Mindtree Kalinga

Spring Security Basic

May 17, 2018

By

Shashi Kant Kumar(M1044558)

Spring Security Introduction

- Spring Security is a framework which provides various security features like: authentication, authorization to create secure Java Enterprise Applications.
- **Authorization** is the process to allow authority to perform actions in the application.
- **Authentication**: Process of checking the user, who they claim to be
- 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 (Lightweight Directory Access Protocol)
    - Form-based authentication
    - Open ID authentication
    - Automatic remember-me authentication
    - Kerberos
    - JOSSO (Java Open Source Single Sign-On)
    - App Fuse
    - Andro MDA
    - 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.
- A legacy system, in the context of computing, refers to outdated computer systems, programming languages or application software that are used instead of available upgraded versions.

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.
- 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 worldwide.

Spring Security Features

- 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) Login Module

  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.

- 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.

- Authorization

  Spring Security provides 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.

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.

- spring-security-core.jar

  It contains top level packages-

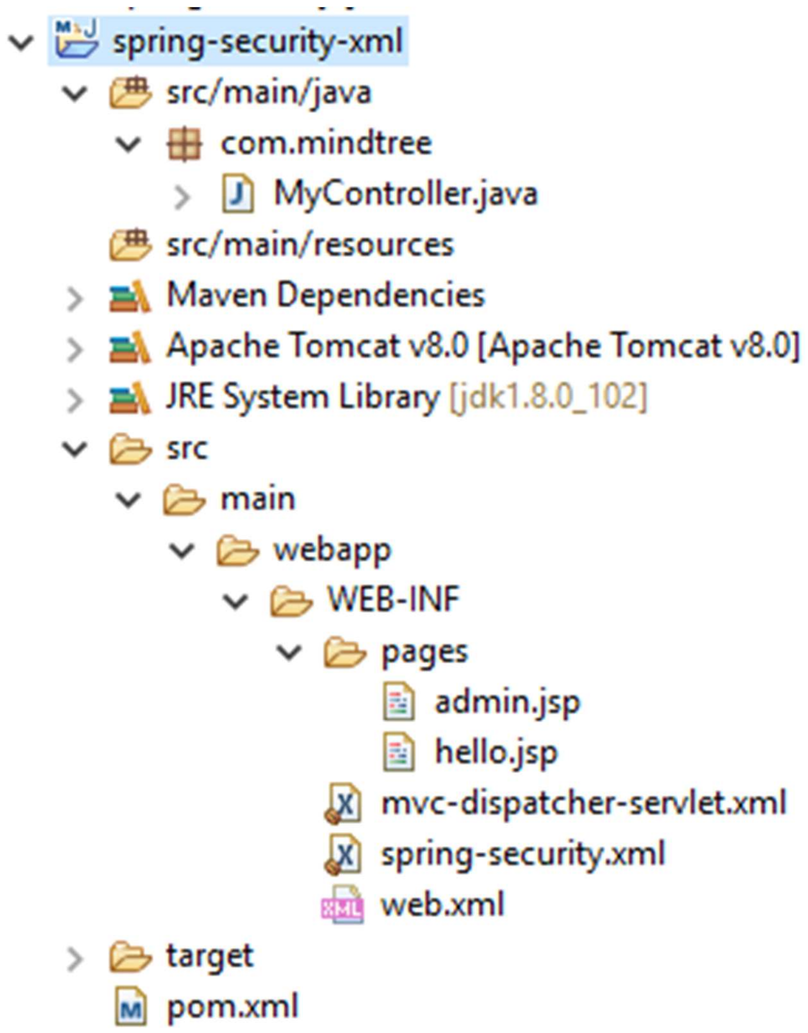
  - org.springframework.security.core
  - org.springframework.security.access
  - org.springframework.security.authentication

- 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

Simple Login Page Using XML

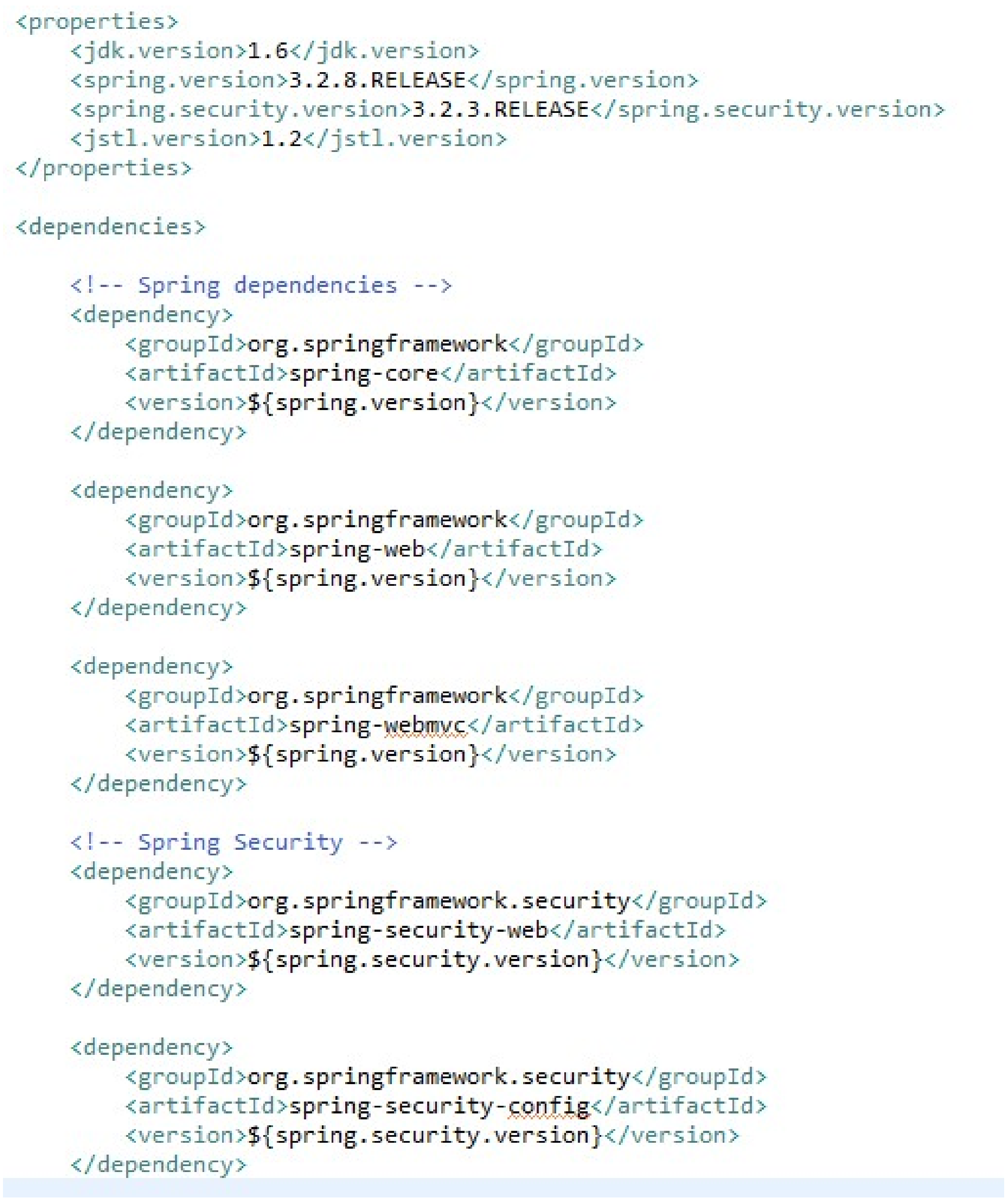
➢ Folder Structure



```
spring-security-xml
  src/main/java
    com.mindtree
      MyController.java
  src/main/resources
  Maven Dependencies
  Apache Tomcat v8.0 [Apache Tomcat v8.0]
  JRE System Library [jdk1.8.0_102]
  src
    main
      webapp
        WEB-INF
          pages
            admin.jsp
            hello.jsp
          mvc-dispatcher-servlet.xml
          spring-security.xml
          web.xml
  target
  pom.xml
```

