Spring Security Introduction

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

It is a sub-project of Spring framework which was started in 2003 by Ben Alex. Later on, in 2004, It was released under the Apache License as Spring Security 2.0.0.

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.

C++ vs Java

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

We can apply authorization to authorize web request, methods and access to individual domain.

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)

The beauty of this framework is its flexible authentication nature to integrate with any software solution. Sometimes, developers want to integrate it with a legacy system that does not follow any security standard, there Spring Security works nicely.

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

Spring Security History

In late 2003, a project **Acegi Security System for Spring** started with the intention to develop a Spring-based security system. So, a simple security system was implemented but not released officially. Developers used that code internally for their solutions and by 2004 about 20 developers were using that.

Initially, authentication module was not part of the project, around a year after, module was added and complete project was reconfigure to support more technologies.

After some time this project became a subproject of Spring framework and released as 1.0.0 in 2006.

in 2007, project is renamed to Spring Security and widely accepted. Currently, it is recognized and supported by developers open community world wide.

# Spring Security Features

* LDAP (Lightweight Directory Access Protocol)
* Single sign-on
* JAAS (Java Authentication and Authorization Service) LoginModule
* Basic Access Authentication
* Digest Access Authentication
* Remember-me
* Web Form Authentication
* Authorization
* Software Localization
* HTTP Authorization

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

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

### **Single sign-on**

This feature allows a user to access multiple applications with the help of single account(user name and password).

### **JAAS (Java Authentication and Authorization Service) LoginModule**

It is a Pluggable Authentication Module implemented in Java. Spring Security supports it for its authentication process.

### **Basic Access Authentication**

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

13.4M

216

Triggers in SQL (Hindi)

### **Digest Access Authentication**

This feature allows us to make authentication process more secure than Basic Access Authentication. It asks to the browser to confirm the identity of the user before sending sensitive data over the network.

### **Remember-me**

Spring Security supports this feature with the help of HTTP Cookies. It remember to the user and avoid login again from the same machine until the user logout.

### **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.

### **Software Localization**

This feature allows us to make application user interface in any language.

### **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**.

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.

24.1M

501

Java Try Catch

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.

OAuth 2.0 Core - spring-security-oauth2-core.jar

This jar is required to integrate Oauth 2.0 Authorization Framework and OpenID Connect Core 1.0 into the application. This jar file includes the core classes for OAuth 2.0 and classes are stored into the **org.springframework.security.oauth2.core** package.

OAuth 2.0 Client - spring-security-oauth2-client.jar

This jar file is required to get client support for OAuth 2.0 Authorization Framework and OpenID Connect Core 1.0. This module provides OAuth login and OpenID client support. All the classes and interfaces are available from **org.springframework.security.oauth2.client** package.

OAuth 2.0 JOSE - spring-security-oauth2-jose.jar

It provides Spring Security's support for the JOSE (Javascript Object Signing and Encryption) framework. The JOSE framework provides methods to establish secure connection between clients. It contains following collection of specifications:

* JWT (JSON Web Token)
* JWS (JSON Web Signature)
* JWE (JSON Web Encryption)
* JWK (JSON Web Key)

All the classes and interfaces are available into these two packages:

**org.springframework.security.oauth2.jwt** and **org.springframework.security.oauth2.jose.**

ACL - spring-security-acl.jar

This jar is used to apply security to domain object in the application. We can access classes and code from the **org.springframework.security.acls** package.

CAS - spring-security-cas.jar

It is required for Spring Security?s CAS client integration. We can use it to integrate Spring Security web authentication with CAS single sign-on server. The source code is located into **org.springframework.security.cas** package.

OpenID - spring-security-openid.jar

This jar is used for OpenID web authentication support. We can use it to authenticate users against an external OpenID server. It requires OpenID4Java and top level package is **org.springframework.security.openid**.

Test - spring-security-test.jar

This jar provides support for testing Spring Security application.

# Spring Security Project

In this tutorial, we will implement **Spring Security with the Spring MVC** framework. All the examples are Spring MVC and **created using Maven project**.

We are using **Spring Security 5.0.0.RELEASE** version and following are the maven dependencies, we used in all the examples.

1. **<dependency>**
2. **<groupId>**org.springframework.security**</groupId>**
3. **<artifactId>**spring-security-web**</artifactId>**
4. **<version>**5.0.0.RELEASE**</version>**
5. **</dependency>**
6. **<dependency>**
7. **<groupId>**org.springframework.security**</groupId>**
8. **<artifactId>**spring-security-core**</artifactId>**
9. **<version>**5.0.0.RELEASE**</version>**
10. **</dependency>**
11. **<dependency>**
12. **<groupId>**org.springframework.security**</groupId>**
13. **<artifactId>**spring-security-config**</artifactId>**
14. **<version>**5.0.0.RELEASE**</version>**
15. **</dependency>**

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

Let's see an example, in which we will use XML to configure the Spring Security.

23.4M

488

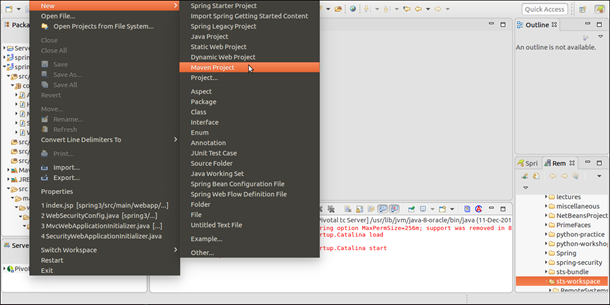
HTML Tutorial

**Next**

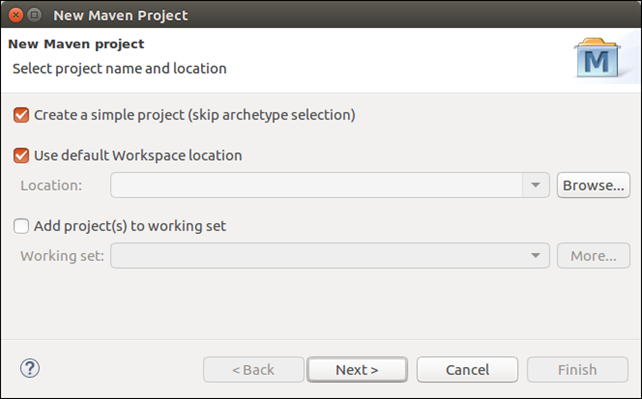
**Stay**

### **Create a Maven Project**

Click on **File** menu locate to **New→Maven Project**, as we did in the following screen shot.

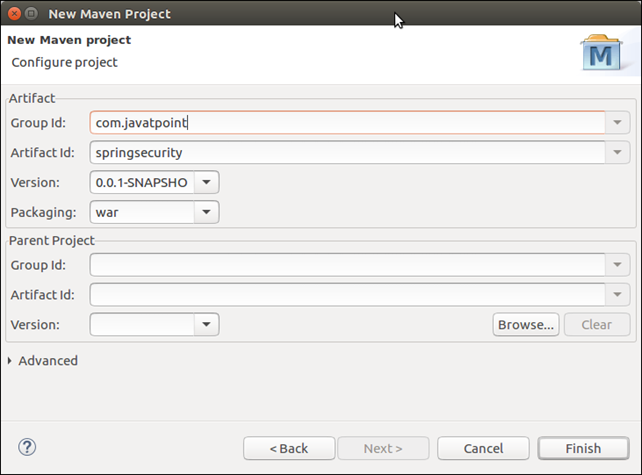


### **Select Project Name and Location**

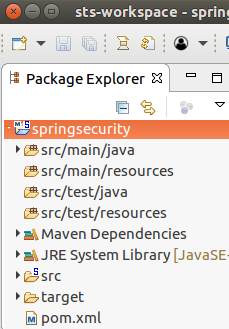


### **Provide Project Name**

Provide project name and select packaging type as **war (Web Archive)** as we did below.

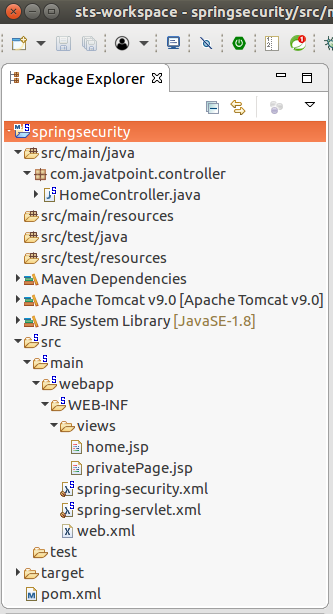


**Finish** the project and it will create an empty directory structure of the project like below.



Initially, it is empty. So, let's create a Spring MVC application and integrate with Spring Security.

This is our project layout. It contains a controller, three XML files and two JSP files.



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

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

### **Controller**

**HomeController.java**

1. **package** com.javatpoint.controller;
3. **import** org.springframework.stereotype.Controller;
4. **import** org.springframework.web.bind.annotation.RequestMapping;
5. **import** org.springframework.web.bind.annotation.RequestMethod;
7. @Controller
8. **public** **class** HomeController {
10. @RequestMapping(value="/", method=RequestMethod.GET)
11. **public** String home() {
12. **return** "home";
13. }
15. @RequestMapping(value="/admin", method=RequestMethod.GET)
16. **public** String privateHome() {
17. **return** "privatePage";
18. }
19. }

### **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>**
20. **<listener>**
21. **<listener-class>**org.springframework.web.context.ContextLoaderListener**</listener-class>**
22. **</listener>**
24. **<filter>**
25. **<filter-name>**springSecurityFilterChain**</filter-name>**
26. **<filter-class>**org.springframework.web.filter.DelegatingFilterProxy**</filter-class>**
27. **</filter>**
28. **<filter-mapping>**
29. **<filter-name>**springSecurityFilterChain**</filter-name>**
30. **<url-pattern>**/\***</url-pattern>**
31. **</filter-mapping>**
33. **<context-param>**
34. **<param-name>**contextConfigLocation**</param-name>**
35. **<param-value>**
36. /WEB-INF/spring-servlet.xml
37. /WEB-INF/spring-security.xml
38. **</param-value>**
39. **</context-param>**
40. **</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**

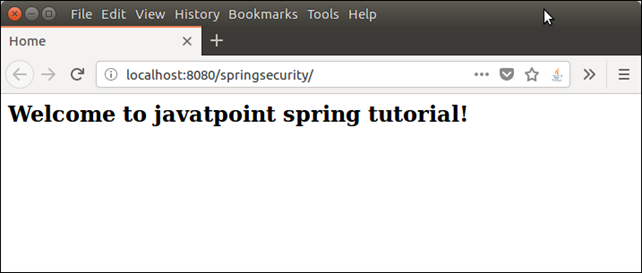
**home.jsp**

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

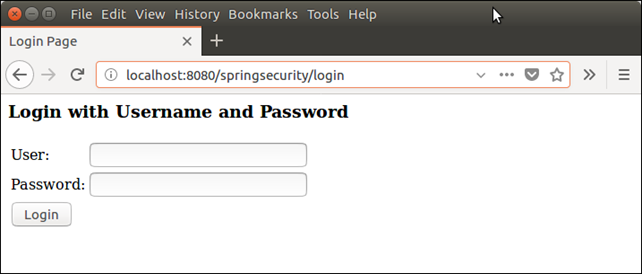
This example is executed using Apache Tomcat v9.0. After running it produces the following output to the browser.

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.

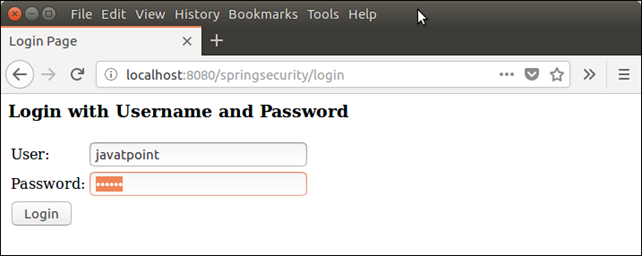
**Request URL : http://localhost:8080/springsecurity/admin**



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

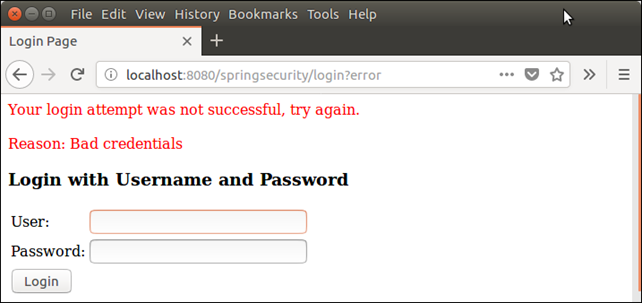
This is spring security provided module, we did not create it. It also validate the user input.

**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.

# Spring Security Project using Java Configuration

Spring Framework added Java configuration support in Spring 3.1. In Spring Security, Java configuration was added to Spring Security 3.2 that allows us to configure Spring Security **without writing single line of XML.**

Here, we will create an example that implements Spring Security and configured without using XML. It includes the following steps.

### **Step 1**

The first step is to create a Spring Security Java configuration. A simple basic Java Configuration is given below.

