ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2024/2

Luiz Fernando Souza

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 **ImagemUsuarioViewModel.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.

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

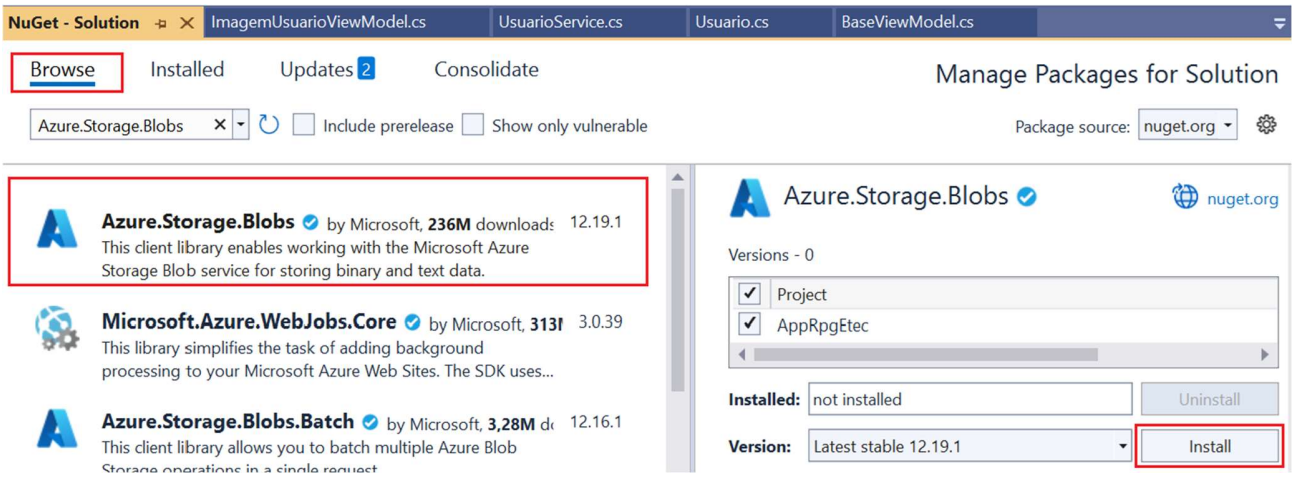
4. Crie o método para fotografar com a estrutura do try/catch. Use o atalho try + TAB + TAB

```
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 do 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



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 SalvarImagemAzure()
{
    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 armazeda
        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 inicialize-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);
    SalvarImagemCommand = new Command(SalvarImagem);
}

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

9. Na pasta Views/Usuarios, crie uma *Content Page* (.Net Maui) chamada **ImagemUsuarioView.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 SalvarImagemCommand}" />
                        </Grid>
                    </VerticalStackLayout>
                </ScrollView>
                    <Image Source="{Binding FonteImagem}" Margin="10"/>
```

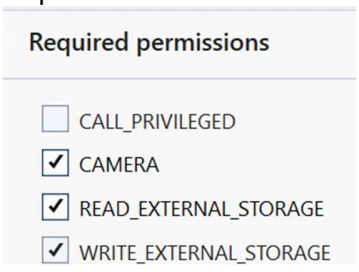
11. Na parte de Código da view iremos fazer a vinculação entre a *ViewModel* e a *View*. Necessário o using de AppRpgEtec.ViewModels.Usuarios

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

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

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

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

    <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 *Plataforms/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 **ImagemUsuarioViewModel.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.

