Table of Contents

[Project 1: Product Service Spring boot application 4](#_Toc134693897)

[Step 1: create the Spring boot application with details: 4](#_Toc134693898)

[Step 2: Add the Dependencies 5](#_Toc134693899)

[Step 3: Configuration in application.yaml: 5](#_Toc134693900)

[Database Configuration: 5](#_Toc134693901)

[microservices Integration in application.yaml file 5](#_Toc134693902)

[Step 4: add @EnableEurekaClient at main class 5](#_Toc134693903)

[step 5: Create the Packages 6](#_Toc134693904)

[Step 6: Create the Entity classes 6](#_Toc134693905)

[Step 7: Create the Model classes 7](#_Toc134693906)

[Step 8: Create the Repository layer 8](#_Toc134693907)

[Step 9: Create the Rest Controller 8](#_Toc134693908)

[Step 10: Create the Service layer 9](#_Toc134693909)

[Step 11: Create the Custom Exception Class 10](#_Toc134693910)

[Step 12: Create the Exception Handler Class(@ControllerAdvice) 11](#_Toc134693911)

[========================================================= 11](#_Toc134693912)

[Project 2: Service Registry Spring boot application 11](#_Toc134693913)

[Step 1: create the Spring boot application with below details: 11](#_Toc134693914)

[Step 2: Add the Dependencies 12](#_Toc134693915)

[Step 3: Eureka server details added into application.yaml file 12](#_Toc134693916)

[Step 4: add @EnableEurekaServer at main class 12](#_Toc134693917)

[========================================================= 13](#_Toc134693918)

[Project 3: Order Service Spring boot application 13](#_Toc134693919)

[Step 1: create the Spring boot application with details: 13](#_Toc134693920)

[Step 2: Add the Dependencies 13](#_Toc134693921)

[Step 3: Configuration in application.yaml: 13](#_Toc134693922)

[Database Configuration: 13](#_Toc134693923)

[microservices Integration in application.yaml file 13](#_Toc134693924)

[Step 4: add @EnableEurekaClient at main class 14](#_Toc134693925)

[step 5: Create the Packages 14](#_Toc134693926)

[Step 6: Create the Entity classes 14](#_Toc134693927)

[Step 7: Create the Model classes 15](#_Toc134693928)

[Step 8: Create the Repository layer 16](#_Toc134693929)

[Step 9: Create the Service layer 17](#_Toc134693930)

[Step 10: Create the Rest Controller 18](#_Toc134693931)

[========================================================= 18](#_Toc134693932)

[Project 4: Config Server Spring boot application 18](#_Toc134693933)

[Step 1: create the Spring boot application with details: 18](#_Toc134693934)

[Step 2: Add the Dependencies 19](#_Toc134693935)

[Step 3: Add git hub configuration in application.yaml file 19](#_Toc134693936)

[Step 4: Remove the eureka configuration in OrderService and ProductService projects in application.yaml file 19](#_Toc134693937)

[Step 5: Add the Config client dependency into ProductService and OrderService projects in pom.xml file 20](#_Toc134693938)

[Step 6: add the cloud server configuration to ProductService and OrderService Projects in Application.yaml 20](#_Toc134693939)

[======================================================== 20](#_Toc134693940)

[Update Project 1: Reduce Quantity API implementation in Production Service 20](#_Toc134693941)

[Step 1 : add the Api for Reduce Quantity Service in ProductController 20](#_Toc134693942)

[Step 2: Add into Service Layer 21](#_Toc134693943)

[ProductService 21](#_Toc134693944)

[ProductServiceImpl 21](#_Toc134693945)

[======================================================== 21](#_Toc134693946)

[Update Project 2: Feign Client for ProductService configured into OrderService Project 21](#_Toc134693947)

[Step1:Add the Dependency OpenFeign 21](#_Toc134693948)

[Step 2: Enable Feign Client 22](#_Toc134693949)

[Step 3: Add the External Client Interface(ProductService) 22](#_Toc134693950)

[Step 4: update OrderService to reduce quantity for product 22](#_Toc134693951)

[======================================================== 23](#_Toc134693952)

[Update Project 2: Exception handling into OrderService Project 23](#_Toc134693953)

[Step 1: add ErrorResponse class 23](#_Toc134693954)

[Step 2: add CustomOrderException class 24](#_Toc134693955)

[Step 3: Add the Exception Handler Class 24](#_Toc134693956)

[Step 4 : Add ErrorDecoder class 25](#_Toc134693957)

[Step 5: Configure ErrorDecoder 26](#_Toc134693958)

[=========================================================== 26](#_Toc134693959)

[Software: Install Docker and Zipkin via Docker 26](#_Toc134693960)

[Step 1: install the Docker 26](#_Toc134693961)

[Step 2 :install the Zipkin 26](#_Toc134693962)

[=========================================================== 27](#_Toc134693963)

[Project 1,2 : add the Dependencies for Zipkin and Slueth 27](#_Toc134693964)

[=========================================================== 27](#_Toc134693965)

[Project 5: Payment Service Spring boot application 27](#_Toc134693966)

[Step 1: create the Spring boot application with details: 27](#_Toc134693967)

[Step 2: Add the Dependencies 27](#_Toc134693968)

[Step 3: Configuration in application.yaml: 28](#_Toc134693969)

[Step 4: add @EnableEurekaClient at main class 28](#_Toc134693970)

[step 5: Create the Packages 29](#_Toc134693971)

[Step 6: Create the Entity classes 29](#_Toc134693972)

[Step 7: Create the Model classes 30](#_Toc134693973)

[Step 8: Create the Repository layer 31](#_Toc134693974)

[Step 9: Create the Service layer 31](#_Toc134693975)

[Step 10: Create the Rest Controller 32](#_Toc134693976)

[========================================================= 33](#_Toc134693977)

[Update Project 2: Feign Client for PaymentService configured into OrderService Project 33](#_Toc134693978)

[Step 1: Add the External Client Interface(PaymentService) 33](#_Toc134693979)

[Step 4: Add PaymentRequest into Order Project 33](#_Toc134693980)

[Step 5: update OrderService to payment details sent to payment db 34](#_Toc134693981)

[============================================================== 35](#_Toc134693982)

[Update Project 2: Implementing GetOrder Details in Order Service Project 35](#_Toc134693983)

[Step 1: Add the OrderResponse class to send order Details 35](#_Toc134693984)

[Step 2: to get Order Details from OrderController class and OrderService ,OrderServiceImpl classes 36](#_Toc134693985)

[============================================================== 39](#_Toc134693986)

[Update Project 2: Fetching Product Data for getOrder Details API 39](#_Toc134693987)

[Step 1: Add the restTemplate to fetch the Product Details from ProductService project 39](#_Toc134693988)

[Step 2: create the ProductResponse Class 40](#_Toc134693989)

[Step 3: Update the OrderResponse class to add the ProductDetails 40](#_Toc134693990)

[Step 4: Update the OrderServiceImpl class to add the ProductDetails 41](#_Toc134693991)

[============================================================== 44](#_Toc134693992)

[Update Project 2, 5: Fetching Payment Details for getOrder Details API 44](#_Toc134693993)

[Step 1: Add the PaymentResponse into PaymentService Project 44](#_Toc134693994)

[Step 2: get PaymentResponse from PaymentController based on orderId in PaymentService Project 44](#_Toc134693995)

[Step 3: get PaymentResponse from PaymentService, PaymentServiceImpl based on orderid in PaymentService Project 45](#_Toc134693996)

[Step 4: get PaymentResponse from PaymentRepository based on orderid in PaymentService Project 47](#_Toc134693997)

[Step 5: create the PaymentResponse Class in OrderService Project 47](#_Toc134693998)

[Step 6: Update the OrderResponse class to add the PaymentDetails 48](#_Toc134693999)

[Step 7: Update the OrderServiceImpl class to add the PaymentDetails 49](#_Toc134694000)

[============================================================== 52](#_Toc134694001)

[Project 6: API Gateway Service Spring boot application 52](#_Toc134694002)

[Step 1: create the Spring boot application with details: 52](#_Toc134694003)

[Step 2: Add the Dependencies 52](#_Toc134694004)

[Step 3: Configuration in application.yaml: 53](#_Toc134694005)

[Step 4: add @EnableEurekaClient at main class 53](#_Toc134694006)

[============================================================ 54](#_Toc134694007)

[Update project 4: Implement Circuit Breaker in API GateWay project 54](#_Toc134694008)

[Step 1: Add the Dependencies for Circuit Breaker (with Reactor) 54](#_Toc134694009)

[Step 2: Add the Configuration for Circuit Breaker in Application.yaml 55](#_Toc134694010)

