Exercise 1: Centralized Authentication with OAuth 2.1/OIDC

**CentralizedAuthLoginApplication.java**

**package org.example.authlogin;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class CentralizedAuthLoginApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(CentralizedAuthLoginApplication.class, args);  
 }  
}**

**SecurityConfig.java**

**package org.example.authlogin.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.Customizer;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.web.SecurityFilterChain;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .authorizeHttpRequests(auth -> auth  
 .anyRequest().authenticated()  
 )  
 .oauth2Login(Customizer.*withDefaults*());  
  
 return http.build();  
 }  
}**

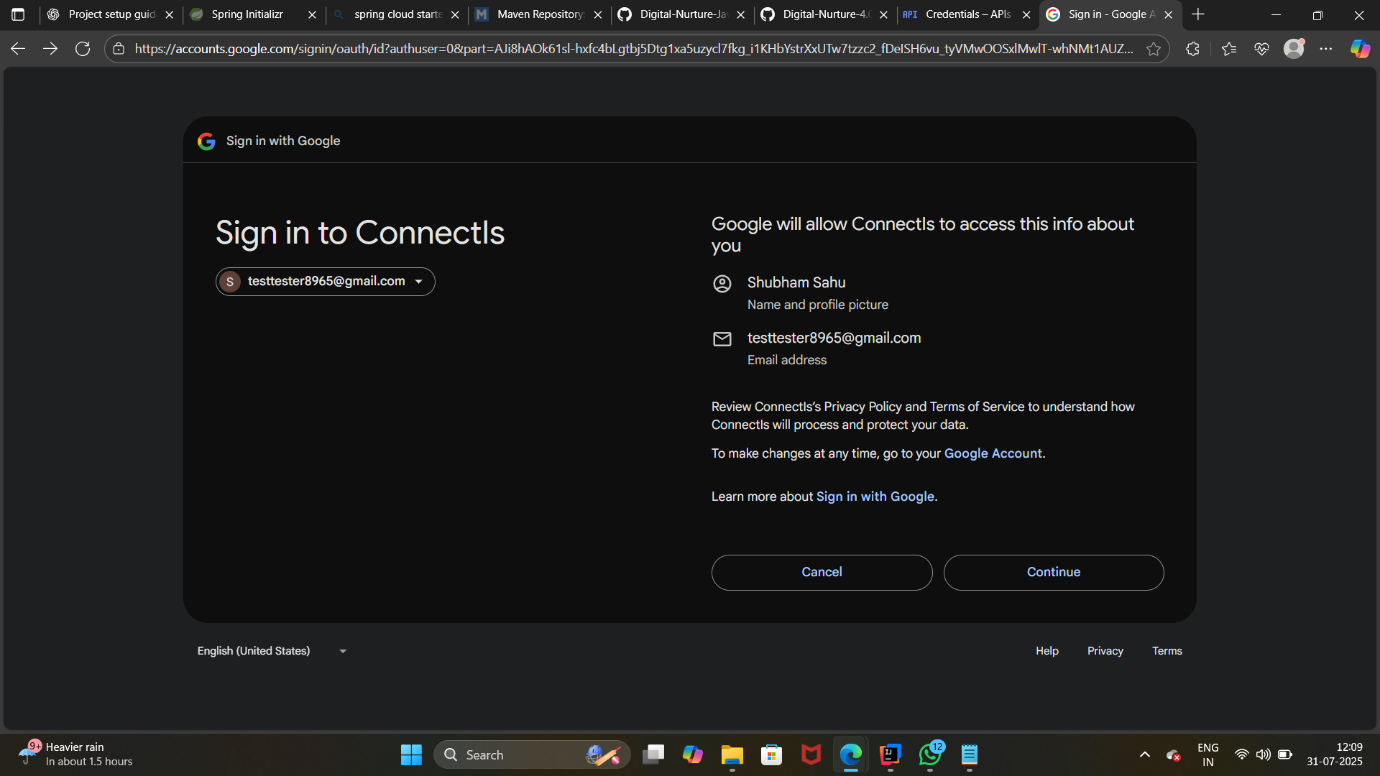
**UserController.java**

**package org.example.authlogin.controller;  
  
import org.springframework.security.core.Authentication;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class UserController {  
  
