**Spring security 동작 과정. 간단한 ver. JWT기반 인증 방식.**  
장점 : 인증을 유지하지 않아도 돼. 서버에 부담이 없다. 즉 세션을 전혀 사용하지 않는다.  
로그인 방식 : rest API(요청 바디에 json 형식으로 넣어서 전송) + post  
인증 처리 : JWT토큰을 생성. 응답 바디에 json형식으로 포함하여 전송  
인증 유지 방식 : 클라는 요청에 JWT를 authorization헤더에 담아 요청. 서버는 요청마다 해당 JWT를 파싱하고 검증 후 인증 정보를 요청 처리 중에만 임시적으로 저장하고 활용함. 요청이 끝나면 사라진다. 🡺 클라가 토큰(인증 정보) 유지.  
단점 : 필터나 다른 것들을 필히 커스터마이징 해야함.

jwt자체와 jwt기반 인증 동작에 대해서.  
쓰는 이유 : 서버 부담 안주려고. 분산 서버(MSA)에 유리. 특히 서버 확장에 유리.   
폼기반 인증/인가는 서버 세션 의존적이지만, jwt는 statless기반이므로 로그인 이후 상태 저장은 전적으로 클라에게 의존함.  
프론트에서 서버로 뜸뜸히 요청하는 경우 세션 타임 아웃된다.  
jwt는 위조 변조에 강하다. 서명 떄문에. 도청당해서 통째로 읽히는 거는 약하다.  
그래서 https + JWT를 많이 사용함.

JWT는 커스텀할 클래스와 새로 만들어야할 보조 클래스들이 많다. 아래는 정리 요약본  
새로 만들어야 하는거  
JwtProcessor – JWT 작업을 위한 helper클래스. 토큰 생성. 토큰에서 추출. 토큰 유효한지 검사.  
JWT방식(API기반)에 맞는 DTO들 정의. LoginDTO(로그인 할때 온 id,pw정보) AuthResultDTO(UserInfo와토큰정보를 묶은 response에 담아서 보낼) UserInfoDTO(인증성공시유저의정보를담을)  
JsonResponse – 로그인 결과를 필터에서 직접 JSON응답하기 위한 유틸 클래스. jasckson객체 이용하여 전송.  
JwtAuthenticationFilter - 모든 요청에 대해서 헤더에 토큰이 있는지 검사  
AuthenticationErrorFilter - 토큰 처리시 예외 발생을 처리하는 필터. 만기, 그외에  
커스텀해야하는거  
UsernamePasswordAuthenticationFilter=> JwtUsernamePasswordAuthenticationFilter  
AuthenticationSuccessHandler=>LoginSuccessHandler  
AuthenticationFailureHandler=>LoginFailureHandler  
AuthenticationEntryPoint=>CustomAuthenticationEntryPoint  
AccessDeniedHandler=>CustomAccessDeniedHandler  
수정 : security config, ???Controller

