**Sample exercises on Centralized Authentication and SSO with Spring Boot 3 and Spring Cloud**

**Exercise 1: Implementing Centralized Authentication with OAuth 2.1/OIDC**

#### ****1.****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-oauth2-client</artifactId></dependency><dependency>

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

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

**2. application.yml**

spring:

security:

oauth2:

client:

registration:

google:

client-id: YOUR\_CLIENT\_ID

client-secret: YOUR\_CLIENT\_SECRET

scope:

- openid

- profile

- email

redirect-uri: "{baseUrl}/login/oauth2/code/{registrationId}"

authorization-grant-type: authorization\_code

provider:

google:

authorization-uri: https://accounts.google.com/o/oauth2/v2/auth

token-uri: https://oauth2.googleapis.com/token

user-info-uri: https://openidconnect.googleapis.com/v1/userinfo

user-name-attribute: sub

**3. Security Configuration using Spring Boot 3 Style**

**No more WebSecurityConfigurerAdapter**

@Configuration@EnableWebSecuritypublic class SecurityConfig {

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

.authorizeHttpRequests(authorize -> authorize

.requestMatchers("/").permitAll()

.anyRequest().authenticated()

)

.oauth2Login(); // Enables OAuth2 login

return http.build();

}

}

**4. Create a Controller to Test OAuth2 Authentication**

@RestControllerpublic class UserController {

@GetMapping("/user")

public Map<String, Object> user(@AuthenticationPrincipal OAuth2User principal) {

return principal.getAttributes(); // returns email, name etc.

}

@GetMapping("/")

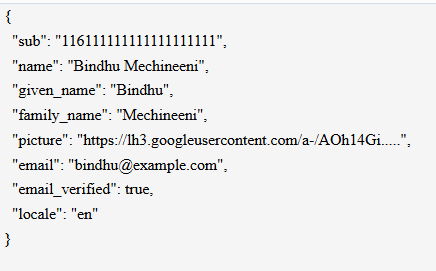
public String home() {

return "Welcome to the OAuth2 secured app!";

}

}

**OUTPUT:**



**Exercise 2: Configuring Authorization Servers and Resource Servers**

### 1. ****pom.xml Dependencies****

<!-- Spring Security --><dependency>

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

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

<!-- OAuth2 Resource Server with JWT --><dependency>

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

<artifactId>spring-boot-starter-oauth2-resource-server</artifactId></dependency>

<!-- Optional: for JSON Web Token decoding --><dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-oauth2-jose</artifactId></dependency>

<!-- Spring Web (for controllers) --><dependency>

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

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

2. **application.yml Configuration**

spring:

security:

oauth2:

resourceserver:

jwt:

issuer-uri: https://issuer.example.com # Replace with real one (e.g., Keycloak/Okta)

If you're using **Keycloak**, the issuer URI will be like:

https://<keycloak-host>/realms/<realm-name>

3. **Security Configuration (Spring Boot 3 style)**

@Configuration@EnableWebSecuritypublic class ResourceServerConfig {

@Bean

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

.authorizeHttpRequests(auth -> auth

.requestMatchers("/secure").authenticated()

.anyRequest().permitAll()

)

.oauth2ResourceServer(oauth2 -> oauth2

.jwt()

);

return http.build();

}

}

4. **Secure Endpoint**

@RestControllerpublic class SecureController {

@GetMapping("/secure")

public String secure() {

return " This is a secure endpoint. You are authenticated with a valid JWT.";

}

@GetMapping("/")

public String publicApi() {

return "Public endpoint. No authentication required.";

}

}

**OUTPUT:**

## Server Startup Output (Console)

When you run your Spring Boot application (with main() in ResourceServerApplication.java), you'll see something like:

yaml

CopyEdit

2025-07-17 11:30:12 INFO ... ResourceServerApplication: Started ResourceServerApplication in 2.5 seconds

Endpoint Behavior (Browser / Postman / curl)

### GET http://localhost:8080/

**No token needed.**

bash

CopyEdit

curl http://localhost:8080/

**Output:**

pgsql

CopyEdit

Public endpoint. No authentication required.