✓ **Step 1.**

Add required Dependencies in POM.xml file



```xml
<properties>
    <jdk.version>1.6</jdk.version>
    <spring.version>3.2.8.RELEASE</spring.version>
    <spring.security.version>3.2.3.RELEASE</spring.security.version>
    <jstl.version>1.2</jstl.version>
</properties>

<dependencies>

    <!-- Spring dependencies -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring.version}</version>
    </dependency>

    <!-- Spring Security -->
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-web</artifactId>
        <version>${spring.security.version}</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-config</artifactId>
        <version>${spring.security.version}</version>
    </dependency>
```

You may can add Jstl dependency if required

```
<!-- jstl for jsp page -->
<dependency>
      <groupId> jstl </groupId>
      <artifactId> jstl </artifactId>
      <version> ${jstl.version} </version>
</dependency>
```

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.

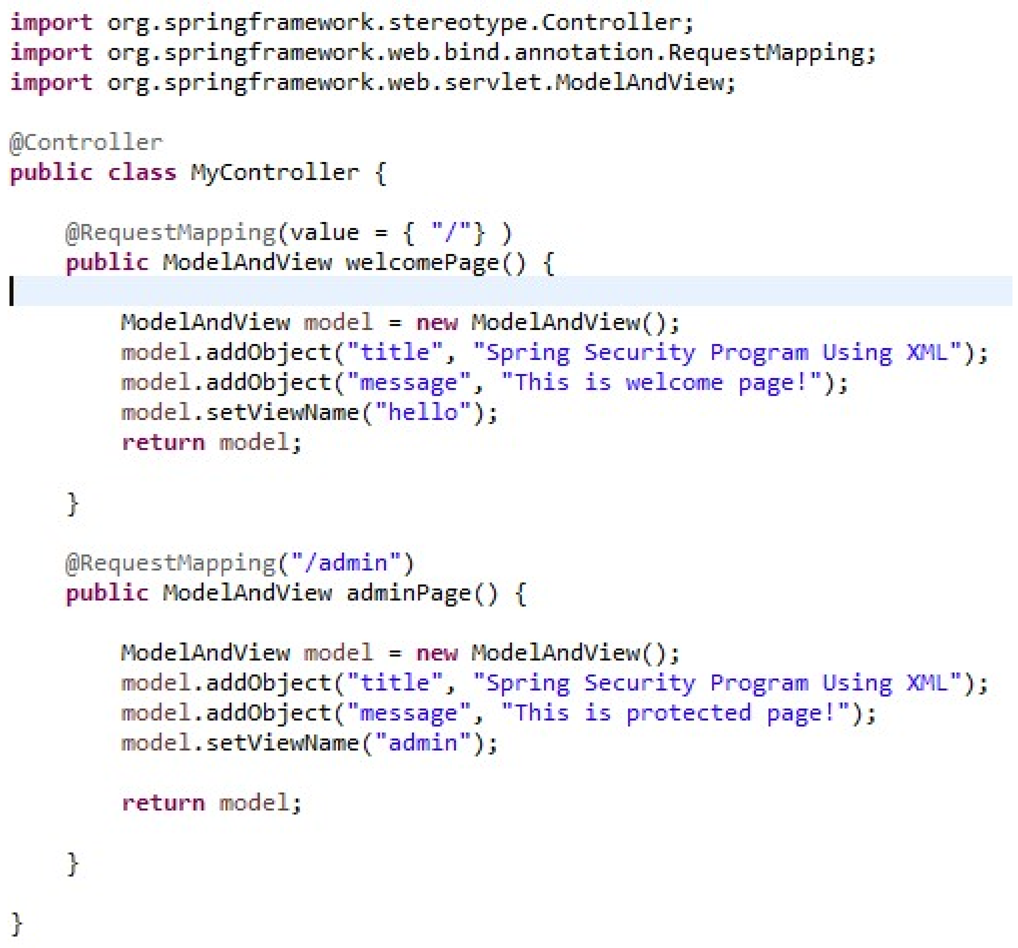
All the classes and interfaces are located in the **org.springframework.security.web** package.

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.

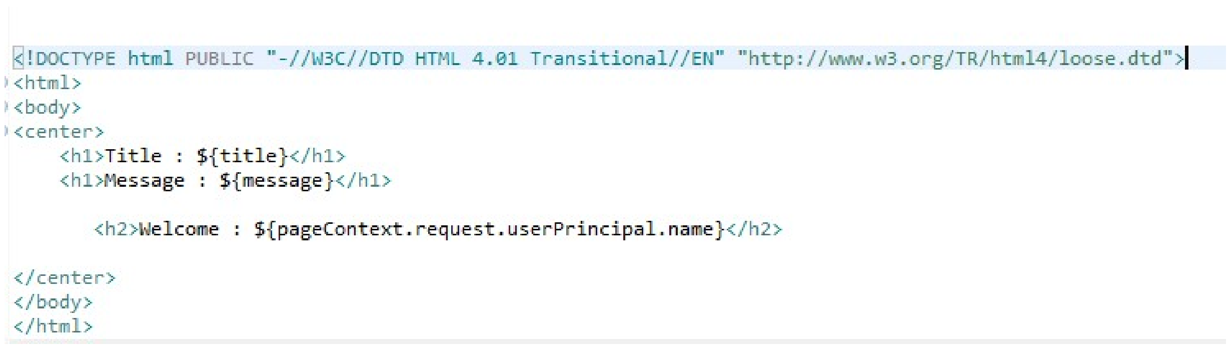
All the classes and interfaces are stored in org.springframework.security.config package.

