**WEEK-5 HANDS-ON**

**MICROSERVICES**

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

**Source code:**

<!-- 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>

</dependencies>

# 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

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;

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.oauth2Login();

}

}

// UserController.java

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

import java.security.Principal;

@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.

**Source code:**

<!-- 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-resource-server</artifactId>

</dependency>

</dependencies>

# application.yml

spring:

security:

oauth2:

resourceserver:

jwt:

issuer-uri: https://issuer.example.com # Replace with actual issuer like https://accounts.google.com

// ResourceServerConfig.java

import org.springframework.context.annotation.Configuration;

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

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

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

@Configuration

@EnableWebSecurity

public class ResourceServerConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.oauth2ResourceServer()

.jwt();

}

}

// SecureController.java

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class SecureController {

@GetMapping("/secure")

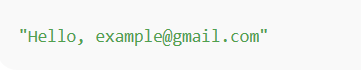
public String secure() {

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.

**Source code:**

<!-- pom.xml -->

<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>

# application.yml

spring:

security:

jwt:

secret: YOUR\_SECRET\_KEY

// JwtConfig.java

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

import io.jsonwebtoken.\*;

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

import org.springframework.stereotype.Component;

import java.util.Date;

@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

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 Authentication getAuthentication(String token) {

String username = Jwts.parser()

.setSigningKey(jwtConfig.getSecret())

.parseClaimsJws(token)

.getBody()

.getSubject();

return new UsernamePasswordAuthenticationToken(username, "", new ArrayList<>());

}

}

// JwtTokenFilter.java

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

import org.springframework.security.core.Authentication;

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

import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.IOException;

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

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

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.\*;

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

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private JwtTokenFilter jwtTokenFilter;

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.csrf().disable()

.authorizeRequests()

.anyRequest().authenticated()

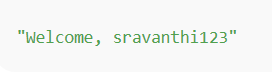
.and()

.addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class);

}

}

**Output**:



**Sample Hands-on Exercises on Edge Services and API Gateway with Spring Boot 3 and Spring Cloud**

**Exercise 1:** Implementing Edge Services for Routing and Filtering \*\*Task:\*\* Implement an edge service for routing and filtering requests in a microservices architecture using Spring Boot 3 and Spring Cloud.

**Source code:**

<!-- pom.xml -->

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-gateway</artifactId>

</dependency>

</dependencies>

# application.properties

spring.cloud.gateway.routes[0].id=example\_route

spring.cloud.gateway.routes[0].uri=http://example.org

spring.cloud.gateway.routes[0].predicates[0]=Path=/example/\*\*

// LoggingFilter.java

import org.springframework.cloud.gateway.filter.GlobalFilter;

import org.springframework.cloud.gateway.filter.GatewayFilterChain;

import org.springframework.stereotype.Component;

import org.springframework.web.server.ServerWebExchange;

import reactor.core.publisher.Mono;

@Component

public class LoggingFilter implements GlobalFilter {

@Override

public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {

System.out.println("Request URI: " + exchange.getRequest().getURI());

return chain.filter(exchange);

}

}

**Output:**



**Exercise 2**: Load Balancing in an API Gateway \*\*Task:\*\* Implement load balancing in an API Gateway using Spring Boot 3 and Spring Cloud.

**Source code:**

<!-- pom.xml -->

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-gateway</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-loadbalancer</artifactId>

</dependency>

</dependencies>

# application.properties

spring.cloud.gateway.routes[0].id=load\_balanced\_route

spring.cloud.gateway.routes[0].uri=lb://example-service

spring.cloud.gateway.routes[0].predicates[0]=Path=/loadbalanced/\*\*

// LoadBalancerConfiguration.java

import org.springframework.cloud.client.ServiceInstance;

import org.springframework.cloud.client.loadbalancer.\*;

import org.springframework.cloud.loadbalancer.core.\*;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.env.Environment;

@Configuration

public class LoadBalancerConfiguration {

@Bean

public ReactorLoadBalancer<ServiceInstance> randomLoadBalancer(

Environment environment,

LoadBalancerClientFactory loadBalancerClientFactory) {

String name = environment.getProperty(LoadBalancerClientFactory.PROPERTY\_NAME);

return new RandomLoadBalancer(

loadBalancerClientFactory.getLazyProvider(name, ServiceInstanceListSupplier.class),

name

);

}

}

Output:



A close-up of a text

AI-generated content may be incorrect.

**Exercise 3:** Resilience Patterns in an API Gateway \*\*Task:\*\* Implement resilience patterns in an API Gateway using Spring Boot 3 and Spring Cloud.

