



Visual Studio para desenvolvimento Mobile

O desenvolvimento mobile para este semestre será realizado através do Visual Studio, com o .Net Maui (.NET Multi-Plataform App UI) evolução do Xamarin Forms, plataforma incorporada ao Visual Studio e que permite o desenvolvimento multiplataforma, com o ponto positivo da criação de uma única Solution podendo conter vários projetos de classes que podem ser executados no Android, IOS e Windows, utilizando-se de uma mesma estrutura, além de customizações de interface que cada sistema operacional oferece.

O que era o Xamarin.Forms

Xamarin.Forms é um kit de ferramentas de interface do usuário multiplataforma que permite que os desenvolvedores criem, com eficiência, layouts de interface de usuário nativos que possam ser compartilhados entre iOS, Android e aplicativos da Plataforma Universal do Windows.

Estrutura do Front-end .NET MAUI

Feita em XAML (Linguagem de marcação de aplicativo extensivo) – Linguagem declarativa baseada em XML desenvolvida pela Microsoft para criação de Layouts. É facilmente integrável com arquiteturas de aplicativo populares como o MVVM.

Controles XAML: BoxView, ListView, Button, Label, Maps, Image Button Search Bar, Progress Bar, Slidesr, etc.

Instalação do Visual Studio

1. Acessar a página de Download do Visual Studio no link:

<https://visualstudio.microsoft.com/pt-br/downloads>

2. Desça um pouco a página até a parte de Downloads e faça a escolha para baixar a versão Comunidade (Community). Esta versão permite logar com o e-mail institucional. Se você já tem o Visual Studio 2019 instalado, procure no Windows por “Visual Studio Installer”, execute o programa e verifique se as configurações descritas na etapa 3 estão selecionadas, se alguma não estiver, basta selecionar e executar.

Downloads

The screenshot shows the Visual Studio 2022 download page. On the left, there's a summary for "Visual Studio 2022 | Windows". Below it, a paragraph describes the IDE as the best for .NET and C++ development on Windows. In the center, three editions are listed: "Comunidade" (Community), "Professional", and "Enterprise". The "Comunidade" box is highlighted with a green border. It contains a "Download gratuito" button. The "Professional" and "Enterprise" boxes also have "Avaliação gratuita" buttons. At the bottom, there are links for "Notas de versão", "Comparar Edições", and "Como fazer a instalação offline".

- O download se iniciará automaticamente. Verifique se o seu navegador não está bloqueando pop-ups.
- Esse arquivo se trata de um instalador online que gerencia o que será instalado ou atualizando e é onde escolhemos algumas configurações que falaremos adiante.



3. Ao Executar o arquivo baixado, a tela a seguir aparecerá e selecionaremos os seguintes itens no menu “Cargas de Trabalho”.

Web e Nuvem (4)

ASP.NET e desenvolvimento Web Crie aplicativos Web usando ASP.NET Core, ASP.NET, HTML/JavaScript e Contêineres, incluindo suporte a Doc...	Desenvolvimento para Azure SDKs, ferramentas e projetos do Azure para desenvolver aplicativos na nuvem e criar recursos usando o .NET e o
Desenvolvimento Python Edição, depuração, desenvolvimento interativo e controle do código-fonte para Python.	Desenvolvimento em node.js Crie aplicativos de rede escalonáveis usando Node.js, um tempo de execução do JavaScript conduzido por evento...

Área de trabalho e Dispositivos móveis (5)

Desenvolvimento de .NET Multi-Platform App UI Crie aplicativos Android, iOS, Windows e Mac a partir de uma única base de código usando C# com .NET MAUI.	Desenvolvimento para desktop com .NET Crie WPF, Windows Forms e aplicativos de console usando C#, Visual Basic e F# com .NET e .NET Framework.
Desenvolvimento para desktop com C++ Crie aplicativos C++ modernos para Windows usando as ferramentas de sua escolha, incluindo MSVC, Clang, CMA...	Desenvolvimento com a Plataforma Universal do Wind... Crie aplicativos para a Plataforma Universal do Windows usando C#, VB, JavaScript ou, como alternativa, C++.

4. Selecione a aba “pacote de idiomas”, mantenha apenas a opção “Inglês” selecionada. Para prosseguir a instalação selecione a opção “Baixar tudo e instalar” e mantenha seu computador conectado a internet.

Instalando — Visual Studio Community 2019 — 16.10.4

Cargas de trabalho Componentes individuais **Pacotes de idiomas** Locais de instalação

Você pode adicionar pacotes de idiomas adicionais à instalação do Visual Studio.

<input type="checkbox"/> Alemão	<input type="checkbox"/> Ferramentas de desenvolvimento .NET
<input type="checkbox"/> Chinês (Simplificado)	<input type="checkbox"/> Ferramentas de desenvolvimento do .NET...
<input type="checkbox"/> Chinês (Tradicional)	<input type="checkbox"/> ASP.NET e ferramentas de desenvolvim...
<input type="checkbox"/> Coreano	<input type="checkbox"/> ASP.NET e ferramentas de desenvolvim...
<input type="checkbox"/> Espanhol	<input type="checkbox"/> Opcional
<input type="checkbox"/> Francês	<input checked="" type="checkbox"/> Ferramentas de desenvolvimento do .NET...
<input checked="" type="checkbox"/> Inglês	<input checked="" type="checkbox"/> Ferramentas de nuvem para desenvolvime...
<input type="checkbox"/> Italiano	<input checked="" type="checkbox"/> Ferramentas de criação de perfil do .NET
<input type="checkbox"/> Japonês	<input checked="" type="checkbox"/> Ferramentas do Entity Framework 6
<input type="checkbox"/> Polonês	<input checked="" type="checkbox"/> Recursos avançados do ASP.NET
<input checked="" type="checkbox"/> Português (Brasil)	<input checked="" type="checkbox"/> Developer Analytics Tools
<input type="checkbox"/> Russo	<input checked="" type="checkbox"/> Implantação da Web
<input type="checkbox"/> Tcheco	<input checked="" type="checkbox"/> Live Share
<input type="checkbox"/> Turco	<input checked="" type="checkbox"/> Como depurar o .NET com o WSL 2
	<input checked="" type="checkbox"/> IntelliCode
	<input type="checkbox"/> Windows Communication Foundation
	<input type="checkbox"/> Ferramentas de desenvolvimento do .NET...
	<input type="checkbox"/> Ferramentas de desenvolvimento do .NET...

Detalhes da instalação

ASP.NET e desenvolvimento Web

Incluído

- Ferramentas de desenvolvimento .NET
- Ferramentas de desenvolvimento do .NET...
- ASP.NET e ferramentas de desenvolvim...
- ASP.NET e ferramentas de desenvolvim...

Opcional

- Ferramentas de desenvolvimento do .NET...
- Ferramentas de nuvem para desenvolvime...
- Ferramentas de criação de perfil do .NET
- Ferramentas do Entity Framework 6
- Recursos avançados do ASP.NET
- Developer Analytics Tools
- Implantação da Web
- Live Share
- Como depurar o .NET com o WSL 2
- IntelliCode
- Windows Communication Foundation
- Ferramentas de desenvolvimento do .NET...
- Ferramentas de desenvolvimento do .NET...

Localização
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community Alterar...

Ao continuar, você concorda com a licença [licença](#) da edição do Visual Studio selecionada. Também oferecemos a possibilidade de baixar outro software com o Visual Studio. Esse software é licenciado separadamente, conforme [3rd Party Notices](#) ou na licença que o acompanha. Ao continuar, você também concorda com essas licenças.

Espaço total necessário 9.9 GB

Baixar tudo, depois instalar Instalar



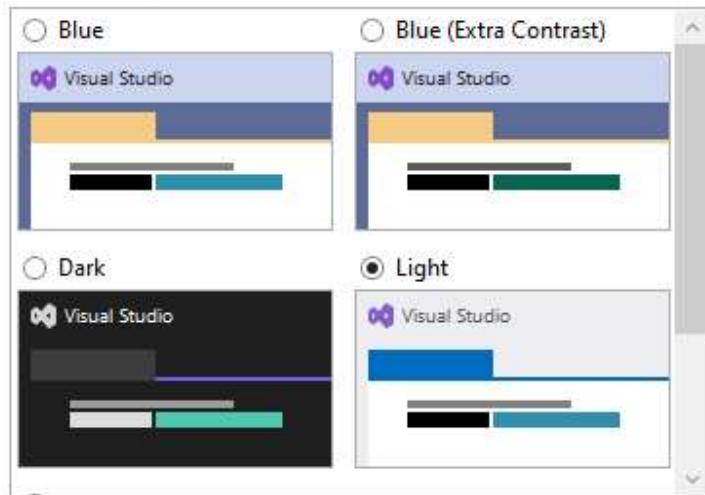
-
5. Quando executar o Visual Studio pela primeira vez, aparecerá uma caixa de seleção para escolher a linguagem de programação padrão. Você deve escolher o C#.

Visual Studio

Start with a familiar environment

Development Settings:

Choose your color theme



You can always change these settings later.

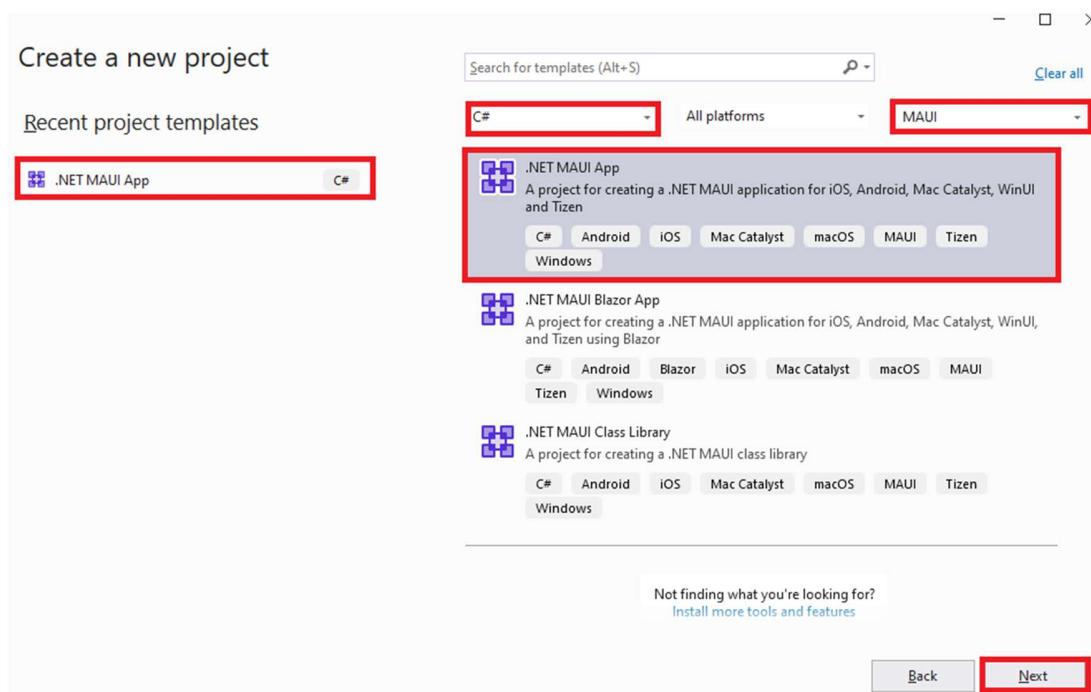


Atividade Prática

- Configuração para execução no próprio aparelho ou no emulador
- Criação do primeiro projeto .NET MAUI
- Entendimento dos elementos XAML para a interface do Aplicativo.
- Aplicativo básico para sala de Aula:

Primeiro Aplicativo:

Programação de um Aplicativo que exiba uma Label e Botão e exibindo o número de vezes que o usuário clicou no botão e demais acréscimos.



Configure your new project

.NET MAUI App C# Android iOS Mac Catalyst macOS MAUI Tizen Windows

Project name

Location
 ...

Solution name ?

Place solution and project in the same directory

Back Next



Escolher a versão do .NET e clique em criar.

Additional information

.NET MAUI App C# Android iOS Mac Catalyst macOS MAUI Tizen Windows

Framework ⓘ

.NET 8.0 (Long Term Support)

Back

Create

- Para inicialização do emulador, navegue até o menu Tools → Android → Android Device Manager, clicando no botão Iniciar no dispositivo existente.
 - ATENÇÃO: O emulador será iniciado uma vez durante a aula para que possamos executar o projeto diversas vezes. Não feche o emulador.
 - Para executar o projeto selecione a triângulo ao lado botão Play e selecione o emulador iniciado. Agora sim, você poderá clicar no botão play e conferir a execução do aplicativo.
- Abra a ToolBox ao lado esquerdo, e adicione mais controles na View chamada MainPage.Xaml. Ao digitar a propriedade Clicked, dentro das aspas duplas, selecione a opção sinalizada.

```
<Entry x:Name="txtNome" Placeholder="Digite seu nome" />
<Button x:Name="btnVerificar" Text="Verificar" Clicked="" />
```

Bind event to a newly created method called 'btnVerificar_Clicked'. Use 'Go To Definition' to navigate to the newly created method.

- Toda view (arquivo .xaml) possui um arquivo de código C#, com extensão .xaml.cs. Abra (F7) o arquivo de código MainPage.Xaml.cs e perceba que o evento clicando na opção anterior na View, foi criado na parte de código. Realize a programação dentro do evento conforme a seguir

```
private void btnVerificar_Clicked(object sender, EventArgs e)
{
    string texto = $"O nome tem {txtNome.Text.Length} caracteres";

    DisplayAlert("Mensagem", texto, "Ok");
}
```

- Arraste mais um botão e crie o evento Clicked

```
<Button x:Name="btnLimpar" Text="Limpar" />
```



4. Realize a programação do evento, na parte de código da View

```
private async void btnLimpar_Clicked(object sender, EventArgs e)
{
    if(await DisplayAlert("Pergunta", "Deseja realmente limpar a tela", "Yes", "No"))
    {
        txtNome.Text = string.Empty;
    }
}
```

5. Insira os controles abaixo para verificar os dias vividos.

```
<Label Text="Selecione sua data de nascimento" />
<DatePicker x:Name="txtDtNascimento" />
<Button x:Name="btnVerificarDiasVividos"
        Text="Verificar Dias Vividos" Clicked="btnVerificarData_Clicked"/>
```

6. Crie o evento para o botão e realize a programação para exibição dos dias vividos

```
private async void btnVerificarData_Clicked(object sender, EventArgs e)
{
    int diasVividos = DateTime.Now.Subtract(txtDtNascimento.Date).Days;

    await Application.Current.MainPage
        .DisplayAlert("Mensagem", $"Você já viveu {diasVividos} dias", "Ok");
}
```

7. Crie mais um botão com o nome a seguir, criando um evento para o mesmo

```
<Button x:Name="btnCalcular" Text="Calcular" />
```

8. Programa a primeira parte do evento coletando os números digitados pelo usuário

```
private async void btnCalcular_Clicked(object sender, EventArgs e)
{
    string n1 = await Application.Current.MainPage
        .DisplayPromptAsync("Mensagem", "Digite o primeiro número", "Ok");

    string n2 = await Application.Current.MainPage
        .DisplayPromptAsync("Mensagem", "Digite o segundo número", "Ok");

    string operacao = await Application.Current.MainPage
        .DisplayActionSheet("Mensagem", "Selecione uma opção",
        "Cancelar", "Somar", "Subtrair", "Multiplicar", "Dividir");
```



-
9. Realize a segunda parte do evento, coletando qual a operação escolhida pelo usuário e realizando o cálculo.

```
string operacao = await Application.Current.MainPage
    .DisplayActionSheet("Mensagem", "Selecione uma opção",
    "Cancelar", "Somar", "Subtrair", "Multiplicar", "Dividir");

if(operacao != null)
{
    if (operacao == "Somar")
    {
        int resultado = int.Parse(n1) + int.Parse(n2);

        await Application.Current.MainPage
            .DisplayAlert("Mensagem", $"O resultado é {resultado}", "Ok");
    }
    else if(operacao == "Subtrair")
    {
        int resultado = int.Parse(n1) - int.Parse(n2);

        await Application.Current.MainPage
            .DisplayAlert("Mensagem", $"O resultado é {resultado}", "Ok");
    }
    //Realize as demais operações.
}
```



Atividade:

Entregar quatro prints (não precisa zipar) nesta tarefa:

- Print da View (ContentPage)
- Print da parte de código da View
- Print do dispositivo funcionando com a soma dos dias
- Print do dispositivo funcionando com a soma dos meses

Use o aplicativo "PrimeiroApp" para incrementar as seguintes funcionalidades:

- Caixa de texto para armazenar uma data no formato "dd/MM/aaaa"
- Botão com o texto "Adicionar Dez dias"
- Botão com o texto "Adicionar Dez meses"
- Deve ser criado um evento no botão de texto "Adicionar Dez dias" para programação na parte de código
- A programação na parte de código deve pegar a data digitada, converter para armazenar em uma variável DateTime, exibir ao usuário com dez dias adicionados.
- Deve ser criado um evento no botão de texto "Adicionar Dez meses"
- A programação na parte de código deve perguntar ao usuário se ele realmente quer somar dez meses, se sim, pegar a data digitada, converter para armazenar em uma variável DateTime e exibir ao usuário adicionado dez meses.

Deixo como referência as seguintes pesquisas para dar suporte ao trabalhar com datas para guardar em variáveis e somar:

<https://www.portugal-a-programar.pt/forums/topic/42216-como-somar-a-data>

<https://www.arquivodecodigos.com.br/dicas/2811-c-csharp-como-adicionar-ou-subtrair-dias-de-uma-data-usando-c.html>



Aula 03 - Ciclo de Vida de um App - Preferences - MVVM (Model View ViewModel) - Binding e Command

- Criar Solution .NET MAUI App no Framework 6.0 chamada **AppBindingCommands**

A classe Base Application apresentada na classe App.Xaml.cs possui três métodos virtuais que podem ser sobreescritos para ditar o ciclo de vida do app.

- OnStart: Chamado quando o aplicativo é iniciado.
- OnSleep: Chamado sempre que o aplicativo vai para segundo plano.
- OnResume: Chamado quando o aplicativo é retornado, após ter sido enviado para segundo plano.

Observações:

- Propriedade *MainPage* na classe *App.xaml.cs* define qual a página inicial do Aplicativo

No construtor da classe App, codifique o recurso de Preferences:

```
public App()
{
    InitializeComponent();

    DateTime data = DateTime.Now;
    Preferences.Set("dtAtual", data);
    Preferences.Set("AcaoInicial", string.Format("* App executado às {0}. \n", data));

    MainPage = new AppShell();
}
```

Sobrecreva os eventos do cílico de vida do App conforme abaixo:

```
protected override void OnStart()
{
    base.OnStart();
    Preferences.Set("AcaoStart", string.Format("* App iniciado às {0}. \n", DateTime.Now));
}

0 references
protected override void OnSleep()
{
    base.OnSleep();
    Preferences.Set("AcaoSleep", string.Format("* App em segundo plano às {0}. \n", DateTime.Now));
}

0 references
protected override void OnResume()
{
    base.OnResume();
    Preferences.Set("AcaoResume", string.Format("* App reativado às {0}. \n", DateTime.Now));
}
```



Abra a view MainPage.xaml e arraste uma Label um Button, configurando conforme abaixo

```
<Label x:Name="lblInformacoes" Text="" />
<Button x:Name="btnAtualizarInformacoes" Text="Atualizar Informações" />
```

- Crie o evento Clicked conforme abaixo. Ao digitar aparecerá a opção “New Event Handler” que já criará o evento na parte de código da view

```
<Label x:Name="lblInformacoes" Text="" />
<Button x:Name="btnAtualizarInformacoes" Text="Atualizar Informações" Clicked="" />
</VerticalStackLayout>
```

The screenshot shows the XAML code for a vertical stack layout containing a label and a button. The button's Clicked attribute is highlighted with a red rectangle. A context menu is open over the Clicked attribute, with the "New Event Handler" option also highlighted with a red rectangle. Other options in the menu include "OnCounterClicked".

Localize o evento na parte de Código desta Content Page e realize a programação para recuperar o valor das *preferences*

```
private void btnAtualizarInformacoes_Clicked(object sender, EventArgs e)
{
    string informacoes = string.Empty;

    if (preferences.ContainsKey("AcaoInicial"))
        informacoes += Preferences.Get("AcaoInicial", string.Empty); //string.empty --> valor default.

    if (preferences.ContainsKey("AcaoStart"))
        informacoes += Preferences.Get("AcaoStart", string.Empty);

    if (preferences.ContainsKey("AcaoSleep"))
        informacoes += Preferences.Get("AcaoSleep", string.Empty);

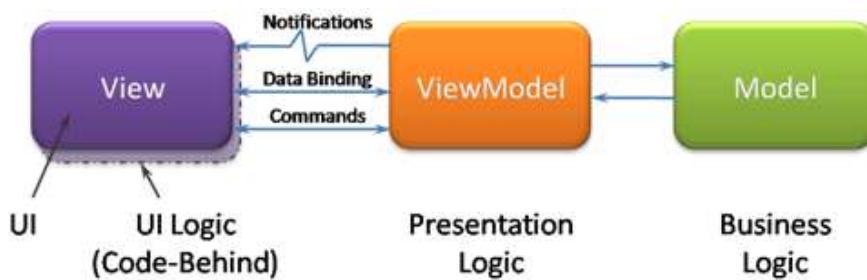
    if (preferences.ContainsKey("AcaoResume"))
        informacoes += Preferences.Get("AcaoResume", string.Empty);

    lblInformacoes.Text = informacoes;
}
```

Insira breakpoints nos construtores das classes mencionadas e nos Métodos OnStart, OnSleep e OnResume, executar o aplicativo e depurar (debugar) alternando para a área de trabalho do emulador/celular e voltando o aplicativo para o primeiro plano. Lembrete sobre depuração: F9 → Insere e remove breakpoints; F10 → passa linha a linha do código sem adentrar métodos; F11 → Entra dentro de métodos quando o breakpoint está na linha de um método; F5 → Pula de um breakpoint para o outro.



