#### Add Spring Security Configuration Class

The first and foremost step to add spring security in our application is to create **Spring Security Java Configuration**. This configuration creates a Servlet Filter known as the springSecurityFilterChain which is responsible for all the security (protecting the application URLs, validating submitted username and passwords, redirecting to the log in form, etc) within our application.

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter {

@Autowired

**public** **void** configureGlobalSecurity(AuthenticationManagerBuilder auth)

**throws** Exception {

auth.inMemoryAuthentication()

.withUser("user").password("password")

.roles("USER");

auth.inMemoryAuthentication()

.withUser("admin").password("password")

.roles("ADMIN");

auth.inMemoryAuthentication()

.withUser("dba").password("password")

.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()

.and()

.exceptionHandling().accessDeniedPage("/Access\_Denied");

}

}

Method configureGlobalSecurity in above class configures AuthenticationManagerBuilder with user credentials and allowed roles. This AuthenticationManagerBuilder creates AuthenticationManager which is responsible for processing any authentication request. Notice that in above example, we have used in-memory authentication while you are free to choose from JDBC, LDAP and other authentications.

The overridden Method Configure configures HttpSecurity which allows configuring web based security for specific http requests. By default it will be applied to all requests, but can be restricted using requestMatcher(RequestMatcher)/antMathchers or other similar methods.

In above configuration, we say that URL’s ‘/’ & ‘/home’ are not secured, anyone can access them. URL ‘/admin/\*\*’ can only be accessed by someone who have ADMIN role. URL ‘/db/\*\*’ can only be accessed by someone who have both ADMIN and DBA roles.

Method formLogin provides support for form based authentication and will generate a default form asking for user credentials. You are allowed to configure your own login form.We will see examples for the same in subsequent posts.

We have also used exceptionHandling().accessDeniedPage() which in this case will catch all 403 [http access denied] exceptions and display our user defined page instead of showing default HTTP 403 page [ which is not so helpful anyway].

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

<http auto-config=*"true"*>

<intercept-url pattern=*"/"* access=*"permitAll"* />

<intercept-url pattern=*"/home"* access=*"permitAll"* />

<intercept-url pattern=*"/admin\*\*"* access=*"hasRole('ADMIN')"* />

<intercept-url pattern=*"/dba\*\*"*

access=*"hasRole('ADMIN') and hasRole('DBA')"* />

<form-login authentication-failure-url=*"/Access\_Denied"* />

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name=*"user"* password=*"password"* authorities=*"ROLE\_USER"* />

<user name=*"admin"* password=*"password"* authorities=*"ROLE\_ADMIN"* />

<user name=*"dba"* password=*"password"* authorities=*"ROLE\_ADMIN,ROLE\_DBA"* />

</user-service>

</authentication-provider>

</authentication-manager>

Below specified initializer class registers the springSecurityFilter with application war

**public** **class** SecurityInitializer **extends** AbstractSecurityWebApplicationInitializer {

}

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

<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>

@Controller

**public** **class** HelloWorldController {

@RequestMapping(value = { "/", "/home" }, method = RequestMethod.***GET***)

**public** String homePage(ModelMap model) {

model.addAttribute("greeting", "Hi, Welcome to my application. ");

**return** "welcome";

}

@RequestMapping(value = "/admin", method = RequestMethod.***GET***)

**public** String adminPage(ModelMap model) {

model.addAttribute("user", getPrincipal());

**return** "admin";

}

@RequestMapping(value = "/db", method = RequestMethod.***GET***)

**public** String dbaPage(ModelMap model) {

model.addAttribute("user", getPrincipal());

**return** "dba";

}

@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** "welcome";

}

@RequestMapping(value = "/Access\_Denied", method = RequestMethod.***GET***)

**public** String accessDeniedPage(ModelMap model) {

model.addAttribute("user", getPrincipal());

**return** "accessDenied";

}

**private** String getPrincipal(){

String userName = **null**;

Object principal = SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();

**if** (principal **instanceof** UserDetails) {

userName = ((UserDetails)principal).getUsername();

} **else** {

userName = principal.toString();

}

**return** userName;

}

}

Methods in controller class are trivial. Method getPrincipal is a generic function which returns the logged in user name from Spring SecurityContext. Method logoutPage handles the logging out with a simple call to**SecurityContextLogoutHandler().logout(request, response, auth);**. It’s handy and saves you from putting cryptic logout logic in your JSP’s which is not really manageable. You might have noticed that ‘/login’ is missing, it is because it will be generated and handled by default by Spring Security.

#### Add SpringMVC Configuration Class

@Configuration

@EnableWebMvc

@ComponentScan(basePackages = "in.spring4buddies.application")

**public** **class** WebApplicationConfiguration {

@Bean(name = "HelloWorld")

**public** ViewResolver viewResolver() {

InternalResourceViewResolver viewResolver = **new** InternalResourceViewResolver();

viewResolver.setViewClass(JstlView.**class**);

viewResolver.setPrefix("/WEB-INF/views/");

viewResolver.setSuffix(".jsp");

**return** viewResolver;

}

}

#### Add Initializer class

**public** **class** WebApplicationInitializer **extends** AbstractAnnotationConfigDispatcherServletInitializer {

@Override

**protected** Class<?>[] getRootConfigClasses() {

**return** **new** Class[] { WebApplicationConfiguration.**class** };

}

@Override

**protected** Class<?>[] getServletConfigClasses() {

**return** **null**;

}

@Override

**protected** String[] getServletMappings() {

**return** **new** String[] { "/" };

}

}

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.

welcome.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"* pageEncoding=*"ISO-8859-1"*%>

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>HelloWorld page</title>

</head>

<body>

Greeting : ${greeting}

This is a welcome page.

</body>

</html>

admin.jsp

<%@ 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>HelloWorld Admin page</title>

</head>

<body>

Dear <strong>${user}</strong>, Welcome to Admin Page.

<a href=*"*<c:url value=*"/logout"* />*"*>Logout</a>

</body>

</html>

dba.jsp

<%@ 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>DBA page</title>

</head>

<body>

Dear <strong>${user}</strong>, Welcome to DBA Page.

<a href=*"*<c:url value=*"/logout"* />*"*>Logout</a>

</body>

</html>

accessDenied.jsp

<%@ 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>AccessDenied page</title>

</head>

<body>

Dear <strong>${user}</strong>, You are not authorized to access this page

<a href=*"*<c:url value=*"/logout"* />*"*>Logout</a>

</body>

</html>

Mvn clean install - > mvn jetty:run

* localhost:8080/ sf-security-default-login-form/
* Now try to access admin page on localhost:8080/ sf-security-default-login-form/admin, you will be prompted for login.
  + Provide credentials of a ‘USER’ role. Submit, you will see AccessDenied Page
  + Now logout and try to access admin page again, Provide wrong password,

we have seen the default login form provided by Spring Security in case we don’t specify one. In this post, we will create our own Custom login form. Basically, the idea is, in Security Configuration, attach a call to **loginPage(URL)** function with **formLogin()** like shown below

.and().formLogin().loginPage("/login")

And then, Map this ‘/login’ URL in your Spring MVC Controller which will return the login view defined by you. Now, on login attempt, the specified login view will be displayed.Rest of the login functionality remains same. Below provided is complete example for this scenario.

.and().formLogin().loginPage("/login")

        .usernameParameter("ssoId").passwordParameter("password")

        .and().csrf()

This code creates a custom login page with ‘/login’ url, which will accept ssoId as username and password Http request parameters. We have also shown a call to **csrf()** which is optional as it is by default active in Spring Security 4. This call is, however, required if you want to disable CSRF protection by using **csrf().disable()** although it is not a good idea to disable it.

<http auto-config=*"true"*>

<intercept-url pattern=*"/"* access=*"permitAll"* />

<intercept-url pattern=*"/home"* access=*"permitAll"* />

<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" />

        <csrf/>

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name=*"user"* password=*"password"* authorities=*"ROLE\_USER"* />

<user name=*"admin"* password=*"password"* authorities=*"ROLE\_ADMIN"* />

<user name=*"dba"* password=*"password"* authorities=*"ROLE\_ADMIN,ROLE\_DBA"* />

</user-service>

</authentication-provider>

</authentication-manager>

only changes are new loginPage method to handle ‘/login’ requests and adapting logout to redirect to login page on logout, as shown below:

@RequestMapping(value = "/login", method = RequestMethod.***GET***)

**public** String loginPage() {

**return** "login";

}

@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";

}

WebApplicationConfiguration class extend from **WebMvcConfigurerAdapter** [just a convenience class] and implementing method **addResourceHandlers** which handles static resources(CSS/images/..) to be used in views.

