1. **Create authentication service that returns JWT**

pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>3.5.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.cognizant</groupId>

<artifactId>spring-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>spring-learn</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

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

</properties>

<dependencies>

<!-- Spring Web -->

<dependency>

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

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

</dependency>

<!-- Spring 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>

<dependency>

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

<artifactId>jaxb-api</artifactId>

<version>2.3.1</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

JwtAuthApplication.java

package com.example.jwt;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class JwtAuthApplication {

public static void main(String[] args) {

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

}

}

AuthController.java

package com.example.jwt;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestHeader;

import org.springframework.web.bind.annotation.RestController;

import java.util.Base64;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

*@RestController*

public class AuthController {

private final String SECRET\_KEY = "MySecretKeyForJWTSigning";

*@GetMapping*("/authenticate")

public ResponseEntity<?> authenticate(*@RequestHeader*("Authorization") String authHeader) {

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

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

}

// Decode Base64

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

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

String credentials = new String(credDecoded);

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

String username = parts[0];

String password = parts[1];

// Validate username/password (here, we hardcode "user"/"pwd")

if (!"user".equals(username) || !"pwd".equals(password)) {

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

}

// Generate JWT

String jwt = Jwts.*builder*()

.setSubject(username)

.setIssuedAt(new Date())

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

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

.compact();

Map<String, String> response = new HashMap<>();

response.put("token", jwt);

return ResponseEntity.*ok*(response);

}

}

SecurityConfig.java

package com.example.jwt;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

import org.springframework.security.web.SecurityFilterChain;

*@Configuration*

public class SecurityConfig {

*@Bean*

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http

.csrf().disable()

.authorizeRequests()

.anyRequest().permitAll();

return http.build();

}

}

