**Hands on 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

Let incorporate the above as separate hands on exercises.

**Solution:**

**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>3.3.1</version> <relativePath/> </parent>

<groupId>com.example</groupId>

<artifactId>jwt-auth-service</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>jwt-auth-service</name>

<description>Authentication service that returns JWT</description>

<properties>

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

</properties>

<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.projectlombok</groupId>

<artifactId>lombok</artifactId>

<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>0.12.5</version> </dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.12.5</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId>

<version>0.12.5</version>

<scope>runtime</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

**SecurityConfig.java**

package com.example.jwtauthservice.config;

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.configuration.AuthenticationConfiguration;

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

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

import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

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

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

import org.springframework.security.web.SecurityFilterChain;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

.csrf(AbstractHttpConfigurer::disable) // Disable CSRF for API

.authorizeHttpRequests(authorize -> authorize

.requestMatchers("/authenticate").permitAll() // Allow unauthenticated access to /authenticate

.anyRequest().authenticated() // All other requests require authentication

)

.sessionManagement(session -> session

.sessionCreationPolicy(SessionCreationPolicy.STATELESS) // Use stateless sessions for JWT

);

return http.build();

}

@Bean

public UserDetailsService userDetailsService() {

UserDetails user = User.builder()

.username("user")

.password(passwordEncoder().encode("pwd")) // Encode the password

.roles("USER")

.build();

return new InMemoryUserDetailsManager(user);

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticationConfiguration) throws Exception {

return authenticationConfiguration.getAuthenticationManager();

}

}

**AuthenticationController.java**

package com.example.jwtauthservice.controller;

import org.springframework.http.HttpHeaders;

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.AuthenticationException;

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

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

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

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

import java.nio.charset.StandardCharsets;

import java.util.Base64;

@RestController

@RequestMapping("/authenticate")

public class AuthenticationController {

private final AuthenticationManager authenticationManager;

public AuthenticationController(AuthenticationManager authenticationManager) {

this.authenticationManager = authenticationManager;

}

@PostMapping

public ResponseEntity<String> authenticate(@RequestHeader(HttpHeaders.AUTHORIZATION) String authorizationHeader) {

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

return ResponseEntity.badRequest().body("Invalid Authorization header. Basic authentication expected.");

}

try {

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

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

String credentials = new String(decodedCredentials, StandardCharsets.UTF\_8);

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

if (parts.length != 2) {

return ResponseEntity.badRequest().body("Invalid Basic authentication format.");

}

String username = parts[0];

String password = parts[1];

// Now, use Spring Security's AuthenticationManager to authenticate

Authentication authentication = authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(username, password)

);

// If authentication is successful, you can proceed to generate JWT

// For this exercise, we'll just confirm successful authentication

return ResponseEntity.ok("Authentication successful for user: " + authentication.getName());

} catch (IllegalArgumentException e) {

return ResponseEntity.badRequest().body("Malformed Base64 credentials.");

} catch (AuthenticationException e) {

return ResponseEntity.status(401).body("Authentication failed: " + e.getMessage());

