Acabamos de ver como usar Java equals y hashcode

El uso de Spring Security JDBC es un clásico. En muchas ocaciones ocasiones necesitamos configurar Spring Security para que almacene los usuarios y los roles en una base de datos . Hoy por hoy esto es bastante sencillo de hacer si nos apoyamos en el manejo de anotaciones y Spring Boot. . Vamos a ver un ejemplo sencillo de como configurar lo básico. El primer paso es crearnos un proyecto de Spring Boot que contenga las siguientes dependencias.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<groupId>com.arquitecturajava</groupId>
	<artifactId>security1</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>war</packaging>

	<name>security1</name>
	<description>Demo project for Spring Boot</description>

	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>1.5.11.BUILD-SNAPSHOT</version>
		<relativePath /> <!-- lookup parent from repository -
->
	</parent>
```

```
        <properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncod
ing>
                <java.version>1.8</java.version>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
security</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
web</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
test</artifactId>
                        <scope>test</scope>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
jdbc</artifactId>
                </dependency>
```

```
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<scope>runtime</scope>
		</dependency>
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>jstl</artifactId>
		</dependency>

		<dependency>
<groupId>org.springframework.security</groupId>
			<artifactId>spring-security-test</artifactId>
			<scope>test</scope>
		</dependency>

		<dependency>
			<groupId>org.apache.tomcat.embed</groupId>
			<artifactId>tomcat-embed-jasper</artifactId>
			<scope>provided</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-
plugin</artifactId>
			</plugin>
		</plugins>
```

```
</build>

<repositories>
        <repository>
                <id>spring-snapshots</id>
                <name>Spring Snapshots</name>
                <url>https://repo.spring.io/snapshot</url>
                <snapshots>
                        <enabled>true</enabled>
                </snapshots>
        </repository>
        <repository>
                <id>spring-milestones</id>
                <name>Spring Milestones</name>
                <url>https://repo.spring.io/milestone</url>
                <snapshots>
                        <enabled>false</enabled>
                </snapshots>
        </repository>
</repositories>

<pluginRepositories>
        <pluginRepository>
                <id>spring-snapshots</id>
                <name>Spring Snapshots</name>
                <url>https://repo.spring.io/snapshot</url>
                <snapshots>
                        <enabled>true</enabled>
                </snapshots>
        </pluginRepository>
        <pluginRepository>
```

```
                        <id>spring-milestones</id>
                        <name>Spring Milestones</name>
                        <url>https://repo.spring.io/milestone</url>
                        <snapshots>
                                <enabled>false</enabled>
                        </snapshots>
                </pluginRepository>
        </pluginRepositories>

</project>
```

El siguiente paso es configurar el fichero de configuración de Spring con un ViewResolver que nos permite gestionar JSP.

```
package com.arquitecturajava.security1;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.web.servlet.ViewResolver;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import
org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

@EnableWebMvc
@Configuration
@ComponentScan("com.arquitecturajava.*")
```

```
@Import({ SpringConfiguracionSeguridad.class })
public class SpringConfiguracion {

	@Bean
	public ViewResolver viewResolver() {

		InternalResourceViewResolver viewResolver = new
InternalResourceViewResolver();
		viewResolver.setViewClass(JstlView.class);
		viewResolver.setPrefix("/");
		viewResolver.setSuffix(".jsp");

		return viewResolver;
	}

}
```

A partir de este momento ya podremos gestionar vistas JSP . Al tratarse de un proyecto basado en Spring Boot deberemos modificar el fichero de application.properties para que define la carpeta de vistas y el sufijo de estas.

```
spring.mvc.view.prefix: /
spring.mvc.view.suffix: .jsp
```

Spring Security JDBC

Es momento de ver el contenido del fichero que se encarga de configurar la seguridad.