[Step 3: Add the Filter into main class for Circuit Breaker 55](#_Toc134694011)

[Step 4: Add the FallBackController for FallBack Methods 56](#_Toc134694012)

[============================================================ 57](#_Toc134694013)

[Update project 4: Implement Circuit Breaker into Order Service project 57](#_Toc134694014)

[Step 1: Add the Dependencies for Circuit Breaker (with out reactor) 57](#_Toc134694015)

[Step 2: Add the Circuit Breaker added into PaymentService 57](#_Toc134694016)

[Step 3: Add the Circuit Breaker added into ProductService 58](#_Toc134694017)

[Step 4: Add the resilience4j circuit breaker configured into application.yaml 59](#_Toc134694018)

# Project 1: Product Service Spring boot application

## Step 1: create the Spring boot application with details:

* **Project Name:** ProductService
* **Project Type:**Maven
* **Choose jdk:** java 8
* **Package name:**com.example.demo

## Step 2: Add the Dependencies

* **Spring boot version :** 2.7.11
* **Dependencies**
  + **Spring boot starter Web**
  + **Spring boot starter Data jpa**
  + **Spring boot starter DevTools**
  + **Spring boot starter Cloud Bootstrap**
  + **mySQl Driver**
  + **lombok**

For Microservices 🡺 add the Dependencies

* **Spring boot starter Eureka Discovery Client**

## Step 3: Configuration in application.yaml:

Note :Change the application.properties into application.yaml file and add the database configuration details

### Database Configuration:

spring:

datasource:

url: jdbc:mysql://${DB\_HOST:localhost}:3306/productdb

username: root

password:

driver-class-name: com.mysql.cj.jdbc.Driver

jpa:

database-platform: org.hibernate.dialect.MySQL57InnoDBDialect

hibernate:

ddl-auto: update

application:

name: PRODUCT-SERVICE

### microservices Integration in application.yaml file

server:

port: 8080

eureka:

instance:

prefer-ip-address: true

client:

fetch-registry: true

register-with-eureka: true

service-url:

defaultZone: ${EUREKA\_SERVER\_ADDRESS:http://localhost:8761/eureka}

## Step 4: add @EnableEurekaClient at main class

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

import com.example.demo.entity.Product;

@SpringBootApplication

**@EnableEurekaClient**

public class ProductServiceApplication {

public static void main(String[] args) {

SpringApplication.run(ProductServiceApplication.class, args);

}

}

## step 5: Create the Packages

* com.example.demo.controller
* com.example.demo.service
* com.example.demo.repository
* com.example.demo.entity
* com.example.demo.model
* com.example.demo.exception
* com.example.demo.exception.handler

## Step 6: Create the Entity classes

Product class has a Product entity class has a Productid, product name, product price and quantity.

package com.example.demo.entity;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long productId;

@Column(name ="PRODUCT\_NAME")

private String productName;

@Column(name ="PRICE")

private long price;

@Column(name ="QUANTITY")

private long quantity;

}

## Step 7: Create the Model classes

1.request class and response for product

package com.example.demo.model;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class ProductRequest {

private String name;

private long price;

private long quanity;

}

package com.example.demo.model;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class ProductResponse {

private long productId;

private String productName;

private long price;

private long quantity;

}

package com.example.demo.model;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class ErrorResponse {

private String errorMessage;

private String errorCode;

}

## Step 8: Create the Repository layer

Name is :ProductRepository

package com.example.demo.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.example.demo.entity.Product;

@Repository

public interface ProductRepository extends JpaRepository< Product, Long> {

}

## Step 9: Create the Rest Controller

package com.example.demo.controller;

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

import org.springframework.http.HttpStatus;

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.PostMapping;

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

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

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

import com.example.demo.model.ProductRequest;

import com.example.demo.model.ProductResponse;

import com.example.demo.service.ProductService;

@RestController

@RequestMapping("product")

public class ProductController {

@Autowired

ProductService productService;

@PostMapping

public ResponseEntity<Long> addProduct(@RequestBody ProductRequest productRequest)

{

long productid =productService.addProduct(productRequest);

return new ResponseEntity<Long>(productid, HttpStatus.CREATED);

}

@GetMapping("/{id}")

public ResponseEntity<ProductResponse> getProductById(@PathVariable("id") long productId)

{

ProductResponse productResponse=productService.getProductById(productId);

return new ResponseEntity<>(productResponse,HttpStatus.OK);

}

}

## Step 10: Create the Service layer

Name : ProductService interface and ProductServiceImpl Class

package com.example.demo.service;

import com.example.demo.model.ProductRequest;

import com.example.demo.model.ProductResponse;

public interface ProductService {

long addProduct(ProductRequest productRequest);

ProductResponse getProductById(long productId);

}

package com.example.demo.service;

import org.springframework.beans.BeanUtils;

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

import org.springframework.stereotype.Service;

import com.example.demo.entity.Product;

import com.example.demo.exception.ProductServiceCustomException;

import com.example.demo.model.ProductRequest;

import com.example.demo.model.ProductResponse;

import com.example.demo.repository.ProductRepository;

import lombok.extern.log4j.Log4j2;

@Service

@Log4j2

public class ProductServiceImpl implements ProductService{

@Autowired

ProductRepository productRepo;

@Override

public long addProduct(ProductRequest productRequest) {

log.info("ProductServiceImpl.addProduct started");

Product product=Product.builder()

.productName(productRequest.getName())

.price(productRequest.getPrice())

.quantity(productRequest.getQuanity()).build();

productRepo.save(product);

log.info("ProductServiceImpl.addProduct ended");

return product.getProductId();

}

@Override

public ProductResponse getProductById(long productId) {

log.info("ProductServiceImpl.getProductById started");

log.info("Get Product for ProductID : {}",productId);

Product product=productRepo.findById(productId)

.orElseThrow( ()->new ProductServiceCustomException ( "Product with given id is not found","PRODUCT\_NOT\_FOUND"));

ProductResponse productResponse=new ProductResponse();

BeanUtils.copyProperties(product, productResponse);

log.info("ProductServiceImpl.getProductById ended");

return productResponse;

}

}

## Step 11: Create the Custom Exception Class

package com.example.demo.exception;

import lombok.Data;

@Data

public class ProductServiceCustomException extends RuntimeException{

private String errorCode;

public ProductServiceCustomException(String message,String errorCode)

{

super(message);

this.errorCode=errorCode;

}

}

## Step 12: Create the Exception Handler Class(@ControllerAdvice)

package com.example.demo.exception.handler;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

import com.example.demo.exception.ProductServiceCustomException;

import com.example.demo.model.ErrorResponse;

@ControllerAdvice

public class RestReponseEntityExceptionHandler extends ResponseEntityExceptionHandler{

@ExceptionHandler(ProductServiceCustomException.class)

public ResponseEntity<ErrorResponse> handleProductServiceException(ProductServiceCustomException exception)

{

return new ResponseEntity<ErrorResponse>(new ErrorResponse().builder()

.errorMessage(exception.getMessage())

.errorCode(exception.getErrorCode())

.build(),HttpStatus.NOT\_FOUND);

}

}

# =========================================================

# Project 2: Service Registry Spring boot application

## Step 1: create the Spring boot application with below details:

* **Project Name:** service-regisrty
* **Project Type:**Maven
* **Choose jdk:** java 8
* **Package name:**com.example.demo

## Step 2: Add the Dependencies

* **Spring boot version :** 2.7.11
* **Dependencies**
  + **Spring boot starter Eureka Server**
  + **Spring boot starter Cloud Bootstrap**
  + **Lombok**

## Step 3: Eureka server details added into application.yaml file

server:

port: 8761

eureka:

instance:

hostname: "${HOSTNAME:localhost}.eureka"

client:

register-with-eureka: false

fetch-registry: false

## Step 4: add @EnableEurekaServer at main class

package com.example.demo.serviceregistry;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer

public class ServiceRegistryApplication {

public static void main(String[] args) {

SpringApplication.run(ServiceRegistryApplication.class, args);

}

}

# =========================================================

# Project 3: Order Service Spring boot application

## Step 1: create the Spring boot application with details:

* **Project Name:** OrderService
* **Project Type:**Maven
* **Choose jdk:** java 8
* **Package name:**com.example.demo

## Step 2: Add the Dependencies

* **Spring boot version :** 2.7.11
* **Dependencies**
  + **Spring boot starter Web**
  + **Spring boot starter Data jpa**
  + **Spring boot starter DevTools**
  + **Spring boot starter Cloud Bootstrap**
  + **mySQl Driver**
  + **lombok**

For Microservices 🡺 add the Dependencies

* **Spring boot starter Eureka Discovery Client**

## Step 3: Configuration in application.yaml:

Note :Change the application.properties into application.yaml file and add the database configuration details

