Role based Authorization - (Annotation)

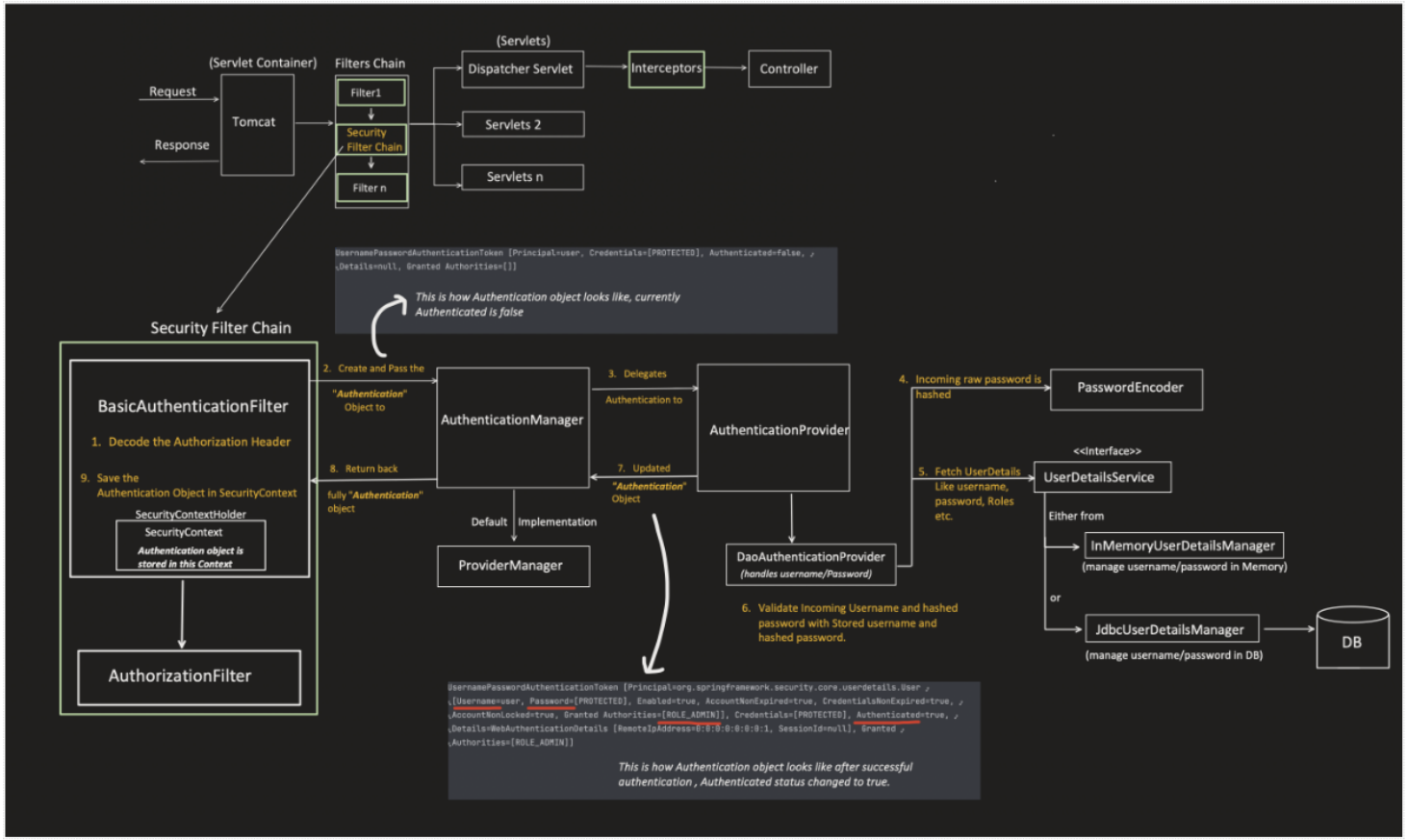
- We have already covered different type of Authentications.
- And with that, we have also covered, how at Security Filter layer itself we can do authorization.

```
@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.

Controller class.



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

