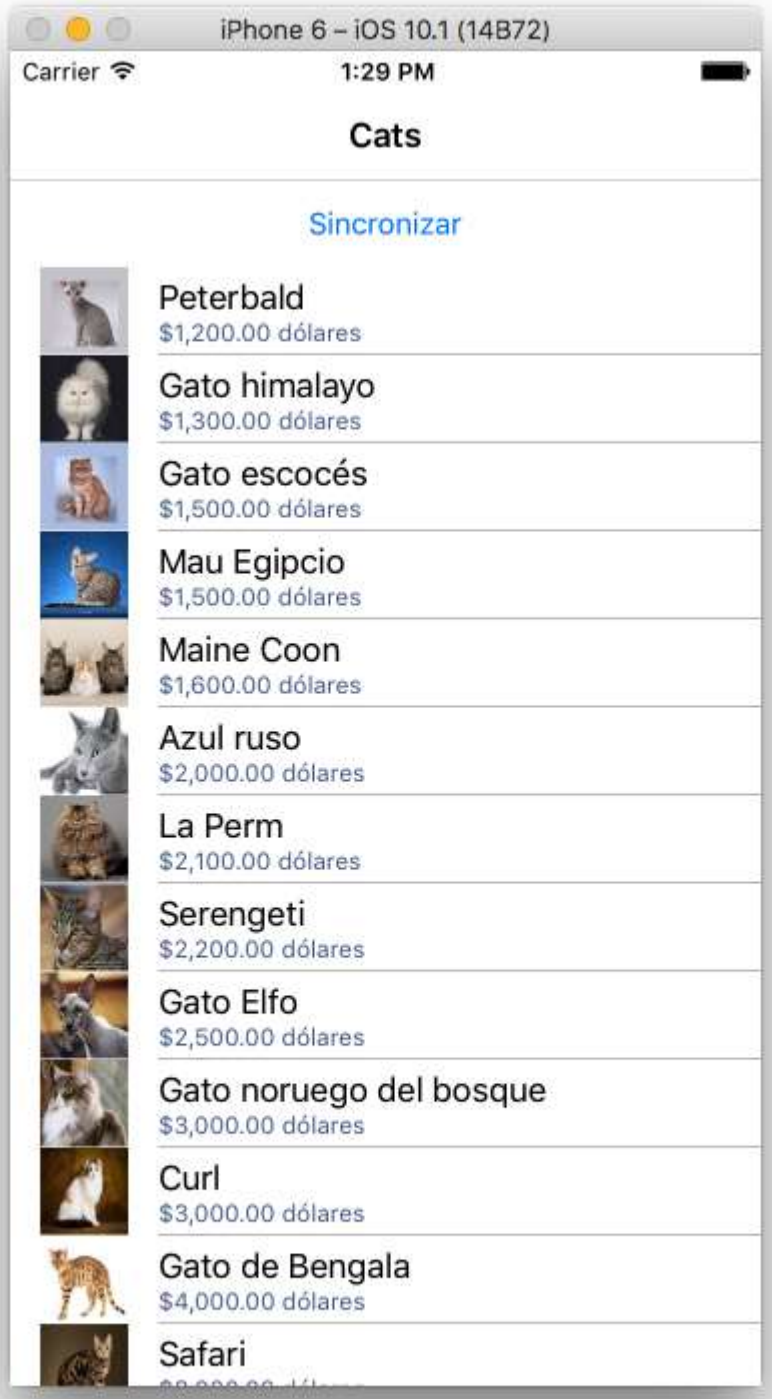
5. Toque o botão sincronizar. O indicador será mostrado.



Depois de carregar os dados a partir do serviço Web, será mostrada uma tela similar à seguinte.



6. Encerre a aplicação e retorne ao Visual Studio.
7. Teste a aplicação nas demais plataformas. A seguinte imagem mostra a aplicação executando-se no emulador de iOS.

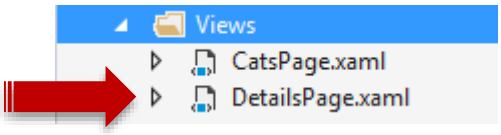


A seguinte imagem mostra a aplicação executando-se no emulador Windows 10 Mobile.