✓ **Step 2.**



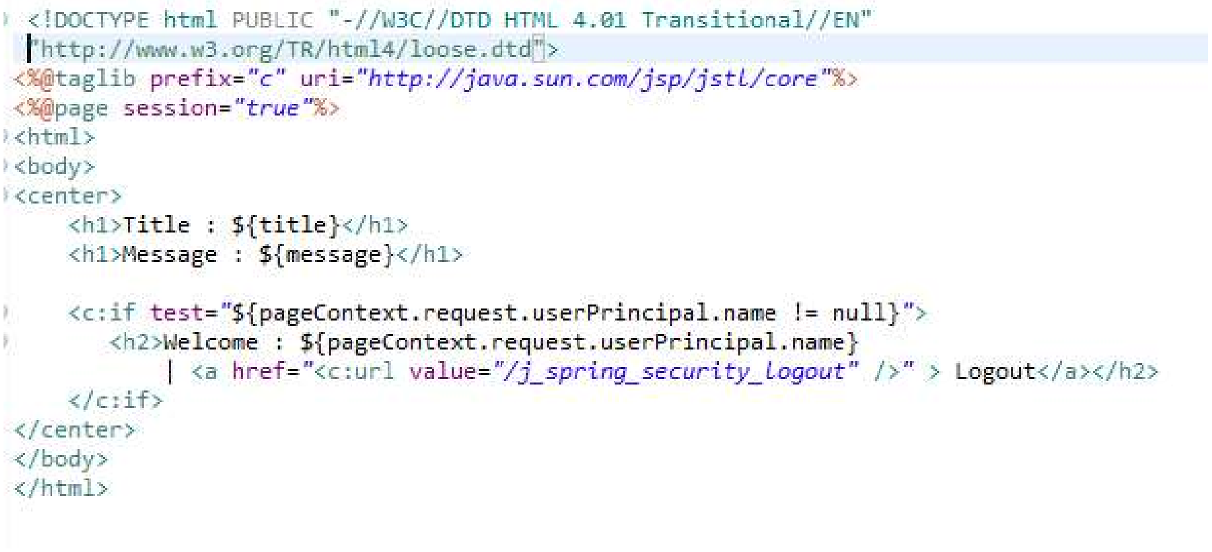
```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class MyController {

    @RequestMapping(value = { "/"} )
    public ModelAndView welcomePage() {

        ModelAndView model = new ModelAndView();
        model.addObject("title", "Spring Security Program Using XML");
        model.addObject("message", "This is welcome page!");
        model.setViewName("hello");
        return model;

    }

    @RequestMapping("/admin")
    public ModelAndView adminPage() {

        ModelAndView model = new ModelAndView();
        model.addObject("title", "Spring Security Program Using XML");
        model.addObject("message", "This is protected page!");
        model.setViewName("admin");

        return model;

    }

}
```

✓ **Step 3.**
Create admin.jsp page



```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<body>
<center>
    <h1>Title : ${title}</h1>
    <h1>Message : ${message}</h1>

       <h2>Welcome : ${pageContext.request.userPrincipal.name}</h2>

</center>
</body>
</html>
```

If session is required (So that you can use logout) admin.jsp page can be created like this



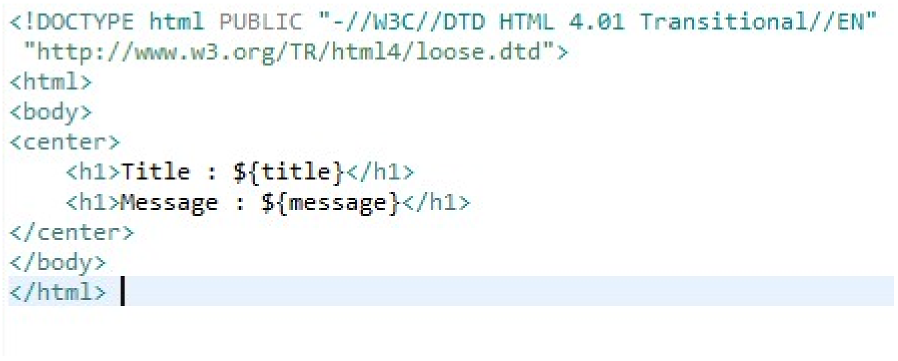
```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@page session="true"%>
<html>
<body>
<center>
    <h1>Title : ${title}</h1>
    <h1>Message : ${message}</h1>

    <c:if test="${pageContext.request.userPrincipal.name != null}">
       <h2>Welcome : ${pageContext.request.userPrincipal.name}
            | <a href="<c:url value="/j_spring_security_logout" />" > Logout</a></h2>
    </c:if>
</center>
</body>
</html>
```

The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page.

Where, the **uri** attribute value resolves to a location the container understands

