JWT Hands on

Solution:

1. **Create Spring Boot Project**

* Use Spring Initializr or create a new Maven project
* Group ID: com.cognizant.springlearnjwt
* Artifact ID: jwt-auth-service
* Java version: 17

1. **Add Dependencies to pom.xml**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-api</artifactId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId>

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

</dependencies>

**Configure Application Properties**

**File:** src/main/resources/application.properties

spring.application.name=jwt-auth-service

server.port=8091 //update this to 8090

**Create Security Configuration**

**File:** src/main/java/com/cognizant/springlearnjwt/jwt\_auth\_service/security/SecurityConfig.java

package com.cognizant.springlearnjwt.jwt\_auth\_service.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.config.annotation.web.builders.HttpSecurity;

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

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

import org.springframework.security.web.SecurityFilterChain;

import com.cognizant.springlearnjwt.jwt\_auth\_service.security.CustomUserDetailsService;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

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

@Bean

public PasswordEncoder passwordEncoder() {

LOGGER.info("Start");

// For testing purposes, using a simple encoder

return org.springframework.security.crypto.password.NoOpPasswordEncoder.getInstance();

}

@Bean

public SecurityFilterChain filterChain(HttpSecurity httpSecurity, CustomUserDetailsService userDetailsService) throws Exception {

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

.httpBasic(httpBasic -> {})

.authorizeHttpRequests(auth -> auth

.requestMatchers("/authenticate").permitAll()

.requestMatchers("/countries").hasRole("USER")

.anyRequest().authenticated()

)

.userDetailsService(userDetailsService);

return httpSecurity.build();

}

@Bean

public CustomUserDetailsService userDetailsService() {

return new CustomUserDetailsService(passwordEncoder());

}

}

**Create Custom User Details Service**

**File:** src/main/java/com/cognizant/springlearnjwt/jwt\_auth\_service/security/CustomUserDetailsService.java

package com.cognizant.springlearnjwt.jwt\_auth\_service.security;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

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

import org.springframework.stereotype.Service;

import java.util.Arrays;

@Service

public class CustomUserDetailsService implements UserDetailsService {

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

private final PasswordEncoder passwordEncoder;

public CustomUserDetailsService(PasswordEncoder passwordEncoder) {

this.passwordEncoder = passwordEncoder;

LOGGER.info("CustomUserDetailsService initialized with password encoder: {}", passwordEncoder.getClass().getSimpleName());

}

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

LOGGER.info("Loading user by username: {}", username);

if ("admin".equals(username)) {

LOGGER.info("Creating admin user with password: pwd");

return new User("admin", "pwd",

Arrays.asList(new SimpleGrantedAuthority("ROLE\_ADMIN")));

} else if ("user".equals(username)) {

LOGGER.info("Creating user with password: pwd");

return new User("user", "pwd",

Arrays.asList(new SimpleGrantedAuthority("ROLE\_USER")));

} else {

LOGGER.error("User not found: {}", username);

throw new UsernameNotFoundException("User not found: " + username);

}

}

}

**Create JWT Token Utility**

**File:** src/main/java/com/cognizant/springlearnjwt/jwt\_auth\_service/security/JwtTokenUtil.java

package com.cognizant.springlearnjwt.jwt\_auth\_service.security;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import org.springframework.stereotype.Component;

import javax.crypto.SecretKey;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

@Component

public class JwtTokenUtil {

private static final SecretKey key = Keys.secretKeyFor(SignatureAlgorithm.HS512);

private static final int JWT\_TOKEN\_VALIDITY = 5 \* 60 \* 60; // 5 hours

public String getUsernameFromToken(String token) {

return getClaimFromToken(token, Claims::getSubject);

}

public Date getExpirationDateFromToken(String token) {

return getClaimFromToken(token, Claims::getExpiration);

}

public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {

final Claims claims = getAllClaimsFromToken(token);

return claimsResolver.apply(claims);

}

private Claims getAllClaimsFromToken(String token) {

return Jwts.parserBuilder()

.setSigningKey(key)

.build()

.parseClaimsJws(token)

.getBody();

}

private Boolean isTokenExpired(String token) {

final Date expiration = getExpirationDateFromToken(token);

return expiration.before(new Date());

