ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE II – 2023/1

Luiz Fernando Souza / Elaine Marion

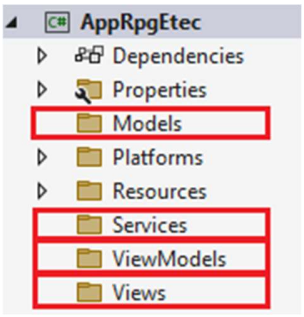
Aula 13 – Criação do Projeto AppRpgEtec – Listagem de Personagens

Crie um projeto **.NET MAUI App** com as seguintes recomendações:

Project name: **AppRpgEtec -** Framework: .**NET 7.0**

Faremos criação das classes que vão consumir a API e alimentar as demais camadas, esta camada se chamará *Services* e ficará no projeto C#. Crie uma pasta com o nome ***Services***. Também crie as pastas ***Models***, ***ViewModels*** e ***Views*** dentro do projeto C#. Usaremos essa estrutura de pastas no projeto genérico para que a divisão de tarefas fique visivelmente organizada.

1. Crie as seguintes pastas clicando com direito no projeto C#



2. Crie uma pasta chamada **Enuns** dentro da pasta Models, adicionando uma classe **ClasseEnum** dentro da pasta Enuns, e transformando em um *enum* conforme abaixo

```
public enum ClasseEnum
{
    NaoSelecionada = 0,
    Cavaleiro = 1,
    Mago = 2,
    Clerigo = 3
}
```

3. Cria uma classe **Personagem** dentro da pasta Models e adicione as propriedades a seguir. Faça o using de AppRgpEtec.Models.Enuns

```
public int Id { get; set; }
        public string Nome { get; set; }
        public int PontosVida { get; set; }
        public int Forca { get; set; }
        public int Defesa { get; set; }
        public int Inteligencia { get; set; }
        public ClasseEnum Classe { get; set; }
```

4. Crie uma classe **Request** dentro da pasta **Services** e programe os métodos abaixo

```
public async Task<int> PostReturnIntAsync<TResult>(string uri, TResult data)
        {
            HttpClient httpClient = new HttpClient();

            var content = new StringContent(JsonConvert.SerializeObject(data));
            content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
            HttpResponseMessage response = await httpClient.PostAsync(uri, content);

            string serialized = await response.Content.ReadAsStringAsync();

            if (response.StatusCode == System.Net.HttpStatusCode.OK)
                return int.Parse(serialized);
            else
                return 0;
        }

        public async Task<TResult> PostAsync<TResult>(string uri, TResult data, string token)
        {
            HttpClient httpClient = new HttpClient();
            httpClient.DefaultRequestHeaders.Authorization
                = new AuthenticationHeaderValue("Bearer", token);

            var content = new StringContent(JsonConvert.SerializeObject(data));
            content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
            HttpResponseMessage response = await httpClient.PostAsync(uri, content);

            string serialized = await response.Content.ReadAsStringAsync();
            TResult result = data;

            if (response.StatusCode == System.Net.HttpStatusCode.OK)
                result = await Task.Run(() => JsonConvert.DeserializeObject<TResult>(serialized));

            return result;
        }

        public async Task<int> PostReturnIntTokenAsync<TResult>(string uri, TResult data, string
token)
        {
            HttpClient httpClient = new HttpClient();
            httpClient.DefaultRequestHeaders.Authorization
                = new AuthenticationHeaderValue("Bearer", token);

            var content = new StringContent(JsonConvert.SerializeObject(data));
            content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
            HttpResponseMessage response = await httpClient.PostAsync(uri, content);

            string serialized = await response.Content.ReadAsStringAsync();

            if (response.StatusCode == System.Net.HttpStatusCode.OK)
                return int.Parse(serialized);
            else
                return 0;
        }
```

```
        public async Task<int> PutAsync<TResult>(string uri, TResult data, string token)
        {
            HttpClient httpClient = new HttpClient();
            httpClient.DefaultRequestHeaders.Authorization
                = new AuthenticationHeaderValue("Bearer", token);

            var content = new StringContent(JsonConvert.SerializeObject(data));
            content.Headers.ContentType = new MediaTypeHeaderValue("application/json");
            HttpResponseMessage response = await httpClient.PutAsync(uri, content);

            string serialized = await response.Content.ReadAsStringAsync();

            if (response.StatusCode == System.Net.HttpStatusCode.OK)
                return int.Parse(serialized);
            else
                return 0;
        }

        public async Task<TResult> GetAsync<TResult>(string uri, string token)
        {
            HttpClient httpClient = new HttpClient();
            httpClient.DefaultRequestHeaders.Authorization
                = new AuthenticationHeaderValue("Bearer", token);

            HttpResponseMessage response = await httpClient.GetAsync(uri);
            string serialized = await response.Content.ReadAsStringAsync();
            TResult result = await Task.Run(() =>
                JsonConvert.DeserializeObject<TResult>(serialized));
            return result;
        }

        public async Task<int> DeleteAsync(string uri, string token)
        {
            HttpClient httpClient = new HttpClient();
            httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", token);
            HttpResponseMessage response = await httpClient.DeleteAsync(uri);
            string serialized = await response.Content.ReadAsStringAsync();
            if (response.StatusCode == System.Net.HttpStatusCode.OK)
                return int.Parse(serialized);
            else
                return 0;
            }
```