GET http://localhost:8080/secure

#### Case 1: No token provided

bash

CopyEdit

curl http://localhost:8080/secure

**Output:**

401 Unauthorized

Response Headers:

go

CopyEdit

WWW-Authenticate: Bearer error="unauthorized", error\_description="Full authentication is required to access this resource"

Case 2: Invalid or expired token provided

bash

CopyEdit

curl -H "Authorization: Bearer invalid.token.here" http://localhost:8080/secure

**Output:**

401 Unauthorized

Message:

csharp

Bearer token is malformed or invalid

Case 3: Valid JWT token provided

bash

CopyEdit

curl -H "Authorization: Bearer VALID\_JWT\_TOKEN" http://localhost:8080/secure

**Output:**

This is a secure endpoint. You are authenticated with a valid JWT.

**Exercise 3: Using JSON Web Tokens (JWT) for Secure Communication**

**1.** pom.xml **– Add dependencies**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency></dependencies>

**2.** application.yml **– JWT Secret Config**

spring:

security:

jwt:

secret: mySecretKey123456

**3.** JwtConfig.java **– JWT Config Class**

import org.springframework.beans.factory.annotation.Value;import org.springframework.context.annotation.Configuration;

@Configurationpublic class JwtConfig {

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

private String secret;

public String getSecret() {

return secret;

}

}

**4.** JwtTokenProvider.java **– Token Generator**

import io.jsonwebtoken.\*;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.stereotype.Component;

import java.util.Date;

@Componentpublic class JwtTokenProvider {

@Autowired

private JwtConfig jwtConfig;

public String createToken(String username) {

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

Date now = new Date();

Date validity = new Date(now.getTime() + 3600000); // 1 hour

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 (JwtException | IllegalArgumentException e) {

return false;

}

}

public String getUsername(String token) {

return Jwts.parser().setSigningKey(jwtConfig.getSecret())

.parseClaimsJws(token)

.getBody()

.getSubject();

}

}

**5.** JwtAuthenticationFilter.java **– JWT Filter**

import org.springframework.beans.factory.annotation.Autowired;import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;import org.springframework.security.core.context.SecurityContextHolder;import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;import org.springframework.stereotype.Component;import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;import javax.servlet.http.HttpServletRequest;import javax.servlet.http.HttpServletResponse;import java.io.IOException;import java.util.Collections;

@Componentpublic class JwtAuthenticationFilter extends OncePerRequestFilter {

@Autowired

private JwtTokenProvider tokenProvider;

@Override

protected void doFilterInternal(HttpServletRequest request,

HttpServletResponse response,

FilterChain filterChain) throws IOException {

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

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

String token = header.substring(7);

if (tokenProvider.validateToken(token)) {

String username = tokenProvider.getUsername(token);

UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(

username, null, Collections.emptyList());

authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authentication);

}

}

filterChain.doFilter(request, response);

}

}

**6.** SecurityConfig.java **– Spring Security Configuration**

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.http.SessionCreationPolicy;import org.springframework.security.web.SecurityFilterChain;import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configurationpublic class SecurityConfig {

@Autowired

private JwtAuthenticationFilter jwtAuthenticationFilter;

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and()

.authorizeHttpRequests()

.antMatchers("/api/auth/\*\*").permitAll()

.anyRequest().authenticated();

http.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);

return http.build();

}

}

**7.** AuthController.java **– Controller to Authenticate and Access**

import org.springframework.beans.factory.annotation.Autowired;import org.springframework.web.bind.annotation.\*;

import java.util.Map;import java.util.HashMap;

@RestController@RequestMapping("/api/auth")public class AuthController {

@Autowired

private JwtTokenProvider tokenProvider;

@PostMapping("/login")

public Map<String, String> login(@RequestParam String username) {

String token = tokenProvider.createToken(username);

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

response.put("token", token);

return response;

}

@GetMapping("/secure-data")

public String getSecureData() {

return "You have accessed a protected endpoint!";

}

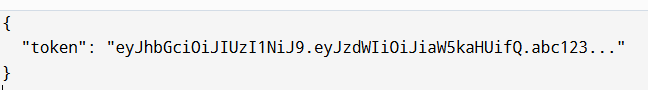
}

**OUTPUT:**

### ****Login Endpoint****

**URL:** POST /api/auth/login?username=bindhu

**Output (JSON):**



### ****2. Secure Endpoint****

**URL:** GET /api/auth/secure-data