@Override

**public** **void** addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/static/\*\*").addResourceLocations( "/static/");

}

Notice the CSRF related line in above jsp:

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

This is required to protect against CSRF attacks. As you can see, the CSRF parameters are accessed using EL Expressions in your JSP, you may additionally prefer to force EL expressions to be evaluated, by adding following to the top of your JSP:

<%@ page isELIgnored="false"%>

# Secure View Fragments using taglibs

This tutorial shows you how to secure view layer, show/hide parts of jsp/view based on logged-in user’s roles, using Spring Security tags in Spring MVC web application.

First of all, in order to use Spring Security tags, we need to include spring-security-taglibs dependency in pom.xml as shown below:

<dependency>

    <groupId>org.springframework.security</groupId>

    <artifactId>spring-security-taglibs</artifactId>

    <version>4.0.1.RELEASE</version>

</dependency>

Then the next step would be to include taglib in your views/JSP’s.

|  |
| --- |
| <%@ taglib prefix="sec" uri="<http://www.springframework.org/security/tags>"%>  Finally, we can use Spring Security expresssions like hasRole, hasAnyRole, etc.. in Views as 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"*%>

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

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Welcome page</title>

</head>

<body>

Dear <strong>${user}</strong>, Welcome to Home Page.

<a href=*"*<c:url value=*"/logout"* />*"*>Logout</a>

<br/>

<br/>

<div>

<label>View all information| This part is visible to Everyone</label>

</div>

<br/>

<div>

<sec:authorize access=*"hasRole('ADMIN')"*>

<label><a href=*"#"*>Edit this page</a> | This part is visible only to ADMIN</label>

</sec:authorize>

</div>

<br/>

<div>

<sec:authorize access=*"hasRole('ADMIN') and hasRole('DBA')"*>

<label><a href=*"#"*>Start backup</a> | This part is visible only to one who is both ADMIN & DBA</label>

</sec:authorize>

</div>

</html>

That’s all you need to conditionally show/hide view fragments based on roles, using Spring Security expressions in your Views. Below is the Security Configuration used for this example:

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter {

@Autowired

**public** **void** configureGlobalSecurity(AuthenticationManagerBuilder auth)

**throws** Exception {

auth.inMemoryAuthentication().withUser("user").password("password")

.roles("USER");

auth.inMemoryAuthentication().withUser("admin").password("password")

.roles("ADMIN");

auth.inMemoryAuthentication().withUser("dba").password("password")

.roles("ADMIN", "DBA");

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests()

.antMatchers("/", "/home")

.access("hasRole('USER') or hasRole('ADMIN') or hasRole('DBA')")

.and()

.formLogin().loginPage("/login")

.usernameParameter("ssoId").passwordParameter("password")

.and()

.exceptionHandling().accessDeniedPage("/Access\_Denied");

}

}

All other configuration same as above…

**Spring Security 4 Role Based Login**

Spring Security’s role based login. That means redirecting users to different URLs upon login according to their assigned roles. Basically what we have to do is to create a custom Success-Handler which will be responsible for **redirecting**the logged-in user to appropriate URL based on his/her role. Spring Security already providesSimpleUrlAuthenticationSuccessHandler which contains the generic logic for success handler. We will just extend this with our own redirect logic to achieve our goal.

Once we got this success handler, we will register it with **formLogin() or loginPage()** et voila. Complete example is shown below

The first and foremost step to add spring security in our application is to create **Spring Security Java Configuration**. This configuration creates a Servlet Filter known as the springSecurityFilterChain which is responsible for all the security (protecting the application URLs, validating submitted username and passwords, redirecting to the log in form, etc) within our application.

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter {

@Autowired

CustomSuccessHandler customSuccessHandler;

@Autowired

**public** **void** configureGlobalSecurity(AuthenticationManagerBuilder auth)

**throws** Exception {

auth.inMemoryAuthentication().withUser("user").password("password")

.roles("USER");

auth.inMemoryAuthentication().withUser("admin").password("password")

.roles("ADMIN");

auth.inMemoryAuthentication().withUser("dba").password("password")

.roles("ADMIN", "DBA");

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests()

.antMatchers("/", "/home").access("hasRole('USER')")

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

.antMatchers("/db/\*\*").access("hasRole('ADMIN') and hasRole('DBA')")

.and()

.formLogin().loginPage("/login").successHandler(customSuccessHandler)

.usernameParameter("ssoId").passwordParameter("password")

.and()

.csrf()

.and()

.exceptionHandling().accessDeniedPage("/Access\_Denied");

}

}

This class is similar to previous posts except one major difference:  
formLogin().loginPage("/login").successHandler(customSuccessHandler). Look at successHandler. This is the class [shown below] responsible for eventual redirection based on any custom logic, which in our case will be to redirect the user [to home/admin/db ] based on his role [USER/ADMIN/DBA].

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

<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-success-handler-ref=*"customSuccessHandler"*

authentication-failure-url=*"/Access\_Denied"* />

<csrf />

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name=*"user"* password=*"password"* authorities=*"ROLE\_USER"* />

<user name=*"admin"* password=*" password "* authorities=*"ROLE\_ADMIN"* />

<user name=*"dba"* password=*" password "* authorities=*"ROLE\_ADMIN,ROLE\_DBA"* />

</user-service>

</authentication-provider>

</authentication-manager>

<bean id="customSuccessHandler" class=" in.spring4buddies.application.configuration.CustomSuccessHandler" />

Below is the Success-Handler referred in above class

@Component

**public** **class** CustomSuccessHandler **extends** SimpleUrlAuthenticationSuccessHandler {

**private** RedirectStrategy redirectStrategy = **new** DefaultRedirectStrategy();

@Override

**protected** **void** handle(HttpServletRequest request,

HttpServletResponse response, Authentication authentication)

**throws** IOException {

String targetUrl = determineTargetUrl(authentication);

**if** (response.isCommitted()) {

System.***out***.println("Can't redirect");

**return**;

}

redirectStrategy.sendRedirect(request, response, targetUrl);

}

/\*

\* This method extracts the roles of currently logged-in user and returns

\* appropriate URL according to his/her role.

\*/

**protected** String determineTargetUrl(Authentication authentication) {

String url = "";

Collection<? **extends** GrantedAuthority> authorities = authentication

.getAuthorities();

List<String> roles = **new** ArrayList<String>();

**for** (GrantedAuthority a : authorities) {

roles.add(a.getAuthority());

}

**if** (isDba(roles)) {

url = "/db";

} **else** **if** (isAdmin(roles)) {

url = "/admin";

} **else** **if** (isUser(roles)) {

url = "/home";

} **else** {

url = "/accessDenied";

}

**return** url;

}

**private** **boolean** isUser(List<String> roles) {

**if** (roles.contains("ROLE\_USER")) {

**return** **true**;

}

**return** **false**;

}

**private** **boolean** isAdmin(List<String> roles) {

**if** (roles.contains("ROLE\_ADMIN")) {

**return** **true**;

}

**return** **false**;

}

**private** **boolean** isDba(List<String> roles) {

**if** (roles.contains("ROLE\_DBA")) {

**return** **true**;

}

**return** **false**;

}

**public** **void** setRedirectStrategy(RedirectStrategy redirectStrategy) {

**this**.redirectStrategy = redirectStrategy;

}

**protected** RedirectStrategy getRedirectStrategy() {

**return** redirectStrategy;

}

}

Notice how we are extending Spring SimpleUrlAuthenticationSuccessHandler class and overridinghandle() method which simply invokes a redirect using configured RedirectStrategy [default in this case] with the URL returned by the user defined **determineTargetUrl** method. This method extracts the Roles of currently logged in user from Authentication object and then construct appropriate URL based on there roles. Finally RedirectStrategy , which is responsible for all redirections within Spring Security framework , redirects the request to specified URL.

Other class and views same as first one.

**Spring Security 4 Hibernate Integration**

we will learn Spring Security database authentication using Hibernate annotation+xml based approach. Previous posts discussed about Spring Security **in-memory authentication**. But in real-world projects, credentials are often stored in database or LDAP. In this post we will go through a complete example of setting up Spring security and verifying credentials directly against database using Hibernate.

