**WEEK 4&5**

**EXERCISE 12**

**Online Bookstore - Securing RESTful Endpoints with Spring Security**

**Business Scenario**

The task is to secure the bookstore's RESTful endpoints using Spring Security with JWT-based authentication. This ensures that only authenticated and authorized users can access certain resources.

**Instructions**

**1. Add Spring Security:**

**Task:** Integrate Spring Security into your project.

**Implementation:**

* **Add Spring Security Dependency:**

**For Maven:**

<dependency>

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

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

</dependency>

**For Gradle:**

implementation 'org.springframework.boot:spring-boot-starter-security'

**Configure Basic Security Settings:**

Spring Security automatically secures your application with default settings. You can customize these settings in your security configuration class.

**2. JWT Authentication:**

**Task:** Implement JWT-based authentication and authorization.

**Implementation:**

1. **Add Dependencies for JWT:**

**For Maven:**

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

**For Gradle:**

implementation 'io.jsonwebtoken:jjwt:0.9.1'

2.**Create JWT Utility Class:**

**JwtUtil.java:**

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.stereotype.Component;

import java.util.Date;

@Component

public class JwtUtil {

private final String SECRET\_KEY = "your\_secret\_key";

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10)) // 10 hours

.signWith(SignatureAlgorithm.HS256, SECRET\_KEY)

.compact();

}

public Claims extractClaims(String token) {

return Jwts.parser()

.setSigningKey(SECRET\_KEY)

.parseClaimsJws(token)

.getBody();

}

public String extractUsername(String token) {

return extractClaims(token).getSubject();

}

public boolean isTokenExpired(String token) {

return extractClaims(token).getExpiration().before(new Date());

}

public boolean validateToken(String token, String username) {

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

}

}

**3.Create JWT Filter Class:**

**JwtRequestFilter.java:**

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

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

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

import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

public class JwtRequestFilter extends OncePerRequestFilter {

@Autowired

private JwtUtil jwtUtil;

@Autowired

private MyUserDetailsService myUserDetailsService;

@Override

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

throws ServletException, IOException {

final String authorizationHeader = request.getHeader("Authorization");

String username = null;

String jwtToken = null;

if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {

jwtToken = authorizationHeader.substring(7);

username = jwtUtil.extractUsername(jwtToken);

}

if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

UserDetails userDetails = myUserDetailsService.loadUserByUsername(username);

if (jwtUtil.validateToken(jwtToken, username)) {

}

}

chain.doFilter(request, response);

}

}

**4.Configure Spring Security:**

**WebSecurityConfig.java:**

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

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.builders.WebSecurity;

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

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

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

@Configuration

@EnableWebSecurity

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private JwtRequestFilter jwtRequestFilter;

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.antMatchers("/authenticate").permitAll() // Allow public access to authentication endpoint

.anyRequest().authenticated() // Secure all other endpoints

.and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS); // Stateless session

http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

}

}

**5.Create Authentication Controller:**

**AuthenticationController.java:**

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

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.AuthenticationException;

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

import javax.servlet.http.HttpServletResponse;

@RestController

public class AuthenticationController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private JwtUtil jwtUtil;

@PostMapping("/authenticate")

public ResponseEntity<?> createToken(@RequestBody AuthenticationRequest authRequest) throws Exception {

try {

Authentication authentication = authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(authRequest.getUsername(), authRequest.getPassword())

);

String token = jwtUtil.generateToken(authRequest.getUsername());

return ResponseEntity.ok(new AuthenticationResponse(token));

} catch (AuthenticationException e) {

throw new Exception("Invalid username/password", e);

}

}

}

**AuthenticationRequest.java:**

public class AuthenticationRequest {

private String username;

private String password;

}

**AuthenticationResponse.java:**

public class AuthenticationResponse {

private String jwtToken;

public AuthenticationResponse(String jwtToken) {

this.jwtToken = jwtToken;

}

// Getter

public String getJwtToken() {

return jwtToken;

}

}

**3. CORS Handling:**

**Task:** Configure CORS to handle cross-origin requests.

**Implementation:**

* **Configure CORS in WebSecurityConfig:**

**WebSecurityConfig.java:**

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.annotation.web.configuration.WebSecurityConfigurerAdapter;

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

import org.springframework.web.servlet.config.annotation.CorsRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

@EnableWebSecurity

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private JwtRequestFilter jwtRequestFilter;

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.antMatchers("/authenticate").permitAll() // Allow public access to authentication endpoint

.anyRequest().authenticated() // Secure all other endpoints

.and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS) // Stateless session

http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

http.cors(); // Enable CORS

}

@Bean

public WebMvcConfigurer corsConfigurer() {

return new WebMvcConfigurer() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*")

.allowedOrigins("\*")

.allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")

.allowedHeaders("\*");

}

};

}

}

**Explanation:**

* + **http.cors():** This enables CORS support in Spring Security.
  + **corsConfigurer():** This method configures CORS settings, allowing all origins, methods, and headers. You can customize this as needed to restrict origins or methods.

**Conclusion:**

With the Spring Security configuration and JWT-based authentication,the RESTful endpoints are now secured. The CORS configuration allows cross-origin requests as specified, which is crucial for enabling frontend applications hosted on different domains to interact with your backend.