**Spring security의 JWT기반 동작 과정 deep ver. Refresh token없이 access token만 있는 버전.**

1. 로그인 정보를 json형식으로 요청 바디에 넣고 post함.
2. AuthenticationErrorFilter(커스텀)가 가장 먼저 맞이. 해당 필터가 다음 필터를 넘기면서 토큰 처리시 발생하는 예외를 try catch로 예외처리. 일반적인 인증/인가 에러말고 토큰 관련 에러시에만 해당 필터가 JSON형식으로 에러 응답을 보낸다.
3. JwtAuthenticationFilter(커스텀)가 오는 요청 마다 헤더에 토큰이 있는지 검사. 있다면 authentication정보를 JwtProcessor(커스텀)을 통해서 토큰에서 추출하고 AuthenticationManager에서 얻고(폼기반로그인처럼)(얻었다면 로그인돼있는 상태에서 요청) SecurityContextHolder에 임시저장. 요청을 처리하는 동안만 저장되므로 세션에 저장하는게 아님. 토큰이 있든 말든 다음 필터로.
4. 폼기반 UsernamePasswordAuthenticationFilter를 끄고 그것을 상속받아   
   특정 로그인 요청 url에서만 동작하게끔(로그인 요청시에만 동작함) 정의한  
   JwtUsernamePasswordAuthenticationFilter(커스텀)에서 로그인 요청 작업을 처리.  
   loginDTO(커스텀)로 로그인 요청 정보를 받고 해당 정보들로 토큰을 만들고   
   AuthenticationManager가 토큰을 통해 인증 처리를 하게함(폼기반 로그인처럼)  
   로그인 성공하면 authentication객체 얻음.
5. 로그인에 성공하면 AuthenticationSuccessHandler을 구현한 LoginSuccessHandler(커스텀)가 JwtProcessor(커스텀)를 이용하여 토큰을 만들고, 얻은 authentication객체를 통해 userdetails를 얻어 UserInfoDTO(커스텀)를 채우고, 토큰과 UserInfoDTO로 AuthResultDTO(커스텀)를 채워 이를 response의 바디에 채워서 응답한다. 응답받은 클라는 토큰을 저장하여 다음 요청때마다 토큰을 포함하여 보낸다.  
   클라에 저장될 토큰은 만료시간을 지정할 수 있다.  
   로그인에 실패하면 AuthenticationFailureHandler을 구현한 LoginFailureHandler(커스텀)가 에러를 JSON형식 응답으로 보낸다.
6. 기본적인 인증/인가 에러는 ExceptionTranslationFilter가 받아 처리한다. 기본적인 에러 핸들러는 JSON형식으로 응답하지 않기 때문에  
   CustomAuthenticationEntryPoint(커스텀)로 401에러는 json으로 응답보내고  
   CustomAccessDeniedHandler(커스텀)로 403에러는 json으로 응답보낸다.
7. 명시되어있지 않은 나머지 과정과 디테일들은 폼기반 인증/인가 과정과 같으므로 명시하지 않았다.

**Spring security의 JWT기반 동작 과정 구현 및 설명. Refresh token없이 access token만 있는 버전.**

**상황 :**

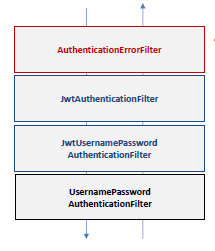
refresh token없어.  
요청 바디에 JSON형식으로 username과 password가 들어있고 post로 온 상황.  
최초 로그인 요청(토큰 발급 받고 얻는 과정) 혹은 인증된 후의 요청(토큰 유효 확인 후 진행)

**준비 세팅 : API 서버를 위한 기본 security 설정 + @**

Build.grade  
ext{ … springSecurityVersion=’5.8.13’ … }  
dependenccies{ …   
implementiation(“org.springframework.security:spring-security-web:${springSecurityVersion}”)//servlet용  
implementiation(“org.springframework.security:spring-security-config:${springSecurityVersion}”) )//servlet용  
implementiation(“org.springframework.security:spring-security-core:${springSecurityVersion}”) )//servlet용  
implementiation(“org.springframework.security:spring-security-taglibs:${springSecurityVersion}”)//jsp용  
implementation("io.jsonwebtoken:jjwt-api:0.11.5") //자바가 jwt를 이용하기 위해 필요한 라이브러리  
runtimeOnly("io.jsonwebtoken:jjwt-impl:0.11.5") //자바가 jwt를 이용하기 위해 필요한 라이브러리  
implementation("io.jsonwebtoken:jjwt-jackson:0.11.5") //JWT관련 자바객체<->json 해주는 라이브러리.  
 … }

Src>main>java>org.scoula>security>config>SecurityInitializer.java  
public class SecurityInitializer extends AbstractSecurityWebApplicationInitializer{ } //비어있어도 필수

