



Aula 07 – Tela de registro e login – Classe de Serviço - Configurações do App

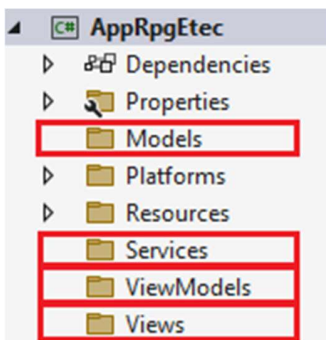
Crie um projeto **.NET MAUI App** com as seguintes recomendações:

Project name: **AppRpgEtec**

Framework: **.NET 7.0**

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
{
    0 references
    public int Id { get; set; }
    0 references
    public string Username { get; set; }
    0 references
    public string PasswordString { get; set; }
    0 references
    public string Perfil { get; set; }
    0 references
    public string Token { get; set; }
    0 references
    public byte[] Foto { get; set; }
    0 references
    public string Email { get; set; }
    0 references
    public double? Latitude { get; set; }
    0 references
    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)
{
    HttpClient httpClient = new HttpClient();

    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
        return 0;
}

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));

    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**

```
1 reference
public class UsuarioService : Request
{
    private readonly Request _request;
    private const string apiUriBase = "http://xyz.somee.com/RpgApi/Usuarios";
    //xyz --> Site da sua API

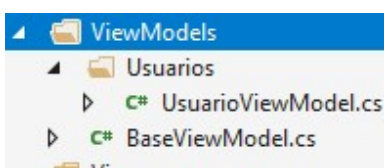
    0 references
    public UsuarioService()
    {
        _request = new Request();
    }

    0 references
    public async Task<Usuario> PostRegistrarUsuarioAsync(Usuario u)
    {
        //Registrar: Rota para o método na API que registrar o usuário
        string urlComplementar = "/Registrar";
        u.Id = await _request.PostReturnIntAsync(apiUriBase + urlComplementar, u);
        return u;
    }

    0 references
    public async Task<Usuario> PostAutenticarUsuarioAsync(Usuario u)
    {
        //Autenticar: Rota para o método na API que autentica com login e senha
        string urlComplementar = "/Autenticar";
        u = await _request.PostAsync(apiUriBase + 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.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Text;

namespace AppRpgHAS.ViewModels
{
    1 reference
    public class BaseViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler PropertyChanged;

        0 references
        public void OnPropertyChanged([CallerMemberName] string name = "")
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
        }
    }
}
```

- 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

```
0 references
public class UsuarioViewModel : BaseViewModel
{
    private UsuarioService uService;
    0 references
    public ICommand RegistrarCommand { 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 consulta 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.

```
#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");
    }
}
#endregion
```



10. Continue a codificação do método programando a codificação abaixo dentro do try. Faça o using de AppRpgEtec.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
    }
}
```

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 RegistrarCommand { get; set; }

    //ctor + TAB + TAB: Atalho para criar o construtor
    1 reference
    public UsuarioViewModel()
    {
        uService = new UsuarioService();
        InicializarCommands();
    }
    2

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

- 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. Crie mais um método, desta vez para fazer a autenticação do usuário. 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.

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



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

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

14. 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*.

15. 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;  
    }  
}
```




16. 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());
}
```

- Compile para certificar que está sem erros e execute o app tentando realizar o login, faça uma tentativa também com dados incorretos.

17. Clique com o direito na pasta *Views/Usuarios* e crie uma View chamada de **CadastroView.xaml**, adicionando 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="Registrar" Command="{Binding RegistrarCommand}" Margin="0,10,0,0"/>
```

18. 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;
    }
}
```

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

```
public async Task DirecionarParaCadastro()//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");
    }
}
```



20. Declare mais um command e realize a vinculação do mesmo com o método DirecionarParaCadastro

```
1 reference
public ICommand DirecionarCadastroCommand { get; set; }

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

21. 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>
```

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

```
[Application(UsesCleartextTraffic=true)]

1 reference
public class MainApplication : MauiApplication
{
```

- 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->