



# Spring Boot Security JWT

Rawlabs Academy

# JSON Web Token (JWT)

JWT is an open standard (**RFC 7519**) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

This information can be verified and trusted because it is digitally signed.

JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.

More detail information please read the official documentation at [JWT.io](https://jwt.io).

# Before Implement JWT

m_user	
id	serial
created_date	datetime
modified_date	datetime
is_deleted	boolean
username	varchar(255)
password	varchar(255)

# Dependency


- spring-boot-starter-security
- jjwt-api
- jjwt-impl
- jjwt-jackson

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-security</artifactId>
4 </dependency>
5 <dependency>
6   <groupId>io.jsonwebtoken</groupId>
7   <artifactId>jjwt-api</artifactId>
8   <version>0.11.5</version>
9 </dependency>
10 <dependency>
11   <groupId>io.jsonwebtoken</groupId>
12   <artifactId>jjwt-impl</artifactId>
13   <version>0.11.5</version>
14   <scope>runtime</scope>
15 </dependency>
16 <dependency>
17   <groupId>io.jsonwebtoken</groupId>
18   <artifactId>jjwt-jackson</artifactId>
19   <version>0.11.5</version>
20   <scope>runtime</scope>
21 </dependency>
```

# Project Structure

```
1 .
2 | com.rawlabs.demospringboot/
3 | | component/           # Component class like a service but not for business logic
4 | | | ...
5 | | | config/           # Configuration
6 | | | | ...
7 | | | | constant/       # Constant class
8 | | | | | ...
9 | | | | controller/     # Presentation layer
10 | | | | | ...
11 | | | | domain/
12 | | | | | dao/         # Representation of Table
13 | | | | | | ...
14 | | | | | dto/         # POJO Class
15 | | | | | | ...
16 | | | | repository/    # Data access layer -> to database
17 | | | | | ...
18 | | | | service/       # Business logic layer
19 | | | | | ...
20 | | DemoSpringbootApplication.java
```

# Spring Boot Properties



```
1 jwt.token.validity=600000
2 jwt.signing.key=LLz0wjFXoLhdj4xfGX4gc192029JBRkcSF9DmPkyYV0n6gCAUa
3
4 spring.main.allow-bean-definition-overriding=true
5 spring.main.allow-circular-references=true
```

- **jwt.token.validity** value in milliseconds
- **jwt.signing.key** is random string value that should be **>=** 256 bit. At least **50 characters**
- Generate random string at [Random string generator](#)

# **Data Transfer Object**

## **(DTO)**

```
1 @Data
2 @Builder
3 @NoArgsConstructor
4 @AllArgsConstructor
5 public class LoginRequestDto {
6
7     @Schema(
8         requiredMode = Schema.RequiredMode.REQUIRED,
9         example = "user"
10    )
11    private String username;
12
13    @Schema(
14        requiredMode = Schema.RequiredMode.REQUIRED,
15        example = "password"
16    )
17    private String password;
18
19 }
```

```
1 @Data
2 @Builder
3 @NoArgsConstructor
4 @AllArgsConstructor
5 public class LoginResponseDto {
6
7     @Schema(
8         requiredMode = Schema.RequiredMode.REQUIRED,
9         example = "anyToken"
10    )
11    private String accessToken;
12
13    @Schema(
14        requiredMode = Schema.RequiredMode.REQUIRED
15    )
16    private LocalDateTime expiresIn;
17
18 }
```




# User DAO

`User` class will implement methods of `UserDetails` class like :

- `getAuthorities()` set return to **null**
- `isAccountNonExpired()` set return to **true**
- `isAccountNonLocked()` set return to **true**
- `isCredentialsNonExpired()` set return to **true**
- `isEnabled()` set return to **true**

```
1 @Data
2 @Builder
3 @NoArgsConstructor
4 @AllArgsConstructor
5 @Entity
6 @Table(name = "m_user")
7 @SQLDelete(sql = "update m_user set is_deleted = true where id = ?")
8 @Where(clause = "is_deleted = false")
9 public class User implements UserDetails {
10
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private Long id;
14
15     @Column(name = "created_date", nullable = false)
16     private LocalDateTime createdDate;
17
18     @Column(name = "modified_date")
19     private LocalDateTime modifiedDate;
20
21     @Column(name = "isDeleted", nullable = false)
22     private Boolean isDeleted;
23
24     @Column(name = "username", nullable = false)
25     private String username;
26
27     @Column(name = "password", nullable = false)
28     private String password;
29
30     // UserDetails implementation methods
31
32 }
```

# User Repository



```
1 @Repository
2 public interface UserRepository extends JpaRepository<User, Long> {
3
4     User findUserByUsername(String username);
5
6 }
```

# User Service

UserService

implemented the

UserDetailsService

interface to **override**  
the method

loadUserByUsername()

to be used for login  
validation.