### Database Configuration:

spring:

datasource:

url: jdbc:mysql://${DB\_HOST:localhost}:3306/orderdb

username: root

password:

driver-class-name: com.mysql.cj.jdbc.Driver

jpa:

database-platform: org.hibernate.dialect.MySQL57InnoDBDialect

hibernate:

ddl-auto: update

application:

name: ORDER-SERVICE

### microservices Integration in application.yaml file

server:

port: 8082

eureka:

instance:

prefer-ip-address: true

client:

fetch-registry: true

register-with-eureka: true

service-url:

defaultZone: ${EUREKA\_SERVER\_ADDRESS:http://localhost:8761/eureka}

## Step 4: add @EnableEurekaClient at main class

package com.example.demo.orderservice;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class OrderServiceApplication {

public static void main(String[] args) {

SpringApplication.run(OrderServiceApplication.class, args);

}

}

## step 5: Create the Packages

* com.example.demo.controller
* com.example.demo.service
* com.example.demo.repository
* com.example.demo.entity
* com.example.demo.model
* com.example.demo.exception
* com.example.demo.exception.handler

## Step 6: Create the Entity classes

package com.example.demo.orderservice.entity;

import java.time.Instant;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Table(name="ORDER\_DETAILS")

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class Order {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(name="PRODUCT\_ID")

private long productId;

@Column(name="QUANITY")

private long quantity;

@Column(name="ORDER\_DATE")

private Instant orderDate;

@Column(name="ORDER\_STATUS")

private String orderStatus;

@Column(name="TOTAL\_AMOUNT")

private long amount;

}

## Step 7: Create the Model classes

1.request class and response for Order

package com.example.demo.orderservice.model;

public enum PaymentMode {

CASH,

PAYPAL,

DEBIT\_CARD,

CREDIT\_CARD,

UPI

}

package com.example.demo.orderservice.model;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class OrderRequest {

private long productId;

private long totalAmount;

private long quantity;

private PaymentMode paymentMode;

}

package com.example.demo.orderservice.model;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class ErrorResponse {

private String errorMessage;

private String errorCode;

}

## Step 8: Create the Repository layer

package com.example.demo. orderservice.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.example.demo.orderservice.entity.Order;

@Repository

public interface OrderRepository extends JpaRepository<Order, Long>{

}

## Step 9: Create the Service layer

package com.example.demo.orderservice.service;

import com.example.demo.orderservice.model.OrderRequest;

public interface OrderService {

long getPlaceOrder(OrderRequest orderRequest);

}

package com.example.demo.orderservice.service;

import java.time.Instant;

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

import org.springframework.stereotype.Service;

import com.example.demo.orderservice.controller.OrderController;

import com.example.demo.orderservice.entity.Order;

import com.example.demo.orderservice.model.OrderRequest;

import com.example.demo.orderservice.repository.OrderRepository;

import lombok.extern.log4j.Log4j2;

@Service

@Log4j2

public class OrderServiceImpl implements OrderService {

@Autowired

OrderRepository orderrepo;

@Override

public long getPlaceOrder(OrderRequest orderRequest) {

//Order Entity ->save the data with Status order created

//Product Service -> Block Products (reduce the quantity)

//Payment Service-> payment->Success->COMPLETE,Else CANCELLED

log.info("Placing order request : {}", orderRequest);

Order order=Order.builder()

.amount(orderRequest.getTotalAmount())

.orderStatus("CREATED")

.productId(orderRequest.getProductId())

.orderDate(Instant.now())

.quantity(orderRequest.getQuantity())

.build();

orderrepo.save(order);

log.info("order placed Successfully with order id : {}",order.getId());

return order.getId();

}

}

## Step 10: Create the Rest Controller

package com.example.demo.orderservice.controller;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

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

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

import com.example.demo.orderservice.model.OrderRequest;

import com.example.demo.orderservice.service.OrderService;

import lombok.extern.log4j.Log4j2;

@RestController

@Log4j2

@RequestMapping("/order")

public class OrderController {

@Autowired

OrderService orderService;

@PostMapping("/placeOrder")

public ResponseEntity<Long> getPlaceOrder(@RequestBody OrderRequest orderRequest)

{

long orderId=orderService.getPlaceOrder(orderRequest);

log.info("Order Id : {}", orderId);

return new ResponseEntity<Long>(orderId,HttpStatus.CREATED);

}

}

# =========================================================

# Project 4: Config Server Spring boot application

## Step 1: create the Spring boot application with details:

* **Project Name:** ConfigServer
* **Project Type:**Maven
* **Choose jdk:** java 8
* **Package name:**com.example.demo

## Step 2: Add the Dependencies

* **Spring boot version :** 2.7.11
* **Dependencies**
  + **Spring boot starter Config Server**
  + **Spring boot starter Eureka Discovery Client**

## Step 3: Add git hub configuration in application.yaml file

server:

port: 9296

spring:

application:

name: CONFIG-SERVER

cloud:

config:

server:

git:

uri: https://github.com/javatrainingms/springbootMSProperties

clone-on-start: true

#eureka:

# instance:

# prefer-ip-address: true

# client:

# fetch-registry: true

# register-with-eureka: true

# service-url:

# defaultZone: ${EUREKA\_SERVER\_ADDRESS:http://localhost:8761/eureka}

## Step 4: Remove the eureka configuration in OrderService and ProductService projects in application.yaml file

Below code needs to comments in application.yaml file (including above yaml file also)

#eureka:

# instance:

# prefer-ip-address: true

# client:

# fetch-registry: true

# register-with-eureka: true

# service-url:

# defaultZone: ${EUREKA\_SERVER\_ADDRESS:http://localhost:8761/eureka}

#

## Step 5: Add the Config client dependency into ProductService and OrderService projects in pom.xml file

<dependency>

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

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

</dependency>

## Step 6: add the cloud server configuration to ProductService and OrderService Projects in Application.yaml

spring:

datasource:

url: jdbc:mysql://${DB\_HOST:localhost}:3306/productdb

username: root

password:

driver-class-name: com.mysql.cj.jdbc.Driver

jpa:

database-platform: org.hibernate.dialect.MySQL57InnoDBDialect

hibernate:

ddl-auto: update

application:

name: PRODUCT-SERVICE

**config:**

**import: configserver:${CONFIG\_SERVER\_URL:http://localhost:9296}**

# ========================================================

# Update Project 1: Reduce Quantity API implementation in Production Service

## Step 1 : add the Api for Reduce Quantity Service in ProductController

@PutMapping("/reduceQuantity/{id}")

public ResponseEntity<Void> reduceQuantity(@PathVariable("id") long productId,

@RequestParam long quantity)

{

productService.reduceQuantity(productId,quantity);

return new ResponseEntity<Void>(HttpStatus.OK);

}

## Step 2: Add into Service Layer

### ProductService

void reduceQuantity(long productId, long quantity);

### ProductServiceImpl

@Override

public void reduceQuantity(long productId, long quantity) {

log.info("ProductServiceImpl.reduceQuantity started");

log.info("Reduce Quantity {} for product id {}",quantity,productId);

Product product=productRepo.findById(productId)

.orElseThrow(()->new ProductServiceCustomException ("Product with given id is not found" , "PRODUCT\_NOT\_FOUND"));

if(product.getQuantity() <quantity)

{

throw new ProductServiceCustomException("Product does not have sufficient Quantity" , "INSUFFICIENT\_QUANTITY");

}

product.setQuantity(product.getQuantity() - quantity);

productRepo.save(product);

log.info("Product Quantity updated Successfully..");

log.info("ProductServiceImpl.reduceQuantity Ended");

}

# ========================================================

# Update Project 2: Feign Client for ProductService configured into OrderService Project

## Step1:Add the Dependency OpenFeign

<dependency>

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

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

</dependency>

## Step 2: Enable Feign Client

package com.example.demo.orderservice;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**@EnableFeignClients**

public class OrderServiceApplication {

public static void main(String[] args) {

SpringApplication.run(OrderServiceApplication.class, args);

}

}

## Step 3: Add the External Client Interface(ProductService)

package com.example.demo.orderservice.external.client;

import org.springframework.cloud.openfeign.FeignClient;

import org.springframework.http.ResponseEntity;

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

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

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

@FeignClient(name="PRODUCT-SERVICE/product")

public interface ProductService {

@PutMapping("/reduceQuantity/{id}")

ResponseEntity<Void> reduceQuantity(@PathVariable("id") long productId,

@RequestParam long quantity);

}

## Step 4: update OrderService to reduce quantity for product

package com.example.demo.orderservice.service;

import java.time.Instant;

