**JWT Authentication with Spring Boot 3.0**



* We will see how to configure InMemory user and jwt authentication using latest spring boot 3.0.
* We will create one protected endpoint and try to secure endpoint using spring boot security.

**Create new Spring Boot Project**

* Go to spring initializer and create new project with dependencies
* add the following dependencies
* For Web

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

Copy

* For security

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

Copy

* Lombok

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

Copy

* For JWT

<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api -->

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-api</artifactId>

<version>0.11.5</version>

</dependency>

<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-impl -->

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId> <!-- or jjwt-gson if Gson is preferred -->

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

Copy

**Create End Point to be secured**

@RestController

public class HomeController {

Logger logger = LoggerFactory.getLogger(HomeController.class);

@RequestMapping("/test")

public String test() {

this.logger.warn("This is working message");

return "Testing message";

}

}

Copy

**Use can create the same that we developed in video.**

**Create InMemory user with UserDetailService Bean**

Create UserDetailService bean and write the InMemory user implementation

Create CustomConfig class and create bean and also create two important bean PasswordEncoder and AuthenticationManager so that we can use later.

@Configuration

class MyConfig {

@Bean

public UserDetailsService userDetailsService() {

UserDetails userDetails = User.builder().

username("DURGESH")

.password(passwordEncoder().encode("DURGESH")).roles("ADMIN").

build();

return new InMemoryUserDetailsManager(userDetails);

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

public AuthenticationManager authenticationManager(AuthenticationConfiguration builder) throws Exception {

return builder.getAuthenticationManager();

}

}

Copy

Now we can login with given username and password by default spring security provide form login .

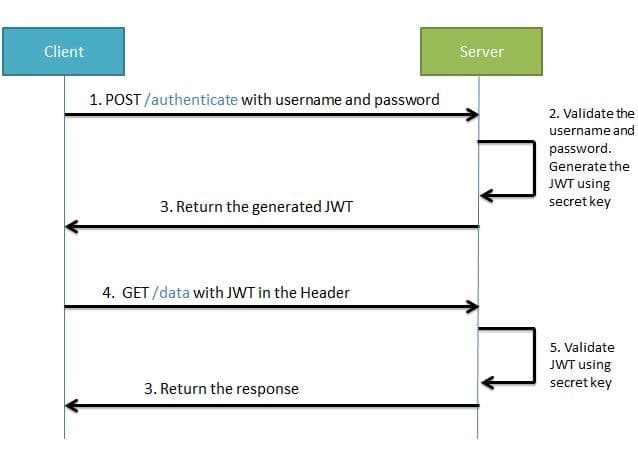
open browser and open

http://localhost:8080/test

Copy

when login form is prompted just login with username and password as given .

**JWT Authentication Flow**



**Steps to implement jwt token:**

**1)** Make sure spring-boot-starter-security is there in pom.xml

**2)**Create Class JWTAthenticationEntryPoint that implement AuthenticationEntryPoint. Method of this class is called whenever as exception is thrown due to unauthenticated user trying to access the resource that required authentication.

@Component

public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {

@Override

public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException) throws IOException, ServletException {

response.setStatus(HttpServletResponse.SC\_UNAUTHORIZED);

PrintWriter writer = response.getWriter();

writer.println("Access Denied !! " + authException.getMessage());

}

}

Copy

**3)**  Create **JWTHelper**  class This class contains method related to perform operations with jwt token like generateToken, validateToken etc.

@Component

public class JwtHelper {

//requirement :

public static final long JWT\_TOKEN\_VALIDITY = 5 \* 60 \* 60;

// public static final long JWT\_TOKEN\_VALIDITY = 60;

private String secret = "afafasfafafasfasfasfafacasdasfasxASFACASDFACASDFASFASFDAFASFASDAADSCSDFADCVSGCFVADXCcadwavfsfarvf";

//retrieve username from jwt token

public String getUsernameFromToken(String token) {

return getClaimFromToken(token, Claims::getSubject);

}

//retrieve expiration date from jwt token

public Date getExpirationDateFromToken(String token) {

return getClaimFromToken(token, Claims::getExpiration);

}

public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {

final Claims claims = getAllClaimsFromToken(token);

return claimsResolver.apply(claims);

}

//for retrieveing any information from token we will need the secret key

private Claims getAllClaimsFromToken(String token) {

return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();

}

//check if the token has expired

private Boolean isTokenExpired(String token) {

final Date expiration = getExpirationDateFromToken(token);

return expiration.before(new Date());

}

//generate token for user

public String generateToken(UserDetails userDetails) {

Map<String, Object> claims = new HashMap<>();

return doGenerateToken(claims, userDetails.getUsername());

}

//while creating the token -

//1. Define claims of the token, like Issuer, Expiration, Subject, and the ID

//2. Sign the JWT using the HS512 algorithm and secret key.