```
1 @Service
2 public class UserService implements UserDetailsService {
3
4     private final UserRepository userRepository;
5
6     @Autowired
7     public UserService(UserRepository userRepository) {
8         this.userRepository = userRepository;
9     }
10
11     @Override
12     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
13         User user = userRepository.findUserByUsername(username);
14         if (user == null) {
15             throw new UsernameNotFoundException("invalid username or password");
16         }
17
18         return user;
19     }
20
21 }
```

# **Component Package**

# Token Provider

TokenProvider is a utility class as component which can be used to generate token, validate token and get token claims

Annotation `@Value` is used to get `application.properties` value

```
1 @Component
2 public class TokenProvider {
3
4     @Value("${jwt.token.validity}")
5     private long tokenExpiresIn;
6
7     @Value("${jwt.signing.key}")
8     private String jwtSigningKey;
9
10    private Key key;
11
12    @PostConstruct
13    public void init() {
14        final byte[] jwtSigningKeyBytes = Base64.getDecoder().decode(jwtSigningKey);
15        key = new SecretKeySpec(jwtSigningKeyBytes, 0, jwtSigningKeyBytes.length,
16                                SignatureAlgorithm.HS256.getJcaName());
17    }
18
19    public String getUsername(String token) {
20        return getClaimFromToken(token, Claims::getSubject);
21    }
22
23    public LocalDateTime getExpirationDate(String token) {
24        Date expiresIn = getClaimFromToken(token, Claims::getExpiration);
25        return expiresIn.toInstant().atZone(ZoneId.systemDefault()).toLocalDateTime();
26    }
27
28    private <T> T getClaimFromToken(String token, Function<Claims, T> claimResolver) {
29        final Claims claims = getAllClaimsFromToken(token);
30        return claimResolver.apply(claims);
31    }
32
33    private Claims getAllClaimsFromToken(String token) {
34        return Jwts.parserBuilder()
35            .setSigningKey(key)
36            .build()
37            .parseClaimsJws(token)
38            .getBody();
39    }
40
41    private boolean isExpired(String token) {
42        final LocalDateTime expirationDate = getExpirationDate(token);
43        return expirationDate.isBefore(LocalDateTime.now());
44    }
45
46    public String generateToken(Authentication authentication) {
47        return Jwts.builder()
48            .setSubject(authentication.getName())
49            .signWith(key)
50            .setIssuedAt(new Date())
51            .setExpiration(new Date(System.currentTimeMillis() + tokenExpiresIn))
52            .compact();
53    }
54
55    public boolean isTokenValid(String token, UserDetails userDetails) {
56        final String username = getUsername(token);
57        return (username.equals(userDetails.getUsername()) && !isExpired(token));
58    }
59
60    public Authentication getAuthenticationToken(final UserDetails userDetails) {
61        List<GrantedAuthority> authorities = new ArrayList<>();
62        authorities.add(new SimpleGrantedAuthority("USER"));
63        return new UsernamePasswordAuthenticationToken(userDetails, "", authorities);
64    }
65 }
```

# Unauthorized Entry Point

```
1 @Component
2 public class UnauthorizedEntryPoint implements AuthenticationEntryPoint {
3
4     @Override
5     public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException
6         authException) throws IOException, ServletException {
7         response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized");
8     }
9 }
```

`UnauthorizedEntryPoint` is used for error handling when get unauthorized client.

# **Configuration**

## **Package**

```

1 @Slf4j
2 public class JwtAuthenticationFilter extends GenericFilterBean {
3
4     private final UserDetailsService userDetailsService;
5     private final TokenProvider tokenProvider;
6
7     public JwtAuthenticationFilter(UserDetailsService userDetailsService, TokenProvider tokenProvider) {
8         this.userDetailsService = userDetailsService;
9         this.tokenProvider = tokenProvider;
10    }
11
12
13    @Override
14    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain
filterChain) throws IOException, ServletException {
15        HttpServletRequest httpRequest = (HttpServletRequest) servletRequest;
16        String authorization = httpRequest.getHeader(HttpHeaders.AUTHORIZATION);
17        String token = null;
18        String username = null;
19        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
20
21        final String BEARER = "Bearer ";
22        if (authorization != null && authorization.startsWith(BEARER)) {
23            token = authorization.replace(BEARER, "");
24
25            try {
26                username = tokenProvider.getUsername(token);
27            } catch (IllegalArgumentException e) {
28                log.error("An error occurred during getting username from token", e);
29            } catch (ExpiredJwtException e) {
30                log.error("Token is expired", e);
31            } catch (SignatureException e) {
32                log.error("Authentication Failed. Username or Password not valid.");
33            }
34        }
35
36        if (username != null && authentication == null) {
37            User user = (User) userDetailsService.loadUserByUsername(username);
38
39            if (tokenProvider.isTokenValid(token, user)) {
40                Authentication authenticationToken = tokenProvider.getAuthenticationToken(user);
41
42                log.debug("Authenticated user: {}, setting security context", username);
43                SecurityContextHolder.getContext().setAuthentication(authenticationToken);
44            }
45        }
46
47        filterChain.doFilter(servletRequest, servletResponse);
48    }
49 }

```



