Spring Security (Jwt/Role)

Mục lục

1. **Jwt**

1.1 Khái niệm

JWT (JSON Web Token) là một tiêu chuẩn mã nguồn mở (RFC 7519) dùng để truyền tải thông tin an toàn, gọn nhẹ và khép kín giữa các bên tham gia dưới format JSON.

1.2 Cấu tạo của Jwt

JWT gồm 3 phần chính, và phần tách nhau bằng một dấu chấm (.):

- Header
- Payload
- Signature

1.2.1 Header

Header thông thường sẽ bao gồm 2 thành phần, gồm : Kiểu token, mà với trường hợp này luôn luôn là JWT, và thuật toán mã hoá được sử dụng.

1.2.2 Payload

Payload trong JWT chứa các claims.

Trong lĩnh vực an toàn thông tin, claims đề cập đến sự tuyên bố quyền truy cập hoặc quyền sử dụng tài nguyên.

Claims là tập hợp các thông tin đại diện cho một thực thể (object) (ví dụ : user_id) và một số thông tin đi kèm. Claims sẽ có dạng Key - Value. Do đó, chúng ta có thể hiểu rằng, claims ám chỉ việc yêu cầu truy xuất tài nguyên cho object tương ứng.

Có 3 kiểu claims, bao gồm registered, public, and private claims.

1.2.3 Signature

Signature là phần cuối cùng của JWT, có chức năng xác thực danh tính người gửi. Để tạo ra một signature chính xác, ta cần encode phần header, phần payload, chọn cryptography (mật mã khoá) thích hợp đã được xác định ở header kèm secret key và thực hiện sign.

1.3 Ưu và nhược điểm Jwt

1.3.1 Ưu điểm

- Gọn nhẹ: JWT nhỏ gọn, chi phí truyền tải thấp giúp tăng hiệu suất của các ứng dụng.
- Bảo mật: JWT sử dụng các mật mã khoá để tiến hành xác thực người danh tính người dùng. Ngoài ra, cấu trúc của JWT cho phép chống giả mạo nên thông tin được đảm bảo an toan trong quá trình trao đổi.
- Phổ thông: JWT được sử dụng dựa trên JSON, là một dạng dữ liệu phổ biến, có thể sử dụng ở hầu hết các ngôn ngữ lập trình. Ngoài ra, triển khai JWT tương đối dễ dàng và tích hợp được với nhiều thiết bị, vì JWT đã tương đối phổ biến.

