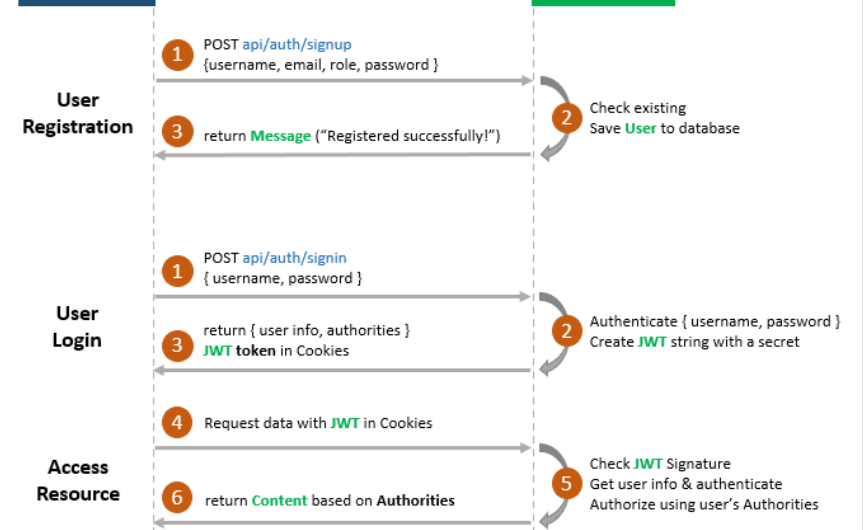
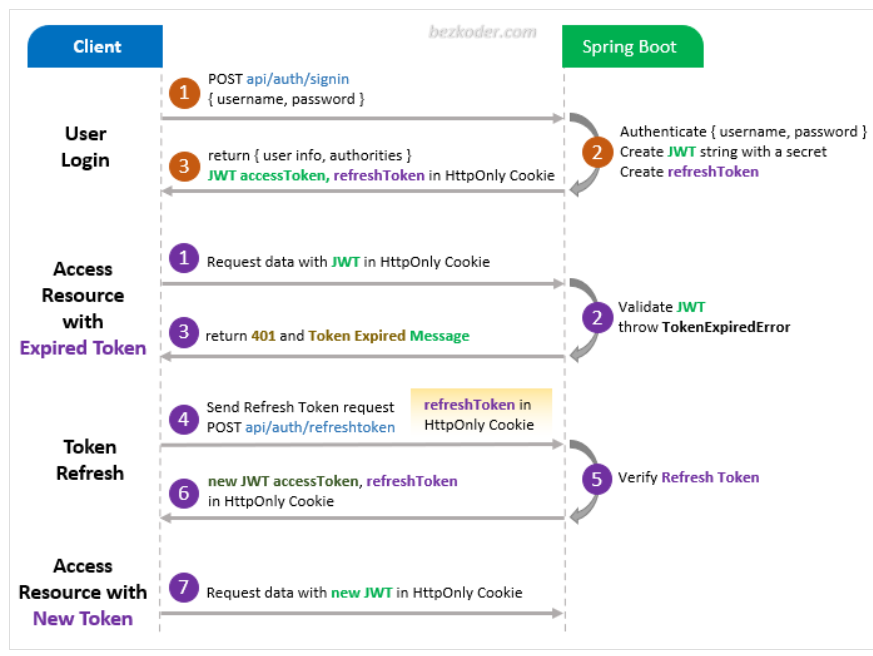
**More info**: <https://www.bezkoder.com/spring-boot-security-login-jwt/>

https://www.bezkoder.com/spring-security-refresh-token/

**The flow of Spring Boot Security using refresh/access token:**



## Flow for Refresh Token with JWT



* **Register / Login user:**

Controller / Post / AuthenticationResponse{username, password}

@ApiOperation("Login with admin user credentials")  
@PostMapping(value = "/login", consumes = MediaType.*APPLICATION\_JSON\_VALUE*)  
public ResponseEntity<AuthenticationResponse> login(@RequestBody AuthenticationRequest request) {  
 return ResponseEntity.*ok*(authenticationService.authenticateUser(request, Role.*ROLE\_ADMIN*));

