# Spring Security 4 Logout Example

Generally, In your views, you should be providing a simple **logout link** to logout a user, something like shown below:

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>  <%@ taglib prefix="c" uri="<http://java.sun.com/jsp/jstl/core>"%>  <html>  <head>      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">      <title>Admin page</title>  </head>  <body>      Dear <strong>${user}</strong>, Welcome to Admin Page.      <a href="<c:url value="/logout" />">Logout</a>  </body>  </html> |

Nothing fancy about it. Now , We just need to map this **/logout** link in our controller. Create a new method like shown below:

|  |
| --- |
| @RequestMapping(value="/logout", method = RequestMethod.GET)  public String logoutPage (HttpServletRequest request, HttpServletResponse response) {      Authentication auth = SecurityContextHolder.getContext().getAuthentication();      if (auth != null){          new SecurityContextLogoutHandler().logout(request, response, auth);      }      return "redirect:/login?logout";//You can redirect wherever you want, but generally it's a good practice to show login screen again.  } |

Here firstly we identified if user was authenticated before using**SecurityContextHolder.getContext().getAuthentication()**. If he/she was, then we called**SecurityContextLogoutHandler().logout(request, response, auth)** to logout user properly.

This **logout** call performs following:

* Invalidates HTTP Session ,then unbinds any objects bound to it.
* Removes the Authentication from the SecurityContext to prevent issues with concurrent requests.
* Explicitly clears the context value from the current thread.

That’s it. You don’t need anything else anywhere in your application to handle logout. Notice that you don’t even need to do anything special in your spring configuration(xml or annotation based), shown below just for information:

|  |
| --- |
| import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.Configuration;  import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  import org.springframework.security.config.annotation.web.builders.HttpSecurity;  import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;    @Configuration  @EnableWebSecurity  public class SecurityConfiguration extends WebSecurityConfigurerAdapter {          @Autowired      public void configureGlobalSecurity(AuthenticationManagerBuilder auth) throws Exception {          auth.inMemoryAuthentication().withUser("bill").password("abc123").roles("USER");          auth.inMemoryAuthentication().withUser("admin").password("root123").roles("ADMIN");          auth.inMemoryAuthentication().withUser("dba").password("root123").roles("ADMIN","DBA");      }        @Override      protected void configure(HttpSecurity http) throws Exception {          http.authorizeRequests()          .antMatchers("/", "/home").permitAll()          .antMatchers("/admin/\*\*").access("hasRole('ADMIN')")          .antMatchers("/db/\*\*").access("hasRole('ADMIN') and hasRole('DBA')")          .and().formLogin().loginPage("/login")          .usernameParameter("ssoId").passwordParameter("password")          .and().exceptionHandling().accessDeniedPage("/Access\_Denied");      }  } |

There is no special logout handling mentioned in above configuration.

**Above security configuration in XML configuration format would be:**

|  |
| --- |
| <beans:beans xmlns="<http://www.springframework.org/schema/security>"      xmlns:beans="<http://www.springframework.org/schema/beans>"      xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"      xsi:schemaLocation="<http://www.springframework.org/schema/beans> <http://www.springframework.org/schema/beans/spring-beans-4.1.xsd>  <http://www.springframework.org/schema/security> <http://www.springframework.org/schema/security/spring-security-4.0.xsd>">        <http auto-config="true" >          <intercept-url pattern="/" access="hasRole('USER')" />          <intercept-url pattern="/home" access="hasRole('USER')" />          <intercept-url pattern="/admin\*\*" access="hasRole('ADMIN')" />          <intercept-url pattern="/dba\*\*" access="hasRole('ADMIN') and hasRole('DBA')" />          <form-login  login-page="/login"                       username-parameter="ssoId"                       password-parameter="password"                       authentication-failure-url="/Access\_Denied" />      </http>        <authentication-manager >          <authentication-provider>              <user-service>                  <user name="bill"  password="abc123"  authorities="ROLE\_USER" />                  <user name="admin" password="root123" authorities="ROLE\_ADMIN" />                  <user name="dba"   password="root123" authorities="ROLE\_ADMIN,ROLE\_DBA" />              </user-service>          </authentication-provider>      </authentication-manager>    </beans:beans> |

