"Spring Security con Base de Datos"

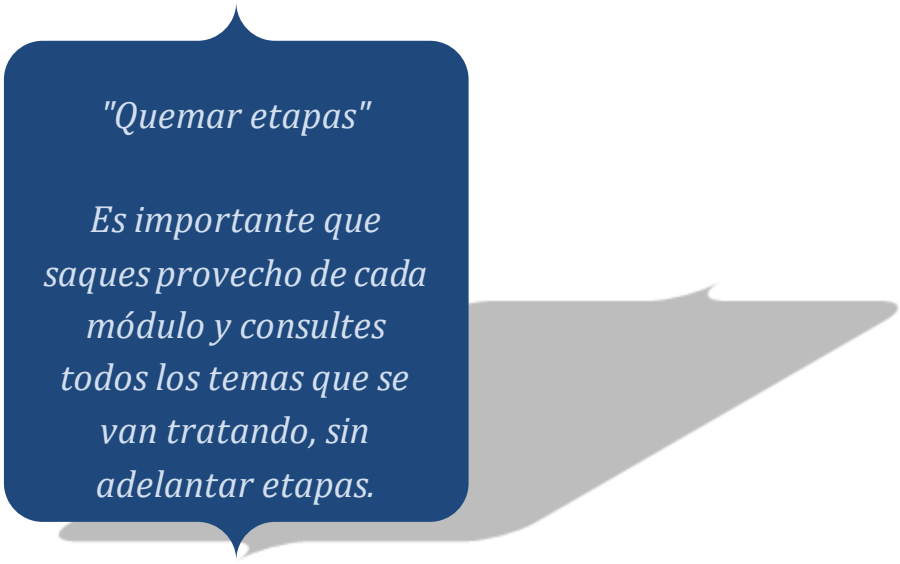
Módulo 8 / 2

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com

Objetivo

El objetivo de esta semana es entender cómo funciona el componente Spring Security y conocer una forma de implementar un sistema de autenticación con **Spring Farmework**.

En esta segunda parte veremos todo lo relacionado a Spring Security con base de datos (JDBC y con Hibernate).

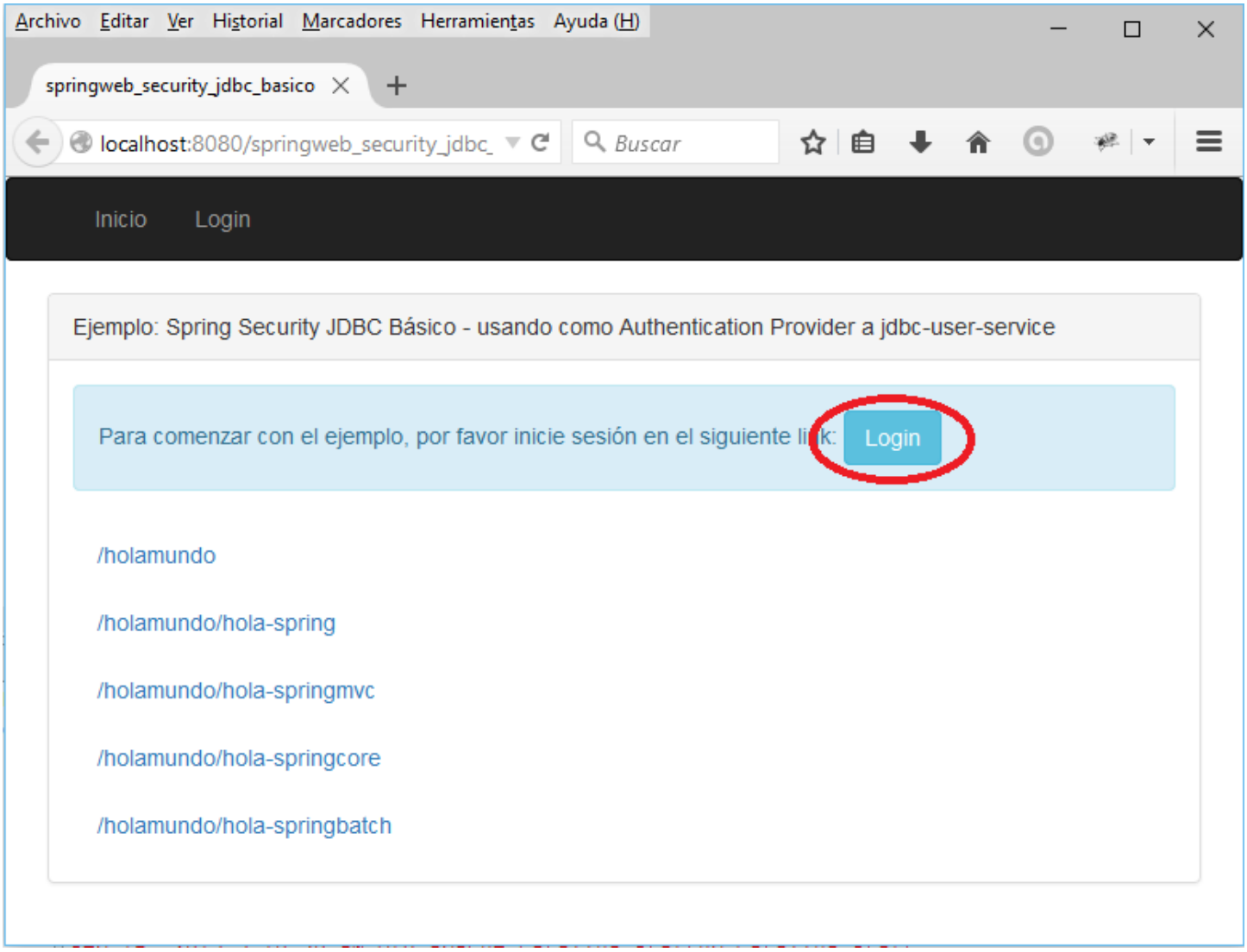


AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com

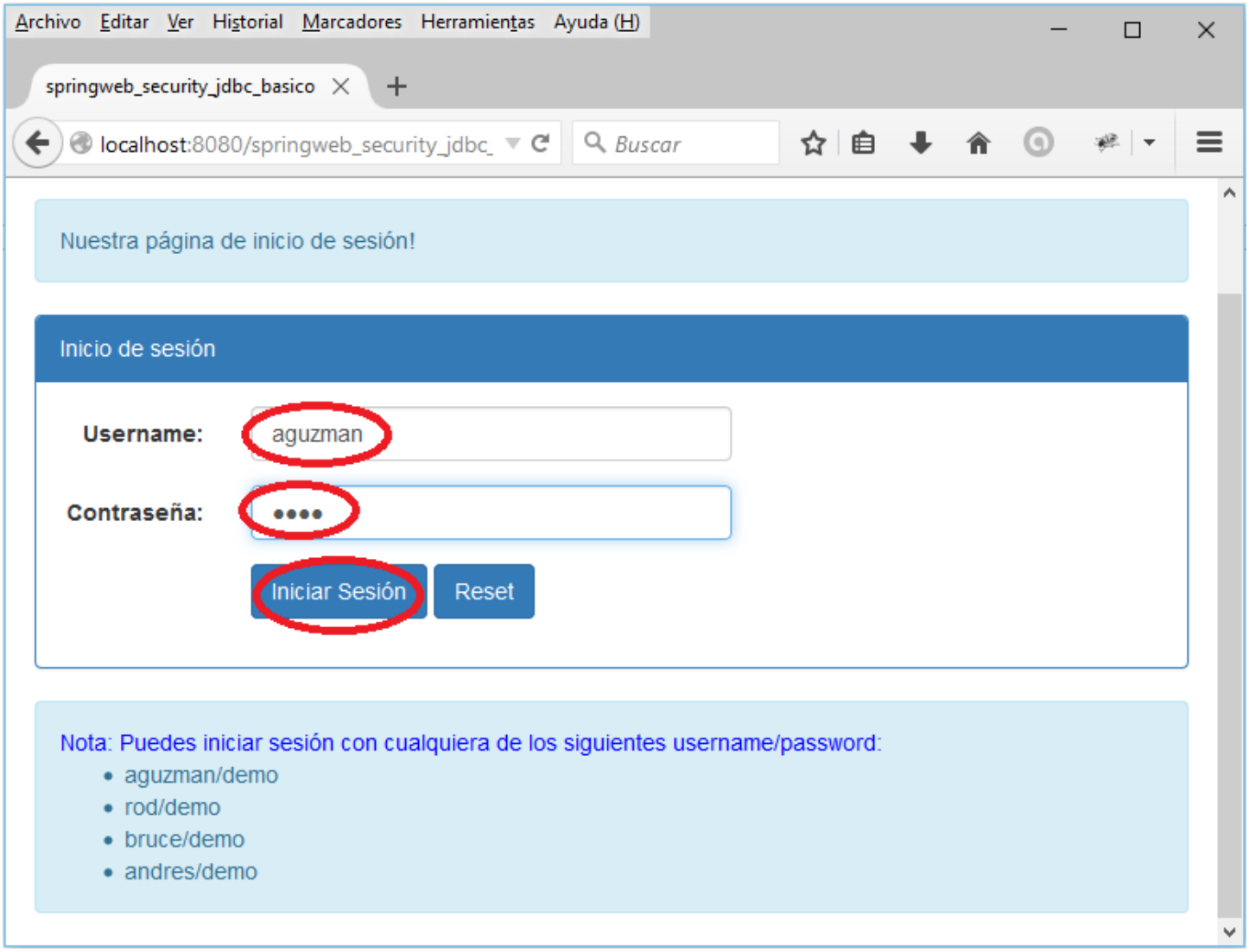
Ejercicio 1: Implementar seguridad simple con JDBC