Padrão MVVM (Model View ViewModel)



Fonte da Imagem: <https://www.devmedia.com.br/entendendo-o-pattern-model-view-viewmodel-mvvm/18411>

O padrão de desenvolvimento MVVM é um pattern de desenvolvimento que oferece uma maior divisão de responsabilidades às camadas envolvidas.

Data Binding

Data Binding é uma forma de carregar elementos de uma View com dados fornecidos por uma classe, com isso não precisaremos dar nome aos componentes de uma View mas sim dar o nome da vinculação a qual o componente ficará atrelado.

Crie uma pasta chamada **ViewModels** e dentro desta pasta, crie uma classe chamada **UsuarioViewModel.cs** que deverá implementar uma interface chamada **INotifyPropertyChanged** com o evento que a classe oferece. A programação, por enquanto, deverá ficar conforme abaixo:

- (CTRL + .) → Using System.ComponentModel (caso não apareça automático) e Implement Interface

```
0 references
public class UsuarioViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    0 references
    void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```



Crie um atributo chamado name inicializando-o vazio.

```
private string name = string.Empty; //CTRL + R, E
```

Crie a propriedade correspondente (CTRL + R, E). Após a criação da propriedade, modifique o set conforme o sinalizado. Veja que estaremos usando o método OnPropertyChanged, responsável por refletir as mudanças desse dado da Classe na View

```
public string Name {  
    get => name;  
    set  
    {  
        if (name == null)  
            return;  
  
        name = value;  
        OnPropertyChanged(nameof(name));  
    }  
}
```

Crie uma propriedade chamada DisplayName inicializando conforme abaixo:

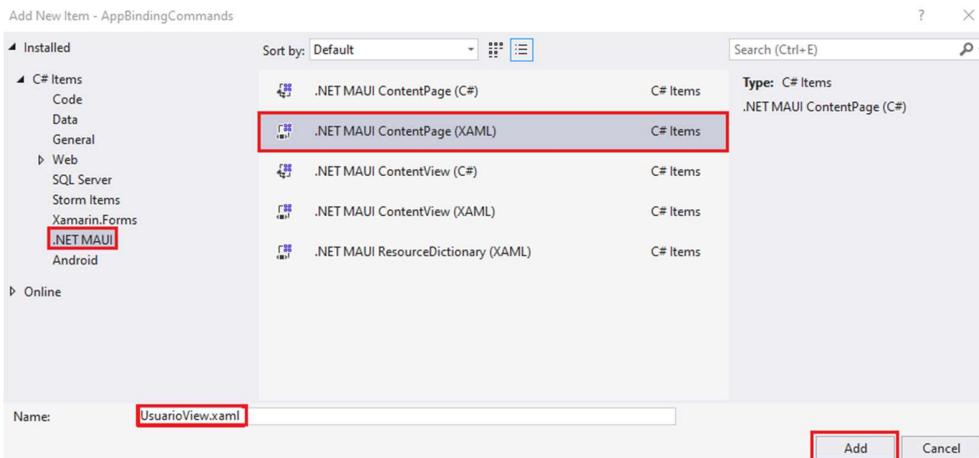
```
public string DisplayName => $"Nome digitado : {Name}";
```

No Set da propriedade Name, inclua o OnPropertyChanged referente ao DisplayName

```
public string Name {  
    get => name;  
    set  
    {  
        if (name == null)  
            return;  
  
        name = value;  
        OnPropertyChanged(nameof(Name));  
        OnPropertyChanged(nameof(DisplayName));  
    }  
}
```



Crie uma pasta chamada **Views** e dentro desta pasta crie uma Content Page chamada **UsuarioView**



Vincule a View (ContentPage) à classe criada na pasta ViewModels através construtor da parte de código da View, conforme abaixo. Será necessário um using para a pasta *AppBindingCommands.ViewModels*.

```
public UsuarioView()
{
    InitializeComponent();
    BindingContext = new UsuarioViewModel();
}
```

Volte até à View *UsuarioView.Xaml* e insira os controles a seguir e inclua na propriedade Text, o nome das propriedades criadas na classe ViewModel, junto com a palavra chave Binding

```
<Entry Text="{Binding Name}" Placeholder="Digite seu nome" />
<Label Text="{Binding DisplayName}" />
```

Abra a view *AppShell.Xaml*, faça referência a pasta Views (A) para poder referenciar a view de usuário (B) e atribuir uma rota (C).

```
xmlns:local="clr-namespace:AppBindingCommands"          A
xmlns:views="clr-namespace:AppBindingCommands.Views"
Shell.FlyoutBehavior="Disabled"

<ShellContent
    Title="Home"                                     B
    ContentTemplate="{DataTemplate views:UsuarioView}"
    C Route="UsuarioView" />
</Shell>
```

- Execute o aplicativo para confirmar que a digitação irá refletir imediatamente na label.



Commands

Commands são formas de acionar dados em uma classe (ViewModel) através de ações em uma View. Uma forma de não criarmos eventos clique para os botões. Serão criados métodos acionados através destes Commands nas classes ViewModel.

Abra a classe UsuarioViewModel e crie um atributo chamado **displayMessage** e a propriedade (CTRL + R, E) conforme abaixo:

```
string displayMessage = string.Empty;  
2 references  
public string DisplayMessage { get => displayMessage; set => displayMessage = value; }
```

Faça as modificações no set conforme abaixo. Perceba que o método OnPropertyChanged está referenciando o atributo/propriedade criado anteriormente.

```
public string DisplayMessage  
{  
    get => displayMessage;  
    set  
    {  
        if (displayMessage == null)  
            return;  
  
        displayMessage = value;  
        OnPropertyChanged(nameof(DisplayMessage));  
    }  
}
```

Ainda na classe UsuarioViewModel crie uma propriedade do tipo **ICommand** chamada **ShowMessageCommand** apenas com get (Prop + TAB + TAB). Necessitará do “using System.Windows.Input”.

```
public ICommand ShowMessageCommand { get; }
```

E o método ShowMessage. Perceba que usamos a data atual guardada no início da execução do aplicativo através das preferences(Classe App).

```
public void ShowMessage()  
{  
    DateTime data = Preferences.Get("dtAtual", DateTime.MinValue);  
    DisplayMessage = $"Boa noite {Name}, Hoje é {data}";  
}
```



Crie o Construtor na classe. Dentro do construtor está sendo definido que o Botão declarado como variável vai executar o método ShowMessage. Utilize o atalho ctor

```
public UsuarioViewModel()
{
    ShowMessageCommand = new Command(ShowMessage);
}
```

Abra a view UsuarioView.Xaml e arraste um Button e uma Label. Será feita a vinculação da Label com a Propriedade **DisplayMessage** e do Button com o evento **ShowMessageCommand**.

```
<Label Text="{Binding DisplayMessage}" HorizontalOptions="CenterAndExpand"
       VerticalOptions="CenterAndExpand"/>

<Button Command="{Binding ShowMessageCommand}" Text="Mensagem" />
```

- Execute o aplicativo para testar.



Command – Continuação

1. Crie o método **CountCharacters** na classe **UsuarioViewModel** conforme abaixo

```
public async Task CountCharacters()
{
    string nameLength =
        string.Format("Seu nome tem {0} Letras", name.Length);

    await Application.Current
        .MainPage.DisplayAlert("Informação", nameLength, "Ok");
}
```

2. Declare um *ICommand* chamado **CountCommand**

```
public ICommand CountCommand { get; }
```

3. Faça a vinculação entre o comando e o método criado:

```
public UsuarioViewModel()
{
    ShowMessageCommand = new Command(ShowMessage);
    CountCommand = new Command(async() => await CountCharacters());
}
```

4. Insira um Button na view UsuarioView.xaml, realizando a referência ao Command da classe ViewModel

```
<Button Command="{Binding CountCommand}" Text="Contar Caracteres" />
```



Confirmação do usuário

5. Crie o método abaixo para exibir um alert ao usuário que a resposta sendo positiva, limpará os campos da tela. Faça o using de System.Threading.Tasks.

```
public async Task CleanConfirmation()
{
    if (await Application.Current.MainPage
        .DisplayAlert("Confirmação", "Confirma limpeza dos dados?", "Yes", "No"))
    {
        Name = string.Empty;
        DisplayMessage = string.Empty;
        OnPropertyChanged(Name);
        OnPropertyChanged(DisplayMessage);

        await Application.Current.MainPage
            .DisplayAlert("Informação", "Limpeza realizada com sucesso", "Ok");
    }
}
```

6. Declare um *ICommand* conforme a seguir:

```
public ICommand CleanCommand { get; }
```

7. Vincule o comando a uns dos métodos criados anteriormente no construtor da *ViewModel*

```
public UsuarioViewModel()
{
    ShowMessageCommand = new Command>ShowMessage();
    CountCommand = new Command(async() => await CountCharacters());
    CleanCommand = new Command(async () => await CleanConfirmation());
}
```

8. Arraste um Button para a View referenciando o método *CleanCommand* na propriedade *Command*

```
<Button Command="{Binding CleanCommand}" Text="Limpar Campos" />
```



Opções através dos Commands

9. Crie o método **ShowOptions** conforme abaixo

```
public async Task ShowOptions()
{
    string result = await Application.Current.MainPage
        .DisplayActionSheet("Selecione uma opção: ", "", 
        "Cancelar", "Limpar", "Contar Caracteres", "Exibir Saudação");

    if (result != null)
    {
        if(result.Equals("Limpar"))
            await CleanConfirmation();
        if (result.Equals("Contar Caracteres"))
            await CountCharacters();
        if (result.Equals("Exibir Saudação"))
            ShowMessage();
    }
}
```

10. Declare o *ICommand OptionCommand*

```
public ICommand OptionCommand { get; }
```

11. Vincule o Comando ao método

```
public UsuarioViewModel()
{
    ShowMessageCommand = new Command>ShowMessage);
    CountCommand = new Command(async() => await CountCharacters());
    CleanCommand = new Command(async () => await CleanConfirmation());
    OptionCommand = new Command(async () => await ShowOptions());
}
```



12. Arraste um Button para a View referenciando o método **OptionCommand** na propriedade *Command*

```
<Button Command="{Binding OptionCommand}" Text="Exibir Opções" />
```

- Execute o App para testar a programação

13. Na parte de código da View, faremos uma modificação no construtor para permitir que a ViewModel possa ser acessada em outras partes do código. Para isso iremos declarar de forma global a ViewModel, instanciaremos ela dentro do construtor e atribuiremos ao contexto da View.

```
private UsuarioViewModel viewModel;  
0 references  
public UsuarioView()  
{  
    InitializeComponent();  
    viewModel = new UsuarioViewModel();  
    BindingContext = viewModel;  
}
```

Com isso fechamos o entendimento do ciclo de vida da aplicação e o uso inicial do MVVM. A partir das próximas aulas aprofundaremos esses conhecimentos na construção de aplicações mais completas.



Aula 04/05 – Tela de registro e login – Classe de Serviço - Configurações do App

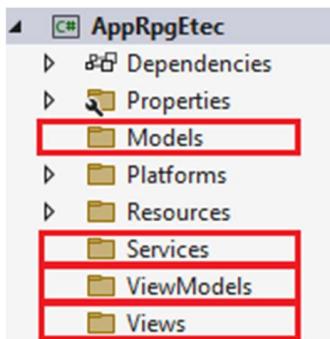
Crie um projeto **.NET MAUI App** com as seguintes recomendações:

Project name: **AppRpgEtec**

Framework: **.NET 8.0**

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 a classe **Usuario** dentro da pasta Models

```
public class Usuario
{
    public int Id { get; set; }
    public string Username { get; set; }
    public string PasswordString { get; set; }
    public string Perfil { get; set; }
    public string Token { get; set; }
    public byte[] Foto { get; set; }
    public string Email { get; set; }
    public double? Latitude { get; set; }
    public double? Longitude { get; set; }
}
```



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

```
public async Task<int> PostReturnIntAsync<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
        throw new Exception(serialized);
}

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));
    else
        throw new Exception(serialized);

    return result;
}
```

- 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.



4. Crie uma pasta chamada **Usuarios** dentro de **Services** e dentro da pasta Usuarios crie a classe **UsuarioService**

```
3 references
public class UsuarioService : Request
{
    private readonly Request _request;

    private const string apiUrlBase = "https://xyz.azurewebsites.net/Usuarios";

    //ctor + TAB: Atalho para criar um construtor
    1 reference
    public UsuarioService()
    {
        _request = new Request();
    }

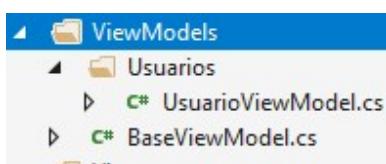
    1 reference
    public async Task<Usuario> PostRegistrarUsuarioAsync(Usuario u)
    {
        string urlComplementar = "/Registrar";
        u.Id = await _request.PostReturnIntAsync(apiUrlBase + urlComplementar, u, string.Empty);

        return u;
    }

    1 reference
    public async Task<Usuario> PostAutenticarUsuarioAsync(Usuario u)
    {
        string urlComplementar = "/Autenticar";
        u = await _request.PostAsync(apiUrlBase + urlComplementar, u, string.Empty);

        return u;
    }
}
```

- Esta classe herda a classe anterior, configura o endereço da API e usa o método Post para consumir a API
 - Faça using para a pasta Models já que temos uma classe de modelo.
5. Na pasta **ViewModels**, crie a classe **BaseViewModel** e uma pasta **Usuarios** com uma classe **UsuarioViewModel**. A estrutura ficará conforme abaixo:





-
6. 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.

```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Runtime.CompilerServices;
5 using System.Text;

6 namespace AppRpgHAS.ViewModels
7 {
8     1 reference
9     public class BaseViewModel : INotifyPropertyChanged
10    {
11        1 reference
12        public event PropertyChangedEventHandler PropertyChanged;
13
14        0 references
15        public void OnPropertyChanged([CallerMemberName] string name = "")
16        {
17            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
18        }
19    }
20 }
```

- A programação exigirá os *usings* sinalizados em verde. Verifique se eles já existem.
7. Na classe *UsuarioViewModel*, realize a herança da classe *BaseViewModel* e declare um *command*, uma variável do tipo Usuario e a classe de serviço do tipo UsuarioService

```
public class UsuarioViewModel : BaseViewModel
{
    1 reference
    private UsuarioService uService;
    0 references
    public ICommand AutenticarCommand { get; set; }
```

- Realize as referências às classes para que sejam reconhecidas através do *using*.



-
8. Crie as propriedades que serão usadas futuramente na View

```
#region AtributosPropriedades
//As propriedades serão chamadas na View futuramente

private string login = string.Empty;
0 references
public string Login
{
    get { return login; }
    set
    {
        login = value;
        OnPropertyChanged();
    }
}

private string senha = string.Empty;
0 references
public string Senha
{
    get { return senha; }
    set
    {
        senha = value;
        OnPropertyChanged();
    }
}
#endregion
```

9. Inicie o método que fará a autenticação na API. A estrutura try/catch evitará que o app feche caso aconteça erro. Digite try e clique no TAB duas vezes para completar o bloco try. Faça a edição conforme abaixo.

```
public async Task AutenticarUsuario() //Método para autenticar um usuário
{
    try
    {
        //Próxima codificação aqui
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage
            .DisplayAlert("Informação", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
```



-
10. Realize a programação dentro do bloco try. Perceba que ao realizar a autenticação, iremos guardar alguns dos dados do usuário que retorna da API em Preferences, para poder usar essas informações mais à frente.

```
try
{
    Usuario u = new Usuario();
    u.Username = Login;
    u.PasswordString = Senha;

    Usuario uAutenticado = await uService.PostAutenticarUsuarioAsync(u);

    if (!string.IsNullOrEmpty(uAutenticado.Token))
    {
        string mensagem = $"Bem-vindo(a) {uAutenticado.Username}.";

        //Guardando dados do usuário para uso futuro
        Preferences.Set("UsuarioId", uAutenticado.Id);
        Preferences.Set("UsuarioUsername", uAutenticado.Username);
        Preferences.Set("UsuarioPerfil", uAutenticado.Perfil);
        Preferences.Set("UsuarioToken", uAutenticado.Token);

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

        Application.Current.MainPage = new MainPage();
    }
    else
    {
        await Application.Current.MainPage
            .DisplayAlert("Informação", "Dados incorretos :(", "Ok");
    }
}
```



11. Crie um método para vincular o método da etapa anterior para o Comando declarado no início da classe conforme (1) e inicialize os objetos e fazendo chamada para o método que inicializa os commands conforme em (2).

```
public class UsuarioViewModel : BaseViewModel
{
    private UsuarioService uService;
    1 reference
    public ICommand AutenticarCommand { get; set; }

    //ctor + TAB + TAB: Atalho para criar o construtor
    0 references
    public UsuarioViewModel()
    {
        uService = new UsuarioService();
        InicializarCommands();
    }
    1 reference
    public void InicializarCommands()
    {
        AutenticarCommand = new Command(async () => await AutenticarUsuario());
    }
}
```

- Até a aula anterior tudo era feito diretamente no construtor, mas agora faremos da forma acima para deixar o código mais organizado, e no construtor ficará apenas a chamada para este método.

12. Na pasta Views, crie uma pasta chamada **Usuarios** e dentro dela, uma *content page* (.Net MAUI) chamada **LoginView.Xaml**. Remova a *label* que está dentro do *StackLayout* e insira o layout abaixo:

```
<Entry Placeholder="Digite seu nome de usuário" Text="{Binding Login}"
       Margin="0,10,0,0" VerticalOptions="FillAndExpand"
       HorizontalOptions="FillAndExpand">
</Entry>
<Entry Placeholder="Digite a senha" Text="{Binding Senha}" IsPassword="True"
       Margin="0,10,0,0" VerticalOptions="FillAndExpand" HorizontalOptions="FillAndExpand" >
</Entry>
<Button Text="Entrar" Command="{Binding AutenticarCommand}" Margin="0,10,0,0"/>
```

- Perceba que já colocamos o *Binding* para Login e Senha também já declarados como propriedade na *ViewModel*.



13. Na parte de código da *View* (LoginView.xaml.cs), faça a declaração da *ViewModel* e logo após inicialize a mesma no construtor, atribuindo-a como contexto da *View*.

```
public partial class LoginView : ContentPage
{
    UsuarioViewModel usuarioViewModel;
    1 reference
    public LoginView()
    {
        InitializeComponent();

        usuarioViewModel = new UsuarioViewModel();
        BindingContext = usuarioViewModel;
    }
}
```

14. Na classe App.xaml.cs, altere para que a página inicial seja a recém-criada:

```
2 references
public App()
{
    InitializeComponent();
    MainPage = new NavigationPage(new Views.Usuarios.LoginView());
}
```

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

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

- Confirme que o endereço da API está inserido na classe UsuarioViewModel.
- Compile para certificar que está sem erros e execute o app tentando realizar o login, faça uma tentativa também com dados incorretos.



16. Inicie o método de registro. A estrutura try/catch evitárá que o app feche caso aconteça erro. Digite try e clique no TAB duas vezes para completar o bloco try. Faça a edição conforme abaixo.

```
#region Métodos
0 references
public async Task RegistrarUsuario()//Método para registrar um usuário
{
    try
    {
        //Próxima codificação aqui
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage
            .DisplayAlert("Informação", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
#endif
```

17. Continue a codificação do método programando a codificação abaixo dentro do try. Faça o using de AppRpqEtec.Models para que a classe Usuario seja reconhecida

```
try
{
    Usuario u = new Usuario();
    u.Username = Login;
    u.PasswordString = Senha;

    Usuario uRegistrado = await uService.PostRegistrarUsuarioAsync(u);

    if (uRegistrado.Id != 0)
    {
        string mensagem = $"Usuário Id {uRegistrado.Id} registrado com sucesso.";
        await Application.Current.MainPage.DisplayAlert("Informação", mensagem, "Ok");

        await Application.Current.MainPage
            .Navigation.PopAsync(); //Remove a página da pilha de visualização
    }
}
```

18. Declare um ICommand (1) e depois faça a vinculação dele ao método RegistrarUsuario (2).

```
1 reference
public ICommand AutenticarCommand { get; set; }
1 reference
1 public ICommand RegistrarCommand { get; set; }

1 reference
public void InicializarCommands()
{
    AutenticarCommand = new Command(async () => await AutenticarUsuario());
    RegistrarCommand = new Command(async () => await RegistrarUsuario());
2
```



19. Clique com o direito na pasta *Views/Usuarios* e crie uma View chamada de **CadastroView.xaml**, adicionando o layout abaixo dentro do *VerticalStackLayout*:

```
<Entry Placeholder="Digite seu nome de usuário" Text="{Binding Login}"  
       Margin="0,10,0,0" VerticalOptions="FillAndExpand"  
       HorizontalOptions="FillAndExpand">  
</Entry>  
<Entry Placeholder="Digite a senha" Text="{Binding Senha}" IsPassword="True"  
       Margin="0,10,0,0" VerticalOptions="FillAndExpand"  
       HorizontalOptions="FillAndExpand" >  
</Entry>  
<Button Text="Registrar" Command="{Binding RegistrarCommand}" Margin="0,10,0,0"/>
```

20. Clique em F7 (View Code) para navegar até a parte de programação desta view e realize a programação a seguir responsável por fazer a vinculação com viewModel.

```
public partial class CadastroView : ContentPage  
{  
    UsuarioViewModel viewModel;  
    0 references  
    public CadastroView()  
    {  
        InitializeComponent();  
  
        viewModel = new UsuarioViewModel();  
        BindingContext = viewModel;  
    }  
}
```

21. Volte para a classe *UsuarioViewModel* e crie o método abaixo. Faça o using para *Views.Usuarios*.

