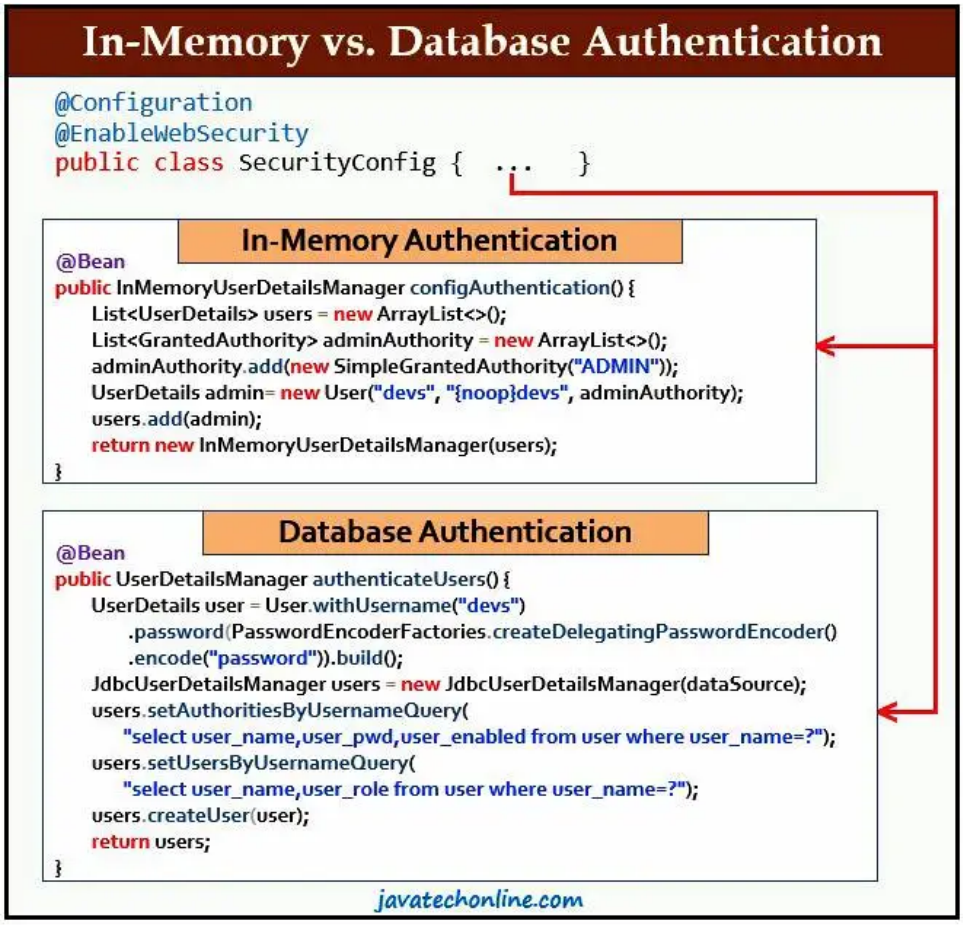
Friday, February 17, 2023

Making Java easy to learn

Java Technology and Beyond

How To Implement Security In Spring Boot Project?



Now a days, almost every client demands for implementation of powerful security feature in real time application. Demand of security feature is very valid to maintain confidentiality, integrity and availability. There are many types of security in the real world, but we as a developer will focus on the Application/Software Security.

Furthermore, in Application Security, our job is to ensure basically two things. First, only valid user can access the application. Second, If the user is valid, he/she can access only permitted data/information in that application. I consider, there is nothing to explain more about them as you must already be aware of these two terminologies i.e. Authentication & Authorization. You might already have guessed what we will discuss in our current topic 'How to implement Security in S

TOP

Boot Project?'

In this article, we will start learning with basic fundamentals. Following step by step, we will end it until we feel confident in implementing Security features in a Spring Boot Application. Accordingly, Let's start discussing about 'How to implement Security in Spring Boot Project?' step by step. A Series of tutorials on Spring Boot Security are on Spring Boot Tutorials page.

We have covered the example of 'How to implement Security in Spring Boot Project?' for both versions: With WebSecurityConfigurerAdapter and Without WebSecurityConfigurerAdapter in this article. Moreover, the example of 'How to implement Security in Spring Boot Project?' Using Spring Boot 3.0 and higher versions is also covered.

Table of Contents

TOP

What will you learn from this article?

1) Why do we need to implement security in a Spring Boot application?

2) How does security work internally in a Spring Boot application?

3) What is a role of javax.servlet.Filter in implementing security in a Spring Boot application?

4) How many types of authorization are used in a Spring Boot Project?

5) How many ways are there to implement security in a Spring Boot Project?

6) What are the steps to implement web security in a Spring Boot Project?

7) Also, How to use @EnableWebSecurity, @Configuration, @Bean in a Spring Boot Project?

8) Example of How to implement in-memory authentication security

9) Example of How to implement JDBC authentication security

10) How to work with Thymeleaf in Spring Boot Project ?

11) How to test the security-enabled feature ?

12) On the contrary, How to disable security feature of the application?

13) Last but not the least you will learn “How to implement Security in Spring Boot Project?” using WebSecurityConfigurerAdapter and Without using WebSecurityConfigurerAdapter.

TOP

Why do we need Security in an Application?