Aprenderemos a dar seguridad a nuestra aplicación spring web usando JDBC, básicamente tenemos que configurar **jdbc-user-service** como proveedor de autenticación (**authentication provider**) y además tener el esquema de tablas user y authorities que usa spring por defecto.

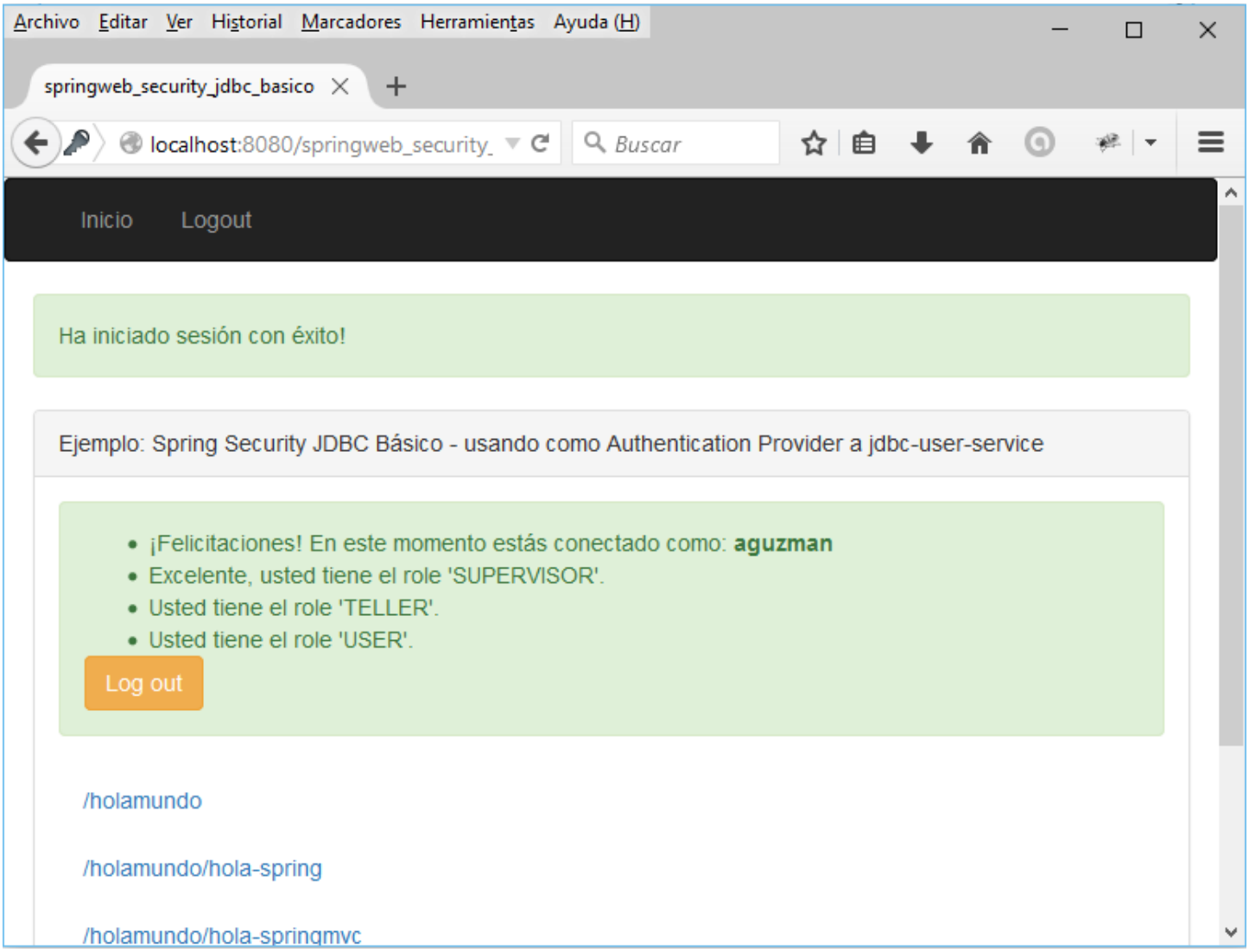
1. Clic derecho sobre el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
2. Clic derecho sobre el proyecto y **Maven->Update Project…**
3. Clic derecho sobre el proyecto **springweb_security_jdbc_basico -> Run As on Server**
4. Observe el resultado en el navegador:
5. Clic en Login para comenzar.



- Observe la página de login personalizada (credenciales de seguridad.)
- Iniciar sesión con username **aguzman** y password **demo**



- Notamos que el usuario **aguzman**, tiene acceso a todo, ya que es dueño de los roles SUPERVISOR y USER:



6. Observar el archivo pom.xml

```
ETC …
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <!-- Spring Security -->
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-core</artifactId>
        <version>${spring.security.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-web</artifactId>
        <version>${spring.security.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-config</artifactId>
        <version>${spring.security.version}</version>
    </dependency>
ETC …
```

7. Estudiar web XML **/springweb_security_basico/src/main/webapp/WEB-INF/web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-
app_3_1.xsd" version="3.1">
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/spring/root-context.xml,
            /WEB-INF/spring/applicationContext-security.xml
        </param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <servlet>
        <servlet-name>springweb_security_jdbc_basico_query</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>springweb_security_jdbc_basico_query</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <!-- Spring Security -->
    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
    </filter>

    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

8. Estudiar archivo de configuración de seguridad spring **/WEB-INF/spring/applicationContext-security.xml**
    - Usamos el proveedor **jdbc-user-service** para definir el query encargado de realizar la autenticación con base de datos, el query lo maneja internamente spring y debe tener cierta estructura/esquema de tablas predefinidas.
    - Además observamos que es necesario pasar el bean **dataSource** configurado en el XML **applicationContext-dataSource.xml (estudiado en el módulo de base de datos)**:

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

    <!-- Podemos usar multiples elementos <intercept-url> para definir los diferentes
        requerimientos de accesos para el conjunto de URLs, pero serán evaluadas
        en el orden de la lista y a la primera coincidencia será usada. -->
    <http auto-config="true">
        <access-denied-handler error-page="/mi_pagina_error_403" />
        <intercept-url pattern="/holamundo/hola*" access="hasRole('ROLE_SUPERVISOR')" />
        <intercept-url pattern="/holamundo*" access="hasRole('ROLE_USER')" />
        <form-login login-page="/mi_pagina_login"
            default-target-url="/?success=1"
            authentication-failure-url="/mi_pagina_login?error" />
        <logout logout-success-url="/mi_pagina_login?logout" />
    </http>

    <authentication-manager>
        <authentication-provider>
              <jdbc-user-service data-source-ref="dataSource" />
        </authentication-provider>
    </authentication-manager>

</beans:beans>
```

- Abrir y estudiar el script SQL initDB.txt:
  /src/main/webapp/WEB-INF/classes/db/hsqldb/initDB.txt
- Para llevar a cabo la autenticación de base de datos, tenemos que crear las tablas predefinidas que usa Spring Security para almacenar los usuarios y roles.
- Debe tener el esquema predefinido por defecto de Spring Security.
- A continuación los scripts SQL para crear las tablas users y authorities.

