

# Contents

- Introduction to Spring Security
- Spring Security Terminologies
- Basic Authentication with System Generated Credentials
- Authorisation with InMemory Authentication
- Authorisation with Database user Authentication



### Introduction to Spring Security :

Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications. Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements. Some of the key features of Spring Security are:

1. Comprehensive and extensible support for both Authentication and Authorization.  
Authentication(401) : Check if the entity who is requesting for a resource is valid or not . Authorization(403) : Check if the valid entity has the required permissions or not . Password is for authentication .Authority is for authorisation.
2. Protection against attacks like session fixation, clickjacking, cross-site request forgery, etc
3. Servlet API integration
4. Optional integration with Spring Web MVC

### Spring Security Terminologies :

Spring Security secures our web application by default and further, we can customize it as per our needs. So important terminologies in Spring Security are as follows:

1. Authentication
2. Authorization
3. Filter

## Session 14



Let us discuss each of them individually as follows:

### Terminology 1: Authentication

The identity of users is verified for providing access to the system. If the user is verified as per the saved credentials, the request is accepted and the desired response is given to the client from the server. Some of the methods are as follows:

- Login form
- HTTP authentication
- Customer authentication method

**1.1 Login Form:** It is a web page to a website that requires user identification and authentication, performed by entering a username and password.

**1.2 HTTP authentication –** In this, the server can request authentication information (user ID and password) from a client.

**1.3 Customer Authentication Method –** customer authentication is a new regulation designed to prevent online transaction fraud.



### Terminology 2: Authorization

Giving the user the permission to access a specific resource or function. Some of the methods: –

**2.1 Access Control for URLs –** Security of URLs allows you to restrict access to specific Internet sites based on the contents of the URL(keywords).

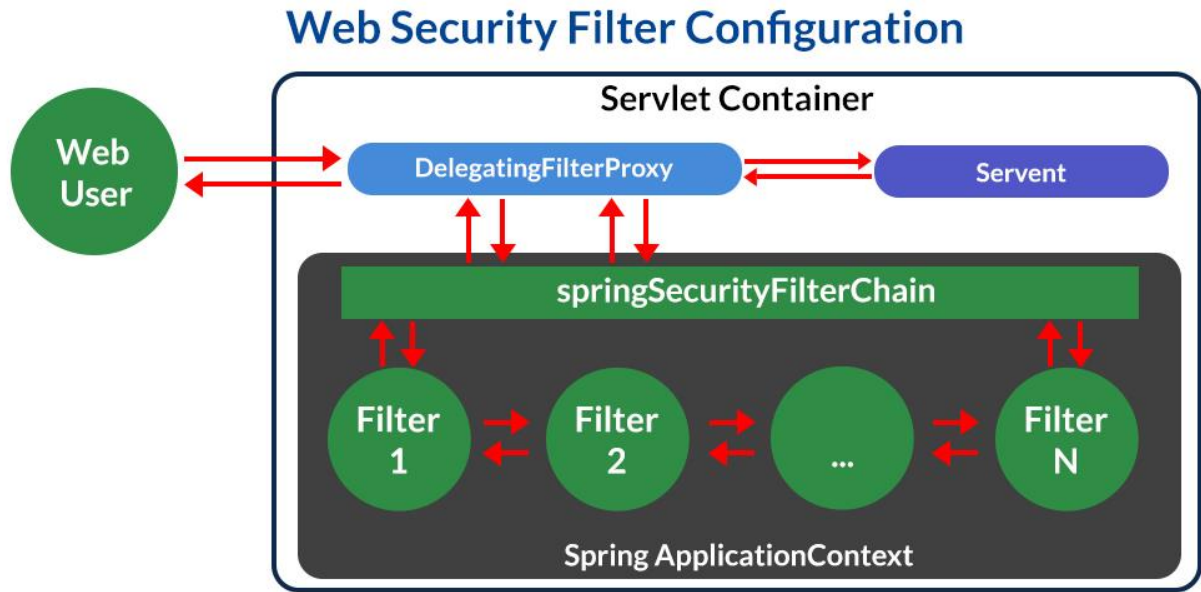
**2.2 Secure Objects and Methods** – The Class method is called by a security interceptor implementation to ensure that the configured AccessDecisionManager supports the type of secure object or not.

