09

Editar

Transcrição

Quando o usuário clica na aba "Editar" para alterar seus dados cadastrais, queremos que haja um formulário com os campos bloqueados para edição. Estes campos sejam habilitados para isto com um botão "Editar", que ainda não existe.

Para implementarmos este novo botão, abriremos `MasterView.xaml`, em que se localizam os `EntryCell`s em `TableView`, que possuem uma propriedade que permite que os campos sejam habilitados ou não para edição: o `IsEnabled`. Criaremos um `Binding` neles para a nova propriedade do `ViewModel`:

```
<TableSection Title="Dados Pessoais">
    <EntryCell Placeholder="Nome" Text="{Binding Nome}" IsEnabled="{Binding Editando}"></EntryCe
    <EntryCell Placeholder="Data de Nascimento" Text="{Binding DataNascimento}" IsEnabled="{Binc
    <EntryCell Placeholder="Telefone" Text="{Binding Telefone}" IsEnabled="{Binding Editando}">
    <EntryCell Placeholder="E-mail" Text="{Binding Email}" IsEnabled="{Binding Editando}"></Entr
</TableSection>
```

Em `MasterViewModel.cs` também precisaremos acrescentar a propriedade do tipo `boolean` chamada `Editando`, logo após `Email`, modificando-se o *setter* para privado, pois esta propriedade não pode ser alterada fora desta classe.

Definiremos `editando` com valor padrão "falso", já que inicialmente o usuário não poderá editar seus dados.

```
public string Email
{
    get { return this.usuario.email; }
    set { this.usuario.email = value; }
}

private bool editando = false;
public bool Editando
{
    get { return editando; }
    private set { editando = value; }
}
```

Vamos rodar a app logando com "joao@alura.com.br" e "alura123", como foi feito até então, indo à página de perfil, clicando em "Perfil", com os campos bloqueados para edição.

O botão "Salvar" ainda funciona, nos levando à aba anterior, mas queremos bloqueá-lo também, exibindo-se outro botão nesta mesma página chamado "Editar", responsável por habilitar a edição. No momento, não temos o que salvar, já que nada foi alterado. Pausando-se a aplicação e indo a `MasterView.xaml`, criaremos outro botão embaixo de "Salvar".

```
<Button Text="Editar" Command="{Binding EditarPerfilCommand}"></Button>
```

Porém, temos um problema: `EditarPerfilCommand` já existe, com outra função, sendo um comando que leva à segunda aba, e que não que vai habilitar a edição dos campos. Chamaremos `SalvarPerfilCommand` de `SalvarCommand`, em:

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

Substituiremos `SalvarPerfilCommand` por `SalvarCommand` em `public ICommand SalvarCommand { get; private set; }` e, no arquivo `MasterView.xaml`, faremos o mesmo onde se encontra o botão "Salvar", alterando também o `Binding` do botão "Editar":

```
<Button Text="Salvar" Command="{Binding SalvarCommand}"></Button>
<Button Text="Editar" Command="{Binding EditarCommand}"></Button>
```

Agora precisaremos criar um comando no `ViewModel` para o novo botão ("Editar"). Voltando a `MasterViewModel.cs`, portanto, teremos:

```
public ICommand EditarPerfilCommand { get; private set; }
public ICommand SalvarCommand { get; private set; }
public ICommand EditarCommand { get; private set; }
```

Definiremos também uma instância deste `EditarCommand` algumas linhas abaixo. Como já alteramos a propriedade `IsEnabled` para amarrá-la à propriedade `Editando` do `ViewModel`, o que precisaremos fazer é simplesmente modificar a propriedade `Editando` para poder habilitá-la para edição.

```
private void DefinirComandos(Usuario usuario)
{
  //...

  EditarCommand = new Command(() =>
  {
    this.Editando = true;
  });
}
```

Feito isto, rodaremos a aplicação e veremos o que acontece. Temos ambos os botões implementados, porém quando clicamos em "Editar", os campos ainda não são habilitados, como deveria ocorrer. O formulário "não sabe" que houve uma alteração na propriedade `Editando`.

Para fazermos isto por meio do Xamarin Forms, em um padrão MVM, chamaremos um método de notificação. A `MasterViewModel` precisa ser alterada para que ela herde, de uma classe já utilizada anteriormente (o `BaseViewModel.cs`), uma interface chamada `INotifyPropertyChanged`. Com ela, conseguimos implementar um método chamado `OnPropertyChanged`, responsável por notificar a *view* de que houve mudança em uma propriedade (a ser mencionada como parâmetro).

Em `MasterViewModel`, portanto, alteraremos a linha:

```
public class MasterViewModel : BaseViewModel
```

E, algumas linhas abaixo, chamaremos o `OnPropertyChanged`, rodando a aplicação em seguida:

```
public bool Editando
{
    get { return editando = false; }
    private set
    {
        editando = value;
        OnPropertyChanged(nameof(Editando));
    }
}
```

O botão "Editar" ainda não está funcionando. O problema é que definimos o `get` sempre com atribuição de valor `false`, ou seja, impossibilitando que ele fique `true`. Deixaremos o código assim:

```
public bool Editando
{
    get { return editando; }
    private set
    {
        editando = value;
        OnPropertyChanged(nameof(Editando));
    }
}
```

Rodaremos novamente a app e, ao chegarmos no botão "Editar", desta vez os campos ficaram habilitados para edição.