```
/* Create tables */
create table users(
    username varchar_ignorecase(50) not null primary key,
    password varchar_ignorecase(50) not null,
    enabled boolean not null);

create table authorities (
    username varchar_ignorecase(50) not null,
    authority varchar_ignorecase(50) not null,
    constraint fk_authorities_users foreign key(username)
    references users(username));
create unique index ix_auth_username on authorities
    (username,authority);
```

9. Abrir y estudiar el script SQL populateDB.txt.
   /src/main/webapp/WEB-INF/classes/db/hsqldb/populateDB.txt

```
    /* Populate tables */
    INSERT INTO users (username, password, enabled) VALUES('aguzman', 'demo', 1);
    INSERT INTO users (username, password, enabled) VALUES('rod', 'demo', 1);
    INSERT INTO users (username, password, enabled) VALUES('bruce', 'demo', 1);
    INSERT INTO users (username, password, enabled) VALUES('james', 'demo', 1);
    INSERT INTO users (username, password, enabled) VALUES('andres', 'demo', 1);

    INSERT INTO authorities (username, authority) VALUES('aguzman', 'ROLE_SUPERVISOR');
    INSERT INTO authorities (username, authority) VALUES('aguzman', 'ROLE_USER');
    INSERT INTO authorities (username, authority) VALUES('aguzman', 'ROLE_TELLER');
    INSERT INTO authorities (username, authority) VALUES('rod', 'ROLE_SUPERVISOR');
    INSERT INTO authorities (username, authority) VALUES('rod', 'ROLE_USER');
    INSERT INTO authorities (username, authority) VALUES('rod', 'ROLE_TELLER');
    INSERT INTO authorities (username, authority) VALUES('bruce', 'ROLE_USER');
    INSERT INTO authorities (username, authority) VALUES('bruce', 'ROLE_TELLER');
    INSERT INTO authorities (username, authority) VALUES('james', 'ROLE_USER');
    INSERT INTO authorities (username, authority) VALUES('andres', 'ROLE_USER');
```

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com

10. Abrir y estudiar la clase Controller LoginController:

/src/main/java/com.bolsadeideas.ejemplos.controllers/LoginController.java

- Un simple controlador para manejar de mejor forma los errores del login y retornar la vista con el formulario de login personalizado.

```
package com.bolsadeideas.ejemplos.controllers;

import java.security.Principal;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class LoginController {

    protected final Logger logger = LoggerFactory.getLogger(LoginController.class);

    @RequestMapping(value = "mi_pagina_login", method = RequestMethod.GET)
    public String login(Model model,
            @RequestParam(value = "error", required = false) String error,
            @RequestParam(value = "logout", required = false) String logout,
            Principal principal) {

        if(principal != null){
            return "redirect:/";
        }

        if (error != null) {
            model.addAttribute("error", "Error en el login:
                                Nombre de usuario o contraseña inválida, vuelva a intentarlo!");
            logger.info("Error en el login: Nombre de usuario o contraseña inválida!");
        }

        if (logout != null) {
            model.addAttribute("msg", "Ha cerrado la sesión con éxito!");
            logger.info("Ha cerrado la sesión con éxito!");
        }
        return "mi_pagina_login";
    }

}
```

11. Abrir y estudiar la vista **inicio.jsp**.
    src/main/webapp/WEB-INF/views/inicio.jsp
    - Página de inicio por defecto, muestra el uso de las etiquetas **JSP taglib de Spring Security <sec:authorize…>** para mostrar contenido privado de usuarios con inicio de sesión y de los roles **ROLE_SUPERVISOR** / **ROLE_USER**.

```
ETC...
<body>
    <jsp:include page="menu.jsp" />
    <div class="container">
        <c:if test="${not empty param.success}">
            <div class="alert alert-success">
                <p>Ha iniciado sesión con éxito!</p>
            </div>
        </c:if>
        <div class="panel panel-default">
            <div class="panel-heading">
                 Ejemplo: Spring Security JDBC Básico - usando como Authentication Provider
                 a jdbc-user-service
            </div>

            <div class="panel-body">
                <sec:authorize access="!isAuthenticated()">
                    <div class="alert alert-info">
                        Para comenzar con el ejemplo, por favor inicie sesión en el
                        siguiente link: <a class="btn btn-info" role="button"
                            href="<%=request.getContextPath()%>/mi_pagina_login">Login</a>
                    </div>
                </sec:authorize>

                <sec:authorize access="isAuthenticated()">

                    <div class="alert alert-success">
                        <ul>
                            <li>¡Felicitaciones! En este momento estás conectado como:
                                <strong>${pageContext.request.userPrincipal.name}</strong>
                            </li>
                            <sec:authorize access="hasRole('ROLE_SUPERVISOR')">
                                <li>Excelente, usted tiene el role 'SUPERVISOR'.</li>
                            </sec:authorize>

                            <sec:authorize access="hasRole('ROLE_TELLER')">
                                <li>Usted tiene el role 'TELLER'.</li>
                            </sec:authorize>

                            <sec:authorize access="hasRole('ROLE_USER')">
                                <li>Usted tiene el role 'USER'.</li>
                            </sec:authorize>
                        </ul>
```

```
                        <form id="logoutForm"
                            action="${pageContext.request.contextPath}/logout" method="post">
                            <input class="btn btn-warning" role="button" type="submit"
                                value="Log out" /> <input type="hidden"
                                name="${_csrf.parameterName}" value="${_csrf.token}" />
                        </form>
                    </div>
                </sec:authorize>
                <ul class="nav nav-pills nav-stacked">
                    <li><a href="holamundo">/holamundo</a></li>
                    <li><a href="holamundo/hola-spring">/holamundo/hola-spring</a></li>
                    <li><a href="holamundo/hola-springmvc">/holamundo/hola-springmvc</a></li>
                    <li><a href="holamundo/hola-springcore">
                              /holamundo/hola-springcore</a>
                    </li>
                    <li><a href="holamundo/hola-springbatch">
                              /holamundo/hola-springbatch</a>
                    </li>
                </ul>
            </div>
        </div>

