

더조은 컴퓨터 아카데미 강남캠퍼스
빅데이터 기반 AI활용 앱&웹 개발자 취업캠프

신발 가게 운영 앱 만들기

MySQL 데이터 베이스 + FastAPI 연동

BCD
MART



고용노동부

THE JOEUN

TEAM4 과제 프리젠테이션

이권형 / 이학현 / 전감성 / 전종익

목차

1

요구 조건 분석,
프로젝트 개요와 목적

03 - 07

PAGE

2

팀 구성 및 역할

08

PAGE

3

수행 절차 및 방법

09 - 13

PAGE

4

기능 및 UI/UX 구성 방식

14 - 17

PAGE

5

수행 결과

18 - 24

PAGE

6

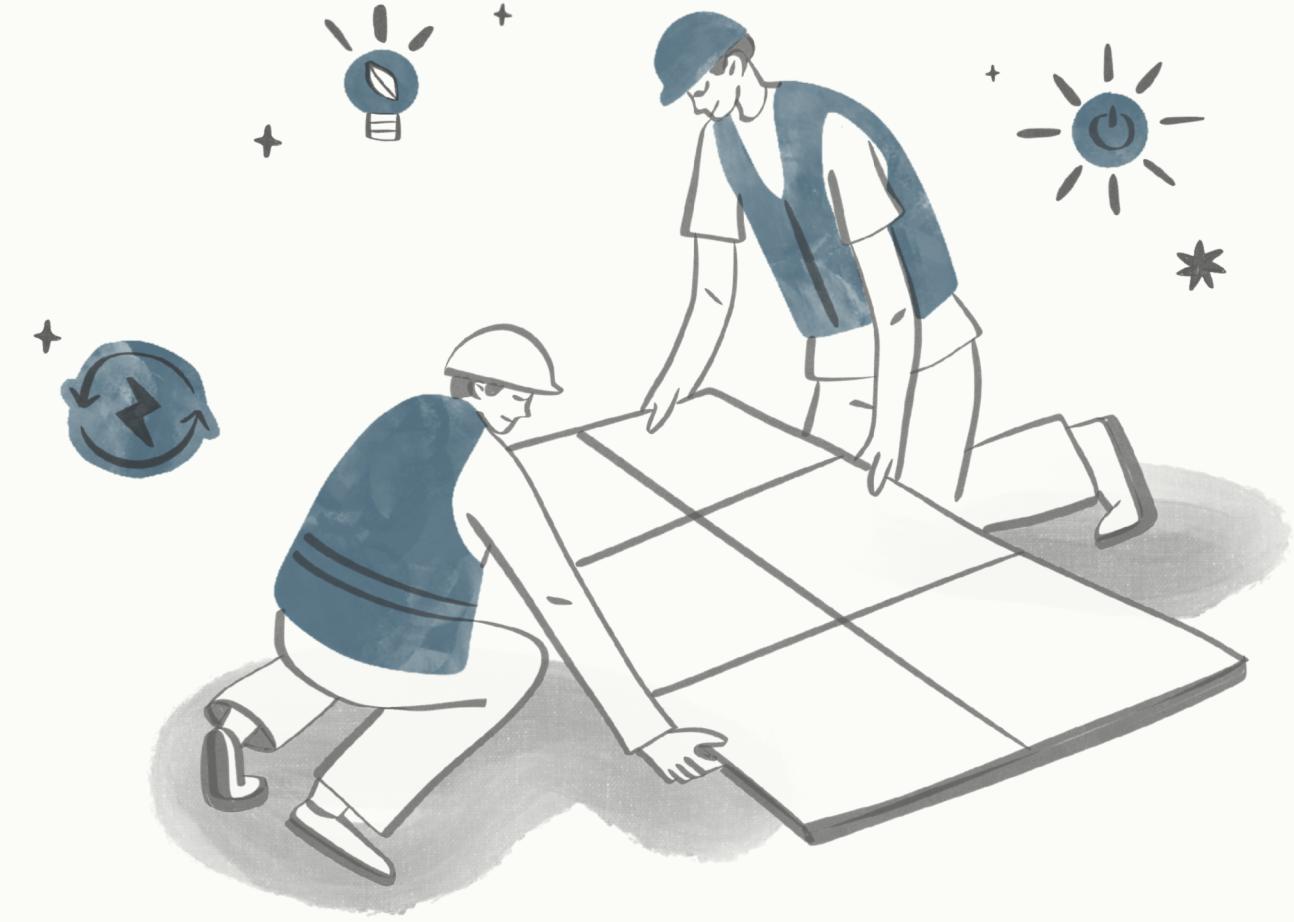
자체 평가 의견

25

PAGE

요구 사항 분석

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 신청 후 즉시 해당 대리점으로 신발을 발송한다.
8. 대리점장은 해당 월일의 매출을 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,
이때 사원이 품의를 작성하여 팀장, 이사 까지 결재를 득하면 자동으로 발주가 발생한다.



프로젝트의 개요 – 목적

개요

신발 가게를 위한
고객 + 대리점 + 본사
매장 운영 통합 관리 시스템



중요성

매장 운영을 디지털화하여 고객 응대를 통합관리
소비자 → 상품을 손쉽게 탐색 및 구매
판매자 → 매출, 재고 실시간 파악 가능



목적

고객 중심의 모바일 신발 쇼핑 경험을 구축
매장에서 재고, 반품, 매출 데이터 효율적 운영



“우리는 경쟁자가 아니라 고객에게 집착한다. 고객이 뭘을 원하는지 출발점으로 삼고 거꾸로 서비스를 설계한다”

– 아마존 창업자: 제프 베조스 –

프로젝트의 개요 - 구현내용

FastAPI와 MySQL DB를 연동한 고객 신발구매 - 대리점 관리 - 본사 관리 시스템

고객

- 상품 탐색 및 즉시 구매
- 장바구니 기능
- 개인정보 변경
- 본인의 구매내역 확인
- 대리점 위치확인
- 반품 내용 확인

대리점

- permission에 따른 로그인 페이지 차별화
- 대리점 관리 페이지로 연결
- 대리점 별 매출 관리
- 반품 접수 가능

본사

- permission에 따른 로그인 페이지 차별화
- 본사 관리 페이지로 연결
- 전체적인 매출 관리 확인 가능
- 날짜, 매장, 상품 별 매출 그래프
- 반품 내용 확인
- 상품 추가 가능
- 재고가 부족한 상품 존재 시
- 사원 → 팀장 → 임원 순서로 기안 상신, 승인
- 임원 승인 시 즉시 발주, 재고보충

프로젝트의 개요 – 개발환경, 활용 패키지

개발환경

VsCode(Flutter)

Python 3.12.7(anaconda3)

MySQL 8.0.42



활용 패키지

```
#카카오API: remedi_kopo: ^0.0.2
# 위치 정보: geocoding: ^3.0.0
# GPS 신호: geolocator: ^14.0.0
# 지도: flutter_map: ^8.1.1
# 지도를 현위치로 옮기기: latlong2: ^0.9.1
# Path : path: ^1.9.0
# path_provider: path_provider: ^2.1.5
#Getx: get: ^4.7.2
#get storage: get_storage: ^2.1.1
# Slider: flutter_slidable: ^4.0.0
#image_picker: image_picker: ^1.1.2
#syncfusion_flutter_charts: syncfusion_flutter_charts: ^29.1.38
#http: http: ^1.4.0
```

프로젝트의 개요 - 구조

FastAPI와 MySQL DB를 연동한 고객 신발구매 - 대리점 관리 - 본사 관리 시스템

Flutter(프론트)

team4shoeshop_refactoring

기존 sqlite DB를 이용하는 구조에서 리팩토링

Python(백엔드)

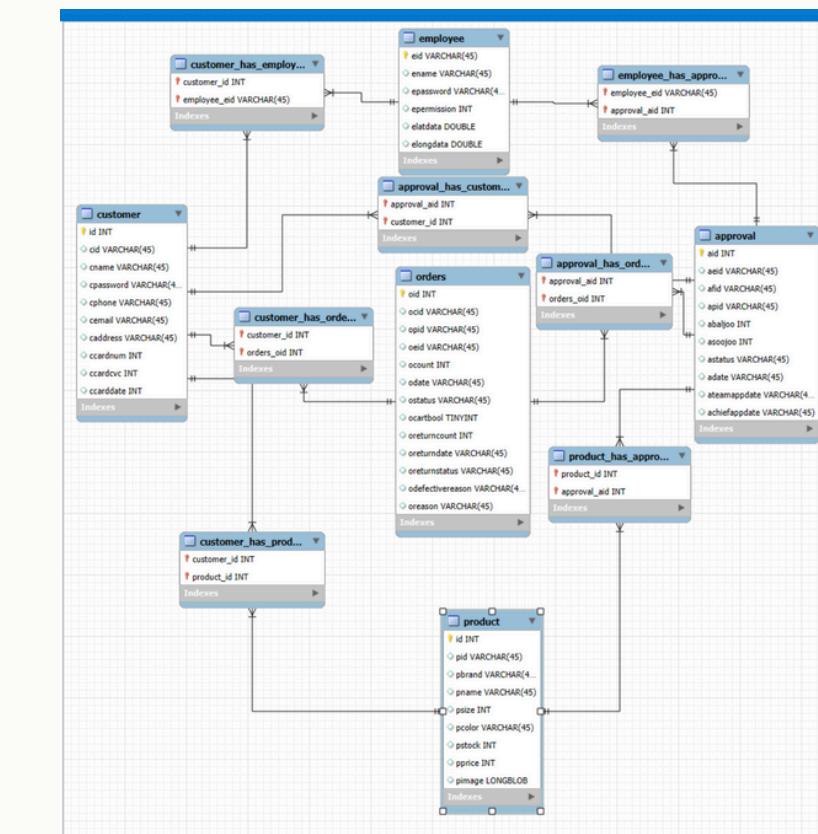
shoeshop.py

MySQL DB와 프론트를 이어주는
Python Code

MySQL(DB)