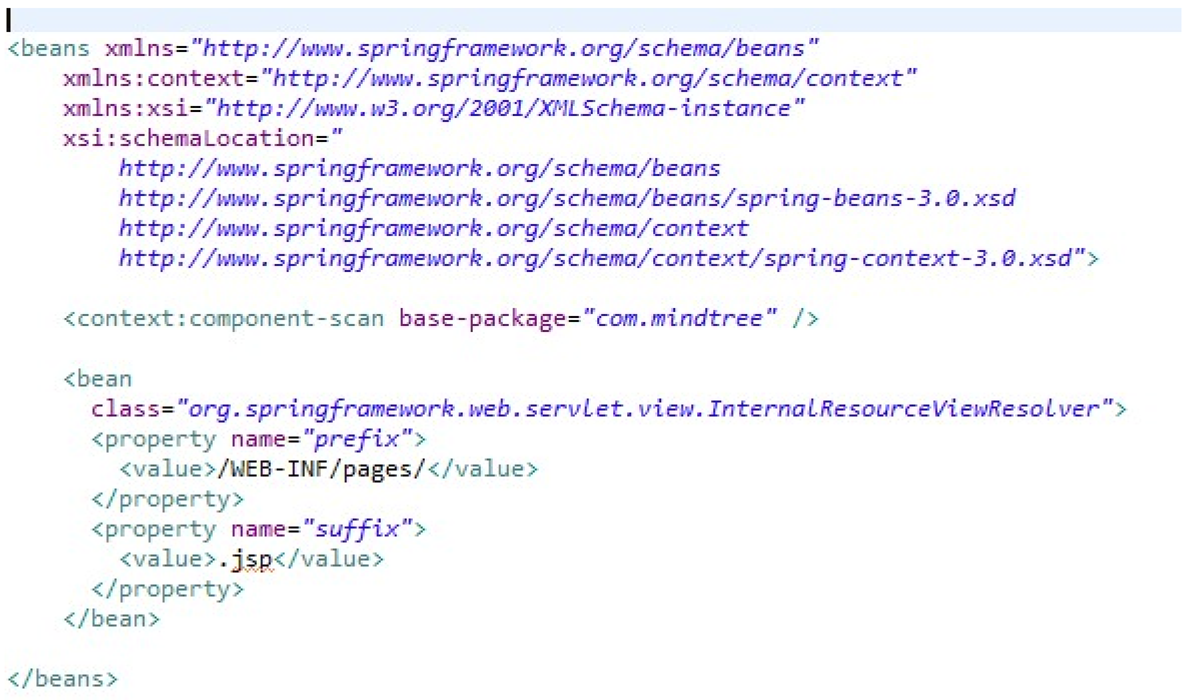
pageContext.request.userPrincipal.name is responsible to get login attributes from a jsp.

Create hello.jsp page



```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<body>
<center>
    <h1>Title : ${title}</h1>
    <h1>Message : ${message}</h1>
</center>
</body>
</html>
```

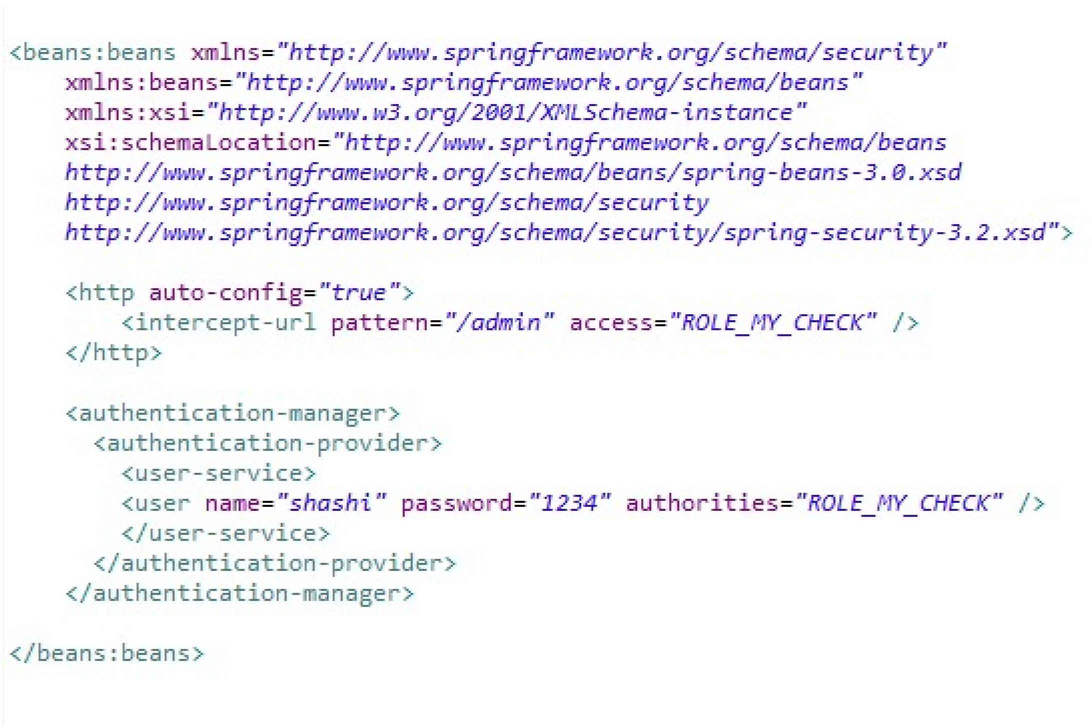
✓ **Step 4.**
Create mvc-dispatcher-servlet.xml



```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:component-scan base-package="com.mindtree" />

    <bean
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix">
        <value>/WEB-INF/pages/</value>
      </property>
      <property name="suffix">
        <value>.jsp</value>
      </property>
    </bean>

</beans>
```

InternalResourceViewResolver is used to resolve "internal resource view" (in simple, it's final output, jsp page) based on a predefined URL pattern. In additional, it allow you to add some predefined prefix or suffix to the view name (prefix + view name + suffix), and generate the final view page URL.

