



# Construcción de una aplicación empresarial con Spring

---

Guía del participante

# Índice

<b>1. Tabla de cambios</b>	<b>3</b>
<b>2. Resumen</b>	<b>4</b>
<b>3. Servicios Web REST</b>	<b>5</b>
<b>4. Preparación de la base de datos</b>	<b>7</b>
<b>5. Inicialización del proyecto</b>	<b>8</b>
<b>6. Spring webmvc</b>	<b>11</b>
6.1. Configuración . . . . .	11
6.2. Controlador de prueba . . . . .	13
6.3. Conclusión . . . . .	14
<b>7. Spring data</b>	<b>15</b>
7.1. Prerrequisitos . . . . .	15
7.2. Configuración . . . . .	15
7.3. Entidades . . . . .	17
7.4. Repositorios . . . . .	19
7.5. Capa se servicios . . . . .	19
7.6. Servicio Rest . . . . .	22
7.7. Conclusión . . . . .	23
<b>8. Spring security</b>	<b>25</b>
8.1. Conceptos básicos de seguridad . . . . .	25
8.2. JSON Web Token . . . . .	26
8.3. Flujo de seguridad . . . . .	27
8.4. Seguridad a nivel web y seguridad a nivel de método . . . . .	28
8.5. Configuración . . . . .	29
8.6. Conclusión . . . . .	41
<b>9. Siguientes pasos</b>	<b>42</b>
<b>Appendices</b>	<b>43</b>
<b>Apéndice A. Scripts de la base de datos</b>	<b>44</b>
A.1. Esquema . . . . .	44
A.2. Creación de usuario . . . . .	45
A.3. Datos iniciales . . . . .	45
<b>Apéndice B. Archivos de configuración de Spring</b>	<b>46</b>
B.1. spring-context.xml . . . . .	46
B.2. spring-mvc-context.xml . . . . .	46
B.3. spring-data-context.xml . . . . .	46

B.4. spring-security-context.xml . . . . .	47
<b>Apéndice C. Otros archivos de configuración</b>	<b>49</b>
C.1. pom.xml . . . . .	49
C.2. web.xml . . . . .	51
C.3. jetty-env.xml . . . . .	52

## 1. Tabla de cambios

Versión	Autor	Autorización	Fecha
1.0	Edgar Ramos		2017/05/24

## 2. Resumen

En la actualidad se puede notar un auge en el uso de servicios web REST<sup>1</sup>, por lo que el ser capaz de desarrollar este tipo de sistemas se ha convertido en una habilidad valiosa en el mercado laboral.

En este curso aprenderemos como configurar algunos frameworks de Spring para crear de forma rápida este tipo de sistemas.

- Con *Spring webmvc* crearemos puntos de acceso – en forma de URI – a nuestro sistema. Este tipo de interfaz es el fundamento de todo servicio REST.
- Luego usaremos *Spring data* para comunicar nuestra aplicación con la capa de almacenamiento de datos.
- Finalmente haremos uso de la flexibilidad de *Spring security* para poder asegurar nuestra aplicación.

---

<sup>1</sup>Representational State Transfer / Transferencia de Estado Representacional

### 3. Servicios Web REST

En esencia un servicio Web es una forma de publicar nuestro sistema para que sea consumido – ya sea de forma pública o privada– por otros sistemas como pueden ser: el frontend de un sitio web escrito en javascript, una aplicación móvil, o simplemente cualquier otro sistema al que se preste el servicio. (fig. 1)

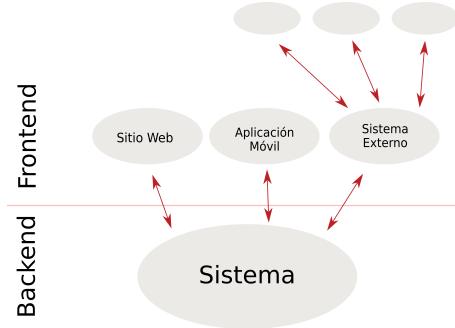


Figura 1: Estructura de servicios Web

Un servicio Web REST es un subconjunto de WWW (basado en HTTP) en el que los agentes proporcionan una interfaz con una semántica uniforme – esencialmente crear, recuperar, actualizar y eliminar– en vez de interfaces arbitrarias o específicas de una aplicación; y manipula recursos solamente intercambiando representaciones. Aún más, las interacciones REST no tienen un *estado* en el sentido de que el mensaje no depende del estado de la conversación. [1]

De lo anterior podemos entender que hacer nuestro sistema *RESTful* significa usar una forma estándar para el diseño de la interfaz que tendrá.

Este estándar se basa en los verbos<sup>2</sup> de HTML, generalmente *GET*, *POST*, *PUT* y *DELETE*. Además, las peticiones son respondidas con la representación del recurso – o una confirmación en caso de *DELETE* –. Esta representación puede estar en una variedad de formatos entre los que sobresalen XML y JSON. (tabla 1)

Verbo	Descripción	Resultado esperado
GET	Solicita al servidor un recurso	La representación del recurso solicitado
POST	Solicita al servidor la creación de un nuevo recurso	La representación del nuevo recurso
PUT	Solicita al servidor la modificación de un recurso	La representación actualizada del recurso
DELETE	Solicita al servidor la eliminación de un recurso	Una confirmación de que el recurso fue eliminado

Tabla 1: Principales verbos de HTML

<sup>2</sup>También llamados *métodos*

Para poner un ejemplo de lo anterior, supongamos que queremos interactuar con la información de un estudiante con matrícula 123456 mediante un servicio ubicado en <http://www.escuela.mx>. Si es un sistema *RESTful* se esperaría que tuviera la interfaz que se muestra en la tabla 2.

Acción	Verbo	URI	Respuesta
Crear un nuevo estudiante	POST	<a href="http://www.escuela.mx/estudiante">http://www.escuela.mx/estudiante</a> <sup>3</sup>	<code>{'nombre': 'Pedro', 'apellido': 'Pérez', 'matricula': '123456'}</code>
Consultar un estudiante	GET	<a href="http://www.escuela.mx/estudiante/123456">http://www.escuela.mx/estudiante/123456</a>	<code>{'nombre': 'Pedro', 'apellido': 'Pérez', 'matricula': '123456'}</code>
Modificar un estudiante	PUT	<a href="http://www.escuela.mx/estudiante/123456">http://www.escuela.mx/estudiante/123456</a> <sup>4</sup>	<code>{'nombre': 'Juan', 'apellido': 'Pérez', 'matricula': '123456'}</code>
Eliminar un estudiante	DELETE	<a href="http://www.escuela.mx/estudiante/123456">http://www.escuela.mx/estudiante/123456</a>	'Estudiante 123456 eliminando'

Tabla 2: Ejemplo de una interfaz *REST*

<sup>3</sup>En las peticiones POST los parámetros se envían en el cuerpo de la misma

<sup>4</sup>En las peticiones PUT los parámetros se envían en el cuerpo de la misma

## 4. Preparación de la base de datos

Antes de comenzar con el desarrollo de la aplicación es necesario preparar la base de datos que se usará. En esta ocasión se optó por implementar la administración de usuarios, ya que mantiene un esquema de base de datos sencillo a la vez que proporciona una funcionalidad de uso general.

El esquema de la base de datos (fig. 2) consta de dos tablas:

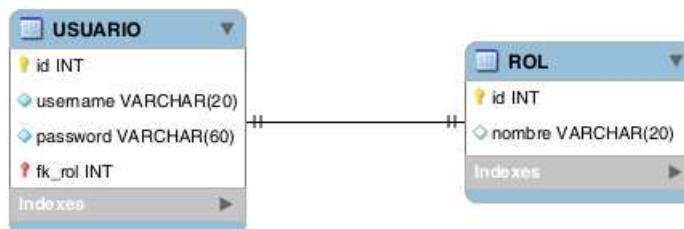


Figura 2: Esquema de la base de datos

**USUARIO** está conformado por un identificador, un nombre de usuario, una contraseña y una referencia a un Rol.

**ROL** está conformado simplemente por un identificador y el nombre del Rol.

Se proporcionan tres scripts en el Apéndice A<sup>5</sup>:

- A.1 Crea el esquema *SpringDemo* con las tablas necesarias y establece las restricciones. Este script debe ser ejecutado como *root*. Este script puede ejecutarse para limpiar la base de datos<sup>6</sup>.
- A.2 Crea el usuario *DemoUser* con contraseña *Password123*<sup>7</sup>. Este script debe ser ejecutado como *root*.
- A.3 Carga los datos iniciales. En este caso sólamente dos roles *ADMINISTRADOR* y *USUARIO*.

La tabla *USUARIO* tiene un espacio de 60 caracteres en el campo *password* ya que usaremos *bcrypt* para almacenar esa información.

*Bcrypt* es una función *hash* usada para guardar contraseñas de forma segura en una base de datos. Como característica importante tiene el agregado de un parámetro *salt* – un valor aleatorio – que ayuda a generar una cadena distinta cada vez, aún cuando las contraseñas de dos o más usuarios lleguen a coincidir.

<sup>5</sup>Para este curso se usará MySQL como manejador de base de datos.

<sup>6</sup>Esto es posible ya que el esquema es eliminado y recreado en caso de que ya exista

<sup>7</sup>En un ambiente de producción se debe establecer un proceso de autenticación seguro. Este es sólo un ejemplo, por lo que nos permitimos usar una contraseña insegura.

## 5. Inicialización del proyecto

Como primer paso debemos crear e inicializar nuestro proyecto. Esto puede variar dependiendo del IDE<sup>8</sup> que se utilice pero en términos generales buscamos crear un nuevo proyecto de maven (fig 3).

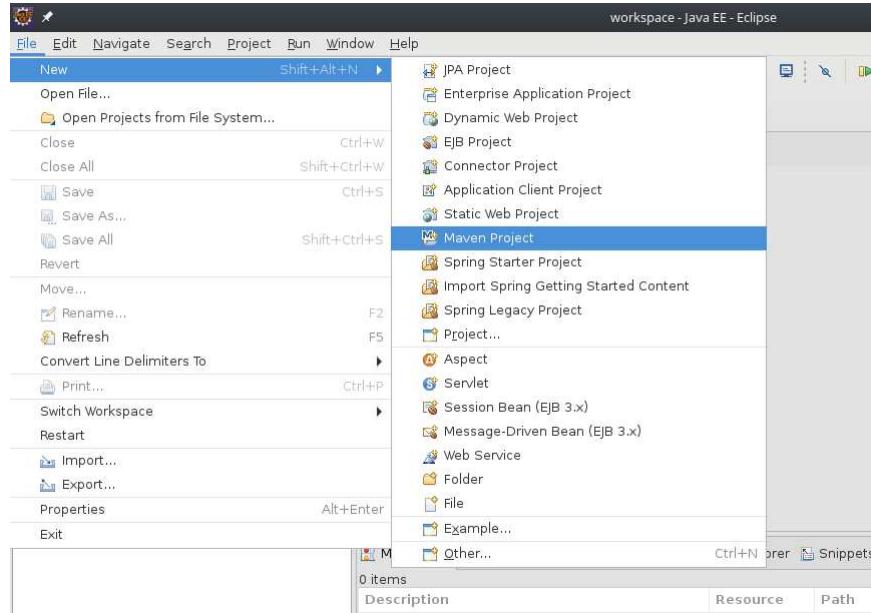


Figura 3: Creación de un nuevo proyecto de Maven en Eclipse neon.3

Independientemente del ambiente de desarrollo, maven cuenta con una serie de plantillas para proyectos llamadas *arquetipos*. En este caso usaremos *maven-archetype-webapp* (fig 4).