shoeshop.db

ERD를 토대로 제작된 DB구조



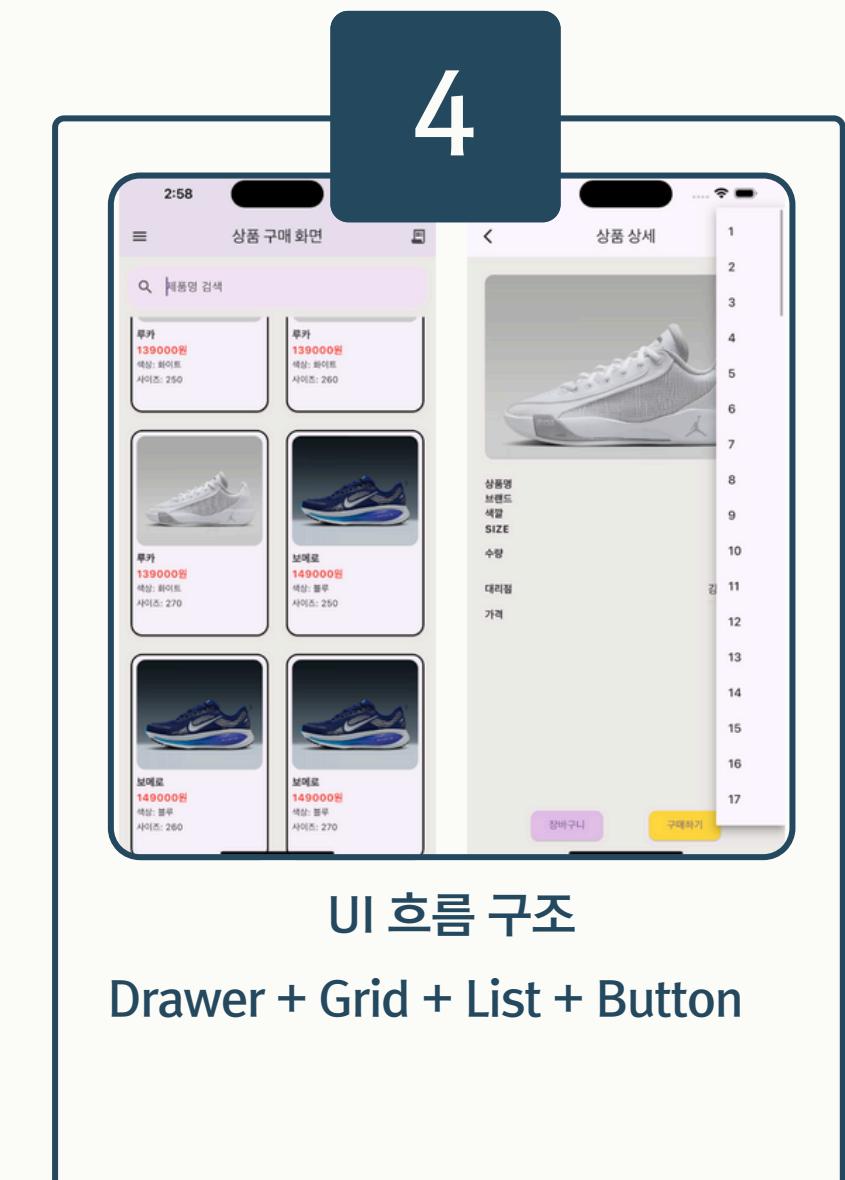
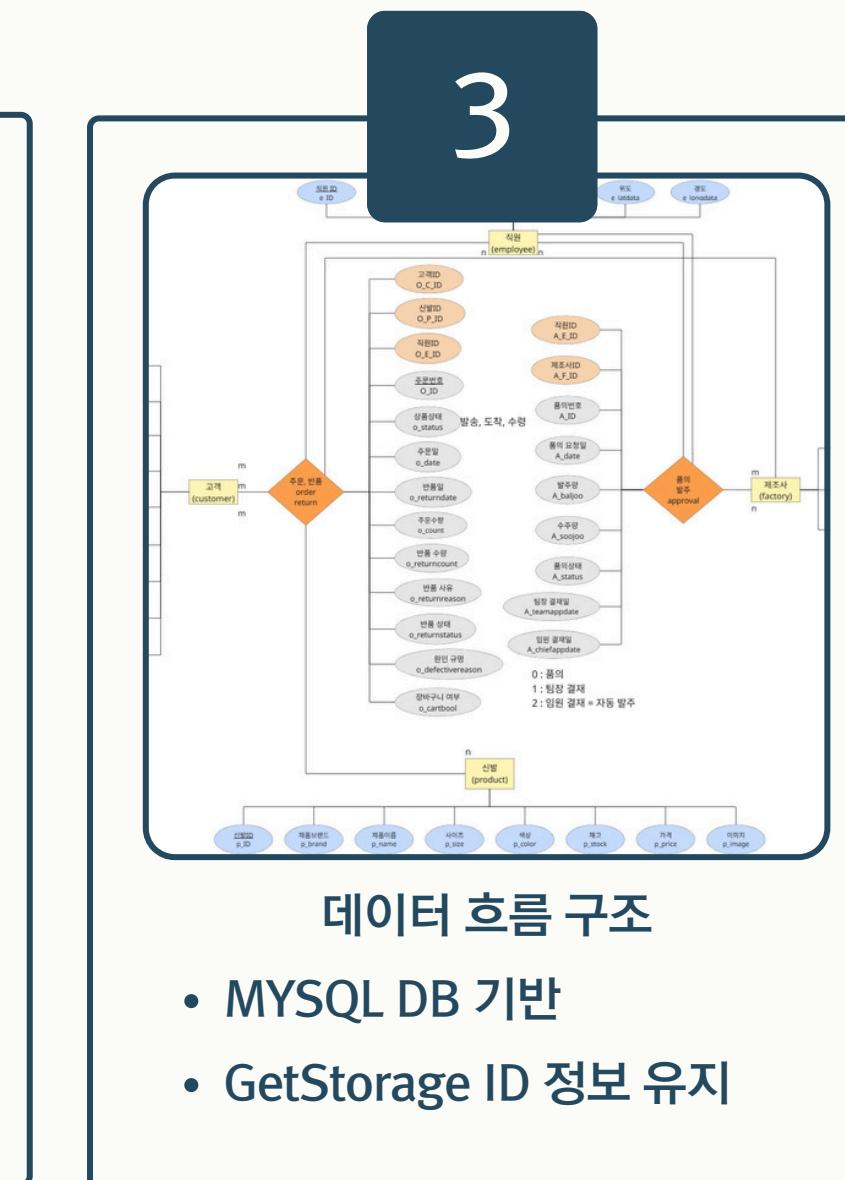
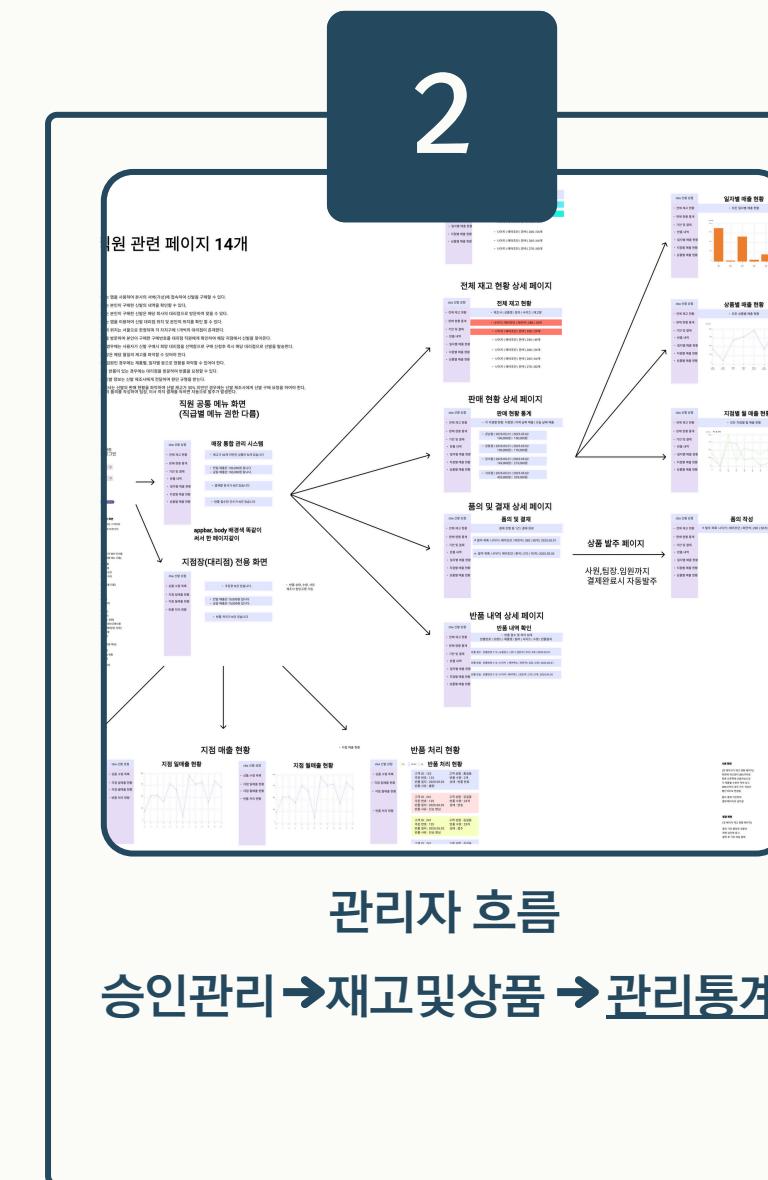
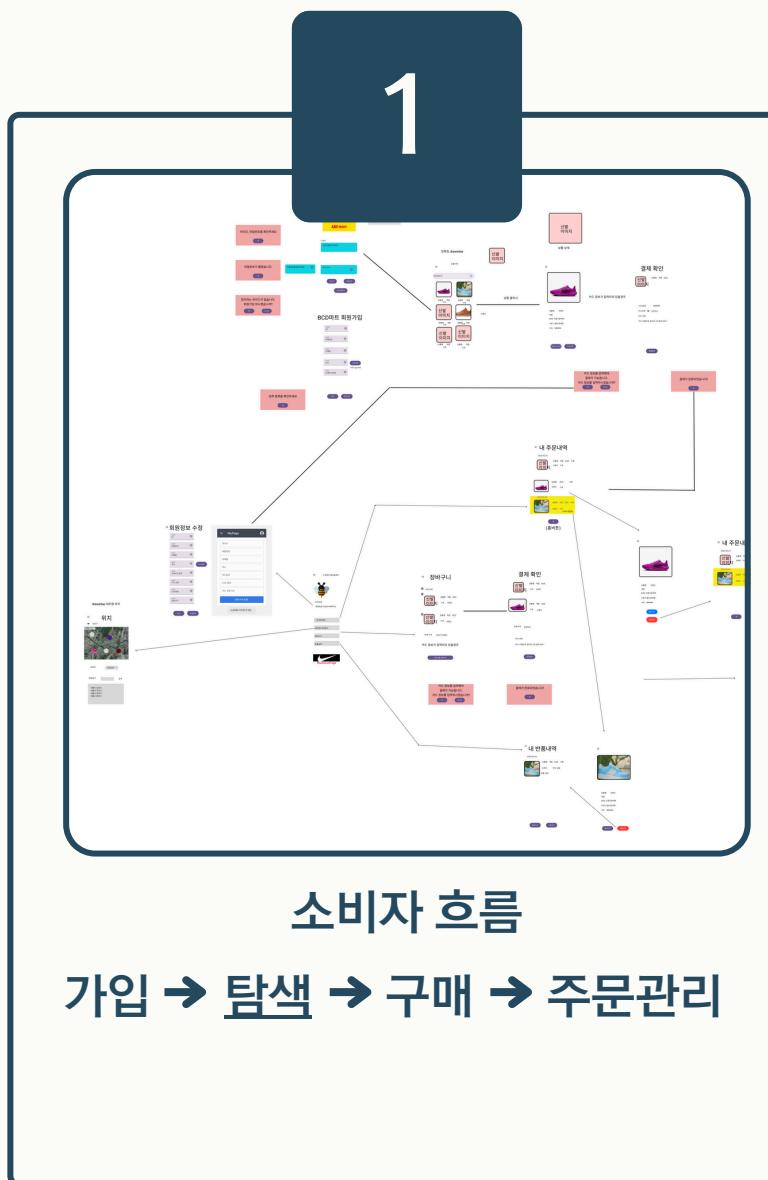
프로젝트 팀 구성 및 역할

