Spring security :

Spring Security is a framework which provides various security features like: authentication, authorization to create secure Java Enterprise Applications.

It overcomes all the problems that come during creating non spring security applications and manage new server environment for the application.

This framework targets two major areas of application are authentication and authorization. Authentication is the process of knowing and identifying the user that wants to access.

**Authorization** is the process to allow authority to perform actions in the application.

**Technologies that support Spring Security Integration**

Spring Security framework supports wide range of authentication models. These models either provided by third parties or framework itself. Spring Security supports integration with all of these technologies.

* HTTP BASIC authentication headers
* HTTP Digest authentication headers
* HTTP X.509 client certificate exchange
* LDAP (Lighweight Directory Access Protocol)
* Form-based authentication
* OpenID authentication
* Automatic remember-me authentication
* Kerberos
* JOSSO (Java Open Source Single Sign-On)
* AppFuse
* AndroMDA
* Mule ESB
* DWR(Direct Web Request)

Advantages

Spring Security has numerous advantages. Some of that are given below.

* Comprehensive support for authentication and authorization.
* Protection against common tasks
* Servlet API integration
* Integration with Spring MVC
* Portability
* CSRF protection
* Java Configuration support

### **LDAP (Lightweight Directory Access Protocol)**

It is an open application protocol for maintaining and accessing distributed directory information services over an Internet Protocol.

### **Basic Access Authentication**

Spring Security supports Basic Access Authentication that is used to provide user name and password while making request over the network.

### **Web Form Authentication**

In this process, web form collect and authenticate user credentials from the web browser. Spring Security supports it while we want to implement web form authentication.

### **Authorization**

Spring Security provides the this feature to authorize the user before accessing resources. It allows developers to define access policies against the resources.

### **HTTP Authorization**

Spring provides this feature for HTTP authorization of web request URLs using Apache Ant paths or regular expressions.

## Features added in Spring Security 5.0

### **OAuth 2.0 Login**

This feature provides the facility to the user to login into the application by using their existing account at GitHub or Google. This feature is implemented by using the Authorization Code Grant that is specified in the OAuth 2.0 Authorization Framework.

### **Reactive Support**

From version Spring Security 5.0, it provides reactive programming and reactive web runtime support and even, we can integrate with Spring WebFlux.

### **Modernized Password Encoding**

Spring Security 5.0 introduced new Password encoder **DelegatingPasswordEncoder** which is more modernize and solve all the problems of previous encoder **NoOpPasswordEncoder**.

In Spring Security 3.0, the Security module is divided into separate jar files. The purpose was to divide jar files based on their functionalities, so, the developer can integrate according to their requirement.

It also helps to set required dependency into pom.xml file of maven project.

**The following are the jar files that are included into Spring Security module**.

* spring-security-core.jar
* spring-security-remoting.jar
* spring-security-web.jar
* spring-security-config.jar
* spring-security-ldap.jar
* spring-security-oauth2-core.jar
* spring-security-oauth2-client.jar
* spring-security-oauth2-jose.jar
* spring-security-acl.jar
* spring-security-cas.jar
* spring-security-openid.jar
* spring-security-test.jar

# **Spring Project Modules**

In Spring Security 3.0, the Security module is divided into separate jar files. The purpose was to divide jar files based on their functionalities, so, the developer can integrate according to their requirement.

It also helps to set required dependency into pom.xml file of maven project.

The following are the jar files that are included into Spring Security module.

* **spring-security-core.jar**
* spring-security-remoting.jar
* **spring-security-web.jar**
* **spring-security-config.jar**
* spring-security-ldap.jar
* spring-security-oauth2-core.jar
* spring-security-oauth2-client.jar
* spring-security-oauth2-jose.jar
* spring-security-acl.jar
* spring-security-cas.jar
* spring-security-openid.jar
* spring-security-test.jar

## Core - spring-security-core.jar

This is core jar file and required for every application that wants to use Spring Security. This jar file includes core **access-control** and **core authentication classes and interfaces**. We can use it in standalone applications or remote clients applications.

It contains top level packages:

* org.springframework.security.core
* org.springframework.security.access
* org.springframework.security.authentication
* org.springframework.security.provisioning

## Remoting - spring-security-remoting.jar

This jar is used to integrate security feature into the Spring remote application. We don't need it until or unless we are creating remote application. All the classes and interfaces are located into **org.springframework.security.remoting** package.

## Web - spring-security-web.jar

This jar is useful for Spring Security web authentication and URL-based access control. It includes filters and web-security infrastructure.

All the classes and interfaces are located into the **org.springframework.security.web** package.

## Config - spring-security-config.jar

This jar file is required for Spring Security configuration using XML and Java both. It includes Java configuration code and security namespace parsing code. All the classes and interfaces are stored in **org.springframework.security.config** package.

## LDAP - spring-security-ldap.jar

This jar file is required only if we want to use LDAP (Lighweight Directory Access Protocol). It includes authentication and provisioning code. All the classes and interfaces are stored into **org.springframework.security.ldap** package

**To implement Spring Security in Spring application, we can configure it either by using XML or Java based configuration**.