```
package com.arquitecturajava.security1;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import
org.springframework.security.config.annotation.authentication.builders
.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.builders.HttpSecuri
ty;
import
org.springframework.security.config.annotation.web.configuration.Enabl
eWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSe
curityConfigurerAdapter;

@EnableWebSecurity
public class SpringConfiguracionSeguridad extends
WebSecurityConfigurerAdapter {

         @Autowired
         DataSource dataSource;

        @Override
        protected void configure(HttpSecurity http) throws Exception {
```

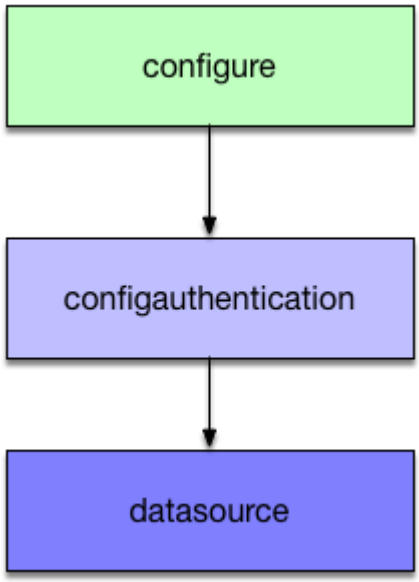
```
http.authorizeRequests().antMatchers("/**").hasRole("basico").and().fo
rmLogin();
        }

        @Bean(name = "dataSource")
        public DriverManagerDataSource dataSource() {
                DriverManagerDataSource driverManagerDataSource = new
DriverManagerDataSource();
driverManagerDataSource.setDriverClassName("com.mysql.jdbc.Driver");
driverManagerDataSource.setUrl("jdbc:mysql://localhost:8889/mibasedato
s");
                driverManagerDataSource.setUsername("cecilio");
                driverManagerDataSource.setPassword("cecilio");
                return driverManagerDataSource;
        }

        @Autowired
        public void configAuthentication(AuthenticationManagerBuilder
auth) throws Exception {
                auth.jdbcAuthentication().dataSource(dataSource)
                                .usersByUsernameQuery("select
username,password, enabled from users where username=?")
                                .authoritiesByUsernameQuery("select
username, role from user_roles where username=?");
        }

}
```

Este fichero se encarga de definir la configuración completa de Spring Framework a nivel de seguridad .



1. El primer método se encarga de configurar el acceso a los recursos y los roles admitidos. En este caso se admite el acceso a todos los recursos a cualquier usuario que disponga de un rol básico.
2. El segundo método se encarga de configurar el tipo de autenticación soportado por Spring Security . En este caso se trata de hacer uso de JDBC , por lo tanto veremos como delega en el uso de un dataSource para conectarse a la base de datos. No solo eso sino que configura ademas cuales son las consultas que se ejecutarán contra unas tablas concretas.
3. Por último el método que se encarga de definir el DataSource y como nos conectamos a la base de datos.
   Una vez tenemos todo esto configurado el siguiente paso es dar de alta en la base de datos en la tabla users:



	username	password	enabled
Editar Copiar Borrar	cecilio	superclave	1

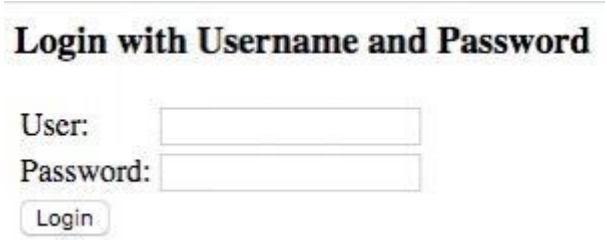
   Este usuario además dispondrá de una serie de roles por lo tanto también tenemos que insertar información en esa tabla:



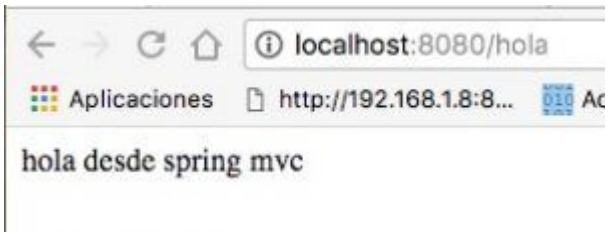
+ Opciones

	user_role_id	username	role
Editar Copiar Borrar	1	cecilio	ROLE_basico

Aquí podemos ver la relación entre la tabla de users y la de users_roles . En este caso nuestro usuario solo dispone de un rol muy concreto el básico . Eso si por defecto en Spring Security todos llevan el prefijo de ROLE_. Acabamos de configurar Spring Security para gestionar la seguridad apoyándonos en una base de datos es momento de arrancar la aplicación y ver como nos solicita usuario y clave por pantalla para acceder a los recursos.



Introducimos nuestros datos y accedemos:



Otros artículos relacionados:

1. Spring Security Annotation y su configuración
2. Spring Security (I) configuracion
3. Spring Security Login (II)

Links Externos:

Spring Security