Spring Security Example UserDetailsService

APRIL 2, 2018 BY PANKAJ — 23 COMMENTS

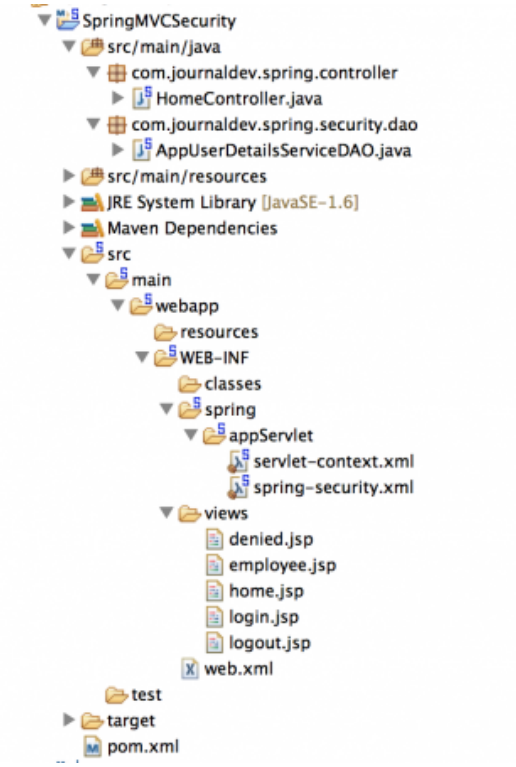
Welcome to Spring Security Example using UserDetailsService. In the last post we learned how to use Spring Security in Web Application. Today we will look into how we can integrate Spring Security in Spring MVC Projects for authentication purposes.

Table of Contents [hide]

Spring Security Example

Integrating Spring Security with Spring MVC Framework is very easy, because we already have Spring Beans configuration file. All we need is to create spring security authentication related changes to get it working. Today we will look into how we can implement authentication in Spring MVC application using in-memory, `UserDetailsService` DAO implementation and JDBC based authentication.

First create a simple Spring MVC project in the Spring Tool Suite, that will give us the base spring MVC application to build our Spring security example application. Once we will be done with all the changes, our application will look like below image.



Let's look into each of the components of our Spring security example project.

Spring Security Maven Dependencies

Our final pom.xml file looks like below.

```
<?xml version="1.0" encoding="UTF-8"?>
<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/maven-v4_0_0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.journaldev.spring</groupId>
	<artifactId>SpringMVCSecurity</artifactId>
	<name>SpringMVCSecurity</name>
	<packaging>war</packaging>
	<version>1.0.0-BUILD-SNAPSHOT</version>
	<properties>
		<java-version>1.6</java-version>
		<org.springframework-version>4.0.2.RELEASE</org.springframework-
version>
		<org.aspectj-version>1.7.4</org.aspectj-version>
		<org.slf4j-version>1.7.5</org.slf4j-version>
	</properties>
	<dependencies>
		<!-- Spring -->
		<dependency>
			<groupId>org.springframework</groupId>
```

We have included `spring-security-config` and `spring-security-web` dependencies for Spring Security. Apart from that we have `spring-jdbc` dependency because we will be using Spring JDBC authentication too.

Rest of the dependencies are related to Spring MVC, logging, AOP etc.

Spring Security Example Deployment Descriptor

Our web.xml file looks like below.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

	<!-- Spring Security Configuration File -->
	<context-param>
```

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

        <!-- Creates the Spring Container shared by all Servlet and Filters -->
        <listener>
                <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
        </listener>
        <listener>
                <listener-
class>org.springframework.security.web.session.HttpSessionEventPublisher</listener-
class>
```

`contextConfigLocation` is the context parameter where we provide the spring security beans configuration file name. It is used by `ContextLoaderListener` to configure authentication in our application.

We have also added `HttpSessionEventPublisher` listener to publish session created/destroyed events to the Spring Root WebApplicationContext.

