

TUTORIAL SPRING SECURITY

PROGRAMAÇÃO COM FRAMEWORKS

Versão 1.0

Responsáveis:
Ana Luíza Cruvinel, Maikon Franczak e Wendel Borges

Data: 01/12/2014

SUMÁRIO

1.	INTRODUÇÃO	2
2.	O QUE É SPRING SECURITY?	2
3.	INSTALANDO E CONFIGURANDO O SPRING SECURITY NO PROJETO.....	2
3.1.	Instalação do Spring Security	2
4.	ADAPTAÇÃO DA CONEXÃO DO HIBERNATE PARA O SPRING.....	3
4.1.	Criação do arquivo context.xml.....	3
4.2.	Alteração do arquivo web.xml para a conexão	4
4.3.	Alteração do arquivo hibernate.cfg.xml	4
5.	CONFIGURAÇÃO DO SPRING SECURITY	5
5.1.	Alteração do arquivo web.xml	5
5.2.	Criação dos arquivos de configuração do Spring Security.....	6
5.2.1.	Criação do arquivo applicationContext.xml.....	6
5.2.2.	Criação do arquivo applicationContext-security.xml	7
5.2.3.	Exemplos de login e os recursos liberados para os respectivos usuários, segundo suas permissões configuradas no Spring Security:.....	9
6.	CONCLUSÃO	12

1. INTRODUÇÃO

O objetivo do aplicativo desenvolvido na disciplina de Programação com Frameworks foi implementar meios de torná-lo mais seguro. Sendo assim, foi utilizado o framework de segurança **Spring Security**, com isso, permitirá que apenas usuários autorizados tenham acesso aos recursos restritos.

2. O QUE É SPRING SECURITY?

O Spring Security se concentra em fornecer autenticação e autorização para aplicações Java. Sua principal funcionalidade é a injeção de dependências nas instâncias feitas, com base em definições em um arquivo XML.

Características:

- Suporte abrangente e extensível para autenticação e autorização de proteção contra ataques como fixação de sessão, furto de click, falsas solicitações, etc.;
- Integração API Servlet;
- Integração opcional com o Spring Web MVC.

3. INSTALANDO E CONFIGURANDO O SPRING SECURITY NO PROJETO

3.1. Instalação do Spring Security

Passo 1: Para utilizar o Spring Security no projeto, primeiramente é preciso fazer o download do arquivo no site <http://projects.spring.io/spring-security/>.

Segue a estrutura interna do arquivo baixado:

Spring-security-3.0

+docs

- Na pasta **docs** você encontrará o manual de referência em PDF E HTML, além da documentação da API Spring Security.

+dist

- Na pasta **dist** estão as bibliotecas do Spring Security

Os projetos de exemplo são os dois arquivos **.war** existentes na pasta dist, que são **spring-security-samples-contacts-<versão>.war** e **spring-security-samples-tutorial-<versão>.war**.

Passo 2: Sendo assim, extraia o arquivo **spring-security-samples-contacts-<versão>.war** (renomeie para **.zip**) e, de dentro dele, obtenha os arquivos jars do diretório **WEB-INF/lib**.

4. ADAPTAÇÃO DA CONEXÃO DO HIBERNATE PARA O SPRING

Como as configurações do banco de dados estão declaradas no arquivo *hibernate.cfg.xml*, deve-se , com a introdução do Spring Security, que este se conecte ao banco de dados. Para evitar que seja necessário ter dois arquivos de configuração de banco para o mesmo projeto, é necessário tirar essa funcionalidade do Hibernate e compartilhá-la entre o Hibernate e o Spring. Com isso, estaremos disponibilizando um DataSource JNDI para os dois frameworks.

Para aplicarmos todas as vantagens do DataSource , será necessário criar no projeto o arquivo *context.xml*, alterar os arquivos *web.xml* e *hibernate.cfg.xml*.

O arquivo *context.xml* permite que configurações sobre o aplicativo, que estariam no arquivo *CATALINA_HOME\conf\server.xml*, possam ser declaradas no próprio aplicativo.

Além disso, será necessário copiar o arquivo *mysql-connctor-java-<versão>-bin.jar*, para a pasta *lib* em *WEB-INF*.

4.1. Criação do arquivo context.xml

O arquivo *context.xml* deve ser criado em *WebContent/META-INF*, conforme o exemplo a seguir.