- Clique com o direito em JsonConvert ou em CTRL + . (Ponto) e escolha Install package Newtonsoft.Json. Isso instalará a última versão da biblioteca sugerida.

- Este método será o primeiro método genérico para consumir APIs e será usado pelas demais classes de serviço.

- Confira se será necessário using de *System.Threading.Tasks*, *System.Net.Http* e *System.Net.Http.Headers*;

5. Crie uma pasta chamada **Personagens** dentro da pasta Services, e crie a classe chamada **PersonagemService** dentro da pasta recém-criada. Adicione a herança para a classe service, a variável global (_request) que representa a instancia da requisição a string que guardará o caminho da sua API, a variável global de token e o construtor que receberá o token passado para alimentar o token da classe.

```
public class PersonagemService : Request
{
    private readonly Request _request;
    private const string apiUrlBase = "http://xyz.somee.com/RpgApi/Personagens";
    //xyz --> Site da sua API

    private string _token;
    0 references
    public PersonagemService(string token)
    {
        _request = new Request();
        _token = token;
    }

    //Próximos métodos aqui

}
```

6. Adicione os métodos que consomem a API com o usings que o Visual Studio sugerir. Faça o using de AppRpgEtec.Models

```
public async Task<ObservableCollection<Personagem>> GetPersonagensAsync()
        {
            string urlComplementar = string.Format("{0}", "/GetAll");
            ObservableCollection<Models.Personagem> listaPersonagens = await
 _request.GetAsync<ObservableCollection<Models.Personagem>>(apiUrlBase + urlComplementar, _token);
            return listaPersonagens;
        }
        public async Task<Personagem> GetPersonagemAsync(int personagemId)
        {
            string urlComplementar = string.Format("/{0}", personagemId);
            var personagem = await _request.GetAsync<Models.Personagem>(apiUrlBase +
urlComplementar, _token);
            return personagem;
        }
public async Task<int> PostPersonagemAsync(Personagem p)
        {
            return await _request.PostReturnIntTokenAsync(apiUrlBase, p, _token);
        }
public async Task<int> PutPersonagemAsync(Personagem p)
        {
            var result = await _request.PutAsync(apiUrlBase, p, _token);
            return result;
        }
        public async Task<int> DeletePersonagemAsync(int personagemId)
        {
            string urlComplementar = string.Format("/{0}", personagemId);
            var result = await _request.DeleteAsync(apiUrlBase + urlComplementar, _token);
            return result;
        }
```

