02

Criando uma autenticação de usuários

Nossa aplicação atualmente permite a todos `adicionar`, `remover` e `listar` produtos. Mas em uma aplicação real isso não deveria ser possível! É importante para nossa regra de negócio que apenas os usuários autorizados e que estejam logados em nosso sistema possam fazer qualquer consulta ou modificação em nosso estoque. Para que isso seja possível vamos começar esse trabalho fornecendo um meio de o usuário se logar em nossa aplicação.

Criando um formulário de Login

Vamos criar a classe `LoginController` no pacote `br.com.caelum.vraptor.controller` que será a responsável por disponibilizar uma página de login e também terá a lógica de autenticação da nossa aplicação. A princípio, vamos apenas adicionar o método `formulario`:

```
@Controller
public class LoginController {

@Get
public void formulario() {
}
```

Precisamos agora criar o `formulario.jsp` para qual esse método será mapeado. Lembrando que esse formulario deve estar dentro da pasta `WEB-INF/jsp/login/`:

```
<html>
<link rel="stylesheet" type="text/css" href="../bootstrap/css/bootstrap.css">

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Formulário de Login</title>
</head>
<body>
<div class="container">
 <form class="form-signin" action="<c:url value='/login/autentica'/>" method="post">
  <h2 class="form-signin-heading">Faça login para acessar o VRaptor-Produtos</h2>
  <input type="text" class="form-control" name="usuario.nome" placeholder="Nome"/>
  <input type="password" class="form-control" name="usuario.senha" placeholder="Senha"/>
  <button class="btn btn-lg btn-primary btn-block" type="submit">Login</button>
 </form>
</div>
</body>
</html>
```

Note que para facilitar o trabalho já estilizamos esse formulário utilizando o *twitter bootstrap*.

Fazendo a autenticação de usuários

Agora precisamos criar o método de autorização de acesso dos usuários. Esse método deve validar o `nome` e a `senha` enviado pelo formulário, se tudo der certo vamos encaminhar a requisição para a listagem de produtos, caso contrário

validar os dados e reenchaminhar para a tela de login. Já deixamos criadas as classes `Usuario` e `UsuarioDao` em nosso projeto. Só precisamos agora criar o método `autentica` que executará nossa regra de autenticação. Ele deve receber um `Usuario` como parâmetro, e validar se esse usuário é cadastrado chamando o método `existe` da classe `UsuarioDao`:

```
@Post
public void autentica(Usuario usuario) {
    if(!dao.existe(usuario)){
        // se não existe, vamos redirecionar para página
    }
    // ok, podemos encaminhar para a listagem de produtos
}
```

Vamos utilizar a classe `Validator` do VRaptor para nos auxiliar a incluir a mensagem de validação e redirecionar para o formulário. E também precisaremos do `Result` para redirecionar o usuário para a listagem caso não exista nenhum erro de validação. Pedindo esses objetos injetados, assim como fizemos nos outros capítulos, o nosso código da classe `LoginController` deve ficar assim:

```
@Controller
public class LoginController {

    private final UsuarioDao dao;
    private final Result result;
    private final Validator validator;

    @Inject
    public LoginController(UsuarioDao dao, Result result,
            Validator validator) {
        this.dao = dao;
        this.result = result;
        this.validator = validator;
    }

    @Deprecated
    LoginController() {
        this(null, null, null, null); // para uso do CDI
    }

    @Get
    public void formulario() {
    }

    @Post
    public void autentica(Usuario usuario) {
        if(!dao.existe(usuario)){
            validator.add(new I18nMessage("login", "login.invalido"));
            validator.onErrorUsePageOf(this).formulario();
        }
        result.redirectTo(ProdutoController.class).lista();
    }
}
```

Como estamos usando o `I18nMessage`, precisamos adicionar a chave `login.invalido` no arquivo `messages.properties`:

```
// outras mensagens omitidas
login.invalido = Nome ou senha inválido
```

E para exibir essa mensagem, vamos adicionar o EL `errors` após a tag `form` do `formulario.jsp` :

```
<!-- inicio do jsp omitido -->
    </form>

    <c:if test="${not empty errors}">
        <div class="alert alert-danger">
            <c:forEach var="error" items="${errors}">
                ${error.category} - ${error.message}<br />
            </c:forEach>
        </div>
    </c:if>
</div>
```

Agora basta reiniciar o servidor e acessar a url do nosso formulário: http://localhost:8080/vraptor-produtos/login/formulario (http://localhost:8080/vraptor-produtos/login/formulario). Tente logar com um usuário que não existe para ver que a validação está funcionando, e logo em seguida com um usuário existente (pode ser o **vraptor**, que utilizamos no vídeo) para que você seja redirecionado para a lista de produtos.

