**JWT\_HANDSON**

**EXERCISE 1 :**

**Create authentication service that returns JWT**   
  
As part of first step of JWT process, the user credentials needs to be sent to authentication service request that generates and returns the JWT.  
  
Ideally when the below curl command is executed that calls the new authentication service, the token should be responded. Kindly note that the credentials are passed using -u option.  
  
**Request**

curl -s -u user:pwd http://localhost:8090/authenticate

**Response**

{"token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOjE1NzAzODA2NzR9.t3LRvlCV-hwKfoqZYlaVQqEUiBloWcWn0ft3tgv0dL0"}

This can be incorporated as three major steps:

* Create authentication controller and configure it in SecurityConfig
* Read Authorization header and decode the username and password
* Generate token based on the user retrieved in the previous step

Let incorporate the above as separate hands on exercises.

**SOLUTION :**

**Step 1: Create Authentication Controller**

@RestController  
public class AuthenticationController {  
  
 @Autowired  
 private JwtUtil jwtUtil;  
  
 @PostMapping("/authenticate")  
 public ResponseEntity<?> generateToken(@RequestHeader("Authorization") String authHeader) {  
 String[] credentials = decodeBasicAuth(authHeader);  
 String username = credentials[0];  
 String password = credentials[1];  
  
 // Ideally validate against DB or in-memory store  
 if ("user".equals(username) && "pwd".equals(password)) {  
 String token = jwtUtil.generateToken(username);  
 return ResponseEntity.ok(Collections.singletonMap("token", token));  
 } else {  
 return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();  
 }  
 }  
  
 private String[] decodeBasicAuth(String authHeader) {  
 String base64Credentials = authHeader.substring("Basic".length()).trim();  
 byte[] credDecoded = Base64.getDecoder().decode(base64Credentials);  
 String credentials = new String(credDecoded, StandardCharsets.UTF\_8);  
 return credentials.split(":", 2);  
 }  
}

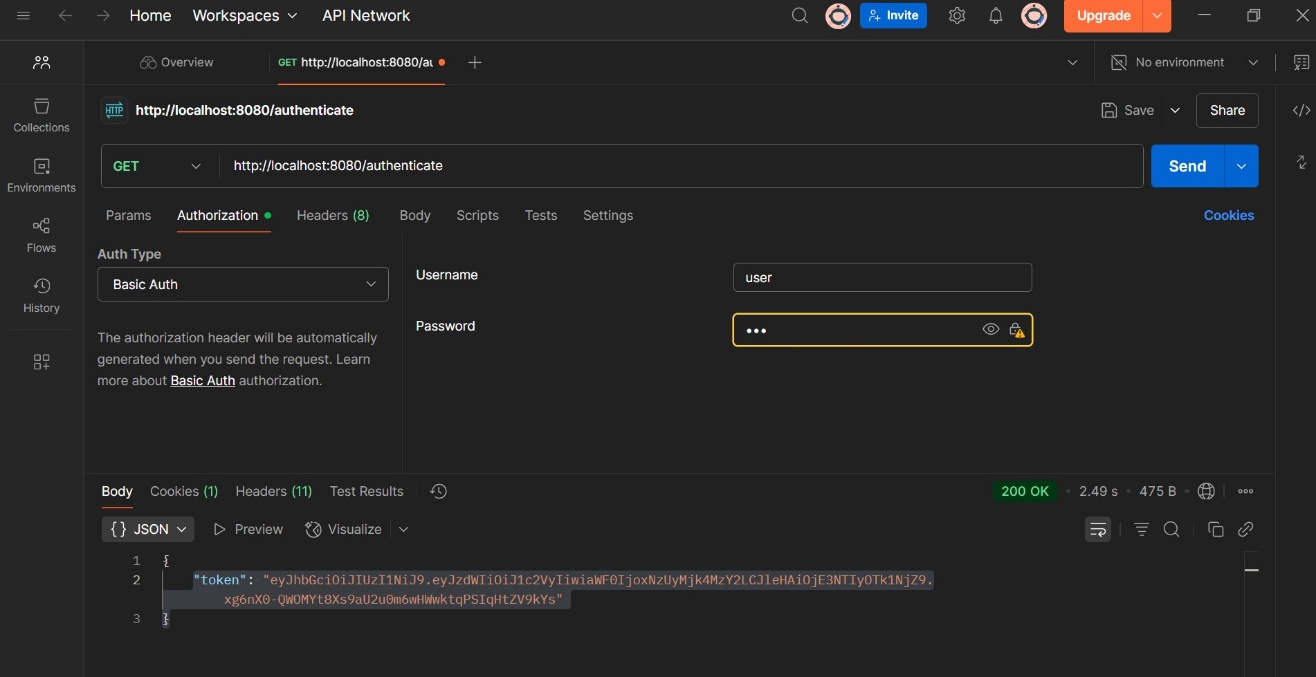
**Step 2: Configure Security in SecurityConfig**

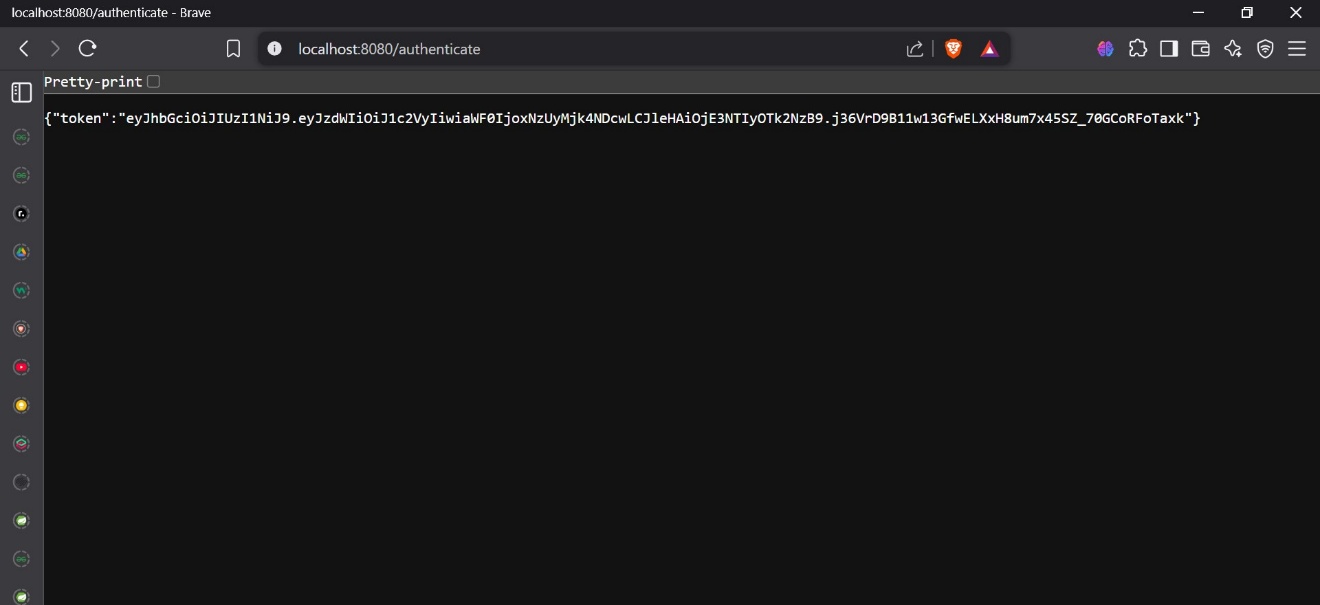
@Configuration  
@EnableWebSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.csrf().disable()  
 .authorizeRequests()  
 .antMatchers("/authenticate").permitAll()  
 .anyRequest().authenticated();  
 }  
}

**Step 3: JWT Utility Class**

@Component  
public class JwtUtil {  
  
 private String SECRET\_KEY = "secret";  
  
 public String generateToken(String username) {  
 return Jwts.builder()  
 .setSubject(username)  
 .setIssuedAt(new Date(System.currentTimeMillis()))  
 .setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10))  
 .signWith(SignatureAlgorithm.HS256, SECRET\_KEY)  
 .compact();  
 }  
}

**OUTPUT :**

****

****