# Version History

|  |  |  |
| --- | --- | --- |
| Version | Date | Release Note |
| V0.1 | 10-Mar-2020 | Initial Draft |
| V1.0 | 2-Apr-2020 | 1st Release |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

index

1 Version History 2

2 Exercise 1 – Form Submission 4

2.1 Form Submission Exercise 4

3 Exercise 1 – bonus 13

# Exercise 1 – Form Submission

The exercise will demostrate how to use spring security module to protect a Spring Application

Prerequisite

Java installed (e.g. JDK 11)

Maven installed (e.g. version 3.6.3)

Spring Tool Suite (STS) (e.g. 4.5.1)

## Form Submission Exercise

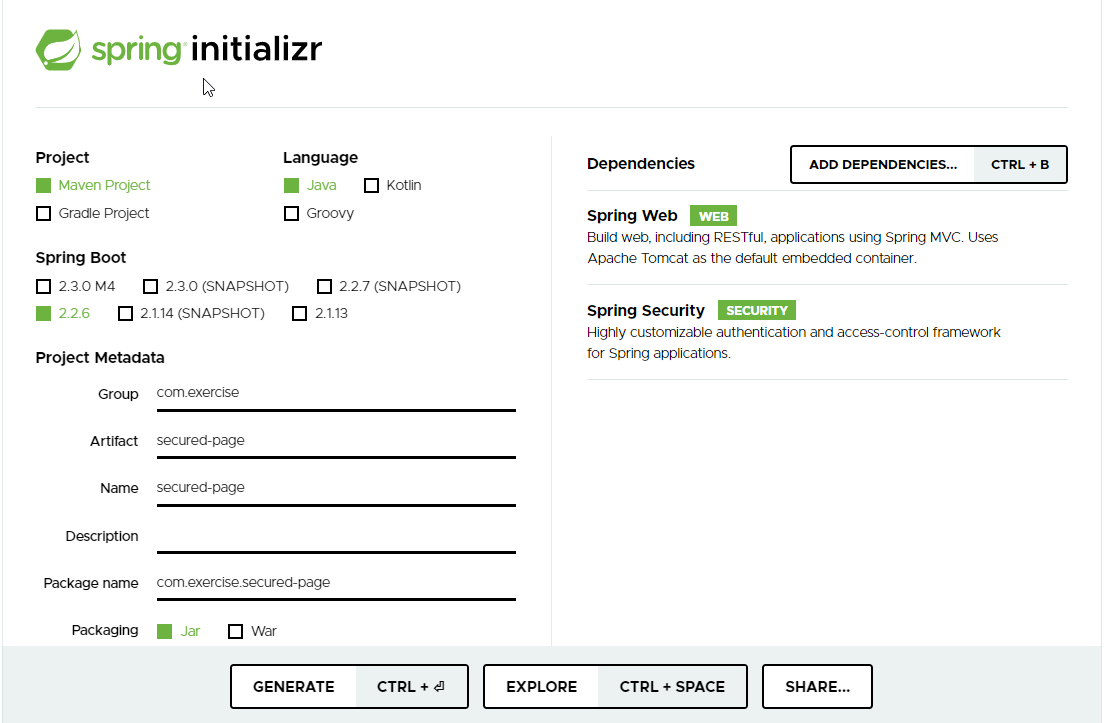
### Generate sample pom.xml

. Goto <https://start.spring.io> and input the following fields:

Group: com.exercise

Artifact: secured-page

Dependencies: Spring Web, Spring Security



Or you may use this one:



2. Click “Generate” and extract the zipped file to your local directory, e.g. C:\java-ex:

C:.

├───.mvn

│ └───wrapper

└───src

├───main

│ ├───java

│ │ └───com

│ │ └───exercise

│ │ └───securedpage

│ └───resources

│ ├───static

│ └───templates

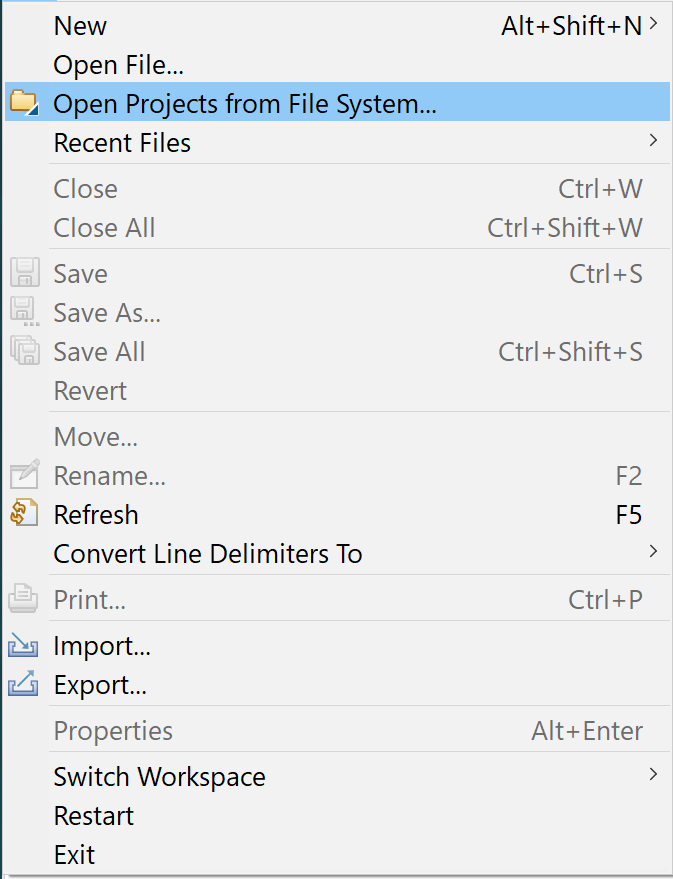
└───test

└───java

└───com

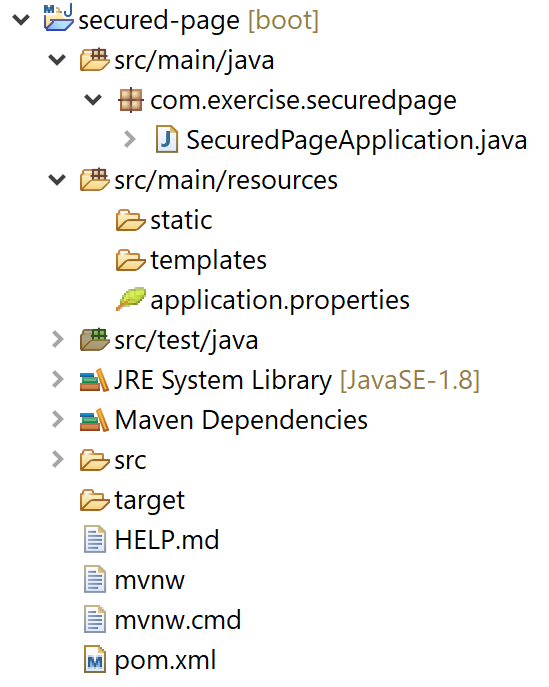
└───exercise

└───securedpage

3. Start STS, click “File” and select “Open Projects from File System…”:

4. Import the project “secured-page” into STS

5. The project structure should be shown:



6. In order to support JSP, we need to add dependency to the project. Now open the file “pom.xml”, and insert the following code to the <dependencies> tag.

