Exercise 1- Create an authentication service that returns JWT

**Solution:**

**JwtAuthApplication.java:**

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

**SecurityConfig.java:**

package com.example.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.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.http.SessionCreationPolicy;  
import org.springframework.security.web.SecurityFilterChain;  
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
import org.springframework.web.cors.CorsConfiguration;  
import org.springframework.web.cors.CorsConfigurationSource;  
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;  
  
import java.util.Arrays;  
  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .csrf(csrf -> csrf.disable())  
 .cors(cors -> cors.configurationSource(corsConfigurationSource()))  
 .sessionManagement(session ->  
 session.sessionCreationPolicy(SessionCreationPolicy.*STATELESS*))  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/authenticate", "/health").permitAll()  
 .anyRequest().authenticated()  
 )  
 .httpBasic(httpBasic -> httpBasic.disable())  
 .formLogin(form -> form.disable());  
  
 return http.build();  
 }  
  
 @Bean  
 public CorsConfigurationSource corsConfigurationSource() {  
 CorsConfiguration configuration = new CorsConfiguration();  
 configuration.setAllowedOriginPatterns(Arrays.*asList*("\*"));  
 configuration.setAllowedMethods(Arrays.*asList*("GET", "POST", "PUT", "DELETE", "OPTIONS"));  
 configuration.setAllowedHeaders(Arrays.*asList*("\*"));  
 configuration.setAllowCredentials(true);  
  
 UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();  
 source.registerCorsConfiguration("/\*\*", configuration);  
 return source;  
 }  
}

**AuthController.java:**

package com.example.controller;  
  
import com.example.dto.TokenResponse;  
import com.example.exception.AuthenticationException;  
import com.example.service.JwtService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import jakarta.servlet.http.HttpServletRequest;  
import java.nio.charset.StandardCharsets;  
import java.util.Base64;  
  
@RestController  
@RequestMapping("/")  
public class AuthController {  
  
 @Autowired  
 private JwtService jwtService;  
  
 @PostMapping("/authenticate")  
 public ResponseEntity<?> authenticate(HttpServletRequest request) {  
 try {  
 // Extract credentials from Authorization header  
 String[] credentials = extractCredentials(request);  
 String username = credentials[0];  
 String password = credentials[1];  
  
 // Validate credentials (in real app, use UserDetailsService)  
 if (isValidCredentials(username, password)) {  
 // Generate JWT token  
 String token = jwtService.generateToken(username);  
 return ResponseEntity.*ok*(new TokenResponse(token));  
 } else {  
 return ResponseEntity.*status*(HttpStatus.*UNAUTHORIZED*)  
 .body("{\"error\":\"Invalid credentials\"}");  
 }  
  
 } catch (Exception e) {  
 return ResponseEntity.*status*(HttpStatus.*UNAUTHORIZED*)  
 .body("{\"error\":\"Authentication failed: " + e.getMessage() + "\"}");  
 }  
 }  
  
 private String[] extractCredentials(HttpServletRequest request) {  
 String authHeader = request.getHeader("Authorization");  
  
 if (authHeader == null || !authHeader.startsWith("Basic ")) {  
 throw new AuthenticationException("Missing or invalid Authorization header");  
 }  
  
 // Extract Base64 encoded credentials  
 String base64Credentials = authHeader.substring(6);  
 String credentials = new String(Base64.*getDecoder*().decode(base64Credentials),  
 StandardCharsets.*UTF\_8*);  
  
 String[] values = credentials.split(":", 2);  
 if (values.length != 2) {  
 throw new AuthenticationException("Invalid credentials format");  
 }  
  
 return values;  
 }  
  
 private boolean isValidCredentials(String username, String password) {  
 // Simple validation for demo purposes  
 // In production, use proper user authentication service  
 return "user".equals(username) && "pwd".equals(password);  
 }  
  
 @GetMapping("/health")  
 public ResponseEntity<String> health() {  
 return ResponseEntity.*ok*("JWT Authentication Service is running!");  
 }  
}

**TokenResponse.java:**

package com.example.dto;  
  
public class TokenResponse {  
 private String token;  
  