Src>main>java>org.scoula>security>config>SecurityConfig.java  
@Configuration @EnableWebSecurity @Log4j2 @ComponentScan(basePackages={“쓸컴포넌트경로”})  
@MapperScan(basePAckages={“userdetails 갖고 올때 사용할 매퍼 경로”})  
@RequiredArgsConstructor  
public class SecurityConfig extends WebSecurityConfigurerAdapter{  
 //userdetails를 매퍼를 통해 가져올 서비스활용  
 private final UserDetailsService userDetailsService;  
 private final AuthenticationErrorFilter authenticationErrorFilter; //새로 만든 필터 주입받기  
 private final JwtAuthenticationFilter jwtAuthenticationFilter;//새로 만든 필터 주입받기  
 @Autowired//해당 타입은 생성자로 주입 안돼. 새로 커스텀한 필터 주입받기  
 private JwtUsernamePasswordAuthenticationFilter jwtUsernamePasswordAuthenticationFilter;  
 private final CustomAccessDeniedHandler accessDeniedHandler;  
 private final CustomAuthenticationEntryPoint authenticationEntryPoint;  
 //WebConfig에 등록된 문자셋 필터보다 security filter가 먼저 동작하기에  
 //security filter에도 문자셋 설정 필터를 정의. CsrfFilter보다 앞에 등록 필요.  
 public CharacterEncodingFilter encodingFilter( ){//문자셋 설정 필터  
 CharacterEncodingFilter encodingFilter = new CharacterEncodingFilter( );  
 encodingFilter.setEncoding(“UTF-8”);//한글깨짐 방지  
 encodingFilter.setForceEncoding(true);  
 return encodingFilter;  
 }  
 @Override  
 public void configure(HttpSecurity http//실질적인 설정 객체) throws Exception{   
 http.addFilterBefore(encodingFilter( ), CsrfFilter.class);//인코딩 필터 배치. 필터 순서지정.  
 //인증 에러 필터  
 .addFilterBefore(authenticationErrorFilter, UsernamePasswordAuthenticationFilter.class)  
 //jwt 인증 필터 앞에다 배치  
 .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class)  
 //커스텀한 필터를 기존 필터 앞에다 배치  
 .addFilterBefore(jwtUsernamePasswordAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);  
 http.httpBasic( ).disable( ) //기본 http인증 비활성화  
 .csrf( ).disable( ) //CSRF 비활성화  
 .formLogin( ).disable( ) //form기반 로그인 비활성화 == 관련 필터 해제  
 //UsernamePasswordAuthenticationFilter포함해서 해제됨 /login에 반응도 안하겠죠?  
 //security에서 세션에 상태정보를 저장하지 않게끔 지정.  
 .sessionManagement( ).sessionCreationPolicy(SessionCreationPolicy.STATELESS);//세션 생성모드 설정.  
 //토큰 예외 말고 인증/인가 관련 에러 처리하는 커스텀 핸들러들 등록.  
 http.exceptionHandling( ).authenticationEntryPoint(authenticationEntryPoint)  
 .accessDeniedHandler(accessDeniedHandler);  
 }  
 @Override protected void configure(AuthenticationManagerBuilder auth) throws Exception{  
 //인증 정보를 설정. 사용자 정보를 어디서 얻을지 설정. 암호화 설정.  
 auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder( ));  
 }  
 @Bean //암호화 동작객체를 어디서나 쓰려고 컨텍스트에 빈 등록  
 public PasswordEncoder passwordEncoder( ){  
 return new BcryptPasswordEncoder( ); //암호화하는 encode메서드와 맞는지 확인하는 matches메서드 지원.  
 }  
 @Bean//AuthenticationManager 빈등록  
 public AuthenticationManager authenticationManager( ) throws Exception{ return super.authenticationManager( ); }  
 @Bean//모든 오리진과 헤더에 대해서 허용하겠다. Cross origin 접근 허용.  
 public CorsFilter corsFilter( ){  
 UrlBasedCorsConfigurationSource source=new UrlBasedCorsConfigurationSource( );  
 CorsConfiguration config=new CorsConfiguration( );  
 config.setAllowCredentials(true); config.addAllowedOriginPattern("\*");  
 config.addAllowedHeader("\*"); config.addAllowedMethod("\*");  
 source.registerCorsConfiguration("/\*\*", config);  
 return new CorsFilter(source);   
 }  
 @Override //접근제한 무시경로 설정 == 등록한 경로는 security 관여하지 말라 지정.  
 public void configure(WebSecurity web) throws Exception{  
 web.ignoring( ).antMatchers(“/assets/\*\*”, “/\*”, “/api/member/\*\*”); //해당 예시에서는  
 //vue에서의 static경로, 루트레벨 경로, 회원가입 및 조회 페이지 경로.  
 }  
}