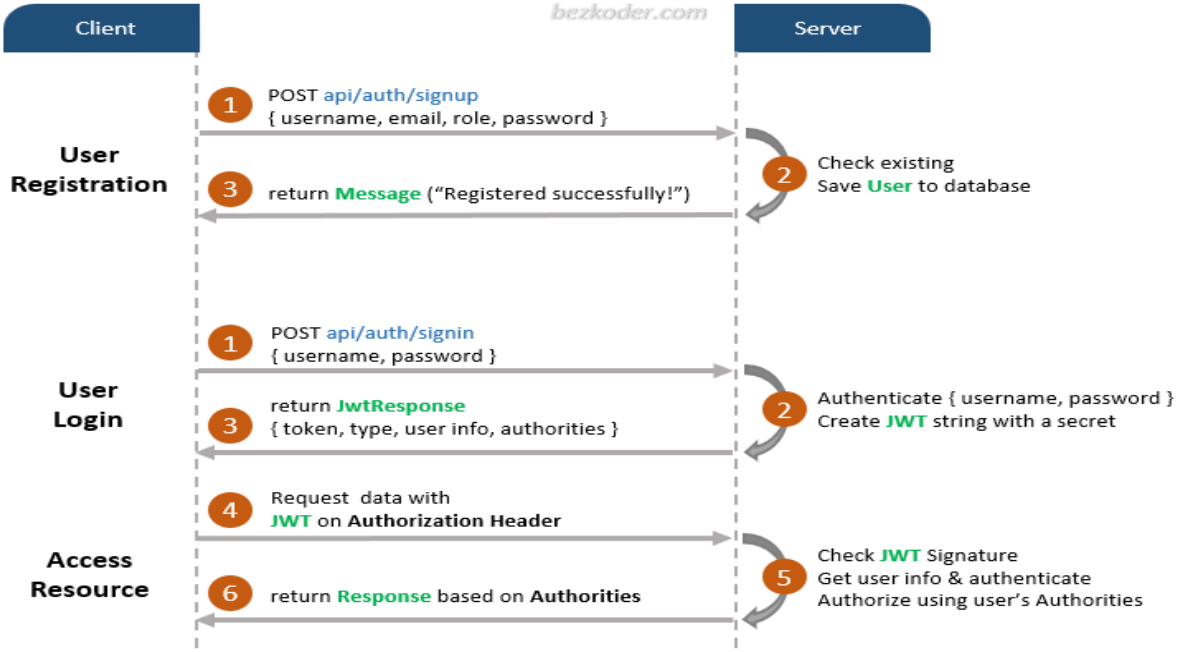
1.3.2 Nhược điểm

- Kích thước: Mặc dù trong tài liệu không ghi cụ thể giới hạn, nhưng do được truyền trên HTTP Header, vì thế, JWT có giới hạn tương đương với HTTP Header (khoảng 8KB).

- Rủi ro bảo mật: Khi sử dụng JWT không đúng cách, ví dụ như không kiểm tra tính hợp lệ của signature, không kiểm tra expire time, kẻ tấn công có thể lợi dụng sơ hở để truy cập vào các thông tin trái phép.

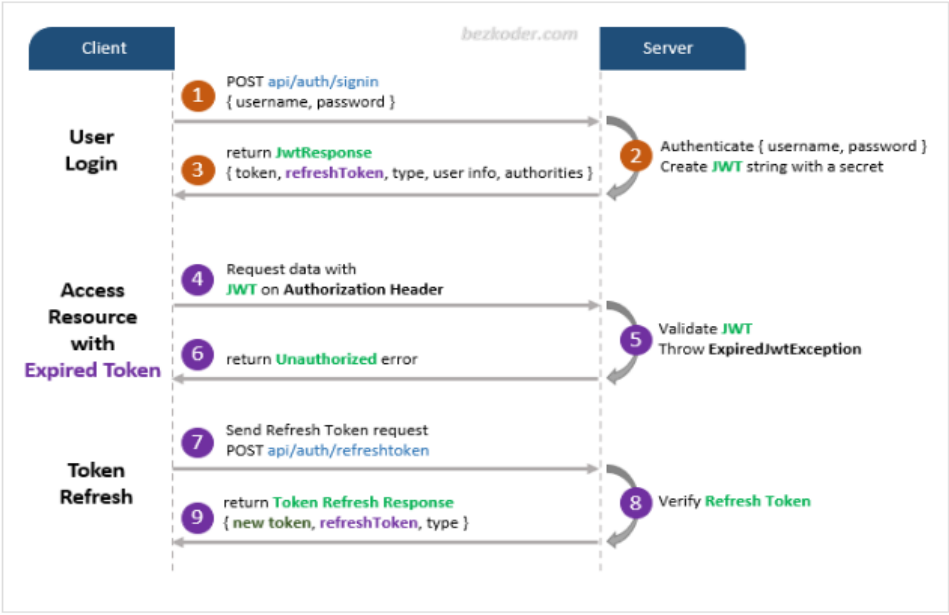
1.4 Luồng hoạt động đăng ký, đăng nhập và refresh token trong Jwt

1.4.1 Luồng hoạt động đăng nhập và đăng ký.



1.4.2 Luồng hoạt động refresh token

Refresh Token.