**WebSecurityConfig.java**

30M

585

Difference between JDK, JRE, and JVM

1. **package** com.javatpoint;
3. **import** org.springframework.context.annotation.\*;
4. //import org.springframework.security.config.annotation.authentication.builders.\*;
5. **import** org.springframework.security.config.annotation.web.builders.HttpSecurity;
6. **import** org.springframework.security.config.annotation.web.configuration.\*;
7. **import** org.springframework.security.core.userdetails.User;
8. **import** org.springframework.security.core.userdetails.UserDetailsService;
9. **import** org.springframework.security.provisioning.InMemoryUserDetailsManager;
10. **import** org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
12. @EnableWebSecurity
13. @ComponentScan("com.javatpoint")
14. **public** **class** WebSecurityConfig **implements** WebMvcConfigurer {
16. @Bean
17. **public** UserDetailsService userDetailsService() **throws** Exception {
18. InMemoryUserDetailsManager manager = **new** InMemoryUserDetailsManager();
19. manager.createUser(User.withDefaultPasswordEncoder().username("javatpoint").
20. password("java123").roles("USER").build());
21. **return** manager;
22. }
24. **protected** **void** configure(HttpSecurity http) **throws** Exception {
26. http
27. .antMatcher("/")
28. .authorizeRequests()
29. .anyRequest().hasRole("ADMIN")
30. .and()
31. .httpBasic();
32. }
33. }

This configuration creates a Servlet Filter known as the **springSecurityFilterChain**. It is **responsible for** protecting the application URLs, validating submit username and password, redirecting to the login form etc.

The above Java Configuration do the following for our application.

* Require authentication for every URL
* Creates a login form
* Allow user to authenticate using form based authentication
* Allow to logout
* Prevent from CSRF attack
* Security Header Integration, etc

### **Step 2**

Now, we will register **springSecurityFilterChain** with the war. To register, Spring Security provides a base class AbstractSecurityWebApplicationInitializer that we need to extend.

For Spring MVC application, SecurityWebApplicationInitializer will look like below.

**SecurityWebApplicationInitializer.java**

1. **package** com.javatpoint;
2. **import** org.springframework.security.web.context.\*;
4. **public** **class** SecurityWebApplicationInitializer
5. **extends** AbstractSecurityWebApplicationInitializer {
7. }

This code will register the springSecurityFilterChain for every URL in our application.

### **Step 3**

Now, load WebSecurityConfig in our existing ApplicationInitializer and add into the getRootConfigClasses() method.

**MvcWebApplicationInitializer.java**

1. **package** com.javatpoint;
3. **import** org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
4. **public** **class** MvcWebApplicationInitializer **extends**
5. AbstractAnnotationConfigDispatcherServletInitializer {
6. @Override
7. **protected** Class<?>[] getRootConfigClasses() {
8. **return** **new** Class[] { WebSecurityConfig.**class** };
9. }
10. @Override
11. **protected** Class<?>[] getServletConfigClasses() {
12. // TODO Auto-generated method stub
13. **return** **null**;
14. }
15. @Override
16. **protected** String[] getServletMappings() {
17. **return** **new** String[] { "/" };
18. }
19. }

### **Step 4**

**WebSecurityConfigurerAdapter** class provides a configure(HttpSecurity http) method that contains the following default configuration. Default definition looks like below.

1. **protected** **void** configure(HttpSecurity http) **throws** Exception {
2. http
3. .authorizeRequests()
4. .anyRequest().authenticated()
5. .and()
6. .formLogin()
7. .and()
8. .httpBasic();
9. }

It is similar to the given XML.

1. **<http>**
2. **<intercept-url** pattern="/\*\*" access="authenticated"**/>**
3. **<form-login** **/>**
4. **<http-basic** **/>**
5. **</http>**

This method does the following things.

* It ensures that each request made by the user requires to the user to be authenticated
* It allows user to authenticate by using form based login
* It allows user to authenticate with HTTP Basic authentication

### **Step 5**

Creating a controller to handle user requests.

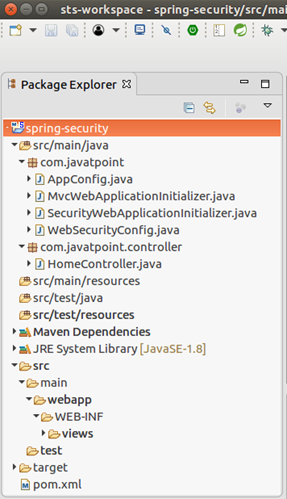
**HomeController.java**

1. **package** com.javatpoint.controller;
2. **import** org.springframework.stereotype.Controller;
3. **import** org.springframework.web.bind.annotation.RequestMapping;
4. **import** org.springframework.web.bind.annotation.RequestMethod;
5. @Controller
6. **public** **class** HomeController {
8. @RequestMapping(value="/", method=RequestMethod.GET)
9. **public** String index() {
11. **return** "index";
12. }
13. }

We have one view (.jsp) page **index.jsp**, it contains the following source code.

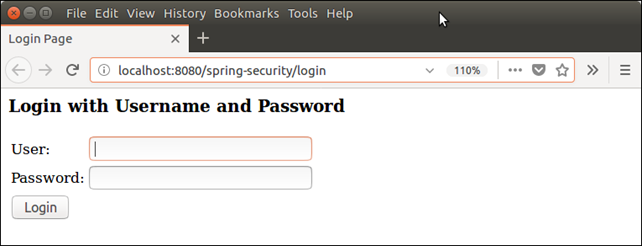
1. **<html>**
2. **<head>**
3. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8"**>**
4. **<title>**Home Page**</title>**
5. **</head>**
6. **<body>**
7. Welcome to home page!
8. **</body>**
9. **</html>**

Our complete project looks like the below.



Output:

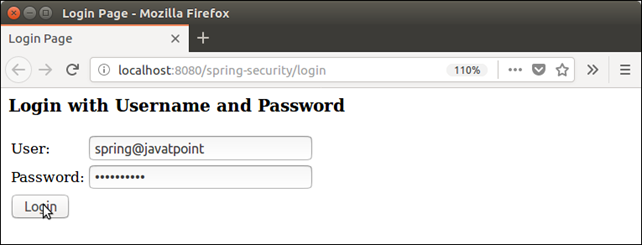
We have a single action in our controller and it can be accessed only by authentic user. So, when we run the application, it prompts for the login credentials. The output is given below.



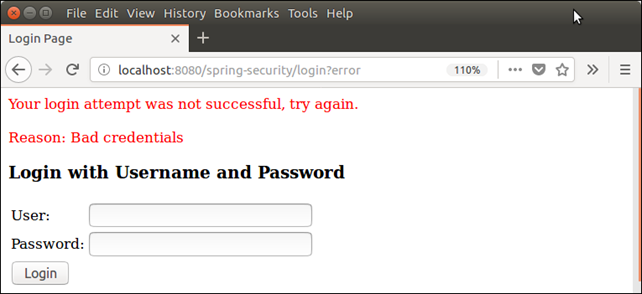
This is **default login** page provided by the Spring Security, we did not create it. Although we can create our own login page and configure with the application. We will do this in our next topics.

Well, now, provide the login credentials to get into the application resource. Spring Security validate user credentials and make sure that user is authentic.

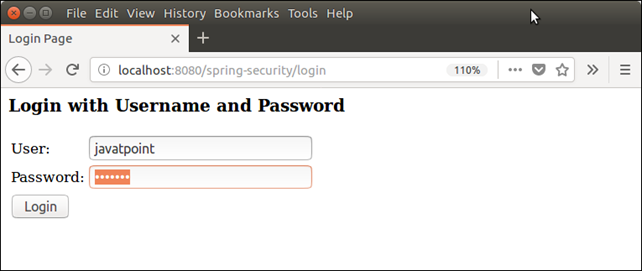
Let's see, what happen? If we enter wrong credentials.



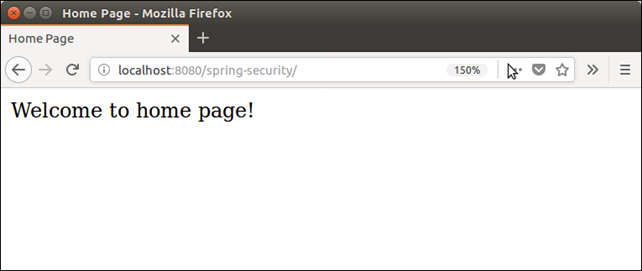
After click on login button, it throws **Bad Credentials** error.



Now, login with **correct credentials.**



This time credentials are matched and shows our home page (index.jsp).



# Spring Security Login-Logout Module Example

Spring Security provides login and logout features that we can use in our application. It is helpful to create secure Spring application.

Here, we are creating a Spring MVC application with Spring Security and implementing login and logout features.

First we created a maven project and provided following project dependencies in pom.xml file.

### **Project Dependencies**

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
2. **<modelVersion>**4.0.0**</modelVersion>**
3. **<groupId>**com.javatpoint**</groupId>**
4. **<artifactId>**springSecurityLoginOut**</artifactId>**
5. **<version>**0.0.1-SNAPSHOT**</version>**
6. **<packaging>**war**</packaging>**
7. **<properties>**
8. **<maven.compiler.target>**1.8**</maven.compiler.target>**
9. **<maven.compiler.source>**1.8**</maven.compiler.source>**
10. **</properties>**
11. **<dependencies>**
12. **<dependency>**
13. **<groupId>**org.springframework**</groupId>**
14. **<artifactId>**spring-webmvc**</artifactId>**
15. **<version>**5.0.2.RELEASE**</version>**
16. **</dependency>**
17. **<dependency>**
18. **<groupId>**org.springframework.security**</groupId>**
19. **<artifactId>**spring-security-web**</artifactId>**
20. **<version>**5.0.0.RELEASE**</version>**
21. **</dependency>**
22. **<dependency>**
23. **<groupId>**org.springframework.security**</groupId>**
24. **<artifactId>**spring-security-core**</artifactId>**
25. **<version>**5.0.0.RELEASE**</version>**
26. **</dependency>**
27. **<dependency>**
28. **<groupId>**org.springframework.security**</groupId>**
29. **<artifactId>**spring-security-config**</artifactId>**
30. **<version>**5.0.0.RELEASE**</version>**
31. **</dependency>**
33. <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
34. **<dependency>**
35. **<groupId>**javax.servlet**</groupId>**
36. **<artifactId>**javax.servlet-api**</artifactId>**
37. **<version>**3.1.0**</version>**
38. **<scope>**provided**</scope>**
39. **</dependency>**
40. **<dependency>**
41. **<groupId>**javax.servlet**</groupId>**
42. **<artifactId>**jstl**</artifactId>**
43. **<version>**1.2**</version>**
44. **</dependency>**
45. **</dependencies>**
46. **<build>**
47. **<plugins>**
48. **<plugin>**
49. **<groupId>**org.apache.maven.plugins**</groupId>**
50. **<artifactId>**maven-war-plugin**</artifactId>**
51. **<version>**2.6**</version>**
52. **<configuration>**
53. **<failOnMissingWebXml>**false**</failOnMissingWebXml>**
54. **</configuration>**
55. **</plugin>**
56. **</plugins>**
57. **</build>**
58. **</project>**

### **Spring Security Configuration**

After that we created configuration files to enable login feature and allowed access to the authorized user only.

30M

585

Difference between JDK, JRE, and JVM

This project contains the following four Java files.

