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

**Exercise 1: Implementing Centralized Authentication with OAuth 2.1/OIDC** Task: Implement centralized authentication using OAuth 2.1/OIDC in a Spring Boot application.

## Step-by-Step Explanation:

1. Add dependencies for Spring Security and OAuth2 Client in your `pom.xml`.
2. Configure OAuth2 client properties in `application.yml`.
3. Create a security configuration class to set up OAuth2 login.
4. Implement a controller to handle login and display user information.

## Solution Code:

\*\*pom.xml\*\*

<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>

\*\*application.yml\*\* spring:

security:

oauth2:

client:

registration:

my-client:

client-id: YOUR\_CLIENT\_ID

client-secret: YOUR\_CLIENT\_SECRET scope: openid, profile, email

authorization-grant-type: authorization\_code

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

my-provider:

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

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

\*\*SecurityConfig.java\*\* @EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter { @Override

protected void configure(HttpSecurity http) throws Exception { http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.oauth2Login();

}

}

\*\*UserController.java\*\* @RestController

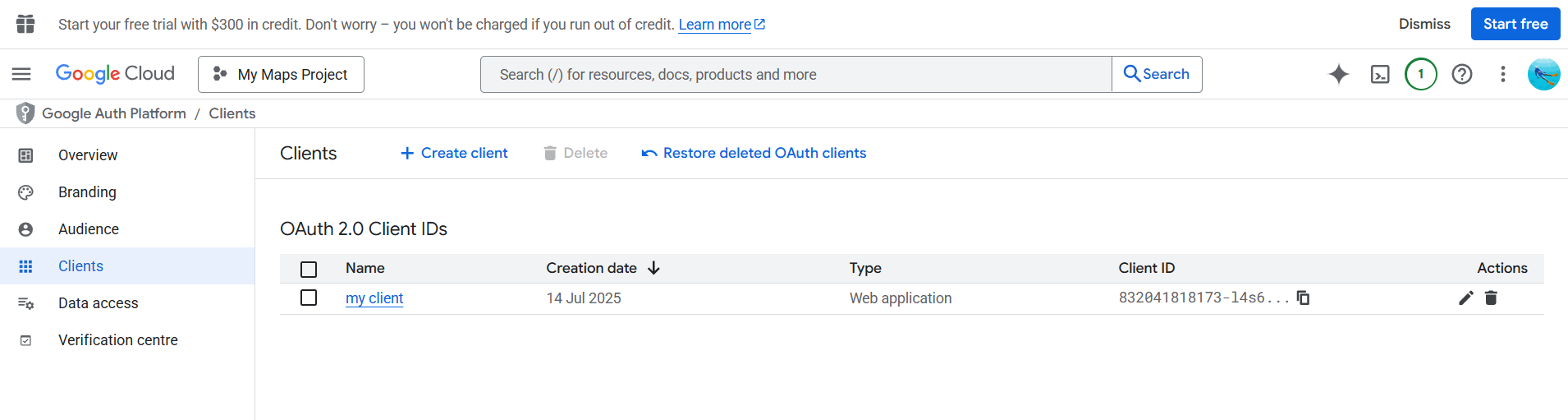
public class UserController { @GetMapping("/user")

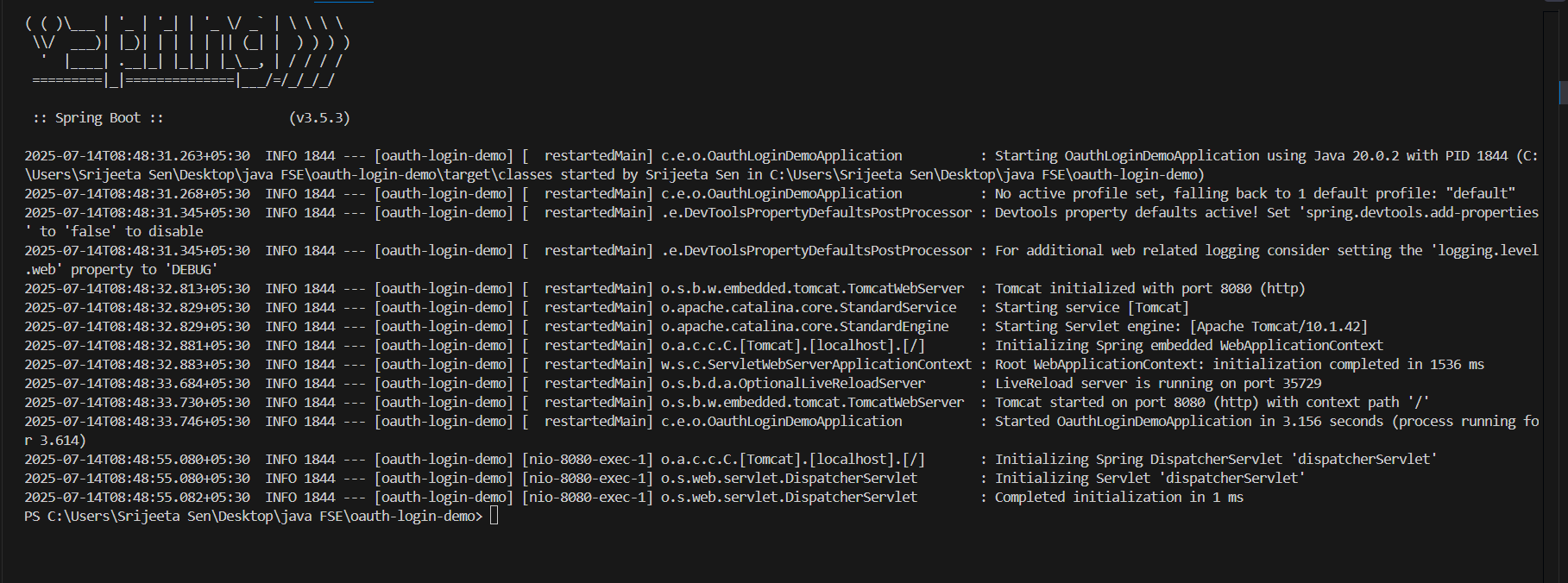
public Principal user(Principal principal) { return principal;