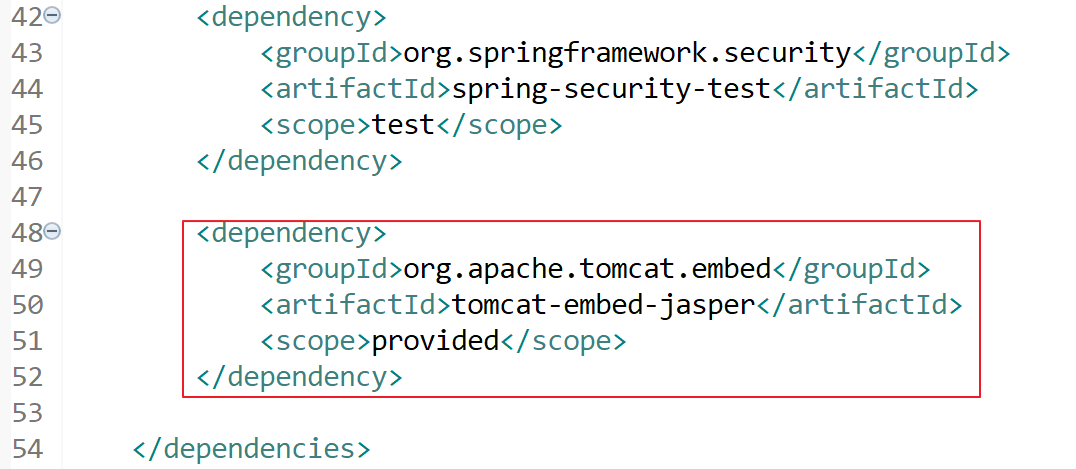
<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

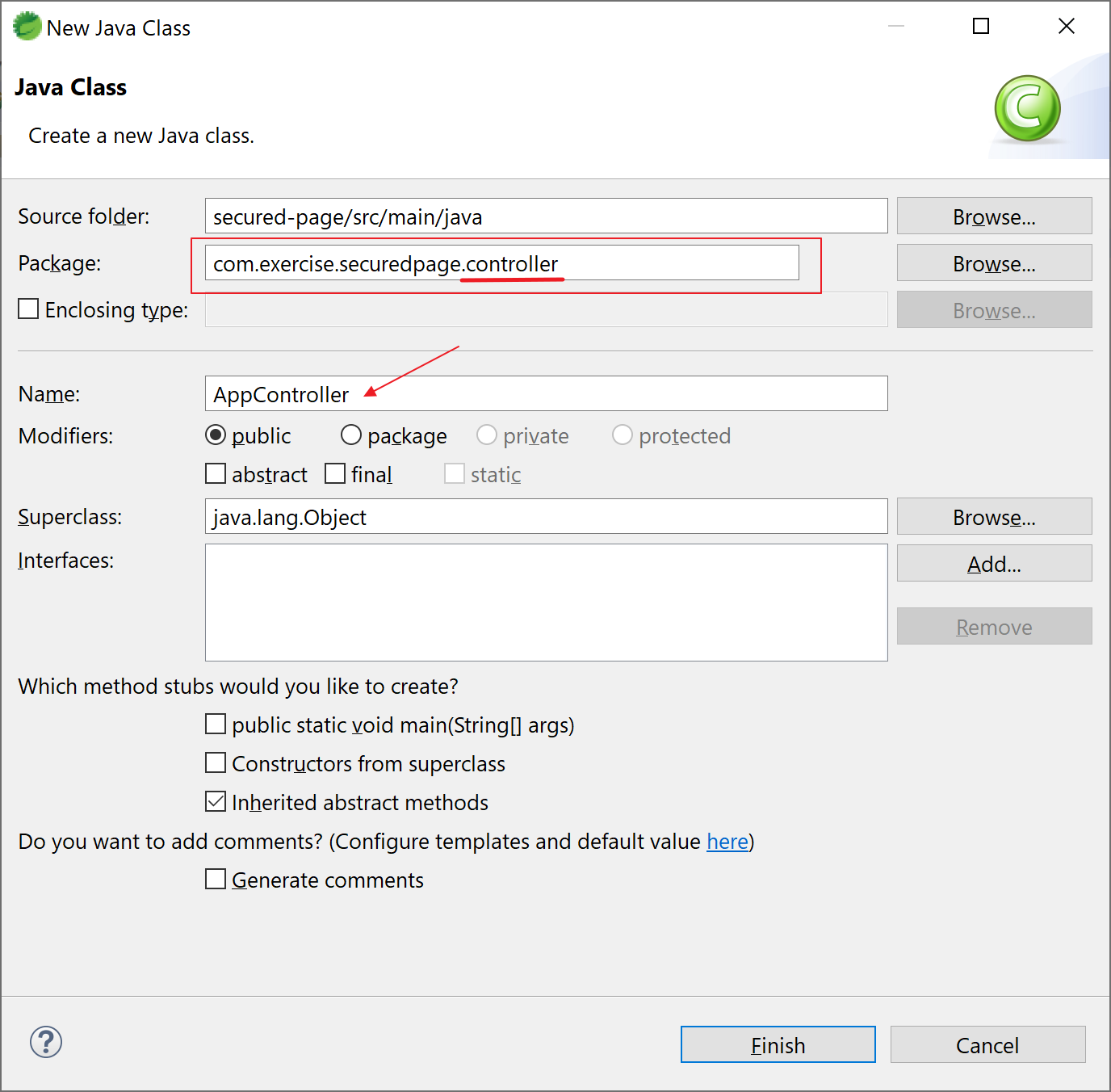


7. Open application.properties (.\src\main\resources\application.properties) to configure the MVC’s view. Add the following code to the file:

spring.mvc.view.prefix=/WEB-INF/jsp/

spring.mvc.view.suffix=.jsp

8. Create the Controller class “AppController”, right the package “com.exercise.securedpage”, select “New” and then click “Class”. Add a class “AppController” under the package “com.exercise.securedpage.controller”:



9. Open the file “AppController.java” and insert the following code:

package com.exercise.securedpage.controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

@Controller

public class AppController {

@GetMapping("/public/welcome")

public String getWelcome() {

return "welcome";

}

@GetMapping("/admin/secret")

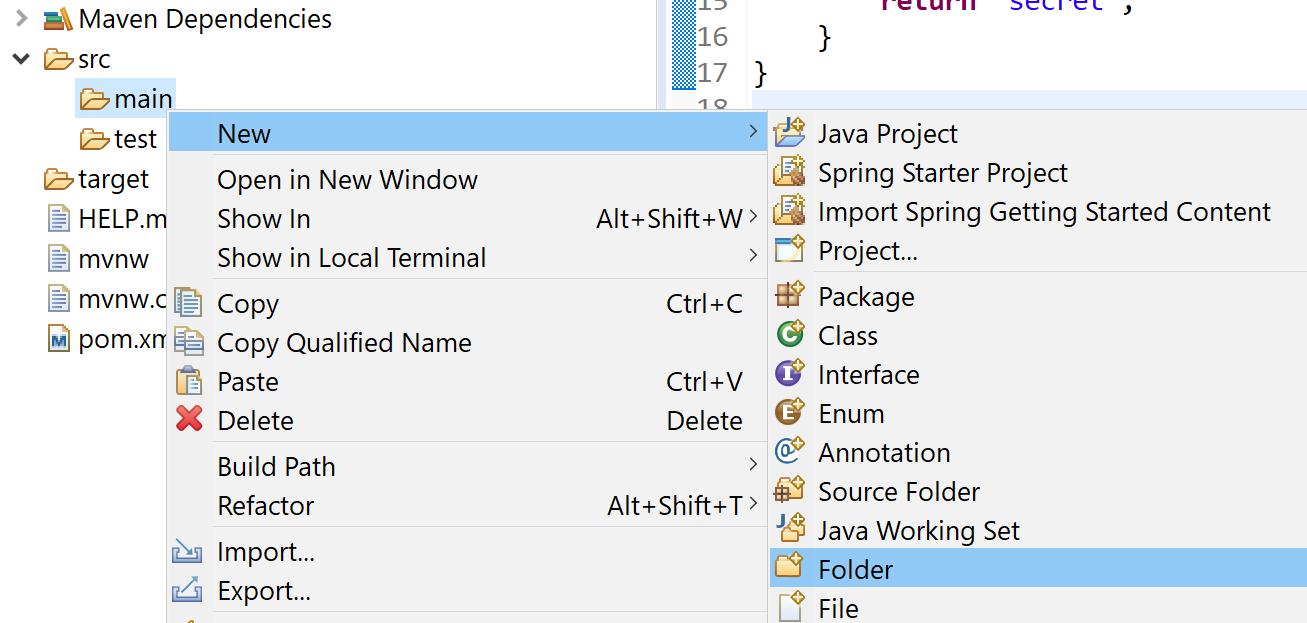
public String getSecret() {

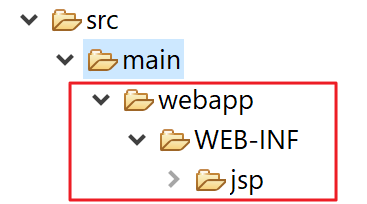
return "secret";