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

import org.springframework.stereotype.Service;

import com.example.demo.orderservice.controller.OrderController;

import com.example.demo.orderservice.entity.Order;

import com.example.demo.orderservice.model.OrderRequest;

import com.example.demo.orderservice.repository.OrderRepository;

import lombok.extern.log4j.Log4j2;

@Service

@Log4j2

public class OrderServiceImpl implements OrderService {

@Autowired

OrderRepository orderrepo;

@Autowired

ProductService prodService;

@Override

public long getPlaceOrder(OrderRequest orderRequest) {

//Order Entity ->save the data with Status order created

//Product Service -> Block Products (reduce the quantity)

//Payment Service-> payment->Success->COMPLETE,Else CANCELLED

log.info("Placing order request : {}", orderRequest);

prodService.reduceQuantity(orderRequest.getProductId(), orderRequest.getQuantity());

log.info("creating order with status is CREATED...");

Order order=Order.builder()

.amount(orderRequest.getTotalAmount())

.orderStatus("CREATED")

.productId(orderRequest.getProductId())

.orderDate(Instant.now())

.quantity(orderRequest.getQuantity())

.build();

orderrepo.save(order);

log.info("order placed Successfully with order id : {}",order.getId());

return order.getId();

}

}

# ========================================================

# Update Project 2: Exception handling into OrderService Project

## Step 1: add ErrorResponse class

package com.example.demo.orderservice.external.response;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class ErrorResponse {

private String errorMessage;

private String errorCode;

}

## Step 2: add CustomOrderException class

package com.example.demo.orderservice.exception;

import lombok.Data;

@Data

public class CustomOrderException extends RuntimeException{

private String errorCode;

private int status;

public CustomOrderException(String message,String errorCode,int status)

{

super(message);

this.errorCode=errorCode;

this.status=status;

}

}

## Step 3: Add the Exception Handler Class

package com.example.demo.orderservice.exception.handler;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

import com.example.demo.orderservice.exception.CustomOrderException;

import com.example.demo.orderservice.external.response.ErrorResponse;

@ControllerAdvice

public class RestReponseEntityExceptionHandler extends ResponseEntityExceptionHandler{

@ExceptionHandler(CustomOrderException.class)

public ResponseEntity<ErrorResponse> handleProductServiceException(CustomOrderException exception)

{

return new ResponseEntity<ErrorResponse>(new ErrorResponse().builder()

.errorMessage(exception.getMessage())

.errorCode(exception.getErrorCode())

.build(),HttpStatus.valueOf(exception.getStatus()));

}

}

## Step 4 : Add ErrorDecoder class

package com.example.demo.orderservice.external.decoder;

import java.io.IOException;

import com.example.demo.orderservice.exception.CustomOrderException;

import com.example.demo.orderservice.external.response.ErrorResponse;

import com.fasterxml.jackson.databind.ObjectMapper;

import feign.Response;

import feign.codec.ErrorDecoder;

import lombok.extern.log4j.Log4j2;

@Log4j2

public class CustomErrorDecoder implements ErrorDecoder{

@Override

public Exception decode(String methodKey, Response response) {

ObjectMapper objectMapper=new ObjectMapper();

log.info("::{}",response.request().url());

log.info("::{}",response.request().headers());

try {

ErrorResponse errorResponse = objectMapper.readValue(

response.body().asInputStream(), ErrorResponse.class);

return new CustomOrderException(errorResponse.getErrorMessage(),

errorResponse.getErrorCode(),response.status());

} catch (IOException e) {

return new CustomOrderException("Internal Server Error","INTERNAL\_SERVER\_ERROR",500);

}

}

}

## Step 5: Configure ErrorDecoder

package com.example.demo.orderservice.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import com.example.demo.orderservice.external.decoder.CustomErrorDecoder;

import feign.codec.ErrorDecoder;

@Configuration

public class FeignConfig {

@Bean

public ErrorDecoder errorDecoder()

{

return new CustomErrorDecoder();

}

}

# ===========================================================

# Software: Install Docker and Zipkin via Docker

## Step 1: install the Docker

URL : <https://docs.docker.com/desktop/install/windows-install/>

Please follow above url inside steps

## Step 2 :install the Zipkin

URL: <https://zipkin.io/pages/quickstart>

Please follow above url inside steps

**using Docker**: already docker installed in my system🡪 I am using command like (in cmd)

docker run -d -p 9411:9411 openzipkin/zipkin

# ===========================================================

# Project 1,2 : add the Dependencies for Zipkin and Slueth

<dependency>

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

<artifactId>spring-cloud-sleuth-zipkin</artifactId>

</dependency>

<dependency>

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

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

</dependency>

# ===========================================================

# Project 5: Payment Service Spring boot application

## Step 1: create the Spring boot application with details:

* **Project Name:** PaymentService
* **Project Type:**Maven
* **Choose jdk:** java 8
* **Package name:**com.example.demo

## Step 2: Add the Dependencies

* **Spring boot version :** 2.7.11
* **Dependencies**
  + **Spring boot starter Web**
  + **Spring boot starter Data jpa**
  + **Spring boot starter DevTools**
  + **Spring boot starter Cloud Bootstrap**
  + **mySQl Driver**
  + **lombok**

For Microservices 🡺 add the Dependencies

* **Spring boot starter Eureka Discovery Client**
* **Spring boot config client**
* **Zipkin**
* **Sleuth(not available)**

<dependency>

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

<artifactId>spring-cloud-sleuth-zipkin</artifactId>

</dependency>

<dependency>

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

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

</dependency>

## Step 3: Configuration in application.yaml:

Note :Change the application.properties into application.yaml file and add the database configuration and microservice details

server:

port: 8081

spring:

datasource:

url: jdbc:mysql://${DB\_HOST:localhost}:3306/paymentdb

username: root

password:

driver-class-name: com.mysql.cj.jdbc.Driver

jpa:

database-platform: org.hibernate.dialect.MySQL57InnoDBDialect

hibernate:

ddl-auto: update

application:

name: PAYMENT-SERVICE

config:

import: configserver:${CONFIG\_SERVER\_URL:http://localhost:9296}

## Step 4: add @EnableEurekaClient at main class

package com.example.demo.paymentService;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

import com.example.demo.entity.Product;

@SpringBootApplication

**@EnableEurekaClient**

public class PaymentServiceApplication {

public static void main(String[] args) {

SpringApplication.run(PaymentServiceApplication.class, args);

}

}

## step 5: Create the Packages

* com.example.demo.paymentService.controller
* com.example.demo.paymentService.service
* com.example.demo.paymentService.repository
* com.example.demo.paymentService.entity
* com.example.demo.paymentService.model
* com.example.demo.paymentService.exception
* com.example.demo.paymentService.exception.handler

## Step 6: Create the Entity classes

package com.example.demo.paymentService.entity;

import java.time.Instant;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Table(name ="TRANSACTION\_DETAILS")

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class TransactionDetails {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(name ="ORDER\_ID")

private long orderid;

@Column(name ="MODE")

private String paymentMode;

@Column(name ="REFERENCE\_NUMBER")

private String referenceNumber;

@Column(name ="PAYMENT\_DATE")

private Instant paymentDate;

@Column(name ="STATUS")

private String paymentStatus;

@Column(name ="AMOUNT")

private long amount;

}

## Step 7: Create the Model classes

package com.example.demo.paymentService.model;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class PaymentRequest {

private long orderid;

private long amount;

private String referenceNumber;

private PaymentMode paymentMode;

}

package com.example.demo.paymentService.model;

public enum PaymentMode {

CASH,

PAYPAL,

DEBIT\_CARD,

CREDIT\_CARD,

UPI

}

## Step 8: Create the Repository layer

package com.example.demo.paymentService.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.example.demo.paymentService.entity.TransactionDetails;

@Repository

public interface TransactionDetailsRepository extends JpaRepository<TransactionDetails, Long>{

}

## Step 9: Create the Service layer

Name : PaymentService interface and PaymentServiceImpl Class

package com.example.demo.paymentService.service;

import com.example.demo.paymentService.model.PaymentRequest;

public interface PaymentService {

Long doPayment(PaymentRequest paymentRequest);

}

package com.example.demo.paymentService.service;

import java.time.Instant;

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

import org.springframework.stereotype.Service;

import com.example.demo.paymentService.entity.TransactionDetails;

import com.example.demo.paymentService.model.PaymentRequest;

import com.example.demo.paymentService.repository.TransactionDetailsRepository;

import lombok.extern.log4j.Log4j2;

@Service

@Log4j2

public class PaymentServiceImpl implements PaymentService{

@Autowired

private TransactionDetailsRepository transactionDetailsRepository;

