Spring MVC Tutorial – Contact List Application

Step 20: Incorporating Spring Security

SOFTWARE GUILD

The Learning House

427 S 4th Street #300

Louisville KY 40202

Step 20: Incorporating Spring Security

Overview

In this step, we will add Spring Security to our application. Spring Security is a comprehensive security solution that provides authentication and authorization at both the web request and method invocation levels. Like other Spring components, Spring Security uses both the dependency injection and aspect-oriented programming features of the core Spring platform.

In this step, we will add user authentication (via a login form) and authorization (via user roles) and will demonstrate these features by limiting the ability to add, update, and delete Contacts to users with particular roles. We will also show how you can use Spring Security to personalize the application for logged-in users.

Spring Security Approach

We will demonstrate the ability for Spring to limit access to web request endpoints in our application. Most of the Spring Security configuration will reside in a separate file (spring-security.xml), but we will have to add some configuration to Tomcat's web.xml file. Spring Security uses servlet filters to intercept incoming requests, which is why the Tomcat configuration changes are necessary (we must register the filter with Tomcat). The filter looks at each incoming request and decides whether or not to grant access based on the security configuration and the information of the currently logged-in user.

User and role information is stored in two new tables in our database. The tables for this example contain only the minimal information necessary for authentication and authorization, but additional user information can be stored in these tables if needed for more complex applications.

Preliminaries - Adding a Landing (Home) Page

To better show the features and capabilities of Spring Security, we need to make a couple changes to our application. In this section, we'll add a welcome/landing page that anyone can see (whether logged in or not). We'll put the rest of the pages (both Ajax and non-Ajax) behind Spring Security, which will require users to log in if they want to gain access.

Remapping the Ajax Main Page

Right now, '/' and '/home' both map to the main page of the Ajax-powered pages of our application. Our first step in adding a new landing page is to change the mapping for this endpoint. We will do this by adding the following method (and associated annotations) to the HomeController. After this method is added, we'll rename home.jsp to mainAjaxPage.jsp.

Add the following method to your HomeController:

```
// This method will be invoked by Spring MVC when it sees a request for
// ContactListMVC/mainAjaxPage.
@RequestMapping(value={"/mainAjaxPage"}, method=RequestMethod.GET)
public String displayMainAjaxPage() {
    // This method does nothing except return the logical name of the
    // view component (/jsp/home.jsp) that should be invoked in response
    // to this request.
    return "mainAjaxPage";
}
```

Now, rename home.jsp to mainAjaxPage.jsp.

Create New home.jsp

In this section, we'll create a new home.jsp file that will be our welcome/landing page for the application. Create a new JSP file called home.jsp and enter the following content:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="s" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Company Contacts</title>
        <!-- Bootstrap core CSS -->
        <link href="${pageContext.request.contextPath}/css/bootstrap.min.css" rel="stylesheet">

        <!-- SWC Icon -->
        <link rel="shortcut icon" href="${pageContext.request.contextPath}/img/icon.png">

    </head>
    <body>
        <div class="container">
            <h1>Company Contacts</h1>
            <hr/>
            <div class="navbar">
                <ul class="nav nav-tabs">
                    <li role="presentation" class="active">
                        <a href="${pageContext.request.contextPath}/home">Home</a>
                    </li>
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/mainAjaxPage">Contact List (Ajax)</a>
                    </li>
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/search">Search</a>
                    </li>
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/stats">Stats</a>
                    </li>
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/displayContactListNoAjax">
                            Contact List (No Ajax)
                        </a>
                    </li>
                </ul>
            </div>
        </div>

        <div class="container">
            <p><a href="${pageContext.request.contextPath}/login">Log In</a></p>

            <p>
                Welcome to the Company Contact demonstration project for the SWC Guild Java Cohort.
            </p>
        </div>

        <!-- Placed at the end of the document so the pages load faster -->
        <script src="${pageContext.request.contextPath}/js/jquery-1.11.1.min.js"></script>
        <script src="${pageContext.request.contextPath}/js/bootstrap.min.js"></script>

    </body>
</html>
```

Modifying the Nav Bar

The Home link in our nav bar will now go to our new welcome/landing page, which means we must modify our site navigation to include a link to the main Ajax page — we'll label this link "Contact List (Ajax)." Modify all pages containing the main nav so that the nav looks like the following (make sure you set class="active" properly on each page):