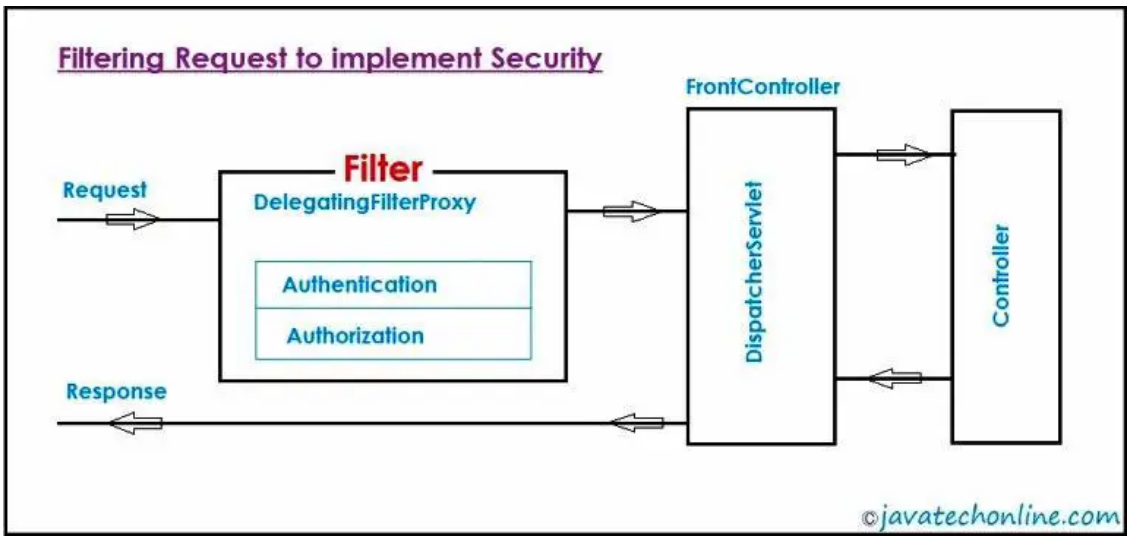
Now a day's data are at most risk as malicious attacks moving their focus to applications/software and mobiles/devices from operating systems & networks. Also, from the business/client perspective, application security plays an important role in maintaining trust, building brand image & mitigating risks. In fact, without any one of these don't think about a successful business. In conclusion, it does not matter if you are creating the app for in-house use, selling purpose, or buying purpose, Security is the most important feature for each & every app.

As per the most recent Verizon Data Breach Investigations Report-2020, 43% of all data breaches were attacks on web applications. This represented a doubling of the number from 2019. In addition, 86% of all breaches were financially motivated.

How does security work internally in a Spring Boot Application ?

Fundamentally, spring security works on a concept called JAAS(Java Authentication and Authorization Services). In brief, it works on Filter (javax.servlet.Filter) concept. We all know that we can use the Filters if we want to apply some pre-processing logic before a servlet request. The same concept has been applied to implement security features in Spring boot projects as well. DelegatingFilterProxy is a predefined class under package org.springframework.web.filter provided by Spring Security module which acts as a filter here.

As shown in the diagram below when a user sends a request to the application, it passes through a security filter before going to DispatcherServlet. If user validates the security of this filter, then only the request will go to DispatcherServlet to serve the user's purpose.



How many types of authorization are used in Spring B

TOP

Application?

There are four mostly used types of authorization implementation available. They are :

1) permitAll

permitAll represents that No need of any authorization to access current page.
Example URLs: /login, /home, /contactUs, /aboutUs ..etc.

2) authenticated

It represents that Login(username/password) is Required & no authorization(role based access) is required.
Examle URLs: /updateUserDetails, /inbox, /settings, ..etc.

3) hasAuthority

hasAuthority represents that user should have both authentication and role based authorization access.
Example URLs+role: /approveRequest + MANAGER, /blockUser + ADMIN, /addUser + ADMIN ...etc.

4) hasAnyAuthority

It represents that user should have authentication and multiple role based authorization access. It's like Manager & HR both roles have access to a particular page.
Example URLs+role: /approveRequest + (MANAGER & HR)

What are the various ways to implement security in a Spring Boot Web Application?

Briefly, there are the three most common ways to implement security in a Spring Boot web application. They are

In-Memory Security

In this type of implementation, we store data in RAM (in-memory) and validate data when there is a request. We use this method in test or development environment. This method is not recommended in production applications.

Using Database(JDBC)

In this type of implementation, we store data in the database and validate data/login when a request comes. But it works based on SQL queries provided by the programmer.

