11

Refatoração

Transcrição

O que faremos agora é extrair o método para deixar o construtor de `EntrarCommand` menor e mais enxuto. Selecionamos parte do código que queremos modificar, clicamos com o botão direito do mouse e selecionamos "*Quick Actions and Refactorings...*", clicando na opção "Extrair método", ou "*Extract Method*". Quando fazemos isto, aparece a opção de renomearmos o novo método, o qual chamaremos de `FazerLogin`. Temos um construtor de `LoginViewModel` mais conciso.

```
public LoginViewModel()
{
  EntrarCommand = new Command(async () =>
  {
    await FazerLogin();
    }, () =>
    {
      return !string.IsNullOrEmpty(usuario)
              && !string.IsNullOrEmpty(senha);
    });
}
```

Idealmente, seria melhor se pudessemos mover este método para uma classe especializada. No lado direito, em "*Solution Explorer*", clicaremos com o lado direito do mouse em cima de "TestDrive (Portable)", e em "*Add* > *Class*", criando a classe de nome `LoginService.cs`.

Moveremos para esta classe o método que acabamos de extrair do `LoginViewModel`. Neste arquivo, teremos que criar uma nova instância, uma declaração de uma variável `loginService`, a partir da qual faremos a chamada do `FazerLogin`:

```
public LoginViewModel()
{
  EntrarCommand = new Command(async () =>
  {
    var loginService = new LoginService();
    await loginService.FazerLogin();
    }, () =>
    {
      return !string.IsNullOrEmpty(usuario)
              && !string.IsNullOrEmpty(senha);
    });
}
```

Este método (`FazerLogin`) é privado, portanto precisamos torná-lo público, em:

```
public class LoginService
{
  public async Task FazerLogin()
  {
```

```
      ...
    }
  }
```

`"email"` e `"senha"` são valores fixos no código, então chamaremos uma variável de `email` e outra de `senha`. Neste método, passamos uma `string` chamada `email`, e outra chamada `senha`.

```
public class LoginService
{
  public async Task FazerLogin(string email, string senha)
  {
    using (var cliente = new HttpClient())
    {
      var camposFormulario = new FormUrlEncodedContent(new[]
        {
              new KeyValuePair<string, string>("email", email),
              new KeyValuePair<string, string>("senha", senha)
          });

      cliente.BaseAdress = new Uri("https://aluracar.herokuapp.com");
      await cliente.PostAsync("/login", camposFormulario);
      MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");
    }
  }
}
```

Eles estão sendo recebidos pelo `FazerLogin`, e serão repassados na chamada do post. Ao fazermos isto, na chamada do `FazerLogin` em `LoginViewModel.cs`, há um erro por não termos passado nenhum parâmetro para este método. Assim, colocaremos dois argumentos para esta chamada:

```
public LoginViewModel()
{
  EntrarCommand = new Command(async () =>
  {
    var loginService = new LoginService();
    await loginService.FazerLogin(usuario, senha);
    }, () =>
    {
      return !string.IsNullOrEmpty(usuario)
              && !string.IsNullOrEmpty(senha);
    });
}
```

Poderíamos melhorar este código refatorando a função `FazerLogin`, pois temos dois argumentos (`email` e `senha`), sempre juntos, em todo lugar da nossa aplicação. O e-mail estará sempre acompanhado da senha, e vice-versa. Em vez de passarmos dois argumentos separadamente, criaremos outra classe chamada `Login`, com estas duas propriedades.

Clicando com o lado direito do mouse em `Models` na listagem de diretórios que se encontra do lado direito da app, selecionaremos "*Add* > *Class*", nomeando esta classe de `Login`. Criaremos nela as suas propriedades. No arquivo `Login.cs`, então, teremos:

```
namespace TestDrive.Models
{
    public class Login
    {
      public string email { get; set; }
      public string senha { get; set; }
    }
}
```

Agora, iremos substituir por `Login` os locais em que aparecem `email` e `senha`, como em:

```
public async Task FazerLogin(Login login)
{
  using (var cliente = new HttpClient())
  {
    var camposFormulario = new FormUrlEncodedContent(new[]
    {
          new KeyValuePair<string, string>("email", login.email),
          new KeyValuePair<string, string>("senha", login.senha)
      });

    cliente.BaseAdress = new Uri("https://aluracar.herokuapp.com");
    await cliente.PostAsync("/login", camposFormulario);
    MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");
  }
}
```

Na chamada do método `FazerLogin` teremos que passar uma instância do objeto `Login`. Em `LoginViewModel.cs`, deletaremos `usuario, senha` substituindo-o por `new Login()`, como abaixo:

```
var loginService = new LoginService();
await loginService.FazerLogin(new Login(usuario, senha));
...
```