```
<div class="navbar">
   <ul class="nav nav-tabs">
       <li role="presentation" class="active">
           <a href="${pageContext.request.contextPath}/home">Home</a>
       </li>
       <li role="presentation">
           <a href="${pageContext.request.contextPath}/mainAjaxPage">Contact List (Ajax)</a>
       </li>
       <li role="presentation">
           <a href="${pageContext.request.contextPath}/search">Search</a>
       </li>
       <li role="presentation">
           <a href="${pageContext.request.contextPath}/stats">Stats</a>
       </li>
       <li role="presentation">
           <a href="${pageContext.request.contextPath}/displayContactListNoAjax">
               Contact List (No Ajax)
           </a>
       </li>
   </ul>
</div>
```

Maven Dependencies

Spring Security requires access to additional libraries. Add the following entries to your POM file:

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
    <version>3.1.4.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>3.1.4.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>3.1.4.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>3.1.4.RELEASE</version>
    <type>jar</type>
    <scope>compile</scope>
</dependency>
```

Tomcat Configuration

As mentioned above, Spring Security uses a servlet filter to intercept requests and make security decisions. We must register this component with Tomcat in the web.xml. We will be using a separate configuration file for Spring Security and must tell Spring about this new file — this is also done in the web.xml file. Make the following changes to your web.xml file:

Notes:

1. Security servlet filter entry
    a. /* tells Tomcat that this filter should be applied to all requests to this application
2. Register the Spring Security configuration file (spring-configuration.xml) with the Spring framework

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd"
         id="WebApp_ID" version="2.5">
    <display-name>Archetype Created Spring MVC Web Application</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>

    <error-page>
        <location>/error</location>
    </error-page>

    <servlet>
        <servlet-name>spring-dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>spring-dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.js</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.css</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.eot</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.svg</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.ttf</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.woff</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.jpg</url-pattern>
    </servlet-mapping>
```

```
    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.png</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>default</servlet-name>
        <url-pattern>*.gif</url-pattern>
    </servlet-mapping>

   <!-- #1 - Spring Security Filter changes START -->
    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <!-- #1a - Intercept ALL requests to this application -->
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/spring-dispatcher-servlet.xml
            <!-- #2 register spring-security.xml file with Spring -->
            /WEB-INF/spring-security.xml
            classpath:spring-persistence.xml
        </param-value>
    </context-param>
    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>

</web-app>
```

Spring Security Configuration

All Spring Security specific configuration information will be contained in **spring-security.xml**. We registered this file with Spring in the web.xml file above, telling Spring that it will be located in the WEB-INF folder. Create the file now and copy the following entries:

Notes:

1. Namespace for Spring Security configuration. This is set as the default XML namespace for this file so we don't have to use prefixes on our XML tags.
2. Make sure the login and home pages are not protected by Spring Security. All protected pages are redirected to the login page if the user is not already logged in. If the login page is protected, we get into an infinite loop. If the home page is protected, there is no part of our site accessible to non-logged-in users.

3. Authentication, login form, and protected endpoint configuration:
    a. Use an HTML form for authentication, use Spring Security to perform authentication
    b. Use this endpoint for the login form (this endpoint maps to our LoginController which simply displays our custom login JSP)
    c. Go back to the login form with a failure message if authentication fails
    d. Go back to the login form when the user logs out
    e. Protect these endpoints — they all require ROLE_ADMIN
    f. All other endpoints require ROLE_USER
4. Configure the Authentication Manager:
    a. We're using the default JDBC provider.
    b. We supply a dataSource (defined in the spring-persistence.xml file — Spring config files can see all the other Spring config files that are loaded) and tell Spring where to find the user's credentials. In this case, user passwords are in the Users table and user roles are in the Authorities table. We will create these tables in the next section.