Rest of application code is same as mentioned in every post in this series

**Following technologies being used:**

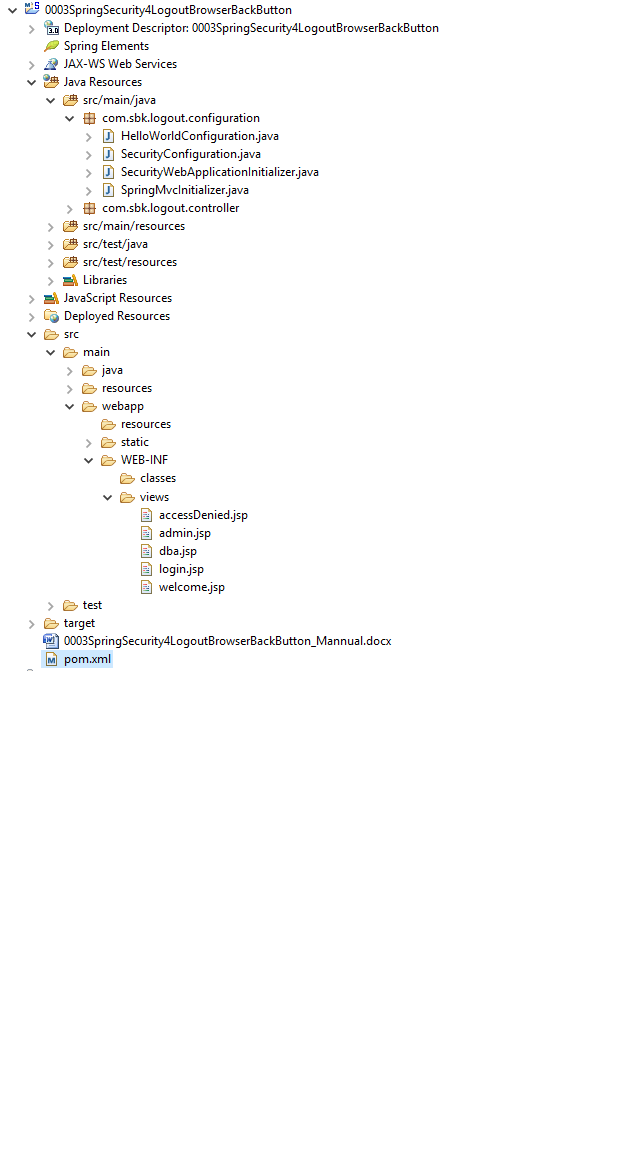
* Spring 4.1.6.RELEASE
* Spring Security 4.0.1.RELEASE
* Maven 3
* JDK 1.7
* Tomcat 7
* STS

Let’s begin.

1. **Create a Spring MVC project using STS**
2. **Delete the web.xml as we will be doing java Configuration**
3. **Delete the spring folder from WEB-INF**
4. **Update pom.xml**

* *First thing to notice here is the maven-war-plugin declaration. As we are using full annotation configuration,****we don’t even use web.xml****, so we will need to configure this plugin in order to avoid maven failure to build war package. We are using latest versions(at time of writing) of Spring and Spring Security.*
* *Along with that, we have also included JSP/Servlet/Jstl dependencies which we will be needing as we are going to use servlet api’s and jstl view in our code. In general, containers might already contains these libraries, so we can set the scope as ‘provided’ for them in pom.xml.*

1. **Following is the project structure**

****

1. **Let’s now add the content mentioned in above structure explaining each in detail**.
2. **Create *com.sbk.logout.configuratio.SecurityConfiguration***  **class**
3. ****

#### Register the springSecurityFilter with war com.sbk.logout.configuration.SecurityWebApplicationInitializer

Below specified initializer class registers the springSecurityFilter [created in previous step] with application war.



|  |
| --- |
| package com.websystique.springsecurity.configuration;  import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;  public class SecurityWebApplicationInitializer extends AbstractSecurityWebApplicationInitializer {    } |

This class is exactly same as in Previous demo[0001SpringSecurity4HelloWorldAnnotation](file:///D:\Q3_Spring_Security-Level2\ws\0001SpringSecurity4HelloWorldAnnotation)

**Above setup in XML configuration format would be(In web.xml):**

|  |
| --- |
| <filter>      <filter-name>springSecurityFilterChain</filter-name>      <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>  </filter>  <filter-mapping>      <filter-name>springSecurityFilterChain</filter-name>      <url-pattern>/\*</url-pattern>  </filter-mapping> |

#### Add Controller com.sbk.logout.controller

#### 

#### Add SpringMVC Configuration Class

**com.sbk.logout.configuration**



#### Add Initializer class(This class is exactly same as in [previous post](file:///D:\Q3_Spring_Security-Level2\ws\0002SpringSecurity4CustomLoginFormAnnotation\src\main\java\com\sbk\customform\configuration\SpringMvcInitializer.java).)

