ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2025/2

Luiz Fernando Souza / Eliane Marion

Display Action Sheets - Atualizando dados das disputas e dos personagens

Nesta aula, faremos a configuração para restaurar os pontos de vida dos personagens e os dados das disputas.

1. Adicione os métodos abaixo na classe **PersonagemService** que consumirão o serviço da API.

```
public async Task<int> PutRestaurarPontosAsync(Personagem p)
{
    string urlComplementar = "/RestaurarPontosVida";
    var result = await _request.PutAsync(apiUrlBase + urlComplementar, p, _token);
    return result;
}
1 reference
public async Task<int> PutZerarRankingAsync(Personagem p)
{
    string urlComplementar = "/ZerarRanking";
    var result = await _request.PutAsync(apiUrlBase + urlComplementar, p, _token);
    return result;
}
0 references
public async Task<int> PutZerarRankingRestaurarVidasGeralAsync()
{
    string urlComplementar = "/ZerarRankingRestaurarVidas";
    var result = await _request.PutAsync(apiUrlBase + urlComplementar, new Personagem(), _token);
    return result;
}
```

2. Abra a **ListagemPersonagemViewModel** e crie os métodos que acionarão a classe de serviço

```
public async Task ExecutarRestaurarPontosPersonagem(Personagem p)
{
    await pService.PutRestaurarPontosAsync(p);
}
1 reference
public async Task ExecutarZerarRankingPersonagem(Personagem p)
{
    await pService.PutZerarRankingAsync(p);
}
0 references
public async Task ExecutarZerarRankingRestaurarVidasGeral()
{
    await pService.PutZerarRankingRestaurarVidasGeralAsync();
}
```

3. Adicione o método que processará todas as operações possíveis para o personagem

```
public async void ProcessarOpcaoRespondidaAsync(Personagem personagem, string result)
        {
            if (result.Equals("Editar Personagem"))
            {
                await Shell.Current
                        .GoToAsync($"cadPersonagemView?pId={personagem.Id}");
            }
            else if (result.Equals("Remover Personagem"))
            {
              if (await Application.Current.MainPage.DisplayAlert("Confirmação",
               $"Deseja realmente remover o personagem {personagem.Nome.ToUpper()}?",
"Yes", "No"))
                {
                    await RemoverPersonagem(personagem);
                    await Application.Current.MainPage.DisplayAlert("Informação",
                        "Personagem removido com sucesso!", "Ok");

                    await ObterPersonagens();
                }
            }
            else if (result.Equals("Restaurar Pontos de Vida"))
            {
                if (await Application.Current.MainPage.DisplayAlert("Confirmação",
                    $"Restaurar os pontos de vida de {personagem.Nome.ToUpper()}?", "Yes", "No"))
                {
                    await ExecutarRestaurarPontosPersonagem(personagem);
                    await Application.Current.MainPage.DisplayAlert("Informação",
                        "Os pontos foram restaurados com sucesso.", "Ok");

                    await ObterPersonagens();
                }
            }
            else if (result.Equals("Zerar Ranking do Personagem"))
            {
                if (await Application.Current.MainPage.DisplayAlert("Confirmação",
                    $"Zerar o ranking de {personagem.Nome.ToUpper()}?", "Yes", "No"))
                {
                    await ExecutarZerarRankingPersonagem(personagem);
                    await Application.Current.MainPage.DisplayAlert("Informação",
                        "O ranking foi zerado com sucesso.", "Ok");

                    await ObterPersonagens();
                }
            }
      }
```

4. Adicione o método que vai exibir as opções ao usuário e passará ao método de processamento, a palavra selecionada no click da caixa de opções.

```
public async Task ExibirOpcoesAsync(Personagem personagem)
{
    try
    {
        personagemSelecionado = null;
        string result = string.Empty;

        result = await Application.Current.MainPage
            .DisplayActionSheet("Opções para o personagem " + personagem.Nome,
                "Cancelar",
                "Editar Personagem",
                "Restaurar Pontos de Vida",
                "Zerar Ranking do Personagem",
                "Remover Personagem");

        if (result != null)
            ProcessarOpcaoRespondidaAsync(personagem, result);
    }
    catch (Exception ex)
    {
        await Application.Current
            .MainPage.DisplayAlert("Ops...", ex.Message, "Ok");
    }
}
```

5. Altere a propriedade **PersonagemSelecionado** inserindo o trecho sinalizado em verde e removendo o trecho em vermelho. Neste caso estamos invocando a exibição da caixa de opções.