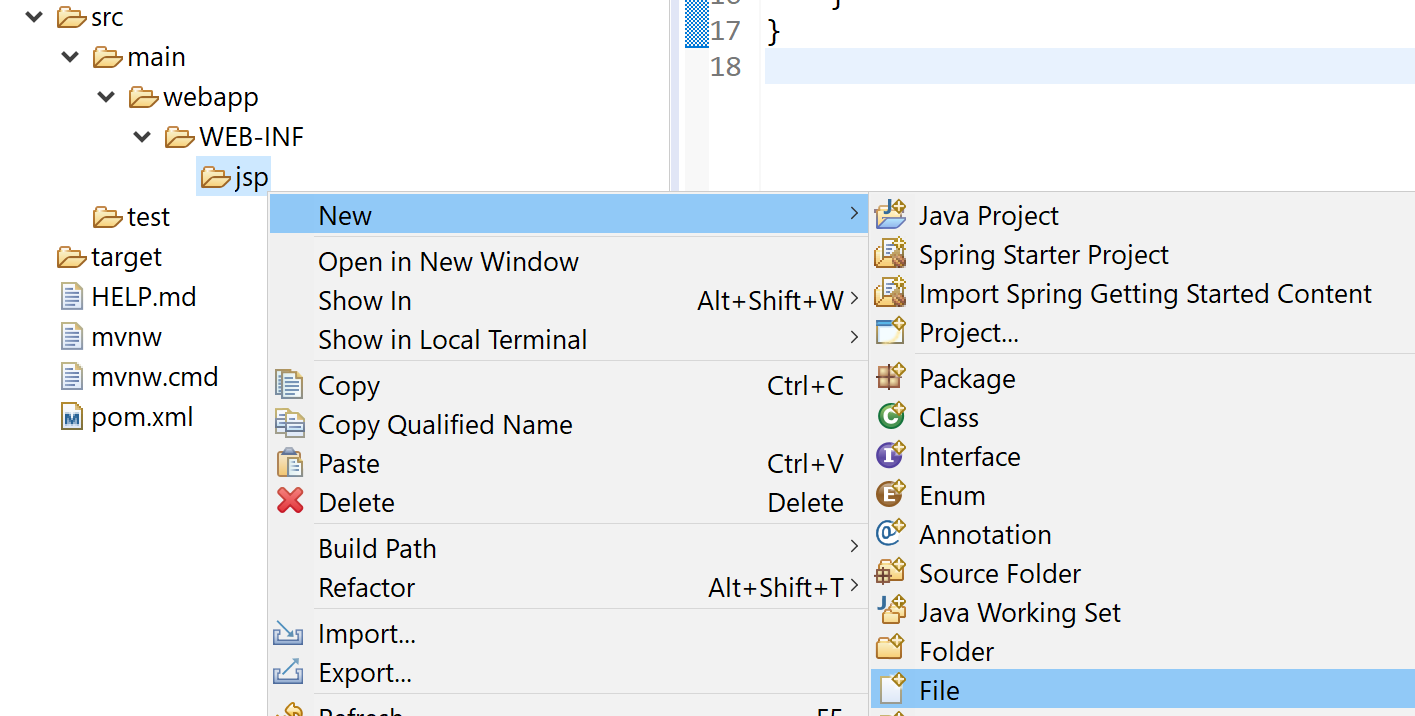
}

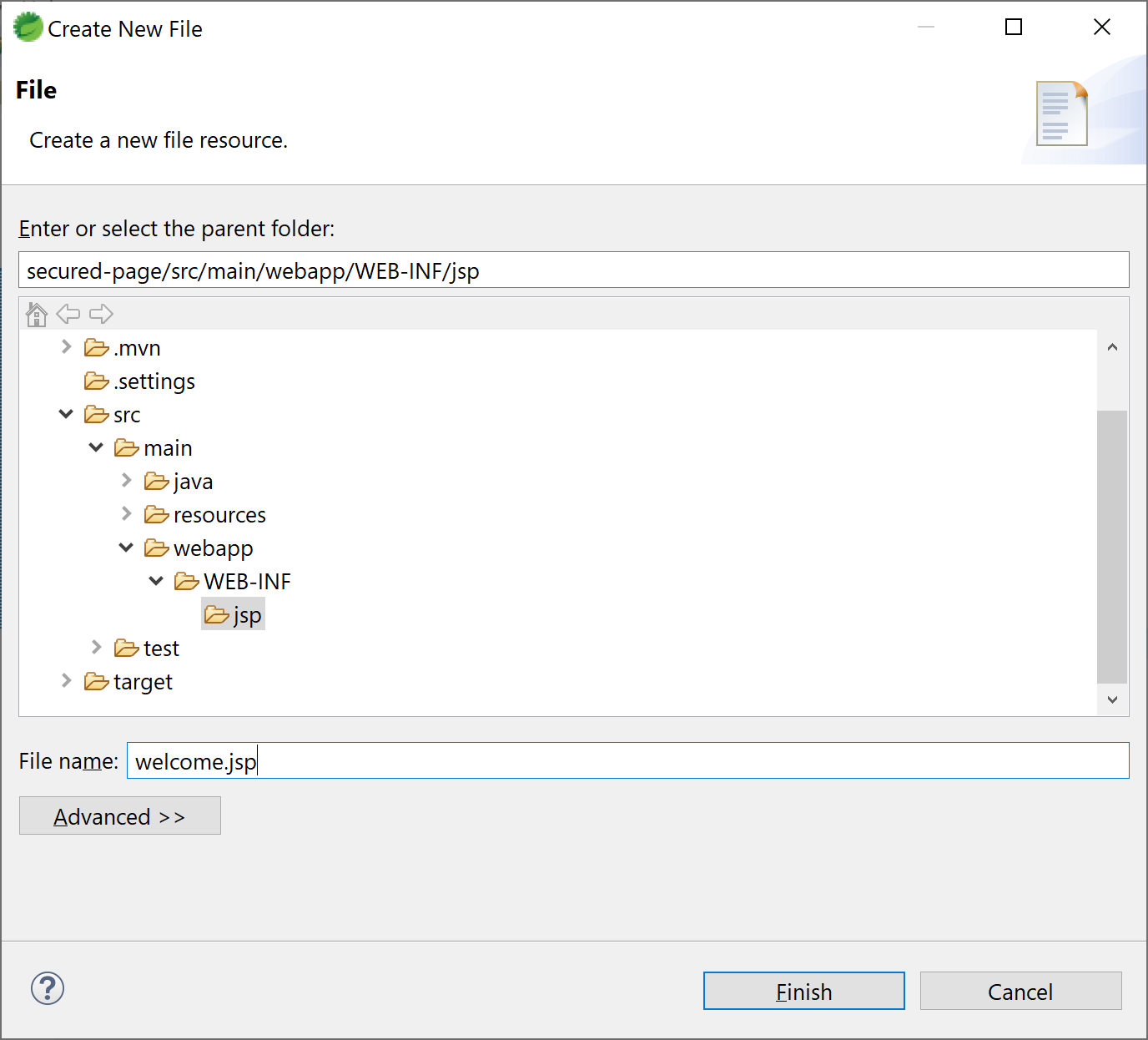
}

10. Implementing the views. Create a new folder “webapp” under “./src/main/”:



Repeat the steps to create the following structure (src/main/webapp/WEB-INF/jsp/):

10a. Create the file welcome.jsp under “src/main/webapp/WEB-INF/jsp/”:



Open the welcome.jsp and insert the following code:

<html>

<body>

<h2>Welcome - public page</h2>

</body>

</html>

10b. Repeat the step to create the file secret.jsp under “src/main/webapp/WEB-INF/jsp/”:

<html>

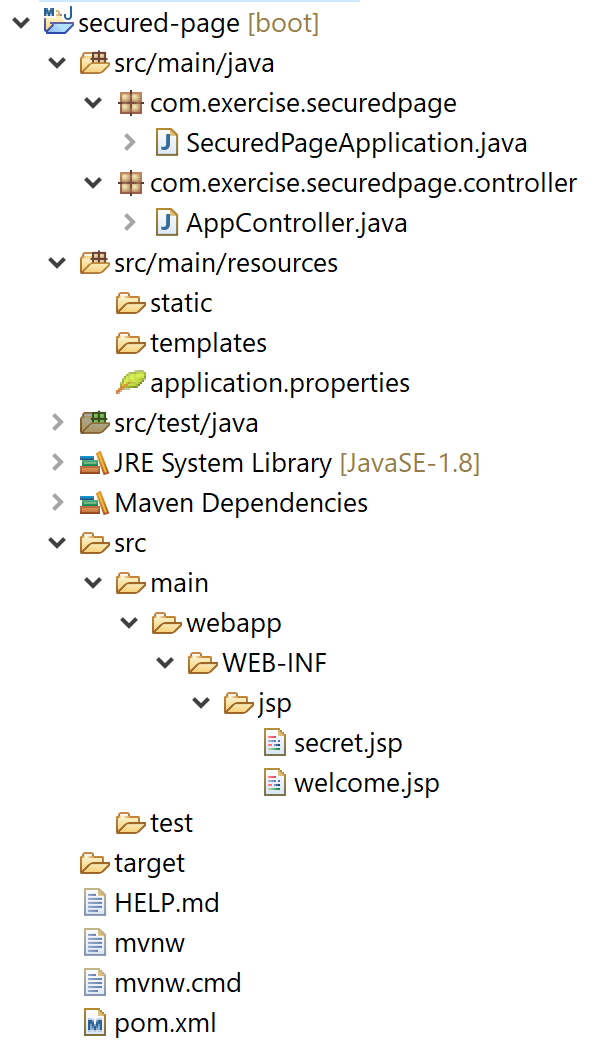
<body>

<h2>Secret - secured page</h2>

</body>

</html>

11. The final project look like:



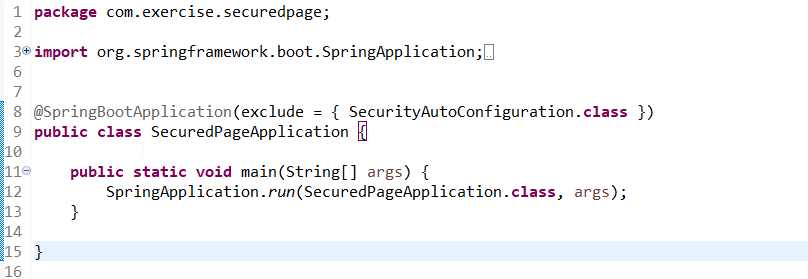
12. Run the application and try to visit the 2 pages:

<http://localhost:8080/public/welcome>

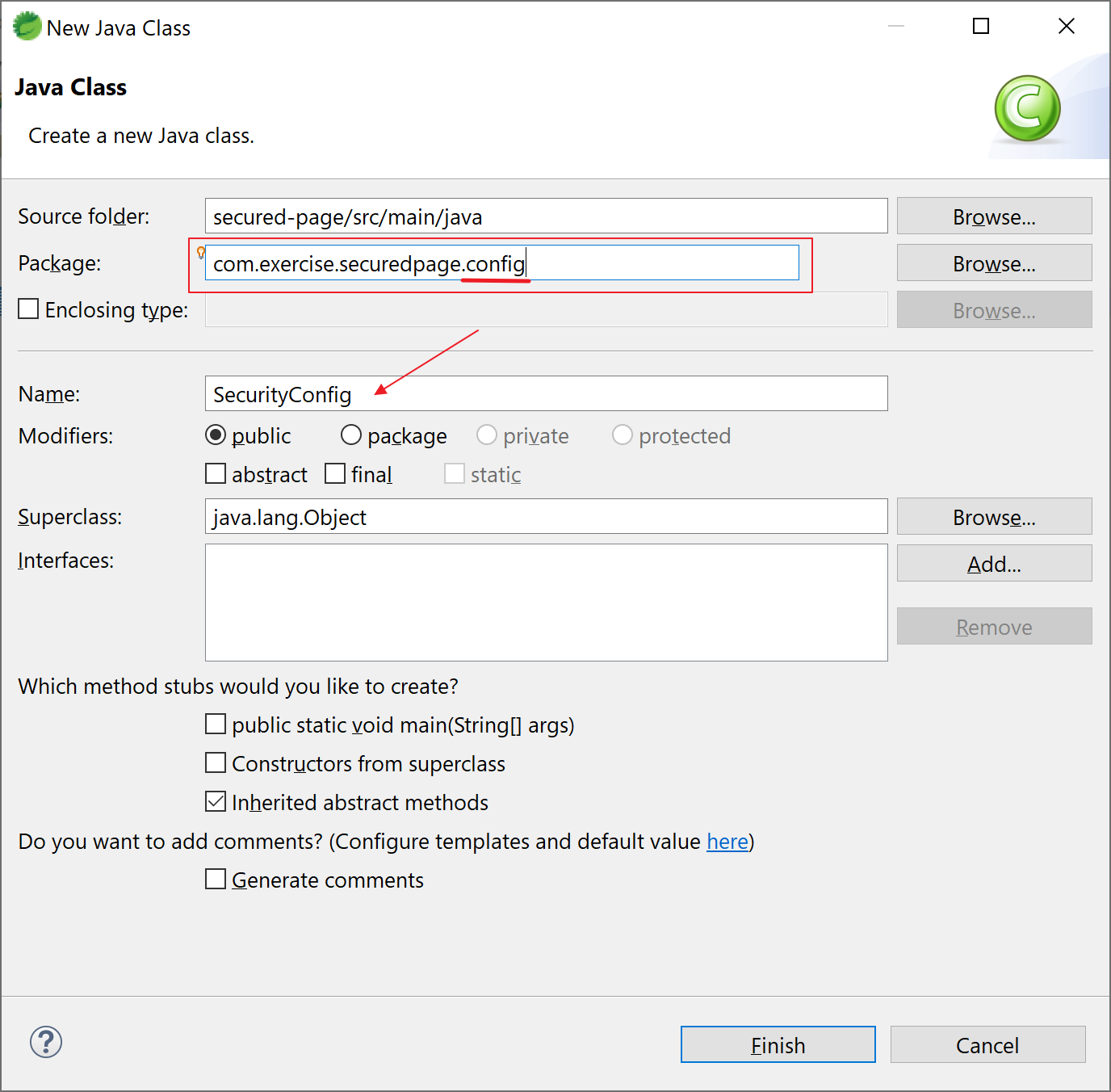
<http://localhost:8080/admin/secret>

Can you go to these pages? Why?

By default, the auto Configuration in Spring Security module will enabe the authentication for all urls. We can disable it by exclude the auto-configuration

@SpringBootApplication(exclude = { SecurityAutoConfiguration.class })in the main class

13. Configure the Spring Security. Create a class “SecurityConfig” under package “com.exercise.securedpage.config”:



Open the file and insert the following code:

package com.exercise.securedpage.config;

import org.springframework.beans.factory.annotation.Autowired;

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;

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests()

.antMatchers("/admin/\*\*").hasRole("ADMIN")

.antMatchers("/public/\*\*").permitAll();

}

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication()

.withUser("admin").password("{noop}password").roles("ADMIN").and()

.withUser("user").password("{noop}password").roles("USER");

}

}

Note:

1. URL with pattern /admin/\*\* needs role ADMIN to access
2. URL with pattern /public/\*\* is allowed for everyone
3. user “admin” with password “password” has role “ADMIN”
4. user “user” with password “password” has role “USER”
5. {noop} is used for plain password (no encryption)

14. Restart the application and visit the 2 pages again:

<http://localhost:8080/public/welcome>

<http://localhost:8080/admin/secret>

Can you visit the secret page? Why? What’s the HTTP status?