```

1 @Configuration
2 @EnableWebSecurity
3 public class WebSecurityConfig extends WebSecurityConfiguration {
4
5     private final UnauthorizedEntryPoint unauthorizedEntryPoint;
6
7     @Autowired
8     public WebSecurityConfig(UnauthorizedEntryPoint unauthorizedEntryPoint) {
9         this.unauthorizedEntryPoint = unauthorizedEntryPoint;
10    }
11
12    @Bean
13    public SecurityFilterChain filterChain(HttpSecurity http, TokenProvider tokenProvider, UserRepository
14    userRepository) throws Exception {
15        http.httpBasic().and().cors().and().csrf().disable()
16            .authorizeHttpRequests()
17            .requestMatchers("/auth/**").permitAll()
18            .anyRequest().authenticated().and()
19            .exceptionHandling().authenticationEntryPoint(unauthorizedEntryPoint).and()
20            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
21        http.addFilterBefore(new JwtAuthenticationFilter(userDetailsService(userRepository), tokenProvider),
22        UsernamePasswordAuthenticationFilter.class);
23        return http.build();
24    }
25
26    @Bean
27    public AuthenticationManager authenticationManager() {
28        return authentication -> {
29            String username = authentication.getPrincipal().toString();
30            return new UsernamePasswordAuthenticationToken(username, "", authentication.getAuthorities());
31        };
32    }
33
34    @Bean
35    public WebMvcConfigurer corsConfigurer() {
36        return new WebMvcConfigurer() {
37            @Override
38            public void addCorsMappings(CorsRegistry registry) {
39                registry.addMapping("/**")
40                    .allowedMethods("*");
41            }
42        };
43    }
44
45    @Bean
46    public UserDetailsService userDetailsService(UserRepository userRepository) {
47        return new UserService(userRepository);
48    }
49
50    @Bean
51    public WebSecurityCustomizer webSecurityCustomizer() {
52        return (web) -> web.ignoring()
53            .requestMatchers("/swagger-ui/**")
54            .requestMatchers("/api-docs/**");
55    }
56 }

```

# Login Service

The flow is user will be login and call method `doLogin()` and check if password has mathced from database user.

It will returning error when password does not match.

