

### Role based Authorization - (Annotation)

- We have already covered different type of Authentications.
- And with that, we have also covered, how at Security Filter layer itself we can do authorization.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(...patterns: "/users").hasRole("USER")
                .anyRequest().authenticated()
            )
            .formLogin(Customizer.withDefaults());
        return http.build();
    }
}
```

- But in large scale applications, where we might have 100s of APIs, in that scenarios managing the roles at Security Filter might become difficult and cause scalable issue.

That's where, Annotation based "Role Based Authorization" comes into the picture. And these annotations are used within our Controller class.

SpringBoot (ConceptAndCoding) x

SpringBoot (ConceptAndCoding) x

SpringBoot (ConceptAndCoding) x

+

onedrive.live.com/view.aspx?resid=6B364628C29FEB52lsb06d0b4b3c6e43888c560d8ed130ea41&migratedtospo=true&redeem=aHR0cHM6Ly8xZHI2Lm1zL28vYy82YjM2NDYyOGMyOWZiYUyL0Vrc0xiYk1UEloRGpGWU5qdEV3NmtFQjd3UnhINXdkM...

SpringBoot (ConceptAndCoding) v

Three-tier, architecture based. Here basic authentication comes into the picture and these annotations are used within our Controller class.

(Servlet Container)

Tomcat

Request

Response

Filters Chain

Filter1

Security Filter Chain

Filter n

(Servlets)

Dispatcher Servlet

Servlets 2

Servlets n

Interceptors

Controller

```
UsernamePasswordAuthenticationToken [Principal=user, Credentials={PROTECTED}, Authenticated=false,
Details=null, Granted Authorities=[]]
```

This is how Authentication object looks like, currently Authenticated is false

Security Filter Chain

BasicAuthenticationFilter

1. Decode the Authorization Header

9. Save the Authentication Object in SecurityContext

SecurityContextHolder

Authentication object is stored in this context

AuthorizationFilter

2. Create and Pass the "Authentication" Object to

AuthenticationManager

3. Delegates Authentication to

AuthenticationProvider

4. Incoming raw password is hashed

PasswordEncoder

5. Fetch UserDetails Like username, password, Roles etc.

<<Interface>>

UserDetailsService

Either from

InMemoryUserDetailsManager

(manage username/password in Memory)

or

JdbcUserDetailsManager

(manage username/password in DB)

DB

6. Validate Incoming Username and hashed password with Stored username and hashed password.

DaoAuthenticationProvider

(handles username/Password)

7. Updated "Authentication" Object

AuthenticationManager

8. Return back fully "Authentication" object

BasicAuthenticationFilter

Default Implementation

ProviderManager

