PAM-I

Programação de Aplicativos Mobile - I

Padrão MVVM

Objetivos de aprendizagem

Entender e implementar o padrão MVVM

O que são padrões de desenvolvimento?

Os padrões de projeto (*design patterns*) são soluções generalistas para problemas recorrentes durante o desenvolvimento de um software. Um Design Pattern é um conceito que serve como base para se desenvolver uma solução.

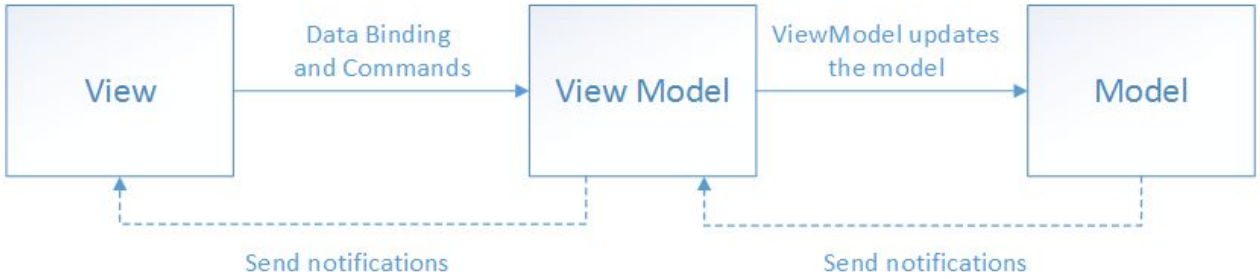
Para se implementar um padrão de projeto você deve **seguir o conceito dos padrões escolhidos (dentre todos os existentes)** e ajustá-lo ao problema que deseja resolver.

Por que usar o padrão MVVM

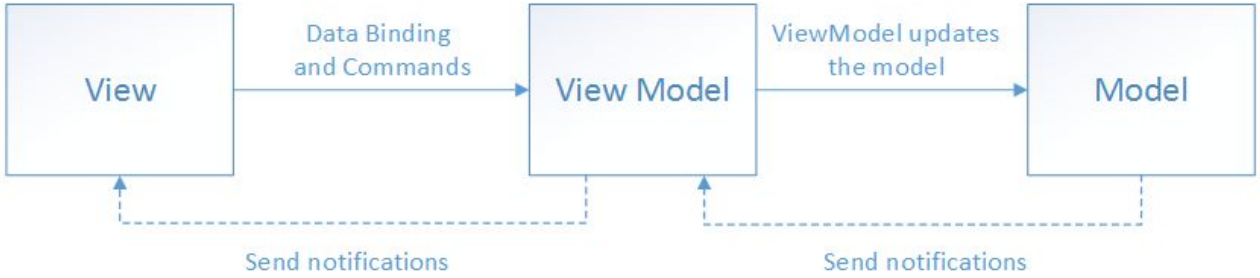
A experiência do desenvolvedor .NET MAUI normalmente envolve a criação de uma interface do usuário em XAML e, em seguida, a adição de code-behind que opera na interface do usuário.

Problemas complexos de manutenção podem surgir à medida que os aplicativos são modificados e crescem em tamanho e escopo, como um forte acoplamento entre os controles de interface do usuário e a lógica de negócios, o que aumenta o custo de fazer modificações na interface do usuário e a dificuldade de testar esse código.

O que é o padrão MVVM?



O que é o padrão MVVM?



Há três componentes principais no padrão MVVM: o modelo (*model)*, a exibição (*view)* e o modelo de exibição (*viewModel)*.

Model

Classes de modelo são classes que encapsulam os dados do aplicativo.
As *models* podem ser considerado como representando o modelo de domínio do aplicativo, que geralmente inclui um modelo de dados junto com a lógica de negócios e validação.

View

A View é responsável por definir a estrutura, o layout e a aparência do que o usuário vê na tela.
O ideal é que cada View seja definida em XAML, com um code-behind limitado que não contém lógica de negócios. No entanto, em alguns casos, o code-behind pode conter lógica de interface do usuário que implementa um comportamento visual difícil de expressar em XAML, como animações.

