Exercise-5: Create authentication service that returns JWT

AuthenticationController.java

package com.cognizant.spring\_learn.controller;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.web.bind.annotation.\*;

import java.security.Key;

import java.util.\*;

@RestController

@RequestMapping

public class AuthenticationController {

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

@Value("${jwt.secret}")

private String jwtSecret;

@GetMapping("/authenticate")

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

LOGGER.info("Start Authentication");

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

String user = extractUserFromBasicAuth(authHeader);

String token = generateJwt(user);

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

response.put("token", token);

LOGGER.info("End Authentication");

return response;

}

@GetMapping("/test")

public String test() {

return "Controller is working";

}

private String extractUserFromBasicAuth(String authHeader) {

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

throw new IllegalArgumentException("Invalid Authorization header");

}

String base64Credentials = authHeader.substring("Basic ".length());

String decoded = new String(Base64.getDecoder().decode(base64Credentials));

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

return decoded.split(":")[0]; // Extract username

}

private String generateJwt(String user) {

Key key = Keys.hmacShaKeyFor(jwtSecret.getBytes());

return Jwts.builder()

.setSubject(user)

.setIssuedAt(new Date())

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

.signWith(key, SignatureAlgorithm.HS256)

.compact();

}

}

**JwtAuthorizationFilter.java**

package com.cognizant.spring\_learn.security;

import io.jsonwebtoken.\*;

import io.jsonwebtoken.security.Keys;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;

import java.io.IOException;

import java.nio.charset.StandardCharsets;

import java.util.ArrayList;

public class JwtAuthorizationFilter extends BasicAuthenticationFilter {

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

private static final String SECRET\_KEY = "secretkeysecretkeysecretkey123456"; // Must be >= 256 bits (32 chars)

public JwtAuthorizationFilter(AuthenticationManager authManager) {

super(authManager);

LOGGER.info("JwtAuthorizationFilter Initialized");

}

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

throws IOException, ServletException {

LOGGER.info("Start JWT filter");

String header = request.getHeader("Authorization");

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

if (header == null || !header.startsWith("Bearer ")) {

chain.doFilter(request, response);

return;

}

UsernamePasswordAuthenticationToken authentication = getAuthentication(header);

if (authentication != null) {

SecurityContextHolder.getContext().setAuthentication(authentication);

}

chain.doFilter(request, response);

LOGGER.info("End JWT filter");

}

private UsernamePasswordAuthenticationToken getAuthentication(String header) {

try {

String token = header.replace("Bearer ", "");

Jws<Claims> parsedToken = Jwts.parserBuilder()

.setSigningKey(Keys.hmacShaKeyFor(SECRET\_KEY.getBytes(StandardCharsets.UTF\_8)))

.build()

.parseClaimsJws(token);

String user = parsedToken.getBody().getSubject();

LOGGER.debug("User from token: {}", user);

if (user != null) {

return new UsernamePasswordAuthenticationToken(user, null, new ArrayList<>());

}

} catch (JwtException e) {

LOGGER.error("JWT token is invalid: {}", e.getMessage());

}

return null;

}

}

**SecurityConfig.java**

package com.cognizant.spring\_learn.security;

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.method.configuration.EnableMethodSecurity;

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

import org.springframework.security.core.userdetails.\*;

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

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

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration

@EnableMethodSecurity

public class SecurityConfig {

@Bean

public UserDetailsService 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() {

// ⚠️ Not for production

return NoOpPasswordEncoder.getInstance();

}

@Bean

public AuthenticationManager authenticationManager(AuthenticationConfiguration configuration)

throws Exception {

return configuration.getAuthenticationManager();

}

@Bean

public SecurityFilterChain filterChain(HttpSecurity http, AuthenticationManager authManager) throws Exception {

http

.csrf(csrf -> csrf.disable())

.httpBasic(basic -> {})

.authorizeHttpRequests(auth -> auth

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

.anyRequest().authenticated()

)

.addFilterBefore(new JwtAuthorizationFilter(authManager), UsernamePasswordAuthenticationFilter.class);

return http.build();

}

}

**Output:**