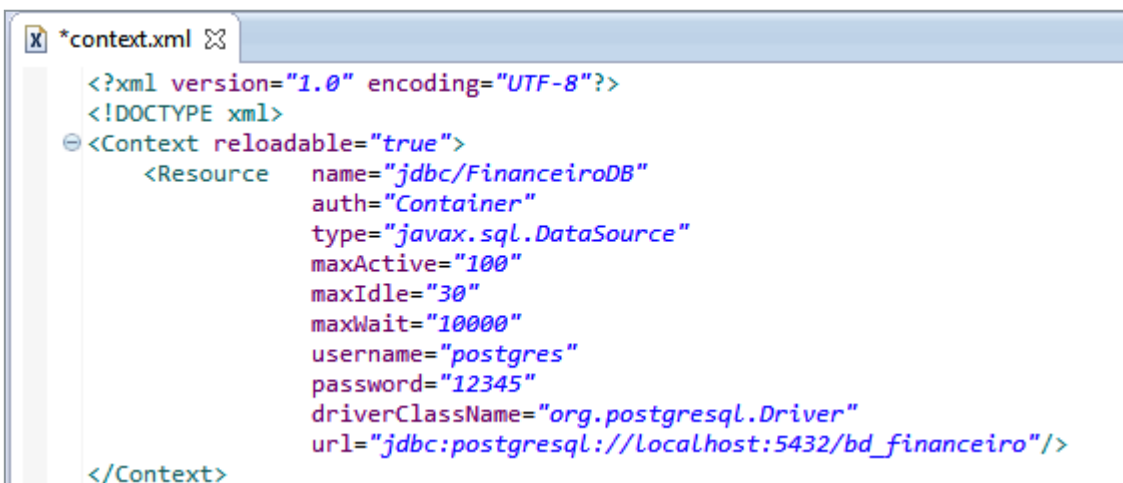
Criando o arquivo *context.xml*:

Passo 01: Clique com o botão inverso do mouse sobre a pasta *META-INF*, clique na opção *New* → e, em seguida, *File*.

Passo 02: Na janela *New File* defina o nome do arquivo em *File Name* como *context.xml* e clique no botão *Finish*.

Passo 03: Clique duas vezes no arquivo criado e deixe-o igual à figura abaixo.

Passo 04: Deixe o arquivo *context.xml*, igual à figura abaixo:



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xml>
<Context reloadable="true">
  <Resource name="jdbc/FinanceiroDB"
    auth="Container"
    type="javax.sql.DataSource"
    maxActive="100"
    maxIdle="30"
    maxWait="10000"
    username="postgres"
    password="12345"
    driverClassName="org.postgresql.Driver"
    url="jdbc:postgresql://localhost:5432/bd_financeiro"/>
</Context>
```

Figura 1: Arquivo contexto.xml

Com isso acabamos de configurar um DataSource JNDI com o nome ***jdbc/FinanceiroDB***.

Segue os atributos e as respectivas configurações para o funcionamento do pool de conexão:

- **maxActive** - Número máximo de conexões em uso.
- **maxIdle** - Número máximo de conexões aguardando uso.
- **maxWait** – Tempo máximo de espera por uma conexão disponível.

No exemplo apresentado, foi utilizado também o login postgres e senha 12345.

4.2. Alteração do arquivo **web.xml** para a conexão

É necessário adicionar ao arquivo **web.xml** uma referência para o **DataSource JNDI** criado. Com isso, avisará ao Tomcat que o **DataSource JNDI** deve estar disponível para o aplicativo em questão.

Segue o bloco de configuração necessário para habilitar o projeto ao respectivo recurso:

WEB-INF/web.xml

```
[...]
<resource-ref>
<description>DataSource Financeiro</description>
<res-ref-name>jdbc/FinanceiroDB</res-ref-name>
<res-type>java.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
[...]
```

4.3. Alteração do arquivo **hibernate.cfg.xml**

Como foi criado um **DataSource JNDI** com as respectivas funções de acesso ao banco, é necessário editar o **hibernate.cfg.xml** para não mais conter as configurações do banco de dados, e sim apontar para o DataSource criado no arquivo **context.xml**.

Quando a isso, é necessário remover as seguintes linhas do arquivo:

```
[...]
<property name="hibernate.connection.driver_class">
org.postgresql.Driver
</property>
<property name="hibernate.connection.password">12345</property>
<property name="hibernate.connection.url">
jdbc:postgresql://localhost:5432/bd_financeiro
</property>
```