ViewModel

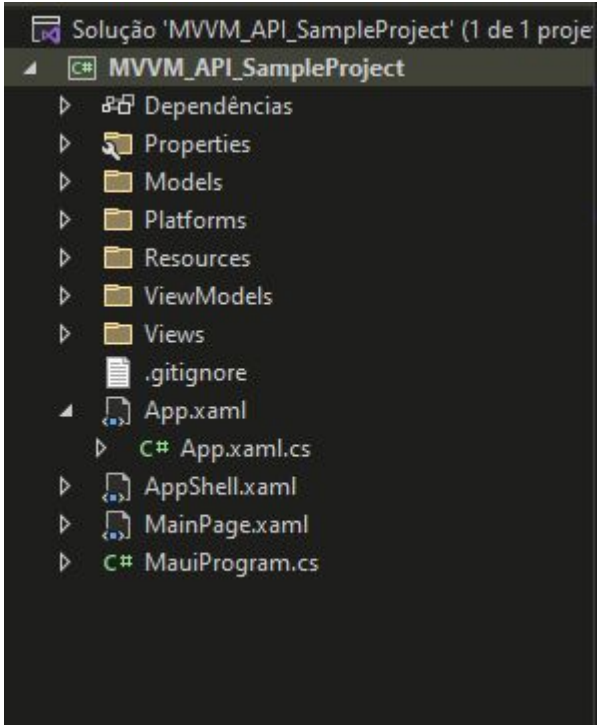
O modelo de exibição implementa propriedades e comandos aos quais a exibição pode associar dados e notifica a exibição de quaisquer alterações de estado por meio de eventos de notificação de alteração.
As propriedades e os comandos fornecidos pela viewModel definem a funcionalidade a ser oferecida pela interface do usuário, mas a view determina como essa funcionalidade deve ser exibida

Prática!
Entendendo como o MVVM funciona

Criação do projeto

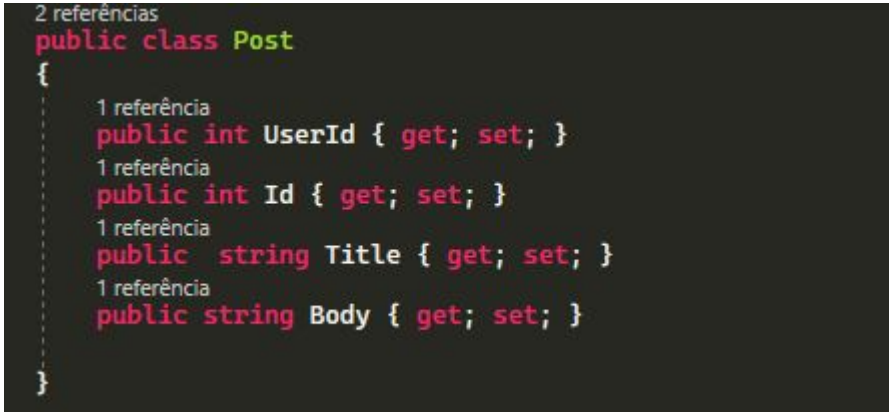
1. Crie um novo projeto chamado “Aula_MVVM-HTTPClient”.

