02

Autenticação e Autorização

Agora que já temos um cadastro de usuários pronto, vamos desenvolver as duas fases da segurança da aplicação, a autenticação e a autorização.

Autenticação

Para desenvolvermos a lógica de autenticação, precisaremos de uma nova tela na aplicação que mostrará o formulário de login, essa tela será gerada por um novo controller chamado `LoginController` em seu método `form` :

```
@Controller
public class LoginController {

    public void form(){}
}
```

Na view desse método (`WEB-INF/jsp/login/form.jsp`) queremos gerar o html para o formulário de login da aplicação:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib tagdir="/WEB-INF/tags" prefix="alura" %>

<c:import url="/WEB-INF/jsp/header.jsp"/>

<form action="${linkTo[LoginController].login(null, null)}" method="post">
    <alura:validationMessage name="login_invalido"/>

    <label for="login">Login:</label>
    <input type="text" id="login" name="login" class="form-control"/>

    <label for="senha">Senha:</label>
    <input type="password" id="senha" name="senha" class="form-control"/>

    <input type="submit" value="Autenticar" class="btn"/>
</form>

<c:import url="/WEB-INF/jsp/footer.jsp"/>
```

Agora precisamos criar o método `login` na classe `LoginController` recebendo o login e a senha que são enviados pelo formulário. Dentro desse método utilizaremos o `UsuarioDAO` para verificar se os dados enviados pelo formulário de login estão de acordo com os cadastrados no banco de dados. Para realizar essa busca, criaremos um novo método dentro do `UsuarioDAO` chamado `busca` :

```
public Usuario busca(String login, String senha){
    String jpql = "select u from Usuario u where u.login = :login and u.senha = :senha";
    TypedQuery<Usuario> query = manager.createQuery(jpql, Usuario.class);
    query.setParameter("login", login);
    query.setParameter("senha", senha);
```

```
        return query.getSingleResult();
    }
```

Se o login for realizado com sucesso, queremos armazenar o usuário logado dentro de um componente `@SessionScoped` :

```
@SessionScoped
@Named
public class UsuarioLogado implements Serializable{

    private Usuario usuario;

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

    public void desloga(){
        this.usuario = null;
    }

    public Usuario getUsuario() {
        return this.usuario;
    }

    public boolean isLogado() {
        return this.usuario != null;
    }
}
```

Como o `Usuario` será colocado dentro de um componente `@SessionScoped` , ele também precisa implementar a interface `Serializable` do java:

```
@Entity
public class Usuario implements Serializable{
    // implementação da classe
}
```

Agora vamos utilizar esse novo método do DAO dentro do `LoginController` :

```
@Controller
public class LoginController {

    private UsuarioLogado usuarioLogado;
    private UsuarioDao usuarioDao;
    private Validator validator;
    private Result result;

    @Inject
    public LoginController(UsuarioLogado usuarioLogado, UsuarioDao usuarioDao,
            Validator validator, Result result) {
        this.usuarioLogado = usuarioLogado;
        this.usuarioDao = usuarioDao;
        this.validator = validator;
```

```
        this.result = result;
    }
    public LoginController() {
    }

    public void form(){}

    @Post
    public void login(String login, String senha){
        Usuario usuario = usuarioDao.busca(login, senha);
        if(usuario != null){
            usuarioLogado.fazLogin(usuario);
            result.redirectTo(IndexController.class).index();
        }else{
            validator.add(new SimpleMessage("login_invalid", "Login ou senha incorretos"));
            validator.onErrorRedirectTo(this).form();
        }
    }

    public void desloga(){
        usuarioLogado.desloga();
        result.redirectTo(this).form();
    }
}
```

Autorização

Para fazer a autorização na aplicação, podemos utilizar um interceptor do VRaptor. Nesse interceptor, se o usuário estiver logado, queremos executar normalmente o código do controller, senão, faremos um redirecionamento para o login da aplicação.

