**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.

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.

**Program:**

**Pom.xml:**

<dependencies>

<!-- Spring Boot Web -->

<dependency>

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

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

</dependency>

<!-- Spring Boot Security -->

<dependency>

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

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

</dependency>

<!-- JWT -->

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

</dependencies>

**JwtUtil.java:**

package com.cognizant.auth.util;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.stereotype.Component;

import java.util.Date;

@Component

public class JwtUtil {

private String secretKey = "mysecretkey"; // Should be in config

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date())

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

.signWith(SignatureAlgorithm.HS256, secretKey)

.compact();

}

}

**AuthenticationController.java:**

package com.cognizant.auth.controller;

import com.cognizant.auth.util.JwtUtil;

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

import org.springframework.http.ResponseEntity;

import org.springframework.util.Base64Utils;

import org.springframework.web.bind.annotation.\*;

import javax.servlet.http.HttpServletRequest;

@RestController

public class AuthenticationController {

@Autowired

private JwtUtil jwtUtil;

@GetMapping("/authenticate")

public ResponseEntity<?> authenticate(HttpServletRequest request) {

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

if (header == null || !header.startsWith("Basic ")) {

return ResponseEntity.status(401).body("Missing or invalid Authorization header");

}

// Decode Base64 credentials

String base64Credentials = header.substring("Basic ".length());

String credentials = new String(Base64Utils.decodeFromString(base64Credentials));

String[] values = credentials.split(":", 2);

String username = values[0];

String password = values[1];

// Hardcoded credentials (you can use UserDetailsService later)

if ("user".equals(username) && "pwd".equals(password)) {

String token = jwtUtil.generateToken(username);

return ResponseEntity.ok("{\"token\":\"" + token + "\"}");

} else {

return ResponseEntity.status(401).body("Invalid credentials");

}

}

}

**SecurityConfig.java:**

package com.cognizant.auth.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.\*;

@EnableWebSecurity

@Configuration

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.csrf().disable()

.authorizeRequests()

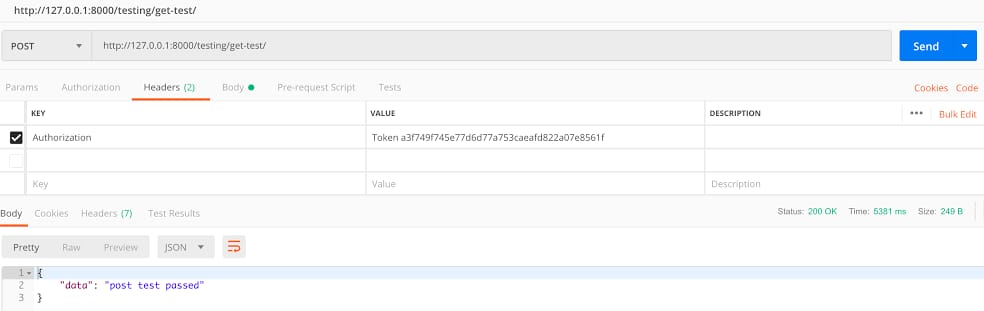
.antMatchers("/authenticate").permitAll()

.anyRequest().authenticated();

}

}

**Output:**

****