**com.sbk.logout.configuration**



Notice that above initializer class extends AbstractAnnotationConfigDispatcherServletInitializerwhich is the base class for all WebApplicationInitializer implementations. Implementations of WebApplicationInitializer configures ServletContext programatically, for Servlet 3.0 environments. It means we won’t be using web.xml and we will deploy the app on Servlet 3.0 container.

#### Add Views



login.jsp



login.jsp additionally contains CSS for login panel layout.

***Notice the CSRF (*Cross Site Request Forgery*)related line in above jsp:***

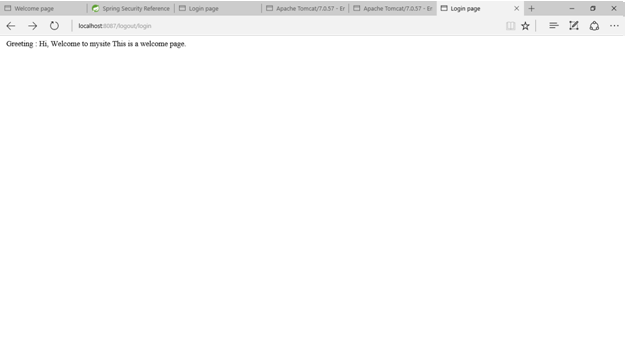
|  |
| --- |
| ***<input type="hidden" name="${\_csrf.parameterName}" value="${\_csrf.token}" /></strong>*** |

***This is required to protect against***[***CSRF attacks***](http://docs.spring.io/spring-security/site/docs/3.2.x/reference/htmlsingle/#csrf)***.***