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter {

@Autowired

@Qualifier("customUserDetailsService")

UserDetailsService userDetailsService;

@Autowired

**public** **void** configureGlobalSecurity(AuthenticationManagerBuilder auth)

**throws** Exception {

auth.userDetailsService(userDetailsService);

}

@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()

.csrf()

.and()

.exceptionHandling().accessDeniedPage("/Access\_Denied");

}

}

All the credentials are now stored in database, and will be accessible to Spring Security viaorg.springframework.security.core.userdetails.UserDetailsService implementations. We will provide an implementation of **UserDetailsService** which will eventually use our fully transactional user defined **userService**to access data from database.

That’s it. Rest of the setup for this post is (deja vu) usual Spring Security, Spring MVC and trivial Hibernate Setup which we have seen many times in previous tutorials. Below is the full example code for this post. We have divided the responsibilities into separate layers(service/dao) to make it manageable.

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

<http auto-config=*"true"*>

<intercept-url pattern=*"/"* access=*"permitAll"* />

<intercept-url pattern=*"/home"* access=*"permitAll"* />

<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"* />

<csrf />

</http>

<authentication-manager>

<authentication-provider user-service-ref=*"customUserDetailsService"* />

</authentication-manager>

<bean id=*"customUserDetailsService"*

class=*"in.spring4buddies.application.service.CustomUserDetailsService"* />

Below specified initializer class registers the springSecurityFilter [created in Step 3] with application war.

**public** **class** SecurityInitializer **extends**

AbstractSecurityWebApplicationInitializer {

}

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

<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>

**Define UserDetailsService implementation**

This service is responsible for providing authentication details to Authentication Manager. It implements Spring’sUserDetailsService interface, which contains only one method **loadUserByUsername**, taking username[ssoId in our example] and returns a org.springframework.security.core.userdetails.User object. We will populate this object using our own UserService which gets data from db using UserDao object.

@Service("customUserDetailsService")

**public** **class** CustomUserDetailsService **implements** UserDetailsService {

@Autowired

**private** UserService userService;

@Transactional(readOnly = **true**)

**public** UserDetails loadUserByUsername(String ssoId)

**throws** UsernameNotFoundException {

User user = userService.findBySso(ssoId);

System.***out***.println("User : " + user);

**if** (user == **null**) {

System.***out***.println("User not found");

**throw** **new** UsernameNotFoundException("Username not found");

}

**return** **new** org.springframework.security.core.userdetails.User(

user.getSsoId(), user.getPassword(), user.getState().equals(

"Active"), **true**, **true**, **true**,

getGrantedAuthorities(user));

}

**private** List<GrantedAuthority> getGrantedAuthorities(User user) {

List<GrantedAuthority> authorities = **new** ArrayList<GrantedAuthority>();

**for** (UserProfile userProfile : user.getUserProfiles()) {

System.***out***.println("UserProfile : " + userProfile);

authorities.add(**new** SimpleGrantedAuthority("ROLE\_"

+ userProfile.getType()));

}

System.***out***.print("authorities :" + authorities);

**return** authorities;

}

}

**Add Controller**

@Controller

**public** **class** HelloWorldController {

@RequestMapping(value = { "/", "/home" }, method = RequestMethod.***GET***)

**public** String homePage(ModelMap model) {

model.addAttribute("greeting", "Hi, Welcome to my appication");

**return** "welcome";

}

@RequestMapping(value = "/admin", method = RequestMethod.***GET***)

**public** String adminPage(ModelMap model) {

model.addAttribute("user", getPrincipal());

**return** "admin";

}

@RequestMapping(value = "/db", method = RequestMethod.***GET***)

**public** String dbaPage(ModelMap model) {

model.addAttribute("user", getPrincipal());

**return** "dba";

}

@RequestMapping(value = "/Access\_Denied", method = RequestMethod.***GET***)

**public** String accessDeniedPage(ModelMap model) {

model.addAttribute("user", getPrincipal());

**return** "accessDenied";

}

@RequestMapping(value = "/login", method = RequestMethod.***GET***)

**public** String loginPage() {

**return** "login";

}

@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";

}

**private** String getPrincipal() {

String userName = **null**;

Object principal = SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();

**if** (principal **instanceof** UserDetails) {

userName = ((UserDetails) principal).getUsername();

} **else** {

userName = principal.toString();

}

**return** userName;

}

}

**Add SpringMVC Configuration Class**

@Configuration

@EnableWebMvc

@ComponentScan(basePackages = "in.spring4buddies.application")

**public** **class** WebApplicationConfiguration **extends** WebMvcConfigurerAdapter {

@Bean

**public** ViewResolver viewResolver() {

InternalResourceViewResolver viewResolver = **new** InternalResourceViewResolver();

viewResolver.setViewClass(JstlView.**class**);

viewResolver.setPrefix("/WEB-INF/views/");

viewResolver.setSuffix(".jsp");

**return** viewResolver;

}

@Override

**public** **void** addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/static/\*\*").addResourceLocations(

"/static/");

}

}

**Add Initializer class**

**public** **class** WebApplicationInitializer **extends**

AbstractAnnotationConfigDispatcherServletInitializer {

@Override

**protected** Class<?>[] getRootConfigClasses() {

**return** **new** Class[] { WebApplicationConfiguration.**class** };

}

@Override

**protected** Class<?>[] getServletConfigClasses() {

**return** **null**;

}

@Override

**protected** String[] getServletMappings() {

**return** **new** String[] { "/" };

}

}

**Create Model classes**

A User can have multiple roles [DBA,ADMIN,USER]. And a Role can be assigned to more than one user. Hence there is a Many-To-Many relationship between a User and UserProfile[role]. We kept this relationship uni-directional [User to UserProfile] as we are only interested in finding Roles for a give user (and not vice-versa). We will be using Many-To-Many association using Join table.

@Entity

@Table(name = "APP\_USER")

**public** **class** User {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

@Column(name = "SSO\_ID", unique = **true**, nullable = **false**)

**private** String ssoId;

@Column(name = "PASSWORD", nullable = **false**)

**private** String password;

@Column(name = "FIRST\_NAME", nullable = **false**)

**private** String firstName;

@Column(name = "LAST\_NAME", nullable = **false**)

**private** String lastName;

@Column(name = "EMAIL", nullable = **false**)

**private** String email;

@Column(name = "STATE", nullable = **false**)

**private** String state = State.***ACTIVE***.getState();

@ManyToMany(fetch = FetchType.***EAGER***)

@JoinTable(name = "APP\_USER\_USER\_PROFILE", joinColumns = { @JoinColumn(name = "USER\_ID") }, inverseJoinColumns = { @JoinColumn(name = "USER\_PROFILE\_ID") })

**private** Set<UserProfile> userProfiles = **new** HashSet<UserProfile>();

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getSsoId() {

**return** ssoId;

}

**public** **void** setSsoId(String ssoId) {

**this**.ssoId = ssoId;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getState() {

**return** state;

}

**public** **void** setState(String state) {

**this**.state = state;

}

**public** Set<UserProfile> getUserProfiles() {

**return** userProfiles;

}

**public** **void** setUserProfiles(Set<UserProfile> userProfiles) {

**this**.userProfiles = userProfiles;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + id;

result = prime \* result + ((ssoId == **null**) ? 0 : ssoId.hashCode());

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (!(obj **instanceof** User))

**return** **false**;

User other = (User) obj;

**if** (id != other.id)

**return** **false**;

**if** (ssoId == **null**) {

**if** (other.ssoId != **null**)

**return** **false**;

} **else** **if** (!ssoId.equals(other.ssoId))

**return** **false**;

**return** **true**;

}

@Override

**public** String toString() {

**return** "User [id=" + id + ", ssoId=" + ssoId + ", password=" + password

+ ", firstName=" + firstName + ", lastName=" + lastName

+ ", email=" + email + ", state=" + state + ", userProfiles="

+ userProfiles + "]";

}

}

@Entity

@Table(name = "USER\_PROFILE")

**public** **class** UserProfile {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** id;

@Column(name = "TYPE", length = 15, unique = **true**, nullable = **false**)

**private** String type = UserProfileType.***USER***.getUserProfileType();

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getType() {

**return** type;

}

**public** **void** setType(String type) {

**this**.type = type;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + id;

result = prime \* result + ((type == **null**) ? 0 : type.hashCode());