Na pasta ViewModels, crie a classe **BaseViewModel**. Como trabalharemos com o padrão *MVVM*, estruturaremos a classe *BaseViewModel* para centralizar o método *OnPropertyChanged* para as demais classes *ViewModel*. Essa configuração, como vimos, é a que reflete as alterações das classes espontaneamente para as *Views* e vice-versa.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Text;

namespace AppRpgHAS.ViewModels
{
    1 reference
    public class BaseViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        0 references
        public void OnPropertyChanged([CallerMemberName] string name = "")
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
        }
    }
}
```

- A programação exigirá os *usings* sinalizados em verde. Verifique se eles já existem.

7. Cria uma pasta **Personagens** dentro da pasta ViewModels. Adicione uma classe com o nome **ListagemPersonagemViewModel.cs** herdando a classe *BaseViewModel* e deixando a classe pública. Prossiga na viewModel realizando a programação a seguir dentro da classe

```
public class ListagemPersonagemViewModel : BaseViewModel
{
    1 private PersonagemService pService;
      1 reference
    2 public ObservableCollection<Personagem> Personagens { get; set; }
      0 references
    3 public ListagemPersonagemViewModel()
      {
          string token = Preferences.Get("UsuarioToken", string.Empty);
          pService = new PersonagemService(token);
          Personagens = new ObservableCollection<Personagem>();
      }

    //Próximos elementos da classe aqui

}//Fim da classe
```

(1) Declaração da variável de serviço que consumirá a API
(2) Declaração da Coleção de Personagens como propriedade
(3) Construtor pegando token, inicializando o serviço passando o token e inicializando a lista de personagens

- Usings de *AppRpg.Models; AppRpg.Services.Personagens e System.Collections.ObjectModel*

8. Programe o método de busca dos personagens.

```
public async Task ObterPersonagens()
{
    try //Junto com o Cacth evitará que erros fechem o aplicativo
    {
        Personagens = await pService.GetPersonagensAsync();
        OnPropertyChanged(nameof(Personagens)); //Informará a View que houve carregamento
    }
    catch (Exception ex)
    {
        //Captará o erro para exibir em tela
        await Application.Current.MainPage
            .DisplayAlert("Ops", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
```

9. Adicione no construtor a chamada para o método

```
public ListagemPersonagemViewModel()
{
    string token = Preferences.Get("UsuarioToken", string.Empty);
    pService = new PersonagemService(token);
    Personagens = new ObservableCollection<Personagem>();

    _ = ObterPersonagens();
}
```

- O "_" (underline) descarta a operação assíncrona de usar o operador await para armazenar um retorno.

10. Crie uma pasta **Personagens** dentro da pasta Views e dentro da pasta Persoagens crie uma view (.NET Maui) chamada **ListagemView.xaml**, remova o VerticalStackLayout que está dentro da tag ContentPage.Content e adicione o conteúdo abaixo.

```
<VerticalStackLayout Padding="10, 0, 0, 0" VerticalOptions="FillAndExpand">
        <ListView x:Name="listView" HasUnevenRows="True" ItemsSource="{Binding Personagens}" >
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <StackLayout Padding="10">
                            <Label Text="{Binding Nome}" FontSize="18" FontAttributes="Bold"/>
                            <Label Text="{Binding PontosVida}" FontSize="14"/>
                        </StackLayout>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
        </VerticalStackLayout>
```

11. Na parte de código da *view*, declare a viewModel de personagem, inserindo o using para a pasta da viewModel, faça as inicializações necessárias no construtor e atribua ao contexto da view para que seja feita a operação para trazer a lista de personagens

```
public partial class ListagemView : ContentPage
{
    ListagemPersonagemViewModel viewModel;

    1 reference
    public ListagemView()
    {
        InitializeComponent();

        viewModel = new ListagemPersonagemViewModel();
        BindingContext = viewModel;
        Title = "Personagens - App Rpg Etec";
    }
}
```

- Abra o arquivo App.xaml.cs para definir a view **ListagemView.xaml** como inicial. Será necessário o using AppRpgEtec.Views.Personagens.

- Abra a classe **MainApplication**, da pasta Plataforms/Android e habilite o emulador trafegar dados JSON conforme a instrução sinalizada abaixo:

```
[Application(UsesCleartextTraffic=true)]
1 reference
public class MainApplication : MauiApplication
{
```

- Execute o aplicativo e confirme que os registros serão listados. Abaixo está o resultado esperado