이름	역할	담당 업무
전종익	팀장	고객 화면 구성, 깃허브 관리
이권형	팀원	고객 화면 구성, 본사 재고 및 상품 추가 구성
이학현	팀원	본사 메인화면, 품의 및 결재 시스템 구현
전감성	팀원	대리점 화면 지점별 총매출 반품시스템

프로젝트 수행 절차 및 방법(개요)

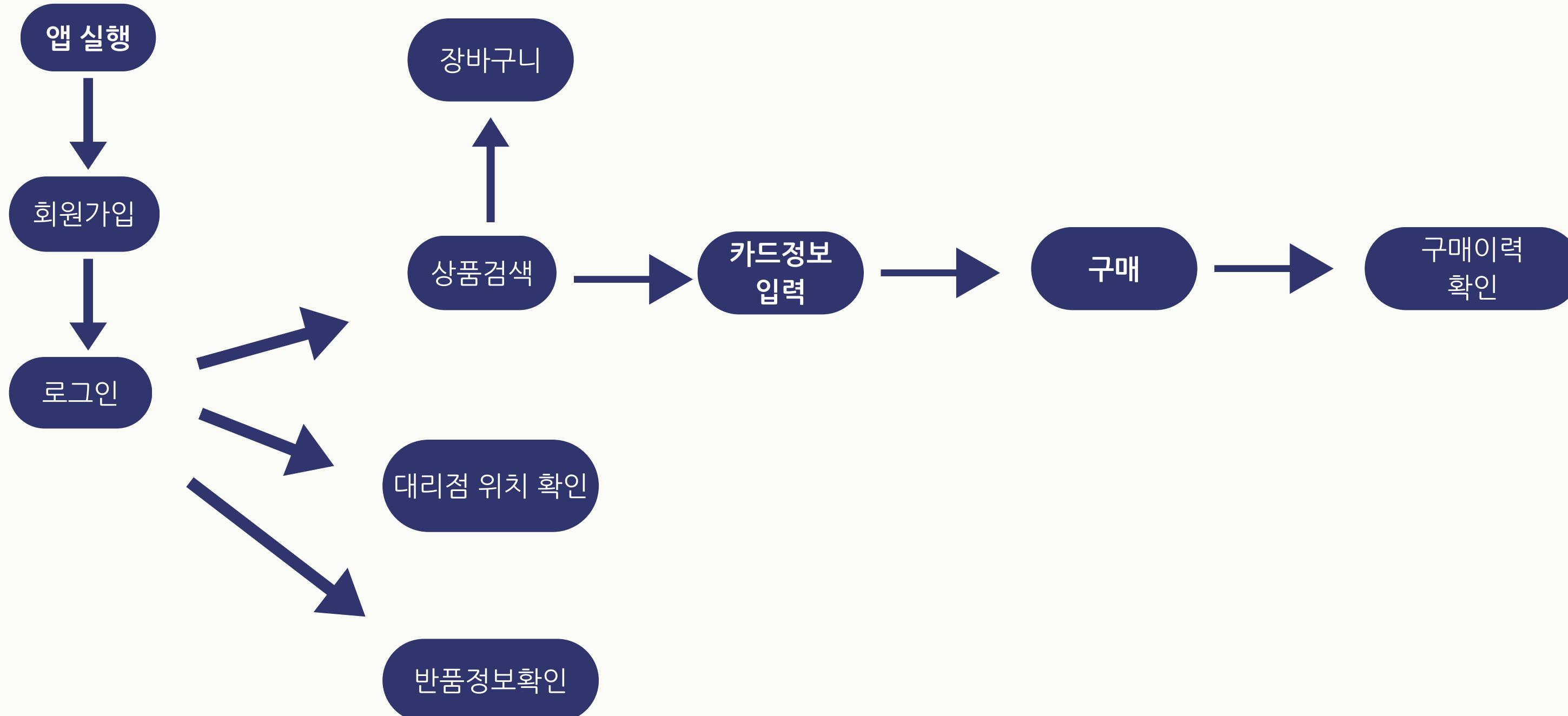
구분	기간	활동	비고
사전 기획	2025년 5월 17일	<ul style="list-style-type: none">ERD구성을 토대로 DB생성역할 분담 선정	<ul style="list-style-type: none">MY SQL 데이터베이스 사용
백엔드 구조 만들기	2025년 5월 18~19일	<ul style="list-style-type: none">Python을 사용한 코드 만들기데이터베이스 연동 여부 확인	<ul style="list-style-type: none">Fast API 사용
프론트엔드 구조 만들기	2025년 5월 18-19일	<ul style="list-style-type: none">vsCode사용 페이지 구성	<ul style="list-style-type: none">flutter 사용
전체 코드 구조작업 및 작동 여부 점검	2025년 5월 20일	<ul style="list-style-type: none">페이지 별 오류수정서브유닛테스트	<ul style="list-style-type: none">최적화 및 오류 수정
앱 제작 완료 및 발표 자료 제작	2025년 5월 20일	<ul style="list-style-type: none">코드 취합, 최종 구동 점검발표 자료 작성 및 구동 영상 촬영	<ul style="list-style-type: none">최종점검

프로젝트 수행 절차 및 방법 (사전기획 1/3)

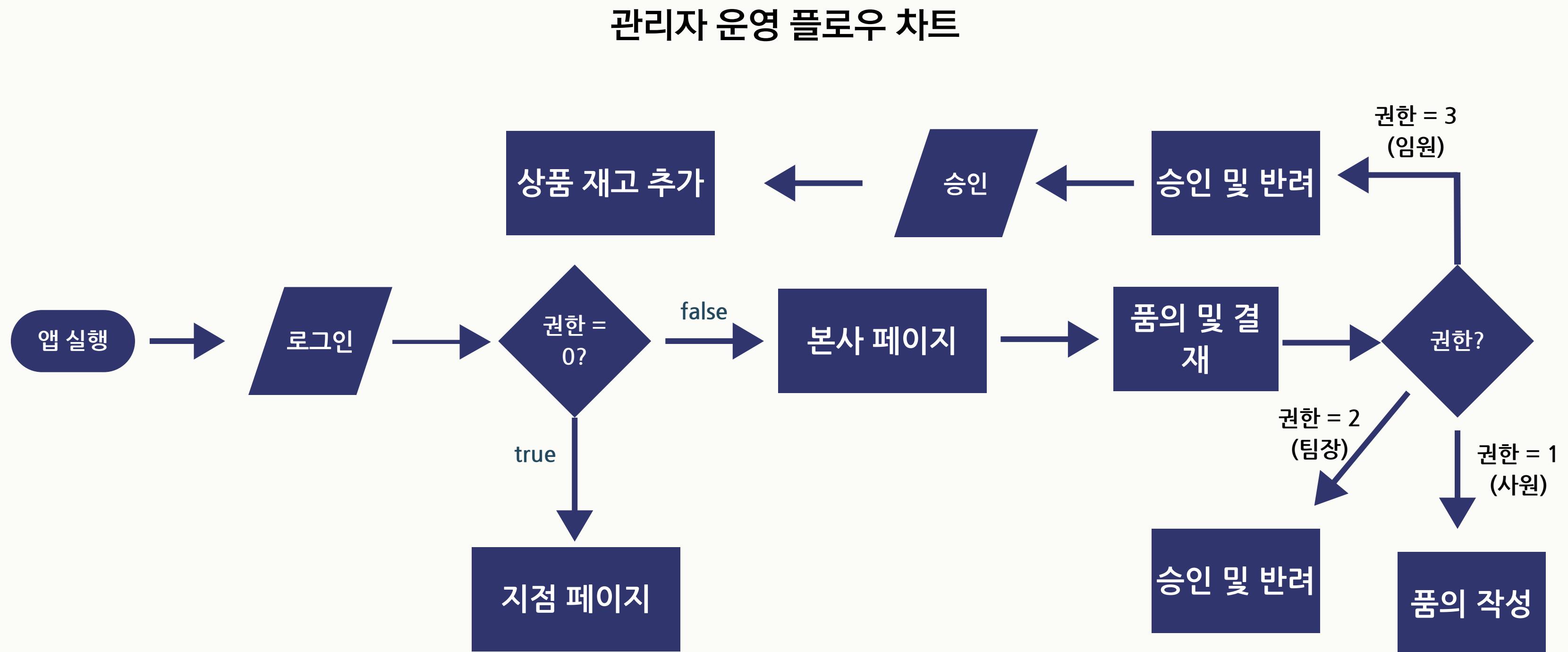


프로젝트 수행 절차 및 방법 (사전기획 2/3)

