Final-Project

Catch-A-Bite

Get Started Order Now

김정호, 박성철, 이주호, 이주희

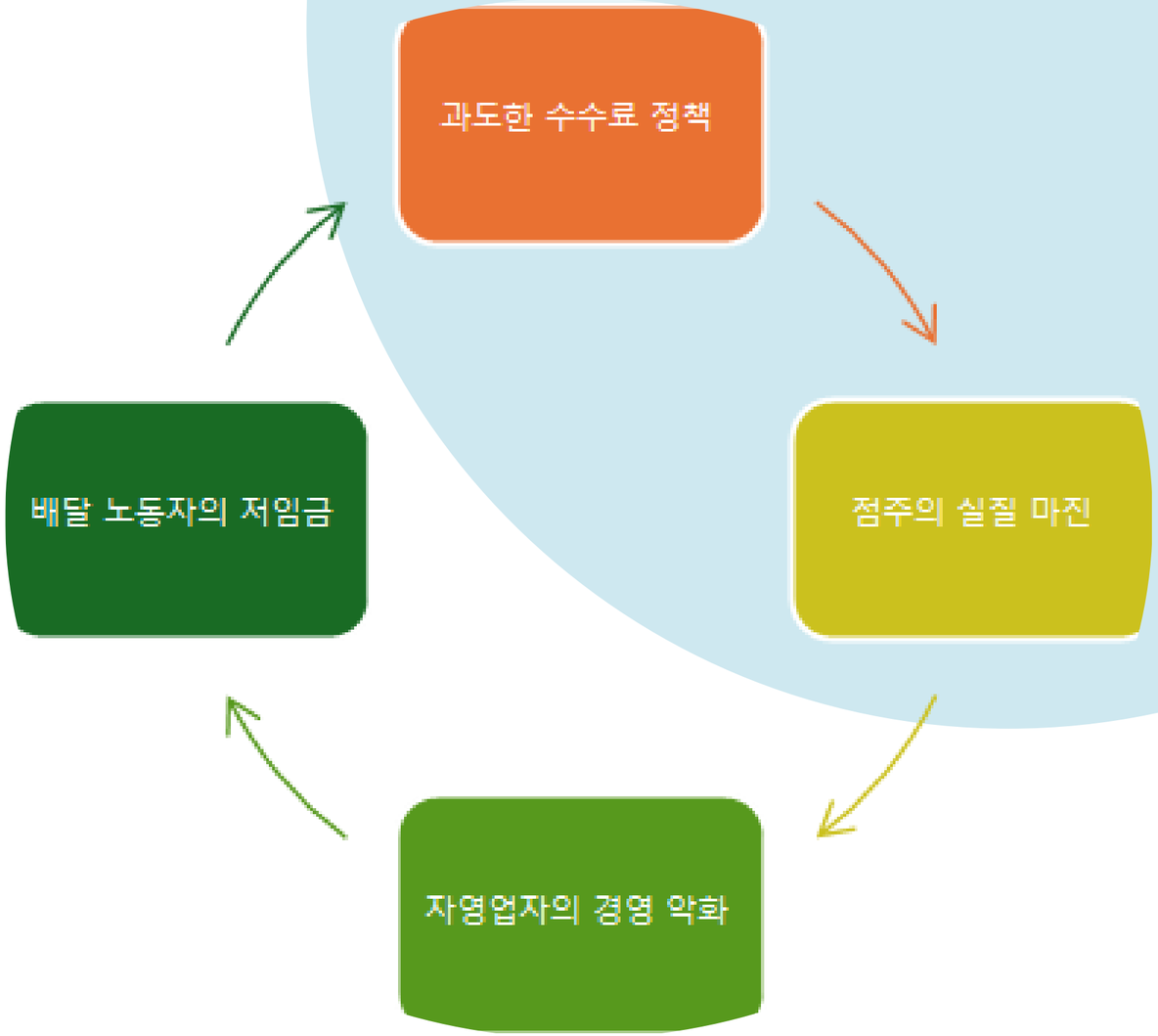
개발 배경

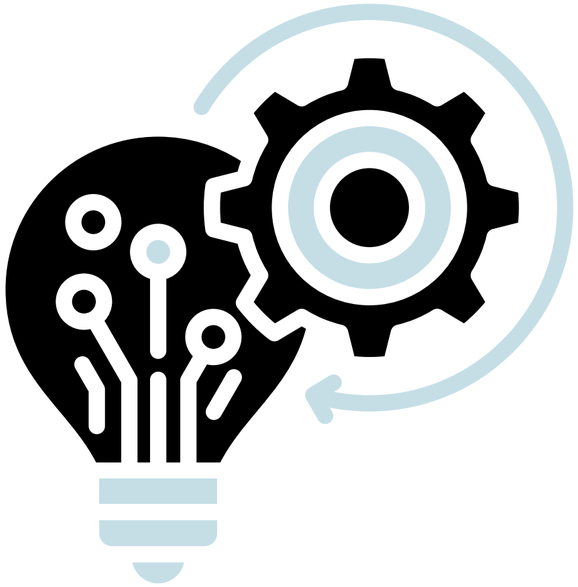
본 프로젝트는 현재 한국의 음식 배달 시장에서 배달의민족과 요기요 등 대형 플랫폼이 펼치고 있는 과도한 수수료 정책과 공격적인 시장 지배 행위로 인한 구조적 문제를 개선하고자 합니다.

기존 배달 플랫폼은 중개 수수료, 배달료 전가, 광고비 강제 등을 통해 점주의 실질 마진을 25~35% 이상 훼손하고 있으며, 이는 자영업자의 경영 악화, 소비자 음식값 인상, 배달 노동자의 저임금 악순환으로 이어지고 있습니다.

본 프로젝트는 이러한 현실을 직시하고, 기존 플랫폼의 한계를 보완하고 투명한 대안 플랫폼을 제시하려고 합니다.