```
UsernamePasswordAuthenticationToken [Principal=org.springframework.security.core.userdetails.User,
Username=user, Password={PROTECTED}, Enabled=true, AccountNonExpired=true, CredentialsNonExpired=true,
AccountNonLocked=true, Granted Authorities=[ROLE_ADMIN], Credentials={PROTECTED}, Authenticated=true,
Details=WebAuthenticationDetails [RemoteIpAddress=0:0:0:0:0:0:0:1, SessionId=null], Granted
Authorities=[ROLE_ADMIN]]
```

This is how Authentication object looks like after successful authentication, Authenticated status changed to true.

Snipping Tool

Screenshot copied to clipboard

Automatically saved to screenshots folder.

Markup and share

35°C

Haze

Search

5:54 PM

6/2/2025

## Annotations for "Role Based Authorization"

@PreAuthorize

Does Authorization, before execution of the API.

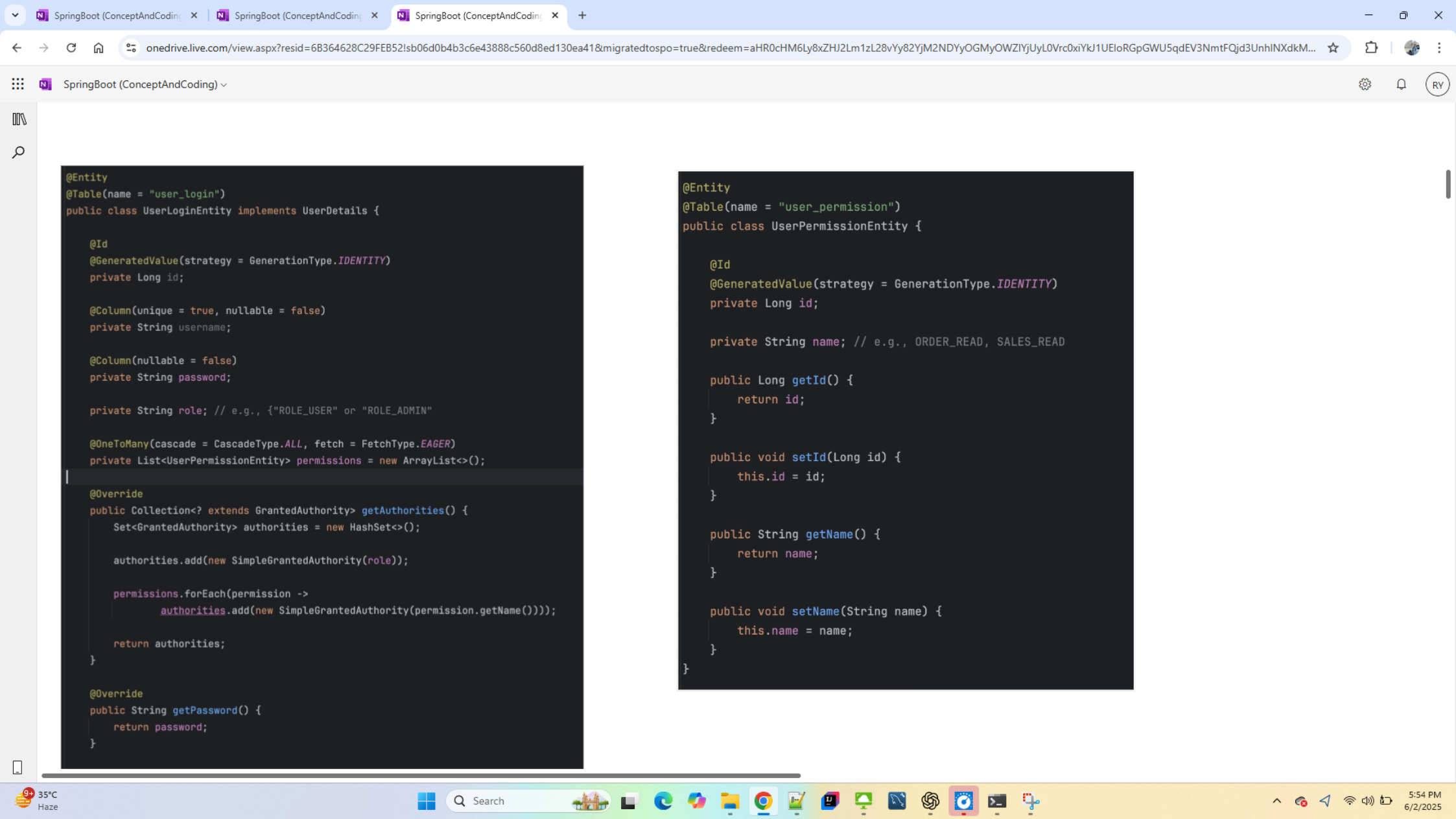
@PostAuthorize

Does Authorization after execution of the API but before sending the response back to the user.

Lets create User First (Dynamic one)

We have already seen its implementation in





```
@Entity
@Table(name = "user_login")
public class UserLoginEntity implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String username;

    @Column(nullable = false)
    private String password;

    private String role; // e.g., {"ROLE_USER" or "ROLE_ADMIN"}

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    private List<UserPermissionEntity> permissions = new ArrayList<>();

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        Set<GrantedAuthority> authorities = new HashSet<>();

        authorities.add(new SimpleGrantedAuthority(role));

        permissions.forEach(permission ->
            authorities.add(new SimpleGrantedAuthority(permission.getName())));

        return authorities;
    }

    @Override
    public String getPassword() {
        return password;
    }
}
```

```
@Entity
@Table(name = "User_permission")
public class UserPermissionEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name; // e.g., ORDER_READ, SALES_READ

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

SpringBoot (ConceptAndCoding) xSpringBoot (ConceptAndCoding) xSpringBoot (ConceptAndCoding) x

onedrive.live.com/view.aspx?resid=6B364628C29FEB52lsb06d0b4b3c6e43888c560d8ed130ea41&migratedtospo=true&redeem=aHR0cHM6Ly8xZHI2Lm1zL28vYy82YjM2NDYyOGMyOWZlYyUyL0Vrc0xiYk1UEloRGpGWU5qdEV3NmtFQjd3UnhINXdkm... RY