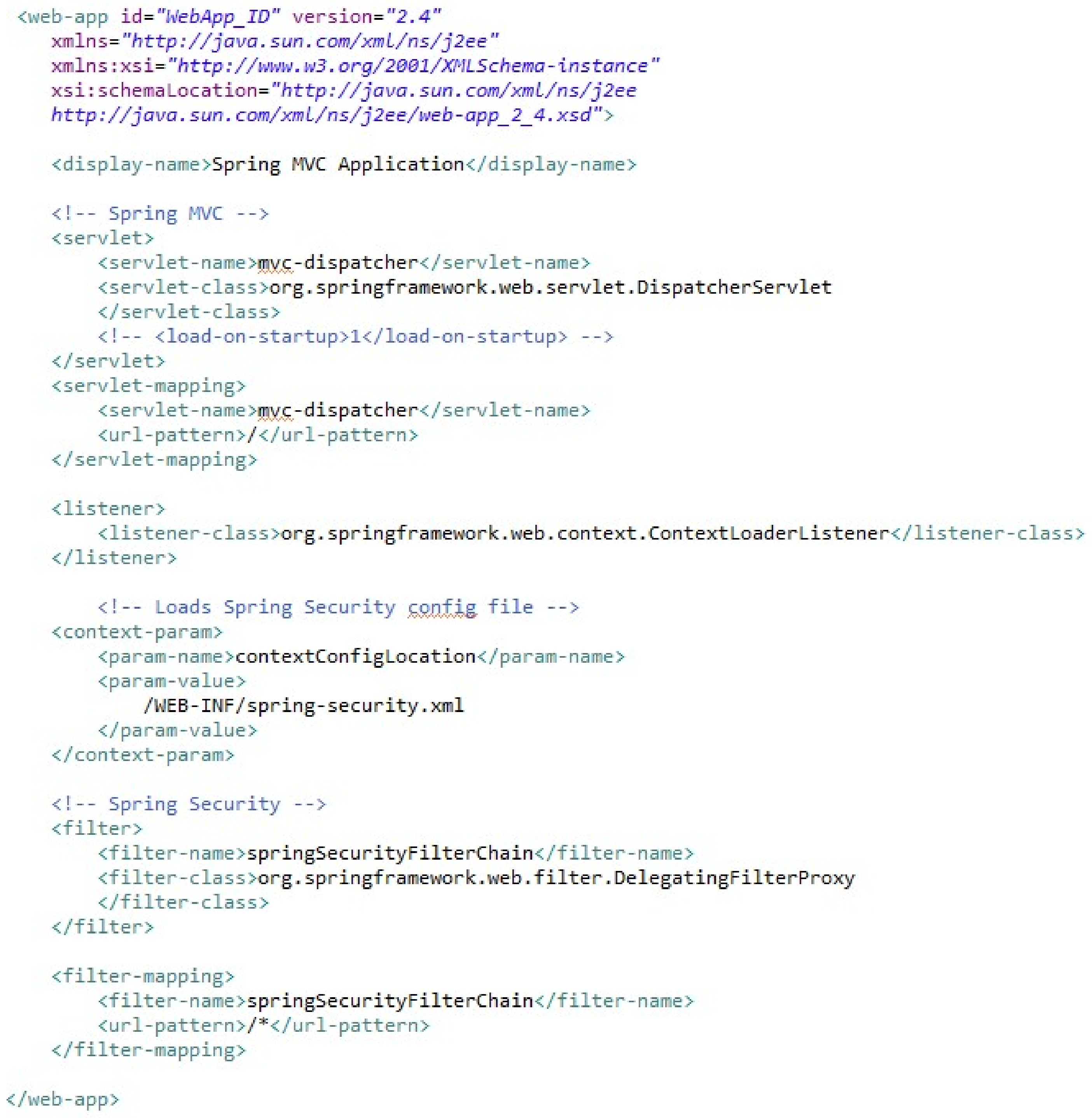
- ✓ **Step 5.**
  Create spring-security.xml



```xml
<beans:beans xmlns="http://www.springframework.org/schema/security"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security-3.2.xsd">

    <http auto-config="true">
        <intercept-url pattern="/admin" access="ROLE_MY_CHECK" />
    </http>

    <authentication-manager>
      <authentication-provider>
        <user-service>
        <user name="shashi" password="1234" authorities="ROLE_MY_CHECK" />
        </user-service>
      </authentication-provider>
    </authentication-manager>

</beans:beans>
```

Access name can be any thing but it should match with authorities. When intercept-url matches with pattern a login page will popup for authentication.

✓ **Step 6.**
Configure web.xml file



```
<web-app id="WebApp_ID" version="2.4"
   xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

   <display-name>Spring MVC Application</display-name>

   <!-- Spring MVC -->
   <servlet>
       <servlet-name>mvc-dispatcher</servlet-name>
       <servlet-class>org.springframework.web.servlet.DispatcherServlet
       </servlet-class>
       <!-- <load-on-startup>1</load-on-startup> -->
   </servlet>
   <servlet-mapping>
       <servlet-name>mvc-dispatcher</servlet-name>
       <url-pattern>/</url-pattern>
   </servlet-mapping>

   <listener>
       <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
   </listener>

       <!-- Loads Spring Security config file -->
   <context-param>
       <param-name>contextConfigLocation</param-name>
       <param-value>
           /WEB-INF/spring-security.xml
       </param-value>
   </context-param>

   <!-- Spring Security -->
   <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>

</web-app>
```

The Spring security filter chain is a very complex and flexible engine.

Key filters in the chain are-

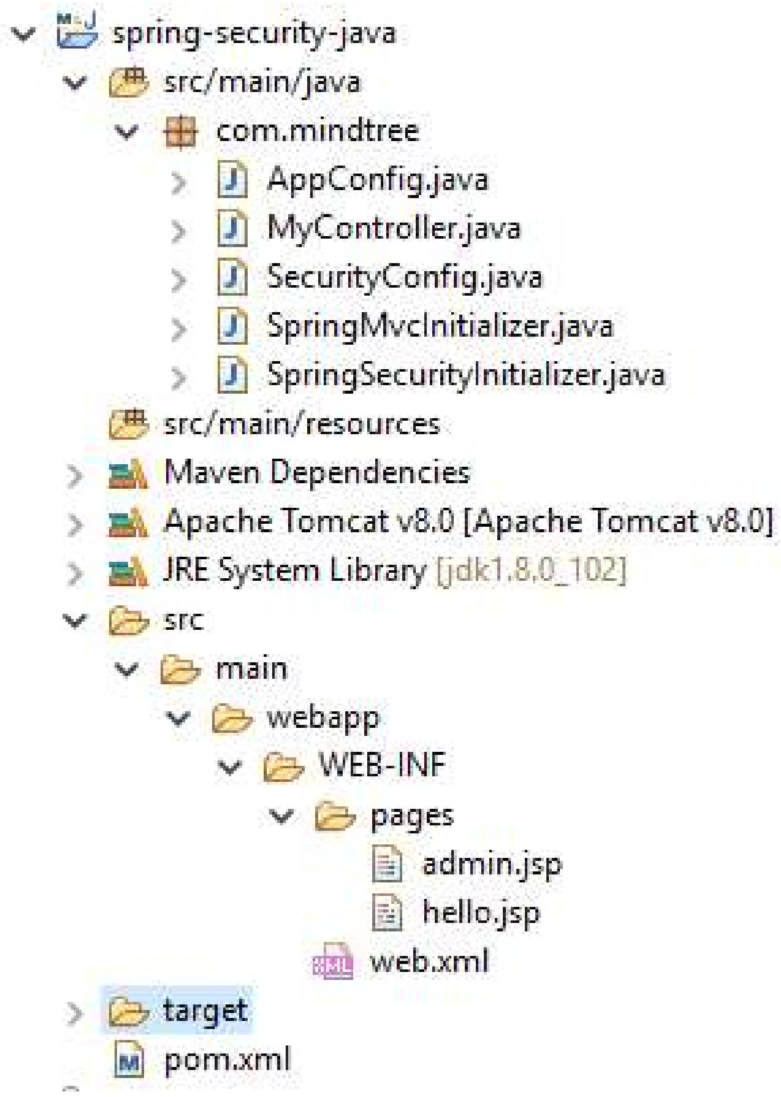
performs authentication

ExceptionTranslationFilter (catch security exceptions from FilterSecurityInterceptor)