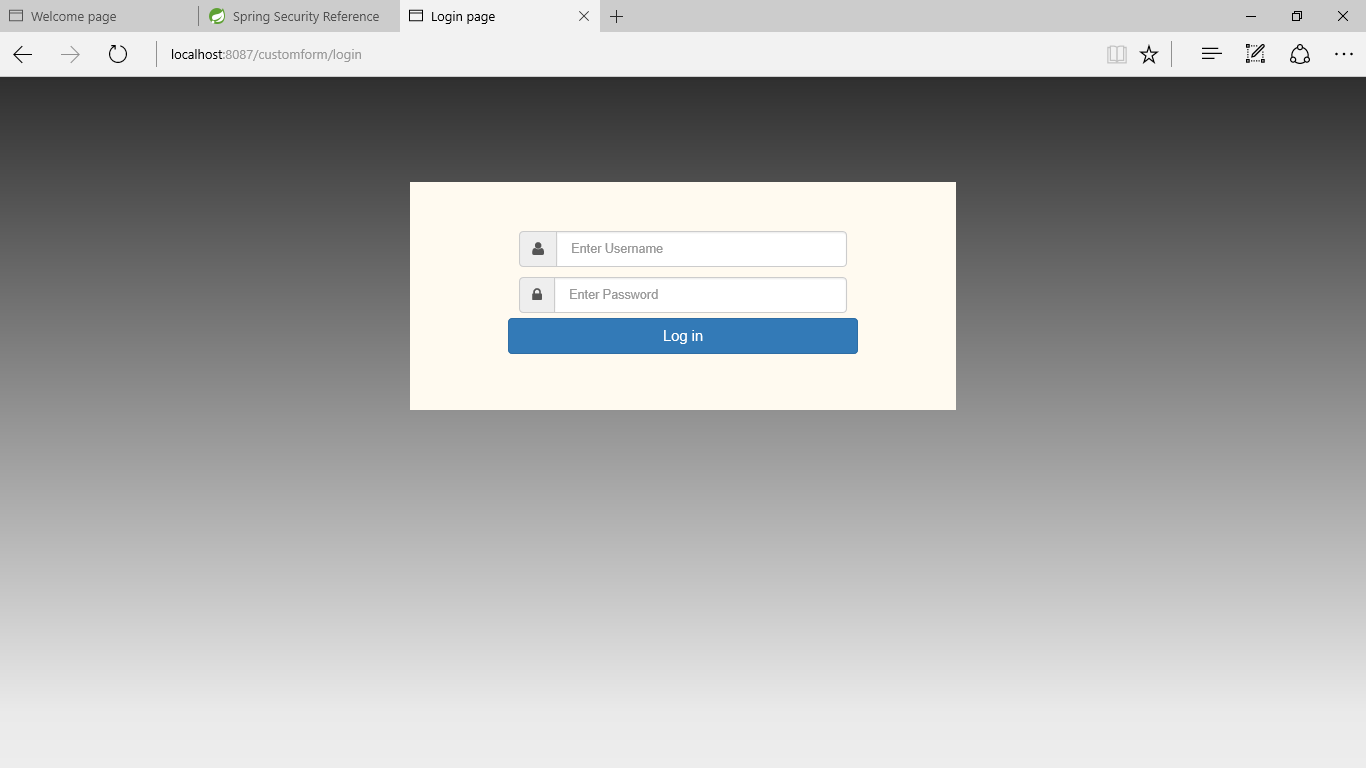
#### Build and Deploy the application

***NOTE : Before deploying, check the java build path(1.7), Check java compiler(1.7), java facets(jdk1.7,web module version 3.0), targeted runtime(tom cat7) and Add Maven dependency to build path***

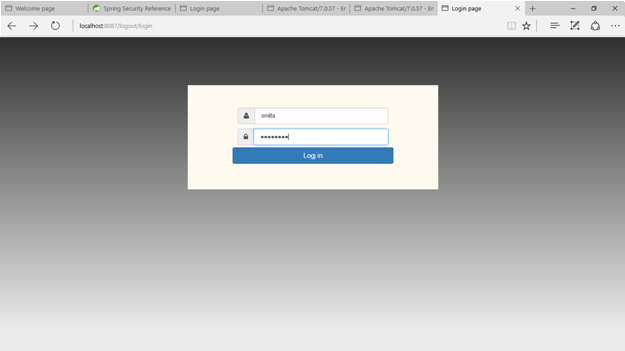
#### Run the application(http://localhost:8087/logout/)



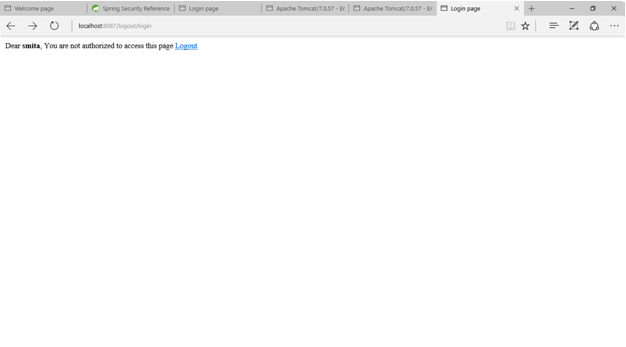
Now try to access admin page on http://localhost:8087/logout/admin, you will be prompted for login.



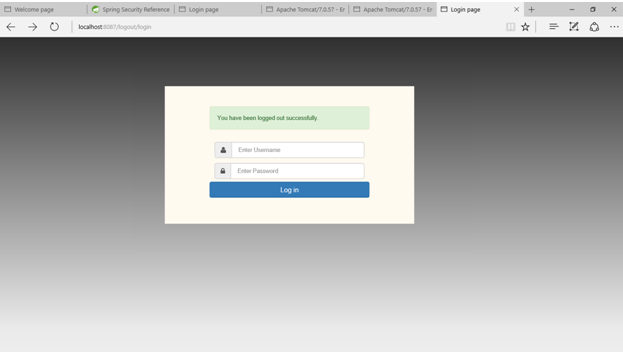
Provide credentials of a ‘USER’ role.*enter [smita,password]*



Submit, you will see AccessDenied Page



1. Logout. http://localhost:8087/logout/login?logout



click on **browser back button**, you will remain at login screen.