SpringBoot (ConceptAndCoding) v

```
public String getUsername() {
    return username;
}
```

```
@RestController
public class UserLoginController {

    @Autowired
    UserLoginEntityService userLoginEntityService;

    @Autowired
    PasswordEncoder passwordEncoder;

    @PostMapping("/user-login")
    public ResponseEntity<String> login(@RequestBody UserLoginEntity userLoginEntity) {
        // Hash the password before saving
        userLoginEntity.setPassword(passwordEncoder.encode(userLoginEntity.getPassword()));

        userLoginEntityService.save(userLoginEntity);

        return ResponseEntity.ok( body: "User registered successfully!");
    }
}
```

```
@Service
public class UserLoginEntityService implements UserDetailsService {

    @Autowired
    private UserLoginEntityRepository userLoginEntityRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        return userLoginEntityRepository.findByUsername(username)
            .orElseThrow(() -> new UsernameNotFoundException("user not found"));
    }

    public UserDetails save(UserLoginEntity userLoginEntity) {
        return userLoginEntityRepository.save(userLoginEntity);
    }
}
```

```
@Repository
public interface UserLoginEntityRepository extends JpaRepository<UserLoginEntity, Long> {

    Optional<UserLoginEntity> findByUsername(String username);
}
```

Now, we should be able to create user, and in this one User can have 1 role like ROLE\_ADMIN, ROLE\_USER etc.  
But we can give many permissions like ORDER\_READ, SALES\_DELETE etc.... (more granular level permissions)

Snipping Tool

Screenshot copied to clipboard  
Automatically saved to screenshots folder.

Markup and share

35°C  
Haze

Search

5:54 PM  
6/2/2025

SpringBoot (ConceptAndCoding)SpringBoot (ConceptAndCoding)SpringBoot (ConceptAndCoding)

onedrive.live.com/view.aspx?resid=6B364628C29FEB52lsb06d0b4b3c6e43888c560d8ed130ea41&migratedtospo=true&redeem=aHR0cHM6Ly8xZHI2Lm1zL28vYy82YjM2NDYyOGMyOWZlYjUyL0Vrc0xiYk11UEloRGpGWU5qdEV3NmtFQjd3UnhINXdkM...

SpringBoot (ConceptAndCoding)

But we can give many permissions like ORDER\_READ, SALES\_DELETE etc.... (more granular level permissions)

POSTlocalhost:8080/user-login

ParamsAuthorizationHeaders (9)BodyScriptsSettings

☐ none☐ form-data☐ x-www-form-urlencoded☒ raw☐ binary☐ GraphQLJSON

```
1 {
2   "username": "sj",
3   "password": "123",
4   "role": "ROLE_USER",
5   "permissions": [
6     {
7       "name": "ORDER_READ"
8     }
9   ]
10 }
11
```

Now, User Creation part is done, lets update our Security Config file.

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
public class SecurityConfig{

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

    @Bean
```

Without enabling this, @PreAuthorize and @PostAuthorize annotations will be ignored.

35°C Haze

Search

5:55 PM 6/2/2025

Now, User Creation part is done, lets update our Security Config file.

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
public class SecurityConfig{

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

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        http.authorizeHttpRequests(auth -> auth
            .requestMatchers(...patterns: "/user-login").permitAll()
            .anyRequest().authenticated())
            .sessionManagement(session -> session
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .csrf(csrf -> csrf.disable())
            .httpBasic(Customizer.withDefaults());

        return http.build();
    }
}
```

Without enabling this, @PreAuthorize and @PostAuthorize annotations will be ignored.

Now, lets create 2 Controller class, one for ORDER and another for SALES for testing purpose

```
@RestController
@RequestMapping("/api")
public class OrderController {

    @GetMapping("/orders")
    @PreAuthorize("hasRole('USER') and hasAuthority('ORDER_READ')")
    public ResponseEntity<String> readOrders() {
        return ResponseEntity.ok("All orders has been fetched successfully");
    }
}
```

```
@RestController
@RequestMapping("/api")
public class SalesController {

    @GetMapping("/sales")
    @PreAuthorize("hasAuthority('SALES_READ')")
    public ResponseEntity<String> readSalesDetails() {
        return ResponseEntity.ok("All Sales details has been fetched successfully");
    }
}
```

Above we have created user, who has both the permissions `ROLE_USER` and `ORDER_READ`, so API is successfully executed.

GET localhost:8080/api/orders

Params Authorization Headers (8) Body Scripts Settings

Type Basic Auth Username sj Password 123

The authorization header will be automatically generated when you send the request. Learn more about [Basic Auth](#) authorization.