I am also setting `session-timeout` to 15 minutes, this is used for auto timeout when user is inactive for 15 minutes.

`DelegatingFilterProxy` is the application filter defined, it is used for intercepting the HTTP requests and performing authentication related tasks.

`DispatcherServlet` servlet is the front controller for the Spring MVC application.

UserDetailsService

If we want to use any DAO class for authentication, we need to implement `UserDetailsService` interface. Once the DAO is configured, it's `loadUserByUsername()` is used to validate the user.

```
package com.journaldev.spring.security.dao;

import java.util.Collection;
import java.util.List;

import org.apache.commons.logging.Log;
```

```
import org.apache.commons.logging.LogFactory;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;

public class AppUserDetailsServiceDAO implements UserDetailsService {

        protected final Log logger = LogFactory.getLog(getClass());

        @Override
        public UserDetails loadUserByUsername(final String username)
                        throws UsernameNotFoundException {

                logger.info("loadUserByUsername username="+username);
```

Note that I am returning `UserDetails` instance by using anonymous inner class implementation. Ideally, we should have an implementation class for `UserDetails` that can have other user data also, such as emailID, user name, address etc.

Notice that the only combination that will work is when user name is “pankaj” and password is “pankaj123”.

Spring Security Example Controller Class

Here is our controller class that defines two URIs that we can access.

```
package com.journaldev.spring.controller;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;

import javax.servlet.http.HttpServletRequest;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HomeController {
```

```
        private static final Logger logger =
LoggerFactory.getLogger(HomeController.class);
```

In our example, we will apply authentication to URI "/emp/get/{id}" only. All other URIs will be accessible without any authentication. login, logout and denied URIs are used to send corresponding response pages when secured URL is requested.

Spring Security Example Bean Configuration File

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

        <!-- DispatcherServlet Context: defines this servlet's request-processing
infrastructure -->

        <!-- Enables the Spring MVC @Controller programming model -->
        <annotation-driven />

        <!-- Handles HTTP GET requests for /resources/** by efficiently serving up
static resources in the ${webappRoot}/resources directory -->
        <resources mapping="/resources/**" location="/resources/" />
```

Our spring bean configuration file is simple, it has configurations related to Spring MVC application only.

Spring MVC Security Configuration

This is the most important part of our tutorial, let's have a look at our file. We will understand each of the parts one by one.

spring-security.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

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

        <!-- Configuring RoleVoter bean to use custom access roles, by default roles
                should be in the form ROLE_{XXX} -->
        <beans:bean id="roleVoter"
                class="org.springframework.security.access.vote.RoleVoter">
                <beans:property name="rolePrefix" value=""></beans:property>
        </beans:bean>

        <beans:bean id="accessDecisionManager"
                class="org.springframework.security.access.vote.AffirmativeBased">
                <beans:constructor-arg name="decisionVoters"
                        ref="roleVoter" />
        </beans:bean>
```

`accessDecisionManager` bean is defined so that we can have our custom roles, by default all the roles should start with ROLE_ and we are overriding this setting in the `roleVoter` bean property `rolePrefix`.

We can have multiple authentication managers defined in the spring security configuration. I have defined `in-memory-auth` for in-memory authentication, `dao-auth` for UserDetailsService DAO implementation and `jdbc-auth` for JDBC authentication. For JDBC authentication, I have provided configuration for DataSource defined in the application as well as if we want to use JNDI resource defined in the servlet container.

http `authentication-manager-ref` is used to define the authentication manager that will be used for authenticating the user. Currently it's configured to use the JDBC based authentication.

http `access-decision-manager-ref` is used to specifying the ID of the AccessDecisionManager implementation which should be used for authorizing HTTP requests.

`intercept-url` is used to define the URL pattern and authorities of the user who can access this page. For example, we have defined that URI "/emp/**" can be accessible only by users having "Admin" access.

