




[WHITE PAPER] How do we describe the APIs powering the growth of realtime data? Learn more about...

[Read Now](#)

DZone > Security Zone > Spring Security 5 Form Login With Database Provider

Spring Security 5 Form Login With Database Provider

by Yogen Rai  MVB · Jan. 16, 19 · Security Zone · Tutorial

Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. It is one of the most powerful and highly customizable authentication and access control frameworks in the Java ecosystem.

This article is going to focus on **Spring Security Form Login** which is one of the most necessary parts of web applications. The example I am presenting here is a part of **pdf (Programming Discussion Forum)**, a web application built with Spring 5, Hibernate 5, Tiles, and i18n.

1. Setting Up Maven Dependencies

The main Maven dependencies required for form login are `spring-security-web` and `spring-security-config`. However, to provide database backed `UserDetailsService`, we need to have dependencies to support that as well.

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>com.bitMiners</groupId>
5   <artifactId>pdf-app</artifactId>
6   <version>0.0.1</version>
7   <packaging>war</packaging>
8
9   <properties>
10    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
11    <spring.version>5.1.3.RELEASE</spring.version>
12    <hibernate.version>5.2.17.Final</hibernate.version>
13    <c3p0.version>0.9.5.2</c3p0.version>
14    <spring-security.version>5.1.2.RELEASE</spring-security.version>
15  </properties>
16
17  <dependencies>
18    <!-- spring -->
19    <dependency>
20      <groupId>org.springframework</groupId>
21      <artifactId>spring-tx</artifactId>
```

```

2      <version>${spring.version}</version>
3    </dependency>
4    <dependency>
5      <groupId>org.springframework</groupId>
6      <artifactId>spring-orm</artifactId>
7      <version>${spring.version}</version>
8    </dependency>
9    <dependency>
0      <groupId>org.springframework</groupId>
1      <artifactId>spring-webmvc</artifactId>
2      <version>${spring.version}</version>
3    </dependency>
4
5    <!--security-->
6    <dependency>
7      <groupId>org.springframework.security</groupId>
8      <artifactId>spring-security-config</artifactId>
9      <version>${spring-security.version}</version>
0      <exclusions>
1        <exclusion>
2          <artifactId>spring-asm</artifactId>
3          <groupId>org.springframework</groupId>
4        </exclusion>
5      </exclusions>
6    </dependency>
7    <dependency>
8      <groupId>org.springframework.security</groupId>
9      <artifactId>spring-security-web</artifactId>
0      <version>${spring-security.version}</version>
1    </dependency>
2    <dependency>
3      <groupId>org.springframework.security</groupId>
4      <artifactId>spring-security-taglibs</artifactId>
5      <version>${spring-security.version}</version>
6    </dependency>
7
8    <!-- servlets and jps -->
9    <dependency>
0      <groupId>javax.servlet</groupId>
1      <artifactId>jstl</artifactId>
2      <version>1.2</version>
3    </dependency>
4    <dependency>
5      <groupId>javax.servlet</groupId>
6      <artifactId>javax.servlet-api</artifactId>
7      <version>4.0.1</version>
8      <scope>provided</scope>
9    </dependency>
0
1    <dependency>
2      <groupId>org.apache.tiles</groupId>
3      <artifactId>tiles-extras</artifactId>
4      <version>3.0.8</version>
5      <exclusions>
6        <exclusion>
7          <groupId>org.slf4j</groupId>
8          <artifactId>jcl-over-slf4j</artifactId>
9        </exclusion>
0      </exclusions>

```