고객 서비스 이용 플로우 차트



프로젝트 수행 절차 및 방법 (사전기획 3/3)



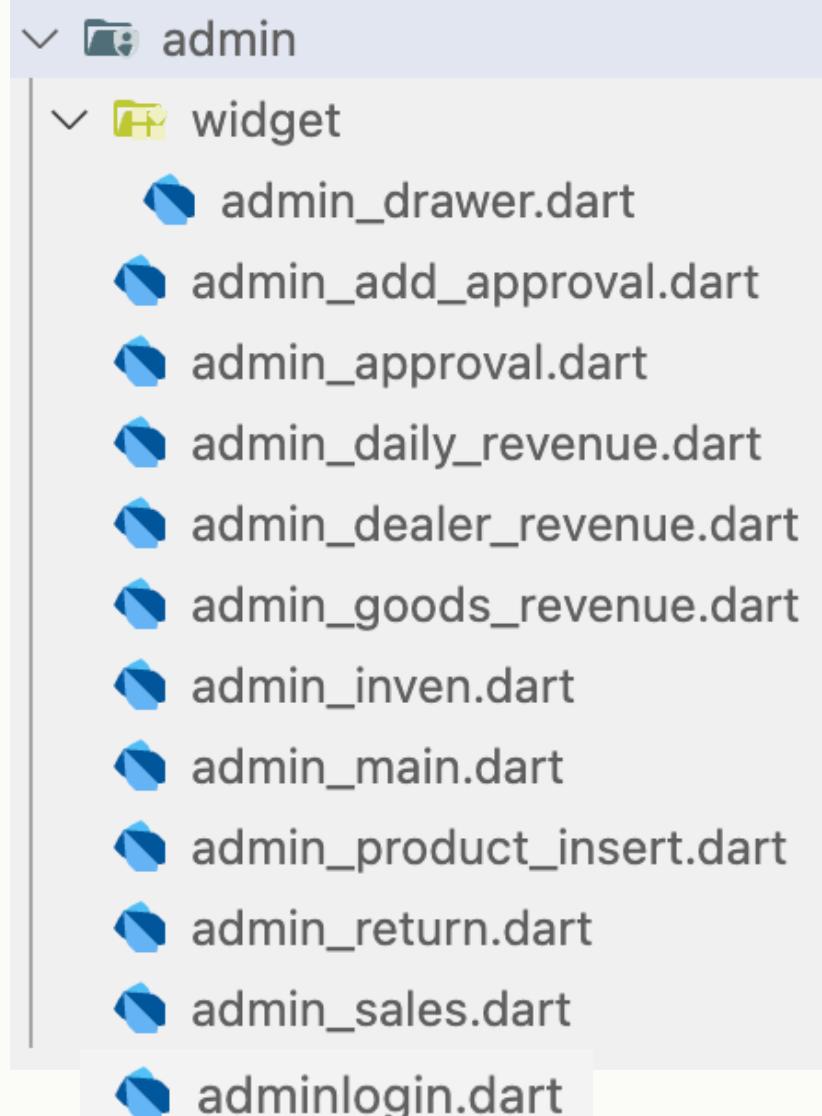
프로젝트 실행 절차 및 방법 (수행)

백엔드 파일

1

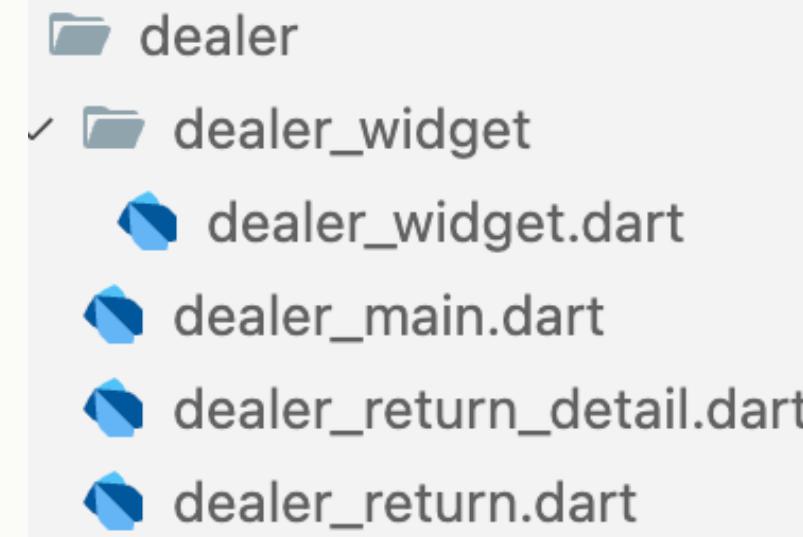


2

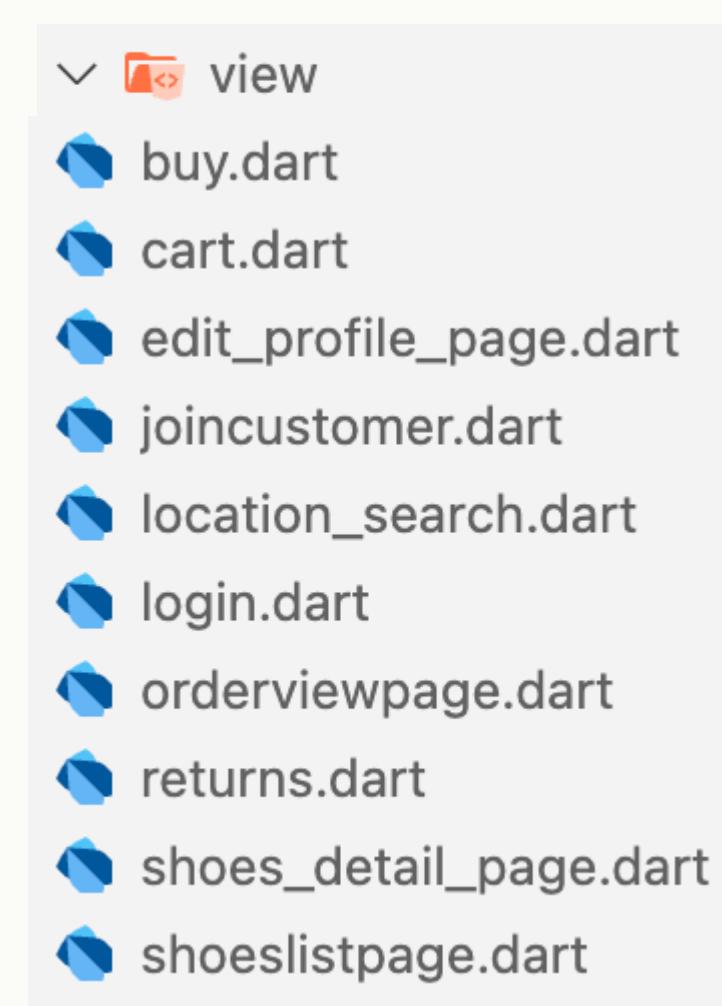


프론트 엔드 파일

3



4



기능 및 UI/UX 구성 방식(고객 기능)



기능 및 UI/UX 구성 방식(관리자 기능)



전체 재고 현황

상품별 재고 수량 확인
30%이하의 경우 시각화 알림

결재 목록 관리

발주 품의서 작성, 품의 목록 확인

판매 통계 현황

오늘의 매출 현황 파악
각 지점별 및 상품별 매출 파악

반품 내역 확인

반품 사유 및 제조사 귀책 여부 확인

상품 등록 기능

신규 상품 추가 가능

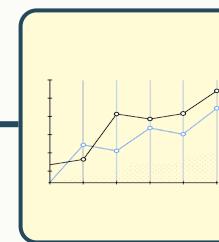
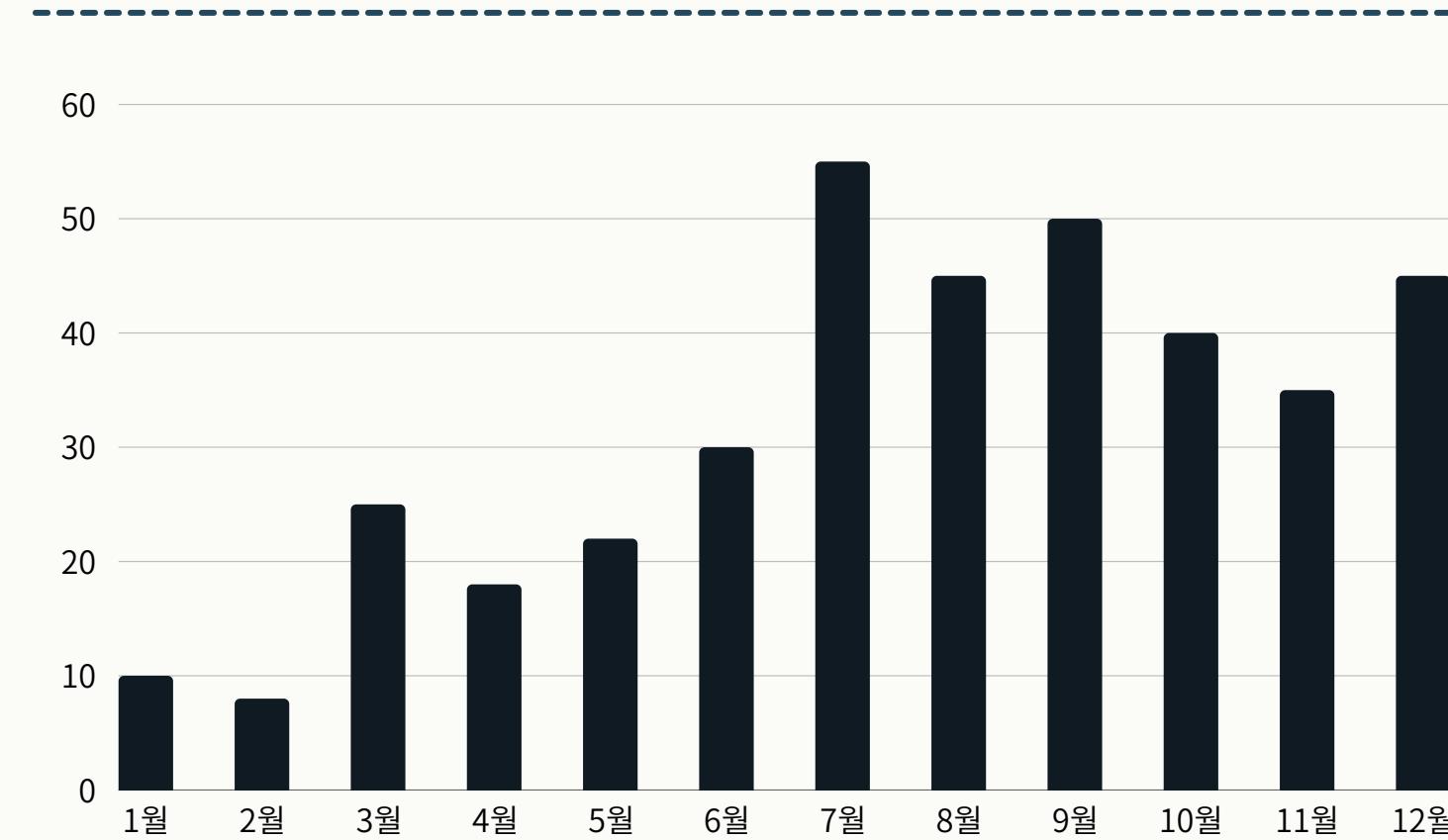
매출 현황 확인

일별, 지점별, 상품별 매출 차트 비교

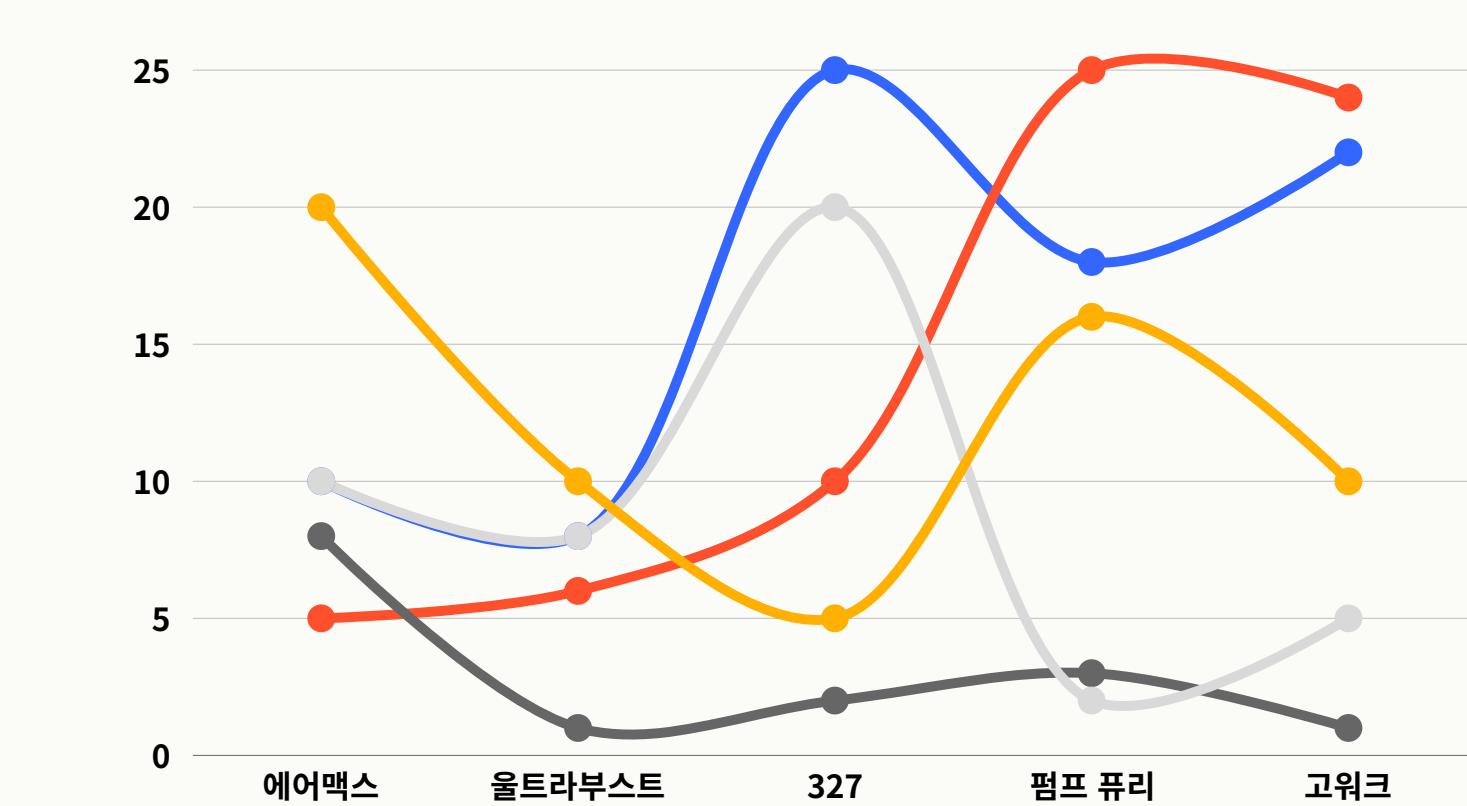
기능 및 UI/UX 구성 방식(통계 기능)



일,월별 매출 현황



상품별 매출 현황



기능 및 UI/UX 구성 방식(UI / UX)

✓ 드로워 메뉴(직관적)

고객과 관리자 모두 Drawer 메뉴를 통해 주요 기능에 접근
(로그인 사용자의 정보도 Drawer 상단에 표시)

✓ 카드 기반 목록(명확성)

GridView 및 ListView 형태의 상품 목록 및 정보 알림
(상품의 시각적 비교 가능)

✓ 상태 및 정보 유지(지속적)

페이지 이동 시 사용자의 정보를 계속 유지
(GetStorage, GetX 활용)

✓ 반응형 입력 요소(선택적)

Textfield, CheckBox, DropdownButton으로 고객이 값 선택
(입력 즉시 반응, 입력값 검증)

프로젝트 수행 결과

① 요구 사항 분석 후 데이터베이스 생성



Table: customer	
Columns:	
id	int AI PK
cid	varchar(45)
cname	varchar(45)
cpassword	varchar(45)
cphone	varchar(45)
cemail	varchar(45)
caddress	varchar(45)
ccardnum	varchar(45)
ccardcvc	int
ccarddate	int

id	pid	pbrand	pname	psize	pcolor	pstock	pprice	pimage
1	prd001	나이키	에어맥스	250	화이트	28	129000	BLOB
2	prd002	나이키	에어맥스	260	화이트	100	129000	BLOB
3	prd003	나이키	에어맥스	270	화이트	100	129000	BLOB
4	prd004	나이키	루카	250	화이트	25	135000	BLOB
5	prd005	나이키	루카	260	화이트	100	135000	BLOB
6	prd006	나이키	루카	270	화이트	100	135000	BLOB
7	prd007	나이키	보메로	250	블루	28	139000	BLOB
8	prd008	나이키	보메로	260	블루	100	139000	BLOB
9	prd009	나이키	보메로	270	블루	100	139000	BLOB
10	prd010	나이키	빅토리투어	250	블랙	28	159000	BLOB
11	prd011	나이키	빅토리투어	260	블랙	100	159000	BLOB
12	prd012	뉴발란스	327	250	화이트	15	139000	BLOB
13	prd013	뉴발란스	327	260	화이트	50	139000	BLOB
14	prd014	뉴발란스	327	270	화이트	50	139000	BLOB
15	prd015	뉴발란스	327	280	화이트	60	139000	BLOB
NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE	NONE

프로젝트 수행 결과 - 고객 회원가입

백엔드 코드 작성

```
@app.get("/check_customer_id") # 회원가입시 cid로 중복 체크
async def check_customer_id(cid: str):
    try:
        conn = connect()
        curs = conn.cursor()
        sql = "SELECT * FROM customer WHERE cid = %s"
        curs.execute(sql, (cid,))
        row = curs.fetchone()
        conn.close()

        if row:
            return {"result": "exists"} # 중복
        else:
            return {"result": "available"} # 사용 가능
    except Exception as e:
        print("Error:", e)
        return {"result": "error"}
```



```
@app.post("/insert_customer") # 회원가입시 회원정보 insert
async def insert_customer(
    cid: str = Form(...),
    cname: str = Form(...),
    cpassword: str = Form(...),
    cphone: str = Form(...),
    cemail: str = Form(...),
    caddress: str = Form(...)):
    try:
        conn = connect()
        curs = conn.cursor()
        sql = """
            INSERT INTO customer (cid, cname, cpassword, cphone,
            VALUES (%s, %s, %s, %s, %s)
            """
        curs.execute(sql, (cid, cname, cpassword, cphone, cemail))
        conn.commit()
        conn.close()
```

FastAPI docs에서 입력테스트

POST /insert_customer Insert Customer

Parameters

No parameters

Request body required

cid * required string

cname * required string

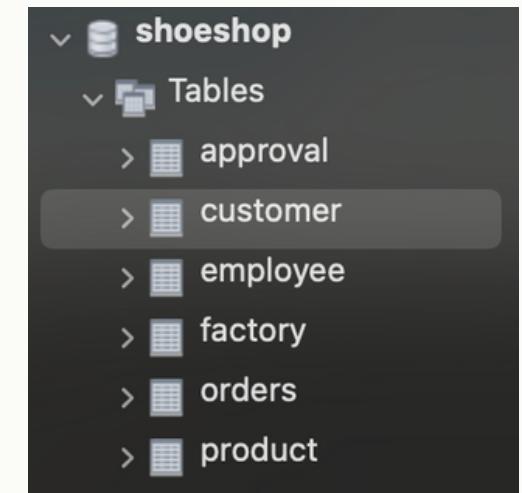
cpassword * required string

cphone * required string

cemail * required string

caddress * required string

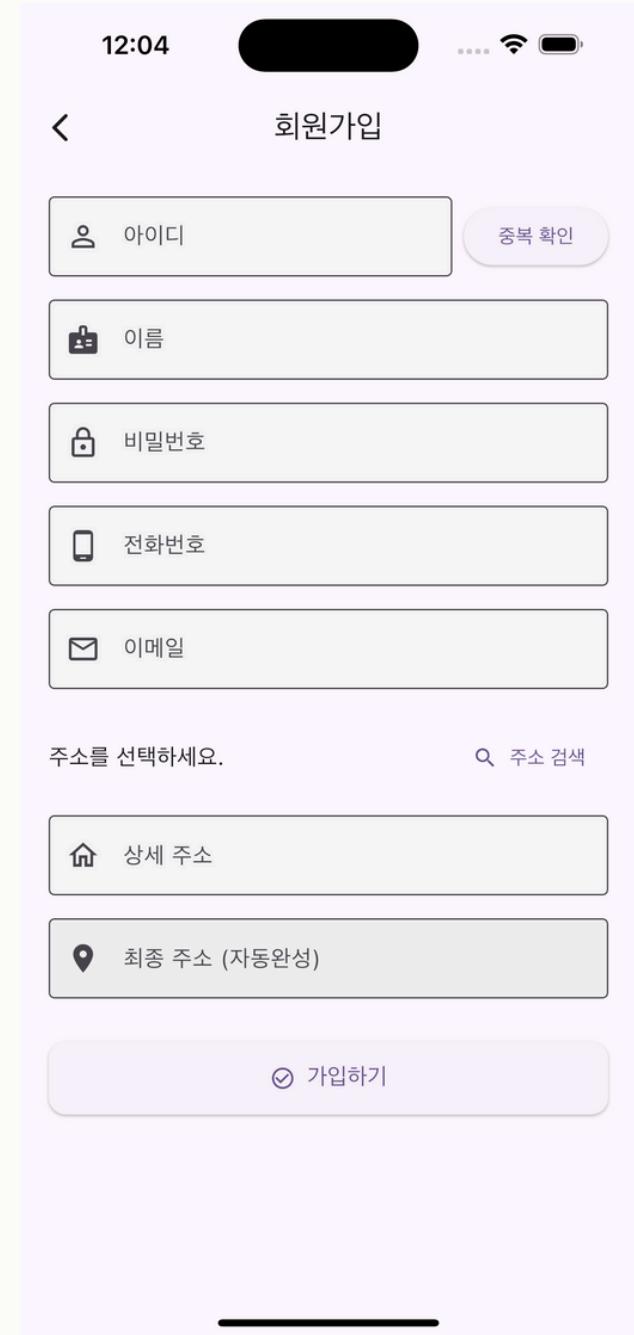
DB 데이터 확인



Result Grid Filter Rows: Search

ID	CID	CNAME	CPASSWORD
6	yubee	유비	1234
8	zangbee	장비	1234
9	kwanwoo	관우	1234

프론트 작성



프로젝트 수행 결과 – 상품 목록

1. 백+프론트 엔드 코드 만들기

```

@app.get("/product_list") # 상품 정보를 확인
async def product_list():
    try:
        conn = connect()
        curs = conn.cursor()
        curs.execute("SELECT pid, pname, pprice, pcolor, psize, pbrand,
                     ORDER BY pname")
        rows = curs.fetchall()
        conn.close()

        result = [
            {
                "pid": row[0],
                "pname": row[1],
                "pprice": row[2],
                "pcolor": row[3],
                "psize": row[4],
                "pbrand": row[5],
                "pstock": row[6],
                "image_url": f"http://127.0.0.1:8000/view/{row[0]}"
            } for row in rows
        ]

        return {"result": result}
    except Exception as e:
        print("Error:", e)
        return {"result": "error", "message": str(e)}

    class Shoeslistpage extends StatefulWidget {
        const Shoeslistpage({super.key});

        @override
        State<Shoeslistpage> createState() => _ShoeslistpageState();
    }

    You, 3시간 전 | 2 authors (smitepaladin and one other)
    class _ShoeslistpageState extends State<Shoeslistpage> {
        final box = GetStorage(); // GetStorage 모듈
        String userId = "";

        List<Map<String, dynamic>> _products = [];
        List<Map<String, dynamic>> _filteredProducts = [];
        String _searchText = "";

        @override
        void initState() {
            super.initState();
            userId = box.read('pUserId') ?? '';
            fetchProducts();
        }

        Future<void> fetchProducts() async {
            final url = Uri.parse("http://192.168.50.236:8000/product_list");
            final response = await http.get(url);
            final data = jsonDecode(utf8.decode(response.bodyBytes));

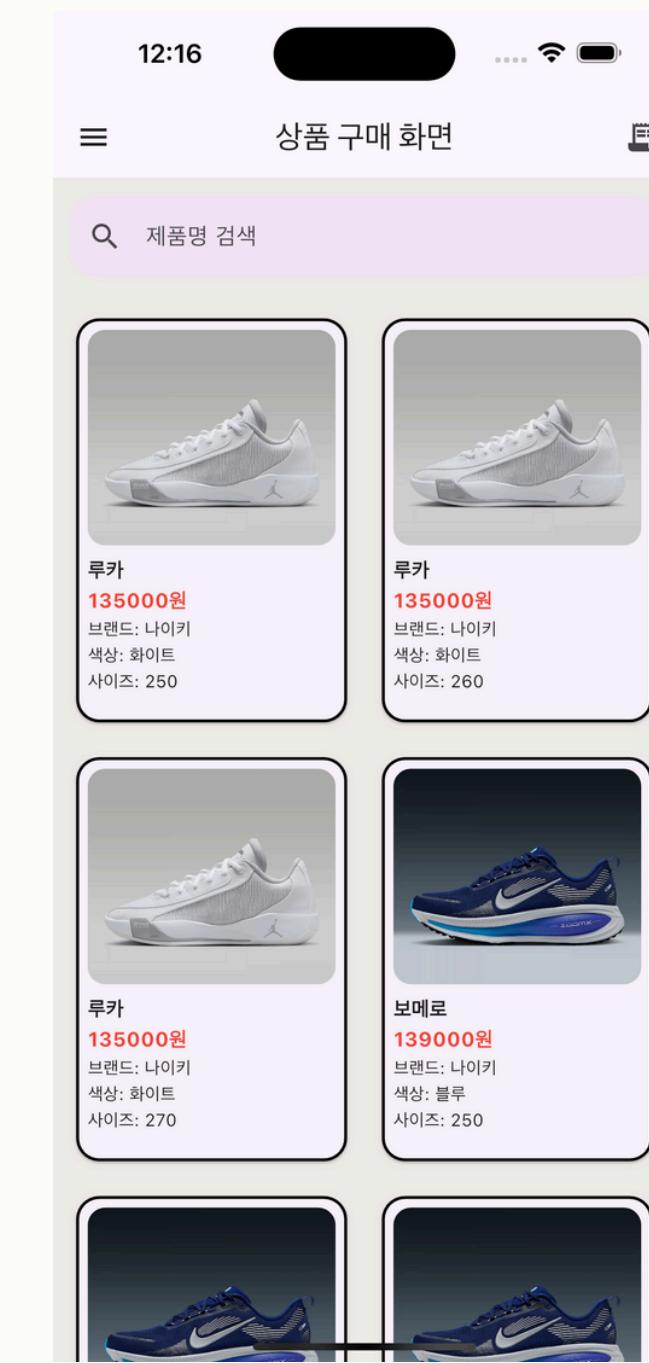
            if (data["result"] is List) {
                setState(() {
                    _products = List<Map<String, dynamic>>.from(data["result"]);
                    _filteredProducts = _products;
                });
            }
        }

        Widget _buildProductCard(Map<String, dynamic> product) {
            final selectedSize = product['psize'] ?? 250;
    
```

2. MySQL 입력 데이터로 상품 목록 확인

pid	pbrand	pname	psize	pcolor	pstock
prd001	나이키	에어맥스	250	화이트	28
prd002	나이키	에어맥스	260	화이트	100
prd003	나이키	에어맥스	270	화이트	100
prd004	나이키	루카	250	화이트	25
prd005	나이키	루카	260	화이트	100
prd006	나이키	루카	270	화이트	100
prd007	나이키	보메로	250	블루	28
prd008	나이키	보메로	260	블루	100
prd009	나이키	보메로	270	블루	100
prd010	나이키	빅토리투어	250	블랙	28
prd011	나이키	빅토리투어	260	블랙	100
prd012	뉴발란스	327	250	화이트	15
prd013	뉴발란스	327	260	화이트	50
prd014	뉴발란스	327	270	화이트	50
prd015	뉴발란스	327	280	화이트	60

3. 앱 화면에서 상품 목록 클릭



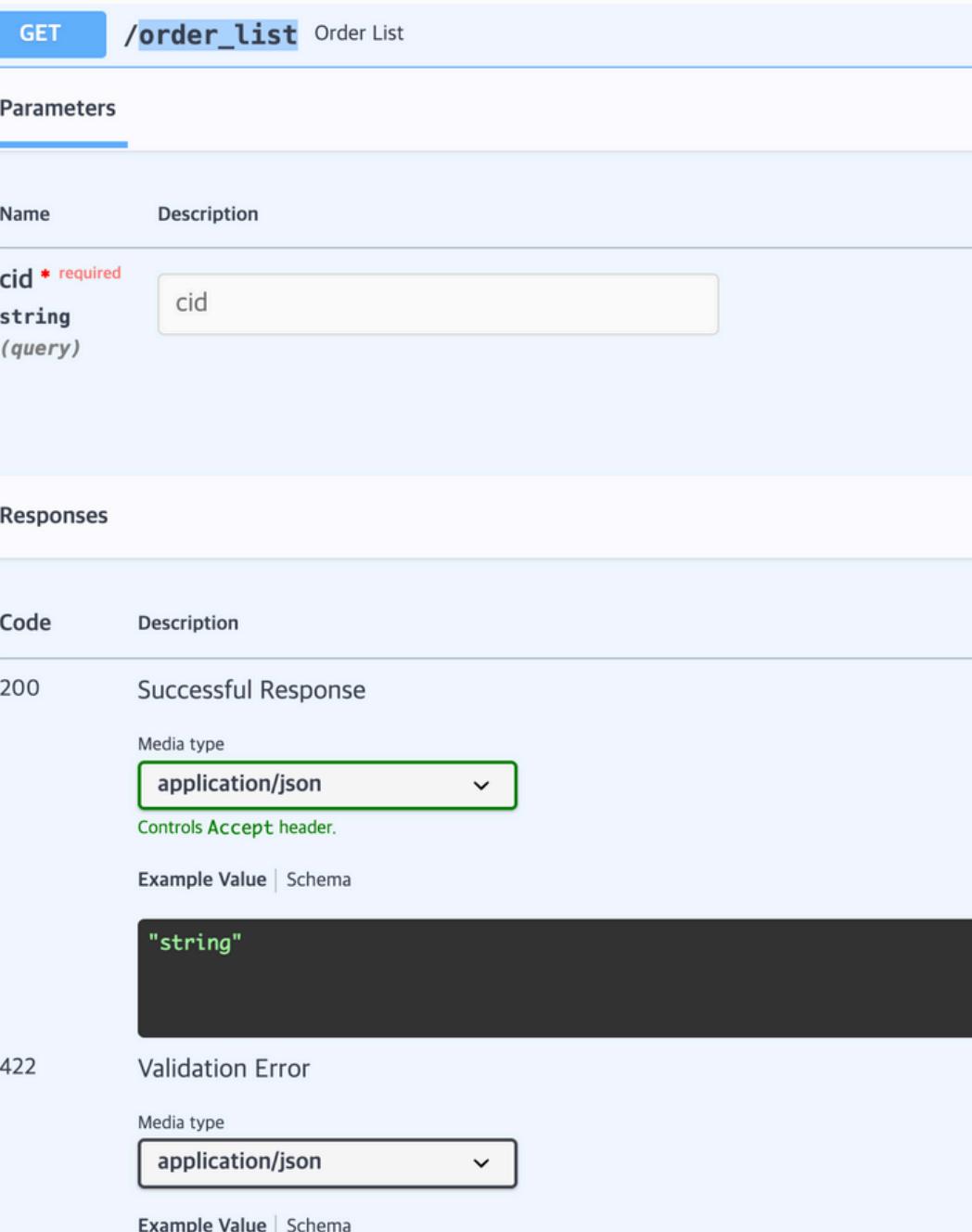
프로젝트 수행 결과 - 고객 주문내역확인

백엔드 코드 작성

```
app.get("/order_list") # 사용자가 구매내역을 확인하는 코드
async def order_list(cid: str):
    try:
        conn = connect()
        curs = conn.cursor()
        sql = """
            SELECT o.oid, o.opid, o.ocount, o.odate, o.ostatus,
                   p.pname, p.pprice, p.pcolor, p.psize,
                   e.ename
            FROM orders o
            JOIN product p ON o.opid = p.pid
            LEFT JOIN employee e ON o.oeid = e.eid
            WHERE o.ocid = %s AND o.ostatus = '결제완료'
            ORDER BY o.odate DESC
        """
        curs.execute(sql, (cid,))
        rows = curs.fetchall()
        conn.close()

        result = [
            {
                "oid": row[0],
                "pid": row[1],
                "count": row[2],
                "odate": row[3],
                "ostatus": row[4],
                "pname": row[5],
                "pprice": row[6],
                "pcolor": row[7],
                "psize": row[8],
                "ename": row[9],
                "image_url": f"http://127.0.0.1:8000/view/{row[1]}"
            } for row in rows
        ]
        return {"result": result}
    except Exception as e:
        print("Error:", e)
        return {"result": "error"}
```

FastAPI docs에서 입력테스트



The screenshot shows the FastAPI documentation for the `/order_list` endpoint. It includes a table for parameters and a table for responses.

Parameters

Name	Description
cid * required	string (query)

Responses

Code	Description
200	Successful Response
422	Validation Error

DB 데이터 확인

ocount	odate	ostatus
2	2025-05-19	결제완료
1	2025-05-19	결제완료
3	2025-05-19	결제완료
3	2025-05-19	결제완료
3	2025-05-19	결제완료
10	2025-05-19	결제완료
1	2025-05-19	결제완료
25	2025-05-19	결제완료
19	2025-05-19	결제완료
25	2025-05-19	결제완료
25	2025-05-19	결제완료
27	NULL	장바구니
27	2025-05-19	결제완료
1	NULL	장바구니

프론트 작성



프로젝트 수행 결과 - 대리점화면

백엔드 코드구성 및 DB 관리.

```
127.0.0.1:8001/list
pretty print 적용 ✕

{
  "results": [
    {
      "oid": 1,
      "ocid": "1",
      "opid": "prd012",
      "oid": "1",
      "ocount": 2,
      "odate": "2025-05-18",
      "ostatus": "반품요청",
      "oreason": "2",
      "oreturndate": "2025-05-19",
      "oreturnstatus": "반품",
      "oreason": "2",
      "pprice": 150000,
      "psize": "L",
      "pname": "조던",
      "odefectivereason": "2"
    },
    {
      "oid": 1,
      "ocid": "1",
      "opid": "prd012",
      "oid": "1",
      "ocount": 2,
      "odate": "2025-05-18",
      "ostatus": "반품요청",
      "oreason": "2",
      "oreturndate": "2025-05-19",
      "oreturnstatus": "반품",
      "oreason": "2",
      "pprice": 150000,
      "psize": "L",
      "pname": "조던",
      "odefectivereason": "2"
    },
    {
      "oid": 2,
      "ocid": "2",
      "opid": "prd012",
      "oid": "2",
      "ocount": 1,
      "odate": "2025-05-19",
      "ostatus": "반품요청",
      "oreason": "2",
      "oreturndate": "2025-05-19",
      "oreturnstatus": "반품",
      "oreason": "2",
      "pprice": 150000,
      "psize": "L",
      "pname": "조던",
      "odefectivereason": "2"
    }
  ]
}

@app.post("/update_return") # 딜러 리턴 페이지에 쓰는 함수
async def update_return(oid: int = Form(...), oreturncount: int = Form(...), oreason: str = Form("반품")):
    try:
        conn = connect()
        curs = conn.cursor()
        sql = "UPDATE orders SET oreturncount = %s, oreason = %s, oreturnstatus = %s, odate = %s WHERE oid = %s"
        curs.execute(sql, (oreturncount, oreason, oreturnstatus, odate, oid))
        conn.commit()
        conn.close()
        return {"result": "OK"}
    except Exception as e:
        print("Error:", e)
        return {"result": "ERROR"}
```

```
@app.get('/list') # 딜러메인에 화면 구성
async def d_orders():
    try:
        conn = connect()
        curs = conn.cursor()
        curs.execute("SELECT oid, ocid, opid, oeid, ocount, odate, ostatus, oreturncount, oreturnstatus, oreason FROM orders")
        rows = curs.fetchall()
        conn.close()
        return {"results": [{"oid": d[0], "ocid": d[1], "opid": d[2], "oeid": d[3], "ocount": d[4], "odate": d[5], "ostatus": d[6], "oreason": d[7], "oreturncount": d[8], "oreturnstatus": d[9]} for d in rows]}
    except Exception as e:
        print("Error:", e)
        return {"result": f"Error: {str(e)}"}
```

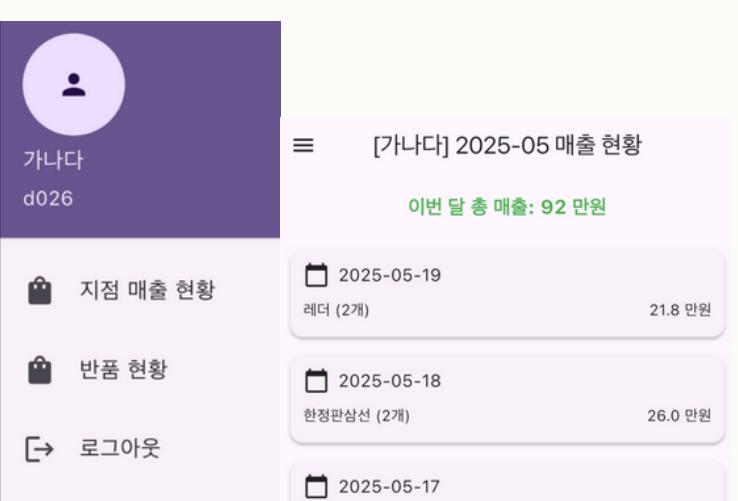
```
@app.get('/dreturns') # 딜러 리턴에 화면구성.
async def d_returns():
```

프론트엔드 구성 및 DB 연결 작업

```
body: Column(
  mainAxisAlignment: MainAxisAlignment.start,
  children: [
    const SizedBox(height: 16),
    Center(
      child: Text(
        "이번 달 총 매출: ${totalsales ~/ 10000} 만원",
        style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold, color: Colors.purple),
      ), // Text
    ), // Center
    const SizedBox(height: 16),
    Expanded(
      child: groupedData.isEmpty
        ? const Center(child: Text('이번 달 매출이 없습니다.'))
        : ListView.builder(
            itemCount: groupedData.length,
            itemBuilder: (context, index) {
              final day = groupedData[index];
              return Card(
                margin: const EdgeInsets.symmetric(horizontal: 16, vertical: 8),
                elevation: 2,
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.start,
                  children: [
                    const SizedBox(height: 16),
                    Text(day['date']),
                    Text("매출: ${day['sales']}"),
                    const SizedBox(height: 16),
                  ],
                ),
              );
            },
          ),
    ),
  ],
);
```

ID	상품ID	제조사	상품명	사이즈	색상	재고수	판매가
14	prd014	丕마	레더	250	silver	40	109000
13	prd013	아디다스	한정판...	260	red	30	130000
12	prd012	나이키	조던	260	red	100	150000

프론트엔드 디자인



반품 요청 관리

- 상품: 조던 / 3개
주문일: 2025-05-10 W450000
- 상품: 조던 / 2개
주문일: 2025-05-26 W300000
- 상품: 레더 / 2개
주문일: 2025-05-19 W218000
- 상품: 한정판삼선 / 2개
주문일: 2025-05-18 W260000
- 상품: 조던 / 3개
주문일: 2025-05-17 W450000

프로젝트 수행 결과 - Fast API

Fast API 메인 화면

FastAPI 0.1.0 OAS 3.1
[/openapi.json](#)

default

- POST** /login Login
- GET** /check_customer_id Check Customer Id
- POST** /insert_customer Insert Customer
- GET** /customer_info Customer Info
- POST** /update_customer Update Customer
- GET** /employee_list Employee List
- POST** /add_to_cart Add To Cart
- GET** /cart_items Cart Items
- DELETE** /delete_cart_item/{oid} Delete Cart Item
- POST** /buy_direct Buy Direct
- POST** /buy_selected Buy Selected
- GET** /order_list Order List
- GET** /returns Get Returns
- GET** /product_list Product List
- GET** /view/prd01 View

