ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2024/1

Luiz Fernando Souza

ICommands e Métodos para Disputa dos Personagens

Ataque com armas

1. Crie uma pasta chamada Disputas dentro da pasta Services e dentro da pasta Disputas crie uma classe chamada DisputaService, com a programação abaixo

```
public class DisputaService :  Request
{
    private readonly Request _request;
    private string _token;

    //xyz --> site da sua API
    private const string _apiUrlBase = "https://xyz/Disputas";

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

}
```

2. Crie os métodos referentes aos tipos de disputas abaixo do construtor. Será necessários os usings para AppRpgEtec.Models e System.Threading.Tasks.

```
public async Task<Disputa> PostDisputaComArmaAsync(Disputa d)
{
    string urlComplementar = "/Arma";
    return await _request.PostAsync(_apiUrlBase + urlComplementar, d, _token);
}
0 references
public async Task<Disputa> PostDisputaComHabilidadesAsync(Disputa d)
{
    string urlComplementar = "/Habilidade";
    return await _request.PostAsync(_apiUrlBase + urlComplementar, d, _token);
}
0 references
public async Task<Disputa> PostDisputaGeralAsync(Disputa d)
{
    string urlComplementar = "/DisputaEmGrupo";
    return await _request.PostAsync(_apiUrlBase + urlComplementar, d, _token);
}
```

3. Abra a classe DisputaViewModel, declare um objeto do tipo DisputaService, e crie uma propriedade do tipo Disputa. inicializando os objetos no construtor. DisputaService exigirá o using de AppRpgEtec.Services.Disputas

```
public Personagem Oponente { get; set; }
private DisputaService dService;
1 reference
public Disputa DisputaPersonagens { get; set; }
1 reference
public DisputaViewModel()
{
    string token = Application.Current.Properties["UsuarioToken"].ToString();
    pService = new PersonagemService(token);
    dService = new DisputaService(token);

    Atacante = new Personagem();
    Oponente = new Personagem();
    DisputaPersonagens = new Disputa();
}
```

4. Crie o método que vai executar a disputa acionando a classe de serviços. Task exigirá o using System.Threading.Tasks.

```
private async Task ExecutarDisputaArmada()
{
    try
    {
        DisputaPersonagens.AtacanteId = Atacante.Id;
        DisputaPersonagens.OponenteId = Oponente.Id;
        DisputaPersonagens = await dService.PostDisputaComArmaAsync(DisputaPersonagens);

        await Application.Current.MainPage
            .DisplayAlert("Resultado", DisputaPersonagens.Narracao, "Ok");
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage
            .DisplayAlert("Ops", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
```

5. Declare o ICommand conforme (1). Tente agrupar este item abaixo do ICommand de pesquisa. E dentro do construtor, inicialize o ICommand vinculando ao método que realizará a disputa e apresentará a narração do embate (2).

```
    PesquisarPersonagensCommand =
        new Command<string>(async (string pesquisa) => { await PesquisarPersonagens(pesquisa); });

2   DisputaComArmaCommand =
        new Command(async () => { await ExecutarDisputaArmada(); });
}
1 reference
public ICommand PesquisarPersonagensCommand { get; set; }
1 reference
1 public ICommand DisputaComArmaCommand { get; set; }
```

6. Abra a parte de design da View, adicione mais uma `<RowDefinition Height="Auto"/>` na parte de configuração do Grid e insira o botão abaixo antes do fechamento do Grid

```
<Button Text="Disputar com Arma" FontAttributes="Bold"
                Grid.Row="2" Grid.Column="0" Grid.ColumnSpan="2"
               VerticalOptions="FillAndExpand" Command="{Binding DisputaComArmaCommand}"/>
```

- Execute o aplicativo. É necessário que existam Armas cadastradas para os Personagens envolvidos na disputa, principalmente o atacante.

- Opcional: Caso o personagem seja derrotado. Altere a coluna de pontos de vida para que ele volte a ter 100 pontos.