**2.3 Access Control Lists** – An ACLs specifies which users are granted access to objects, as well as what operations are allowed to them.



### Terminology 3: Filter

A filter is a function that is invoked at the time of preprocessing and postprocessing of a request. Spring Security maintains a filter chain where all filters have different responsibilities and filters are added or removed depending on which services are required.



## Basic Authentication with System Generated Credentials

Now Let's start coding . Add the following dependencies in the pom of your new maven project :

- spring web
- spring security

**TASK : Now lets create a controller and a simple greeter GET api which returns Hello .Will I be able to call this API without logging in ?**

HINT : No, because with Spring Security dependency all the apis are secured now .

Challenge 1 : What is the status it returned to our API ?

HINT : 302 . It means that it has been redirected to some other API which is present in the response headers LOCATION. This means that the API we hit wanted the user to be signed in .



## What is a cookie?

Cookie in simpler terms means just the textual information about some website. When you visit a particular website, some information is saved in your local system so that when you visit the same website again, this website is able to recognize you and show you the results according to your preferences. Cookies have been long used in internet history and have developed in a magnificent way. When you visit a website you actually request the web page from the server. For a server, every request is a unique request. So if you visit a hundred times, the server will consider each and every request unique. Since the intensity of requests that arrive at a server is high, it is obvious and logical not to store every user's information on the server. Maybe you never visit again and the same information will be redundant. So, to uniquely remember you, the server sends the cookies along with the response which is saved in your local machine. Now the next time you hit the same server, you will get a response according to you as the server will recognize you. This cookie is unique to every server (some exceptions exist today because of advertisements). So you might have many cookies in your system but a server will recognize its own cookie and can analyze it for you. How this evolved over time and used today is discussed in the next section.

## Why use cookies?

Cookies developed in the initial days because developers needed some information about the client to make their experience better. Let say you visit a website which is not in your local language (let's say English). You choose the English option in the language section of the website. Now if you visit the same website 5 times a day, you might have to change the language 5 times. Therefore, this information is saved as a cookie in your system. So the next time you send the request, the server will know that you want to see the website in english. This is where cookies play a vital role. But this is a very minute example of the scale cookies are used today.

**TASK : Lets analyze the Developer Tools Network tab . Are we sending any cookies ?**

**HINT :** We can see that for the first time , there is no cookie that is sent in the Request Headers .Cookies can help to check the authenticity of that user .Notice that we are getting a cookie from the server in response header. This will be sent next time in the Request Headers when we again hit subsequent apis in this authenticated in session.

**TASK : Lets analyze the server logs before and after login .**

**TASK :( Before we login )In response headers , can you see the backend is returning a cookie (SET-COOKIE) ?**

**HINT :** JSESSIONID is a cookie generated by Servlet containers like Tomcat or Jetty and used for session management in the J2EE web application for HTTP protocol. Since HTTP is a stateless protocol there is no way for Web Server to relate two separate requests coming from the same client and Session management is the process to track user sessions using different session management techniques like Cookies and URL Rewriting. If a Web server is using a cookie for session management it creates and sends a JSESSIONID cookie to the client and then the client sends it back to the server in subsequent HTTP requests.  
We can also see this jsessionid cookie in the Application tab in the cookie section . Note : This session id is of an unauthenticated user .

**TASK : Where is the username and password for this login page ?**

**HINT :** username is user and password is the server logs .

**TASK : Will the JSESSIONID change once we login ?**

**TASK : What happens once we login ?**

**HINT :** See it returns 302 again and it redirects to our initial greeter api which we requested for .We can see that this time , server has passed a different jsessionid in the response headers .

Note : This Jsessionid is of an authenticated user .

**TASK : Now lets hit the greeter api once more . Do I have to login again ? Which Jsessionid will be present in the Application tab under the cookie section ?**

**HINT :** No , because frontend will authenticate the user by the same jsessionid it received in the response header . We can see it is passed by frontend in request headers to the backend .