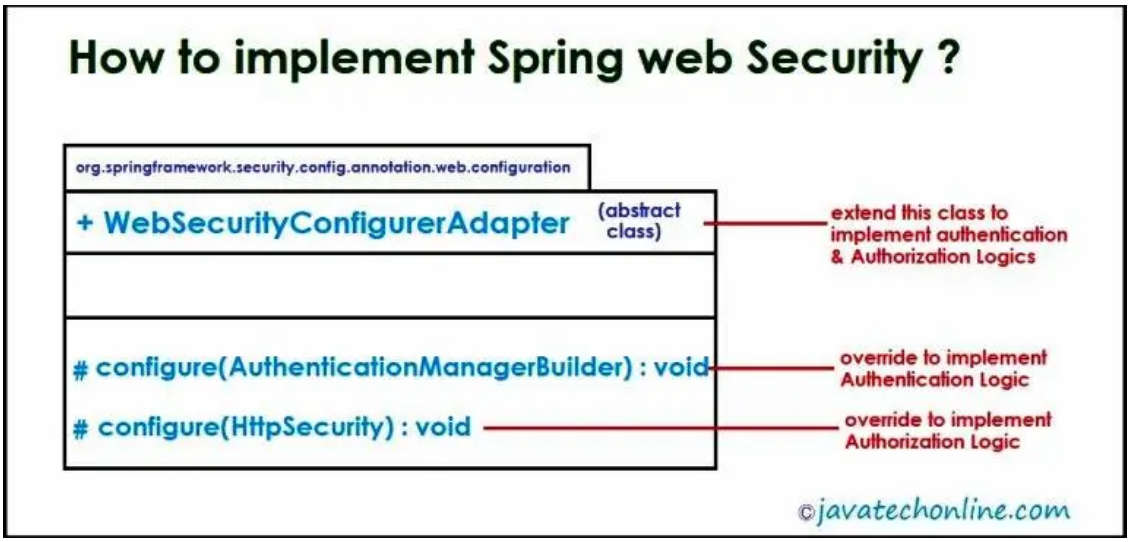
TOP

Using UserDetailsService

We store data in the database and validate data when a request comes. But UserDetailsService works based on ORM(Spring Data JPA). In short, UserDetailsService is an interface provided by the Spring Security module. After entering the username in Login form, when we click on login button this service is called. Subsequently, It locates the user based on the provided username. It contains a method loadUserByUsername(String username) which returns UserDetails object. Furthermore, the UserDetails object provides us the username.

What are the steps to implement web security in Spring Boot Application?

Below are the steps:



1) Write a class as 'SecurityConfig.java' that extends a predefined abstract class WebSecurityConfigurerAdapter.java

2) Apply annotations @Configuration and @EnableWebSecurity on top of 'SecurityConfig.java' accordingly. Needless to say, annotation @EnableWebSecurity is for enabling the security feature in web application.

3) Override below two methods to implement authentication & authorization logics simultaneously.
configure(AuthenticationManagerBuilder auth)
configure(HttpSecurity http)

4) Furthermore, Inject other objects dependency(if any), by using @Autowired like DataSource, BCryptPasswordEncoder etc. and use where you need them.

5) Equally important, Write controller classes and view pages consequently as needed.

TOP

6) In the end, update database properties in application.properties file if you are not implementing in-memory authentication.

Note: Spring Security 5.7.0-M2 onward, WebSecurityConfigurerAdapter has been deprecated. In order to learn the new way of implementing custom configuration class, visit a separate article on Spring Security without WebSecurityConfigurerAdapter.

Example of How to implement in-memory authentication security

For example, Let's consider an application of a small organization where we have three roles : EMPLOYEE, MANAGER and ADMIN. In addition, we will have role based restricted access to some particular pages. On the one hand, some pages will be accessible to multiple roles and on the other hand some with no restrictions(accessible to all roles). Moreover, we will use thymeleaf to create pages. Additionally, we will have a controller class to serve the user's request.

Software/Technologies Used

♦STS (Spring Tool Suite) : Version-> 4.7.1.RELEASE
♦JDK14 (JDK8 or later versions are sufficient)

Step#1 : Create a Spring Boot Starter Project in STS(Spring Tool Suite)

While creating Starter Project, select 'Spring Security', 'Thymeleaf', 'Spring Web' and 'Spring Boot DevTools' as starter project dependencies. In order to know 'how to create Spring Boot Starter Project?', kindly visit Internal Link.

Step#2 : Write a Controller class

Please check below code as a Controller class.

HomeController.java

```
1  package com.dev.springboot.security.controller;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.GetMapping;
5
6  @Controller
7  public class HomeController {
```

TOP

```
8
9       @GetMapping("/home")
10      public String getHomePage() {
11          return "homePage";
12      }
13
14      @GetMapping("/welcome")
15      public String getWelcomePage() {
16          return "welcomePage";
17      }
18
19      @GetMapping("/admin")
20      public String getAdminPage() {
21          return "adminPage";
22      }
23
24      @GetMapping("/emp")
25      public String getEmployeePage() {
26          return "empPage";
27      }
28
29      @GetMapping("/mgr")
30      public String getManagerPage() {
31          return "mgrPage";
32      }
33
34      @GetMapping("/common")
35      public String getCommonPage() {
36          return "commonPage";
37      }
38
39      @GetMapping("/accessDenied")
40      public String getAccessDeniedPage() {
41          return "accessDeniedPage";
42      }
43  }
```

Step#3 : Write UI pages(Thymeleaf)

Below files are UI pages. So, place them inside 'src/main/resources/templates' folder only.

TOP

homepage.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3> welcome to Home Page </h3>
This page is accessible to ALL.
</body>
</html>
```

welcomePage.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3> Welcome Page after successful Login</h3>
<a th:href="@{/logout}" >LOGOUT</a>
</body>
</html>
```

adminPage.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3> Admin Page </h3>
```

TOP

```
Only Admins are allowed to access this page.!!!
<a th:href="@{/logout}" >LOGOUT</a>
</body>
</html>
```

empPage.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3> Employee Page </h3>
<a th:href="@{/logout}" >LOGOUT</a>
</body>
</html>
```

mgrPage.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3> Manager Page </h3>
<a th:href="@{/logout}" >LOGOUT</a>
</body>
</html>
```

commonPage.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
```