**AppConfig.java**

1. **package** com.javatpoint;
2. **import** org.springframework.context.annotation.Bean;
3. **import** org.springframework.context.annotation.ComponentScan;
4. **import** org.springframework.context.annotation.Configuration;
5. **import** org.springframework.web.servlet.config.annotation.EnableWebMvc;
6. **import** org.springframework.web.servlet.view.InternalResourceViewResolver;
7. **import** org.springframework.web.servlet.view.JstlView;
9. @EnableWebMvc
10. @Configuration
11. @ComponentScan({ "com.javatpoint.controller.\*" })
12. **public** **class** AppConfig {
13. @Bean
14. **public** InternalResourceViewResolver viewResolver() {
15. InternalResourceViewResolver viewResolver
16. = **new** InternalResourceViewResolver();
17. viewResolver.setViewClass(JstlView.**class**);
18. viewResolver.setPrefix("/WEB-INF/views/");
19. viewResolver.setSuffix(".jsp");
20. **return** viewResolver;
21. }
22. }

**MvcWebApplicationInitializer.java**

1. **package** com.javatpoint;
2. **import** org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
3. **public** **class** MvcWebApplicationInitializer **extends**
4. AbstractAnnotationConfigDispatcherServletInitializer {
5. @Override
6. **protected** Class<?>[] getRootConfigClasses() {
7. **return** **new** Class[] { WebSecurityConfig.**class** };
8. }
9. @Override
10. **protected** Class<?>[] getServletConfigClasses() {
11. // TODO Auto-generated method stub
12. **return** **null**;
13. }
14. @Override
15. **protected** String[] getServletMappings() {
16. **return** **new** String[] { "/" };
17. }
18. }

**SecurityWebApplicationInitializer.java**

1. **package** com.javatpoint;
2. **import** org.springframework.security.web.context.\*;
3. **public** **class** SecurityWebApplicationInitializer
4. **extends** AbstractSecurityWebApplicationInitializer {
5. }

**WebSecurityConfig.java**

1. **package** com.javatpoint;
2. **import** org.springframework.context.annotation.\*;
3. //import org.springframework.security.config.annotation.authentication.builders.\*;
4. **import** org.springframework.security.config.annotation.web.builders.HttpSecurity;
5. **import** org.springframework.security.config.annotation.web.configuration.\*;
6. **import** org.springframework.security.core.userdetails.User;
7. **import** org.springframework.security.core.userdetails.UserDetailsService;
8. **import** org.springframework.security.provisioning.InMemoryUserDetailsManager;
9. **import** org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
10. @EnableWebSecurity
11. @ComponentScan("com.javatpoint")
12. **public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {
14. @Bean
15. **public** UserDetailsService userDetailsService() {
16. InMemoryUserDetailsManager manager = **new** InMemoryUserDetailsManager();
17. manager.createUser(User.withDefaultPasswordEncoder()
18. .username("irfan").password("khan").roles("ADMIN").build());
19. **return** manager;
20. }
22. @Override
23. **protected** **void** configure(HttpSecurity http) **throws** Exception {
25. http
26. .authorizeRequests()
27. .anyRequest().hasRole("ADMIN")
28. .and().formLogin().and()
29. .httpBasic()
30. .and()
31. .logout()
32. .logoutUrl("/j\_spring\_security\_logout")
33. .logoutSuccessUrl("/")
34. ;
35. }
36. }

### **Controller**

**HomeController:** Controller to handle user requests.

1. **package** com.javatpoint.controller;
2. **import** javax.servlet.http.HttpServletRequest;
3. **import** javax.servlet.http.HttpServletResponse;
4. **import** org.springframework.security.core.Authentication;
5. **import** org.springframework.security.core.context.SecurityContextHolder;
6. **import** org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler;
7. **import** org.springframework.stereotype.Controller;
8. **import** org.springframework.web.bind.annotation.RequestMapping;
9. **import** org.springframework.web.bind.annotation.RequestMethod;
10. @Controller
11. **public** **class** HomeController {
12. @RequestMapping(value = "/", method = RequestMethod.GET)
13. **public** String index() {
14. **return** "index";
15. }
17. @RequestMapping(value="/logout", method=RequestMethod.GET)
18. **public** String logoutPage(HttpServletRequest request, HttpServletResponse response) {
19. Authentication auth = SecurityContextHolder.getContext().getAuthentication();
20. **if** (auth != **null**){
21. **new** SecurityContextLogoutHandler().logout(request, response, auth);
22. }
23. **return** "redirect:/";
24. }
25. }

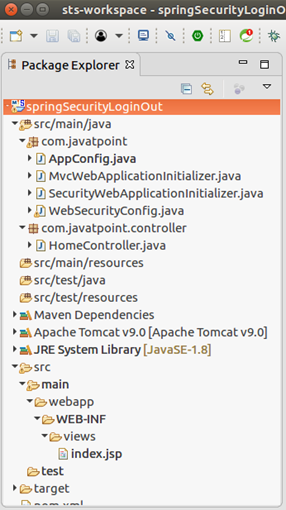
### **Views**

We have a JSP file **index.jsp** that contains the following code.

1. **<**%@ page language="java" contentType="text/html; charset=UTF-8"
2. pageEncoding="UTF-8"%**>**
3. **<**%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%**>**
4. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
5. "http://www.w3.org/TR/html4/loose.dtd"**>**
6. **<html>**
7. **<head>**
8. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8"**>**
9. **<title>**Home**</title>**
10. **</head>**
11. **<body>**
12. **<h3>** Hello ${pageContext.request.userPrincipal.name}, **</h3>**
13. **<h4>**Welcome to Javatpoint! **</h4>**
14. **<a** href="<c:url value='/logout' />"**>**Click here to logout**</a>**
15. **</body>**
16. **</html>**

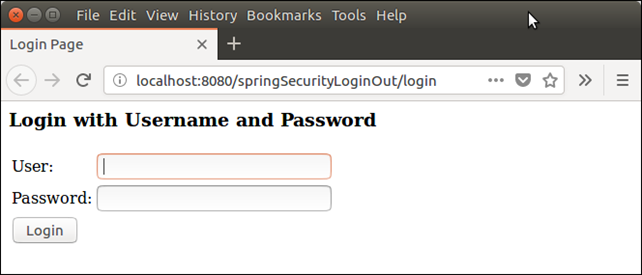
### **Project Structure**

After creating above files, our project structure looks like this:



Output

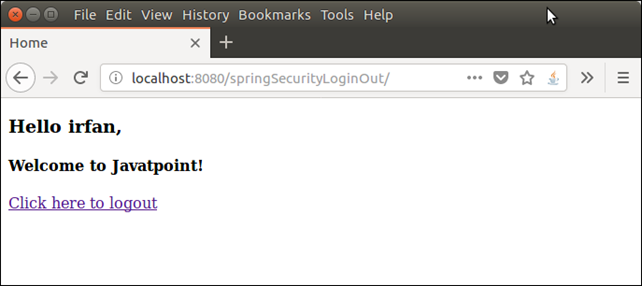
When run using apache tomcat, it produces the following output to the browser.



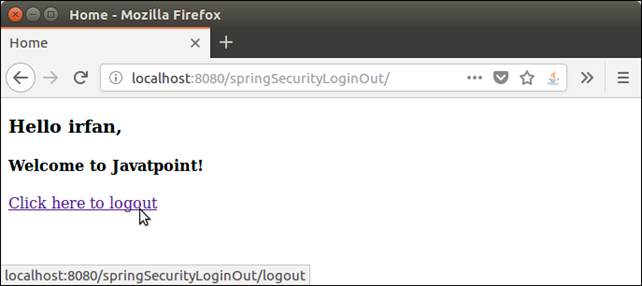
Now, providing user credentials to get logged in.



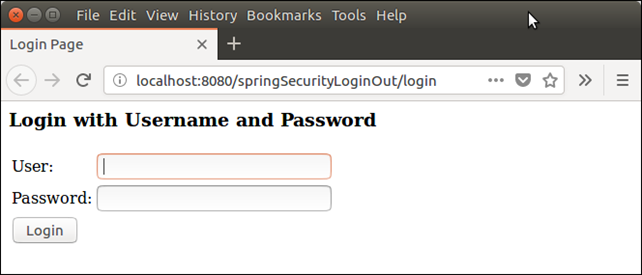
It shows home page after logged in successfully, see below.



Here, we are created a logout link which can be used to get logged out. Let's check out and log out from the application.



And it redirect back to the login page.



Well, we have created a successfully Spring MVC application that uses Spring Security to implement login and logout features.

# Spring Security Custom Login

Spring Security provides it's own built-in login module to authenticate the user. It validates the user credentials and provide accessibility into the application.

The login page rendered by the module is built-in. So, we does not require to create new jsp page. But if we want to customize the login page then how we can?

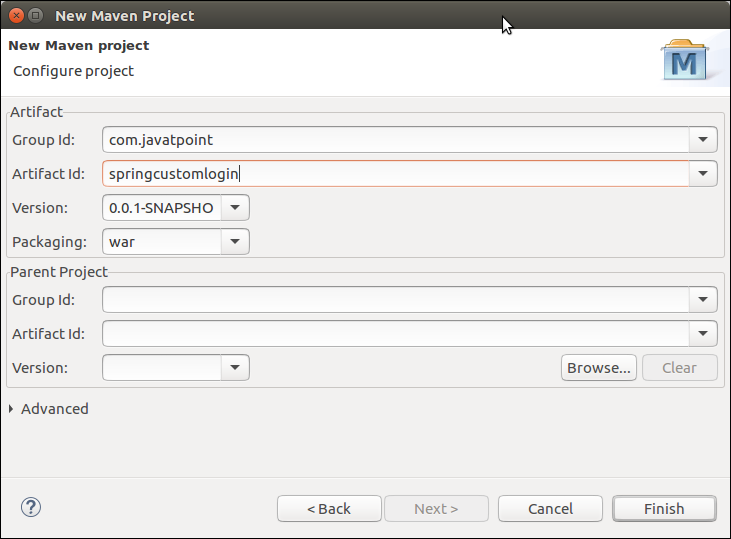
The answer is, we can create our own jsp login page and integrate to the application. In this topic we will create a custom login page and will use it to get login.

See, an example. Create a maven project by providing following details.

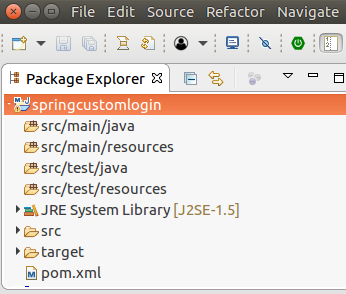
24.1M

501

Java Try Catch



After finishing, it creates the following project structure.



### **Security Configuration**

Configure project to apply spring security. It require following four files. Create a package **com.javatpoint** and put these files into this.

**// AppConfig.java**

1. package com.javatpoint;
3. import org.springframework.context.annotation.Bean;
4. import org.springframework.context.annotation.ComponentScan;
5. import org.springframework.context.annotation.Configuration;
6. import org.springframework.web.servlet.config.annotation.EnableWebMvc;
7. import org.springframework.web.servlet.view.InternalResourceViewResolver;
8. import org.springframework.web.servlet.view.JstlView;
9. @EnableWebMvc
10. @Configuration
11. @ComponentScan({ "com.javatpoint.controller.\*" })
12. public class AppConfig {
13. @Bean
14. public InternalResourceViewResolver viewResolver() {
15. InternalResourceViewResolver viewResolver
16. = new InternalResourceViewResolver();
17. viewResolver.setViewClass(JstlView.class);
18. viewResolver.setPrefix("/WEB-INF/views/");
19. viewResolver.setSuffix(".jsp");
20. return viewResolver;
21. }
22. }

**// MvcWebApplicationInitializer.java**

1. package com.javatpoint;
3. import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
4. public class MvcWebApplicationInitializer extends
5. AbstractAnnotationConfigDispatcherServletInitializer {
6. @Override
7. protected Class**<?>**[] getRootConfigClasses() {
8. return new Class[] { WebSecurityConfig.class };
9. }
10. @Override
11. protected Class**<?>**[] getServletConfigClasses() {
12. // TODO Auto-generated method stub
13. return null;
14. }
15. @Override
16. protected String[] getServletMappings() {
17. return new String[] { "/" };
18. }
19. }

**// SecurityWebApplicationInitializer.java**

1. package com.javatpoint;
2. import org.springframework.security.web.context.\*;
4. public class SecurityWebApplicationInitializer
5. extends AbstractSecurityWebApplicationInitializer {
7. }

**// WebSecurityConfig.java**

1. package com.javatpoint;
3. import org.springframework.context.annotation.\*;
4. //import org.springframework.security.config.annotation.authentication.builders.\*;
5. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
6. import org.springframework.security.config.annotation.web.configuration.\*;
7. import org.springframework.security.core.userdetails.\*;
8. //import org.springframework.security.core.userdetails.UserDetailsService;
9. import org.springframework.security.provisioning.InMemoryUserDetailsManager;
10. import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
11. import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
12. @EnableWebSecurity
13. @ComponentScan("com.javatpoint")
14. public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
16. @Bean
17. public UserDetailsService userDetailsService() {
18. InMemoryUserDetailsManager manager = new InMemoryUserDetailsManager();
19. manager.createUser(User.withDefaultPasswordEncoder()
20. .username("irfan").password("khan123").roles("ADMIN").build());
21. return manager;
22. }
24. @Override
25. protected void configure(HttpSecurity http) throws Exception {
27. http.authorizeRequests().
28. antMatchers("/index", "/user","/").permitAll()
29. .antMatchers("/admin").authenticated()
30. .and()
31. .formLogin()
32. .loginPage("/login")
33. .and()
34. .logout()
35. .logoutRequestMatcher(new AntPathRequestMatcher("/logout"));
36. }
37. }

See, in configure method, after formLogin() a method **loginPage("/login")** is used. It is the actual method that required to call custom login page.

### **View**

First create a login page our own. According to the spring official, the login page should looks like the below.