15. Configure to access the secured page. Open “SecurityConfig.java” and add the following code in configure(HttpSecurity http):

.and()

.formLogin();



Note: Spring boot will automatically create the login form for you.

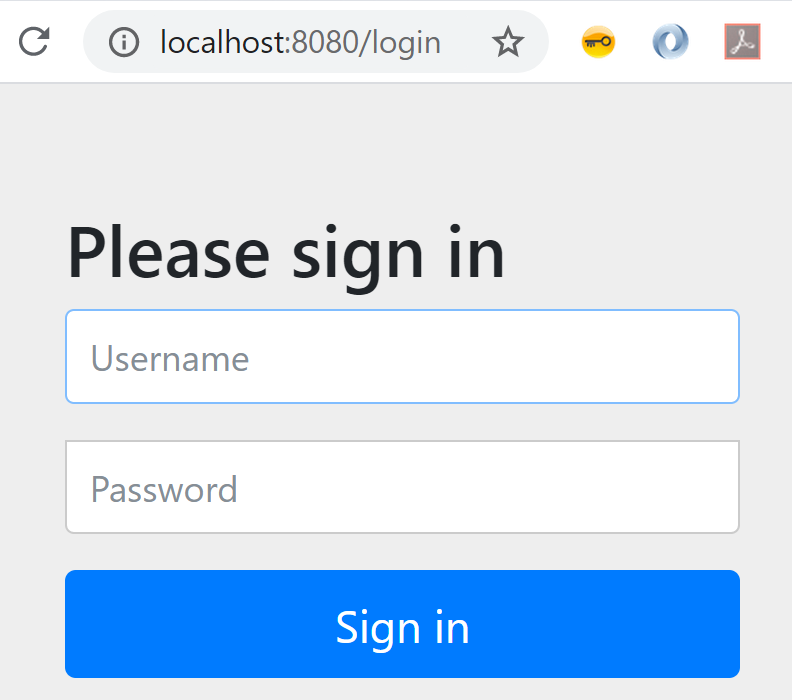
16. Restart the application and visit the 2 pages again:

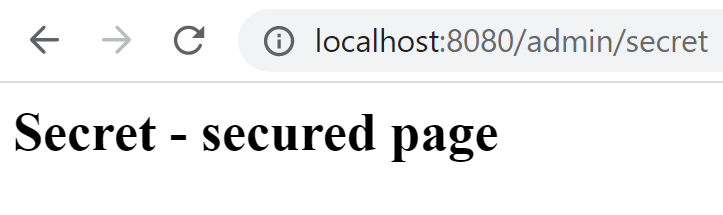
<http://localhost:8080/public/welcome>



<http://localhost:8080/admin/secret>

input Username: admin and Password: password, then click Sign in.





17. Congratulations. Try this one if you have problem with this exercise:



# Exercise 1 – bonus

Prerequisites:

Java installed (e.g. JDK 1.8)

Maven installed (e.g. version 3.6.3)

Spring Tool Suite (STS) (e.g. 4.5.1)

Exercise 1 is finished

1. Create your own login page. Create a JSP file “login.jsp” under “./webapp/WEB-INF/jsp”, and insert the following code:

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

<html>

<body onload='document.loginForm.username.focus();'>

<h1>Login Form</h1>

<c:if test="${not empty errorMessge}"><div style="color:red; font-weight: bold; margin: 30px 0px;">${errorMessge}</div></c:if>

<form name='login' action="/login" method='POST'>

<table>

<tr>

<td>UserName:</td>

<td><input type='text' name='username' value=''></td>

</tr>

<tr>

<td>Password:</td>

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

</tr>

<tr>

<td colspan='2'><input name="submit" type="submit" value="submit" /></td>

</tr>

</table>

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

</form>

</body>

</html>

2. As we use JSTL in the JSP, we need to add the dependency. Open pom.xml and add the following dependency:

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>



3. Configure Spring Security to use your own login.jsp. Open SecurityConfig.java and add the following code:

.loginPage("/login")

.failureUrl("/login?error=true");



4. The login page is now set to “/login”, let’s define “/login” in the controller. Open “AppController.java” and add the following code:

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

@GetMapping(value = "/login")

public String loginPage(@RequestParam(value = "error", required = false) String error,

@RequestParam(value = "logout", required = false) String logout,

ModelMap model) {

String errorMessge = null;

if(error != null) {

errorMessge = "Username or Password is incorrect !!";