@Override

public Long doPayment(PaymentRequest paymentRequest) {

log.info("Recording payment Details : {}",paymentRequest);

TransactionDetails transactionDetails=TransactionDetails.builder()

.paymentDate(Instant.now())

.paymentMode(paymentRequest.getPaymentMode().name())

.paymentStatus("SUCCESS")

.orderid(paymentRequest.getOrderid())

.referenceNumber(paymentRequest.getReferenceNumber())

.amount(paymentRequest.getAmount())

.build();

transactionDetailsRepository.save(transactionDetails);

log.info("Transaction completed with id : {}",transactionDetails.getId());

return transactionDetails.getId();

}

}

## Step 10: Create the Rest Controller

package com.example.demo.paymentService.controller;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

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

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

import com.example.demo.paymentService.service.PaymentService;

@RestController

@RequestMapping("/payment")

public class PaymentController {

@Autowired

private PaymentService paymentService;

@PostMapping

public ResponseEntity<Long> doPayment(@RequestBody com.example.demo.paymentService.model.PaymentRequest paymentRequest)

{

return new ResponseEntity<Long>(paymentService.doPayment(paymentRequest),HttpStatus.OK);

}

}

# =========================================================

# Update Project 2: Feign Client for PaymentService configured into OrderService Project

## Step 1: Add the External Client Interface(PaymentService)

package com.example.demo.orderservice.external.client;

import org.springframework.cloud.openfeign.FeignClient;

import org.springframework.http.ResponseEntity;

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

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

import com.example.demo.orderservice.external.request.PaymentRequest;

@FeignClient(name="PAYMENT-SERVICE/payment")

public interface PaymentService {

@PostMapping

public ResponseEntity<Long> doPayment(@RequestBody PaymentRequest paymentRequest);

}

## Step 4: Add PaymentRequest into Order Project

package com.example.demo.orderservice.external.request;

import com.example.demo.orderservice.model.PaymentMode;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class PaymentRequest {

private long orderid;

private long amount;

private String referenceNumber;

private PaymentMode paymentMode;

}

## Step 5: update OrderService to payment details sent to payment db

package com.example.demo.orderservice.service;

import java.time.Instant;

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

import org.springframework.stereotype.Service;

import com.example.demo.orderservice.controller.OrderController;

import com.example.demo.orderservice.entity.Order;

import com.example.demo.orderservice.model.OrderRequest;

import com.example.demo.orderservice.repository.OrderRepository;

import lombok.extern.log4j.Log4j2;

@Service

@Log4j2

public class OrderServiceImpl implements OrderService {

@Autowired

OrderRepository orderrepo;

@Autowired

ProductService prodService;

@Autowired

PaymentService paymentService;

@Override

public long getPlaceOrder(OrderRequest orderRequest) {

//Order Entity ->save the data with Status order created

//Product Service -> Block Products (reduce the quantity)

//Payment Service-> payment->Success->COMPLETE,Else CANCELLED

log.info("Placing order request : {}", orderRequest);

prodService.reduceQuantity(orderRequest.getProductId(), orderRequest.getQuantity());

log.info("creating order with status is CREATED...");

Order order=Order.builder()

.amount(orderRequest.getTotalAmount())

.orderStatus("CREATED")

.productId(orderRequest.getProductId())

.orderDate(Instant.now())

.quantity(orderRequest.getQuantity())

.build();

orderrepo.save(order);

log.info("calling payment service to complete the Payment service");

PaymentRequest paymentRequest =PaymentRequest.builder()

.orderid(order.getId())

.paymentMode(orderRequest.getPaymentMode())

.amount(orderRequest.getTotalAmount())

.build();

String orderStatus= null;

try

{

paymentService.doPayment(paymentRequest);

log.info("payment done successfully. Chanding the order status to PLACED");

orderStatus="PLACED";

}

catch (Exception e) {

log.info("Error occured in payment. Changing order status to PAYMENT\_FAILED");

orderStatus="PAYMENT\_FAILED";

}

order.setOrderStatus(orderStatus);

orderrepo.save(order);

log.info("order placed Successfully with order id : {}",order.getId());

return order.getId();

}

}

# ==============================================================

# Update Project 2: Implementing GetOrder Details in Order Service Project

## Step 1: Add the OrderResponse class to send order Details

package com.example.demo.orderservice.model;

import java.time.Instant;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class OrderResponse {

private long orderId;

private Instant orderDate;

private String orderStatus;

private long amount;

}

## Step 2: to get Order Details from OrderController class and OrderService ,OrderServiceImpl classes

package com.example.demo.orderservice.controller;

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

import org.springframework.http.HttpStatus;

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.PostMapping;

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

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

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

import com.example.demo.orderservice.model.OrderRequest;

import com.example.demo.orderservice.model.OrderResponse;

import com.example.demo.orderservice.service.OrderService;

import lombok.extern.log4j.Log4j2;

@RestController

@Log4j2

@RequestMapping("/order")

public class OrderController {

@Autowired

OrderService orderService;

@PostMapping("/placeOrder")

public ResponseEntity<Long> getPlaceOrder(@RequestBody OrderRequest orderRequest)

{

long orderId=orderService.getPlaceOrder(orderRequest);

log.info("Order Id : {}", orderId);

return new ResponseEntity<Long>(orderId,HttpStatus.CREATED);

}

@GetMapping("/{orderId}")

public ResponseEntity<OrderResponse> getOrderDetails(@PathVariable long orderId)

{

OrderResponse orderResponse=orderService.getOrderDetails(orderId);

return new ResponseEntity<OrderResponse>(orderResponse,HttpStatus.OK);

}

}

package com.example.demo.orderservice.service;

import com.example.demo.orderservice.model.OrderRequest;

import com.example.demo.orderservice.model.OrderResponse;

public interface OrderService {

long getPlaceOrder(OrderRequest orderRequest);

OrderResponse getOrderDetails(long orderId);

}

package com.example.demo.orderservice.service;

import java.time.Instant;

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

import org.springframework.stereotype.Service;

import com.example.demo.orderservice.controller.OrderController;

import com.example.demo.orderservice.entity.Order;

import com.example.demo.orderservice.exception.CustomOrderException;

import com.example.demo.orderservice.external.client.PaymentService;

import com.example.demo.orderservice.external.client.ProductService;

import com.example.demo.orderservice.external.request.PaymentRequest;

import com.example.demo.orderservice.model.OrderRequest;

import com.example.demo.orderservice.model.OrderResponse;

import com.example.demo.orderservice.repository.OrderRepository;

import lombok.extern.log4j.Log4j2;

@Service

@Log4j2

public class OrderServiceImpl implements OrderService {

@Autowired

OrderRepository orderrepo;

@Autowired

ProductService prodService;

@Autowired

PaymentService paymentService;

@Override

public long getPlaceOrder(OrderRequest orderRequest) {

//Order Entity ->save the data with Status order created

//Product Service -> Block Products (reduce the quantity)

//Payment Service-> payment->Success->COMPLETE,Else CANCELLED

log.info("Placing order request : {}", orderRequest);

prodService.reduceQuantity(orderRequest.getProductId(), orderRequest.getQuantity());

log.info("creating order with status is CREATED...");

Order order=Order.builder()

.amount(orderRequest.getTotalAmount())

.orderStatus("CREATED")

.productId(orderRequest.getProductId())

.orderDate(Instant.now())

.quantity(orderRequest.getQuantity())

.build();

orderrepo.save(order);

log.info("calling payment service to complete the Payment service");

PaymentRequest paymentRequest =PaymentRequest.builder()

.orderid(order.getId())

.paymentMode(orderRequest.getPaymentMode())

.amount(orderRequest.getTotalAmount())

.build();

String orderStatus= null;

try

{

paymentService.doPayment(paymentRequest);

log.info("payment done successfully. Chanding the order status to PLACED");

orderStatus="PLACED";

}

catch (Exception e) {

log.info("Error occured in payment. Changing order status to PAYMENT\_FAILED");

orderStatus="PAYMENT\_FAILED";

}

order.setOrderStatus(orderStatus);

orderrepo.save(order);

log.info("order placed Successfully with order id : {}",order.getId());

return order.getId();

}

@Override

public OrderResponse getOrderDetails(long orderId) {

log.info("Get order details for Order Id :{}",orderId);

Order order=orderrepo.findById(orderId)

.orElseThrow(()->new CustomOrderException("order not found for the order id : "+orderId,

"NOT\_FOUND", 404));

OrderResponse orderResponse

=OrderResponse.builder()

.orderId(order.getId())

.orderStatus(order.getOrderStatus())

.amount(order.getAmount())

.orderDate(order.getOrderDate())

.build();

return orderResponse;

