09

Validação

Transcrição

Após a navegação a partir da tela de login, precisamos implementar a autenticação do usuário. Antes disto, precisamos ainda validar os dados, pois rodando-se a aplicação, temos os campos de "Usuário" e "Senha" que, mesmo sem preenchimento, permite que se clique no botão "Entrar", passando-se à tela seguinte sem nenhum problema. Ou seja, o botão "Entrar" invariavelmente aceita qualquer ação.

Precisamos validar estes campos para verificarmos se estão sendo preenchidos de fato. Como visto na parte `1` do curso, há uma maneira de validarmos o preenchimento por meio do comando `EntrarCommand`, que possui como parâmetro da classe `Command o (Action execute)`, a ser executado assim que o usuário clicar no botão. Há também um *overload* que permite a definição da habilitação do mesmo.

Portanto, podemos colocar uma função que retornará verdadeiro ou falso definindo se este botão ficará desabilitado ou não. Por padrão, esta instância do comando possui uma função interna que sempre retorna `true`, o que quer dizer que ela está habilitada em qualquer situação. O que faremos agora é implementar uma função que informa se este formulário está preenchido corretamente. Caso não seja o caso, retornaremos um valor `false` indicando que este botão precisa ficar desabilitado enquanto o formulário não for preenchido.

Então vamos inserir outro argumento deste construtor do comando, uma função anônima, em cujo corpo retornaremos `true` ou `false`. No caso, retornaremos `false`, pois queremos que este botão fique desabilitado. Vamos testar o código e ver o que acontece:

```
public ICommand EntrarCommand { get; private set; }

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

Ao rodarmos a aplicação, vemos os campos de "Usuário", "Senha" e o botão "Entrar", que a app não deixa mais clicar, não acontece nada! Ele está desabilitado pois na função que indica se ele pode ser executado ou não, estamos retornando `false`. Não importa o que se digite, o botão "Entrar" permanece desabilitado.

Vamos modificar a função anônima, pausando a aplicação e voltando ao cósigo para retornar o valor de acordo com o que for colocado ali. Em vez de retornar um valor fixo (verdadeiro ou falso), incluímos uma expressão que indica se ambos os campos estão preenchidos corretamente.

Para o campo "Usuário", é necessário que se verifique se não se trata de uma `string` vazia ou com valor nulo. Também precisamos validar a senha. Acrescentamos o operador lógico `and` copiando a linha de cima e substituindo-se `usuario` por `senha`:

```
public ICommand EntrarCommand { get; private set; }

    public LoginViewModel()
    {
      EntrarCommand = new Command(() =>
      {
        MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");
      }, () =>
      {
        return !string.IsNullOrEmpty(usuario)
                && !string.IsNullOrEmpty(senha);
        });
```

Rodamos novamente a aplicação para conferirmos seu funcionamento. Com os campos não preenchidos, o botão encontra-se desabilitado, e quando preenchemos "Usuário", o botão permanece inalterado. O mesmo ocorre ao preenchimento de "Senha". Algo está errado!

O problema é que quando os valores destes campos são modificados, o botão não é notificado sobre a mudança, portanto ele não sabe que deve ficar habilitado assim que o usuário preenche seus dados. Corrigiremos isto indo às propriedades `usuario` e `senha`. Precisamos chamar o `EntrarCommand` setando uma propriedade, o `ChangeCanExecute()`, o qual não aparece na lista de opções dadas pelo Visual Studio. `EntrarCommand` é uma interface `ICommand`.

O `ChangeCanExecute()`, que queremos utilizar neste caso, não faz parte do `ICommand`, e sim da classe `Command`. É preciso realizarmos um *cast*, uma conversão para uma classe, chamada de `Command`, com o método `ChangeCanExecute()`. Faremos o mesmo algumas linhas abaixo, quando modificamos a propriedade `senha`.

```
private string usuario;
public string Usuario
{
    get { return usuario; }
    set
    {
      usuario = value;
      ((Command)EntrarCommand).ChangeCanExecute();
    }
}

private string senha;
public string Senha
{
    get { return senha; }
    set
    {
      senha = value;
      ((Command)EntrarCommand).ChangeCanExecute();
    }
}
```

Feito isto, rodaremos a aplicação mais uma vez, e vamos ver o resultado. Neste momento, os valores estão vazios, os campos não estão preenchidos. Clicaremos em "Entrar", e vê-se que o botão está desabilitado. Preenchendo-se os campos, o botão é habilitado, permitindo-se o acesso à tela de listagem de veículos. Assim, vimos como implementar a

validação, modificando-se um comando e retornando um valor verdadeiro ou falso, indicando se o formulário está validado ou não.