FilterSecurityInterceptor (may throw authentication and authorization exceptions)

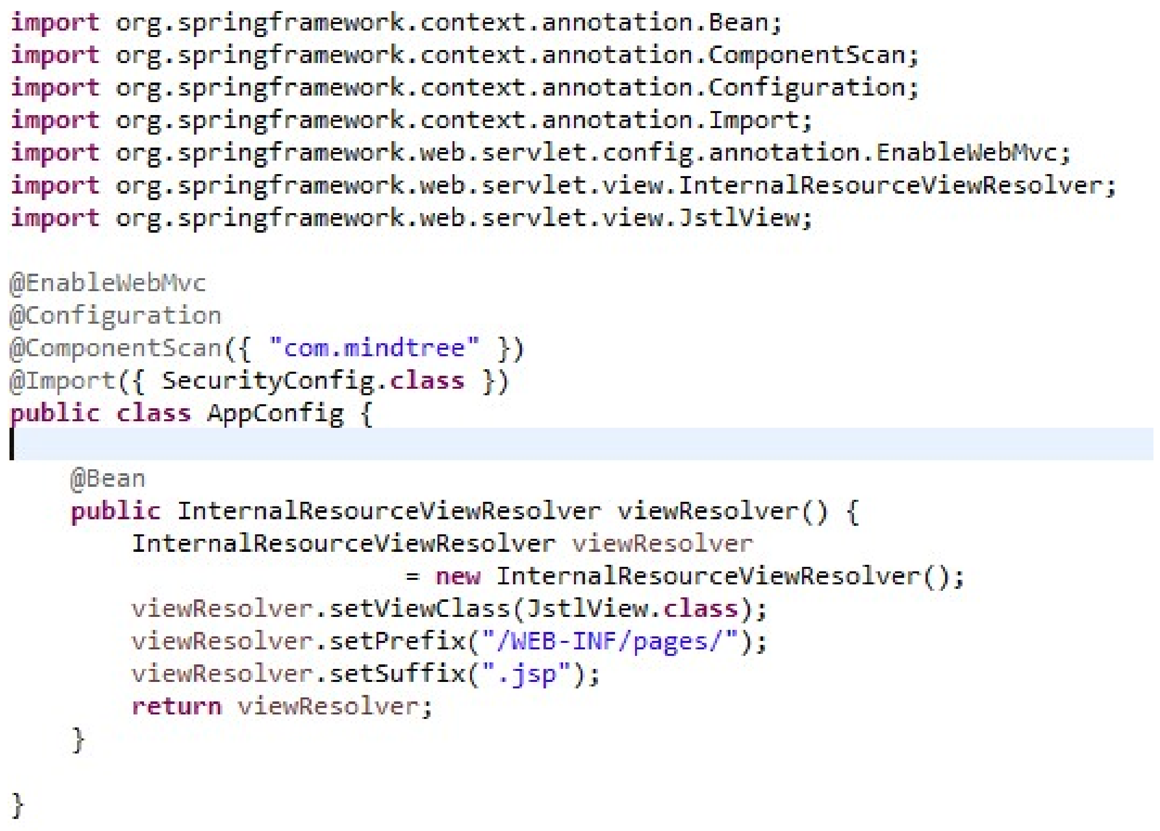
Simple Login Page Using Java Configuration

➢ **Folder Structure**



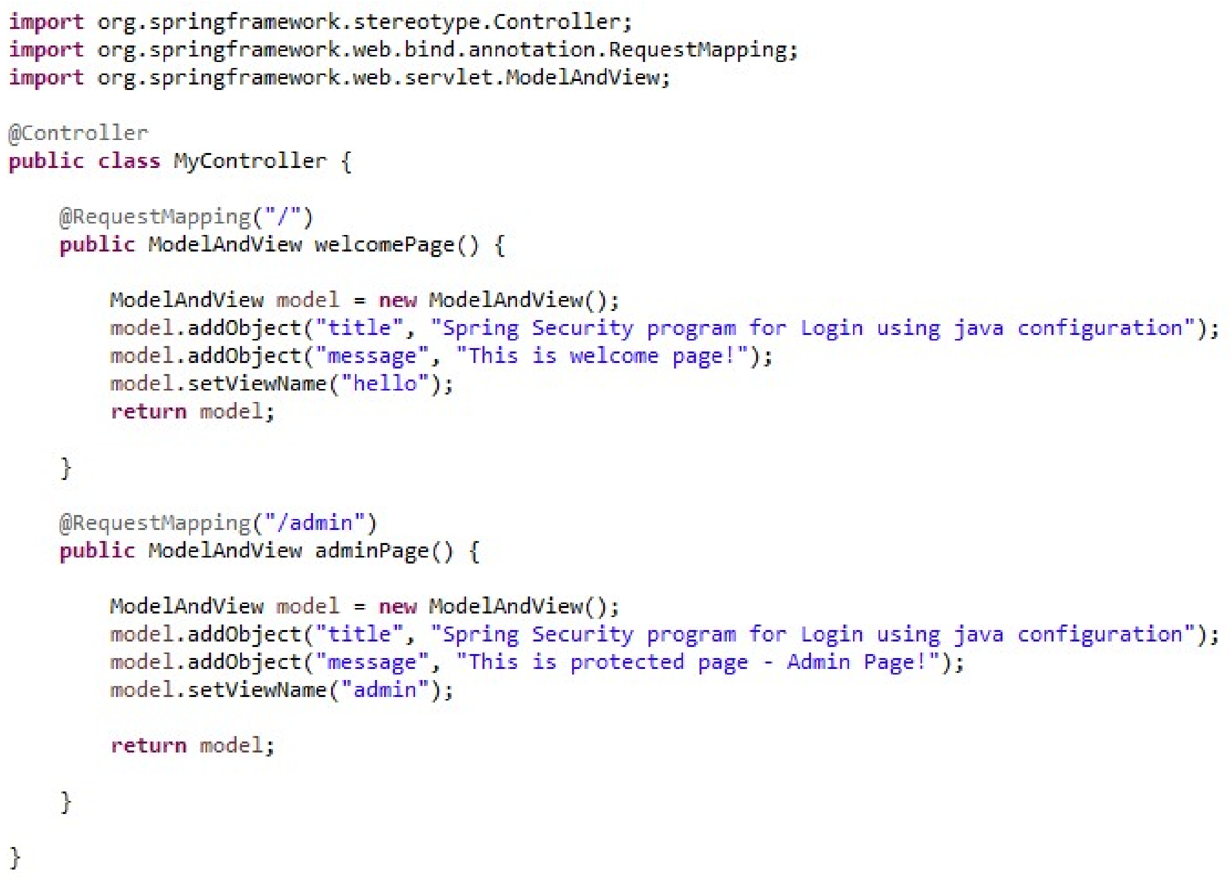
```
spring-security-java
  src/main/java
    com.mindtree
      AppConfig.java
      MyController.java
      SecurityConfig.java
      SpringMvcInitializer.java
      SpringSecurityInitializer.java
  src/main/resources
  Maven Dependencies
  Apache Tomcat v8.0 [Apache Tomcat v8.0]
  JRE System Library [jdk1.8.0_102]
  src
    main
      webapp
        WEB-INF
          pages
            admin.jsp
            hello.jsp
          web.xml
  target
  pom.xml
```