    </div>
    <script type="text/javascript">
        function formSubmit() {
            $("#logoutForm").submit();
        }
    </script>
</body>
ETC...
```

12. Abrir y estudiar la vista **mi_pagina_login.jsp**.
    src/main/webapp/WEB-INF/views/mi_pagina_login.jsp
    - Vista formulario de login personalizado con los mensajes de errores.

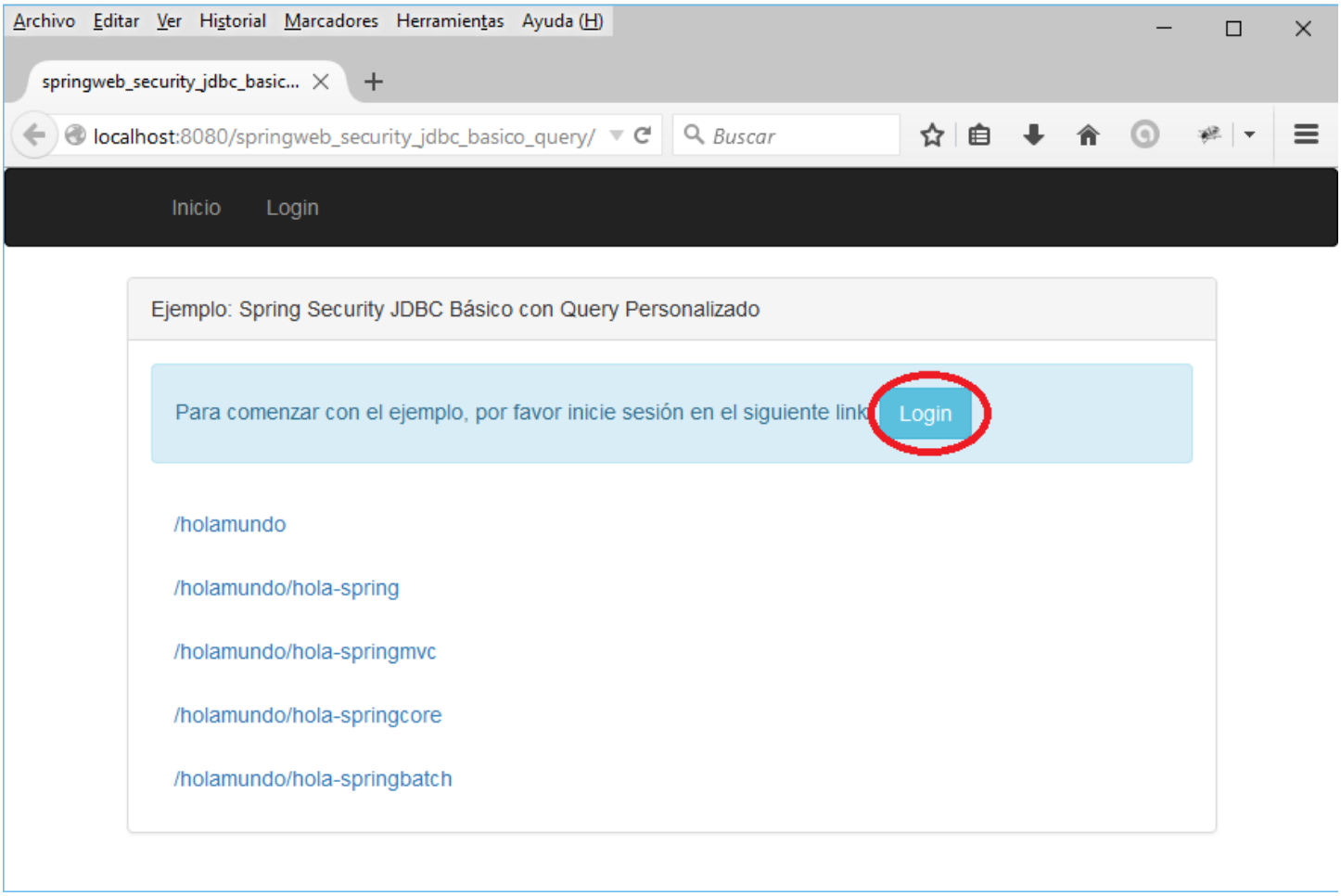
```
ETC...
<body>
    <jsp:include page="menu.jsp" />
    <div class="container">
        <sec:authorize access="!isAuthenticated()">

            <div class="alert alert-info">
                <p>Nuestra página de inicio de sesión!</p>
            </div>

            <c:if test="${not empty error}">
                <div class="alert alert-danger">
                    <p>${error}</p>
                </div>
            </c:if>

            <c:if test="${not empty msg}">
                <div class="alert alert-warning">
                    <p>${msg}</p>
                </div>
            </c:if>

            <div class="panel panel-primary">
                <div class="panel-heading">Inicio de sesión</div>
                <div class="panel-body">

                    <form name='f' action="${pageContext.request.contextPath}/login"
                        method='post' class="form-horizontal" role="form">

                        <input type="hidden" name="${_csrf.parameterName}"
                            value="${_csrf.token}" />

                        <div class="form-group">
                            <label for="username"
                                   class="col-sm-2 control-label">Username:</label>
                            <div class="col-sm-10">
                                <input style="width: 300px;" class="form-control" type='text'
                                    name='username' />
                            </div>
                        </div>

                        <div class="form-group">
                            <label for="password" class="col-sm-2 control-label">
                                Contraseña:
                            </label>
                            <div class="col-sm-10">
                                <input style="width: 300px;" class="form-control"
```

```
                                type='password' name='password' />
                        </div>
                    </div>

                    <div class="form-group">
                        <div class="col-sm-offset-2 col-sm-10">
                            <input type="submit" value="Iniciar Sesión"
                                   class="btn btn-primary" role="button" />
                            <input type="reset" value="Reset" class="btn btn-primary"
                                   role="button" />
                        </div>
                    </div>
                </form>
            </div>
        </div>
        <div class="alert alert-info">
            <p style="color: blue">Nota: Puedes iniciar sesión con
                cualquiera de los siguientes username/password:</p>
            <ul>
                <li>aguzman/demo</li>
                <li>rod/demo</li>
                <li>bruce/demo</li>
                <li>andres/demo</li>
            </ul>
        </div>
    </sec:authorize>
</div>
</body>
ETC...
```

Ejercicio 2: Implementar seguridad JDBC con consulta/query personalizado

Similar al ejemplo anterior, pero veremos cómo personalizar nuestra consulta de autenticación y esquema de tablas, algo más personalizado que el anterior.

1. Clic derecho en el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
2. Clic derecho en el proyecto y **Maven->Update Project…**
3. Clic derecho en el proyecto **springweb_security_jdbc_basico_query-> Run As on Server**
4. Observe el resultado en el navegador:
5. Clic en Login para comenzar.



6. Estudiar archivo de configuración de seguridad spring

   /WEB-INF/spring/applicationContext-security.xml

   - Usamos el proveedor **jdbc-user-service** y definimos un query personalizado con nuestro propio esquema de tablas:

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

    <!-- Podemos usar multiples elementos <intercept-url> para definir los diferentes
        requerimientos de accesos para el conjunto de URLs, pero serán evaluadas
        en el orden de la lista y a la primera coincidencia será usada. -->
    <http auto-config="true">
        <access-denied-handler error-page="/mi_pagina_error_403" />
        <intercept-url pattern="/holamundo/hola*" access="hasRole('ROLE_SUPERVISOR')" />
        <intercept-url pattern="/holamundo*" access="hasRole('ROLE_USER')" />
        <form-login login-page="/mi_pagina_login"
            default-target-url="/?success=1"
            authentication-failure-url="/mi_pagina_login?error" />
        <logout logout-success-url="/mi_pagina_login?logout" />
    </http>

    <authentication-manager>
        <authentication-provider>
            <jdbc-user-service data-source-ref="dataSource"
            users-by-username-query=
                  "select username, password, enabled from usuarios where username=?"
            authorities-by-username-query=
                 "select username, role from usuarios_roles where username=?" />
        </authentication-provider>
    </authentication-manager>

</beans:beans>
```

- Abrir y estudiar el script SQL initDB.txt:
  /src/main/webapp/WEB-INF/classes/db/hsqldb/initDB.txt
- Observamos los scripts SQL personalizados para el query.

```
/* Create tables */
create table usuarios (
    username varchar_ignorecase(50) not null primary key,
    password varchar_ignorecase(50) not null,
    enabled boolean not null);

create table usuarios_roles (
    username varchar_ignorecase(50) not null,
    role varchar_ignorecase(50) not null,
    constraint fk_roles_usuarios foreign key(username)
    references usuarios(username));
create unique index ix_auth_username on usuarios_roles
    (username,role);
```

