



Aula 15 - Consumindo APIs - Cadastro de Personagens do jogo RPG

1. Na pasta models crie uma classe chamada **TipoClasse** e codifique conforme abaixo:

```
public class TipoClasse
{
    0 references
    public int Id { get; set; }
    0 references
    public string Descricao { get; set; }
}
```

2. Crie a classe **CadastroPersonagemViewModel.cs** na pasta ViewModels/Personagens. Usings de *AppRpgHAS.Servicos.Personagens*;

```
public class CadastroPersonagemViewModel : BaseViewModel
{
    1 private PersonagemService pService;

    0 references
    2 public CadastroPersonagemViewModel()
    {
        string token = Preferences.Get("UsuarioToken", string.Empty);
        pService = new PersonagemService(token);
    }
}
```

- (1) Declaração da propriedade que referência a classe de serviço e (2) Criação do Construtor para receber o objeto que será enviado da View para alimentar esta classe, resgate do token da aplicação para passar para a instância do serviço.

3. Crie os atributos abaixo, depois selecione todos eles de uma vez e crie as propriedades (CTRL + R,E)

```
private int id;
private string nome;
private int pontosVida;
private int forca;
private int defesa;
private int inteligencia;
private int disputas;
private int vitorias;
private int derrotas;
```



4. Altere as propriedades para que fiquem conforme o modelo. Se atente ao que se trata de atributo e o que se trata de propriedade conforme as cores. Repita a alteração nas demais propriedades.

```
public int Id
{
    get => id;
    set
    {
        id = value;
        OnPropertyChanged();
    }
}
```

5. Criaremos um atributo e propriedade que se trata de uma coleção de **TipoClasse**. Exigirá o using de *AppRpgEtec.Models*, e *System.Collections.ObjectModel*.

```
private ObservableCollection<TipoClasse> listaTiposClasse;
0 references
public ObservableCollection<TipoClasse> ListaTiposClasse
{
    get { return listaTiposClasse; }
    set
    {
        if (value != null)
        {
            listaTiposClasse = value;
            OnPropertyChanged();
        }
    }
}
```

6. Programe o método que carregará as classes na View. Exigirá o using de *System.Threading.Tasks*

```
public async Task ObterClasses()
{
    try
    {
        ListaTiposClasse = new ObservableCollection<TipoClasse>();
        ListaTiposClasse.Add(new TipoClasse() { Id = 1, Descricao = "Cavaleiro" });
        ListaTiposClasse.Add(new TipoClasse() { Id = 2, Descricao = "Mago" });
        ListaTiposClasse.Add(new TipoClasse() { Id = 3, Descricao = "Clerigo" });
        OnPropertyChanged(nameof(ListaTiposClasse));
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage
            .DisplayAlert("Ops", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
```



7. Volte até o construtor da página e faça a chamada para o método recém-criado. Isso fará com que quando o objeto viewModel for criado em memória, já carreguemos a lista de classes no mesmo momento.

```
public CadastroPersonagemViewModel()  
{  
    string token = Preferences.Get("UsuarioToken", string.Empty);  
    pService = new PersonagemService(token);  
    _ = ObterClasses();  
}
```

8. Crie um atributo e uma propriedade que armazenará o tipo de classe selecionado pelo usuário na view.

```
private TipoClasse tipoClasseSelecionado;  
0 references  
public TipoClasse TipoClasseSelecionado  
{  
    get { return tipoClasseSelecionado; }  
    set  
    {  
        if (value != null)  
        {  
            tipoClasseSelecionado = value;  
            OnPropertyChanged();  
        }  
    }  
}
```



9. Crie o método que acionará a classe de Serviços para salvar o Personagem. Exigirá o using para *AppRpgEtec.Models.Enums* e *System*.

```
public async Task SalvarPersonagem()
{
    try
    {
        Personagem model = new Personagem()
        {
            Nome = this.nome,
            PontosVida = this.pontosVida,
            Defesa = this.defesa,
            Derrotas = this.derrotas,
            Disputas = this.disputas,
            Forca = this.forca,
            Inteligencia = this.inteligencia,
            Vitorias = this.vitorias,
            Id = this.id,
            Classe = (ClasseEnum)tipoClasseSelecionado.Id
        };
        if (model.Id == 0)
            await pService.PostPersonagemAsync(model);

        await Application.Current.MainPage
            .DisplayAlert("Mensagem", "Dados salvos com sucesso!", "Ok");

        await Shell.Current.GoToAsync(".."); //Remove a página atual da pilha de páginas
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage
            .DisplayAlert("Ops", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
```



