04

Logout e o escopo Flash

Começando daqui? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/JSF/livraria-capitulo13.zip) do projeto completo do capítulo anterior e continuar seus estudos a partir deste capítulo.

Com a autenticação e autorização do usuário completas, o objetivo agora é realizar o **logout** do usuário.

Realizando o logout do usuário

A primeira coisa que devemos fazer é criar um botão para essa tarefa, como é um botão que terá que aparecer em todas as páginas, o colocaremos no nosso `_template.xhtml`, logo abaixo do logo. Como é um `commandLink`, ele precisa estar dentro de um formulário:

```
<!-- restante do código -->
<h:body>
    <div id="cabecalho">
        <h:graphicImage library="img" name="logo.png" />

        <h:form>
            <h:commandLink value="Logout" action="#{loginBean.deslogar}" />
        </h:form>
        <h1>
            <ui:insert name="titulo"></ui:insert>
        </h1>
    </div>
    <!-- restante do código -->
</h:body>
```

O `LoginBean` já está criado, falta implementar o método `deslogar`. Mas como implementá-lo? Primeiramente, o método deve redirecionar o usuário para a página de login, logo o retorno dele será `"login?faces-redirect=true"`. E para realizar o logout do usuário, basta remover a chave `usuarioLogado` do `sessionMap`, logo, sem a chave na sessão, o nosso autorizador não permitirá o acesso às páginas do sistema, para isso o usuário precisará realizar login novamente. Ou seja:

```
public String deslogar() {

    FacesContext context = FacesContext.getCurrentInstance();
    context.getExternalContext().getSessionMap().remove("usuarioLogado");

    return "login?faces-redirect=true";
}
```

Podemos testar agora, vamos realizar o login e logo em seguida o logout. Depois, se tentarmos acessar as páginas da aplicação, não conseguimos! Logo o logout foi realizado com sucesso.

Porém, repare uma coisa, o botão de logout aparece na página de login, e isso não é o ideal, já que não tem como o usuário realizar o logout se ele ainda nem fez o login, precisamos mudar isso, não queremos que esse botão apareça na página de login.

Mas o botão está no template, e o template está sendo utilizado por `login.xhtml`, então como alterar isso? Para essas situações, podemos usar o atributo `rendered`, definindo uma condição para quando o formulário do botão de logout será renderizado. A nossa condição é que esse formulário apareça somente quando o `usuarioLogado` **não** for nulo, então vamos deixar isso explícito:

```
<!-- restante do código -->
<h:body>
    <div id="cabecalho">
        <h:graphicImage library="img" name="logo.png" />

        <h:form rendered="#{usuarioLogado != null}">
            <h:commandLink value="Logout" action="#{loginBean.deslogar}" />
        </h:form>
        <h1>
            <ui:insert name="titulo"></ui:insert>
        </h1>
    </div>
    <!-- restante do código -->
</h:body>
```

Podemos testar mais uma vez e ver que tudo está funcionando como o esperado!

Mensagens de validação

O último detalhe que falta na nossa aplicação está referente ao formulário de login. Se clicarmos no botão "Efetuar Login", mas com os campos "Email" e "Senha" em branco, aparecem mensagens de validação, dizendo o motivo que o login não foi realizado.

Porém, se tentarmos realizar login com uma senha incorreta, por exemplo, nada aparece! O usuário fica sem saber o que realmente aconteceu. Precisamos melhorar isso, precisamos informar o usuário para o mesmo saber o que ele fez de errado.

Para melhorar isso, precisamos mudar o método `efetuaLogin`, da classe `LoginBean`. Como queremos mostrar mensagens de validação, de erro, precisamos colocá-las após o `if`, pois o código após o `if` só será executado se o usuário não existir no banco.

A primeira coisa que iremos mudar é o retorno do método, ele não retornará mais `null`, vamos deixar explícito para onde o usuário deve ser redirecionado caso o login dê falha, que é a página de login.

Agora, para exibir uma mensagem para o usuário, utilizaremos o já conhecido `facesContext`, chamando o seu método `addMessage`, que recebe dois parâmetros. O primeiro parâmetro é o nome do componente. Se você está criando uma mensagem e quer associá-la a um componente específico, ele precisa ficar explicitado aqui. Porém, não é o nosso caso, queremos uma mensagem global, relacionada ao formulário, então deixaremos o valor `null`; o segundo parâmetro é a mensagem em si, que é uma `FacesMessage`:

```
public String efetuaLogin() {
    System.out.println("Fazendo login do usuário "
            + this.usuario.getEmail());

    FacesContext context = FacesContext.getCurrentInstance();
    boolean existe = new UsuarioDao().existe(this.usuario);
```

```
    if (existe) {

        context.getExternalContext().getSessionMap()
                .put("usuarioLogado", this.usuario);

        return "livro?faces-redirect=true";
    }

    context.addMessage(null, new FacesMessage("Usuário não encontrado"));

    return "login";
}
```

Mas não basta só isso, agora precisamos alterar o formulário de login para exibir essa mensagem, e o componente do JSF que a exibe é o componente `h:message`. Vamos adicioná-lo **antes** do formulário de login:

```
<!-- restante do código -->

<ui:define name="conteudo">

    <h:messages />

        <h:form id="login">
<!-- restante do código -->
```

Podemos fazer agora o teste de tentar fazer o login com um usuário de senha errada... Ótimo, recebemos a mensagem. Mas e se realizarmos o primeiro teste que fizemos neste capítulo, se deixarmos os campos de e-mail e senha em branco, e tentarmos fazer o login... As mensagens também aparecem acima do formulário.

Não é isso que queremos, queremos que as mensagens específicas apareçam ao lado dos seus componentes específicos, e as mensagens globais acima do formulário. Para resolver isso é muito fácil, basta adicionar o atributo `globalOnly` no componente das mensagens e deixá-lo ativo, `true`:

```
<!-- restante do código -->

<ui:define name="conteudo">

    <h:messages globalOnly="true" />

        <h:form id="login">
<!-- restante do código -->
```

Ótimo! Agora conseguimos realizar tudo o que queríamos, vamos mudar apenas mais um detalhe. O correto, a boa prática, é fazer um redirecionamento após submeter um formulário, para limpar os dados da requisição, se não fizermos isso, um outro usuário poderia repetir a requisição, ou ver os dados da mesma, como e-mail e senha. Vamos então fazer um redirecionamento para a página de login, caso o usuário faça login incorretamente:

```
public String efetuaLogin() {
    System.out.println("Fazendo login do usuário "
            + this.usuario.getEmail());
```

```
        FacesContext context = FacesContext.getCurrentInstance();
        boolean existe = new UsuarioDao().existe(this.usuario);

        if (existe) {

            context.getExternalContext().getSessionMap()
                    .put("usuarioLogado", this.usuario);

            return "livro?faces-redirect=true";
        }

        context.addMessage(null, new FacesMessage("Usuário não encontrado"));

        return "login?faces-redirect=true";
    }
```

Mas com isso criamos um outro problema, as mensagens globais de validação não são mais exibidas! Acontece que o `FacesContext` só existe em uma requisição, e com o `faces-redirect` estamos fazendo uma segunda requisição, então a mensagem não existe mais.

Porém, temos como resolver isso com o JSF 2, para isso temos que acessar novamente o `externalContext` e nele há um escopo chamado `flash`. Esse escopo dura duas requisições, exatamente o que precisamos. Mas para quê queremos usar o `flash`? Para manter as mensagens. Então chamaremos mais um método, o `setKeepMessages`, passando `true` como parâmetro, isso quer dizer que as mensagens que estamos adicionando no `flash` serão mantidas por duas requisições:

```
    public String efetuaLogin() {
        System.out.println("Fazendo login do usuário "
                + this.usuario.getEmail());

        FacesContext context = FacesContext.getCurrentInstance();
        boolean existe = new UsuarioDao().existe(this.usuario);

        if (existe) {

            context.getExternalContext().getSessionMap()
                    .put("usuarioLogado", this.usuario);

            return "livro?faces-redirect=true";
        }

        context.getExternalContext().getFlash().setKeepMessages(true);
        context.addMessage(null, new FacesMessage("Usuário não encontrado"));

        return "login?faces-redirect=true";
    }
```

Agora, mesmo com o redirecionamento, as mensagens são mantidas para mais uma requisição. Com isso completamos a nossa aplicação!

O que aprendemos:

- Definimos condição com o atributo `rendered`;

- Relembramos o `FacesMessage` ;
- Utilizamos o escopo `flash` do JSF 2;