포장 기능 추가.

쿠폰 전체에서 쓸 수 있는지, 특정 가게에 대한 건지 추가.

프론트에서 대기열 확인 추가.

부하테스트 추가.

스케일 아웃 확장도 해보기.

포장 기능에서 쿠폰 선택해서 결제하는 것까지.

나중에는 배달까지?

지금 할일.

1. 쿠폰 전체에서 쓸 수 있는지, 특정 가게에 대한 건지 추가.
2. 나중에 포장 기능 넣을거니까 프론트 넣을 때 나중에 확장하기 편하게 설계하기.
3. 프론트 넣고 테스트해보기.
4. 부하테스트로 대기열 제대로 되는지 확인하기.
5. 트래픽 늘려서 얼마까지 견디나 확인하기.
6. 스케일 아웃방식으로 확장해보기.
7. 카프카 병렬성 확장해보기.

--

1. 포장 기능 추가하기. 쿠폰 가게 확인은 이 단계에서 해도 될 듯. 위에서는 부하테스트, 기능 테스트 중점으로.

이 기능 추가할 때 카프카로 설계해볼까?

실제 결제 api까지 넣기.

결제 api 구현할때는 실무처럼 보안과 트랜잭션에 신경쓰기.

1. 인증은 세션으로 하기. jwt는 프론트하기 귀찮으니.

쿠폰을 가게별로 저장할건데 연관관계를 매핑해야하나, 아니면 쿠폰에 가게 id만 저장할까?

shop 엔티티를 만들어줘.

shop 관리 하는 어드민 페이지.

order. 나중에 결제 api 추가할때.

장바구니. 특정 가게의 메뉴를 담을 테이블.

사용자가 현재 음식의 주문 상태, 준비 됐는지, 준비 완료인지 확인하는 테이블을 추가해야하나?

---

추후 추가.

배달 기능.

배달원 상태. – 배달원이 조작.

내일 할일.

쿠폰 두개로 나누기.

배달은 너무 많아지니, 포장까지만 하기.

주문 결제 장바구니 쿠폰까지 일단 크루드부터 하기.

근데, 프론트는 너무 부담이 되네.

gpt로 간단하게만 해야하나…

그냥 api만 만들까…

프론트에서 쿠폰 발급할 때 어떻게 쿠폰 종류를 선택할까?

쿠폰 발급 방식

프론트에서 백엔드 요청시.

리퀘스트 파람 : 쿼리가 노출됨. 사용자가 조작가능. 서버에서 검증해야함.

post – 사용자가 조작할 수 없어 안전.

쿠폰 검증 방식

레디스 + db를 사용한 중복검증.

redis만으로는 불안정하기 때문.

<!-- Thymeleaf 모델에 담긴 CouponType.id -->

<input type="hidden" id="couponTypeId" th:value="${couponType.id}" />

서버에서 프론트에 쿠폰 id를 넘겨줘야하는데.

무슨 쿠폰을 사용할지 알고?

쿠폰 발급을 어디서 할건지?

1. 전체 쿠폰 모아보기 창.
2. 각 가게별로 발급 받을 수 잇는 쿠폰 확인.

각 가게에서 유저가 받을 수 있는 쿠폰을 가게 메인에서 보여준다.

유저가 받을 수 있는지 아닌지는 어떻게 구분할까?

여러 쿠폰 중 하나만 받을 수 있게 하는 로직은?

일단 그냥 쿠폰 전부 받을 수 있게 할까. 떠있는 쿠폰은.

가게별로 가져오고, 전체창에서는 가게별로 정렬하기만 하면 되네.

그럼 주는 id는 정해졌고.

전체 페이지에서는 전체 쿠폰을 shop 별로 보여주기.

shop 페이지에서는 해다 가게에 쿠폰의 id를 페이지로 전달하기.

이건 shop 페이지 만들고 구현하기.

테스트시에는 전체 페이지에서 하기.

유저가 발급 받았는지 확인하는건 어떡하지?

해당 id를 가진 유저의 쿠폰의 valid를 조작해서 사용 여부 확인.

발급 여부 확인은 쿠폰이 있는지 없는지 확인.

발급 받았으면 더 발급 받지 못하게하기.

지금 할거는 서버에서 프론트로 타입 보내고, 프론트에서 유저가 선택한 타입이랑 user id 서버로 보내서 발급 로직 실행하기.

고려한 것.

실무에서는 어떤 방식으로 구현할지에 대해 고민했다.

api 주소에 typeid를 포함하는 방식. 쿼리 파라미터보다 restful한 방식으로 결정함.

확장성을 고려하면 json에 typeid를 저장하는 방식도 고려하였으나, 아직까지는 필요치 않아 추후 필요하다면 리팩토링 예정.

주소에 데이터를 포함하거나 쿼리파라미터를 사용하는 경우 사용자가 이를 변경하여 접근할 수 있음. 허가 되지 않은 접근을 방지하기 위해 service에 검증로직을 구현함.

처음에는 컨트롤러에 구현하였으나, 추후 재사용성을 고려하여, service에 검증로직을 이동하고, 전역 advice를 통해 서비스에서 exception을 발생했을 때 http 에러 응답을 전송하는 방식으로 변경함.

service에 검증 로직을 구현하고 컨트롤러는 간소화하는 방식으로 service의 재사용성을 높이고, 컨트롤러에서 중복 코드를 줄임.

사용자가 실제 사용할 때 어떤 오류가 발생할 수 있을지.

실제 운영 환경에서 어떤 문제가 발생할 수 있을지 고민함.

선착순 쿠폰 발급 기능의 경우 단시간에 높은 트래픽이 발생할 수 있음.

그래서 카프카, redis를 사용해 쿠폰 발급시 대기열을 구현하여 단시간에 높은 트래픽을 감당할 수 있도록 구현함.

k6로 부하테스트 시, 로컬 환경에서 운영체제의 포트번호 리소스 한도인 27000명의 트래픽을 10초간 입력했을 때 쿠폰이 오류 없이 발급됨을 확인하였음.

또한 대기열과 쿠폰 발급 구현시 멀티 스레딩으로 인한 문제가 발생할 수 있음을 생각하고, lua 스크립트를 사용해 원자적으로 재고 확인, 재고 차감, 쿠폰 발급 등의 로직을 구현하여 멀티스레딩 환경에서 오류를 방지하고, 쿠폰 발급 오류가 발생했을 때의 상황을 고려하여 예외 처리를 하였음.