```
<property name="hibernate.connection.username">postgres</property>
[...]
```

E acrescentar a linha seguinte:

```
[...]
<property name="connection.datasource">
java:comp/env/jdbc/FinanceiroDB
</property>
[...]
```

5. CONFIGURAÇÃO DO SPRING SECURITY

A criação do **Spring Security** abrange a criação de dois arquivos de configuração próprios do Spring e a alteração do arquivo **web.xml**.

5.1. Alteração do arquivo web.xml

Para que o **Spring Security** possa interceptar as requisições realizadas e avaliar as permissões de acesso requisitadas, é necessário alterar o arquivo **web.xml**.

Segue as alterações necessárias:

```
[...]
<!-- Spring Security -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/applicationContext.xml
        /WEB-INF/applicationContext-security.xml
    </param-value>
</context-param>
<context-param>
    <param-name>com.sun.faces.expressionFactory</param-name>
    <param-value>com.sun.el.ExpressionFactoryImpl</param-value>
</context-param>
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
```

```

    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern> /*</url-pattern>
</filter-mapping>
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
[...]
```

Segue as funcionalidades das tags destacadas:

- ✓ **<context-param>** com o nome **contextoConfigLocation** informa quais são os arquivos de configuração disponíveis.
- ✓ **<filter>/<filter-mapping>** As configurações nas respectivas tags permitirão ao Spring Security interceptar todas as requisições realizadas.
- ✓ **<url-pattern>** A configuração **/*** faz com que todas as requisições sejam avaliadas.
- ✓ **<listener>** Habilita ao **Spring Security** carregar os arquivos de configuração no momento em que o arquivo web for solicitado.

5.2. Criação dos arquivos de configuração do Spring Security

Como foi falado anteriormente, é necessária a criação de dois arquivos de configuração para a instalação do **Spring Security**. São eles, **applicationContext.xml** e **applicationContext-security.xml**

5.2.1. Criação do arquivo applicationContext.xml

As configurações do Spring Framework serão implementadas no arquivo **applicationContext.xml**. Este faz referência ao **DataSource JNDI** e será criado no diretório **WebContent/WEB-INF**.

Passo 01: Clique com o botão inverso do mouse sobre a pasta **WEB-INF**, clique na opção **New** → e, em seguida, **File**.

Passo 02: Na janela **New File** defina o nome do arquivo em **File Name** como **applicationContext.xml** e clique no botão **Finish**.

Passo 03: Clique duas vezes no arquivo criado e deixe-o igual à figura abaixo.

Passo 04: Deixe o arquivo **applicationContext.xml**, igual à figura abaixo:

WEB-INF/applicationContext.xml

```
*applicationContext.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
  <bean id="financeiroDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName">
      <value>java:comp/env/jdbc/FinanceiroDB</value>
    </property>
  </bean>
</beans>
```

Figura 2: Arquivo applicationContext.xml

- ✓ **<bean>** Este elemento cria a referência **financeiroDataSource** do tipo **JndiObjectFactoryBean**, apontando para o **DataSource JNDI jdbc/FinanceiroDB**.

5.2.2. Criação do arquivo applicationContext-security.xml

O arquivo **applicationContext-security.xml** conterá as configurações de permissão para as pastas e , também, as configurações de origem dos dados dos usuários e suas respectivas permissões. Este deverá ser criado no mesmo diretório do arquivo anterior, **WebContent/WEB-INF**.

Passo 01: Clique com o botão inverso do mouse sobre a pasta **WEB-INF**, clique na opção **New** → e, em seguida, **File**.

Passo 02: Na janela **New File** defina o nome do arquivo em **File Name** como **applicationContext-security.xml** e clique no botão **Finish**.

Passo 03: Clique duas vezes no arquivo criado e deixe-o igual à figura abaixo.

Passo 04: Deixe o arquivo *applicationContext-security.xml*, igual à figura abaixo:

```

applicationContext-security.xml
<?xml version="1.0" encoding="UTF-8"?>
<b:beans xmlns="http://www.springframework.org/schema/security"
  xmlns:b="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security-3.0.xsd">