Usaremos los siguientes parámetros para crear el proyecto:

Parámetro	Valor
Group Id	com.mozcalti.cursos
Artifact Id	springdemo
Version	0.0.1-SNAPSHOT

Con esto, se generará la estructura de archivos que se muestra en la figura 5. A esto hay que eliminar el archivo *src/main/webapp/index.jsp* y crear el directorio *src/main/java*<sup>9</sup>.

Como parte final de la inicialización del proyecto nos queda agregar un par de plugins al archivo pom.xml<sup>10</sup>.

De forma predeterminada maven compila el código para que cumpla con las especificaciones de Java 1.5, para poder usar las ventajas de la versión más reciente necesitamos incluir este plugin.

<sup>8</sup>Para esta guía se usará Eclipse neon.3

<sup>9</sup>Esto hay que hacerlo ya que el arquetipo que utilizamos sólo se aproxima a la estructura que buscamos. Esto se puede solucionar generando un arquetipo personalizado, pero eso está fuera del alcance de este curso.

<sup>10</sup>La versión final del archivo pom.xml se encuentra en el Apéndice C.1

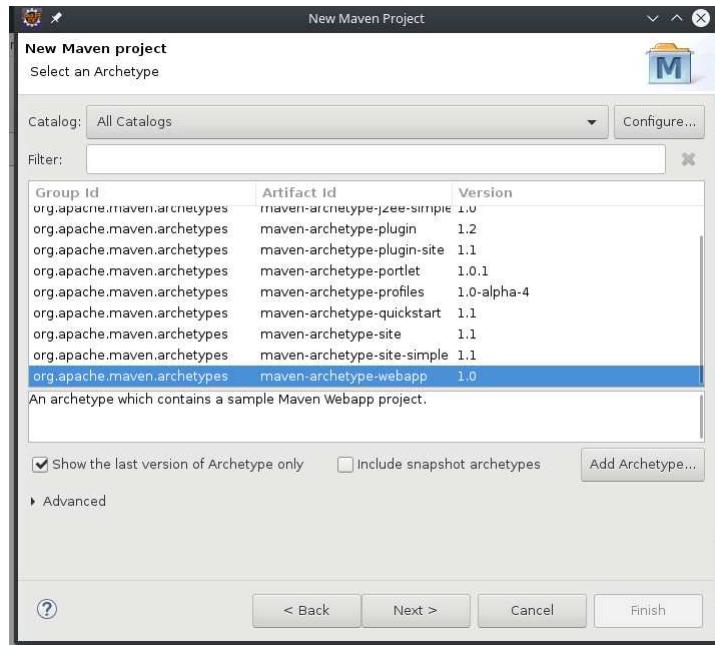


Figura 4: maven-archetype-webapp



Figura 5: Estructura inicial

```

1   <plugin>
2     <groupId>org.apache.maven.plugins</groupId>
3     <artifactId>maven-compiler-plugin</artifactId>
4     <version>3.6.1</version>
5     <configuration>
6       <source>1.8</source>
7       <target>1.8</target>
8     </configuration>
9   </plugin>

```

Jetty es un servidor desarrollado en Java que puede ser embeddo en el código. Esto también permite a maven instanciarlo para poder probar nuestra aplicación sin necesidad de instalar algún servidor en el sistema. Para eso necesitamos el siguiente plugin:

```
1   <plugin>
```

```

2      <groupId>org.eclipse.jetty</groupId>
3      <artifactId>jetty-maven-plugin</artifactId>
4      <version>9.4.5.v20170502</version>
5      <dependencies>
6          <dependency>
7              <groupId>org.eclipse.jetty</groupId>
8              <artifactId>jetty-plus</artifactId>
9              <version>9.4.5.v20170502</version>
10         </dependency>
11         <dependency>
12             <groupId>org.eclipse.jetty</groupId>
13             <artifactId>jetty-jndi</artifactId>
14             <version>9.4.5.v20170502</version>
15         </dependency>
16         <dependency>
17             <groupId>com.zaxxer</groupId>
18             <artifactId>HikariCP</artifactId>
19             <version>2.6.1</version>
20         </dependency>
21         <dependency>
22             <groupId>mysql</groupId>
23             <artifactId>mysql-connector-java</artifactId>
24             <version>6.0.6</version>
25         </dependency>
26         <dependency>
27             <groupId>org.eclipse.jetty</groupId>
28             <artifactId>jetty-servlets</artifactId>
29             <version>9.4.5.v20170502</version>
30         </dependency>
31     </dependencies>
32 </plugin>
33

```

Para probar que la configuración inicial haya sido exitosa, haremos que nuestro IDE ejecute el siguiente comando de maven<sup>11</sup>.

```
1 mvn clean package jetty:run
```

Si no hay ningun error, entonces podemos proceder a la siguiente etapa.

---

<sup>11</sup>Una vez más, la forma de lograrlo depende del IDE utilizado

## 6. Spring webmvc

Con los preparativos listos, es hora de comenzar a desarrollar nuestra aplicación.

En esta sección crearemos un primer servicio web que puede ser consumido por una interfaz REST. Aunque no cumplirá ninguna función real, nos permitirá estudiar el proceso de configuración de forma aislada para poder usarla luego en funciones más complejas.

### 6.1. Configuración

En primer término debemos agregar las siguientes dependencias<sup>12</sup> al archivo pom.xml

- **Spring webmvc** Contiene las clases y anotaciones necesarias para poder vincular métodos de objetos java a los puntos de acceso – URL– de nuestro sistema.

```

1   <dependency>
2     <groupId>org.springframework</groupId>
3     <artifactId>spring-webmvc</artifactId>
4     <version>4.3.8.RELEASE</version>
5   </dependency>
6

```

- **jackson-databind** Le permitirá a nuestra aplicación convertir entre objetos java y objetos JSON de forma sencilla y generalmente transparente para nosotros.

```

1   <dependency>
2     <groupId>com.fasterxml.jackson.core</groupId>
3     <artifactId>jackson-databind</artifactId>
4     <version>2.9.0.pr3</version>
5   </dependency>
6

```

El siguiente paso es configurar, en nuestro servidor, un *servlet* que contenga nuestra aplicación. Esto se hace en el archivo web.xml, donde debemos agregar el siguiente contenido:

```

1   <web-app>
2
3   <servlet>
4
5     <servlet-name>demoServlet</servlet-name>
6     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
7
8     <init-param>
9       <param-name>contextConfigLocation</param-name>
10      <param-value>/WEB-INF/context/spring-context.xml</param-value>
11    </init-param>
12

```

<sup>12</sup>Para tener las dependencias más actualizadas es recomendable buscarlas en un sitio como <http://mvnrepository.com/>

```

13 <load-on-startup>1</load-on-startup>
14
15 </servlet>
16
17 <servlet-mapping>
18 <servlet-name>demoServlet</servlet-name>
19 <url-pattern>/*</url-pattern>
20 </servlet-mapping>
21 </web-app>
22

```

Con lo anterior estamos instanciando un *servlet* de spring – DispatcherServlet – con un archivo de configuración<sup>13</sup>. Luego, le decimos a nuestro servidor cuales URL estarán manejados por este servlet.

En el paso anterior hicimos referencia a un archivo de inicialización que aún no existe, por lo que hay que crearlo.

Los archivos de configuración de Spring son archivos XML que generalmente declaran un *namespace* haciendo referencia a un xsd. Para facilitar el futuro mantenimiento haremos uso de un archivo principal que configurará el contexto general de Spring y luego cargará un archivo de configuración para cada módulo que use nuestra aplicación.

Dicho lo anterior, crearemos el directorio /WEB-INF/context y dentro de él, el archivo spring-context.xml<sup>14</sup> con el siguiente contenido

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3 xmlns:context="http://www.springframework.org/schema/context"
4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"
6 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"
7
8 <context:component-scan base-package="com.mozcalti.cursos.springdemo" />
9
10 <import resource="spring-mvc-context.xml"/>
11
12 </beans>

```

Con la línea *context:component-scan* le estamos diciendo a Spring que usaremos anotaciones para marcar nuestras clases y que deberá analizar el árbol de paquetes – iniciando en el que proporcionamos – para buscar los *beans*.

Finalmente, con el elemento *import* indicamos que hay configuraciones adicionales en ese otro archivo.

El contenido de spring-mvc-context.xml es el siguiente:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"

```

---

<sup>13</sup>De forma predeterminada el archivo de configuración corresponde al nombre del servlet seguido de *-servlet.xml* (*demoServlet-servlet.xml* en nuestro ejemplo), pero se considera buena práctica colocar los archivos de configuración en un directorio específico, por lo que es necesario agregar este parámetro de inicialización

<sup>14</sup>La mayoría de los IDE cuentan con herramientas para crear archivos de configuración de Spring. Si se cuenta con él, es recomendable usarlo.

```

3  xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www.w3.org/2001/XMLSchema
4  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/sch
5  http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xs
6
7  <mvc:annotation-driven />
8
9  </beans>
10

```

Con la línea *annotation-driven* señalamos que usaremos anotaciones en las clases para las configuraciones.

## 6.2. Controlador de prueba

Para crear servicios web con Spring webmvc basta anotar una clase con *@RestController* y *@RequestMapping*.

La anotación *@RequestMapping* tiene un argumento obligatorio que indica la ruta con la que se asociará la ejecución del método. Se puede usar también en los métodos, y generalmente se usa en ambos. Cuando se usa en la clase esta parte de la ruta será común para todos los métodos.

Con esto en mente, podemos crear nuestro primer controlador.

Definamos la clase ControladorPrueba en el paquete com.mozcalti.cursos.springdemo.controladores

```

1  package com.mozcalti.cursos.springdemo.controladores;
2
3  import java.util.Date;
4
5  import org.springframework.web.bind.annotation.PathVariable;
6  import org.springframework.web.bind.annotation.RequestMapping;
7  import org.springframework.web.bind.annotation.RequestMethod;
8  import org.springframework.web.bind.annotation.RestController;
9
10 @RestController
11 @RequestMapping("/prueba")
12 public class ControladorPrueba {
13
14     @RequestMapping(value = "/ping", method = RequestMethod.GET)
15     public String metodoPrueba() {
16         return (new Date()).toString();
17     }
18
19     @RequestMapping(value="/concatenar/{parametroUno}/{parametroDos}", method=RequestMethod.G
20     public String concatenar(@PathVariable String parametroUno,@PathVariable String parametro
21     {
22         return parametroUno + " " + parametroDos;
23     }
24
25
26 }

```

Es de resaltar que en la anotación `@RequestMapping` se especifica el verbo HTML con que se ligará este método. Ya que simplemente estamos solicitando un recurso – la fecha del servidor – le corresponde una solicitud `GET`.