```java
@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;
    }
```

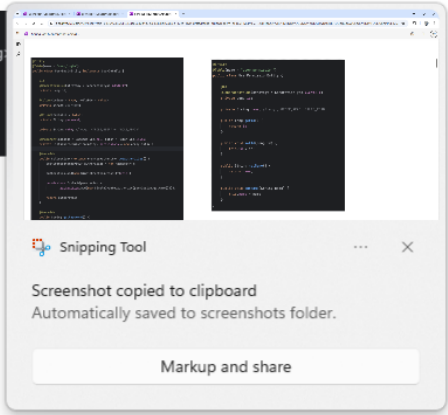
```java
@Entity
@Table(name = "user_permission")
public class UserPermissionEntity {

    @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;
    }
}
```

```
    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);
    }
}
```

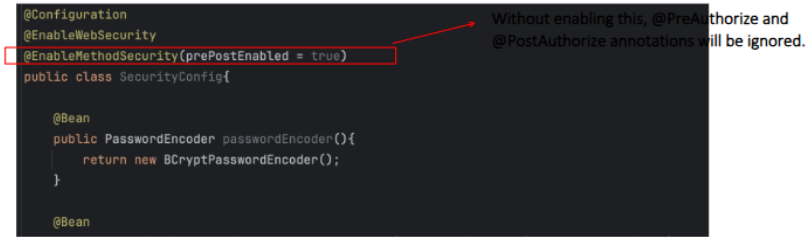
```
@Repository
public interface UserLoginEntityRepository extends JpaRepository<UserLoginEntity, Long

    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)

But we can give many permissions like ORDER_READ, SALES_DELETE etc.... (more granular level permissions)

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



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

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

```java
@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();
    }
}
```

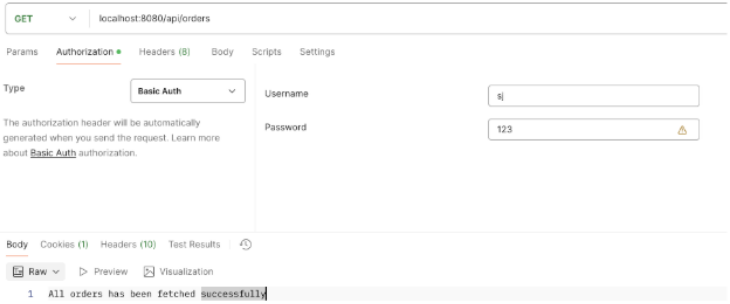
Without enabling this, @PreAuthorize and @PostAuthorize annotations will be ignored.

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

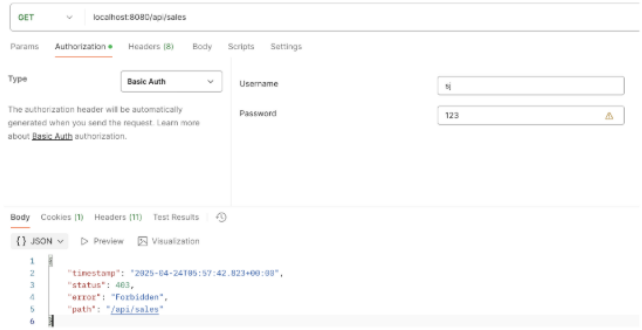
```java
@RestController
@RequestMapping("/api")
public class OrderController {

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

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

    @GetMapping("/sales")
    @PreAuthorize("hasAuthority('SALES_READ')")
    public ResponseEntity<String> readSalesDetails() {
        return ResponseEntity.ok( body: "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.



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



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

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

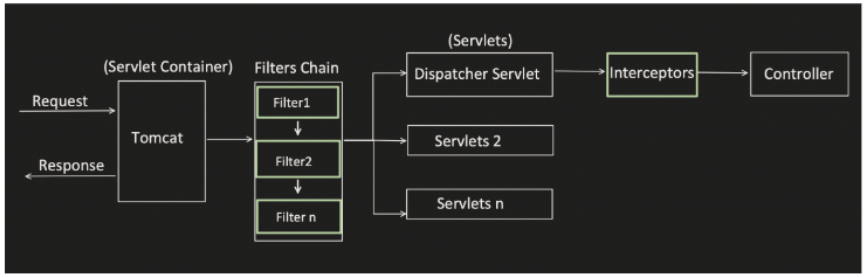
For "hasRole" : "ROLE_" is appended

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;
}
```

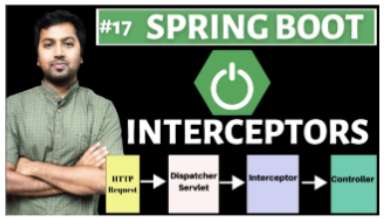
Now, one question comes to mind is, how this Authorization methods invokes before invocation of the Controller method.

