**5. JWT-handson**

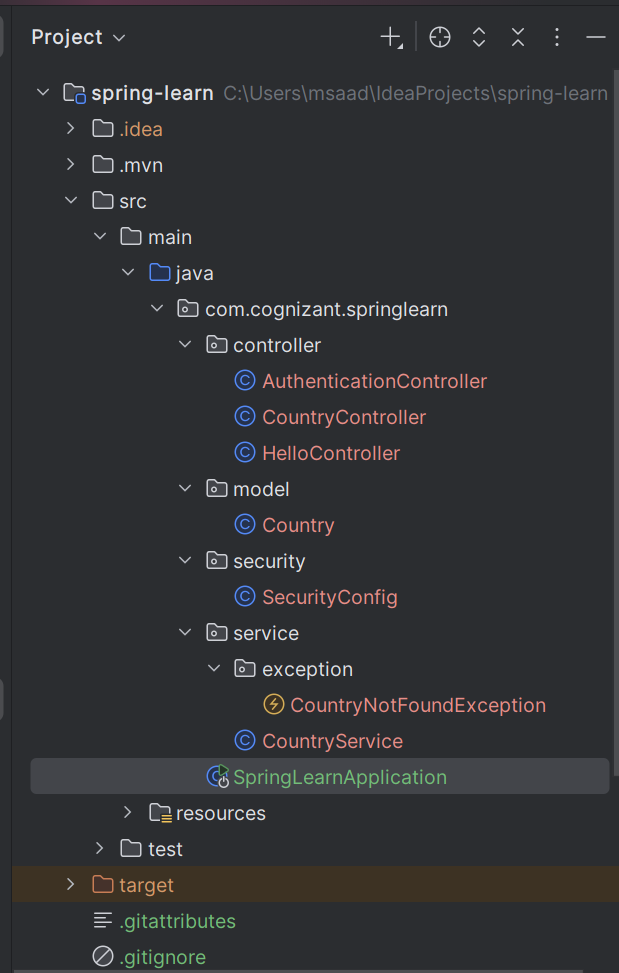
**Create authentication service that returns JWT**   
As part of first step of JWT process, the user credentials needs to be sent to authentication service request that generates and returns the JWT.  
  
Ideally when the below curl command is executed that calls the new authentication service, the token 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"}

**Folder Structure:**

****

**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>spring-learn</description>  
 <url/>  
 <licenses>  
 <license/>  
 </licenses>  
 <developers>  
 <developer/>  
 </developers>  
 <scm>  
 <connection/>  
 <developerConnection/>  
 <tag/>  
 <url/>  
 </scm>  
 <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-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>  
  
 **<!-- Spring Security -->** <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
  
 <!-- JWT Library -->  
 <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 -->  
 <version>0.11.5</version>  
 <scope>runtime</scope>  
 </dependency>  
  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

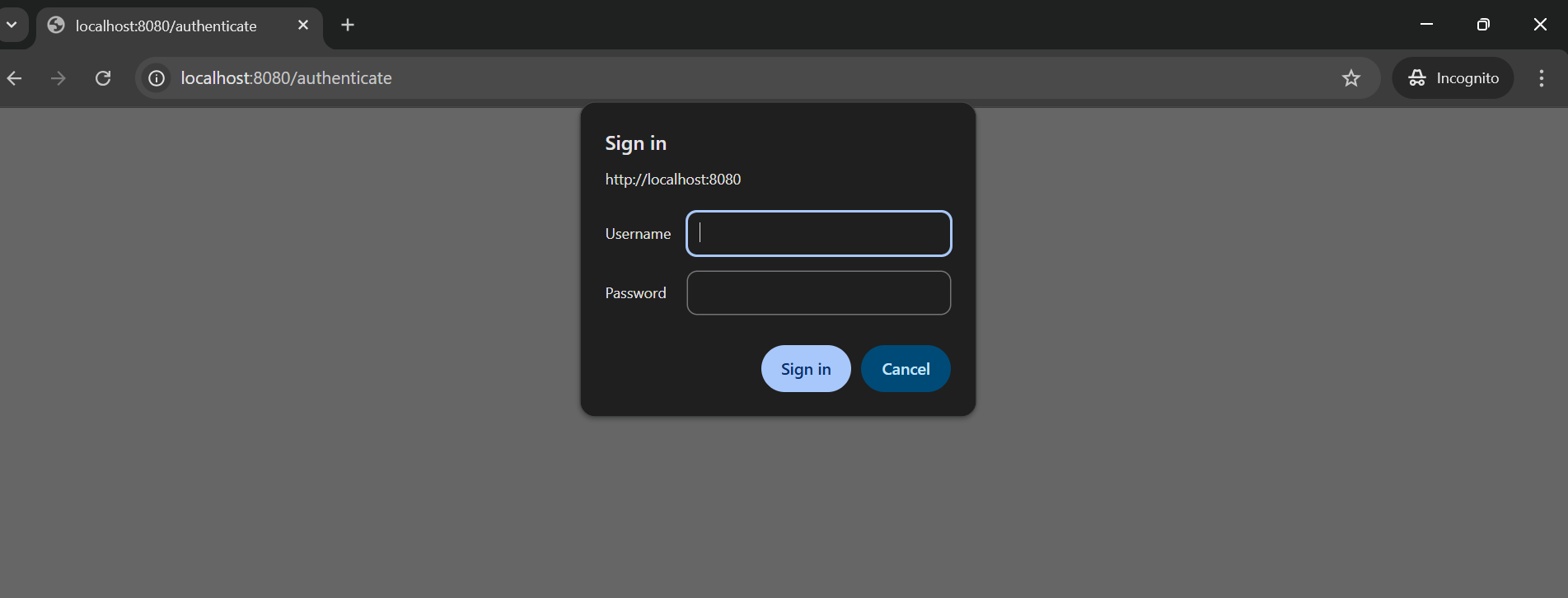
**SecurityConfig.java (in package security)**

package com.cognizant.springlearn.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.configuration.AuthenticationConfiguration;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.web.SecurityFilterChain;  
import org.springframework.security.provisioning.InMemoryUserDetailsManager;  
import org.springframework.security.core.userdetails.User;  
import org.springframework.security.core.userdetails.UserDetails;  
  
@Configuration  
public class SecurityConfig {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(SecurityConfig.class);  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public InMemoryUserDetailsManager userDetailsService() {  
 UserDetails admin = User  
 .*withUsername*("admin")  
 .password(passwordEncoder().encode("pwd"))  
 .roles("ADMIN")  
 .build();  
  
 UserDetails user = User  
 .*withUsername*("user")  
 .password(passwordEncoder().encode("pwd"))  
 .roles("USER")  
 .build();  
  
 return new InMemoryUserDetailsManager(admin, user);  
 }  
  
 @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 AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {  
 return config.getAuthenticationManager();  
 }  
}

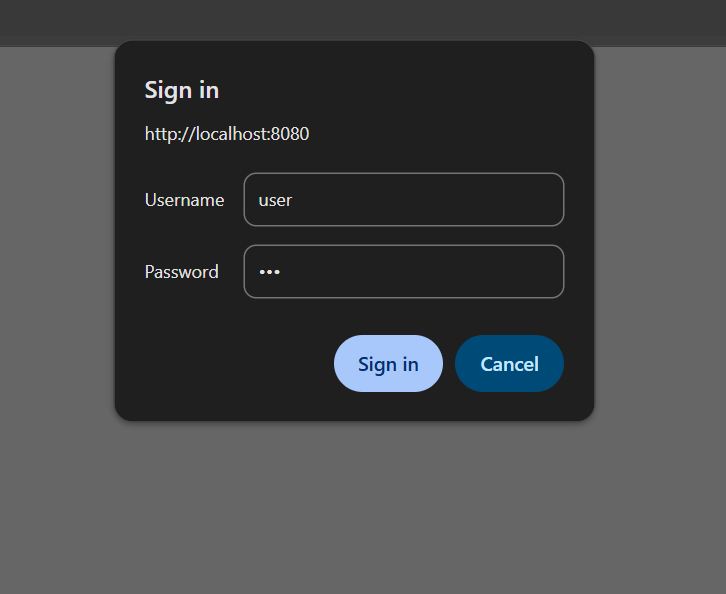
**AuthenticationController.java (in the package controller)**

package com.cognizant.springlearn.controller;  
  
import io.jsonwebtoken.JwtBuilder;  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
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.security.Keys;  
  
  
import java.util.Base64;  
import java.util.Date;  
import java.util.HashMap;  
import java.util.Map;  
  
import io.jsonwebtoken.security.Keys;  
import java.security.Key;  
  
@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> map = new HashMap<>();  
 map.put("token", token);  
  