**// login.jsp**

1. **<**%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %**>**
2. **<c:url** value="/login" var="loginUrl"**/>**
3. **<form** action="${loginUrl}" method="post"**>**
4. **<c:if** test="${param.error != null}"**>**
5. **<p>**
6. Invalid username and password.
7. **</p>**
8. **</c:if>**
9. **<c:if** test="${param.logout != null}"**>**
10. **<p>**
11. You have been logged out.
12. **</p>**
13. **</c:if>**
14. **<p>**
15. **<label** for="username"**>**Username**</label>**
16. **<input** type="text" id="username" name="username"**/>**
17. **</p>**
18. **<p>**
19. **<label** for="password"**>**Password**</label>**
20. **<input** type="password" id="password" name="password"**/>**
21. **</p>**
22. **<input** type="hidden"
23. name="${\_csrf.parameterName}"
24. value="${\_csrf.token}"**/>**
25. **<button** type="submit" class="btn"**>**Log in**</button>**
26. **</form>**

**// index.jsp**

1. **<html>**
2. **<head>**
3. **<title>**Home Page**</title>**
4. **</head>**
5. **<body>**
6. **<h3>** Welcome to Javatpoint! **<br>** **</h3>**
7. **<a** href="admin"**>**Login here**</a>**
8. **</body>**
9. **</html>**

**// admin.jsp**

1. **<html>**
2. **<head>**
3. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8"**>**
4. **<title>**Home Page**</title>**
5. **</head>**
6. **<body>**
7. Login Successful!
8. **<a** href="logout"**>**logout**</a>**
9. **</body>**
10. **</html>**

### **Controller**

Create a controller HomeController inside the com.javatpoint.controller package.

**// HomeController.java**

1. package com.javatpoint.controller;
2. import org.springframework.stereotype.Controller;
3. import org.springframework.web.bind.annotation.RequestMapping;
4. import org.springframework.web.bind.annotation.RequestMethod;
5. @Controller
6. public class HomeController {
8. @RequestMapping(value="/", method=RequestMethod.GET)
9. public String index() {
11. return "index";
12. }
13. @RequestMapping(value="/login", method=RequestMethod.GET)
14. public String login() {
16. return "login";
17. }
18. @RequestMapping(value="/admin", method=RequestMethod.GET)
19. public String admin() {
21. return "admin";
22. }
23. }

### **Project Dependencies**

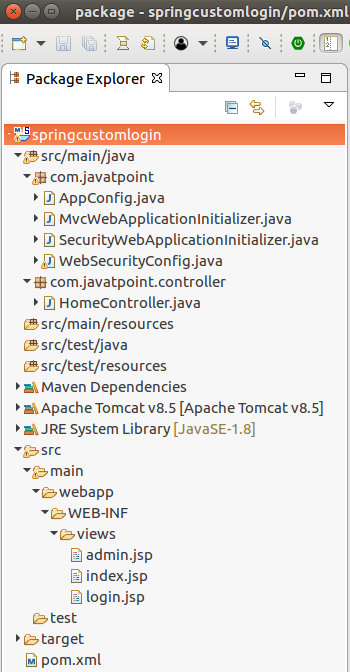
**// pom.xml**

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
2. **<modelVersion>**4.0.0**</modelVersion>**
3. **<groupId>**com.javatpoint**</groupId>**
4. **<artifactId>**springcustomlogin**</artifactId>**
5. **<version>**0.0.1-SNAPSHOT**</version>**
6. **<packaging>**war**</packaging>**
7. **<properties>**
8. **<maven.compiler.target>**1.8**</maven.compiler.target>**
9. **<maven.compiler.source>**1.8**</maven.compiler.source>**
10. **</properties>**
11. **<dependencies>**
12. **<dependency>**
13. **<groupId>**org.springframework**</groupId>**
14. **<artifactId>**spring-webmvc**</artifactId>**
15. **<version>**5.0.2.RELEASE**</version>**
16. **</dependency>**
17. **<dependency>**
18. **<groupId>**org.springframework.security**</groupId>**
19. **<artifactId>**spring-security-web**</artifactId>**
20. **<version>**5.0.0.RELEASE**</version>**
21. **</dependency>**
22. **<dependency>**
23. **<groupId>**org.springframework.security**</groupId>**
24. **<artifactId>**spring-security-core**</artifactId>**
25. **<version>**5.0.4.RELEASE**</version>**
26. **</dependency>**
27. <!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-config -->
28. **<dependency>**
29. **<groupId>**org.springframework.security**</groupId>**
30. **<artifactId>**spring-security-config**</artifactId>**
31. **<version>**5.0.4.RELEASE**</version>**
32. **</dependency>**

35. <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
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. **<dependency>**
43. **<groupId>**javax.servlet**</groupId>**
44. **<artifactId>**jstl**</artifactId>**
45. **<version>**1.2**</version>**
46. **</dependency>**
47. **</dependencies>**
48. **<build>**
49. **<plugins>**
50. **<plugin>**
51. **<groupId>**org.apache.maven.plugins**</groupId>**
52. **<artifactId>**maven-war-plugin**</artifactId>**
53. **<version>**2.6**</version>**
54. **<configuration>**
55. **<failOnMissingWebXml>**false**</failOnMissingWebXml>**
56. **</configuration>**
57. **</plugin>**
58. **</plugins>**
59. **</build>**
60. **</project>**

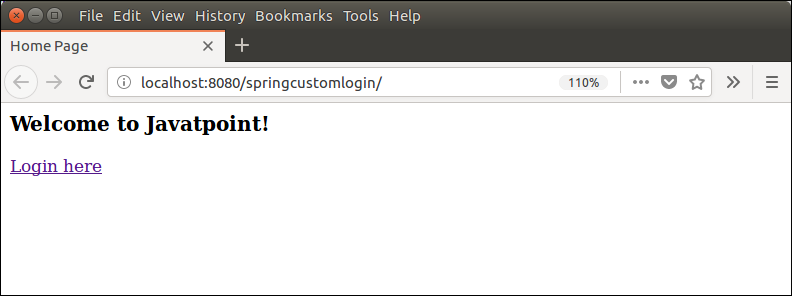
### **Project Structure**

Our project looks like the this:

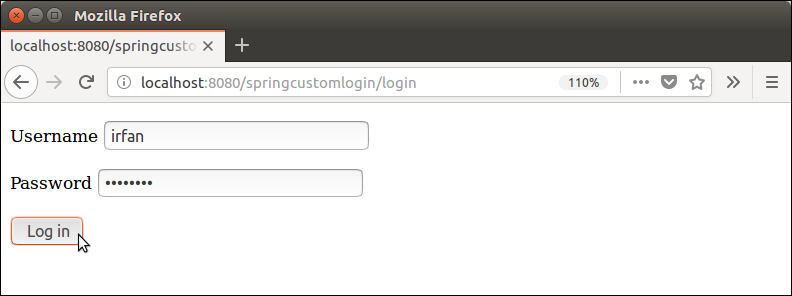
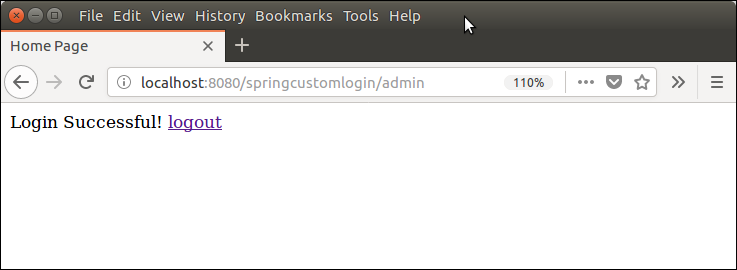


### **Run the Server**

**Output:**



Now, login by providing user credentials.

See, it's working fine. Now, we can create it more decorative and custom according to the need.

# Spring Security Form-Based Authentication

Form-Based authentication is a way in which user's authentication is done by login form. This form is built-in and provided by spring security framework.

The HttpSecurity class provide a method formLogin() which is responsible to render login form and validate user credentials.

In this tutorial, we will create an example that implements form-based authentication. Lets start the example.

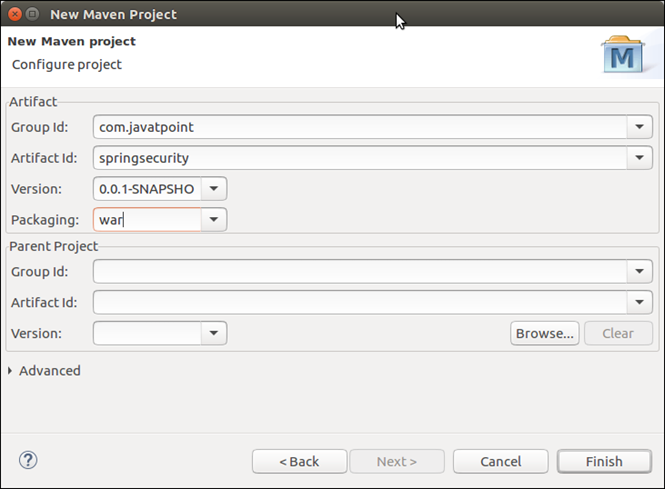
### **Create a Maven Project**

First create a maven project by providing project details.

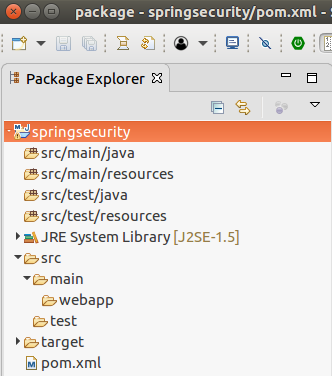
30.3M

681

Exception Handling in Java - Javatpoint



This project initially looks like this:



### **Spring Security Configuration**

Configure spring security in the application by using the following Java files. Create a package **com.javatpoint** and put all the files into it.

**// AppConfig.java**

1. package com.javatpoint;
2. import org.springframework.context.annotation.Bean;
3. import org.springframework.context.annotation.ComponentScan;
4. import org.springframework.context.annotation.Configuration;
5. import org.springframework.web.servlet.config.annotation.EnableWebMvc;
6. import org.springframework.web.servlet.view.InternalResourceViewResolver;
7. import org.springframework.web.servlet.view.JstlView;
8. @EnableWebMvc
9. @Configuration
10. @ComponentScan({ "com.javatpoint.controller.\*" })
11. public class AppConfig {
12. @Bean
13. public InternalResourceViewResolver viewResolver() {
14. InternalResourceViewResolver viewResolver
15. = new InternalResourceViewResolver();
16. //viewResolver.setViewClass(JstlView.class);
17. viewResolver.setPrefix("/WEB-INF/views/");
18. viewResolver.setSuffix(".jsp");
19. return viewResolver;
20. }
21. }

**// MvcWebApplicationInitializer.java**

1. package com.javatpoint;
2. import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
3. public class MvcWebApplicationInitializer extends
4. AbstractAnnotationConfigDispatcherServletInitializer {
5. @Override
6. protected Class**<?>**[] getRootConfigClasses() {
7. return new Class[] { WebSecurityConfig.class };
8. }
9. @Override
10. protected Class**<?>**[] getServletConfigClasses() {
11. // TODO Auto-generated method stub
12. return null;
13. }
14. @Override
15. protected String[] getServletMappings() {
16. return new String[] { "/" };
17. }
18. }

**// SecurityWebApplicationInitializer.java**

1. package com.javatpoint;
2. import org.springframework.security.web.context.\*;
3. public class SecurityWebApplicationInitializer extends AbstractSecurityWebApplicationInitializer {
4. }

**// WebSecuiryConfig.java**

1. package com.javatpoint;
2. import org.springframework.context.annotation.\*;
3. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
4. import org.springframework.security.config.annotation.web.configuration.\*;
5. import org.springframework.security.core.userdetails.\*;
6. import org.springframework.security.provisioning.InMemoryUserDetailsManager;
7. import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
8. @EnableWebSecurity
9. @ComponentScan("com.javatpoint")
10. public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
11. @Bean
12. public UserDetailsService userDetailsService() {
13. InMemoryUserDetailsManager manager = new InMemoryUserDetailsManager();
14. manager.createUser(User.withDefaultPasswordEncoder()
15. .username("admin").password("admin123").roles("ADMIN").build());
16. return manager;
17. }
18. @Override
19. protected void configure(HttpSecurity http) throws Exception {
20. http.authorizeRequests().
21. antMatchers("/index", "/user","/").permitAll()
22. .antMatchers("/admin").authenticated()
23. .and()
24. .formLogin() // It renders a login form
25. .and()
26. .logout()
27. .logoutRequestMatcher(new AntPathRequestMatcher("/logout"));
28. }
29. }

### **Controller**

Create a controller HomeController and put inside the **com.javatpoint.controller** package. It contains the following code.

**// HomeController.java**

1. package com.javatpoint.controller;
2. import org.springframework.stereotype.Controller;
3. import org.springframework.web.bind.annotation.RequestMapping;
4. import org.springframework.web.bind.annotation.RequestMethod;
6. @Controller
7. public class HomeController {
9. @RequestMapping(value="/", method=RequestMethod.GET)
10. public String index() {
12. return "index";
13. }
14. @RequestMapping(value="/admin", method=RequestMethod.GET)
15. public String admin() {
17. return "admin";
18. }
19. }

### **Views**

This project contains the following two view (JSP pages). Put these into **WEB-INF/views** folder.