}

if(logout != null) {

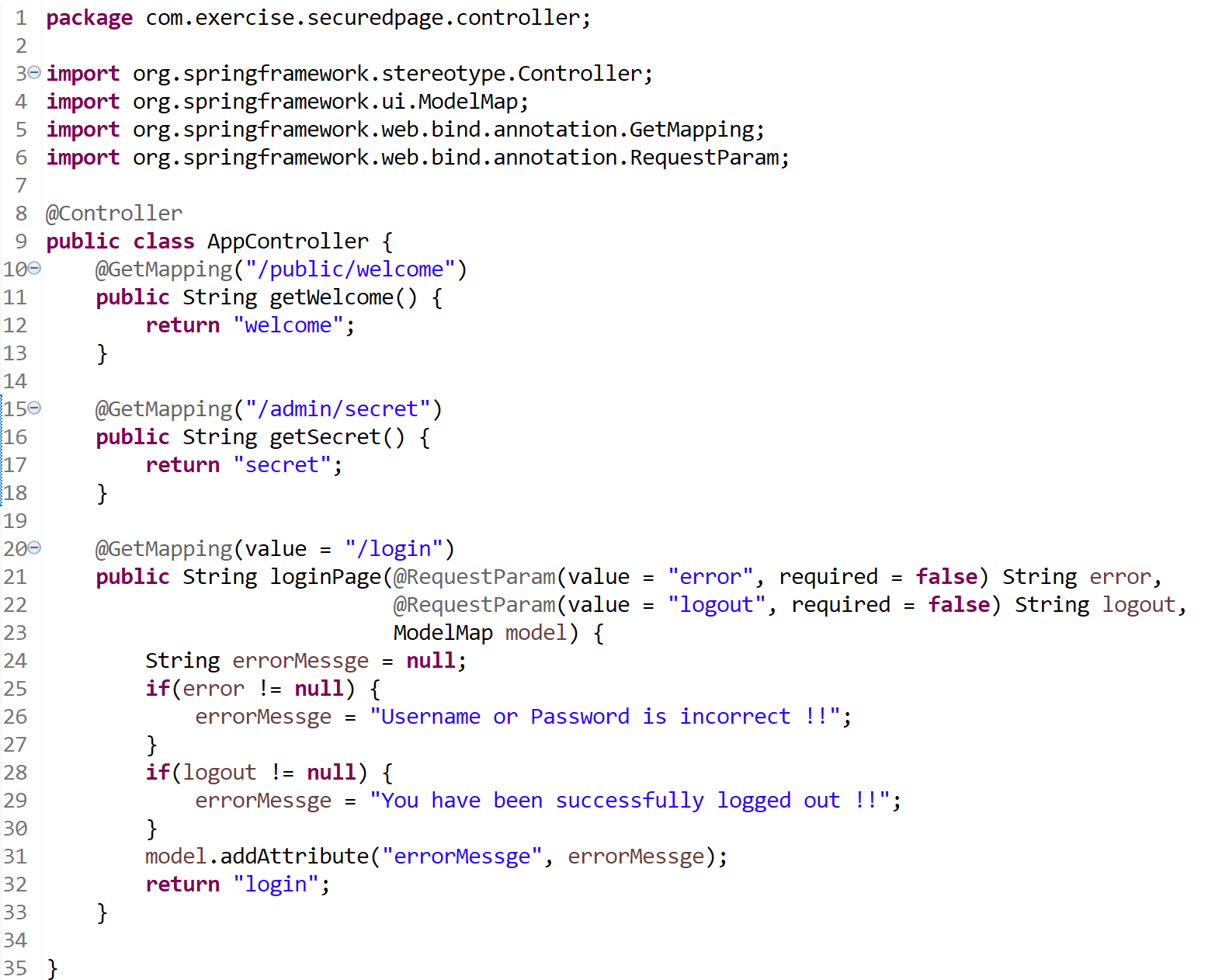
errorMessge = "You have been successfully logged out !!";

}

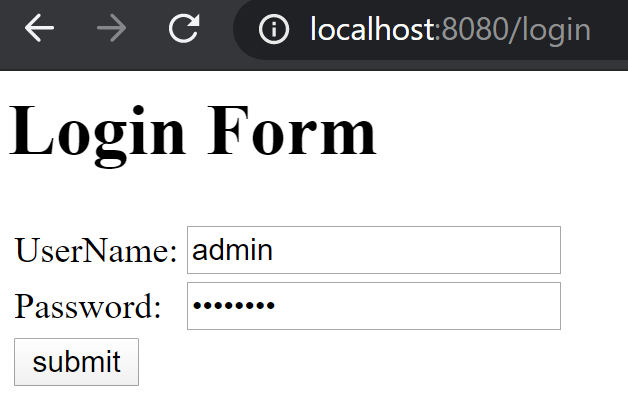
model.addAttribute("errorMessge", errorMessge);

return "login";

}



5. Restart the application and visit

<http://localhost:8080/admin/secret>

6. Logout. Open SecurityConfig.java and add the following:

.and()

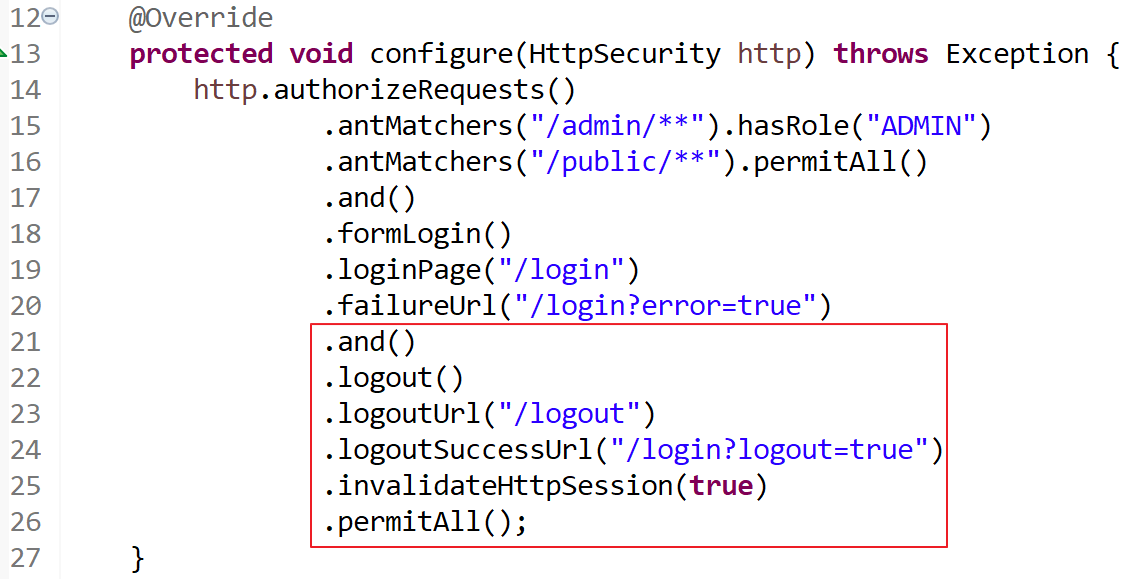
.logout()

.logoutUrl("/logout")

.logoutSuccessUrl("/login?logout=true")

.invalidateHttpSession(true)

.permitAll();



7. The logout URL is now set to “/logout”, let’s define “/logout” in the controller. Open “AppController.java” and add the following code:

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler;

@GetMapping(value="/logout")

public String logoutPage (HttpServletRequest request, HttpServletResponse response) {

Authentication auth = SecurityContextHolder.getContext().getAuthentication();

