**WEEK 4: 5. JWT-handson**

**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.

**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.cognizant.spring\_learn</groupId>

<artifactId>jwt-auth</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>jwt-auth</name>

<description>JWT Authentication Project</description>

<parent>

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

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

<version>3.5.3</version>

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

</parent>

<properties>

<java.version>21</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-api</artifactId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId>

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</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>

<version>3.5.3</version>

</plugin>

</plugins>

</build>

</project>

**SpringLearnApplication.java**

package com.cognizant.spring\_learn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

public static void main(String[] args) {

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

}

}

**JwtUtil.java**

package com.cognizant.spring\_learn.util;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import org.springframework.stereotype.Service;

import java.security.Key;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

@Service

public class JwtUtil {

private final Key key = Keys.secretKeyFor(SignatureAlgorithm.*HS256*);

public String generateToken(String username) {

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

return Jwts.builder()

.setClaims(claims)

.setSubject(username)

.setIssuedAt(new Date(System.currentTimeMillis()))

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

.signWith(key)

.compact();

}

}

**AuthenticationController.java**

package com.cognizant.spring\_learn.controller;

import com.cognizant.spring\_learn.util.JwtUtil;

import jakarta.servlet.http.HttpServletRequest;

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

import org.springframework.http.ResponseEntity;

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

import java.util.Base64;

import java.util.HashMap;

import java.util.Map;

@RestController

public class AuthenticationController {

@Autowired

private JwtUtil jwtUtil;

@RequestMapping("/authenticate")

public ResponseEntity<?> authenticate(HttpServletRequest request) {

try {

System.out.println("Inside authenticate method");

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

System.out.println("Authorization Header: " + header);

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

System.out.println(" Missing or invalid Authorization header");

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

}

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

String decoded = new String(Base64.getDecoder().decode(base64Credentials));

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

System.out.println("Decoded username:password = " + decoded);

if (parts.length != 2) {

return ResponseEntity.status(401).body("Invalid Authorization format");

}

String username = parts[0];

String password = parts[1];

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

System.out.println(" Valid credentials, generating token...");

String token = jwtUtil.generateToken(username);

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

response.put("token", token);

return ResponseEntity.ok(response);

} else {

System.out.println(" Invalid credentials");

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

}

} catch (Exception e) {

e.printStackTrace();

return ResponseEntity.status(500).body("Internal server error: " + e.getMessage());

}

}

}

**SecurityConfig.java**

package com.cognizant.spring\_learn.config;

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()

.authorizeHttpRequests()

.anyRequest().permitAll(); // Allow all requests

return http.build();

}

}

**HelloController.java**

package com.cognizant.spring\_learn.controller;

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

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

@RestController

public class HelloController {

@GetMapping("/hello")

public String sayHello() {

return "Hello World!! (Secured)";

}

}

**Output:**