}

}

# ==============================================================

# Update Project 2: Fetching Product Data for getOrder Details API

## Step 1: Add the restTemplate to fetch the Product Details from ProductService project

package com.example.demo.orderservice;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.web.client.RestTemplateBuilder;

import org.springframework.cloud.client.loadbalancer.LoadBalanced;

import org.springframework.cloud.openfeign.EnableFeignClients;

import org.springframework.context.annotation.Bean;

import org.springframework.web.client.RestTemplate;

@SpringBootApplication

@EnableFeignClients

public class OrderServiceApplication {

public static void main(String[] args) {

SpringApplication.run(OrderServiceApplication.class, args);

}

@Bean

@LoadBalanced

public RestTemplate restTemplate(RestTemplateBuilder builder) {

return builder.build();

}

}

## Step 2: create the ProductResponse Class

package com.example.demo.orderservice.external.response;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@Builder

@AllArgsConstructor

@NoArgsConstructor

public class ProductResponse {

private String productName;

private long productId;

private long quantity;

private long price;

}

## Step 3: Update the OrderResponse class to add the ProductDetails

package com.example.demo.orderservice.model;

import java.time.Instant;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class OrderResponse {

private long orderId;

private Instant orderDate;

private String orderStatus;

private long amount;

private ProductDetails productDetails;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public static class ProductDetails {

private long productId;

private String productName;

private long price;

private long quantity;

}

}

## Step 4: Update the OrderServiceImpl class to add the ProductDetails

package com.example.demo.orderservice.service;

import java.time.Instant;

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

import org.springframework.stereotype.Service;

import org.springframework.web.client.RestTemplate;

import com.example.demo.orderservice.entity.Order;

import com.example.demo.orderservice.exception.CustomOrderException;

import com.example.demo.orderservice.external.client.PaymentService;

import com.example.demo.orderservice.external.client.ProductService;

import com.example.demo.orderservice.external.request.PaymentRequest;

import com.example.demo.orderservice.external.response.ProductResponse;

import com.example.demo.orderservice.model.OrderRequest;

import com.example.demo.orderservice.model.OrderResponse;

import com.example.demo.orderservice.repository.OrderRepository;

import lombok.extern.log4j.Log4j2;

@Service

@Log4j2

public class OrderServiceImpl implements OrderService {

@Autowired

OrderRepository orderrepo;

@Autowired

ProductService prodService;

@Autowired

PaymentService paymentService;

@Autowired

RestTemplate restTemplate;

@Override

public long getPlaceOrder(OrderRequest orderRequest) {

//Order Entity ->save the data with Status order created

//Product Service -> Block Products (reduce the quantity)

//Payment Service-> payment->Success->COMPLETE,Else CANCELLED

log.info("Placing order request : {}", orderRequest);

prodService.reduceQuantity(orderRequest.getProductId(), orderRequest.getQuantity());

log.info("creating order with status is CREATED...");

Order order=Order.builder()

.amount(orderRequest.getTotalAmount())

.orderStatus("CREATED")

.productId(orderRequest.getProductId())

.orderDate(Instant.now())

.quantity(orderRequest.getQuantity())

.build();

orderrepo.save(order);

log.info("calling payment service to complete the Payment service");

PaymentRequest paymentRequest =PaymentRequest.builder()

.orderid(order.getId())

.paymentMode(orderRequest.getPaymentMode())

.amount(orderRequest.getTotalAmount())

.build();

String orderStatus= null;

try

{

paymentService.doPayment(paymentRequest);

log.info("payment done successfully. Chanding the order status to PLACED");

orderStatus="PLACED";

}

catch (Exception e) {

log.info("Error occured in payment. Changing order status to PAYMENT\_FAILED");

orderStatus="PAYMENT\_FAILED";

}

order.setOrderStatus(orderStatus);

orderrepo.save(order);

log.info("order placed Successfully with order id : {}",order.getId());

return order.getId();

}

@Override

public OrderResponse getOrderDetails(long orderId) {

log.info("Get order details for Order Id :{}",orderId);

Order order=orderrepo.findById(orderId)

.orElseThrow(()->new CustomOrderException("order not found for the order id : "+orderId,

"NOT\_FOUND", 404));

log.info("Invoking Product Service to fetch the product for id:{}",order.getProductId());

ProductResponse productResponse

=restTemplate.getForObject("http://PRODUCT-SERVICE/product/"+order.getProductId(),

ProductResponse.class);

OrderResponse.ProductDetails productDetails=OrderResponse.ProductDetails

.builder()

.productName(productResponse.getProductName())

.productId(productResponse.getProductId())

.build();

OrderResponse orderResponse

=OrderResponse.builder()

.orderId(order.getId())

.orderStatus(order.getOrderStatus())

.amount(order.getAmount())

.orderDate(order.getOrderDate())

.productDetails(productDetails)

.build();

return orderResponse;

}

}

You can check the URL

<http://localhost:8082/order/4>

Get method

Output :

{

    "orderId": 4,

    "orderDate": "2023-04-26T16:28:04Z",

    "orderStatus": "PLACED",

    "amount": 13000,

    "productDetails": {

        "productId": 1,

        "productName": "laptop",

        "price": 0,

        "quantity": 0

    }

}

# ==============================================================

# Update Project 2, 5: Fetching Payment Details for getOrder Details API

## Step 1: Add the PaymentResponse into PaymentService Project

package com.example.demo.paymentService.model;

import java.time.Instant;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class PaymentResponse {

private long paymentId;

private String status;

private PaymentMode paymentMode;

private long amount;

private Instant paymentDate;

private long orderId;

}

## Step 2: get PaymentResponse from PaymentController based on orderId in PaymentService Project

package com.example.demo.paymentService.controller;

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

import org.springframework.http.HttpStatus;

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.PostMapping;

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

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

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

import com.example.demo.paymentService.model.PaymentResponse;

import com.example.demo.paymentService.service.PaymentService;

@RestController

@RequestMapping("/payment")

public class PaymentController {

@Autowired

private PaymentService paymentService;

@PostMapping

public ResponseEntity<Long> doPayment(@RequestBody com.example.demo.paymentService.model.PaymentRequest paymentRequest)

{

return new ResponseEntity<Long>(paymentService.doPayment(paymentRequest),HttpStatus.OK);

}

@GetMapping("/order/{orderId}")

public ResponseEntity<PaymentResponse> getPaymentDetailsByOrderId(@PathVariable("orderId") String orderId)

{

return new ResponseEntity<PaymentResponse>(paymentService.getPaymentDetailsByOrderId(orderId) ,HttpStatus.OK);

}

}

## Step 3: get PaymentResponse from PaymentService, PaymentServiceImpl based on orderid in PaymentService Project

package com.example.demo.paymentService.service;

import com.example.demo.paymentService.model.PaymentRequest;

import com.example.demo.paymentService.model.PaymentResponse;

public interface PaymentService {

Long doPayment(PaymentRequest paymentRequest);

PaymentResponse getPaymentDetailsByOrderId(String orderId);

}

package com.example.demo.paymentService.service;

import java.time.Instant;

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

import org.springframework.stereotype.Service;

import com.example.demo.paymentService.entity.TransactionDetails;

import com.example.demo.paymentService.model.PaymentMode;

import com.example.demo.paymentService.model.PaymentRequest;

import com.example.demo.paymentService.model.PaymentResponse;

import com.example.demo.paymentService.repository.TransactionDetailsRepository;

import lombok.extern.log4j.Log4j2;

@Service

@Log4j2

public class PaymentServiceImpl implements PaymentService{

@Autowired

private TransactionDetailsRepository transactionDetailsRepository;

@Override

public Long doPayment(PaymentRequest paymentRequest) {

log.info("Recording payment Details : {}",paymentRequest);

TransactionDetails transactionDetails=TransactionDetails.builder()

.paymentDate(Instant.now())

.paymentMode(paymentRequest.getPaymentMode().name())

.paymentStatus("SUCCESS")

.orderid(paymentRequest.getOrderid())

.referenceNumber(paymentRequest.getReferenceNumber())

.amount(paymentRequest.getAmount())

.build();

transactionDetailsRepository.save(transactionDetails);

log.info("Transaction completed with id : {}",transactionDetails.getId());

return transactionDetails.getId();

}

@Override

public PaymentResponse getPaymentDetailsByOrderId(String orderId) {

log.info("Getting payment details for the Order id :",orderId);

TransactionDetails transactionDetails =transactionDetailsRepository.findByOrderid(Long.valueOf(orderId));

PaymentResponse paymentResponse=PaymentResponse.builder()

.paymentId(transactionDetails.getId())

.paymentMode(PaymentMode.valueOf(transactionDetails.getPaymentMode()))