**TASK : What will happen if I go into the Application tab and clear the cookie from the cookie section and then hit the greeter api ?**

**HINT :** We will have to login again .

**TASK : Whenever my app restarts , is a new password generated or is it the same one as the previous one from the logs ?**

Challenge 1: Till what time will this cookie stay alive ?

HINT : 60 minutes .Check the keep-alive response header.

Challenge 2: What is the Security Context in Spring Security ?

HINT : The SecurityContext and SecurityContextHolder are two fundamental classes of Spring Security. The SecurityContext is used to store the details of the currently authenticated user, also known as a principle. So, if you have to get the username or any other user details, you need to get this SecurityContext first. The SecurityContextHolder is a helper class, which provides access to the security context.

Challenge 3: How to get the current logged-in Username in Spring Security ?

HINT :

User user =

(User)SecurityContextHolder.getContext().getAuthentication().getPrincipal();

String username = user.getUsername();

Challenge 4: Can we tell Spring that this API should not be secured ?

HINT : Yes , by creating a configuration class containing the authentication and authorisation function.

References :

<https://www.geeksforgeeks.org/how-to-implement-simple-authentication-in-spring-boot/>



## Authorisation with InMemory Authentication

**TASK :** In the same project , let's create a configuration class . Extend the `WebSecurityConfigurerAdapter` class .

**TASK :** Define Authorities in the `application.properties` . Note : Ideally , Authorities should not be designations , it should be verbs .

**TASK :** Override the 2 configure methods . One will help us for authentication and the other will help us facilitate Authorisation .

`protected void configure(AuthenticationManagerBuilder auth) —> Authentication`

```
@Configuration
public class DemoConfig extends WebSecurityConfigurerAdapter {

    private static final String STUDENT_AUTHORITY = "student";
    private static final String ADMIN_AUTHORITY = "admin";

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication() InMemoryUserDetailsManagerConfigurer<AuthenticationManagerBuilder>
            .withUser( username: "a" ) UserDetailsManagerConfigurer<...>.UserDetailsBuilder
            .password(new BCryptPasswordEncoder().encode( rawPassword: "123" ))
            .authorities(ADMIN_AUTHORITY)
            .and() InMemoryUserDetailsManagerConfigurer<AuthenticationManagerBuilder>
            .withUser( username: "s" ) UserDetailsManagerConfigurer<...>.UserDetailsBuilder
            .password(new BCryptPasswordEncoder().encode( rawPassword: "123" ))
            .authorities(STUDENT_AUTHORITY);
    }
}
```

**TASK :** Let us start the application and try to hit the api with the users we have defined . Does the app start ?

**TASK:** Let's define the logic inside the configuration function for Authorisation .

protected void configure(HttpSecurity http) —> Authorisation

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .csrf().disable()
        .httpBasic()
        .and()
        .authorizeHttpRequests()
        .antMatchers(HttpMethod.POST, ...antPatterns: "/student/attendance/**").hasAuthority(ADMIN_AUTHORITY)
        .antMatchers(HttpMethod.GET, ...antPatterns: "/student/attendance/**").hasAnyAuthority(ADMIN_AUTHORITY,
            STUDENT_AUTHORITY)
        .antMatchers(...antPatterns: "/student/**").hasAuthority(STUDENT_AUTHORITY)
        .antMatchers(...antPatterns: "/admin/**").hasAuthority(ADMIN_AUTHORITY)
        .antMatchers(...antPatterns: "/*").permitAll()
        .and()
        .formLogin();
}
```

**TASK :** Let us start the application and try to hit the api with the users we have defined . Does the app start ?

Challenge 1 : What is PasswordEncoder and why do we need a bean of PasswordEncoder ?

**HINT :**

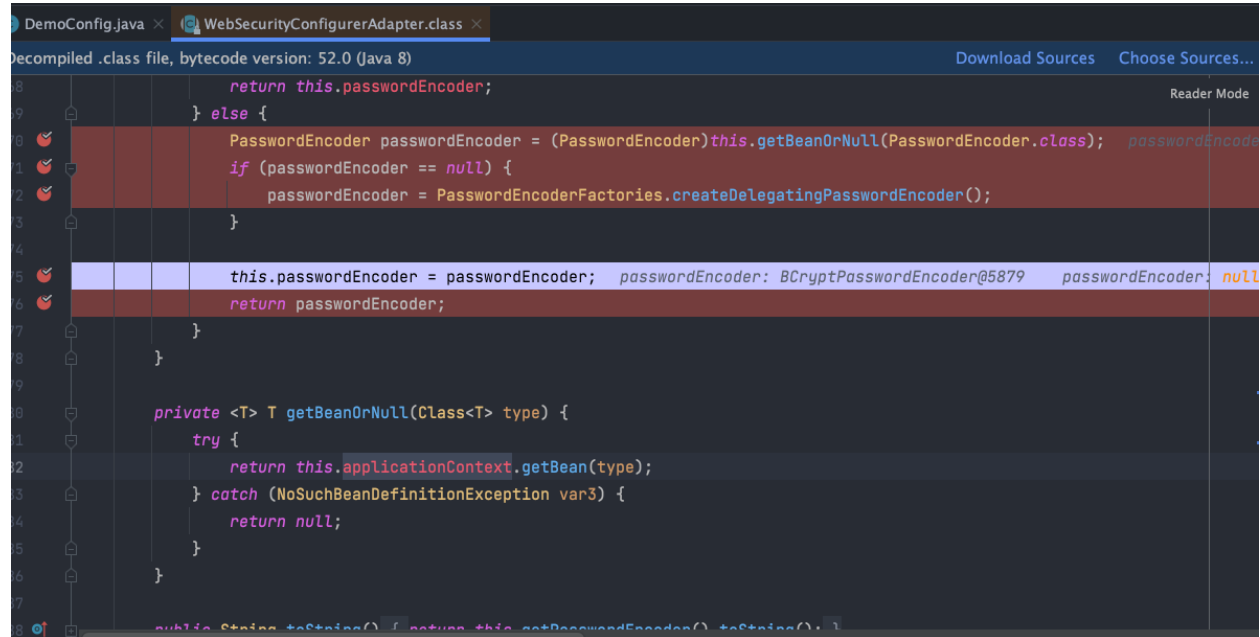
NoOpPasswordEncoder : If, for any reason, we don't want to encode the configured password, we can make use of the NoOpPasswordEncoder.

```
@Bean
public PasswordEncoder getPE(){
    return NoOpPasswordEncoder.getInstance();
}
```

BCryptPasswordEncoder : Implementation of PasswordEncoder that uses the BCrypt strong hashing function.

```
@Bean
public PasswordEncoder getPE(){
    return new BCryptPasswordEncoder();
}
```

Challenge 2: Why do we need to have the getPasswordEncoder() function in the config class as a @Bean ?



```

68         return this.passwordEncoder;
69     } else {
70         PasswordEncoder passwordEncoder = (PasswordEncoder)this.getBeanOrNull(PasswordEncoder.class);
71         if (passwordEncoder == null) {
72             passwordEncoder = PasswordEncoderFactories.createDelegatingPasswordEncoder();
73         }
74
75         this.passwordEncoder = passwordEncoder; // passwordEncoder: BCryptPasswordEncoder@5879
76         return passwordEncoder;
77     }
78 }
79
80 private <T> T getBeanOrNull(Class<T> type) {
81     try {
82         return this.applicationContext.getBean(type);
83     } catch (NoSuchBeanDefinitionException var3) {
84         return null;
85     }
86 }
87
88 public String toString() { return this.getPasswordEncoder().toString(); }

```

Challenge 3: If I have logged in through the Browser client and got authentication cookies in response headers , what will happen when I hit the API through Postman ? Do I need to login again through postman ?

References :

<https://www.geeksforgeeks.org/spring-security-project-example-using-java-configuration/>  
<https://www.geeksforgeeks.org/spring-security-custom-login/>  
<https://www.geeksforgeeks.org/spring-security-in-memory-authentication/>

## **Authorisation with Database user Authentication**

**TASK 0.** Add mysql-connector-java,Validation ,Lombok ,spring data jpa dependencies in pom.xml . Also setup the application.properties file .

**TASK 1.** In the config class inside the configure function, let's use auth.userDetailsService() function for authentication. This requires the instance of a class implementing UserDetailsService interface.

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    // we can define how we want to authenticate
    auth.userDetailsService(myUserService);
}
```

