08

Spring Security JWT인증

01. JWT 인증

Dependency 추가

```
implementation 'io.jsonwebtoken:jjwt-api:0.12.3'

implementation 'io.jsonwebtoken:jjwt-impl:0.12.3'

implementation 'io.jsonwebtoken:jjwt-jackson:0.12.3'
```

JWT 토큰 방식

- 세션이 없는 인증 방식 (Stateless)
- JWT 자체에 인증 정보를 포함하고 있어 서버가 별도로 세션을 저장하지 않음.
- 요청마다 JWT를 클라이언트가 헤더(보통 Authorization)에 포함해 전송.
- 서버는 이 토큰을 매번 검증(parsing & signature check)만 하면 됨

01. JWT 인증

- **JwtFilter**
  - 가장 앞단에 꽂을 필터로 요청시 전달되는 JWT토큰을 검증
- **JwtLoginFiler**
  - JwtFilter 바로 뒤에 꽂을 필터로 대부분의 인증과정을 포함
- **JwtUtil**
  - Jwt토큰을 생성하거나 Jwt토큰에서 인증정보 추출하는 기능을 담당

01. JWT 인증

SecurityConfig.java

```
http.csrf(csrf->csrf.disable())
        .formLogin(formLogin->formLogin.disable())
        .httpBasic(httpBasic->httpBasic.disable())
```

```
.cors(cors->cors.configurationSource(request -> {
    CorsConfiguration corsConfiguration = new CorsConfiguration();
    corsConfiguration.setAllowCredentials(true);
    corsConfiguration.addAllowedHeader("*");
   //클라이언트가 요청을 보낼때 보낼수 있는 헤더


    corsConfiguration.setExposedHeaders(List.of("Authorization"));
    //서버가 응답을 보낼때 브라우저가 접근할수 있는 헤더
    corsConfiguration.addAllowedMethod("*");
    corsConfiguration.addAllowedOrigin("http://localhost:3000");
    return corsConfiguration;
}))
```

01. JWT 인증

SecurityConfig.java

```
private final AuthenticationConfiguration authenticationConfiguration;
private final JwtUtil jwtUtil;

@Bean
public AuthenticationManager
authenticationManager(AuthenticationConfiguration configuration) throws
Exception {
    return configuration.getAuthenticationManager();
}
```

```
.sessionManagement(session->
        session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

.addFilterBefore(new JwtFilter(jwtUtil), JwtLoginFilter.class)

.addFilterAt(new
JwtLoginFilter(authenticationManager(authenticationConfiguration),
jwtUtil), UsernamePasswordAuthenticationFilter.class)
```

01. JWT 인증

frontend

```
apiClient.interceptors.request.use((config) => {
    // 1. 요청 데이터가 URLSearchParams 타입인지 확인
        if (config.data instanceof URLSearchParams) {
        config.headers["Content-Type"] = "application/x-www-form-urlencoded";
    }
    const jwtToken=store.getState().token.token;
    config.headers["authorization"]=jwtToken;

    return config; // 수정된 config 반환
}, (error) => {
    // 요청을 가로채는 중에 에러 발생 시 처리
        return Promise.reject(error);
});
```

02. Refresh 토큰

- JWT 토큰을 Access와 Refresh로 나눔
- Access 토큰
  - 요청을 보낼때 헤더에 담을 토큰으로 유효시간을 짧게 함
- Refresh 토큰
  - Access 토큰의 유효시간이 만료되면 다시 토큰을 발급받을 때 사용하는 토큰으로 쿠키에 담아 사용
  - 유효시간은 길게 잡음
- 사용자가 인증에 성공하면 서버에서 Access 토큰(Header)과 Refresh토큰(Cookie)을 함께 발급하여 전달

02. Refresh 토큰

JwtLoginFilter

```
@Override
public void successfulAuthentication(HttpServletRequest request,
HttpServletResponse response, FilterChain chain, Authentication authResult) throws
IOException, ServletException {
    …
    …
    String access_token = this.jwtUtil.createToken("access", username, role,
    5 * 1000L);
    String refresh_token = this.jwtUtil.createToken("refresh", username, role,
    60 * 60 * 24 * 1000L);
    response.addHeader("Authorization", "Bearer " + access_token);
    response.addCookie(this.createCookie("refresh", refresh_token));
    response.setCharacterEncoding("UTF-8");
    response.getWriter().write(jsonMessage);
}
```

02. Refresh 토큰

JwtLoginFilter

```
private Cookie createCookie(String key, String value) {
    Cookie cookie = new Cookie(key, value);
    cookie.setPath("/");
   // 루트 경로부터 시작되는 모든 요청에 대해 쿠키가 포함되도록
   //설정

    cookie.setHttpOnly(true);
   //클라이언트의 자바스크립트 코드에서 접근할 수 없음

    cookie.setMaxAge(60 * 60 * 24);
    return cookie;
}
```

02. Refresh 토큰

JwtFilter

```
try{
    this.jwtUtil.isExpired(token);
}catch(ExpiredJwtException e){
    response.getWriter().write("access token expired");
    response.setStatus(456);
    response.setCharacterEncoding("UTF-8");
    return;
}

String category = this.jwtUtil.getCategory(token);
if(!category.equals("access")){
    response.getWriter().write("invalid access token");
    response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
    response.setCharacterEncoding("UTF-8");
    return;
}
```

02. Refresh 토큰

ReissueController

```
@PostMapping(value = "/reissue")
public ResponseEntity<String> reissue(HttpServletRequest request, HttpServletResponse response) {
    String refreshToken= null;
    Cookie[] cookies = request.getCookies();
    for(Cookie cookie : cookies) {
        if(cookie.getName().equals("refresh")) {
            refreshToken = cookie.getValue();
            break;
        }
    }
    if(refreshToken == null) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("토큰 null");
    }
    try{
        jwtUtil.isExpired(refreshToken);
    }catch(ExpiredJwtException ex){
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("만료된  토큰");
    }
    String category=this.jwtUtil.getCategory(refreshToken);
    if(!category.equals("refresh")){
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("유효하지  않는  토큰");
    }
    String username=jwtUtil.getUserName(refreshToken);
    String role=jwtUtil.getRole(refreshToken);
    String newAccessToken=this.jwtUtil.createToken("access", username, role, 5000L);
    response.addHeader("Authorization", "Bearer "+newAccessToken);
    return ResponseEntity.status(HttpStatus.OK).body("토큰  발급  성공");
}
```

02. Refresh 토큰

frontend

```
apiClient.interceptors.response.use((response)=> response,
    async  (error)=>{
    const originalReguest = error.config;
    if(error.response && error.response.status ===456 && ! originalReguest._retry){
        originalReguest._retry = true;
            try{
                const response = await axios.post("http://localhost:8080/reissue", null , {
                    withCredentials:true,
                });

                const newAccess = response.headers['authorization'];
                store.dispatch(setToken(newAccess));
                console.log("만료된  요청  재시도");
                return apiClient(originalReguest);
            }catch(error){
                console.error('리프레시  토큰으로  재발급  실패:', error);
                // 재발급  실패  시  에러  전달 (로그아웃  처리  등  추가  가능)
                return Promise.reject(error);
            }
    }
    return Promise.reject(error); // 다른  에러는  그대로  반환
});
```