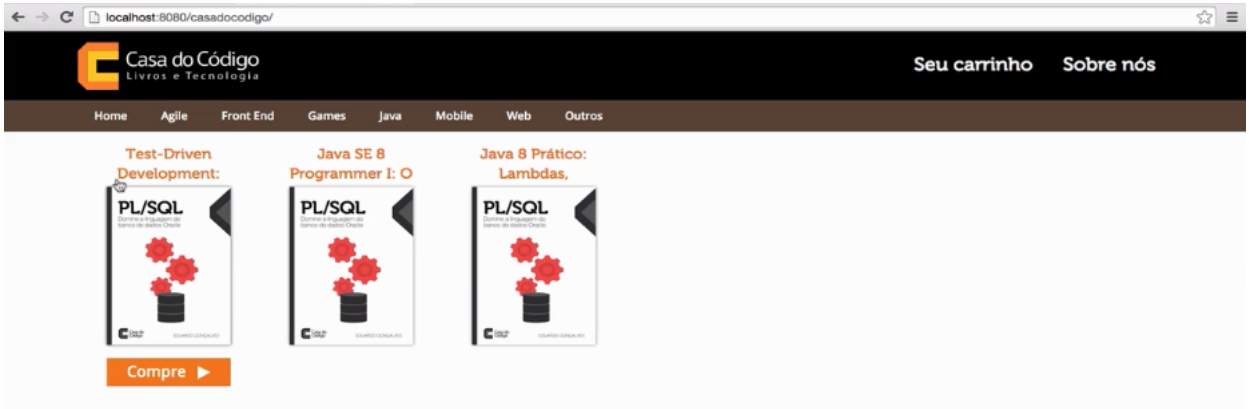
02

# Habilitando Spring Security

## Transcrição

Já temos a listagem e cadastro de produtos funcionando perfeitamente, mas note que não temos os links de acesso a estas páginas, na Home do projeto. Atualmente, a página inicial ( `home.jsp` ) encontra-se dessa forma:




Vamos adicionar os links para as páginas de cadastro e listagem de produtos junto aos links de `Seu carrinho` e `Sobre nós` . No arquivo `home.jsp` , busque o seguinte trecho de código:

```
<ul class="clearfix">
    <li><a href="/cart" rel="nofollow">Carrinho</a></li>
    <li><a href="/pages/sobre-a-casa-do-codigo" rel="nofollow">Sobre Nós</a></li>
</ul>
```

Vamos criar mais duas opções neste menu com os links que criamos na aula passada apontando para as páginas de listagem e cadastro de produtos da seguinte forma:

```
<ul class="clearfix">

    <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>

    <li><a href="/cart" rel="nofollow">Carrinho</a></li>
    <li><a href="/pages/sobre-a-casa-do-codigo" rel="nofollow">Sobre Nós</a></li>
</ul>
```

E como resultado teremos os links sendo exibidos na página inicial da aplicação:


Vamos refletir sobre o que acabamos de fazer: com os links expostos desta maneira, possibilitamos que qualquer usuário possa cadastrar livros na Casa do Código. Imagine a Casa do Código recebendo pedidos de livros que não existem realmente. Precisamos ter um controle de acesso e somente pessoas autorizadas poderão entrar nas páginas de cadastro de livros, assim como na de listagem.

Há várias estratégias que resolvem este tipo de problema. Poderíamos criar um filtro de requisições, um interceptador etc. Mas usaremos um recurso que o próprio *Spring* nos fornece. Um filtro já pronto, capaz de fazer o trabalho de verificar se o usuário tem ou não autorização de acessar determinadas páginas.

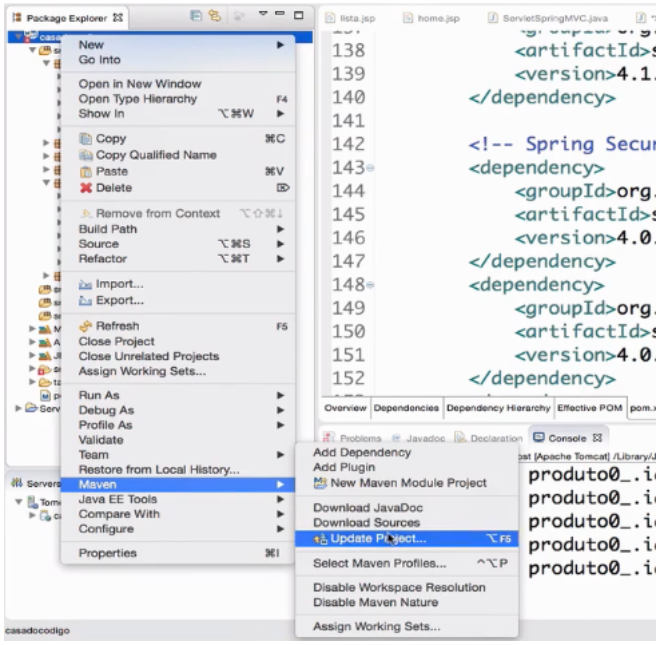
Para utilizarmos este filtro, precisamos configura-lo no projeto de forma semelhante ao que fizemos quando criamos a classe `ServletSpringMVC` , ou seja, criar uma outra classe que herde da classe do filtro que já se encontra configurado.