2. Crie as pastas View, Model e ViewModel na sua estrutura da solução



Criação do projeto

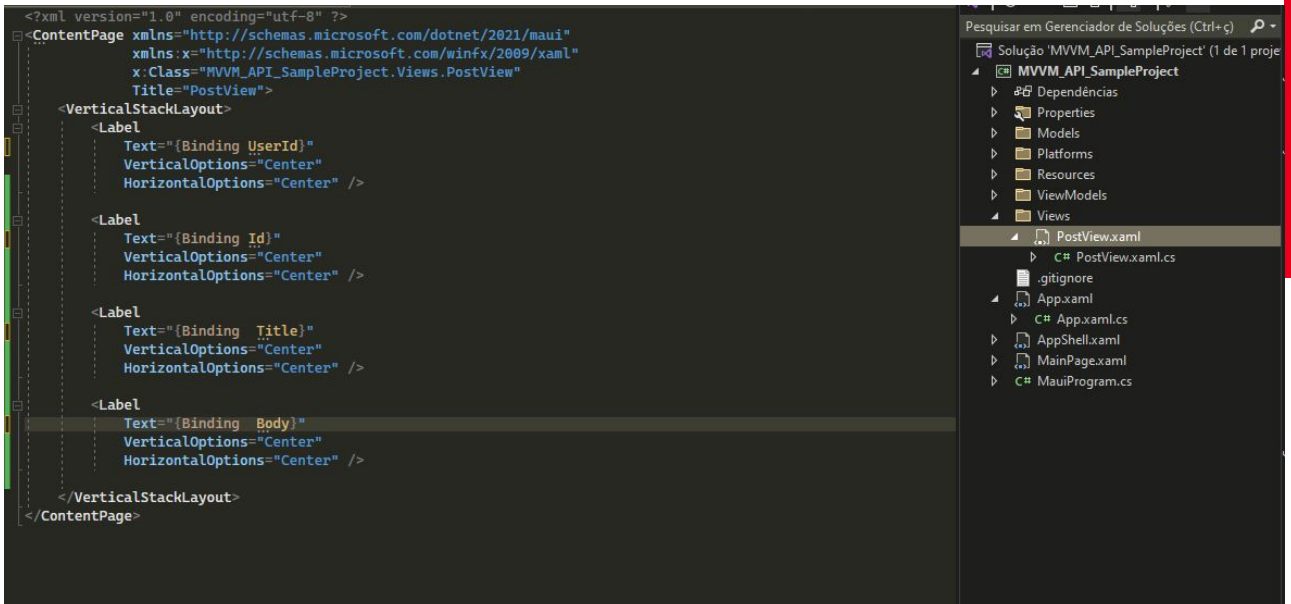
3. Crie uma nova classe dentro de Models com o nome Posts.