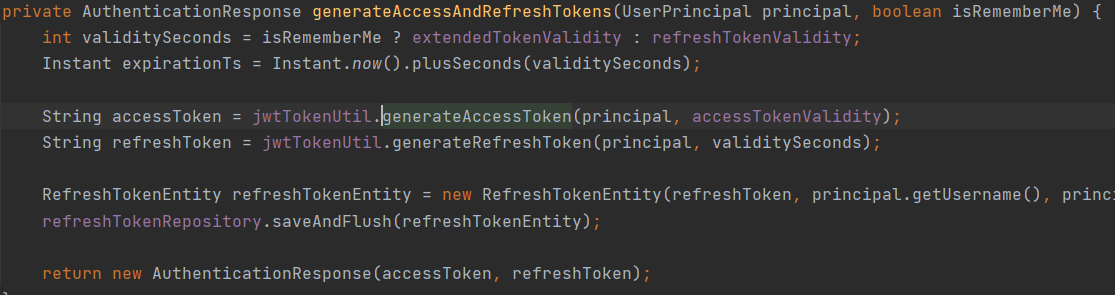
AuthenticationService| authenticateUser(): *Authenticate user based on username and password from request. If the credentials are correct access and refresh tokens are generated and returned as a result.*

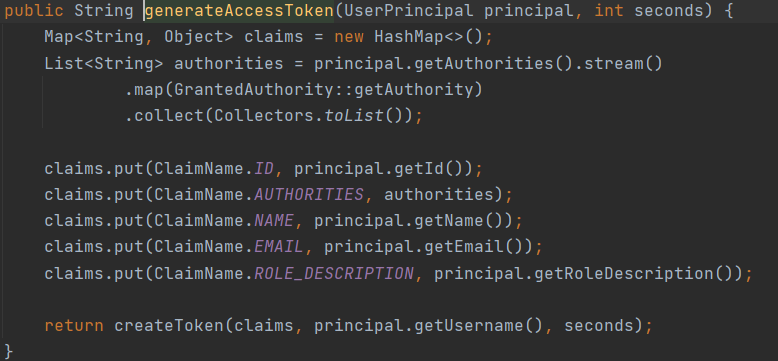
@Override  
public AuthenticationResponse authenticateUser(AuthenticationRequest request, Role role) {  
 String username = request.getUsername().trim();  
 UserPrincipal principal;  
 try {  
 authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, request.getPassword()));

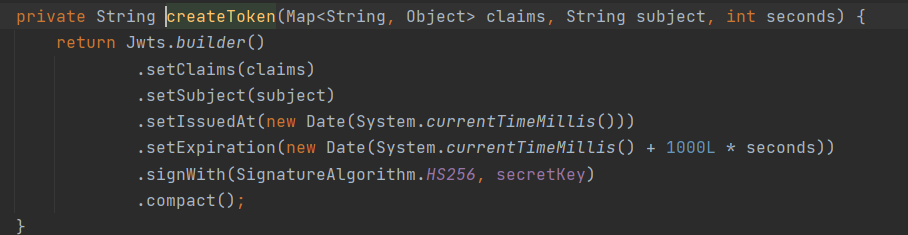
principal = (UserPrincipal) userDetailsService.loadUserByUsername(username);  
 } catch (BadCredentialsException e) {  
 String messageKey = "authentication." + (Role.*ROLE\_ADMIN*.equals(role) ? "admin" : "app") + ".invalid.credentials";  
 throw new BadCredentialsException(*getMessageByKey*(messageSource, messageKey));  
 } catch (DisabledException e) {  
 throw new DisabledException(*getMessageByKey*(messageSource, "authentication.disabled.user"));  
 }  
  
 if (Role.*ROLE\_ADMIN*.equals(role)) {  
 auditService.logEvent(AuditEventSource.*ADMINISTRATION*, AuditEventType.*ADMIN\_USER\_LOGIN*, null, principal, AdministrationModule.*DASHBOARD*);  
 } else {  
 auditService.logEvent(AuditEventSource.*APPLICATION*, AuditEventType.*APP\_USER\_LOGIN*, null, principal);  
 }  
  
 return generateAccessAndRefreshTokens(principal, request.isRememberMe());  
}

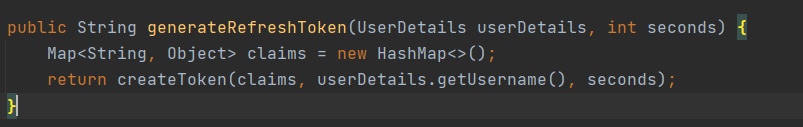
authenticationManager.authenticate- Attempts to authenticate the user based on the provided credentials. If the authentication is successful, the method returns an **Authentication** object. If the authentication fails then catch the specific exception.