- ✓ **Step 1.**
  Create AppConfig.java



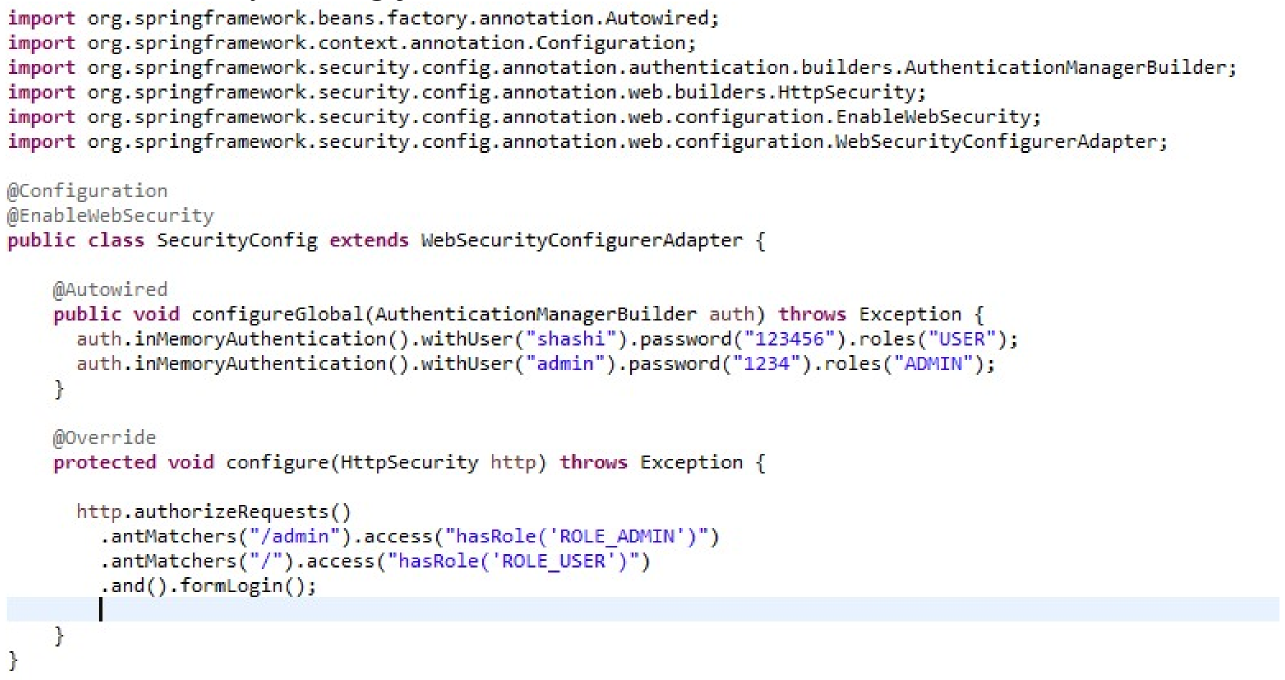
```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;
import org.springframework.web.servlet.view.JstlView;

@EnableWebMvc
@Configuration
@ComponentScan({ "com.mindtree" })
@Import({ SecurityConfig.class })
public class AppConfig {

    @Bean
    public InternalResourceViewResolver viewResolver() {
        InternalResourceViewResolver viewResolver
                          = new InternalResourceViewResolver();
        viewResolver.setViewClass(JstlView.class);
        viewResolver.setPrefix("/WEB-INF/pages/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }

}
```

✓ **Step 2.**
Create MyController.java



```java
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class MyController {

    @RequestMapping("/")
    public ModelAndView welcomePage() {

        ModelAndView model = new ModelAndView();
        model.addObject("title", "Spring Security program for Login using java configuration");
        model.addObject("message", "This is welcome page!");
        model.setViewName("hello");
        return model;

    }

    @RequestMapping("/admin")
    public ModelAndView adminPage() {

        ModelAndView model = new ModelAndView();
        model.addObject("title", "Spring Security program for Login using java configuration");
        model.addObject("message", "This is protected page - Admin Page!");
        model.setViewName("admin");

        return model;

    }

}
```

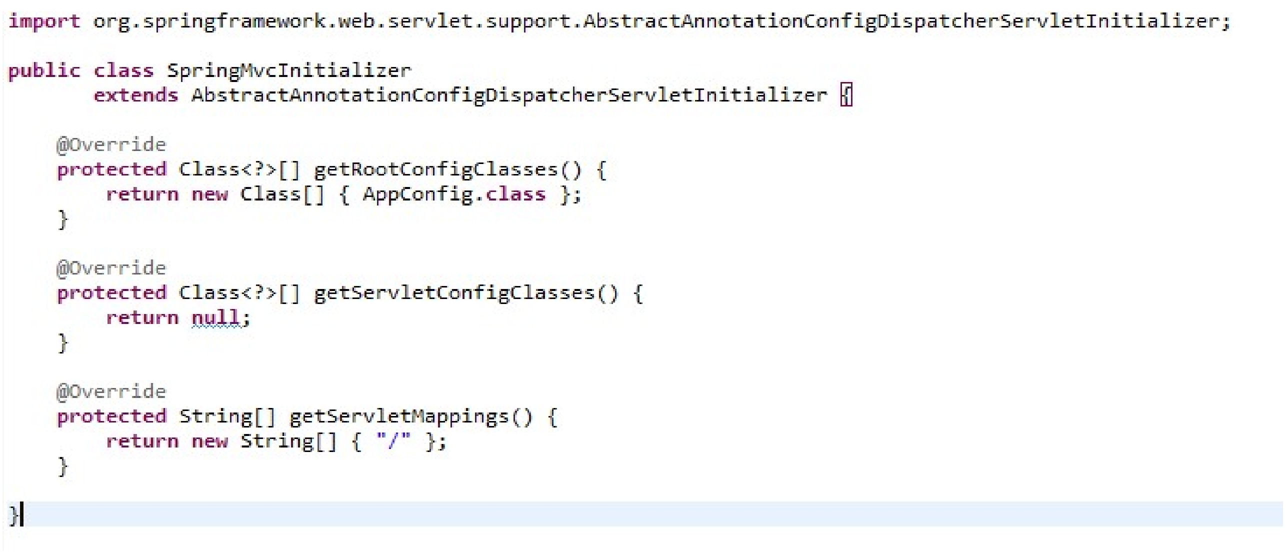
✓ **Step 3.**
Create SecurityConfig.java



```java
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 SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
      auth.inMemoryAuthentication().withUser("shashi").password("123456").roles("USER");
      auth.inMemoryAuthentication().withUser("admin").password("1234").roles("ADMIN");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {

      http.authorizeRequests()
        .antMatchers("/admin").access("hasRole('ROLE_ADMIN')")
        .antMatchers("/").access("hasRole('ROLE_USER')")
        .and().formLogin();

    }
}
```