**// index.jsp**

1. **<html>**
2. **<head>**
3. **<title>**Index Page**</title>**
4. **</head>**
5. **<body>**
6. Welcome to Javatpoint! **<br>** **<br>**
7. **<a** href="admin"**>**Admin login**</a>**
8. **</body>**
9. **</html>**

**// admin.jsp**

1. **<html>**
2. **<head>**
3. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8"**>**
4. **<title>**Home Page**</title>**
5. **</head>**
6. **<body>**
7. **<span** style="color: green;"**>**login successful!**</span>**
8. **<a** href="logout"**>**Logout**</a>**
9. **<hr>**
10. **<h3>**Welcome Admin**</h3>**
11. **</body>**
12. **</html>**

### **Project Dependencies**

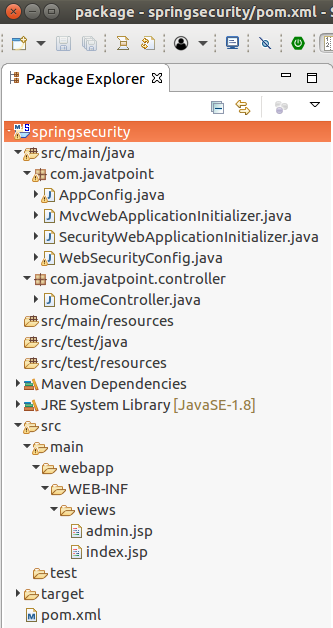
**// pom.xml**

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
2. **<modelVersion>**4.0.0**</modelVersion>**
3. **<groupId>**com.javatpoint**</groupId>**
4. **<artifactId>**springsecurity**</artifactId>**
5. **<version>**0.0.1-SNAPSHOT**</version>**
6. **<packaging>**war**</packaging>**
7. **<properties>**
8. **<maven.compiler.target>**1.8**</maven.compiler.target>**
9. **<maven.compiler.source>**1.8**</maven.compiler.source>**
10. **</properties>**
11. **<dependencies>**
12. **<dependency>**
13. **<groupId>**org.springframework**</groupId>**
14. **<artifactId>**spring-webmvc**</artifactId>**
15. **<version>**5.0.2.RELEASE**</version>**
16. **</dependency>**
17. **<dependency>**
18. **<groupId>**org.springframework.security**</groupId>**
19. **<artifactId>**spring-security-web**</artifactId>**
20. **<version>**5.0.0.RELEASE**</version>**
21. **</dependency>**
22. **<dependency>**
23. **<groupId>**org.springframework.security**</groupId>**
24. **<artifactId>**spring-security-core**</artifactId>**
25. **<version>**5.0.4.RELEASE**</version>**
26. **</dependency>**
27. <!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-config -->
28. **<dependency>**
29. **<groupId>**org.springframework.security**</groupId>**
30. **<artifactId>**spring-security-config**</artifactId>**
31. **<version>**5.0.4.RELEASE**</version>**
32. **</dependency>**

35. <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
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. **<dependency>**
43. **<groupId>**javax.servlet**</groupId>**
44. **<artifactId>**jstl**</artifactId>**
45. **<version>**1.2**</version>**
46. **</dependency>**
47. **</dependencies>**
48. **<build>**
49. **<plugins>**
50. **<plugin>**
51. **<groupId>**org.apache.maven.plugins**</groupId>**
52. **<artifactId>**maven-war-plugin**</artifactId>**
53. **<version>**2.6**</version>**
54. **<configuration>**
55. **<failOnMissingWebXml>**false**</failOnMissingWebXml>**
56. **</configuration>**
57. **</plugin>**
58. **</plugins>**
59. **</build>**
60. **</project>**

### **Project Structure**

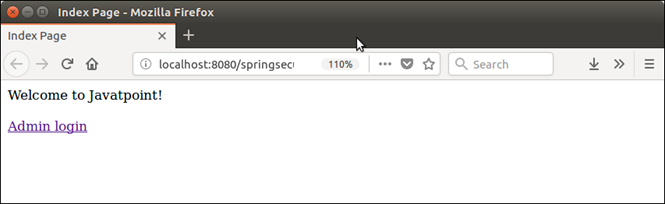
After adding all these files the project structure looks like this:



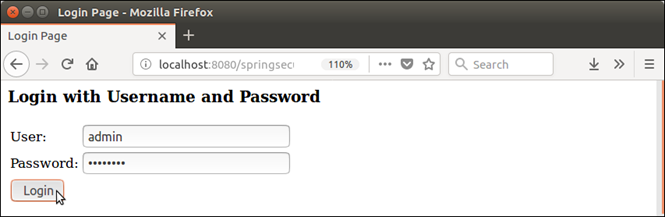
### **Run Server**

Run the application over the server and see it produces the following output to the browser.

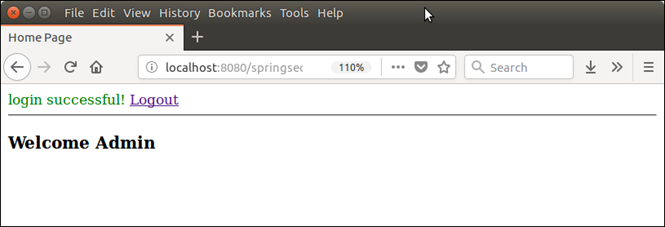
**Output:**



Click on link, a login form is rendered that will use for form-based authentication.



After validating credentials it authenticate the user and render to the admin page.



# Spring Security Remember Me

Remember me is a feature that allows a user to access into application without re-login. User's login session terminates after closing the browser and if user again access the application by opening browser, it prompts for login.

But we can avoid this re-login by using remember me feature. It stores user's identity into the Cookie or database and use to identity the user.

We are implementing this into the following example. Lets see an example.

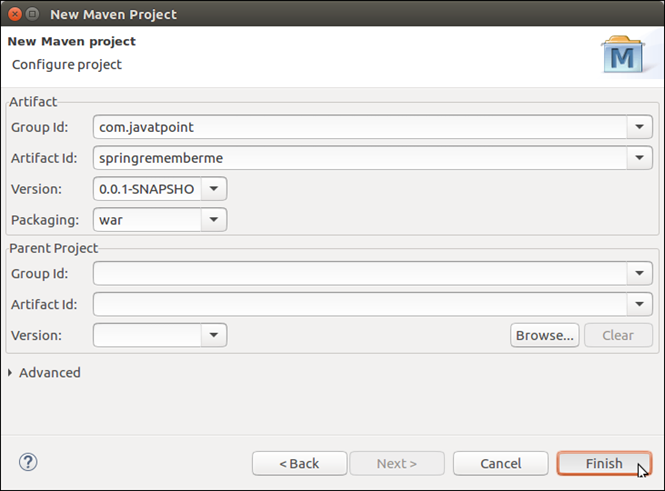
### **Create a Maven Project**

First create a maven project and provide the project details.

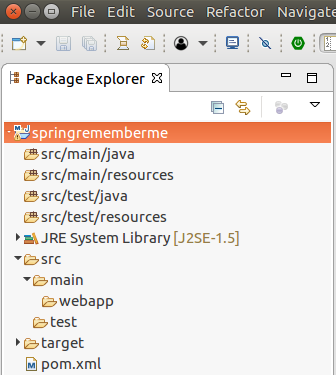
20.5M

390

Prime Ministers of India | List of Prime Minister of India (1947-2020)



Initially, project looks like this:



### **Spring Security Configuration**

Configure the project to implement spring security. It requires following four Java files. First create a package **com.javatpoint** and put all the files into this.

**// AppConfig.java**

1. package com.javatpoint;
2. import org.springframework.context.annotation.Bean;
3. import org.springframework.context.annotation.ComponentScan;
4. import org.springframework.context.annotation.Configuration;
5. import org.springframework.web.servlet.config.annotation.EnableWebMvc;
6. import org.springframework.web.servlet.view.InternalResourceViewResolver;
7. import org.springframework.web.servlet.view.JstlView;
8. @EnableWebMvc
9. @Configuration
10. @ComponentScan({ "com.javatpoint.controller.\*" })
11. public class AppConfig {
12. @Bean
13. public InternalResourceViewResolver viewResolver() {
14. InternalResourceViewResolver viewResolver
15. = new InternalResourceViewResolver();
16. viewResolver.setViewClass(JstlView.class);
17. viewResolver.setPrefix("/WEB-INF/views/");
18. viewResolver.setSuffix(".jsp");
19. return viewResolver;
20. }
21. }

**// MvcWebApplicationInitializer.java**

1. package com.javatpoint;
2. import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
3. public class MvcWebApplicationInitializer extends
4. AbstractAnnotationConfigDispatcherServletInitializer {
5. @Override
6. protected Class**<?>**[] getRootConfigClasses() {
7. return new Class[] { WebSecurityConfig.class };
8. }
9. @Override
10. protected Class**<?>**[] getServletConfigClasses() {
11. // TODO Auto-generated method stub
12. return null;
13. }
14. @Override
15. protected String[] getServletMappings() {
16. return new String[] { "/" };
17. }
18. }

**// SecurityWebApplicationInitializer.java**

1. package com.javatpoint;
2. import org.springframework.security.web.context.\*;
3. public class SecurityWebApplicationInitializer
4. extends AbstractSecurityWebApplicationInitializer {
5. }

**// WebSecurityConfig.java**

In this class, we are creating user and authenticating as well. The rememberMe() method inside the configure() method is responsible to remember and store user identity.

1. package com.javatpoint;
2. import org.springframework.context.annotation.\*;
3. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
4. import org.springframework.security.config.annotation.web.configuration.\*;
5. import org.springframework.security.core.userdetails.\*;
6. import org.springframework.security.provisioning.InMemoryUserDetailsManager;
7. import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
8. @EnableWebSecurity
9. @ComponentScan("com.javatpoint")
10. public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
11. @Bean
12. public UserDetailsService userDetailsService() {
13. InMemoryUserDetailsManager manager = new InMemoryUserDetailsManager();
14. manager.createUser(User.withDefaultPasswordEncoder()
15. .username("admin").password("admin123").roles("ADMIN").build());
16. return manager;
17. }
19. @Override
20. protected void configure(HttpSecurity http) throws Exception {
22. http.authorizeRequests().
23. antMatchers("/index", "/user","/").permitAll()
24. .antMatchers("/admin").authenticated()
25. .and()
26. .formLogin()
27. .loginPage("/login")
28. .and()
29. .rememberMe()
30. .key("rem-me-key")
31. .rememberMeParameter("remember") // it is name of checkbox at login page
32. .rememberMeCookieName("rememberlogin") // it is name of the cookie
33. .tokenValiditySeconds(100) // remember for number of seconds
34. .and()
35. .logout()
36. .logoutRequestMatcher(new AntPathRequestMatcher("/logout"));
37. }
38. }

### **Controller**

Create a controller HomeController inside the **com.javatpoint.controller** package. See the controller code.

**// HomeController.java**

1. package com.javatpoint.controller;
2. import org.springframework.stereotype.Controller;
3. import org.springframework.web.bind.annotation.RequestMapping;
4. import org.springframework.web.bind.annotation.RequestMethod;
5. @Controller
6. public class HomeController {
7. @RequestMapping(value = "/", method = RequestMethod.GET)
8. public String index() {
9. return "index";
10. }
11. @RequestMapping(value = "/login", method = RequestMethod.GET)
12. public String login() {
13. return "login";
14. }
15. @RequestMapping(value = "/admin", method = RequestMethod.GET)
16. public String admin() {
17. return "admin";
18. }
19. }

### **View**

Create view (JSP pages) to produce output to the browser.

**// index.jsp**

1. **<html>**
2. **<head>**
3. **<title>**Home Page**</title>**
4. **</head>**
5. **<body>**
6. Welcome to Javatpoint! **<br>** **<br>**
7. **<a** href="admin"**>**Admin login**</a>**
8. **</body>**
9. **</html>**

**// admin.jsp**

1. **<html>**
2. **<head>**
3. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8"**>**
4. **<title>**Home Page**</title>**
5. **</head>**
6. **<body>**
7. Welcome Admin! ?
8. **<a** href="logout"**>**logout**</a>**
9. **</body>**
10. **</html>**

**// login.jsp**

This is our custom login page in which we added remember me check box. See the code.

1. **<**%@ taglib
2. prefix="c"
3. uri="http://java.sun.com/jsp/jstl/core"
4. %**>**
5. **<c:url** value="/login" var="loginUrl"**/>**
6. **<form** action="${loginUrl}" method="post"**>**
7. **<c:if** test="${param.error != null}"**>**
8. **<p>**
9. Invalid username and password.
10. **</p>**
11. **</c:if>**
12. **<c:if** test="${param.logout != null}"**>**
13. **<p>**
14. You have been logged out.
15. **</p>**
16. **</c:if>**
17. **<p>**
18. **<label** for="username"**>**Username**</label>**
19. **<input** type="text" id="username" name="username"**/>**
20. **</p>**
21. **<p>**
22. **<label** for="password"**>**Password**</label>**
23. **<input** type="password" id="password" name="password"**/>**
24. **</p>**
25. **<p>**
26. **<label** for="remember"**>** Remember me**</label>**
27. **<input** type="checkbox" name="remember" **/>**
28. **</p>**
29. **<input** type="hidden"
30. name="${\_csrf.parameterName}"
31. value="${\_csrf.token}"**/>**
32. **<button** type="submit" class="btn"**>**Log in**</button>**
33. **</form>**

### **Project Dependencies**

Following is our pom.xml file that contains all required dependencies.