Si volvemos a ejecutar nuestra aplicación, podemos hacer una solicitud `GET` a la ruta `http://localhost:8080/prueba/ping`, lo que nos devolverá la fecha y hora.

Nuestro primer método no recibe ningún parámetro, lo cual es un caso muy poco frecuente. Para poder recibir valores por medio de la petición Spring webmvc nos proporciona tres anotaciones:

`@PathVariable` para recibir un parámetro que aparezca en la ruta URI. En este caso, el parámetro se indica como parte del valor de la anotación `@RequestMapping`, colocándolo entre llaves (`{...}`).

`@RequestParam` para recibir un parámetro en el cuerpo de la petición. Esta opción se utiliza para procesar peticiones `POST` y `PUT`.

`@RequestHeader(<CABECERA>)` para recibir un valor incluído en una de las cabeceras de la solicitud.

Para proporcionar un ejemplo sencillo, agregaremos el siguiente método a la clase `ControladorPrueba`:

```
1  @RequestMapping(value="/concatenar/{parametroUno}/{parametroDos}", method=RequestMethod.GET)
2      public String concatenar(@PathVariable String parametroUno,@PathVariable String parametroDos)
3      {
4          return parametroUno + " " + parametroDos;
5      }
6
```

Con esto, si solicitamos el recurso `http://localhost:8080/prueba/Hola/Mundo` obtendremos la cadena `Hola Mundo`.

### 6.3. Conclusión

Hemos aprendido a realizar las configuraciones necesarias para definir la interfaz de nuestro sistema. Spring webmvc nos proporciona una serie de herramientas para poder hacerlo con un mínimo esfuerzo.

En la siguiente etapa aprenderemos a configurar Spring data para llevar a nuestro sistema un paso más allá y realizar operaciones reales, en el sentido de que estaremos interactuando con una base de datos.

## 7. Spring data

Ya que sabemos cómo publicar nuestros servicios web, es momento de establecer los mecanismos para poder persistir la información de nuestro sistema en una base de datos.

Para esto nos podemos ayudar con Spring data, un framework que facilita estas operaciones tanto como Spring webmvc nos ayudó en la sección anterior.

En particular usaremos dos conceptos en esta sección:

**Entidad** Es una clase POJO<sup>15</sup> que corresponde a una entidad en la base de datos. Esto se logra con anotaciones en la clase correspondiente.

**Repositorio** es una interfaz que define métodos para poder operar en la base de datos. Spring luego crea las implementaciones correspondientes y las inyecta donde son necesarias.

### 7.1. Prerrequisitos

En un ambiente de producción la información de la conexión a bases de datos no debe ser parte de la aplicación; para esto existe JNDI<sup>16</sup>.

En pocas palabras, con JNDI el servidor crea un recurso – en este caso una conexión a la base de datos – y lo publica con un nombre, para que después las aplicaciones puedan solicitarlo y sacar provecho de él.

La configuración de JNDI es dependiente del servidor y su explicación queda fuera del alcance de este curso, bastará para nosotros el crear el archivo /WEB-INF/jetty-env.xml con el contenido que se proporciona en el apéndice C.3 y el siguiente contenido al archivo web.xml dentro del elemento web-app.

```

1 <resource-ref>
2   <res-ref-name>jdbc/DemoCP</res-ref-name>
3   <res-type>javax.sql.DataSource</res-type>
4   <res-auth>Container</res-auth>
5 </resource-ref>
```

Con esto tenemos listo nuestro recurso con el nombre *jdbc/DemoCP* listo para que sea usado en nuestra aplicación.

### 7.2. Configuración

Al igual que con Spring webmvc es necesario agregar algunas dependencias al archivo pom.xml:

- *spring-data-jpa* Proporciona las clases y anotaciones que estaremos usando principalmente en esta sección.

```

1 <dependency>
2   <groupId>org.springframework.data</groupId>
3   <artifactId>spring-data-jpa</artifactId>
4   <version>1.11.3.RELEASE</version>
5 </dependency>
6
```

---

<sup>15</sup>Plain Old Java Object

<sup>16</sup>Java Naming and Directory Interface

- *hibernate-core* Proporciona herramientas para poder convertir objetos Java en entidades para la base de datos.

```

1 <dependency>
2   <groupId>org.hibernate</groupId>
3   <artifactId>hibernate-core</artifactId>
4   <version>5.2.10.Final</version>
5 </dependency>
6

```

- *spring-security-core* Conforma la parte central de Spring security, pero en esta ocasión nos proporcionará la clase BCrypt para poder manipular las contraseñas de los usuarios.

```

1 <dependency>
2   <groupId>org.springframework.security</groupId>
3   <artifactId>spring-security-core</artifactId>
4   <version>4.2.2.RELEASE</version>
5 </dependency>
6

```

Con esto, podemos proceder al archivo de configuración *spring-data-context.xml*, el cual debe contener lo siguiente

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jpa="http://www.springframework.org/schema/jpa"
4   xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans.xsd
5   http://www.springframework.org/schema/jpa http://www.springframework.org/schema/jpa.xsd">
6
7   <jpa:repositories base-package="com.mozcalti.cursos.springdemo.repositorios" />
8
9   <bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
10      <property name="jndiName" value="java:comp/env/jdbc/DemoCP" />
11   </bean>
12
13   <bean id="jpaVendorAdapter"
14     class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" />
15
16   <bean id="entityManagerFactory"
17     class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
18     <property name="packagesToScan" value="com.mozcalti.cursos.springdemo.entidades" />
19     <property name="dataSource" ref="dataSource" />
20     <property name="jpaVendorAdapter" ref="jpaVendorAdapter" />
21     <property name="jpaProperties">
22       <props>
23         <prop key="hibernate.dialect">org.hibernate.dialect.MySQL5InnoDBDialect</prop>
24         <prop key="javax.persistence.schema-generation.database.action">none</prop>
25       </props>

```

```

26          </property>
27      </bean>
28
29      <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
30          <property name="entityManagerFactory" ref="entityManagerFactory" />
31      </bean>
32
33  </beans>

```

En este archivo de configuración primero indicamos dónde deberán buscarse los repositorios; luego declaramos una serie de beans que interconectamos para preparar, finalmente, el manejador de transacciones que usará el sistema. Nótese que el bean *dataSource* lo solicitamos como un recurso JNDI. También, en el bean *entityManagerFactory* indicamos dónde estarán las entidades del sistema.

Finalmente, no olvidemos importar este archivo de configuración desde *spring-context.xml* con la línea

```
1  <import resource="spring-data-context.xml"/>
```

### 7.3. Entidades

Las entidades usadas en este curso son sencillas (fig. 2), pero son un buen ejemplo de cómo configurar una clase para que sea reconocida por Spring data.

En primer término tenemos la clase Rol, que cuenta sólo con un identificador y un nombre

```

1  package com.mozcalti.cursos.springdemo.entidades;
2
3  import java.io.Serializable;
4
5  import javax.persistence.Column;
6  import javax.persistence.Entity;
7  import javax.persistence.GeneratedValue;
8  import javax.persistence.GenerationType;
9  import javax.persistence.Id;
10 import javax.persistence.Table;
11
12 @Entity
13 @Table(name = "ROL")
14 public class Rol implements Serializable {
15
16     private Long id;
17     private String nombre;
18
19     public Rol() {
20     }
21
22     @Id
23     @GeneratedValue(strategy = GenerationType.IDENTITY)
24     @Column(name = "id")
25     public Long getId() {

```

```

26             return id;
27     }
28
29     @Column(name = "nombre")
30     public String getNombre() {
31         return nombre;
32     }
33
34     public void setId(Long id) {
35         this.id = id;
36     }
37
38     public void setNombre(String nombre) {
39         this.nombre = nombre;
40     }
41
42 }
```

Una entidad debe implementar la interfaz *Serializable* y tener un constructor vacío. Además de eso, sólo se utilizan algunas anotaciones:

**@Entity** Marca la clase para que Spring la identifique como una entidad.

**@Table** Indica el nombre de la tabla en la que se almacenará la información de esta entidad.

**@Id** Señala el atributo como el identificador de la entidad.

**@GeneratedValue** Significa que este valor será proporcionado por la base de datos. Se puede indicar una estrategia como *SEQUENCE*, *TABLE* o en este caso *IDENTITY* para los valores autoincrementales de MySQL.

**@Column** Indica que este atributo es una de las columnas de la tabla correspondiente. Se puede proporcionar el nombre específico de la columna.

Las anotaciones de los atributos pueden hacerse indistintamente al momento de declararlos, en los *setters* o en los *getters*, pero esta última opción es la que permite un mejor desempeño.

Ahora nos queda la entidad Usuario:

En esta ocasión podemos ver un par de anotaciones nuevas

**@JsonIgnore** Esta anotación no es de Spring data, sino de jackson-databind. Como dijimos antes, esta dependencia nos permite transformar fácilmente entre objetos java y objetos JSON para poder enviarlos en la respuesta de nuestra aplicación. Si existe un atributo que no querremos incluir – como en este caso la contraseña, por cuestiones de seguridad – basta con agregar esta anotación.

**@ManyToOne** Este es un indicador de multiplicidad. Estos se utilizan cuando el atributo es otro objeto y puede ser *@OneToOne*, *@ManyToOne*, *@OneToMany* o *@ManyToMany*.

**@JoinColumn** Esta anotación se usa en conjunto con *@ManyToOne* y define la forma en que las dos entidades se relacionan. En el caso de *@ManyToMany* se usa *@JoinTable*.

## 7.4. Repositorios

Con las entidades listas, es momento de crear nuestros repositorios.

*Spring data* nos proporciona una serie de interfaces predefinidas con los métodos más generales. Nosotros debemos extenderlas para especificar métodos específicos de nuestras entidades.

Como dijimos antes, los repositorios son interfaces y es Spring quien crea las clases concretas. Al pensarlo, podría parecer muy complicado definir estos métodos específicos, ya que ¿cómo sabría Spring la forma de implementarlos?

Afortunadamente para nosotros podemos establecer el comportamiento de la (efímera) clase por medio del nombre del método.

Si, por ejemplo, tenemos una entidad *Pelota* con un atributo de tipo *String* llamado *color* podemos definir el siguiente método en el repositorio correspondiente

```
1 Set<Pelota> findByColor(String color);
```

el cual nos devolvería el Set de Pelotas que tengan el color que proporcionamos.

Además, si por alguna razón no podemos hacer uso de esta característica, también disponemos de la anotación *@Query* para definir, en jpql, nuestro query.

Sabiendo esto, definiremos el repositorio para la entidad Rol sin mayor modificación.

```
1 package com.mozcalti.cursos.springdemo.repositorios;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 import com.mozcalti.cursos.springdemo.entidades.Rol;
6
7 public interface RolRepository extends CrudRepository<Rol, Long> {
8
9     Rol findByNombre(String nombre);
10
11 }
```

Y agregaremos un método específico al repositorio para la entidad Usuario que devuelva un usuario dado su *username*.

Al extender la interfaz *CrudRepository* se indican, por medio de *generics* de java, la entidad que manejará y el tipo de objeto que usa como identificador.

También cabe señalar que no agregamos un método que recupere una entidad por su identificador, ya que este es uno de los métodos generales que mencionamos (*findOne*) junto con el que guarda una entidad (*save*).

## 7.5. Capa se servicios

En un sistema real, es en la capa de servicios donde se implementan todas las reglas de negocio. Es quizás la parte más importante, y también donde Spring nos deja más trabajo.

Repasemos los requisitos de nuestro sistema.

- La aplicación debe permitir la administración de usuarios (creación, modificación, recuperación y eliminación).

- Los parámetros que se pueden modificar son el rol y la contraseña.
- La contraseña deberá estar cifrada con *BCrypt*.

En primer término crearemos una clase de soporte que nos permita cifrar la contraseña.