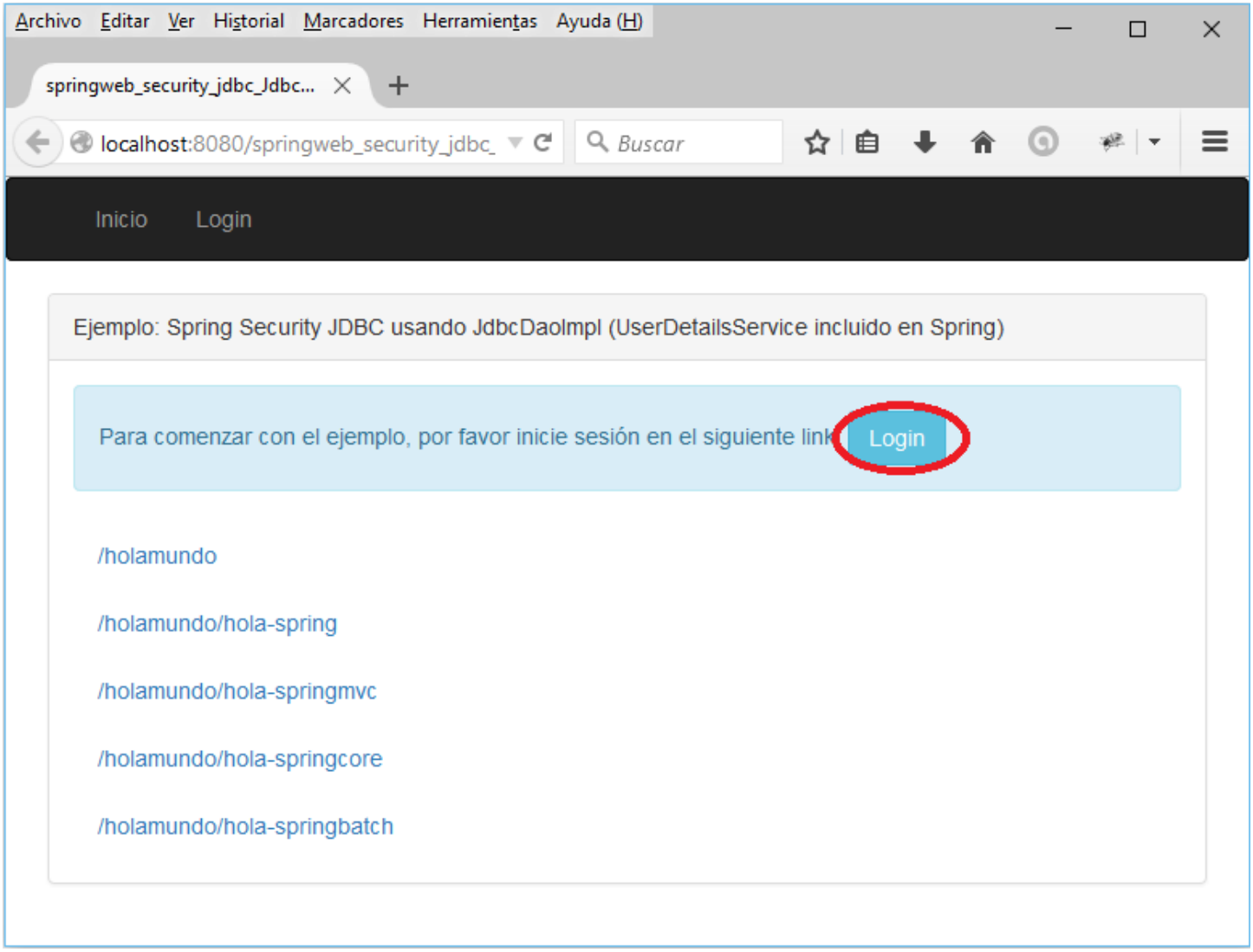
7. Abrir y estudiar el script SQL populateDB.txt.
   /src/main/webapp/WEB-INF/classes/db/hsqldb/populateDB.txt

```
    /* Populate tables */
    INSERT INTO usuarios (username, password, enabled) VALUES('aguzman', 'demo', 1);
    INSERT INTO usuarios (username, password, enabled) VALUES('rod', 'demo', 1);
    INSERT INTO usuarios (username, password, enabled) VALUES('bruce', 'demo', 1);
    INSERT INTO usuarios (username, password, enabled) VALUES('james', 'demo', 1);
    INSERT INTO usuarios (username, password, enabled) VALUES('andres', 'demo', 1);

    INSERT INTO usuarios_roles (username, role) VALUES('aguzman', 'ROLE_SUPERVISOR');
    INSERT INTO usuarios_roles (username, role) VALUES('aguzman', 'ROLE_USER');
    INSERT INTO usuarios_roles (username, role) VALUES('aguzman', 'ROLE_TELLER');
    INSERT INTO usuarios_roles (username, role) VALUES('rod', 'ROLE_SUPERVISOR');
    INSERT INTO usuarios_roles (username, role) VALUES('rod', 'ROLE_USER');
    INSERT INTO usuarios_roles (username, role) VALUES('rod', 'ROLE_TELLER');
    INSERT INTO usuarios_roles (username, role) VALUES('bruce', 'ROLE_USER');
    INSERT INTO usuarios_roles (username, role) VALUES('bruce', 'ROLE_TELLER');
    INSERT INTO usuarios_roles (username, role) VALUES('james', 'ROLE_USER');
    INSERT INTO usuarios_roles (username, role) VALUES('andres', 'ROLE_USER');
```

Ejercicio 3: Implementar seguridad JDBC usando JdbcDaoImpl

Aprenderemos a implementar seguridad con JDBC usando la clase **JdbcDaoImpl**, proveída por Spring Security.

1. Clic derecho en el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
2. Clic derecho en el proyecto y **Maven->Update Project…**
3. Clic derecho en el proyecto **springweb_security_jdbc_JdbcDaoImpl-> Run As on Server**
4. Observe el resultado en el navegador:
5. Clic en Login para comenzar.



6. Estudiar archivo de configuración de seguridad spring
   /WEB-INF/spring/applicationContext-security.xml
   - En el elemento **authentication-provider** usamos el atributo **user-service-ref** para hacer referencia al bean **JdbcDaoImpl** de Spring, encargado de realizar la autenticación:

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

    <!-- Podemos usar multiples elementos <intercept-url> para definir los diferentes
        requerimientos de accesos para el conjunto de URLs, pero serán evaluadas
        en el orden de la lista y a la primera coincidencia será usada. -->
    <http auto-config="true">
        <access-denied-handler error-page="/mi_pagina_error_403" />
        <intercept-url pattern="/holamundo/hola*" access="hasRole('ROLE_SUPERVISOR')" />
        <intercept-url pattern="/holamundo*" access="hasRole('ROLE_USER')" />
        <form-login login-page="/mi_pagina_login"
            default-target-url="/?success=1"
            authentication-failure-url="/mi_pagina_login?error" />
        <logout logout-success-url="/mi_pagina_login?logout" />
    </http>

    <authentication-manager>
        <authentication-provider user-service-ref='userDetailsService' />
    </authentication-manager>

    <beans:bean id="userDetailsService"
        class="org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl">
        <beans:property name="dataSource" ref="dataSource" />
    </beans:bean>

</beans:beans>
```

7. Abrir y estudiar el script SQL initDB.txt:
   /src/main/webapp/WEB-INF/classes/db/hsqldb/initDB.txt
   - Observamos los scripts SQL correspondientes.