Read More





개발 목표

가입 및 운영 프로세스 최적화를 통한
사업자 업무 효율 제고 및 수익성 강화

벤치마킹

개발환경

프론트엔드

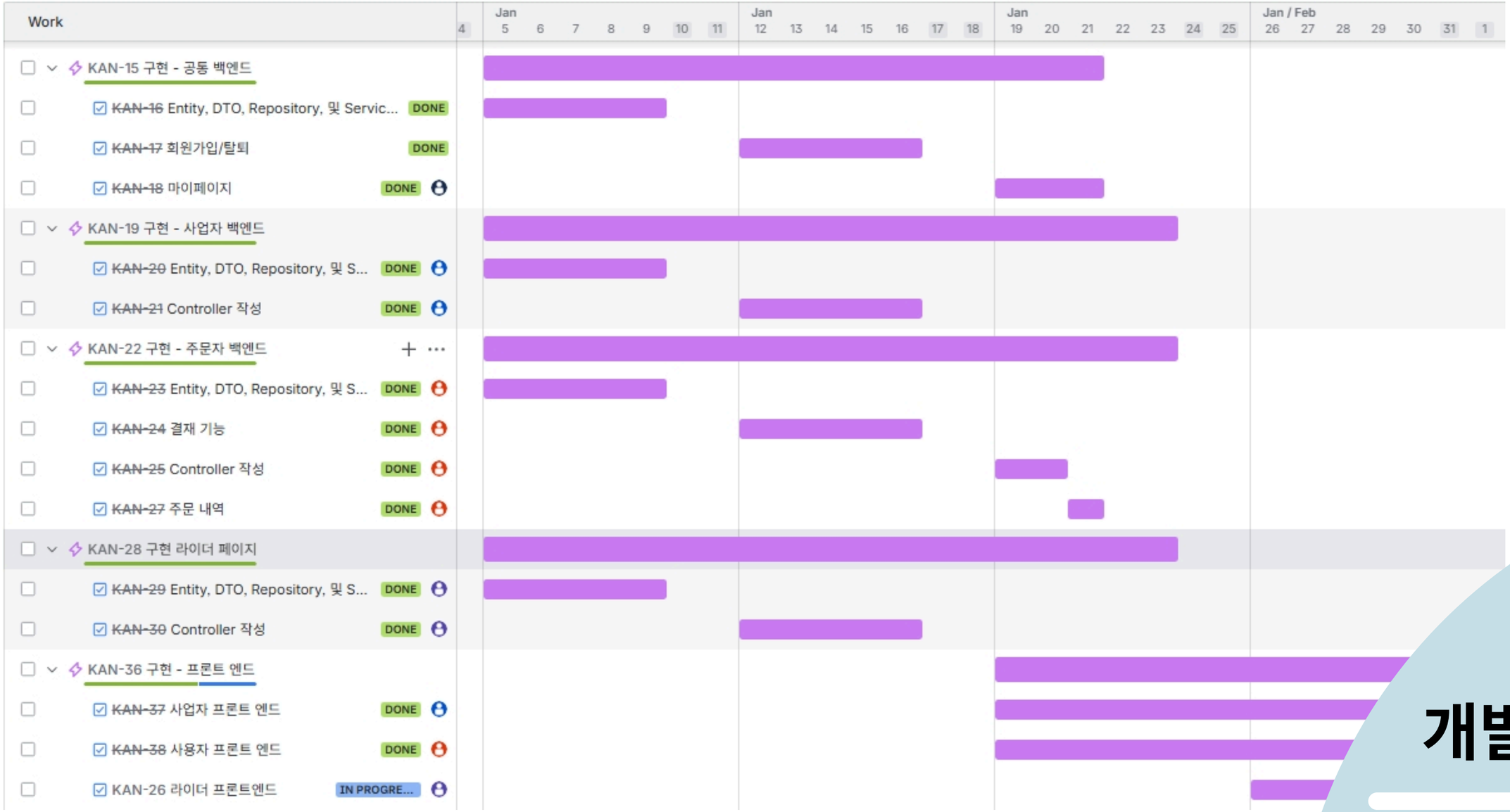
HTML
CSS
JavaScript
React
Axios

백엔드

Java
Spring Boot/Spring
Security
Lombok
MariaDB

개발도구

GitHub
Notion
ERMaster
Jira
Figma

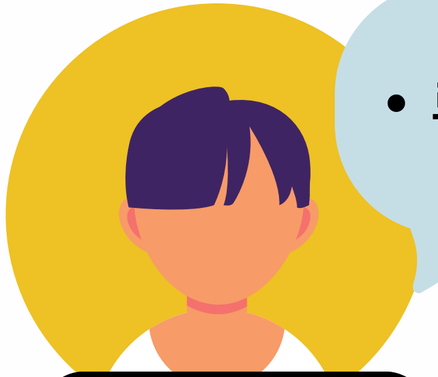

Work
Jan
4 5 6 7 8 9 10 11
Jan
12 13 14 15 16 17 18
Jan
19 20 21 22 23 24 25
Jan / Feb
26 27 28 29 30 31 1
KAN-15 구현 - 공통 백엔드
~~KAN-16~~ Entity, DTO, Repository, 및 Servic... DONE
~~KAN-17~~ 회원가입/탈퇴 DONE
~~KAN-18~~ 마이페이지 DONE
KAN-19 구현 - 사업자 백엔드
~~KAN-20~~ Entity, DTO, Repository, 및 S... DONE
~~KAN-21~~ Controller 작성 DONE
KAN-22 구현 - 주문자 백엔드
~~KAN-23~~ Entity, DTO, Repository, 및 S... DONE
~~KAN-24~~ 결재 기능 DONE
~~KAN-25~~ Controller 작성 DONE
~~KAN-27~~ 주문 내역 DONE
KAN-28 구현 라이더 페이지
~~KAN-29~~ Entity, DTO, Repository, 및 S... DONE
~~KAN-30~~ Controller 작성 DONE
KAN-36 구현 - 프론트 엔드
~~KAN-37~~ 사업자 프론트 엔드 DONE
~~KAN-38~~ 사용자 프론트 엔드 DONE
KAN-26 라이더 프론트엔드 IN PROGRE...