```

1    </dependency>
2    <!-- Hibernate -->
3    <dependency>
4        <groupId>org.hibernate</groupId>
5        <artifactId>hibernate-core</artifactId>
6        <version>${hibernate.version}</version>
7    </dependency>
8    <!-- Hibernate-C3P0 Integration -->
9    <dependency>
10        <groupId>org.hibernate</groupId>
11        <artifactId>hibernate-c3p0</artifactId>
12        <version>${hibernate.version}</version>
13    </dependency>
14
15    <!-- c3p0 -->
16    <dependency>
17        <groupId>com.mchange</groupId>
18        <artifactId>c3p0</artifactId>
19        <version>${c3p0.version}</version>
20    </dependency>
21
22    <dependency>
23        <groupId>mysql</groupId>
24        <artifactId>mysql-connector-java</artifactId>
25        <version>5.1.34</version>
26    </dependency>
27 </dependencies>
28 </project>

```

You can checkout pom.xml in my GitHub repository for other plugin details.

2. Entity Mapping

Let us create two @Entity classes, named as User and Authority , to map with database tables as follows.

```

1 package com.bitMiners.pdf.domain;
2
3 import org.hibernate.validator.constraints.NotEmpty;
4
5 import javax.persistence.*;
6 import java.io.Serializable;
7 import java.util.Date;
8 import java.util.HashSet;
9 import java.util.Set;
10
11 @Entity
12 @Table(name = "user")
13 public class User implements Serializable {
14     @Id
15     @GeneratedValue(strategy = GenerationType.IDENTITY)
16     private Integer id;
17
18     @NotEmpty
19     @Column(nullable = false, unique = true)
20     private String username;
21
22     @SetOf
23     private Set<Authority> authorities;
24 }

```

```

1  @NotEmpty
2  private String password;
3
4  private Date dateCreated;
5
6  @ManyToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
7  @JoinTable(name = "user_authority",
8      joinColumns = { @JoinColumn(name = "user_id") },
9      inverseJoinColumns = { @JoinColumn(name = "authority_id") })
10 private Set<Authority> authorities = new HashSet<>();
11
12 public User() {
13 }
14 // getters and setters
15 }

```

```

1 package com.bitMiners.pdf.domain;
2
3 import com.bitMiners.pdf.domain.types.AuthorityType;
4
5 import javax.persistence.*;
6
7 @Entity
8 @Table(name = "authority")
9 public class Authority {
10     @Id
11     @GeneratedValue(strategy = GenerationType.AUTO)
12     private Integer id;
13
14     @Enumerated(EnumType.STRING)
15     private AuthorityType name;
16
17     public Authority() {}
18
19     public Authority(AuthorityType name) {
20         this.name = name;
21     }
22     // getters and setters
23 }

```

It is always good to have authorities as configurable value, but for ease, let us define this as an enum for now.

```

1 package com.bitMiners.pdf.domain.types;
2
3 public enum AuthorityType {
4     ROLE_ADMIN,
5     ROLE_USER
6 }

```

3. Populate a Database Table

Once entities are defined, you can put `import.sql` under the resources folder in the

project structure so that Hibernate can populate tables with SQL statements inside.

```

1 INSERT INTO `authority`(`name`, `id`) VALUES ('ROLE_ADMIN', 1);
2 INSERT INTO `authority`(`name`, `id`) VALUES ('ROLE_USER', 2);
3
4 INSERT INTO `user_authority`(`authority_id`, `user_id`) VALUES (1, 1);
5 INSERT INTO `user_authority`(`authority_id`, `user_id`) VALUES (2, 2);
6
7 INSERT INTO `user` (`id`, `username`, `password`, `dateCreated`) VALUES (1,'ironman','$2a$10$jXlure,
8
9 INSERT INTO `user` (`id`, `username`, `password`, `dateCreated`) VALUES (2,'rabi','$2a$10$0tFJKcOV/1

```

4. Retrieving a User

In order to retrieve a user associated with a username, let us create `UserRepositoryImpl` which implements `UserRepository` as below:

```

1 package com.bitMiners.pdf.repositories;
2 import com.bitMiners.pdf.domain.User;
3
4 public interface UserRepository extends CrudRepository<User, Integer> {
5     User getUserByUsername(String username);
6 }

