ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2024/1

Luiz Fernando Souza

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 sobrescritos 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();
}
```

Sobrescreva os eventos do clico 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));
}
```

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2024/1

Luiz Fernando Souza

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 aperecerá 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="" />
                                                                        <New Event Handler>
/erticalStackLayout>                                                    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;
}
```

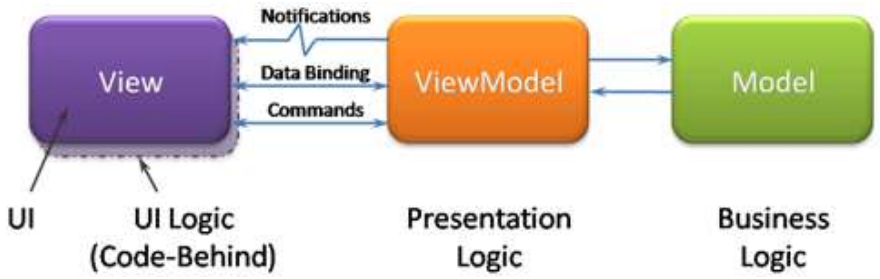
Insira breakpoints nos construtores das classes mencionadas e nos Métodos OnStart, OnSleep e OnResume, executar o aplicativo e depurar (debugar) alternando para a área de trabalho do emulador/celular e voltando o aplicativo para o primeiro plano. Lembrete sobre depuração: F9 → Insere e remove breakpoints; 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.

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2024/1

Luiz Fernando Souza

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.ComponenteModel (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));
    }
}
```

Luiz Fernando Souza

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}";
```

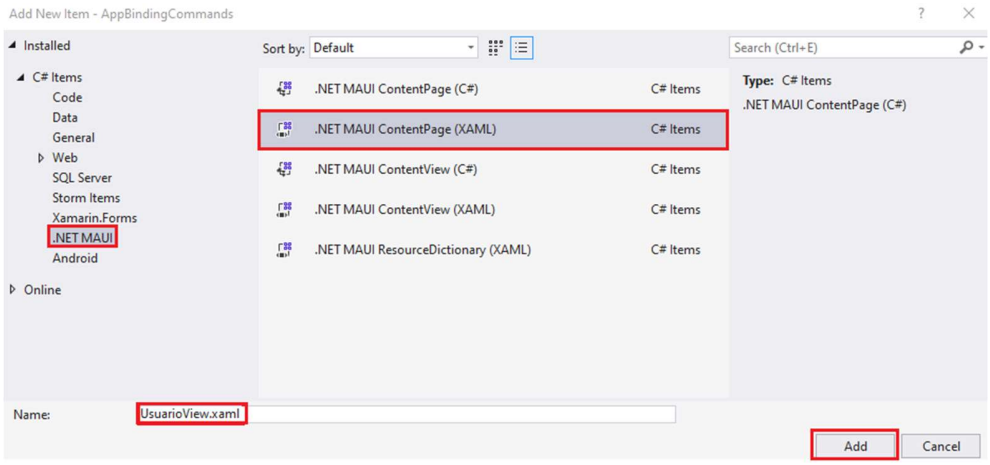
No Set da propriedade Name, inclua o OnPropertyChanged referente ao DisplayName

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

        name = value;
        OnPropertyChanged(nameof(Name));
        OnPropertyChanged(nameof(DisplayName));
    }
}
```

Luiz Fernando Souza

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 imedintamente na label.

ETEC PROF. HORÁCIO AUGUSTO DA SILVEIRA

PROGRAMAÇÃO DE APLICATIVOS MOBILE – II – 2024/1

Luiz Fernando Souza

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.