**SecurityConfig.java**

package com.cognizant.spring\_learn.security;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Bean;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

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

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

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

import org.springframework.security.config.annotation.web.configuration.\*;

import com.cognizant.spring\_learn.security.JwtAuthorizationFilter;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

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

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication()

.withUser("admin").password(passwordEncoder().encode("pwd")).roles("ADMIN")

.and()

.withUser("user").password(passwordEncoder().encode("pwd")).roles("USER");

}

@Override

protected void configure(HttpSecurity httpSecurity) throws Exception {

httpSecurity.csrf().disable().httpBasic().and()

.authorizeRequests()

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

.anyRequest().authenticated()

.and()

.addFilter(new JwtAuthorizationFilter(authenticationManager()));

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

**JwtAuthorizationFilter.java**

package com.cognizant.spring\_learn.security;

import java.io.IOException;

import java.util.ArrayList;

import javax.servlet.\*;

import javax.servlet.http.\*;

import org.slf4j.\*;

import org.springframework.security.authentication.\*;

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

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

import io.jsonwebtoken.\*;

public class JwtAuthorizationFilter extends BasicAuthenticationFilter {

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

public JwtAuthorizationFilter(AuthenticationManager authManager) {

super(authManager);

}

@Override

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

throws IOException, ServletException {

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

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

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

chain.doFilter(request, response);

return;

}

UsernamePasswordAuthenticationToken authentication = getAuthentication(request);

SecurityContextHolder.getContext().setAuthentication(authentication);

chain.doFilter(request, response);

}

private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {

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

if (token != null) {

try {

Jws<Claims> claims = Jwts.parser()

.setSigningKey("secretkey")

.parseClaimsJws(token.replace("Bearer ", ""));

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

if (user != null) {

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

}

} catch (JwtException e) {

return null;

}

}

return null;

}

}

**AuthenticationController.java**

package com.cognizant.spring\_learn.controller;

import java.util.\*;

import java.util.Base64;

import org.slf4j.\*;

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

import io.jsonwebtoken.\*;

@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 authentication");

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

String user = getUser(authHeader);

String token = generateJwt(user);

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

map.put("token", token);

LOGGER.info("End authentication");

return map;

}

private String getUser(String authHeader) {

String encodedCredentials = authHeader.replace("Basic ", "");

byte[] decoded = Base64.getDecoder().decode(encodedCredentials);

String credentials = new String(decoded);

return credentials.split(":")[0]; // Get username

}

private String generateJwt(String user) {

JwtBuilder builder = Jwts.builder();

builder.setSubject(user);

builder.setIssuedAt(new Date());

builder.setExpiration(new Date(System.currentTimeMillis() + 1200000)); // 20 mins

builder.signWith(SignatureAlgorithm.HS256, "secretkey");

return builder.compact();

}

}