Criação do projeto

4. Crie uma nova View dentro da pasta Views chamada PostView.

5. Adicione controles para representar as informações de um Post.



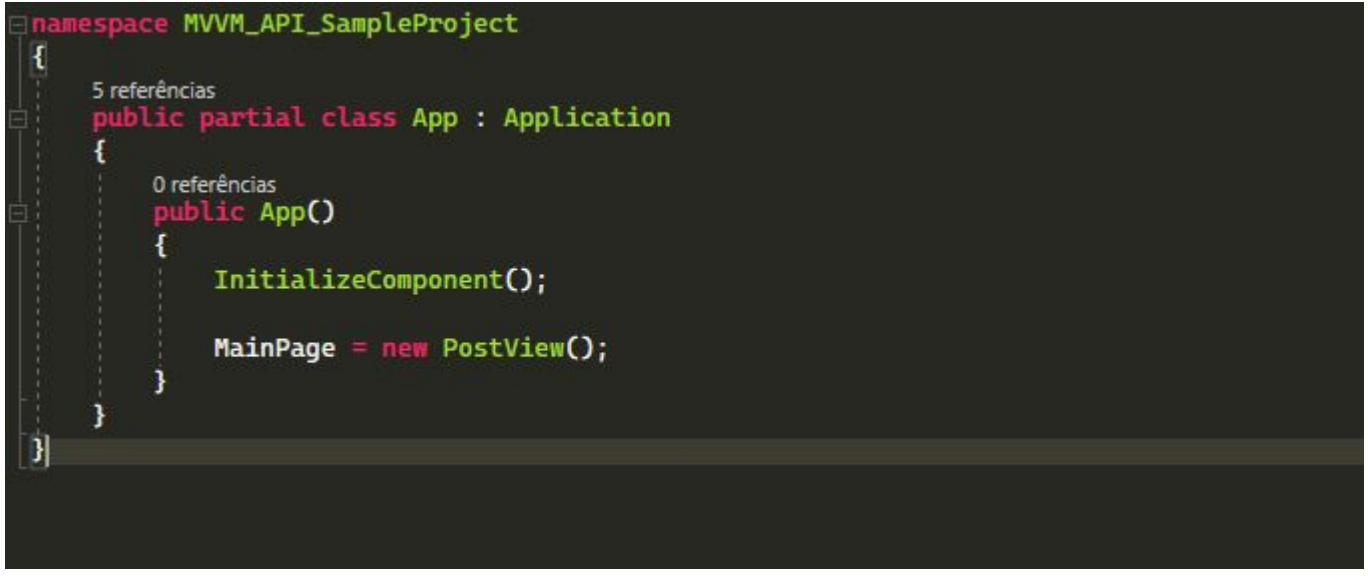
Criação do projeto

6. No Arquivo PostView.cs adicione um BindingContext chamado post para possibilitar a visualização dos dados pela View

```
5 referências
public partial class PostView : ContentPage
{
    1 referência
    public PostView()
    {
        InitializeComponent();
        var post = new Post
        {
            Id = 1,
            UserId = 1,
            Title = "Test",
            Body = "Test"
        };
        BindingContext = post;
    }
}
```

Criação do projeto

7. Modifique no Arquivo AppXaml.cs qual View deve ser exibida dentro da view principal e execute o código.



Atividade!

Criar uma view e uma modelagem seguindo o padrão MVVM que represente a seguinte estrutura de lista de tarefas:

ToDo:
Deve ter um ID de usuário associado a cad tarefa, um ID de tarefa, um título e um status de conclusão:


```
{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
 },
```


Na view, ID, UserID e Title podem ser Labels, status de conclusão deve ser um checkbox (ver documentação)

Referências e ajudas

https://learn.microsoft.com/pt-br/dotnet/architecture/maui/mvvm
https://coodesh.com/blog/dicionario/o-que-e-arquitetura-mvvm/
https://www.youtube.com/watch?v=waUne0fOz3s

CheckBox - .NET MAUI | Microsoft Learn

JsonSerializerOptions Class (System.Text.Json) | Microsoft Learn

Consumindo uma API Rest

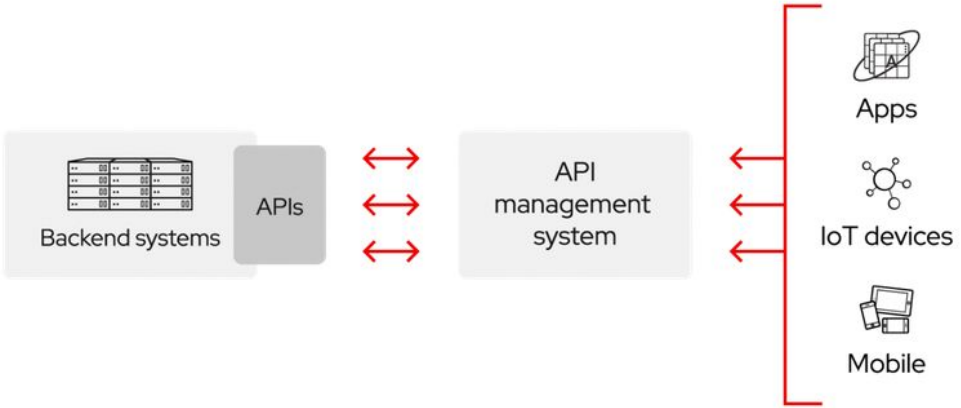
Objetivos de aprendizagem

Consumir uma API Rest através do http client

O que é uma API?

API significa **Application Programming Interface** e em resumo é um conjunto de definições e protocolos para criar e integrar aplicações.

As APIs funcionam como se fossem contratos, com documentações que representam um acordo entre as partes interessadas. Se uma dessas partes enviar uma solicitação remota estruturada de uma forma específica, isso determinará como a aplicação da outra parte responderá.



API Rest

API REST, também chamada de API RESTful, **é uma API que está em conformidade com as restrições do estilo de arquitetura REST** (Representational State Transfer).

Quando um cliente faz uma solicitação usando uma API RESTful, essa API **transfere uma representação do estado** do recurso ao solicitante ou endpoint. Essa informação é entregue via HTTP utilizando um dos vários formatos possíveis: Javascript Object Notation (JSON), HTML, XLT, Python, PHP ou texto sem formatação.

