**Topic:- JWT-handson**

**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

**Solution**

**Code:**

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>jwt-auth-demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>JWT Authentication Demo</name>

<description>Demo Spring Boot project with JWT auth</description>

<parent>

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

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

<version>3.2.4</version>

<relativePath/>

</parent>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<dependency>

<groupId>commons-codec</groupId>

<artifactId>commons-codec</artifactId>

<version>1.15</version>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**AuthenticationController.java**

package com.example.jwt;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.Base64;

import java.util.Collections;

@RestController

public class AuthenticationController {

@GetMapping("/authenticate")

public ResponseEntity<?> authenticate(@RequestHeader(value = "Authorization", required = false) String authHeader) {

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

return ResponseEntity.*status*(HttpStatus.*BAD\_REQUEST*).body("Missing or invalid Authorization header");

}

try {

String[] credentials = decodeBasicAuth(authHeader);

String username = credentials[0];

String password = credentials[1];

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*).body("Invalid credentials");

}

} catch (Exception e) {

return ResponseEntity.*status*(HttpStatus.*BAD\_REQUEST*).body("Invalid authentication format");

}

}

private String[] decodeBasicAuth(String authHeader) {

String base64Credentials = authHeader.substring("Basic ".length()).trim();

byte[] decodedBytes = Base64.*getDecoder*().decode(base64Credentials);

String decodedString = new String(decodedBytes);

return decodedString.split(":", 2);

}

}

**JwtUtil.java**

package com.example.jwt;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import java.util.Date;

public class JwtUtil {

private static final String *SECRET\_KEY* = "mySecretKey123456";

private static final long *EXPIRATION\_TIME* = 10 \* 60 \* 1000;

public static String generateToken(String username) {

long currentTimeMillis = System.*currentTimeMillis*();

return Jwts.*builder*()

.setSubject(username)

.setIssuedAt(new Date(currentTimeMillis))

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

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

.compact();

}

}

**SecurityConfig.java**

package com.example.jwt;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.Customizer;

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

import org.springframework.security.web.SecurityFilterChain;

@Configuration

public class SecurityConfig {

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

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

.authorizeHttpRequests(auth -> auth

.requestMatchers("/authenticate").permitAll()

.anyRequest().authenticated()

)

.httpBasic(Customizer.*withDefaults*());

return http.build();

}

}

**JwtAuthDemoApplication.java**

package com.example.jwt;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

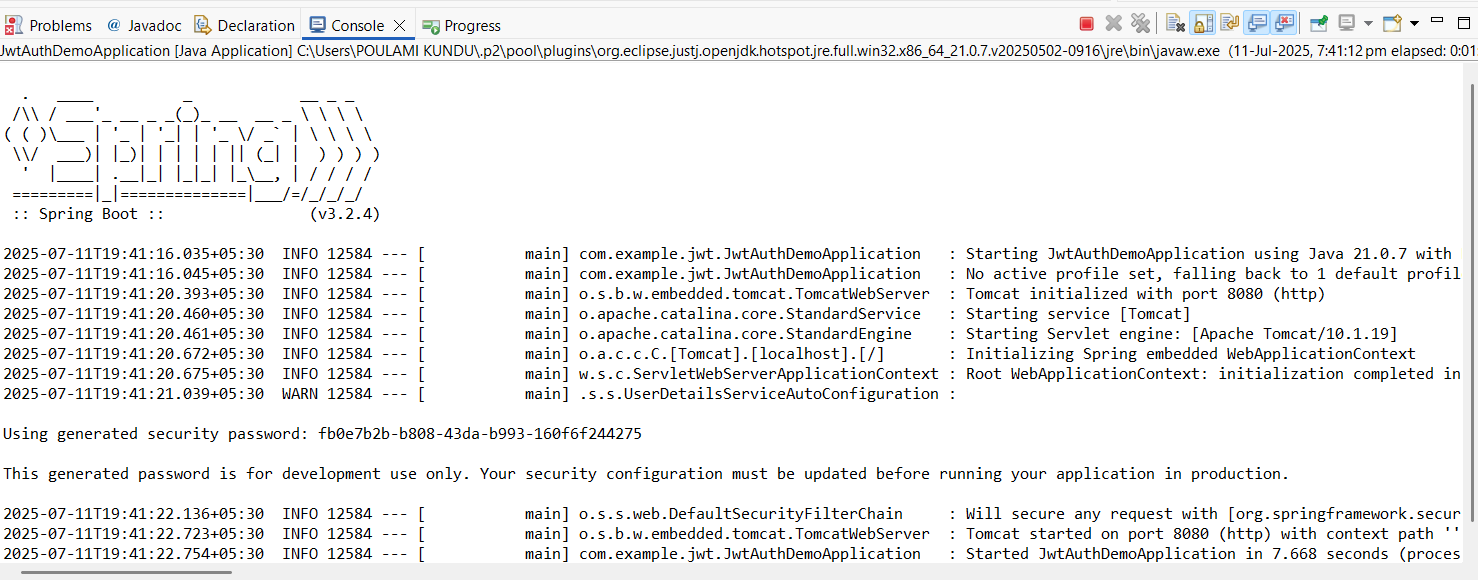
public class JwtAuthDemoApplication {

public static void main(String[] args) {

SpringApplication.*run*(JwtAuthDemoApplication.class, args);

}

}

**OUTPUT:-**