ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE II – 2023/1

Luiz Fernando Souza / Elaine Marion

Aula 15 - Consumindo APIs – Atualização e remoção de Personagens do jogo RPG

1. Abra a classe **CadastroPersonagemViewModel** e programe o método que carregará o personagem. Exigirá o using de System.Linq

```
public async void CarregarPersonagem()
{
    try
    {
        Personagem p = await pService.GetPersonagemAsync(int.Parse(personagemSelecionadoId));

        this.Nome = p.Nome;
        this.PontosVida = p.PontosVida;
        this.Defesa = p.Defesa;
        this.Forca = p.Forca;
        this.Inteligencia = p.Inteligencia;
        this.Id = p.Id;

        TipoClasseSelecionado = this.ListaTiposClasse
        .FirstOrDefault(tClasse => tClasse.Id == (int)p.Classe);
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage
            .DisplayAlert("Ops", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
```

2. Altere o ICommand **SalvarPersonagem** da classe **CadastroPersonagemViewModel** para caso a propriedade Id não seja 0 ano cadastro, realizar o *put* no serviço que consome a API.

```
if (model.Id == 0)
    await pService.PostPersonagemAsync(model);
else
    await pService.PutPersonagemAsync(model);

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

3. Adicione no topo da classe **CadastroPersonagemViewModel** uma diretiva que servirá como uma pesquisa para relacionar o id do personagem clicado na view de listagem para que esse dado seja recuperado na view de cadastro

```
[QueryProperty("PersonagemSelecionadoId", "pId")]
3 references
public class CadastroPersonagemViewModel : BaseViewModel
{
    private PersonagemService pService;
```

4. Ainda na classe **CadastroPersonsagemViewModel**, crie um atributo e propriedade chamado **PersonagemSelecionadoId**

```
private string personagemSelecionadoId;//CTRL + R,E
0 references
public string PersonagemSelecionadoId
{
    set
    {
        if (value != null)
        {
            personagemSelecionadoId = Uri.UnescapeDataString(value);
            CarregarPersonagem();
        }
    }
}
```

5. Adicione mais campos na View de Cadastro para exibir o Id do Personagem. Observe que esse campo não permite edição.

```
<Label Text="Id" FontSize="Medium" />
<Entry Text="{Binding Id}" IsEnabled="False" FontSize="Medium" />

<Label Text="Nome" FontSize="Medium" />
<Entry Text="{Binding Nome}" FontSize="Medium" />
```

6. Abra a viewModel de listagem de personagens e declare um atributo e uma propriedade para armazenar o personagem selecionado. Observe que quando acontecer mudança no valor da propriedade, chamaremos uma rota para que o aplicativo exiba outra view, passando um id do personagem como parâmetro.

```
private Personagem personagemSelecionado;//CTRL + R,E

0 references
public Personagem PersonagemSelecionado
{
    get { return personagemSelecionado; }
    set
    {
        if (value != null)
        {
            personagemSelecionado = value;

            Shell.Current
                .GoToAsync($"cadPersonagemView?pId={personagemSelecionado.Id}");
        }
    }
}
```

Referência sobre rotas: https://lalorosas.com/blog/shell-routing

7. Na *view* de listagem dos personagens, modifique o *listview* para guardar o personagem quando houver o toque na tela, selecionando os dados para a propriedade criada na etapa anterior.

```
<ListView x:Name="listView" HasUnevenRows="True" ItemsSource="{Binding Personagens}"
          SelectedItem="{Binding PersonagemSelecionado}" >
```

Removendo um personagem

8. Programe a *viewModel* de listagem de personagens o método de remoção conforme a seguir

```
public async Task RemoverPersonagem(Personagem p)
{
    try
    {
        if (await Application.Current.MainPage
                .DisplayAlert("Confirmação", $"Confirma a remoção de {p.Nome}?", "Sim", "Não"))
        {
            await pService.DeletePersonagemAsync(p.Id);

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

            _ = ObterPersonagens();
        }
    }
    catch (Exception ex)
    {

        await Application.Current.MainPage
            .DisplayAlert("Ops", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
```

9. Também na viewModel de listagem de personagens, declare um ICommand (1) e inicialize-o (2) no construtor da classe

```
public ListagemPersonagemViewModel()
{
    string token = Application.Current.Properties["UsuarioToken"].ToString();
    pService = new PersonagemService(token);
    Personagens = new ObservableCollection<Personagem>();
    _ = ObterPersonagens();

    NovoPersonagem = new Command(async () => { await ExibirCadastroPersonagem(); });
2   RemoverPersonagemCommand =
        new Command<Personagem>(async (Personagem p) => { await RemoverPersonagem(p); });
}
1 reference
public ICommand NovoPersonagem { get; } //using System.Windows.Input
1 reference
1 public ICommand RemoverPersonagemCommand { get; }
```

10. Atualize o *listView* presente na *view* de listagem de personagens para que se conecte ao command RemoverPersonagemCommand

```
<ListView.ItemTemplate>
    <DataTemplate>
        <ViewCell>
            <ViewCell.ContextActions>
                <MenuItem Text="Remover" IsDestructive="True"
                        Command="{Binding Path=BindingContext.RemoverPersonagemCommand, Source={x:Reference listView}}"
                        CommandParameter="{Binding .}"></MenuItem>
            </ViewCell.ContextActions>
```

- Execute o aplicativo para realizar os testes.

- Para deixar o listview expansivo, abra uma tag *ScrollView* antes e depois do *VerticalStackLayout* e feche a tag depois do fechamento do *VerticalStackLayout.*