TOP

```
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3> COMMON Page </h3>
Both Manager & Employee are allowed to access this page. !
<a th:href="@{/logout}" >LOGOUT</a>
</body>
</html>
```

accessDeniedPage.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h3>You are not allowed to access this page. Please go to Welcome Page</h3>
<a th:href="@{/welcome}" >Welcome</a>
<a th:href="@{/logout}" >LOGOUT</a>
</body>
</html>
```

Step#4 : Write SecurityConfig class

This class is the most important with respect to implementing security features that contains all security related logics. As shown in the code example below, we use '{noop}' before password value if we don't want to encode the password.

Step#4A: Code For versions lower than Spring Security 5.7.0

SecurityConfig.java

```
package com.dev.springboot.security;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.bu
```

TOP

```
5  import org.springframework.security.config.annotation.web.builders.HttpSecurity
6  import org.springframework.security.config.annotation.web.configuration.EnableW
7  import org.springframework.security.config.annotation.web.configuration.WebSecu
8  import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
9
10 @Configuration
11 @EnableWebSecurity
12 public class SecurityConfig extends WebSecurityConfigurerAdapter {
13
14     @Override
15     protected void configure(AuthenticationManagerBuilder auth) throws Exceptio
16
17         // {noop} => No operation for password encoder  (no password encoding m
18         auth.inMemoryAuthentication().withUser("devs").password("{noop}devs").a
19         auth.inMemoryAuthentication().withUser("ns").password("{noop}ns").autho
20         auth.inMemoryAuthentication().withUser("vs").password("{noop}vs").autho
21     }
22
23     @Override
24     protected void configure(HttpSecurity http) throws Exception {
25
26         //declares which Page(URL) will have What access type
27         http.authorizeRequests()
28         .antMatchers("/home").permitAll()
29         .antMatchers("/welcome").authenticated()
30         .antMatchers("/admin").hasAuthority("ADMIN")
31         .antMatchers("/emp").hasAuthority("EMPLOYEE")
32         .antMatchers("/mgr").hasAuthority("MANAGER")
33         .antMatchers("/common").hasAnyAuthority("EMPLOYEE","MANAGER")
34
35         // Any other URLs which are not configured in above antMatchers
36         // generally declared aunthenticated() in real time
37         .anyRequest().authenticated()
38
39         //Login Form Details
40         .and()
41         .formLogin()
42         .defaultSuccessUrl("/welcome", true)
43
44         //Logout Form Details
45         .and()
46         .logout()
```

↑ TOP

```
47          .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
48
49          //Exception Details
50          .and()
51          .exceptionHandling()
52          .accessDeniedPage("/accessDenied")
53          ;
54      }
55  }
```

Step#4B: Code For versions higher than Spring Security 5.7 and lower than Spring Security 6.0

As WebSecurityConfigurerAdapter has been deprecated from Spring Security 5.7.0-M2 as per an announcement posted in the Spring Official website, on 21st Feb, 2022, we will write SecurityConfig class without using WebSecurityConfigurerAdapter as shown below:

SecurityConfig.java

```
1   package com.dev.springboot.security;
2
3   import java.util.ArrayList;
4   import java.util.List;
5
6   import org.springframework.context.annotation.Bean;
7   import org.springframework.context.annotation.Configuration;
8   import org.springframework.security.config.annotation.authentication.builders.A
9   import org.springframework.security.config.annotation.web.builders.HttpSecurity
10  import org.springframework.security.config.annotation.web.configuration.EnableW
11  import org.springframework.security.core.GrantedAuthority;
12  import org.springframework.security.core.authority.SimpleGrantedAuthority;
13  import org.springframework.security.core.userdetails.User;
14  import org.springframework.security.core.userdetails.UserDetails;
15  import org.springframework.security.provisioning.InMemoryUserDetailsManager;
16  import org.springframework.security.web.SecurityFilterChain;
17  import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
18
19  @Configuration
20  @EnableWebSecurity
21  public class SecurityConfig {
22
```

TOP

```
23      @Bean
24      protected InMemoryUserDetailsManager configAuthentication() {
25
26          List<UserDetails> users = new ArrayList<>();
27          List<GrantedAuthority> adminAuthority = new ArrayList<>();
28          adminAuthority.add(new SimpleGrantedAuthority("ADMIN"));
29          UserDetails admin= new User("devs", "{noop}devs", adminAuthority);
30          users.add(admin);
31
32          List<GrantedAuthority> employeeAuthority = new ArrayList<>();
33          adminAuthority.add(new SimpleGrantedAuthority("EMPLOYEE"));
34          UserDetails employee= new User("ns", "{noop}ns", employeeAuthority);
35          users.add(employee);
36
37          List<GrantedAuthority> managerAuthority = new ArrayList<>();
38          adminAuthority.add(new SimpleGrantedAuthority("MANAGER"));
39          UserDetails manager= new User("vs", "{noop}vs", managerAuthority);
40          users.add(manager);
41
42          return new InMemoryUserDetailsManager(users);
43      }
44
45      @Bean
46      protected SecurityFilterChain filterChain(HttpSecurity http) throws Except
47
48          //declares which Page(URL) will have What access type
49          http.authorizeRequests()
50          .antMatchers("/home").permitAll()
51          .antMatchers("/welcome").authenticated()
52          .antMatchers("/admin").hasAuthority("ADMIN")
53          .antMatchers("/emp").hasAuthority("EMPLOYEE")
54          .antMatchers("/mgr").hasAuthority("MANAGER")
55          .antMatchers("/common").hasAnyAuthority("EMPLOYEE","MANAGER")
56
57          // Any other URLs which are not configured in above antMatchers
58          // generally declared aunthenticated() in real time
59          .anyRequest().authenticated()
60
61          //Login Form Details
62          .and()
63          .formLogin()
64          .defaultSuccessUrl("/welcome", true)
```

↑ TOP

