# Haomibo프로젝트

## 사용자 로그인, 인증, 로그아웃에 대한 설명

2019.10.17

백엔드 API 참고 : <http://39.97.175.200:8080/haomibo/swagger-ui.html>

### - 로그인

사용자는 이메일과 암호를 가지고 로그인을 진행한다

서버는 사용자의 정보를 자료기지에서 찾아본다.

이메일과 암호가 정확한 경우 사용자에 대한 간단한 정보와 JWT에 기초한 인증토크을 생성하여 응답내용으로 돌려준다.

등록된 사용자가 아니거나 암호가 틀린 경우 해당한 오유코드를 돌려준다.

@RequestMapping(value = **"/login"**, method = RequestMethod.***POST***)  
**public** Object login(  
 @RequestBody @Valid LoginRequestBody requestBody,  
 BindingResult bindingResult) {  
 **if** (bindingResult.hasErrors()) { //로그인요청파라메터유효성검사  
 **return new** CommonResponseBody(Constants.ResponseMessages.***INVALID\_PARAMETER***);  
 }  
  
 QSysUser qSysUser = QSysUser.***sysUser***;  
 Predicate predicate = qSysUser.**email**.eq(requestBody.getEmail());  
// 로그인 요청파라메터중 이메일을 선택하여 체계에 등록된 메일인가를 체크한다.  
 Optional<SysUser> optionalSysUser = **sysUserRepository**.findOne(predicate);   
  
 **if** (!optionalSysUser.isPresent()) { // 만일 체계에 등록되지 않은 이메일인경우  
 **return new** CommonResponseBody(Constants.ResponseMessages.***USER\_NOT\_FOUND***);

//등록되지 않은 사용자임을 나타내는 오유문자렬을 응답으로 전송한다.  
 }

//체계에 등록된 이메일인 경우  
 SysUser sysUser = optionalSysUser.get();  
  
 **if** (!sysUser.getPassword().equals(requestBody.getPassword())) {

//사용자의 암호가 일치하는가를 체크한다.  
 **return new** CommonResponseBody(Constants.ResponseMessages.***INVALID\_PASSWORD***);

//암호가 일치하지 않은 경우 사용자암호가 불일치하다는 오유문자렬을 응답으로 전송한다.  
 }  
  
//암호가 일치하는 경우 JWT 토큰을 생성한다. 토큰은 API에 접근할수 있는 여부를 제공하며 일정한 시간이 지나면 재발급받아야 한다. 재발급에 대한 사항은 refresh-token을 참고할 것.

// 토큰의 유효기간은 2시간이며 2시간전에 FrontEnd는 자동적으로 재발급받게 설계되여 있다.

String token = **jwtUtil**.generateTokenForSysUser(sysUser);  
  
 **return new** CommonResponseBody(  
 Constants.ResponseMessages.***OK***,  
 **new** LoginResponseBody(  
 **new** User(sysUser.getId(), sysUser.getName()),  
 **new** Token(token, **jwtUtil**.getExpirationDateFromToken(Utils.*removePrefixFromToken*(token)))  
 )  
 );  
}

### - 인증

로그인, 사용자등록을 제외한 임의의 요청은 인증을 거친다. 인증은 로그인시 발급된 토큰에 기준하여 진행한다. 요청 헤더에 “X-AUTH-TOKEN” 마당에 토큰정보를 넣어 API를 호출해야 한다.

서버는 요청헤더에서 토큰정보를 읽어 이 토큰이 tbl\_forbidden\_token표에 있는지 없는지를 검사한다.( tbl\_forbidden\_token표에 대해선 아래에서 설명)

요청에 함께 보내진 토큰이 표에 존재한다면 이 토큰은 이미 로그아웃한 사용자의 토큰이므로 INVALID\_TOEKN으로 간주되고 요청은 거절된다.

표에 존재하지 않는경우 토큰에 대한 expire date를 계산하여 이 토큰이 유효한가를 검사한다.

유효기간이 지난 토큰에 대하여 서버는 TOKEN\_EXPIRED오유로 응답한다.

유효한 경우 토큰의 내용에서 사용자 id 와 role 을 얻어 사용자의 유무성 검사를 진행하고 요청에 role을 부여한다.

Controller 의 메소드들은 role을 부여받은 요청들에만 해당 처리를 진행하여 응답결과를 준다.

// 모든 요청은 이 필터를 거쳐 콘트롤러에 전달되며 토큰이 없는 요청은 이 필터에 의해 무시되며 API에 접근할수 없게 된다.