```

1 package com.mozcalti.cursos.springdemo.herramientas;
2
3 import org.springframework.security.crypto.bcrypt.BCrypt;
4 import org.springframework.stereotype.Component;
5
6 @Component
7 public class SoportePassword {
8
9     public String cifrar(String password) {
10         String salt = BCrypt.gensalt();
11
12         return BCrypt.hashpw(password, salt);
13     }
14
15 }
```

Esta clase cuenta con un único método que, a su vez, utiliza dos métodos de *BCrypt* para cifrar nuestra contraseña.

Vale la pena señalar la anotación *@Component* en la clase. Esta indica a Spring que *SoportePassword* formará parte – es un componente – de otras clases, y deberá crear un *bean* cuando escanee el árbol de paquetes.

Ahora toca crear la clase de servicio de Usuario

```

1 package com.mozcalti.cursos.springdemo.servicios;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.security.access.annotation.Secured;
5 import org.springframework.stereotype.Service;
6
7 import com.mozcalti.cursos.springdemo.entidades.Rol;
8 import com.mozcalti.cursos.springdemo.entidades.Usuario;
9 import com.mozcalti.cursos.springdemo.herramientas.SoportePassword;
10 import com.mozcalti.cursos.springdemo.repositorios.RolRepositorio;
11 import com.mozcalti.cursos.springdemo.repositorios.UsuarioRepositorio;
12
13 @Service
14 public class ServicioUsuario {
15
16     private SoportePassword soportePassword;
17     private UsuarioRepositorio usuarioRepositorio;
18     private RolRepositorio rolRepositorio;
19
20     @Secured("ROLE_ADMINISTRADOR")
21     public Usuario crearUsuario(String username, String password) {
```

```

22     Usuario usuario = new Usuario();
23     usuario.setUsername(username);
24     usuario.setPassword(soportePassword.cifrar(password));
25
26         return usuarioRepositorio.save(usuario);
27     }
28
29     @Secured({"ROLE_ADMINISTRADOR", "ROLE_USUARIO"})
30     public Usuario recuperarUsuario(Long idUsuario) {
31         return usuarioRepositorio.findOne(idUsuario);
32     }
33
34     public Usuario recuperarUsuario(String username) {
35         return usuarioRepositorio.findByUsername(username);
36     }
37
38     @Secured("ROLE_ADMINISTRADOR")
39     public Usuario asignarRol(Long idUsuario, Long idRol) {
40         Usuario usuario = usuarioRepositorio.findOne(idUsuario);
41         Rol rol = rolRepositorio.findOne(idRol);
42
43         usuario.setRol(rol);
44
45         return usuarioRepositorio.save(usuario);
46     }
47
48
49     @Secured("ROLE_ADMINISTRADOR")
50     public Usuario cambiarPassword(Long idUsuario, String password) {
51         Usuario usuario = usuarioRepositorio.findOne(idUsuario);
52
53         usuario.setPassword(soportePassword.cifrar(password));
54
55         return usuarioRepositorio.save(usuario);
56     }
57
58     @Secured("ROLE_ADMINISTRADOR")
59     public String eliminarUsuario(Long idUsuario) {
60         usuarioRepositorio.delete(idUsuario);
61
62         return "Usuario eliminado";
63     }
64
65     @Autowired
66     public void setSoportePassword(SoportePassword soportePassword) {
67         this.soportePassword = soportePassword;
68     }

```

```

69
70     @Autowired
71     public void setUsuarioRepositorio(UsuarioRepositorio usuarioRepositorio) {
72         this.usuarioRepositorio = usuarioRepositorio;
73     }
74
75     @Autowired
76     public void setRolRepositorio(RolRepositorio rolRepositorio) {
77         this.rolRepositorio = rolRepositorio;
78     }
79
80 }

```

La clase *ServicioUsuario* hace uso de los repositorios y de la clase de soporte para proporcionar la funcionalidad requerida.

En esta ocasión marcamos la clase con la anotación `@Service` que, aunque es esencialmente equivalente a `@Component`, denota de mejor manera la intención. Esta etiqueta se prefiere justamente en las clases de la capa de servicios.

También usamos por primera vez la anotación `@Autowired`, que le indica a Spring que debe buscar en el contexto un bean de un tipo dado e insertarlo automáticamente. Esta anotación puede agregarse en la declaración de cada elemento, en su *getter* o en su *setter*, pero hay un mejor desempeño si se hace en este último.

## 7.6. Servicio Rest

Como lo hicimos en la sección anterior, crearemos los puntos de entrada para nuestra nueva funcionalidad. Tendremos `/usuario` como base común de los URL, y crearemos cada método para que cumpla con las reglas de *REST*.

```

1 package com.mozcalti.cursos.springdemo.controladores;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.web.bind.annotation.PathVariable;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RequestMethod;
7 import org.springframework.web.bind.annotation.RequestParam;
8 import org.springframework.web.bind.annotation.RestController;
9
10 import com.mozcalti.cursos.springdemo.entidades.Usuario;
11 import com.mozcalti.cursos.springdemo.servicios.ServicioUsuario;
12
13 @RestController
14 @RequestMapping("/usuario")
15 public class UsuarioControlador {
16
17     private ServicioUsuario servicioUsuario;
18
19     @RequestMapping(value = "", method = RequestMethod.POST)
20     public Usuario nuevoUsuario(String username, String password) {

```

```

21         return servicioUsuario.crearUsuario(username, password);
22     }
23
24     @RequestMapping(value = "/{idUsuario}", method = RequestMethod.GET)
25     public Usuario obtenerUsuario(@PathVariable Long idUsuario) {
26         return servicioUsuario.recuperarUsuario(idUsuario);
27     }
28
29     @RequestMapping(value = "/{idUsuario}/password", method = RequestMethod.PUT)
30     public Usuario cambiarPassword(@PathVariable Long idUsuario, @RequestParam String password) {
31         return servicioUsuario.cambiarPassword(idUsuario, password);
32     }
33
34     @RequestMapping(value = "/{idUsuario}/rol", method = RequestMethod.PUT)
35     public Usuario cambiarRol(@PathVariable Long idUsuario, @RequestParam Long idRol) {
36         return servicioUsuario.asignarRol(idUsuario, idRol);
37     }
38
39     @RequestMapping(value = "/{idUsuario}", method = RequestMethod.DELETE)
40     public String eliminarUsuario(@PathVariable Long idUsuario) {
41         return servicioUsuario.eliminarUsuario(idUsuario);
42     }
43
44     @Autowired
45     public void setServicioUsuario(ServicioUsuario servicioUsuario) {
46         this.servicioUsuario = servicioUsuario;
47     }
48
49 }
```

Queda poco que explicar sobre la configuración, pero sí cabe señalar que hay dos métodos vinculados con la url /usuario/{idUsuario}, sin embargo se diferencian por el verbo HTML que esperan, esto es suficiente para que Spring puede redirigir la petición al método correcto.

Es también por eso que los métodos que modifican el usuario tienen un nivel extra en la URI, para poder distinguir entre ellos.

Con todo esto listo, podemos volver a iniciar el servidor y probar que todo esté funcionando adecuadamente.

#### **Nota Importante**

La contraseña se cifra sólo para ser almacenada, y es un dato que viaja desde el cliente hasta el servidor en *texto claro*, por lo que la seguridad depende de que la comunicación no quede comprometida. Lo mismo aplica cuando se hace una petición de login.

Es por esto que nuestros servidores de producción siempre deben estar configurados para usar un canal seguro (HTTPS).

## 7.7. Conclusión

Una vez más, *Spring data* nos proporciona un *framework* con el que, luego de un poco de configuración, estamos listos para manipular nuestra base de datos.



Así, con *Spring webmvc* y *Spring data* podemos concentrarnos en la implementación de la capa de servicios, que es la parte fundamental de toda aplicación.

Ahora estamos listos para la etapa final de este curso: implementar una capa de seguridad basada en roles que nos permita delimitar el acceso a la aplicación.

## 8. Spring security

Como ya hemos visto, los diferentes módulos de Spring nos facilitan la implementación de tareas rutinarias, algunas veces reduciéndolo a un problema de configuración.

La seguridad de las aplicaciones no es la excepción, *Spring security* se encarga de la mayoría de las tareas necesarias para que nosotros podamos concentrarnos en asuntos más específicos.

Spring Security es un *framework* dedicado a proveer un conjunto completo de servicios de seguridad para aplicaciones Java de una forma flexible y amistosa con el desarrollador. Se apega a prácticas bien establecidas presentadas por *Spring Framework*. Spring Security administra todas las capas de seguridad dentro de tu aplicación. Además, cuenta con un extenso conjunto de opciones configurables que lo hacen flexible y poderoso.

[...] Puede decirse que Spring Security es simplemente un *framework* completo de autenticación/autorización construido sobre *Spring Framework*. Aunque la mayoría de las aplicaciones que lo usan son web, el núcleo de Spring Security se puede usar en aplicaciones *standalone* [2]

### 8.1. Conceptos básicos de seguridad

El siguiente es un extracto de Spring Security Pro [2].

#### Autenticación

El proceso de autenticación permite validar que un usuario en particular es quien dice ser. En este proceso, un usuario le presenta a la aplicación información sobre si mismo (normalmente, un nombre de usuario y una contraseña) que nadie más conoce. La aplicación toma esto e intenta hacerla coincidir con la que tiene almacenada – normalmente una base de datos o un servidor LDAP. Si la información proporcionada coincide con un registro, se dice que el usuario se ha autenticado exitosamente en el sistema.

#### Autorización

Cuando un usuario se autentica sólo significa que el usuario ha sido reconocido por el sistema. No significa que el usuario tiene permitido hacer lo que quiera. El siguiente paso lógico para asegurar una aplicación es determinar qué acciones puede ejecutar un usuario, a qué recursos tiene acceso y asegurar que no pueda realizar ninguna acción no permitida. Este trabajo corresponde al proceso de autorización. En el caso más común, este proceso compara el conjunto de permisos de un usuario con el permiso requerido para ejecutar una acción en la aplicación, y lo permite sólo si hay una coincidencia.

#### Conceptos generales de Autenticación y Autorización

**Usuario** El primer paso para asegurar un sistema de atacantes maliciosos es identificar a los usuarios legítimos y sólo permitirle el acceso a ellos. Una abstracción *User* se crea en el sistema y se le asigna su propia identidad. Este es el *usuario* que luego será permitido en el sistema.

**Credenciales** Las credenciales son la forma en que un usuario demuestra quién es. Normalmente es una contraseña (aunque los certificados también son comunes), y es la información que sólo conoce el propietario.

**Rol** En el contexto de seguridad de una aplicación un rol puede interpretarse como una agrupación lógica de usuarios. Esta agrupación se hace de modo que los usuarios comparten un grupo de permisos para ciertos recursos. Por ejemplo, todos los usuarios con rol "admin" tendrán los

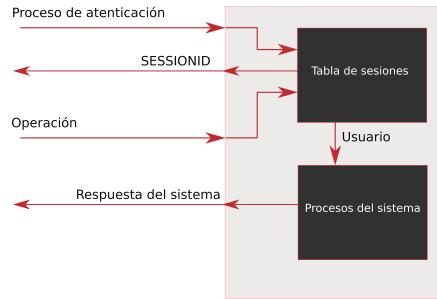


Figura 6: Proceso de autenticación con SESSIONID.

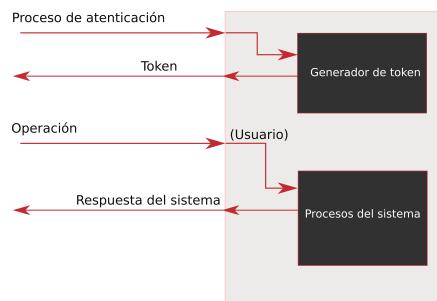


Figura 7: Proceso de autenticación con un Token de autenticación.

mismos permisos en los mismos recursos. Los roles sirven simplemente como un modo de agrupar permisos para ejecutar determinadas acciones, haciendo que los usuarios con ese rol hereden esos permisos.

**Recurso** Por recurso se entiende, en este contexto, cualquier parte de la aplicación a la que queremos acceder y que necesita estar asegurada adecuadamente para prevenir su uso no autorizado, por ejemplo una URL, un método de negocio o un objeto particular.

## 8.2. JSON Web Token

En servidores con estado (*statefull*) suele usarse un *SESSIONID*, esto es, un identificador que se asocia en el servidor con la información del usuario. Cuando posteriormente el cliente solicita una operación al sistema, envía este identificador que, de forma interna, es usado para recuperar una abstracción del Usuario (fig. 6).

Estos identificadores deben almacenarse en el servidor – generalmente en una base de datos en memoria – lo que representa un problema de escalabilidad.

Como una alternativa se propuso la implementación de un *token* de autenticación. En muchos aspectos es similar al identificador de sesión, siendo la principal diferencia que el *token* lleva consigo mismo la información del usuario, haciendo innecesario el almacenarla en el servidor y convirtiéndolo en un servidor sin estado (*stateless*) (fig. 7).

Como es usual, cada enfoque tiene sus ventajas y desventajas. En un servidor con *SESSIONID* invalidar una sesión es tan sencillo como eliminar el registro de la base de datos, mientras que invalidar un *Token* puede resultar

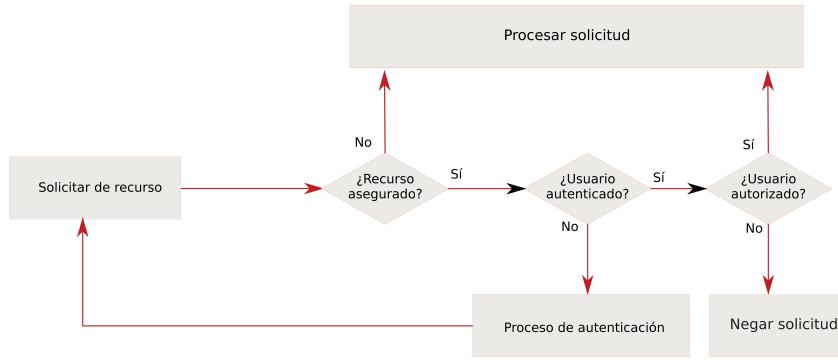


Figura 8: Flujo predeterminado de Spring security

más complicado<sup>17</sup>. Ambos pueden almacenarse en una *cookie*<sup>18</sup> en el navegador y también ambos pueden prestarse a un ataque MITM para el robo de sesión<sup>19</sup>.

JSON Web Token (JWT) es una representación compacta y compatible con URL para transferir *pretensiones* entre dos partes. Las *pretensiones* en un JWT están codificadas como un objeto JSON que se usa como *payload* en la estructura de un *JSON Web Signature* (JWS) o como el texto plano de la estructura de un *JSON Web Encryption* (JWE), permitiendo que las *pretensiones* sean firmadas digitalmente o su integridad protegida con un Mensaje de Autenticación de Código (MAC) y/o cifradas.[3]

JWT es un estandar abierto que ofrece una implementación de un token de autenticación. Esencialmente consiste en un objeto JSON codificado en base64 y que opcionalmente puede ser firmado y/o cifrado por el servidor para garantizar su integridad y seguridad<sup>20</sup>.

### 8.3. Flujo de seguridad

El flujo predeterminado de Spring security (fig. 8) establece que si un usuario no autenticado solicita un recurso protegido, debe ser redirigido al proceso de autenticación – generalmente una página de login –, sin embargo este no es adecuado para un servicio REST ya que no se cuenta con una capa de vista propiamente dicha.

Deberemos modificar el flujo para que en este escenario la petición simplemente sea denegada (fig. 9).

Similarmente, cuando se tiene una autenticación exitosa, Spring envía un redireccionamiento – ya sea a una página predeterminada o a la solicitud que inició el proceso de autenticación –, pero por la misma razón que el caso anterior esto no se adecúa a nuestras necesidades, por lo que deberemos modificar también este caso para que simplemente regrese un estatus 200 Ok.

<sup>17</sup>Se recomienda agregar siempre una vigencia al token, forzando una reautenticación periódica para limitar el mal uso de un token interceptado.

<sup>18</sup>Almacenar el token de autenticación de esta forma, aunque posible, no debería hacerse en servicios *RESTfull* ya que abre la puerta a ataques CSRF. Pero en realidad no tenemos control sobre el código del cliente, por lo que sólo podemos hacer la recomendación correspondiente.

<sup>19</sup>Una vez más resaltamos la importancia de usar un canal seguro (HTTPS) para las comunicaciones de nuestra aplicación una vez en ambiente de producción.

<sup>20</sup>Aunque ambas características son opcionales, se recomienda tratar la firma como obligatoria, para prevenir que el token pueda ser falsificado.

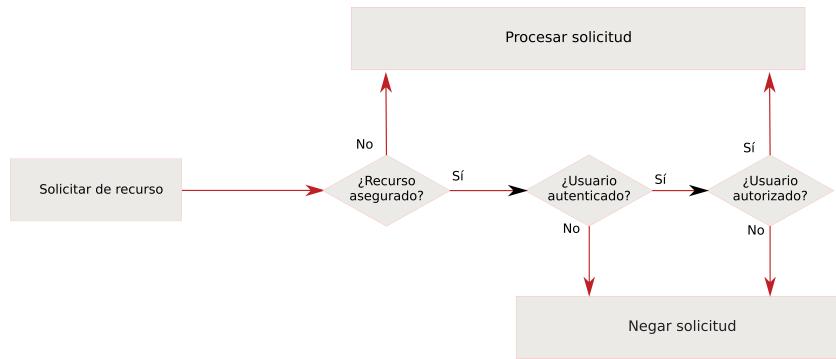


Figura 9: Flujo de Spring security para servicios REST

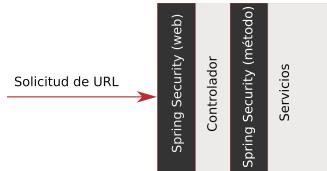


Figura 10: Diferentes puntos de intercepción de Spring security

#### 8.4. Seguridad a nivel web y seguridad a nivel de método

Spring security nos permite dos enfoques para asegurar nuestra aplicación. Ambas conceden o rechazan el uso de un recurso según las reglas establecidas, pero lo hacen en momentos diferentes. Si lo hacemos a *nivel web*, la decisión será tomada antes de que la clase controladora (`@RestController`) sea llamada, y si lo hacemos a *nivel de método*, la decisión será antes de invocar el método en la capa de servicios<sup>21</sup>.

El siguiente es un extracto de Spring Security Pro [2].

[La seguridad a nivel web] es muy conveniente y poderosa; sin embargo, no es una solución 100 % efectiva para todos los casos. Las mayores preocupaciones al aplicar únicamente este tipo de seguridad son tanto funcionales como de conveniencia, como se va a explicar.

Primero, por definición, la seguridad en la capa web aplica sólo a aplicaciones web, lo que la hace inutilizable para otro tipo de aplicaciones Java. Aunque Spring Security se enfoca principalmente en asegurar aplicaciones web, no hay motivo por el que algunas de sus partes no puedan ser usadas para otro tipo de aplicaciones.

Segundo, el mecanismo de seguridad por patrones de URL, aunque flexible, requiere que el desarrollador adopte ciertas reglas o convenciones solamente por razones de seguridad (como crear una ruta `/admin/` para administradores y establecer las reglas correspondientes).

Tercero, la seguridad al nivel de URL establece un criterio de *grano grueso*, ya que las URL son los puntos de entrada a la aplicación. Esto significa que las limitaciones de seguridad se ejecutan por

<sup>21</sup>Este nivel de seguridad no es exclusivo de la capa de servicios y puede intervenir en cualquier parte del flujo de nuestra aplicación, pero se acostumbra limitarse a este nivel y/o en los repositorios.

cada solicitud, reduciendo notablemente la flexibilidad. Por ejemplo, si se desea garantizar que un Objeto de Acceso de Datos (DAO) en la aplicación sea llamado sólamente por un administrador, no se puede lograr sólamente con el nivel de seguridad web. Se necesitaría garantizar que todas las URL que llaman a ese DAO estuvieran limitadas a administradores. Si, por alguna razón, existe una URL que no esté asegurada, esa solicitud alcanzará el DAO libremente, y ejecutará una operación potencialmente peligrosa.

Por esta razón, y al menos para este ejemplo, estableceremos reglas de acceso a nivel web sólo pidiendo que un usuario esté autenticado, mientras que aseguraremos los métodos de la capa de servicios para que solamente puedan invocarlos los usuarios autorizados.

## 8.5. Configuración

Una vez más, el primer paso es agregar las dependencias de maven al archivo pom.xml

**spring-security-core** Proporciona las funcionalidades básicas de Spring security. En nuestro caso esta dependencia ya fue agregada en la sección anterior para poder hacer uso de la clase BCrypt.

```

1 <dependency>
2   <groupId>org.springframework.security</groupId>
3   <artifactId>spring-security-core</artifactId>
4   <version>4.2.2.RELEASE</version>
5 </dependency>
6

```

**spring-security-web** Características específicas para servicios web.

```

1 <dependency>
2   <groupId>org.springframework.security</groupId>
3   <artifactId>spring-security-web</artifactId>
4   <version>4.2.2.RELEASE</version>
5 </dependency>
6

```

**jjwt** Implementación de JWT para java.

```

1 <dependency>
2   <groupId>io.jsonwebtoken</groupId>
3   <artifactId>jjwt</artifactId>
4   <version>0.6.0</version>
5 </dependency>
6

```

**javax-servlet-api** Esta librería es proveída por el servidor, así que la agregaremos con ese alcance (provided). Es necesaria para tener acceso a clases como HttpServletRequest y HttpServletResponse.