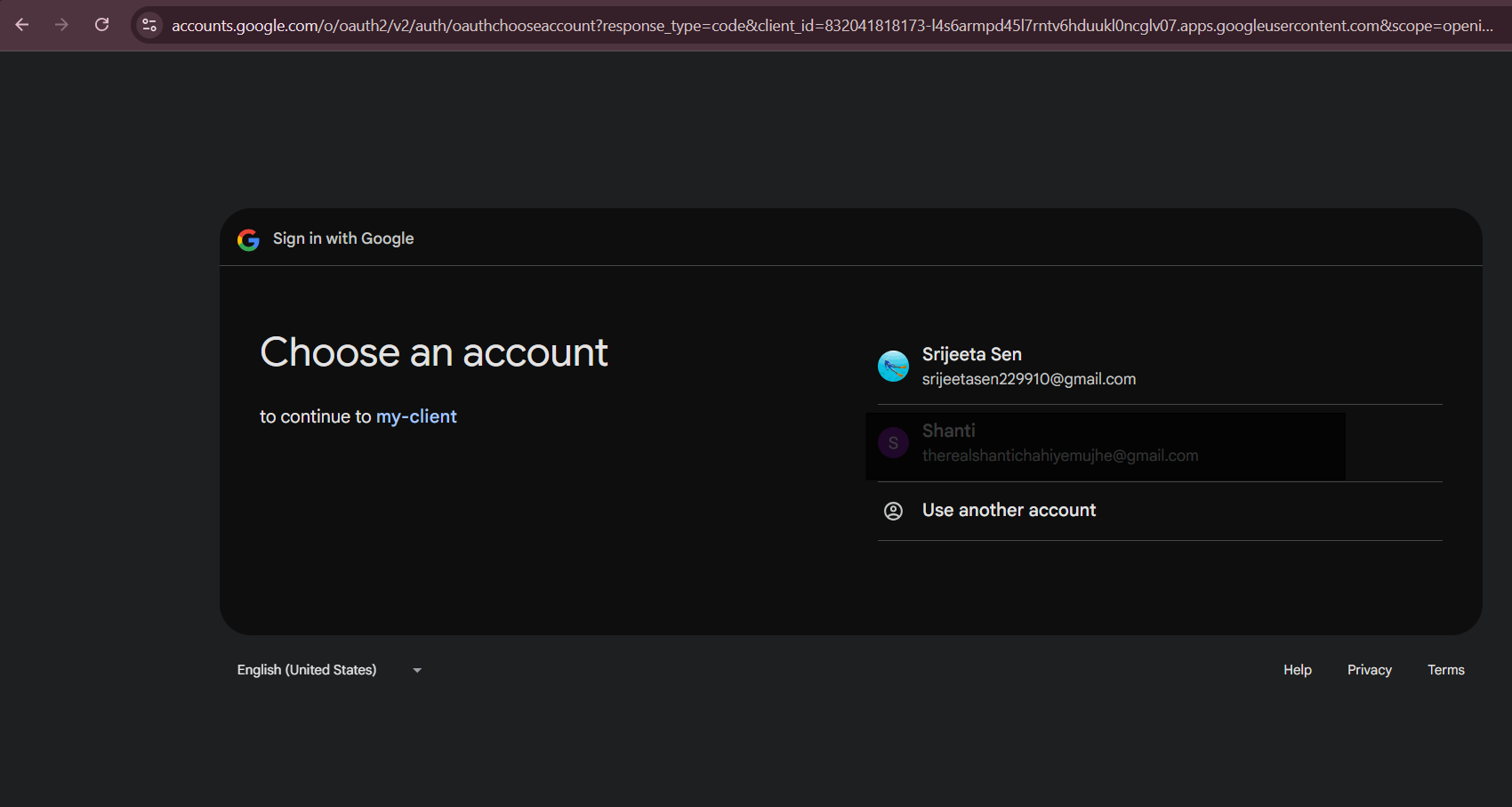
}

}

Output:







# Exercise 2: Configuring Authorization Servers and Resource Servers

Task: Configure Authorization Servers and Resource Servers in a Spring Boot application.

