08

Entrycell

Transcrição

A seguir, precisamos amarrar os campos do formulário com os dados da `ViewModel`. Além disso, iremos validá-los, já que não podemos clicar em "Salvar" aleatoriamente, uma vez que os dados ali contidos precisam ser válidos para serem efetivamente salvos após as alterações.

Para isto, pausaremos a aplicação e, em `MasterView.xaml`, associaremos o conjunto de tags `EntryCell` com propriedades do `ViewModel`. Inicialmente, devemos saber qual a propriedade do componente visual que queremos amarrar. Para `EntryCell`, trata-se de `Text`. Para cada uma delas, criaremos um `Binding`.

```
<TableView>
  <TableRoot>
    <TableSection Title="Dados Pessoais">
      <EntryCell Placeholder="Nome" Text="{Binding Nome}"></EntryCell>
      <EntryCell Placeholder="Data de Nascimento" Text="{Binding DataNascimento}"></EntryCell>
      <EntryCell Placeholder="Telefone" Text="{Binding Telefone}"></EntryCell>
      <EntryCell Placeholder="E-mail" Text="{Binding Email}"></EntryCell>
    </TableSection>
  </TableRoot>
</TableView>
```

Para garantir que estas propriedades que estamos amarrando existam em `MasterViewModel.cs`, verificaremos que ali já existem `Nome` e `Email`. Portanto, faltam as propriedades relativas a "data de nascimento" e "telefone", os quais acrescentaremos:

```
public string DataNascimento
{
    get { return this.usuario.dataNascimento; }
    set { this.usuario.dataNascimento = value; }
}

public string Telefone
{
    get { return this.usuario.telefone; }
    set { this.usuario.telefone = value; }
}
```

O próximo passo consiste em criarmos o comando a ser executado quando o usuário clicar no botão "Salvar". Neste mesmo arquivo, vamos adicionar um novo `Command` chamado `SalvarPerfilCommand`, da mesma forma como anteriormente fizemos com o `EditarPerfilCommand`.

```
private readonly Usuario usuario;

public ICommand EditarPerfilCommand { get; private set; }
public ICommand SalvarPerfilCommand { get; private set; }
```

Ao clicar em "Salvar", o comando será executado; definiremos também sua `Action` e o instanciaremos, pois até então apenas sua propriedade foi definida. No construtor de classe, precisaremos declará-lo como um novo comando, dentro do qual estará a `Action`, sem parâmetros e com expressão lambda.

```
public MasterViewModel(Usuario usuario)
{
    this.usuario = usuario;

    EditarPerfilCommand = new Command(() =>
    {
        MessagingCenter.Send<Usuario>(usuario, "EditarPerfil");
    });

    SalvarPerfilCommand = new Command(() =>
    {

    });
}
```

Então, rodaremos a aplicação de novo para verificarmos seu funcionamento. Todos os dados cadastrados do usuário foram trazidos à página de edição de perfil, o que demonstra que o `Binding` foi bem-sucedido, e o botão "Salvar" executa a `Action` definida a partir do comando `SalvarPerfilCommand`.

Se adicionarmos um *breakpoint* nesta `Action` e clicarmos no botão, nada acontecerá, porque ainda não associamos o `Binding` deste comando no XAML.

Faremos isto agora abrindo `MasterView.xaml` e incluindo mais informações na tag referente a `Button`:

```
<Button Text="Salvar" Command="{Binding SalvarPerfilCommand}"></Button>
```

A partir daí, pode-se rodar novamente a aplicação. Na aba "Editar", clicaremos em "Salvar" e, agora sim, conseguiremos chegar à `Action` definida em `SalvarPerfilCommand`.

Ao clicarmos neste botão, queremos ser redirecionados à aba "Usuário". Para isto, faremos algo parecido com o processo que tivemos ao selecionar "Perfil" na primeira aba, lançando uma nova mensagem no `MessagingCenter`, denominada `SucessoSalvarUsuario`. Acrescentaremos o código abaixo em `MasterViewModel.cs`:

```
SalvarPerfilCommand = new Command(() =>
{
    MessagingCenter.Send<Usuario>(usuario, "SucessoSalvarUsuario");
});
```

Além de enviar a mensagem, também é necessário interceptá-la, chegando-se à *view* capaz de fazer a troca entre as abas, a `TabbedPage`. Indo à `MasterView.xaml` e acessando seu *code behind*, no qual se localizam os eventos `OnAppearing` e `OnDisappearing`, acrescentaremos a assinatura desta mensagem a partir de `MessagingCenter`. Em seguida, vamos inserir também uma `Action` que receberá a instância de `usuario`, dentro do qual faremos a troca de aba, passando da aba `2` para a `1`. Com isso, trocaremos a página atual do `TabbedPage` (`CurrentPage`).

```
MessagingCenter.Subscribe<Usuario>(this, "SucessoSalvarUsuario", (usuario) =>
{
    this.CurrentPage = this.Children[0];
});
```

Ainda precisamos cancelar a assinatura quando a página deixar de ser exibida na tela. Dentro de `OnDisappearing`, então, acrescentaremos:

```
MessagingCenter.Unsubscribe<Usuario>(this, "SucessoSalvarUsuario");
```

Agora o código está ficando muito grande no método `OnAppearing`. Para evitarmos que este tenha muitas responsabilidades, vamos extrair os dois `MessagingCenter`s para um novo método. Após selecioná-los, clicaremos com o botão direito do mouse e em "*Quick Actions and Refactorings...*", depois, em "*Extract Method*" e nomeando-o `AssinarMensagens()`.

O código ficará assim:

```
protected override void OnAppearing()
{
    base.OnAppearing();

    AssinarMensagens();
}

private void AssinarMensagens()
//...
```

Onde aparece `OnDisappearing`, teremos as duas chamadas para cancelamento das mensagens, portanto faremos a mesma extração, chamando o novo método de `CancelarMensagens()`:

```
protected override void OnDisappearing()
{
    base.OnDisappearing();

    CancelarMensagens();
}

private void CancelarMensagens()
...
```

Moveremos o método `AssinarMensagens()` para baixo, desta forma, ele ficará junto com o outro, tornando o código mais claro. De volta ao `MasterViewModel.cs`, vimos que ocorre o mesmo: estamos enviando dois comandos. Extrairemos um método novo a partir deles, chamando-o de `DefinirComandos`:

```
public MasterViewModel(Usuario usuario)
{
    this.usuario = usuario;
```

```
            DefinirComandos(usuario);
        }
```

Rodando a aplicação, veremos que ele está realmente fazendo a troca entre as abas. Somos capazes disto, tanto quando clicamos em "Perfil", na primeira aba (o que resulta no redirecionamento a segunda página do `TabbedPage`), quanto em "Salvar", na segunda aba (indo à primeira).