}

public String generateToken(String username) {

Map<String, Object> claims = new HashMap<>();

return doGenerateToken(claims, username);

}

private String doGenerateToken(Map<String, Object> claims, String subject) {

return Jwts.builder()

.setClaims(claims)

.setSubject(subject)

.setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + JWT\_TOKEN\_VALIDITY \* 1000))

.signWith(key, SignatureAlgorithm.HS512)

.compact();

}

public Boolean validateToken(String token, String username) {

final String tokenUsername = getUsernameFromToken(token);

return (username.equals(tokenUsername) && !isTokenExpired(token));

}

}

**Create Authentication Controller**

**File:** src/main/java/com/cognizant/springlearnjwt/jwt\_auth\_service/controller/AuthenticationController.java

package com.cognizant.springlearnjwt.jwt\_auth\_service.controller;

import com.cognizant.springlearnjwt.jwt\_auth\_service.security.JwtTokenUtil;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestHeader;

import org.springframework.web.bind.annotation.RestController;

import java.util.Base64;

import java.util.HashMap;

import java.util.Map;

@RestController

public class AuthenticationController {

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

private final JwtTokenUtil jwtTokenUtil;

private final UserDetailsService userDetailsService;

private final PasswordEncoder passwordEncoder;

public AuthenticationController(JwtTokenUtil jwtTokenUtil,

UserDetailsService userDetailsService,

PasswordEncoder passwordEncoder) {

this.jwtTokenUtil = jwtTokenUtil;

this.userDetailsService = userDetailsService;

this.passwordEncoder = passwordEncoder;

}

@GetMapping("/authenticate")

public ResponseEntity<?> authenticate(@RequestHeader("Authorization") String authorizationHeader) {

LOGGER.info("Start - Authorization header: {}", authorizationHeader);

try {

// Extract username and password from Authorization header

String[] credentials = extractCredentials(authorizationHeader);

String username = credentials[0];

String password = credentials[1];

LOGGER.info("Extracted credentials - Username: {}, Password: {}", username, password);

// Load user details and verify password

UserDetails userDetails = userDetailsService.loadUserByUsername(username);

LOGGER.info("Loaded user details - Username: {}, Stored password: {}", userDetails.getUsername(), userDetails.getPassword());

boolean passwordMatches = passwordEncoder.matches(password, userDetails.getPassword());

LOGGER.info("Password match result: {}", passwordMatches);

if (!passwordMatches) {

LOGGER.error("Invalid credentials for user: {}", username);

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid credentials");

}

// Generate JWT token

final String token = jwtTokenUtil.generateToken(userDetails.getUsername());

LOGGER.info("Generated token: {}", token);

// Create response

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

response.put("token", token);

LOGGER.info("End - Success");

return ResponseEntity.ok(response);

} catch (UsernameNotFoundException e) {

LOGGER.error("User not found: {}", e.getMessage());

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid credentials");

} catch (Exception e) {

LOGGER.error("Authentication error: {}", e.getMessage(), e);

return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body("Authentication failed");

}

}

private String[] extractCredentials(String authorizationHeader) {

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

throw new IllegalArgumentException("Invalid Authorization header");

}

String encodedCredentials = authorizationHeader.substring(6);

String decodedCredentials = new String(Base64.getDecoder().decode(encodedCredentials));

String[] credentials = decodedCredentials.split(":");

if (credentials.length != 2) {

throw new IllegalArgumentException("Invalid credentials format");

}

return credentials;

}

}

**Create Country Controller**

**File:** src/main/java/com/cognizant/springlearnjwt/jwt\_auth\_service/controller/CountryController.jav

package com.cognizant.springlearnjwt.jwt\_auth\_service.controller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

import java.util.Arrays;

import java.util.List;

import java.util.Map;

@RestController

public class CountryController {

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

@GetMapping("/countries")

public List<Map<String, String>> getCountries() {

LOGGER.info("Start");

List<Map<String, String>> countries = Arrays.asList(

Map.of("code", "US", "name", "United States"),

Map.of("code", "DE", "name", "Germany"),

Map.of("code", "IN", "name", "India"),

Map.of("code", "JP", "name", "Japan")

);

LOGGER.info("End");

return countries;

}

}

Test the application’s end point

Output:

A computer screen with white text

AI-generated content may be incorrect.