Tarefa 7. Adicionar a página de detalhes.

Agora é o momento para fazer alguma navegação e mostrar alguns detalhes dos dados.

1. Dentro do diretório **Views** adicione um novo elemento **Forms Xaml Page** chamado **DetailsPage.xaml**. Esta página permitirá mostrar o detalhe de um elemento da lista selecionado pelo usuário.



Tal como acontece com a página **CatsPage**, utilizaremos um **StackLayout** mas iremos colocá-lo dentro de um **ScrollView** se houver muito texto a ser apresentado.

2. Substitua o elemento **Label** pelo seguinte código.

```
<ScrollView Padding="10">
  <StackLayout Spacing="10">

  </StackLayout>
</ScrollView>
```

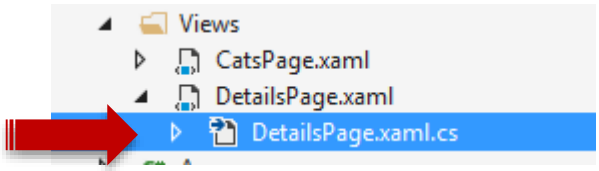
3. Dentro de **StackLayout** adicione agora o seguinte código para definir controles e ligações para as propriedades do objeto **Cat**.

```
<Image Source="{Binding Image}" HeightRequest="200" WidthRequest="200"/>
<Label Text="{Binding Name}" FontSize="24"/>
<Label Text="{Binding Price, StringFormat='{0:c} dólares'}" TextColor="Red"/>
<Label Text="{Binding Description}" />
```

4. Adicione o seguinte código para permitir ao usuário navegar no site Web do elemento selecionado.

```
<Button Text="Ir al Sitio Web" x:Name="ButtonWebSite"/>
```

5. Salve as alterações.
6. Abra o arquivo code-behind de **DetailsPage.xaml** chamado **DetailsPage.xaml.cs**.



7. Modifique o construtor da classe para que aceite como parâmetro um objeto **Cat** que representa o objeto selecionado pelo usuário.

```
public DetailsPage(Models.Cat selectedCat)
{
    InitializeComponent();

}
```

8. Adicione o seguinte código dentro da classe para declarar uma variável que armazene os dados do objeto **Cat** selecionado.

```
Models.Cat SelectedCat;
```

9. No construtor da classe, adicione o seguinte código para armazenar os dados do elemento selecionado e definir o contexto de ligação da página.

```
Models.Cat SelectedCat;
public DetailsPage(Models.Cat selectedCat)
{
    InitializeComponent();
    this.SelectedCat = selectedCat;
    BindingContext = this.SelectedCat;
 }
```

10. Xamarin.Forms tem predefinidas algumas APIs interessantes para funcionalidade multiplataforma, tal como abrir uma URL no navegador predeterminado.

    No construtor da classe, adicione o seguinte código para definir um manipulador do evento **Clicked** do botão **ButtonWebSite**.

```
ButtonWebSite.Clicked += ButtonWebSite_Clicked;
```

11. Adicione o seguinte código para implementar o manipulador do evento **Clicked** utilizando a classe **Device** para invocar o método **OpenUri**.

```
private void ButtonWebSite_Clicked(object sender, EventArgs e)
{
    if (SelectedCat.WebSite.StartsWith("http"))
    {
        Device.OpenUri(new Uri(SelectedCat.WebSite));
    }
}
```

12. Abra o arquivo code-behind de **CatsPage.xaml** chamado **CatsPage.xaml.cs**.
13. Adicione o seguinte código dentro do construtor da classe para definir um manipulador do evento **ItemSelected** do controle **ListViewCats**. Isto nos permitirá ser informados quando um elemento da lista for selecionado.

```
public CatsPage()
{
    InitializeComponent();
    ListViewCats.ItemSelected += ListViewCats_ItemSelected;
}
```