**Source code:**

<!-- pom.xml -->

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-gateway</artifactId>

</dependency>

<dependency>

<groupId>io.github.resilience4j</groupId>

<artifactId>resilience4j-spring-boot2</artifactId>

</dependency>

</dependencies>

# application.properties

resilience4j.circuitbreaker.instances.exampleCircuitBreaker.registerHealthIndicator=true

resilience4j.circuitbreaker.instances.exampleCircuitBreaker.slidingWindowSize=10

resilience4j.circuitbreaker.instances.exampleCircuitBreaker.failureRateThreshold=50

spring.cloud.gateway.routes[0].id=resilient\_route

spring.cloud.gateway.routes[0].uri=http://localhost:8081

spring.cloud.gateway.routes[0].predicates[0]=Path=/resilient/\*\*

spring.cloud.gateway.routes[0].filters[0]=name=CircuitBreaker

spring.cloud.gateway.routes[0].filters[0].args.name=exampleCircuitBreaker

spring.cloud.gateway.routes[0].filters[0].args.fallbackUri=forward:/fallback

// ResilienceConfiguration.java

import io.github.resilience4j.circuitbreaker.CircuitBreakerConfig;

import io.github.resilience4j.timelimiter.TimeLimiterConfig;

import org.springframework.cloud.client.circuitbreaker.Customizer;

import org.springframework.cloud.circuitbreaker.resilience4j.ReactiveResilience4JCircuitBreakerFactory;

import org.springframework.cloud.circuitbreaker.resilience4j.Resilience4JConfigBuilder;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class ResilienceConfiguration {

@Bean

public Customizer<ReactiveResilience4JCircuitBreakerFactory> defaultCustomizer() {

return factory -> factory.configureDefault(id -> new Resilience4JConfigBuilder(id)

.circuitBreakerConfig(CircuitBreakerConfig.ofDefaults())

.timeLimiterConfig(TimeLimiterConfig.ofDefaults())

.build());

}

}

// FallbackController.java

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class FallbackController {

@GetMapping("/fallback")

public ResponseEntity<String> fallback() {

return ResponseEntity.ok("Fallback response: Service is temporarily unavailable.");

}

}

**Output:**

Request:

A black text with a white background

AI-generated content may be incorrect.

Response:

A close up of a word

AI-generated content may be incorrect.

**Exercises on Microservices with Spring Boot 3.0**

1.Build a User and Order Management System  
**source code:**

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId> <!-- or postgresql -->

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

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

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

</dependency>

@Entity

public class User {

@Id @GeneratedValue

private Long id;

private String name;

private String email;

}

@Entity

public class Order {

@Id @GeneratedValue

private Long id;

private Long userId;

private String product;

private Integer quantity;

}

@Autowired

private WebClient.Builder webClientBuilder;

public User getUser(Long userId) {

return webClientBuilder.build()

.get()

.uri("http://localhost:8081/users/" + userId)

.retrieve()

.bodyToMono(User.class)

.block();

}

spring:

datasource:

url: jdbc:mysql://localhost:3306/userdb

username: root

password: yourpass

jpa:

hibernate:

ddl-auto: update

show-sql: true

**output:**



**2. Inventory Management System with Service Discovery  
source code:**

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

@EnableConfigServer

@SpringBootApplication

public class ConfigServerApplication {

public static void main(String[] args) {

SpringApplication.run(ConfigServerApplication.class, args);

}

}

server:

port: 8888

spring:

cloud:

config:

server:

git:

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

server:

port: 8761

eureka:

client:

register-with-eureka: false

fetch-registry: false

spring:

application:

name: product-service

datasource:

url: jdbc:mysql://localhost:3306/productdb

username: root

password: your\_password

jpa:

hibernate:

ddl-auto: update

eureka:

client:

service-url:

defaultZone: <http://localhost:8761/eureka>

@EnableDiscoveryClient

@SpringBootApplication

public class ProductServiceApplication {

public static void main(String[] args) {

SpringApplication.run(ProductServiceApplication.class, args);

}

}

@Entity

@Data

public class Product {

@Id @GeneratedValue

private Long id;

private String name;

private String description;

private Integer initialStock;

}

@RestController

@RequestMapping("/products")

public class ProductController {

@Autowired private ProductRepository repo;

@PostMapping

public Product add(@RequestBody Product product) {

return repo.save(product);

}

@GetMapping

public List<Product> list() {

return repo.findAll();

}

}

@EnableDiscoveryClient

@SpringBootApplication

public class InventoryServiceApplication {

public static void main(String[] args) {

SpringApplication.run(InventoryServiceApplication.class, args);

}

}

@Entity

@Data

public class Inventory {

@Id

private Long productId;

private Integer quantityAvailable;

}

@RestController

@RequestMapping("/inventory")

public class InventoryController {

@Autowired private InventoryRepository repo;

@GetMapping("/{productId}")

public Inventory checkStock(@PathVariable Long productId) {

return repo.findById(productId).orElse(new Inventory(productId, 0));

}

@PutMapping("/{productId}")

public Inventory updateStock(@PathVariable Long productId, @RequestParam int qty) {

Inventory inv = repo.findById(productId).orElse(new Inventory(productId, 0));

inv.setQuantityAvailable(qty);

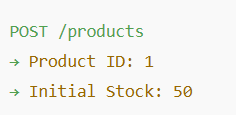
return repo.save(inv);

}

}

**Output:**

**Add Product:**

****

**Check Inventory:**

**A close-up of words

AI-generated content may be incorrect.**

**Update Inventory:**

**A close up of black text

AI-generated content may be incorrect.**