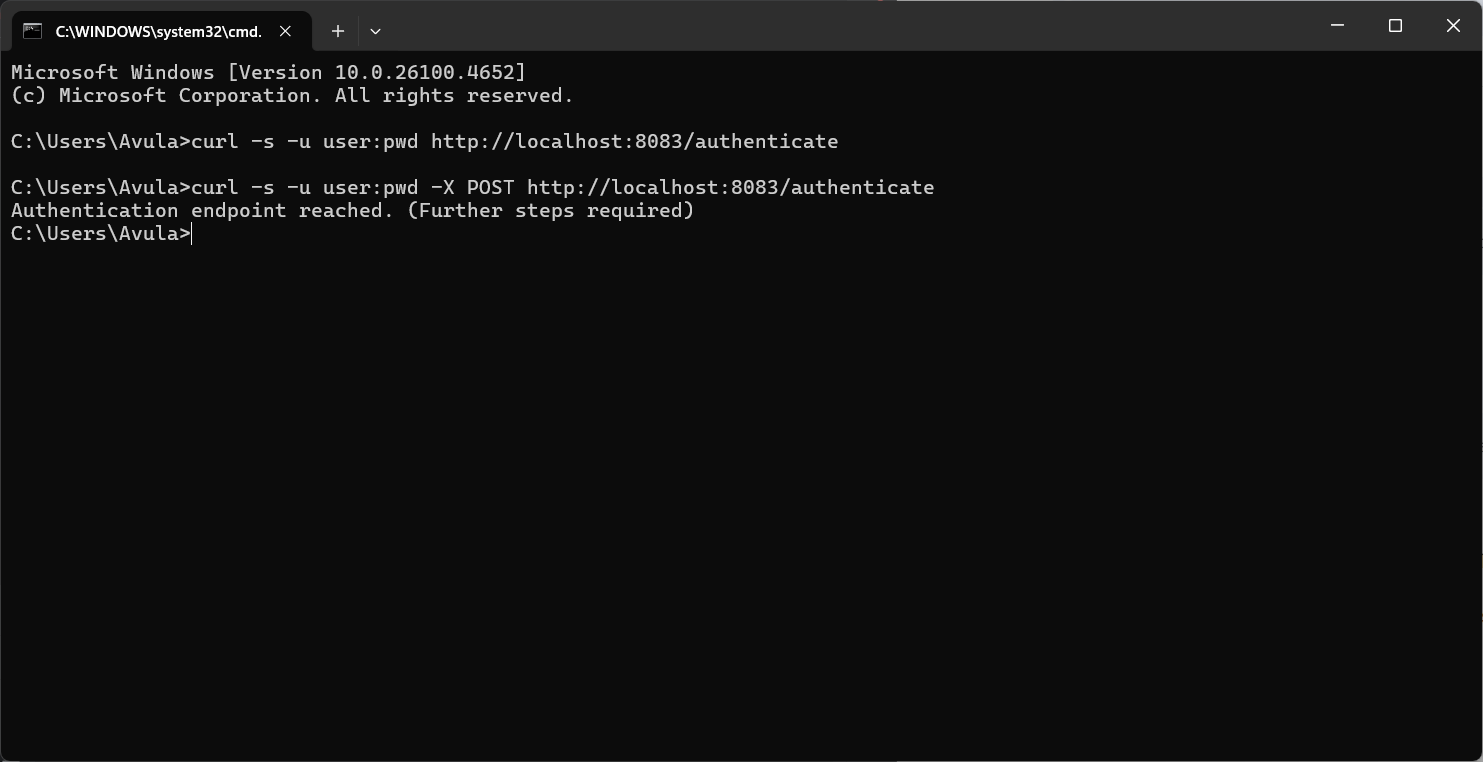
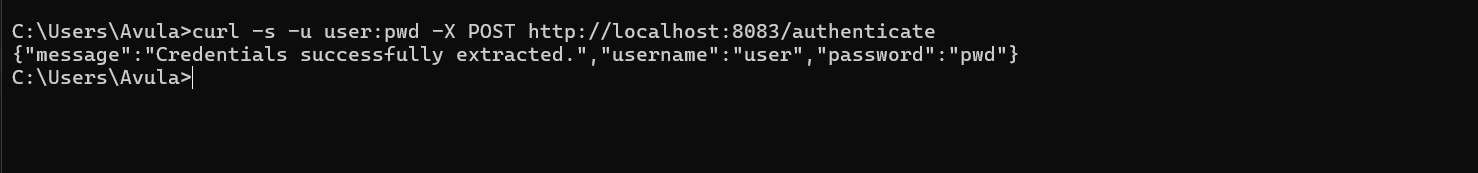
}

}

}

**OUTPUT:**

**curl -s -u user:pwd** [**http://localhost:8083/authenticate**](http://localhost:8083/authenticate)

****