```
65
66          //Logout Form Details
67          .and()
68          .logout()
69          .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
70
71          //Exception Details
72          .and()
73          .exceptionHandling()
74          .accessDeniedPage("/accessDenied")
75          ;
76
77          return http.build();
78      }
79  }
```

Step#4C: Code For versions Spring Security 6.0.0 and higher (Spring Boot 3.0+)

From Spring Security 6.0 (released in November, 2022), the WebSecurityConfigurerAdapter is completely removed from the Spring Security API. It has also impacted the newly released Spring Boot 3.0 in November, 2022. Hence, if you are using Spring Framework 6.0+ or Spring Boot 3.0+, in either case, your implementation of SecurityConfig.java should look like below. Additionally, you can also check the Spring Security related changes in Spring Framework 6.0.

SecurityConfig.java

```
1  package com.dev.springboot.security;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import org.springframework.context.annotation.Bean;
7  import org.springframework.context.annotation.Configuration;
8  import org.springframework.security.config.annotation.authentication.builders.A
9  import org.springframework.security.config.annotation.web.builders.HttpSecurity
10 import org.springframework.security.config.annotation.web.configuration.EnableW
11 import org.springframework.security.core.GrantedAuthority;
12 import org.springframework.security.core.authority.SimpleGrantedAuthority;
13 import org.springframework.security.core.userdetails.User;
14 import org.springframework.security.core.userdetails.UserDetails;
15 import org.springframework.security.provisioning.InMemoryUserDetailsMan
```

TOP

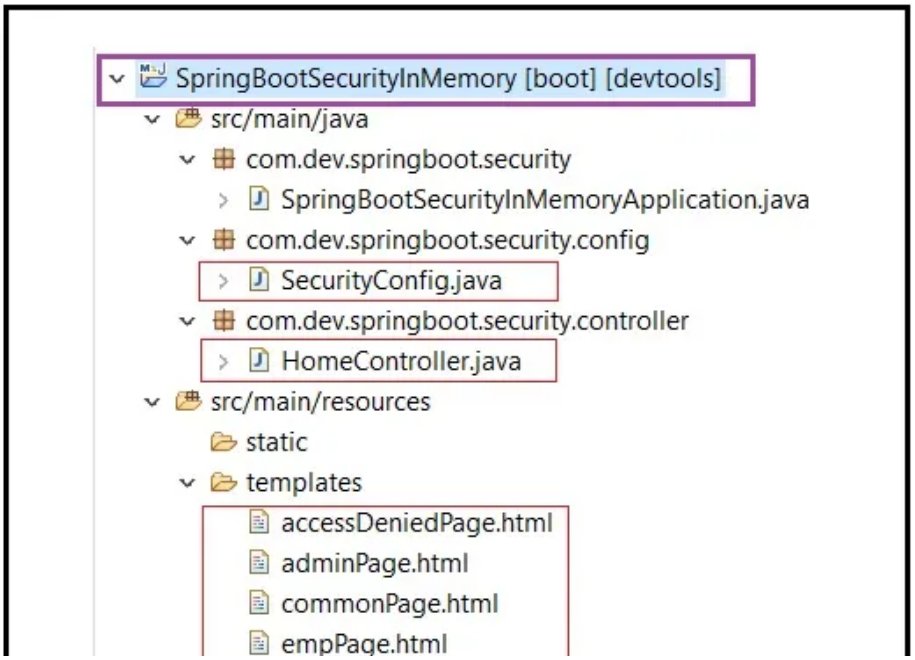
```
16  import org.springframework.security.web.SecurityFilterChain;
17  import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
18
19  @Configuration
20  @EnableWebSecurity
21  public class SecurityConfig {
22
23      @Bean
24      protected InMemoryUserDetailsManager configAuthentication() {
25
26          List<UserDetails> users = new ArrayList<>();
27          List<GrantedAuthority> adminAuthority = new ArrayList<>();
28          adminAuthority.add(new SimpleGrantedAuthority("ADMIN"));
29          UserDetails admin= new User("devs", "{noop}devs", adminAuthority);
30          users.add(admin);
31
32          List<GrantedAuthority> employeeAuthority = new ArrayList<>();
33          adminAuthority.add(new SimpleGrantedAuthority("EMPLOYEE"));
34          UserDetails employee= new User("ns", "{noop}ns", employeeAuthority);
35          users.add(employee);
36
37          List<GrantedAuthority> managerAuthority = new ArrayList<>();
38          adminAuthority.add(new SimpleGrantedAuthority("MANAGER"));
39          UserDetails manager= new User("vs", "{noop}vs", managerAuthority);
40          users.add(manager);
41
42          return new InMemoryUserDetailsManager(users);
43      }
44
45      @Bean
46      protected SecurityFilterChain filterChain(HttpSecurity http) throws Except
47
48          //declares which Page(URL) will have What access type
49          http.authorizeHttpRequests()
50          .requestMatchers("/home").permitAll()
51          .requestMatchers("/welcome").authenticated()
52          .requestMatchers("/admin").hasAuthority("ADMIN")
53          .requestMatchers("/emp").hasAuthority("EMPLOYEE")
54          .requestMatchers("/mgr").hasAuthority("MANAGER")
55          .requestMatchers("/common").hasAnyAuthority("EMPLOYEE","MANAGER")
56
57          // Any other URLs which are not configured in above antMatchers
```

TOP

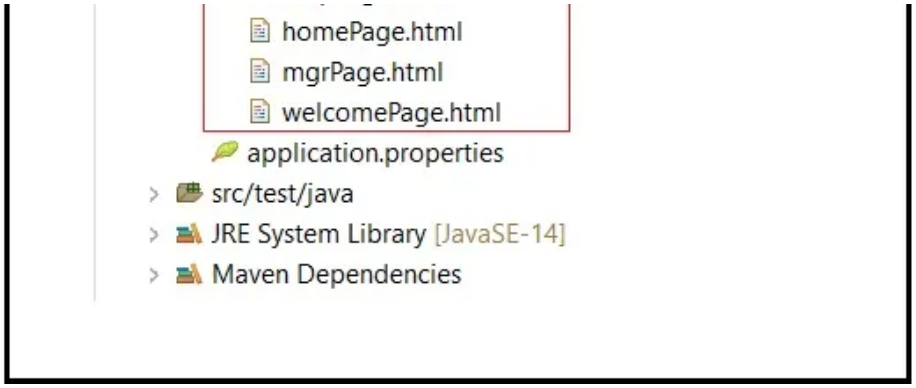
```
58          // generally declared aunthenticated() in real time
59          .anyRequest().authenticated()
60
61          //Login Form Details
62          .and()
63          .formLogin()
64          .defaultSuccessUrl("/welcome", true)
65
66          //Logout Form Details
67          .and()
68          .logout()
69          .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
70
71          //Exception Details
72          .and()
73          .exceptionHandling()
74          .accessDeniedPage("/accessDenied")
75          ;
76
77          return http.build();
78      }
79  }
```

Note: In the code above, observe the API changes in Spring Security 6.0: antMatchers() replaced by requestMatchers(), authorizeRequests() replaced by authorizeHttpRequests().

Your project structure will look like below snapshot.



TOP



Example of How to implement JDBC authentication security

In this method of authentication we validate credentials and roles from existing values in database. Also, we will work on password encryption logic as well in this example. First of all we will insert some dummy data in the database, then test the security feature accordingly. In the final analysis, If we compare the functionality of the previous example from this is that we will save user credentials & roles into the database rather than RAM. Also, we will have additional code for implementing password encryption logic in this example. So all files from the previous example will also be valid in this example.

Additionally, we will have one config class to create object of BCryptPasswordEncoder to encode the password and also entries of database properties in application.properties to connect with the database. Needless to say, the implementation of SecurityConfig class will differ this time.

Software/Technologies Used

♦ STS (Spring Tool Suite) : Version-> 4.7.1.RELEASE
♦ MySQL Database : Version ->8.0.19 MySQL Community Server
♦ JDK14 (JDK8 or later versions are sufficient)

Step#1 : Insert some dummy records in database

Please note that this step is required only for testing purpose. If you already have a database created with user credentials and roles mapping recently, ignore this step. In Step#1A we will create some encoded values of password, then we insert them into database in Step#1B.

Step#1A : Create encoded password values by using BCryptPasswordEncoder

We will use BCryptPasswordEncoder to generate some encoded password values as below

TOP

BCryptPasswordEncoderTest.java

```
1  package com.dev.springboot.security.util;
2
3  import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
4
5  public class BCryptPasswordEncoderTest {
6
7      public static void main(String[] args) {
8          BCryptPasswordEncoder bpe = new BCryptPasswordEncoder();
9          String encodedPWD = bpe.encode("devs@A!5003");
10         System.out.println(encodedPWD);
11     }
12 }
```

Output

$2a$10$qnOB2PH1CqRvw8f5epvHzOlrounRkVGi.Y5ho6ENdmj/C1DmPdAsy

Step#1B : create a database and insert dummy records

Execute MySQL DB Commands as shown below:

To create the database as 'testbootsecurity'

♦ `create database testbootsecurity;`

To take 'testbootsecurity' in use:

♦ `use testbootsecurity;`

To create the user table:

♦ `create table user (user_id int, user_name varchar(30), user_pwd varchar(100), user_r`

To check the description of user table:

♦ `desc user;`

TOP

```
INSERT INTO user values(501,'devs','$2a$10$qnOB2PH1CqRvw8f5epvHzOlrounRkVGi.Y5ho6ENdmj
INSERT INTO user values(502,'ns','$2a$10$mmUMC5ZwoVnEQYV7/R6m.uWWtj7EiIo3lKasBObkOCc12
INSERT INTO user values(503,'vs','$2a$10$kqEC/fhQ7SNDhnncOQ9pb.yXXxJ/c7a1SQx2QPNZ.47fU
```

♦ `select * from user;` **(To check if values are inserted into DB)**

In addition, we will require below two queries in implementing configure(AuthenticationManagerBuilder auth) method of SecurityConfig.java class.

Query#1 : Retrieve username,password,enabled using username ;

♦ `select user_name,user_pwd,user_enabled from user where user_name=?;`

Query#2 : Retrieve username,role using username;

♦ `select user_name,user_role from user where user_name=?;`

Step#2 : Create a Spring Boot Starter Project in STS(Spring Tool Suite)

While creating Starter Project select 'Spring Security', 'Thymeleaf', 'Spring Web', 'JDBC API', 'MySQL Driver' and 'Spring Boot DevTools' as starter project dependencies.

Step#3 : Copy Controller class & UI pages from previous Example

Copy HomeController.java and also all thymeleaf pages from previous example.

Step#4 : Create AppConfig.java to create object of BCryptPasswordEncoder

Create AppConfig.java as below. We will require object of BCryptPasswordEncoder while implementing SecurityConfig.java class.

AppConfig.java