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

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

            _ = ExibirOpcoesAsync(personagemSelecionado);
        }
    }
}
```

6. Programe o método que será responsável por zerar o ranking geral

```
public async Task ZerarRankingResturarVidasGeral()
{
    try
    {
        if (await Application.Current.MainPage.DisplayAlert("Confirmação",
           $"Deseja realmente zerar todo o ranking?", "Yes", "No"))
        {
            await ExecutarZerarRankingRestaurarVidasGeral();

            await Application.Current.MainPage
                .DisplayAlert("Informação", "Ranking zerado com sucesso.", "Ok");

            await ObterPersonagens();
        }
    }
    catch (Exception ex)
    {

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

7. Declare o ICommand abaixo do fechamento do construtor, no grupo em que existe outros ICommands

```
public ICommand ZerarRankingRestaurarVidasGeralCommand { get; set; }
```

8. Atribua o método ao ICommand dentro do construtor

```
ZerarRankingRestaurarVidasGeralCommand =
    new Command(async () => { await ZerarRankingResturarVidasGeral(); });
```

9. Posicione o botão para zerar o Ranking geral no Design da View, antes do ListView.

```
<Button Text="Zerar Ranking Geral" Command="{Binding ZerarRankingRestaurarVidasGeralCommand}"
          FontAttributes="Bold" VerticalOptions="FillAndExpand"/>
```

- Execute o aplicativo e faça os testes das funcionalidades programadas.

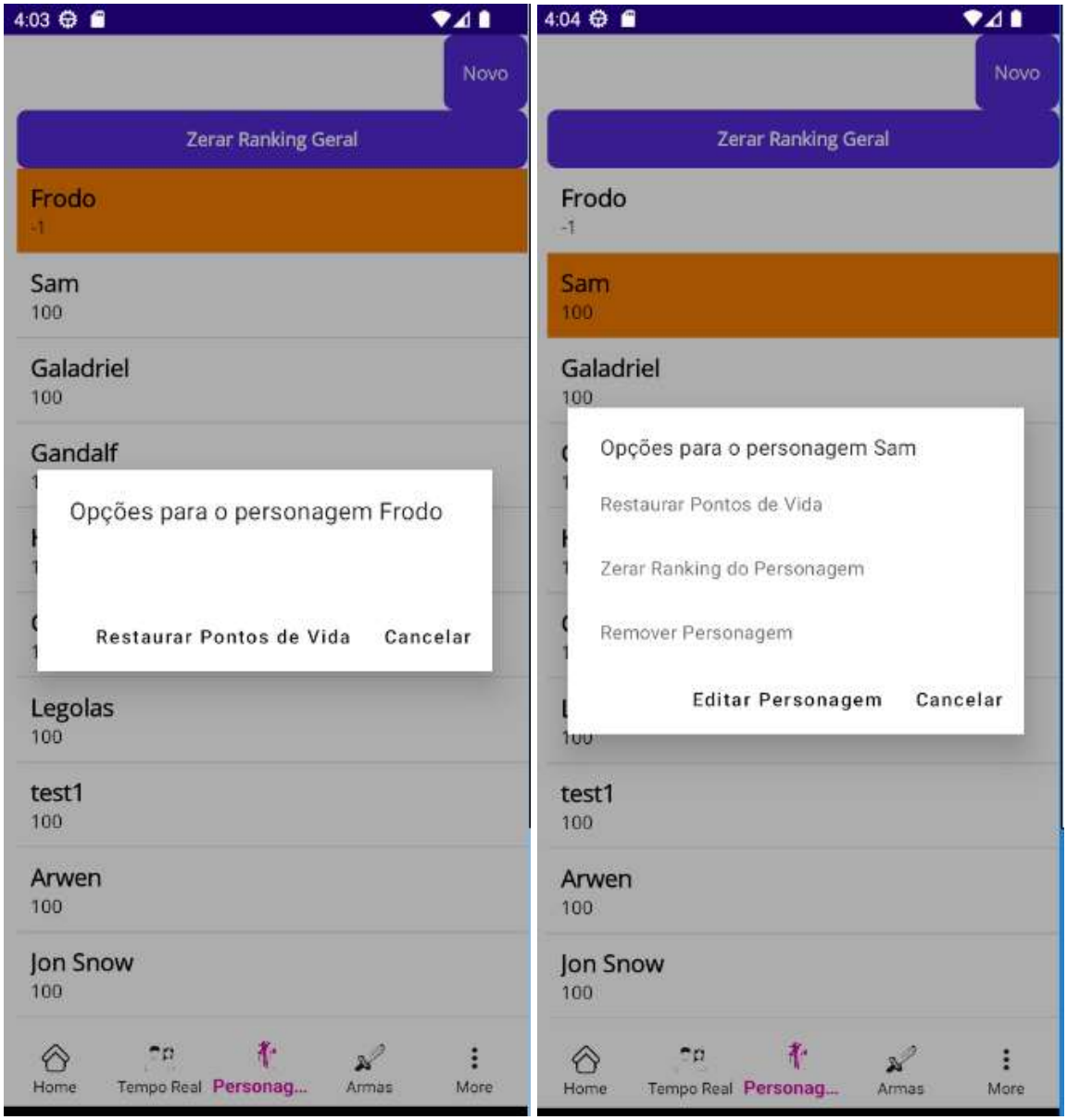
10. Adicione uma regra no método ExibirOpcoesAsync de View/Personagens/ListagemView.xaml.cs para caso os pontos de vida do Personagem estejam menores ou iguais a zero, o menu do personagem em questão seja exibido com menos opções.

```
public async Task ExibirOpcoesAsync(Personagem personagem)
{
    try
    {
        personagemSelecionado = null;
        string result = string.Empty;

        if (personagem.PontosVida > 0)
        {
            result = await Application.Current.MainPage
                .DisplayActionSheet("Opções para o personagem " + personagem.Nome,
                    "Cancelar",
                    "Editar Personagem",
                    "Restaurar Pontos de Vida",
                    "Zerar Ranking do Personagem",
                    "Remover Personagem");
        }
        else
        {
            result = await Application.Current.MainPage
                .DisplayActionSheet("Opções para o personagem " + personagem.Nome,
                    "Cancelar",
                    "Restaurar Pontos de Vida");
        }

        if (result != null)
            ProcessarOpcaoRespondidaAsync(personagem, result);
    }
    catch (Exception ex)
    {
        await Application.Current
            .MainPage.DisplayAlert("Ops...", ex.Message, "Ok");
    }
}
```

- Resultados esperados



ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2025/2

Luiz Fernando Souza / Eliane Marion

Validando a visualização de um botão – Conversores para cores

Na regra de negócios do app, não permitiremos que o botão de salvamento esteja ativo caso os pontos de vida sejam negativos. Definiremos uma propriedade para ser referenciada à view de vínculo através do binding

1. Crie uma propriedade chamada CadastroHabilitado na classe CadastroPersonagemViewModel

```
public bool CadastroHabilitado
{
    get
    {
        return (PontosVida > 0);
    }
}
```

- Perceba que esta propriedade só tem get, pois apenas retornará o true ou false não sendo possível atribuir valores, já que não temos o set.

2. Insira a notificação de mudança da propriedade CadastroHabilitado para os pontos de vida

```
public int PontosVida
{
    get => pontosVida;
    set
    {
        pontosVida = value;
        OnPropertyChanged();
        OnPropertyChanged(nameof(CadastroHabilitado));
    }
}
```

3. Abra a View de cadastro dos personagens e atribua IsEnabled ao binding do botão de Imagem.

```
<Button Text="Salvar" Command="{Binding SalvarCommand}"
        IsEnabled="{Binding CadastroHabilitado}"></Button>
```

- Realize o teste entrando no menu para cadastrar um novo personagem, como neste caso ele não terá Id ainda (será 0) o botão ficará invisível, pois a propriedade retornará false.

Luiz Fernando Souza / Eliane Marion

Validação de Campos

Podemos inserir validações baseadas nas propriedades de uma classe viewModel para habilitar ou desabilitar o botão de gravação.

4. Crie um método que verifica se o nome do personagem está preenchido e se os valores de força e defesa são diferentes de zero, além da propriedade que verifica os pontos de vida.

```
public bool ValidarCampos()
{
    return !string.IsNullOrEmpty(Nome)
        && CadastroHabilitado
        && Forca != 0
        && Defesa != 0;
}
```

5. Procure o construtor e altere o command do botão de salvar para examinar as condições, conforme abaixo

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

    SalvarCommand = new Command(async () => { await SalvarPersonagem(); }, () => ValidarCampos());
    CancelarCommand = new Command(async => CancelarCadastro());
}
```

- É necessário que na linha seguinte ao OnPropertyChanged de cada propriedade envolvida na validação, nós possamos inserir a codificação sinalizada abaixo, para que quando a viewModel identifique que a propriedade foi alterada, possa rodar a validação novamente. Repita o trecho sinalizado nas propriedades PontosVida, Forca e Defesa.