`form-login` defines the login form configuration and we can provide the username and password parameter names. `authentication-failure-url` is used to define the URL for the authentication failure page. If no login failure URL is specified, Spring Security will automatically create a failure login URL at /spring_security_login?login_error and a corresponding filter to render that login failure URL when requested.

`default-target-url` is used to define the default URL that will be redirected to after successful authentication, if the user's previous action could not be resumed. This generally happens if the user visits a login page without having first requested a secured operation that triggers authentication. If unspecified, it defaults to the root of the application.

`logout` is used to define the logout processing filter. Here we are invalidating the session and sending the user to login page after successful logout. `logout-url` is used to define the URL to be used for logout action.

`access-denied-handler` defines the global error page if the user is denied the access, because he is not authorized to perform the specified action.

`session-management` will add a SessionManagementFilter filter to the filter stack for Session Management.

There are some other configurations also, but I have included most of the important ones that we use.

Spring Security Example View Pages

Let's have a quick look at our view pages, before we deploy and test our application.

`home.jsp`

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page session="false"%>
<html>
<head>
<title>Home</title>
</head>
<body>
        <h1>Hello world!</h1>

        <P>The time on the server is ${serverTime}.</P>
</body>
</html>
```

home.jsp is returned for "/home" URI and it should not require any authentication.

`employee.jsp`

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page session="false"%>
<html>
```

```
<head>
<title>Get Employee Page</title>
</head>
<body>
        <h1>Employee Information</h1>
        <p>
                Employee ID:${id}<br> Employee Name:${name}<br>
        </p>
        <c:if test="${pageContext.request.userPrincipal.name != null}">
        Hi ${pageContext.request.userPrincipal.name}<br>

        <c:url var="logoutAction" value="/j_spring_security_logout"></c:url>

        <form action="${logoutAction}" method="post">
                <input type="submit" value="Logout" />
        </form>
        </c:if>
</body>
```

This page is returned when we are accessing URI that requires authentication. Here I have provided logout option so that user can logout and terminate the session. Once logout is successful, user should be sent back to login page as configured.

login.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<html>

<head>
<title>Login Page</title>
</head>
<body>
        <h3>Login with Username and Password</h3>
        <c:url var="loginUrl" value="/j_spring_security_check"></c:url>
        <form action="${loginUrl}" method="POST">
                <table>
                        <tr>
                                <td>User ID:</td>
                                <td><input type='text' name='username' /></td>
                        </tr>
                        <tr>
                                <td>Password:</td>
```

```
                <td><input type='password' name='password' /></td>
            </tr>
            <tr>
```

There are few important points to note here. The first one is that the login URL is "**/j_spring_security_check**". This is the default login processing URL, just like the logout-url.

Another important point is the form parameters name for username and password. They should be same as configured in the spring security configurations.

logout.jsp

```
<html>
<head>
        <title>Logout Page</title>
</head>
<body>
<h2>
        Logout Successful!
</h2>

</body>
</html>
```

denied.jsp

```
<html>
<head>
        <title>Access Denied</title>
</head>
<body>
<h1>
        Access Denied!
</h1>