## Step-by-Step Explanation:

1. Add dependencies for Spring Security and OAuth2 Resource Server in your `pom.xml`.
2. Configure the Authorization Server properties in `application.yml`.
3. Create a security configuration class for the Resource Server.
4. Implement a controller to secure endpoints.

## Solution Code:

\*\*pom.xml\*\*

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

\*\*application.yml\*\* spring:

security:

oauth2:

resourceserver:

jwt:

issuer-uri: https://issuer.example.com

\*\*ResourceServerConfig.java\*\* @EnableWebSecurity

public class ResourceServerConfig extends WebSecurityConfigurerAdapter { @Override

protected void configure(HttpSecurity http) throws Exception { http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.oauth2ResourceServer()

.jwt();

}

}

\*\*SecureController.java\*\* @RestController

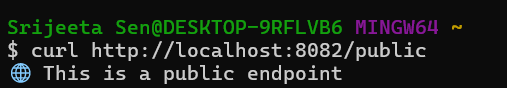
public class SecureController { @GetMapping("/secure") public String secure() {

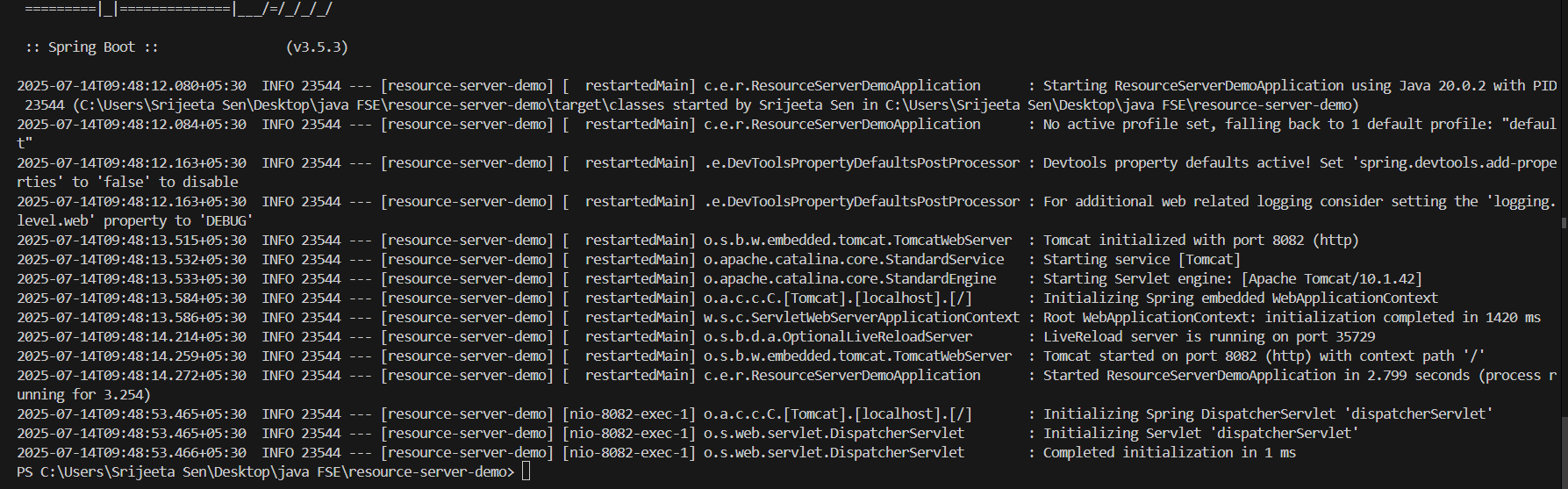
return "This is a secure endpoint";

}

}

Output:





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

Task: Use JSON Web Tokens (JWT) for secure communication in a Spring Boot application.

## Step-by-Step Explanation:

1. Add dependencies for Spring Security and JWT in your `pom.xml`.
2. Configure JWT properties in `application.yml`.
3. Create a security configuration class to set up JWT authentication.
4. Implement a controller to secure endpoints using JWT.

## Solution Code:

\*\*pom.xml\*\*

<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>

\*\*application.yml\*\* spring:

security:

jwt:

secret: YOUR\_SECRET\_KEY

\*\*JwtConfig.java\*\* @Configuration

public class JwtConfig { @Value("${spring.security.jwt.secret}") private String secret;

public String getSecret() { return secret;

}

}

\*\*JwtTokenProvider.java\*\* @Component

public 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 validity

return Jwts.builder()

.setClaims(claims)

.setIssuedAt(now)

.setExpiration(validity)

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

.compact();

}

}

\*\*JwtTokenFilter.java\*\*

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;

}

}

\*\*SecurityConfig.java\*\* @EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter { @Autowired

private JwtTokenFilter jwtTokenFilter;

@Override

protected void configure(HttpSecurity http) throws Exception { http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class);

}

}