Adicionando o usuário logado na sessão

Da forma que está, o usuário está sendo autenticado e logo em seguida perdemos essa informação. Como podemos fazer para que a aplicação se lembre que esse usuário já fez o login em nosso sistema? Precisamos guardar essa informação em algum objeto que viva por mais tempo que uma *request*, ou seja, um objeto com escopo maior.

Vamos criar a classe `UsuarioLogado` , e determinar o escopo dela como `@SessionScoped` . Assim, cada sessão de usuário terá uma classe dessa mantendo suas informações em memória.

```
@SessionScoped
@Named
public class UsuarioLogado {

    private Usuario usuario;

    public Usuario getUsuario() {
        return usuario;
    }

    public void setUsuario(Usuario usuario) {
        this.usuario = usuario;
    }
}
```

Note que além da anotação de escopo, utilizamos a anotação `@Named` . Essa é uma anotação do CDI utilizada para disponibilizar objetos na jsp. Dessa forma, não precisamos fazer o `result.include(...)` dessa classe para que ela fique acessível em qualquer página do nosso sistema!

O último passo é `settar` o usuário na classe `UsuarioLogado` quando ele for autenticado:

```
@Post
public void autentica(Usuario usuario) {
    if(!dao.existe(usuario)){
        validator.add(new I18nMessage("login", "login.invalido"));
        validator.onErrorUsePageOf(this).formulario();
    }
    usuarioLogado.setUsuario(usuario);
    result.redirectTo(ProdutoController.class).lista();
}
```

Para garantir que o usuário está sendo mantido na sessão do navegador, vamos modificar a `lista.jsp` para exibir o nome desse usuário quando ele estiver logado. Basta modificar o texto da tag `h1` da listagem, para:

```
<h1>Listagem de Produtos do ${usuarioLogado.usuario.nome}</h1>
```

Excelente, vamos tentar reiniciar o servidor agora. Note que recebemos a seguinte exception:

```
org.jboss.weld.exceptions.DeploymentException: WELD-000072:
Bean declaring a passivating scope must be passivation capable.
Bean: Managed Bean [class br.com.caelum.vraptor.model.UsuarioLogado]
```

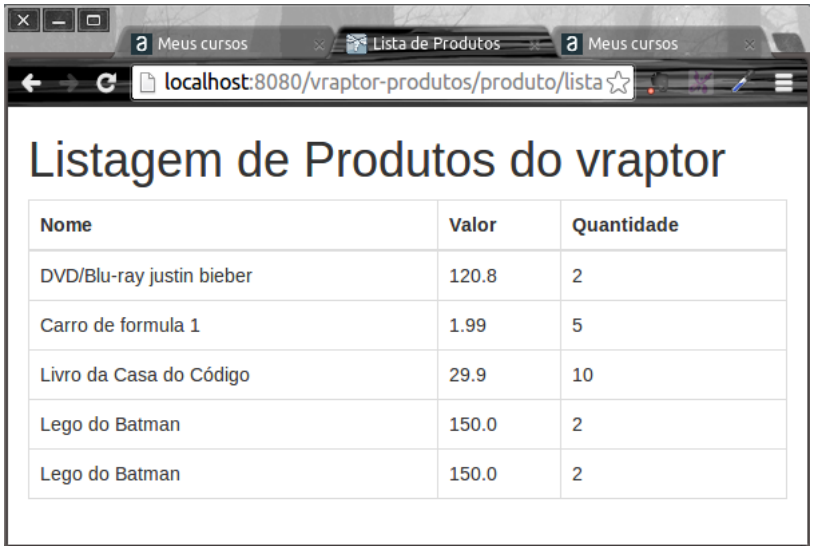
Isso aconteceu pois o CDI determina que sempre que você anota uma classe com `@SessionScoped`, essa classe precisa ser serializável, ou seja, implementar `Serializable`. Vamos modificar nossa classe `UsuarioLogado` para atender essa condição:

```
@SessionScoped
@Named
public class UsuarioLogado implements Serializable {
    // restante do código omitido
}
```

Agora sim podemos restartar o tomcat e acessar a página da lista de produtos:

- uma vez antes de logar no sistema (o nome do usuário não estará disponível)
- e mais uma vez logo após se autenticar (agora sim, o nome deve estar na lista)

Note que os dados do usuário logado estão sendo mantidos na sessão e disponibilizados na jsp. Mesmo quando fazemos varios *refresh*s na página!


Meus cursos
Lista de Produtos
Meus cursos
localhost:8080/vraptor-produtos/produto/lista
Listagem de Produtos do vraptor
Nome
Valor
Quantidade
DVD/Blu-ray justin bieber
120.8
2
Carro de formula 1
1.99
5
Livro da Casa do Código
29.9
10
Lego do Batman
150.0
2
Lego do Batman
150.0
2