```
/* Create tables */
create table users(
    username varchar_ignorecase(50) not null primary key,
    password varchar_ignorecase(50) not null,
    enabled boolean not null);

create table authorities (
    username varchar_ignorecase(50) not null,
    authority varchar_ignorecase(50) not null,
    constraint fk_authorities_users foreign key(username)
    references users(username));
create unique index ix_auth_username on authorities
    (username,authority);
```

8. Abrir y estudiar el script SQL populateDB.txt.
   /src/main/webapp/WEB-INF/classes/db/hsqldb/populateDB.txt

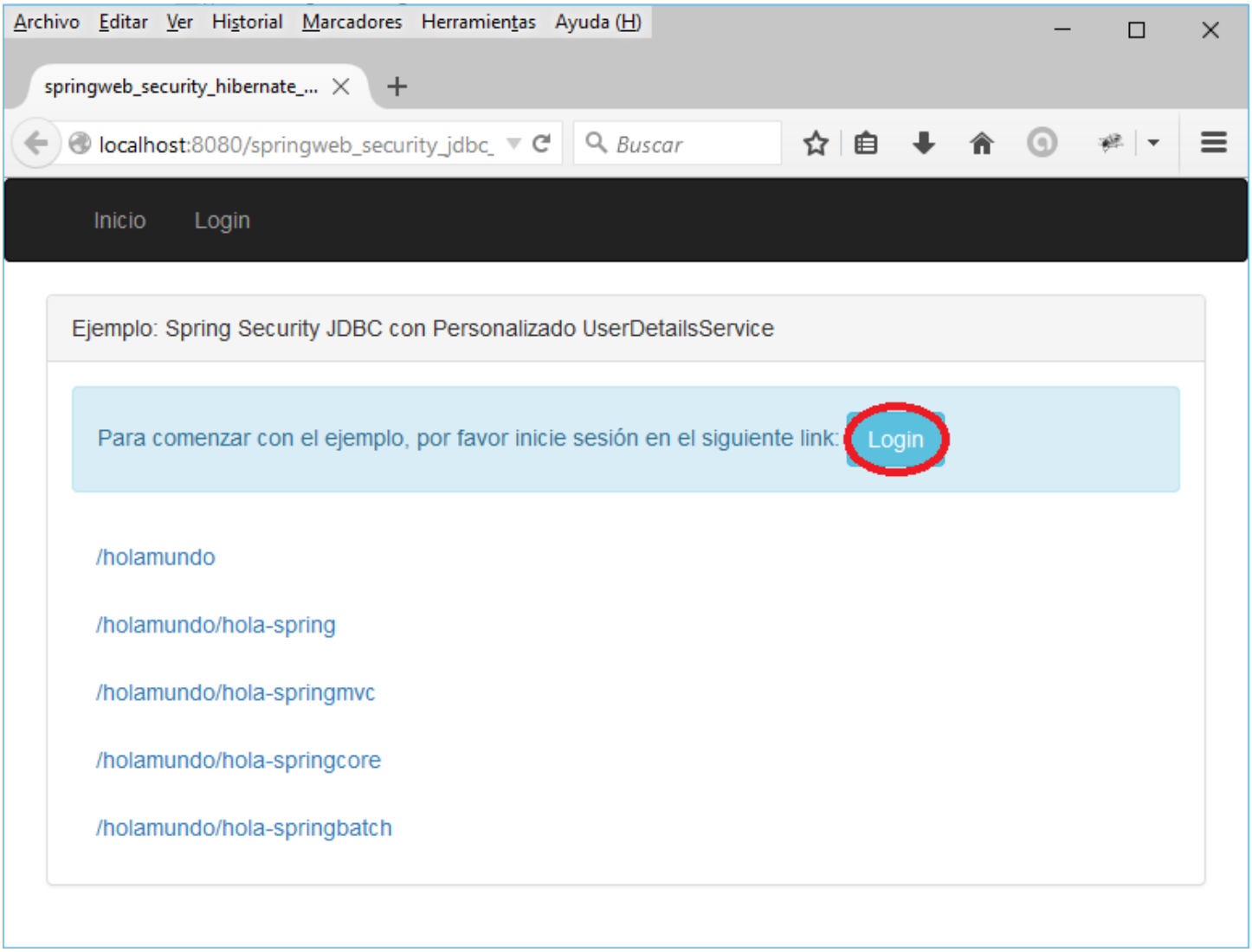
```
    /* Populate tables */
    INSERT INTO users (username, password, enabled) VALUES('aguzman', 'demo', 1);
    INSERT INTO users (username, password, enabled) VALUES('rod', 'demo', 1);
    INSERT INTO users (username, password, enabled) VALUES('bruce', 'demo', 1);
    INSERT INTO users (username, password, enabled) VALUES('james', 'demo', 1);
    INSERT INTO users (username, password, enabled) VALUES('andres', 'demo', 1);

    INSERT INTO authorities (username, authority) VALUES('aguzman', 'ROLE_SUPERVISOR');
    INSERT INTO authorities (username, authority) VALUES('aguzman', 'ROLE_USER');
    INSERT INTO authorities (username, authority) VALUES('aguzman', 'ROLE_TELLER');
    INSERT INTO authorities (username, authority) VALUES('rod', 'ROLE_SUPERVISOR');
    INSERT INTO authorities (username, authority) VALUES('rod', 'ROLE_USER');
    INSERT INTO authorities (username, authority) VALUES('rod', 'ROLE_TELLER');
    INSERT INTO authorities (username, authority) VALUES('bruce', 'ROLE_USER');
    INSERT INTO authorities (username, authority) VALUES('bruce', 'ROLE_TELLER');
    INSERT INTO authorities (username, authority) VALUES('james', 'ROLE_USER');
    INSERT INTO authorities (username, authority) VALUES('andres', 'ROLE_USER');
```

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com

Ejercicio 4: Implementar seguridad JDBC usando nuestra personalizada clase DAO

Aprenderemos a implementar seguridad con JDBC usando nuestra propia y personalizada clase Dao usando JdbcTemplate para personalizar las consultas y esquema SQL de tablas.

1. Clic derecho en el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
2. Clic derecho en el proyecto y **Maven->Update Project…**
3. Clic derecho en **springweb_security_jdbc_userDetailsService-> Run As on Server**
4. Observe el resultado en el navegador:
5. Clic en Login para comenzar.



6. Estudiar archivo de configuración de seguridad spring
   /WEB-INF/spring/applicationContext-security.xml
   - En el elemento **authentication-provider** usamos el atributo **user-service-ref** para hacer referencia al bean personalizado **JdbcUserDetailsService**, encargado de realizar la autenticación:

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

    <!-- Podemos usar multiples elementos <intercept-url> para definir los diferentes
        requerimientos de accesos para el conjunto de URLs, pero serán evaluadas
        en el orden de la lista y a la primera coincidencia será usada. -->
    <http auto-config="true">
        <access-denied-handler error-page="/mi_pagina_error_403" />
        <intercept-url pattern="/holamundo/hola*" access="hasRole('ROLE_SUPERVISOR')" />
        <intercept-url pattern="/holamundo*" access="hasRole('ROLE_USER')" />
        <form-login login-page="/mi_pagina_login"
            default-target-url="/?success=1"
            authentication-failure-url="/mi_pagina_login?error" />
        <logout logout-success-url="/mi_pagina_login?logout" />
    </http>

    <authentication-manager>
        <authentication-provider user-service-ref='jdbcUserDetailsService' />
    </authentication-manager>

</beans:beans>
```