10. Volte ao topo da classe e declare um ICommand chamado SalvarCommand (1). Isso exigirá o using de System.Windows.Input. Faça a vinculação do ICommand ao método SalvarPersonagem (2)

```
1 reference
1 public ICommand SalvarCommand { get; }

1 reference
public CadastroPersonagemViewModel()
{
    string token = Preferences.Get("UsuarioToken", string.Empty);
    pService = new PersonagemService(token);
    _ = ObterClasses();

2 SalvarCommand = new Command(async () => { await SalvarPersonagem(); });
}
```

11. Crie uma *content page* (selecione **.NET MAUI** no menu a esquerda) chamada **CadastroPersonagemView.xaml** na pasta Views/Personagens, remova o VerticalStackLayout junto com a label que existia originalmente e insira o layout abaixo dentro da tag *ContentPage*. Observe que os campos têm uma propriedade Keyboard para que o teclado seja adaptado ao tipo de caractere que queremos ter na entrada de dados.

```
<ScrollView>
    <VerticalStackLayout Spacing="3" Padding="15">
        <Label Text="Nome" FontSize="Medium" />
        <Entry Text="{Binding Nome}" FontSize="Medium" />

        <Label Text="Classe" FontSize="Medium" />
        <Picker Title="---Selecione---" ItemsSource="{Binding ListaTiposClasse}"
ItemDisplayBinding="{Binding Descricao}" SelectedItem="{Binding TipoClasseSelecionado}" />

        <Label Text="Pontos de Vida" FontSize="Medium" />
        <Entry Text="{Binding PontosVida}" FontSize="Medium" Keyboard="Numeric" />

        <Label Text="Força" FontSize="Medium" />
        <Entry Text="{Binding Forca}" FontSize="Medium" Keyboard="Numeric" />

        <Label Text="Defesa" FontSize="Medium" />
        <Entry Text="{Binding Defesa}" FontSize="Medium" Keyboard="Numeric" />

        <Label Text="Inteligência" FontSize="Medium" />
        <Entry Text="{Binding Inteligencia}" FontSize="Medium" Keyboard="Numeric" />

        <Label Text="Disputas" FontSize="Medium" />
        <Entry Text="{Binding Disputas}" FontSize="Medium" Keyboard="Numeric" />

        <Label Text="Vitórias" FontSize="Medium" />
        <Entry Text="{Binding Vitorias}" FontSize="Medium" Keyboard="Numeric" />

        <Label Text="Derrotas" FontSize="Medium" />
        <Entry Text="{Binding Derrotas}" FontSize="Medium" Keyboard="Numeric" />

        <HorizontalStackLayout Spacing="20">
            <Button Text="Salvar" Command="{Binding SalvarCommand}"></Button>
        </HorizontalStackLayout>
    </VerticalStackLayout>
</ScrollView>
```




12. Na parte de Código da *View* realize a programação abaixo para declarar a classe *ViewModel*, inicializar o atributo vinculando à *View* e carregar os tipos de Classes.

```
private CadastroPersonagemViewModel cadViewModel;  
0 references  
public CadastroPersonagemView()  
{  
    InitializeComponent();  
  
    cadViewModel = new CadastroPersonagemViewModel();  
    BindingContext = cadViewModel;  
    Title = "Novo Personagem";  
}
```

13. Abra a classe da view *AppShell.xaml.cs* e programe uma rota de referência para a view de cadastro.

```
public AppShell()  
{  
    InitializeComponent();  
  
    Routing.RegisterRoute("cadPersonagemView", typeof(CadastroPersonagemView));  
}
```

