REPORT

“음식배달”

과제 URL : https://github.com/rayseo88/food-delivery-jh

과정 : Cloud Developing Level 2

작성일 : 2022.11.22

작성자 : 서 정 호

목 차

[서비스 시나리오 3](#_Toc120026354)

[체크포인트 5](#_Toc120026355)

[분석/설계 6](#_Toc120026356)

# 서비스 시나리오

**기능적 요구사항**

1. 고객이 평점 높은 메뉴를 조회한다.
2. 고객이 메뉴를 선택하여 주문한다
3. 고객이 선택한 메뉴에 대해 결제한다
4. 주문이 되면 주문 내역이 입점상점주인에게 주문정보가 전달된다
5. 상점주는 주문을 수락하거나 거절할 수 있다
6. 상점주는 요리시작 때와 완료 시점에 시스템에 상태를 입력한다.
7. 고객이 아직 요리가 시작되지 않은 주문을 취소할 수 있다
8. 요리가 완료되면 고객의 지역 인근의 라이더 들에 의해 배송건 조회가 가능하다
9. 라이더가 해당 요리를 pick한 후, pick했다고 앱을 통해 통보한다.
10. 고객이 주문상태를 중간중간 조회한다
11. 주문상태가 바뀔 때 마다 카톡으로 알림을 보낸다
12. 고객이 요리를 배달 받으면 배송확인 버튼을 탭하여, 모든 거래가 완료된다
13. 고객이 주문/배송된 메뉴에 대해 평점을 입력한다.

**비기능적 요구사항**

1. 장애격리
   1. 상점관리 기능이 수행되지 않더라도 주문은 365일 24시간 받을 수 있어야 한다 Async (event-driven), Eventual Consistency

🡪 각 서비스의 이벤트와 policy들은 pub/sub 방식으로 구현하여 특정 서비스에서 이벤트가 발생하면 이를 구독하는 서비스 들은 Asynchronous 하게 동작하여 해당 이벤트에 대한 처리를 하게 되고, 결과적으로 처리 결과가 일치하게 되도록 함

* 1. 결제시스템이 과중되면 사용자를 잠시동안 받지 않고 결제를 잠시후에 하도록 유도한다 Circuit breaker, fallback

1. 성능
   1. 고객이 자주 상점관리에서 확인할 수 있는 배달상태를 주문시스템(프론트엔드)에서 확인할 수 있어야 한다 CQRS

🡪 customer service에서 view를 구현하여 order status를 조회할 수 있도록 구현

* 1. 배달상태가 바뀔때마다 카톡 등으로 알림을 줄 수 있어야 한다 Event driven

🡪 각 service에서 발생한 상태 변경 이벤트에 따라서 안내메시지를 구성하여 message를 전송하도록 구현

# 체크포인트

* Saga (Pub/Sub)

- order, store, deliver 마이크로 서비스 들은 기본적으로 pub/sub 방식으로 이벤트를 publish하고 구독하는 구조로 구현되어 있으며, 이로 인해 순간적인 데이터 또는 상태 불일치가 발생할 수 있으나, 결과적으로는 동일한 정보로 처리될 수 있도록 구현되어 있음.

front서비스에서 주문시,

    @PostPersist

    public void onPostPersist(){

        OrderPlaced orderPlaced = new OrderPlaced(this);

        orderPlaced.publishAfterCommit();

    }

store 서비스에서, 수락

    public void accept(){

        StoreAccepted storeAccepted = new StoreAccepted(this);

        storeAccepted.publishAfterCommit();

        setStatus("주문수락됨");

    }

* CQRS

- 고객은 주문 후에 수시로 상태를 조회할 수 있어야 하고, 또 별개로 주문에 앞서 메뉴를 고르기 위해 메뉴별 평점을 조회할 수 있어야 한다. 이를 위해 read model(view) 방식으로 orderStatus view와 TopFood view를 구현함

- 생성

@StreamListener(KafkaProcessor.INPUT)

    public void whenOrderPlaced\_then\_CREATE\_1 (@Payload OrderPlaced orderPlaced) {

        try {

            if (!orderPlaced.validate()) return;

            // view 객체 생성

            TopFood topFood = new TopFood();

            // view 객체에 이벤트의 Value 를 set 함

            topFood.setFoodId(orderPlaced.getFoodId());

            topFood.setCount(1L);

            topFood.setId(orderPlaced.getId());

            // view 레파지 토리에 save

            topFoodRepository.save(topFood);

        }catch (Exception e){

            e.printStackTrace();

        }

    }

- 업데이트

 @StreamListener(KafkaProcessor.INPUT)

    public void whenScoreAdded\_then\_UPDATE\_1(@Payload ScoreAdded scoreAdded) {

        try {

            if (!scoreAdded.validate()) return;

                // view 객체 조회

            Optional<TopFood> topFoodOptional = topFoodRepository.findById(scoreAdded.getOrderid());

            if( topFoodOptional.isPresent()) {

.                 TopFood topFood = topFoodOptional.get();