```
public async Task DireccionarParaCadastro() //Método para exibição da view de Cadastro  
{  
    try  
    {  
        await Application.Current.MainPage.  
            Navigation.PushAsync(new CadastroView());  
    }  
    catch (Exception ex)  
    {  
        await Application.Current.MainPage  
            .DisplayAlert("Informação", ex.Message + " Detalhes: " + ex.InnerException, "Ok");  
    }  
}
```

22. Declare mais um command e realize a vinculação do mesmo com o método *DireccionarParaCadastro*

```
1 reference  
public ICommand DireccionarCadastroCommand { get; set; }  
  
public void InicializarCommands()  
{  
    RegistrarCommand = new Command(async ()=> await RegistrarUsuario());  
    AutenticarCommand = new Command(async () => await AutenticarUsuario());  
    DireccionarCadastroCommand = new Command(async () => await DireccionarParaCadastro());  
}
```



23. Abra a view **LoginView.Xaml** e adicione o trecho de layout abaixo do botão de login para exibir opção de registro do usuário. Perceba que estamos usando um StackLayout horizontal e que ele permite reconhecimento de gestos. Ou seja, ao clicar neste trecho da tela, invocaremos um Command na ViewModel.

```
<HorizontalStackLayout
    HorizontalOptions="Center"
    Spacing="20" Margin="30">
    <Label Text="Não possui login?"></Label>
    <Label Text="Cadastre-se" FontAttributes="Bold"></Label>
    <HorizontalStackLayout.GestureRecognizers>
        <TapGestureRecognizer Command="{Binding DirecionarCadastroCommand}">
        </TapGestureRecognizer>
    </HorizontalStackLayout.GestureRecognizers>
</HorizontalStackLayout>
```

- Faça o teste no dispositivo, cadastrando um novo usuário e depois autenticando o usuário recém-cadastrado.

Indicação de conteúdos:

- Construção de página de Login MAUI: <https://youtu.be/LzORBRQfeFU>
- Playlist construção de Layout inicial MAUI: <https://www.youtube.com/playlist?list=PLn-SpzWnVxDdJvDS1RZeISI0kBbhOYua->



Aula 06 - Consumindo APIs – Listagem de Personagens do jogo RPG

1. 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
}
```

2. Clique com o botão direito na pasta Models e crie uma classe chamada **Personagem**, com as propriedades abaixo. Será necessário o using para *AppRpgEtec.Models.Enuns*

```
public class Personagem
{
    0 references
    public int Id { get; set; }
    0 references
    public string Nome { get; set; }
    0 references
    public int PontosVida { get; set; }
    0 references
    public int Forca { get; set; }
    0 references
    public int Defesa { get; set; }
    0 references
    public int Inteligencia { get; set; }
    0 references
    public byte[] FotoPersonagem { get; set; }
    0 references
    public int Disputas { get; set; }
    0 references
    public int Vitorias { get; set; }
    0 references
    public int Derrotas { get; set; }
    0 references
    public ClasseEnum Classe { get; set; }
}
```



3. Abra a classe **Request** e adicione os demais métodos que a API utilizará.

- Método Put

```
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
        throw new Exception(serialized);
}
```

- Método Get

```
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();

    if (response.StatusCode != System.Net.HttpStatusCode.OK)
        throw new Exception(serialized);

    TResult result = await Task.Run(() => JsonConvert.DeserializeObject<TResult>(serialized));
    return result;
}
```

- Método Delete

```
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
        throw new Exception(serialized);
}
```



4. 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 instância 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.

```
1 reference
public class PersonagemService : Request
{
    private readonly Request _request;
    private const string apiUrlBase = "https://xyz.azurewebsites.net/Personagens";
    //xyz --> site da sua API

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

    //Próximos métodos aqui
}
```

5. Adicione os métodos que consomem a API com o usings que o Visual Studio sugerir.

```
public async Task<int> PostPersonagemAsync(Personagem p)
{
    return await _request.PostReturnIntAsync(apiUrlBase, p, _token);
}

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> 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;
}
```

6. 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

```
1 reference
public class ListagemPersonagemViewModel : BaseViewModel
{
    1 private PersonagemService pService;
    2 public ObservableCollection<Personagem> Personagens { get; set; }
    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*;



7. Programe o método de busca dos personagens.

```
public async Task ObterPersonagens()
{
    try //Junto com o Cacth evitara 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");
    }
}
```

8. 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 e armazenar um retorno.

9. Crie uma pasta Personagens dentro de Views e dentro crie a .Net MAUI/Content Page **ListagemView.xaml**, remova o VerticalStackLayout que está dentro da tag ContentPage.Content e adicione o conteúdo abaixo.

```
<ScrollView>
    <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>
</ScrollView>
```



10. 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();

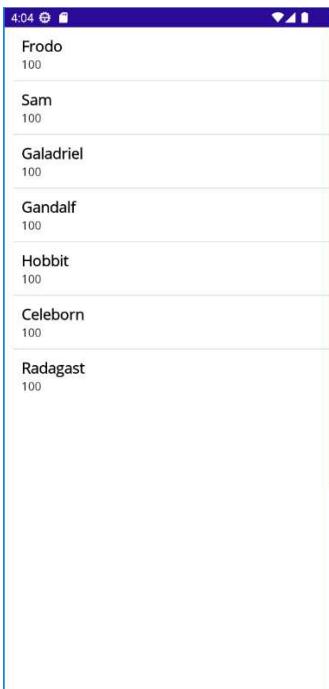
        viewModel = new ListagemPersonagemViewModel();
        BindingContext = viewModel;
        Title = "Personagens - App Rpg Etec";
    }
}
```

- Altere o método de login para que a view inicial possa ser a de listagem.
`Preferences.Set("UsuarioPerfil", uAutenticado.Perfil);`

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

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

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





Aula 07 - 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 referencia 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 inteligencia;
private int pontosVida;  private int disputas;
private int forca;        private int vitorias;
private int defesa;       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 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" A
    Shell.FlyoutBehavior="Disabled"
    Title="AppRpgEtec">

    <ShellContent
        Title="Home" B
        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(); });
    CancelarCommand = new Command(async => CancelarCadastro());
3 }
```

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



Aula 10 - Consumindo APIs – Atualização e remoção de Personagens do jogo RPG

1. Abra a classe **CadastroPersonagemViewModel** e insira 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;
```

2. 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.Derrotas = p.Derrotas;
        this.Disputas = p.Disputas;
        this.Forca = p.Forca;
        this.Inteligencia = p.Inteligencia;
        this.Vitorias = p.Vitorias;
        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");
    }
}
```



3. Ainda na classe **CadastroPersonagemViewModel**, 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();
        }
    }
}
```

4. Altere o ICommand **SalvarPersonagem** da classe **CadastroPersonagemViewModel** para caso a propriedade Id não seja 0 no 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");
```

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. Programa 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 initialize-o (2) no construtor da classe

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

    _ = ObterPersonagens();
    NovoPersonagemCommand = new Command(async () => { await ExibirCadastroPersonagem(); });
    RemoverPersonagemCommand =
    2    new Command<Personagem>(async (Personagem p) => { await RemoverPersonagem(p); });
}

1 reference
public ICommand NovoPersonagemCommand { get; }

0 references
1 public ICommand RemoverPersonagemCommand { get; set; }
```



-
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>
        </ViewCell>
    </DataTemplate>
</ListView.ItemTemplate>
```

- Execute o aplicativo para realizar os testes.



Atividade - Busca de Personagens para o CRUD de armas

1. Arraste a classe Arma da pasta da aula para dentro da pasta Models
2. Clique com o botão direito na pasta Services e crie uma pasta **Armas**
3. Arraste a classe **ArmasService** da pasta da aula para dentro da pasta Services/Armas
4. Clique com o botão direito na pasta ViewModels e crie uma pasta chamada **Armas**
5. Arraste a classe **CadastroArmaViewModel** da pasta da aula para dentro da pasta ViewMoldes/Armas
6. Clique com o botão direito na pasta Views e crie uma pasta chamada **Armas**
7. Clique com o direito na pasta Views/Armas e adicione uma content page chamada **CadastroArmaView.xaml**
8. Copie o layout inserido dentro da pasta da aula insira dentro da tag ContentPage.Content
9. Abra a parte de código da View (F7) e programe a vinculação da viewModel com a view

```
private CadastroArmaViewModel cadViewModel;
0 references
public CadastroArmaView()
{
    InitializeComponent();

    cadViewModel = new CadastroArmaViewModel();
    BindingContext = cadViewModel;
    Title = "Nova Arma";
}
```

10. Abra a view AppShell.Xaml e insira um item referenciando a pasta Views/Armas

```
xmlns:viewsDisputas ="clr-namespace:AppRpgEtec.Views.Disputas"
xmlns:viewsArmas ="clr-namespace:AppRpgEtec.Views.Armas"
xmlns:conv="clr-namespace:AppRpgEtec.Converters"
xmlns:local="clr-namespace:AppRpgEtec">

<Shell.Resources>
```



11. Insira um item de menu dentro da tag FlyoutItem, abaixo do item que aponta para listagem de personagens

```
<ShellContent Title="Personagens" Icon="MenuPersonagens.png"
    ContentTemplate="{DataTemplate viewsPersonagens:ListagemView}" />

<ShellContent Title="Armas" Icon="MenuArmas.png"
    ContentTemplate="{DataTemplate viewsArmas:CadastroArmaView}" />

<ShellContent Title="Disputas" Icon="MenuDisputas.png"
    ContentTemplate="{DataTemplate viewsDisputas:ListagemView}" />
```

- Com estas etapas você pode executar o aplicativo para verificar se a tela de cadastro de arma estará carregando e se o picker estará carregado com os personagens.

Atividade para entrega no Teams

12. Crie a view para listar as armas cadastradas ([Views/Armas/ListagemView.xaml](#)), adicione as funcionalidades para listar as armas cadastradas, selecionar uma arma cadastrada para edição e remover uma arma. (viewModel da listagem de armas estará postada no exercício).

13. Faça o mapeamento da página de cadastro na parte de código da Content page [AppShell.xaml](#).

14. Altere o menu flyout para que o ícone de espada agora abra a listagem de armas e posteriormente, ao clicar na arma ou no botão de adicionar, carregar o cadastro de armas.

Lembre-se: Como existe um relacionamento 1:1 entre Personagem e Arma, se você tentar salvar uma nova arma para um personagem que já possui uma, acontecerá um erro.



Atividade - Listagem de Armas

1. Arraste a classe Arma da pasta da aula para dentro da pasta **Models**
2. Clique com o botão direito na pasta **Services** e crie uma pasta **Armas**
3. Arraste a classe **ArmasService** da pasta da aula para dentro da pasta **Services/Armas**
4. Clique com o botão direito na pasta **ViewModels** e crie uma pasta chamada **Armas**
5. Arraste a classe **ListagemArmaViewModel** da pasta da aula para dentro da pasta **ViewMoldes/Armas**
6. Clique com o botão direito na pasta **Views** e crie uma pasta chamada **Armas**
7. Clique com o direito na pasta **Views/Armas** e adicione uma content page chamada **ListagemView.xaml**
8. Crie o Layout para apresentar um listView de Armas que apresente o Nome e o Dano da arma, conforme exemplo que fizemos com os personagens.
9. Abra a parte de código da View (F7) e programe a vinculação da viewModel com a view
10. Abra a view AppShell.Xaml e insira um item referenciando a pasta Views/Armas

```
xmlns:viewsDisputas ="clr-namespace:AppRpgEtec.Views.Disputas"  
xmlns:viewsArmas ="clr-namespace:AppRpgEtec.Views.Armas"  
xmlns:conv="clr-namespace:AppRpgEtec.Converters"  
xmlns:local="clr-namespace:AppRpgEtec">
```

```
<Shell.Resources>
```

11. Insira um item de menu dentro da tag FlyoutItem, abaixo do item que aponta para listagem de personagens

```
<ShellContent Title="Personagens" Icon="MenuPersonagens.png"  
ContentTemplate="{DataTemplate viewsPersonagens:ListagemView}" />  
  
<ShellContent Title="Armas" Icon="MenuArmas.png"  
ContentTemplate="{DataTemplate viewsArmas:ListagemView }" />  
  
<ShellContent Title="Disputas" Icon="MenuDisputas.png"  
ContentTemplate="{DataTemplate viewsDisputas:ListagemView}" />
```

- Correção do print: o nome da imagem deve ter a extensão **.svg**
- Com estas etapas você pode executar o aplicativo para verificar se a tela de listagem de arma estará carregando e se o picker estará carregado com os personagens.



PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2024/2

Luiz Fernando Souza

Como deverá ser a entrega da Avaliação:

- Chamar o professor para demostrar o funcionamento da View de armas
- Criar arquivo Word/Pdf. Mencionar o nome da dupla no arquivo. Relacionar o print da listagem de armas.
- Realizar a entrega via Teams



Atividade - Busca de Personagens para o Cadastro de armas

1. Arraste a classe **CadastroArmaViewModel** da pasta da aula para dentro da pasta ViewMoldes/Armas
2. Clique com o direito na pasta Views/Armas e adicione uma content page chamada **CadastroArmaView.xaml**
3. Copie o layout inserido dentro da pasta da aula insira dentro da tag ContentPage.Content
4. Abra a parte de código da View (F7) e programe a vinculação da viewModel com a view

```
private CadastroArmaViewModel cadViewModel;
0 references
public CadastroArmaView()
{
    InitializeComponent();

    cadViewModel = new CadastroArmaViewModel();
    BindingContext = cadViewModel;
    Title = "Nova Arma";
}...
```

- Com estas etapas você pode executar o aplicativo para verificar se a tela de cadastro de arma estará carregando e se o picker estará carregado com os personagens.

Atividade para entrega no Teams

5. Altere a viewModel e a view de listagem de armas para conter um botão de adicionar, que ao clicar carregará o cadastro de armas.

Lembre-se: Como existe um relacionamento 1:1 entre Personagem e Arma, se você tentar salvar uma nova arma para um personagem que já possui uma, acontecerá um erro.



Configurações do App: Splash Screen, Ícone e nome do Aplicativo

A criação de uma Splash Screen tem por objetivo fazer uma apresentação inicial enquanto outros componentes do aplicativo são carregados. O template do projeto já possui uma criada, mas mostraremos como customizá-la.

1. Arraste a imagem chamada `has_logo_transparente.svg` para a pasta `Resources/Splash`, para a pasta `Resources/AppIcon` e para a pasta `Resources/Images` do projeto. Você pode usar o site <https://convertio.co/pt/png-svg/> para converter imagens de outros tipos em svg, formato recomendado para imagens no app.
 2. Clique no arquivo de projeto  **AppRpgEtec** e altere o nome da imagem para o ícone e para a Splash. Aproveite e altere as cores de fundo para Branco.

```
<ItemGroup>
    <!-- App Icon -->
    <MauiIcon Include="Resources\AppIcon\has_logo_transparente.svg" ForegroundFile="Resources\AppIcon\appiconfg.svg" Color="#FFFFFF" />

    <!-- Splash Screen -->
    <MauiSplashScreen Include="Resources\Splash\has_logo_transparente.svg" Color="#FFFFFF" BaseSize="128,128" />
```

- Remova a tag ForeGroundFile para que a palavra “.NET” não fique sobre o ícone

- ### 3. Altere o nome de exibição do Aplicativo

```
<!-- Display name -->
<ApplicationTitle>Rpg Etec</ApplicationTitle>
```

- Desinstale o aplicativo do emulador ou dispositivo e execute novamente.

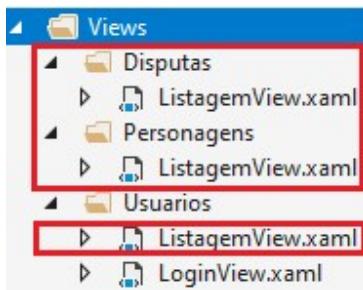
- Entre no arquivo Platforms/Android/AndroidManifest.xml e realize a menção para a nova imagem, conforme sinalizado.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <application android:allowBackup="true" android:icon="@mipmap/has_logo_transparente"
        android:roundIcon="@mipmap/has_logo_transparente_round"
        android:supportsRtl="true">
    </application>
```



Aula 12 – Menu Flyout em páginas Shell – Layout, Estilos e Cores

- Atualize a *Solution* com as seguintes pastas (clicar com direito na pasta Views → add → new Folder) e ContentPages (clicar com o direito nas pastas → add → new item → content page).



- Abra a View AppShell.Xaml, remova a tag FlyoutBehaviour (1) e adicione o um template que exibirá no topo da página shell uma imagem e informações do App (2). O Layout (2) encontra-se copiável na sequência deste arquivo.

```
<Shell x:Class="AppRpgEtec.AppShell"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:AppRpgEtec"
    FlyoutBehavior="Disabled"> 1
2
<Shell.FlyoutHeaderTemplate>
    <DataTemplate>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="{OnPlatform Android=130, iOS=200}"/>
                <RowDefinition Height="*"/>
                <RowDefinition Height="40"/>
            </Grid.RowDefinitions>
            <FlexLayout Grid.Row="0" Direction="Row" AlignItems="Center" >
                <Frame Padding="0" CornerRadius="40"
                    HeightRequest="80" WidthRequest="80" >
                    <Image Source="has_logo_transparente.svg"/>
                </Frame>
                <VerticalStackLayout Padding="10,0,0,0" Spacing="0" >
                    <Label Text="App RPG Etec" FontAttributes="Bold" FontSize="Medium" />
                    <Label Text="Disciplina PAM-II"/>
                </VerticalStackLayout>
            </FlexLayout>
        </Grid>
    </DataTemplate>
</Shell.FlyoutHeaderTemplate>

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



```
<Shell.FlyoutHeaderTemplate>
    <DataTemplate>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="{OnPlatform Android=130, iOS=200}" />
                <RowDefinition Height="*" />
                <RowDefinition Height="40" />
            </Grid.RowDefinitions>
            <FlexLayout Grid.Row="0" Direction="Row" AlignItems="Center" >
                <Frame Padding="0" CornerRadius="40"
                    HeightRequest="80" WidthRequest="80" >
                    <Image />
                </Frame>
                <VerticalStackLayout Padding="10,0,0,0" Spacing="0">
                    <Label Text="App RPG Etec" FontAttributes="Bold" FontSize="Medium" />
                    <Label Text="Disciplina PAM-II" />
                </VerticalStackLayout>
            </FlexLayout>
        </Grid>
    </DataTemplate>
</Shell.FlyoutHeaderTemplate>
```

3. Adicione mais referências para as pastas das views que usaremos na próxima etapa.

```
<Shell x:Class="AppRpgEtec.AppShell"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:viewsUsuarios ="clr-namespace:AppRpgEtec.Views.Usuarios"
    xmlns:viewsPersonagens ="clr-namespace:AppRpgEtec.Views.Personagens"
    xmlns:viewsDisputas ="clr-namespace:AppRpgEtec.Views.Disputas"
    xmlns:local ="clr-namespace:AppRpgEtec">

    <Shell.FlyoutHeaderTemplate>
        <DataTemplate>
            <!-- -->
```

4. Adicione uma imagem para o item de menu existe e adicione mais itens de menu para as páginas de listagem

```
</DataTemplate>
</Shell.FlyoutHeaderTemplate>

<ShellContent Title="Home" Icon="home.svg"
    ContentTemplate="{DataTemplate local: MainPage}" Route="MainPage" />

<ShellContent Title="Usuários" Icon="menuusuarios.svg"
    ContentTemplate="{DataTemplate viewsUsuarios: ListagemView}" />

<ShellContent Title="Personagens" Icon="menupersonagens.svg"
    ContentTemplate="{DataTemplate viewsPersonagens: ListagemView}" />

<ShellContent Title="Disputas" Icon="menudisputas.svg"
    ContentTemplate="{DataTemplate viewsDisputas: ListagemView}" />

</Shell>
```



5. Insira as imagens da pasta da aula dentro da pasta Resources → Images.
6. Abra a classe UsuarioViewModel dentro da pasta ViewModels/Usuarios e confirme se a página que será chamada após o login será a página que contém o menu.

```
public async Task AutenticarUsuario() //Método para autenticar um usuário
{
    try
    {
        Usuario u = new Usuario();
        u.Username = Login;
        u.PasswordString = Senha;

        Usuario uAutenticado = await uService.PostAutenticarUsuarioAsync(u);

        if (!string.IsNullOrEmpty(uAutenticado.Token))
        {
            string mensagem = $"Bem-vindo(a) {uAutenticado.Username}.";

            //Guardando dados do usuário para uso futuro
            Preferences.Set("UserId", uAutenticado.Id);
            Preferences.Set("UsuarioUsername", uAutenticado.Username);
            Preferences.Set("UsuarioPerfil", uAutenticado.Perfil);
            Preferences.Set("UsuarioToken", uAutenticado.Token);

            await Application.Current.MainPage
                .DisplayAlert("Informação", mensagem, "Ok");
            Application.Current.MainPage = new AppShell(); ←
        }
        else
        {
            await Application.Current.MainPage
                .DisplayAlert("Informação", "Dados incorretos :(", "Ok");
        }
    }
}
```

- Execute o aplicativo para realizar os testes necessários.



7. Clique com o direito no projeto (C#) e adicione duas *content pages*: **AboutView.xaml** e **DescriptionView.xaml**.
8. Faça as melhorias abaixo para que possamos criar ícones dos menus no rodapé da página Shell (FlyOutItem - 1), para criar uma página de menu que contenha duas abas (2 - Tab), para criar um item de menu para sair do app que aparecerá apenas no menu lateral (3) e um rodapé para o menu (4 - Shell.FlyOutFooter).