개발 일정

업무 분담

- 사용자 백엔드
- 사용자 가게관련 프론트엔드

김정호



- 로그인 및 회원가입

김진덕

- 사업자 풀스택

박성철

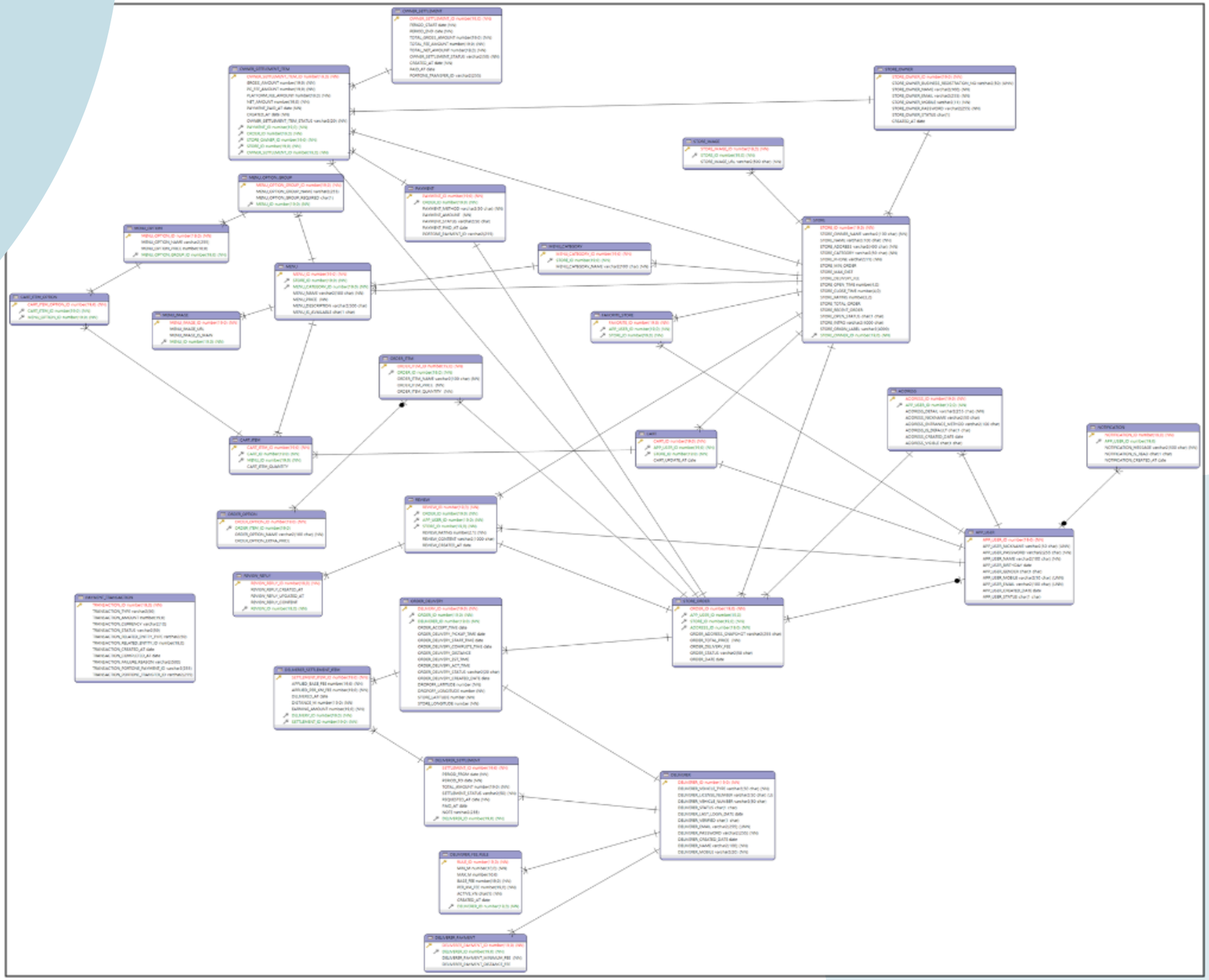
- 라이더 풀스택

이주호

- 사용자 메인페이지
- 마이페이지
- 검색

이주희

ERD



사용자 기능 정의서

	Location			Detail	
	1Depth	2Depth	3Depth	관련 DB	기능 설명
P					
	주소	주소 검색	GPS로 주소 찾기	APP_USER - 사용자가 누구인지 확인 ADDRESS - 주소 등록/수정/삭제	GPS로 주소 찾기 버튼을 클릭시 핸드폰의 GPS에게 위치 정보를 받아와서 KAKAO API에게 제공함. KAKAO API가 제공해준 주소를 뛰우고 상세정보를 요청함.
2			일반 주소로 검색/저장		사용자 마이페이지 주소관리에서 일반 또는 지번 주소를 작성 시 KAKAO API 에게 제공함. KAKAO API가 제공해준 주소를 뛰우고 상세정보를 요청함.
3			지번 주소로 검색/저장		
4			상세 정보 저장		사용자 주소의 상세정보를 ADDRESS 테이블의 ADDRESS_DETAIL에 저장함
5			주소 등록		ADDRESS를 테이블에 저장함. APP_USER_ID로 사용자와 연결함.
6			주소 불러오기		APP_USER가 주소를 성공적으로 가지고 옴.
7		주소 저장	사용자 홈에서 저정		사용자 홈에서 주소 버튼을 클릭 후 주소를 저장하는 것
8			마이페이지에서 저장		마이페이지에서 주소를 저장하는 것
9		배달 받을 주소 변경	홈페이지에서 변경		사용자 홈에서 주소 버튼을 클릭하여 주소를 변경하는 것
1	검색/분류	검색	글 검색	ADDRESS STORE STORE_IMAGE	검색창에 글을 작성 후 검색 버튼을 클릭 시 검색결과가 뜨는 것
2			음식 분류		사용자 홈 에서 음식 분류를 클릭 시 맞는 음식 분류만 뜨는 것
3			즐겨 찾기		즐겨찾기 버튼을 클릭 시 즐겨찾기에 설정한 음식점만 뜨는 것
4			검색 기록 보여주기		사용자 홈에서 검색 버튼을 클릭 시 검색기록이 있는 페이지가 뜨는 것
5			검색 기록 삭제		이전에 작성한 검색기록을 "x"버튼으로 삭제하는 것
6			최소 주문 기능		검색결과에서 각 필터로 맞는 음식점을 검색하는 것
1	가게	음식	음식 분류로 나누기	STORE, STORE_IMAGE, MENU MENU_CATEGORY, REVIEW MENU_OPTION_GROUP, MENU_OPTION	음식점의 모든 아이템 (메뉴)를 분류로 나누는 것
2			추가 옵션		메뉴 클릭시 추가 옵션이 뜨고 장바구니에 추가 시 맞는 옵션이 포함되는 것
3			장바구니에 추가하기		카트에 성공적으로 추가되는 것
4		리뷰	확인 가능		리뷰가 떠서 음식점의 리뷰를 확인할 수 있는 것
1	장바구니 및 주문	장바구니	장바구니에 메뉴 수 변경	APP_USER ADDRESS STORE MENU MENU_OPTION CART CART_ITEM PAYMENT	장바구니에서 메뉴 아이템의 수 변경하는 것
2			장바구니에 메뉴 삭제		장바구니에서 메뉴 아이템을 삭제하는 것
3			최소 주문 금액 확인		사용자가 주문할 수 있는 컨디션을 만족하는지 확인하는 것
4			최대 거리 확인		
5		결제	결제		API를 통하여 결재하고 사업자에게 알림을 보내는 것 (승인요청)
6		사업자에서 정보 받기 (시간을 많이 소모할 수 있음)	승인		사업자가 주문 승인 시 알림 받는 것
7		배달	배달완료 후 완료로 변경		배달원에게서 배달 받은 후 리뷰화면으로 변경되는 것
8			리뷰 작성 기능		리뷰가 성공적으로 작성되어 사업자 페이지에 보이는 것
9			리뷰 수정 기능		
10			주문내역에 저장		배달 완성 후 CART의 정보가 ORDER로 전달되는 것 -이전에는 카트에 유지

사업자 기능 정의서

No	Location			Detail
	1Depth	2Depth	3Depth	기능 설명
APP				
1	주소	주소 검색	GPS로 주소 찾기	GPS로 주소 찾기 버튼을 클릭시 핸드폰의 GPS에게 위치 정보를 받아와서 KAKAO API에게 제공함. KAKAO API가 제공해준 주소를 뛰우고 상세정보를 요청함.
2			일반 주소로 검색/저장	사용자 마이페이지 주소관리에서 일반 또는 지번 주소를 작성 시 KAKAO API에게 제공함. KAKAO API가 제공해준 주소를 뛰우고 상세정보를 요청함.
3			지번 주소로 검색/저장	
4			상세 정보 저장	사용자 주소의 상세정보를 ADDRESS 테이블의 ADDRESS_DETAIL에 저장함
5			주소 등록	ADDRESS를 테이블에 저장함. APP_USER_ID로 사용자와 연결함.
6			주소 불러오기	APP_USER가 주소를 성공적으로 가지고 옴.
7		주소 저장	사용자 홈에서 저장	사용자 홈에서 주소 버튼을 클릭 후 주소를 저장하는 것
8			마이페이지에서 저장	마이페이지에서 주소를 저장하는 것
9		배달 받을 주소 변경	홈페이지에서 변경	사용자 홈에서 주소 버튼을 클릭하여 주소를 변경하는 것
1	검색/분류	검색	글 검색	검색창에 글을 작성 후 검색 버튼을 클릭 시 검색결과가 뜨는 것
2			음식 분류	사용자 홈 에서 음식 분류를 클릭 시 맞는 음식 분류만 뜨는 것
3			즐겨 찾기	즐겨찾기 버튼을 클릭 시 즐겨찾기에 설정한 음식점만 뜨는 것
4			검색 결과 + 음식 분류	검색결과에서 음식 분류를 클릭 시 맞는 음식 분류만 뜨는 것
5			검색 기록 보여주기	사용자 홈에서 검색 버튼을 클릭 시 검색기록이 있는 페이지가 뜨는 것
6			검색 기록 삭제	이전에 작성한 검색기록을 "x"버튼으로 삭제하는 것
7			거리	검색결과에서 각 필터로 맞는 음식점을 검색하는 것
8			별 수	
9			트렌드	
10			총 주문 수	
11			최소 주문 기능	
1	가게관리	가게정보	기본정보	상호명, 소개글, 영업시간 수정
2			영업상태	영업중/준비중/마감 설정
3			배달조건	최소주문금액, 최대배달거리 등 설정
4		카테고리	관리	메뉴 분류 등록/수정/삭제
5		메뉴	등록	메뉴 등록
6			수정/삭제	메뉴 수정 및 판매중지
7		옵션	그룹	메뉴 옵션 그룹 관리
8			항목	메뉴 옵션 항목 관리
9		주문	목록 조회	실시간 주문 목록 조회
10			상세 조회	주문 상세 정보 조회(음식, 주문정보, 주소 등)
11			승인	주문 승인 및 조리 시작
12			거절	주문 거절 및 사유 전달
13		배달상태	조리완료	조리 완료 상태 변경
14			배달완료	배달 완료 처리
15		리뷰	조회	리뷰 목록 확인
16			댓글	리뷰 답글 작성
17		매출	조회	기간별 매출 조회
18		결제	내역	결제 및 정산 내역 확인

