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.

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`

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
```

Project Structure

```
.
└── com.rawlabs.demospringboot/
    ├── component/          # Component class like a service but not for business logic
    │   └── ...
    ├── config/             # Configuration
    │   └── ...
    ├── constant/           # Constant class
    │   └── ...
    ├── controller/         # Presentation layer
    │   └── ...
    ├── domain/
    │   ├── dao/            # Representation of Table
    │   │   └── ...
    │   └── dto/            # POJO Class
    │       └── ...
    ├── repository/         # Data access layer -> to database
    │   └── ...
    ├── service/            # Business logic layer
    │   └── ...
    └── DemoSpringbootApplication.java
```

Spring Boot Properties

```
1 jwt.token.validity=600000
2 jwt.signing.key=lLz0wjFXoLhdj4xfGX4gc192O29JBRkcSF9DmPkyYVOn6gCAUa
3
4 spring.main.allow-bean-definition-overriding=true
5 spring.main.allow-circular-references=true
```

- `jwt.token.validity` value in miliseconds
- `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)

```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class LoginRequestDto {

    @Schema(
            requiredMode = Schema.RequiredMode.REQUIRED,
            example = "user"
    )
    private String username;

    @Schema(
            requiredMode = Schema.RequiredMode.REQUIRED,
            example = "password"
    )
    private String password;

}
```

```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class LoginResponseDto {

    @Schema(
            requiredMode = Schema.RequiredMode.REQUIRED,
            example = "anyToken"
    )
    private String accessToken;

    @Schema(
            requiredMode = Schema.RequiredMode.REQUIRED
    )
    private LocalDateTime expiresIn;

}
```

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**

```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "m_user")
@SQLDelete(sql = "update m_user set is_deleted = true where id = ?")
@Where(clause = "is_deleted = false")
public class User implements UserDetails {

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

    @Column(name = "created_date", nullable = false)
    private LocalDateTime createdDate;

    @Column(name = "modified_date")
    private LocalDateTime modifiedDate;

    @Column(name = "isDeleted", nullable = false)
    private Boolean isDeleted;

    @Column(name = "username", nullable = false)
    private String username;

    @Column(name = "password", nullable = false)
    private String password;

    // UserDetails implementation methods

}
```

User Repository

```java
@Repository
public interface UserRepository extends JpaRepository<User, Long> {

    User findUserByUsername(String username);

}
```

User Service

`UserService` implemented the `UserDetailService` interface to **override** the method `loadUserByUsername()` to be used for login validation.

```java
@Service
public class UserService implements UserDetailsService {

    private final UserRepository userRepository;

    @Autowired
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findUserByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException("invalid username or password");
        }

        return user;
    }

}
```

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

```java
@Component
public class TokenProvider {

    @Value("${jwt.token.validity}")
    private long tokenExpiresIn;

    @Value("${jwt.signing.key}")
    private String jwtSigningKey;

    private Key key;

    @PostConstruct
    public void init() {
        final byte[] jwtSigningKeyBytes = Base64.getDecoder().decode(jwtSigningKey);
        key = new SecretKeySpec(jwtSigningKeyBytes, 0, jwtSigningKeyBytes.length,
SignatureAlgorithm.HS256.getJcaName());
    }

    public String getUsername(String token) {
        return getClaimFromToken(token, Claims::getSubject);
    }

    public LocalDateTime getExpirationDate(String token) {
        Date expiresIn = getClaimFromToken(token, Claims::getExpiration);
        return expiresIn.toInstant().atZone(ZoneId.systemDefault()).toLocalDateTime();
    }

    private <T> T getClaimFromToken(String token, Function<Claims, T> claimResolver) {
        final Claims claims = getAllClaimsFromToken(token);
        return claimResolver.apply(claims);
    }

    private Claims getAllClaimsFromToken(String token) {
        return Jwts.parserBuilder()
                .setSigningKey(key)
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    private boolean isExpired(String token) {
        final LocalDateTime expirationDate = getExpirationDate(token);
        return expirationDate.isBefore(LocalDateTime.now());
    }

    public String generateToken(Authentication authentication) {
        return Jwts.builder()
                .setSubject(authentication.getName())
                .signWith(key)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + tokenExpiresIn))
                .compact();
    }

    public boolean isTokenValid(String token, UserDetails userDetails) {
        final String username = getUsername(token);
        return (username.equals(userDetails.getUsername()) && !isExpired(token));
    }

    public Authentication getAuthenticationToken(final UserDetails userDetails) {
        List<GrantedAuthority> authorities = new ArrayList<>();
        authorities.add(new SimpleGrantedAuthority("USER"));
        return new UsernamePasswordAuthenticationToken(userDetails, "", authorities);
    }

}
```

Unauthorized Entry Point

```java
@Component
public class UnauthorizedEntryPoint implements AuthenticationEntryPoint {

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException) throws IOException, ServletException {
        response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized");
    }

}
```

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

Configuration

Package