userDetailsService.loadUserByUsername- We override LoadUserByUsername() method in our custom UserDetailServiceImpl which implement UserDetailService.

generateAccessAndRefreshTokens(principal, request.isRememberMe(): Base logic for creating tokens:

jwtTokenUtil.generateAccessToken: Set all necessary claims 

createToken Init token details **SIGN** WITH provided **SECRET KEY** 

jwtTokenUtil.generateRefreshToken: Simular to access token but without claims and different expiration 

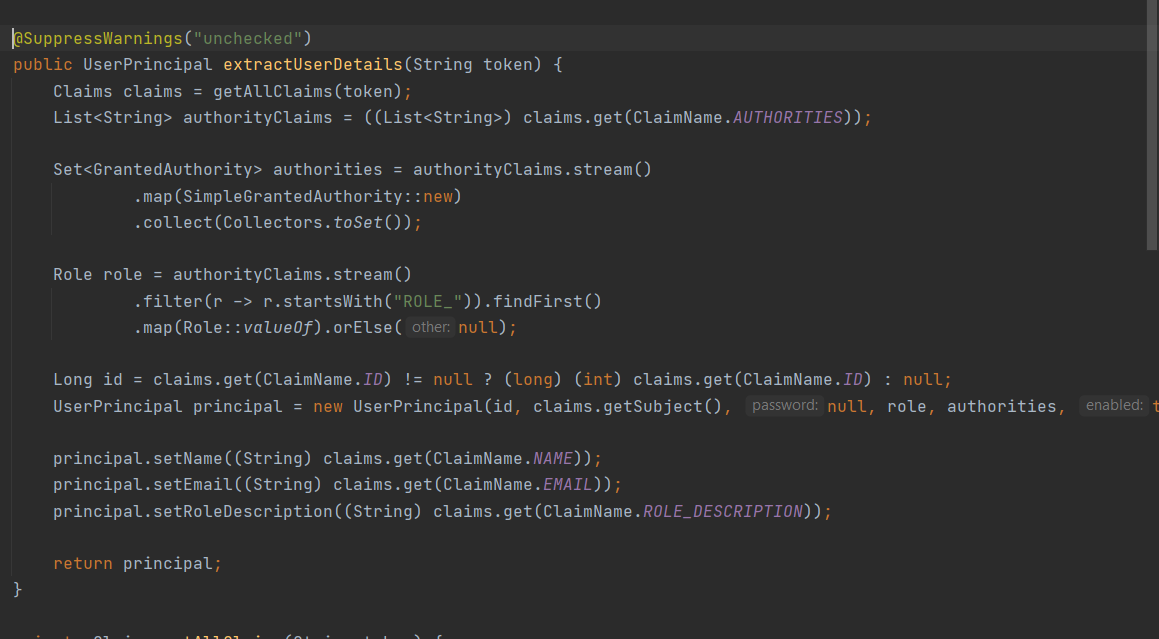
* **Аccess Resource:**
* @Component  
  public class JwtTokenFilter extends OncePerRequestFilter