```

1 <dependency>
2   <groupId>javax.servlet</groupId>
3   <artifactId>javax.servlet-api</artifactId>
4   <version>4.0.0-b06</version>
5   <scope>provided</scope>
6 </dependency>
7

```

**spring-security-config** Proporciona el manejador del namespace necesario para poder configurar el contexto de seguridad.

```

1 <dependency>
2   <groupId>org.springframework.security</groupId>
3   <artifactId>spring-security-config</artifactId>
4   <version>4.2.2.RELEASE</version>
5 </dependency>

```

Spring security se implementa como una serie de filtros en el *servlet*, así que para activarlo debemos agregar las siguientes líneas en el archivo web.xml

```

1 <filter>
2   <filter-name>springSecurityFilterChain</filter-name>
3   <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
4 </filter>
5
6 <filter-mapping>
7   <filter-name>springSecurityFilterChain</filter-name>
8   <url-pattern>/*</url-pattern>
9 </filter-mapping>

```

Con esto crearmo un filtro llamado *springSecurityFilterChain* – el filtro debe tener exactamente ese nombre – que se aplicará a todas las rutas de nuestra aplicación.

Como se dijo, Spring security requerirá un poco más de trabajo del que hemos hecho hasta ahora ya que hay que adaptarlo para nuestras necesidades<sup>22</sup>.

Tendremos que implementar algunas interfaces y crear algunas clases de soporte:

**CustomGrantedAuthority** Representa un conjunto de permisos en el sistema. En nuestro caso sólamente será un *wrapper* del Rol.

```

1 package com.mozcalti.cursos.springdemo.seguridad;
2 import org.springframework.security.core.GrantedAuthority;
3
4 import com.mozcalti.cursos.springdemo.entidades.Rol;
5
6 public class CustomGrantedAuthority implements GrantedAuthority {
7

```

---

<sup>22</sup>Toda la familia de Spring está en constante evolución, no sería sorprendente que dentro de poco exista una forma más sencilla de proteger servicios REST, pero por ahora aún tenemos que ensuciarnos las manos.

```

8     private Rol rol;
9
10    public CustomGrantedAuthority(Rol rol) {
11        this.rol = rol;
12    }
13
14    public String getAuthority() {
15        return "ROLE_" + rol.getNombre();
16    }
17
18 }

```

**CustomUserDetails** Representa la información detallada de un Usuario.

```

1 package com.mozcalti.cursos.springdemo.seguridad;
2
3 import java.util.Collection;
4 import java.util.HashSet;
5 import java.util.Set;
6
7 import org.springframework.security.core.GrantedAuthority;
8 import org.springframework.security.core.userdetails.UserDetails;
9
10 import com.mozcalti.cursos.springdemo.entidades.Usuario;
11
12 public class CustomUserDetails implements UserDetails {
13
14     private Usuario usuario;
15
16     public CustomUserDetails(Usuario usuario) {
17         this.usuario = usuario;
18     }
19
20     public Collection<? extends GrantedAuthority> getAuthorities() {
21
22         Set<GrantedAuthority> authorities = new HashSet<>();
23
24         authorities.add(new CustomGrantedAuthority(usuario.getRol()));
25
26         return authorities;
27     }
28
29     public String getPassword() {
30         return usuario.getPassword();
31     }
32
33     public String getUsername() {

```

```

34         return usuario.getUsername();
35     }
36
37     public boolean isAccountNonExpired() {
38
39         return true;
40     }
41
42     public boolean isAccountNonLocked() {
43
44         return true;
45     }
46
47     public boolean isCredentialsNonExpired() {
48
49         return true;
50     }
51
52     public boolean isEnabled() {
53         return true;
54     }
55
56 }
```

Como se puede observar, existe una amplia variedad de parámetros que podemos personalizar en la cuenta de un usuario, pero para este ejemplo mantendremos el ejemplo sencillo y devolveremos valores default en la mayoría de los métodos.

**CustomUserDetailsService** Esta interfaz define un servicio que, dado un nombre de usuario, devuelve un objeto UserDetails.

```

1 package com.mozcalti.cursos.springdemo.seguridad;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.security.core.userdetails.UserDetails;
5 import org.springframework.security.core.userdetails.UserDetailsService;
6 import org.springframework.security.core.userdetails.UsernameNotFoundException;
7
8 import com.mozcalti.cursos.springdemo.servicios.ServicioUsuario;
9
10 public class CustomUserDetailsService implements UserDetailsService {
11
12     private ServicioUsuario servicioUsuario;
13
14     @Override
15     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
16         return new CustomUserDetails(servicioUsuario.recuperarUsuario(username));
17     }
18 }
```

```

17     }
18
19     @Autowired
20     public void setServicioUsuario(ServicioUsuario servicioUsuario) {
21         this.servicioUsuario = servicioUsuario;
22     }
23
24 }
```

**SoporteToken** Se encarga de crear y validar los token.

```

1 package com.mozcalti.cursos.springdemo.seguridad;
2
3 import java.util.Calendar;
4 import java.util.Date;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.security.core.userdetails.UserDetails;
8 import org.springframework.stereotype.Component;
9
10 import com.mozcalti.cursos.springdemo.servicios.ServicioUsuario;
11
12 import io.jsonwebtoken.Claims;
13 import io.jsonwebtoken.Jwts;
14 import io.jsonwebtoken.SignatureAlgorithm;
15
16 @Component
17 public class SoporteToken {
18
19     private static final String LLAVE = "v1X9IngjtjNxNlQOZKOPiyTnhu7YK00omCldPIMhXHCn030B";
20     private static final int HORAS_VIGENCIA = 8;
21
22     private ServicioUsuario servicioUsuario;
23
24     public String crearToken(UserDetails userDetails) {
25         Claims claims = Jwts.claims();
26
27         claims.setSubject(userDetails.getUsername());
28         claims.setExpiration(calcularVigencia());
29
30         return Jwts.builder().setClaims(claims).signWith(SignatureAlgorithm.HS512, LLAVE);
31     }
32
33
34     public UserDetails recuperarUsuario(String token) {
35         Claims claims = Jwts.parser().setSigningKey(LLAVE).parseClaimsJws(token).get();
36 }
```

```

37         if (esVigente(claims)) {
38
39             String username = claims.getSubject();
40
41             return new CustomUserDetails(servicioUsuario.recuperarUsuario(username));
42         } else {
43             return null;
44         }
45     }
46
47
48     private boolean esVigente(Claims claims) {
49         return claims.getExpiration().after(new Date());
50     }
51
52     private Date calcularVigencia() {
53         Calendar hoy = Calendar.getInstance();
54
55         hoy.add(Calendar.HOUR, HORAS_VIGENCIA);
56
57         return hoy.getTime();
58     }
59
60     @Autowired
61     public void setServicioUsuario(ServicioUsuario servicioUsuario) {
62         this.servicioUsuario = servicioUsuario;
63     }
64
65 }
```

Cabe señalar que para este ejemplo los valores de *LLAVEy HORAS\_VIGENCIA* han sido escritas en código duro, pero en una aplicación real deberían ser leídas desde un archivo de configuración.

**ServicioAutenticacionToken** Provee métodos para agregar el token a la respuesta del servidor y para recuperar el objeto Authentication de la solicitud si se encuentra un token.

```

1 package com.mozcalti.cursos.springdemo.seguridad;
2
3 import javax.servlet.http.HttpServletRequest;
4 import javax.servlet.http.HttpServletResponse;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.security.core.Authentication;
8 import org.springframework.security.core.userdetails.UserDetails;
9 import org.springframework.stereotype.Component;
10
11 @Component
```

```

12 public class ServicioAutenticacionToken {
13
14     private static final String HEADER_NAME = "X-TOKEN";
15
16     private SoporteToken soporteToken;
17
18     public void agregarToken(HttpServletRequest response, Authentication authentication) {
19
20         UserDetails userDetails = (UserDetails) authentication.getPrincipal();
21
22         response.addHeader(HEADER_NAME, soporteToken.crearToken(userDetails));
23
24     }
25
26     public Authentication validarToken(HttpServletRequest request) {
27         String token = request.getHeader(HEADER_NAME);
28
29         if (token != null) {
30             UserDetails userDetails = soporteToken.recuperarUsuario(token);
31
32             if (userDetails != null) {
33                 return new CustomAuthentication(userDetails);
34             }
35
36         }
37
38         return null;
39     }
40
41     @Autowired
42     public void setSoporteToken(SoporteToken soporteToken) {
43         this.soporteToken = soporteToken;
44     }
45
46 }
```

**CustomEntryPoint** Establece el comportamiento del flujo de seguridad como lo queremos.

```

1 package com.mozcalti.cursos.springdemo.seguridad;
2
3 import java.io.IOException;
4
5 import javax.servlet.ServletException;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8
9 import org.springframework.security.core.AuthenticationException;
```

```

10 import org.springframework.security.web.AuthenticationEntryPoint;
11
12 public class CustomEntryPoint implements AuthenticationEntryPoint {
13
14     @Override
15     public void commence(HttpServletRequest request, HttpServletResponse response,
16                         AuthenticationException authException) throws IOException, ServletException {
17         response.sendError(HttpStatus.SC_UNAUTHORIZED);
18     }
19 }
20
21 }
```

**CustomAuthenticationSuccessHandler** Establece el comportamiento del flujo de seguridad como lo requerimos.

```

1 package com.mozcalti.cursos.springdemo.seguridad;
2
3 import java.io.IOException;
4
5 import javax.servlet.ServletException;
6 import javax.servlet.http.HttpServletRequest;
7 import javax.servlet.http.HttpServletResponse;
8
9 import org.springframework.beans.factory.annotation.Autowired;
10 import org.springframework.security.core.Authentication;
11 import org.springframework.security.web.authentication.AuthenticationSuccessHandler;
12
13 public class CustomAuthenticationSuccessHandler implements AuthenticationSuccessHandler {
14
15     private ServicioAutenticacionToken servicioAutenticacionToken;
16
17     @Override
18     public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
19                                         Authentication authentication) throws IOException, ServletException {
20
21         servicioAutenticacionToken.agregarToken(response, authentication);
22
23         response.setStatus(HttpStatus.SC_OK);
24     }
25
26
27     @Autowired
28     public void setServicioAutenticacionToken(ServicioAutenticacionToken servicioAutenticacionToken) {
29         this.servicioAutenticacionToken = servicioAutenticacionToken;
30     }
31
32 }
```

**TokenValidationFilter** Es el filtro que revisará las peticiones y respuestas y manejará la autenticación por token.