API Rest

A Arquitetura REST se baseia em 6 restrições:

- Arquitetura Cliente-Servidor
- Sem monitoramento de estado
- Capacidade de Cache
- Sistema em camadas
- Interface Uniforme

HTTP

HTTP é um protocolo que permite a obtenção de recursos, como documentos HTML. É a base de qualquer troca de dados na Web e um protocolo cliente-servidor, o que significa que as requisições são iniciadas pelo destinatário, geralmente um navegador da Web.

O protocolo HTTP define um conjunto de **métodos de requisição** responsáveis por indicar a ação a ser executada para um dado recurso. Cada um deles implementa uma semântica diferente e essa semântica deve ser respeitada durante a criação de Endpoints.

Métodos HTTP

`GET`: usado para solicitar dados de um de recursos.

`POST`: usado para enviar dados para um servidor para criar/atualizar um recurso.

`PUT`: usado para enviar dados para um servidor para criar/atualizar um recurso de maneira idempoente.

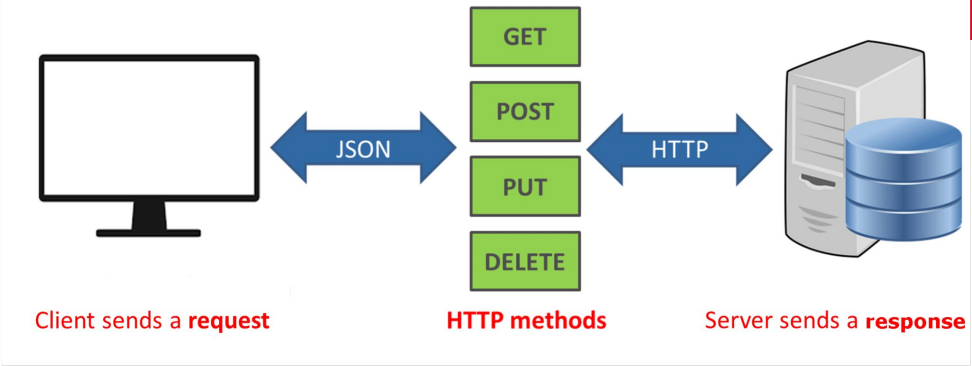
`DELETE`: exclui o recurso especificado.

`PATCH`: usado para aplicar modificações parciais em um recurso.

Padrão REST

REST (Transferência de Estado Representacional) é um estilo de arquitetura para a criação de serviços Web. As solicitações REST normalmente são feitas via HTTPS usando os mesmos verbos HTTP que os navegadores da Web usam para recuperar páginas da Web e enviar dados para servidores. Os verbos são:

- GET – essa operação é usada para recuperar dados do serviço Web.
- POST – essa operação é usada para criar um item de dados no serviço Web.
- PUT – essa operação é usada para atualizar um item de dados no serviço Web.
- PATCH – essa operação é usada para atualizar um item de dados no serviço Web, descrevendo um conjunto de instruções sobre como o item deve ser modificado.
- DELETE – essa operação é usada para excluir um item de dados no serviço Web.



Os serviços Web baseados em REST normalmente usam mensagens **JSON** para retornar dados ao cliente.

JSON é um formato de intercâmbio de dados baseado em texto que produz cargas compactas, o que resulta em requisitos de largura de banda reduzidos ao enviar dados.

A simplicidade do REST ajudou a torná-lo o método principal para acessar serviços Web em aplicativos móveis.