```
</Shell.FlyoutHeaderTemplate>

1 <FlyoutItem FlyoutDisplayOptions="AsMultipleItems">

    <ShellContent Title="Home" Icon="home.svg"
        ContentTemplate="{DataTemplate local:MainPage}" Route="MainPage" />

    <ShellContent Title="Usuários" Icon="menuusuarios.svg"
        ContentTemplate="{DataTemplate viewsUsuarios:ListagemView}" />

    <ShellContent Title="Personagens" Icon="menupersonagens.svg"
        ContentTemplate="{DataTemplate viewsPersonagens:ListagemView}" />

    <ShellContent Title="Disputas" Icon="menudisputas.svg"
        ContentTemplate="{DataTemplate viewsDisputas:ListagemView}" />

2 <Tab Title="Info" Route="info" Icon="info.svg">
    <ShellContent Title="Sobre" Route="sobre" ContentTemplate="{DataTemplate local:AboutView}" />
    <ShellContent Title="Descrição" Route="desc" ContentTemplate="{DataTemplate local:DescriptionView}" />
</Tab>

1 </FlyoutItem>

3 <ShellContent Title="Sair" Icon="Exit.png" ContentTemplate="{DataTemplate viewsUsuarios:LoginView}"
    Shell.FlyoutBehaviour="Disabled"/>

4 <Shell.FlyoutFooter>
    <Label x:Name="lblLogin" TextColor="Black" FontAttributes="Bold" HorizontalOptions="Center" />
</Shell.FlyoutFooter>

</Shell>
```

- Observe que o item de menu “sair” possui uma propriedade Shell.FlyoutBehaviour com valor false, para que quando o usuário clique em sair o menu não mais fique disponível.



9. Entre na parte de código da página AppShell.xaml (F7) para recuperar dados do usuário da Preference e atribuir ao label que adicionamos no design

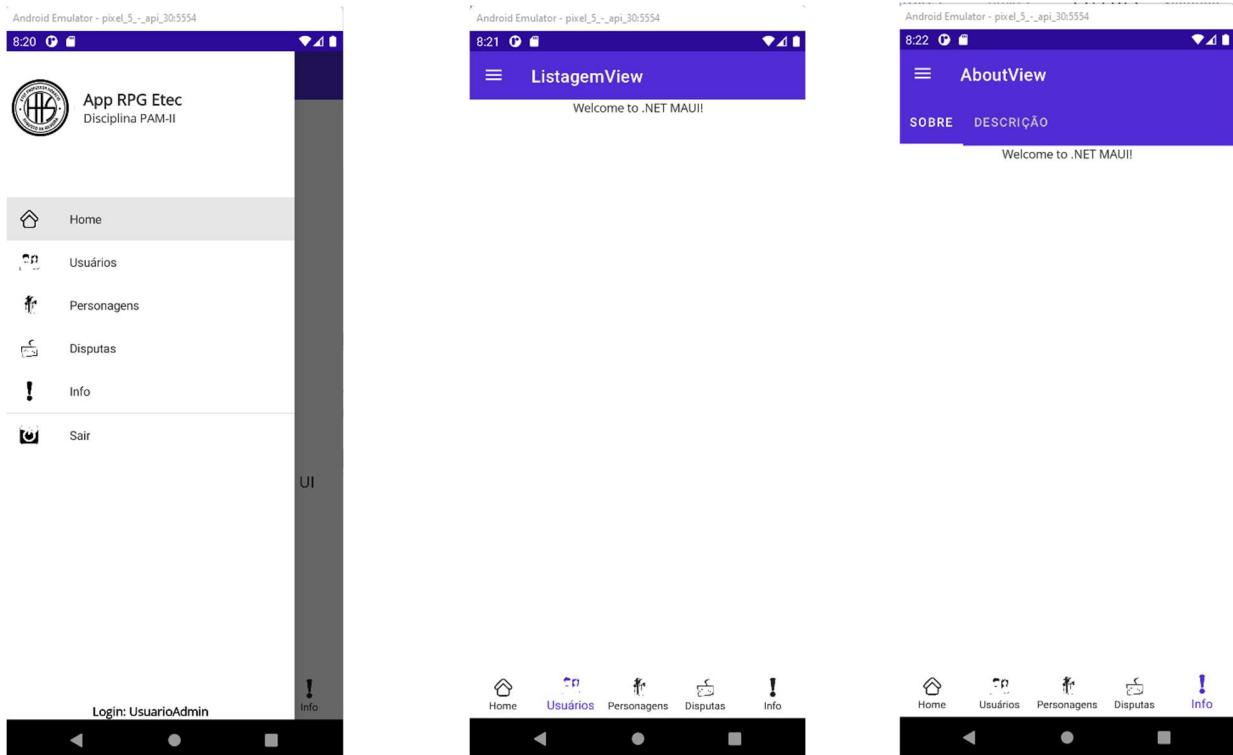
```
using AppRpgEtec.Views.Personagens;

namespace AppRpgEtec
{
    5 references
    public partial class AppShell : Shell
    {
        1 reference
        public AppShell()
        {
            InitializeComponent();

            string login = Preferences.Get("UsuarioUsername", string.Empty);
            lblLogin.Text = login;

            Routing.RegisterRoute("cadPersonagemView", typeof(CadastroPersonagemView));
        }
    }
}
```

Resultados esperados:





Cores e Estilos

1. Abra o arquivo Resources/Styles/Colors.xaml e crie as tags abaixo. Definiremos neste arquivo cores nomeadas por nomenclaturas que poderemos usar em outras partes do projeto.

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

    <Color x:Key="MainColor">#f8cc45</Color>
    <Color x:Key="DarkColor">Black</Color>
    <Color x:Key="DarkTextColor">Black</Color>

    <Color x:Key="Primary">#512BD4</Color>
    <Color x:Key="Secondary">#DFD8F7</Color>
    <Color x:Key="Tertiary">#2B0B98</Color>
```

2. Abra o arquivo Resources/Styles/Styles.xaml e crie o estilo abaixo

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

    <Style x:Key="WhiteButtonSmall" TargetType="Button">
        <Setter Property="BackgroundColor" Value="White" />
        <Setter Property="TextColor" Value="Black" />
        <Setter Property="HeightRequest" Value="36" />
        <Setter Property="CornerRadius" Value="18" />
        <Setter Property="FontSize" Value="10" />
        <Setter Property="WidthRequest" Value="80" />
    </Style>
```

3. Abra a view MainPage.Xaml e delete os elementos dentro da tag ContentPage. Insira o layout abaixo

```
<ContentPage.Content>
    <VerticalStackLayout Padding="25" Spacing="30">
        <Frame
            BackgroundColor="{StaticResource MainColor}"
            HeightRequest="{OnPlatform Android=190, iOS=220}"
            Padding="0" HasShadow="false" CornerRadius="10">
            <Grid>
                <Image Source="dotnet_bot.svg"/>
                <FlexLayout Direction="Row" Margin="20" AlignItems="Stretch">
                    <Label Text="RPG App" FlexLayout.Grow="1" FontAttributes="Bold"
                           FontSize="20" VerticalTextAlignment="Center"/>
                    <FlexLayout FlexLayout.Grow="2" Direction="Column"
                               AlignItems="End" JustifyContent="Center">
                        <Label Text="Usuários" FontSize="20"/>
                        <Button Style="{StaticResource WhiteButtonSmall}" Text="VER"/>
                    </FlexLayout>
                </Grid>
            </Frame>
        </VerticalStackLayout>
    </ContentPage.Content>
```



A montagem do layout ficará conforme abaixo. Perceba que temos a configuração da cor *MainColor*, inserida no arquivo de cores sendo mencionada no *BackgroundColor* através do *StaticResource*. Perceba também que o botão presente no layout está recebendo o estilo *WhiteButtonSmall* configurado no arquivo de estilos. Execute o aplicativo para testar as alterações.

```
<?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.MainPage">

    <ContentPage.Content>
        <VerticalStackLayout Padding="25" Spacing="30">
            <Frame
                BackgroundColor="{StaticResource MainColor}"
                HeightRequest="{OnPlatform Android=190, iOS=220}"
                Padding="0" HasShadow="false" CornerRadius="10">
                <Grid>
                    <Image Source="dotnet_bot.svg"/>
                    <FlexLayout Direction="Row" Margin="20" AlignItems="Stretch">
                        <Label Text="RPG App" FlexLayout.Grow="1" FontAttributes="Bold"
                            FontSize="20" VerticalTextAlignment="Center"/>
                        <FlexLayout FlexLayout.Grow="2" Direction="Column"
                            AlignItems="End" JustifyContent="Center">
                            <Label Text="Usuários" FontSize="20"/>
                            <Button Style="{StaticResource WhiteButtonSmall}" Text="VER"/>
                        </FlexLayout>
                    </FlexLayout>
                </Grid>
            </Frame>
        </VerticalStackLayout>
    </ContentPage.Content>
</ContentPage>
```

4. Insira as tags abaixo antes de “*ContentPage.Content*” para testarmos mais uma forma declarar recursos de layout.

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="AppRpgEtec.MainPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <Style x:Key="FrameOptions" TargetType="Frame">
                <Setter Property="Margin" Value="10" />
                <Setter Property="HasShadow" Value="True" />
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>

    <ContentPage.Content>
```



5. Insira as tags abaixo do Frame mais externo

```
<Label Text="Este App oferece recursos para" FontAttributes="Bold" FontSize="Medium"/>
    <ScrollView Orientation="Horizontal">
        <HorizontalStackLayout>
            <Frame Style="{StaticResource FrameOptions}" Padding="0"
HeightRequest="{OnPlatform Android=150, iOS=140}"
                WidthRequest="{OnPlatform Android=120, iOS=140}">
                    <VerticalStackLayout Spacing="0" VerticalOptions="Center">
                        <Image Source="menupersonagens.svg"/>
                        <Label Text="Personagens" FontSize="Small"
HorizontalTextAlignment="Center" Margin="10,0"/>
                    </VerticalStackLayout>
                </Frame>
                <Frame Style="{StaticResource FrameOptions}" Padding="0"
HeightRequest="{OnPlatform Android=150, iOS=140}"
                WidthRequest="{OnPlatform Android=120, iOS=140}">
                    <VerticalStackLayout Spacing="30" VerticalOptions="Center">
                        <Image Source="menuarmas.png"/>
                        <Label Text="Armas" FontSize="Small" HorizontalTextAlignment="Center"
Margin="10,0"/>
                    </VerticalStackLayout>
                </Frame>
                <Frame Style="{StaticResource FrameOptions}" Padding="0"
HeightRequest="{OnPlatform Android=150, iOS=140}"
                WidthRequest="{OnPlatform Android=120, iOS=140}">
                    <VerticalStackLayout Spacing="0" VerticalOptions="Center">
                        <Image Source="menudisputas.png"/>
                        <Label Text="Disputas" FontSize="Small"
HorizontalTextAlignment="Center" Margin="10,0"/>
                    </VerticalStackLayout>
                </Frame>
            </HorizontalStackLayout>
        </ScrollView>

<HorizontalStackLayout Margin="10,100">
    <Image Source="exit.svg" WidthRequest="50"/>
    <Label Text="Sair" Margin="10,0,0,0" VerticalTextAlignment="Center" />
</HorizontalStackLayout>
```



O layout deverá ficar conforme abaixo. Execute o aplicativo para testar

```
</FlexLayout>
</Grid>
</Frame>

<Label Text="Este App oferece recursos para" FontAttributes="Bold" FontSize="Medium"/>

<ScrollView Orientation="Horizontal">
    <HorizontalStackLayout>
        <Frame Style="{StaticResource FrameOptions}" Padding="0" HeightRequest="{OnPlatform Android=150, iOS=140}"
            WidthRequest="{OnPlatform Android=120, iOS=140}">
            <VerticalStackLayout Spacing="0" VerticalOptions="Center">
                <Image Source="menupersonagens.svg"/>
                <Label Text="Personagens" FontSize="Small" HorizontalTextAlignment="Center" Margin="10,0"/>
            </VerticalStackLayout>
        </Frame>

        <Frame Style="{StaticResource FrameOptions}" Padding="0" HeightRequest="{OnPlatform Android=150, iOS=140}"
            WidthRequest="{OnPlatform Android=120, iOS=140}">
            <VerticalStackLayout Spacing="30" VerticalOptions="Center">
                <Image Source="menuarmas.png"/>
                <Label Text="Armas" FontSize="Small" HorizontalTextAlignment="Center" Margin="10,0"/>
            </VerticalStackLayout>
        </Frame>

        <Frame Style="{StaticResource FrameOptions}" Padding="0" HeightRequest="{OnPlatform Android=150, iOS=140}"
            WidthRequest="{OnPlatform Android=120, iOS=140}">
            <VerticalStackLayout Spacing="0" VerticalOptions="Center">
                <Image Source="menudisputas.png"/>
                <Label Text="Disputas" FontSize="Small" HorizontalTextAlignment="Center" Margin="10,0"/>
            </VerticalStackLayout>
        </Frame>
    </HorizontalStackLayout>
</ScrollView>

<HorizontalStackLayout Margin="10,100">
    <Image Source="exit.svg" WidthRequest="50"/>
    <Label Text="Sair" Margin="10,0,0" VerticalTextAlignment="Center" />
</HorizontalStackLayout>
</ContentPage.Content>
</ContentPage>
```



Recomendações de estudos para Criação de layout, estilos e cores – Canal dotNET

Documentação: <https://learn.microsoft.com/en-us/dotnet/maui/user-interface/layouts/?view=net-maui-7.0>
Parte I https://www.youtube.com/watch?v=x_sNPEwS3kA
Parte II <https://www.youtube.com/watch?v=k3oediCzZvs>



Mapas

1. Acesse o endereço <https://console.developers.google.com> e faça autenticação com uma conta google.
Clique para concordar com os termos de serviços, caso seja solicitado.
2. No menu superior esquerdo, clique em “Selecione um projeto” e na janela que se abrirá, clique em “Novo Projeto”

Novo projeto

⚠ Voce tem 19 projects restantes na sua cota. Solicite um aumento ou exclua projetos. [Saiba mais](#)

[MANAGE QUOTAS](#)

Nome do projeto * [?](#)

ID do projeto: mauimaps-387418. Não é possível alterá-lo depois. [EDITAR](#)

Local * [PROCURAR](#)

Pasta ou organização pai

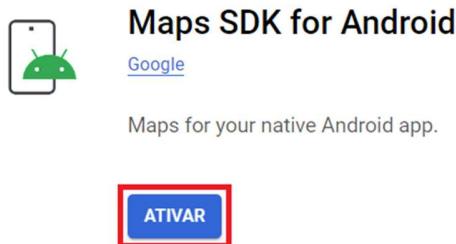
CRIAR [CANCELAR](#)

3. Clique em “APIs e serviços” → “API e serviços ativados” e depois em “Ativar APIs e serviços”

1 API APIs e serviços	2 APIs e serviços										
APIs e serviços ativados: <ul style="list-style-type: none">BibliotecaCredenciaisTela de permissão OAuthVerificação de domínioContratos de uso de página	+ ATIVAR APIs E SERVIÇOS <table border="1"><thead><tr><th>Tráfego</th><th>...</th></tr></thead><tbody><tr><td>1/s</td><td></td></tr><tr><td>0,6/s</td><td></td></tr><tr><td>0,6/s</td><td></td></tr><tr><td>No data is available for the selected time frame.</td><td>0,4/s</td></tr></tbody></table>	Tráfego	...	1/s		0,6/s		0,6/s		No data is available for the selected time frame.	0,4/s
Tráfego	...										
1/s											
0,6/s											
0,6/s											
No data is available for the selected time frame.	0,4/s										



4. Em mapas, selecione “Maps SDK for Android” e depois clique na opção “Ativar”



VISÃO GERAL DOCUMENTAÇÃO SUPORTE

5. Volte para a Home do Google Cloud Platform e com seu projeto selecionado, clique em Credenciais → Criar credenciais → Chave de API. Copie esta chave para um lugar seguro, pois usaremos ela na Solution do Visual Studio.

The screenshot shows the Google Cloud Platform dashboard. The top navigation bar has 'Google Cloud' and a search bar. The left sidebar has 'Métrica', 'Cotas', 'Credenciais' (highlighted with a red box), 'Biblioteca', and 'Exibições de mapa'. The main area shows 'APIs e serviços' with 'MauiMaps' selected (highlighted with a red box). Under 'Credenciais', there is a sub-section 'Chave de API criada' (highlighted with a red box) containing a key value: 'AIzaSyCwX...'. Other sections include 'IDs do cliente' and 'Contas de serviço'.

- Existem maneiras de deixar sua chave mais segura, para que ela não seja usada por robôs ou não seja clonada, para isso, deve ser feita uma vinculação da chave a uma única solution do Visual Studio. Isso será abordado mais à frente.



6. Abra o arquivo **AndroidManifest.xml** (pasta Platforms/Android/Resources) para adicionar as permissões : **ACCESS_NETWORK_STATE**, **ACCESS_COARSE_LOCATION**, **ACCESS_FINE_LOCATION**, **ACCESS_MOCK_LOCATION**, conforme abaixo

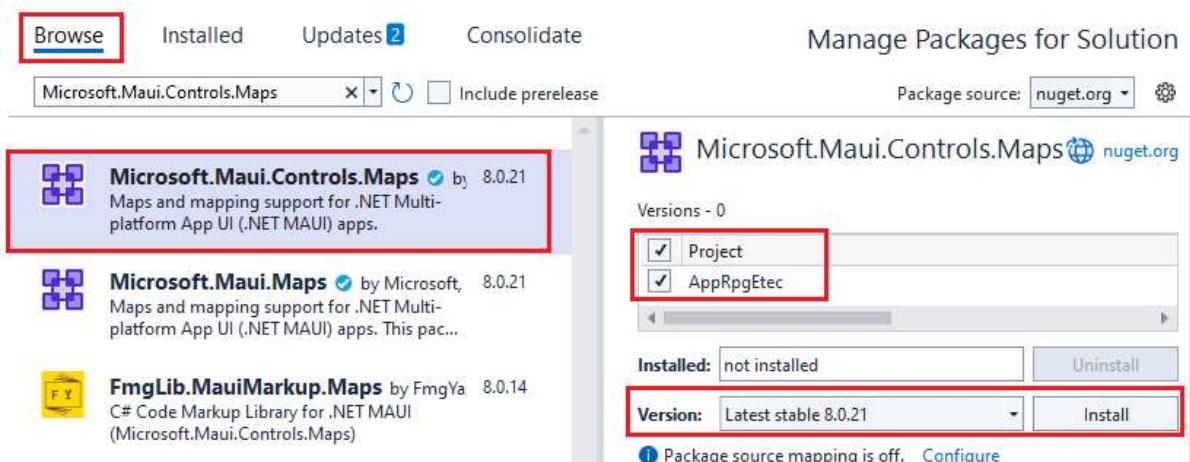
Required permissions

- ACCESS_BLOBS_ACROSS_USERS
- ACCESS_CHECKIN_PROPERTIES
- ACCESS_COARSE_LOCATION
- ACCESS_FINE_LOCATION
- ACCESS_LOCATION_EXTRA_COMMANDS
- ACCESS_MEDIA_LOCATION
- ACCESS_MOCK_LOCATION
- ACCESS_NETWORK_STATE

7. Clique com o direito neste mesmo arquivo e escolha → Open with → XML (Text) Editor. Iremos incluir as informações da chave da API. portanto abra um espaço na tag application e codifique conforme abaixo.

```
<application android:allowBackup="true"  
            android:icon="@mipmap/has_logo_transparente"  
            android:roundIcon="@mipmap/has_logo_transparente_round"  
            android:supportsRtl="true">  
  
    <meta-data android:name="com.google.android.geo.API_KEY"  
              android:value="CHAVE_DA_SUA_API_GOOGLE_MAPS">  
    </meta-data>  
  
</application>
```

8. Clique com o botão direito na Solution e em “Manage Nuget Packages for Solution”. Na aba Browser pesquise por Microsoft.Maui.ControlsMaps, selecione o projeto e depois clique em Install.





9. Abra o arquivo MauiProgram.cs para configurar o uso de mapas.

```
public static MauiApp CreateMauiApp()
{
    var builder = MauiApp.CreateBuilder();
    builder
        .UseMauiApp<App>()
        .ConfigureFonts(fonts =>
    {
        fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
        fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
    }).UseMauiMaps();
```

10. Clique com o direito na pasta Views/Usuarios e crie uma view (.NET MAUI) chamada **LocalizacaoView.xaml**. Faça a declaração do namespace para o uso de mapas e crie um controle do tipo Map conforme sinalizado:

```
<?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.Usuarios.LocalizacaoView"
    xmlns:maps="clr-namespace:Microsoft.Maui.Controls.Maps;assembly=Microsoft.Maui.Controls.Maps"
    Title="LocalizacaoView">

    <maps:Map x:Name="mapa" ItemsSource="{Binding MeuMapa}">
    </maps:Map>
</ContentPage>
```

11. Na view AppShell.xaml, realize a edição para que o menu de Usuários vire uma tab

```
<ShellContent Title="Usuários" Icon="menuusuarios.svg"
    ContentTemplate="{DataTemplate viewsUsuarios:LocalizacaoView}" />
```



- Execute o aplicativo para realizar os testes.