 *LOGGER*.info("END - authenticate()");  
 return map;  
 }  
  
 private String getUser(String authHeader) {  
 String encodedCredentials = authHeader.replace("Basic ", "");  
 byte[] decodedBytes = Base64.*getDecoder*().decode(encodedCredentials);  
 String decodedCredentials = new String(decodedBytes); // "user:pwd"  
 *LOGGER*.debug("Decoded credentials: {}", decodedCredentials);  
 return decodedCredentials.split(":")[0]; // return "user"  
 }  
  
 private String generateJwt(String user) {  
 String secret = "my-secret-key-that-is-long-enough-123456"; // At least 32 chars  
 Key key = Keys.*hmacShaKeyFor*(secret.getBytes());  
  
 return Jwts.*builder*()  
 .setSubject(user)  
 .setIssuedAt(new Date())  
 .setExpiration(new Date(System.*currentTimeMillis*() + 20 \* 60 \* 1000)) // 20 minutes  
 .signWith(key, SignatureAlgorithm.*HS256*)  
 .compact();  
 }  
}

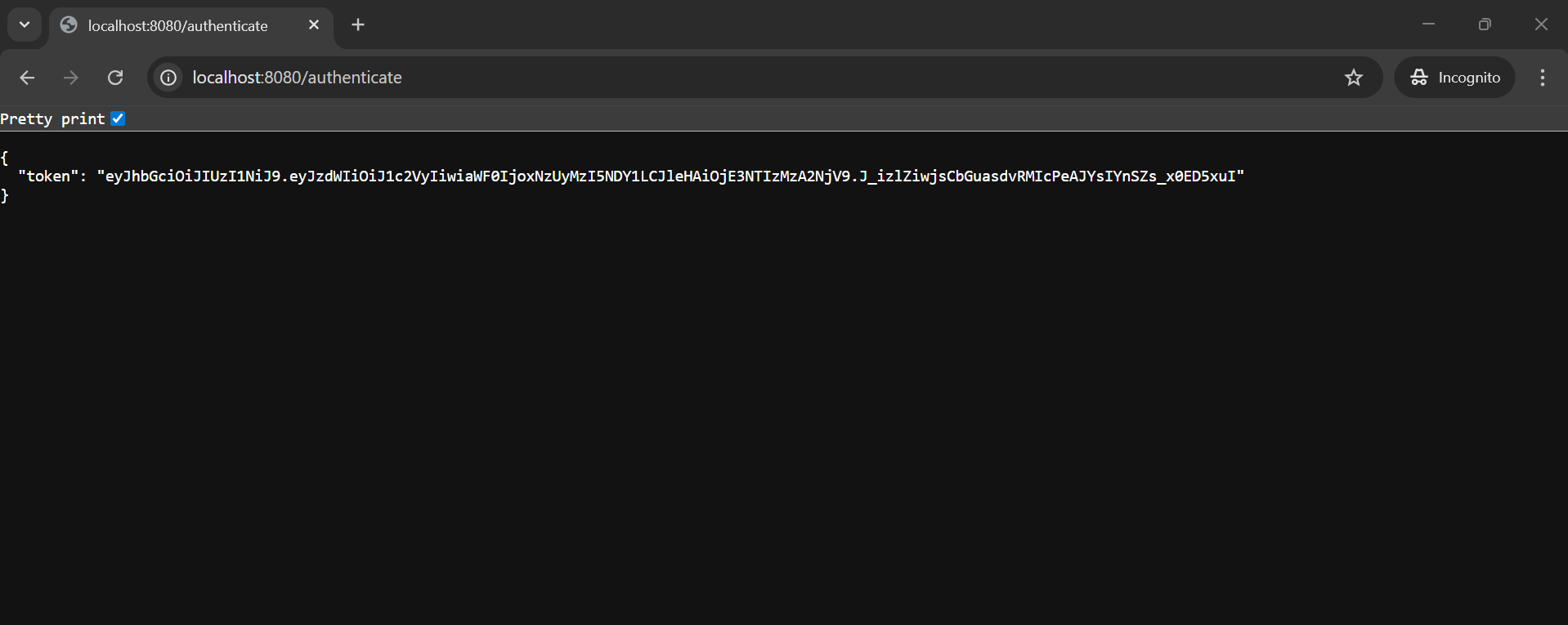
**Enter credentials:**

****

**I have entered “user” for username and “rak” for password**

****

**Output:**

****

**JWT has been successfully fetched .**