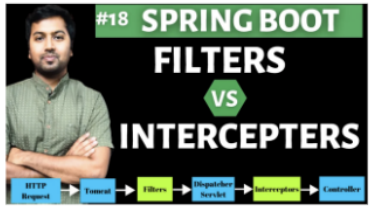
Its because of INTERCEPTORS



@PreAuthorize Annotation is intercepted by ***AuthorizationManagerBeforeMethodInterceptor***

What and how Interceptors works and how they are different from filters?
We have already covered both the topics in depth.





`@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:

AuthorizationManagerBeforeMethod Interceptor

1. Reads String from @PreAuthroize annotation → `@PreAuthorize("hasRole('USER') and hasAuthority('ORDER_READ')")`

2. Parse it using SpelExpressionParser → **Converts String into AST(Abstract Syntax Tree)**

3. Resolves the above Expression →
   - **References are resolved Which are encountered in above expression.**

OpAnd.java

```
@Override
public TypedValue getValueInternal(ExpressionState state) throws EvaluationException {
    if (!getBooleanValue(state, getLeftOperand())) {
        // no need to evaluate right operand
        return BooleanTypedValue.FALSE;
    }
    return BooleanTypedValue.forValue(getBooleanValue(state, getRightOperand()));
}
```

4. → Validates the result and decide whether to invoke the method or not

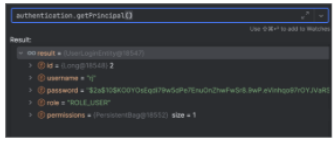
More Logical Operators

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);
}
```



@PostAuthorize

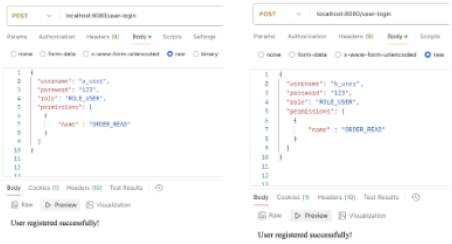
- Does Authorization after execution of the API but before sending the response back to the user.

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

Let's see with an example

Let's see with an example

Created 2 Users



SELECT * FROM USER_LOGIN

SELECT * FROM USER_LOGIN;

ID	PASSWORD	ROLE	USERNAME
1	$2a$10$oLOsKKbsbEsC83.MmGuNbO2CYqN8kvcCZW8/fwMvBB0p6mtKnQZ5u	ROLE_USER	a_user
2	$2a$10$Eb/CtSeC0uUpTd09iyTUOcXDjdKTNiX.9C6CjZj2zZ9L2OI0YW.m	ROLE_USER	b_user

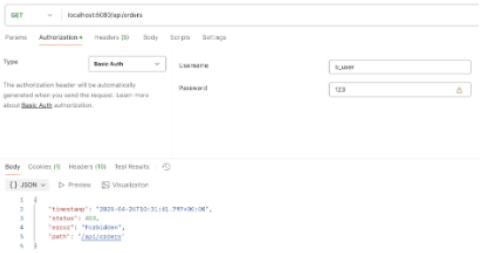
(2 rows, 6 ms)

```java
@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 = 11; //hardcoding for now
        orderDTO.orderID = 100001;
        return orderDTO;
    }
}
```

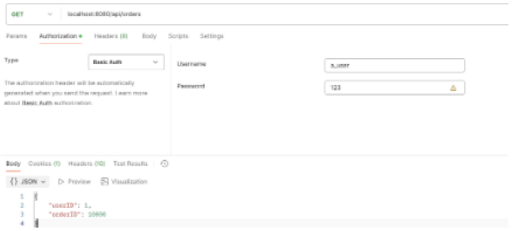
For testing purpose, I have hardcoded the user ID, and added the userID of "a_user"

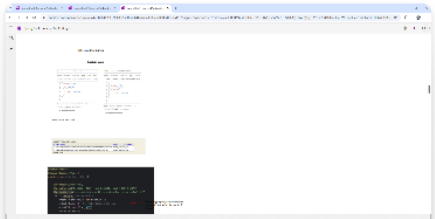
```
    }
}
```

Now, try to invoke this api with "b_users" credentials



Now, try to invoke this api with "a_users" credentials





Snipping Tool

Screenshot copied to clipboard
Automatically saved to screenshots folder.

Markup and share