 @GetMapping("/user")  
 public String user(Authentication authentication) {  
 return "Logged in as: " + authentication.getName();  
 }  
}**

**application.yml  
server:  
 port: 8080  
  
spring:  
 security:  
 oauth2:  
 client:  
 registration:  
 google:  
 client-id:client\_id client-secret: client\_secret  
 scope: openid, profile, email  
 authorization-grant-type: authorization\_code  
 redirect-uri: "{baseUrl}/login/oauth2/code/{registrationId}"  
 provider:  
 google:  
 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**

**pom.xml**

**<dependencies>  
 <!-- 🔐 Security -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
  
 <!-- 🌐 OAuth2 Login Client -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-oauth2-client</artifactId>  
 </dependency>  
  
 <!-- 🌍 Web -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <!-- 🌱 Optional: Dev Tools -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
</dependencies>**

****

**Exercise 2: Authorization Server + Resource Server**

**Pom.xml**

**<dependencies>  
 <!-- ✅ Web -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <!-- ✅ Spring Security -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
  
 <!-- ✅ OAuth2 Resource Server (JWT) -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>  
 </dependency>  
</dependencies>**

**ResourceServerAuthApplication.java**

**package org.example.resourceserver;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class ResourceServerAuthApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(ResourceServerAuthApplication.class, args);  
 }  
}**

**SecurityConfig.java**

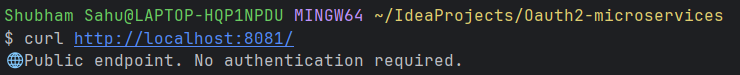
**package org.example.resourceserver.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.oauth2.jwt.JwtDecoder;  
import org.springframework.security.oauth2.jwt.NimbusJwtDecoder;  
import org.springframework.security.web.SecurityFilterChain;  
  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
  
 @Bean  
 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
 http  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/secure").authenticated()  
 .anyRequest().permitAll()  
 )  
 .oauth2ResourceServer(oauth2 -> oauth2  
 .jwt(jwt -> jwt.decoder(jwtDecoder())) // 👈 provide JwtDecoder  
 );  
  
 return http.build();  
 }  
  
 // 👇 Explicit JwtDecoder bean  
 @Bean  
 public JwtDecoder jwtDecoder() {  
 String jwkSetUri = "https://demo.duendesoftware.com/.well-known/openid-configuration/jwks";  
 return NimbusJwtDecoder.*withJwkSetUri*(jwkSetUri).build();  
 }  
}**

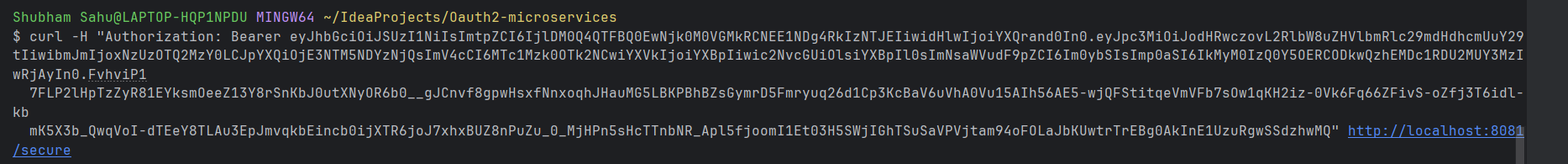
**SecureController.java**

**package org.example.resourceserver.controller;  
  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class SecureController {  
  
 @GetMapping("/secure")  
 public String secure() {  
 return "✅ You accessed a secure endpoint with a valid JWT.";  
 }  
  
 @GetMapping("/")  
 public String home() {  
 return "🌐 Public endpoint. No authentication required.";  
 }  
}**