Para que sejamos capazes de utilizar os recursos de segurança do *Spring*, teremos que deixar estes recursos declarados nas depêndencias do nosso projeto. Sendo assim, devemos abrir o arquivo `pom.xml` e adicionar as seguintes linhas de dependência já usando a versão `RELEASE` (sem ser milestone):

```
<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>
```

**Observação:** No video usamos a versão **Milestone** por ser a que estava disponível durante a criação do curso.

Após as adições no arquivo `pom.xml` lembre-se de atualizar o projeto da seguinte forma:




O próximo passo para começarmos a utilizar realmente os recursos de segurança do *Spring* é criar a classe responsável por inicializar o filtro de segurança. No pacote `br.com.casadocodigo.loja.conf`, criaremos a classe `SpringSecurityFilterConfiguration`, que faremos extender a classe `AbstractSecurityWebApplicationInitializer`.

```
public class SpringSecurityFilterConfiguration extends AbstractSecurityWebApplicationInitializer

}
```

Apenas isto não será o suficiente. A classe da forma que está já funciona, mas ela apenas inicializa o filtro de segurança do *Spring*. Onde estão realmente as configurações de segurança? Em nenhum lugar! Para armazenar as configurações de segurança, criaremos uma nova classe chamada `SecurityConfiguration` no mesmo pacote, iremos anotá-la com `EnableWebSecurity`.

```
@EnableWebSecurity
public class SecurityConfiguration {

}
```

*Obs: Note que no video usamos a anotação `@EnableWebMvcSecurity`. A anotação `@EnableWebMvcSecurity` foi deprecada em novas versões do Spring Security e decidimos já usa a anotação mais atual.*

Desta forma, o *Spring* através da classe com esta anotação já configura alguns detalhes de segurança de forma automática. Mas para que isso funcione, o *Spring* precisa saber que a classe existe. Lembra qual classe que carrega todas as configurações de nossa aplicação? É a `ServletSpringMVC`. Nesta usamos o método `getServletConfigClasses` para carregar as configurações da aplicação e do **JPA** no primeiro módulo deste curso.

Mas agora, o método que usaremos é o `getRootConfigClasses` que carrega configurações logo ao iniciar a aplicação. Este método simplesmente retorna um **Array** de classes do mesmo jeito que o método `getServletConfigClasses` faz. Assim faremos a classe `SecurityConfiguration` ser reconhecida pelo *Spring*:

```
protected Class<?>[] getRootConfigClasses() {
    return new Class[]{SecurityConfiguration.class};
```

```
}
```

Agora poderemos fazer o teste! Mas ao reiniciarmos o servidor, teremos um erro indicando que nossas configurações não foram suficientes, veja:

```
Hint try extending WebSecurityConfigurerAdapter
```

O erro explica que deve haver uma classe que herde da classe `WebsecurityConfigurerAdapter`. Se olharmos nossa classe de configuração ( `SecurityConfiguration` ), veremos que ela só habilita as configurações, mas não configura nada. Veja:

```
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter{

}
```

A anotação é bem clara. Ela simplesmente habilita o recurso de **Web MVC Secutiry** e quem configura o recurso é justamente a classe `WebSecurityConfigurerAdapter`. Então, para solucionar o erro, precisaremos fazer com que a classe `SecurityConfiguration` herde da classe `WebSecurityConfigurerAdapter` da seguinte maneira:

```
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter{

}
```

Se reiniciarmos o servidor agora, já poderemos ver que o erro foi resolvido e que o recurso de segurança já funciona. Ao abrirmos nossa aplicação em `localhost:8080/casadocodigo` teremos: