**WEEK 4 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.  
  
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.

**CODES AND OUTPUTS :**

**AuthenticationController.java**

package com.cognizant.jwt\_auth.controller;

import java.util.Base64;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

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

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.security.Keys;

@RestController

public class AuthenticationController {

private static final Logger LOGGER = LoggerFactory.getLogger(AuthenticationController.class);

@GetMapping("/authenticate")

public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {

LOGGER.info("Start authenticate()");

LOGGER.debug("Authorization header: {}", authHeader);

String user = getUser(authHeader);

String token = generateJwt(user);

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

response.put("token", token);

LOGGER.info("End authenticate()");

return response;

}

private String getUser(String authHeader) {

LOGGER.info("Start getUser()");

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

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

String decoded = new String(decodedBytes);

String user = decoded.split(":")[0];

LOGGER.debug("Decoded user: {}", user);

return user;

}

private String generateJwt(String user) {

byte[] secret = "my-secret-key-which-is-long-enough".getBytes(); // at least 256 bits (32 chars)

return Jwts.builder()

.setSubject(user)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + 20 \* 60 \* 1000))

.signWith(Keys.hmacShaKeyFor(secret)) // Use Keys class with newer version

.compact();

}

}

**SecurityConfig.java**

package com.cognizant.jwt\_auth.security;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

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

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

private static final Logger LOGGER = LoggerFactory.getLogger(SecurityConfig.class);

@Bean

public AuthenticationManager authManager(HttpSecurity http, PasswordEncoder encoder) throws Exception {

return http.getSharedObject(AuthenticationManagerBuilder.class)

.inMemoryAuthentication()

.withUser("user").password(encoder.encode("pwd")).roles("USER")

.and()

.withUser("admin").password(encoder.encode("pwd")).roles("ADMIN")

.and()

.passwordEncoder(encoder)

.and()

.build();

}

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.csrf().disable()

.httpBasic().and()

.authorizeHttpRequests()

.requestMatchers("/authenticate").hasAnyRole("USER", "ADMIN")

.anyRequest().authenticated();

return http.build();

}

@Bean

public PasswordEncoder passwordEncoder() {

LOGGER.info("Password encoder created");

return new BCryptPasswordEncoder();

}

}

Dependenices (pom.xml)

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-test</artifactId>

<scope>test</scope>

</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> <!-- or jjwt-gson if you prefer -->

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

</dependencies>

A screenshot of a computer

AI-generated content may be incorrect.