HTTP Client

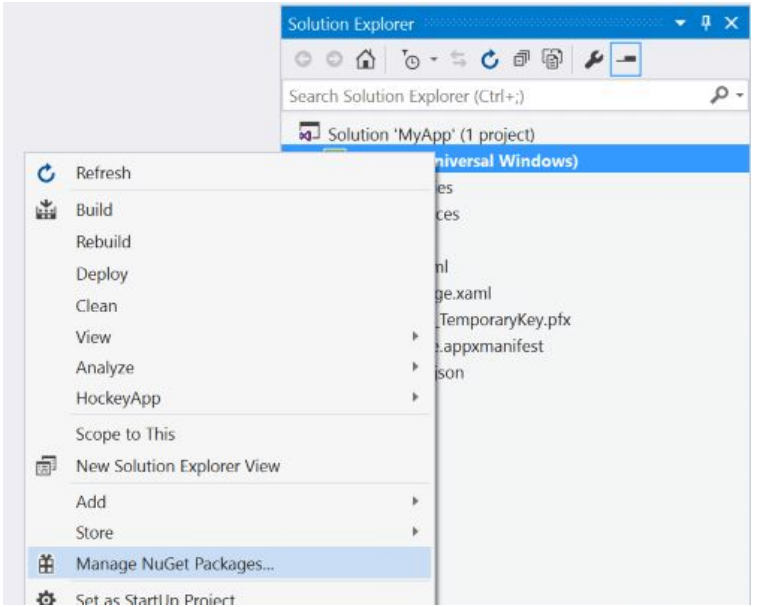
Um aplicativo .NET MAUI (interface do usuário de aplicativo multiplataforma) pode consumir um serviço Web baseado em REST enviando solicitações para o serviço Web com classe HTTPClient.

Essa classe fornece funcionalidade para enviar solicitações HTTP e receber respostas HTTP de um recurso identificado por URI.

Cada solicitação é enviada como uma operação **assíncrona**.

Comunity Toolkit mvvm

Instalando o Community Toolkit MVVM



Adaptando a viewModel para usar as requisições

```
namespace MVVM_API_SampleProject.ViewModels
{
    //Herdando ObservableObject do Community Toolkit Mvvm
    3 referências
    internal partial class PostViewModel : ObservableObject, IDisposable
    {
        HttpClient client;

        JsonSerializerOptions _serializerOptions;
        string baseUrl = "https://jsonplaceholder.typicode.com";

        [ObservableProperty]
        public int _UserId;
        [ObservableProperty]
        public int _Id;
        [ObservableProperty]
        public string _Title;
        [ObservableProperty]
        public string _Body;
        //Uma coleção de Post
        [ObservableProperty]
        public ObservableCollection<Post> _posts;


        1 referência
        public PostViewModel()
        {
            client = new HttpClient();
            Posts = new ObservableCollection<Post>();
            _serializerOptions = new JsonSerializerOptions { PropertyNameCaseInsensitive = true };
        }
```

Adaptando a viewModel para usar as requisições

```
    1 referência
    public PostViewModel()
    {
        client = new HttpClient();
        Posts = new ObservableCollection<Post>();
        _serializerOptions = new JsonSerializerOptions { PropertyNameCaseInsensitive = true };
    }

    //Consumir a API Rest -> Criação dos Commands

    0 referências
    public ICommand GetPostsCommand => new Command(async () => await LoadPostsAsync());

    1 referência
    private async Task LoadPostsAsync()
    {
        var url = $"{baseUrl}/posts";
        try
        {
            var response = await client.GetAsync(url);
            if (response.IsSuccessStatusCode)
            {
                string content = await response.Content.ReadAsStringAsync();
                Posts = JsonSerializer.Deserialize<ObservableCollection<Post>>(content, _serializerOptions);
            }
        }
        catch (Exception e)
        {
            Debug.WriteLine(e.Message);
        }
    }

    0 referências
    public void Dispose()
    {
        throw new NotImplementedException();
    }
}
```