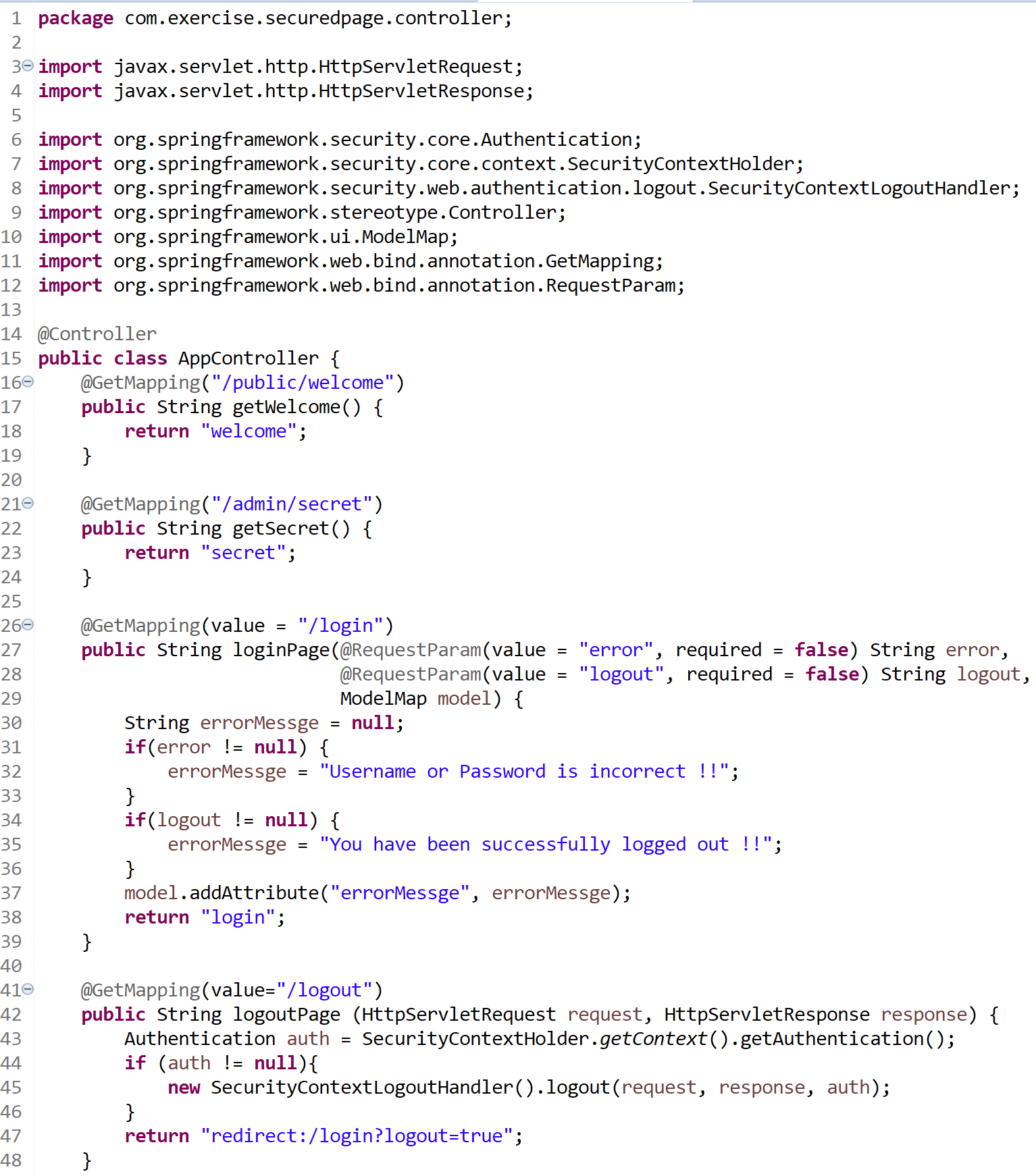
if (auth != null){

new SecurityContextLogoutHandler().logout(request, response, auth);

}

return "redirect:/login?logout=true";

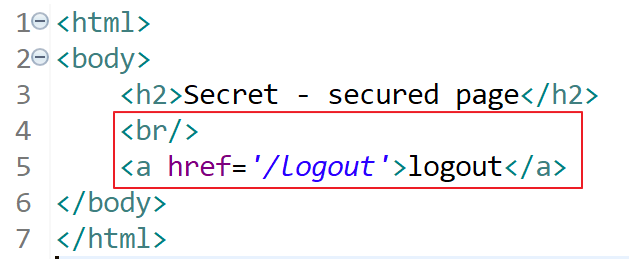
}



8. Add a “logout” link in “secret.jsp”. Open “secret.jsp” and add the following code:

<br/>

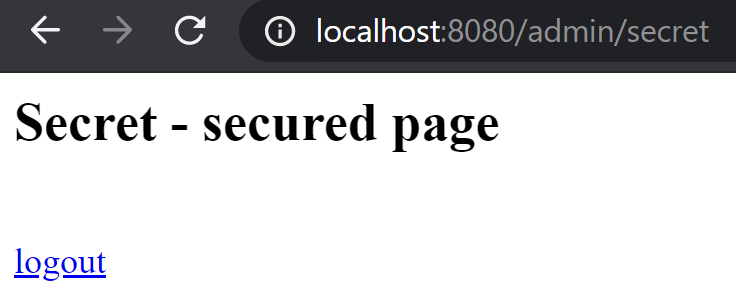
<a href='/logout'>logout</a>

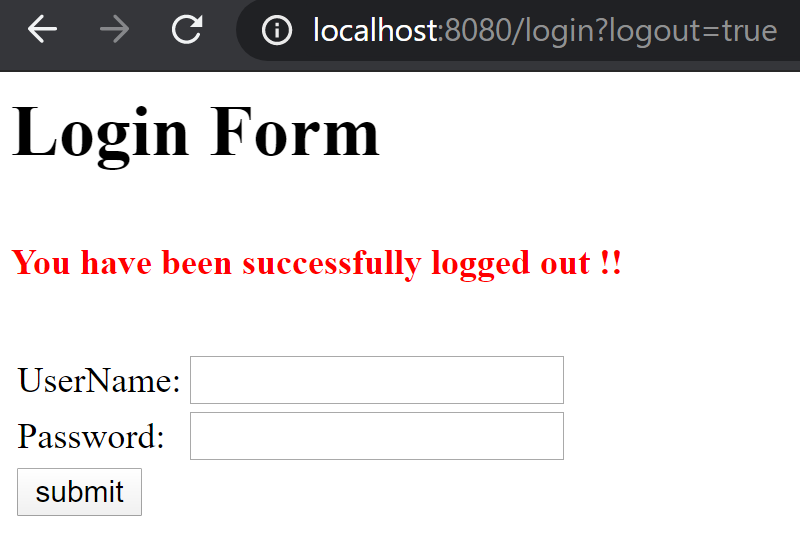


9. Restart the application and visit:

<http://localhost:8080/admin/secret>

After logging in, click the logout link to sign out:





10. Congratulations! Try this one if you have problem with this exercise:



~ End of the exercises ~