12. Na pasta ViewModels/Usuarios, crie uma classe chamada **LocalizacaoViewModel.cs** herdando de **BaseViewModels**. Realize o using a seguir:

```
| using Map = Microsoft.Maui.Controls.Maps.Map;
```

13. Faça a programações iniciais declarando um atributo e propriedade do tipo Map

```
private Map meuMapa;  
9 references  
public Map MeuMapa  
{  
    get => meuMapa;  
    set  
    {  
        if (value != null)  
        {  
            meuMapa = value;  
            OnPropertyChanged();  
        }  
    }  
}
```



14. Ainda na classe **LocalizacaoViewModel.cs**, crie um método para ativar a chamar a localização atual

```
public async void InicializarMapa()
{
    try
    {
        //Próxima etapa aqui
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage
            .DisplayAlert("Erro", ex.Message, "OK");
    }
}
```

15. Faça a programação abaixo dentro do bloco try

```
//Coordenadas geográficas da escola
Location location = new Location(-23.5200241d, -46.596498d);
Pin pinEtec = new Pin()
{
    Type = PinType.Place,
    Label = "Etec Horácio",
    Address = "Rua alcântara, 113, Vila Guilherme",
    Location = location
};

Map map = new Map();
MapSpan mapSpan = MapSpan
    .FromCenterAndRadius(location, Distance.FromKilometers(5));
map.Pins.Add(pinEtec);
map.MoveToRegion(mapSpan);

MeuMapa = map;
```



16. Abra a parte de código da ContentPage LocalizacaoView.xaml e defina a ligação com a classe ViewModel

```
public partial class LocalizacaoView : ContentPage
{
    LocalizacaoViewModel viewModel;
    1 reference
    public LocalizacaoView()
    {
        InitializeComponent();

        viewModel = new LocalizacaoViewModel();
        viewModel.InicializarMapa();

        BindingContext = viewModel;
    }
}
```

17. Abra o layout da view e altere o controle de mapa deixando como segue

```
<maps:Map x:Name="mapa" ItemsSource="{Binding MeuMapa}">
    <maps:Map.ItemTemplate>
        <DataTemplate>
            <maps:Pin Location="{Binding Location}"
                      Address="{Binding Address}"
                      Label="{Binding Label}" />
        </DataTemplate>
    </maps:Map.ItemTemplate>
</maps:Map>
```

- Execute para testar a visualização do Pin no mapa.

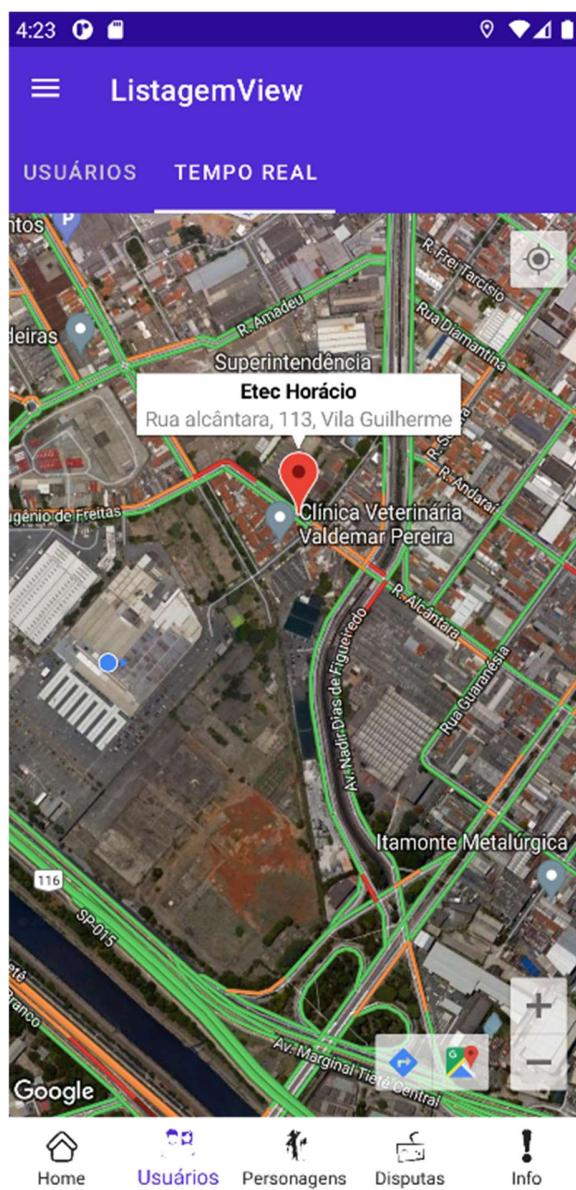




18. Acrescente as propriedades sinalizadas abaixo e execute para observar as mudanças.

```
<maps:Map x:Name="mapa" ItemsSource="{Binding MeuMapa}"  
MapType="Hybrid" IsShowingUser="true"  
IsZoomEnabled="True" IsTrafficEnabled="True">  
<maps:Map.ItemTemplate>  
<DataTemplate>
```

Abaixo está o resultado esperado





Geolocalização

1. Abra classe **Models/Usuarios.cs** e confirme a existência das propriedades referente a geolocalização

```
public double? Latitude { get; set; }  
2 references  
public double? Longitude { get; set; }
```

2. Abra a classe **UsuarioService** e crie um construtor que receba um token como parâmetro conforme a sinalização. Ao final deste procedimento a classe terá dois construtores.

```
public UsuarioService()  
{  
    _request = new Request();  
}
```

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

3. Abra a classe **UsuarioService** e adicione os métodos abaixo que vão atualizar a localização de um usuário, listar todos os usuários e buscar um usuário por id.

```
public async Task<int> PutAtualizarLocalizacaoAsync(Usuario u)  
{  
    string urlComplementar = "/AtualizarLocalizacao";  
    var result = await _request.PutAsync(apiUrlBase + urlComplementar, u, _token);  
    return result;  
}  
  
//using System.Collections.ObjectModel  
public async Task<ObservableCollection<Usuario>> GetUsuariosAsync()  
{  
    string urlComplementar = string.Format("{0}", "/GetAll");  
    ObservableCollection<Models.Usuario> listaUsuarios = await  
    _request.GetAsync<ObservableCollection<Models.Usuario>>(apiUrlBase + urlComplementar,  
_token);  
    return listaUsuarios;  
}
```



-
4. Abra a classe **UsuarioViewModel** e antes do método AutenticarUsuario, adicione dois atributos para controlar a operação de coleta de geolocalização

```
private CancellationTokenSource _cancelTokenSource;
private bool _isCheckingLocation;
```

1 reference

```
public async Task AutenticarUsuario() //Método para autenticar um usuário
{
```

5. Adicione a codificação para coletar a geolocalização do usuário e salvar a informação na API. A codificação deverá ficar antes da mensagem de boas-vindas. Será necessário deixar a localização do dispositivo ativada para os testes.

```
//Início da coleta de Geolocalização atual para Atualização na API
_isCheckingLocation = true;
_cancelTokenSource = new CancellationTokenSource();
GeolocationRequest request =
    new GeolocationRequest(GeolocationAccuracy.Medium, TimeSpan.FromSeconds(10));

Location location = await Geolocation
    .Default.GetLocationAsync(request, _cancelTokenSource.Token);

Usuario uLoc = new Usuario();
uLoc.Id = uAutenticado.Id;
uLoc.Latitude = location.Latitude;
uLoc.Longitude = location.Longitude;

UsuarioService uServiceLoc = new UsuarioService(uAutenticado.Token);
await uServiceLoc.PutAtualizarLocalizacaoAsync(uLoc);
//Fim da coleta de Geolocalização atual para Atualização na API

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

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



-
6. Abra a classe **LocalizacaoViewModel.cs** criando o atributo do tipo UsuarioService que consumirá a API e construtor que vai ler o token das Preferences passando para a classe de serviço.

```
public class LocalizacaoViewModel : BaseViewModel
{
    private UsuarioService uService;
    1 reference
    public LocalizacaoViewModel()
    {
        string token = Preferences.Get("UsuarioToken", string.Empty);
        uService = new UsuarioService(token);
    }
}
```



-
7. Ainda na classe **LocalizacaoViewModel**, crie um método que busque os usuários na API e apresente no mapa

```
public async void ExibirUsuariosNoMapa()
{
    try
    {
        //using AppRpgEtec.Models
        ObservableCollection<Usuario> ocUsuarios = await uService.GetUsuariosAsync();
        List<Usuario> listaUsuarios = new List<Usuario>(ocUsuarios);
        Map map = new Map();

        foreach (Usuario u in listaUsuarios)
        {
            if (u.Latitude != null && u.Longitude != null)
            {
                double latitude = (double)u.Latitude;
                double longitude = (double)u.Longitude;
                Location location = new Location(latitude, longitude);

                Pin pinAtual = new Pin()
                {
                    Type = PinType.Place,
                    Label = u.Username,
                    Address = $"E-mail: {u.Email}",
                    Location = location
                };
                map.Pins.Add(pinAtual);
            }
        }
        MeuMapa = map;
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage.DisplayAlert("Erro", ex.Message, "OK");
    }
}
```

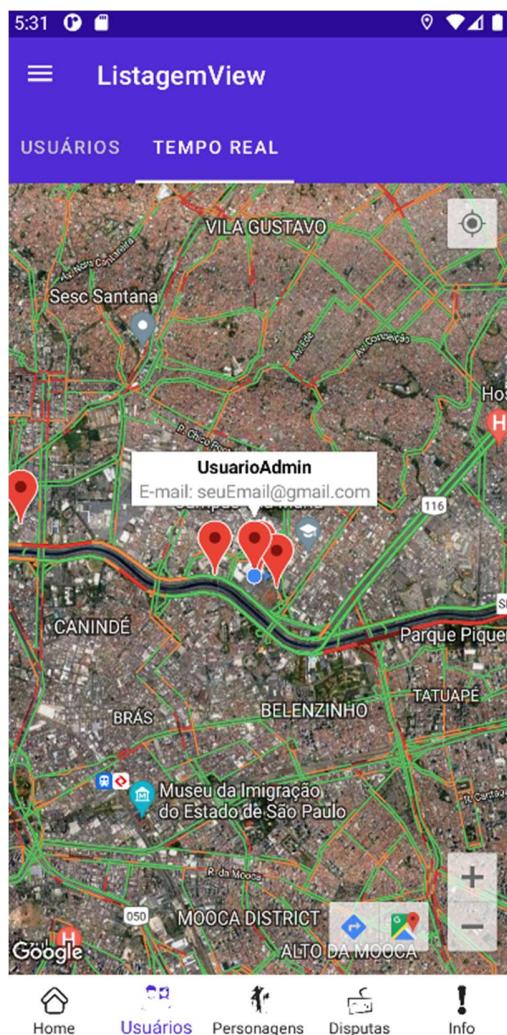


8. Faça a chamada do método no construtor da view **LocalizacaoView.xaml.cs** no lugar do método que inicializava o mapa.

```
public LocalizacaoView()
{
    InitializeComponent();

    viewModel = new LocalizacaoViewModel();
    BindingContext = viewModel;
    //viewModel.InicializarMapa();
    viewModel.ExibirUsuariosNoMapa();
}
```

- Execute o aplicativo para realizar os testes





Referências:

[Microsoft Learning](#)

Mapas:

<https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/map>

Geolocalização:

<https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/device/geolocation?tabs=android>

Geocoding - Transcrição de endereço em pin ou de pin em endereço:

<https://learn.microsoft.com/en-us/dotnet/maui/platform-integration/device/geocoding?tabs=android>

Refefrências – Xamarin (Alguns componentes não funcionarão no MAUI)

https://www.youtube.com/watch?v=T9g_fkm7mhA

Plugin Permission:

<https://social.msdn.microsoft.com/Forums/en-US/8294f783-4182-4e3e-bff8-c80610331c81/need-a-fix-on-current-location-access-on-xamarinforms?forum=xamarinforms>

Google Maps:

<https://bertuzzi.medium.com/o-x-do-xamarin-forms-mapas-mas-com-google-maps-d-e9b57071b4ec>

<https://julianocustodio.com.br/2017/08/09/gps-xamarin-forms/>

Mapas com MVVM:

https://medium.com/@pedro_jesus/usando-mapas-em-xamarin-forms-com-mvvm-7f4cab769725

<https://stackoverflow.com/questions/28098020/bind-to-xamarin-forms-maps-map-from-viewmodel>

Geolocator:

<https://stackoverflow.com/questions/36630509/how-to-get-current-location-or-move-to-current-location-in-xamarin-forms-map>

<http://rsamorim.azurewebsites.net/2018/03/01/xamarin-forms-capturando-mudanca-de-localizacao/>

Aplicação com exemplos do Plugin de Mapas:

<https://github.com/amay077/Xamarin.Forms.GoogleMaps/tree/master/XFGoogleMapSample>

Aplicação clone do Uber:

<https://www.xamboy.com/2019/07/03/introducing-xuber-uber-clone-using-xamarin-forms/>



Criação de chave de API com restrição por aplicativo

Abra o prompt de comando do Windows e navegue até a pasta abaixo através do comando cd

C:\Program Files\Android\jdk\microsoft_dist_openjdk_1.8.0.25\bin → 1.8.0.25 é a versão do exemplo, na sua máquina pode ser outra.

```
>cd C:\Program Files\Android\jdk\microsoft_dist_openjdk_1.8.0.25\bin
```

Execute o comando

```
keytool.exe -list -v -keystore "C:\Users\[USERNAME]\AppData\Local\Xamarin\Mono for Android\debug.keystore"  
-alias androiddebugkey -storepass android -keypass android
```

```
keytool.exe -list -v -keystore "C:\Users\Luiz\AppData\Local\Xamarin\Mono for Android\debug.keystore" -alias  
androiddebugkey -storepass android -keypass android
```

- Username → nome do seu usuário

Copie o código gerado na linha SHA1 para um bloco de notas. Usaremos o mesmo mais tarde