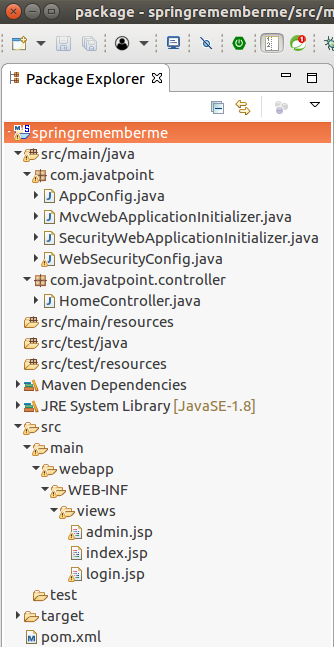
**// pom.xml**

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
2. **<modelVersion>**4.0.0**</modelVersion>**
3. **<groupId>**com.javatpoint**</groupId>**
4. **<artifactId>**springrememberme**</artifactId>**
5. **<version>**0.0.1-SNAPSHOT**</version>**
6. **<packaging>**war**</packaging>**
7. **<properties>**
8. **<maven.compiler.target>**1.8**</maven.compiler.target>**
9. **<maven.compiler.source>**1.8**</maven.compiler.source>**
10. **</properties>**
11. **<dependencies>**
12. **<dependency>**
13. **<groupId>**org.springframework**</groupId>**
14. **<artifactId>**spring-webmvc**</artifactId>**
15. **<version>**5.0.2.RELEASE**</version>**
16. **</dependency>**
17. **<dependency>**
18. **<groupId>**org.springframework.security**</groupId>**
19. **<artifactId>**spring-security-web**</artifactId>**
20. **<version>**5.0.0.RELEASE**</version>**
21. **</dependency>**
22. **<dependency>**
23. **<groupId>**org.springframework.security**</groupId>**
24. **<artifactId>**spring-security-core**</artifactId>**
25. **<version>**5.0.4.RELEASE**</version>**
26. **</dependency>**
27. <!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-config -->
28. **<dependency>**
29. **<groupId>**org.springframework.security**</groupId>**
30. **<artifactId>**spring-security-config**</artifactId>**
31. **<version>**5.0.4.RELEASE**</version>**
32. **</dependency>**

35. <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
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. **<dependency>**
43. **<groupId>**javax.servlet**</groupId>**
44. **<artifactId>**jstl**</artifactId>**
45. **<version>**1.2**</version>**
46. **</dependency>**
47. **</dependencies>**
48. **<build>**
49. **<plugins>**
50. **<plugin>**
51. **<groupId>**org.apache.maven.plugins**</groupId>**
52. **<artifactId>**maven-war-plugin**</artifactId>**
53. **<version>**2.6**</version>**
54. **<configuration>**
55. **<failOnMissingWebXml>**false**</failOnMissingWebXml>**
56. **</configuration>**
57. **</plugin>**
58. **</plugins>**
59. **</build>**
60. **</project>**

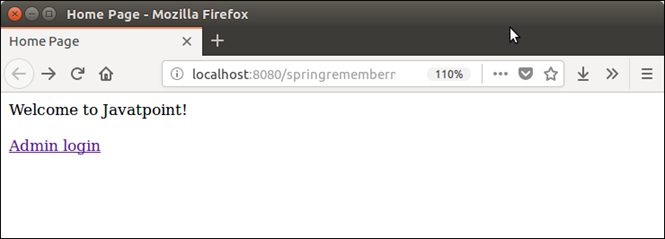
### **Project Structure**

After adding all the files the project structure looks like this:

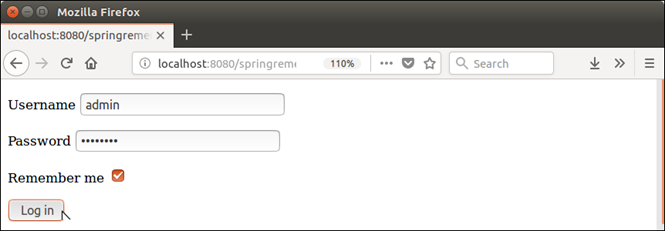


### **Run Server**

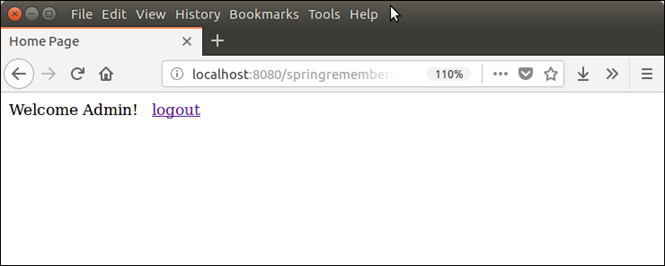
**Output:**



Click on Admin login link and login.



See, we have **clicked on remember me** check box.



Copy the URL: **http://localhost:8080/springrememberme/admin** and close the browser completely. After a second open browser and paste the copied URL.

See, it does not ask for login and land us on the same page. Because we did check remember me button during login.

# Spring Security at Method Level

Apart from authentication, spring security also check authorization of the logged in user. After login which user is authorize to access the resource is done on the bases of user's ROLE.

At the time of creating user in WebSecurityConfig class, we can specify user?s ROLE as well.

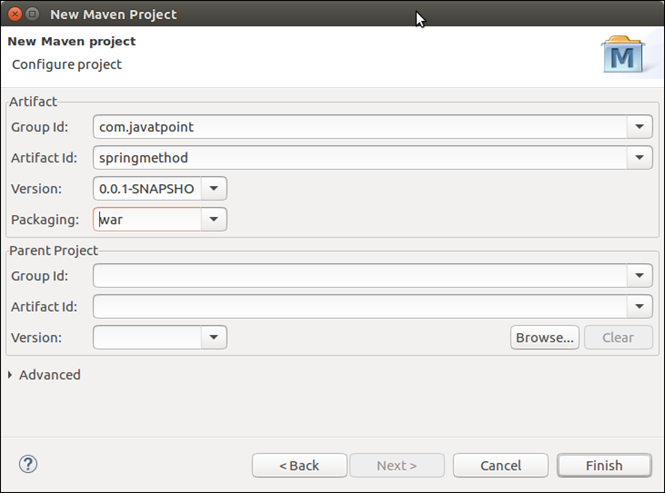
Security applied on a method restricts to unauthorized user and allow only authentic user.

Let's see an example. First create a maven project by providing details.

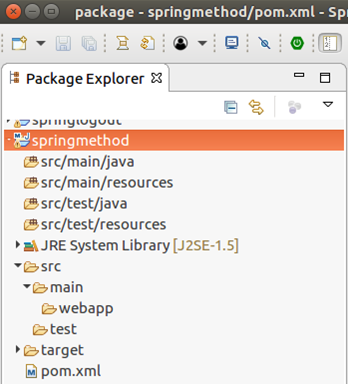
20.5M

390

Prime Ministers of India | List of Prime Minister of India (1947-2020)



This project initially look like this:



### **Spring Security Configuration**

Now, configure the application to protect from unauthorized and unauthentic users. It requires four Java files that are given below, create a package com.javatpoint and place all these files inside it.

**// AppConfig.java**

This class is used to set view suffix and prefix with the help of view resolver.

1. package com.javatpoint;
2. import org.springframework.context.annotation.Bean;
3. import org.springframework.context.annotation.ComponentScan;
4. import org.springframework.context.annotation.Configuration;
5. import org.springframework.web.servlet.config.annotation.EnableWebMvc;
6. import org.springframework.web.servlet.view.InternalResourceViewResolver;
7. import org.springframework.web.servlet.view.JstlView;
8. @EnableWebMvc
9. @Configuration
10. @ComponentScan({ "com.javatpoint.controller.\*" })
11. public class AppConfig {
12. @Bean
13. public InternalResourceViewResolver viewResolver() {
14. InternalResourceViewResolver viewResolver
15. = new InternalResourceViewResolver();
16. viewResolver.setViewClass(JstlView.class);
17. viewResolver.setPrefix("/WEB-INF/views/");
18. viewResolver.setSuffix(".jsp");
19. return viewResolver;
20. }
21. }

**// MvcWebApplicationInitializer.java.java**

1. package com.javatpoint;
2. import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
3. public class MvcWebApplicationInitializer extends
4. AbstractAnnotationConfigDispatcherServletInitializer {
5. @Override
6. protected Class**<?>**[] getRootConfigClasses() {
7. return new Class[] { WebSecurityConfig.class };
8. }
9. @Override
10. protected Class**<?>**[] getServletConfigClasses() {
11. // TODO Auto-generated method stub
12. return null;
13. }
14. @Override
15. protected String[] getServletMappings() {
16. return new String[] { "/" };
17. }
18. }

**// SecurityWebApplicationInitializer.java**

1. package com.javatpoint;
2. import org.springframework.security.web.context.\*;
3. public class SecurityWebApplicationInitializer
4. extends AbstractSecurityWebApplicationInitializer {
5. }

**// WebSecurityConfig.java**

This class is used to create user and set their authentication. Each time login is required when user want to access into the application.

1. package com.javatpoint;
2. import org.springframework.context.annotation.\*;
3. import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
4. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
5. import org.springframework.security.config.annotation.web.configuration.\*;
6. import org.springframework.security.core.userdetails.\*;
7. import org.springframework.security.core.userdetails.User.UserBuilder;
8. import org.springframework.security.provisioning.InMemoryUserDetailsManager;
9. import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
10. @EnableWebSecurity
11. @ComponentScan("com.javatpoint")
12. @EnableGlobalMethodSecurity(prePostEnabled=true)
13. public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
14. @Bean
15. public UserDetailsService userDetailsService() {
16. // ensure the passwords are encoded properly
17. UserBuilder users = User.withDefaultPasswordEncoder();
18. InMemoryUserDetailsManager manager = new InMemoryUserDetailsManager();
19. manager.createUser(users.username("irfan").password("user123").roles("USER").build());
20. manager.createUser(users.username("admin").password("admin123").roles("ADMIN").build());
21. return manager;
22. }
23. @Override
24. protected void configure(HttpSecurity http) throws Exception {
25. http.authorizeRequests().
26. antMatchers("/index","/").permitAll()
27. .antMatchers("/admin","/user").authenticated()
28. .and()
29. .formLogin()
30. .and()
31. .logout()
32. .logoutRequestMatcher(new AntPathRequestMatcher("/logout"));
33. }
34. }

### **Controller**

Create a controller HomeController and place into **com.javatpoint.controller** package.

**// HomeController.java**

1. package com.javatpoint.controller;
2. import org.springframework.security.access.prepost.PreAuthorize;
3. import org.springframework.stereotype.Controller;
4. import org.springframework.web.bind.annotation.RequestMapping;
5. import org.springframework.web.bind.annotation.RequestMethod;
6. import org.springframework.web.bind.annotation.ResponseBody;
7. @Controller
8. public class HomeController {
9. @RequestMapping(value="/", method=RequestMethod.GET)
10. public String index() {
11. return "index";
12. }
13. @RequestMapping(value="/user", method=RequestMethod.GET)
14. public String user() {
15. return "admin";
16. }
17. @RequestMapping(value="/admin", method=RequestMethod.GET)
18. public String admin() {
19. return "admin";
20. }
21. // Only, a person having ADMIN role can access this method.
22. @RequestMapping(value="/update", method=RequestMethod.GET)
23. @ResponseBody
24. @PreAuthorize("hasRole('ROLE\_ADMIN')")
25. public String update() {
26. return "record updated ";
27. }
28. }

### **View**

Create following views (JSP pages) to produce output for the user. Place all the views into **WEB-INF/views** folder.

**// index.jsp**

1. **<html>**
2. **<head>**
3. **<title>**Home Page**</title>**
4. **</head>**
5. **<body>**
6. Welcome to Javatpoint! **<br>** **<br>**
7. Login as:
8. **<a** href="admin"**>**Admin**</a>** **<a** href="user"**>**User**</a>**
9. **</body>**
10. **</html>**

**// admin.jsp**

1. **<html>**
2. **<head>**
3. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8"**>**
4. **<title>**Home Page**</title>**
5. **</head>**
6. **<body>**
7. **<span** style="color: green"**>**Login Successful!**</span>** ? **<a** href="logout" style="text-decoration: none;"**>**logout**</a>**  **<br>** **<br>**
8. **<a** href="update" style="text-decoration: none;"**>**Update Record**</a>**
9. **</body>**
10. **</html>**

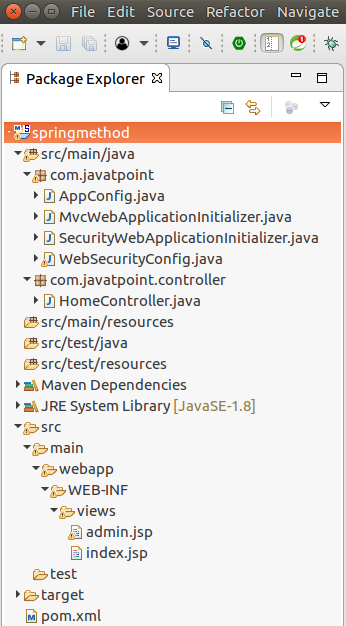
### **Package Dependencies**

Following are the dependencies that are required to create this project.