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (!(obj **instanceof** UserProfile))

**return** **false**;

UserProfile other = (UserProfile) obj;

**if** (id != other.id)

**return** **false**;

**if** (type == **null**) {

**if** (other.type != **null**)

**return** **false**;

} **else** **if** (!type.equals(other.type))

**return** **false**;

**return** **true**;

}

@Override

**public** String toString() {

**return** "UserProfile [id=" + id + ", type=" + type + "]";

}

}

**public** **enum** State {

***ACTIVE***("Active"),

***INACTIVE***("Inactive"),

***DELETED***("Deleted"),

***LOCKED***("Locked");

**private** String state;

**private** State(**final** String state) {

**this**.state = state;

}

**public** String getState() {

**return** **this**.state;

}

@Override

**public** String toString() {

**return** **this**.state;

}

**public** String getName() {

**return** **this**.name();

}

}

**public** **enum** UserProfileType {

***USER***("USER"),

***DBA***("DBA"),

***ADMIN***("ADMIN");

String userProfileType;

**private** UserProfileType(String userProfileType) {

**this**.userProfileType = userProfileType;

}

**public** String getUserProfileType() {

**return** userProfileType;

}

}

**Create Dao Layer**

@SuppressWarnings("unchecked")

**public** **abstract** **class** AbstractDao<PK **extends** Serializable, T> {

**private** **final** Class<T> persistentClass;

**public** AbstractDao() {

**this**.persistentClass = (Class<T>) ((ParameterizedType) **this**.getClass()

.getGenericSuperclass()).getActualTypeArguments()[1];

}

@Autowired

**private** SessionFactory sessionFactory;

**protected** Session getSession() {

**return** sessionFactory.getCurrentSession();

}

**public** T getByKey(PK key) {

**return** (T) getSession().get(persistentClass, key);

}

**public** **void** persist(T entity) {

getSession().persist(entity);

}

**public** **void** delete(T entity) {

getSession().delete(entity);

}

**protected** Criteria createEntityCriteria() {

**return** getSession().createCriteria(persistentClass);

}

}

**public** **interface** UserDao {

User findById(**int** id);

User findBySSO(String sso);

}

@Repository("userDao")

**public** **class** UserDaoImpl **extends** AbstractDao<Integer, User> **implements** UserDao {

**public** User findById(**int** id) {

**return** getByKey(id);

}

**public** User findBySSO(String sso) {

Criteria crit = createEntityCriteria();

crit.add(Restrictions.*eq*("ssoId", sso));

**return** (User) crit.uniqueResult();

}

}

**Create Service Layer**

**public** **interface** UserService {

User findById(**int** id);

User findBySso(String sso);

}

@Service("userService")

@Transactional

**public** **class** UserServiceImpl **implements** UserService {

@Autowired

**private** UserDao dao;

**public** User findById(**int** id) {

**return** dao.findById(id);

}

**public** User findBySso(String sso) {

**return** dao.findBySSO(sso);

}

}

**Create Hibernate Configuration**

Hibernate configuration class contains @Bean methods for DataSource, SessionFactory & Transaction Manager. Datasource properties are taken from application.properties file and contains connection details for ORACLE database.

@Configuration

@EnableTransactionManagement

@ComponentScan({ "in.spring4buddies.application.configuration" })

@PropertySource(value = { "classpath:application.properties" })

**public** **class** HibernateConfiguration {

@Autowired

**private** Environment environment;

@Bean

**public** LocalSessionFactoryBean sessionFactory() {

LocalSessionFactoryBean sessionFactory = **new** LocalSessionFactoryBean();

sessionFactory.setDataSource(dataSource());

sessionFactory.setPackagesToScan(**new** String[] { "in.spring4buddies.application.model" });

sessionFactory.setHibernateProperties(hibernateProperties());

**return** sessionFactory;

}

@Bean

**public** DataSource dataSource() {

DriverManagerDataSource dataSource = **new** DriverManagerDataSource();

dataSource.setDriverClassName(environment.getRequiredProperty("jdbc.driverClassName"));

dataSource.setUrl(environment.getRequiredProperty("jdbc.url"));

dataSource.setUsername(environment.getRequiredProperty("jdbc.username"));

dataSource.setPassword(environment.getRequiredProperty("jdbc.password"));

**return** dataSource;

}

**private** Properties hibernateProperties() {

Properties properties = **new** Properties();

properties.put("hibernate.dialect", environment.getRequiredProperty("hibernate.dialect"));

properties.put("hibernate.show\_sql", environment.getRequiredProperty("hibernate.show\_sql"));

properties.put("hibernate.format\_sql", environment.getRequiredProperty("hibernate.format\_sql"));

**return** properties;

}

@Bean

@Autowired

**public** HibernateTransactionManager transactionManager(SessionFactory s) {

HibernateTransactionManager txManager = **new** HibernateTransactionManager();

txManager.setSessionFactory(s);

**return** txManager;

}

}

application.properties

jdbc.driverClassName = oracle.jdbc.driver.OracleDriver

jdbc.url = jdbc:oracle:thin:@localhost:1522:orcl

jdbc.username = srlp

jdbc.password = srlp

hibernate.dialect = org.hibernate.dialect.Oracle10gDialect

hibernate.show\_sql = true

hibernate.format\_sql = true

**Add Views**

login.jsp

<%@ 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>Login page</title>

<link href=*"*<c:url value=*'/static/css/bootstrap.css'* />*"* rel=*"stylesheet"*></link>

<link href=*"*<c:url value=*'/static/css/app.css'* />*"* rel=*"stylesheet"*></link>

<link rel=*"stylesheet"* type=*"text/css"* href=*"//cdnjs.cloudflare.com/ajax/libs/font-awesome/4.2.0/css/font-awesome.css"* />

</head>

<body>

<div id=*"mainWrapper"*>

<div class=*"login-container"*>

<div class=*"login-card"*>

<div class=*"login-form"*>

<c:url var=*"loginUrl"* value=*"/login"* />

<form action=*"*${loginUrl}*"* method=*"post"* class=*"form-horizontal"*>

<c:if test=*"*${param.error != null}*"*>

<div class=*"alert alert-danger"*>

<p>Invalid username and password.</p>

</div>

</c:if>

<c:if test=*"*${param.logout != null}*"*>

<div class=*"alert alert-success"*>

<p>You have been logged out successfully.</p>

</div>

</c:if>

<div class=*"input-group input-sm"*>

<label class=*"input-group-addon"* for=*"username"*><i class=*"fa fa-user"*></i></label>

<input type=*"text"* class=*"form-control"* id=*"username"* name=*"ssoId"* placeholder=*"Enter Username"* required>

</div>

<div class=*"input-group input-sm"*>

<label class=*"input-group-addon"* for=*"password"*><i class=*"fa fa-lock"*></i></label>

<input type=*"password"* class=*"form-control"* id=*"password"* name=*"password"* placeholder=*"Enter Password"* required>

</div>

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

<div class=*"form-actions"*>

<input type=*"submit"*

class=*"btn btn-block btn-primary btn-default"* value=*"Log in"*>

</div>

</form>

</div>

</div>

</div>

</div>

</body>

</html>

As you can see, the CSRF parameters are accessed using EL expressions in your JSP, you may additionally prefer to force EL expressions to be evaluated, by adding following to the top of your JSP:

<%@ page isELIgnored="false"%>

welcome.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"* pageEncoding=*"ISO-8859-1"*%>

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Welcome page</title>

</head>

<body>

Greeting : ${greeting}

This is a welcome page.

</body>

</html>

admin.jsp

<%@ 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>

dba.jsp

<%@ 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>DBA page</title>

</head>

<body>

Dear <strong>${user}</strong>, Welcome to DBA Page.

<a href=*"*<c:url value=*"/logout"* />*"*>Logout</a>

</body>

</html>

accessDenied.jsp

<%@ 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>AccessDenied page</title>

</head>

<body>

Dear <strong>${user}</strong>, You are not authorized to access this page

<a href=*"*<c:url value=*"/logout"* />*"*>Logout</a>

</body>

</html>

**Spring Security 4 Hibernate Role Based Login**

how to use role based login in Spring Security 4 using Hibernate setup. That means redirecting users to different URLs upon login according to their assigned roles, this time along with Hibernate setup.

**Create a NEW Custom Success Handler**

Goal of this class is to provide custom redirect functionality we are looking for.