Faremos alterações em `Login.cs` também, criando um construtor que receberá estes parâmetros:

```
public string email { get; private set; }
public string senha { get; private set; }

public Login(string email, string senha)
{
  this.email = email;
  this.senha = senha;
}
```

Com isto, conseguimos passar ao serviço de autenticação o usuário e a senha digitados pelo cliente na tela de login. Precisamos verificar o resultado da requisição POST para sabermos se o login, a autenticação, foi bem sucedida. Abrindo o arquivo `LoginService.cs`, na chamada do `PostAsync`, armazenaremos o resultado em uma variável, chamada `resultado`.

Investigamos uma propriedade para este resultado, `IsSuccessStatusCode`, que indicará um valor boolean (verdadeiro ou falso) e, caso seja verdadeiro, obtivemos sucesso na autenticação. Se for falso, precisamos indicar que tivemos uma falha, ou seja, lançaremos outra mensagem, de outro tipo. Para isto, criaremos outra classe indicando que houve uma exceção no login (`LoginException`), que será herdada da classe `Exception` do C# do .NET, e a qual tornaremos pública.

```
public class LoginService
{
  public async Task FazerLogin(Login login)
  {
    using (var cliente = new HttpClient())
    {
      var camposFormulario = new FormUrlEncodedContent(new[]
        {
            new KeyValuePair<string, string>("email", login.email),
            new KeyValuePair<string, string>("senha", login.senha)
            });

            cliente.BaseAdress = new Uri("https://aluracar.herokuapp.com");

            var resultado = await cliente.PostAsync("/login", camposFormulario);
            if (resultado.IsSuccessStatusCode)
            MessagingCenter.Send<Usuario>(new Usuario(), "SucessoLogin");

            else
            MessagingCenter.Send<LoginException>(new LoginException("Usuário ou senha incorreto'
        }
      }
}

public class LoginException : Exception
{
  public LoginException(string message) : base(message)
  {
  }
}
```

Com o lançamento de uma nova falha, precisamos de algum meio de interceptarmos esta mensagem, e fazer algo a partir disto. Abrindo o arquivo `LoginView.xaml.cs` e acessando o *code behind* dele, neste caso, teremos um `OnAppearing` , dentro do qual assinaremos uma mensagem. Para fazê-lo, chamaremos o `MessagingCenter` , fazendo o `Subscribe` , passando-se o tipo da mensagem (`LoginException`). Nele, precisamos passar quem assina, que no caso é a própria classe, e o nome do tipo da mensagem (`"FalhaLogin"`).

O método que tratará a interceptação desta mensagem é o `DisplayAlert` e, neste caso, a mensagem que será mostrada será "Usuário ou senha incorreto", e o nome do botão será "Ok". Além disto, precisamos passar um `await` , pois queremos que ele aguarde o clique do usuário no botão "Ok". Como estamos utilizando o operador `await` , também precisamos marcar o método anônimo com a palavra chave `async` .

Tendo a assinatura desta mensagem, precisamos cancelá-la assim que a *view* deixa de aparecer na tela. Criaremos então outro `override` , ou seja, sobrescreveremos outro método, `OnDisappearing()` . Chamaremos o método `Unsubscribe` , passando-se como tipo da mensagem justamente `LoginException` que estávamos recebendo antes.

```
namespace TestDrive.Views
{
  public partial class LongView : ContentPage
  {
    public LoginView()
    {
      InitializeComponent();
    }

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

      MessagingCenter.Subscribe<LoginException>(this, "FalhaLogin",
      async (exc) =>
      {
          await DisplayAlert("Login", exc.Message, "Ok");
      });
    }

    protected override void OnDisappearing()
    {
    base.OnDisappearing();
    MessagingCenter.Unsubscribe<LoginException>(this, "FalhaLogin");
    }
  }
}
```

Deste modo, cancelamos a mensagem assim que a tela desaparece. Agora, vamos rodar a aplicação e ver se funciona nos dois casos, de sucesso e falha. Preencheremos os campos de "Usuário" e "Senha" com "joao@alura.com.br" e "alura123", respectivamente, apertando "Entrar" em seguida. Com os dados corretos, somos direcionados à página de listagem de veículos, conforme esperado.

Abrindo novamente a app, digitaremos nos mesmos campos informações quaisquer. Abre-se uma janela com a mensagem "Login. Usuário ou senha incorreto.", e um "OK" embaixo. Ou seja, conseguimos realizar a implementação da autenticação com sucesso.