            // view 객체에 이벤트의 eventDirectValue 를 set 함

                topFood.setScore(scoreAdded.getScore() + topFood.getScore());

                topFood.setCount(1L + topFood.getScore());

                // view 레파지 토리에 save

                 topFoodRepository.save(topFood);

                }

        }catch (Exception e){

            e.printStackTrace();

        }

    }

3. 조회

@RepositoryRestResource(collectionResourceRel="topFoods", path="topFoods")

public interface TopFoodRepository extends PagingAndSortingRepository<TopFood, Long> {

}

* Compensation / Correlation

- 고객이 주문이후에 취소를 한 경우 OrderCanceled 이벤트를 publish해서 rollback 처리를 할 수 있도록 함

점주가 주문거부을 거부하면(storeOrder서비스) reject 이벤트를 발생시킴

    public void reject(){

        StoreRejected storeRejected = new StoreRejected(this);

        storeRejected.publishAfterCommit();

        setStatus("주문거부됨");

    }

이어서, front서비스의 cancel policy를 수행함.

    public static void cancel(StoreRejected storeRejected){

        repository().findById(storeRejected.getOrderid()).ifPresent(order->{

            order.setStatus("주문거부됨");

            repository().save(order);

         });

* Request / Response

- 주문 상태에 따라 메시지 알림을 보내기 위해 주문 정보를 가져오기 위해 order aggregate(서비스)로 Req/Resp 요청을 보냄

    public static void sendTalk(StoreAccepted storeAccepted){

        fooddeliveryjh.external.Order order = CustomerApplication.applicationContext

            .getBean(fooddeliveryjh.external.OrderService.class)

            .getOrder(storeAccepted.getId());

        NotificationLog notificationLog = new NotificationLog();

            notificationLog.setMessage("주문접수");

            notificationLog.setCustomerId(order.getCustomerId());

        repository().save(notificationLog);

    }

* Circuit Breaker

- 주문이 이뤄지고 결제 처리를 진행할 때 외부 결제 기관과의 인터페이스에서 부하가 걸릴 때, 결제 처리를 진행하지 않도록 조건을 지정함.

- 설정값 (application.yaml)

feign:

  hystrix:

    enabled: true

hystrix:

  command:

    default:

      execution.isolation.thread.timeoutInMilliseconds: 100

- 구현

@FeignClient(name = "front", url = "${api.url.front}", fallback = OrderServiceImpl.class)

public interface OrderService {

    @RequestMapping(method= RequestMethod.GET, path="/orders/{id}")

    public Order getOrder(@PathVariable("id") Long id);

}

- fallback

@Service

public class OrderServiceImpl implements OrderService {

    /\*\*

     \* Fallback

     \*/

    public Order getOrder(Long id) {

        Order order = new Order();

        order.setCustomerId("");

        order.setId(id);

        return order;

    }

}

- 호출

    public static void sendTalk(StoreAccepted storeAccepted){

        fooddeliveryjh.external.Order order = CustomerApplication.applicationContext

            .getBean(fooddeliveryjh.external.OrderService.class)

            .getOrder(storeAccepted.getId());

        NotificationLog notificationLog = new NotificationLog();

            notificationLog.setMessage("주문접수");

            notificationLog.setCustomerId(order.getCustomerId());

        repository().save(notificationLog);

    }

* Gateway / Ingress

- Gateway 설정정보

spring:

  profiles: default

  cloud:

    gateway:

      routes:

        - id: front

          uri: http://localhost:8081

          predicates:

            - Path=/orders/\*\*, /payments/\*\*,

        - id: store

          uri: http://localhost:8082

          predicates:

            - Path=/storeOrders/\*\*, /menus/\*\*,

        - id: delivery

          uri: http://localhost:8083

          predicates:

            - Path=/deliveries/\*\*,

        - id: customer

          uri: http://localhost:8084

          predicates:

            - Path=/notificationLogs/\*\*, /evaluations/\*\*, /orderStatuses/\*\*, /topFoods/\*\*

        - id: frontend

          uri: http://localhost:8080

          predicates:

            - Path=/\*\*

      globalcors:

        corsConfigurations:

          '[/\*\*]':

            allowedOrigins:

              - "\*"

            allowedMethods:

              - "\*"

            allowedHeaders:

              - "\*"

            allowCredentials: true

- 결과

C:\Users\comtec>curl https://8088-rayseo88-fooddeliveryjh-spu2ocr84sq.ws-us77.gitpod.io/storeOrders

{

"\_embedded" : {

"storeOrders" : [ ]

},

"\_links" : {

"self" : {

"href" : "http://localhost:8082/storeOrders"

},

"profile" : {

"href" : "http://localhost:8082/profile/storeOrders"

},

"search" : {

"href" : "http://localhost:8082/storeOrders/search"

}

},

"page" : {

"size" : 20,

"totalElements" : 0,

"totalPages" : 0,

"number" : 0

}

}

C:\Users\comtec>

# 분석/설계

* MSAEz 로 모델링한 결과

https://labs.msaez.io/#/storming/84KolU4aNjYR6Tz89wgTmeGige43/e19dd43d7a94246f1a61324a831df794

**완성된 모델**