```
 <!-- #1 - Make security the default namespace -->
<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-3.0.xsd
  http://www.springframework.org/schema/security
  http://www.springframework.org/schema/security/spring-security-3.1.xsd">

    <!-- #2 - Make sure we don't need authorization to get to the login or home page -->
    <http pattern="/login" security="none" />
    <http pattern="/home" security="none" />
    <http pattern="/" security="none" />
    <http pattern="/css/**" security="none" />
    <http pattern="/js/**" security="none" />
    <http pattern="/img/**" security="none" />
    <!-- #3 - Authentication/login form and protected endpoint configuration -->
    <http auto-config="true" use-expressions="false">
        <!-- #3a - Login via html form, use Spring to do the security check -->
        <!-- #3b - Use the login page at this endpoint -->
        <!-- #3c - Redirect here if login fails -->
        <form-login login-processing-url="/j_spring_security_check"
                    login-page="/login"
                    authentication-failure-url="/login?login_error=1"/>
        <!-- #3d - Go back to home page when user logs out -->
        <logout logout-success-url="/home" />
        <!-- #3e - Access to these endpoints require admin role -->
        <intercept-url pattern="/displayEditContactFormNoAjax" access="ROLE_ADMIN" />
        <intercept-url pattern="/displayNewContactFormNoAjax" access="ROLE_ADMIN" />
        <intercept-url pattern="/addNewContactNoAjax" access="ROLE_ADMIN" />
        <intercept-url pattern="/deleteContactNoAjax" access="ROLE_ADMIN" />
        <intercept-url pattern="/editContactNoAjax" access="ROLE_ADMIN" />
        <intercept-url pattern="/mainAjaxPage" access="ROLE_ADMIN" />
        <intercept-url pattern="/contacts" access="ROLE_ADMIN" />
        <intercept-url pattern="/contact" access="ROLE_ADMIN" />
        <intercept-url pattern="/contact/**" access="ROLE_ADMIN" />
        <!-- #3f - Access to all other controller endpoints require user role -->
        <intercept-url pattern="/**" access="ROLE_USER" />
    </http>
    <!-- #4 - Authentication Manager config -->
    <authentication-manager>
        <!-- #4a - Authentication Provider - we're using the JDBC service -->
        <authentication-provider>
            <!-- #4b - Tells Spring Security where to look for user information -->
            <!--       We use the dataSource defined in spring-persistence.xml  -->
            <!--       and we give Spring Security the query to use to lookup   -->
            <!--       the user's credentials (get the password from the users  -->
            <!--       tables and get the roles from the authorities table)     -->
            <jdbc-user-service id="userService"
                               data-source-ref="dataSource"
                               users-by-username-query=
          "select username, password, enabled from users where username=?"
                               authorities-by-username-query=
          "select username, authority from authorities where username=?" />
        </authentication-provider>
    </authentication-manager>
</beans:beans>
```

Database Changes

We have to create tables to hold the users and authorities information. Run the following script against your **contact_list** database. This script will create two new tables and populate them with test information.

```
--
-- Table structure for table `users`
--
CREATE TABLE IF NOT EXISTS `users` (
  `user_id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(20) NOT NULL,
  `password` varchar(20) NOT NULL,
  `enabled` tinyint(1) NOT NULL,
  PRIMARY KEY (`user_id`),
  KEY `username` (`username`)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=3 ;

--
-- Dumping data for table `users`
--
INSERT INTO `users` (`user_id`, `username`, `password`, `enabled`) VALUES
(1, 'test', 'password', 1),
(2, 'test2', 'password', 1);

--
-- Table structure for table `authorities`
--
CREATE TABLE IF NOT EXISTS `authorities` (
  `username` varchar(20) NOT NULL,
  `authority` varchar(20) NOT NULL,
  KEY `username` (`username`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Dumping data for table `authorities`
--
INSERT INTO `authorities` (`username`, `authority`) VALUES
('test', 'ROLE_ADMIN'),
('test', 'ROLE_USER'),
('test2', 'ROLE_USER');

--
-- Constraints for table `authorities`
--
ALTER TABLE `authorities`
  ADD CONSTRAINT `authorities_ibfk_1` FOREIGN KEY (`username`) REFERENCES `users` (`username`);
```

Application Code

We need to add some controller and view code to handle the login form for our application. We will also make some minor changes to existing view components. These changes will personalize the application for the user that is logged in.

Controllers

We need to create a new controller to handle login requests. Create a new Java class called LoginController in your controller package and insert the following code:

Notes:

1. Respond to all GET requests to /login, which will then go to our custom login JSP (login.jsp)

```
package com.swcguild.contactlistmvc.controller;

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

@Controller
public class LoginController {
    // #1 - respond to all GET requests for /login
    @RequestMapping(value="/login", method = RequestMethod.GET)
    public String showLoginForm() {
        return "login";
    }
}
```

Views

Although Spring Security can provide us with a generated login page, we want to use our own. Create a new JSP file called login.jsp (in the jsp folder) and insert the following code:

Notes:

1. This block allows us to display a message if the login attempt fails.
2. We must POST this form to the Spring Security endpoint j_spring_security_check. Spring will look at the submitted information and determine if the username/password combination is correct.
    a. The j_spring_security_check endpoint looks for the request parameter j_username, so that is what we must name the username field.
    b. Similar for j_password — Spring expects the password in a request parameter called j_password.