//3. According to JWS Compact Serialization(https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-41#section-3.1)

// compaction of the JWT to a URL-safe string

private String doGenerateToken(Map<String, Object> claims, String subject) {

return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + JWT\_TOKEN\_VALIDITY \* 1000))

.signWith(SignatureAlgorithm.HS512, secret).compact();

}

//validate token

public Boolean validateToken(String token, UserDetails userDetails) {

final String username = getUsernameFromToken(token);

return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));

}

}

Copy

**4)**Create JWTAuthenticationFilter that extends OncePerRequestFilter and override method and write the logic to check the token that is comming in header. We have to write 5 important logic

* 1. Get Token from request
  2. Validate Token
  3. GetUsername from token
  4. Load user associated with this token
  5. set authentication

@Component

public class JwtAuthenticationFilter extends OncePerRequestFilter {

private Logger logger = LoggerFactory.getLogger(OncePerRequestFilter.class);

@Autowired

private JwtHelper jwtHelper;

@Autowired

private UserDetailsService userDetailsService;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {

// try {

// Thread.sleep(500);

// } catch (InterruptedException e) {

// throw new RuntimeException(e);

// }

//Authorization

String requestHeader = request.getHeader("Authorization");

//Bearer 2352345235sdfrsfgsdfsdf

logger.info(" Header : {}", requestHeader);

String username = null;

String token = null;

if (requestHeader != null && requestHeader.startsWith("Bearer")) {

//looking good

token = requestHeader.substring(7);

try {

username = this.jwtHelper.getUsernameFromToken(token);

} catch (IllegalArgumentException e) {

logger.info("Illegal Argument while fetching the username !!");

e.printStackTrace();

} catch (ExpiredJwtException e) {

logger.info("Given jwt token is expired !!");

e.printStackTrace();

} catch (MalformedJwtException e) {

logger.info("Some changed has done in token !! Invalid Token");

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

}

} else {

logger.info("Invalid Header Value !! ");

}

//

if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

//fetch user detail from username

UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

Boolean validateToken = this.jwtHelper.validateToken(token, userDetails);

if (validateToken) {

//set the authentication

UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authentication);

} else {

logger.info("Validation fails !!");

}

}

filterChain.doFilter(request, response);

}

}

Copy

**5)** Configure spring security in configuration file:

@Configuration

public class SecurityConfig {

@Autowired

private JwtAuthenticationEntryPoint point;

@Autowired

private JwtAuthenticationFilter filter;

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http.csrf(csrf -> csrf.disable())

.authorizeRequests().

requestMatchers("/test").authenticated().requestMatchers("/auth/login").permitAll()

.anyRequest()

.authenticated()

.and().exceptionHandling(ex -> ex.authenticationEntryPoint(point))

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

http.addFilterBefore(filter, UsernamePasswordAuthenticationFilter.class);

return http.build();

}

}

Copy

**6)** Create JWTRequest and JWTResponse to receive request data and send Login success response.

**7)**  Create login api to accept username and password and return token if username and password is correct.

@RestController

@RequestMapping("/auth")

public class AuthController {

@Autowired

private UserDetailsService userDetailsService;

@Autowired

private AuthenticationManager manager;

@Autowired

private JwtHelper helper;

private Logger logger = LoggerFactory.getLogger(AuthController.class);

@PostMapping("/login")

public ResponseEntity<JwtResponse> login(@RequestBody JwtRequest request) {

this.doAuthenticate(request.getEmail(), request.getPassword());

UserDetails userDetails = userDetailsService.loadUserByUsername(request.getEmail());

String token = this.helper.generateToken(userDetails);

JwtResponse response = JwtResponse.builder()

.jwtToken(token)

.username(userDetails.getUsername()).build();

return new ResponseEntity<>(response, HttpStatus.OK);

}

private void doAuthenticate(String email, String password) {

UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(email, password);

try {

manager.authenticate(authentication);

} catch (BadCredentialsException e) {

throw new BadCredentialsException(" Invalid Username or Password !!");

}

}

@ExceptionHandler(BadCredentialsException.class)

public String exceptionHandler() {

return "Credentials Invalid !!";

}

}