</body>
</html>
```

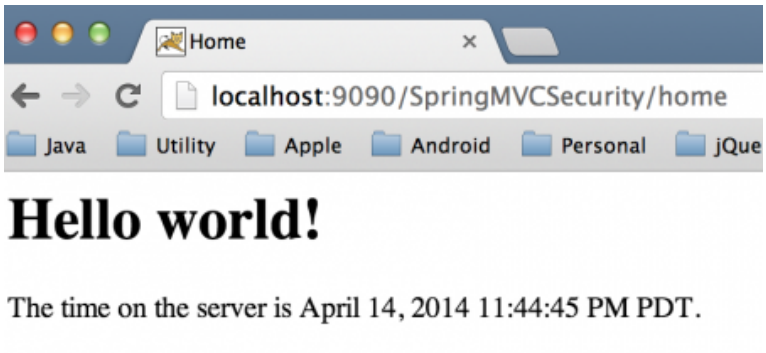
logout.jsp and denied.jsp pages are simple, but we could have included some information here based on the user details.

Our spring security example application is ready to test, note that for JDBC authentication I am using the same setup as our previous Spring Security Example. So if you have landed directly here, you should check that out.

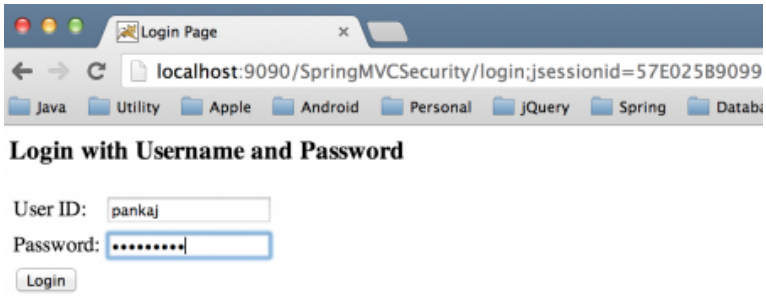
Spring Security MVC Example Testing

Just deploy the application in your favorite servlet container, mine is Apache Tomcat 7. Below images show us the different outputs for different URLs.

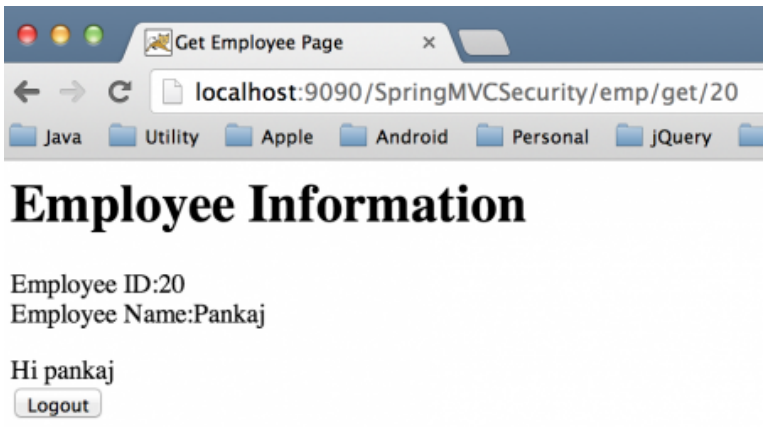
Spring Security Example – Home Page without Authentication



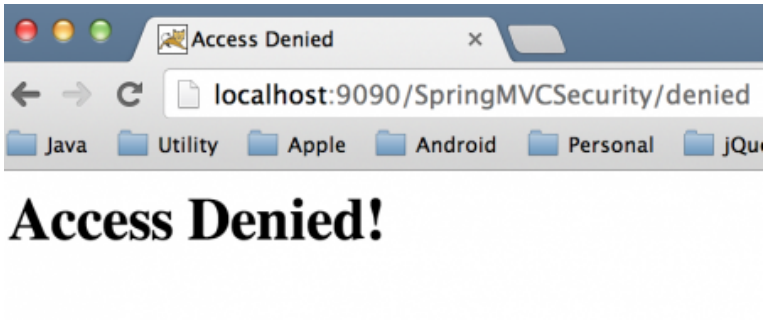
Spring Security Example – Login Page when Authentication Enabled Page is requested (/emp/get/{20})



Spring Security Example – Response Page when Authentication is Successful



Spring Security Example – Access Denied Page when Authentication is Failed



That's all for the Spring Security Example using UserDetailsService, please download the sample project from below link and explore it to learn more.

Download Spring MVC Security Project

« PREVIOUS
Spring Security Example Tutorial

NEXT »
Spring Tutorial – Spring Core Framework Tutorials

About Pankaj

If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: SPRING

Comments

Sanjeev Kumar says
JULY 9, 2018 AT 6:33 PM
HI Pankaj
I implemented your code; but even after passing the right credential, it is redirecting to the home page every time . I used jdb_auth and in-memory_auth both ; but it is not not showing successful login and always redirecting to home page. Is there any more configuration required in it?
Reply

Nidhi Sable says
JANUARY 13, 2018 AT 9:39 PM
Hey, I was trying to run this program but I am getting bean factory error.
Reply

Shefali says
JULY 26, 2017 AT 4:18 AM
Hi Pankaj,
I am getting Bad Credentials exception everytime I try to log in
Reply

Arvind says
MARCH 30, 2017 AT 3:00 AM
Hi,
I want to know , how we can know the use name and his id in case of the say amazon like portal send product recommendation links to e mails to the multiple users different product.. Can we do through Spring security?
Reply

Paritosh says

FEBRUARY 16, 2017 AT 5:25 AM

I'm always getting error while running this code. Can you please help me regarding this?
Exception/ Error shown is :
Error creating bean with name 'org.springframework.security.filterChains': Cannot resolve reference to bean 'org.springframework.security.web.DefaultSecurityFilterChain#0' while setting bean property 'sourceList' with key [0]; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'org.springframework.security.web.DefaultSecurityFilterChain#0': Cannot resolve reference to bean 'org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter#0' while setting constructor argument with key [4]; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter#0': Cannot resolve reference to bean 'org.springframework.security.authentication.ProviderManager#0' while setting bean property 'authenticationManager'; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'org.springframework.security.authentication.ProviderManager#0': Cannot resolve reference to bean 'jdbc-auth' while setting constructor argument; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'jdbc-auth': Cannot resolve reference to bean 'org.springframework.security.authentication.dao.DaoAuthenticationProvider#2' while setting constructor argument with key [0]; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'org.springframework.security.authentication.dao.DaoAuthenticationProvider#2': Cannot resolve reference to bean 'org.springframework.security.provisioning.JdbcUserDetailsManager#0' while setting bean property 'userDetailsService'; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'org.springframework.security.provisioning.JdbcUserDetailsManager#0': Cannot resolve reference to bean 'dataSource' while setting bean property 'dataSource'; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'dataSource' defined in ServletContext resource [/WEB-INF/spring/appServlet/spring-security.xml]: Error setting property values; nested exception is org.springframework.beans.PropertyBatchUpdateException; nested PropertyAccessExceptions (1) are:

Reply

tim says

FEBRUARY 15, 2016 AT 7:14 AM

OMG When I run this (code unaltered) in eclipse using tomcat 7 it throws 12 exceptions. ?????

Reply

rakesh says
OCTOBER 5, 2015 AT 3:48 AM
nice one.
Reply

parakarmi says
JULY 15, 2015 AT 4:22 AM
private static final long serialVersionUID = 2059202961588104658L what is used .
Please provide spring security with rest tutorial for Login module
Reply

deepak says
JUNE 20, 2015 AT 2:55 AM
Hi Pankaj,
Many Many Thanks for this tutorial.
Reply

suraj says
MAY 19, 2015 AT 2:36 AM
Hi Pankaj,
Firstly thank you for your excellent document. I met some questions:
1. After I donwloaded the file, I found there was missed MySQL dependency in pom.xml
2. I always get "org.springframework.web.servlet.PageNotFound – No mapping found for HTTP request with URI [/SpringMVCSecurity/] in DispatcherServlet with name 'appServlet'" error. I had to update dispatch part of web.xml
appServlet
/SpringMVCSecurity/*
and input "http://localhost:8080/SpringMVCSecurity/SpringMVCSecurity/login" in browser.
Could you please help me to find the reason?
Thanks.
Michael Liu
Reply
Reply

sai mani kiran says

APRIL 26, 2015 AT 11:18 AM

thanks boss… good job

Reply

justapis says

OCTOBER 31, 2014 AT 2:36 AM

Hi Pankaj,

Can you help me with the problem using MyUserDetailService to get authorized user from DB. I have configured files below, but I can't get user data from DB via login form. I have crypted password to DB using bcrypt. Actually I can't see any log events during login process. Only login error page.

servlet-context