라이더
기능 정의서

No	Location			Page위치	Page Name	Detail	
	1Depth	2Depth	3Depth			관련 DB/Entity	페이지 설명
1	로그인	ID,PW 입력		라이더 로그인 페이지	rider_login	deliverer	라이더 전용 앱 실행시 초기화면, 로그인 후 [마이페이지]로 이동
2		ID 찾기 페이지로 이동	이메일 주소, 전화번호 입력	ID 찾기 페이지	rider_find_id		이메일 주소, 전화번호를 입력하면 ID 출력(모달창)
3		PW 찾기 페이지로 이동	ID와 이메일 주소 입력	PW 찾기 페이지	rider_find_pw		ID와 이메일 주소를 입력하면 PW 전송(이메일 주소로 전송)
4		회원가입 페이지로 이동	ID, PW, 이메일, 이름, 전화번호 입력	회원가입 페이지	rider_register		회원가입 페이지, 계정 생성 후 [마이페이지]로 이동
5	마이페이지	배달현황(조회)	(조회)로 이동	마이페이지	rider_myPage	deliverer, order_delivery	라이더의 현재 상태(대기/배달중)와 진행 중 배달의 단계별 상태를 조회
6		내 수입	나의 수입 표시(월별/일별)			deliverer _payment	수입을 일별/월별로 표시. 2순위
7		내 평점	나의 평점 표시			delivery_review	나의 종합 평점을 표시. 3순위
8	조회	대기(상태)	배차 수락(기능)	배달 현황 페이지	rider_main	deliverer, order_delivery	대기(조회) -> .. -> 배달 완료의 과정마다 라이더 현재 위치를 전송
9		진행(배차 수락)	픽업 완료(기능)	배달 접수 페이지	rider_progress	order_delivery	배달 과정 표시(라이더 화면_1순위). 고객이 배달 취소를 하면 [대기]로 이동(2순위)
10		배달지 도착(기능)					
11		배달 완료(기능)					
12		완료		배달 완료 페이지	rider_finish		배달 완료한 주문 표시. [대기]로 이동.

API정의서

No	카테고리	Method	URI	기능 설명	(Params/Body	Response (Data Type)
1	회원 관리	POST	/	사용자 회원가입	Body: AppUserDTO	AppUserDTO
2	회원 관리	GET	/{userId}	회원 정보 조회 (마이 페이지)	Path: userId	AppUserDTO
3	회원 관리	PUT	/{userId}	회원 정보 수정	Path: userId, Body: AppUserDTO	AppUserDTO
4	회원 관리	DELETE	/{userId}	회원 탈퇴	Path: userId	Void (Message)
5	가게	GET	/stores/search	키워드로 가게 검색	Query: keyword	List<UserStoreSummaryDTO>
6	가게	GET	/stores/category	카테고리별 가게 목록 조회	Query: storeCategory	List<UserStoreSummaryDTO>
7	가게	GET	/stores/random	랜덤 가게 추천 목록 조회	-	List<UserStoreSummaryDTO>
8	가게	GET	/stores/{storeId}	가게 상세 정보 조회 (메뉴, 리뷰 수 등)	Path: storeId, Header: Auth (Optional)	UserStoreResponseDTO
9	가게	GET	/stores/{storeId}/menus	가게 메뉴판(카테고리별 메뉴) 조회	Path: storeId	List<MenuCategoryWithMenusDTO>
10	메뉴	GET	/menus/{menuId}	메뉴 상세 정보(옵션 포함) 조회	Path: menuId	UserMenuDetailDTO
11	즐겨찾기	GET	/favorites	내 즐겨찾기(찜) 가게 목록 조회	Header: Auth (Required)	List<UserFavoriteStoreResponseDTO>
12	즐겨찾기	POST	/favorites	가게 즐겨찾기 추가	Body: FavoriteStoreDTO (storeId)	FavoriteStoreDTO
13	즐겨찾기	DELETE	/favorites/{favoriteId}	즐겨찾기 해제	Path: favoriteId	Void (Message)

사용자

주문 시점의 메뉴

가게 상세 정보, 카테고리별 메뉴 구성, 실시간 리뷰 수 등 흩어진 도메인 데이터를 하나의 DTO로 캡슐화하여 클라이언트의 복잡한 연산 로직을 제거

```java
@Override
public UserStoreResponseDTO getStoreDetailsForUser(Long storeId, String userLoginId) {
    // 1. 가게 조회
    Store store = storeRepository.findStoreWithCategoriesById(storeId)
            .orElseThrow(() -> new IllegalArgumentException(s: "존재하지 않는 가게id입니다."));

    // 2. 가게 대표 이미지 처리
    String storeImageUrl = null;
    if (store.getImages() != null && !store.getImages().isEmpty()) {
        storeImageUrl = store.getImages().get(index: 0).getStoreImageUrl();
    }

    // 3. 리뷰 수 조회
    Integer reviewCount = (int) reviewRepository.countByStore_StoreId(storeId);

    // 4. 메뉴 이미지 일괄 조회 (N+1 문제 방지 최적화)
    // 4-1. 가게의 모든 메뉴 ID 추출
    List<Long> allMenuIds = store.getMenuCategories().stream()
        .flatMap(cat -> cat.getMenus().stream())
        .map(Menu::getMenuId)
        .collect(Collectors.toList());

    // 4-2. 메뉴 ID로 이미지 조회
    List<MenuImage> menuImages = menuImageRepository.findByMenu_MenuIdIn(allMenuIds);

    // 4-3. 조회된 이미지를 Map으로 변환 (Key: MenuId, Value: ImageUrl)
    Map<Long, String> imageMap = menuImages.stream()
    .sorted((a, b) -> Boolean.compare(b.getMenuImageIsMain(), a.getMenuImageIsMain()))
    .collect(Collectors.toMap(
        img -> img.getMenu().getMenuId(),
        MenuImage::getMenuImageUrl,
        (existing, replacement) -> existing // 중복 시 첫 번째(정렬에 의해 대표이미지) 사용
    ));
```

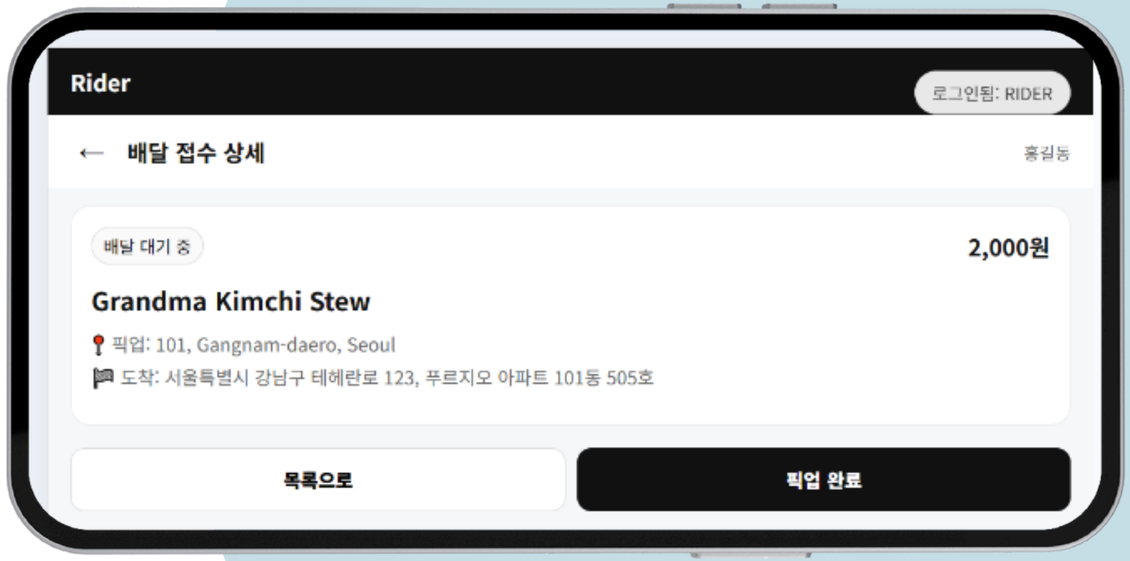
가게 상세 정보, 카테고리별 메뉴 구성

가게 상세 정보, 카테고리별 메뉴 구성, 실시간 리뷰 수 등 흩어진 도메인 데이터를 하나의 DTO로 캡슐화하여 클라이언트의 복잡한 연산 로직을 제거

```
/ 1. 결과를 담을 빈 리스트를 만듭니다.
ist<MenuCategoryWithMenusDTO> categoryDTOs = new ArrayList<>();

/ 2. 가게의 모든 카테고리를 하나씩 꺼냅니다.
or (MenuCategory category : store.getMenuCategories()) {

  // 2-1. 해당 카테고리에 속한 메뉴들을 담을 빈 리스트를 만듭니다.
  List<UserMenuImageDTO> menuDtos = new ArrayList<>();

  // 2-2. 카테고리 안의 메뉴들을 하나씩 꺼내서 변환합니다.
  for (Menu menu : category.getMenus()) {

      // 아까 만들어둔 지도(Map)에서 메뉴 ID에 맞는 이미지 URL을 가져옵니다.
      String imgUrl = imageMap.get(menu.getMenuId());

      // 메뉴 엔티티를 DTO로 변환합니다.
      UserMenuImageDTO menuDto = menuImageConverter.toDto(
              menu,
              storeId,
              category.getMenuCategoryId(),
              imgUrl
      );

      // 변환된 메뉴를 리스트에 담습니다.
      menuDtos.add(menuDto);
  }

  // 2-3. 메뉴 리스트가 완성되었으니, 카테고리 DTO를 만듭니다.
  MenuCategoryWithMenusDTO categoryDTO = MenuCategoryWithMenusDTO.builder()
          .menuCategoryId(category.getMenuCategoryId())
          .menuCategoryName(category.getMenuCategoryName())
          .menus(menuDtos) // 완성된 메뉴 리스트 연결
          .build();
```

라이더

배달접수 상세 페이지



DB.sql | DeliveryController.java 2, M

catch-a-bite > src > main > java > com > deliveryapp > catchabite > controller > DeliveryController.java > Language Support for Java(TM)

```java
25  public class DeliveryController {
57
58
59      // 매장에서 픽업완료(배달원)
60      @PostMapping("/{deliveryId}/pickup-complete")
61      public ResponseEntity<DeliveryApiResponseDTO<Void>> pickupComplete(
62              @PathVariable Long deliveryId,
63              Authentication authentication
64      ) {
65          if (authentication == null || authentication.getPrincipal() == null) {
66              return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
67                      .body(DeliveryApiResponseDTO.fail(message: "인증 정보가 없습니다."));
68          }
69
70          String principal = authentication.getPrincipal().toString();
71          deliveryService.pickupCompleteByPrincipal(deliveryId, principal);
72
73          return ResponseEntity.ok(DeliveryApiResponseDTO.success(message: "픽업이 완료되었습니다."));
74      }
```

DB.sql | OrderDeliveryService.java 1, M

catch-a-bite > src > main > java > com > deliveryapp > catchabite > service > OrderDeliveryService.java > Language Support for Java(TM) by

```java
25  public class OrderDeliveryService {
106     @Transactional
107     public void pickupCompleteByPrincipal(Long deliveryId, String principal) {
108         Long delivererId = resolveDelivererIdFromPrincipal(principal);
109         pickupComplete(deliveryId, delivererId);
110     }
111
112     @Transactional
113     public void pickupComplete(Long deliveryId, Long delivererId) {
114
115         OrderDelivery od = orderDeliveryRepository.findByIdForUpdate(deliveryId)
116             .orElseThrow(() -> new IllegalArgumentException(s: "배달이 없습니다."));
117
118         if (od.getDeliverer() == null || !od.getDeliverer().getDelivererId().equals(delivererId)) {
119             throw new SecurityException(s: "본인 배달이 아닙니다.");
120         }
121
122         if (od.getOrderDeliveryStatus() != DeliveryStatus.PENDING) {
123             throw new IllegalStateException(s: "픽업 완료는 대기(PENDING) 상태에서만 가능합니다.");
124         }
125
126         StoreOrder so = od.getStoreOrder();
127         if (so == null) throw new IllegalStateException(s: "주문 정보가 없습니다.");
128
129         if (so.getOrderStatus() != OrderStatus.COOKED) {
130             throw new IllegalStateException(s: "픽업 완료는 주문상태가 COOKED일 때만 가능합니다.");
131         }
132
133         od.setOrderDeliveryPickupTime(LocalDateTime.now());
134         od.setOrderDeliveryStartTime(LocalDateTime.now());
135         od.setOrderDeliveryStatus(DeliveryStatus.IN_DELIVERY);
136
137         so.changeStatus(OrderStatus.DELIVERING);
138
139         orderDeliveryRepository.save(od);
```

```java
public interface OrderDeliveryRepository extends JpaRepository<OrderDelivery, Long> {

    // 배달기사가 동시 수락/중복 수락을 막기위해 PESSIMISTIC_WRITE(비관적 락)를 이용
    // '비관적 락'은 트랜젝션이 끝나기 전까지는 해당 트랙젝션이 이용하고 있는 DB의 값을 변경하지 않고 놔두다가, 이용이 끝나면 변경한다
    @Lock(LockModeType.PESSIMISTIC_WRITE)
    @Query("select od from OrderDelivery od where od.deliveryId = :deliveryId")
    Optional<OrderDelivery> findByIdForUpdate(@Param("deliveryId") Long deliveryId);
```

배달완료 페이지

DB.sql | OrderDeliveryService.java 1, M

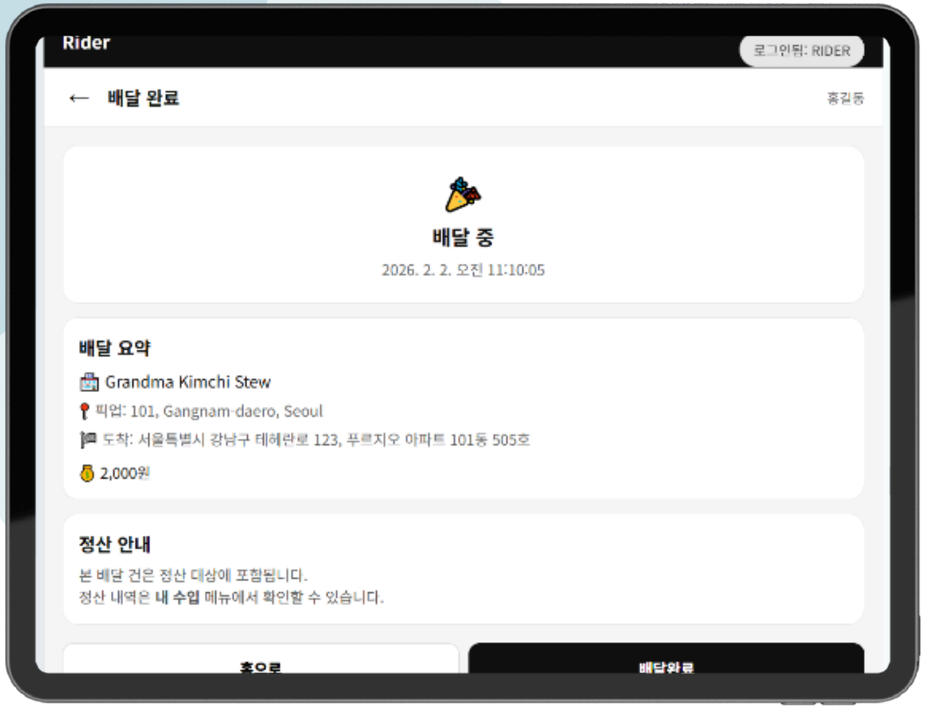
catch-a-bite > src > main > java > com > deliveryapp > catchabite > service > OrderDeliveryService.java > Language Support for Java(TM)

```java
public class OrderDeliveryService {

    @Transactional
    public void completeDeliveryByPrincipal(Long deliveryId, String principal) {
        Long delivererId = resolveDelivererIdFromPrincipal(principal);
        completeDelivery(deliveryId, delivererId);
    }

    @Transactional
    public void completeDelivery(Long deliveryId, Long delivererId) {

        OrderDelivery od = orderDeliveryRepository.findByIdForUpdate(deliveryId)
            .orElseThrow(() -> new IllegalArgumentException(s: "배달이 없습니다."));

        if (od.getDeliverer() == null || !od.getDeliverer().getDelivererId().equals(delivererId)) {
            throw new SecurityException(s: "본인 배달이 아닙니다.");
        }

        if (od.getOrderDeliveryStatus() != DeliveryStatus.IN_DELIVERY) {
            throw new IllegalStateException(s: "배달완료는 배달중(IN_DELIVERY) 상태에서만 가능합니다.");
        }

        od.setOrderDeliveryCompleteTime(LocalDateTime.now());
        od.setOrderDeliveryStatus(DeliveryStatus.DELIVERED);

        StoreOrder so = od.getStoreOrder();
        if (so == null) throw new IllegalStateException(s: "주문 정보가 없습니다.");

        if (so.getOrderStatus() != OrderStatus.DELIVERING) {
            throw new IllegalStateException(s: "배달완료는 주문상태가 DELIVERING인 때만 가능합니다.");
        }
        so.changeStatus(OrderStatus.DELIVERED);

    }
```

DB.sql | DeliveryController.java 2, M

catch-a-bite > src > main > java > com > deliveryapp > catchabite > controller > DeliveryController.java > Language Support for Java(TM

```java
public class DeliveryController {

    @PostMapping("/{deliveryId}/complete")
    public ResponseEntity<DeliveryApiResponseDTO<Void>> complete(
            @PathVariable Long deliveryId,
            Authentication authentication
    ) {
        if (authentication == null || authentication.getPrincipal() == null) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                    .body(DeliveryApiResponseDTO.fail(message: "인증 정보가 없습니다."));
        }

        String principal = authentication.getPrincipal().toString();
        deliveryService.completeDeliveryByPrincipal(deliveryId, principal);

        return ResponseEntity.ok(DeliveryApiResponseDTO.success(message: "배달이 완료되었습니다."));
    }
```



