Create a Java Web Application and integrate Spring MVC + Spring Security with XML configuration

1. Creating Dynamic Web Project with Maven

First, create a project in Eclipse. You can create a dynamic web project first (**File > New > Dynamic WebProject**), the convert the project to Maven.

In this project, the following technologies are used:

- Java 8.
- Tomcat 8 with Servlet 3.1.
- Spring framework 5.1.4.RELEASE.
- Spring Security 5.1.4.RELEASE.
- JSTL 1.2

Let’s add code to the project now.

2. Adding Maven Dependencies

2.1 In the Maven project file (**pom.xml**), declare the following properties:

```
<properties>
          <spring.version>5.1.4.RELEASE</spring.version>
          <spring.security.version>5.1.4.RELEASE</
spring.security.version>
          <jstl.version>1.2</jstl.version>
     </properties>
```

These are the versions of Spring framework, Spring security and JSTL.

2.2 Next, declare the dependencies for Spring and Spring Web MVC:

```xml
<!-- Spring framework -->
<dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-core</artifactId>
     <version>${spring.version}</version>
</dependency>

<dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-context</artifactId>
     <version>${spring.version}</version>
</dependency>

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

2.3 Dependencies for Spring Security Web and Spring Security Config:

```xml
<!-- Spring Security -->
<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>
```

2.4 And dependency for JSTL as we'll write JSTL expressions in JSP pages:

```
<!-- JSTL -->
<dependency>
     <groupId>jstl</groupId>
     <artifactId>jstl</artifactId>
     <version>${jstl.version}</version>
</dependency>
```

3. Coding Index Page

Create a directory called **views** under the **/WEB-INF** directory, then create an **index.jsp** file with the following HTML code:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
     "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Security Basic Demo (XML)</title>
</head>
<body>
     <div align="center">
          <h1>Spring Security Basic Demo (XML)</h1>
          <a href="admin">Go to Administrator Page</a>
     </div>
</body>
</html>
```

As you can see, this is very simple page with a heading "*Spring Security Basic Demo (XML)*" and a hyperlink to the administrator page.

4. Coding Admin Page

Next, create an **admin.jsp** file under the **/WEB-INF/views** directory with the following code:

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ page language="java" session="true" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
     "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">
<title>Spring Security Basic Demo (XML)</title>
</head>
<body>
     <div align="center">
          <h1>${title}</h1>
          <h2>${message}</h2>
          <c:if test="${pageContext.request.userPrincipal.name !
= null}">
               <h2>Welcome : $
{pageContext.request.userPrincipal.name} |
                    <a href="<c:url value="/logout" />" >
Logout</a></h2>
     </c:if>
     </div>
</body>
</html>
```

This is the administrator page which requires authentication and authorization to access. We use JSTL expressions to display the title and message in the model. If the user is logged in, display his username along with a logout link.

5. Coding Spring MVC Controller

Next, we write code for a Spring controller in order to handle requests coming to the application. Create a Java class named **AdminController** under the package **com.demo.controller** with the following code:

```
package com.demo.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class AdminController {

	@RequestMapping(value="/", method = RequestMethod.GET)
	public ModelAndView visitHome() {
		return new ModelAndView("index");
	}

	@RequestMapping(value="/admin", method = RequestMethod.GET)
	public ModelAndView visitAdmin() {
		ModelAndView model = new ModelAndView("admin");
		model.addObject("title", "Admministrator Control 
Panel");
		model.addObject("message", "This page demonstrates how 
to use Spring security.");

		return model;
	}
}
```

As you can see, this controller is designed to handle 2 requests:

- **"/"**: the request to the application's context root, or home page.
- **"/admin"**: the request to the administrator page, which will be secured by Spring security.

The annotations **@Controller** and **@RequestMapping** are used to declare this is a controller which has two HTTP GET handle methods. These annotations will be scanned by Spring as we will configure in the Spring's application context file.

6. Configuring Web Deployment Descriptor (web.xml)

In the web deployment descriptor file (**web.xml** under **/WEB-INF** folder), we configure how Spring MVC and Spring Security are loaded when during the application startup time.

Here's content of the **web.xml** file:

```
<servlet>
          <servlet-name>SpringController</servlet-name>
          <servlet-
class>org.springframework.web.servlet.DispatcherServlet</
servlet-class>
          <init-param>
               <param-name>contextConfigLocation</param-name>
               <param-value>/WEB-INF/spring-mvc.xml</param-
value>
          </init-param>
          <load-on-startup>1</load-on-startup>
     </servlet>

     <servlet-mapping>
          <servlet-name>SpringController</servlet-name>
          <url-pattern>/</url-pattern>
     </servlet-mapping>

     <listener>
          <listener-
class>org.springframework.web.context.ContextLoaderListener</
listener-class>
     </listener>

     <context-param>
          <param-name>contextConfigLocation</param-name>
          <param-value>
               /WEB-INF/spring-security.xml
          </param-value>
     </context-param>

     <!-- Spring Security  Filter -->
     <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>
```

As standard, we declare a Spring dispatcher servlet that handles all URLs coming to the application, and a Spring Web Context Loader Listener to loads Spring security configuration (in a Spring security configuration file named **spring-security.xml** file under **/WEB-INF** folder).

The interesting point here is the configuration of the **Spring Security Filter:**

The responsibility of the Spring Security Filter is to intercept some URL patterns in order to apply authentication and authorization as configured in the Spring security configuration file.

7. Configuring Spring MVC Application Context

Next, we configure Spring MVC to scan Java classes for controllers and resolve view names. Create a **spring-mvc.xml**file under the **/WEB-INF** folder with the following code:

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

     <mvc:annotation-driven />
     <context:component-scan base-
package="com.demo.controller" />

     <bean
class="org.springframework.web.servlet.view.InternalResourceView
Resolver">
          <property name="prefix" value="/WEB-INF/views/" />
          <property name="suffix" value=".jsp" />
     </bean>

</beans>
```

As you are familiar with Spring MVC, the **<mvc:annotation-driven />** element tells Spring to analyze annotations for loading configurations and controllers.

The **<context:component-scan />** element specifies which Java package to search for Spring components.

And an **InternalResourceViewResolver** bean is declared to tell Spring how to resolve logical view names to physical view pages.

8. Configuring Spring Security

The last step is to declare authentication (who can login) and authorization (who can access which page). Create a **spring-security.xml** file under **/WEB-INF** folder with the following code:

```
<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-4.2.xsd
     http://www.springframework.org/schema/security
     http://www.springframework.org/schema/security/spring-
security-4.2.xsd">

     <http auto-config="true">
          <intercept-url pattern="/admin**"
access="hasRole('ROLE_ADMIN')" />
          <csrf disabled="true" />
     </http>

     <authentication-manager>
          <authentication-provider>
               <user-service>
                    <user name="admin" password="{noop}nimda"
authorities="ROLE_ADMIN" />
               </user-service>
          </authentication-provider>
     </authentication-manager>

</beans:beans>
```

Here, there are two elements are used for authentication and authorization:

- The **<authentication-manager>** element declares a user with username, password and role (**ROLE_ADMIN** per this configuration). This user can be authenticated to access the application.
- In the **<http>** element, we declare which URL pattern will be intercepted by Spring security filter, using the **<intercept-url>** element. As per this configuration, all the URL patterns **/admin**** are secured, and only the users having role **ROLE_ADMIN** can be authorized to access these URLs.

Note that the **<csrf disabled="true" />** element tells the Spring security filter to intercept the */logout* link as HTTP GET request.