```
fi.isokaju.webshop.model.User
fi.isokaju.webshop.model.Product
fi.isokaju.webshop.model.UserRole
org.hibernate.dialect.MySQLDialect
true
```

spring-security

MyUserDetailService

```
public class MyUserDetailsService implements UserDetailsService {
/*
* This class is for getting and transferring User to spring framework
* userdetailuser
*
*
*/
private UserDao userDao;
@Override
public UserDetails loadUserByUsername(final String username)
throws UsernameNotFoundException {
fi.isokaju.webshop.model.User user = userDao.findByUserName(username);
List authorities = buildUserAuthority(user
.getRoles());
return buildUserForAuthentication(user, authorities);
}
// Converts fi.webshop.users.model.User user to
// org.springframework.security.core.userdetails.User
private User buildUserForAuthentication(fi.isokaju.webshop.model.User user,
List authorities) {
```

```
return new User(user.getUsername(), user.getPassword(),
user.isEnabled(), true, true, true, authorities);
}
private List buildUserAuthority(Set userRoles) {
Set setAuths = new HashSet();
// Build user's authorities
for (UserRole userRole : userRoles) {
setAuths.add(new SimpleGrantedAuthority(userRole.getRole()));
}
List Result = new ArrayList(
setAuths);
return Result;
}
public UserDao getUserDao() {
return userDao;
}
public void setUserDao(UserDao userDao) {
this.userDao = userDao;
}
}
```

web.xml