@Override  
**protected void** doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) **throws** IOException, ServletException {  
  
 AtomicReference<String> tokenStatus = **new** AtomicReference<>(Constants.ResponseMessages.***OK***);  
  
 Utils.*getTokenString*(request.getHeader(Constants.***REQUEST\_HEADER\_AUTH\_TOKEN\_KEY***)).ifPresent(token -> { //API요청파켓으로부터 토큰을 추출한다.  
  
 tokenStatus.set(**jwtUtil**.getTokenStatus(token)); // 토큰의 상태 즉 체계가 허용하는 토큰인가를 체크한다.  
 **if** (!Constants.ResponseMessages.***OK***.equals(tokenStatus.get())) {  
 **return**;  
 }  
  
 **jwtUtil**.getIdFromToken(token).ifPresent(id -> {

//토큰이 유효한 경우 토큰으로부터 사용자아이디와 Role을 얻는다. 토큰은 생성시 사용자아이디와 Role과 함께 암호화된다.  
 **jwtUtil**.getRoleFromToken(token).ifPresent(role -> {  
  
 **if** (Constants.Roles.***SYS\_USER***.equals(role)) {  
 **sysUserRepository**.findById(id).ifPresent(sysUser -> {

//토큰으로부터 얻은 사용자의 아이디와 Role에 따라 API에 접근할수 있는 권한을 부여한다. 만약 권한이 없는경우 API요청은 무시되게 된다.   
 UsernamePasswordAuthenticationToken authentication =  
 **new** UsernamePasswordAuthenticationToken(sysUser, **null**, Arrays.*asList*(**new** SimpleGrantedAuthority(Constants.Roles.***SYS\_USER***)));  
 authentication.setDetails(**new** WebAuthenticationDetailsSource().buildDetails(request));  
  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 });  
 }  
  
 });  
 });  
 });

// 토큰이 유효한 경우 권한이 부여된 요청은 다음 공정인 콘트럴에 전달되며 콘트럴은 자기의 접근권한과 비교하여 요청을 수락하게 된다.

**if** (Constants.ResponseMessages.***OK***.equals(tokenStatus.get())) {  
 filterChain.doFilter(request, response);  
} **else** {  
 **try** {

// 토큰이 유효하지 않은 경우 요청은 오유로 인식하고 무시된다.  
 response.getWriter().write(Utils.*convertObjectToJson*(**new** CommonResponseBody(tokenStatus.get())));  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
}

// 토큰의 유효성을 체크한다.

**public** String getTokenStatus(String token) {  
  
 **try** {  
 getAllClaimsFromToken(token);  
 } **catch** (ExpiredJwtException e) {  
 **return** Constants.ResponseMessages.***TOKEN\_EXPIRED***; //토큰의 기간만료  
 } **catch** (Exception e) {  
 **return** Constants.ResponseMessages.***INVALID\_TOKEN***; // 유효하지 않은 토큰  
 }  
  
 **final** Date expiration = getExpirationDateFromToken(token);  
 **if** (expiration.before(**new** Date())) {  
 **return** Constants.ResponseMessages.***TOKEN\_EXPIRED***;  
 }  
  
 QForbiddenToken qForbiddenToken = QForbiddenToken.***forbiddenToken***; //로그아웃된 토큰  
 **boolean** isForbidden = **forbiddenTokenRepository**.count(qForbiddenToken.**token**.eq(token)) > 0;  
  
 **if** (isForbidden) {  
 **return** Constants.ResponseMessages.***INVALID\_TOKEN***;  
 }  
  
 **return** Constants.ResponseMessages.***OK***; // 정상토큰  
}

### - 로그아웃

로그아웃요청에 토큰을 넣어 보낸다.

서버는 토큰 정보를 tbl\_forbidden\_token표에 넣어 보관한다.

// 로그인한 사용자에 대하여 로그아웃 하기

@Secured({Constants.Roles.***SYS\_USER***})  
@RequestMapping(value = **"/logout"**, method = RequestMethod.***POST***)  
**public** Object logout(  
 @RequestHeader(value = Constants.***REQUEST\_HEADER\_AUTH\_TOKEN\_KEY***, defaultValue = **""**) String authToken) {  
  
 ForbiddenToken forbiddenToken = **new** ForbiddenToken(authToken, expirationTimestamp);

// 토큰을 로그아웃된 토큰테이블에 보관한다.  
 **forbiddenTokenRepository**.save(forbiddenToken);  
 **forbiddenTokenRepository**.flush();  
  
 **return new** CommonResponseBody(Constants.ResponseMessages.***OK***);  
}

tbl\_forbidden\_token은 로그아웃한 사용자의 토큰이 재리용되는것을 방지하기 위한 블랙리스트이다.이 표의 매 행은 로그아웃한 유저들의 토큰과 토큰의 유효기간을 보관하며 시간에 따라 표의 자료량이 늘어나는것을 방지하기 위해 서버는 하루에 한번씩 이 표를 정리한다.