03

Autorização pelo PhaseListener

Começando daqui? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/JSF/livraria-capitulo12.zip) do projeto completo do capítulo anterior e continuar seus estudos a partir deste capítulo.

No capítulo anterior, foi falado sobre a **autenticação do usuário**, como fazer o seu login. Mas nós só verificamos se o usuário estava correto ou não, o problema ainda é que podemos acessar livremente qualquer página da nossa aplicação através da sua URL específica, como localhost:8080/livraria/livro.xhtml ou localhost:8080/livraria/autor.xhtml, ou seja, atualmente não é necessário estar logado para acessar as páginas da nossa aplicação. Precisamos fechar o acesso às nossa páginas, somente um usuário com login feito que pode acessá-las, precisamos fazer a **autorização do usuário**, e é isso que faremos nesse capítulo.

O PhaseListener

Precisamos verificar cada requisição para nossa aplicação. Ou seja, antes de fazer qualquer coisa devemos checar se o usuário já fez login, se sim, ele pode continuar, se não, o usuário será redirecionado para a página de login. Se você já assistiu algum treinamento sobre Servlets, Spring MVC ou VRaptor, por exemplo, já usou um recurso para essa autorização do usuário, esse recurso é o **filtro**, ou **interceptador**. Precisamos filtrar, interceptar **todas** as requisições, para checar se o usuário já fez login.

Só que nós estamos no mundo JSF, então vamos focar no que o JSF nos oferece. Vamos utilizar um recurso já visto para filtrar as requisições, o **PhaseListener**, nada nos impede de utilizá-lo para algo mais sofisticado, como a autorização do usuário.

Implementando o nosso Autorizador

Nós já utilizamos o `PhaseListener` uma vez, na classe `LogPhaseListener`, então vamos copiá-la, mas mudando o seu nome, a nova classe se chamará `Autorizador`. Mas um `PhaseListener` precisa ser cadastrado no arquivo `WebContent/WEB-INF/faces-config.xml`, então adicione dentro da tag `<lifecycle>`:

```
<phase-listener>br.com.caelum.livraria.util.Autorizador</phase-listener>
```

Pronto, agora podemos focar no `Autorizador`. O `PhaseListener` anterior foi configurado para rodar em cada fase, mas no `Autorizador` queremos que ele rode apenas na primeira fase, a fase `RESTORE_VIEW`, pois é no início que iremos verificar se o usuário existe ou não.

Precisamos do nome da página pare verificar se ela é protegida ou não, e para isso é preciso que tenhamos em mãos a árvore de componentes. Para ter essa árvore de componentes, precisamos executar o `Autorizador` **depois** da fase, logo implementaremos o método `afterPhase`. A classe `Autorizador` ficará assim:

```
public class Autorizador implements PhaseListener {

    private static final long serialVersionUID = 1L;

    @Override
    public void afterPhase(PhaseEvent event) {
    }
```

```
    @Override
    public void beforePhase(PhaseEvent event) {
    }

    @Override
    public PhaseId getPhaseId() {
        return PhaseId.RESTORE_VIEW;
    }
}
```

O primeiro passo é recuperar a árvore programaticamente, porque queremos saber o nome da página. Através do evento, pegamos o `facesContext`, que já conhecemos, ele é criado a cada requisição. E através desse `facesContext` que pegamos o elemento raiz da árvore de componentes, que programaticamente se dá através do método `getViewRoot()`, e através desse `viewRoot` que pegamos o nome da página, através do método `getViewId()`. Depois vamos imprimir esse nome para ver se está correto:

```
@Override
public void afterPhase(PhaseEvent event) {
    FacesContext context = event.getFacesContext();
    String nomePagina = context.getViewRoot().getViewId();

    System.out.println(nomePagina);
}
```

Mas porque queremos saber o nome da página? Porque há uma página que não queremos fazer a autorização, a página `login.xhtml`. Se o usuário está acessando essa página, ele está querendo se autenticar, então devemos deixá-lo continuar:

```
@Override
public void afterPhase(PhaseEvent event) {
    FacesContext context = event.getFacesContext();
    String nomePagina = context.getViewRoot().getViewId();

    System.out.println(nomePagina);

    if ("/login.xhtml".equals(nomePagina)) {
        return;
    }
}
```

O `return` significa que não iremos fazer nada e que o código continuará normalmente com as fases seguintes. Se o código não entrar no `if`, significa que o usuário está tentando acessar uma outra página da nossa aplicação, então precisamos fazer uma verificação de acesso.

Como saber se o usuário fez login?

Na classe `LoginBean`, no método `efetuaLogin`, verificamos em um `if` se o usuário existe, então a ideia é que se o usuário existe, guardamos a informação em algum lugar, para depois poder testar nas próximas requisições se o usuário se autenticou ou não. Então é dentro desse `if` que guardaremos essa informação, mas como?

Geralmente, em uma aplicação web, usamos a **sessão HTTP** para guardar essa informação sobre o usuário, e aqui não será diferente. Mas como acessar essa sessão? Primeiro pegamos a instância do `facesContext`, e como já vimos anteriormente, pegamos-o em qualquer lugar através do método `getCurrentInstance()`.

Com o `context` em mãos, podemos acessar objetos que rodam ao nível das servlets, através do método `getExternalContext()`. E com isso podemos pegar a sessão HTTP, utilizando o método `getSessionMap()`. A sessão HTTP na verdade é um mapa que podemos guardar dados através da relação chave-valor.

Então, na *session map*, adicionaremos através do método `put`, uma chave **usuarioLogado**, e o seu valor, que será o usuário que fez o login:

```
public class LoginBean {

    // restante do código

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

        FacesContext context = FacesContext.getCurrentInstance();

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

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

        return null;
    }
}
```

Pronto! Então se o usuário existe banco, se ele se autenticou, guardamos as informações dele numa chave "usuarioLogado".

Agora precisamos acessar essas informações no `Autorizador`. Recuperamos as informações com quase o mesmo código, só que ao invés de fazer um `put`, fazemos um `get` na chave. E se esse usuário logado existir, ou seja, se ele não é nulo, então sabemos que ele realmente fez login e assim ele pode continuar:

```
@Override
public void afterPhase(PhaseEvent event) {

    FacesContext context = event.getFacesContext();
    String nomePagina = context.getViewRoot().getViewId();

    System.out.println(nomePagina);

    if ("/login.xhtml".equals(nomePagina)) {
        return;
    }

    Usuario usuarioLogado = (Usuario) context.getExternalContext()
            .getSessionMap().get("usuarioLogado");
```

```
        if(usuarioLogado != null) {
            return;
        }
    }
```

Mas se chegamos ao final do código, significa que o usuário não acessou a página de login, e ele não se autenticou, ou seja, devemos redirecioná-lo para a página de login. Para isso, precisamos fazer uma **navegação programaticamente** com o JSF, e para isso o JSF possui um objeto específico para fazer essa navegação, o `NavigationHandler`.

Esse navegador vem do nosso `context`, mas antes temos que pegar os dados da aplicação, através do método `getApplication()`, para aí acessá-lo, chamando o método `getNavigationHandler()`. E a partir desse `handler` que chamamos o método `handleNavigation`, que recebe três parâmetros. O primeiro parâmetro é o `context`, que já temos; o segundo parâmetro seria o nome da nossa página caso a mesma estivesse cadastrada no `faces-config.xml`, mas como não utilizamos isso, utilizamos a string completa de redirecionamento da página, passaremos o valor `null`; e o terceiro é a página para onde queremos ir, e queremos ir para `login.xhtml`, logo `"/login?faces-redirect=true"`.

Por fim, não basta definir somente a navegação, também precisamos renderizar a resposta, através do método `renderResponse()` do `context`, isso quer dizer que o JSF pulará todas as fases e irá logo para a último, para renderizar a resposta:

```
@Override
public void afterPhase(PhaseEvent event) {

    FacesContext context = event.getFacesContext();
    String nomePagina = context.getViewRoot().getViewId();

    System.out.println(nomePagina);

    if ("/login.xhtml".equals(nomePagina)) {
        return;
    }

    Usuario usuarioLogado = (Usuario) context.getExternalContext()
            .getSessionMap().get("usuarioLogado");

    if(usuarioLogado != null) {
        return;
    }

    NavigationHandler handler = context.getApplication().getNavigationHandler();
    handler.handleNavigation(context, null, "/login?faces-redirect=true");

    context.renderResponse();
}
```

Agora o nosso `Autorizador` está pronto! Podemos testá-lo. Se acessamos a página `livro.xhtml`, somos redirecionados para a página de login! E a mesma coisa acontece caso acessemos a página `autor.xhtml`. Se fizermos login, conseguimos acessar as duas páginas! O que falta agora é fazer um botão para realizar o logout do usuário, e é isso que faremos no próximo capítulo.

O que aprendemos

- Adquirir a `root view` ;
- Mais utilizações do `PhaseListener` ;
- Salvando dados na sessão HTTP;
- Navegação automática;