```
contextConfigLocation
/WEB-INF/spring/root-context.xml
<!-- /WEB-INF/spring/appServlet/servlet-context.xml
/WEB-INF/spring/appServlet/spring-security.xml -->
org.springframework.web.context.ContextLoaderListener
org.springframework.security.web.session.HttpSessionEventPublisher
15
contextConfigLocation
/WEB-INF/spring/appServlet/spring-security.xml
org.springframework.web.context.ContextLoaderListener
appServlet
org.springframework.web.servlet.DispatcherServlet
contextConfigLocation
/WEB-INF/spring/appServlet/servlet-context.xml
1
appServlet
/
springSecurityFilterChain
org.springframework.web.filter.DelegatingFilterProxy
springSecurityFilterChain
/*
```

Thank in advance!
Reply

justapis says
OCTOBER 31, 2014 AT 2:44 AM
Hi Pankaj,
There is tag fails in mys post Can you remove it so I send again □
Br, justapis
Reply

Antonio Maggiore says
JULY 28, 2015 AT 4:03 AM
I've the same problem
Reply

Sateesh Karampoodi says
OCTOBER 7, 2014 AT 12:30 AM
Thank you very much Pankaj.
Its really excellent tutorial and very helpful to us.
Once again thank you very much.
Reply

Peter says
OCTOBER 6, 2014 AT 6:23 PM
Hi Pankaj,
Thanks so much for the tutorial. I was lost trying to implement Spring Security into my Spring MVC app. This was extremely helpful. I have question, though, about csrf. The login page doesn't include the csrf token:
Is Spring Security handling this on its own?
Reply

Pankaj says
OCTOBER 6, 2014 AT 11:03 PM

check out index.jsp in https://www.journaldev.com/2715/spring-security-in-servlet-web-application-using-dao-jdbc-in-memory-authentication

Reply

michael says

SEPTEMBER 21, 2014 AT 7:24 PM

Hi Pankaj,

Firstly thank you for your excellent document. I met some questions:

1. After I donwloaded the file, I found there was missed MySQL dependency in pom.xml

2. I always get **"org.springframework.web.servlet.PageNotFound – No mapping found for HTTP request with URI [/SpringMVCSecurity/] in DispatcherServlet with name 'appServlet'"** error. I had to update dispatch part of web.xml

appServlet

/SpringMVCSecurity/*

and input "http://localhost:8080/SpringMVCSecurity/SpringMVCSecurity/login" in browser.

Could you please help me to find the reason?

Thanks.

Michael Liu

Reply

Ezhil says

SEPTEMBER 10, 2014 AT 11:14 AM

Wonderful post. Learned lot of things

Reply

shuo says

AUGUST 30, 2014 AT 12:14 AM

Thanks a lot Sir. Your article is very useful and learn much from it.

Reply

micheal says

AUGUST 20, 2014 AT 4:08 AM

Hi Pankaj,

first of all thanks
i downloaded the project, after successful login its getting redirected to /home,
but according to your screeshots it should land on emp/** page for username pankaj and pwd pankaj123.
if try to login with wrong credential its redirecting to access denied page, but for successful login its redirectng to /home.
please help
Reply

Sandeep says
JULY 30, 2014 AT 8:00 PM
Thanks Pankaj, I could run the application you have provided here successfully, but when I did following change; I am not able to get into the application, every time getting "Access Denied".
Database Tables –
1) USERS (USER_ID, USER_PASS, FIRST_NAME, LAST_NAME, DOB, MOBILE_NO,HOME_PHONE_NO,EMAIL_ID).
USER_ID is PK here
2) USER_ROLES(USER_ID, USER_ROLE).
USER_ID & USER_ROLE is composite primary key here and USER_ID is reference key of USERS table.
I have made following changes in spring-security.xml:
I have made following change in login.jsp file:
User ID:
Password:
Now, where I deploy and run the application, its always fails with "Access Denied", so your help here is highly appreciated please. Please let me know where am I missing configuration?
Thanking you in advance!!
Reply

Sandeep says
JULY 17, 2014 AT 7:48 PM
Hi Pankaj, excellent tutorial and it helped me to start with Spring Security!! Appreciated your efforts!!
Reply

Leave a Reply

Your email address will not be published. Required fields are marked *
Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

SPRING FRAMEWORK

Spring Tutorial

Spring Core

- Spring Framework
- Spring Dependency Injection
- Spring IoC and Bean
- Spring Bean Life Cycle
- Spring REST
- Spring REST XML
- Spring RestTemplate

- Spring AOP
- Spring AOP Method Profiling
- Spring Annotations
- Spring @Autowired
- Spring @RequestMapping

Spring MVC

- Spring MVC Example
- Spring MVC Tutorial
- Spring MVC Exception Handling
- Spring MVC Validator
- Spring MVC Intercepter
- Spring MVC File Upload
- Spring MVC i18n
- Spring MVC Hibernate MqSQL

Spring ORM

- Spring ORM
- Spring ORM JPA
- Spring Data JPA
- Spring Transaction
- Spring JdbcTemplate

Spring Security

- Spring Security Overview
- Spring Security Example Tutorial
- Spring Security UserDetailsService
- Spring MVC Login Logout
- Spring Security Roles

Spring Boot

- Spring Boot Tutorial
- Spring Boot Components
- Spring Boot CLI Hello World
- Spring Boot Initilizr Web
- Spring Boot Initilizr IDE
- Spring Boot Initilizr CLI
- Spring Boot Initilizr Tools
- Spring Boot MongoDB
- Spring Boot Redis Cache
- Spring Boot Interview Questions

Spring Batch

- Spring Batch
- Spring Batch Example

Spring AMQP

- Spring AMQP
- Spring RabbitMQ
- Spring AMQP RabbitMQ
- Apache ActiveMQ
- Spring ActiveMQ Tutorial

- Spring ActiveMQ Example

Spring Integrations

- Spring JDBC
- Spring DataSource JNDI
- Spring Hibernate
- Spring Primefaces JPA
- Spring Primefaces MongoDB
- Spring Primefaces Hibernate
- SpringRoo Primefaces Hibernate
- Spring JSF
- Spring JDF Hibernate

Miscellaneous

- Spring Data MongoDB
- Spring Interview Questions

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial