

강남 더조은컴퓨터학원 K-Digital Training 풀스택 과정

신발가게 API

블랙마켓_APP

2조 다다익선

팀장: 이창준

팀원: 김수아

팀원: 강인환

팀원: 이상훈



CONTENTS

- 1 개요
- 2 팀 구성 및 역할
- 3 프로젝트 진행 과정
- 4 수행 결과
- 5 자체 평가 의견

01

개요

주제 및 기획 의도 :

- 신발가게 어플을 만들고 각자의 유저가 원하는 기능을 만들 수 있다

활용 장비 및 재료 :

- Python, Flutter(Dart), MySQLWorkbench

프로젝트 구조 :

- Python을 이용해 백엔드 서버를 구축하고 Query가 필요한 기능들을 추가한뒤 Flutter를 이용해 화면 디자인과 http를 이용해 필요한 정보를 불러와 MySQLWorkbench에 저장되는 앱을 구현



기대 효과 :

- Flutter & Dart, SQLite, API 연동, UI/UX 설계 등 실무형 기술을 폭넓게 경험
- 데이터 모델링, 상태관리, 다단계 결재 구조 등 복잡한 로직을 직접 구현 → 엔지니어링 자신감 향상
- 앱 전체 구조(본사–대리점–고객)의 설계 경험 → 앱 설계 능력 체득
- 실제 기업 운영에 적용되는 시스템을 스스로 리딩 → 기획 + 개발 + 운영까지 경험
- 러 테이블 간 관계 설계, 사용자의 업무 흐름 고려 → 비즈니스 로직 구현 능력 향상
- 은 기능부터 전체 앱 아키텍처까지 스스로 책임지는 경험 → 전문성과 신뢰도 상승

01

개요

구현 내용 :

- 신발가게 회원, 대리점 점주, 본사 직원이 각각 신발 가게 (BlackMarket) 을 이용하기 위한 기능들을 구현 한 앱과 앱에 연결되는 데이터베이스를 구축 하였습니다.

컨셉 :

- 일관성 있는 디자인과 가시성을 확보 할 수 있고 남녀노소 모두 거부감 없이 접근 할 수 있는 검은색과 흰색의 조합으로 구성 하였으며 이를 투영하여 앱의 이름을 Black Market 이라고 작명하게 되었습니다.



관련성 :

- 팀으로 함께 앱을 제작함으로써 팀원들과 함께 협업하는 경험을 쌓고 성장할 수 있었습니다.
- 앱에 필요한 기능을 구현하기 위해 그리고 앱의 구동에 따라 저장 및 이동되는 data 를 컨트롤 하기 위하여 전반적인 앱 개발 능력과 database 활용 능력을 기를 수 있었습니다.
- 강의 시간 동안 배운 내용을 활용하는 능력을 기르고 스스로 부족했던 부분에 대해서 공부하여 성장 할 수 있었습니다.

02

팀 구성



팀장 : 이창준

- 전반적인 GITHUB 관리
- 로그인 및 회원가입 화면 구현
- 고객의 물품 구매 관련, 장바구니, 구매 내역, 공지사항 확인 화면 구현
- 제작한 화면들을 데이터베이스를 통한 데이터 이동 및 저장 기능 구현



팀원 : 강인환

- 대리점에 필요한 백엔드와 프론트엔드 구성
- DB에 들어가는 정보 수집

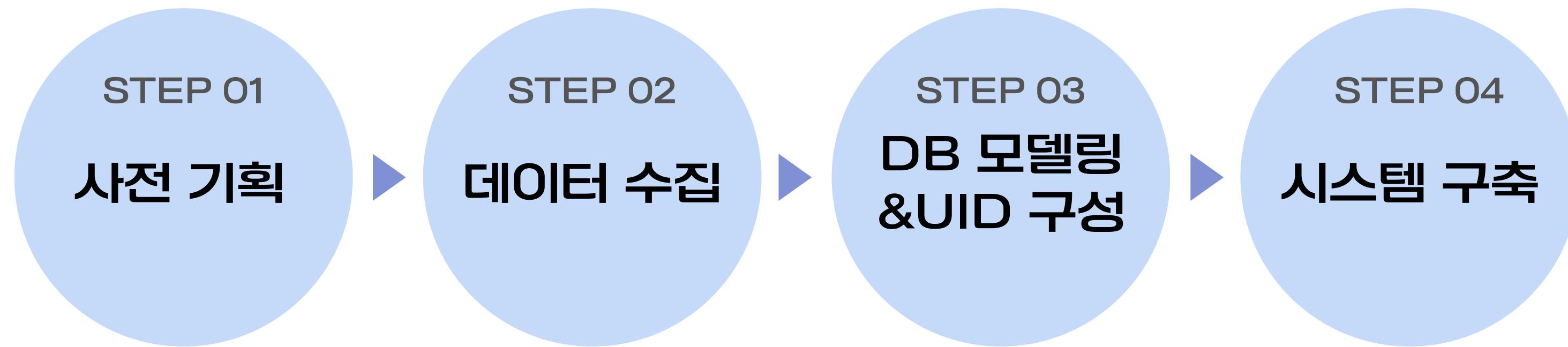


팀원 : 김수아

- 본사의 업무의 효율성을 극대화 할 수 있는 페이지를 구현
- 본사의 전사적 협업 체계 강화 구현
- 본사 데이터 기반 의사결정 구현
- 본사의 확정성과 재사용성 확보

03 프로젝트 진행과정

WORK FLOW



진행기간
2025.05.01~2025.05.02

- 요구분석
- 시나리오 설계

진행기간
2025.05.03~2025.05.04

- 신발 사진 정보 수집
- 대리점 위치 선정
- 시장 정보 확보

진행기간
2025.05.05~2025.05.06
2025.05.16~2025.05.17

- sqlite
- mysql

전체기간
2025.05.01~2025.05.20

- sqlite를 이용한 앱 만들기
- mysql을 이용한 앱 만들기

프로젝트 진행 과정

STEP 01

요구 조건

1. 사용자는 앱을 사용하여 본사의 서버(가상)에 접속하여 신발을 구매할 수 있다.
2. 사용자는 본인이 구매한 신발의 내역을 확인할 수 있다,
3. 사용자는 본인이 구매한 신발은 해당 회사의 대리점으로 방문하여 찾을 수 있다.
4. 사용자는 앱을 이용하여 신발 대리점 위치 및 본인의 위치를 확인 할 수 있다.
5. 대리점의 위치는 서울으로 한정되며 각 자치구에 1개씩의 대리점이 존재한다.
6. 대리점을 방문하여 본인이 구매한 구매번호를 대리점 직원에게 확인하여 해당 지점에서 신발을 찾아온다.
7. 본사의 경우에는 사용자가 신발 구매시 희망 대리점을 선택함으로 구매 신청후 즉시 해당 대리점으로 신발을 발송한다.
8. 대리점장은 해당 월일의 재고를 파악할 수 있어야 한다.
9. 본사의 임원인 경우에는 제품별, 일자별 등으로 현황을 파악할 수 있어야 한다.
10. 고객의 반품이 있는 경우에는 대리점을 방문하여 반품을 요청할 수 있다.
11. 반품 신발 정보는 신발 제조사에게 전달하여 원인 규명을 받는다.
12. 본사에서는 신발의 판매 현황을 파악하여 신발 재고가 30% 미만인 경우에는 신발 제조사에게 신발 구매 요청을 하여야 한다

03

프로젝트 진행 과정 STEP 01

회원

→ 회원가입, 로그인, 상품리스트, 후기, 관심, 구매내역, 회원정보수정, 상품상태, 대리점 선택, 장바구니
회원이 가능한 기능

필요 UI

회원가입, 로그인, 상품리스트, 상품상세 페이지, 마이페이지, 구매내역, 회원정보수정, 구매내역 상세페이지,
장바구니 담기, 구매 페이지, ~~후기 작성 페이지, 후기 보기, 대리점 선택페이지, 대리점 상세 정보~~
~~추천 페이지~~

= 14 페이지.

대리점 관리자

→ 로그인, 회원가입, 오늘 매장 주문, 매장 재고 확인, 본사 반품, 회원 누령 했을 때
상태 변경이 가능하다.

UI

→ 로그인, 오늘 매장 주문 확인 페이지, 매장 재고 확인 페이지, 반품 내역 보는 페이지, 반품을 신청하는 페이지.
현황 토탈 페이지, 회원 누령 상태 변경 페이지.

본사 관리자

→ 전체 재고를 확인할 수 있다, 입고를 할 수 있다, 출고도 할 수 있다, 주문서 작성, 승인, 반려, 대리점마다
재고, 총 구매 내역 볼 수 있다, 반품사유가 제조사, ~~판매인~~ 규명을 가능, 회원 관리가 가능하다.

리스트

→ 결제 확인 페이지, 결제 확인 페이지, 발주 페이지, 입고 페이지, 발주 페이지
→ 대리점 관리자 관리

요구분석&시나리오 설계

요구조건을 분석하여 필요한 사용자를 정의하고
사용자를 실체화하여 시나리오를 설계하였다

사용자 : 일반회원, 대리점관리자, 본사 관리자

시나리오 설계 :

• 일반 회원 :

- 회원가입 및 로그인이 가능하다
- 제품의 정보를 확인 할 수 있고 구매 및 장바구니 활용 할 수 있다
- 구매 내역, 공지사항 확인 가능

• 대리점 관리자 :

- 대리점 매장 주문, 매장 재고를 확인 할 수 있다.
- 반품 처리를 도와 드릴 수 있다.

• 본사 관리자 :

- 본사에 제품, 대리점, 대리점 회원, 제조사를 등록할 수 있다
- 구매내역을 확인할 수 있고, 이사직급을 가진 관리자는 기간에 따른
매출을 확인 할 수 있다.
- 발주 결재서는 전자 결재서로 승인이 이루어지면 자동 발주가 가능하다
- 본사 재고를 입출고, 발주로 재고 관리가 가능하고 내역 로그를 확인 할 수
있다

03 프로젝트 진행 과정 STEP 02 : 데이터 수집

제조사(4개)								
제조사 정보								
제품(신발) 속성 리스트								
순번	제조사(한글)	제조사(영문)	제품코드	제품명	색상코드	사이즈	가격	이미지경로
1	나이키	NIKE	1010113353	나이키 에어 맥스 액시 XMFW	204	250 ~ 290	129000	nike_01
2	나이키	NIKE	1010113264	우먼스 나이키 코트 비전 로우 넥스트 네이처	100	220 ~ 250	99000	nike_02
3	나이키	NIKE	1010111098	나이키 에어 줌 업턴 SC	004	250 ~ 290	119000	nike_03
4	나이키	NIKE	1010113265	나이키 다운쉬프터 13	010	250 ~ 290	89000	nike_04
5	나이키	NIKE	1010105452	나이키 저니 런	001	250 ~ 300	119000	nike_05
6	나이키	NIKE	1010113412	주니어 티업포 레전드 10 아카데미 TF	401	200 ~ 240	69000	nike_06
7	나이키	NIKE	1010105355	나이키 에어 원플로 11	002	250 ~ 290	129000	nike_07
8	나이키	NIKE	1010111081	맨 나이키 프로미나	004	250 ~ 290	79000	nike_08
9	나이키	NIKE	1010107408	나이키 인터랙트 런	104	250 ~ 290	99000	nike_09
10	나이키	NIKE	1010111100	나이키 코스믹 러너 그레이드스쿨	002	225 ~ 250	65000	nike_10
11	아디다스	ADIDAS	1010114419	오즈웨이브 샌들	GRETWO/SILVMT/OWHITE	225 ~ 275	79000	adidas_01
12	아디다스	ADIDAS	1010112661	VL 코트 베이스	FTWWHT/CBLACK/GREONE	225 ~ 290	69000	adidas_02
13	아디다스	ADIDAS	1010110559	클라우드풀 워크 라운저	CBLACK/LUCBLU/GOLDMT	250 ~ 290	85000	adidas_03
14	아디다스	ADIDAS	1010104613	그라디스	FTWWHT/CBLACK/MAGOLD	220 ~ 300	89000	adidas_04
15	아디다스	ADIDAS	1010113987	슈타트	EARSTR/TRUPNK/GUM3	220 ~ 290	109000	adidas_05
16	아디다스	ADIDAS	1010113984	삼바 오리지널 우먼스	FTWWHT/OWHITE/GUM5	220 ~ 250	149000	adidas_06
17	아디다스	ADIDAS	1010114668	울트라란 5	FTWWHT/BRIRED/CBLACK	250 ~ 300	79000	adidas_07
18	아디다스	ADIDAS	1010113942	클라이마풀 벤티스	CBLACK/CARBON/GREFIV	230 ~ 300	99000	adidas_08
19	아디다스	ADIDAS	1010114279	서브존	FTWWHT/CBLACK/FTWWHT	250 ~ 300	99000	adidas_09
20	아디다스	ADIDAS	1010112898	클라이마풀 벤티나	GREONE/CBLACK/GREONE	230 ~ 300	119000	adidas_10
21	뉴발란스	NEW BALANCE	1010111721	U740WN2	WN2	220 ~ 290	139000	nb_01
22	뉴발란스	NEW BALANCE	1010109541	WARISTS4	S4	220 ~ 250	109000	nb_02
23	뉴발란스	NEW BALANCE	1010114609	PV878PB1	PB1	200 ~ 220	99000	nb_03
24	뉴발란스	NEW BALANCE	1010114606	PAFCPRC5	C5	180 ~ 220	109000	nb_04
25	뉴발란스	NEW BALANCE	1010112858	MW1880L1	L1	250 ~ 290	149000	nb_05
26	뉴발란스	NEW BALANCE	1010088506	MW1880C1	C1	250 ~ 290	149000	nb_06
27	뉴발란스	NEW BALANCE	1010111731	URC42PMA	PMA	225 ~ 290	139000	nb_07
28	뉴발란스	NEW BALANCE	1010103934	MR530EWB	EWB	220 ~ 290	129000	nb_08
29	뉴발란스	NEW BALANCE	1010111716	U740GR2	GR2	220 ~ 290	139000	nb_09
30	뉴발란스	NEW BALANCE	1010111705	WS327TBA	BA	230 ~ 250	129000	nb_10
31	푸마	PUMA	1010113011	벨라 V 블러쉬	Black-Smokey Gray-Shadow Gray	220 ~ 260	79000	puma_01
32	푸마	PUMA	1010113004	푸마 클럽 II 에라	Alpine Snow-PUMA Black-PUMA Gold	225 ~ 290	79000	puma_02

Database 제품 등록에 필요한 data 를 시장 조사를 통해 수집 하였다.

조사 시장 :ABC 마켓

조사한 사람: 이상훈 (취업으로 중도하차)

수집 데이터 :

- 각 브랜드 별 신발 제품의 data 수집
- 각 신발의 이미지 데이터 수집

03 프로젝트 진행 과정 STEP 02 : 데이터 수집

ST_GN	강남점	서울특별시 강남구 테헤란로	127.0473	37.5172
ST_JN	종로점	서울특별시 종로구 세종대로	126.9794	37.5725
ST_MP	마포점	서울특별시 마포구 양화로 16	126.9268	37.5594
ST_SC	서초점	서울특별시 서초구 남부순환로	127.0215	37.4959
ST_SP	송파점	서울특별시 송파구 올림픽로	127.1058	37.5145
ST_YD	영등포점	서울특별시 영등포구 의사당로	126.9268	37.5206
ST_JG	중구점	서울특별시 중구 명동길 1	126.9809	37.5635
ST_DDM	동대문점	서울특별시 동대문구 천호대로	127.0152	37.5801
ST_GR	구로점	서울특별시 구로구 디지털로	126.856	37.4954
ST_NW	노원점	서울특별시 노원구 동일로 15	127.0636	37.6538

Database 대리점 등록에 필요한 data 를 csv 파일로 만들어서 등록하였다.

수집 데이터 :

- 각 서울 자치구별 위도경도 정보 및 주소

03 프로젝트 진행 과정

STEP 02 : 데이터 수집

예시 sql문

```
-- =====  
-- 3. 샘플 회원 (users 테이블) 삽입  
-- (userid는 TEXT PRIMARY KEY, memberType INTEGER)  
-- (아이디와 비밀번호를 영어+숫자 조합으로 수정)  
-- =====  
-- 일반 회원 (memberType=1)  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('cust001', 'pw123a', '김길동', '010-1111-0001', 1, '1990-01-15', '남');  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('cust002', 'pw456b', '박영희', '010-1111-0002', 1, '1992-03-22', '여');  
  
-- 본사 직원 (memberType=2)  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('office001', 'staff1pw', '본사일사원', '010-2222-0001', 2, '1988-04-10', '남');  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('office002', 'daeri2pw', '본사이대리', '010-2222-0002', 2, '1985-07-25', '여');  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('office003', 'bujang3pw', '본사삼부장', '010-2222-0003', 2, '1975-11-03', '남');  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('office004', 'isa4pw', '본사사이사', '010-2222-0004', 2, '1968-09-18', '여');  
  
-- 대리점 회원 (memberType >= 3, 순차적으로 3부터 부여)  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('storegn01', 'gnpw123', '강남점지점장', '010-3333-0001', 3, '1980-01-01', '남'); -- memberType 3  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('storejn01', 'jnpw456', '종로점지점장', '010-3333-0002', 4, '1981-02-02', '여'); -- memberType 4  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('storemp01', 'mppw789', '마포점지점장', '010-3333-0003', 5, '1982-03-03', '남'); -- memberType 5  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('storesc01', 'scpwabc', '서초점지점장', '010-3333-0004', 6, '1983-04-04', '여'); -- memberType 6  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('storesp01', 'sppwdef', '송파점지점장', '010-3333-0005', 7, '1984-05-05', '남'); -- memberType 7  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('storeyd01', 'ydpwghi', '영등포점지점장', '010-3333-0006', 8, '1985-06-06', '여'); -- memberType 8  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('storejg01', 'jgpwjk1', '중구점지점장', '010-3333-0007', 9, '1986-07-07', '남'); -- memberType 9  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('storeddm01', 'ddmpw14', '동대문점지점장', '010-3333-0008', 10, '1987-08-08', '여'); -- memberType 10  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('storegr01', 'grpw257', '구로점지점장', '010-3333-0009', 11, '1988-09-09', '남'); -- memberType 11  
INSERT INTO users (userid, password, name, phone, memberType, birthDate, gender) VALUES  
    ('storenw01', 'nwpw368', '노원점지점장', '010-3333-0010', 12, '1989-10-10', '여'); -- memberType 12
```

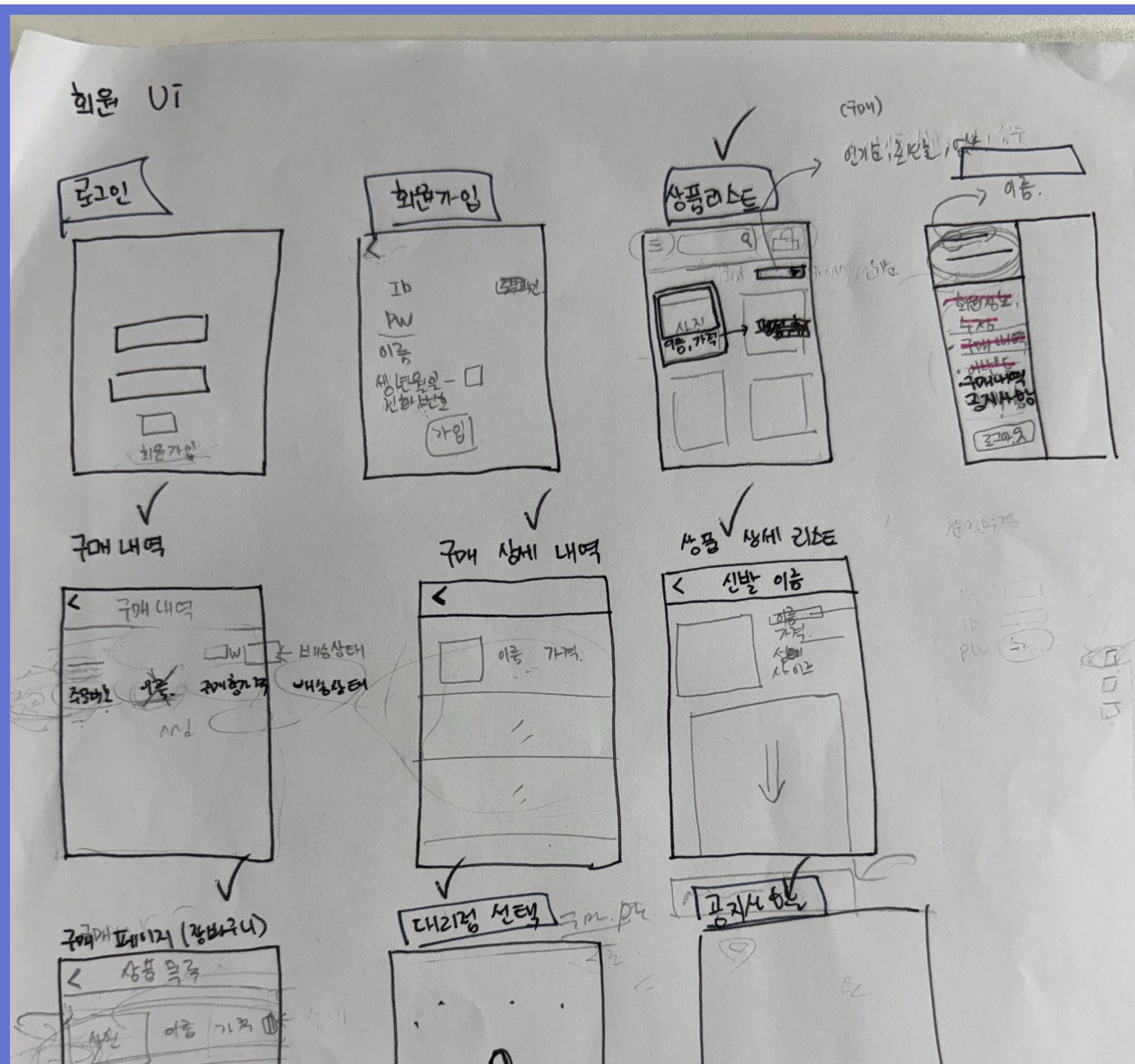
Database 를 활용하여 앱에서 확인해볼 수있게 임의의 데이터 수집

수집 데이터 :

- 각 테이블에 들어갈 sql문을 작성

03

프로젝트 진행 과정 STEP 03 :UID 설계



회원 UID

사용자가 앱을 이용 함으로써 이용 할 수 있는 서비스와 관련된 화면들을 구성하였다.

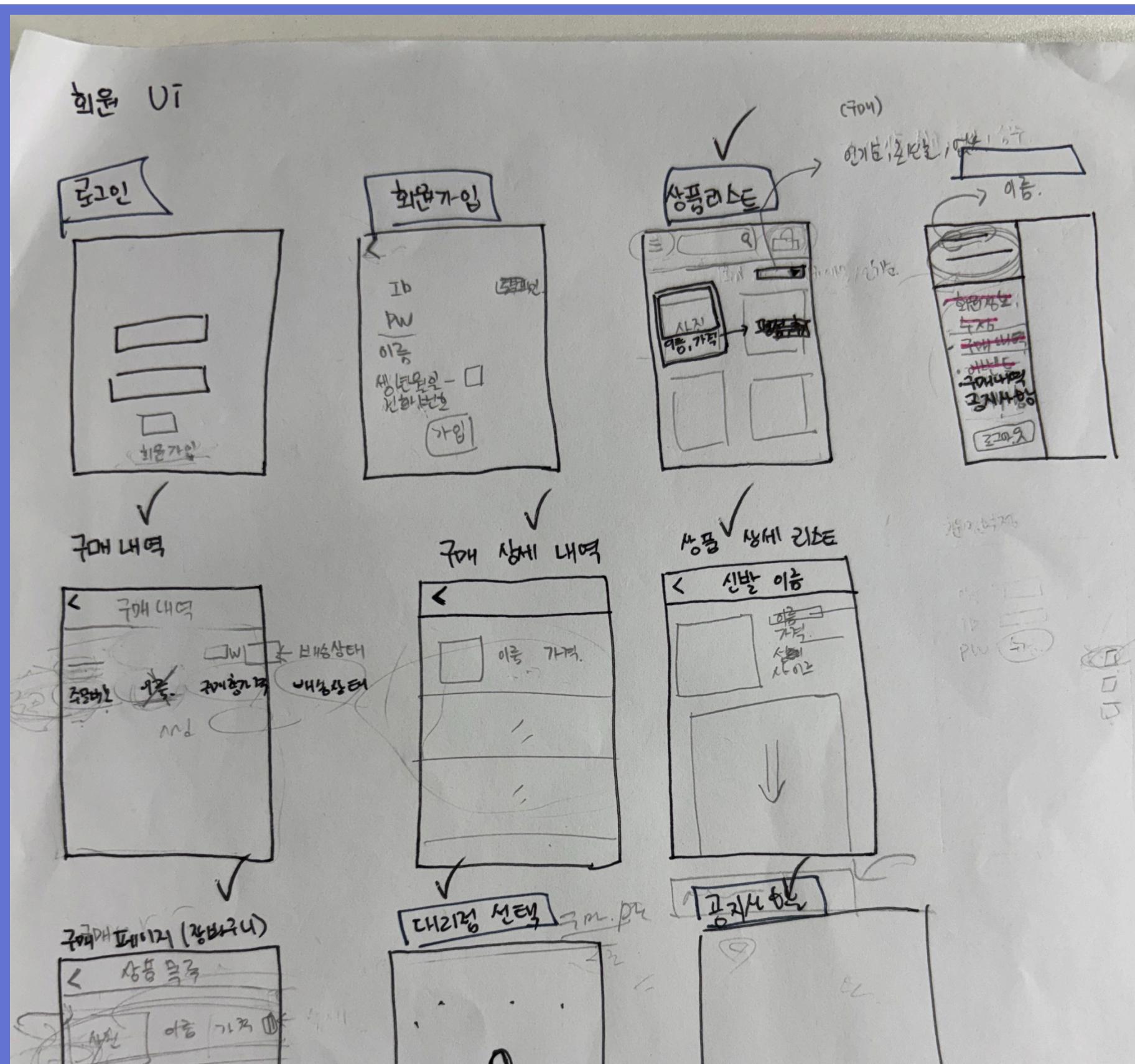
사용자 : **일반회원**

화면 설계 :

- **로그인**: 회원가입으로 이동 및 로그인이 가능하다.
- **회원가입**: 중복된 ID를 체크할 수 있고 가입이 가능하다.
- **상품 리스트**: 본사에서 등록한 제품 게시글을 볼 수 있고 검색을 활용해 원하는 제품이 있는지 검색 할 수 있다.
- **상품 상세**: 상품 리스트에서 선택한 상품을 소개하는 이미지를 볼 수 있고 사이즈, 수량 및 픽업 대리점을 선택하여 장바구니 담기 혹은 주문을 할 수 있다.
- **장바구니**: 사용자가 장바구니에 담아둔 상품 리스트를 볼 수 있으며 해당 상품을 주문 할 수 있고 내역을 삭제 할 수 있다.

03

프로젝트 진행 과정 STEP 03 :UID 설계



회원 UID

사용자가 앱을 이용 함으로써 이용 할 수 있는 서비스와 관련된 화면들을 구성하였다.

사용자 : **일반회원**

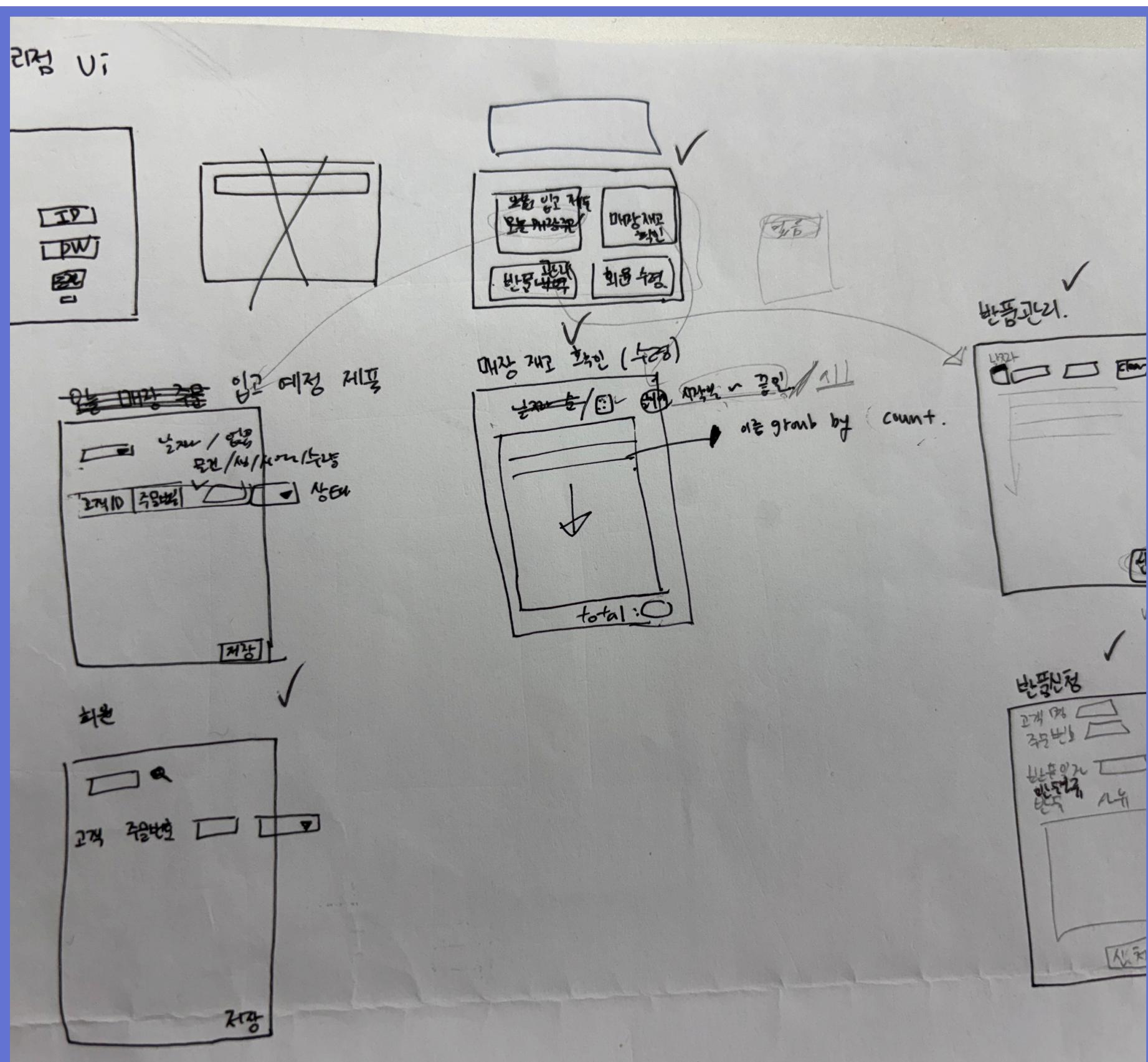
화면 설계 :

(Drawer)

- 구매 내역:** 사용자가 주문한 제품 리스트를 볼 수 있으며 진행상황에 따라 상태가 갱신된다.
- 구매 내역 상세:** 해당 주문의 상세 정보를 볼 수 있다.
- 공지사항:** 본사에서 작성한 공지사항들의 리스트를 볼 수 있다.
- 공지사항 상세:** 해당 공지사항의 상세 내용을 볼 수 있다.

03

프로젝트 진행 과정 STEP 03 :UID 설계



대리점 UID

대리점 관리자가 사용할수있는 앱의 화면을 구현

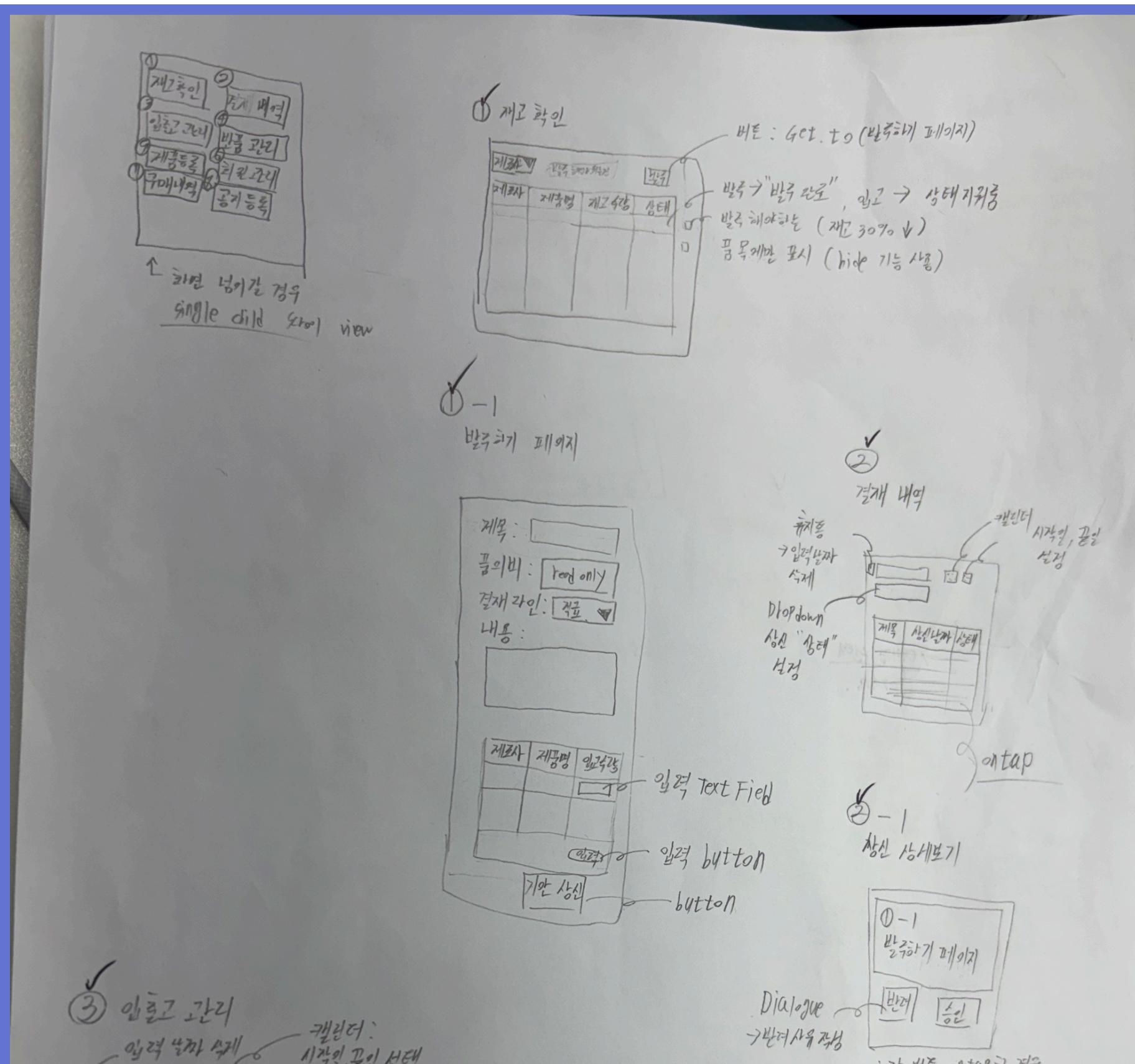
사용자 :**대리점관리자**

화면 설계 : 본사에서 생성한 아이디를 통해 로그인을 하면 그 아이디에 맞는 대리점에 연결되고 그 대리점의 본사로 부터의 입고 예정 제품, 매장 재고, 반품 내역, 회원 수령 페이지를 구성하였다.

- **입고 예정 제품:** 날짜를 선택하여 그날의 입고예정 제품과 수량을 확인할수 있다.
- **매장 재고:** 날짜의 범위를 선택하여 범위내에 얼마만큼의 재고가 있는지 총개수를 확인할수 있다.
- **반품 내역:** 해당 매장의 날짜별로 반품된 날짜 처리상태 반품 사유등을 확인할 수 있고 반품신청 페이지로 이동하여 주문번호를 조회하여 고객의 이름을 찾고 반품을 등록 할 수 있다.
- **픽업 대기목록:** 선택된 매장에서 픽업될 물건들을 검색하여 확인하고 받아간 물품을 업데이트 할 수 있다.

03

프로젝트 진행 과정 STEP 03 :UID 설계



본사 UID

본사 관리자가 사용할수있는 앱의 화면을 구현

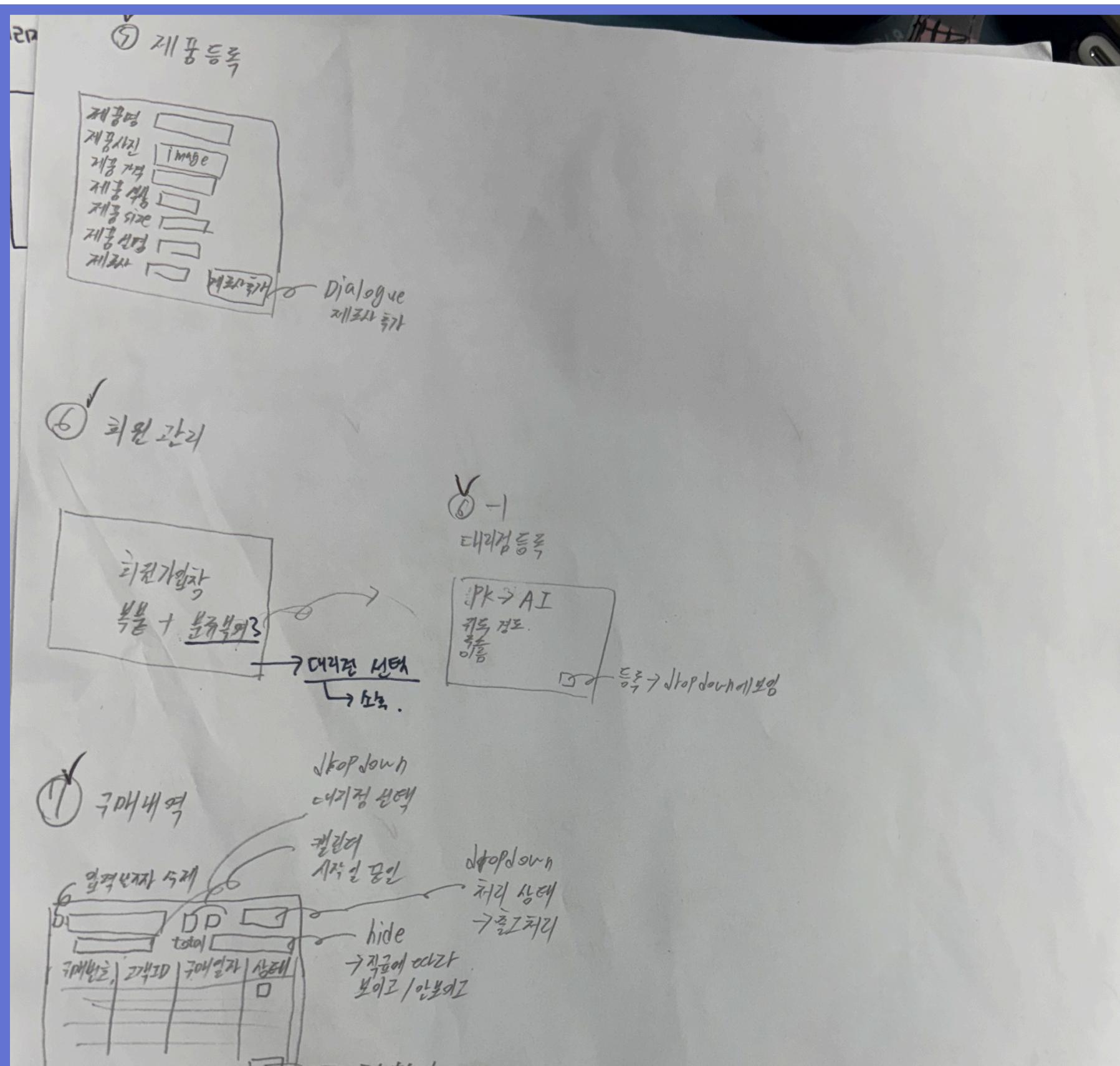
사용자 : **본사 관리자**

화면 설계 : 본사 관리자의 업무효율성을 올릴 수 있는 화면을 구성해 보려고 노력하였다.

- **HOME**: 관리할 수 있는 페이지를 연결해준다
- **재고 관리**: 재고의 이름이나 제품 코드로 검색이 가능하고 입고와 출고가 가능하다
- **발주서 작성**: 관리자 직급에 따른 결재 라인을 선택하여 결재서를 작성한다. 결재서는 전자 결재 시스템으로 결재 가능한 직급을 가진 회원이 승인, 혹은 반려가 가능하다. 승인이 되면 물건은 자동 발주가 이루어 진다
- **반품관리** : 대리점에서 반품이 등록된 제품이 어떤 사유로 반품으로 처리되었는지 확인하여 다시 물건을 보내거나 제조사를 통해 일을 해결 했을 경우 그에 따른 업무 처리를 기록 할 수 있다.

03

프로젝트 진행 과정 STEP 03 :UID 설계



본사 UID

본사 관리자가 사용할 수 있는 앱의 화면을 구현

사용자 : **본사 관리자**

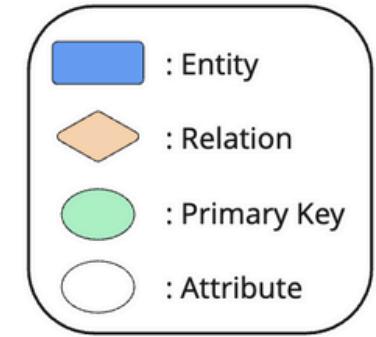
화면 설계 : 본사 관리자의 업무 효율성을 올릴 수 있는 화면을 구성해 보려고 노력하였다.

- **등록:** 제품, 제조사, 대리점, 대리점회원, 공지사항을 등록할 수 있다.
- **구매내역:** 고객의 주문내역을 확인 할 수 있고 직급이 이사인 본사 회원은 주문내역에서 날짜를 선택 하여 총 매출을 볼 수 있다.

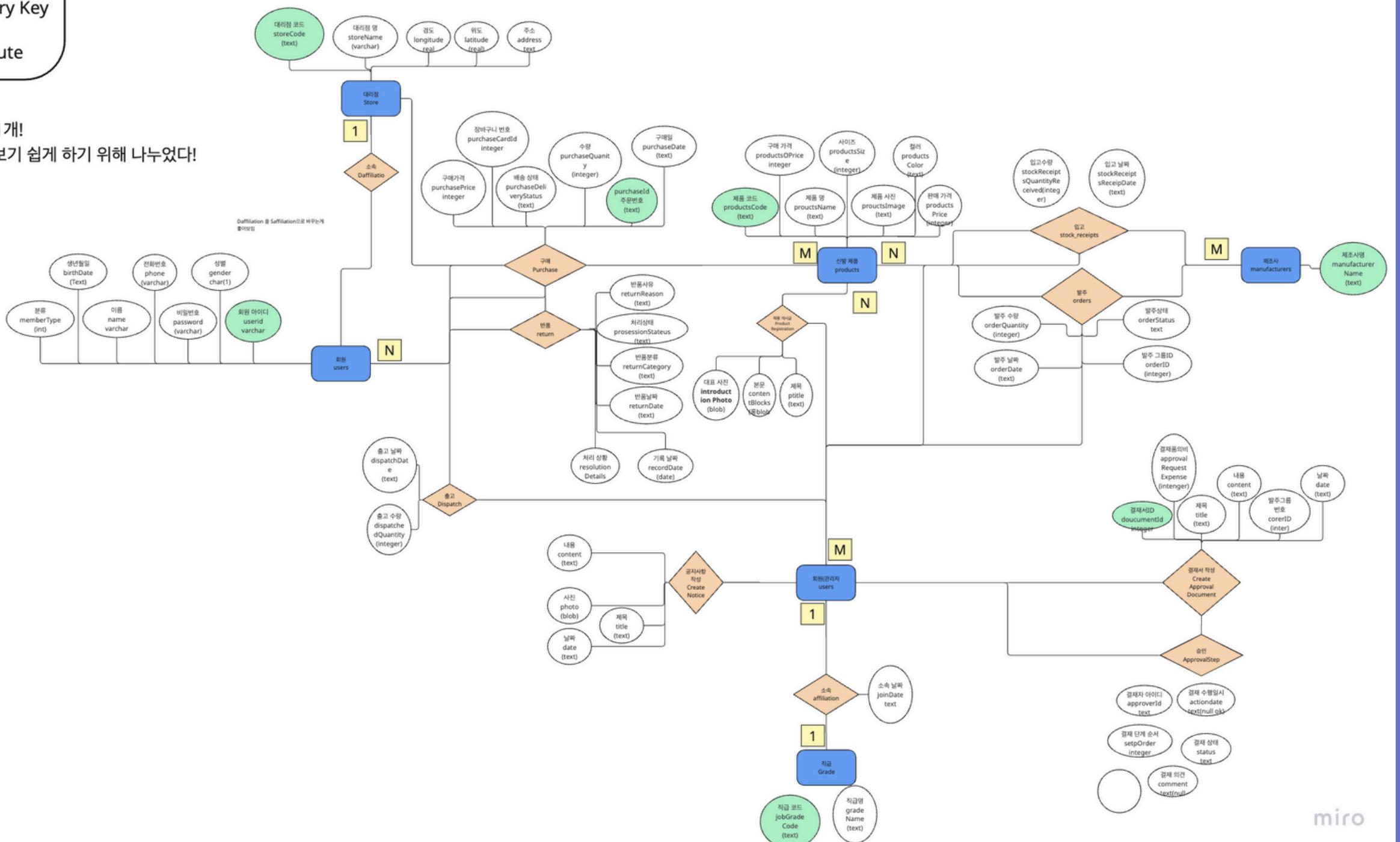
03 프로젝트 진행 과정

STEP 03 : DB 설계

회원, 대리점, 본사 UID 구상을 기반
으로 개념적 ERD 를 구성



회원 : 테이블은 1개!
회원 (관리자) : ERD 에서 보기 쉽게 하기 위해 나누었다!



이용 서비스 : MIRO

개념적 ERD 설계 :

• Entity:

- 회원
- 대리점
- 신발 제품
- 직급
- 제조사

• Relation:

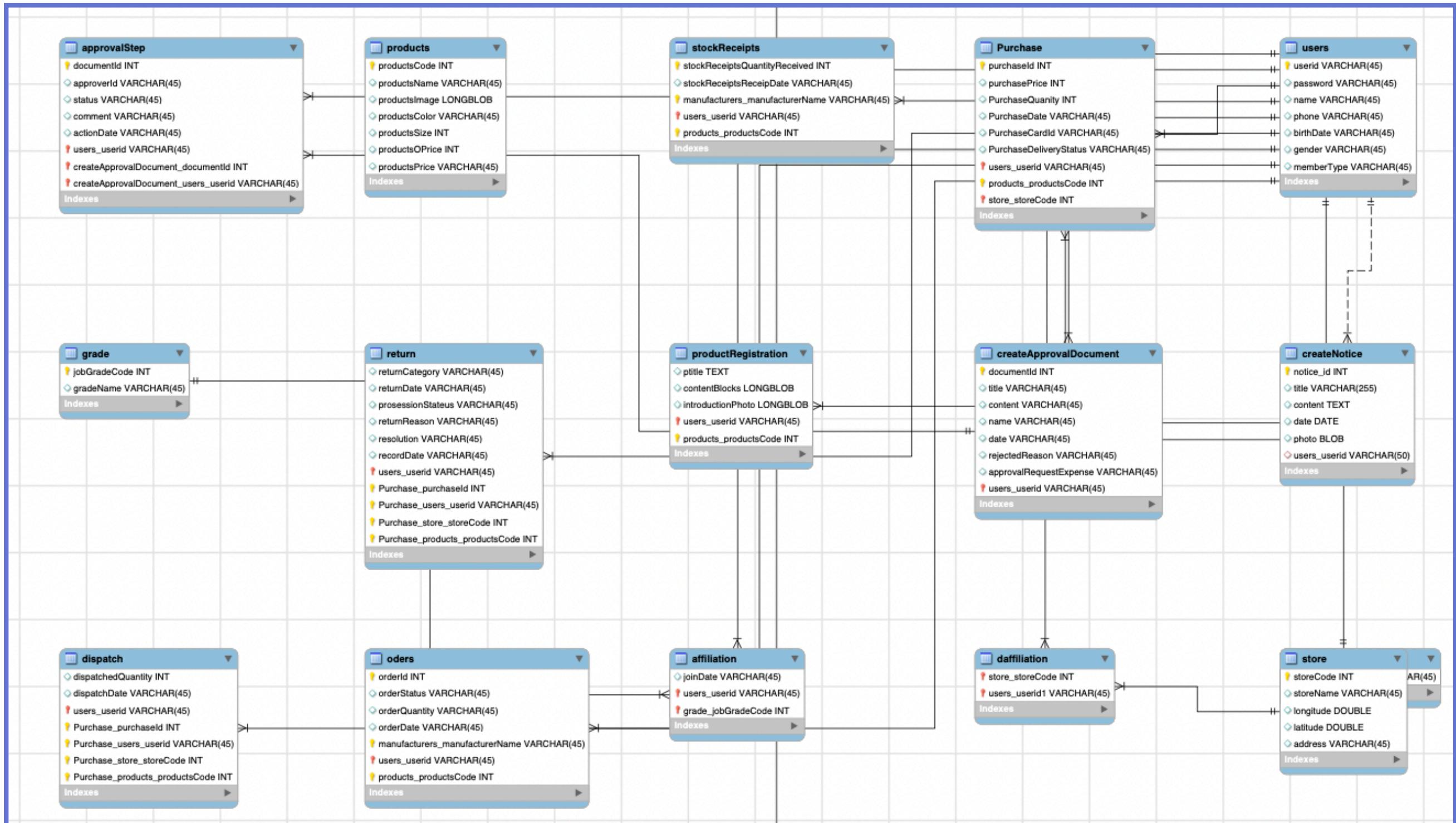
- 소속
- 구매
- 반품
- 출고
- 발주
- 승인
- 입고
- 제품 게시글 작성
- 공지사항 작성
- 결재서 작성

03 프로젝트 진행 과정

STEP 03 : DB 설계

개념적 ERD 구상을 기반으로 물리적 ERD 구현

이용 서비스 : MySQL workbench



개념적 ERD 설계 :

- **Entity:**

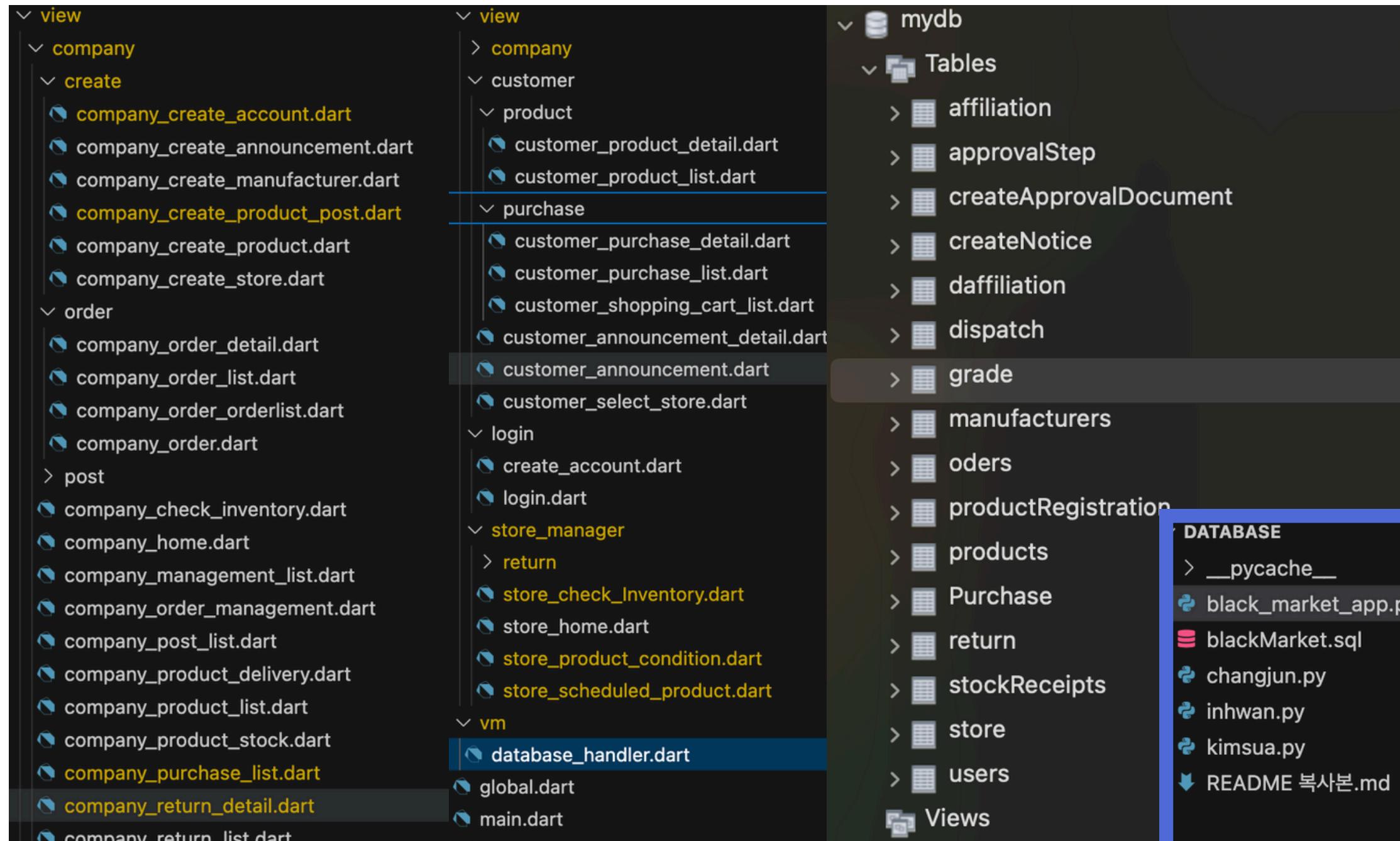
- 회원
- 대리점
- 신발 제품
- 직급
- 제조사

- **Relation:**

- 소속
- 구매
- 반품
- 출고
- 발주
- 승인
- 입고
- 제품 게시글 작성
- 공지사항 작성
- 결재서 작성

03 프로젝트 진행 과정

STEP 04 시스템구축



개발 시스템 환경 구축



```
black_market_app.py > ...
1  from fastapi import FastAPI
2  from kimsua import router as kimsua_router
3  from inhwain import router as inhwain_router
4  from changjun import router as changjun_router
5
6  ip = "127.0.0.1"
7
8  app = FastAPI()
9  app.include_router(kimsua_router,prefix="/kimsua")
10 app.include_router(inhwain_router,prefix="/inhwan")
11 app.include_router(changjun_router,prefix="/changjun")
12
13 if __name__ == "__main__":
14     import uvicorn
15     uvicorn.run(app,host=ip,port=8000)
```

04

프로젝트 결과

01

UI 구현



Dart



02

SQLITE 구현



SQLite



03

MySQL 구현



MySQL™



PYTHON

04

프로젝트 결과



회원

04 프로젝트 결과

```
// 사용자 정보 입력
Future<int> insertUserInfo(Users account) async {
    int result = 0;
    final Database db = await initializeDB();
    result = await db.rawInsert(
        "INSERT INTO users (userid, password, name, birthDate, gender, phone, memberType) VALUES (?, ?, ?, ?, ?, ?, ?)",
        [
            account.userid,
            account.password,
            account.name,
            account.birthDate,
            account.gender,
            account.phone,
            account.memberType,
        ],
    );
    return result;
}
```

UID 구상을 통한 ERD 설계를 확인하기 위한 Local database test

이용 패키지 : **SQLite**

구현 내용 :

- **Database_Handler:**
 - database에 있는 table로부터 data를 CRUD하는 함수들을 작성함

04 프로젝트 결과

POST /changjun/insertUserAccount Insertuseraccount

Parameters

Try it out

```

# ----- Functions -----
# ----- create_account.dart ----- #
# 1. 사용자가 앱을 사용하기 위해 회원가입을 할 때 입력한 정보를 Database 에 insert 하는 함수
@router.post("/insertUserAccount")
async def insertUserAccount(
    userid : str=Form(...), password : str=Form(...), name : str=Form(...), phone : str=Form(...),
    birthDate : str=Form(...), gender : str=Form(...), memberType : str=Form(...)):
    try:
        conn = connect()
        curs = conn.cursor()
        sql = 'INSERT INTO users (userid, password, name, phone, birthDate, gender, memberType) VALUES (%s,%s,%s,%s,%s,%s,%s)'
        curs.execute(sql, (userid, password, name, phone, birthDate, gender, memberType))
        conn.commit()
        conn.close()
        return {'result' : 'OK'}
    except Exception as e:
        print("Error : ", e)
        return {"result" : "Error" }

# ----- #
# 2. 사용자가 회원가입을 할 때 아이디의 중복을 확인하기 위해 Database 에 입력한 값의 유무를 확인하는 함수
@router.get('/selectUserIdDoubleCheck')
async def selectUserIdDoubleCheck(userid : str):
    conn = connect()
    curs = conn.cursor()
    curs.execute("SELECT count(*) From users WHERE userid =%s", (userid, ))
    rows = curs.fetchall()
    conn.close()
    result = [{ 'count' : row[0]}for row in rows]
    return {'results' : result}

# ----- #

```

**Local database 로 확인한 SQL문
을 Python 에서 작성하여 server 와
database 의 연결을 test**

이용 패키지 : **Pyhon, mySQL,
FastAPI**

구현 내용 :

- **Pyhon:**

- Database 의 data 를 App 으로 혹은
App 에서 입력한 data 를 Database 로
불러오거나 입력하는 함수를 구현

- **FastAPI:**

- FastApi 를 활용하여 web 을 이용해
함수의 정상작동 여부를 확인

04 프로젝트 결과

```
// ----- Functions ----- //
// 1. 시작은 전체 리스트가 나오고 이후 검색어를 입력 한 뒤 검색 버튼을 누르게 되면 검색어가 포함된 data 들을 불러오는 함수
// 아무것도 입력하지 않고 검색 버튼을 눌러도 전체 data 가 나타난다.
getJSONData()async{
  searchKeyword = searchCon.text.trim().isEmpty
  ? ''
  : searchCon.text;
  var response = await http.get(Uri.parse("http://$globalip:8000/changjun/select/allProductsRegistration/$searchKeyword"));
  data.clear();
  data.addAll(json.decode(utf8.decode(response.bodyBytes))['results']);
  setState(() {});
}
// -----
```

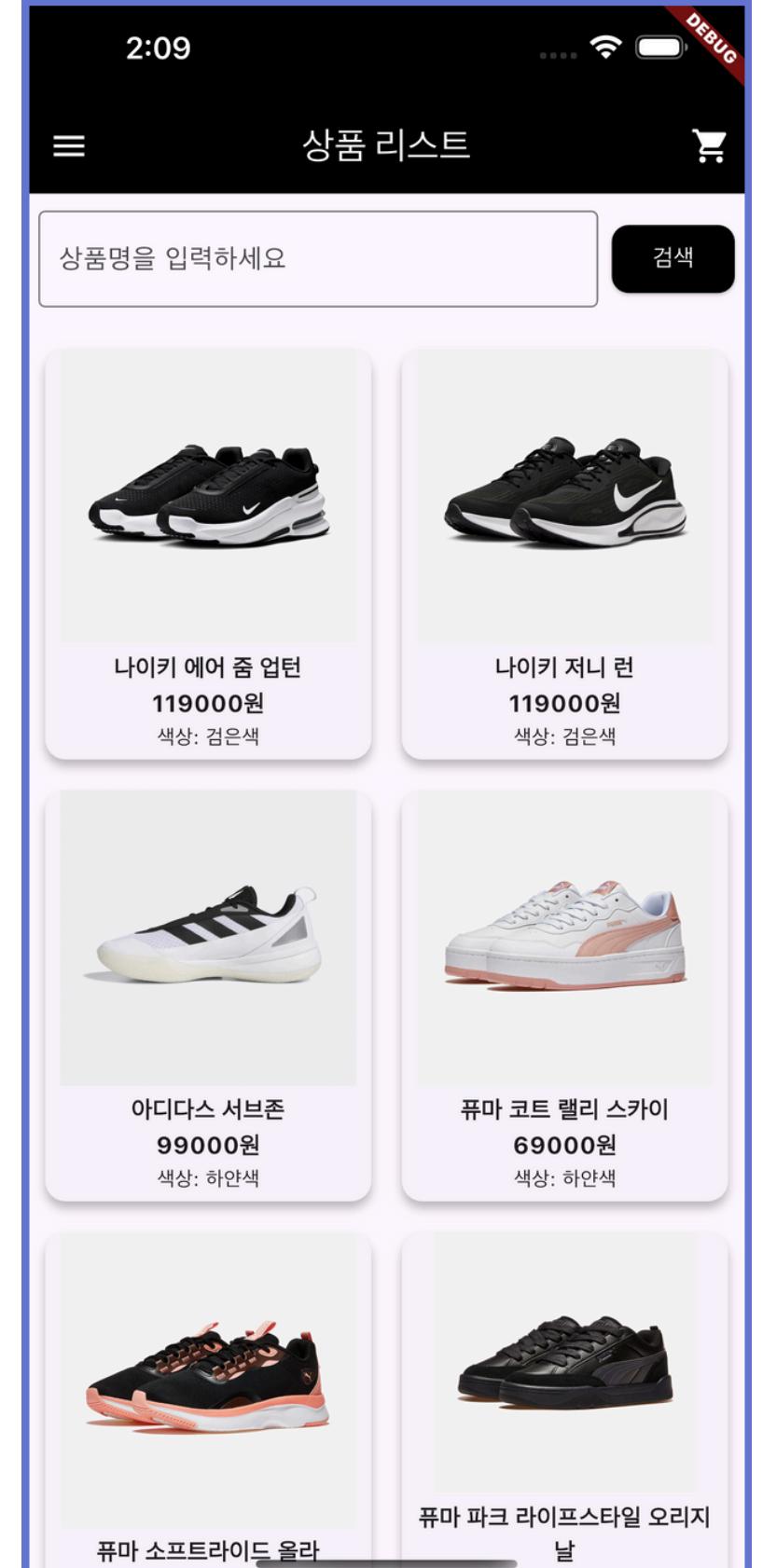
Python 으로 작성하여 확인한 database CRUD 함수를 App 에서 적용하여 Database 와 App 간의 연결 확인

이용 패키지 : **Http**

구현 내용 :

- **Data CRUD:**

- Database 에서 data 를 불러와 원하는 Form 으로 App 에서 출력 및 삭제 하는 기능 구현
- App 에서 입력한 data 를 database 로 입력, 수정 하는 기능 구현



04 프로젝트 결과

자체 평가 의견 :

1. 부족한 UX:

- 사용자가 물건을 구매 할 때의 편의성을 개선 할 필요가 있음.
- ex) 선택한 대리점의 상세 주소를 볼 수 있는 UI, 장바구니의 제품을 한번에 결제하는 UI 등.

2. 사용자 정보의 수정 기능:

- 사용자가 회원 가입 시에 입력한 정보 중 전화번호, 비밀번호 와 같이 갱신이 필요한 부분에 대한 수정 기능의 구현 필요.

3. 팀원 간의 업무 분배의 미흡:

- 프로젝트를 진행하면서 업무 역량을 고려하여 업무량을 분배하는 것이 효율적일 수 있지만 이러한 업무량의 불균등이 팀원에게 부담을 주게되면 이러한 부분을 해소하기 위해 다른 팀원 역시 업무 분담에 적극적이어야 하며 팀장으로서 이러한 부분을 고려하여 업무를 고르게 분배하는 부분에서 미흡함.

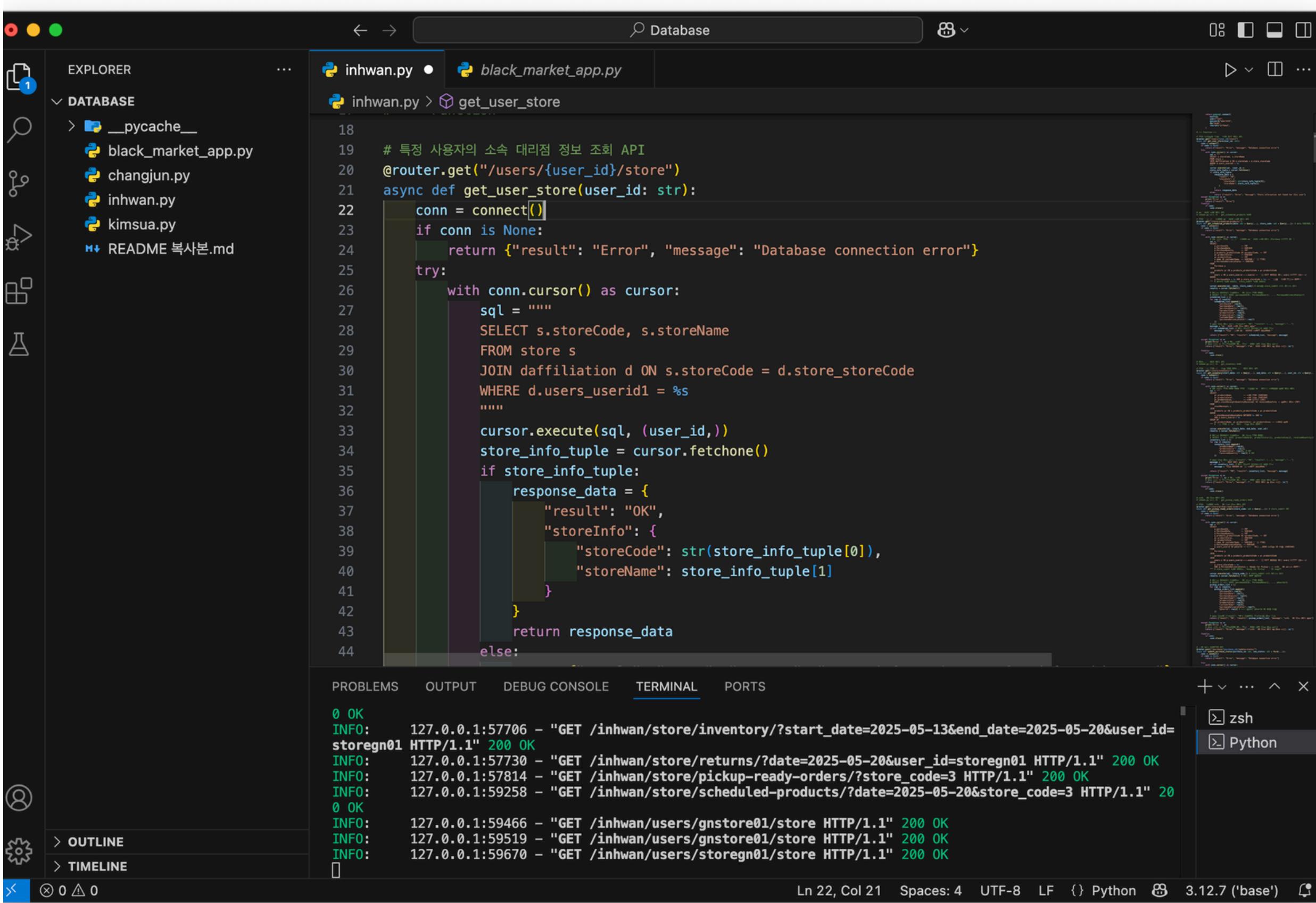
04

프로젝트 결과



대리점 관리자

프로젝트 결과 화면 구성 예: 대리점 정보 찾기



```

Database
inhwan.py black_market_app.py
inhwan.py > get_user_store

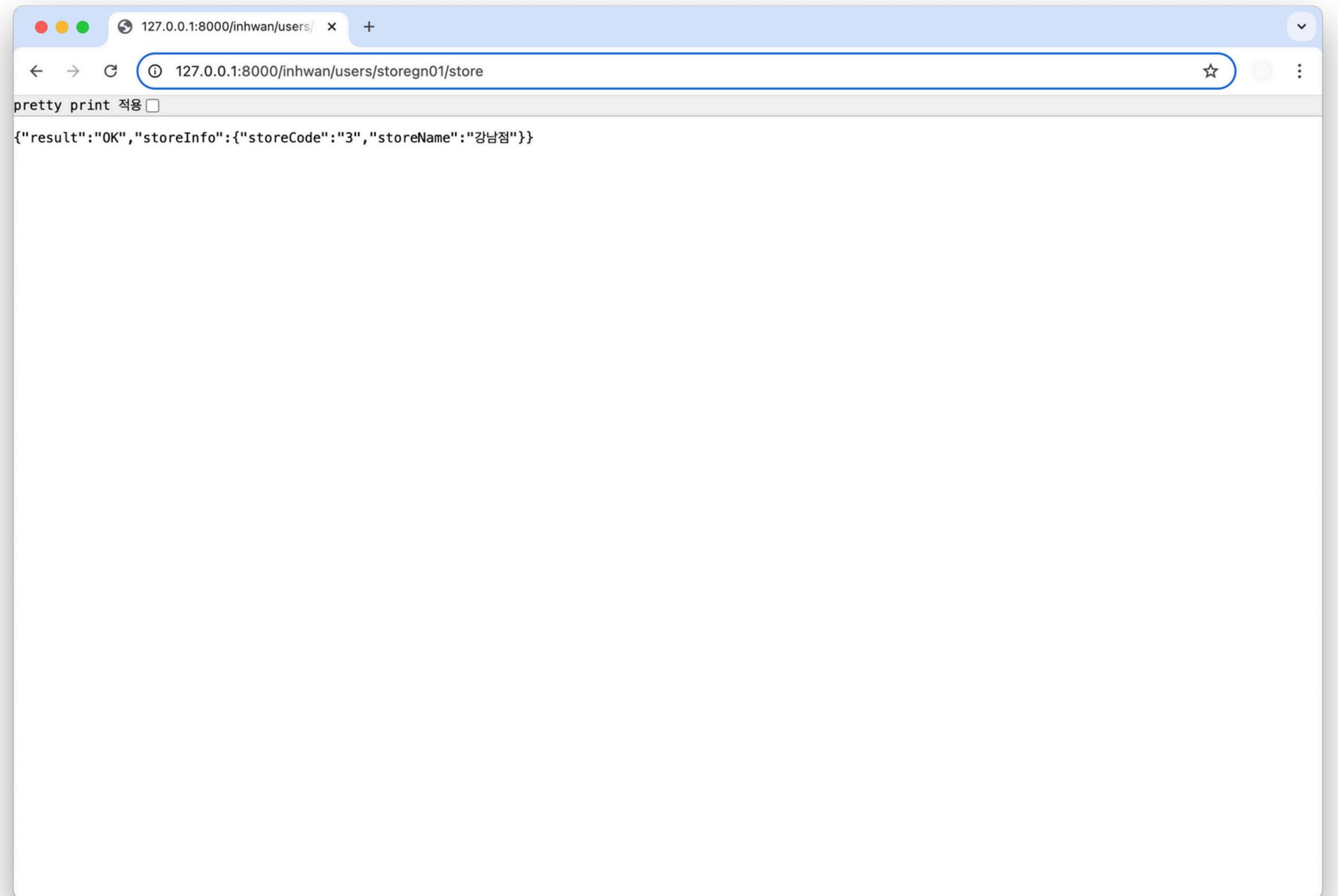
18
19 # 특정 사용자의 소속 대리점 정보 조회 API
20 @router.get("/users/{user_id}/store")
21 async def get_user_store(user_id: str):
22     conn = connect()
23     if conn is None:
24         return {"result": "Error", "message": "Database connection error"}
25     try:
26         with conn.cursor() as cursor:
27             sql = """
28                 SELECT s.storeCode, s.storeName
29                 FROM store s
30                 JOIN daffiliation d ON s.storeCode = d.store_storeCode
31                 WHERE d.users_userid1 = %s
32             """
33             cursor.execute(sql, (user_id,))
34             store_info_tuple = cursor.fetchone()
35             if store_info_tuple:
36                 response_data = {
37                     "result": "OK",
38                     "storeInfo": {
39                         "storeCode": str(store_info_tuple[0]),
40                         "storeName": store_info_tuple[1]
41                     }
42                 }
43             return response_data
44         else:
45
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+ ... ^ x
zsh
Python
0 OK
INFO: 127.0.0.1:57706 - "GET /inhwan/store/inventory/?start_date=2025-05-13&end_date=2025-05-20&user_id=storegn01 HTTP/1.1" 200 OK
INFO: 127.0.0.1:57730 - "GET /inhwan/store/returns/?date=2025-05-20&user_id=storegn01 HTTP/1.1" 200 OK
INFO: 127.0.0.1:57814 - "GET /inhwan/store/pickup-ready-orders/?store_code=3 HTTP/1.1" 200 OK
INFO: 127.0.0.1:59258 - "GET /inhwan/store/scheduled-products/?date=2025-05-20&store_code=3 HTTP/1.1" 200 OK
0 OK
INFO: 127.0.0.1:59466 - "GET /inhwan/users/gnstore01/store HTTP/1.1" 200 OK
INFO: 127.0.0.1:59519 - "GET /inhwan/users/gnstore01/store HTTP/1.1" 200 OK
INFO: 127.0.0.1:59670 - "GET /inhwan/users/storegn01/store HTTP/1.1" 200 OK
Ln 22, Col 21 Spaces: 4 UTF-8 LF {} Python 3.12.7 ('base')

```

Python 을 이용해 내 라우터에 필요 한 기능 구현

구현 내용 : 특정 사용자의 소속 대리점
정보를 가져온다

프로젝트 결과



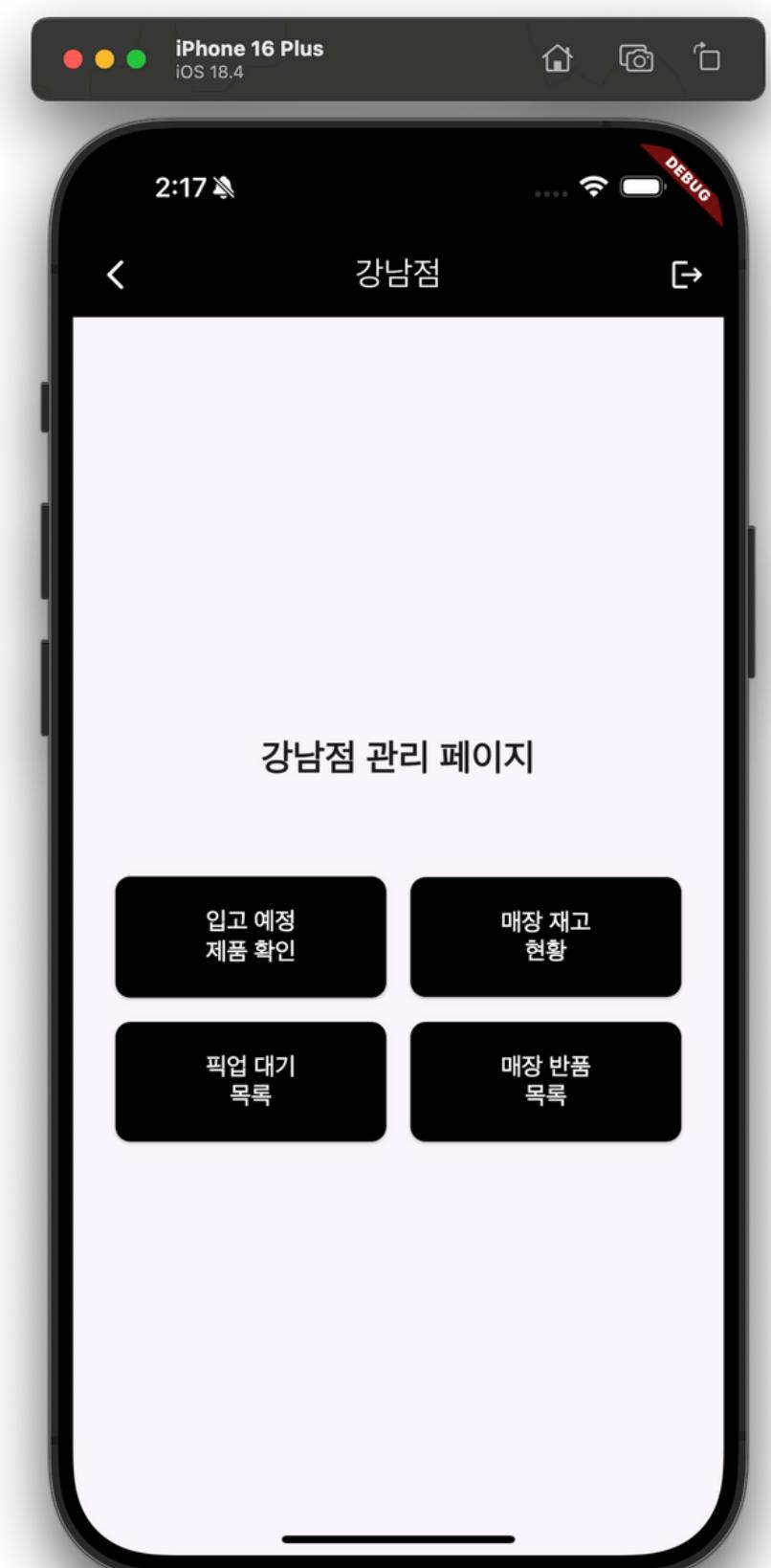
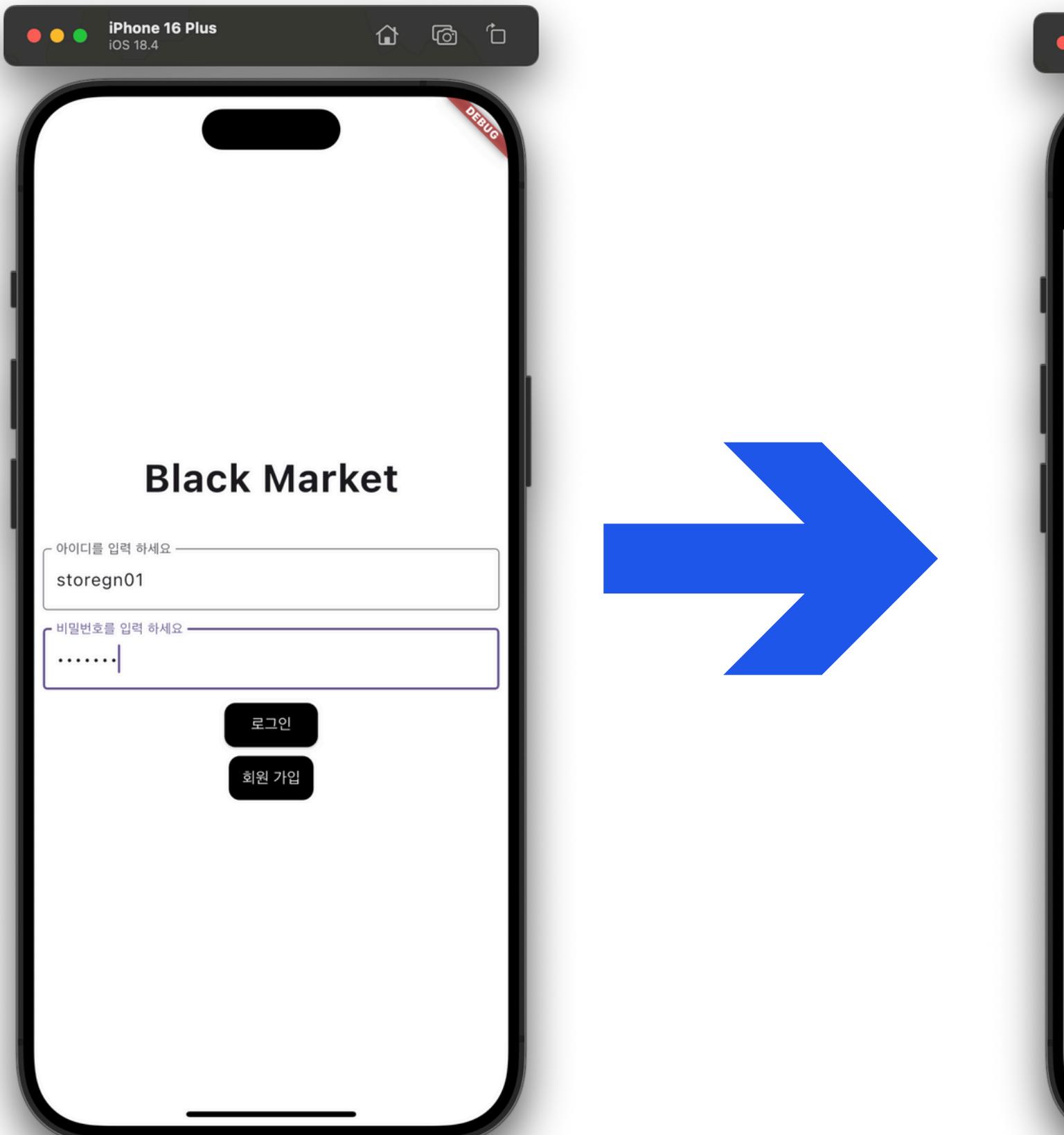
A screenshot of a web browser window. The address bar shows two tabs: the first tab is at `127.0.0.1:8000/inhwani/users/` and the second tab is at `127.0.0.1:8000/inhwani/users/storegn01/store`. Below the tabs, there is a checkbox labeled "pretty print 적용". The main content area displays the following JSON response:

```
{"result": "OK", "storeInfo": {"storeCode": "3", "storeName": "강남점"}}
```

실제 인터넷 창에 어떤 정보가 들어오는지 확인

구현 내용 : 내가 만든 기능이 정상적으로 동작하는지 확인

프로젝트 결과



내가 만든 앱으로 구현 결과 확인하기

구현 내용 : 로그인 하였을때
memberType 확인하여 강남점 대리점
화면으로 넘어감

자체 평가 의견

부족한 기능:

- 고객이 학업할 때 싸인을 받거나 무언가 확인받을 수 있는 페이지 구현을 하지 못한 것이 아쉬움
- 보여주는 UI에 정보를 날짜를 세세하게 적어야 할 껌 같다.

04

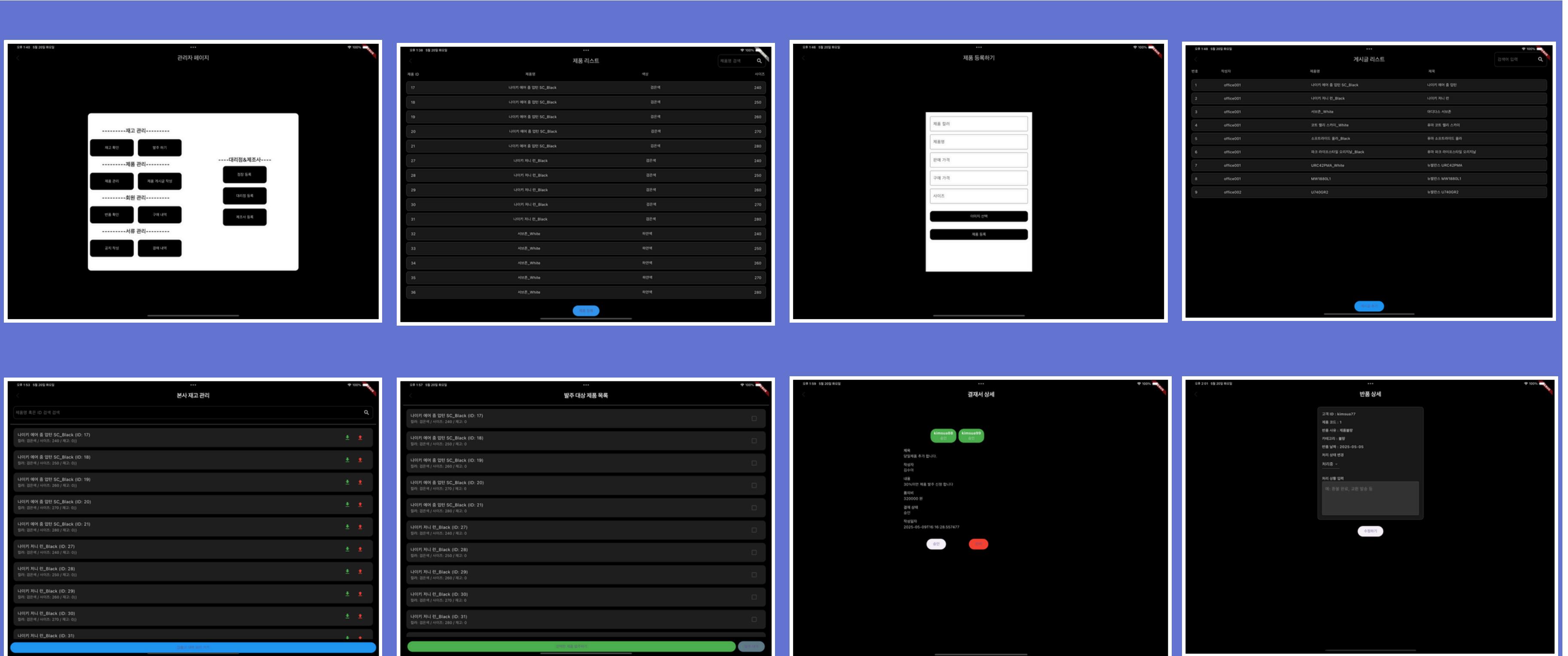
프로젝트 결과



본사 관리자

04

프로젝트 결과



프로젝트 결과



Dart

create,insert



필요한 테이블을 만들 수 있다.
정보를 저장할 수 있다

제품, 제조사, 입고, 출고, 발주, 대리점회원, 공지사항 등록 등의 페이지에 쓰였다

```
//입고 넣기
Future<int> insertStockReceipt(StockReceipts receipt) async {
  final db = await initializeDB();
  return await db.insert('stockReceipts', {
    'saUserId': receipt.saUserId,
    'stockReceiptsQuantityReceived': receipt.stockReceiptsQuantityReceived,
    'stockReceiptsReceipDate':
      receipt.stockReceiptsReceipDate.toIso8601String(),
    'sproductCode': receipt.sproductCode,
    'smanufacturerName': receipt.smanufacturerName,
  }, conflictAlgorithm: ConflictAlgorithm.replace);
}
```



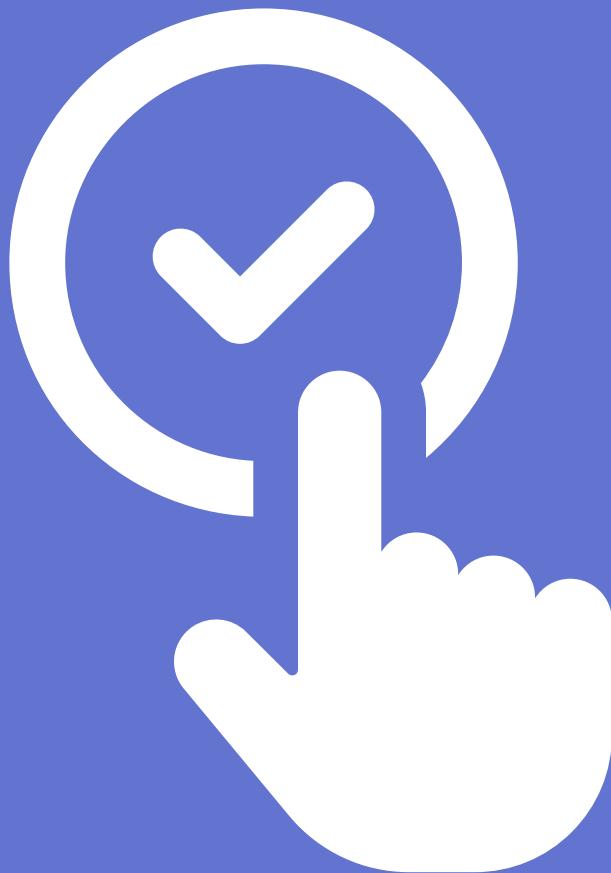
Dart

select



필요한 정보를 가져와 읽을 수 있다.

입출고 내역, 발주 내역, 발주 딕테일 재고관리페이지, 게시글 리스트 등 정보를 들고와야 하는 페이지에 쓰임



```
//입고 리스트
Future<List<Map<String, dynamic>>> getAllStockReceipts() async {
    final db = await initializeDB();
    return await db.rawQuery('''
        SELECT * FROM stockReceipts
    ''');
}
```

```
//출고 리스트
Future<List<Map<String, dynamic>>> getAllDispatches() async {
    final db = await initializeDB();
    return await db.rawQuery('''
        SELECT d.*, s.storeName
        FROM dispatch d
        LEFT JOIN store s ON d.dstoreCode = s.storeCode
    ''');
}
```



Dart

update



원래 있던 정보를 업데이트할 수 있다.

배송상태, 반품상태, 승인상태 등 변경되는 정보가 있을 때 쓰임

```
//반품 상태 변경 쿼리문
Future<void> updateReturnStatus({
    required int returnCode,
    required String newStatus,
}) async {
    final db = await initializeDB();
    await db.rawUpdate(
        '''
        UPDATE return
        SET processionStatus = ?
        WHERE returnCode = ?
        ''',
        [newStatus, returnCode],
    );
}
```



Dart

delete



필요 없는 정보는 삭제 할 수 있다.



필요 없는 데이터를 삭제하는데 쓰이지만 정보통신법상 유저에게는 지워진걸로 보이고 정보를 가지고 있어야 되기 때문에 관리자 페이지에는 딜렉트가 쓰지 않았다.

```
// ----- //
Future<void> deletePurchaseItem(int purchaseId) async {
    final db = await initializeDB();
    await db.delete(
        'purchase',
        where: 'purchaseId = ?',
        whereArgs: [purchaseId],
    );
}
```

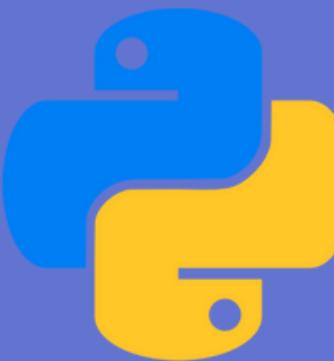
프로젝트 결과



PYTHON



프로젝트 결과



PYTHON

create,insert



필요한 테이블을 만들 수 있다.
정보를 저장할 수 있다

제품, 제조사, 입고, 출고, 발주, 대리점회원, 공지사항 등록 등의 페이지에 쓰였다

```
@router.post('/insert/products')
async def insert_products(productsName : str =Form(...),productsColor : str = Form(...),
productsSize : int = Form(...),productsOPrice : int = Form(...),productsPrice : str = Form(...),productsImage:UploadFile = File(...)):
    try:
        image_data = await productsImage.read()
        conn = connect()
        curs = conn.cursor()
        sql = """
        INSERT INTO products(productsName,productsColor,productsSize,
        productsOPrice,productsPrice,productsImage) VALUES (%s,%s,%s,%s,%s,%s)
        """
        curs.execute(sql,(productsName,productsColor,productsSize,productsOPrice,productsPrice,image_data))
        conn.commit()
        conn.close()
        return{"result" : "OK"}
    except Exception as e:
        print("Error : ",e)
        return{"resule":"Error"}
```

프로젝트 결과



PYTHON



select



필요한 정보를 가져와 읽을 수 있다.

입출고 내역, 발주 내역, 발주 딕테일 재고관리페이지, 게시글 리스트 등 정보를 들고와야 하는 페이지에 쓰임

```
#입고 검색
@router.post('/select/products/stockReceipts')
async def select_stockReceipts():
    try:
        conn = connect()
        curs = conn.cursor()
        sql = "SELECT * FROM stockReceipts"
        curs.execute(sql)
        rows = curs.fetchall()
        conn.close()
        result = [{"stockReceiptsQuantityReceived": row[0], "stockReceiptsReceiptDate": row[1],
                   "manufacturers_manufacturerName": row[2], "users_userid": row[3],
                   "products_productsCode": row[4], } for row in rows]

        return {"result": result}
    except Exception as e:
        print("Error : ", e)
        return {"result": "Error"}
```

프로젝트 결과



PYTHON



update



원래 있던 정보를 업데이트할 수 있다.

배송상태, 반품상태, 승인상태 등 변경되는 정보가 있을 때 쓰임

```
#출고 후 주문 상태 업데이트
@router.post("/update/Purchase/state")
async def update_Purchase_state(purchaseId:int =Form(...)):
    try:
        conn = connect()
        curs = conn.cursor()
        sql = "UPDATE Purchase SET PurchaseDeliveryStatus ='본사출고완료' WHERE purchaseId =%s"
        curs.execute(sql,(purchaseId,))
        conn.commit()
        conn.close()
        return {"result": "OK"}
    except Exception as e:
        print("Error : ",e)
        return{"result":"Error"}
```

프로젝트 결과



PYTHON



delete



필요 없는 정보는 삭제 할 수 있다.

필요 없는 데이터를 삭제하는데 쓰이지만 정보통신법상 유저에게는 지워진걸로 보이고 정보를 가지고 있어야 되기 때문에 관리자 페이지에는 딜렉트가 쓰지 않았다.

```
@app.route('/delete')
def delete():
    # 요청에서 파라미터 추출
    code = request.args.get("code")

    # 데이터베이스 연결 및 커서 생성
    conn = connection()
    curs = conn.cursor()
    try:
        sql = "DELETE FROM student WHERE scode = %s"
        curs.execute(sql, (code,))
        conn.commit()
    except Exception as e:
        conn.rollback()
        return jsonify([{'result': 'error', 'message': str(e)}])
    finally:
        conn.close()
    return jsonify([{'result': 'OK'}])
```

04

프로젝트 결과



Github

04 프로젝트 결과 GitHub

The screenshot shows the GitHub repository page for 'Black_Market'. At the top, it displays the repository name 'ChangJun0716 / Black_Market' and a search bar with placeholder text 'Type / to search'. Below the search bar are navigation links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, and Insights. The main content area shows a project card for 'Black_Market' (Public). It includes a 'Pin' button and an 'Unwatch' button with a count of 1. Below the card, there's a summary of branches and tags: 'main' (1 Branch, 0 Tags). A 'Code' dropdown menu is open, showing options like 'Go to file', 't', '+', and 'Code'. The repository's commit history is listed, starting with a commit from 'ChangJun0716 CJ_Work_17' made 20 hours ago. Other commits include 'Update blackMarketData' by 'Data' folder, 'CJ_Work_17' by 'Database' folder, 'CJ_Work_17' by 'Project/black_market_app' folder, 'Update .gitignore' by '.gitignore' file, and 'Update README.md' by 'README.md' file.

The screenshot shows the GitHub README page for the 'Black Market App'. The title 'README' is at the top, followed by a horizontal line. The page contains the following text:
Project Name : Black Market App
Description : Local 데이터베이스와 flutter 를 이용하여 고객, 대리점장, 본사 직원 모두가 이용 가능한 앱 제작.
사용자는 로그인 시 분류 값에 의해 각각 다른 페이지로 이동하여 앱을 이용하는 형식.
고객 : 상품의 검색과 구매
대리점장 : 고객이 구매한 상품의 입고 확인 및 상태 관리, 반품 정보를 본사에 전달
본사 직원 : 재고관리를 통한 제품 물량 관리, 반품 원인규명 요청, 주문에 따른 대리점 별 재고 배송, 매출 확인 (대리점, 기간 등)
Date :
1차 db라이트를 사용한 개발 2025.05.02 ~ 2025.05.08 2차 mysql을 사용한 개발 2025.05.16 ~ 2025.05.20
Author : ChangJun Lee, Sua Kim, Inwhan Kang, Sanghoon Lee
Init Version : 2025.05.02

Github web service 를 이용해 팀원들이 project 와 관련된 file 들을 서로 공유함

- **Data:** Database 에 입력할 data 를 보관하는 폴더
- **Database:** Python 함수들을 작성한 파일 및 SQL database 를 보관하는 폴더
- **Project/black_market_app:** 앱 제작을 진행하는 flutter file 이 보관된 폴더!
- **README:** 프로젝트에 대한 내용을 작성한 페이지

04 프로젝트 결과 Git Hub

Commits on May 19, 2025

CJ_Work_17 · ChangJun0716 committed 20 hours ago
수아님 pull request #20 from kimsua88/main · ChangJun0716 authored yesterday
SA-work11 · kimsua88 committed yesterday
CJ_Work_16 · ChangJun0716 committed yesterday
CJ_Work_15 · ChangJun0716 committed yesterday
인환님 pull request #19 from littlehippo01/main · ChangJun0716 authored 2 days ago
대리점 api로 변경 · littlehippo01 committed 2 days ago

CJ_Work_15

Update:

1. Database:

1-1: customer_products_detail.dart:
1-1-1: 사용자가 선택한 제품의 소개글에 포함된 이미지들의 url index 를 포함하여 보내주는 함수 제작
1-1-2: 사용자가 선택한 제품의 소개글에 포함된 이미지 list 를 보내주는 함수 제작
1-1-3: 사용자가 값들을 지정한 뒤 장바구니에 담거나 주문을 진행할 때 purchase table 로 Insert 하는 함수.

1-2: customer_shopping_cart.dart:
1-2-1: 장바구니 리스트에서 사용자의 userid 와 결제 상태: '장바구니' 인 data 들을 불러오는 함수 제작
1-2-2: 위 1-2-1 함수의 이미지를 불러오는 함수 제작
1-2-3: 사용자가 등록한 장바구니 리스트의 data 를 삭제하는 함수 제작
1-2-4: 사용자가 등록한 장바구니 리스트에서 원하는 제품을 주문하는 함수 제작

1-3: customer_purchase_list.dart:
1-3-1: 구매 목록 페이지에서 사용자가 주문한 내역을 불러오는 함수 제작

1-4: customer_purchase_detail.dart:
1-4-1: 사용자가 구매 목록에서 구매한 data 를 ontap 하여 상세보기 페이지로 넘어갔을 때 해당 내역의 data 를 넘겨주는 함수 제작
1-4-2: 위 1-4-1 의 image 를 보내주는 함수 제작

1-5: customer_select_store.dart:
1-5-1: 사용자가 대리점을 선택하기 위해 대리점 선택 페이지로 들어갔을 때 대리점들의 data 를 불러오는 함수 제작

Commit 을 할 때에는 comment 로 작업 내용을 작성하여 다른 팀원들이 update 내용을 알 수 있도록 함

- **Comment:** 작업 내용을 CRUD 로 대분류, Database, App 과 Data 로 중분류 하여 작성

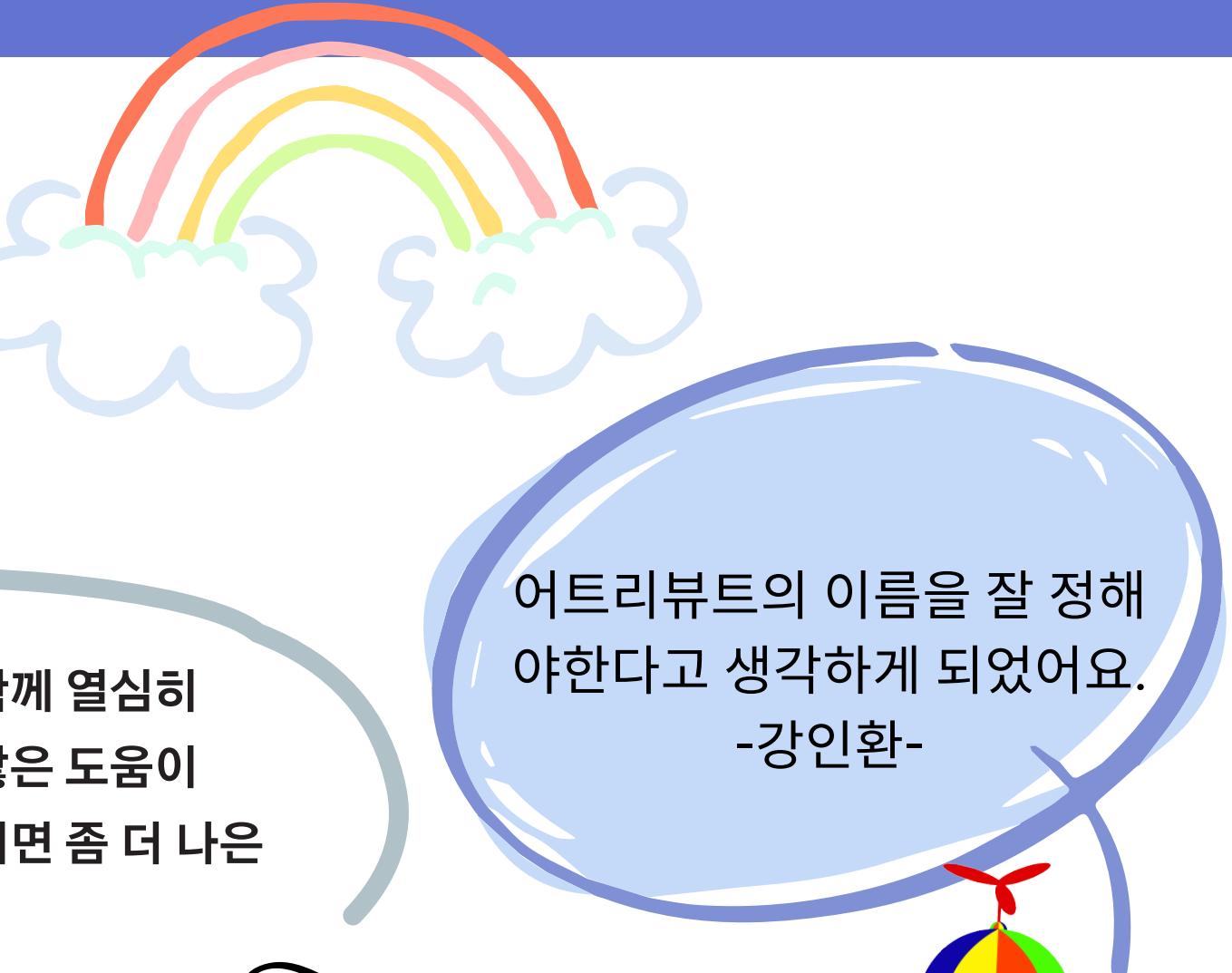
05 자체평가 의견

거의 2주간 잠도 못 자고
만들었어요
DB를 정말 신경써서 만들어야 된다는
걸 알게 되었고
다음엔 ui을 손으로 그리는게 아닌
피그마를 이용해 보려고 해요



팀장으로서 많이 부족 했는데 함께 열심히
진행한 팀원 분들께 감사하고 많은 도움이
되었어요 다음에 다시 만나게 되면 좀 더 나은
모습으로 함께 해요~

- 창준 -



어트리뷰트의 이름을 잘 정해
야한다고 생각하게 되었어요.
-강인환-

THANK YOU