@Component

**public** **class** CustomSuccessHandler **extends** SimpleUrlAuthenticationSuccessHandler {

**private** RedirectStrategy redirectStrategy = **new** DefaultRedirectStrategy();

@Override

**protected** **void** handle(HttpServletRequest request,

HttpServletResponse response, Authentication authentication)

**throws** IOException {

String targetUrl = determineTargetUrl(authentication);

**if** (response.isCommitted()) {

System.***out***.println("Can't redirect");

**return**;

}

redirectStrategy.sendRedirect(request, response, targetUrl);

}

**protected** String determineTargetUrl(Authentication authentication) {

String url = "";

Collection<? **extends** GrantedAuthority> authorities = authentication

.getAuthorities();

List<String> roles = **new** ArrayList<String>();

**for** (GrantedAuthority a : authorities) {

roles.add(a.getAuthority());

}

**if** (isDba(roles)) {

url = "/db";

} **else** **if** (isAdmin(roles)) {

url = "/admin";

} **else** **if** (isUser(roles)) {

url = "/home";

} **else** {

url = "/accessDenied";

}

**return** url;

}

**public** **void** setRedirectStrategy(RedirectStrategy redirectStrategy) {

**this**.redirectStrategy = redirectStrategy;

}

**protected** RedirectStrategy getRedirectStrategy() {

**return** redirectStrategy;

}

**private** **boolean** isUser(List<String> roles) {

**if** (roles.contains("ROLE\_USER")) {

**return** **true**;

}

**return** **false**;

}

**private** **boolean** isAdmin(List<String> roles) {

**if** (roles.contains("ROLE\_ADMIN")) {

**return** **true**;

}

**return** **false**;

}

**private** **boolean** isDba(List<String> roles) {

**if** (roles.contains("ROLE\_DBA")) {

**return** **true**;

}

**return** **false**;

}

}

Notice how we are extending Spring SimpleUrlAuthenticationSuccessHandler class and overridinghandle() method which simply invokes a redirect using configured RedirectStrategy [default in this case] with the URL returned by the user defined **determineTargetUrl** method. This method extracts the Roles of currently logged in user from Authentication object and then construct appropriate URL based on there roles. Finally RedirectStrategy , which is responsible for all redirections within Spring Security framework , redirects the request to specified URL.

**Register Custom Success Handler with [existing] Security Configuration**

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter {

@Autowired

@Qualifier("customUserDetailsService")

UserDetailsService userDetailsService;

@Autowired

CustomSuccessHandler customSuccessHandler;

@Autowired

**public** **void** configureGlobalSecurity(AuthenticationManagerBuilder auth)

**throws** Exception {

auth.userDetailsService(userDetailsService);

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests()

// .antMatchers("/", "/home").permitAll()

.antMatchers("/", "/home").access("hasRole('USER')")

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

.antMatchers("/db/\*\*").access("hasRole('ADMIN') and hasRole('DBA')")

// .and().formLogin().loginPage("/login")

.and()

.formLogin().loginPage("/login")

.successHandler(customSuccessHandler)

.usernameParameter("ssoId").passwordParameter("password")

.and()

.csrf()

.and()

.exceptionHandling().accessDeniedPage("/Access\_Denied");

}

}

What changes compare to previous Hibernate post is an extra call to **successHandler()** as highlighted below:  
formLogin().loginPage("/login").successHandler(customSuccessHandler).  
Look at successHandler. This is the class responsible for eventual redirection based on any custom logic, which in our case will be to redirect the user [to home/admin/db ] based on his role [USER/ADMIN/DBA].

Additionally, we have also protected the home page, under USER role, to make example more realistic.That’s it. Just add this class in configuration package and register it as success-handler with Security Configuration (as shown above)

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

<http auto-config=*"true"*>

<intercept-url pattern=*"/"* access=*"permitAll"* />

<intercept-url pattern=*"/home"* access=*"permitAll"* />

<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-success-handler-ref=*"customSuccessHandler"*

authentication-failure-url=*"/Access\_Denied"* />

<csrf />

</http>

<authentication-manager>

<authentication-provider user-service-ref=*"customUserDetailsService"* />

</authentication-manager>

<bean id=*"customUserDetailsService"*

class=*"in.spring4buddies.application.service.CustomUserDetailsService"* />

<bean id=*"customSuccessHandler"*

class=*"in.spring4buddies.application.configuration.CustomSuccessHandler"* />

**Spring Security 4 Hibernate Password Encoder Bcrypt**

**A word on Password Encoding** Any application, which takes Security seriously, should **NEVER** store passwords in plain text format. Passwords should always be encoded using a secure hashing algorithm. There are many standard algorithms like SHA or MD5 which combined with a proper SALT can be a good choice for password encoding. Spring Security providesBCryptPasswordEncoder, and implementation of Spring’s PasswordEncoder interface that uses the BCrypt strong hashing function to encode the password

**Where all possibly do we need Password Encoding in application?**

**1.**During password comparison. Encode input password before comparing with the one stored in database(which is encoded)  
**2.**During New user creation/existing user password updation. Encode the new input password before saving/updating in database.

#### Changes compare to previous posts?

**1.**Create & Inject PasswordEncoder into AuthenticationProvider & set it as authentication provider onAuthenticationManagerBuilder

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter {

@Autowired

@Qualifier("customUserDetailsService")

UserDetailsService userDetailsService;

@Autowired

**public** **void** configureGlobalSecurity(AuthenticationManagerBuilder auth)

**throws** Exception {

auth.userDetailsService(userDetailsService);

auth.authenticationProvider(authenticationProvider());

}

@Bean

**public** PasswordEncoder passwordEncoder() {

**return** **new** BCryptPasswordEncoder();

}

@Bean

**public** DaoAuthenticationProvider authenticationProvider() {

DaoAuthenticationProvider authenticationProvider = **new** DaoAuthenticationProvider();

authenticationProvider.setUserDetailsService(userDetailsService);

authenticationProvider.setPasswordEncoder(passwordEncoder());

**return** authenticationProvider;

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests()

.antMatchers("/", "/home").permitAll()

.antMatchers("/admin/\*\*", "/newuser").access("hasRole('ADMIN')")

.antMatchers("/db/\*\*").access("hasRole('ADMIN') and hasRole('DBA')")

.and()

.formLogin().loginPage("/login").usernameParameter("ssoId")

.passwordParameter("password")

.and()

.csrf()

.and()

.exceptionHandling().accessDeniedPage("/Access\_Denied");

}

}

Above setup will take care of password comparisons during Authentication anywhere in application.

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

<http auto-config=*"true"*>

<intercept-url pattern=*"/"* access=*"permitAll"* />

<intercept-url pattern=*"/home"* access=*"permitAll"* />

<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"* />

<csrf />

</http>

<authentication-manager>

<authentication-provider user-service-ref=*"customUserDetailsService"*>

<password-encoder ref=*"bcryptEncoder"* />

</authentication-provider>

</authentication-manager>

<bean id=*"bcryptEncoder"*

class=*"org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"* />

<bean id=*"customUserDetailsService"*

class=*"in.spring4buddies.application.service.CustomUserDetailsService"* />

**2.**Update UserService to include password encoding before saving new password in database.

@Service("userService")

@Transactional

**public** **class** UserServiceImpl **implements** UserService {

@Autowired

**private** UserDao dao;

@Autowired

**private** PasswordEncoder passwordEncoder;

**public** **void** save(User user) {

user.setPassword(passwordEncoder.encode(user.getPassword()));

dao.save(user);

}

**public** User findById(**int** id) {

**return** dao.findById(id);

}

**public** User findBySso(String sso) {

**return** dao.findBySSO(sso);

}

}

That’s all you have to do to setup password encoding in your application using Spring Security’s BCrypt implementation.