```java
@Slf4j
public class JwtAuthenticationFilter extends GenericFilterBean {

    private final UserDetailsService userDetailsService;
    private final TokenProvider tokenProvider;

    public JwtAuthenticationFilter(UserDetailsService userDetailsService, TokenProvider tokenProvider) {
        this.userDetailsService = userDetailsService;
        this.tokenProvider = tokenProvider;
    }


    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
        HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;
        String authorization = httpServletRequest.getHeader(HttpHeaders.AUTHORIZATION);
        String token = null;
        String username = null;
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

        final String BEARER = "Bearer ";
        if (authorization != null && authorization.startsWith(BEARER)) {
            token = authorization.replace(BEARER, "");

            try {
                username = tokenProvider.getUsername(token);
            } catch (IllegalArgumentException e) {
                log.error("An error occured during getting username from token", e);
            } catch (ExpiredJwtException e) {
                log.error("Token is expired", e);
            } catch (SignatureException e) {
                log.error("Authentication Failed. Username or Password not valid.");
            }
        }

        if (username != null && authentication == null) {
            User user = (User) userDetailsService.loadUserByUsername(username);

            if (tokenProvider.isTokenValid(token, user)) {
                Authentication authenticationToken = tokenProvider.getAuthenticationToken(user);

                log.debug("Authenticated user: {}, setting security context", username);
                SecurityContextHolder.getContext().setAuthentication(authenticationToken);
            }
        }

        filterChain.doFilter(servletRequest, servletResponse);
    }
}
```

```java
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfiguration {

    private final UnauthorizedEntryPoint unauthorizedEntryPoint;

    @Autowired
    public WebSecurityConfig(UnauthorizedEntryPoint unauthorizedEntryPoint) {
        this.unauthorizedEntryPoint = unauthorizedEntryPoint;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http, TokenProvider tokenProvider, UserRepository userRepository) throws Exception {
        http.httpBasic().and().cors().and().csrf().disable()
                .authorizeHttpRequests()
                .requestMatchers("/auth/**").permitAll()
                .anyRequest().authenticated().and()
                .exceptionHandling().authenticationEntryPoint(unauthorizedEntryPoint).and()
                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        http.addFilterBefore(new JwtAuthenticationFilter(userDetailsService(userRepository), tokenProvider), UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

    @Bean
    public AuthenticationManager authenticationManager() {
        return authentication -> {
            String username = authentication.getPrincipal().toString();
            return new UsernamePasswordAuthenticationToken(username, "", authentication.getAuthorities());
        };
    }

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                        .allowedMethods("*");
            }
        };
    }

    @Bean
    public UserDetailsService userDetailsService(UserRepository userRepository) {
        return new UserService(userRepository);
    }

    @Bean
    public WebSecurityCustomizer webSecurityCustomizer() {
        return (web) -> web.ignoring()
                .requestMatchers("/swagger-ui/**")
                .requestMatchers("/api-docs/**");
    }

}
```

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.

```java
@Service
public class LoginService {

    private final AuthenticationManager authenticationManager;
    private final UserService userService;
    private final TokenProvider tokenProvider;

    @Autowired
    public LoginService(AuthenticationManager authenticationManager, UserService userService,
                        TokenProvider tokenProvider) {
        this.authenticationManager = authenticationManager;
        this.userService = userService;
        this.tokenProvider = tokenProvider;
    }

    public LoginResponseDto doLogin(LoginRequestDto request) {
        User user = (User) userService.loadUserByUsername(request.getUsername());

        if (!user.getPassword().equals(request.getPassword())) {
            throw new RuntimeException("Invalid username or password!");
        }

        final Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(request.getUsername(), request.getPassword())
        );

        SecurityContextHolder.getContext().setAuthentication(authentication);
        final String token = tokenProvider.generateToken(authentication);

        return LoginResponseDto.builder()
                .accessToken(token)
                .expiresIn(tokenProvider.getExpirationDate(token))
                .build();
    }

}
```

Update API Docs Configuration

```java
@Configuration
public class OpenAPIConfiguration {

    @Bean
    public OpenAPI openAPI() {
        return new OpenAPI()
                .components(
                        new Components().addSecuritySchemes("bearer",
                                new
SecurityScheme().type(SecurityScheme.Type.HTTP).scheme("bearer").bearerFormat("JWT"))
                )
                .info(
                        new Info()
                                .title("Rawlabs.ID")
                                .description("Learn OpenAPI Documentation")
                                .version("1.0.0")
                                .contact(
                                        new Contact()
                                                .name("Maverick")
                                                .email("maverick@mail.com")
                                                .url("https://piinalpin.com/")
                                )
                );
    }

}
```

Login Controller

```java
@RestController
@RequestMapping(value = "/auth")
public class LoginController {

    private final LoginService loginService;

    @Autowired
    public LoginController(LoginService loginService) {
        this.loginService = loginService;
    }

    @PostMapping(value = "/login", produces = MediaType.APPLICATION_JSON_VALUE)
    @Operation(summary = "Login")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Success")
    })
    public LoginResponseDto login(@RequestBody LoginRequestDto request) {
        return loginService.doLogin(request);
    }

}
```

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.


THANK YOU