```
1 @Service
2 public class LoginService {
3
4     private final AuthenticationManager authenticationManager;
5     private final UserService userService;
6     private final TokenProvider tokenProvider;
7
8     @Autowired
9     public LoginService(AuthenticationManager authenticationManager, UserService userService,
10                        TokenProvider tokenProvider) {
11         this.authenticationManager = authenticationManager;
12         this.userService = userService;
13         this.tokenProvider = tokenProvider;
14     }
15
16     public LoginResponseDto doLogin(LoginRequestDto request) {
17         User user = (User) userService.loadUserByUsername(request.getUsername());
18
19         if (!user.getPassword().equals(request.getPassword())) {
20             throw new RuntimeException("Invalid username or password!");
21         }
22
23         final Authentication authentication = authenticationManager.authenticate(
24             new UsernamePasswordAuthenticationToken(request.getUsername(), request.getPassword())
25         );
26
27         SecurityContextHolder.getContext().setAuthentication(authentication);
28         final String token = tokenProvider.generateToken(authentication);
29
30         return LoginResponseDto.builder()
31             .accessToken(token)
32             .expiresIn(tokenProvider.getExpirationDate(token))
33             .build();
34     }
35
36 }
```

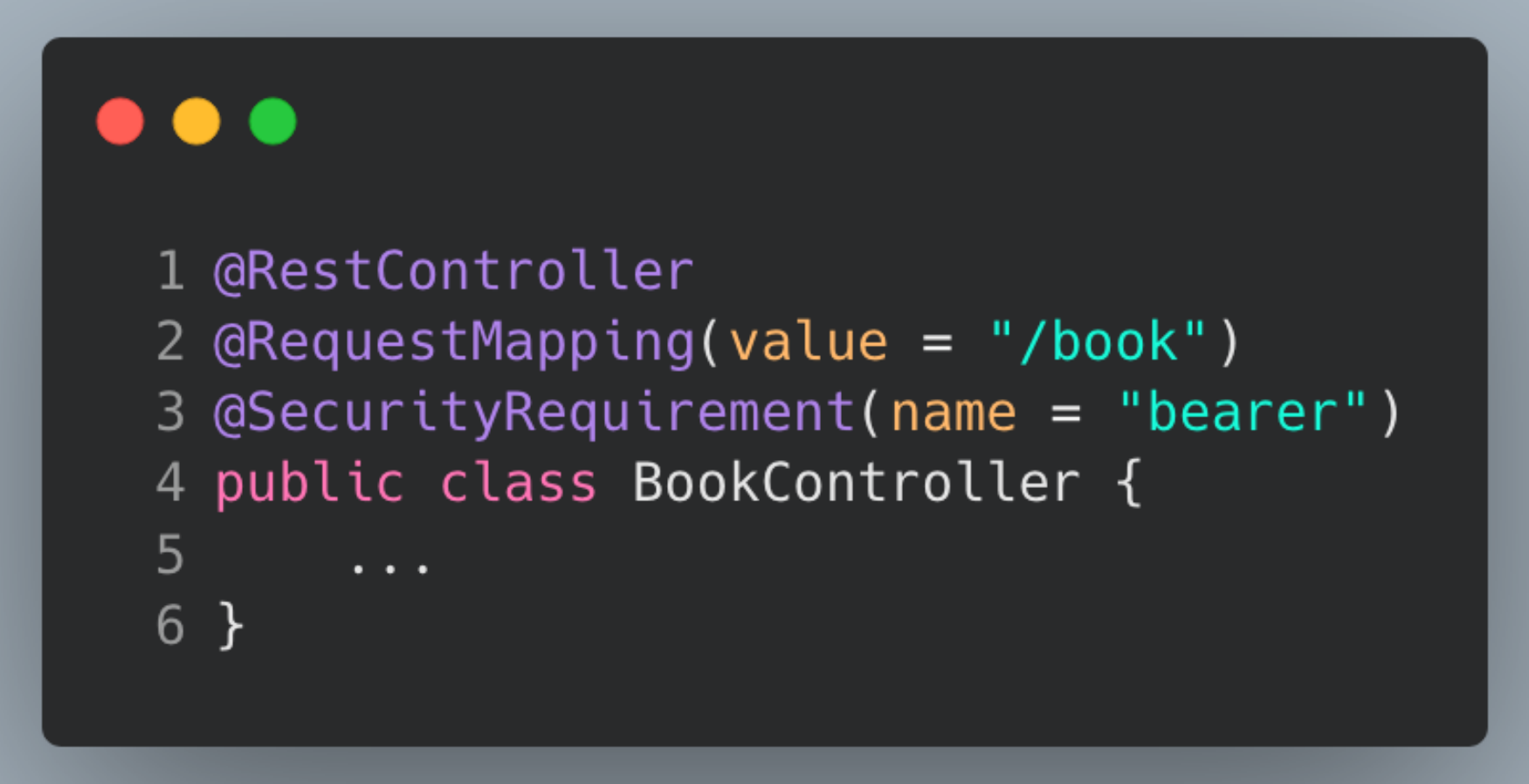
# Update API Docs Configuration

```
1 @Configuration
2 public class OpenAPIConfiguration {
3
4     @Bean
5     public OpenAPI openAPI() {
6         return new OpenAPI()
7             .components(
8                 new Components().addSecuritySchemes("bearer",
9                     new
10 SecurityScheme().type(SecurityScheme.Type.HTTP).scheme("bearer").bearerFormat("JWT"))
11             .info(
12                 new Info()
13                     .title("Rawlabs.ID")
14                     .description("Learn OpenAPI Documentation")
15                     .version("1.0.0")
16                     .contact(
17                         new Contact()
18                             .name("Maverick")
19                             .email("maverick@mail.com")
20                             .url("https://piinalpin.com/")
21                     )
22             );
23     }
24
25 }
```

# Login Controller

```
1 @RestController
2 @RequestMapping(value = "/auth")
3 public class LoginController {
4
5     private final LoginService loginService;
6
7     @Autowired
8     public LoginController(LoginService loginService) {
9         this.loginService = loginService;
10    }
11
12    @PostMapping(value = "/login", produces = MediaType.APPLICATION_JSON_VALUE)
13    @Operation(summary = "Login")
14    @ApiResponses(value = {
15        @ApiResponse(responseCode = "200", description = "Success")
16    })
17    public LoginResponseDto login(@RequestBody LoginRequestDto request) {
18        return loginService.doLogin(request);
19    }
20
21 }
```

# Update Authenticated Controller



```
1 @RestController
2 @RequestMapping(value = "/book")
3 @SecurityRequirement(name = "bearer")
4 public class BookController {
5     ...
6 }
```

# Testing

When everything is done, do testing with inject data to database `username` and `password` and then hit endpoint using authorization.

