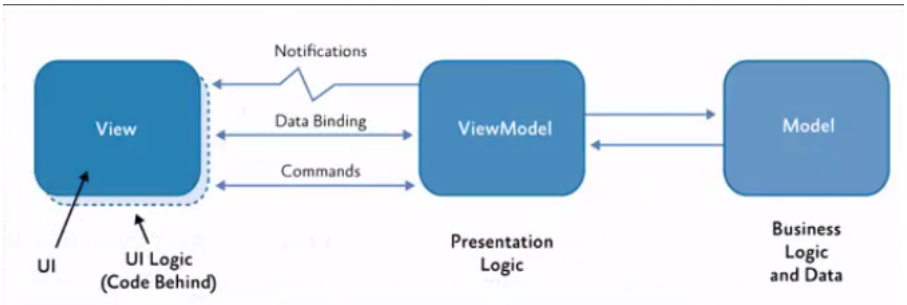
08

Binding context

Transcrição

Aprendemos e nos aprofundamos em relação ao padrão de arquitetura chamado **MVVM** (*Pattern Model-View-ViewModel*), que foi estudado e aplicado em todas as *views* da aplicação de *TestDrive*.



Faremos o mesmo agora aplicando-se este padrão à *view* de login. Entraremos em "*TestDrive (Portable)* > *ViewModels*", clicando nesta pasta com o lado direito do mouse e em "*Add* > *Class*", já que criaremos uma nova classe, chamada `LoginViewModel.cs`.

Com isto, precisamos definir uma instância deste `ViewModel` como contexto de *binding*, de amarração, a ser utilizado na *view* de login. No arquivo `LoginView.xaml`, queremos que estas propriedades sejam amarradas como as que estarão na classe `LoginViewModel.cs`.

No curso anterior vimos como fazer este *binding*, a amarração do contexto (*binding context*) por meio de código C#. Agora, o veremos de outra forma, amarrando o `ViewModel` com a `View` através do código `.xaml`. Definiremos no código `.xaml` da `LoginView` o `namespace`, a localização do `ViewModel`. Ao abrirmos o `LoginViewModel.cs`, copiaremos e utilizaremos seu `namespace`, `TestDrive.ViewModels`, em `LoginView.xaml`.

Para nomearmos um `namespace` em `.xaml`, precisamos do atributo `xmlns`, que significa "*namespace* do xml", um apelido. Queremos uma palavra curta e de fácil acesso. Em vez de colocarmos `TestViewModels`, usaremos `vm` (que significa `ViewModels`). Dentro dele, colaremos o que já encontramos como nome de `namespace`, ou o procuraremos dentre as opções oferecidas pelo Visual Studio:

```
xmlns:vm="clr-namespace:TestDrive.ViewModels"
```

Com isto temos o `namespace` setado no XAML. Agora precisamos definir o contexto de *binding* do `LoginView`, a propriedade `BindingContext` da `ContentPage`. Dentro desta tag, definiremos outra acessando o próprio `ContentPage`, porém colocando-se a propriedade `BindingContext`. Ao fazermos isto, é como se estivessemos usando o código C# para `this.BindingContext`.

Dentro dela, criaremos uma nova instância de `LoginViewModel`, o qual não aparece como sugestão pelo IntelliSense, pois o XAML não reconhece este tipo, nem sabe onde está esta classe. Por isto mesmo vamos utilizar o `namespace` definido anteriormente (`vm`), como prefixo. Dentro da tag `ContentPage`, que já existia antes, acrescentamos as seguintes linhas:

```
<ContentPage.BindingContext>
  <vm:LoginViewModel></vm:LoginViewModel>
```

```
</ContentPage.BindingContext>
```

Criamos a nova classe `LoginViewModel` porém ainda não definimos as propriedades. Faremos isto abrindo este arquivo e definindo uma propriedade para cada um dos campos da tela de login. Primeiramente, deixaremos a classe pública, dentro da qual incluiremos duas propriedades, uma para "Usuário" e outra para "Senha".