Src>main>java>org.scoula>security>util>JwtProcessor.java  
@Component //jwt작업을 위한 헬퍼클래스. 토큰생성, 토큰에서 추출, 토큰 유효성 검증.  
public class JwtProcessor{  
 static private final long TOKEN\_VALID\_MILISECOND = 1000L\*60\*5;//5분 토큰 유효 시간 지정  
 // 서버만 알고 있는 첨가물  
 private String secretKey=”충분히긴 임의의(랜덤한)비밀키문자열”; //개발시에만 직접 지정  
 private Key key=Keys.hmacShaKeyFor(secretKey.getBytes(StandardCharsets.UTF\_8));//개발시에만  
 // private Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256); -- 운영시 사용 }

//jwt생성  
 public String generateToken(String subject){  
 return Jwts.builder( ) //payload 정보 구성  
 .setSubject(subject) // 해당 예시는 유저네임만 토큰에 넣었고 다른 정보는 디비에 넣어서 보관함  
 .setIssuedAt(new Date( )) //정보 구성한 시간 저장  
 .setExpiration(new Date(new Date().getTime() + TOKEN\_VALID\_MILISECOND))// 만료시간 지정  
 .signWith(key) //서버만 알고 있는 첨가물 첨가.  
 .compact( );//토큰 문자열로 만들기  
 //토큰에 추가적인 정보 더 넣고 싶다면 Jwts.Builder( ).claims(키,밸류).claims(키,밸류)…해주면 된다.  
 //권한정보를 토큰 생성할 때 넣어줄 수 있으나 보안이 약해지고, 위 예시는 보안은 좋으나 권한 정보를  
 //얻기 위한 디비 연산 오버헤드가 생긴다.  
 }  
 //JWT subject == username 추출. 해석 불가인 경우 예외 발생(유효 만기, 형식이상, 시그니처, 인자 문제)  
 public String getUsername(String token){  
 return Jwts.parserBuilder( )  
 .setSigningKey(key)//토큰 생성할 때 썼던 키  
 .build( )  
 .parseClaimsJws(token) //해석할 토큰  
 .getBody( )  
 .getSubject( ); //유저네임 추출  
 }  
 //JWT 유효기간 검증. 해석 물가인 경우 예외 발생  
 public boolean validateToken(String token) {  
 Jws<Claims> claims = Jwts.parserBuilder( ) //파싱이 된다면 아직 유효하니까 넘어가고, 안되면 에러 발생됨.  
 .setSigningKey(key)  
 .build( )  
 .parseClaimsJws(token);  
 return true;  
 }  
}

Src>main>java>org.scoula>security>util>JsonResponse.java //응답 JSON형태로 내보내기위한 helpler 유틸클래스  
public class JsonResponse {  
 //올바른 JSON 응답을 보낼 경우  
 public static <T> void send(HttpServletResponse response, T result) throws IOException {  
 ObjectMapper om = new ObjectMapper( );//jackson 라이브러리 객체.  
 response.setContentType("application/json;charset=UTF-8"); //응답 헤더 세팅  
 Writer out = response.getWriter( );  
 out.write(om.writeValueAsString(result)); //java객체 🡺 JSON  
 out.flush( ); //보내버리기  
 }  
 //예외 JSON을 보내야하는 경우  
 public static void sendError(HttpServletResponse response, HttpStatus status, String message) throws IOException{  
 response.setStatus(status.value( )); //에러 상태  
 response.setContentType("application/json;charset=UTF-8"); //응답 헤더 세팅  
 Writer out = response.getWriter( );  
 out.write(message); //에러메세지 쓰고  
 out.flush( ); //보내버리기  
 }  
}