2. Role

2.1 Authorize HttpServletRequests

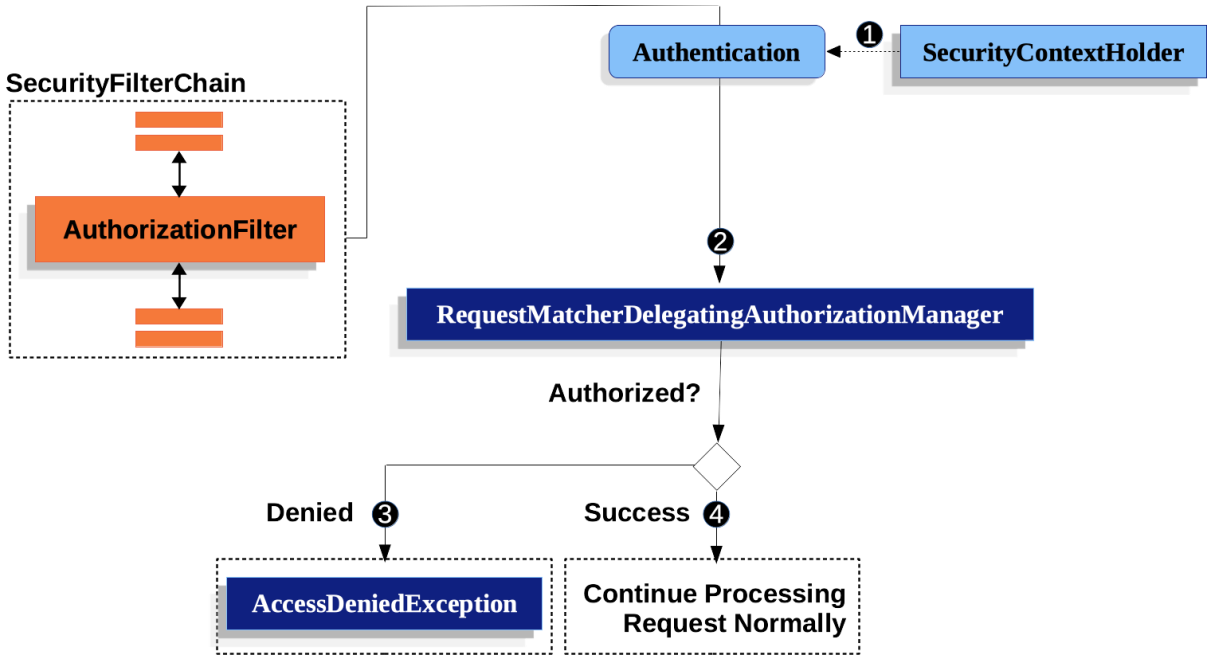
Spring Security cho phép bạn mô hình hóa quyền ở mức độ yêu cầu. Ví dụ, với Spring Security, bạn có thể chỉ định rằng tất cả các trang dưới đường dẫn /admin đều yêu cầu một quyền cụ thể, trong khi tất cả các trang khác chỉ đơn giản yêu cầu xác thực.

Mặc định, Spring Security yêu cầu mọi yêu cầu đều được xác thực. Tuy nhiên, mỗi khi bạn sử dụng một đối tượng HttpSecurity, bạn cần khai báo các quy tắc xác thực của mình.

```java
http
    .authorizeHttpRequests((authorize) -> authorize
        .anyRequest().authenticated()
    )
```

Đoạn mã trên thông báo cho Spring Security rằng mọi điểm cuối trong ứng dụng của bạn đều yêu cầu ngữ cảnh bảo mật ít nhất là được xác thực để cho phép truy cập.

2.2 Cách Authorization làm việc



1. Đầu tiên, AuthorizationFilter xây dựng Nhà cung cấp để truy xuất Xác thực từ SecurityContextHolder.
2. Thứ hai, nó chuyển Nhà cung cấp<Authentication> và HttpServletRequest tới Trình quản lý ủy quyền. AuthorizationManager khớp yêu cầu với các mẫu trong AuthorizeHttpRequests và chạy quy tắc tương ứng.
3. Nếu ủy quyền bị từ chối, AuthorizationDeniedEvent sẽ được xuất bản và AccessDeniedException sẽ được ném ra. Trong trường hợp này, ExceptionTranslationFilter xử lý AccessDeniedException.
4. Nếu quyền truy cập được cấp, AuthorizationGrantedEvent sẽ được xuất bản và AuthorizationFilter tiếp tục với FilterChain cho phép ứng dụng xử lý bình thường.

2.3 Tất cả Dispatches đều được uỷ quyền

AuthorizationFilter không chỉ chạy với mọi request và còn với mọi dispatch.

```java
@Controller
public class MyController {
    @GetMapping("/endpoint")
    public String endpoint() {
        return "endpoint";
    }
}
```

Trong trường hợp này, việc ủy quyền diễn ra hai lần; một lần để ủy quyền/endpoint và một lần để chuyển tiếp tới Thymeleaf để hiển thị mẫu “endpoint”.

```java
@Controller
public class MyController {
    @GetMapping("/endpoint")
    public String endpoint() {
        throw new UnsupportedOperationException("unsupported");
    }
}
```

Trong trường hợp đó, việc ủy quyền cũng xảy ra hai lần; một lần để ủy quyền/endpoint và một lần để gửi lỗi.