  <http>
    <intercept-url pattern="/admin/**" access="ROLE_ADMINISTRADOR" />
    <intercept-url pattern="/restrito/**" access="ROLE_USUARIO" />
    <form-login login-page="/publico/Login.xhtml"
      always-use-default-target="true" default-target-url="/restrito/principal.xhtml"
      authentication-failure-url="/publico/Login.xhtml?login_error=1" />
    <logout/>
    <remember-me />
  </http>

  <authentication-manager>
    <authentication-provider>
      <jdbc-user-service data-source-ref="financeiroDataSource"
        authorities-by-username-query="SELECT u.login, p.permissao
          FROM usuario u, usuario_permissao p
          WHERE u.id = p.usuario
          AND u.login = ?"
        users-by-username-query="SELECT login, senha, ativo
          FROM usuario
          WHERE login = ?" />
    </authentication-provider>
  </authentication-manager>
</b:beans>

```

Figura 3: Arquivo *applicationContext-security.xml*

Segue as funcionalidades das tags acrescentadas no arquivo:

- ✓ **<HTTP>** Agrupa as configurações referentes ao contexto web do sistema.
- ✓ **<intercept-url>** Este elemento tem como característica a configuração de quais páginas ou diretórios serão seguros no qual, o atributo *pattern* expressa o padrão textual da URL e o atributo *access* especifica os nomes de permissão que terão acesso aos recursos.
- ✓ **<form-login>** Configura o funcionamento da página de login do *Spring Security*.
- ✓ **<logout>** Habilita o recurso de logout para o sistema.
- ✓ **<remember-me>** Permite habilitar o login automático.

- ✓ **<authentication-provider>** Esta tag diz ao Spring quais são os usuários válidos do sistema e suas permissões.
- ✓ **<jdbc-user-service>** permite declarar as SQLs que fornecerão os dados que o Spring Security necessita, vindas do banco de dados.

Segue as funcionalidades das tags **<jdbc-user-service>**:

- ✓ **User-by-username-query** – Fornece o login, senha e status de ativo dos usuários do sistema.
- ✓ **Authorities-by-username-query** – Fornece os logins dos usuários e suas permissões .

Obs.: É necessário colocar o bloco de comando `<sec:ifAnyGranted roles="ROLE_ADMINISTRADOR">` na página principal.xhtml para que o sistema redireciona segundo as permissões cadastradas na página ***applicationContext-security.xml*** para que o Spring saiba, por meio da tag **<authentication-provider>** quais são os usuários válidos do sistema e suas permissões, que nesse caso é administrador e usuário.

5.2.3. Exemplos de login e os recursos liberados para os respectivos usuários, segundo suas permissões configuradas no Spring Security:



Figura 4: Página inicial

Nesta página, podemos perceber as restrições configuradas no Spring Security ao utilizar a tag `<form-login>` que é um de seus atributos, para o desenvolvimento da página de Login. E, com isso, ele verifica no banco se existe o usuário e quais são as permissões cadastradas.



Figura 5: Página Principal de usuário comum

Nesta página podemos perceber que o Spring Security direcionou o usuário cadastrado para a respectiva área de permissão que, no caso, não tem permissão de administrador.



Figura 6: Página principal de usuário administrador

Nesta página podemos perceber o mesmo usuário já com a permissão de administrador, com isso, o Spring Security o redirecionou para a página de administrador.



Figura 7: Página de listagem de usuários – Permissão Administrador

Nesta página são demonstradas as permissões cadastradas no respectivo documento do Spring Security, o applicationContext-security.xml, sendo que nesta, é demonstrado a permissão de todos os usuários cadastrados no sistema. Em destaque a permissão Administrador.

Sendo que, neste caso, o usuário teve acesso a esta página por identificar, segundo o código que é ativado apenas caso o usuário seja administrador.

```
<sec:ifAnyGranted roles="ROLE_ADMINISTRADOR">
<h:commandLink action="/admin/principal.xhtml"
                title="Administrativo">
<h:graphicImage library="imagens" name="administrativo16.png"
style="border:0" />
</h:commandLink>
```



Figura 8: Página de listagem de usuários – Permissão Usuário Vip

Em destaque a permissão Usuário VIP cadastrado no Spring Security para o usuário Ana.

6. CONCLUSÃO

Em suma, é de extrema necessidade a implementação do Spring Security, pois com ele, é analisado isoladamente cada requisição de acesso. Com isso, agregaremos maior segurança ao projeto, que protegerão de uma forma mais eficiente os dados e informações do usuário.