Body Cookies (1) Headers (10) Test Results

Raw Preview Visualization

1 All orders has been fetched successfully

Now, when we try to access `/sales` API, which above user do not have permission, request has thrown exception.

GET localhost:8080/api/sales

Params Authorization Headers (8) Body Scripts Settings

Type Basic Auth Username sj Password 123

The authorization header will be automatically generated when you send the request. Learn more about [Basic Auth](#) authorization.

Body Cookies (1) Headers (11) Test Results

JSON Preview Visualization

```
1 {}
2 {"timestamp": "2025-04-24T05:37:42.823+00:00",
3  "status": 403,
4  "error": "Forbidden",
5  "path": "/api/sales"}
6
```



```
@PreAuthorize("hasRole('USER') and hasAuthority('ORDER_READ')")
```

Spring Boot, treat both "hasRole" and "hasAuthority" in a similar way, only difference is that :

For "hasRole" : "ROLE\_" is appended

#### SecurityExpressionRoot.java

```
private String defaultRolePrefix = "ROLE_";

@Override
public final boolean hasAuthority(String authority) {
    return hasAnyAuthority(authority);
}

@Override
public final boolean hasAnyAuthority(String... authorities) {
    return hasAnyAuthorityName(prefix: null, authorities);
}

@Override
public final boolean hasRole(String role) {
    return hasAnyRole(role);
}

@Override
public final boolean hasAnyRole(String... roles) {
    return hasAnyAuthorityName(this.defaultRolePrefix, roles);
}

private boolean hasAnyAuthorityName(String prefix, String... roles) {
    Set<String> roleSet = getAuthoritySet();
    for (String role : roles) {
        String defaultedRole = getRoleWithDefaultPrefix(prefix, role);
        if (roleSet.contains(defaultedRole)) {
            return true;
        }
    }
    return false;
}
```

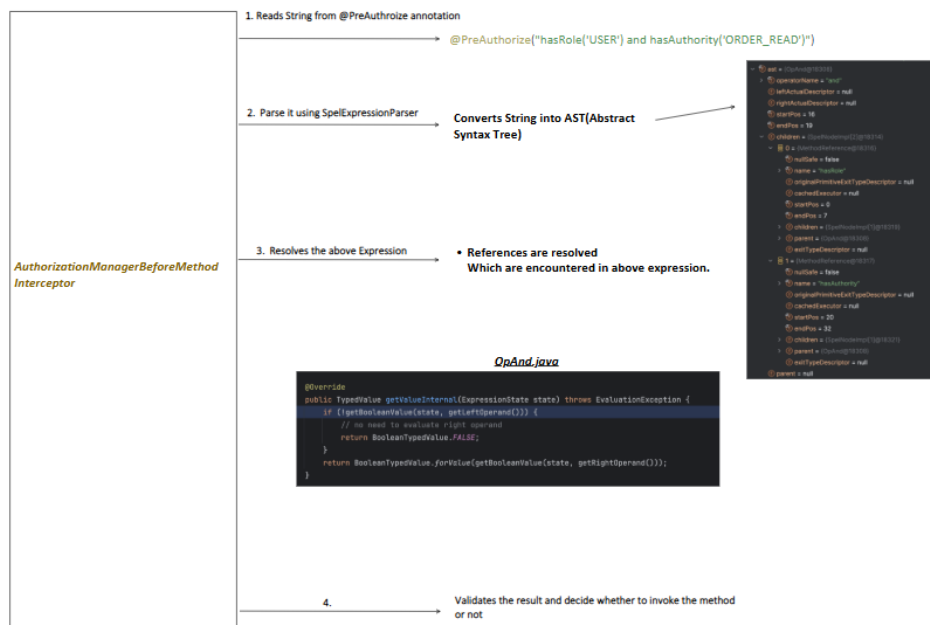


```
@PreAuthorize("hasRole('USER') and hasAuthority('ORDER_READ')")
```

This expression uses SpEL (i.e. Spring Expression Language)

Spring framework has *SpelExpressionParser* class, which help in compilation of these strings.

Below is the high level flow:



onedrive.live.com/view.aspx?resid=6B364628C29FEB52lsb06d0b4b3c6e43888c560d8ed130ea41&migratedtospo=true&redeem=aHR0cHM6Ly8xZHZJ2Lm1zL28vYy82YjM2NDYyOGMyOWZlYjUyL0Vrc0xiYk1UEl0RGpGWU5qdEV3NmtFQjd3UnhINXdkm...