```
<%@ taglib prefix="s" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Contact List</title>
    </head>
    <body>
        <div>
            <h2>Sign in to the Contact List App</h2>
            <!-- #1 - If login_error == 1 then there was a failed login attempt -->
            <!--      so display an error message                                -->
            <c:if test="${param.login_error == 1}">
                <h3>Wrong id or password!</h3>
            </c:if>


            <!-- #2 - Post to Spring security to check our authentication -->
            <form method="post" class="signin" action="j_spring_security_check">
                <fieldset>
                    <table cellspacing="0">
                        <tr>
                            <th>
                                <label for="username">Username
                                </label>
                            </th>

                            <td><input id="username_or_email"
                                       name="j_username"
                                       type="text" />
                            </td>
                        </tr>
                        <tr>
                            <th><label for="password">Password</label></th>
                            <!-- #2b - must be j_password for Spring -->
                            <td><input id="password"
                                       name="j_password"
                                       type="password" />
                            </td>
                        </tr>
                        <tr>
                            <th></th>
                            <td><input name="commit" type="submit" value="Sign In" /></td>
                        </tr>
                    </table>
                </fieldset>
            </form>
        </div>
    </body>
</html>
```

Conditional Rendering of Page Elements

Even though we have protected all of our endpoints from unauthorized access, the links to these features still show up on our pages whether or not the user is logged in. In this step, we'll add code to the displayContactListNoAjax.jsp view component that displays the logged-in user's username and only shows the Edit, Delete, and Add links for authorized users.

JSP

Add the following to your displayContactListNoAjax.jsp (add the Logout link to all pages that have the Nav Bar).

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="s" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Company Contacts</title>
        <!-- Bootstrap core CSS -->
        <link href="${pageContext.request.contextPath}/css/bootstrap.min.css" rel="stylesheet">

        <!-- SWC Icon -->
        <link rel="shortcut icon" href="${pageContext.request.contextPath}/img/icon.png">

    </head>
    <body>
        <div class="container">
            <h1>Company Contacts</h1>
            <hr/>
            <div class="navbar">
                <ul class="nav nav-tabs">
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/home">Home</a>
                    </li>
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/mainAjaxPage">Contact List (Ajax)</a>
                    </li>
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/search">Search</a>
                    </li>
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/stats">Stats</a>
                    </li>
                    <li role="presentation" class="active">
                        <a href="${pageContext.request.contextPath}/displayContactListNoAjax">
                            Contact List (No Ajax)
                        </a>
                    </li>
                    <!-- #1 - Logout link -->
                     <li role="presentation">
                        <a href="${pageContext.request.contextPath}/j_spring_security_logout">Log Out</a>
                    </li>
                </ul>
            </div>
        </div>

        <div class="container">
            <h1>Company Contacts</h1>
            <!-- #2 - Personalized welcome message -->
            Hello <sec:authentication property="principal.username" />!<br/>
            <!-- #3 - Only render this link if user has admin role -->
            <sec:authorize access="hasRole('ROLE_ADMIN')">
            <a href="displayNewContactFormNoAjax">Add a Contact</a><br/>
            </sec:authorize>
            <hr/>
```

```
            <!-- Iterate over contactList: forEach contact in contactList, do something -->
            <c:forEach var="contact" items="${contactList}">
                <!-- Build custom delete url for each contact.  Use the id of the contact -->
                <!-- to specify the contact to delete or update                           -->
                <s:url value="deleteContactNoAjax"
                       var="deleteContact_url">
                    <s:param name="contactId" value="${contact.contactId}" />
                </s:url>
                <!-- Build custom edit url for each contact -->
                <s:url value="displayEditContactFormNoAjax"
                       var="editContact_url">
                    <s:param name="contactId" value="${contact.contactId}" />
                </s:url>
                <!-- A pointless demonstration of the if tag -->
                <c:if test="${contact.lastName == 'Doe'}">
                    CEO<br/>
                </c:if>
                Name: ${contact.firstName} ${contact.lastName}
                <!-- #4 - Only render Edit and Delete links for users with admin role -->
                <sec:authorize access="hasRole('ROLE_ADMIN')">
                | <a href="${deleteContact_url}">Delete</a> |
                <a href="${editContact_url}">Edit</a>
                </sec:authorize>
                <br/>
                Phone: ${contact.phone}<br/>
                Email: ${contact.email}<br/>
                <hr>
            </c:forEach>
        </div>

        <!-- Placed at the end of the document so the pages load faster -->
        <script src="${pageContext.request.contextPath}/js/jquery-1.11.1.min.js"></script>
        <script src="${pageContext.request.contextPath}/js/bootstrap.min.js"></script>

    </body>
</html>
```