```

1 package com.mozcalti.cursos.springdemo.seguridad;
2
3 import java.io.IOException;
4
5 import javax.servlet.FilterChain;
6 import javax.servlet.ServletException;
7 import javax.servlet.ServletRequest;
8 import javax.servlet.ServletResponse;
9 import javax.servlet.http.HttpServletRequest;
10
11 import org.springframework.beans.factory.annotation.Autowired;
12 import org.springframework.security.core.Authentication;
13 import org.springframework.security.core.context.SecurityContextHolder;
14 import org.springframework.web.filter.GenericFilterBean;
15
16 public class TokenValidationFilter extends GenericFilterBean {
17
18     private ServicioAutenticacionToken servicioAutenticacionToken;
19
20     @Override
21     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
22             throws IOException, ServletException {
23
24         HttpServletRequest httpRequest = (HttpServletRequest) request;
25
26         Authentication authentication = servicioAutenticacionToken.validarToken(httpRequest);
27
28         SecurityContextHolder.getContext().setAuthentication(authentication);
29
30         chain.doFilter(request, response);
31
32     }
33
34     @Autowired
35     public void setServicioAutenticacionToken(ServicioAutenticacionToken servicioAutenticacionToken) {
36         this.servicioAutenticacionToken = servicioAutenticacionToken;
37     }
38
39
40
41 }
```

Ya con todas estas clases listas, podemos configurar el archivo `spring-security-context.xml` con el siguiente contenido

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```

2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:security="http://www.springframework.org/schema/security"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans.xsd
6     http://www.springframework.org/schema/security http://www.springframework.org/schema/security.xsd"
7
8
9     <!-- Beans -->
10    <bean id="jwtFilter"
11        class="com.mozcalti.cursos.springdemo.seguridad.TokenValidationFilter" />
12
13    <bean id="userDeailService"
14        class="com.mozcalti.cursos.springdemo.seguridad.CustomUserDetailsService" />
15
16    <bean id="successHandler"
17        class="com.mozcalti.cursos.springdemo.seguridad.CustomAuthenticationSuccessHandler" />
18
19    <bean id="entryPoint"
20        class="com.mozcalti.cursos.springdemo.seguridad.CustomEntryPoint" />
21
22
23    <!-- Autenticación -->
24    <security:authentication-manager>
25        <security:authentication-provider
26            user-service-ref="userDeailService">
27                <security:password-encoder hash="bcrypt" />
28            </security:authentication-provider>
29    </security:authentication-manager>
30
31
32
33    <!-- Autorización -->
34    <security:global-method-security
35        secured-annotations="enabled" />
36
37    <security:http entry-point-ref="entryPoint"
38        create-session="stateless" use-expressions="true">
39
40        <security:csrf disabled="true" />
41
42        <security:custom-filter ref="jwtFilter" before="LOGIN_PAGE_FILTER" />
43
44        <security:intercept-url pattern="/login"
45            access="isAnonymous()" />
46
47        <security:intercept-url pattern="/prueba/**"
48            access="isAnonymous()" />

```

```

49
50      <security:intercept-url pattern="/**"
51          access="isAuthenticated()" />
52
53      <security:form-login
54          authentication-success-handler-ref="successHandler" />
55  </security:http>
56
57
58  </beans>

```

En primer término agregamos la definición de los beans que usaremos en este archivo de configuración<sup>23</sup>. Luego configuraremos la parte de autenticación, esta es bastante sencilla, sólo indicamos qué clase implementa *UserDetailsService* y que estaremos usando BCrypt para el manejo de contraseñas.

Finalmente viene la configuración de autorización. El elemento *security:global-method-security* indica que usaremos seguridad a nivel de método en nuestras clases. Igualmente el elemento *security:http* configura la seguridad a nivel web.

Como atributos de este elemento indicamos que el servicio no tendrá un estado, que usaremos expresiones para definir los criterios de seguridad y, finalmente, nuestro punto de entrada personalizado que modifica el comportamiento predefinido de Spring.

Desactivamos la protección CSRF, ya que este es un mecanismo de seguridad para prevenir el uso de cookies, y nuestra aplicación *RESTfull* no debería tener ese problema.

Agregamos el filtro de autenticación por token a la cadena de filtros de Spring security justo antes de que el sistema intente crear el servicio de login.

El elemento *security:form-login* crea un servicio de login y lo agrega en la ruta /login – este valor puede personalizarse –. Este servicio hará uso del administrador de autenticación que previamente definimos.

Para terminar, establecemos las reglas para el acceso a los recursos. Los URI de login y todos los de pruebas estarán disponibles al público, mientras que el resto requerirán que el usuario esté autenticado.

Ahora sólo queda modificar la clase *ServicioUsuario* para aplicar la seguridad a nivel de método. Esto se logra con la notación *@Secured*, que tiene como argumento el nombre de un rol – o una lista de nombres de roles – que puede invocar el método.

```

1 package com.mozcalti.cursos.springdemo.servicios;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.security.access.annotation.Secured;
5 import org.springframework.stereotype.Service;
6
7 import com.mozcalti.cursos.springdemo.entidades.Rol;
8 import com.mozcalti.cursos.springdemo.entidades.Usuario;
9 import com.mozcalti.cursos.springdemo.herramientas.SoportePassword;
10 import com.mozcalti.cursos.springdemo.repositorios.RolRepositorio;
11 import com.mozcalti.cursos.springdemo.repositorios.UsuarioRepositorio;
12
13 @Service
14 public class ServicioUsuario {

```

<sup>23</sup>El resto de las clases que creamos son usadas de forma interna por estas cuatro.

```

15
16     private SoportePassword soportePassword;
17     private UsuarioRepository usuarioRepository;
18     private RolRepository rolRepository;
19
20     @Secured("ROLE_ADMINISTRADOR")
21     public Usuario crearUsuario(String username, String password) {
22         Usuario usuario = new Usuario();
23         usuario.setUsername(username);
24         usuario.setPassword(soportePassword.cifrar(password));
25
26         return usuarioRepository.save(usuario);
27     }
28
29     @Secured({"ROLE_ADMINISTRADOR", "ROLE_USUARIO"})
30     public Usuario recuperarUsuario(Long idUsuario) {
31         return usuarioRepository.findOne(idUsuario);
32     }
33
34     public Usuario recuperarUsuario(String username) {
35         return usuarioRepository.findByUsername(username);
36     }
37
38     @Secured("ROLE_ADMINISTRADOR")
39     public Usuario asignarRol(Long idUsuario, Long idRol) {
40         Usuario usuario = usuarioRepository.findOne(idUsuario);
41         Rol rol = rolRepository.findOne(idRol);
42
43         usuario.setRol(rol);
44
45         return usuarioRepository.save(usuario);
46     }
47
48     @Secured("ROLE_ADMINISTRADOR")
49     public Usuario cambiarPassword(Long idUsuario, String password) {
50         Usuario usuario = usuarioRepository.findOne(idUsuario);
51
52         usuario.setPassword(soportePassword.cifrar(password));
53
54         return usuarioRepository.save(usuario);
55     }
56
57     @Secured("ROLE_ADMINISTRADOR")
58     public String eliminarUsuario(Long idUsuario) {
59         usuarioRepository.delete(idUsuario);
60
61

```

```

62             return "Usuario eliminado";
63     }
64
65     @Autowired
66     public void setSoportePassword(SoportePassword soportePassword) {
67         this.soportePassword = soportePassword;
68     }
69
70     @Autowired
71     public void setUsuarioRepositorio(UsuarioRepositorio usuarioRepositorio) {
72         this.usuarioRepositorio = usuarioRepositorio;
73     }
74
75     @Autowired
76     public void setRolRepositorio(RolRepositorio rolRepositorio) {
77         this.rolRepositorio = rolRepositorio;
78     }
79
80 }

```

Con esto terminamos de asegurar nuestra aplicación y podemos proceder a hacer las pruebas.

Si intentamos acceder a un URI sin hacer el login antes, entonces el filtro no agregará el objeto *Authentication* al contexto de seguridad y recibiremos una respuesta de recurso denegado.

Entonces, primero debemos hacer login y tomar el token de la cabecera de respuesta. Este token deberá ser enviado en toda solicitud posterior para poder acceder a los recursos (si es que se tienen los privilegios adecuados).

## 8.6. Conclusión

Aunque esta vez tuvimos que hacer un esfuerzo mayor al crear las clases personalizadas para adaptar el comportamiento de Spring security, en realidad es más lo que se gana, ya que podemos aprovechar todo el poder del framework tan solo implementando algunas clases con código sencillo.

También debemos recordar que Spring security, aunque robusto, sólo cubre los aspectos de seguridad a partir de nuestro servlet, por lo que no debemos descartar el aspecto de la seguridad en otros niveles (el servidor, el canal de conexión, etc.).

## 9. Siguientes pasos

En este momento estamos en condiciones de crear una aplicación *RESTfull* protegida agregando objetos de negocio que cumplan los requisitos solicitados por algún cliente.

Este tipo de sistemas suelen ofrecerse como una API para ser consumidos. Un ejemplo cada vez más común es un sitio web estático que se conecta a un sistema backend – como el que acabamos de mostrar – para hacerlo dinámico<sup>24</sup>. El desarrollo de este tipo de sistemas requiere un paso extra: la configuración de las cabeceras CORS (*Cross-Origin-Resource-Sharing*), ya que la política de *mismo origen* de los navegadores bloquea peticiones a otros dominios.

Afortunadamente, tanto Spring webmvc como Spring security son fácilmente configurables en este aspecto.

También queda profundizar en los módulos que se han presentado en esta guía. Spring data, por ejemplo, tiene una variedad de interfaces de las que podemos extender nuestros repositorios, proporcionándonos métodos para ordenar o paginar nuestros datos. Spring webmvc puede recibir no sólo tipos *básicos* como enteros o cadenas, sino objetos complejos – gracias a jackson – como nuestro Usuario. Y no sobra mencionar el norme abanico de configuraciones que tiene Spring security. Estas son sólo algunas características de sólo algunos de los módulos de Spring. Para saber más, el sitio oficial es siempre un buen punto de inicio, sobre todo por la rica documentación que caracteriza al equipo. <https://spring.io/projects>

---

<sup>24</sup>Para esto existen varias opciones dentro del ecosistema de javascript, como son JQuery, ReactJS o AngularJS.

## Referencias

- [1] D. Booth, H. Hass, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web services architecture."
- [2] C. Scarioni, *Pro Spring Security*. Apress, 2013.
- [3] M. Jones, J. Bradley, and N. Sakimura, "Json web token (jwt)," tech. rep., IETF, May 2015.

