08

Mãos na massa: Spring Security

Chegou a hora de você pôr em prática o que foi visto na aula. Para isso, execute os passos listados abaixo.

1) Para criar um login e senha usando o **Spring Security**, primeiramente adicione as seguintes dependências ao seu **pom.xml**:

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>4.0.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>4.0.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>4.0.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
    <version>4.0.0.RELEASE</version>
</dependency>
```

2) Para configurar o **Spring Security**, crie a classe `SpringSecurityFilterConfiguration` no pacote `br.com.casadocodigo.loja.conf`:

```
public class SpringSecurityFilterConfiguration
    extends AbstractSecurityWebApplicationInitializer {

    // a classe ficará vazia
}
```

3) Também no pacote `br.com.casadocodigo.loja.conf`, crie a classe `SecurityConfiguration`, anotando-a com `@EnableWebSecurity` (a anotação `@EnableWebMvcSecurity` foi descontinuada):

```
@EnableWebSecurity
public class SecurityConfiguration
    extends WebSecurityConfigurerAdapter {

}
```

4) Adicione as classe `SecurityConfiguration`, `AppWebConfiguration` e `JPAConfiguration` no método `getRootConfigClasses`, da classe `ServletSpringMVC`:

```
@Override
protected Class<?>[] getRootConfigClasses() {
    return new Class[] {SecurityConfiguration.class,
        AppWebConfiguration.class, JPAConfiguration.class};
}
```

5) Ainda na classe `ServletSpringMVC`, deixe o array de classes do método `getServletConfigClasses` vazio:

```
@Override
protected Class<?>[] getServletConfigClasses() {
    return new Class[] {};
}
```

6) Habilite as páginas públicas. Para isso, sobrescreva o método `configure` da classe `SecurityConfiguration`, para que o acesso a *home*, detalhes do produto e carrinho estejam sempre liberados. Adicione também a regra para liberar o acesso à pasta **resources**. Sem essa liberação, os arquivos CSS e imagens não poderão ser carregados. O mesmo aplica se para a URL de pagamentos:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers("/produtos/form").hasRole("ADMIN")
        .antMatchers("/carrinho/**").permitAll()
        .antMatchers("/produtos/").hasRole("ADMIN")
        .antMatchers("/produtos/**").permitAll()
        .antMatchers("/resources/**").permitAll()
        .antMatchers("/pagamento/**").permitAll()
        .antMatchers("/").permitAll()
        .anyRequest().authenticated()
        .and().formLogin();
}
```

7) Para poder conhecer e trabalhar o usuário, prepare o seu modelo. Para tal, crie as classes `Usuario` e `Role` no pacote `br.com.casadocodigo.loja.models`. O usuário terá um e-mail, senha e nome, além de implementar a interface `UserDetails` do Spring Security. A classe `Role` terá apenas um nome e implementa a interface `GrantedAuthority`. O `Usuario` e `Role` devem se associar (`List<Role>`) para definir as permissões. Use as anotações do JPA para mapear as regras de integridade:

```
@Entity
public class Role implements GrantedAuthority {

    private static final long serialVersionUID = 1L;

    @Id
    private String nome;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
```

```
        this.nome = nome;
    }

    @Override
    public String getAuthority() {
        return this.nome;
    }
}
```

E a classe `Usuario`:

```
@Entity
public class Usuario implements UserDetails {

    private static final long serialVersionUID = 1L;

    @Id
    private String email;
    private String senha;
    private String nome;

    @OneToMany(fetch=FetchType.EAGER)
    private List<Role> roles = new ArrayList<>();

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getSenha() {
        return senha;
    }

    public void setSenha(String senha) {
        this.senha = senha;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public List<Role> getRoles() {
        return roles;
    }

    public void setRoles(List<Role> roles) {
        this.roles = roles;
    }
```

```
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return this.roles;
    }

    @Override
    public String getPassword() {
        return this.senha;
    }

    @Override
    public String getUsername() {
        return this.email;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

8) Agora, crie a classe `UsuarioDAO`, anotando-a com `@Repository` e implementando a interface `UserDetailsService` do Spring Security. Para poder executar a *query*, injete uma instância do `EntityManager` (usando a anotação `@PersistenceContext`). Implemente no método `loadUserByUsername(String email)` a *query* para recuperar os usuários com esse e-mail:

```
@Repository
public class UsuarioDAO implements UserDetailsService {

    @PersistenceContext
    private EntityManager manager;

    public Usuario loadUserByUsername(String email) {
        List<Usuario> usuarios = manager
            .createQuery("select u from Usuario u where email = :email", Usuario.class)
            .setParameter("email", email)
            .getResultList();

        if(usuarios.isEmpty()) {
            throw new UsernameNotFoundException("Usuário " + email + " não foi encontrado");
        }
```

```
        return usuarios.get(0);
    }
}
```

9) Na classe `SecurityConfiguration` injeto o DAO e sobrescreva o método `configure`. Nesse método, passe o `usuarioDao` para o `AuthenticationManagerBuilder` e defina a criptografia da senha. Para tal, use o método `passwordEncoder`, que recebe um objeto do tipo `BCryptPasswordEncoder`:

```
@EnableWebSecurity
public class SecurityConfiguration
    extends WebSecurityConfigurerAdapter {

    @Autowired
    private UsuarioDAO usuarioDao;

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
            throws Exception {

        auth.userDetailsService(usuarioDao)
            .passwordEncoder(new BCryptPasswordEncoder());
    }

    // restante do código da classe omitido
}
```

10) No MySQl, insira um usuário e uma senha. Para facilitar segue o código SQL preparado:

```
insert into Role values('ROLE_ADMIN');

insert into Usuario (email, nome, senha) values ('admin@casadocodigo.com.br', 'Administrador',

insert into Usuario_Role(Usuario_email, roles_nome) values('admin@casadocodigo.com.br', 'ROLE_AI
```

A senha real é **123456**.

11) Caso você queira gerar a sua própria senha, execute a classe abaixo inserindo a senha desejada. O resultado da impressão deve ser inserido no banco de dados:

```
public class GeraSenha {

    public static void main(String[] args) {
        String senhaCriptografado = new BCryptPasswordEncoder().encode("123456");
        System.out.println(senhaCriptografado);
    }
}
```

12) Na página **home.jsp**, adicione a `taglib` do Spring Security:

```
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="security" %>
```

13) Ainda no **home.jsp**, embrulhe os dois links para listar e cadastrar produtos com `security:authorize`, testando o role `ROLE_ADMIN`:

```
<nav id="main-nav">
    <ul class="clearfix">
        <security:authorize access="hasRole('ROLE_ADMIN')">
            <li>
                <a href="${s:mvcUrl('PC#listar').build()}"
                        rel="nofollow">
                    Listagem de Produtos
                </a>
            </li>
            <li>
                <a href="${s:mvcUrl('PC#form').build()}"
                        rel="nofollow">
                    Cadastro de Produtos
                </a>
            </li>
        </security:authorize>
        <li>
            <a href="${s:mvcUrl('CCC#itens').build()}"
                    rel="nofollow">
                Seu carrinho (${carrinhoCompras.quantidade})
            </a>
        </li>
        <li>
            <a href="/pages/sobre-a-casa-do-codigo" rel="nofollow">
                Sobre nós
            </a>
        </li>
    </ul>
</nav>
```

14) Em **lista.jsp**, adicione a `taglib` do Spring Security e logo abaixo da `ul` de listagem e cadastro de produtos, insira uma nova `ul` com as classes padrões do Bootstrap para criação de um menu alinhado à direita da barra de menu. Dentro da `ul`, crie uma nova `li` usando a `security:authentication` para mostrar o nome do usuário logado:

```
<div id="navbar" class="collapse navbar-collapse">
    <ul class="nav navbar-nav">
        <li class="nav-item">
            <a href="${s:mvcUrl('PC#listar').build()}">
                Lista de Produtos
            </a>
        </li>
        <li class="nav-item">
            <a href="${s:mvcUrl('PC#form').build()}">
                Cadastro de Produtos
            </a>
        </li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
```

```
            <li class="nav-item">
                <a href="#">
                    <security:authentication
                        property="principal"
                            var="usuario" />
                    Usuário: ${usuario.username }
                </a>
            </li>
        </ul>
    </div><!-- /.navbar-collapse -->
```

15) Para evitar um ataque CSRF e trabalhar com um token que identifica o formulário, use em todos os formulários sempre a *tag* `form:form` , por exemplo nos arquivos **detalhe.jsp** e **lista.jsp**. Por exemplo, onde há:

```
<form action='<c:url value="/carrinho/add" />'
    method="post" class="container">
```

Ficará:

```
<form:form servletRelativeAction="/carrinho/add"
    method="post" cssClass="container">
```

16) Crie a nova página **formLogin.jsp** dentro da pasta **src/main/webapp/WEB_INF/views** com o seguinte código:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<%@ taglib uri="http://www.springframework.org/tags" prefix="s" %>

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Livros de Java, Android, iPhone, Ruby, PHP e muito mais - Casa do Código</title>
    <c:url value="/resources/css" var="cssPath" />
    <link rel="stylesheet" href="${cssPath}/bootstrap.min.css" />
    <link rel="stylesheet" href="${cssPath}/bootstrap-theme.min.css" />
    <style type="text/css">
        body {
            padding: 60px 0px;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Login Casa do Código</h1>
        <form:form servletRelativeAction="/login" method="POST">
            <div class="form-group">
                <label>E-mail</label>
                <input type="text" name="username"
                    class="form-control" />
            </div>
            <div class="form-group">
```

```
                <label>Senha</label>
                <input type="password" name="password"
                    class="form-control" />
            </div>
            <button type="submit" class="btn btn-primary">
                Logar
            </button>
        </form:form>
    </div>
</body>
</html>
```

17) Crie um novo *controller*, no pacote `br.com.casadocodigo.loja.controllers`, para poder acessar o formulário:

```
@Controller
public class LoginController {

    @RequestMapping(value="/login",
        method=RequestMethod.GET)
    public String loginForm(){
        return "loginForm";
    }

}
```

18) Na classe `SecurityConfiguration`, no método `configure(HttpSecurity http)` logo após da chamada `authenticated()`, adicione novas regras login e logout:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers("/produtos/form").hasRole("ADMIN")
        .antMatchers("/carrinho/**").permitAll()
        .antMatchers("/produtos/").hasRole("ADMIN")
        .antMatchers("/produtos/**").permitAll()
        .antMatchers("/resources/**").permitAll()
        .antMatchers("/pagamento/**").permitAll()
        .antMatchers("/").permitAll()
        .anyRequest().authenticated()
        .and().formLogin().loginPage("/login")
            .defaultSuccessUrl("/produtos").permitAll()
        .and().logout()
            .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
                .permitAll().logoutSuccessUrl("/login");
}
```

19) Na página **lista.jsp**, no menu de navegação, crie um link **Sair** que chama a URL `/logout`:

```
<li class="nav-item">
    <a href="<c:url value="/logout" />">Sair</a></span>
</li>
```