```
1 package com.dev.springboot.security.jdbc.config;
2
```

TOP

```
3  import org.springframework.context.annotation.Bean;
4  import org.springframework.context.annotation.Configuration;
5  import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
6
7  @Configuration
8  public class AppConfig {
9
10     @Bean
11     public BCryptPasswordEncoder encode() {
12         return new BCryptPasswordEncoder();
13     }
14 }
```

Step#5 : Update database properties in application.properties file

Update application.properties to connect to MySQL DB as below. Please note that we can omit driver-class-name as Spring Boot will automatically find it from database URL.

```
#application.properties
-----------------------------------------------------------------
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/testBootSecurity
spring.datasource.username=root
spring.datasource.password=devs
```

Step#6 : Update SecurityConfig.java

Copy SecurityConfig.java from previous example and update configure(AuthenticationManagerBuilder auth) method as below.

Step#6A: Code For versions lower than Spring Security 5.7.0

SecurityConfig.java

```
1  package com.dev.springboot.security.jdbc.config;
2
3  import javax.sql.DataSource;
4
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.context.annotation.Configuration;
```

TOP

```
7  import org.springframework.security.config.annotation.authentication.builders.A
8  import org.springframework.security.config.annotation.web.builders.HttpSecurity
9  import org.springframework.security.config.annotation.web.configuration.EnableW
10 import org.springframework.security.config.annotation.web.configuration.WebSecu
11 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
12 import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
13
14 @Configuration
15 @EnableWebSecurity
16 public class SecurityConfig extends WebSecurityConfigurerAdapter {
17
18     @Autowired
19     private DataSource dataSource;
20
21     @Autowired
22     private BCryptPasswordEncoder passwordEncoder;
23
24     @Override
25     protected void configure(AuthenticationManagerBuilder auth) throws Exceptio
26
27         auth.jdbcAuthentication()
28         .dataSource(dataSource)     //creates database connection
29         .usersByUsernameQuery("select user_name,user_pwd,user_enabled from user
30         .authoritiesByUsernameQuery("select user_name,user_role from user where
31         .passwordEncoder(passwordEncoder);
32     }
33
34     @Override
35     public void configure(HttpSecurity http) throws Exception {
36
37         http.authorizeRequests()
38         .antMatchers("/home").permitAll()
39         .antMatchers("/welcome").authenticated()
40         .antMatchers("/admin").hasAuthority("ADMIN")
41         .antMatchers("/emp").hasAuthority("EMPLOYEE")
42         .antMatchers("/mgr").hasAuthority("MANAGER")
43         .anyRequest().authenticated()
44
45         .and()
46         .formLogin()
47         .defaultSuccessUrl("/welcome",true)
48
```

TOP

```
49          .and()
50          .logout()
51          .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
52
53          .and()
54          .exceptionHandling()
55          .accessDeniedPage("/accessDenied")
56          ;
57
58      }
59  }
```

Step#6B: Code For versions higher than Spring Security 5.7.0 and lower than Spring Security 6.0.0

As WebSecurityConfigurerAdapter has been deprecated from Spring Security 5.7.0-M2 as per an announcement posted in the Spring Official website, on 21st Feb, 2022, we will write SecurityConfig class without using WebSecurityConfigurerAdapter as shown below:

SecurityConfig.java

```
1  package com.dev.springboot.security.jdbc.config;
2
3  import javax.sql.DataSource;
4
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.context.annotation.Bean;
7  import org.springframework.context.annotation.Configuration;
8  import org.springframework.security.config.annotation.authentication.builders.A
9  import org.springframework.security.config.annotation.web.builders.HttpSecurity
10 import org.springframework.security.config.annotation.web.configuration.EnableW
11 import org.springframework.security.core.userdetails.User;
12 import org.springframework.security.core.userdetails.UserDetails;
13 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
14 import org.springframework.security.crypto.factory.PasswordEncoderFactories;
15 import org.springframework.security.provisioning.JdbcUserDetailsManager;
16 import org.springframework.security.provisioning.UserDetailsManager;
17 import org.springframework.security.web.SecurityFilterChain;
18 import org.springframework.security.web.util.matcher.AntPathRequestMatc
```

TOP

```
19
20  @Configuration
21  @EnableWebSecurity
22  public class SecurityConfig {
23
24      @Autowired
25      private DataSource dataSource;
26
27      @Autowired
28      private BCryptPasswordEncoder passwordEncoder;
29
30      @Bean
31      public UserDetailsManager authenticateUsers() {
32
33          UserDetails user = User.withUsername("devs").
34                  password(PasswordEncoderFactories.createDelegatingPasswordEncoc
35          JdbcUserDetailsManager users = new JdbcUserDetailsManager(dataSource);
36          users.setAuthoritiesByUsernameQuery("select user_name,user_pwd,user_ena
37          users.setUsersByUsernameQuery("select user_name,user_role from user whe
38          users.createUser(user);
39          return users;
40      }
41
42      @Bean
43      protected SecurityFilterChain filterChain(HttpSecurity http) throws Except
44
45          http.authorizeRequests()
46          .antMatchers("/home").permitAll()
47          .antMatchers("/welcome").authenticated()
48          .antMatchers("/admin").hasAuthority("ADMIN")
49          .antMatchers("/emp").hasAuthority("EMPLOYEE")
50          .antMatchers("/mgr").hasAuthority("MANAGER")
51          .anyRequest().authenticated()
52
53          .and()
54          .formLogin()
55          .defaultSuccessUrl("/welcome",true)
56
57          .and()
58          .logout()
59          .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
60
```

TOP