## Apéndice A Scripts de la base de datos

### A.1 Esquema

```

1 -- MySQL Script generated by MySQL Workbench
2 -- Mon May 29 12:38:37 2017
3 -- Model: New Model    Version: 1.0
4 -- MySQL Workbench Forward Engineering
5
6 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
7 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
8 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
9
10 -----
11 -- Schema SpringDemo
12 -----
13 DROP SCHEMA IF EXISTS 'SpringDemo' ;
14
15 -----
16 -- Schema SpringDemo
17 -----
18 CREATE SCHEMA IF NOT EXISTS 'SpringDemo' DEFAULT CHARACTER SET utf8 ;
19 USE 'SpringDemo' ;
20
21 -----
22 -- Table 'SpringDemo'.`ROL`
23 -----
24 CREATE TABLE IF NOT EXISTS 'SpringDemo'.`ROL` (
25     `id` INT NOT NULL AUTO_INCREMENT,
26     `nombre` VARCHAR(20) NULL,
27     PRIMARY KEY (`id`)
28 ) ENGINE = InnoDB;
29
30
31 -----
32 -- Table 'SpringDemo'.`USUARIO`
33 -----
34 CREATE TABLE IF NOT EXISTS 'SpringDemo'.`USUARIO` (
35     `id` INT NOT NULL AUTO_INCREMENT,
36     `username` VARCHAR(20) NOT NULL,
37     `password` VARCHAR(60) NOT NULL,
38     `fk_rol` INT NULL,
39     PRIMARY KEY (`id`),
40     INDEX `fk_USUARIO_ROL_idx` (`fk_rol` ASC),
41     CONSTRAINT `fk_USUARIO_ROL`
42       FOREIGN KEY (`fk_rol`)
43       REFERENCES 'SpringDemo'.`ROL` (`id`)

```

```
44      ON DELETE NO ACTION
45      ON UPDATE NO ACTION)
46 ENGINE = InnoDB;
47
48
49 SET SQL_MODE=@OLD_SQL_MODE;
50 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
51 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

## A.2 Creación de usuario

```
1 -----
2 -- Este script otorga privilegios de SELECT, INSERT, --
3 -- UPDATE y DELETE en el esquema SpringDemo      --
4 -----
5
6 grant SELECT,INSERT,UPDATE,DELETE on SpringDemo.* to 'DemoUser'@'localhost' identified by 'Passwo
```

## A.3 Datos iniciales

```
1 insert into ROL(nombre)
2 values ("ADMINISTRADOR"),("USUARIO");
```

## Apéndice B Archivos de configuración de Spring

### B.1 spring-context.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:context="http://www.springframework.org/schema/context"
4      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans.xsd
6          http://www.springframework.org/schema/context http://www.springframework.org/schema/context.xsd">
7
8      <context:component-scan base-package="com.mozcalti.cursos.springdemo" />
9
10     <import resource="spring-mvc-context.xml"/>
11     <import resource="spring-data-context.xml"/>
12     <import resource="spring-security-context.xml"/>
13
14 </beans>
```

### B.2 spring-mvc-context.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans.xsd
5          http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc.xsd">
6
7      <mvc:annotation-driven />
8
9 </beans>
```

### B.3 spring-data-context.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jpa="http://www.springframework.org/schema/jpa"
4      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans.xsd
5          http://www.springframework.org/schema/jpa http://www.springframework.org/schema/jpa.xsd">
6
7      <jpa:repositories base-package="com.mozcalti.cursos.springdemo.repositorios" />
8
9      <bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
10          <property name="jndiName" value="java:comp/env/jdbc/DemoCP" />
11      </bean>
12
13      <bean id="jpaVendorAdapter"
14          class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" />
```

```

15
16 <bean id="entityManagerFactory"
17     class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
18     <property name="packagesToScan" value="com.mozcalti.cursos.springdemo.entidades" />
19     <property name="dataSource" ref="dataSource" />
20     <property name="jpaVendorAdapter" ref="jpaVendorAdapter" />
21     <property name="jpaProperties">
22         <props>
23             <prop key="hibernate.dialect">org.hibernate.dialect.MySQL5InnoDBDialect</prop>
24             <prop key="javax.persistence.schema-generation.database.action">n</prop>
25         </props>
26     </property>
27 </bean>
28
29 <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
30     <property name="entityManagerFactory" ref="entityManagerFactory" />
31 </bean>
32
33 </beans>

```

#### B.4 spring-security-context.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:security="http://www.springframework.org/schema/security"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans.xsd
6     http://www.springframework.org/schema/security http://www.springframework.org/schema/security.xsd">
7
8
9     <!-- Beans -->
10    <bean id="jwtFilter"
11        class="com.mozcalti.cursos.springdemo.seguridad.TokenValidationFilter" />
12
13    <bean id="userDeailService"
14        class="com.mozcalti.cursos.springdemo.seguridad.CustomUserDetailsService" />
15
16    <bean id="successHandler"
17        class="com.mozcalti.cursos.springdemo.seguridad.CustomAuthenticationSuccessHandler" />
18
19    <bean id="entryPoint"
20        class="com.mozcalti.cursos.springdemo.seguridad.CustomEntryPoint" />
21
22
23    <!-- Autenticación -->
24    <security:authentication-manager>
25        <security:authentication-provider

```

```

26          user-service-ref="userDetailService">
27              <security:password-encoder hash="bcrypt" />
28      </security:authentication-provider>
29  </security:authentication-manager>
30
31
32
33      <!-- Autorización -->
34  <security:global-method-security
35      secured-annotations="enabled" />
36
37      <security:http entry-point-ref="entryPoint"
38          create-session="stateless" use-expressions="true">
39
40          <security:csrf disabled="true" />
41
42          <security:custom-filter ref="jwtFilter" before="LOGIN_PAGE_FILTER" />
43
44          <security:intercept-url pattern="/login"
45              access="isAnonymous()" />
46
47          <security:intercept-url pattern="/prueba/**"
48              access="isAnonymous()" />
49
50          <security:intercept-url pattern="/**"
51              access="isAuthenticated()" />
52
53          <security:form-login
54              authentication-success-handler-ref="successHandler" />
55      </security:http>
56
57
58  </beans>

```

## Apéndice C Otros archivos de configuración

### C.1 pom.xml

```

1  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
2      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_
3          <modelVersion>4.0.0</modelVersion>
4          <groupId>com.mozcalti.cursos</groupId>
5          <artifactId>springdemo</artifactId>
6          <packaging>war</packaging>
7          <version>0.0.1-SNAPSHOT</version>
8          <name>springdemo Maven Webapp</name>
9          <url>http://maven.apache.org</url>
10         <dependencies>
11             <dependency>
12                 <groupId>org.springframework</groupId>
13                 <artifactId>spring-webmvc</artifactId>
14                 <version>4.3.8.RELEASE</version>
15             </dependency>
16
17             <dependency>
18                 <groupId>com.fasterxml.jackson.core</groupId>
19                 <artifactId>jackson-databind</artifactId>
20                 <version>2.9.0.pr3</version>
21             </dependency>
22
23             <dependency>
24                 <groupId>org.springframework.data</groupId>
25                 <artifactId>spring-data-jpa</artifactId>
26                 <version>1.11.3.RELEASE</version>
27             </dependency>
28
29             <dependency>
30                 <groupId>org.hibernate</groupId>
31                 <artifactId>hibernate-core</artifactId>
32                 <version>5.2.10.Final</version>
33             </dependency>
34
35             <dependency>
36                 <groupId>org.springframework.security</groupId>
37                 <artifactId>spring-security-core</artifactId>
38                 <version>4.2.2.RELEASE</version>
39             </dependency>
40
41             <dependency>
42                 <groupId>org.springframework.security</groupId>
43                 <artifactId>spring-security-web</artifactId>

```

```

44      <version>4.2.2.RELEASE</version>
45  </dependency>
46
47  <dependency>
48      <groupId>io.jsonwebtoken</groupId>
49      <artifactId>jjwt</artifactId>
50      <version>0.6.0</version>
51  </dependency>
52
53  <dependency>
54      <groupId>javax.servlet</groupId>
55      <artifactId>javax.servlet-api</artifactId>
56      <version>4.0.0-b06</version>
57      <scope>provided</scope>
58  </dependency>
59
60  <dependency>
61      <groupId>org.springframework.security</groupId>
62      <artifactId>spring-security-config</artifactId>
63      <version>4.2.2.RELEASE</version>
64  </dependency>
65
66  </dependencies>
67  <build>
68      <finalName>springdemo</finalName>
69
70      <plugins>
71          <plugin>
72              <groupId>org.apache.maven.plugins</groupId>
73              <artifactId>maven-compiler-plugin</artifactId>
74              <version>3.6.1</version>
75
76              <configuration>
77                  <target>1.8</target>
78                  <source>1.8</source>
79              </configuration>
80          </plugin>
81
82          <plugin>
83              <groupId>org.eclipse.jetty</groupId>
84              <artifactId>jetty-maven-plugin</artifactId>
85              <version>9.4.5.v20170502</version>
86              <dependencies>
87                  <dependency>
88                      <groupId>org.eclipse.jetty</groupId>
89                      <artifactId>jetty-plus</artifactId>
90                      <version>9.4.5.v20170502</version>

```

```

91      </dependency>
92      <dependency>
93          <groupId>org.eclipse.jetty</groupId>
94          <artifactId>jetty-jndi</artifactId>
95          <version>9.4.5.v20170502</version>
96      </dependency>
97      <dependency>
98          <groupId>com.zaxxer</groupId>
99          <artifactId>HikariCP</artifactId>
100         <version>2.6.1</version>
101     </dependency>
102     <dependency>
103         <groupId>mysql</groupId>
104         <artifactId>mysql-connector-java</artifactId>
105         <version>6.0.6</version>
106     </dependency>
107     <dependency>
108         <groupId>org.eclipse.jetty</groupId>
109         <artifactId>jetty-servlets</artifactId>
110         <version>9.4.5.v20170502</version>
111     </dependency>
112   </dependencies>
113 </plugin>
114
115 </plugins>
116 </build>
117 </project>

```

## C.2 web.xml

```

1  <!DOCTYPE web-app PUBLIC
2  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
3  "http://java.sun.com/dtd/web-app_2_3.dtd" >
4
5  <web-app>
6      <display-name>DemoSpring</display-name>
7
8      <resource-ref>
9          <res-ref-name>jdbc/DemoCP</res-ref-name>
10         <res-type>javax.sql.DataSource</res-type>
11         <res-auth>Container</res-auth>
12     </resource-ref>
13
14     <servlet>
15         <servlet-name>demoServlet</servlet-name>
16         <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
17

```

```

18    <init-param>
19        <param-name>contextConfigLocation</param-name>
20        <param-value>/WEB-INF/context/spring-context.xml</param-value>
21    </init-param>
22
23    <load-on-startup>1</load-on-startup>
24
25  </servlet>
26
27  <servlet-mapping>
28      <servlet-name>demoServlet</servlet-name>
29      <url-pattern>/*</url-pattern>
30  </servlet-mapping>
31
32  <filter>
33      <filter-name>springSecurityFilterChain</filter-name>
34      <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
35  </filter>
36
37  <filter-mapping>
38      <filter-name>springSecurityFilterChain</filter-name>
39      <url-pattern>/*</url-pattern>
40  </filter-mapping>
41 </web-app>

```

### C.3 jetty-env.xml

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <Configure id="springRestDemo" class="org.eclipse.jetty.webapp.WebAppContext">
3      <New id="DemoCP" class="org.eclipse.jetty.plus.jndi.Resource">
4          <Arg></Arg>
5          <Arg>jdbc/DemoCP</Arg>
6          <Arg>
7              <New class="com.zaxxer.hikari.HikariDataSource">
8                  <Set name="maximumPoolSize">20</Set>
9                  <Set name="dataSourceClassName">com.mysql.cj.jdbc.MysqlDataSource</Set>
10                 <Set name="username">DemoUser</Set>
11                 <Set name="password">Password123</Set>
12                 <Call name="addDataSourceProperty">
13                     <Arg>url</Arg>
14                     <Arg>jdbc:mysql://localhost:3306/SpringDemo?serverTimezone=UTC</Arg>
15                 </Call>
16             </New>
17         </Arg>
18     </New>
19   </Arg>
20 </Configure>

```