### **Spring Security Project Source Code**

Our project name is **springsecurity** and contains the following source files.

### **Controller**

**HomeController.java**

1. @Controller
2. **public** **class** HomeController {
4. @RequestMapping(value="/", method=RequestMethod.GET)
5. **public** String home() {
6. **return** "home";
7. }
9. @RequestMapping(value="/admin", method=RequestMethod.GET)
10. **public** String privateHome() {
11. **return** "privatePage";
12. }
13. }

### **Spring Security Configuration**

**spring-security.xml**

1. **<beans:beans** xmlns="http://www.springframework.org/schema/security"
2. xmlns:beans="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://www.springframework.org/schema/beans
5. http://www.springframework.org/schema/beans/spring-beans.xsd
6. http://www.springframework.org/schema/security
7. http://www.springframework.org/schema/security/spring-security.xsd"**>**
8. **<http** auto-config="true"**>**
9. **<intercept-url** pattern="/admin" access="hasRole('ROLE\_ADMIN')" **/>**
10. **</http>**
11. **<authentication-manager>**
12. **<authentication-provider>**
13. **<user-service>**
14. **<user** name="admin" password="1234" authorities="hasRole(ROLE\_ADMIN)" **/>**
15. **</user-service>**
16. **</authentication-provider>**
17. **</authentication-manager>**
18. **</beans:beans>**

### **Servlet Dispatcher**

**spring-servlet.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"
4. xmlns:mvc="http://www.springframework.org/schema/mvc"
5. xmlns:context="http://www.springframework.org/schema/context"
6. xsi:schemaLocation="
7. http://www.springframework.org/schema/mvc
8. http://www.springframework.org/schema/mvc/spring-mvc.xsd
9. http://www.springframework.org/schema/beans
10. http://www.springframework.org/schema/beans/spring-beans.xsd
11. http://www.springframework.org/schema/context
12. http://www.springframework.org/schema/context/spring-context.xsd"**>**
13. **<mvc:annotation-driven** **/>**
14. **<context:component-scan** base-package="com.javatpoint.controller"**>**
15. **</context:component-scan>**
16. **<context:annotation-config></context:annotation-config>**
17. **<bean** class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>**
18. **<property** name="prefix" value="/WEB-INF/views/"**></property>**
19. **<property** name="suffix" value=".jsp"**></property>**
20. **</bean>**
21. **</beans>**

### **Web Descriptor**

**web.xml**

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. <!DOCTYPE xml**>**
3. **<web-app** xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
6. http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"
7. version="3.1"**>**
9. <!-- Spring Configuration -->
10. **<servlet>**
11. **<servlet-name>**spring**</servlet-name>**
12. **<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet- class>**
13. **<load-on-startup>**1**</load-on-startup>**
14. **</servlet>**
15. **<servlet-mapping>**
16. **<servlet-name>**spring**</servlet-name>**
17. **<url-pattern>**/**</url-pattern>**
18. **</servlet-mapping>**

  <!—Spring Security configuration filter -->

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

  <!—Listener to load all xml files -->

1. **<listener>**
2. **<listener-class>**org.springframework.web.context.ContextLoaderListener**</listener-class>**
3. **</listener>**
4. **<context-param>**
5. **<param-name>**contextConfigLocation**</param-name>**
6. **<param-value>**
7. /WEB-INF/spring-servlet.xml
8. /WEB-INF/spring-security.xml
9. **</param-value>**
10. **</context-param>**
11. **</web-app>**

### **Project Dependencies**

**pom.xml**

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4. http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
5. **<modelVersion>**4.0.0**</modelVersion>**
6. **<groupId>**com.javatpoint**</groupId>**
7. **<artifactId>**springsecurity**</artifactId>**
8. **<version>**0.0.1-SNAPSHOT**</version>**
9. **<packaging>**war**</packaging>**
10. **<properties>**
11. **<maven.compiler.target>**1.8**</maven.compiler.target>**
12. **<maven.compiler.source>**1.8**</maven.compiler.source>**
13. **</properties>**
14. **<dependencies>**
15. **<dependency>**
16. **<groupId>**org.springframework**</groupId>**
17. **<artifactId>**spring-webmvc**</artifactId>**
18. **<version>**5.0.2.RELEASE**</version>**
19. **</dependency>**
20. **<dependency>**
21. **<groupId>**org.springframework.security**</groupId>**
22. **<artifactId>**spring-security-web**</artifactId>**
23. **<version>**5.0.0.RELEASE**</version>**
24. **</dependency>**
25. **<dependency>**
26. **<groupId>**org.springframework.security**</groupId>**
27. **<artifactId>**spring-security-core**</artifactId>**
28. **<version>**5.0.0.RELEASE**</version>**
29. **</dependency>**
30. **<dependency>**
31. **<groupId>**org.springframework.security**</groupId>**
32. **<artifactId>**spring-security-config**</artifactId>**
33. **<version>**5.0.0.RELEASE**</version>**
34. **</dependency>**
36. **<dependency>**
37. **<groupId>**javax.servlet**</groupId>**
38. **<artifactId>**javax.servlet-api**</artifactId>**
39. **<version>**3.1.0**</version>**
40. **<scope>**provided**</scope>**
41. **</dependency>**
42. **</dependencies>**
43. **<build>**
44. **<plugins>**
45. **<plugin>**
46. **<groupId>**org.apache.maven.plugins**</groupId>**
47. **<artifactId>**maven-war-plugin**</artifactId>**
48. **<version>**2.6**</version>**
49. **<configuration>**
50. **<failOnMissingWebXml>**false**</failOnMissingWebXml>**
51. **</configuration>**
52. **</plugin>**
53. **</plugins>**
54. **</build>**
55. **</project>**

