**Json Web Token (Jwt)**

step 1 : make a fresh spring boot appliaction using spring initializer

step 2 : add dependency for 1)web 2)security 3)lombok

step 3 : import the project in eclipse

step 4: make 2 classes

a) SecurityConfigurer class extending WebSecurityConfigurerAdapter &

b) make a class by the name MyUserDetailsService implements UserDetailsService

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfigurer **extends** ~~WebSecurityConfigurerAdapter~~ {

@Autowired

MyUserDetailsService myUserDetailsService ;

@Override

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

auth.userDetailsService(myUserDetailsService);

}

@Bean

**public** PasswordEncoder getPasswordEncoder()

{

**return** ~~NoOpPasswordEncoder~~.~~getInstance~~();

}

}

@Service

**public** **class** MyUserDetailsService **implements** UserDetailsService{

@Override

**public** UserDetails loadUserByUsername(String userName) **throws** UsernameNotFoundException {

**return** **new** User("alice","alice123", **new** ArrayList<>());

}

}

NOTE : try to run the application and hit the /welcome endpoint passing username password as alice alice

step 5 : add the folowing dependencies (hold on)

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

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/javax.xml.bind/jaxb-api -->

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.3.0</version>

</dependency>

step 6 : make a package utils and make a user defined class JwtUtils as follows

**package** com.enumtech.jwtoken.util;

**import** java.util.Date;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.function.Function;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.stereotype.Service;

**import** io.jsonwebtoken.Claims;

**import** io.jsonwebtoken.Jwts;

**import** io.jsonwebtoken.SignatureAlgorithm;

@Service

**public** **class** JwtUtil {

**private** String SECRET\_KEY = "viratkohli";

**public** String extractUsername(String token) {

**return** extractClaim(token, Claims::getSubject);

}

**public** Date extractExpiration(String token) {

**return** extractClaim(token, Claims::getExpiration);

}

//2nd imp method will take a arg of type claimsResolver + this method is invoked by above 2 functions

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

**final** Claims claims = extractAllClaims(token);

**return** claimsResolver.apply(claims);

}

**private** Claims extractAllClaims(String token) {

**return** Jwts.*parser*().setSigningKey(SECRET\_KEY).parseClaimsJws(token).getBody();

}

**private** Boolean isTokenExpired(String token) {

**return** extractExpiration(token).before(**new** Date());

}

//1st imp method will take a userDetails object and create a Jwt Token

**public** String generateToken(UserDetails userDetails) {

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

**return** createToken(claims, userDetails.getUsername());

}

//Subject is the Principal i.e logged in user ("Alice")

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

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

.setExpiration(**new** Date(System.*currentTimeMillis*() + 1000 \* 60 \* 60 \* 10))

.signWith(SignatureAlgorithm.***HS256***, SECRET\_KEY).compact();

}

**public** Boolean validateToken(String token, UserDetails userDetails) {

**final** String username = extractUsername(token);

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

}

}

step 7 : create an API which will accept the username & password , authenticate the user and return the jwt

for this API create the request & response class in model package

**package** com.enumtech.jwtoken.model;

**public** **class** AuthenticationRequest {

**private** String username;

**private** String password;

**public** AuthenticationRequest() {}

**public** AuthenticationRequest(String username, String password) {

**this**.username = username;

**this**.password = password;

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

}

---------------------------------

**package** com.enumtech.jwtoken.model;

**public** **class** AuthenticationResponse {

**private** **final** String jwt;

**public** AuthenticationResponse(String jwt) {

**this**.jwt = jwt;

}

**public** String getJwt() {

**return** jwt;

}

}

----------------------------------

@Autowired

**private** MyUserDetailsService userDetailsService;

@Autowired

**private** AuthenticationManager authenticationManager;

@Autowired

**private** JwtUtil jwtTokenUtil;

@RequestMapping(value = "/authenticate", method = RequestMethod.***POST***)

**public** ResponseEntity<?> createAuthenticationToken(@RequestBody AuthenticationRequest authenticationRequest) **throws** Exception{

**try** {

authenticationManager.authenticate(**new** UsernamePasswordAuthenticationToken(authenticationRequest.getUsername(), authenticationRequest.getPassword()));