```
Alias name: androiddebugkey  
Creation date: 03/08/2019  
Entry type: PrivateKeyEntry  
Certificate chain length: 1  
Certificate[1]:  
Owner: CN=Android Debug, O=Android, C=US  
Issuer: CN=Android Debug, O=Android, C=US  
Serial number: 4bccd8b9  
Valid from: Sat Aug 03 17:04:08 BRT 2019 until: Mon Jul 26 17:04:08 BRT 2049  
Certificate fingerprints:  
    MD5: [REDACTED]  
    SHA1: [REDACTED]  
    SHA256:  
Signature algorithm name: SHA256withRSA  
Subject Public Key Algorithm: 2048-bit RSA key  
Version: 3
```



Procure a chave de API anteriormente criada e selecione a opção sinalizada abaixo

Chave de API criada

Transfira esta chave com o parâmetro **key=API_KEY** para usá-la no seu aplicativo.

Sua chave de API

⚠ Restrinja sua chave para evitar uso não autorizado na produção.

[FECHAR](#) [RESTRINGIR CHAVE](#)

Nome *

XamarinMapsDemoKey

Restrições de chave

⚠ Esta chave não tem restrições. As restrições ajudam a evitar o uso não autorizado e o roubo de cotas. [Saiba mais](#)

Restrições do aplicativo

Uma restrição de aplicativo controla quais sites, endereços IP ou aplicativos podem usar sua chave de API. Você pode definir apenas uma restrição de aplicativo por chave.

- Nenhuma
- Referenciadores de HTTP (sites da Web)
- Endereços IP (servidores da Web, cron jobs etc.)
- Apps para Android
- Apps para iOS

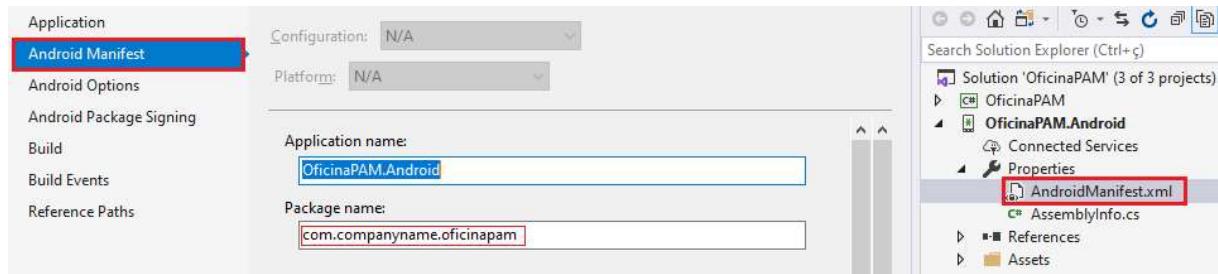
Restringir o uso de seus apps para Android

Adicione o nome do pacote e a impressão digital do certificado de assinatura SHA-1 para restringir o uso dos seus aplicativos para Android

[ADICIONAR UM ITEM](#)

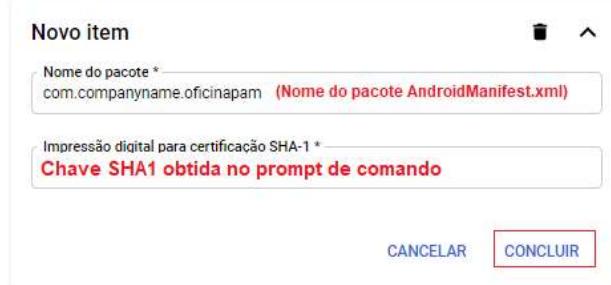


Vá até o arquivo android.manifest da solution e copie o nome do pacote da aplicação



Restringir o uso de seus apps para Android

Adicione o nome do pacote e a impressão digital do certificado de assinatura SHA-1 para restringir o uso dos seus aplicativos para Android.



Observação: pode levar até cinco minutos para que as configurações sejam aplicadas

SALVAR CANCELAR



Conta de Armazenamento do Azure (Blob Storage)

- Entre no endereço do Azure for Students com seu e-mail institucional <https://azure.microsoft.com/pt-br/free/students>
- Navegue até o menu Contas de Armazenamento ou Blob Storage para iniciar os procedimentos selecionando os itens conforme abaixo e após isso clique em avançar. Caso você não tenha um grupo de recursos, poderá criar durante esta etapa. No exemplo foi criado o grupo EtecHAS. No lugar de etecstorage dê outro nome pois o endereço de armazenamento usará esta palavra. Ex: <https://etecstorage.blob.core.windows.net>

The screenshot shows the Azure Storage Account creation process. The top navigation bar includes 'Microsoft Azure', a search bar, and a 'Pesquisar recursos' button. Below the navigation, the breadcrumb path is 'Página inicial > Contas de armazenamento > Criar uma conta de armazenamento'. The main section has tabs for 'Básico', 'Avançado', 'Rede', 'Proteção de dados', 'Criptografia', 'Marcas', and 'Examinar'. The 'Básico' tab is selected. A descriptive text explains that Azure Storage is a managed service for storing and managing data at scale. It mentions Blobs, Files, Tables, and Data Lake Storage Gen2, and notes that costs depend on usage and selected options. The 'Detalhes do projeto' section asks for an 'Assinatura' (Azure para Estudantes Starter) and a 'Grupo de recursos' (EtecHAS). The 'Detalhes da instância' section requires a storage account name ('etecstorage'), region ('(South America) Brazil South'), and performance level ('Standard v2'). Redundancy is set to 'LRS (armazenamento com redundância local)'. At the bottom are buttons for 'Examinar', '< Anterior', and 'Avançar: Avançado >'.



3. Na aba avançado selecione os itens conforme abaixo e clique em prosseguir.

Microsoft Azure Pesquisar recursos

Página inicial > Contas de armazenamento >

Criar uma conta de armazenamento

Básico Avançado Rede Proteção de dados Criptografia Marcas Examinar

ⓘ Determinadas opções foram desabilitadas por padrão devido à combinação de desempenho, redundância e região da conta de armazenamento.

Segurança

Defina as configurações de segurança que afetam sua conta de armazenamento.

Exigir transferência segura para operações da API REST

Permitir a habilitação do acesso anônimo em contêineres individuais

Habilitar o acesso à chave de conta de armazenamento

Padrão para autorização do Microsoft Entra no portal do Azure

Versão mínima do TLS

Escopo permitido para operações de cópia (versão prévia)

Namespace hierárquico

O namespace hierárquico, complementado pelo ponto de extremidade do Data Lake Storage Gen2, permite a semântica de arquivos e diretórios, acelera as cargas de trabalho de análise de big data e habilita ACLS (listas de controle de acesso) [Saiba mais](#)

Habilitar namespace hierárquico

Protocolos de acesso

Os pontos de extremidade de Blob e Data Lake Gen2 são provisionados por padrão [Saiba mais](#)



4. Selecione os itens da aba rede e clique em avançar

Página inicial > Contas de armazenamento >

Criar uma conta de armazenamento ...

Conectividade de rede

Você pode se conectar à conta de armazenamento de forma pública, com endereços IP ou pontos de extremidade de serviço públicos, ou de forma privada, usando um ponto de extremidade privado.

Acesso à rede *

- Permitir acesso público de todas as redes
 Habilitar o acesso público de redes virtuais e endereços IP selecionados
 Desabilitar o acesso público e usar o acesso privado
- i** Habilitar o acesso público de todas as redes pode disponibilizar esse recurso publicamente. A menos que o acesso público seja necessário, recomendamos usar um tipo de acesso mais restrito. [Saiba mais](#)

Ponto de extremidade privado

Crie um ponto de extremidade privado para permitir uma conexão privada com esse recurso. É possível criar conexões de ponto de extremidade privado adicionais na conta de armazenamento ou no centro de link privado.

+ Adicionar ponto de extremidade privado

Nome	Assinatura	Grupo de r...	Região	Tipo de su...	Sub-rede	Zona DNS ...
<small>Clique em Adicionar para criar um ponto de extremidade privado</small>						

Roteamento de rede

Determine como encaminhar o tráfego conforme ele sai da origem em direção ao ponto de extremidade do Azure. O roteamento de rede da Microsoft é recomendado para a maioria dos clientes.

Preferência de roteamento * ①

- Roteamento de rede da Microsoft
 Roteamento da Internet

5. Na aba Proteção de dados nada será alterado.

6. A aba criptografia ficará como abaixo



Página inicial > Contas de armazenamento >

Criar uma conta de armazenamento ...

Básico Avançado Rede Proteção de dados Criptografia Marcas Examinar

- Tipo de criptografia * ①
 MMK (chaves gerenciadas pela Microsoft)
 CMK (chaves gerenciadas pelo cliente)

- Habilitar suporte para chaves gerenciadas pelo cliente ①
 Somente blobs e arquivos
 Todos os tipos de serviço (blobs, arquivos, tabelas e filas)

⚠ Esta opção não poderá ser alterada após a criação dessa conta de armazenamento.

Habilitar a criptografia de infraestrutura ①



7. Nada será alterado no item Marcas. No item examinação será apresentado um resumo dos itens configurados. Após isso clique em avançar

The screenshot shows the Microsoft Azure Storage Account creation interface. The top navigation bar includes 'Microsoft Azure', 'Página inicial > Contas de armazenamento >', 'Criar uma conta de armazenamento', and a '...'. Below the navigation is a breadcrumb trail: 'Básico' (selected), 'Avançado', 'Rede', 'Proteção de dados', 'Criptografia', 'Marcas', and 'Examinar'. The 'Marcas' tab is underlined. The configuration section for 'Marcas' lists the following items:

Assinatura	Azure para Estudantes
Grupo de Recursos	RG_Test
Localização	eastus
Nome da conta de armazenamento	etecstorage
Modelo de implantação	Resource manager
Desempenho	Standard
Replicação	LRS (armazenamento com redundância local)

Below the 'Marcas' section are sections for 'Avançado', 'Rede', and 'Segurança'. At the bottom are buttons for 'Criar', '< Anterior' (disabled), 'Avançar >', and 'Baixar um modelo para automação'.

8. Após esta etapa será exibida a mensagem sobre a implantação do recurso.

The screenshot shows the Microsoft Azure Storage Account implementation status page for the resource 'etecstorage_1702589143226'. The top navigation bar includes 'Microsoft Azure', 'Página inicial >', and a search bar 'Pesquisar recursos, serviços e documentos (G+ /)'. The main title is 'etecstorage_1702589143226 | Visão Geral'. Below the title, there's a message 'A implantação foi concluída' with a checkmark icon. The implementation details are listed as follows:

Nome da implantação:	etecstorage_1702589143226
Assinatura:	Azure para Estudantes
Grupo de recursos:	RG_Test

Implementation details include: Hora de início: 14/12/2023, 18:26:33; ID de Correlação: a66463f2-a1e7-470b-9a98-7e086a1e5322. At the bottom are buttons for 'Ir para o recurso' and 'Detalhes de implantação'.

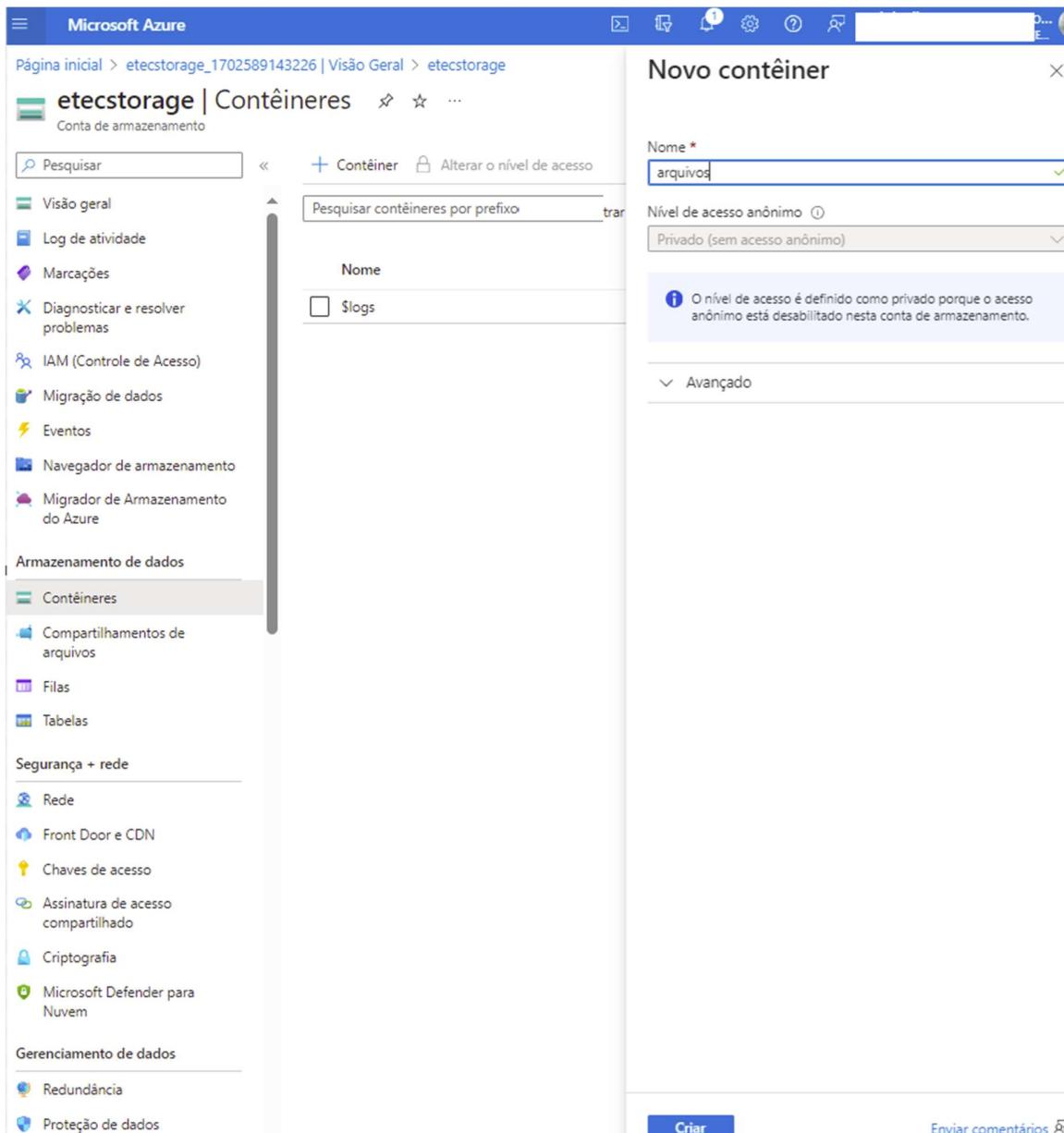


9. Na visão geral do recurso será possível visualizar o menu containers. Os containers funcionarão com pastas.

The screenshot shows the Microsoft Azure Storage blade for the 'etecstorage' account. The left sidebar contains navigation links for 'Visão geral', 'Log de atividade', 'Marcações', 'Diagnosticar e resolver problemas', 'IAM (Controle de Acesso)', 'Migração de dados', 'Eventos', 'Navegador de armazenamento', and 'Migrador de Armazenamento do Azure'. Under 'Armazenamento de dados', there are sections for 'Contêineres', 'Compartilhamentos de arquivos', 'Filas', and 'Tabelas'. Other sections include 'Segurança + rede', 'Gerenciamento de dados', 'Redundância', and 'Proteção de dados'. The main content area displays general information about the storage account, including its resource group (RG Teste), location (eastus), and access signature (Azure para Estudantes). It also lists blob service settings such as Namespace hierárquico (Disabled), Camada de acesso padrão (Hot), and Exclusão reversível do blob (Enabled (7 days)). The 'Serviço Blob' section is expanded, showing detailed configurations for blob storage. The 'Serviço de arquivo' and 'Serviço Fila' sections are also partially visible. At the bottom, there are tabs for 'Propriedades', 'Monitoramento', 'Funcionalidades (7)', 'Recomendações (0)', 'Tutorial', and 'Ferramentas + SDKs'.



10. Crie um novo container com o nome arquivos.



The screenshot shows the Microsoft Azure Storage interface. On the left, there's a sidebar with navigation links like 'Visão geral', 'Log de atividade', 'Marcções', etc. The main area shows a list of containers under 'etecstorage' with one item: '\$logs'. A modal window titled 'Novo contêiner' is open on the right, prompting for a new container name. The input field contains 'arquivos'. Below it, the access level is set to 'Privado (sem acesso anônimo)'. A note indicates that anonymous access is disabled. At the bottom of the modal are 'Criar' and 'Enviar comentários' buttons.



11. Volte para a conta de armazenamento criada e no menu da esquerda, role a tela até a seção Rede + Armazenamento, mostre e copie os dados presentes em cadeia de conexão. Usaremos estes dados mais a frente

Página inicial / etecstorage

etecstorage | Chaves de acesso

Conta de armazenamento

Pesquisar

Definir lembrete de rotação Atualizar Enviar comentários

Segurança + rede

- Rede
- Front Door e CDN
- Chaves de acesso**
- Assinatura de acesso compartilhado
- Criptografia
- Microsoft Defender para Nuvem

Gerenciamento de dados

- Tarefas de armazenamento (versão prévia)
- Redundância
- Proteção de dados
- Replicação de objeto
- Inventário de blobs
- Site estático
- Gerenciamento do ciclo de vida
- Pesquisa de IA do Azure

As chaves de acesso autenticam as solicitações dos seus aplicativos para esta conta de armazenamento. Mantenha suas chaves em um local seguro como o Azure Key Vault e substitua-as frequentemente por novas chaves. As duas chaves permitem que você substitua uma enquanto ainda usa a outra.

Lembre-se de atualizar as chaves com quaisquer recursos e aplicativos do Azure que utilizam essa conta de armazenamento. Saiba mais sobre como gerenciar chaves de acesso de conta de armazenamento ↗

Nome da conta de armazenamento: etecstorage

key1 Chave de Giro

Última vez girado: 14/12/2023 (150 dias atrás)

Chave: ... Mostrar

Cadeia de conexão

key2 Chave de Giro

Última vez girado: 14/12/2023 (150 dias atrás)

Chave: ... Mostrar

Cadeia de conexão

key2 Chave de Giro

Última vez girado: 14/12/2023 (150 dias atrás)

Chave: ... Mostrar

- Agora poderemos fazer o uso das ferramentas de imagens



Aula 15 - Galeria e câmera do App para salvamento de imagens na API

1. Abra a classe Services/Usuarios/**UsuarioService** e adicione um método para atualizar a foto do usuário para opção de armazenamento da imagem na API.

```
public async Task<int> PutFotoUsuarioAsync(Usuario u)
{
    string urlComplementar = "/AtualizarFoto";
    var result = await _request.PutAsync(apiUrlBase + urlComplementar, u, _token);
    return result;
}
```

- Insira também o método responsável por fazer a consulta de um usuário.

```
public async Task<Usuario> GetUsuarioAsync(int usuarioId)
{
    string urlComplementar = string.Format("/{0}", usuarioId);
    var usuario = await
        _request.GetAsync<Models.Usuario>(apiUrlBase + urlComplementar, _token);
    return usuario;
}
```

2. Crie uma classe dentro da pasta **ViewModels/Usuarios** chamada **ImagenUsuarioViewModel.cs**. Herdando de **BaseViewModel** (1). Será necessário o using para AppRpgEtec.Services.Usuarios. Em (2) faremos a declaração da classe de serviço para uso da API. Em (3) colocaremos os dados de armazenamento do azure e em (4) será o construtor da classe.

```
REFERENCE
public class ImagemUsuarioViewModel : BaseViewModel 1
{
    2 private UsuarioService uService;
    3 private static string conexaoAzureStorage = "COLE a chave de acesso da conta de armazenamento";
    4 private static string container = "arquivos"; //nome do container criado
    0 references
    public ImagemUsuarioViewModel()
    {
        string token = Preferences.Get("UsuarioToken", string.Empty);
        uService = new UsuarioService(token);
    }

    //Próximas etapas aqui
}
```

3. Adicione o atributo/propriedade **fonteImagem** para armazenar a imagem para exibir na view e **Foto** para atribuir ao futuro objeto do tipo Usuário.

```
private ImageSource fonteImagem;
1 reference
public ImageSource FonteImagem
{
    get { return fonteImagem; }
    set
    {
        fonteImagem = value;
        OnPropertyChanged();
    }
}
```

```
private byte[] foto; //CTRL + R,E
1 reference
public byte[] Foto
{
    get => foto;
    set
    {
        foto = value;
        OnPropertyChanged();
    }
}
```



-
4. Crie o método para fotografar com a estrutura do try/catch. Use o atalho try + TAB + TAB

```
public async void Fotografar()
{
    try
    {
        //Codificação aqui
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage
            .DisplayAlert("Ops", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
```

5. Faça a programação dentro do bloco try do método criado no item anterior

```
//Verificação se o dispositivo suporta mídia como câmera e galeria.
if (MediaPicker.Default.IsCaptureSupported)
{
    //Chamada para a câmera do dispositivo. Fica aguardando usuário tirar foto.
    FileResult photo = await MediaPicker.Default.CapturePhotoAsync();

    if (photo != null)
    {
        using (Stream sourceStream = await photo.OpenReadAsync())//Leitura dos bytes da foto para Stream
        {
            using (MemoryStream ms = new MemoryStream())
            {
                await sourceStream.CopyToAsync(ms); //Conversão do Stream para MemoryStream (arquivo em memória)

                //Carregamento do array de bytes a partir da memória para a propriedade da ViewModel
                Foto = ms.ToArray();

                //Carregamento do controle que apresenta a imagem para a ViewModel
                FonteImagem = ImageSource.FromStream(() => new MemoryStream(ms.ToArray()));
            }
        }
    }
}
```



6. Abra o repositório do nuget e faça a instalação do pacote do pacote Azure.Storage.Blobs

The screenshot shows the NuGet package manager interface. At the top, there are tabs for 'Browse', 'Installed', 'Updates 2', and 'Consolidate'. Below these, a search bar contains 'Azure.Storage.Blobs' with a dropdown menu, a refresh icon, and checkboxes for 'Include prerelease' and 'Show only vulnerable'. To the right, it says 'Manage Packages for Solution' and 'Package source: nuget.org'. The main area displays three packages:

- Azure.Storage.Blobs** by Microsoft, 236M downloads, version 12.19.1. This client library enables working with the Microsoft Azure Storage Blob service for storing binary and text data.
- Microsoft.Azure.WebJobs.Core** by Microsoft, 313M downloads, version 3.0.39. This library simplifies the task of adding background processing to your Microsoft Azure Web Sites. The SDK uses...
- Azure.Storage.Blobs.Batch** by Microsoft, 3,28M downloads, version 12.16.1. This client library allows you to batch multiple Azure Blob Storage operations in a single request.

On the right side, there is a detailed view for the 'Azure.Storage.Blobs' package. It shows 'Versions - 0' with two checked checkboxes: 'Project' and 'AppRpgEtec'. Under 'Installed', it says 'not installed' with an 'Uninstall' button. A dropdown for 'Version' is set to 'Latest stable 12.19.1', and a large red box highlights the 'Install' button.

7. Crie o método que vai escrever a imagem no armazenamento do azure. Será necessário o using para AppRpgEtec.Models e de Azure.Storage.Blobs.

```
public async void SalvarImagenAzure()
{
    try
    {
        Usuario u = new Usuario();
        u.Foto = foto;
        u.Id = Preferences.Get("UsuarioId", 0);

        string fileName = $"{u.Id}.jpg";

        //define o BLOB no qual a imagem será armazenda
        var blobClient = new BlobClient(conexaoAzureStorage, container, fileName);

        if (blobClient.Exists())
            blobClient.Delete();

        using (var stream = new MemoryStream(u.Foto))
        {
            blobClient.Upload(stream);
        }

        await Application.Current.MainPage.DisplayAlert("Mensagem", "Dados salvos com sucesso!", "Ok");
        await App.Current.MainPage.Navigation.PopAsync();
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage
            .DisplayAlert("Ops", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
```



8. Declare os ICommands (1) e initialize-os no construtor (2), vinculando os mesmos aos métodos criados anteriormente. Será necessário o using de System.Windows.Input.

```
public ImagemUsuarioViewModel()
{
    string token = Preferences.Get("UsuarioToken", string.Empty);
    uService = new UsuarioService(token);

    2 FotografarCommand = new Command(Fotografar);
    SalvarImagenCommand = new Command(SalvarImagen);
}

1 reference
1  public ICommand FotografarCommand { get; }
1 reference
1  public ICommand SalvarImagenCommand { get; }
```

9. Na pasta Views/Usuarios, crie uma *Content Page* (.Net Maui) chamada **ImagenUsuarioView.xaml**. Insira as tags abaixo, dentro da tag ContentPage

```
<ScrollView>
    <VerticalStackLayout>

        </VerticalStackLayout>
    </ScrollView>
```

10. Insira o design abaixo dentro da tag VerticalStackLayout que foi inserido na etapa anterior

```
<ScrollView>
    <VerticalStackLayout HorizontalOptions="FillAndExpand"
VerticalOptions="Start">
        <Grid HorizontalOptions="Fill" Margin="5, 5, 0, 0" Padding="10, 10, 0, 0">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="33*" />
                <ColumnDefinition Width="34*" />
                <ColumnDefinition Width="33*" />
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>
            <Button Text="Câmera" HorizontalOptions="Center" Grid.Column="0"
Grid.Row="0" Command="{Binding FotografarCommand}"/>
            <Button Text="Álbum" HorizontalOptions="Center" Grid.Column="1"
Grid.Row="0" Command="{Binding AbrirGaleriaCommand}"/>
            <Button Text="Gravar" HorizontalOptions="Center" Grid.Column="2"
Grid.Row="0" Command="{Binding SalvarImagenCommand}"/>
        </Grid>
    </VerticalStackLayout>
</ScrollView>
<Image Source="{Binding FonteImagen}" Margin="10"/>
```



11. Na parte de Código da view iremos fazer a vinculação entre a *ViewModel* e a *View*. Necessário o using de AppRpgEtec.ViewModels.Usuarios

```
ImagenUsuarioViewModel viewModel;
0 references
public ImagemUsuarioView()
{
    InitializeComponent();

    viewModel = new ImagemUsuarioViewModel();
    Title = "Imagen do Usuário";
    BindingContext = viewModel;
}
```

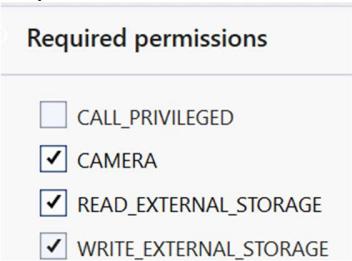
12. Abra a view **AppShell.xaml** e insira um item de menu para chamar a view *ImagenUsuarioView*

```
<Tab Title="Info" Route="info" Icon="info.svg">
    <ShellContent Title="Usuário" Route="usuario"
        ContentTemplate="{DataTemplate viewsUsuarios:ImagenUsuarioView}" />

    <ShellContent Title="Sobre" Route="sobre"
        ContentTemplate="{DataTemplate local:AboutView}" />

    <ShellContent Title="Descrição" Route="desc"
        ContentTemplate="{DataTemplate local:DescriptionView}" />
</Tab>
```

13. Abra o arquivo **AndroidManifest.xml** que fica na pasta *Platforms/Android* (pasta Properties) e adicione o código abaixo antes do fechamento da tag manifest, para as permissões sobre câmera, leitura e escrita de arquivos externos



14. Abra o arquivo do manifest como texto e antes do fechamento da tag manifest adicione as tags abaixo, referente a captura de imagem.

```
<queries>
    <intent>
        <action android:name="android.media.action.IMAGE_CAPTURE" />
    </intent>
</queries>
```

- Execute o aplicativo para testar o funcionamento da câmera e o salvamento da imagem na API.



15. Volte à viewModel **ImagenUsuarioViewModel.cs** e faça uma cópia do método fotografar e altere basicamente as partes sinalizadas abaixo. Como pode perceber vamos utilizar a mesma Biblioteca MediaPicker, alterando apenas o método que desta vez será para escolher uma imagem da galeria.

```
public async void AbrirGaleria()
{
    try
    {
        //Verificação se o dispositivo suporta câmera.
        if (MediaPicker.Default.IsCaptureSupported)
        {
            //Chamada para a galeria do dispositivo. Fica aguardando usuário escolher a foto da galeria.
            FileResult photo = await MediaPicker.Default.PickPhotoAsync();

            if (photo != null)
            {
                using (Stream sourceStream = await photo.OpenReadAsync())//Leitura dos bytes da foto para Stream
```

16. Declare abaixo do construtor um ICommand chamado AbrirGaleriaCommand (1) e faça a vinculação dele com o método AbrirGaleria na última linha do construtor (2)

```
public ImagenUsuarioViewModel()
{
    string token = Preferences.Get("UsuarioToken", string.Empty);
    uService = new UsuarioService(token);

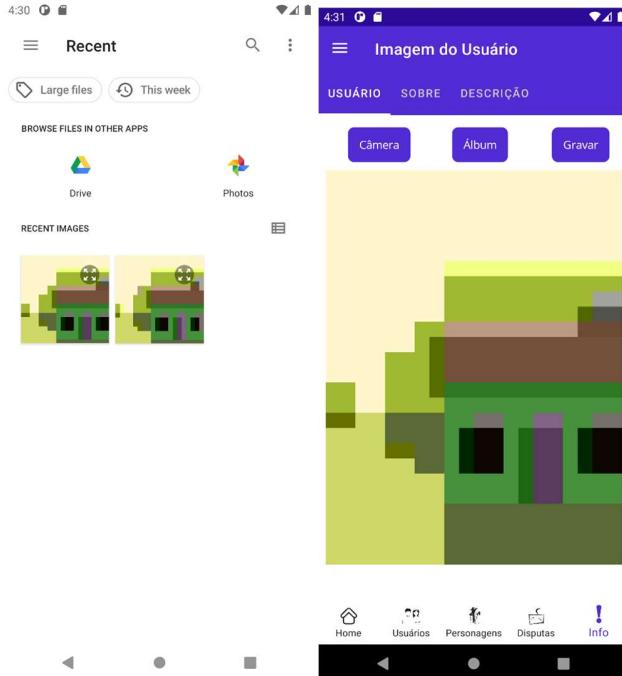
    2 FotografarCommand = new Command(Fotografar);
    SalvarImagenCommand = new Command(SalvarImagenAzure);
    AbrirGaleriaCommand = new Command(AbrirGaleria);
}
```

1 reference

```
1 public ICommand FotografarCommand { get; }
1 reference
public ICommand SalvarImagenCommand { get; }
1 reference
public ICommand AbrirGaleriaCommand { get; }
```



- Resultado esperados para acionamento da câmera e galeria



- Pasta de armazenamento do azure

Página inicial > etecstorage | Contêineres >

arquivos ...

Contêiner

Pesquisar

Visão geral

Diagnosticar e resolver problemas

IAM (Controle de Acesso)

Configurações

Tokens de acesso compartilhado

Política de acesso

Propriedades

Metadados

Carregar

Alterar o nível de acesso

Atualizar

Excluir

Alterar a camada

Adquirir concessão

...

Método de autenticação: Chave de acesso (Alternar para conta de usuário do Microsoft Entra)

Local: arquivos

Pesquisar blobs por prefixo (diferenciar maiúsculas de minúsculas)

Mostrar blobs excluídos

Adicionar o filtro

Nome	Modificado	Camada de acesso	S.	Tipo de blob	Tamanho
<input type="checkbox"/> 1.jpg	12/05/2024, 23:47:59	Principal (Inferidos)		Blob de blocos	142.12 KiB
<input type="checkbox"/> 6b7b9497-322e-4a88-884c-e758e38dbe0....	15/12/2023, 12:57:17	Principal (Inferidos)		Blob de blocos	308.96 KiB
<input type="checkbox"/> ac6ddbce-6233-4a1d-9cd5-45a6c219689d.p...	15/12/2023, 17:36:42	Principal (Inferidos)		Blob de blocos	133.14 KiB
<input type="checkbox"/> logo_horizontal_positivo.png	15/12/2023, 09:20:11	Principal (Inferidos)		Blob de blocos	308.96 KiB
<input type="checkbox"/> Lupa.png	15/12/2023, 09:18:39	Principal (Inferidos)		Blob de blocos	4.33 KiB



Apenas para ciência

- Caso a fossemos inserir a imagem na API teríamos um método pra ler as propriedades da viewModel, atribuir ao objeto do tipo Usuario para que a classe de serviço envie para a API as informações. Necessário using de AppRgpEtec.Models.