Security Configuration

In order for the security tags inserted in displayContactList.jsp to work, we must add a bean definition for a WebSecurityExpressionHandler. This component is needed to interpret the hasRole('ROLE_ADMIN') expressions above. Update your spring-security.xml file so it looks like this:

```
<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-3.0.xsd
  http://www.springframework.org/schema/security
  http://www.springframework.org/schema/security/spring-security-3.1.xsd">

    <!-- Expression Handler bean definition START -->
    <beans:bean id="webexpressionHandler"
        class="org.springframework.security.web.access.expression.DefaultWebSecurityExpressionHandler"/>
    <!-- Expression Handler bean definition END -->

    <!-- #2 - Make sure we don't need authorization to get to the login page -->
    <http pattern="/login" security="none" />
    <http pattern="/home" security="none" />
    <http pattern="/" security="none" />
    <!-- #3 - Authentication/login form and protected endpoint configuration -->
    <http auto-config="true" use-expressions="false">
        <!-- #3a - Login via html form, use Spring to do the security check -->
        <!-- #3b - Use the login page at this endpoint -->
        <!-- #3c - Redirect here if login fails -->
        <form-login login-processing-url="/j_spring_security_check"
                    login-page="/login"
                    authentication-failure-url="/login?login_error=1"/>
        <!-- #3d - Go back to home page when user logs out -->
        <logout logout-success-url="/home" />
        <!-- #3e - Access to these endpoints require admin role -->
        <intercept-url pattern="/displayEditContactFormNoAjax" access="ROLE_ADMIN" />
        <intercept-url pattern="/displayNewContactFormNoAjax" access="ROLE_ADMIN" />
        <intercept-url pattern="/addNewContactNoAjax" access="ROLE_ADMIN" />
        <intercept-url pattern="/deleteContactNoAjax" access="ROLE_ADMIN" />
        <intercept-url pattern="/editContactNoAjax" access="ROLE_ADMIN" />
        <intercept-url pattern="/mainAjaxPage" access="ROLE_ADMIN" />
        <intercept-url pattern="/contacts" access="ROLE_ADMIN" />
        <intercept-url pattern="/contact" access="ROLE_ADMIN" />
        <intercept-url pattern="/contact/**" access="ROLE_ADMIN" />
        <!-- #3f - Access to all other controller endpoints require user role -->
        <intercept-url pattern="/**" access="ROLE_USER" />
    </http>
    <!-- #4 - Authentication Manager config -->
    <authentication-manager>
        <!-- #4a - Authentication Provider - we're using the JDBC service -->
        <authentication-provider>
            <!-- #4b - Tells Spring Security where to look for user information -->
            <!--       We use the dataSource defined in spring-persistence.xml  -->
            <!--       and we give Spring Security the query to use to lookup   -->
            <!--       the user's credentials (get the password from the users  -->
            <!--       tables and get the roles from the authorities table)     -->
            <jdbc-user-service id="userService"
                               data-source-ref="dataSource"
                               users-by-username-query=
          "select username, password, enabled from users where username=?"
                               authorities-by-username-query=
          "select username, authority from authorities where username=?" />
        </authentication-provider>
    </authentication-manager>
</beans:beans>
```

Wrap-up

In this step you have learned how to do the following:

1. Add Spring Security dependencies to the POM
2. Configure Tomcat to use the Spring Security servlet filter to intercept requests and perform security checks
3. Configure Spring Security settings in a separate file and register that file with the Spring framework
4. Create the database tables needed for Spring Security authentication and authorization
5. Create a custom controller and view for the login form
6. Provide a way for users to log out of the application
7. Limit access to selected application endpoints to only users with the required roles
8. Customize the rendering of views based on the user that is logged into the application