**Register the springSecurityFilter with war**

**public** **class** SecurityInitializer **extends**

AbstractSecurityWebApplicationInitializer {

}

**Define UserDetailsService implementation**

@Service("customUserDetailsService")

**public** **class** CustomUserDetailsService **implements** UserDetailsService {

@Autowired

**private** UserService userService;

@Transactional(readOnly = **true**)

**public** UserDetails loadUserByUsername(String ssoId)

**throws** UsernameNotFoundException {

User user = userService.findBySso(ssoId);

System.***out***.println("User : " + user);

**if** (user == **null**) {

System.***out***.println("User not found");

**throw** **new** UsernameNotFoundException("Username not found");

}

**return** **new** org.springframework.security.core.userdetails.User(

user.getSsoId(), user.getPassword(), user.getState().equals(

"Active"), **true**, **true**, **true**,

getGrantedAuthorities(user));

}

**private** List<GrantedAuthority> getGrantedAuthorities(User user) {

List<GrantedAuthority> authorities = **new** ArrayList<GrantedAuthority>();

**for** (UserProfile userProfile : user.getUserProfiles()) {

System.***out***.println("UserProfile : " + userProfile);

authorities.add(**new** SimpleGrantedAuthority("ROLE\_"

+ userProfile.getType()));

}

System.***out***.print("authorities :" + authorities);

**return** authorities;

}

}

**Add Controller**

@Controller

**public** **class** HelloWorldController {

@Autowired

UserProfileService userProfileService;

@Autowired

UserService userService;

@RequestMapping(value = { "/", "/home" }, method = RequestMethod.***GET***)

**public** String homePage(ModelMap model) {

model.addAttribute("greeting", "Hi, Welcome to mysite");

**return** "welcome";

}

@RequestMapping(value = "/admin", method = RequestMethod.***GET***)

**public** String adminPage(ModelMap model) {

model.addAttribute("user", getPrincipal());

**return** "admin";

}

@RequestMapping(value = "/db", method = RequestMethod.***GET***)

**public** String dbaPage(ModelMap model) {

model.addAttribute("user", getPrincipal());

**return** "dba";

}

@RequestMapping(value = "/Access\_Denied", method = RequestMethod.***GET***)

**public** String accessDeniedPage(ModelMap model) {

model.addAttribute("user", getPrincipal());

**return** "accessDenied";

}

@RequestMapping(value = "/login", method = RequestMethod.***GET***)

**public** String loginPage() {

**return** "login";

}

@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";

}

@RequestMapping(value = "/newUser", method = RequestMethod.***GET***)

**public** String newRegistration(ModelMap model) {

User user = **new** User();

model.addAttribute("user", user);

**return** "newuser";

}

/\*

\* This method will be called on form submission, handling POST request It

\* also validates the user input

\*/

@RequestMapping(value = "/newUser", method = RequestMethod.***POST***)

**public** String saveRegistration(@Valid User user, BindingResult result,

ModelMap model) {

**if** (result.hasErrors()) {

System.***out***.println("There are errors");

**return** "newuser";

}

userService.save(user);

System.***out***.println("First Name : " + user.getFirstName());

System.***out***.println("Last Name : " + user.getLastName());

System.***out***.println("SSO ID : " + user.getSsoId());

System.***out***.println("Password : " + user.getPassword());

System.***out***.println("Email : " + user.getEmail());

System.***out***.println("Checking UsrProfiles....");

**if** (user.getUserProfiles() != **null**) {

**for** (UserProfile profile : user.getUserProfiles()) {

System.***out***.println("Profile : " + profile.getType());

}

}

model.addAttribute("success", "User " + user.getFirstName()

+ " has been registered successfully");

**return** "registrationsuccess";

}

**private** String getPrincipal() {

String userName = **null**;

Object principal = SecurityContextHolder.*getContext*()

.getAuthentication().getPrincipal();

**if** (principal **instanceof** UserDetails) {

userName = ((UserDetails) principal).getUsername();

} **else** {

userName = principal.toString();

}

**return** userName;

}

@ModelAttribute("roles")

**public** List<UserProfile> initializeProfiles() {

**return** userProfileService.findAll();

}

}

**Add SpringMVC Configuration Class**

@Configuration

@EnableWebMvc

@ComponentScan(basePackages = "in.spring4buddies.application")

**public** **class** WebApplicationConfiguration **extends** WebMvcConfigurerAdapter {

@Autowired

RoleToUserProfileConverter roleToUserProfileConverter;

@Override

**public** **void** configureViewResolvers(ViewResolverRegistry registry) {

InternalResourceViewResolver viewResolver = **new** InternalResourceViewResolver();

viewResolver.setViewClass(JstlView.**class**);

viewResolver.setPrefix("/WEB-INF/views/");

viewResolver.setSuffix(".jsp");

registry.viewResolver(viewResolver);

}

/\*

\* Configure ResourceHandlers to serve static resources like CSS/ Javascript

\* etc...

\*/

@Override

**public** **void** addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/static/\*\*").addResourceLocations(

"/static/");

}

/\*

\* Configure Converter to be used. In our example, we need a converter to

\* convert string values[Roles] to UserProfiles in newUser.jsp

\*/

@Override

**public** **void** addFormatters(FormatterRegistry registry) {

registry.addConverter(roleToUserProfileConverter);

}

}

Only interesting thing in this class is registration of a converter who will be responsible for converting an id into an Object.This is required to handle one-to-many relationship in JSP. During User creation, A User can be allocated multiple roles/userProfiles, and so we need a converter to map a particular role/userProfile to a user based on profile id. Below provided is the converter class.

Above Converter setup in **XML configuration** will be:

<mvc:annotation-driven conversion-service="conversionService"/>

<bean id="conversionService" class="org.springframework.format.support.FormattingConversionServiceFactoryBean">

    <property name="converters">

        <list>

            <bean id="roleToUserProfile" class="com.websystique.springsecurity.configuration.RoleToUserProfileConverter" />

        </list>

    </property>

</bean>

**Add SpringMVC Converter Class**

@Component

**public** **class** RoleToUserProfileConverter **implements**

Converter<Object, UserProfile> {

@Autowired

UserProfileService userProfileService;

/\*

\* Gets UserProfile by Id

\*

\* @see

\* org.springframework.core.convert.converter.Converter#convert(java.lang

\* .Object)

\*/

**public** UserProfile convert(Object element) {

Integer id = Integer.*parseInt*((String) element);

UserProfile profile = userProfileService.findById(id);

System.***out***.println("Profile : " + profile);

**return** profile;

}

/\*

\* Gets UserProfile by type

\*

\* @see

\* org.springframework.core.convert.converter.Converter#convert(java.lang

\* .Object)

\*/

/\*

\* public UserProfile convert(Object element) { String type =

\* (String)element; UserProfile profile=

\* userProfileService.findByType(type);

\* System.out.println("Profile ... : "+profile); return profile; }

\*/

}

**Add Initializer class**

**public** **class** WebApplicationInitializer **extends**

AbstractAnnotationConfigDispatcherServletInitializer {

@Override

**protected** Class<?>[] getRootConfigClasses() {

**return** **new** Class[] { WebApplicationConfiguration.**class** };

}

@Override

**protected** Class<?>[] getServletConfigClasses() {

**return** **null**;

}

@Override

**protected** String[] getServletMappings() {

**return** **new** String[] { "/" };

}

}

**Create Hibernate Configuration**

@Configuration

@EnableTransactionManagement

@ComponentScan({ "in.spring4buddies.application.configuration" })

@PropertySource(value = { "classpath:application.properties" })

**public** **class** HibernateConfiguration {

@Autowired

**private** Environment environment;

@Bean

**public** LocalSessionFactoryBean sessionFactory() {

LocalSessionFactoryBean sessionFactory = **new** LocalSessionFactoryBean();

sessionFactory.setDataSource(dataSource());

sessionFactory.setPackagesToScan(**new** String[] { "in.spring4buddies.application.model" });

sessionFactory.setHibernateProperties(hibernateProperties());

**return** sessionFactory;

}

@Bean

**public** DataSource dataSource() {

DriverManagerDataSource dataSource = **new** DriverManagerDataSource();

dataSource.setDriverClassName(environment.getRequiredProperty("jdbc.driverClassName"));

dataSource.setUrl(environment.getRequiredProperty("jdbc.url"));

dataSource.setUsername(environment.getRequiredProperty("jdbc.username"));

dataSource.setPassword(environment.getRequiredProperty("jdbc.password"));

**return** dataSource;

}

**private** Properties hibernateProperties() {

Properties properties = **new** Properties();

properties.put("hibernate.dialect", environment.getRequiredProperty("hibernate.dialect"));

properties.put("hibernate.show\_sql", environment.getRequiredProperty("hibernate.show\_sql"));

properties.put("hibernate.format\_sql", environment.getRequiredProperty("hibernate.format\_sql"));

**return** properties;

}

@Bean

@Autowired

**public** HibernateTransactionManager transactionManager(SessionFactory s) {

HibernateTransactionManager txManager = **new** HibernateTransactionManager();

txManager.setSessionFactory(s);