```
public async void SalvarImagen()
{
    try
    {
        Usuario u = new Usuario();
        u.Foto = foto;
        u.Id = Preferences.Get("UsuarioId", 0);

        if (await uService.PutFotoUsuarioAsync(u) != 0)
        {
            await Application.Current.MainPage.DisplayAlert("Mensagem", "Dados salvos com sucesso!", "Ok");
            await App.Current.MainPage.Navigation.PopAsync();
        }
        else { throw new Exception("Erro ao tentar atualizar imagem"); }
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage
            .DisplayAlert("Ops", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
```



Aula 12 - Carregando no App uma imagem Salva

- Crie uma pasta chamada **Converters** e dentro desta pasta, crie uma classe chamada **ByteArrayToImageSourceConverter**, implemente a interface sugerida e realize a programação no método *Convert* a seguir

```
public class ByteArrayToImageSourceConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, System.Globalization.CultureInfo culture)
    {
        ImageSource retSource = null;
        if (value != null)
        {
            byte[] imageAsBytes = (byte[])value;
            retSource = ImageSource.FromStream(() => new MemoryStream(imageAsBytes));
        }
        return retSource;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

- Exigirá o using *System.IO* e *System.Globalization*;
- Abra a view *ImagenUsuarioView*, insirindo o namespace que vai referenciar a classe recém-criada e o recurso que será usado

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="AppRpgEtec.Views.Uuarios.ImagenUsuarioView"
    xmlns:conv="clr-namespace:AppRpgEtec.Converters"
    Title="ImagenUsuarioView">
    <ScrollView>
```

- Adicione o recurso do conversor dentro da seção de recursos da contente page antes deste primeiro ScrollView que aparece no print anterior.

```
<ContentPage.Resources>
    <ResourceDictionary>
        <conv:ByteArrayToImageSourceConverter x:Key="ByteArrayToImage" />
    </ResourceDictionary>
</ContentPage.Resources>
```



4. Troque ou altere o objeto Image que existia pelo sinalizado abaixo, pois existindo um conversor, podemos fazer o binding da viewModel diretamente da propriedade Foto, já que em tempo de execução as informações do array de bytes da imagem serão transformadas em uma imagem visualizável.

```
</ScrollView>

<Image Source="{Binding FonteImagen}" Margin="10"/>

<Image WidthRequest="400" HeightRequest="400" Margin="20"
      Source="{Binding Foto, Converter={StaticResource ByteArrayToImage}}"/>

</VerticalStackLayout>
```

5. Abra a classe *ImagenUsuarioViewModel* e crie o método que vai buscar a foto do usuário através da classe de serviço.

```
public async void CarregarUsuarioAzure()
{
    try
    {
        int usuarioId = Preferences.Get("UserId", 0);
        string filename = $"{usuarioId}.jpg";
        var blobClient = new BlobClient(conexaoAzureStorage, container, filename);

        if (blobClient.Exists())
        {
            Byte[] fileBytes;

            using (MemoryStream ms = new MemoryStream())
            {
                blobClient.OpenRead().CopyTo(ms);
                fileBytes = ms.ToArray();
            }
            Foto = fileBytes;
        }
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage
            .DisplayAlert("Ops", ex.Message + " Detalhes: " + ex.InnerException, "Ok");
    }
}
```

6. Faça a chamada do método no construtor e execute o app para testar.

```
public ImagemUsuarioViewModel()
{
    string token = Preferences.Get("UserToken", string.Empty);
    uService = new UsuarioService(token);

    FotografarCommand = new Command(Fotografar);
    SalvarImagenCommand = new Command(SalvarImagenAzure);
    AbrirGaleriaCommand = new Command(AbrirGaleria);

    CarregarUsuarioAzure();
}
```



- Resultado esperado: View carregando exibindo a imagem salva



7. Clique com o botão direito na pasta ViewModels e crie uma classe chamada **AppShellViewModel.cs**, herdando da classe BaseViewModel.

```
public class AppShellViewModel : BaseViewModel
```

8. Insira o atributo de serviço e o construtor conforme o código abaixo dentro da classe para que tenhamos todos os procedimentos para poder buscar os dados do usuário através da API e principalmente a foto cadastrada.

```
private UsuarioService uService;
public AppShellViewModel()
{
    string token = Preferences.Get("UsuarioToken", string.Empty);
    uService = new UsuarioService(token);

    CarregarUsuarioAzure();
}
```



-
9. Crie um atributo/propriedade para a foto e o método para trazer informações do usuário da API.

```
private byte[] foto;
public byte[] Foto
{
    get => foto;
    set
    {
        foto = value;
        OnPropertyChanged();
    }
}
```

10. Use o mesmo método de carregar de carregamento através do armazenamento do Azure

```
public async void CarregarUsuarioAzure()
{
    try
    {
        int usuarioId = Preferences.Get("UsuarioId", 0);
        string filename = $"{usuarioId}.jpg";
        var blobClient = new BlobClient(conexaoAzureStorage, container, filename);

        if (blobClient.Exists())
        {
            Byte[] fileBytes;

            using (MemoryStream ms = new MemoryStream())
            {
                blobClient.OpenRead().CopyTo(ms);
                fileBytes = ms.ToArray();
            }

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



11. Abra a parte de código da view AppShell (AppShell.xaml.cs), declarando a viewModel recém-criada (1) e atribuindo a viewModel como contexto da View

```
public partial class AppShell : Shell
{
    1 AppShellViewModel viewModel;
    1 reference
    public AppShell()
    {
        InitializeComponent();

        2 viewModel = new AppShellViewModel();
        BindingContext = viewModel;

        string login = Preferences.Get("UsuarioUsername", string.Empty);
        lblLogin.Text = $"Login: {login}";
    }
}
```

12. Abra o design da view AppShell e faça referência a pasta das classes de conversão (1), declare o método de conversão de array de bytes para imagem apelidando através da propriedade Key (2) e altere o objeto Image para que ele use a conversão para exibir a imagem através do Binding.

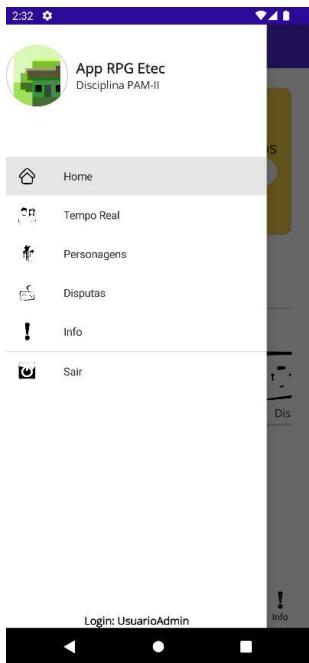
```
1 xmlns:conv="clr-namespace:AppRpgEtec.Converters"
  xmlns:local="clr-namespace:AppRpgEtec"

2 <Shell.Resources>
    <ResourceDictionary>
        <conv:ByteArrayToImageSourceConverter x:Key="ByteArrayToImage" />
    </ResourceDictionary>
</Shell.Resources>

<Shell.FlyoutHeaderTemplate>
    <DataTemplate>
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="{OnPlatform Android=130, iOS=200}"/>
                <RowDefinition Height="*"/>
                <RowDefinition Height="40"/>
            </Grid.RowDefinitions>
            <FlexLayout Grid.Row="0" Direction="Row" AlignItems="Center" >
                <Frame Padding="-20" CornerRadius="40" HeightRequest="80" WidthRequest="80" >
                    3 <Image Source="{Binding Foto, Converter={StaticResource ByteArrayToImage}}"/>
                </Frame>
            </FlexLayout>
        </Grid>
    </DataTemplate>
</Shell.FlyoutHeaderTemplate>
```



-
13. Execute o app e confirme que a imagem que aparecerá no menu será a que foi enviada através do dispositivo



Referências: <https://learn.microsoft.com/pt-br/dotnet/maui/platform-integration/device-media/picker?view=net-maui-8.0&tabs=android>

Salvamento via API/MVC: <https://renicius-pagotto.medium.com/upload-de-arquivos-com-o-azure-blob-storage-76d86710ae3>



Apenas para ciência

14. Para carregar a imagem através de uma API se ela estiver guardada no banco de dados, poderíamos usar o método a seguir:

```
public async void CarregarUsuario()
{
    try
    {
        int usuarioId = Preferences.Get("UsuarioId", 0);
        Usuario u = await uService.GetUsuarioAsync(usuarioId);

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

15. Adaptando o construtor para chamada do método conforme sinalizado abaixo

```
public ImagemUsuarioViewModel()
{
    string token = Preferences.Get("UsuarioToken", string.Empty);
    uService = new UsuarioService(token);

    AbrirGaleriaCommand = new Command(AbrirGaleria);
    SalvarImagemCommand = new Command(SalvarImagem);
    FotografarCommand = new Command(Fotografar);

    CarregarUsuario();
}
```



Pesquisa de Personagens para as Disputas – Controle Search Bar

1. Crie a classe Disputa dentro da pasta Models com as propriedades a seguir

```
public int Id { get; set; }
0 references
public DateTime? DataDisputa { get; set; }
0 references
public int AtacanteId { get; set; }
0 references
public int OponenteId { get; set; }
0 references
public string Narracao { get; set; }
0 references
public int HabilidadeId { get; set; }
0 references
public List<int> ListaIdPersonagens { get; set; } = new List<int>();
0 references
public List<string> Resultados { get; set; } = new List<string>();
```

2. Abra a classe **PersonagemService** e adicione um método para realizar pesquisar pelo nome aproximado, conforme abaixo

```
public async Task<ObservableCollection<Personagem>> GetByNomeAproximadoAsync(string busca)
{
    string urlComplementar = $"{_apiUrlBase}/GetByNomeAproximado/{busca}";

    ObservableCollection<Models.Personagem> listaPersonagens = await
        _request.GetAsync<ObservableCollection<Models.Personagem>>(_apiUrlBase + urlComplementar, _token);

    return listaPersonagens;
}
```

3. Crie uma pasta chamada Disputas dentro da pasta ViewModels e dentro da pasta Disputas crie uma classe chamada DisputaViewModel, deixando-a pública e fazendo herança com a classe BaseViewModel

4. Declare os objetos que serão utilizados, e os dados de inicialização no construtor. Usings para AppRpgEtec.Model, AppRpgEtec.Services.Personagens e System.Collections.ObjectModel

```
private PersonagemService pService;
1 reference
public ObservableCollection<Personagem> PersonagensEncontrados { get; set; }
1 reference
public Personagem Atacante { get; set; }
1 reference
public Personagem Oponente { get; set; }
0 references
public DisputaViewModel()
{
    string token = Preferences.Get("UsuarioToken", string.Empty);
    pService = new PersonagemService(token);

    Atacante = new Personagem();
    Oponente = new Personagem();

    PersonagensEncontrados = new ObservableCollection<Personagem>();
}
```



5. Programe o método que executará a pesquisa.

```
public async Task PesquisarPersonagens(string textoPesquisaPersonagem)
{
    try
    {
        PersonagensEncontrados = await pService.GetByNomeAproximadoAsync(textoPesquisaPersonagem);
        OnPropertyChanged(nameof(PersonagensEncontrados));
    }
    catch (Exception ex)
    {

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

6. Declare um ICommand e o vincule ao método de pesquisa. Faça o using para System.Windows.Input

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

    Atacante = new Personagem();
    Oponente = new Personagem();

    PersonagensEncontrados = new ObservableCollection<Personagem>();

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

1 public ICommand PesquisarPersonagensCommand { get; set; }
```

7. Programe duas propriedades que retornarão o nome do atacante e do oponente

```
public string DescricaoPersonagemAtacante
{
    get => Atacante.Nome;
}

0 references
public string DescricaoPersonagemOponente
{
    get => Oponente.Nome;
}
```



-
8. Programe o método que receberá o personagem selecionado

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

        if(tipoCombatente == "Atacante")
        {
            Atacante = p;
            OnPropertyChanged(nameof(DescricaoPersonagemAtacante));
        }
        else if (tipoCombatente == "Oponente")
        {
            Oponente = p;
            OnPropertyChanged(nameof(DescricaoPersonagemOponente));
        }
    }
    catch (Exception ex)
    {

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

9. Crie o atributo/propriedade que armazenará o personagem selecionado para executar o método criado na etapa anterior

```
private Personagem personagemSelecionado;
0 references
public Personagem PersonagemSelecionado
{
    set
    {
        if(value != null)
        {
            personagemSelecionado = value;
            SelecionarPersonagem(personagemSelecionado);
            OnPropertyChanged();
            PersonagensEncontrados.Clear();
        }
    }
}
```



10. Abra a content page ListagemView na pasta Views/Disputas e na parte de código declare um objeto ViewModel de Disputa e inicialize-o no construtor, vinculando-o à view.

```
public partial class ListagemView : ContentPage
{
    DisputaViewModel viewModel;
    0 references
    public ListagemView()
    {
        InitializeComponent();

        viewModel = new DisputaViewModel();
        BindingContext = viewModel;
    }
}
```

11. Insira o layout no design da View dentro do VerticalStackLayout que já existia.

```
<SearchBar x:Name="searchBar" HeightRequest="25" Placeholder="Digite o nome do Personagem..."  
          TextColor="Black" SearchCommand="{Binding PesquisarPersonagensCommand}"  
          SearchCommandParameter="{Binding Source={x:Reference searchBar}, Path=Text}"/>

    <ListView x:Name="lvPersonagens" HasUnevenRows="True" ItemsSource="{Binding PersonagensEncontrados}"  
              SelectedItem="{Binding PersonagemSelecionado}"  
              Margin="10, 5, 0, 0">  
        <ListView.ItemTemplate>  
            <DataTemplate>  
                <ViewCell>  
                    <StackLayout Orientation="Vertical">  
                        <Label Text="{Binding Id}" TextColor="Blue" IsVisible="False"/>  
                        <Label Text="{Binding Nome}" TextColor="Blue" FontSize="18"/>  
                        <Label Text="{Binding Classe}" TextColor="Blue" FontSize="14"/>  
                    </StackLayout>  
                </ViewCell>  
            </DataTemplate>  
        </ListView.ItemTemplate>  
    </ListView>

    <Grid HorizontalOptions="Fill" Margin="5, 5, 0, 0" Padding="10, 10, 0, 0">  
        <Grid.ColumnDefinitions>  
            <ColumnDefinition Width="Auto"/>  
            <ColumnDefinition Width="*"/>  
        </Grid.ColumnDefinitions>  
        <Grid.RowDefinitions>  
            <RowDefinition Height="Auto"/>  
            <RowDefinition Height="Auto"/>  
        </Grid.RowDefinitions>  
        <Label Text="Atacante: " Grid.Column="0" Grid.Row="0" FontAttributes="Bold" />  
        <Label Text="{Binding DescricaoPersonagemAtacante}" Grid.Column="1" Grid.Row="0"  
              TextColor="Blue" />  
  
        <Label Text="Oponente: " Grid.Column="0" Grid.Row="1" FontAttributes="Bold" />  
        <Label Text="{Binding DescricaoPersonagemOponente}" Grid.Column="1" Grid.Row="1"  
              TextColor="Blue" />  
    </Grid>
```



- Execute o aplicativo para certificar que o campo de pesquisa está sendo exibido e ao digitar o nome aproximado e clicar na lupa ou no botão de checagem do teclado os personagens vão aparecer. Ao clicar no personagem escolhido, você deverá informar se é um Atacante ou Oponente e os dados do personagem devem aparecer na View.

Busca com modificação imediata após a digitação

12. Insira um campo *Entry* abaixo da *SearchBar*

```
<SearchBar x:Name="searchBar" HeightRequest="25" Placeholder="Digite o nome do Personagem..."  
          TextColor="Black" SearchCommand="{Binding PesquisarPersonagensCommand}"  
          SearchCommandParameter="{Binding Source={x:Reference searchBar}, Path=Text}"/>  
  
<Entry Placeholder="Digite o nome do Personagem..." Text="{Binding TextoBuscaDigitado}">  
</Entry>  
  
<ListView x:Name="lvPersonagens" HasUnevenRows="True" ItemsSource="{Binding PersonagensEncontrados}">
```

13. Retorne para a classe *DisputaViewModel* e crie um atributo propriedade com as características abaixo

```
private string textoBuscaDigitado = string.Empty;  
0 references  
public string TextoBuscaDigitado  
{  
    get { return textoBuscaDigitado; }  
    set  
    {  
        //Verifica se não é nulo, se não é vazio e se o tamanho do texto é maior que zero.  
        if ((value != null && !string.IsNullOrEmpty(value) && value.Length > 0))  
        {  
            textoBuscaDigitado = value;  
            _ = PesquisarPersonagens(textoBuscaDigitado);  
        }  
        else  
        {  
            //Limpa o list view que exibe o resultado da pesquisa  
            PersonagensEncontrados.Clear();  
        }  
    }  
}
```

- Agora você duas maneiras de realizar uma busca, use a que achar melhor.
- A próxima etapa será programar o ataque com armas ou habilidades.



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.AtacantId = 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.AtacantId = 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 ViewModel

```
<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.



