**Hands On 7: Generate token based on the user**

**AuthController.java**

package com.example.demo.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.\*;  
  
import java.util.Base64;  
import java.util.Date;  
import java.util.HashMap;  
import java.util.Map;  
  
@RestController  
public class AuthController {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(AuthController.class);  
  
 @GetMapping("/authenticate")  
 public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {  
 *LOGGER*.info("Start of authenticate()");  
 *LOGGER*.debug("Authorization header: {}", authHeader);  
  
 String user = getUser(authHeader);  
 *LOGGER*.debug("Extracted user from header: {}", user);  
  
 String token = generateJwt(user);  
 *LOGGER*.debug("Generated token: {}", token);  
  
 Map<String, String> map = new HashMap<>();  
 map.put("token", token);  
  
 *LOGGER*.info("End of authenticate()");  
 return map;  
 }  
  
 private String getUser(String authHeader) {  
 String encodedCredentials = authHeader.substring("Basic ".length());  
 byte[] decodedBytes = Base64.*getDecoder*().decode(encodedCredentials);  
 String decodedString = new String(decodedBytes);  
 return decodedString.split(":")[0];  
 }  
  
 private String generateJwt(String user) {  
 JwtBuilder builder = Jwts.*builder*();  
 builder.setSubject(user);  
 builder.setIssuedAt(new Date());  
 builder.setExpiration(new Date(new Date().getTime() + 20 \* 60 \* 1000));  
 builder.signWith(SignatureAlgorithm.*HS256*, "ThisIsASecretKeyUsedToSignAndVerifyJWTsWithHmacSha256ItMustBeLongEnoughToProvideSecurityAndAvoidWarningsOrExceptionsRelatedToKeyLengthEspeciallyWhenUsingStrongEncryptionAlgorithmsInProduction");  
 return builder.compact();  
 }  
}

**SecurityConfig.java**

package com.example.demo.security;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
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.core.userdetails.User;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.provisioning.InMemoryUserDetailsManager;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.crypto.password.NoOpPasswordEncoder;  
  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
  
 @Bean  
 public InMemoryUserDetailsManager userDetailsService() {  
 UserDetails user = User.*withUsername*("user")  
 .password("pwd")  
 .roles("USER")  
 .build();  
  
 UserDetails admin = User.*withUsername*("admin")  
 .password("pwd")  
 .roles("ADMIN")  
 .build();  
  
 return new InMemoryUserDetailsManager(user, admin);  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return NoOpPasswordEncoder.*getInstance*();}  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .csrf(csrf -> csrf.disable())  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/countries").hasRole("USER")  
 .requestMatchers("/authenticate").hasAnyRole("USER", "ADMIN")  
 .anyRequest().authenticated()  
 )  
 .httpBasic(customizer -> {});  
  
 return http.build();  
 }  
}

**DemoApplication.java**

package com.example.demo;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class DemoApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(DemoApplication.class, args);  
 }  
  
}

**Output**