2.4 Yêu cầu uỷ quyền.

Tổng kết nhanh về các quy tắc xác thực được tích hợp vào DSL (Domain-Specific Language) của Spring Security:

- permitAll: Yêu cầu không cần xác thực và là một điểm cuối công khai; lưu ý rằng trong trường hợp này, Authentication không bao giờ được lấy từ phiên.
- denyAll: Yêu cầu không được phép dưới mọi tình huống; lưu ý rằng trong trường hợp này, Authentication không bao giờ được lấy từ phiên.
- hasAuthority: Yêu cầu Authentication phải có một GrantedAuthority khớp với giá trị cung cấp.
- hasRole: Một biến thể của hasAuthority giúp ngắn gọn hóa cho trường hợp sử dụng ROLE_ hoặc bất kỳ tiền tố mặc định nào được cấu hình.
- hasAnyAuthority: Yêu cầu Authentication phải có một GrantedAuthority khớp với bất kỳ giá trị nào trong số những giá trị được cung cấp.
- hasAnyRole: Một biến thể của hasAnyAuthority giúp ngắn gọn hóa cho trường hợp sử dụng ROLE_ hoặc bất kỳ tiền tố mặc định nào được cấu hình.
- access: Yêu cầu sử dụng AuthorizationManager tùy chỉnh để xác định quyền truy cập.

```java
@Bean
SecurityFilterChain web(HttpSecurity http) throws Exception {
    http
        // ...
        .authorizeHttpRequests(authorize -> authorize                                    ❶
            .dispatcherTypeMatchers(FORWARD, ERROR).permitAll() ❷
            .requestMatchers("/static/**", "/signup", "/about").permitAll()          ❸
            .requestMatchers("/admin/**").hasRole("ADMIN")                               ❹
            .requestMatchers("/db/**").access(allOf(hasAuthority('db'), hasRole('ADMIN')))    ❺
            .anyRequest().denyAll()                                                      ❻
        );

    return http.build();
}
```

- Dispatches FORWARD và ERROR được phép để cho phép Spring MVC hiển thị views và Spring Boot hiển thị lỗi.
- Các mẫu URL được chỉ định mà bất kỳ người dùng nào cũng có thể truy cập.
- Các URL bắt đầu bằng "/admin/" sẽ bị giới hạn đối với người dùng có vai trò "ADMIN".
- Các URL bắt đầu bằng "/db/" yêu cầu người dùng phải được cấp quyền "db" và là "ROLE_ADMIN".
- Bất kỳ URL nào chưa được khớp sẽ bị từ chối truy cập, giúp đảm bảo rằng các quy tắc xác thực không bị quên.

3. Triển khai Authentication và Authorization cơ bản.

3.1 SecurityConfig

```java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    @Autowired
    private JwtAuthenticationEntrypoint entrypoint;

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

    @Autowired
    private  JwtUtil jwt;

    @Autowired UserDetailService userDetailService;

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

        http.cors(AbstractHttpConfigurer::disable);
        http.csrf(AbstractHttpConfigurer::disable)
        .exceptionHandling(exception -> exception.authenticationEntryPoint(entrypoint))
        .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
        .authorizeHttpRequests(auth -> {
            auth.requestMatchers("/user/**").permitAll();
            auth.requestMatchers("/admin/**").hasRole("ADMIN");
            auth.anyRequest().authenticated();
        });
        http.addFilterBefore(new JwtAuthenticaionFilter( userDetailService,jwt), UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
}
```

1. PasswordEncoder Bean: Định nghĩa một Bean cho PasswordEncoder, trong trường hợp này là BCryptPasswordEncoder, được sử dụng để mã hóa mật khẩu người dùng.
2. Autowired Beans: Inject các Bean cần thiết như JwtAuthenticationEntrypoint, JwtUtil, và UserDetailService.

3. SecurityFilterChain Bean: Định nghĩa một Bean cho SecurityFilterChain, là nơi bạn đặt cấu hình bảo mật. Cấu hình này bao gồm:

- Tắt CORS: Vô hiệu hóa CORS (Cross-Origin Resource Sharing) để cho phép gửi yêu cầu từ các nguồn khác nhau.
- Tắt CSRF Protection: Vô hiệu hóa bảo vệ chống tấn công CSRF (Cross-Site Request Forgery).
- Xử lý Ngoại lệ Xác thực: Đặt một điểm vào điểm (entry point) xác thực dựa trên JwtAuthenticationEntryPoint để xử lý các ngoại lệ xác thực.
- Quản lý Phiên: Đặt chính sách quản lý phiên là STATELESS để chỉ sử dụng xác thực dựa trên token và không lưu trạng thái phiên trên máy chủ.
- Authorize Requests: Xác định quy tắc xác thực cho các yêu cầu. Cụ thể, cho phép mọi yêu cầu bắt đầu bằng "/user/" và "/admin/" mà không cần xác thực, yêu cầu "/admin/" chỉ được phép với vai trò "ADMIN", và yêu cầu còn lại đều yêu cầu xác thực.