Src>main>java>org.scoula>security>filter>AuthenticationErrorFilter.java  
//토큰 관련 예외 처리 필터. 만기 예외인 경우 401. 토큰 관련예외 외로는 안다룸.  
@Component public class AuthenticationErrorFilter extends OncePerRequestFilter {//요청당 한번만 해당 필터 통과  
 @Override  
 protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)  
 throws ServletException, IOException {  
 //다음 필터 동작 명시  
 try { super.doFilter(request, response, filterChain); } catch (ExpiredJwtException e) {//401에러  
 JsonResponse.sendError(response, HttpStatus.UNAUTHORIZED, "토큰의 유효시간이 지났습니다.");  
 } catch (UnsupportedJwtException | MalformedJwtException | SignatureException e) {  
 JsonResponse.sendError(response, HttpStatus.UNAUTHORIZED, e.getMessage());  
 } catch (ServletException e) {  
 JsonResponse.sendError(response, HttpStatus.INTERNAL\_SERVER\_ERROR, e.getMessage());  
 }  
 }  
}

Src>main>java>org.scoula>security>filter>JwtAuthenticationFilter.java  
//모든 요청에 대해서 헤더에 토큰있는지 검사. 있으면 토큰을 추출. 없으면 클라가 로그인 안한 상태.  
//. 토큰 유효성 검사. 토큰 유효하면 그를 통해 SecurityContextHolder에 사용자 로그인 정보 설정.  
@Component @Log4j2 @RequiredArgsConstructor   
public class JwtAuthenticationFilter extends OncePerRequestFilter {   
 public static final String AUTHORIZATION\_HEADER = "Authorization";  
 public static final String BEARER\_PREFIX = "Bearer "; // 끝에 공백 있음  
 private final JwtProcessor jwtProcessor;  
 private final UserDetailsService userDetailsService;  
 private Authentication getAuthentication(String token) {//토큰을 통해 authentication얻기  
 String username = jwtProcessor.getUsername(token);  
 //서비스에서 정보 갖고오기  
 UserDetails princiapl=userDetailsService.loadUserByUsername(username);//usernotfound런타임 예외 발생가능  
 return new UsernamePasswordAuthenticationToken(principal, null, principal.getAuthorities( ));  
 }  
 @Override  
 protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)  
 throws ServletException, IOException {  
 String bearerToken = request.getHeader(AUTHORIZATION\_HEADER);//null이면 로그인 안한 사용자.  
 if (bearerToken != null && bearerToken.startsWith(BEARER\_PREFIX)) { //jwt 처리  
 String token = bearerToken.substring(BEARER\_PREFIX.length());  
 // 토큰에서 사용자 정보 추출 및 Authentication 객체 구성 후 SecurityContext에 저장  
 Authentication authentication = getAuthentication(token);//토큰을 통해 authentication얻고  
 SecurityContextHolder.getContext( ).setAuthentication(authentication); SCH에 얻은 authentication 등록  
 }  
 super.doFilter(request, response, filterChain);  
 }  
}

**데이터 준비 :**

Src>main>java>org.scoula>security>dto>LoginDTO.java //로그인할 때 온 id, pw정보  
@NoArgsConstructor @AllArgsConstructor @Data public class LoginDTO{  
 private String username; private String password; //spring security규약에 따라 저 둘의 멤버를 가져야함.  
 public static LoginDTO of(HttpServletRequest request){ //요청 바디의 JSON 🡺 LoginDTO에 담음  
 ObjectMapper om = new ObjectMapper( ); //jackson 라이브러리 객체  
 try{ return om.readValue(request.getInputStream(), LoginDTO.class); } //json🡺자바객체  
 catch(Exception e){ throw new BadCredentailsException(“username or password missing”); } //필수 프로퍼티X.  
 }  
}