.paymentDate(transactionDetails.getPaymentDate())

.orderId(transactionDetails.getOrderid())

.status(transactionDetails.getPaymentStatus())

.amount(transactionDetails.getAmount())

.build();

return paymentResponse;

}

}

## Step 4: get PaymentResponse from PaymentRepository based on orderid in PaymentService Project

package com.example.demo.paymentService.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.example.demo.paymentService.entity.TransactionDetails;

@Repository

public interface TransactionDetailsRepository extends JpaRepository<TransactionDetails, Long>{

TransactionDetails findByOrderid(long orderid);

}

## Step 5: create the PaymentResponse Class in OrderService Project

package com.example.demo.orderservice.external.response;

import java.time.Instant;

import com.example.demo.orderservice.model.PaymentMode;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class PaymentResponse {

private long paymentId;

private String status;

private PaymentMode paymentMode;

private long amount;

private Instant paymentDate;

private long orderId;

}

## Step 6: Update the OrderResponse class to add the PaymentDetails

package com.example.demo.orderservice.model;

import java.time.Instant;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public class OrderResponse {

private long orderId;

private Instant orderDate;

private String orderStatus;

private long amount;

private ProductDetails productDetails;

private PaymentDetails paymentDetails;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public static class ProductDetails {

private long productId;

private String productName;

private long price;

private long quantity;

}

@Data

@AllArgsConstructor

@NoArgsConstructor

@Builder

public static class PaymentDetails {

private long paymentId;

private PaymentMode paymentMode;

private String paymentStatus;

private Instant paymentDate;

}

}

## Step 7: Update the OrderServiceImpl class to add the PaymentDetails

package package com.example.demo.orderservice.service;

import java.time.Instant;

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

import org.springframework.stereotype.Service;

import org.springframework.web.client.RestTemplate;

import com.example.demo.orderservice.entity.Order;

import com.example.demo.orderservice.exception.CustomOrderException;

import com.example.demo.orderservice.external.client.PaymentService;

import com.example.demo.orderservice.external.client.ProductService;

import com.example.demo.orderservice.external.request.PaymentRequest;

import com.example.demo.orderservice.external.response.PaymentResponse;

import com.example.demo.orderservice.external.response.ProductResponse;

import com.example.demo.orderservice.model.OrderRequest;

import com.example.demo.orderservice.model.OrderResponse;

import com.example.demo.orderservice.repository.OrderRepository;

import lombok.extern.log4j.Log4j2;

@Service

@Log4j2

public class OrderServiceImpl implements OrderService {

@Autowired

OrderRepository orderrepo;

@Autowired

ProductService prodService;

@Autowired

PaymentService paymentService;

@Autowired

RestTemplate restTemplate;

@Override

public long getPlaceOrder(OrderRequest orderRequest) {

//Order Entity ->save the data with Status order created

//Product Service -> Block Products (reduce the quantity)

//Payment Service-> payment->Success->COMPLETE,Else CANCELLED

log.info("Placing order request : {}", orderRequest);

prodService.reduceQuantity(orderRequest.getProductId(), orderRequest.getQuantity());

log.info("creating order with status is CREATED...");

Order order=Order.builder()

.amount(orderRequest.getTotalAmount())

.orderStatus("CREATED")

.productId(orderRequest.getProductId())

.orderDate(Instant.now())

.quantity(orderRequest.getQuantity())

.build();

orderrepo.save(order);

log.info("calling payment service to complete the Payment service");

PaymentRequest paymentRequest =PaymentRequest.builder()

.orderid(order.getId())

.paymentMode(orderRequest.getPaymentMode())

.amount(orderRequest.getTotalAmount())

.build();

String orderStatus= null;

try

{

paymentService.doPayment(paymentRequest);

log.info("payment done successfully. Chanding the order status to PLACED");

orderStatus="PLACED";

}

catch (Exception e) {

log.info("Error occured in payment. Changing order status to PAYMENT\_FAILED");

orderStatus="PAYMENT\_FAILED";

}

order.setOrderStatus(orderStatus);

orderrepo.save(order);

log.info("order placed Successfully with order id : {}",order.getId());

return order.getId();

}

@Override

public OrderResponse getOrderDetails(long orderId) {

log.info("Get order details for Order Id :{}",orderId);

Order order=orderrepo.findById(orderId)

.orElseThrow(()->new CustomOrderException("order not found for the order id : "+orderId,

"NOT\_FOUND", 404));

log.info("Invoking Product Service to fetch the product for id:{}",order.getProductId());

ProductResponse productResponse

=restTemplate.getForObject("http://PRODUCT-SERVICE/product/"+order.getProductId(),

ProductResponse.class);

log.info("Getting payment information from the payment service");

PaymentResponse paymentResponse

=restTemplate.getForObject("http://PAYMENT-SERVICE/payment/order/"+order.getId(),

PaymentResponse.class);

OrderResponse.ProductDetails productDetails=OrderResponse.ProductDetails

.builder()

.productName(productResponse.getProductName())

.productId(productResponse.getProductId())

.build();

OrderResponse.PaymentDetails paymentDetails=OrderResponse.PaymentDetails

.builder()

.paymentId(paymentResponse.getPaymentId())

.paymentStatus(paymentResponse.getStatus())

.paymentDate(paymentResponse.getPaymentDate())

.paymentMode(paymentResponse.getPaymentMode())

.build();

OrderResponse orderResponse

=OrderResponse.builder()

.orderId(order.getId())

.orderStatus(order.getOrderStatus())

.amount(order.getAmount())

.orderDate(order.getOrderDate())

.productDetails(productDetails)

.paymentDetails(paymentDetails)

.build();

return orderResponse;

}

}

You can check the URL

<http://localhost:8082/order/4>

Get method

Output :

{

    "orderId": 4,

    "orderDate": "2023-04-26T16:28:04Z",

    "orderStatus": "PLACED",

    "amount": 13000,

    "productDetails": {

        "productId": 1,

        "productName": "laptop",

        "price": 0,

        "quantity": 0

    },

    "paymentDetails": {

        "paymentId": 1,

        "paymentMode": "CASH",

        "paymentStatus": "SUCCESS",

        "paymentDate": "2023-04-26T16:28:04Z"

    }

}

# ==============================================================

# Project 6: API Gateway Service Spring boot application

## Step 1: create the Spring boot application with details:

* **Project Name:** CloudGateway
* **Project Type:**Maven
* **Choose jdk:** java 8
* **Package name:**com.example.demo

## Step 2: Add the Dependencies

* **Spring boot version :** 2.7.11
* **Dependencies**
  + **Spring boot starter Cloud Bootstrap**
  + **Spring boot starter gateway**
  + **Spring boot starter WebFlux**
  + **Lombok**
  + **Spring boot starter Eureka Discovery Client**
  + **Spring boot actuator**
  + **Spring boot config client**
  + **Zipkin**
  + **Sleuth(not available)**

<dependency>

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

<artifactId>spring-cloud-sleuth-zipkin</artifactId>

</dependency>

<dependency>

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

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

</dependency>

## Step 3: Configuration in application.yaml:

Note :Change the application.properties into application.yaml file and add the database configuration and microservice details

server:

port: 9090

spring:

application:

name: API-GATEWAY

config:

import: configserver:${CONFIG\_SERVER\_URL:http://localhost:9296}

cloud:

gateway:

routes:

- id : ORDER-SERVICE

uri: lb://ORDER-SERVICE

predicates:

- Path=/order/\*\*

- id : PAYMENT-SERVICE

uri: lb://PAYMENT-SERVICE

predicates:

- Path=/payment/\*\*

- id : PRODUCT-SERVICE

uri: lb://PRODUCT-SERVICE

predicates:

- Path=/product/\*\*

## Step 4: add @EnableEurekaClient at main class

package com.example.demo.cloudgateway;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication

**@EnableEurekaClient**

public class CloudGatewayApplication {

public static void main(String[] args) {

SpringApplication.run(CloudGatewayApplication.class, args);

}

}

You can check the URL

<http://localhost:9090/order/4>

Get method

Output :

{

    "orderId": 4,

    "orderDate": "2023-04-26T16:28:04Z",

    "orderStatus": "PLACED",

    "amount": 13000,

    "productDetails": {

        "productId": 1,

        "productName": "laptop",

        "price": 0,

        "quantity": 0

    },

    "paymentDetails": {

        "paymentId": 1,

        "paymentMode": "CASH",

        "paymentStatus": "SUCCESS",

        "paymentDate": "2023-04-26T16:28:04Z"

    }

}

# ============================================================

# Update project 4: Implement Circuit Breaker in API GateWay project

## Step 1: Add the Dependencies for Circuit Breaker (with Reactor)

<dependency>

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

<artifactId>spring-cloud-starter-circuitbreaker-reactor-resilience4j</artifactId>