**return** txManager;

}

}

**Model classes are same as above.**

**public** **abstract** **class** AbstractDao<PK **extends** Serializable, T> {

**private** **final** Class<T> persistentClass;

@SuppressWarnings("unchecked")

**public** AbstractDao() {

**this**.persistentClass = (Class<T>) ((ParameterizedType) **this**.getClass()

.getGenericSuperclass()).getActualTypeArguments()[1];

}

@Autowired

**private** SessionFactory sessionFactory;

**protected** Session getSession() {

**return** sessionFactory.getCurrentSession();

}

@SuppressWarnings("unchecked")

**public** T getByKey(PK key) {

**return** (T) getSession().get(persistentClass, key);

}

**public** **void** persist(T entity) {

getSession().persist(entity);

}

**public** **void** delete(T entity) {

getSession().delete(entity);

}

**protected** Criteria createEntityCriteria() {

**return** getSession().createCriteria(persistentClass);

}

}

**public** **interface** UserDao {

**void** save(User user);

User findById(**int** id);

User findBySSO(String sso);

}

@Repository("userDao")

**public** **class** UserDaoImpl **extends** AbstractDao<Integer, User> **implements** UserDao {

**public** **void** save(User user) {

persist(user);

}

**public** User findById(**int** id) {

**return** getByKey(id);

}

**public** User findBySSO(String sso) {

Criteria crit = createEntityCriteria();

crit.add(Restrictions.*eq*("ssoId", sso));

**return** (User) crit.uniqueResult();

}

}

**public** **interface** UserProfileDao {

List<UserProfile> findAll();

UserProfile findByType(String type);

UserProfile findById(**int** id);

}

@Repository("userProfileDao")

**public** **class** UserProfileDaoImpl **extends** AbstractDao<Integer, UserProfile>

**implements** UserProfileDao {

@SuppressWarnings("unchecked")

**public** List<UserProfile> findAll() {

Criteria crit = createEntityCriteria();

crit.addOrder(Order.*asc*("type"));

**return** (List<UserProfile>) crit.list();

}

**public** UserProfile findById(**int** id) {

**return** getByKey(id);

}

**public** UserProfile findByType(String type) {

Criteria crit = createEntityCriteria();

crit.add(Restrictions.*eq*("type", type));

**return** (UserProfile) crit.uniqueResult();

}

}

**Create Service Layer**

**public** **interface** UserService {

**void** save(User user);

User findById(**int** id);

User findBySso(String sso);

}

@Service("userService")

@Transactional

**public** **class** UserServiceImpl **implements** UserService {

@Autowired

**private** UserDao dao;

@Autowired

**private** PasswordEncoder passwordEncoder;

**public** **void** save(User user) {

user.setPassword(passwordEncoder.encode(user.getPassword()));

dao.save(user);

}

**public** User findById(**int** id) {

**return** dao.findById(id);

}

**public** User findBySso(String sso) {

**return** dao.findBySSO(sso);

}

}

**public** **interface** UserProfileService {

List<UserProfile> findAll();

UserProfile findByType(String type);

UserProfile findById(**int** id);

}

@Service("userProfileService")

@Transactional

**public** **class** UserProfileServiceImpl **implements** UserProfileService {

@Autowired

UserProfileDao dao;

**public** List<UserProfile> findAll() {

**return** dao.findAll();

}

**public** UserProfile findByType(String type) {

**return** dao.findByType(type);

}

**public** UserProfile findById(**int** id) {

**return** dao.findById(id);

}

}

login.jsp

<%@ 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>Login page</title>

<link href=*"*<c:url value=*'/static/css/bootstrap.css'* />*"* rel=*"stylesheet"*></link>

<link href=*"*<c:url value=*'/static/css/app.css'* />*"* rel=*"stylesheet"*></link>

<link rel=*"stylesheet"* type=*"text/css"* href=*"//cdnjs.cloudflare.com/ajax/libs/font-awesome/4.2.0/css/font-awesome.css"* />

</head>

<body>

<div id=*"mainWrapper"*>

<div class=*"login-container"*>

<div class=*"login-card"*>

<div class=*"login-form"*>

<c:url var=*"loginUrl"* value=*"/login"* />

<form action=*"*${loginUrl}*"* method=*"post"* class=*"form-horizontal"*>

<c:if test=*"*${param.error != null}*"*>

<div class=*"alert alert-danger"*>

<p>Invalid username and password.</p>

</div>

</c:if>

<c:if test=*"*${param.logout != null}*"*>

<div class=*"alert alert-success"*>

<p>You have been logged out successfully.</p>

</div>

</c:if>

<div class=*"input-group input-sm"*>

<label class=*"input-group-addon"* for=*"username"*><i class=*"fa fa-user"*></i></label>

<input type=*"text"* class=*"form-control"* id=*"username"* name=*"ssoId"* placeholder=*"Enter Username"* required>

</div>

<div class=*"input-group input-sm"*>

<label class=*"input-group-addon"* for=*"password"*><i class=*"fa fa-lock"*></i></label>

<input type=*"password"* class=*"form-control"* id=*"password"* name=*"password"* placeholder=*"Enter Password"* required>

</div>

<input type=*"hidden"* name=*"*${\_csrf.parameterName}*"*

value=*"*${\_csrf.token}*"* />

<div class=*"form-actions"*>

<input type=*"submit"*

class=*"btn btn-block btn-primary btn-default"* value=*"Log in"*>

</div>

</form>

</div>

</div>

</div>

</div>

</body>

</html>

welcome.jsp

<%@ 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>Welcome page</title>

<link href=*"*<c:url value=*'/static/css/bootstrap.css'* />*"* rel=*"stylesheet"*></link>

<link href=*"*<c:url value=*'/static/css/app.css'* />*"* rel=*"stylesheet"*></link>

</head>

<body>

<div class=*"success"*>

Greeting : ${greeting}

This is a welcome page.

</div>

</body>

</html>

admin.jsp

<%@ 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>

<link href=*"*<c:url value=*'/static/css/bootstrap.css'* />*"* rel=*"stylesheet"*></link>

<link href=*"*<c:url value=*'/static/css/app.css'* />*"* rel=*"stylesheet"*></link>

</head>

<body>

<div class=*"success"*>

Dear <strong>${user}</strong>, Welcome to Admin Page.

<br/>

Would you like to <a href=*"*<c:url value=*'/newUser'* />*"*>Add Some Users</a> to keep yourself busy?

<br/>

<a href=*"*<c:url value=*"/logout"* />*"*>Logout</a>

</div>

</body>

</html>

dba.jsp

<%@ 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>DBA page</title>

<link href=*"*<c:url value=*'/static/css/bootstrap.css'* />*"* rel=*"stylesheet"*></link>

<link href=*"*<c:url value=*'/static/css/app.css'* />*"* rel=*"stylesheet"*></link>

</head>

<body>

<div class=*"success"*>

Dear <strong>${user}</strong>, Welcome to DBA Page.

<br/>

<a href=*"*<c:url value=*"/logout"* />*"*>Logout</a>

</div>

</body>

</html>

newuser.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"* pageEncoding=*"ISO-8859-1"*%>

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

<%@ 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>User Registration Form</title>

<link href=*"*<c:url value=*'/static/css/bootstrap.css'* />*"* rel=*"stylesheet"*></link>

<link href=*"*<c:url value=*'/static/css/app.css'* />*"* rel=*"stylesheet"*></link>

</head>

<body>

<div class=*"form-container"*>

<h1>New User Registration Form</h1>

<form:form method=*"POST"* modelAttribute=*"user"* class=*"form-horizontal"*>

<div class=*"row"*>

<div class=*"form-group col-md-12"*>

<label class=*"col-md-3 control-lable"* for=*"firstName"*>First Name</label>

<div class=*"col-md-7"*>

<form:input type=*"text"* path=*"firstName"* id=*"firstName"* class=*"form-control input-sm"*/>

<div class=*"has-error"*>

<form:errors path=*"firstName"* class=*"help-inline"*/>

</div>

</div>

</div>

</div>

<div class=*"row"*>

<div class=*"form-group col-md-12"*>

<label class=*"col-md-3 control-lable"* for=*"lastName"*>Last Name</label>

<div class=*"col-md-7"*>

<form:input type=*"text"* path=*"lastName"* id=*"lastName"* class=*"form-control input-sm"*/>