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?Id={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 onde existe outros ICommand

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

8. Atribua o método ao ICommand dentro do construtor

```
ZerarRankingResturarVidasGeralCommand =
    new Command(async () => { await ZerarRankingRestaurarVidasGeral(); });
```

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

```
<Button Text="Zerar Ranking Geral" Command="{Binding ZerarRankingResturarVidasGeralCommand}"
    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.

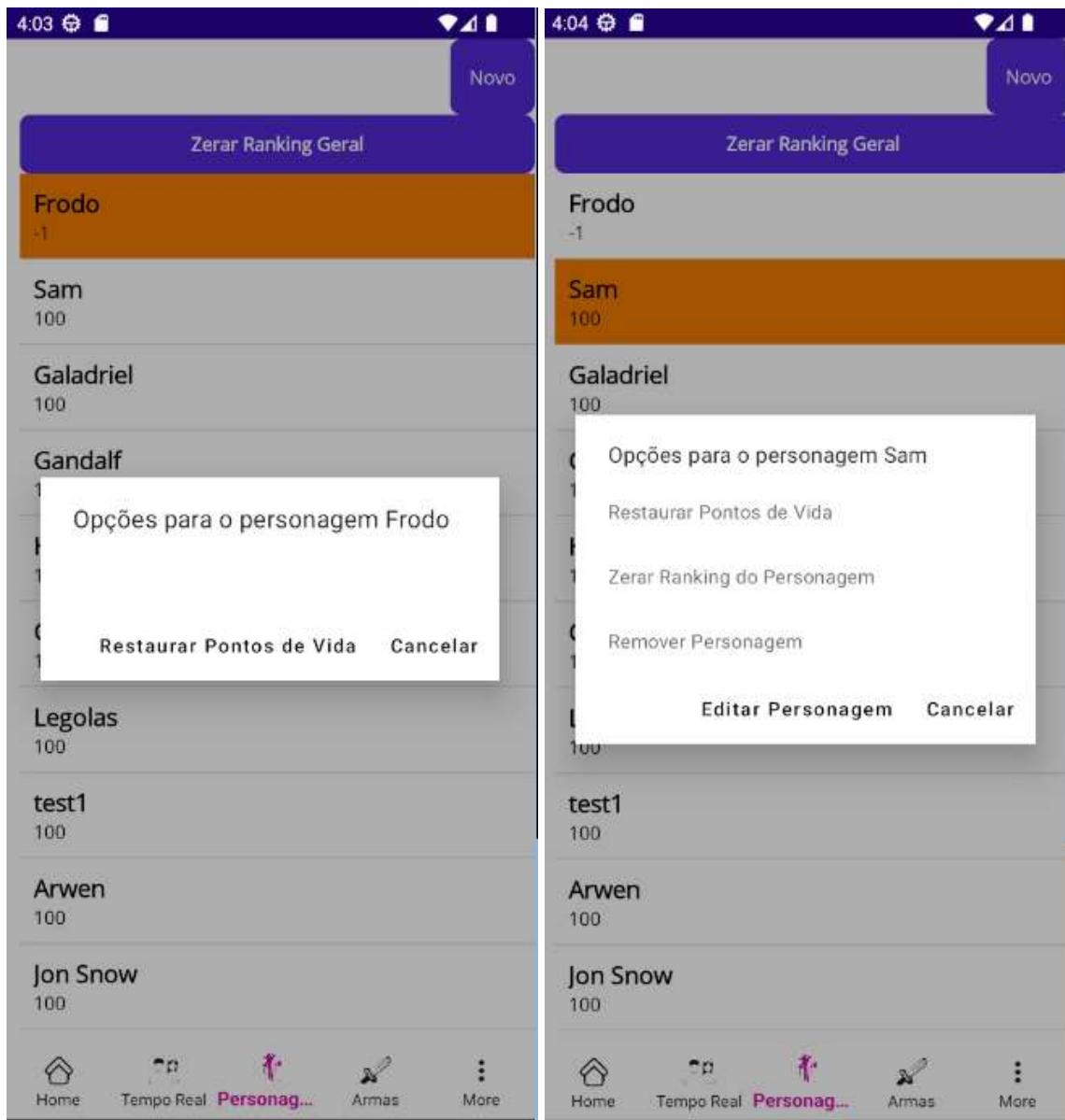
```
public async Task ExibirOpcoesAsync(Personagem personagem)
{
    try
    {
        personagemSelecionado = null;
        string result = string.Empty;

        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)
            ProcessarOpcãoRespondidaAsync(personagem, result);
    }
    catch (Exception ex)
    {
        await Application.Current
            .MainPage.DisplayAlert("Ops...", ex.Message, "Ok");
    }
}
```



- Resultados esperados





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.



Validação de Campos

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

- 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;
}
```

- 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, Força 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>`



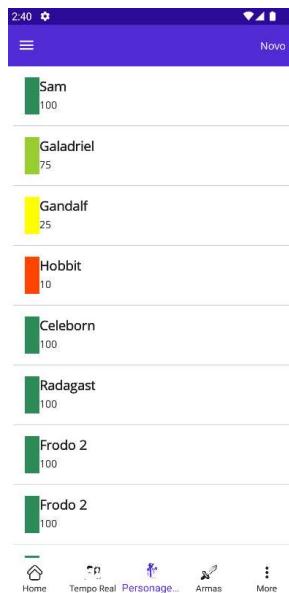
9. Ainda na Listagem de Personagens, procure as labels do nome e pontos de vida dos personagens, modificando o trecho para que tenhamos um Grid e um Box View para exibir a cor sinalizada pelos pontos de vida. Perceba que o Box View conterá a chamada para o Conversor

```
</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>



Envio de E-mail

1. Clique com o direito na pasta Models e adicione uma classe chamada **Email.cs** programando conforme abaixo

```
public class Email
{
    2 references
    public string Remetente { get; set; }
    1 reference
    public string RemententePassword { get; set; }
    1 reference
    public string Destinatario { get; set; }
    2 references
    public string DestinatarioCopia { get; set; }
    1 reference
    public string DominioPrimario { get; set; }
    1 reference
    public int PortaPrimaria { get; set; }
    1 reference
    public string Assunto { get; set; }
    1 referênci
    public string Mensagem { get; set; }
}
```

2. Clique com o direito no projeto C# e crua uma pasta chamada Helpers, adicionando uma nova pasta chamada **Message**. Clique com o direito na pasta Message e crie uma classe chamada **EmailHelper.cs** programando o bloco inicial conforme abaixo.

```
public async Task EnviarEmail(Models.Email email)
{
    try
    {
        ...
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

- Usings: System.Threading.Tasks, AppRpgEtec.Models



-
3. Faça a programação do bloco try. Usings: System.Net.Mail e System.Net. Se for usar anexos exigirá System.IO.

```
string toEmail = email.Destinatario;

MailMessage mailMessage = new MailMessage()
{
    From = new MailAddress(email.Remetente, "App RPG Etec")
};

mailMessage.To.Add(new MailAddress(toEmail));

if (!string.IsNullOrEmpty(email.DestinatarioCopia))
    mailMessage.CC.Add(new MailAddress(email.DestinatarioCopia));

mailMessage.Subject = "App RPG Etec - " + email.Assunto;
mailMessage.Body = GerarCorpoEmail(email.Mensagem);
mailMessage.IsBodyHtml = true;
mailMessage.Priority = MailPriority.High;

//Outras opções --> Anexo:
//byte[] arquivoAnexo = null; string contentType = "application/pdf"; //image/png ou //image/jpeg
//mailMessage.Attachments.Add(new Attachment(new MemoryStream(arquivoAnexo), "nomeArquivo", contentType));

using (SmtpClient smtp = new SmtpClient(email.DominioPrimario, email.PortaPrimaria))
{
    smtp.Credentials = new NetworkCredential(email.Remetente, email.RemetentePassword);
    smtp.EnableSsl = true;
    await smtp.SendMailAsync(mailMessage);
}
```

4. Criaremos o método que vai gerar o corpo da mensagem em HTML, para sanar o erro que deve estar ocorrendo quando você programou o item anterior

```
public string GerarCorpoEmail(string mensagem)
{
    try
    {

    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```



-
5. Use o trecho de código para dentro do bloco try. Faça identação do Código com CTRL + K, D

```
StringBuilder sb = new StringBuilder();
sb.AppendLine("<html> ");
sb.AppendLine("<body> ");
sb.AppendLine("    <div style = 'text-align:center'> ");
sb.AppendLine("        <div style = 'text-align:left'> ");
sb.AppendLine("            <table style = 'width: 600px; border:1px solid #0089cf;' border = '0'
cellspacing = '0' cellpadding = '0'> ");

sb.AppendLine("                <tbody> ");
sb.AppendLine("                    <tr style = 'background-color: #0089cf;'> ");
sb.AppendLine("                        <td> ");
sb.AppendLine("                            <table style = 'width: 100%;' border = '0' cellspacing = '0'
cellpadding = '0'> ");

sb.AppendLine("                                <tbody> ");
sb.AppendLine("                                    <tr> ");
sb.AppendLine("                                        <td style = 'width: 224px;'><img src =
'https://etechoracio.com.br/imagens/logo_sexo_positivo.png' alt = 'Etec Professor Horácio Augusto
da Silveira' width = '224px' height = '148' /></td> ");

sb.AppendLine("                                            <td style = 'font-family: Arial; font-size: 40px;
color: #fff; text-align: center;'> Etec HAS </td> ");

sb.AppendLine("                                    </tr> ");
sb.AppendLine("                                </tbody> ");
sb.AppendLine("                            </table> ");
sb.AppendLine("                        </td> ");
sb.AppendLine("                    </tr> ");
sb.AppendLine("                <tr > ");
sb.AppendLine("                    <td style = 'padding:5px; height: 400px; font-size: 1.2rem; line-
height: 1.467; font-family: 'Segoe UI','Segoe WP',Arial,Sans-Serif; color: #333;
vertical-align:top'> ");

sb.AppendFormat("{0}", mensagem);
sb.AppendLine("                </td> ");
sb.AppendLine("            </tr> ");
sb.AppendLine("            <tr style = 'background-color: #0089cf; color: #fff;'> ");
sb.AppendLine("                <td style = 'padding:5px'> ");
sb.AppendLine("                    Etec Professor Horácio Augusto da Silveira <br/> ");
sb.AppendLine("                    Curso Técnico em Desenvolvimento de Sistemas <br/> ");
sb.AppendLine("                    Rua Alcântara, 113 - Vila Guilherme <br/> São Paulo/SP CEP: 02110-
010 ");
sb.AppendLine("                </td> ");
sb.AppendLine("            </tr> ");
sb.AppendLine("        </tbody> ");
sb.AppendLine("    </table> ");
sb.AppendLine("    </div> ");
sb.AppendLine("    </div> ");
sb.AppendLine("    </body> ");
sb.AppendLine("</html> ");
```



6. Abra a classe UsuarioViewModel e realize a programação abaixo antes da mensagem de boas-vindas.

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

Models.Email email = new Models.Email();
email.Remetente = "luizfernando987@gmail.com";
email.RemententePassword = "*****";
email.Destinatario = "luizfernando987@gmail.com";
email.DominioPrimario = "smtp.gmail.com";
email.PortaPrimaria = 587;
email.Assunto = "Notificação de acesso";
email.Mensagem = $"Usuário {u.Username} acessou o aplicativo" +
    $" em {DateTime.Now:dd/MM/yyyy HH:mm:ss}";

EmailHelper emailHelper = new EmailHelper(); //using AppRpgEtec.Helpers.Message
await emailHelper.EnviarEmail(email);

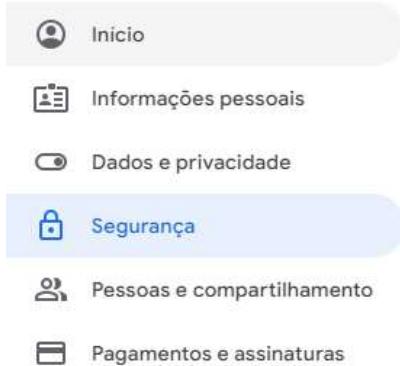
await Application.Current.MainPage
    .DisplayAlert("Informação", mensagem, "Ok");

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

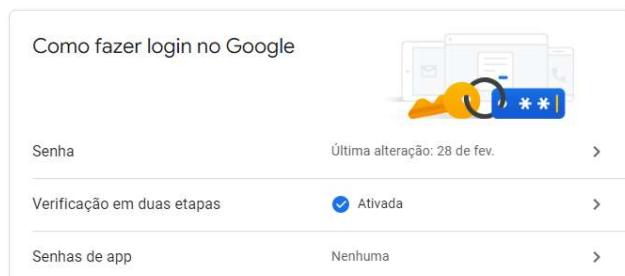
7. Se você usa autenticação em dois fatores para contas google, será necessário criar uma senha de app para que o e-mail consiga ser enviado, veja como habilitar o recurso a seguir:

<https://support.google.com/mail/?p=InvalidSecondFactor>, ou acessando <https://myaccount.google.com>

8. Acesse à esquerda a aba segurança



9. Navegue até “Como fazer login no Google” e selecione “Senhas de App”. Será necessário autenticação.





10. Selecione as opções abaixo

Você não tem nenhuma senha de app.

Selecione o app e o dispositivo para o qual você quer gerar a senha de app.

Gmail ▾ Outro ▾

11. Insira um nome para a senha de app, como no exemplo abaixo

Você não tem nenhuma senha de app.

Selecione o app e o dispositivo para o qual você quer gerar a senha de app.

12. Copie a senha gerada para utilizar no envio de e-mail na classe UsuarioViewModel e volte aos testes.

Senha de app gerada

Sua senha de app para seu dispositivo

Email

Password

Como usar

Acesse as configurações da sua Conta do Google no aplicativo ou dispositivo que você está tentando configurar. Substitua sua senha pela senha de 16 caracteres mostrada acima. Assim como sua senha normal, esta senha de app concede acesso total à sua Conta do Google. Não é necessário memorizá-la, por isso não a anote ou a compartilhe com outras pessoas.



13. Resultados esperados:

Principal Promoções Social

eu App RPG Etec - Notificação de acesso - Etec HAS Usuário usuarioadmin acessou o aplicativo em 2...

App RPG Etec - Notificação de acesso Caixa de entrada

App Rpg Etec para mim



Usuário usuarioadmin acessou o aplicativo em 20/03/2023 02:54:08
Etec Professor Horácio Augusto da Silveira
Curso Técnico em Desenvolvimento de Sistemas
Rua Alcântara, 113 - Vila Guilherme
São Paulo/SP CEP: 02110-010

Responder Encaminhar



Banco de dados no dispositivo com SQLite

1. Expanda o projeto C#, clique com o botão direito em Dependencies e escolha “Manage Nuget Packages”. Selecione a aba Browser, pesquise por **sqlite-net-pcl** e instale o pacote.
2. Crie uma classe chamada **Acesso.cs** dentro da pasta Models com as propriedades abaixo

```
public int Id { get; set; }  
0 references  
public string Login { get; set; }  
1 reference  
public string Dispositivo { get; set; }  
1 reference  
public string Plataforma { get; set; }  
1 reference  
public string Ip { get; set; }  
1 reference  
public DateTime DtAcesso { get; set; }
```

3. Faça a edição da propriedade **Id** para configurar algumas informações para a futura tabela local. Exigirá o using SQLite.

[**PrimaryKey**, **AutoIncrement**]

```
0 references  
public int Id { get; set; }  
0 references  
public string Dispositivo { get; set; }
```

4. Crie uma pasta chamada **Helpers** e dentro dela uma pasta chamada **SQLiteDB**. Você criará a classe **AcessoDB**, codificando a conexão e o construtor que conterá o caminho do banco para criação da tabela. Exibirá os usings SQLite e AppRpgEtec.Models.

```
public class AcessoDB  
{  
    readonly SQLiteAsyncConnection _db;  
    public AcessoDB(string dbPath)  
    {  
        _db = new SQLiteAsyncConnection(dbPath);  
        _db.CreateTableAsync<Acesso>().Wait();  
    }
```



-
5. Ainda na classe AcessoDB, codifique os métodos CRUD para manipulação da tabela. Exigirá os usings System.Collections.Generic e System.Threading.Tasks.

- Buscar todos

```
public Task<List<Acesso>> GetAll()
{
    return _db.Table<Acesso>().OrderByDescending(i => i.Id).ToListAsync();
}
```

- Buscar por Id

```
public Task<Acesso> GetById(int id)
{
    return _db.Table<Acesso>().FirstOrDefaultAsync(i => i.Id == id);
}
```

- Inserir

```
public Task<int> Insert(Acesso model)
{
    return _db.InsertAsync(model);
}
```

- Update

```
public Task<List<Acesso>> Update(Acesso model)
{
    string sql = "Update Acesso set Dispositivo=?, Plataforma=?, Ip=?, DtAcesso=? where Id=?";
    return _db.QueryAsync<Acesso>(sql,
        model.Dispositivo, model.Plataforma, model.Ip, model.DtAcesso, model.Id);
}
```

- Delete

```
public Task<int> Delete(int id)
{
    return _db.Table<Acesso>().DeleteAsync(i => i.Id == id);
}...
```



- Buscar pelo login

```
public Task<List<Acesso>> Search(string q)
{
    string sql = "select * from Acesso where Login like '%" + q + "%'";
    return _db.QueryAsync<Acesso>(sql);
}
```

6. Abra a classe App.xaml.cs e configure um atributo e propriedade responsável por verificar se a classe AcessoDB já criou um banco de dados no dispositivo, caso ele não exista ainda, ele será criado. Exigirá o using AppRpgEtec.Helpers.SQLiteDB e System.IO.

```
static AcessoDB database;
0 references
public static AcessoDB Database
{
    get
    {
        if (database == null)
        {
            database =
                new AcessoDB(Path.Combine(Environment.GetFolderPath(
                    Environment.SpecialFolder.LocalApplicationData), "AppRpgEtecDb"));
        }
        return database;
    }
}
```

7. Abra a classe **UsuarioViewModel** e procure pelo método ConsultarUsuario. Você deverá inserir antes da mensagem de boas-vindas o trecho de código abaixo responsável por coletar informações do usuário e do dispositivo e mandar para a base de dados.

```
Acesso acesso = new Acesso();
acesso.Login = u.Username;
acesso.Dispositivo = $"{DeviceInfo.Manufacturer} - {DeviceInfo.Name}";
acesso.Plataforma = $"{DeviceInfo.Platform} Versão {DeviceInfo.VersionString}";
acesso.DtAcesso = DateTime.Now;
await App.Database.Insert(acesso);

string mensagem = string.Format("Bem-vindo {0}", u.Username);
await Application.Current.MainPage
    .DisplayAlert("Informação", mensagem, "Ok");
```



8. Clique com o botão direito na pasta **ViewModels/Usuarios** e crie uma classe chamada **AcessoViewModel.cs**, fazendo a herança para a classe **BaseViewModel**

```
public class AcessoViewModel : BaseViewModel
```

9. Crie uma coleção do tipo Acesso (1), um método que faça a busca no banco local e abasteça a ObservableCollection (2) e um construtor que inicialize a lista e realize a chamada ao método (3)

```
public AcessoViewModel()
{
    ListaAcessos = new ObservableCollection<Acesso>();
    ObterAcessos();
}
```

2 references

```
1 public ObservableCollection<Acesso> ListaAcessos { get; set; }
```

1 reference

```
public async void ObterAcessos()
{
    List<Acesso> list = await App.Database.GetAll();
    list.ForEach(i => ListaAcessos.Add(i));
}
```

10. Clique com o botão direito na pasta Views/Usuarios e crie uma content page chamada **AcessoView.Xaml**, inserindo o layout abaixo dentro da tag **StackLayout**

```
<ListView ItemsSource="{Binding ListaAcessos}">
    <ListView.Header>
        <Grid RowDefinitions="*,*,*">
            <Label Grid.Row="0" Grid.Column="0" Text="Id"></Label>
            <Label Grid.Row="0" Grid.Column="1" Text="Login"></Label>
            <Label Grid.Row="0" Grid.Column="2" Text="Dispositivo"></Label>
            <Label Grid.Row="0" Grid.Column="3" Text="Acesso"></Label>
        </Grid>
    </ListView.Header>
    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <Grid RowDefinitions="Auto" ColumnDefinitions="*,*,*">
                    <Label Grid.Row="0" Grid.Column="0" Text="{Binding Id}"></Label>
                    <Label Grid.Row="0" Grid.Column="1" Text="{Binding Login}"></Label>
                    <Label Grid.Row="0" Grid.Column="2" Text="{Binding Dispositivo}"></Label>
                    <Label Grid.Row="0" Grid.Column="3" Text="{Binding DtAcesso,
StringFormat='{0:dd/MM/yyyy HH:mm}'}"></Label>
                </Grid>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```



11. Navegue até a parte de código da View (F7) e programe a vinculação da ViewModel com a View. Exigirá o using AppRpgEtec.Models, System.Collections.Generic e System.Collections.ObjectModel.

```
private AcessoViewModel viewModel;  
0 references  
public AcessoView()  
{  
    InitializeComponent();  
  
    viewModel = new AcessoViewModel();  
    BindingContext = viewModel;  
}
```

12. Abra a view FlyoutMenu.xaml e faça a edição do menu de usuários para que tenhamos duas ShellContents

```
<Tab Title="Usuários" Route="usuariosInfo" Icon="MenuUsuarios.png">  
    <ShellContent Title="Usuários"  
        ContentTemplate="{DataTemplate viewsUsuarios:ListagemView}" />  
  
    <ShellContent Title="Acessos"  
        ContentTemplate="{DataTemplate viewsUsuarios:AcessoView}" />  
</Tab>
```

- Execute o aplicativo para confirmar que será exibida uma listagem

13. Abra a classe AcessoViewModel e crie um método para exibir os detalhes do acesso. Exigirá o using Xamarin.Forms

```
public async void ExibirDetalhes()  
{  
    string detalhes =  
        $" Data: {acessoSelecionado.DtAcesso:dd/MM/yyyy HH:mm} " +  
        $"\\n Login: {acessoSelecionado.Login} " +  
        $"\\n Dispositivo: {acessoSelecionado.Dispositivo} " +  
        $"\\n Plataforma: {acessoSelecionado.Plataforma}";  
  
    await Application.Current.MainPage  
        .DisplayAlert("Informação", detalhes, "Ok");  
}
```



-
14. Crie um atributo/propriedade para o acesso selecionado que no *set* faça a chamada do método que exibe os detalhes

```
private Acesso acessoSelecionado; //CTRL + R,E
0 references
public Acesso AcessoSelecionado
{
    get { return acessoSelecionado; }
    set
    {
        if (value != null)
        {
            acessoSelecionado = value;
            ExibirDetalhes();
        }
    }
}
```

15. Volte até a view AcessoView.Xaml e acrescente uma propriedade que aponte para os dados criados na etapa anterior

```
<ListView ItemsSource="{Binding ListaAcessos}"
          SelectedItem="{Binding AcessoSelecionado}">
    <ListView.Header>
```

- Execute o aplicativo para testar a exibição dos detalhes

16. Ainda na view de acessos, insira as tags abaixo, acima do listView referente a busca por login

```
<StackLayout >
    <SearchBar Placeholder="login" Text="{Binding ParametroBusca}">
    </SearchBar>
    <Button Text="Buscar pelo login" Command="{Binding BuscarCommand}"></Button>
</StackLayout>
```

17. Abra a classe AcessoViewModel e crie uma propriedade para guardar os dados da busca

```
public string ParametroBusca { get; set; }
```



18. Crie um método para fazer a busca na base de dados local

```
public async void ObterAcessosPorLogin()
{
    ListaAcessos.Clear();...
    List<Acesso> list = await App.Database.Search(ParametroBusca);
    list.ForEach(i => ListaAcessos.Add(i));
}
```

19. Insira o command que se interligará ao botão de busca (1) e inicialize o command no construtor, vinculando ao método de busca no banco de dados (2). Exigirá o using System.Windows.Input

```
public AcessoViewModel()
{
    ListaAcessos = new ObservableCollection<Acesso>();
    ObterAcessos();

    2 BuscarCommand = new Command(ObterAcessosPorLogin);
}
```

1 reference
1 public ICommand BuscarCommand { get; }

- Para esse teste seria interessante que você tenha dois logins diferentes para acessar o aplicativo em momentos diferentes para realizar o filtro.