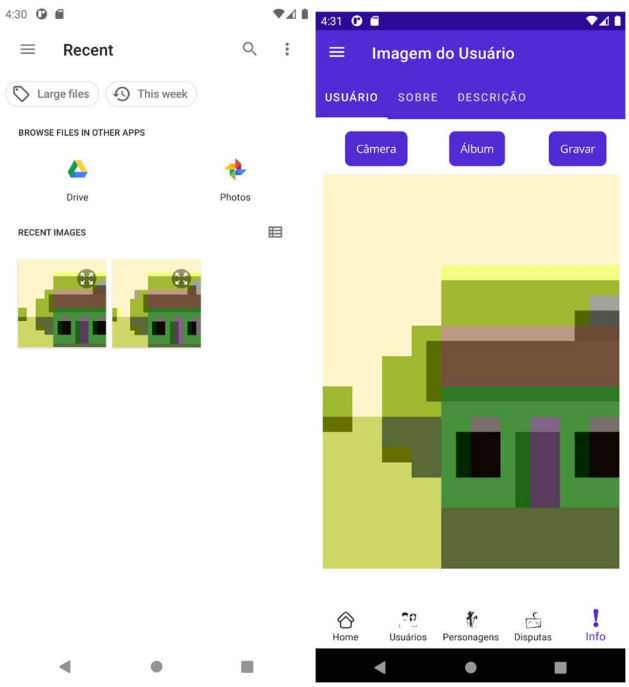
```
public async void AbrirGaleria()
{
    try
    {
        //Verificação se o dispositivo suporta câmera.
        if (MediaPicker.Default.IsCaptureSupported)
        {
            //Chamada para a galeria do dispositivo. Fica aguardando usuário escolher a foto da galeria.
            FileResult photo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

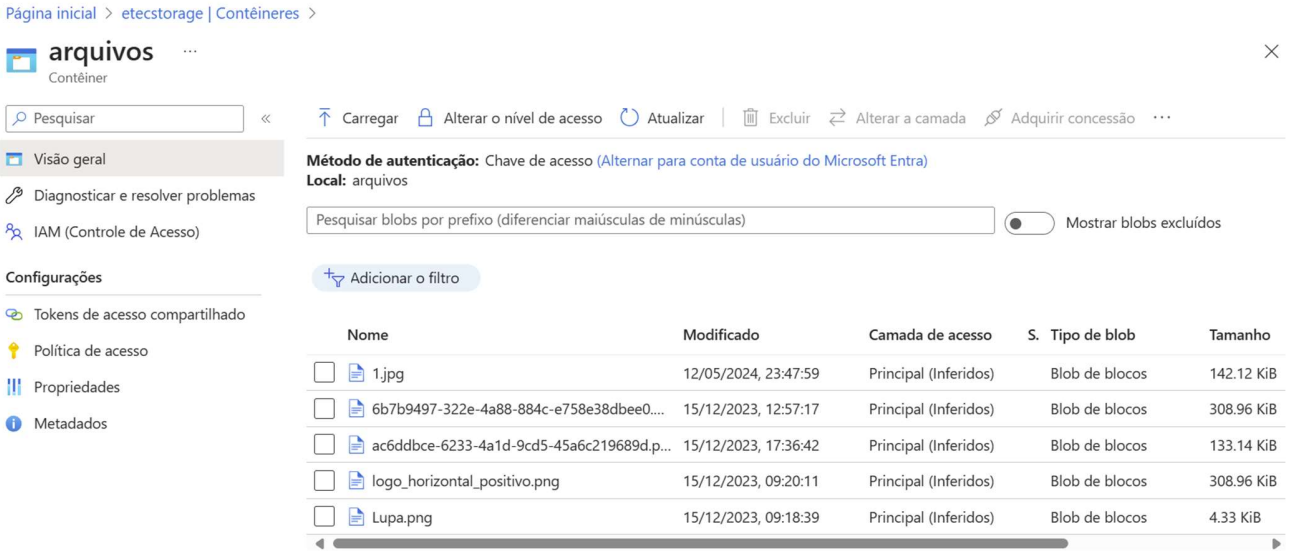
```
public ImagemUsuarioViewModel()
{
    string token = Preferences.Get("UsuarioToken", string.Empty);
    uService = new UsuarioService(token);

2   FotografarCommand = new Command(Fotografar);
    SalvarImagemCommand = new Command(SalvarImagemAzure);
    AbrirGaleriaCommand = new Command(AbrirGaleria);
}
```

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

- Resultado esperados para acionamento da câmera e galeria



- Pasta de armazenamento do azure



Luiz Fernando Souza

Apenas para ciência

- Caso a fossemos inserir a imagem na API teriamos 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 SalvarImagem()
{
    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");
    }
}
```