7. Abrir y estudiar clase **JdbcUserDetailsService**: **com.bolsadeideas.ejemplos.models.services.JdbcUserDetailsService.java**
    - Debemos implementar la interfaz **UserDetailsService** y su método **loadUserByUsername(String username)**.

```
package com.bolsadeideas.ejemplos.models.services;

import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.bolsadeideas.ejemplos.models.dao.IUserDao;
import com.bolsadeideas.ejemplos.models.dao.UsuarioDaoJdbcTemplate;
import com.bolsadeideas.ejemplos.models.entity.Role;
import com.bolsadeideas.ejemplos.models.entity.Usuario;

@Service("jdbcUserDetailsService")
public class JdbcUserDetailsService implements UserDetailsService {

    @Autowired
    private IUserDao usuarioDao;

    @Override
    @Transactional(readOnly = true)
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        Usuario usuario = usuarioDao.findByUsername(username);
        if (usuario == null) {
            throw new UsernameNotFoundException("Username " + username + " no encontrado");
        }

        List<GrantedAuthority> dbAuths = new ArrayList<GrantedAuthority>();
        for (Role role : usuario.getRoles()) {
            dbAuths.add(new SimpleGrantedAuthority(role.getAuthority()));
        }

        if (dbAuths.size() == 0) {
            throw new UsernameNotFoundException("Usuario "+username+" no tiene roles asignados");
        }
        return new User(username, usuario.getPassword(), usuario.getEnabled(),
                        true, true, true, dbAuths);
    }
}
```

- Custom UserDetailsService, carga los usuarios desde IUserDao, luego construye los roles del usuario y retorna UserDetails

8. Abrir y estudiar la clase Dao **UsuarioDaoJdbcTemplate**.
   com.bolsadeideas.ejemplos.models.dao.UsuarioDaoJdbcTemplate

```
package com.bolsadeideas.ejemplos.models.dao;

/* ... ETC IMPORTS ... */

@Repository
public class UsuarioDaoJdbcTemplate implements IUserDao {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Override
    public Usuario findByUsername(String username) {
        Usuario usuario = jdbcTemplate.queryForObject("SELECT id, username, password, enabled
                                                       FROM users WHERE username = ?",
                  BeanPropertyRowMapper.newInstance(Usuario.class), username);
        List<Role> roles = jdbcTemplate.query("SELECT id, authority FROM authorities
                                              WHERE user_id = ?",
                  BeanPropertyRowMapper.newInstance(Role.class), usuario.getId());
        usuario.setRoles(roles);
        return usuario;
    }

    @Override
    public List<Usuario> findAll() {
        return jdbcTemplate.query("SELECT * FROM users",
                                  BeanPropertyRowMapper.newInstance(Usuario.class));
    }

    @Override
    public Usuario findById(Integer id) {
        return jdbcTemplate.queryForObject("SELECT * FROM users WHERE id = ?",
                BeanPropertyRowMapper.newInstance(Usuario.class), id);
    }

/* ... ETC ... */

}
```

- Clase DAO para cargar los usuarios y roles desde la base de datos, vía JdbcTemplate

9. Las vistas JSP, web.xml, xml de spring y controlador se omiten, son absolutamente códigos estándar vistos anteriormente, estudiar estos archivos en los ejemplos que están para la descarga del módulo.

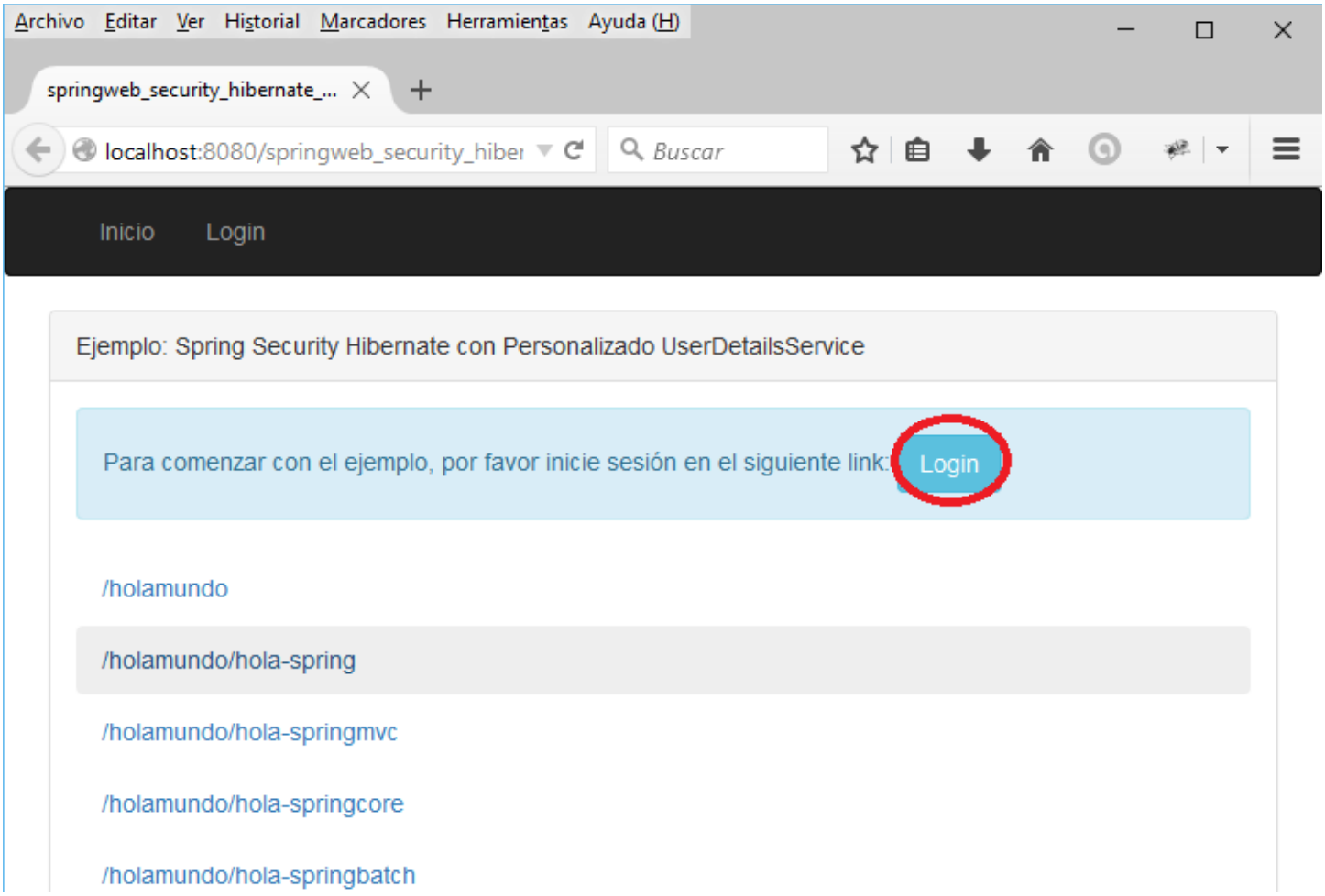
AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com

Ejercicio 5: Implementar seguridad Hibernate usando nuestro personalizado DAO

En el siguiente ejemplo vamos a implementar **seguridad con Hibernate** usando nuestra propia y **personalizada clase Dao** implementada con **HibernateTemplate**.

- ✓ Crear un session factory con hibernate4.LocalSessionFactoryBean
- ✓ Inyectar HibernateTemplate en UsuarioDaoHibernateTemplate
- ✓ Para la integración con Spring Security, crear una clase que implementa la interfaz UserDetailsService, y carga el usuario con UsuarioDaoHibernateTemplate
- ✓ El Transaction manager debe ser declarado, de lo contrario las transacciones Hibernate no van a funcionar en Spring

10. Clic derecho en el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
11. Clic derecho en el proyecto y **Maven->Update Project…**
12. Clic derecho en **springweb_security_hibernate_userDetailsService-> Run As on Server**
13. Observe el resultado en el navegador:
14. Clic en Login para comenzar.