<div class=*"has-error"*>

<form:errors path=*"lastName"* class=*"help-inline"*/>

</div>

</div>

</div>

</div>

<div class=*"row"*>

<div class=*"form-group col-md-12"*>

<label class=*"col-md-3 control-lable"* for=*"ssoId"*>SSO ID</label>

<div class=*"col-md-7"*>

<form:input type=*"text"* path=*"ssoId"* id=*"ssoId"* class=*"form-control input-sm"*/>

<div class=*"has-error"*>

<form:errors path=*"ssoId"* class=*"help-inline"*/>

</div>

</div>

</div>

</div>

<div class=*"row"*>

<div class=*"form-group col-md-12"*>

<label class=*"col-md-3 control-lable"* for=*"password"*>Password</label>

<div class=*"col-md-7"*>

<form:input type=*"password"* path=*"password"* id=*"password"* class=*"form-control input-sm"*/>

<div class=*"has-error"*>

<form:errors path=*"password"* class=*"help-inline"*/>

</div>

</div>

</div>

</div>

<div class=*"row"*>

<div class=*"form-group col-md-12"*>

<label class=*"col-md-3 control-lable"* for=*"email"*>Email</label>

<div class=*"col-md-7"*>

<form:input type=*"text"* path=*"email"* id=*"email"* class=*"form-control input-sm"*/>

<div class=*"has-error"*>

<form:errors path=*"email"* class=*"help-inline"*/>

</div>

</div>

</div>

</div>

<div class=*"row"*>

<div class=*"form-group col-md-12"*>

<label class=*"col-md-3 control-lable"* for=*"userProfiles"*>Roles</label>

<div class=*"col-md-7"*>

<form:select path=*"userProfiles"* items=*"*${roles}*"* multiple=*"true"* itemValue=*"id"* itemLabel=*"type"* class=*"form-control input-sm"*/>

<div class=*"has-error"*>

<form:errors path=*"userProfiles"* class=*"help-inline"*/>

</div>

</div>

</div>

</div>

<div class=*"row"*>

<div class=*"form-actions floatRight"*>

<input type=*"submit"* value=*"Register"* class=*"btn btn-primary btn-sm"*> or <a href=*"*<c:url value=*'/admin'* />*"*>Cancel</a>

</div>

</div>

</form:form>

</div>

</body>

</html>

registrationsuccess.jsp

<%@ 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>User Registration Form</title>

<link href=*"*<c:url value=*'/static/css/bootstrap.css'* />*"* rel=*"stylesheet"*></link>

<link href=*"*<c:url value=*'/static/css/app.css'* />*"* rel=*"stylesheet"*></link>

</head>

<body>

<div class=*"success"*>

Confirmation message : ${success}

<br>

Would you like to <a href=*"*<c:url value=*'/newUser'* />*"*>Add More Users</a>?

<br/>

Go to <a href=*"*<c:url value=*'/admin'* />*"*>Admin Page</a> OR <a href=*"*<c:url value=*"/logout"* />*"*>Logout</a>

</div>

</body>

</html>

accessDenied.jsp

<%@ 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>AccessDenied page</title>

<link href=*"*<c:url value=*'/static/css/bootstrap.css'* />*"* rel=*"stylesheet"*></link>

<link href=*"*<c:url value=*'/static/css/app.css'* />*"* rel=*"stylesheet"*></link>

</head>

<body>

Dear <strong>${user}</strong>, You are not authorized to access this page.

<br/>

<a href=*"*<c:url value=*"/home"* />*"*>Go to home</a> OR <a href=*"*<c:url value=*"/logout"* />*"*>Logout</a>

</body>

</html>

**Spring Security 4 Remember Me**

In Remember-me or **persistent-login authentication**, Applications remember the identity of user between sessions. Basically, during login, when you ask for Remember-Me support, application will **send a cookie to the browser**during login. This cookie will be stored at browser side and will remain there for certain period(defined by cookie lifetime). Next time when you try to access the application, browser will **detect the cookie** (if still valid) and user will be automatically logged in, without providing userid/password e.g.

Spring Security provides two implementations for Remember-Me :

* Simple Hash-Based Token Approach : It uses hashing to preserve the security of cookie-based tokens
* Persistent Token Approach : It uses a database or other persistent storage mechanism to store the generated tokens

In this post, we will be discussing about **Persistent Token Approach**

**Modifications compare to normal login posts:**

**1.**With Persistent Token Approach, the database should contain a **persistent\_logins** table, created using the following SQL (or equivalent):

|  |
| --- |
| CREATE TABLE persistent\_logins (      username VARCHAR(64) NOT NULL,      series VARCHAR(64) NOT NULL,      token VARCHAR(64) NOT NULL,      last\_used TIMESTAMP NOT NULL,      PRIMARY KEY (series)  ); |

This table contains the username, last\_used timestamp of active remember-me, security token and series information which are internals of Spring Bcrypt implementation. For further details, please refer [here](http://jaspan.com/improved_persistent_login_cookie_best_practice).

**2.**Configure Remember-Me in Spring Security

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter {

@Autowired

@Qualifier("customUserDetailsService")

UserDetailsService userDetailsService;

@Autowired

DataSource dataSource;

@Autowired

**public** **void** configureGlobalSecurity(AuthenticationManagerBuilder auth)

**throws** Exception {

auth.userDetailsService(userDetailsService);

}

@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()

.rememberMe().rememberMeParameter("remember-me")

.tokenRepository(persistentTokenRepository())

.tokenValiditySeconds(86400)

.and()

.csrf()

.and()

.exceptionHandling().accessDeniedPage("/Access\_Denied");

}

@Bean

**public** PersistentTokenRepository persistentTokenRepository() {

JdbcTokenRepositoryImpl tokenRepositoryImpl = **new** JdbcTokenRepositoryImpl();

tokenRepositoryImpl.setDataSource(dataSource);

**return** tokenRepositoryImpl;

}

}

Notice how we called rememberMe() to configure Remember-Me authenticationm providing HTTP parameter name attached with remember-me checkbox from view(we will see in view). We have also specified the tokenRepository(where the token will be stored) to be used, and how long (in seconds) this token remains valid ( we have provided 1 day). To configure Repository itself we have injected a DataSource here.This is all you need to activate Remember-Me in your Spring Security based application.

Optionally, you can use Spring Security Built-in expressions along with Spring security tags in your view to customize/show/hide specific logic based on Remember-Me or full-authentication.

<http auto-config=*"true"*>

<intercept-url pattern=*"/"* access=*"permitAll"* />

<intercept-url pattern=*"/home"* access=*"permitAll"* />

<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"* />

<csrf />

</http>

<authentication-manager>

<authentication-provider user-service-ref=*"customUserDetailsService"* />

</authentication-manager>

<remember-me remember-me-parameter=*"remember-me"*

remember-me-cookie=*"remember-me"* token-validity-seconds=*"86400"*

data-source-ref=*"dataSource"* />

<bean id=*"customUserDetailsService"*

class=*"*in.spring4buddies.application*.service.CustomUserDetailsService"* />

**Register the springSecurityFilter with war**

**public** **class** SecurityInitializer **extends**

AbstractSecurityWebApplicationInitializer {

}

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

<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>

@Service("customUserDetailsService")

**public** **class** CustomUserDetailsService **implements** UserDetailsService {

@Autowired

**private** UserService userService;

@Transactional(readOnly = **true**)

**public** UserDetails loadUserByUsername(String ssoId)

**throws** UsernameNotFoundException {

User user = userService.findBySso(ssoId);

System.***out***.println("User : " + user);

**if** (user == **null**) {

System.***out***.println("User not found");

**throw** **new** UsernameNotFoundException("Username not found");

}

**return** **new** org.springframework.security.core.userdetails.User(

user.getSsoId(), user.getPassword(), user.getState().equals(

"Active"), **true**, **true**, **true**,

getGrantedAuthorities(user));

}

**private** List<GrantedAuthority> getGrantedAuthorities(User user) {

List<GrantedAuthority> authorities = **new** ArrayList<GrantedAuthority>();

**for** (UserProfile userProfile : user.getUserProfiles()) {

System.***out***.println("UserProfile : " + userProfile);

authorities.add(**new** SimpleGrantedAuthority("ROLE\_"

+ userProfile.getType()));

}

System.***out***.print("authorities :" + authorities);

**return** authorities;

}

}

Other classes are same as above.