4. Thêm Filter: Thêm một Filter (JwtAuthenticationFilter) vào chuỗi Filter trước UsernamePasswordAuthenticationFilter. Filter này sẽ xử lý việc xác thực dựa trên JWT và cung cấp xác thực cho người dùng.

3.2 Bộ lọc authentication

```java
public class JwtAuthenticaionFilter extends OncePerRequestFilter{

    private final UserDetailService userDetailService;

    private final JwtUtil jwt;


    public JwtAuthenticaionFilter(UserDetailService userDetailService, JwtUtil jwt) {
        super();
        this.userDetailService = userDetailService;
        this.jwt = jwt;
    }
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {

        String token = getToken(request);
        try {
            if(token != null) {
                String username = jwt.extractUsername(token);
                if(username != null && SecurityContextHolder.getContext().getAuthentication() == null ) {
                    UserDetails userDetails = userDetailService.loadUserByUsername(username);
                    if(userDetails != null) {
                        UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(userDet
                        SecurityContextHolder.getContext().setAuthentication(authenticationToken);
                    }
                }
            }

        } catch (Exception e) {
            log.info("token Invalid");
        }
        doFilter(request, response, filterChain);
    }
```

- Lấy Token: Phương thức này bắt đầu bằng việc lấy JWT từ yêu cầu HTTP thông qua phương thức getToken(request).
- Xác thực từ Token: Nếu token không null, phương thức tiếp tục kiểm tra và xác thực người dùng từ token. Đầu tiên, nó rút trích tên người dùng từ token bằng cách sử dụng jwt.extractUsername(token). Sau đó, kiểm tra xem tên người dùng có giá trị không và xem liệu có đối tượng xác thực hiện tại trong SecurityContextHolder không. Nếu có đối tượng xác thực hiện tại, có nghĩa là người dùng đã được xác thực trước đó, không cần thực hiện lại xác thực.
- Xác thực Người dùng: Nếu tên người dùng không null và không có đối tượng xác thực hiện tại, phương thức sẽ gọi userDetailService.loadUserByUsername(username) để lấy chi tiết người dùng. Sau đó, nó tạo một đối tượng

UsernamePasswordAuthenticationToken chứa thông tin người dùng và gán nó vào SecurityContextHolder để xác thực người dùng.

- Xử lý Ngoại lệ: Bất kỳ ngoại lệ nào xảy ra trong quá trình xử lý token đều được bắt và thông báo với thông điệp "token Invalid".
- Chuyển Giao Filter Tiếp Theo: Cuối cùng, phương thức gọi doFilter để chuyển giao quá trình xử lý sang Filter tiếp theo trong chuỗi Filter.

3.3 Xử lí ngoại lệ authentication

```
public class JwtAuthenticationEntrypoint  implements AuthenticationEntryPoint{

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response,
            AuthenticationException authException) throws IOException, ServletException {

        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        String error = "{\"UNAUTHORIZAED\":\"" + authException.getMessage() + "\"}";
        response.getWriter().write(error);
    }

}
```

- Xử Lý Bắt Đầu Xác Thực: Phương thức này được gọi khi xác thực không thành công, nghĩa là người dùng chưa được xác thực và yêu cầu không hợp lệ.
- Cài Đặt Trạng Thái Phản Hồi: Phương thức đặt trạng thái phản hồi của HTTP là SC_UNAUTHORIZED (mã trạng thái 401), thông báo rằng yêu cầu không được chấp nhận do thiếu xác thực.
- Tạo Thông Báo Lỗi: Tạo một thông báo lỗi JSON chứa thông báo "UNAUTHORIZED" và thông tin lỗi từ authException.getMessage().
- Viết Phản Hồi: Ghi thông báo lỗi vào phản hồi bằng cách sử dụng response.getWriter().write(error).

Tài liệu tham khảo

https://www.bezkoder.com/spring-boot-refresh-token-jwt/