Microservices

Exercise 1:

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.oauth2Login();

}

}

@RestController

public class UserController {

@GetMapping("/user")

public Principal user(Principal principal) {

return principal;

}

@GetMapping("/")

public String home() {

return "Welcome to the home page!";

}

}

Exercise 2:

@EnableWebSecurity

public class ResourceServerConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/public").permitAll()

.anyRequest().authenticated()

.and()

.oauth2ResourceServer()

.jwt();

}

}

@RestController

public class SecureController {

@GetMapping("/secure")

public String secure() {

return "This is a secure endpoint";

}

@GetMapping("/public")

public String publicEndpoint() {

return "This is a public endpoint";

}

}

Exercise 3:

@Configuration

public class JwtConfig {

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

private String secret;

@Value("${spring.security.jwt.expiration}")

private long expiration;

public String getSecret() {

return secret;

}

public long getExpiration() {

return expiration;

}

}

@Component

public class JwtTokenProvider {

@Autowired

private JwtConfig jwtConfig;

public String createToken(String username, List<String> roles) {

Claims claims = Jwts.claims().setSubject(username);

claims.put("roles", roles);

Date now = new Date();

Date validity = new Date(now.getTime() + jwtConfig.getExpiration());

return Jwts.builder()

.setClaims(claims)

.setIssuedAt(now)

.setExpiration(validity)

.signWith(SignatureAlgorithm.HS256, jwtConfig.getSecret())

.compact();

}

public boolean validateToken(String token) {

try {

Jwts.parser().setSigningKey(jwtConfig.getSecret()).parseClaimsJws(token);

return true;

} catch (Exception e) {

return false;

}

}

public Authentication getAuthentication(String token) {

Claims claims = Jwts.parser()

.setSigningKey(jwtConfig.getSecret())

.parseClaimsJws(token)

.getBody();

String username = claims.getSubject();

List<String> roles = claims.get("roles", List.class);

List<SimpleGrantedAuthority> authorities = roles.stream()

.map(SimpleGrantedAuthority::new)

.collect(Collectors.toList());

return new UsernamePasswordAuthenticationToken(username, "", authorities);

}

}

public class JwtTokenFilter extends OncePerRequestFilter {

@Autowired

private JwtTokenProvider jwtTokenProvider;

@Override

protected void doFilterInternal(HttpServletRequest request,

HttpServletResponse response,

FilterChain filterChain)

throws ServletException, IOException {

String token = resolveToken(request);

if (token != null && jwtTokenProvider.validateToken(token)) {

Authentication auth = jwtTokenProvider.getAuthentication(token);

SecurityContextHolder.getContext().setAuthentication(auth);

}

filterChain.doFilter(request, response);

}

private String resolveToken(HttpServletRequest request) {

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

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

return bearerToken.substring(7);

}

return null;

}

}

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private JwtTokenFilter jwtTokenFilter;

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and()

.authorizeRequests()

.antMatchers("/auth/login").permitAll()

.anyRequest().authenticated()

.and()

.addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class);

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

@RestController

@RequestMapping("/auth")

public class AuthController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private JwtTokenProvider jwtTokenProvider;

@Autowired

private UserDetailsService userDetailsService;

@PostMapping("/login")

public ResponseEntity<?> login(@RequestBody LoginRequest loginRequest) {

try {

String username = loginRequest.getUsername();

authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(username, loginRequest.getPassword()));

UserDetails userDetails = userDetailsService.loadUserByUsername(username);

List<String> roles = userDetails.getAuthorities().stream()

.map(GrantedAuthority::getAuthority)

.collect(Collectors.toList());

String token = jwtTokenProvider.createToken(username, roles);

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

} catch (AuthenticationException e) {

return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();

}

}

}

public class LoginRequest {

private String username;

private String password;

}

// JwtResponse.java

public class JwtResponse {

private String token;

public JwtResponse(String token) {

this.token = token;

}

}