### **View Pages**

**home.jsp**

1. **<html>**
2. **<head>**
3. **<meta** content="text/html; charset=UTF-8"**>**
4. **<title>**Home**</title>**
5. **</head>**
6. **<body>**
7. **<h2>**Welcome to javatpoint spring tutorial!**</h2>**
8. **</body>**
9. **</html>**

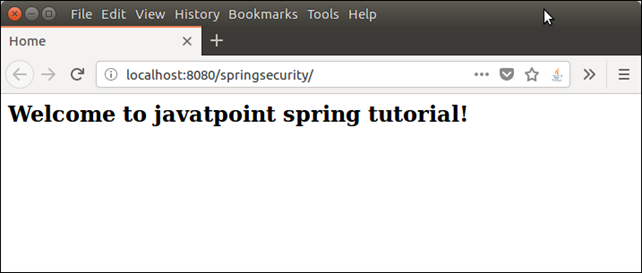
**privatePage.jsp**

1. **<html>**
2. **<head>**
3. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8"**>**
4. **<title>**Admin**</title>**
5. **</head>**
6. **<body>**
7. Hello Admin
8. **</body>**
9. **</html>**

Output

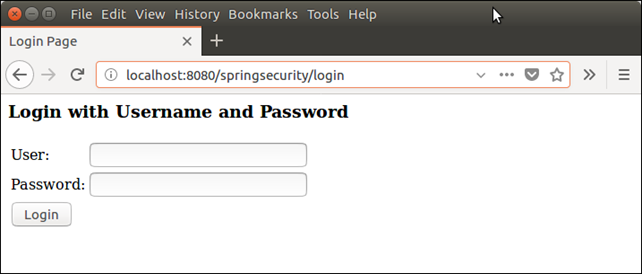
This example is executed using Apache Tomcat v9.0. After running it produces the following output to the browser.

Initially, it renders **home.jsp** page that displays the following output.



We added spring security to admin page, if we enter /**admin** to the browser, application produces the below output.

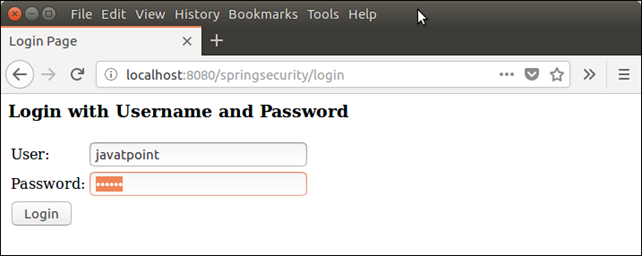
**Request URL : http://localhost:8080/springsecurity/admin**



Now, this the actual magic that spring security provides to protect resources from unauthentic users.

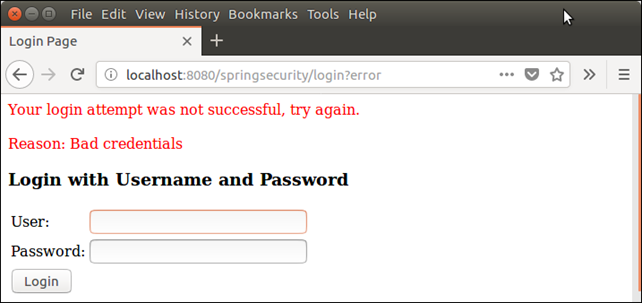
This is spring security provided module, we did not create it. It also validate the user input.

**Providing wrong credentials.**



If we provide wrong login credentials, it will validate with the user name and password, we mentioned in **spring-security.xml** file.

After validating, if login credentials are incorrect, it throws an error message.



Well, in this example, we have seen the Spring Security's login module and how it validates corresponds to the provided user name and password.

In next, topic we will implement further logic like: render user after login successfully.

**Q:->     What is <http auto-config="true"> ?**

Ans:

auto-config="true" is equivalent to:

<http>

<form-login />

<http-basic />

<logout />

</http>

so it gives you a very basic security configuration to boot.

auto-config='true' means for the intercept-url pattern the spring security provides the default login screen