1 package com.bitMiners.pdf.repositories.impl;
2
3 import com.bitMiners.pdf.domain.User;
4 import com.bitMiners.pdf.repositories.UserRepository;
5 import org.hibernate.SessionFactory;
6 import org.hibernate.query.NativeQuery;
7 import org.hibernate.query.Query;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.stereotype.Repository;
0 import org.springframework.transaction.annotation.Transactional;
1
2 @Repository
3 @Transactional
4 public class UserRepositoryImpl implements UserRepository {
5
6     @Autowired
7     private SessionFactory sessionFactory;
8
9     @Override
0     public User getUserByUsername(String username) {
1         Query<User> query = sessionFactory.getCurrentSession().createQuery("FROM User u where u.username = :username");
2         query.setParameter("username", username);
3         return query.uniqueResult();
4     }
5 }

```

5. UserDetailsService Implementation

We need to implement

the `org.springframework.security.core.userdetails.UserDetailsService` interface in order to provide our own service implementation. I have added `UserDetailsServiceImpl` which implements `UserDetailsService` to retrieve the `User` object using the *repository*, and, if it exists, wrap it into a `PdfUserDetails` object, which implements `UserDetails`, and returns it as below:

```

1 package com.bitMiners.pdf.services.impl;
2
3 import com.bitMiners.pdf.domain.PdfUserDetails;
4 import com.bitMiners.pdf.domain.User;
5 import com.bitMiners.pdf.repositories.UserRepository;
6 import org.slf4j.Logger;
7 import org.slf4j.LoggerFactory;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.security.core.userdetails.UserDetails;
10 import org.springframework.security.core.userdetails.UserDetailsService;
11 import org.springframework.security.core.userdetails.UsernameNotFoundException;
12 import org.springframework.stereotype.Service;
13 import org.springframework.transaction.annotation.Transactional;
14
15 @Service("userDetailsService")
16 public class UserDetailsServiceImpl implements UserDetailsService {
17     private static final Logger log = LoggerFactory.getLogger(UserDetailsServiceImpl.class);
18
19     @Autowired
20     private UserRepository userRepository;
21
22     @Transactional(readOnly = true)
23     @Override
24     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
25         User user = userRepository.getUserByUsername(username);
26
27         if (user == null) {
28             throw new UsernameNotFoundException("User not found.");
29         }
30
31         log.info("loadUserByUsername() : {}", username);
32
33         return new PdfUserDetails(user);
34     }
35 }

```

`PdfUserDetails` model is defined as below:

```

1 package com.bitMiners.pdf.domain;
2
3 import org.springframework.security.core.GrantedAuthority;
4 import org.springframework.security.core.authority.SimpleGrantedAuthority;
5 import org.springframework.security.core.userdetails.UserDetails;
6
7 import java.util.Collection;
8 import java.util.stream.Collectors;
9
10

```

```

0 public class PdfUserDetails implements UserDetails {
1     private User user;
2
3     public PdfUserDetails(User user) {
4         this.user = user;
5     }
6
7     @Override
8     public Collection<? extends GrantedAuthority> getAuthorities() {
9         return user.getAuthorities().stream().map(authority -> new SimpleGrantedAuthority(authority
0     }
1
2     public int getId() {
3         return user.getId();
4     }
5
6     @Override
7     public String getPassword() {
8         return user.getPassword();
9     }
0
1     @Override
2     public String getUsername() {
3         return user.getUsername();
4     }
5
6     @Override
7     public boolean isAccountNonExpired() {
8         return true;
9     }
0
1     @Override
2     public boolean isAccountNonLocked() {
3         return true;
4     }
5
6     @Override
7     public boolean isCredentialsNonExpired() {
8         return true;
9     }
0
1     @Override
2     public boolean isEnabled() {
3         return true;
4     }
5
6     public User getUserDetails() {
7         return user;
8     }
9 }

```

6. Spring MVC Controller

Let us add `LoginController` to handle custom success and failure during login.