**application.yml**

**server:  
 port: 8081  
  
spring:  
 security:  
 oauth2:  
 resourceserver:  
 jwt:  
 issuer-uri: https://demo.duendesoftware.com *# ✅ Test Issuer URI***

****

****

**Exercise 3: Using JSON Web Tokens (JWT) for Secure Communication in a standalone Spring Boot 3.2+ application with manual JWT issuance and validation.**

**Pom.xml**

**<dependencies>  
 <!-- Web and Security -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
  
 <!-- JWT library -->  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt</artifactId>  
 <version>0.9.1</version>  
 </dependency>  
</dependencies>**

**JwtAuthApplication.java**

**package org.example.jwtauth;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class JwtTokenAuthApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(JwtTokenAuthApplication.class, args);  
 }  
}**

**JwtConfig.java**

**package org.example.jwtauth.config;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.context.annotation.Configuration;  
  
@Configuration  
public class JwtConfig {  
 @Value("${spring.security.jwt.secret}")  
 private String secret;  
  
 public String getSecret() {  
 return secret;  
 }  
}**

**JwtTokenProvider.java**

**package org.example.jwtauth.jwt;  
  
import io.jsonwebtoken.\*;  
import org.example.jwtauth.config.JwtConfig;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.stereotype.Component;  
  
import java.util.Date;  
import java.util.List;  
  
@Component  
public class JwtTokenProvider {  
  
 private final JwtConfig config;  
  
 public JwtTokenProvider(JwtConfig config) {  
 this.config = config;  
 }  
  
 public String createToken(String username) {  
 Date now = new Date();  
 Date expiry = new Date(now.getTime() + 3600000); // 1 hour  
  
 return Jwts.*builder*()  
 .setSubject(username)  
 .setIssuedAt(now)  
 .setExpiration(expiry)  
 .signWith(SignatureAlgorithm.*HS256*, config.getSecret())  
 .compact();  
 }  
  
 public boolean validateToken(String token) {  
 try {  
 Jwts.*parser*().setSigningKey(config.getSecret()).parseClaimsJws(token);  
 return true;  
 } catch (JwtException e) {  
 return false;  
 }  
 }  
  
 public Authentication getAuthentication(String token) {  
 String username = Jwts.*parser*()  
 .setSigningKey(config.getSecret())  
 .parseClaimsJws(token)  
 .getBody()  
 .getSubject();  
 return new UsernamePasswordAuthenticationToken(username, "", List.*of*());  
 }  
}**

**SecurityConfig.java**

**package org.example.jwtauth.config;  
  
import org.example.jwtauth.jwt.JwtTokenFilter;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.web.SecurityFilterChain;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
  
@Configuration  
public class SecurityConfig {  
  
 private final JwtTokenFilter jwtTokenFilter;  
  
 public SecurityConfig(JwtTokenFilter jwtTokenFilter) {  
 this.jwtTokenFilter = jwtTokenFilter;  
 }  
  
 @Bean  
 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
 http.csrf(csrf -> csrf.disable())  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/token").permitAll()  
 .anyRequest().authenticated()  
 )  
 .addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class);  
  
 return http.build();  
 }  
}**

**AuthController.java**

**package org.example.jwtauth.controller;  
  
import org.example.jwtauth.jwt.JwtTokenProvider;  
import org.springframework.web.bind.annotation.\*;  
  
@RestController  
public class AuthController {  
  
 private final JwtTokenProvider jwtTokenProvider;  
  
 public AuthController(JwtTokenProvider jwtTokenProvider) {  
 this.jwtTokenProvider = jwtTokenProvider;  
 }  
  
 @PostMapping("/token")  
 public String token(@RequestParam String username) {  
 return jwtTokenProvider.createToken(username);  
 }  
  
 @GetMapping("/secure")  
 public String secure() {  
 return "✅ Accessed secure endpoint with valid JWT.";  
 }  
}**

**application.yml**

**server:  
 port: 8082  
  
spring:  
 security:  
 jwt:  
 secret: my-super-secret-key-12345**