SpringBoot (ConceptAndCoding) | Tell me what you want to do | Viewing | Share

More Logical Operators

Operator	Description	Example
and	Logical AND	hasRole('ADMIN') and hasAuthority('READ')
or	Logical OR	hasRole('ADMIN') or hasRole('USER')
not	Logical NOT	not hasRole('ADMIN')
!	Also logical NOT	!hasAuthority('DELETE')

Relational Operators

Operator	Description	Example
==	Equal	#value == 15
!=	Not equal	#value != 15
<	Less than	#value < 100
>	Greater than	#value > 100
<=	Less than or equal	#value <= 15
>=	Greater than or equal	#value >= 90

```
@PreAuthorize("#id == authentication.principal.id")
@GetMapping("/users/{id}")
public User fetchUserDetails(@PathVariable Long id) {
    return userServiceObject.fetchUserDetails(id);
}
```

```
authentication: getPrincipal()
Result:
- id = 1
- username = Y
- password = $1a108a0010d0d7b5d9f70u0n0zhe55-9wP-9n0kq0P0Y0z0R
- role = "ROLE_USER"
- permissions = [Permissions:1000] size = 1
```

@PostAuthorize

- Does Authorization after execution of the API but before sending the response back to the user.

• AuthorizationManagerAfterMethodInterceptor, is the one which intercept the @PostAuthorize annotation.

Let's see with an example

35°C Haze

Search

5:57 PM 6/2/2025

SpringBoot (ConceptAndCoding)SpringBoot (ConceptAndCoding)SpringBoot (ConceptAndCoding)

onedrive.live.com/view.aspx?resid=6B364628C29FEB52lsb06d0b4b3c6e43888c560d8ed130ea41&migratedtospo=true&redeem=aHR0cHM6Ly8xZHZJ2Lm1zL28vYy82YjM2NDYyOGMyOWZiYjUyL0Vrc0xiYk1UEloRGpGWU5qdEV3NmtFQjd3UnhINXdkm...

SpringBoot (ConceptAndCoding)

Let's see with an example

Created 2 Users

POSTlocalhost:8080/user-sign

Params

Authorization

Headers (0)

Body

Script

Settings

name

form-data

view-source:localhost

new

history

1

2

3

4

5

6

7

8

9

10

11

12

13

14

Body

Cookies (0)

Headers (0)

Test Results

Raw

Preview

Visualization

User registered successfully!

POSTlocalhost:8080/user-sign

Params

Authorization

Headers (0)

Body

Script

Settings

name

form-data

view-source:localhost

new

history

1

2

3

4

5

6

7

8

9

10

11

12

13

14

Body

Cookies (0)

Headers (0)

Test Results

Raw

Preview

Visualization

User registered successfully!

SELECT \* FROM USER\_LOGIN;

ID	PASSWORD	ROLE	USERNAME
1	\$2a\$10\$slOxKkIabEic83.MmGuN6O2CYqN8vccZjWMeMvBB0p6m8KQZ5u	ROLE_USER	a_user
2	\$2a\$10\$E8cC8SeChUpT09byTU0cXdykTNKX.9C6G2j2z29L200Yw.m	ROLE_USER	b_user

(2 rows, 6 ms)

@RestController

@RequestMapping("/api")

public class OrderController {

@GetMapping("/orders")

@PreAuthorize("hasRole('USER') and hasAuthority('ORDER\_READ')")

@PostAuthorize("returnObject.userID == authentication.principal.id")

public OrderDTO readOrders() {

OrderDTO orderDTO = new OrderDTO();

orderDTO.userID = 1; //hardcoding for now

orderDTO.orderID = 100001;

return orderDTO;

For testing purpose, I have hardcoded the user ID, and added the userID of "a\_user"

35°C

Haze

Search

5:58 PM

6/2/2025



Now, try to invoke this api with "b\_users" credentials

The screenshot shows the Swagger UI interface. At the top, the 'GET' method is selected for the endpoint 'localhost:8080/pet/index'. The 'Authorization' header is highlighted in red. The 'Type' dropdown is set to 'Basic Auth', and the 'Username' field contains 'j\_doe'. The 'Password' field contains '123'. A red 'X' icon is visible next to the password field. Below the dialog, the 'Body' tab is selected, showing a JSON response: {"id": 1, "name": "Fluffy", "status": "available", "petType": "cat"}. The response is displayed in a light blue background.

Now, try to invoke this api with "a\_users" credentials