사업자

사업자 메뉴 · 옵션 관리

배달앱에서 가장 복잡한 메뉴·옵션을 옵션 그룹 단위로 분리 설계하여 확장 가능하게 구현


```java
// 메뉴 등록
@PostMapping
public ResponseEntity<ApiResponse<MenuDTO>> createMenu(Principal principal,
                    @PathVariable Long storeId,
                    @RequestBody MenuDTO dto) {

  Long storeOwnerId = ownerContext.requireStoreOwnerId(principal);

  MenuDTO result = menuService.createMenu(storeOwnerId, storeId, dto);
  return ResponseEntity.ok(ApiResponse.ok(result));
}
```

```java
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "store_id", nullable = false)
private Store store;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "menu_category_id", nullable = false)
private MenuCategory menuCategory;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "menu_option_group_id", nullable = false)
private MenuOptionGroup menuOptionGroup;

@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "menu_id", nullable = false)
private Menu menu;

@OneToMany(mappedBy = "menuOptionGroup", fetch = FetchType.LAZY)
@BatchSize(size = 100)
@Builder.Default
private List<MenuOption> menuOptions = new ArrayList<>();
```

사업자 주문 관리 (상태 변경)

주문 상태를 기준으로 사업자가
빠르게 처리할 수 있도록
목록 중심 UX와 상태 변경
흐름을 END-TO-END로 구현


```java
// 9-1) 주문 목록 조회 (페이징/기간필터/정렬 추가)
@GetMapping
public ResponseEntity<ApiResponse<PageResponseDTO<OwnerOrderDTO>>> list(
        Principal principal,
        @PathVariable Long storeId,
        @RequestParam(required = false) String status,
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "20") int size,
        @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate from,
        @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate to
) {
    Long storeOwnerId = ownerContext.requireStoreOwnerId(principal);

    Pageable pageable = PageRequest.of(
            page,
            size,
            Sort.by(Sort.Direction.DESC, ...properties: "orderDate") // StoreOrder 필드명 기준
    );

    return ResponseEntity.ok(ApiResponse.ok(
            ownerOrderService.listOrders(storeOwnerId, storeId, status, from, to, pageable)
    ));
}
```

```java
public enum OrderStatus {

    PAYMENTINPROGRESS(value: "payment_in_progress"),
    PAYMENTFAILED(value: "payment_failed"),
    PAYMENTCONFIRMED(value: "payment_confirmed"),
    PENDING(value: "pending"),
    REJECTED(value: "rejected"),
    COOKING(value: "cooking"),
    COOKED(value: "cooked"),
    DELIVERING(value: "delivering"),
    DELIVERED(value: "delivered");

    private final String value;

    OrderStatus(String value) {
        this.value = value;
    }

    /** API 응답 / 화면 표시용 */
    public String getValue() {
        return value;
    }

    /** 외부 입력(String)을 enum으로 변환 */
    public static OrderStatus from(String value) {
        return OrderStatus.valueOf(value.trim().toUpperCase());
    }
}
```

시연

트러블 슈팅

PORTONE 키를 생성하기 위해서는 주문 내역 DATABASE ID가 필요한데, STOREORDERID가 FETCH TYPE LAZY라 데이터가 존재하지 않았습니다.

FETCH TYPE EAGER로 변경하여 문제를 해결하였습니다.

1.4 오류 메시지 (Error Messages)

- 사용자 화면/클라이언트 증상: 서버가 실행되지 않음
- 서버 로그 핵심 메시지 (원문 붙여넣기):

```
WARN 2248 --- [catchabite] [p-nio-80-exec-1] o.m.jdbc.message.server.ErrorPacket      : Error: 1054-
42S22: Unknown column 'au1_0.app_user_login_id' in 'SELECT'
WARN 2248 --- [catchabite] [p-nio-80-exec-1] o.h.engine.jdbc.spi.SqlExceptionHelper   : SQL Error: 1
054, SQLState: 42S22
ERROR 2248 --- [catchabite] [p-nio-80-exec-1] o.h.engine.jdbc.spi.SqlExceptionHelper   : (conn=921)
Unknown column 'au1_0.app_user_login_id' in 'SELECT'
INFO 2248 --- [catchabite] [p-nio-80-exec-1] o.h.e.internal.DefaultLoadEventListener  : HHH000327: E
rror performing load command
```

- 에러 코드/HTTP 상태: 400
- 첨부:

```
{
  "success": false,
  "message": "오류가 발생했습니다.",
  "payment_id": null,
  "order_id": null,
  "imp_uid": null,
  "payment_status": null,
  "payment_amount": null,
  "payment_method": null,
  "payment_paid_at": null,
  "error_code": "PAYMENT_PREPARATION_ERROR",
  "error_message": "Unexpected error during payment preparation"
}
```

Q&A

소감발표

김정호

진행 속도를 중요시하다 보니 소통 과정에서 놓치는 부분들이 있었음에도 불구하고,
팀원분들의 배려와 적극적인 협력 덕분에 프로젝트를 잘 마무리할 수 있었습니다.
부족한 점을 채워주신 팀원분들께 깊이 감사드립니다

박성철

8명이 시작해 4명으로 끝난 여정이었습니다.
고생해주신 모든 팀원분들께 감사의 말씀 올리겠습니다.

이주호

실제로 볼 수 있는 UI화면을 바탕으로 기능을 추가했어야 했는데, 초기에 백엔드 먼저 작업하느라,
프론트엔드 개발에 미흡했습니다. 인원이 중간에 빠지긴 했어도, 제가 담당하는 부분은
프론트엔드도 같이 개발했다면 지금과는 결과가 달랐을 것이라고 생각합니다.

이주희

팀원들이 잘 이끌어 준 덕분에 잘 마무리 할 수 있었습니다.