**5. JWT-handson**

**Create authentication service that returns JWT**

**Code:**

**Pom.XML**

<!-- Spring Security -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<!-- JWT Library -->

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.0</version>

</dependency>

**AuthenticationController**

package com.cognizant.springlearn.controller;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.JwtBuilder;

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

byte[] decodedBytes = Base64.getDecoder().decode(base64Credentials);

String decodedString = new String(decodedBytes);

String[] userDetails = decodedString.split(":");

return userDetails[0]; // Username

}

private String generateJwt(String user) {

JwtBuilder builder = Jwts.builder();

builder.setSubject(user);

builder.setIssuedAt(new Date());

builder.setExpiration(new Date(System.currentTimeMillis() + 20 \* 60 \* 1000)); // 20 min expiry

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

return builder.compact();

}

}

**SecurityConfig.java**

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.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

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.config.annotation.web.configuration.EnableWebSecurity;

@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");

}

@Bean

public PasswordEncoder passwordEncoder() {

LOGGER.info("Start - passwordEncoder()");

return new BCryptPasswordEncoder();

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable().httpBasic().and()

.authorizeRequests()

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

.anyRequest().authenticated()

.and()

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

}

@Bean

@Override

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

}

**JwtAuthorizationFilter**

package com.cognizant.springlearn.security;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jws;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.JwtException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

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.util.ArrayList;

public class JwtAuthorizationFilter extends BasicAuthenticationFilter {

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

public JwtAuthorizationFilter(AuthenticationManager authenticationManager) {

super(authenticationManager);

LOGGER.info("JwtAuthorizationFilter initialized.");

}

@Override

protected void doFilterInternal(HttpServletRequest req, HttpServletResponse res,

FilterChain chain) throws IOException, ServletException {

LOGGER.info("Start JWT filter");

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

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

chain.doFilter(req, res);

return;

}

UsernamePasswordAuthenticationToken authentication = getAuthentication(req);

SecurityContextHolder.getContext().setAuthentication(authentication);

chain.doFilter(req, res);

LOGGER.info("End JWT filter");

}

private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {

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

if (token != null) {

try {

Jws<Claims> jws = Jwts.parser()

.setSigningKey("secretkey")

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

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

if (user != null) {

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

}

} catch (JwtException ex) {

LOGGER.error("JWT validation failed", ex);

return null;

}

}

return null;

}

}

**Sample Response:**

[

{ "code": "IN", "name": "India" },

{ "code": "US", "name": "United States" }

]

**Invalid Token Response**

{

"timestamp": "...",

"status": 403,

"error": "Forbidden",

"message": "Forbidden"

}