</dependency>

## Step 2: Add the Configuration for Circuit Breaker in Application.yaml

server:

port: 9090

spring:

application:

name: API-GATEWAY

config:

import: configserver:${CONFIG\_SERVER\_URL:http://localhost:9296}

cloud:

gateway:

routes:

- id : ORDER-SERVICE

uri: lb://ORDER-SERVICE

predicates:

- Path=/order/\*\*

filters:

- name: CircuitBreaker

args:

name: ORDER-SERVICE

fallbackuri: forward:/orderServiceFallBack

- id : PAYMENT-SERVICE

uri: lb://PAYMENT-SERVICE

predicates:

- Path=/payment/\*\*

filters:

- name: CircuitBreaker

args:

name: PAYMENT-SERVICE

fallbackuri: forward:/paymentServiceFallBack

- id : PRODUCT-SERVICE

uri: lb://PRODUCT-SERVICE

predicates:

- Path=/product/\*\*

filters:

- name: CircuitBreaker

args:

name: PRODUCT-SERVICE

fallbackuri: forward:/productServiceFallBack

## Step 3: Add the Filter into main class for Circuit Breaker

package com.example.demo.cloudgateway;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.circuitbreaker.resilience4j.Resilience4JCircuitBreakerFactory;

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

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

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

import org.springframework.context.annotation.Bean;

import io.github.resilience4j.circuitbreaker.CircuitBreakerConfig;

@SpringBootApplication

@EnableEurekaClient

public class CloudGatewayApplication {

public static void main(String[] args) {

SpringApplication.run(CloudGatewayApplication.class, args);

}

@Bean

public Customizer<Resilience4JCircuitBreakerFactory> defaultCustomizer() {

return factory -> factory.configureDefault(

id -> new Resilience4JConfigBuilder(id)

.circuitBreakerConfig(

CircuitBreakerConfig.ofDefaults()

)

.build()

);

}

}

## Step 4: Add the FallBackController for FallBack Methods

package com.example.demo.cloudgateway.controller;

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

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

@RestController

public class FallBackController {

@GetMapping("orderServiceFallBack")

public String orderServiceFallBack()

{

return "order service is down";

}

@GetMapping("paymentServiceFallBack")

public String paymentServiceFallBack()

{

return "Payment service is down";

}

@GetMapping("productServiceFallBack")

public String productServiceFallBack()

{

return "product service is down";

}

}

**Pre step:** You can stop the Order service.

URL::::

<http://localhost:9090/order/4>

Get method

Output :

**order service is down**

# ============================================================

# Update project 4: Implement Circuit Breaker into Order Service project

## Step 1: Add the Dependencies for Circuit Breaker (with out reactor)

<dependency>

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

<artifactId>spring-cloud-starter-circuitbreaker -resilience4j</artifactId>

</dependency>

## Step 2: Add the Circuit Breaker added into PaymentService

package com.example.demo.orderservice.external.client;

import org.springframework.cloud.openfeign.FeignClient;

import org.springframework.http.ResponseEntity;

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

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

import com.example.demo.orderservice.exception.CustomOrderException;

import com.example.demo.orderservice.external.request.PaymentRequest;

import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;

@CircuitBreaker(name="external",fallbackMethod = "fallback")

@FeignClient(name="PAYMENT-SERVICE/payment")

public interface PaymentService {

@PostMapping

public ResponseEntity<Long> doPayment(@RequestBody PaymentRequest paymentRequest);

default void fallback(Exception e)

{

throw new CustomOrderException("Payment service is not available", "UNAVAILABLE", 500);

}

}

## Step 3: Add the Circuit Breaker added into ProductService

package com.example.demo.orderservice.external.client;

import org.springframework.cloud.openfeign.FeignClient;

import org.springframework.http.ResponseEntity;

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

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

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

import com.example.demo.orderservice.exception.CustomOrderException;

import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;

@CircuitBreaker(name="external",fallbackMethod = "fallback")

@FeignClient(name="PRODUCT-SERVICE/product")

public interface ProductService {

@PutMapping("/reduceQuantity/{id}")

ResponseEntity<Void> reduceQuantity(@PathVariable("id") long productId,

@RequestParam long quantity);

default void fallback(Exception e)

{

throw new CustomOrderException("Product service is not available", "UNAVAILABLE", 500);

}

}

## Step 4: Add the resilience4j circuit breaker configured into application.yaml

server:

port: 8082

spring:

datasource:

url: jdbc:mysql://${DB\_HOST:localhost}:3306/orderdb

username: root

password:

driver-class-name: com.mysql.cj.jdbc.Driver

jpa:

database-platform: org.hibernate.dialect.MySQL57InnoDBDialect

hibernate:

ddl-auto: update

application:

name: ORDER-SERVICE

config:

import: configserver:${CONFIG\_SERVER\_URL:http://localhost:9296}

resilience4j:

circuitbreaker:

instances:

external:

event-consumer-buffer-size: 10

failure-rate-threshold: 50

minimum-number-of-calls: 5

automatic-transition-from-open-to-half-open-enabled: true

wait-duration-in-open-state: 5s

permitted-number-of-calls-in-half-open-state: 3

sliding-window-size: 10

sliding-window-type: COUNT\_BASED

#eureka:

# instance:

# prefer-ip-address: true

# client:

# fetch-registry: true

# register-with-eureka: true

# service-url:

# defaultZone: ${EUREKA\_SERVER\_ADDRESS:http://localhost:8761/eureka}

#

#

# ============================================================

# Update project 4: Implementing Rate Limiter in API Gateway using Resilience4j and Redis

## Step 1: Redis to start into local by using docker

https://hub.docker.com/\_/redis

docker run --name latestredis -d -p6379:6379 redis

## Step 2: Add the Dependencies for Redis-reactive

<dependency>

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

<artifactId>spring-boot-starter-data-redis-reactive</artifactId>

</dependency>

## Step 3: Add the Circuit Breaker added into PaymentService

package com.example.demo.cloudgateway;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.circuitbreaker.resilience4j.Resilience4JCircuitBreakerFactory;

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

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

import org.springframework.cloud.gateway.filter.ratelimit.KeyResolver;

import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

import org.springframework.context.annotation.Bean;

import io.github.resilience4j.circuitbreaker.CircuitBreakerConfig;

import reactor.core.publisher.Mono;

@SpringBootApplication

@EnableEurekaClient

public class CloudGatewayApplication {

public static void main(String[] args) {

SpringApplication.run(CloudGatewayApplication.class, args);

}

@Bean

KeyResolver userKeySolver()

{

return exchange -> Mono.just("userkey");

}

@Bean

public Customizer<Resilience4JCircuitBreakerFactory> defaultCustomizer() {

return factory -> factory.configureDefault(

id -> new Resilience4JConfigBuilder(id)

.circuitBreakerConfig(

CircuitBreakerConfig.ofDefaults()

)

.build()

);

}

}

## Step 4: Add the RequestRateLimiter in application.yaml

server:

port: 9090

spring:

application:

name: API-GATEWAY

config:

import: configserver:${CONFIG\_SERVER\_URL:http://localhost:9296}

cloud:

gateway:

routes:

- id : ORDER-SERVICE

uri: lb://ORDER-SERVICE

predicates:

- Path=/order/\*\*

filters:

- name: CircuitBreaker

args:

name: ORDER-SERVICE

fallbackuri: forward:/orderServiceFallBack

- name: RequestRateLimiter

args:

redis-rate-limiter.replenishRate: 1

redis-rate-limiter.burstCapacity: 1

- id : PAYMENT-SERVICE

uri: lb://PAYMENT-SERVICE

predicates:

- Path=/payment/\*\*

filters:

- name: CircuitBreaker

args:

name: PAYMENT-SERVICE

fallbackuri: forward:/paymentServiceFallBack

- name: RequestRateLimiter

args:

redis-rate-limiter.replenishRate: 1

redis-rate-limiter.burstCapacity: 1

- id : PRODUCT-SERVICE

uri: lb://PRODUCT-SERVICE

predicates:

- Path=/product/\*\*

filters:

- name: CircuitBreaker

args:

name: PRODUCT-SERVICE

fallbackuri: forward:/productServiceFallBack

- name: RequestRateLimiter

args:

redis-rate-limiter.replenishRate: 1

redis-rate-limiter.burstCapacity: 1

URL::::

<http://localhost:9090/order/4>

Get method 🡺hit the multiple times with in 1 second

Output :

**HTTP status code :::429 Too Many Requests**

# ============================================================

# Update project 4: Implementing Rate Limiter in API Gateway using Resilience4j and Redis