Proiect AWBD

# Echipa

Vasu Ionut

Adam Alexandru-Vlad

# Introducere

Am realizat un magazin online care se ocupa de vanzarea de articole sportive. In faza de alpha, acest are implementat doar serviciile de produse, comanda si de utilizatori, urmand ca pe parcusul dezvoltarii ulterioare sa fie incluse si alte servicii precum cel de cos de cumparaturi sau de metoda de plata.

# Servicii

Proiectul consta in 3 microservicii:

* User
* Order
* Product

Microserviciile implementeaza operatiile de tip CRUD si sunt proiectate sa fie modulare si individuale. Exista o interdependinta intre produs si comanda, cat si utilizator.

De asemenea, in cadrul aplicatiei mai sunt incluse:

* Server de configurare care seteaza nivelul de TVA pentru microserviciul de comanda. Produsele nu contin TVA, urmand ca acest cost sa fie adaugat dupa efectuarea unei comenzi.
* Eureka naming server, pentru a inregistra microserviciile

Ca functionalitati aditionale, aplicatia mai dispune de:

* Server SWAGGER pentru documentarea API-ului
* Feign Rest client pentru comunicarea intre microservicii
* Fault tolerance, folosind Resilience4j
* HATEOAS
* Erori custom

Momentan, singura interdependinta se afla intre serviciul Comanda si serviciul Produs. Cu ajutor a Feign Rest Client, comunicarea dintre acestea este realizata. Cand se va crea o comanda se vor transmite in format JSON catre serviciu o lista de id-uri de produse cu cantitatile respective, iar serverul va interoga microserviciul de produse pentru a lua preturile actualizate. Dupa, aceasta le va insuma si va adauga la pretul total cota de TVA stabilita de serverul de configurare

# Server configurare

Acesta tine diferite configuratii. La nivel de microservicii doar cota de TVA este tinuta pentru serviciul de Comanda, desi pe viitor se poate extinde functionalitatea acestuia.

De asemenea, mai sunt tinute configuratii pentru descoperirea de servicii la nivel de eureka naming server.

cost.tva=0.21  
server.port=8081  
eureka.client.register-with-eureka=true  
eureka.client.fetch-registry=true  
eureka.client.service-url.defaultZone=http://localhost:8071/eureka/  
eureka.instance.prefer-ip-address=true  
management.info.env.enabled=true  
info.app.name=order  
info.app.description=order  
info.app.version=1

In mod similar este configurat si serverul de discovery

server.port=8071  
eureka.instance.host=127.0.0.1  
eureka.client.register-with-eureka=false  
eureka.client.fetch-registry=false  
eureka.client.service-url.defaultZone=http://${eureka.instance.host}:${server.port}/eureka/

# Eureka naming server

Acest server are rolul de a descoperii si intregistra servicii. Se pot consulta mai multe informatii pe dashboard-ul acestuai

# Fault tolerance

Am implementat doar pentru o metoda cu scop demonstrativ.

@GetMapping("/order/{orderId}")  
@CircuitBreaker(name="orderId", fallbackMethod = "getOrderFallback")  
public Order getOrder(@PathVariable Long orderId) {  
  
 Order order = orderService.findById(orderId);  
 return order;  
  
}  
  
private Order getOrderFallback(Long orderId, Throwable throwable) {  
 Order order = orderService.findById(orderId);  
 return order;  
}

Se poate vedea ca implementarea este exact la fel.

# Feign Rest Client

Acesta este folosit pentru a comunica intre Comanda si Produs (pentru a lua produsele).

@FeignClient(value = "product", url="localhost:8080")  
public interface ProductServiceProxy {  
 @GetMapping("/product/{productId}")  
 public Product getProduct(@PathVariable Long productId);  
}

Am scris un proxy care sa ia produsele respective dupa id si sa returneze entitatea din baza de date pentru a fi procesata de server.

# Swagger, HATEOAS, error handling

Toate proiectele contin documentatie swagger, HATEOAS cat si error handling. Cateva exemple pot fi vazute mai jos.

@ControllerAdvice  
@RestController  
public class CustomizedExceptionHandler extends ResponseEntityExceptionHandler {  
 @ExceptionHandler(OrderNotFound.class)  
 protected ResponseEntity<Object> handleException(RuntimeException ex, WebRequest request) {  
 ExceptionPattern exception = new ExceptionPattern(new Date(), ex.getMessage(), request.getDescription(true));  
  
 return new ResponseEntity(exception, HttpStatus.*NOT\_FOUND*);  
 }  
  
  
 @Override  
 protected ResponseEntity<Object> handleMethodArgumentNotValid  
 (MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus status, WebRequest request) {  
 ExceptionPattern exception = new ExceptionPattern(new java.util.Date(), ex.getMessage(), request.getDescription(true));  
 return new ResponseEntity(exception, HttpStatus.*BAD\_REQUEST*);  
 }  
}

@SpringBootApplication  
@EnableFeignClients  
@EnableHypermediaSupport(type = EnableHypermediaSupport.HypermediaType.*HAL*)  
@RefreshScope  
@EnableDiscoveryClient  
public class OrderApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(OrderApplication.class, args);  
 }  
  
}