14. Vá até a classe *ViewModel* de listagem de personagem (*ListagemPersonagemViewModel.cs*) para adicionar o método que abrirá a tela de cadastro. Using de *Xamarin.Forms*

```
public async Task ExibirCadastroPersonagem()  
{  
    try..  
    {  
        await Shell.Current.GoToAsync("cadPersonagemView");..  
    }  
    catch (Exception ex)  
    {  
        await Application.Current.MainPage.  
            .DisplayAlert("Ops", ex.Message + " Detalhes: " + ex.InnerException, "Ok");  
    }  
}
```

15. Suba até o topo da classe para declarar um *ICommand* (1) e vinculá-lo ao método criado na etapa anterior

```
public ListagemPersonagemViewModel()  
{  
    string token = Preferences.Get("UsuarioToken", string.Empty);  
    pService = new PersonagemService(token);  
    Personagens = new ObservableCollection<Personagem>();  
    _ = ObterPersonagens();  
  
    2 NovoPersonagem = new Command(async () => { await ExibirCadastroPersonagem(); });  
}  
1 reference  
1 public ICommand NovoPersonagemCommand { get; }
```



16. Vá até a view de Listagem de Personagens (Views/Personagens/ListagemView.xaml) e inclua um botão para adição de participantes na **view**.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="AppRpgEtec.Views.Personagens.ListagemView"
    Title="ListagemView">

    <Shell.TitleView>
        <Button Command="{Binding NovoPersonagemCommand}" Text="Novo" HorizontalOptions="End"></Button>
    </Shell.TitleView>

    <ScrollView>
        <VerticalStackLayout Padding="10, 0, 0, 0" VerticalOptions="FillAndExpand">
```

17. Altere o método de autenticação na classe UsuarioViewModel para que a AppShell.xaml seja a view que apareça após o login

```
Preferences.Set("UsuarioPerfil", uAutenticado.Perfil);

await Application.Current.MainPage
    .DisplayAlert("Informação", mensagem, "Ok");

Application.Current.MainPage = new AppShell();
```

- Abra o AppShell, faça referência a pasta das Views de Personagens (A) e altere o ContentTemplate para que a listagem de personagens seja chamada pelo AppShell.

```
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:local="clr-namespace:AppRpgEtec"
xmlns:viewsPersonagens="clr-namespace:AppRpgEtec.Views.Personagens"
Shell.FlyoutBehavior="Disabled"
Title="AppRpgEtec">

<ShellContent
    Title="Home"
    ContentTemplate="{DataTemplate viewsPersonagens:ListagemView}"
    Route="MainPage" />
```

- Execute o aplicativo, realize o cadastro e confirme que os dados aparecerão na listagem e no banco de dados.
- Caso ao salvar um personagem, você recebe a mensagem de confirmação, mas a lista não seja atualizada na view, programe na parte de código da view de personagens (Views/Personagens/ListagemView.cs) o método abaixo, que se trata de ação que será executada sempre que a view voltar ao primeiro plano.

```
protected override void OnAppearing()
{
    base.OnAppearing();
    _ = viewModel.ObterPersonagens();
}
```



- Experimente colocar o trecho sinalizado na view de cadastro. Ela fará com que o cadastro apareça como um pop up.

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="AppRpgEtec.Views.Personagens.CadastroPersonagemView"
    Shell.PresentationMode="Animated"
    Title="CadastroPersonagemView">
    <VerticalStackLayout>
```

- Insira a tag a seguir abaixo do botão de salvar para que o usuário possa cancelar a ação

```
<Button Text="Cancel" Command="{Binding CancelarCommand}"></Button>
```

- Crie um ICommand chamado CancelarCommand (1), o método para retornar para a view de listagem (2) e a vinculação do command com o método (3)

```
private PersonagemService pService;
1 reference
public ICommand SalvarCommand { get; }
1 reference
1 public ICommand CancelarCommand { get; set; }

1 reference
public CadastroPersonagemViewModel()
{
    string token = Preferences.Get("UsuarioToken", string.Empty);
    pService = new PersonagemService(token);
    _ = ObterClasses();

    SalvarCommand = new Command(async () => { await SalvarPersonagem(); });
    3 CancelarCommand = new Command(async => CancelarCadastro());
}

0 references
2 private async void CancelarCadastro()
{
    await Shell.Current.GoToAsync("..");
}
```