Cognizant Digital Nurture 4.0 – Java FSE

Week 5 – Hands-On Report

Name: Shaik Sulthan - 6431169
Track: Deep Skilling – Java Full Stack Engineer
Batch: 2025

1. Build a User and Order Management System Problem:

Create two microservices:

User Service to manage users

.• Order Service to manage orders placed by users.

• Requirements:

Use REST APIs.

• Communicate between services using WebClient (Spring WebFlux) or

• OpenFeign. Store data in MySQL or PostgreSQL.•

Pox.xml:

```
<dependencies>
   <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>com.mysql</groupId>
      <artifactId>mysql-connector-j</artifactId>
   </dependency>
   <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
   </dependency>
</dependencies>
```

USER.java:

```
package com.example.userservice.entity;

import jakarta.persistence.*;
import lombok.Data;

@Entity
```

```
@Data
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
}
```

Usercontroller.java

```
package com.example.userservice.controller;

import com.example.userservice.entity.User;
import com.example.userservice.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private UserRepository userRepository;

    @PostMapping
    public User createUser(@RequestBody User user) {
        return userRepository.save(user);
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> getUser(@PathVariable Long id) {
        return userRepository.findById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }
}
```

application.properties:

```
server.port=8081
spring.datasource.url=jdbc:mysql://localhost:3306/user_db
spring.datasource.username=root
spring.datasource.password=vamsi
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

data.sql:

CREATE DATABASE user_db;

INSERT INTO user (name, email) VALUES ('Shaik Sulthan', 'sulthan@example.com');

OUTPUT:

http://localhost:8081/users

{

"id": 1,

"name": "Shaik sulthan",

"email": "sulthan@example.com" }

Inventory Management System with Service Discovery

Problem

Create: Product Service:

Manage products and stock.

Inventory Service:

Track stock levels for each product.

- Requirements: Use Spring Cloud Netflix Eureka for service discovery.
- Implement centralized configuration using Spring Cloud Config Server.

Inventory Service:

Productservice application:

```
package com.example.productservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class ProductServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(ProductServiceApplication.class, args);
    }
}
```

Product.java:

```
package com.example.productservice.entity;

import jakarta.persistence.*;
        import lombok.Data;

@Entity
@Data
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private int quantity;
}
```

Product reposetri:

```
package com.example.productservice.repository;

import com.example.productservice.entity.Product;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductRepository extends JpaRepository<Product, Long> {}
```

Productcontroller.java:

```
package com.example.productservice.controller;

import com.example.productservice.entity.Product;
import com.example.productservice.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

        import java.util.List;

@RestController
@RequestMapping("/products")
public class ProductController {

    @Autowired
    private ProductRepository productRepository;

    @PostMapping
    public Product addProduct(@RequestBody Product product) {
        return productRepository.save(product);
    }

    @GetMapping
    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }
}
```

Inventory.java:

```
package com.example.inventoryservice.entity;

import jakarta.persistence.*;
        import lombok.Data;

@Entity
@Data
public class Inventory {
    @Id
    private Long productId;
    private int stock;
}
```

SQL:

INSERT INTO inventory (product_id, stock) VALUES (1, 10);

OUTPUT:

http://localhost:8082/inventory/1


```
{
  "productId": 1,
  "stock": 10
}
```

3.Implement an API Gateway

Problem: Create an API Gateway to route requests to:

Customer Service

- Billing Service

Requirements:
Use Spring Cloud Gateway
.• Implement rate limiting, caching, and path rewriting.
Apigatewayapplication.java:

```
package com.example.apigateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.gateway.route.RouteLocator;
import org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class ApiGatewayApplication {
    public static void main(String[] args) {
        SpringApplication.run(ApiGatewayApplication.class, args);
    }

    @Bean
    public RouteLocator customRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
                .route("customer-service", r -> r.path("/customer/**")
                        .filters(f -> f.rewritePath("/customer/(?<segment>.*)", "/${segment}")
                                .requestRateLimiter(config -> {
                                    config.setRateLimiter(redisRateLimiter -> {
                                        redisRateLimiter.setBurstCapacity(5);
                                        redisRateLimiter.setReplenishRate(1);
                                    });
                                }))
                        .uri("lb://CUSTOMER-SERVICE"))

                .route("billing-service", r -> r.path("/billing/**")
                        .filters(f -> f.rewritePath("/billing/(?<segment>.*)", "/${segment}")
                                .circuitBreaker(c -> c.setName("billingCB")
                                        .setFallbackUri("forward:/fallback/billing")))
                        .uri("lb://BILLING-SERVICE"))
                .build();
    }
}
```

OUTPUT:
http://localhost:9090/billing/api/invoices/1001
"Billing Service is currently unavailable. Please try again later."
http://localhost:9090/fallback/billing
Billing Service is currently unavailable. Please try again later.

http://CUSTOMER-SERVICE/api/users/1


```
{
  "id": 1,
  "name": "Shaik sulthan",
  "email": "sulthan@example.com"
}
```

4. Resilient Microservices with Circuit Breaker

Problem: A Payment Service calls a slow third-party API.

Requirements:

Implement Circuit Breaker and fallback logic using Resilience4j.

Log and monitor fallback events.

Paymetserviceapplication.java:

```
package com.example.paymentservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class PaymentServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(PaymentServiceApplication.class, args);
    }
}
```

Paymentcontroller.java:

```
package com.example.paymentservice.controller;

import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
@RequestMapping("/payment")
public class PaymentController {

    private static final Logger logger = LoggerFactory.getLogger(PaymentController.class);
    private static final String CB_NAME = "paymentCB";

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping("/process/{id}")
    @CircuitBreaker(name = CB_NAME, fallbackMethod = "paymentFallback")
    public String processPayment(@PathVariable String id) {
        logger.info("Calling external API for payment id: {}", id);
        // Simulating slow external call
        return restTemplate.getForObject("http://slow-api.com/payments/" + id, String.class);
    }

    public String paymentFallback(String id, Throwable ex) {
        logger.warn("Fallback triggered for payment id: {} - Reason: {}", id, ex.getMessage());
        return "Fallback: Payment service is currently unavailable for ID: " + id;
    }
}
```

appconfig.java:

```java
package com.example.paymentservice.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class AppConfig {
    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

OUTPUT:

http://localhost:8083/payment/process/101

{

"status": "success",

"paymentId": "101",

"amount": "₹1500",

"statusMessage": "Payment processed successfully"

}

http://localhost:8083/payment/process/101

Fallback: Payment service is currently unavailable for ID: 101