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
2. **<modelVersion>**4.0.0**</modelVersion>**
3. **<groupId>**com.javatpoint**</groupId>**
4. **<artifactId>**springmethod**</artifactId>**
5. **<version>**0.0.1-SNAPSHOT**</version>**
6. **<packaging>**war**</packaging>**
7. **<properties>**
8. **<maven.compiler.target>**1.8**</maven.compiler.target>**
9. **<maven.compiler.source>**1.8**</maven.compiler.source>**
10. **</properties>**
11. **<dependencies>**
12. **<dependency>**
13. **<groupId>**org.springframework**</groupId>**
14. **<artifactId>**spring-webmvc**</artifactId>**
15. **<version>**5.0.2.RELEASE**</version>**
16. **</dependency>**
17. **<dependency>**
18. **<groupId>**org.springframework.security**</groupId>**
19. **<artifactId>**spring-security-web**</artifactId>**
20. **<version>**5.0.0.RELEASE**</version>**
21. **</dependency>**
22. **<dependency>**
23. **<groupId>**org.springframework.security**</groupId>**
24. **<artifactId>**spring-security-core**</artifactId>**
25. **<version>**5.0.4.RELEASE**</version>**
26. **</dependency>**
27. <!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-config -->
28. **<dependency>**
29. **<groupId>**org.springframework.security**</groupId>**
30. **<artifactId>**spring-security-config**</artifactId>**
31. **<version>**5.0.4.RELEASE**</version>**
32. **</dependency>**
33. <!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->
34. <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
35. **<dependency>**
36. **<groupId>**javax.servlet**</groupId>**
37. **<artifactId>**javax.servlet-api**</artifactId>**
38. **<version>**3.1.0**</version>**
39. **<scope>**provided**</scope>**
40. **</dependency>**
41. **<dependency>**
42. **<groupId>**javax.servlet**</groupId>**
43. **<artifactId>**jstl**</artifactId>**
44. **<version>**1.2**</version>**
45. **</dependency>**
46. <!-- https://mvnrepository.com/artifact/org.springframework/spring-framework-bom -->
47. **</dependencies>**
48. **<build>**
49. **<plugins>**
50. **<plugin>**
51. **<groupId>**org.apache.maven.plugins**</groupId>**
52. **<artifactId>**maven-war-plugin**</artifactId>**
53. **<version>**2.6**</version>**
54. **<configuration>**
55. **<failOnMissingWebXml>**false**</failOnMissingWebXml>**
56. **</configuration>**
57. **</plugin>**
58. **</plugins>**
59. **</build>**
60. **</project>**

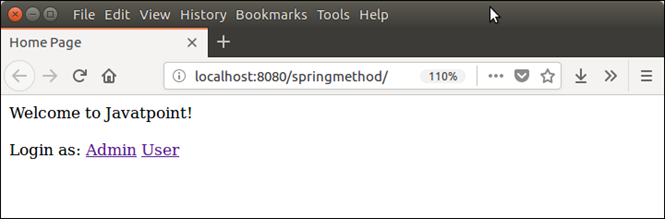
### **Project Structure**

After adding all the above files, our project looks like this:

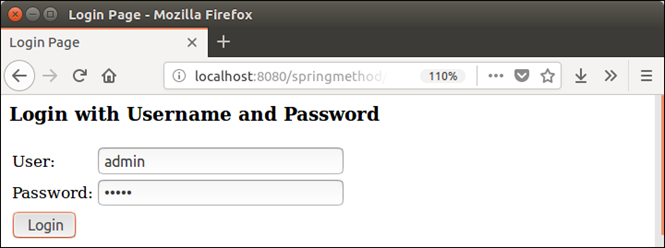


### **Run Server**

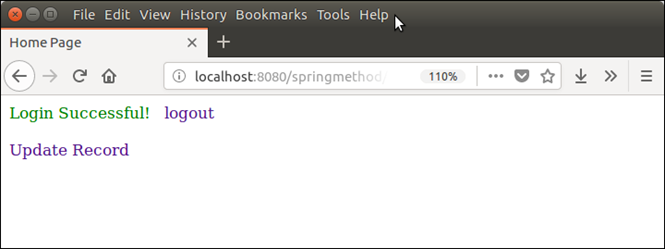
**Output:**



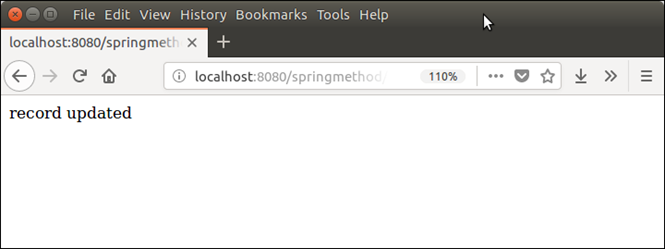
First login as ADMIN



After login,

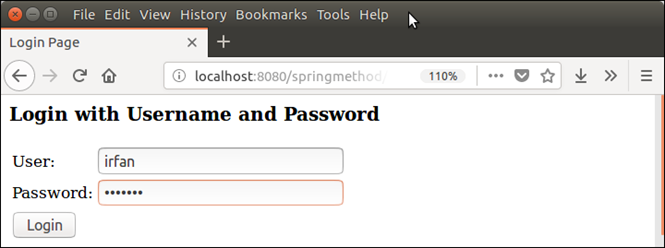
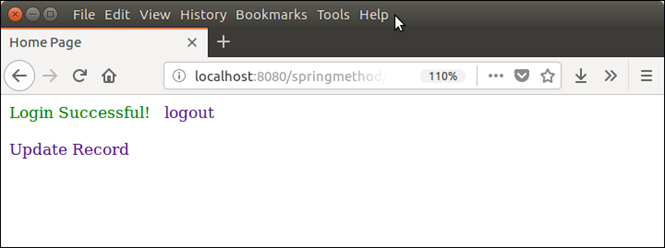


Click on **update record** and see, record is updated because the user's role is ADMIN.

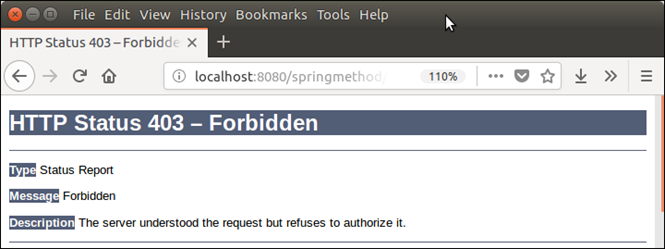


### **User Login**

Now, login as user.

Now, click on **update record** and see server decline the access because the user's role is USER.



# Spring Security JSP Tag Library

Spring Security provides its own tags for jsp pages. These tags are used to access security information and apply security constraints in JSPs.

The following tags are used to secure view layer of the application.

* Authorize Tag
* Authentication Tag
* Accesscontrollist Tag
* Csrfinput Tag
* CsrfMetaTags Tag

### **Authorize Tag**

This tag is used for authorization purpose. This tag evaluates and check whether the request is authorized or not.

It uses two attributes **access** and **URL** to check request authorization. We can pass user's role to evaluate this tag.

20.5M

390

Prime Ministers of India | List of Prime Minister of India (1947-2020)

The content written inside this tag will display only if the attribute is satisfy. For example.

1. **<sec:authorize** access="hasRole('ADMIN')"**>**
2. It will display only is user is admin
3. **</sec:authorize>**

### **Authentication Tag**

This tag is used to access the authentication stored into the security context. It can be used to get current user details if the Authentication is an instance of UserDetails object. For example.

1. **<sec:authentication** property="principal.username"**>**

### **Accesscontrollist Tag**

This tag is used with Spring Security's ACL module. It checks list of required permissions for the specified domains. It executes only if current user has all the permissions. For example.

1. **<sec:accesscontrollist** hasPermission="1,2" domainObject="${someObject}"**>**
2. If user has all the permissions represented by the values "1" or "2" on the given object.
3. **</sec:accesscontrollist>**

### **CsrfInput Tag**

This tag is used to create CSRF tokens for the HTML form. To use it make sure CSRF protection is enabled. We should place this tag inside the <form></form> tag to create CSRF token. For example.

1. **<form** method="post" action="/some/action"**>**
2. **<sec:csrfInput** **/>**
3. Name:**<br** **/>**
4. **<input** type="text" name="username" **/>**
5. ...
6. **</form>**

### **CsrfMetaTags Tag**

It inserts meta tag that contains CSRF token, form field, header name and CSRF token value. These values are useful to set CSRF token within JavaScript in the application.

This tag should place inside the HTML <head> tag.

### **Spring Security Taglib JAR**

To implement any of these tags, we must have spring security taglib jar in our application. It can also be added using following maven dependecy.

1. **<dependency>**
2. **<groupId>**org.springframework.security**</groupId>**
3. **<artifactId>**spring-security-taglibs**</artifactId>**
4. **<version>**5.0.4.RELEASE**</version>**
5. **</dependency>**

### **Spring Security Taglib Declaration**

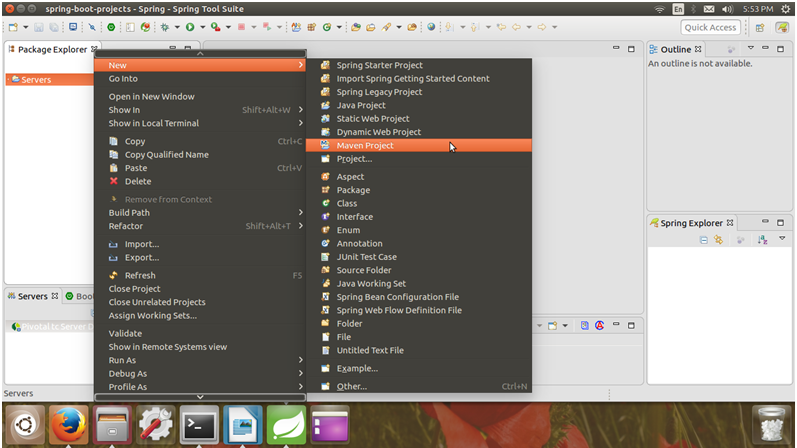
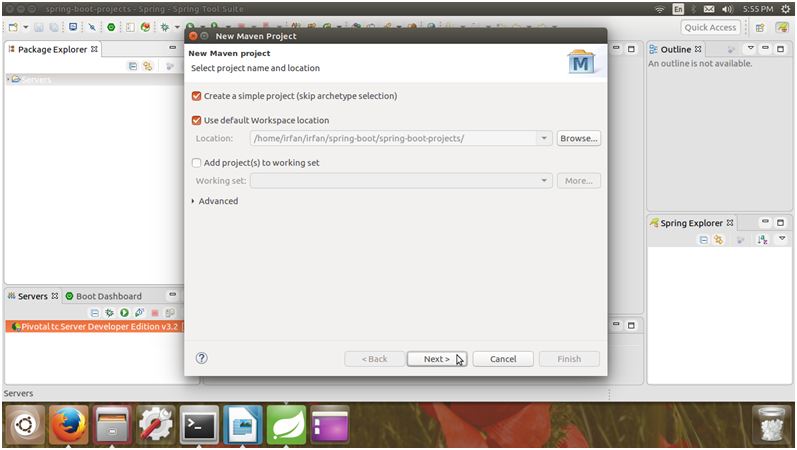
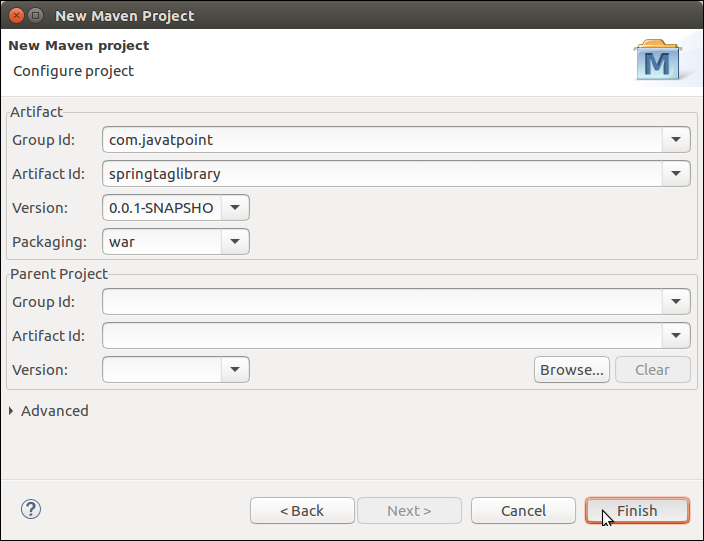
In JSP page, we can use taglib by using the following declaration.

1. **<**%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %**>**

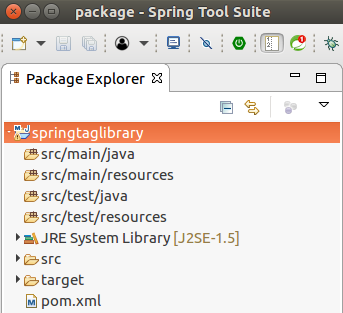
Now, lets see an example to implement these tags in spring security maven project.