Fast API 세부 화면

GET /product_list Product List

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://192.168.50.236:8000/product_list' \
-H 'accept: application/json'
```

Request URL

http://192.168.50.236:8000/product_list

Server response

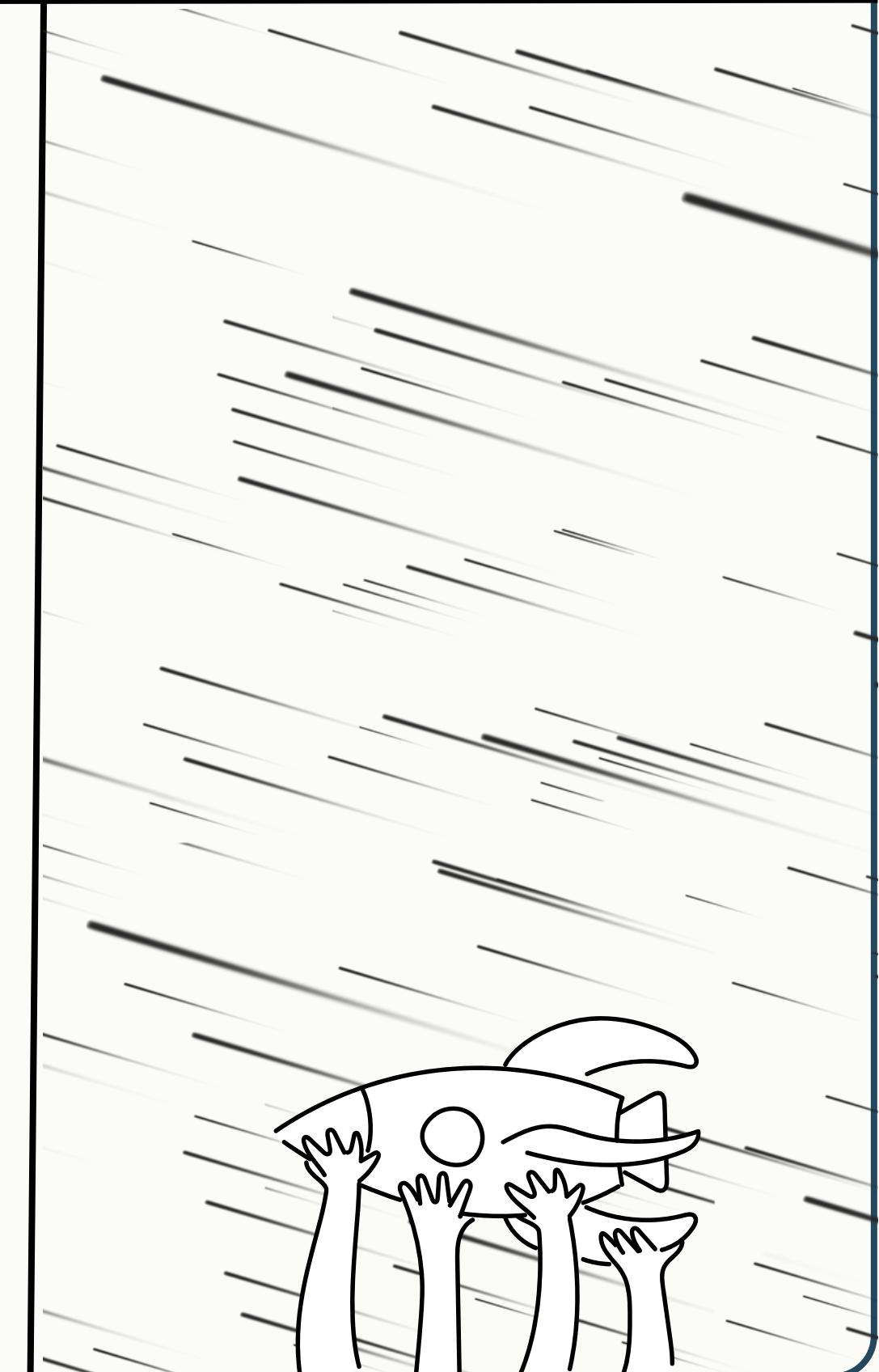
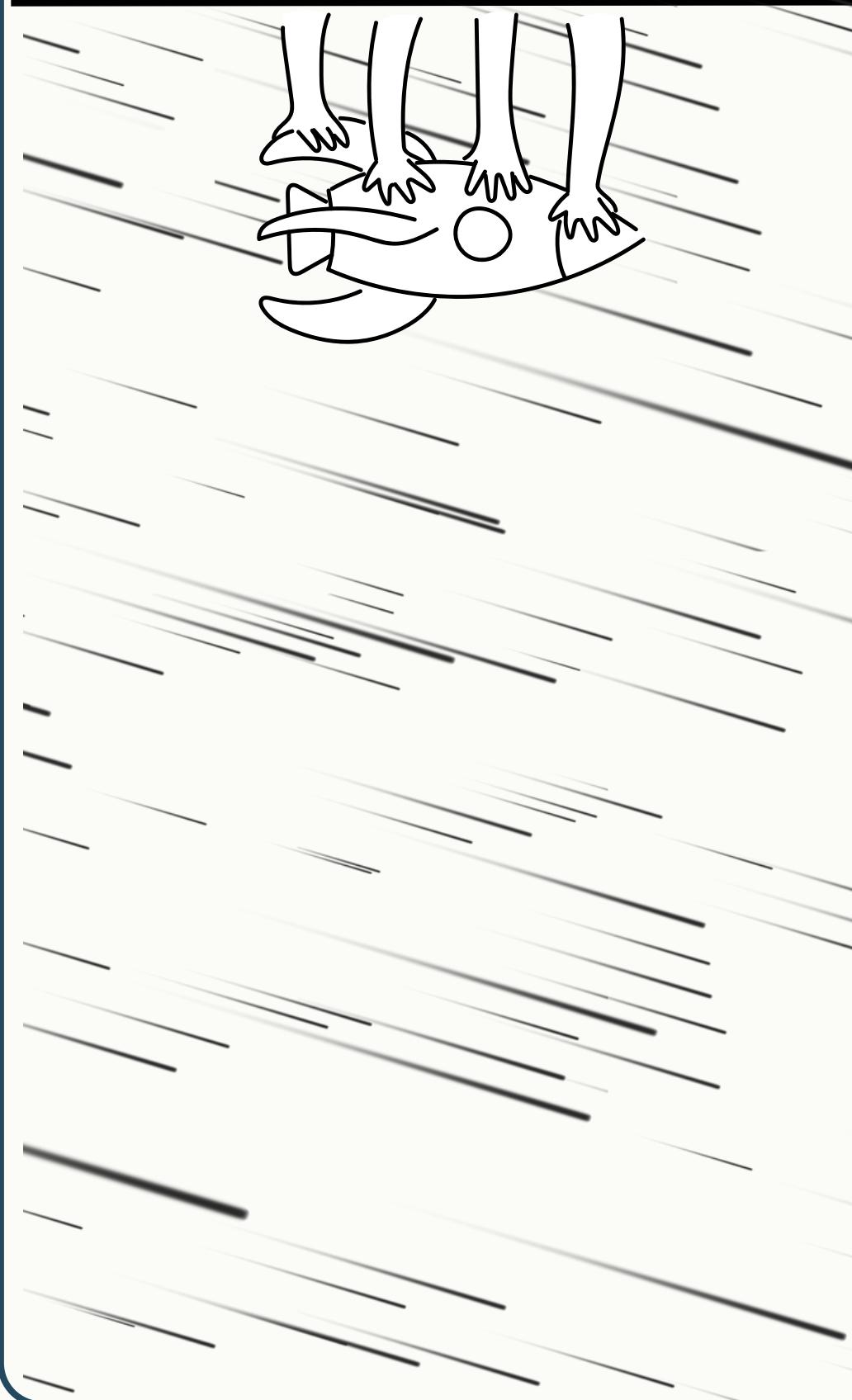
Code Details

200 Response body

```
{
  "result": [
    {
      "pid": "prd012",
      "pname": "327",
      "pprice": 139000,
      "pcolor": "화이트",
      "psize": 250,
      "pbrand": "뉴발란스",
      "pstock": 15,
      "image_url": "http://192.168.50.236:8000/view/prd012"
    },
    {
      "pid": "prd013",
      "pname": "327",
      "pprice": 139000,
      "pcolor": "화이트",
      "psize": 260,
      "pbrand": "뉴발란스",
      "pstock": 50,
      "image_url": "http://192.168.50.236:8000/view/prd013"
    },
    {
      "pid": "prd014",
      "pname": "327",
      "pprice": 139000,
      "pcolor": "화이트",
      "psize": 260,
      "pbrand": "뉴발란스",
      "pstock": 50,
      "image_url": "http://192.168.50.236:8000/view/prd014"
    }
  ]
}
```

Download

프로젝트 수행 결과물



자체 평가 의견

개인 의견

전종익

프론트에서의 거의 모든 기능구현을
백엔드 코드로 대체하려 하다보니
백엔드 코드가 엄청나게
길어지는 문제가 아쉬웠다

이권형

기존에 sqlite로 작성되어 있던 코드를
MySQL로 재구조화 하는 작업이
생각보다 많이 어려웠다. 리모델링이
아니라 새로 건축할 정도로 많은 수정이
필요했고 시간이 촉박하여 여유가 많지
않아 디자인적인 부분까지 신경쓰지
못한 점이 아쉬웠다

이학현

코딩 속도가 느려서
다른 팀원분들에 비해
기여도가 낮은 게 아쉬웠다

하지만 이번 프로젝트를
수행하면서 에러를 해결할 때
성취감이 느껴져
꽤나 재밌었다

전감성

들어간 시간대비작업물이
다른 팀원들보다 미흡하다고 생각.
코딩작업도중 필요한게많아
초기설계에대한 중요성을알게됨.
직접 서버를 만들고
관리하는경험에 흥미가 생김

공통 의견

이번 프로젝트가 TEAM4의 마지막 프로젝트입니다.
다들 고생 많으셨습니다.
다음 조에서도 모두 건승하세요.

THANK
YOU