Ataque com Habilidades

7. Crie a classe **Habilidade** dentro da pasta Models, com as propriedades conforme abaixo. Use o atalho (prop + TAB + TAB) para agilizar a criação das propriedades.

```
public class Habilidade
{
    0 references
    public int Id { get; set; }
    0 references
    public string Nome { get; set; }
    0 references
    public int Dano { get; set; }
}
```

8. Crie uma classe publica **PersonagemHabilidade** dentro da pasta Models, com as propriedades a seguir

```
public int PersonagemId { get; set; }
0 references
public Personagem Personagem { get; set; }
1 reference
public int HabilidadeId { get; set; }
1 reference
public Habilidade Habilidade { get; set; }
0 references
public string HabilidadeNome
{
    get { return Habilidade.Nome; }
}
```

9. Crie uma pasta chamada **PersonagemHabilidades** dentro da pasta Services e dentro desta pasta crie a classe **PersonagemHabilidadeService**, que codificará os métodos que vão consumir a API.

```
public class PersonagemHabilidadeService : Request
    {
        private readonly Request _request = null;

        //Substitua xyz abaixo pelo endereço da sua API:
        private const string _apiUrlBase = "https://xyz/PersonagemHabilidades/";
        private string _token;

        public PersonagemHabilidadeService(string token)
        {
            _request = new Request();
            _token = token;
        }
        //Próximos métodos aqui
         }
```

10. Programaremos um método para buscar a lista de habilidades de acordo com o id do Personagem e outro para buscar a lista de habilidades. Será necessário os usings AppRpgEtec.Models, System.Collections.ObjectModel e System.Threading.Tasks.

```
public async Task<ObservableCollection<PersonagemHabilidade>> GetPersonagemHabilidadesAsync(int personagemId)
        {
            string urlComplementar = string.Format("{0}", personagemId);

            ObservableCollection<Models.PersonagemHabilidade> listaPH = await
                _request.GetAsync<ObservableCollection<Models.PersonagemHabilidade>>(_apiUrlBase + urlComplementar,
_token);
            return listaPH;
        }
        public async Task<ObservableCollection<Habilidade>> GetHabilidadesAsync()
        {
            string urlComplementar = string.Format("{0}", "GetHabilidades");

            ObservableCollection <Models.Habilidade> listaHabilidades = await
                _request.GetAsync<ObservableCollection<Models.Habilidade>>(_apiUrlBase + urlComplementar, _token);
            return listaHabilidades;
             }
```

11. Abra a classe DisputaViewModel, declare uma propriedade de lista de PersonagemHabilidade e para o serviço que consumirá a API. Inicialize-os no construtor. Usings de System.Collections.ObjectModel e AppRpgEtec.Services.PersonagemHabilidades

```
private PersonagemHabilidadeService phService;
0 references
public ObservableCollection<PersonagemHabilidade> Habilidades { get; set; }

1 reference
public DisputaViewModel()
{
    string token = Application.Current.Properties["UsuarioToken"].ToString();
    pService = new PersonagemService(token);
    dService = new DisputaService(token);
    phService = new PersonagemHabilidadeService(token);

    Atacante = new Personagem();
    Oponente = new Personagem();
    DisputaPersonagens = new Disputa();
}
```

12. Ainda em DisputaViewModel, crie um método que buscará a listagem de PersonagemHabilidades através do Id de um personagem

```
public async Task ObterHabilidadesAsync(int personagemId)
{
    try
    {

        Habilidades = await phService.GetPersonagemHabilidadesAsync(personagemId);
        OnPropertyChanged(nameof(Habilidades));

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

13. Altere o método que seleciona o personagem, conforme o trecho indicado, para que depois que o Atacante seja selecionado, possamos executar o método que busca as habilidades deste personagem

```
public async void SelecionarPersonagem(Personagem p)
{
    try
    {
        string tipoCombatente = await Application.Current.MainPage
            .DisplayActionSheet("Atacante ou Oponente?", "Cancelar", "", "Atacante", "Oponente");


        if (tipoCombatente == "Atacante")
        {
            await this.ObterHabilidadesAsync(p.Id);
            Atacante = p;
            OnPropertyChanged(nameof(DescricaoPersonagemAtacante));
```

14. Crie um atributo/propriedade para armazenar a Habilidade que será selecionada na view.