```
public string Nome
{
    get => nome;
    set
    {
        nome = value;
        OnPropertyChanged();
        ((Command)SalvarCommand).ChangeCanExecute();
    }
}
```

- Nesta validação está sendo verificado apenas o nome, quantidade de ponto de vida, força e defesa, ou seja, enquanto estes campos não estiverem preenchidos ou diferente de zero, o botão de salvar não será habilitado. Você pode incluir mais campos na validação observando o que foi feito nas etapas anteriores.

Destacando os personagens

6. Crie uma classe chamada PontosVidaConverter dentro da pasta Converters implementando a interface IValueConverter.

```
public class PontosVidaConverter : IValueConverter
```

16. Clique com CRTL + . (ponto) em IValueConverter e escolha implementar a interface.

7. Com a interface implementada, codifique o método *Convert* que definirá, de acordo com o valor dos pontos de vida, a cor que será retornada. Faça a notação “using Color = Microsoft.Maui.Graphics.Color” no topo do arquivo.

```
public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
{
    ColorTypeConverter converter = new ColorTypeConverter();//using Microsoft.Maui.Graphics.Converters;

    int pontosVida = (int)value;
    if (pontosVida == 100)
        return (Color)converter.ConvertFromInvariantString("SeaGreen");
    else if (pontosVida >= 75)
        return (Color)converter.ConvertFromInvariantString("YellowGreen");
    else if (pontosVida >= 25)
        return (Color)converter.ConvertFromInvariantString("Yellow");
    else if (pontosVida >= 1)
        return (Color)converter.ConvertFromInvariantString("OrangeRed");
    else
        return (Color)converter.ConvertFromInvariantString("Red");
}
```

8. Abra a view de listagem de personagens, faça referência a pasta dos conversores (A) e crie a área de recursos, fazendo a chamada a classe de Conversão de Pontos de Vida em cores (B).

A
```
xmlns:conv="clr-namespace:AppRpgEtec.Converters"
Title="ListagemView">
```

B
```
<ContentPage.Resources>
    <ResourceDictionary>
        <conv:PontosVidaConverter x:Key="ColorConvert" />
    </ResourceDictionary>
</ContentPage.Resources>
```

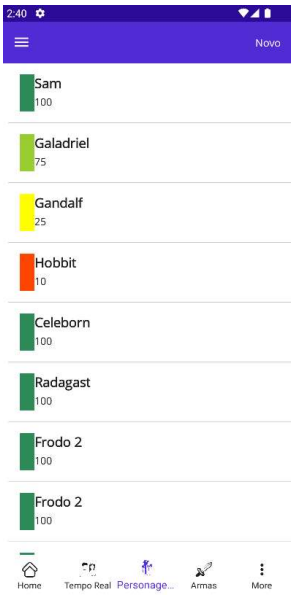
9. Ainda na Listagem de Personagens, procure as labels do nome e pontos de vida dos personagens, modificando o trecho para que tenhamos um Grid e um Box View para exibir a cor sinalizada pelos pontos de vida. Perceba que o Box View conterá a chamada para o Conversor

```
    </ViewCell.ContextActions>
    <Grid Padding="15">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="Auto"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>

        <BoxView Grid.RowSpan="2" HeightRequest="50" WidthRequest="20"
            Color="{Binding PontosVida, Converter={StaticResource ColorConvert}}"/>

        <Label Grid.Row="0" Grid.Column="1" Text="{Binding Nome}" FontSize="18" FontAttributes="Bold"/>
        <Label Grid.Row="1" Grid.Column="1"  Text="{Binding PontosVida}" FontSize="14"/>
    </Grid>
</ViewCell>
```

- Execute o aplicativo e confirme se os personagens disponíveis estarão de acordo com os pontos de vida dos personagens. Outra possibilidade seria usar essa mesma conversão para a propriedade TextColor da Label



Referências

ListView com Grid: https://learn.microsoft.com/pt-br/dotnet/maui/user-interface/controls/listview

BoxView: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/boxview