 public TokenResponse() {}  
  
 public TokenResponse(String token) {  
 this.token = token;  
 }  
  
 public String getToken() {  
 return token;  
 }  
  
 public void setToken(String token) {  
 this.token = token;  
 }  
}

**AuthenticationException.java;**

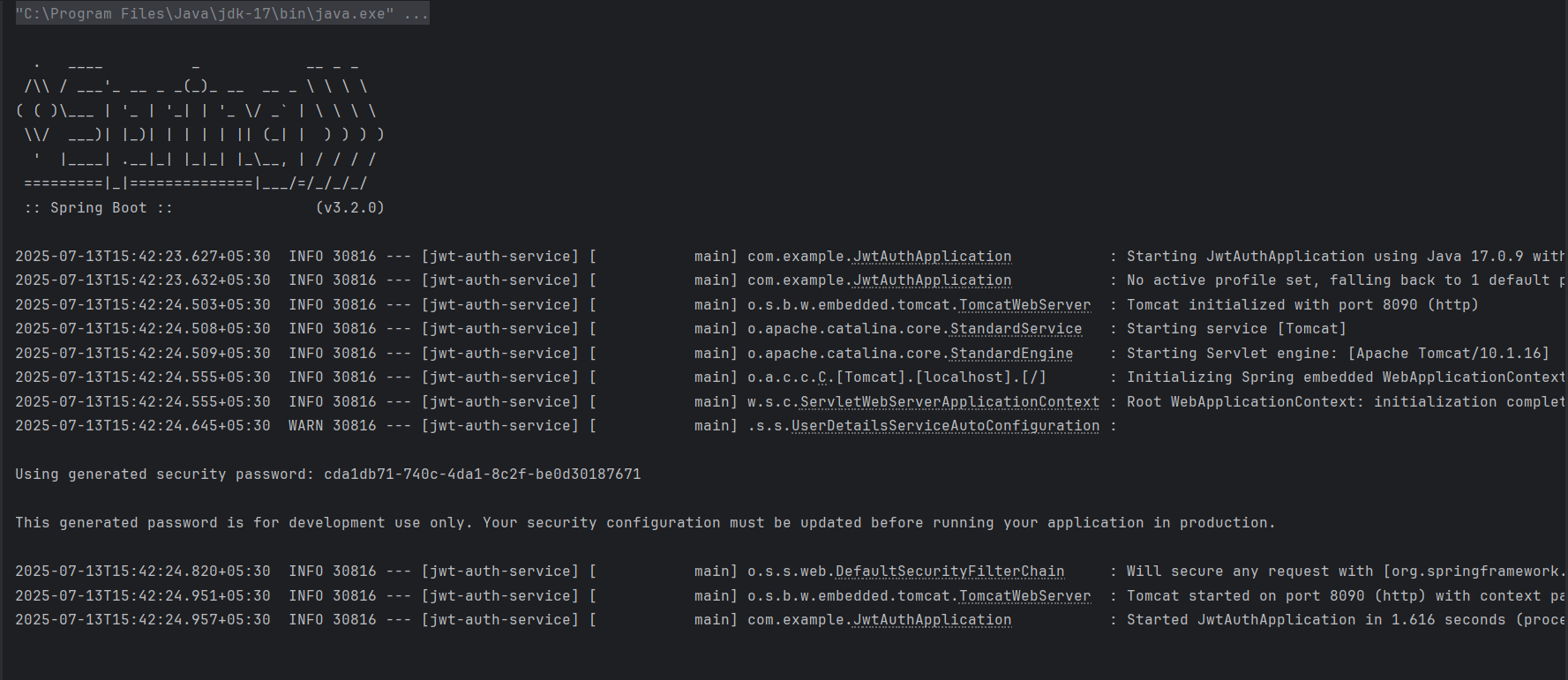
package com.example.exception;  
  
public class AuthenticationException extends RuntimeException {  
 public AuthenticationException(String message) {  
 super(message);  
 }  
  
 public AuthenticationException(String message, Throwable cause) {  
 super(message, cause);  
 }  
}

**JwtService.java:**

package com.example.service;  
  
import io.jsonwebtoken.Claims;  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import io.jsonwebtoken.security.Keys;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Service;  
  
import javax.crypto.SecretKey;  
import java.util.Date;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.function.Function;  
  
@Service  
public class JwtService {  
  
 @Value("${jwt.secret}")  
 private String secret;  
  
 @Value("${jwt.expiration}")  
 private Long expiration;  
  
 private SecretKey getSigningKey() {  
 return Keys.*hmacShaKeyFor*(secret.getBytes());  
 }  
  
 public String generateToken(String username) {  
 Map<String, Object> claims = new HashMap<>();  
 return createToken(claims, username);  
 }  
  
 private String createToken(Map<String, Object> claims, String subject) {  
 Date now = new Date();  
 Date expiryDate = new Date(now.getTime() + expiration);  
  
 return Jwts.*builder*()  
 .setClaims(claims)  
 .setSubject(subject)  
 .setIssuedAt(now)  
 .setExpiration(expiryDate)  
 .signWith(getSigningKey(), SignatureAlgorithm.*HS256*)  
 .compact();  
 }  
  
 public String extractUsername(String token) {  
 return extractClaim(token, Claims::getSubject);  
 }  
  
 public Date extractExpiration(String token) {  
 return extractClaim(token, Claims::getExpiration);  
 }  
  
 public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {  
 final Claims claims = extractAllClaims(token);  
 return claimsResolver.apply(claims);  
 }  
  
 private Claims extractAllClaims(String token) {  
 return Jwts.*parserBuilder*()  
 .setSigningKey(getSigningKey())  
 .build()  
 .parseClaimsJws(token)  
 .getBody();  
 }  
  
 public Boolean isTokenExpired(String token) {  
 return extractExpiration(token).before(new Date());  
 }  
  
 public Boolean validateToken(String token, String username) {  
 final String extractedUsername = extractUsername(token);  
 return (extractedUsername.equals(username) && !isTokenExpired(token));  
 }  
}

**application.yml:**

server:  
 port: 8090  
  
spring:  
 application:  
 name: jwt-auth-service  
  
jwt:  
 secret: mySecretKey123456789012345678901234567890  
 expiration: 1200000 *# 20 minutes in milliseconds*logging:  
 level:  
 com.example.jwtauth: DEBUG  
 org.springframework.security: DEBUG

**Output**;