Adaptando as views para mostrar uma coleção de posts

Criação de um Collection View

Criação de um botão com um *command*

```
<ScrollView>
    <VerticalStackLayout>
        <Entry Placeholder="Id Post" Text="{Binding Post.Id}"/>
        <Entry Placeholder="Titulo do Post" Text="{Binding Post.UserId}"/>
        <Entry Placeholder="Titulo do Post" Text="{Binding Post.Title}"/>
        <Entry Placeholder="Id Post" Text="{Binding Post.Body}"/>

        <HorizontalStackLayout>
            <Button Text="Carregar Posts" Command="{Binding GetPostsCommand}"/>
        </HorizontalStackLayout>

        <CollectionView x:Name="PostsCollection" ItemsSource="{Binding Posts}" EmptyView="Nenhum post encontrado">
            <CollectionView.ItemsLayout>
                <GridItemsLayout Orientation="Vertical"/>
            </CollectionView.ItemsLayout>

            <CollectionView.ItemTemplate>
                <DataTemplate>
                    <VerticalStackLayout>
                        <Frame >
                            <VerticalStackLayout Margin="50" HorizontalOptions="Center" MaximumWidthRequest="180">
                                <Label
                                    Text="{Binding UserId}"
                                    VerticalOptions="Center"
                                    HorizontalOptions="Center" />

                                <Label
                                    Text="{Binding Id}"
                                    VerticalOptions="Center"
                                    HorizontalOptions="Center" />

                                <Label
                                    Text="{Binding  Title}"
                                    VerticalOptions="Center"
                                    HorizontalOptions="Center" />

                                <Label
                                    Text="{Binding  Body}"
                                    VerticalOptions="Center"
                                    HorizontalOptions="Center" />
                            </VerticalStackLayout>
                        </Frame>
                    </VerticalStackLayout>
                </DataTemplate>
            </CollectionView.ItemTemplate>
        </CollectionView>
    </VerticalStackLayout>
</ScrollView>
```

```xml
<ScrollView>
    <VerticalStackLayout>
        <Entry Placeholder="Id Post" Text="{Binding Post.Id}"/>
        <Entry Placeholder="Título do Post" Text="{Binding Post.UserId}"/>
        <Entry Placeholder="Título do Post" Text="{Binding Post.Title}"/>
        <Entry Placeholder="Id Post" Text="{Binding Post.Body}"/>

        <HorizontalStackLayout>
            <Button Text="Carregar Posts" Command="{Binding GetPostsCommand}"/>
        </HorizontalStackLayout>

        <CollectionView x:Name="PostsCollection" ItemsSource="{Binding Posts}" EmptyView="Nenhum post encontrado">
            <CollectionView.ItemsLayout>
                <GridItemsLayout Orientation="Vertical"/>
            </CollectionView.ItemsLayout>

            <CollectionView.ItemTemplate>
                <DataTemplate>
                    <VerticalStackLayout>
                        <Frame >
                            <VerticalStackLayout Margin="50" HorizontalOptions="Center" MaximumWidthRequest="180">
                                <Label
                                    Text="{Binding UserId}"
                                    VerticalOptions="Center"
                                    HorizontalOptions="Center" />

                                <Label
                                    Text="{Binding Id}"
                                    VerticalOptions="Center"
                                    HorizontalOptions="Center" />

                                <Label
                                    Text="{Binding  Title}"
                                    VerticalOptions="Center"
                                    HorizontalOptions="Center" />

                                <Label
                                    Text="{Binding  Body}"
                                    VerticalOptions="Center"
                                    HorizontalOptions="Center" />
                            </VerticalStackLayout>
                        </Frame>
                    </VerticalStackLayout>
                </DataTemplate>
            </CollectionView.ItemTemplate>
        </CollectionView>
    </VerticalStackLayout>
</ScrollView>
```

Organizando a aplicação

Separação das requisições HTTP em uma classe *usuário Service*

Rest Service

```
public class RestService
{
    HttpClient _client;
    JsonSerializerOptions _serializerOptions;

    public List<Classe> Items { get; private set; }

    public RestService()
    {
        _client = new HttpClient();
        _serializerOptions = new JsonSerializerOptions
        {
            PropertyNamingPolicy = JsonNamingPolicy.CamelCase,
            WriteIndented = true
        };
    }
      …
}
```

Recuperando dados

```
public async Task<List<Model>> GetDataAsync()
{
    Items = new List<Model>();

    Uri uri = new Uri("BASEURL");
    try
    {
        HttpResponseMessage response = await _client.GetAsync(uri);
        if (response.IsSuccessStatusCode)
        {
            string content = await response.Content.ReadAsStringAsync();
            Items = JsonSerializer.Deserialize<List<TodoItem>>(content, _serializerOptions);
        }
    }
    catch (Exception ex)
    {
       …
    }

    return Items;
}
```