```

1 package com.bitMiners.pdf.controllers;
2

```

```

3 import com.bitMiners.pdf.domain.PdfUserDetails;
4 import com.bitMiners.pdf.domain.User;
5 import org.apache.logging.log4j.LogManager;
6 import org.apache.logging.log4j.Logger;
7 import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
8 import org.springframework.security.core.context.SecurityContextHolder;
9 import org.springframework.stereotype.Controller;
0 import org.springframework.ui.Model;
1 import org.springframework.web.bind.annotation.RequestMapping;
2 import org.springframework.web.bind.annotation.RequestMethod;
3 import org.springframework.web.bind.annotation.SessionAttributes;
4 import org.springframework.web.bind.support.SessionStatus;
5
6 import javax.servlet.http.HttpSession;
7
8 @SessionAttributes({"currentUser"})
9 @Controller
0 public class LoginController {
1     private static final Logger log = LogManager.getLogger(LoginController.class);
2
3     @RequestMapping(value = "/login", method = RequestMethod.GET)
4     public String login() {
5         return "login";
6     }
7
8     @RequestMapping(value = "/loginFailed", method = RequestMethod.GET)
9     public String loginError(Model model) {
0         log.info("Login attempt failed");
1         model.addAttribute("error", "true");
2         return "login";
3     }
4
5     @RequestMapping(value = "/logout", method = RequestMethod.GET)
6     public String logout(SessionStatus session) {
7         SecurityContextHolder.getContext().setAuthentication(null);
8         session.setComplete();
9         return "redirect:/welcome";
0     }
1
2     @RequestMapping(value = "/postLogin", method = RequestMethod.POST)
3     public String postLogin(Model model, HttpSession session) {
4         log.info("postLogin()");
5
6         // read principal out of security context and set it to session
7         UsernamePasswordAuthenticationToken authentication = (UsernamePasswordAuthenticationToken) ;
8         validatePrinciple(authentication.getPrincipal());
9         User loggedInUser = ((PdfUserDetails) authentication.getPrincipal()).getUserDetails();
0
1         model.addAttribute("currentUser", loggedInUser.getUsername());
2         session.setAttribute("userId", loggedInUser.getId());
3         return "redirect:/wallPage";
4     }
5
6     private void validatePrinciple(Object principal) {
7         if (!(principal instanceof PdfUserDetails)) {
8             throw new IllegalArgumentException("Principal can not be null!");
9         }
0     }
1 }

```


7. Spring Security Java Configuration

Now, let's add a Spring Security configuration class that extends `WebSecurityConfigurerAdapter`. Here, the addition of the annotation `@EnableWebSecurity` provides Spring Security and also provides MVC integration support.

```

1 package com.bitMiners.pdf.config;
2
3 import com.bitMiners.pdf.services.impl.UserDetailsServiceImpl;
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
6 import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
7 import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
8 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
9 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
10 import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
11 import org.springframework.security.core.userdetails.UserDetailsService;
12 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
13
14 @EnableWebSecurity
15 @EnableGlobalMethodSecurity(prePostEnabled = true)
16 public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
17
18     @Bean
19     public UserDetailsService userDetailsService() {
20         return new UserDetailsServiceImpl();
21     }
22
23     @Bean
24     public BCryptPasswordEncoder passwordEncoder() {
25         return new BCryptPasswordEncoder();
26     }
27
28     @Bean
29     public DaoAuthenticationProvider authenticationProvider() {
30         DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
31         authProvider.setUserDetailsService(userDetailsService());
32         authProvider.setPasswordEncoder(passwordEncoder());
33         return authProvider;
34     }
35
36     @Override
37     protected void configure(AuthenticationManagerBuilder auth) {
38         auth.authenticationProvider(authenticationProvider());
39     }
40
41     @Override
42     protected void configure(HttpSecurity http) throws Exception {
43         http
44             .authorizeRequests().antMatchers("/wallPage").hasAnyRole("ADMIN", "USER")
45             .and()
46             .authorizeRequests().antMatchers("/login", "/resource/**").permitAll()
47             .and()

```