```
@Intercepts
public class AutorizacaoInterceptor {

    private UsuarioLogado usuarioLogado;
    private Result result;

    public AutorizacaoInterceptor(){}

    @Inject
    public AutorizacaoInterceptor(UsuarioLogado usuarioLogado, Result result){
        this.usuarioLogado = usuarioLogado;
        this.result = result;
    }

    @AroundCall
    public void intercept(SimpleInterceptorStack stack){
        if(usuarioLogado.isLogado()){
            stack.next();
        }else{
            result.redirectTo(LoginController.class).form();
        }
    }
}
```

Mas se executarmos a aplicação, como na primeira requisição o usuário ainda não está logado, ao entrarmos em qualquer página, seremos redirecionados para o formulário de login (`LoginController` método `form`), mas a requisição para o formulário de autenticação também é protegida pelo interceptor, logo como o usuário ainda não está logado, ele será redirecionado novamente para o formulário de login. Veja que temos um loop de redirecionamentos na aplicação.

Para resolvermos esse problema, podemos marcar quais são os métodos que não serão protegidos pelo interceptor. Para fazer essa marcação, podemos criar uma nova anotação do java.

Para criarmos uma anotação no java, utilizamos a palavra reservada `@interface` :

```
public @interface Open {
}
```

Ao criar a anotação precisamos falar em quais elementos de um programa java ela pode ser utilizada, fazemos isso com a anotação @Target. Como queremos que essa anotação marque métodos, o código ficará da seguinte forma:

```
@Target(ElementType.METHOD)
public @interface Open {
}
```

As anotações do java normalmente não são acessíveis em tempo de execução, elas são descartadas pelo compilador, para indicarmos que a anotação deve sobrever durante a execução do código, utilizamos o `@Retention(RetentionPolicy.RUNTIME)` :

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Open {
}
```

Agora no código do interceptor, precisamos indicar que ele só será executado se o método não estiver marcado com a anotação `@Open` . Para isso podemos criar um método marcado com @Accepts na classe do interceptor:

```
@Accepts
public boolean accepts(){
    // implementação
}
```

Se esse método devolver `true` , o VRaptor executará o código do interceptor, se não o interceptor será ignorado. Para sabermos qual é o método que será invocado na requisição atual, utilizaremos um novo componente do VRaptor chamado `ControllerMethod` :

```
@Intercepts
public class AutorizacaoInterceptor {
    private ControllerMethod controllerMethod;
    private UsuarioLogado usuarioLogado;
    private Result result;

    public AutorizacaoInterceptor(){}
```

```
    @Inject
    public AutorizacaoInterceptor(ControllerMethod controllerMethod,
            UsuarioLogado usuarioLogado, Result result){
        this.controllerMethod = controllerMethod;
        this.usuarioLogado = usuarioLogado;
        this.result = result;
    }
    // Implementação da classe
}
```

Com o `ControllerMethod`, conseguimos saber se o método do controller que será invocado pelo VRaptor possui uma anotação utilizando o método `containsAnnotation`:

```
@Intercepts
public class AutorizacaoInterceptor {

    private ControllerMethod controllerMethod;
    private UsuarioLogado usuarioLogado;
    private Result result;

    public AutorizacaoInterceptor(){}

    @Inject
    public AutorizacaoInterceptor(ControllerMethod controllerMethod, UsuarioLogado usuarioLogad
        this.controllerMethod = controllerMethod;
        this.usuarioLogado = usuarioLogado;
        this.result = result;
    }

    @Accepts
    public boolean accepts(){
        return !controllerMethod.containsAnnotation(Open.class);
    }

    @AroundCall
    public void intercept(SimpleInterceptorStack stack){
        if(usuarioLogado.isLogado()){
            stack.next();
        }else{
            result.redirectTo(LoginController.class).form();
        }
    }
}
```

E agora, para resolvermos o problema do loop de redirecionamentos, precisamos apenas adicionar a anotação @Open nos métodos do `LoginController`:

```
@Controller
public class LoginController{
    @Open
    public void form(){}

    @Open
```

```
    public void login(String login, String senha){
        // implementação
    }

    @Open
    public void desloga(){
        // implementação
    }
}
```

Depois disso, podemos testar a aplicação tentando acessar, por exemplo, a página da lista de usuários, se tudo der certo, o interceptor nos redirecionará para o formulário de login da aplicação.