Src>main>java>org.scoula>security>dto>UserInfoDTO.java //토큰에 안담기는 디비에 담길 유저 정보.  
@NoArgsConstructor @AllArgsConstructor @Data public class UserInfoDTO{  
 //로그인 성공시 응답에 포함시킬 사용자 정보들  
 String username; String email; List<String> roles;  
 //UserDetailsService에서 사용자 정보 디비에서 가져온 정보를 저장한 MemberVO 활용  
 public static UserInfoDTO of(MemberVO mem){ // MemberVO 🡺 UserInfoDTO  
 return new UserInfoDTO( mem.getUsername( ), mem.getEmail( ),  
 mem.getAuthList( ).stream( ).map(a->a.getAuth( )).toList( ) ); //roles멤버까지 채워  
 }  
}

Src>main>java>org.scoula>security>dto>AuthResultDTO.java //UserInfo와 토큰을 묶어 JSON response보낼때  
로그인 성공 결과를 나타내는 응답용 DTO.  
@NoArgsConstructor @AllArgsConstructor @Data public class AuthResultDTO{  
 String token; UserInfoDTO user;  
}

**커스터마이징 해야할 거 :**

Src>main>java>org.scoula>security>filter>JwtUsernamePasswordAuthenticationFilter.java  
약속된 로그인 url요청인 경우 로그인 절차 수행. AuthenticationManager 활용.  
로그인 결과 핸들러 사용. 기존 UsernamePasswordAuthentcationFilter(사용하지 않지만)앞에 위치.  
@Log4j2 @Component  
public class JwtUsernamePasswordAuthenticationFilter extends UsernamePasswordAuthenticationFilter {  
 // 스프링 생성자 주입을 통해 전달  
 public JwtUsernamePasswordAuthenticationFilter(  
 AuthenticationManager authenticationManager, // SecurityConfig가 생성된 이후에 등록됨  
 LoginSuccessHandler loginSuccessHandler, LoginFailureHandler loginFailureHandler) {  
 super(authenticationManager);  
 setFilterProcessesUrl("/api/auth/login"); // POST 로그인 요청 url  
 setAuthenticationSuccessHandler(loginSuccessHandler); // 로그인 성공 핸들러 등록  
 setAuthenticationFailureHandler(loginFailureHandler); // 로그인 실패 핸들러 등록  
 }  
 // 등록한 로그인 요청 URL인 경우 로그인 작업 처리. 요청에서 username, pw 추출  
 @Override   
 public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)  
 throws AuthenticationException {  
 // 요청 BODY의 JSON에서 username, password 🡺 LoginDTO  
 LoginDTO login = LoginDTO.of(request);  
 // 인증 토큰(UsernamePasswordAuthenticationToken) 구성  
 UsernamePasswordAuthenticationToken authenticationToken = //추출 정보로 id, pw로 토큰 생성  
 new UsernamePasswordAuthenticationToken(login.getUsername( ), login.getPassword( ));  
 // AuthenticationManager에게 인증 요청  
 return getAuthenticationManager( ).authenticate(authenticationToken); //인증결과 authentication 반환  
 }  
}

디폴트는 뷰 페이지 리다이렉트인데, 우리는 JSON형식으로 응답을 보내야하는 상황.  
Src>main>java>org.scoula>security>handler>LoginSuccessHandler.java  
@Log4j2 @Component @RequiredArgsConstructor //로그인 결과를 JSON으로 직접 응답  
public class LoginSuccessHandler implements AuthenticationSuccessHandler {  
 private final JwtProcessor jwtProcessor;  
 private AuthResultDTO makeAuthResult(CustomUser user) {//얻은 UserDetails가 들어올 예정  
 String username = user.getUsername( );  
 String token = jwtProcessor.generateToken(username); // 토큰 생성  
 // 토큰 + 사용자 기본 정보 (사용자명, ...)를 묶어서 AuthResultDTO 구성  
 return new AuthResultDTO(token, UserInfoDTO.of(user.getMember( )));   
 }  
 @Override  
 public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,   
 Authentication authentication) throws IOException, ServletException {  
 CustomUser user = (CustomUser) authentication.getPrincipal( ); // 인증 결과 Principal  
 AuthResultDTO result = makeAuthResult(user); // 인증 성공 결과를 JSON으로 직접 응답  
 JsonResponse.send(response, result); //토큰, 사용자 기본정보 json담아 보내  
 }  
}