15. Estudiar archivo de configuración de seguridad spring

/WEB-INF/spring/applicationContext-security.xml

- En el elemento **authentication-provider** usamos el atributo **user-service-ref** para hacer referencia al bean personalizado **HibernateUserDetailsService**, encargado de realizar la autenticación con Hibernate:

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

    <!-- Podemos usar multiples elementos <intercept-url> para definir los diferentes
        requerimientos de accesos para el conjunto de URLs, pero serán evaluadas
        en el orden de la lista y a la primera coincidencia será usada. -->
    <http auto-config="true">
        <access-denied-handler error-page="/mi_pagina_error_403" />
        <intercept-url pattern="/holamundo/hola*" access="hasRole('ROLE_SUPERVISOR')" />
        <intercept-url pattern="/holamundo*" access="hasRole('ROLE_USER')" />
        <form-login login-page="/mi_pagina_login"
            default-target-url="/?success=1"
            authentication-failure-url="/mi_pagina_login?error" />
        <logout logout-success-url="/mi_pagina_login?logout" />
    </http>

    <authentication-manager>
        <authentication-provider user-service-ref='hibernateUserDetailsService' />
    </authentication-manager>

</beans:beans>
```

16. Abrir y estudiar clase **HibernateUserDetailsService**:
    com.bolsadeideas.ejemplos.models.services.HibernateUserDetailsService.java
    - Debemos implementar la interfaz **UserDetailsService** y su método **loadUserByUsername(String username)**.
    - Custom UserDetailsService, carga los usuarios desde IUserDao, luego construye los roles del usuario y retorna UserDetails

```
package com.bolsadeideas.ejemplos.models.services;

import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.bolsadeideas.ejemplos.models.dao.IUserDao;
import com.bolsadeideas.ejemplos.models.dao.UsuarioDaoHibernateTemplate;
import com.bolsadeideas.ejemplos.models.entity.Role;
import com.bolsadeideas.ejemplos.models.entity.Usuario;

@Service("hibernateUserDetailsService")
public class HibernateUserDetailsService implements UserDetailsService {

    @Autowired private IUserDao usuarioDao;

    @Transactional(readOnly = true)
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        Usuario usuario = usuarioDao.findByUsername(username);
        if (usuario == null) {
            throw new UsernameNotFoundException("Username " + username + " no encontrado");
        }

        List<GrantedAuthority> dbAuths = new ArrayList<GrantedAuthority>();
        for (Role role : usuario.getRoles()) {
            dbAuths.add(new SimpleGrantedAuthority(role.getAuthority()));
        }

        if (dbAuths.size() == 0) {
            throw new UsernameNotFoundException("Usuario"+username+" no tiene roles asignados");
        }
        return new User(username, usuario.getPassword(), usuario.getEnabled(),
                        true, true, true, dbAuths);
    }
}
```

17. Abrir y estudiar la clase Dao **UsuarioDaoHibernateTemplate**.
    com.bolsadeideas.ejemplos.models.dao.UsuarioDaoHibernateTemplate
    - Clase DAO para cargar los usuarios y roles desde la base de datos, vía Hibernate

```
package com.bolsadeideas.ejemplos.models.dao;

/* ... ETC IMPORTS ... */

@Repository
public class UsuarioDaoHibernateTemplate implements IUserDao {

    @Autowired
    private HibernateTemplate hibernateTemplate;

    @Override
    public Usuario findByUsername(String username) {
        List<Usuario> results = (List<Usuario>) hibernateTemplate.findByNamedParam("from
                                Usuario u where u.username = :username", "username", username);
        if (results.size() == 1) {
            return results.get(0);
        }
        return null;
    }

    @Override
    public List<Usuario> findAll() {
        return (List<Usuario>) hibernateTemplate.find("from Usuario");
    }

    @Override
    public Usuario findById(Integer id) {
        return (Usuario) hibernateTemplate.get(Usuario.class, id);
    }

    @Override
    public void save(Usuario user) {
        hibernateTemplate.saveOrUpdate(user);
    }

    @Override
    public void delete(Usuario user) {
        hibernateTemplate.delete(user);
    }
}
```

18. Abrir y estudiar la clase Entity **Role**.
    com.bolsadeideas.ejemplos.models.entity.Role

```
package com.bolsadeideas.ejemplos.models.entity;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="authorities")
public class Role implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    private String authority;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getAuthority() {
        return authority;
    }

    public void setAuthority(String authority) {
        this.authority = authority;
    }

    private static final long serialVersionUID = 1L;
}
```

19. Abrir y estudiar la clase Entity **Usuario**.
    com.bolsadeideas.ejemplos.models.entity.Usuario

```
package com.bolsadeideas.ejemplos.models.entity;
import java.io.Serializable;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name="users")
public class Usuario implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;

    private String username;

    private String password;

    private Boolean enabled;

    @OneToMany
    @JoinColumn(name="user_id")
    private List<Role> roles;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
```

```
        }

        public void setPassword(String password) {
            this.password = password;
        }

        public Boolean getEnabled() {
            return enabled;
        }

        public void setEnabled(Boolean enabled) {
            this.enabled = enabled;
        }

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

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

        public static long getSerialversionuid() {
            return serialVersionUID;
        }

        private static final long serialVersionUID = 1L;
    }
```

20. Abrir y estudiar el script SQL initDB.txt:
    /src/main/webapp/WEB-INF/classes/db/hsqldb/initDB.txt
    - Observamos los scripts SQL correspondientes.

```
/* Create tables */
create table users(
    id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    username varchar_ignorecase(50) not null,
    password varchar_ignorecase(50) not null,
    enabled boolean not null);

create table authorities (
    id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    user_id INTEGER NOT NULL,
    authority varchar_ignorecase(50) not null,
    constraint fk_authorities_users foreign key(user_id)
    references users(id));

create unique index ix_auth_user_id on authorities
    (user_id, authority);
```

21. Abrir y estudiar el script SQL populateDB.txt.
    /src/main/webapp/WEB-INF/classes/db/hsqldb/populateDB.txt

```
    /* Populate tables */
    INSERT INTO users (id, username, password, enabled) VALUES(1, 'aguzman', 'demo', 1);
    INSERT INTO users (id, username, password, enabled) VALUES(2, 'rod', 'demo', 1);
    INSERT INTO users (id, username, password, enabled) VALUES(3, 'bruce', 'demo', 1);
    INSERT INTO users (id, username, password, enabled) VALUES(4, 'james', 'demo', 1);
    INSERT INTO users (id, username, password, enabled) VALUES(5, 'andres', 'demo', 1);

    INSERT INTO authorities (id, user_id, authority) VALUES(1, 1, 'ROLE_SUPERVISOR');
    INSERT INTO authorities (id, user_id, authority) VALUES(2, 1, 'ROLE_USER');
    INSERT INTO authorities (id, user_id, authority) VALUES(3, 1, 'ROLE_TELLER');
    INSERT INTO authorities (id, user_id, authority) VALUES(4, 2, 'ROLE_SUPERVISOR');
    INSERT INTO authorities (id, user_id, authority) VALUES(5, 2, 'ROLE_USER');
    INSERT INTO authorities (id, user_id, authority) VALUES(6, 2, 'ROLE_TELLER');
    INSERT INTO authorities (id, user_id, authority) VALUES(7, 3, 'ROLE_USER');
    INSERT INTO authorities (id, user_id, authority) VALUES(8, 3, 'ROLE_TELLER');
    INSERT INTO authorities (id, user_id, authority) VALUES(9, 4, 'ROLE_USER');
    INSERT INTO authorities (id, user_id, authority) VALUES(10, 5, 'ROLE_USER');
```

22. Las vistas JSP, web.xml, xml de spring y controlador se omiten, son absolutamente códigos estándar vistos anteriormente, estudiar estos archivos en los ejemplos que están para la descarga del módulo.

Resumen

En el documento se explica detalladamente como incorporar un sistema de seguridad y autenticación de usuarios en nuestros proyectos con Spring MVC usando base de datos implementado con Spring Security, JDBC e Hibernate. Nos proporciona un sistema de seguridad bastante simple pero potente y con un completo control de errores y configuración de los mensajes de manera sencilla.

¡Dudas, a los foros! ;-)

FIN.

Envía tus consultas a los foros!

Aquí es cuando debes sacarte todas las dudas haciendo consultas en los foros correspondientes

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com