Need to extends OncePerRequestFilter and override method doFilterInternal

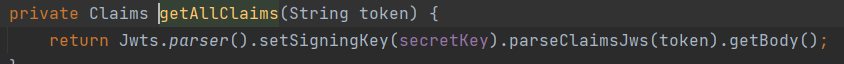
@Override  
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {  
  
 final String authorizationHeader = request.getHeader(*AUTHORIZATION*);  
  
 if (authorizationHeader != null && authorizationHeader.startsWith(*BEARER*)) {  
 String jwt = authorizationHeader.substring(*BEARER*.length());  
 UserPrincipal userDetails;  
 try {  
 userDetails = jwtTokenUtil.extractUserDetails(jwt);  
 } catch (ExpiredJwtException e) {  
 response.sendError(HttpServletResponse.*SC\_UNAUTHORIZED*, *getMessageByKey*(messageSource, "authentication.expired.token"));  
 return;  
 } catch (RuntimeException e) {  
 logger.error(e.getMessage());  
 response.sendError(HttpServletResponse.*SC\_UNAUTHORIZED*, *getMessageByKey*(messageSource, "authentication.unauthorized.user"));  
 return;  
 }  
  
 if (SecurityContextHolder.*getContext*().getAuthentication() == null && userDetails.getAuthorities() != null) {  
 UsernamePasswordAuthenticationToken token = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());  
  
 token.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));  
 SecurityContextHolder.*getContext*().setAuthentication(token);  
 }  
 } else {  
 response.sendError(HttpServletResponse.*SC\_UNAUTHORIZED*, *getMessageByKey*(messageSource, "authentication.unauthorized.user"));  
 return;  
 }  
  
 filterChain.doFilter(request, response);  
}

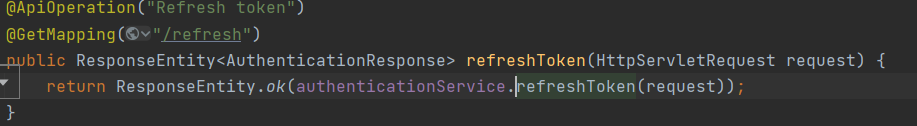
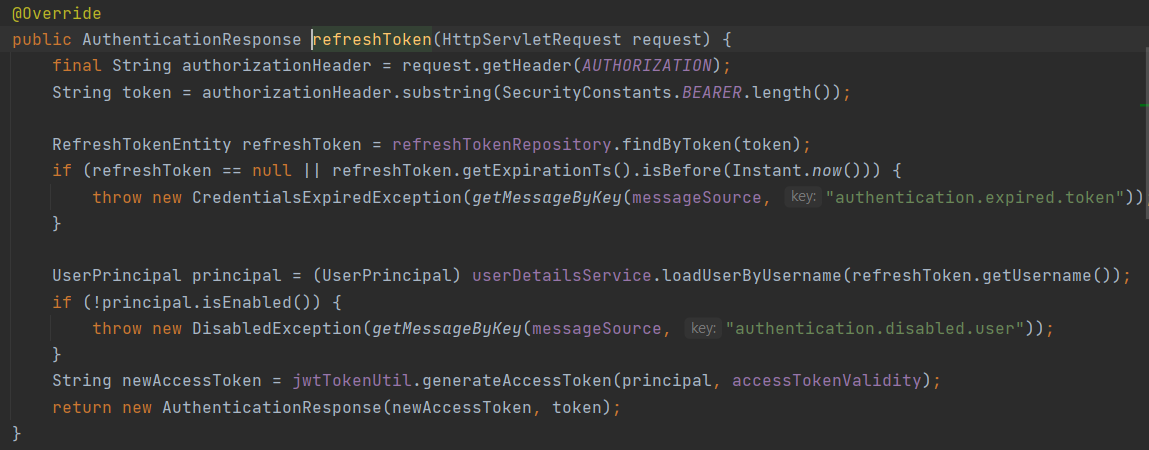
One of **the core logic** in this method:

**jwtTokenUtil**. **extractUserDetails**



In **getAllClaims** method we sign the token with our Secret Key



* **Refresh Token:** Generate a new **access** token for the user found by the **refresh** token. The **refresh token is kept the same** to satisfy product requirements.
* Conroller:Service: 

Refresh token usage - Kога ФЕ викат /refresh за да създане нов access токен? : като получат 401 от access token-a пробват да извикат рефреш, ако не стане чистят и 2та тоукъна от local storage-a. и чак като изтече и рефреш-а се трият тоукъните.. в някои системи се пазят в базата и 2та тоукъна така могат да инвалидират access token-a на някои потребител при нужда примерно сменили са му правата. => https://www.bezkoder.com/spring-security-refresh-token/