Digitaremos "propfull", um atalho muito útil (*snippet*), apertando-se "TAB" duas vezes logo em seguida. Nos aparece toda a estrutura de uma propriedade a ser preenchida, em que trocamos o tipo `int` por `string`, fazendo o mesmo com `myVar` por `usuario`. Com o "TAB", vamos colocar "Usuario" com a primeira letra em caixa alta, e a propriedade é definida em `ViewModel`.

```
namespace TestDrive.ViewModels
{
  public class LoginViewModel
  {
    private string usuario;
    public string Usuario
    {
      get { return usuario; }
      set { usuario = value; }
    }
  }
}
```

Repetiremos o procedimento para a "Senha", e teremos duas propriedades definidas em `ViewModel`, que serão utilizadas para o `BindingContext` com a `LoginView`:

```
namespace TestDrive.ViewModels
{
  public class LoginViewModel
  {
    private string usuario;
    public string Usuario
    {
      get { return usuario; }
      set { usuario = value; }
    }

    private string senha;
    public string Senha
    {
      get { return senha; }
      set { senha = value; }
    }
  }
}
```

No `LoginView` precisaremos definir o `BindingContext` preenchendo-se os campos de entrada de digitação de "Usuário" e "Senha". Como visto no curso anterior, é preciso passar a propriedade, ou controle, deste componente `Entry`, que receberá o valor da propriedade da classe `LoginViewModel`. É o `Text`, a ser preenchido pelo usuário, e que será o *binding*, a amarração com a propriedade na classe de `LoginViewModel`. O código ficará assim:

```
<StackLayout VerticalOptions="Center" Margin="64">
  <Image Source="aluracar.png"></Image>
  <Entry Placeholder="Usuário" Text="{Binding Usuario}"></Entry>
  <Entry Placeholder="Senha" Text="{Binding Senha}" IsPassword="True"></Entry>
  <Button Text="Entrar" Clicked="Button_Clicked"></Button>
</StackLayout>
```

Continuando com o padrão MVVM, falta adequarmos o botão do `LoginView` também. No diagrama mostrado no início deste vídeo, vemos que o botão neste padrão é feito por meio do comando de *binding* para ligar a *view* até `ViewModel`, que é onde deveria estar o código que representa a ação executada assim que o usuário clicar no botão.

Implementaremos o *binding* do botão removendo o evento `Button_Clicked` e trocando-o por um comando, e o *binding* será feito para uma propriedade que será uma instância de um comando do `LoginViewModel`, denominado `EntrarCommand`:

```
<StackLayout VerticalOptions="Center" Margin="64">
  <Image Source="aluracar.png"></Image>
  <Entry Placeholder="Usuário" Text="{Binding Usuario}"></Entry>
  <Entry Placeholder="Senha" Text="{Binding Senha}" IsPassword="True"></Entry>
  <Button Text="Entrar" Command="{Binding EntrarCommand}"></Button>
</StackLayout>
```

Agora vamos até o `LoginViewModel` para colocarmos uma propriedade que será um comando para o login, o botão. Esta propriedade terá o `private set`, pois não conseguiremos alterá-la "do lado de fora", sendo do tipo `ICommand` e de nome `EntrarCommand`.

Precisamos definir a instância do comando de entrada dentro do construtor de `LoginViewModel`. Para isto, digitaremos "ctor", apertando "TAB" duas vezes, e o Visual Studio estrutura o construtor para nós. Dentro do comando, passaremos um método anônimo (uma `Action`), dentro do qual estará o código a ser executado assim que o usuário clicar no botão de "Entrar". Como estamos migrando para o padrão MVVM, podemos simplesmente pegar a linha executada nesta ação, que estávamos referenciando no evento `Button_Clicked`.

Sairemos do `LoginViewModel`, indo ao `LoginView.xaml.cs`, que é o *code behind* da *view* `LoginView`, e copiaremos a linha `MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");`, a qual envia uma mensagem informando sobre o sucesso do login. Colaremos-na no método anônimo onde é executado o comando de entrar.

```
namespace TestDrive.ViewModels
{
  public class LoginViewModel
  {
    private string usuario;
    public string Usuario
    {
      get { return usuario; }
      set { usuario = value; }
    }

    private string senha;
    public string Senha
    {
      get { return senha; }
```

```
        set { senha = value; }
      }

      public ICommand EntrarCommand { get; private set; }

      public LoginViewModel()
      {
        EntrarCommand = new Command(() =>
        {
          MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");
        });
      }
    }
  }
```

Para não deixarmos nenhum "lixo" de código, removeremos o método `Button_Clicked` de `LoginView.xaml.cs`, que não será mais utilizado. Com isto, fizemos o *binding* tanto para os campos de entrada de "Usuário" e "Senha" quanto para o comando do botão "Entrar".

Vamos rodar a aplicação e ver se nada foi quebrado, se está tudo funcionando conforme esperado. É recomendável sempre testarmos a aplicação a cada mudança realizada. Abrindo-se o emulador, digitamos "joao@alura.com.br" no campo de "Usuário" e "alura123" no de "Senha", apertando "Entrar" em seguida. Conseguimos navegar normalmente para a página de listagem de veículos sem nenhum problema. Fizemos a transição do nosso código para o padrão MVVM do Xamarin Forms.