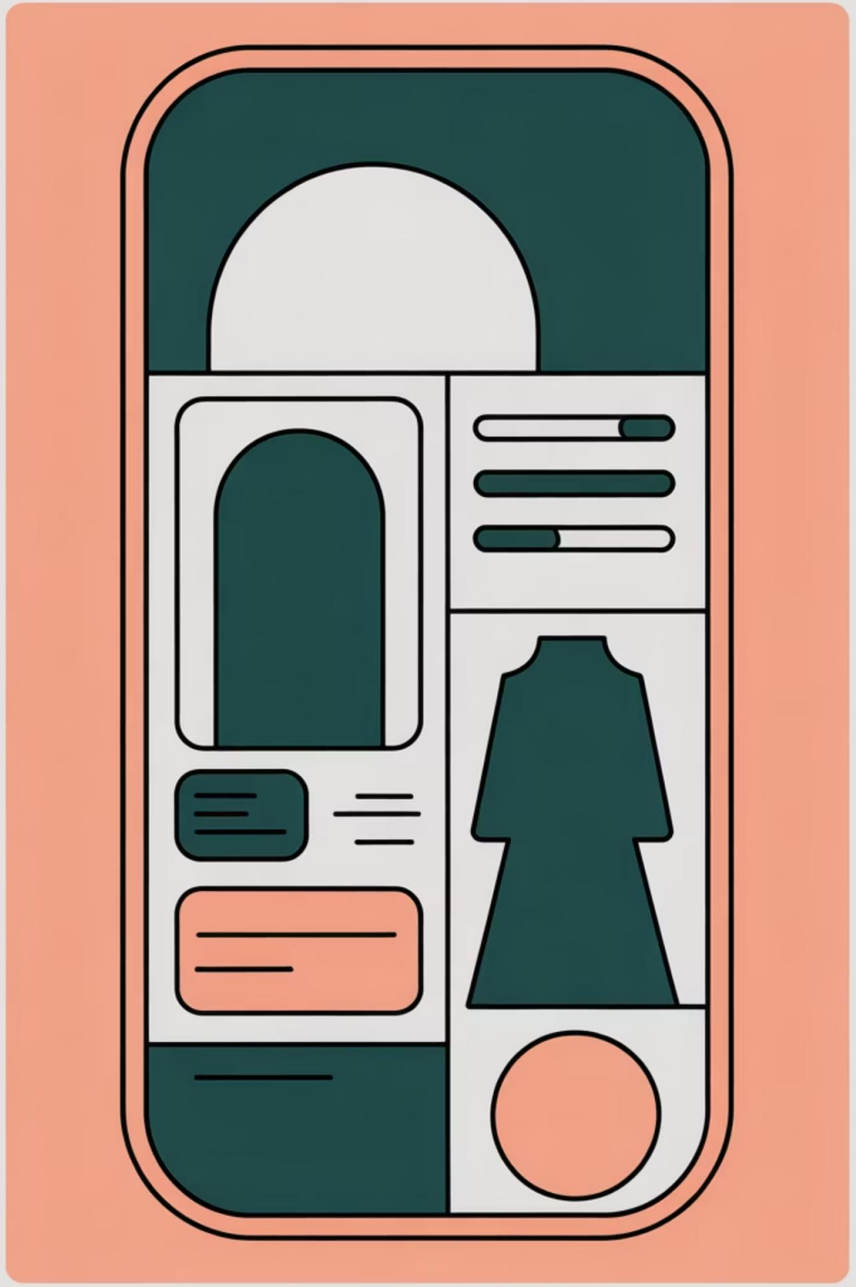


2025-10-20

K-Fashion Demon Hunters

AI 기반 패션 추천 서비스

STC팀 | 배정윤, 이상진, 김준희, 이상엽
IBM x Redhat AI Transformation 3rd

발표 주요 내용

01 프로젝트 배경 및 소개

02 MVP 아키텍처

03 MVP 개발 계획

04 Backend 개발 현황

05 Frontend 개발 현황

06 컨테이너화 및 배포

07 향후 계획

08 Q&A

Target Persona

패션에 관심 많지만 정보 탐색이 번거로운 K직장인 '지민'

분산된 정보, 과도한 정보량

트렌드 - SNS, 가격 비교-다나와, 구매-쇼핑몰
: 방대한 상품과 콘텐츠 그리고 플랫폼

부정확한 이미지 검색

단일 이미지모델이 가진 정밀성의 한계
: 사용자의 의도 불충분

추천의 일반화

다소 일반화된 추천 알고리즘 : 사용자의 세부 취향과 상황 반영 부족

복잡한 사용자 경험

사이트마다 다른 검색 방법 : 비효율적 접근

AI 기반의 통합 패션 플랫폼이 필요하다는 확신

K-Fashion Demon Hunters Project

사용자 취향 분석

이미지 및 텍스트 기반의
정교한 스타일 분석

AI 맞춤 추천

LLM 대화형 스타일링
개인화된 추천 제공

실시간 최저가 비교

여러 쇼핑몰의 가격을
한눈에 비교

단순한 검색 도구를 넘는,

사용자와 대화하며 스타일을 제안하는 AI 패션 어시스턴트

기존 서비스

- 단순 키워드 검색
- 구매 이력 기반 추천
- 수동 가격 비교
- 일방향 정보 제공

K-Fashion Demon Hunters

- AI 이미지 분석 + 자연어 처리
- 대화형 스타일링 추천
- 실시간 자동 최저가 비교
- 양방향 맞춤형 큐레이션

수익 모델 (검토 중)