}

**catch** (BadCredentialsException e) {

**throw** **new** Exception("Incorrect username or password", e);

}

**final** UserDetails userDetails = userDetailsService.loadUserByUsername(authenticationRequest.getUsername());

**final** String jwt = jwtTokenUtil.generateToken(userDetails);

**return** ResponseEntity.*ok*(**new** AuthenticationResponse(jwt));

}

-------------------------------

NOTE : for the **private** AuthenticationManager authenticationManager to work overide the follwoing method in WebSecurity class

@Override

@Bean

**public** AuthenticationManager authenticationManagerBean() **throws** Exception {

**return** **super**.~~authenticationManagerBean~~();

}

-----------------

step 8 : by default spring security is authenticating each request and directs it to the default login page, but this "/authenticate" endpoint has to be by passed as it itself is doing the authentication so hence we will bypass the "/authenticate" from the default login page.

override the configute(HttpSecurity http)

@Override

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

http

.csrf().disable()

.authorizeRequests().antMatchers("/authenticate").permitAll()

.anyRequest().authenticated();

}

step 9 : run and hit the "/authenticate" endpoint and verify whether the token is being returned.

Request Body:

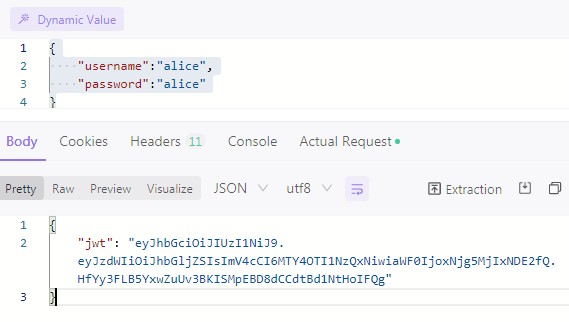
{

    "username":"alice",

    "password":"alice"

}

in response you will get a token



NOTE while generating token use @Controller

While consuming token use @RestController

NOTE : this token is stored on client side in a)local storage or b) cookie and passed in all subsequent requests in the header

**Part 2:**

Step 10: now we have to inform spring security that for every subsequent request

a) read the header for the Key : "Authorization" and

b) read the token

c) verify its a valid token

d) extract the username and put it in the security context(loadUserByUsername).

step 11 : the way we do step 10 is by intercepting the request by creating a filter(there are many filters for specefic tasks, we will use the OncePerRequestFilter

a) create a package "filter"

b) create a user defined class JwtRequestFilter extends OncePerRequestFilter

c) override the doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

**package** com.enumtech.jwtoken.filters;

**import** java.io.IOException;

**import** javax.servlet.FilterChain;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

**import** org.springframework.stereotype.Component;

**import** org.springframework.web.filter.OncePerRequestFilter;

**import** com.enumtech.jwtoken.security.MyUserDetailsService;

**import** com.enumtech.jwtoken.util.JwtUtil;

@Component

**public** **class** JwtRequestFilter **extends** OncePerRequestFilter {

@Autowired

**private** MyUserDetailsService userDetailsService;

@Autowired

**private** JwtUtil jwtUtil;

@Override

**protected** **void** doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

**throws** ServletException, IOException {

**final** String authorizationHeader = request.getHeader("Authorization");

String username = **null**;

String jwt = **null**;

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

jwt = authorizationHeader.substring(7);

username = jwtUtil.extractUsername(jwt);

}

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

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

**if** (jwtUtil.validateToken(jwt, userDetails)) {

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = **new** UsernamePasswordAuthenticationToken(

userDetails, **null**, userDetails.getAuthorities());

usernamePasswordAuthenticationToken

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

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

}

}

chain.doFilter(request, response);

}

}

step 12:

now in the configure(HttpSecurity http) to use this filter

@Autowired

**private** JwtRequestFilter jwtRequestFilter;

@Override

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

httpSecurity.csrf().disable()

.authorizeRequests().antMatchers("/authenticate").permitAll().

anyRequest().authenticated().and().sessionManagement()

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

httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.**class**);

}

step 13: in the Controller make a new endpoint with" /welcome" returning a String "Welcome"

(reason bcoz someone who is having the valid token can only be authorised to access this end point)

@Controller

**public** **class** EmployeeController {

@RequestMapping("/welcome")

**public** String greet()

{

**return** "Welcome";

}

}

step 14: now try to access the ("/welcome") end point by passing the token in the header.

Key : "Authorization"

value : "Bearer JwtToken" //Bearer keyword followed by space char followed by jwttoken