Src>main>java>org.scoula>security>handler>LoginFailureHandler.java  
@Component public class LoginFailureHandler implements AuthenticationFailureHandler {  
 @Override public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response,   
 AuthenticationException exception) throws IOException, ServletException {  
 JsonResponse.sendError(response, HttpStatus.UNAUTHORIZED,  
 "사용자 ID 또는 비밀번호가 일치하지 않습니다.");  
 }  
}

인증/인가 에러에 대한 처리를 JSON 응답으로 변경해야함(디폴트는 에러 페이지). 401에러/403에러.  
Src>main>java>org.scoula>security>handler>CustomAuthenticationEntryPoint.java  
@Log4j2 @Component //401에러. 인증 필요한 페이지인데 인증 안된 채로 접근하려 할때  
public class CustomAuthenticationEntryPoint implements AuthenticationEntryPoint {  
 @Override  
 public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException  
 authException) throws IOException, ServletException {  
 log.error("========== 인증 에러 ============");  
 JsonResponse.sendError(response, HttpStatus.UNAUTHORIZED, authException.getMessage( ));//401에러  
 }  
}

Src>main>java>org.scoula>security>handler>CustomAccessDeniedHandler.java//403에러. 권한에러  
@Component @Log4j2 public class CustomAccessDeniedHandler implements AccessDeniedHandler {  
 @Override public void handle(HttpServletRequest request, HttpServletResponse response,   
 AccessDeniedException accessDeniedException) throws IOException, ServletException {  
 log.error("========== 인가 에러 ============");  
 JsonResponse.sendError(response, HttpStatus.FORBIDDEN, "권한이 부족합니다.");  
 }  
}

**활용 : ??controller에서**

jwt기반 인증/인가 과정은 json을 주고 받는다 🡺 rest api에 맞는 컨트롤러를 정의해야 된다.  
요청이 컨트롤러에 다다랐을 때는 이미 security filter chain을 통과하고 여느 다른 filter들을 통과한 다음이다.  
Src>main>java>org.scoula>handler>??Controller.java  
@Log4j2 @RestController @RequestMapping("/api/security") //해당 컨트롤러를 동작하게하는 url 지정.  
public class ??Controller { //url 지정시 json rest용 컨트롤러에는 앞에 /api를 붙이는게 관용  
 @GetMapping("/all")   
 public ResponseEntity<String> doAll( ) {  
 log.info("do all can access everybody");  
 return ResponseEntity.ok("All can access everybody"); //json형식으로 응답  
 }  
 @GetMapping("/member") //이미 authentication은 security filter chain을 통과하며 확보되어있음  
 public ResponseEntity<String> doMember(Authentication authentication) {//authentication 주입됨  
 UserDetails userDetails = (UserDetails)authentication.getPrincipal( );  
 log.info("username = " + userDetails.getUsername( ));  
 return ResponseEntity.ok(userDetails.getUsername( ));  
 }  
 @GetMapping("/admin")  
 public ResponseEntity<MemberVO> doAdmin(@AuthenticationPrincipal CustomUser customUser) {  
 MemberVO member = customUser.getMember();  
 log.info("username = " + member);  
 return ResponseEntity.ok(member);  
 }  
}

**Spring security의 JWT기반 동작 과정 deep ver. Refresh token 있는 버전.**

PAYPER 프로젝트에 구현되어 있다. 설명하도록 하겠다.