```
private PersonagemHabilidade habilidadeSelecionada;
2 references
public PersonagemHabilidade HabilidadeSelecionada
{
    get { return habilidadeSelecionada; }
    set
    {
        if (value != null)
        {
            try
            {
                habilidadeSelecionada = value;
                OnPropertyChanged();
            }
            catch (Exception ex)
            {
                Application.Current.MainPage.DisplayAlert("Ops", ex.Message, "Ok");
            }
        }
    }
}
```

15. Crie um método para executar a disputa com Habilidades

```
private async Task ExecutarDisputaHabilidades()
{
    try
    {
        DisputaPersonagens.AtacanteId = Atacante.Id;
        DisputaPersonagens.OponenteId = Oponente.Id;
        DisputaPersonagens.HabilidadeId = habilidadeSelecionada.HabilidadeId;
        DisputaPersonagens = await dService.PostDisputaComHabilidadesAsync(DisputaPersonagens);

        await Application.Current.MainPage
           .DisplayAlert("Resultado", DisputaPersonagens.Narracao, "Ok");
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage
            .DisplayAlert("Ops", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
```

16. Declare um ICommand (1) e vincule ao método que faz a chamada para a disputa utilizando habilidades

```
    DisputaComArmaCommand =
        new Command(async () => { await ExecutarDisputaArmada(); });

2   DisputaComHabilidadeCommand =
        new Command(async () => { await ExecutarDisputaHabilidades(); });
}
1 reference
public ICommand PesquisarPersonagensCommand { get; set; }
1 reference
public ICommand DisputaComArmaCommand { get; set; }
1 reference
1 public ICommand DisputaComHabilidadeCommand { get; set; }
```

17. Abra a view de Disputas/ListagemView e adicione duas novas linhas para o grid

```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
```

18. Insira antes do fechamento do Grid as tags para o Picker e o botão para que possamos selecionar a Habilidade do Personagem e executar a disputa

```
<Picker Title="---Selecione a habilidade do primeiro combatente---"
    Grid.Row="3" Grid.Column="0" Grid.ColumnSpan="2"
    ItemsSource="{Binding Habilidades}"
    ItemDisplayBinding="{Binding HabilidadeNome}"
    SelectedItem="{Binding HabilidadeSelecionada}" />

 <Button Text="Disputas com Habilidades" FontAttributes="Bold"
     Grid.Row="4" Grid.Column="0" Grid.ColumnSpan="2"
    VerticalOptions="FillAndExpand" Command="{Binding DisputaComHabilidadeCommand}"/>
```

- Execute o aplicativo para realizar a disputa com Habilidades entre os Personagens

Disputa geral entre os Personagens

19. Crie o método que acionará a classe de Serviço.

```
private async Task ExecutarDisputaGeral()
{
    try
    {
        ObservableCollection<Personagem> lista = await pService.GetPersonagensAsync();
        DisputaPersonagens.ListaIdPersonagens = lista.Select(x => x.Id).ToList();

        DisputaPersonagens = await dService.PostDisputaGeralAsync(DisputaPersonagens);

        string resultados = string.Join(" | ", DisputaPersonagens.Resultados);

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

20. Declare o ICommand **DisputaGeral** conforme abaixo.

```
        2 DisputaGeralCommand =
              new Command(async () => { await ExecutarDisputaGeral(); });
    }
    1 reference
    public ICommand PesquisarPersonagensCommand { get; set; }
    1 reference
    public ICommand DisputaComArmaCommand { get; set; }
    1 reference
    public ICommand DisputaComHabilidadeCommand { get; set; }
    1 reference
1   public ICommand DisputaGeralCommand { get; set; }
```

21. Abra a View Disputas/ListagemView.xaml e adicione mais uma linha ao Grid

```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
```

22. Adicione o Botão que fará a chamada ao método da ViwModel

```
<Button Text="Disputa Geral" FontAttributes="Bold"
                        Grid.Row="5" Grid.Column="0" Grid.ColumnSpan="2"
      VerticalOptions="FillAndExpand" Command="{Binding DisputaGeralCommand}"/>
```

- Execute o aplicativo e realize o teste clicando no botão de disputa geral.