**TASK 2.** Create a custom class (MyUserService) which implements UserDetailsService interface and implements the interface method .

**TASK 3.** Create an entity class MyUser where integer id is Primary Key and username is unique.Entity class is responsible for Authentication and also Authorisation as it specifies Authorities .It has three properties : id,username (UNIQUE),password,authorities(Stored as string with delimiter)

**NOTE :** As MyUserService's loadUserByUsername returns object of class implementing UserDetails interface .Thus , our entity class will implement UserDetails interface and the interface methods .

**TASK 4.** Create a repository interface (MyUserRepository) extending JpaRepository<MyUser,Integer>

```
public interface UserRepository extends JpaRepository<MyUser, Integer> {

    @Query("select u from MyUser u where u.user_name = :s")
    UserDetails find(String s);
}
```

**TASK 5.** Let's write the logic for loadUserByUsername function in MyUserService class and pass the object of MyUserService in auth.userService() in config class.

```
@Service
public class MyUserService implements UserDetailsService {

    @Autowired
    UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // Here you can provide the way to retrieve the data
        return userRepository.find(username);
    }
}
```

**TASK 6.** Now let's create a MyUser object using builder and persist it into the database. We will write the logic in CommandLineRunner.

Result :

```
mysql> select * from my_user;
+-----+-----+-----+-----+
| id | authority | password | username |
+-----+-----+-----+-----+
| 1 | student:admin | $2a$10$P8iJDwpMr7KcX.EBnpNC0.jWSoPEwIM/SVTKjzERJyyc4zMNE9w2C | arjun |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> █
```

Challenge 1: What is the need of CommandLineRunner run function when we can define things in main() ?

Challenge 2: Think if the context will now hold a User object or MyUser object .

Challenge 3: After creating an object class MyUser and also doing database authentication using userServiceDetails , what is the principal returned :

1. If we don't change anything
2. If service function return UserDetails
3. If dao function and service function both return UserDetails

=====

**[V IMP] HOMEWORK TASK :** Create a Post API to sign up a new user into the MyUser table .

HINT : There is a twist here . csrf .

=====

References :

<https://docs.spring.io/spring-security/site/docs/3.0.x/reference/technical-overview.html>

The Official Spring Documentation

## HOMEWORK :

1. What is the difference b/w Role and Authority ? Check your Authorisation and Authentication configure functions in DemoConfig class . (Tinker with it)

### HINT :

In Spring Security, we can **think of each *GrantedAuthority* as an individual privilege**. Examples could include *READ\_AUTHORITY*, *WRITE\_PRIVILEGE*, or even *CAN\_EXECUTE\_AS\_ROOT*. The important thing to understand is that **the name is arbitrary**.

```
// authority
/*
1. A single user can have multiple authorities (CEO /Developer / Manager) -> monitor

      Developer   Manager   CEO   Director   VP
Deploy      Y-----Y       N       N         N
Monitoring  Y         Y       y       y         Y

Designation
.antMatchers("/monitor/**").hasAnyAuthority("Developer", "Manager", "CEO", "Director")
.antMatchers("/deploy/**").hasAnyAuthority("Developer", "Manager")

Roles that a user performs
.antMatchers("/monitor/**").hasAuthority("monitor")
.antMatchers("/deploy/**").hasAuthority("deploy")
*/
```

2. If I put `logging.level.org.springframework=debug` , then will it automatically give `logging.level.org.springframework.security=debug` logs also or do I have to mention both explicitly like below ?

```
logging.level.org.springframework=debug  
logging.level.org.springframework.security=debug
```