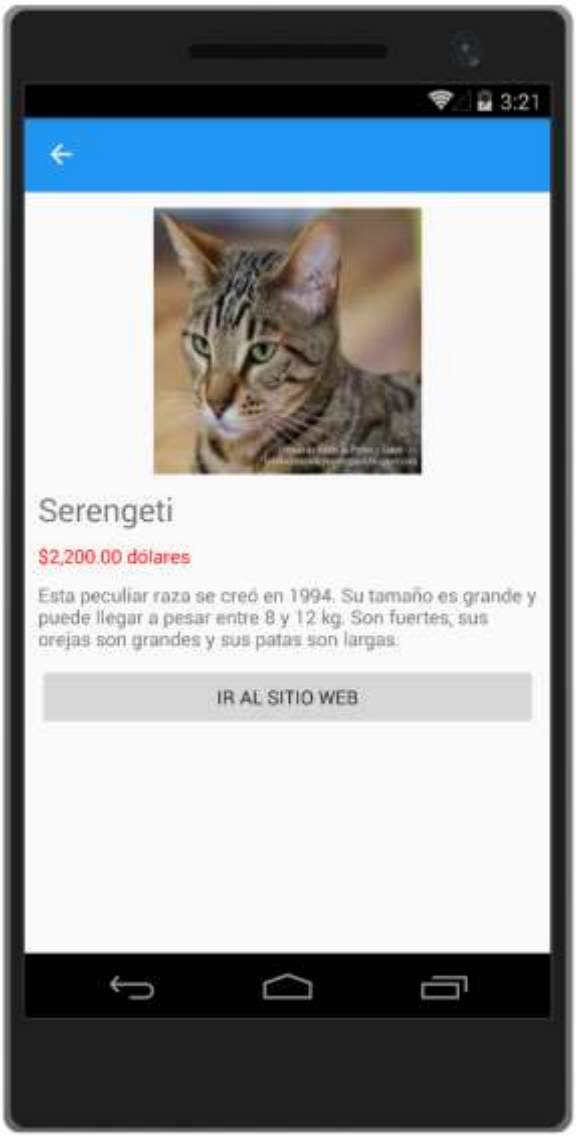
14. Adicione o seguinte código para implementar o manipulador do evento que permita navegar na página **DetailsPage**.

```
private async void ListViewCats_ItemSelected(object sender,
    SelectedItemChangedEventArgs e)
{
```

```
        var SelectedCat = e.SelectedItem as Models.Cat;
        if (SelectedCat != null)
        {
            await Navigation.PushAsync(new Views.DetailsPage(SelectedCat));
            ListViewCats.SelectedItem = null;
        }
    }
```

O código verifica se existe um elemento selecionado e depois utiliza a API predefinida **Navigation** para colocar (push) uma nova página. Finalmente, o código remove a seleção do elemento.