We are using STS (Spring Tool Suite) to create the project. See the example.

### **Create Project**

Click on **finish** button and it will create a maven project that looks like this:



### **Spring Security Configuration**

To configure Spring Security in the Spring MVC application, put the following four files inside the **com.javatpoint** folder.

**AppConfig.java**

1. package com.javatpoint;
2. import org.springframework.context.annotation.Bean;
3. import org.springframework.context.annotation.ComponentScan;
4. import org.springframework.context.annotation.Configuration;
5. import org.springframework.web.servlet.config.annotation.EnableWebMvc;
6. import org.springframework.web.servlet.view.InternalResourceViewResolver;
7. import org.springframework.web.servlet.view.JstlView;
9. @EnableWebMvc
10. @Configuration
11. @ComponentScan({ "com.javatpoint.controller.\*" })
12. public class AppConfig {
13. @Bean
14. public InternalResourceViewResolver viewResolver() {
15. InternalResourceViewResolver viewResolver
16. = new InternalResourceViewResolver();
17. viewResolver.setViewClass(JstlView.class);
18. viewResolver.setPrefix("/WEB-INF/views/");
19. viewResolver.setSuffix(".jsp");
20. return viewResolver;
21. }
22. }

The AppConfig is used to set view location suffix of the view files.

**// MvcWebApplicationInitializer.java**

1. package com.javatpoint;
3. import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
4. public class MvcWebApplicationInitializer extends
5. AbstractAnnotationConfigDispatcherServletInitializer {
6. @Override
7. protected Class**<?>**[] getRootConfigClasses() {
8. return new Class[] { WebSecurityConfig.class };
9. }
10. @Override
11. protected Class**<?>**[] getServletConfigClasses() {
12. // TODO Auto-generated method stub
13. return null;
14. }
15. @Override
16. protected String[] getServletMappings() {
17. return new String[] { "/" };
18. }
19. }

This class is used to initialize servlet dispatcher.

**// SecurityWebApplicationInitializer.java**

1. package com.javatpoint;
2. import org.springframework.security.web.context.\*;
3. public class SecurityWebApplicationInitializer
4. extends AbstractSecurityWebApplicationInitializer {
6. }

Create one more class that is used to create user and apply authentication and authorization on the user accessibility.

**// WebSecurityConfig.java**

1. package com.javatpoint;
3. import org.springframework.context.annotation.\*;
4. import org.springframework.security.config.annotation.web.builders.HttpSecurity;
5. import org.springframework.security.config.annotation.web.configuration.\*;
6. import org.springframework.security.core.userdetails.\*;
7. import org.springframework.security.core.userdetails.User.UserBuilder;
8. import org.springframework.security.provisioning.InMemoryUserDetailsManager;
9. import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
10. @EnableWebSecurity
11. @ComponentScan("com.javatpoint")
12. public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
14. @Bean
15. public UserDetailsService userDetailsService() {
16. // ensure the passwords are encoded properly
17. UserBuilder users = User.withDefaultPasswordEncoder();
18. InMemoryUserDetailsManager manager = new InMemoryUserDetailsManager();
19. manager.createUser(users.username("mohan").password("1mohan23").roles("USER").build());
20. manager.createUser(users.username("admin").password("admin123").roles("ADMIN").build());
21. return manager;
22. }
24. @Override
25. protected void configure(HttpSecurity http) throws Exception {
27. http.authorizeRequests().
28. antMatchers("/index","/").permitAll()
29. .antMatchers("/admin","/user").authenticated()
30. .and()
31. .formLogin()
32. .and()
33. .logout()
34. .logoutRequestMatcher(new AntPathRequestMatcher("/logout"));
36. }
37. }

### **Controller**

Now, create a controller to handle request and respond back.

**// HomeController.java**

1. package com.javatpoint.controller;
2. import org.springframework.stereotype.Controller;
3. import org.springframework.web.bind.annotation.RequestMapping;
4. import org.springframework.web.bind.annotation.RequestMethod;
5. @Controller
6. public class HomeController {
7. @RequestMapping(value="/", method=RequestMethod.GET)
8. public String index() {
9. return "index";
10. }
11. @RequestMapping(value="/user", method=RequestMethod.GET)
12. public String user() {
13. return "admin";
14. }
15. @RequestMapping(value="/admin", method=RequestMethod.GET)
16. public String admin() {
18. return "admin";
19. }
20. }

### **View**

Create view (jsp) files to show the output to the user. We have created three JSP files, see the below.

**// index.jsp**

1. **<html>**
2. **<head>**
3. **<title>**Home Page**</title>**
4. **</head>**
5. **<body>**
6. **<a** href="user"**>**User**</a>** **<a** href="admin"**>**Admin**</a>** **<br>** **<br>**
7. Welcome to Javatpoint!
8. **</body>**
9. **</html>**

**// user.jsp**

1. **<html>**
2. **<head>**
3. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8"**>**
4. **<title>**Home Page**</title>**
5. **</head>**
6. **<body>**
7. Welcome to user page!
8. **</body>**
9. **</html>**

**// admin.jsp**

In the admin page, we have used authorize tag to that evaluates only when the given role is satisfied.

1. **<**%@ taglib uri="http://www.springframework.org/security/tags" prefix="security" %**><html>**
2. **<head>**
3. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8"**>**
4. **<title>**Home Page**</title>**
5. **</head>**
6. **<body>**
7. Welcome to admin page!
8. **<a** href="logout"**>**logout**</a>** **<br><br>**
9. **<security:authorize** access="hasRole('ADMIN')"**>**
10. Hello ADMIN
11. **</security:authorize>**
12. **<security:csrfInput/>**
13. **</body>**
14. **</html>**

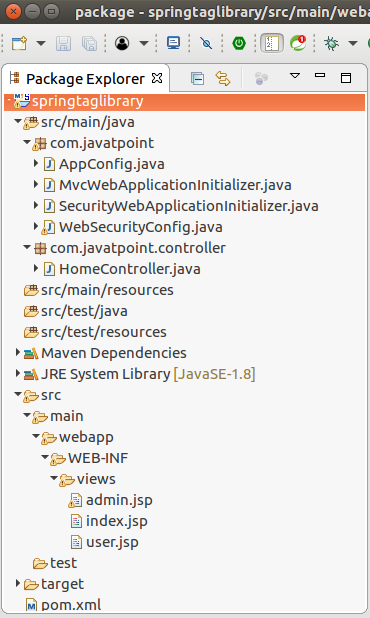
### **Project Dependencies**

Our project contains the following dependencies that are required to build application.

**// pom.xml**

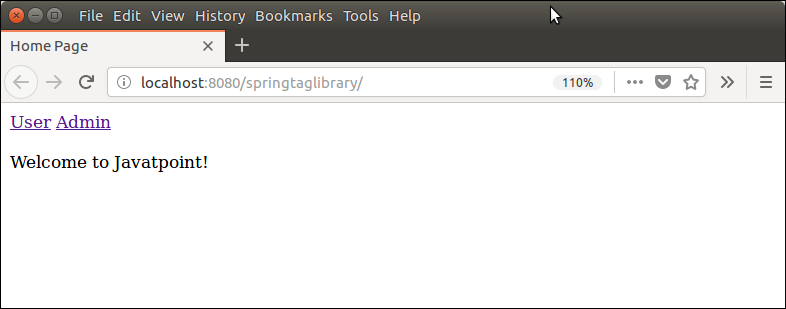
1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
2. **<modelVersion>**4.0.0**</modelVersion>**
3. **<groupId>**com.javatpoint**</groupId>**
4. **<artifactId>**springtaglibrary**</artifactId>**
5. **<version>**0.0.1-SNAPSHOT**</version>**
6. **<packaging>**war**</packaging>**
7. **<properties>**
8. **<maven.compiler.target>**1.8**</maven.compiler.target>**
9. **<maven.compiler.source>**1.8**</maven.compiler.source>**
10. **</properties>**
11. **<dependencies>**
12. **<dependency>**
13. **<groupId>**org.springframework**</groupId>**
14. **<artifactId>**spring-webmvc**</artifactId>**
15. **<version>**5.0.2.RELEASE**</version>**
16. **</dependency>**
17. **<dependency>**
18. **<groupId>**org.springframework.security**</groupId>**
19. **<artifactId>**spring-security-web**</artifactId>**
20. **<version>**5.0.0.RELEASE**</version>**
21. **</dependency>**
22. **<dependency>**
23. **<groupId>**org.springframework.security**</groupId>**
24. **<artifactId>**spring-security-core**</artifactId>**
25. **<version>**5.0.4.RELEASE**</version>**
26. **</dependency>**
27. <!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-taglibs -->
28. **<dependency>**
29. **<groupId>**org.springframework.security**</groupId>**
30. **<artifactId>**spring-security-taglibs**</artifactId>**
31. **<version>**5.0.4.RELEASE**</version>**
32. **</dependency>**
33. <!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-config -->
34. **<dependency>**
35. **<groupId>**org.springframework.security**</groupId>**
36. **<artifactId>**spring-security-config**</artifactId>**
37. **<version>**5.0.4.RELEASE**</version>**
38. **</dependency>**
39. <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
40. **<dependency>**
41. **<groupId>**javax.servlet**</groupId>**
42. **<artifactId>**javax.servlet-api**</artifactId>**
43. **<version>**3.1.0**</version>**
44. **<scope>**provided**</scope>**
45. **</dependency>**
46. **<dependency>**
47. **<groupId>**javax.servlet**</groupId>**
48. **<artifactId>**jstl**</artifactId>**
49. **<version>**1.2**</version>**
50. **</dependency>**
51. <!-- https://mvnrepository.com/artifact/org.springframework/spring-framework-bom -->
52. **</dependencies>**
53. **<build>**
54. **<plugins>**
55. **<plugin>**
56. **<groupId>**org.apache.maven.plugins**</groupId>**
57. **<artifactId>**maven-war-plugin**</artifactId>**
58. **<version>**2.6**</version>**
59. **<configuration>**
60. **<failOnMissingWebXml>**false**</failOnMissingWebXml>**
61. **</configuration>**
62. **</plugin>**
63. **</plugins>**
64. **</build>**
65. **</project>**

After adding all these files, our project looks like this:

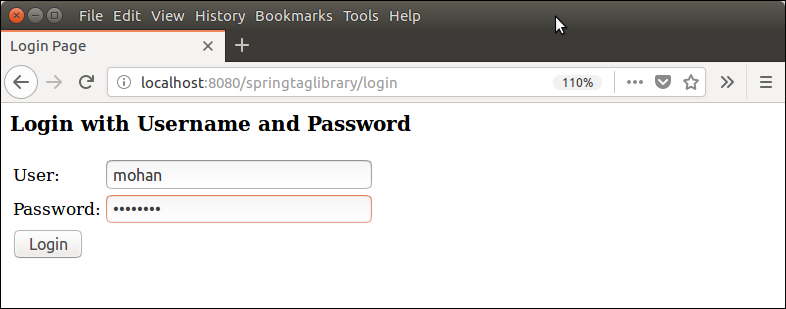


### **Run the Application**

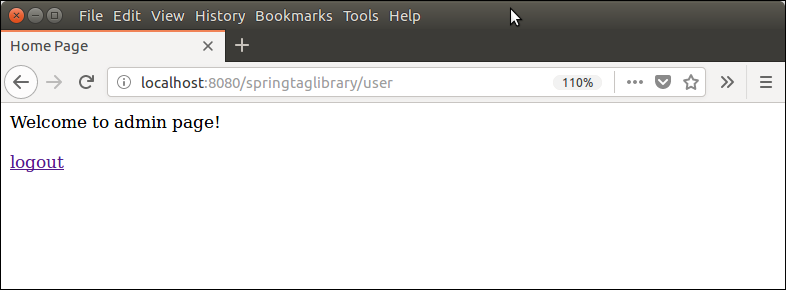
Right click on the project and select **run on server**. It shows the following output to the browser.



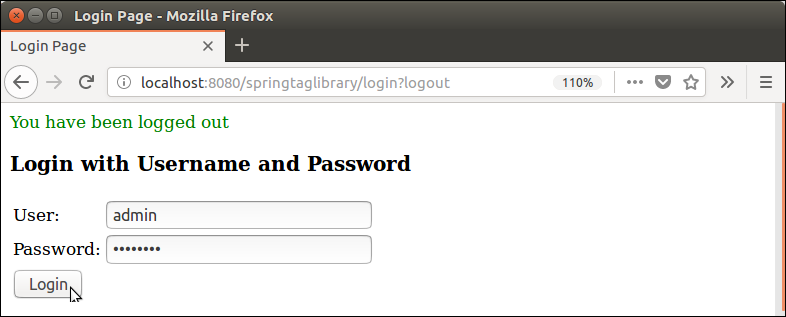
Click on the User and Login by providing credentials that are set in **AppSecurityConfig** file.



After successful login, it shows the admin page that looks like below. Here, notice that the content written inside the authorize tag is not displayed because logged in user has role USER.



Logout and now login as an admin by providing admin credentials.



After logged in as admin, see this time the authorize tag evaluates and it shows the following output.