```
            .and()
            .exceptionHandling()
            .accessDeniedPage("/accessDenied")
            ;
            return http.build();
        }
}
```

Step#6C: Code For versions Spring Security 6.0.0 and higher (Spring Boot 3.0+)

From Spring Security 6.0 (released in November, 2022), the WebSecurityConfigurerAdapter is completely removed from the Spring Security API. It has also impacted the newly released Spring Boot 3.0 in November, 2022. Hence, if you are using Spring Framework 6.0+ or Spring Boot 3.0+, in either case, your implementation of SecurityConfig.java should look like below. Additionally, you can also check the Spring Security related changes in Spring Framework 6.0.

SecurityConfig.java

```
package com.dev.springboot.security.jdbc.config;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.A
import org.springframework.security.config.annotation.web.builders.HttpSecurity
import org.springframework.security.config.annotation.web.configuration.EnableW
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.factory.PasswordEncoderFactories;
import org.springframework.security.provisioning.JdbcUserDetailsManager;
import org.springframework.security.provisioning.UserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

```

TOP

```
24      @Autowired
25      private DataSource dataSource;
26
27      @Autowired
28      private BCryptPasswordEncoder passwordEncoder;
29
30      @Bean
31      public UserDetailsManager authenticateUsers() {
32
33          UserDetails user = User.withUsername("devs").
34                  password(PasswordEncoderFactories.createDelegatingPasswordEncoc
35          JdbcUserDetailsManager users = new JdbcUserDetailsManager(dataSource);
36          users.setAuthoritiesByUsernameQuery("select user_name,user_pwd,user_ena
37          users.setUsersByUsernameQuery("select user_name,user_role from user whe
38          users.createUser(user);
39          return users;
40      }
41
42      @Bean
43      protected SecurityFilterChain filterChain(HttpSecurity http) throws Except
44
45          http.authorizeHttpRequests()
46          .requestMatchers("/home").permitAll()
47          .requestMatchers("/welcome").authenticated()
48          .requestMatchers("/admin").hasAuthority("ADMIN")
49          .requestMatchers("/emp").hasAuthority("EMPLOYEE")
50          .requestMatchers("/mgr").hasAuthority("MANAGER")
51          .anyRequest().authenticated()
52
53          .and()
54          .formLogin()
55          .defaultSuccessUrl("/welcome",true)
56
57          .and()
58          .logout()
59          .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
60
61          .and()
62          .exceptionHandling()
63          .accessDeniedPage("/accessDenied")
64          ;
65          return http.build();
```

TOP

```
66          }
67  }
```

Note: In the code above, observe the API changes in Spring Security 6.0: antMatchers() replaced by requestMatchers(), authorizeRequests() replaced by authorizeHttpRequests().

How to test the security enabled Application ?

After a lot of theory and coding steps simultaneously, its time to test the 'How to implement Security in Spring Boot Project?'. Please follow below steps.

1) Start the application, subsequently Right click on the project, then select "Run As' >> 'Spring Boot App'.

2) Enter the home page URL http://localhost:8080/home, then check if it is accessible to everyone without even login to the application.

3) Enter the admin page URL http://localhost:8080/admin, then it should be redirected to inbuilt login page (provided by Spring Security)

4) Login with providing admin credentials, you will be redirected to welcome page.

5) Enter the URL http://localhost:8080/admin, then you will be able to see the admin page now.

6) As you are logged in with admin credentials, you can see all the pages by hitting the other pages URLs also.

Repeat the above steps subsequently for other roles as well & check whether user can access the page as per the granted roles.

How to implement Security in Spring Boot Project? : Spring Boot 3.0 Onward

Spring Boot 3.0 comes with major API level changes in the Spring Security module. Hence, if you are using Spring Boot 3.0 or later versions, you need to follow below guidelines along with using JDK 17.

1) **In SecurityConfig.java**:

Use authorizeHttpRequests() in place of authorizeRequests()

Use requestMatchers() in place of antMatchers()

TOP

Use RegexRequestMatchers() in place of regexMatchers()

2) **In Entity class**:

In import statements, use 'jakarta' in place of 'javax'. For example: Use 'jakarta.persistence.Entity;' in place of 'javax.persistence.Entity;'.

FAQ

How to disable Security feature of the Application?

Equally important, sometimes we need to disable the security feature, particularly in development & test environment to avoid entering credentials again & again. For that we can comment below two dependencies in your pom.xml file accordingly.

```
<!--
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
-->
```

Summary

Subsequently, going through all the theoretical & examples part of 'How to implement Security in Spring Boot Project?', finally, we are able to implement web security in a Spring Boot project. Of course, in this article we have covered two ways of implementing security feature. Furthermore, we can go through UserDetailsService(the third way of implementing security) in other article. Also, for complete tutorials on Spring Boot Security visit here. If there is any update in the future, we will update the same accordingly. Also Feel free to provide your comments in below comments section.

Spring Security Without WebSecurityConfigurerAdapter
July 23, 2022
In "java"

Spring Security Annotations With Examples
August 4, 2022
In "java"

TOP