```

8      .formLogin().loginPage("/login").usernameParameter("username").passwordParameter("password")
9          .loginProcessingUrl("/doLogin")
0          .successForwardUrl("/postLogin")
1          .failureUrl("/loginFailed")
2          .and()
3          .logout().logoutUrl("/doLogout").logoutSuccessUrl("/logout").permitAll()
4          .and()
5          .csrf().disable();
6    }
7}

```

The above configuration has the following elements to create the login form:

authorizeRequests() is how we allow anonymous access to `/login,/resource/**` and secure the rest of the resource paths.

formLogin() is used to define the login form with username and password input. This has other methods that we can use to configure the behavior of the form login:

- *loginPage()* – the custom login page url.
- *loginProcessingUrl()* – the URL to which we submit the username and password.
- *defaultSuccessUrl()* – the landing page after a successful login.
- *failureUrl()* – the landing page after an unsuccessful login.

Authentication Manager is `DaoAuthenticationProvider`, backed by `UserDetailsService` which is accessing a database via the `UserRepository` repository.

BCryptPasswordEncoder is a password encoder.

8. Add Spring Security to a Web Application

Now, we need to let Spring know about our Spring Security Config by registering on root config as below:

```

1 package com.bitMiners.pdf.config;
2
3 import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;
4
5 public class AppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
6
7     @Override
8     protected Class<?>[] getRootConfigClasses() {
9         return new Class[] { HibernateConfig.class, WebSecurityConfig.class };
0     }
1
2     @Override
3     protected Class<?>[] getServletConfigClasses() {
4         return new Class[] { WebMvcConfig.class };
5     }
6 }

```

```

5    }
6
7    @Override
8    protected String[] getServletMappings() {
9        return new String[] { "/" };
10   }
11}

```

I have skipped including `HibernateConfig.class` and `WebMvcConfig.class` here for brevity. But you can find them on the GitHub repository.

8. Adding a Login Form

Let us add a login form in the `login.jsp` file, as shown below:

```

1 <div class="panel-body">
2   <form action="doLogin" method="post">
3     <fieldset>
4       <legend>Please sign in</legend>
5
6       <c:if test="${not empty error}">
7         <div class="alert alert-danger">
8           <spring:message code="AbstractUserDetailsAuthenticationProvider.badCredentials",
9             <br/>
10          </div>
11        </c:if>
12        <div class="form-group">
13          <input class="form:input-large" placeholder="User Name"
14            name='username' type="text">
15        </div>
16        <div class="form-group">
17          <input class="form:input-large" placeholder="Password"
18            name='password' type="password" value="">
19        </div>
20        <input class="btn" type="submit"
21          value="Login">
22      </fieldset>
23    </form>
24  </div>

```

Note here, the action `doLogin` for form submission is same as of the login processing URL in the security config above.

9. Demoing App

<http://localhost:8080/login> takes you to the login page:



Home Forum Users signup login

Programming Discussion Forum

Adding cutting edge concept to programming

Please sign in

Login

[English](#) | [Nepali](#) | [Chinese](#)

© BitMiners 2023

If you try with bad credentials, you'll see the error message as below:

[Home](#)[Forum](#)[Users](#)[signup](#)[login](#)

Programming Discussion Forum

Adding cutting edge concept to programming

Please sign in

Username or password didn't match!

Login

If login authentication is successful, then it redirects to the homepage.

If you click on logout button once you are logged in, then it will take you to the home page after clearing all the session.

10. Conclusion

In this example, we configured a Spring Security form login authentication process and saw how easily we can configure advanced authentication processes with the methods available.

The project is available on GitHub. You can clone it and run it with Vagrant or an IDE configuration or however you want.

Happy coding!

Get the fastest log management and analysis with Gra
source or enterprise edition free up to 5GB per day.

Presented by Gravelon

<https://dzone.com/articles/spring-security-5-form-login-with-database-provide>