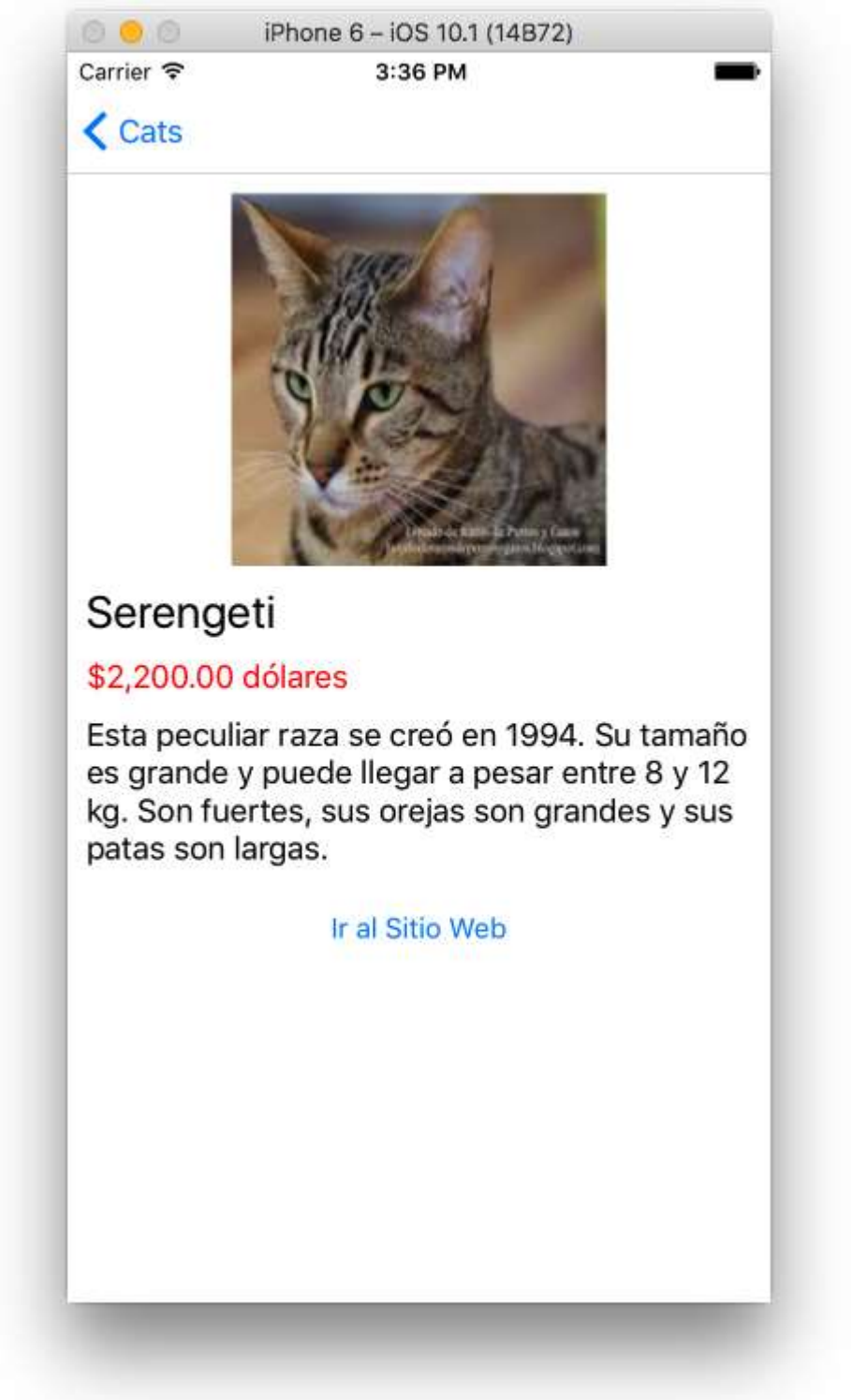
15. Execute novamente a aplicação em algum dos emuladores da plataforma Android.
16. Toque o botão sincronizar.
17. Toque um dos elementos. Note que a página **DetailsPage** aparece mostrando o detalhe do elemento selecionado.



18. Toque o botão **Ir ao Site Web**. Será mostrada a página Web solicitada.

19. Encerre a aplicação e retorne ao Visual Studio.

20. Teste a aplicação nas demais plataformas. A seguinte imagem mostra a aplicação executando-se no emulador de iOS.



A seguinte imagem mostra a aplicação executando-se em un emulador Windows 10 Mobile.


Serengeti
$2,200.00 dólares
Esta peculiar raza se creó en 1994. Su tamaño es grande y puede llegar a pesar entre 8 y 12 kg. Son fuertes, sus orejas son grandes y sus patas son largas.
Ir al Sitio Web

Resumen

Neste laboratório você desenvolveu uma aplicação Xamarin.Forms implementando o padrão MVVM.

A aplicação consome um serviço Web RESTful utilizando a classe **HttpClient**.

No laboratório seguinte você modificará o repositório de dados para consumir os dados a partir de uma aplicação backend hospedada em um **Azure App Service** do Microsoft Azure.

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

¡Finalizei o #Lab04 da #MaratonaXamarin e conheço a forma de consumir um serviço Web RESTful a partir de aplicações Xamarin.Forms implementando o padrão MVVM!