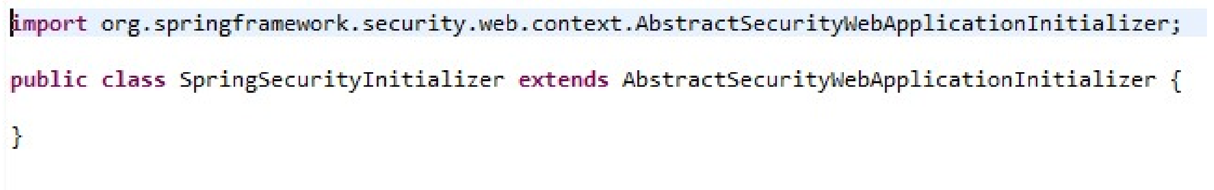
✓ **Step 4.**

Create SpringMvcInitializer.java



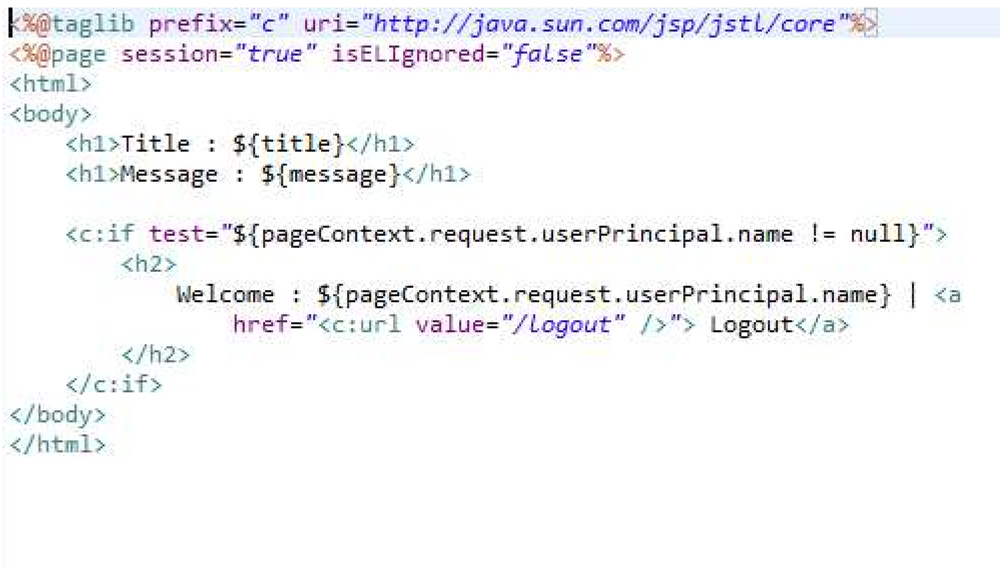
```java
import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class SpringMvcInitializer
       extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { AppConfig.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

}
```

✓ **Step 5.**
Create SpringSecurityInitializer.java



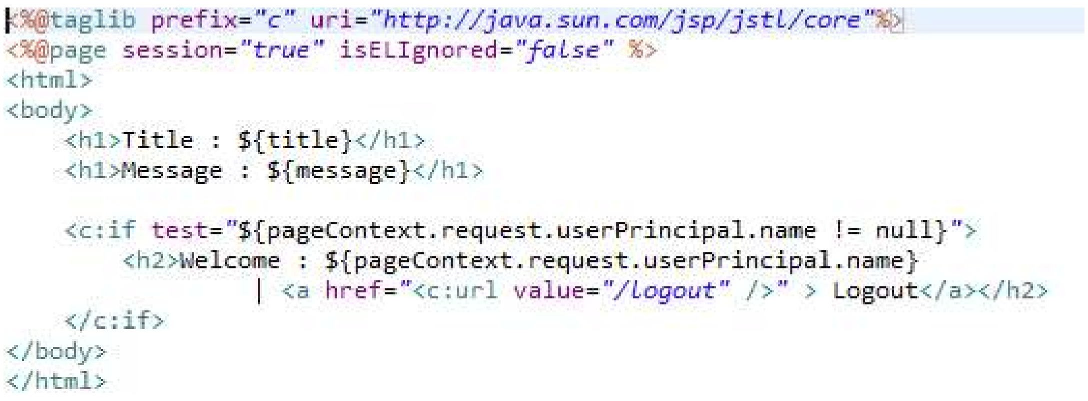
```java
import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;

public class SpringSecurityInitializer extends AbstractSecurityWebApplicationInitializer {

}
```

✓ **Step 6.**
Create hello.jsp page



```jsp
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@page session="true" isELIgnored="false"%>
<html>
<body>
    <h1>Title : ${title}</h1>
    <h1>Message : ${message}</h1>

    <c:if test="${pageContext.request.userPrincipal.name != null}">
        <h2>
            Welcome : ${pageContext.request.userPrincipal.name} | <a
                href="<c:url value="/logout" />"> Logout</a>
        </h2>
    </c:if>
</body>
</html>
```

Create admin.jsp page



```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@page session="true" isELIgnored="false" %>
<html>
<body>
    <h1>Title : ${title}</h1>
    <h1>Message : ${message}</h1>

    <c:if test="${pageContext.request.userPrincipal.name != null}">
        <h2>Welcome : ${pageContext.request.userPrincipal.name}
                   | <a href="<c:url value="/logout" />" > Logout</a></h2>
    </c:if>
</body>
</html>
```