# **Authentication and Authorization using JWT + Spring security + Spring Boot**

1. Create a class for security which extends WebSecurityConfigurerAdapter and annotate the class using @EnableWebSecurity
2. Override configure method of WebSecurityConfigurerAdapter.

/\*

\* method where we can define which resources are public and which are secured

\*/

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.cors().and().csrf().disable().authorizeRequests()

.antMatchers(HttpMethod.***POST***, SecurityConstants.***SIGN\_UP\_URL***).permitAll()

.anyRequest().authenticated()

.and()

.addFilter(**new** JWTAuthenticationFilter(authenticationManager()))

.addFilter(**new** JWTAuthorizationFilter(authenticationManager()))

// this disables session creation on Spring Security

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.***STATELESS***);

}

===========================================================================

/\*

\* method where we defined a custom implementation of UserDetailsService to load user-specific data in the security framework

\*/

@Override

**public** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(bCryptPasswordEncoder);

}

1. **Create a Filter class for JWT Authentication**

**public** **class** JWTAuthenticationFilter **extends** UsernamePasswordAuthenticationFilter {

**private** AuthenticationManager authenticationManager;

**public** JWTAuthenticationFilter(AuthenticationManager authenticationManager) {

**this**.authenticationManager = authenticationManager;

}

==================================================================================

/\*

\* where we parse the user's credentials and issue them to the AuthenticationManager

\*/

@Override

**public** Authentication attemptAuthentication(HttpServletRequest req, HttpServletResponse res) **throws** AuthenticationException {

**try** {

AppUser creds = **new** ObjectMapper().readValue(req.getInputStream(), AppUser.**class**);

**return** authenticationManager.authenticate(**new** UsernamePasswordAuthenticationToken(creds.getUsername(),creds.getPassword(), **new** ArrayList<>()));

} **catch** (IOException e) {

**throw** **new** RuntimeException(e);

}

}

=================================================================================

/\*

\* method called when a user successfully logs in. This method is used to generate a JWT for this user

\*/

@Override

**protected** **void** successfulAuthentication(HttpServletRequest req, HttpServletResponse res, FilterChain chain,Authentication auth) **throws** IOException, ServletException {

String token = Jwts.*builder*().setSubject(((User) auth.getPrincipal()).getUsername())

.setExpiration(**new** Date(System.*currentTimeMillis*() + SecurityConstants.***EXPIRATION\_TIME***))

.signWith(SignatureAlgorithm.***HS512***, SecurityConstants.***SECRET***.getBytes()).compact();

res.addHeader(SecurityConstants.***HEADER\_STRING***, SecurityConstants.***TOKEN\_PREFIX*** + token);

}

}

1. **Create a Filter class for JWT Authorization**

**public** **class** JWTAuthorizationFilter **extends** BasicAuthenticationFilter {

**public** JWTAuthorizationFilter(AuthenticationManager authenticationManager) {

**super**(authenticationManager);

}

=====================================================================================

@Override

**protected** **void** doFilterInternal(HttpServletRequest req, HttpServletResponse res, FilterChain chain)

**throws** IOException, ServletException {

String header = req.getHeader(SecurityConstants.***HEADER\_STRING***);

**if** (header == **null** || !header.startsWith(SecurityConstants.***TOKEN\_PREFIX***)) {

chain.doFilter(req, res);

**return**;

}

UsernamePasswordAuthenticationToken authentication = getAuthentication(req);

SecurityContextHolder.*getContext*().setAuthentication(authentication);

chain.doFilter(req, res);

}

==================================================================================

/\*\*

\* **@param** request

\* This method reads the JWT from the Authorization header, and then uses Jwts

\* to validate the token. If everything is in place, we set the user in the SecurityContext

\* and allow the request to move on.

\*/

**private** UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {

String token = request.getHeader(SecurityConstants.***HEADER\_STRING***);

**if** (token != **null**) {

// parse the token.

String user = Jwts.*parser*().setSigningKey(SecurityConstants.***SECRET***.getBytes())

.parseClaimsJws(token.replace(SecurityConstants.***TOKEN\_PREFIX***, "")).getBody().getSubject();

**if** (user != **null**) {

**return** **new** UsernamePasswordAuthenticationToken(user, **null**, **new** ArrayList<>());

}

**return** **null**;

}

**return** **null**;

}

}

1. **Add a class for static value**

**public** **class** SecurityConstants {

**public** **static** **final** String ***SECRET*** = "SecretKeyToGenJWTs";

**public** **static** **final** **long** ***EXPIRATION\_TIME*** = 864\_000\_000; // 10 days

**public** **static** **final** String ***TOKEN\_PREFIX*** = "Bearer ";

**public** **static** **final** String ***HEADER\_STRING*** = "Authorization";

**public** **static** **final** String ***SIGN\_UP\_URL*** = "/users/sign-up";

}