제휴 커미션

쇼핑몰 파트너십을 통한 구매 연결 수수료

프리미엄 구독

고급 스타일링 분석 및 전문가 추천 서비스
추가 고급 서비스

브랜드 협업

브랜드 대상 타겟 마케팅 및
데이터 인사이트 제공

광고 플랫폼

맞춤형 패션 광고 및 프로모션 지면 제공

단계별 기능 확장 계획

Step 1 :

유사도 비교를 통한 추천

- LLM을 활용한
  자연어 기반 질의 처리
- Vision API 활용,
  이미지 분석 및
  유사도 정확도 확보

Step 2:

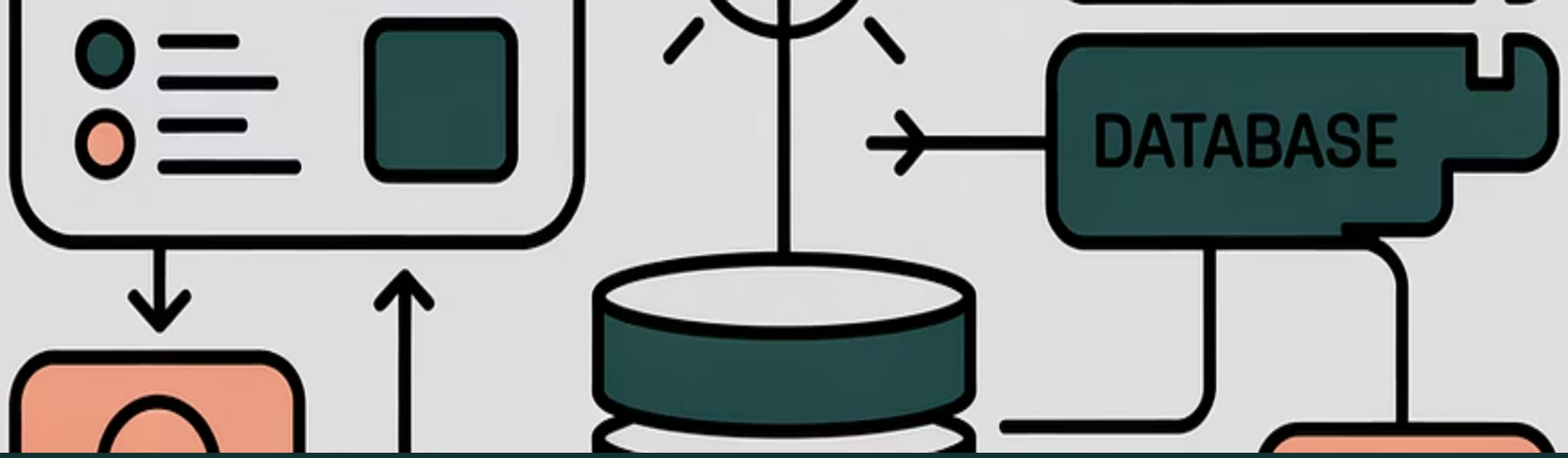
Vision + LLM

- 이미지 + 텍스트 임베딩
- 비슷한 의류 인식

Step 3:

생성형 비전 모델로 확장

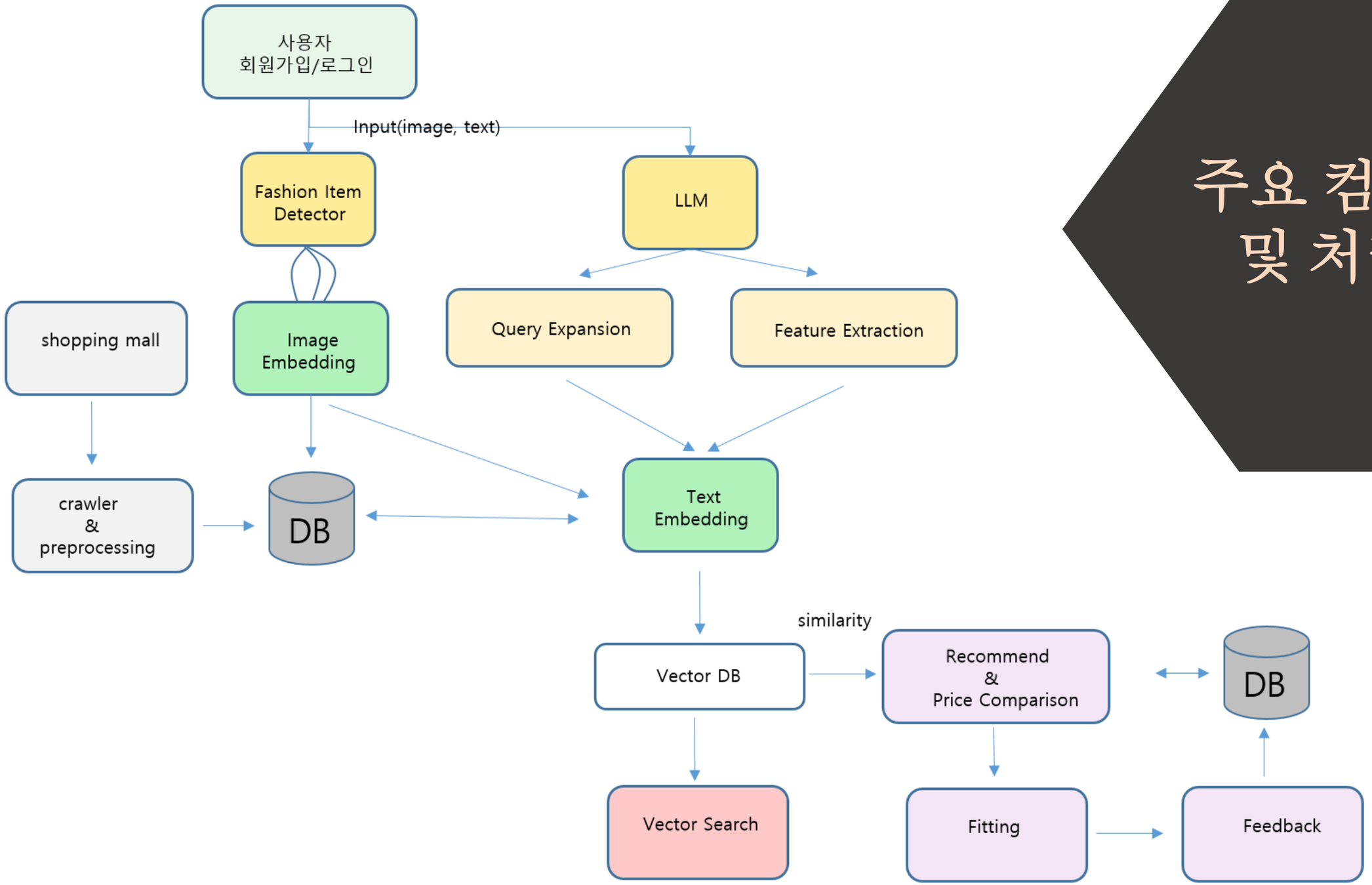
- 이미지 업로드 후
  대화형으로 스타일 추천
- 선택 의류의 가상피팅 구현



MVP

아 키 텍 처

주요 컴포넌트 및 처리 과정



주요 컴포넌트 및 기술 스택

Backend

FastAPI 구축, 고성능 REST API 서버

- 이미지 업로드 및 분석
- LLM 통합 및 데이터 처리 로직

AI 분석 및 추천 모듈

Google Vision API 및 CLIP Embedding

- 이미지 특징 추출 및 유사도 분석
- Vector DB 연동

크롤러

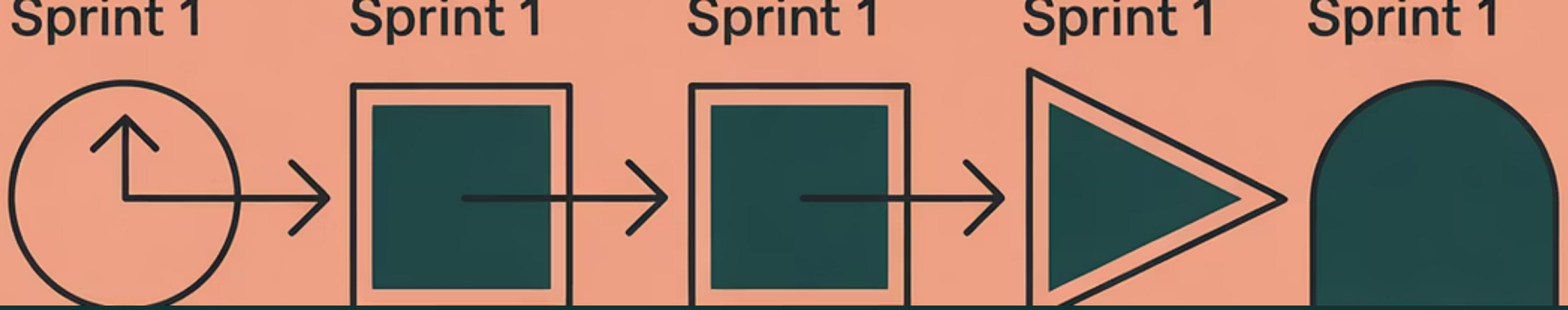
멀티 소스 데이터 수집 엔진

- 무신사, 인스타그램 크롤링
- 실시간 가격 정보

Frontend UI

React

- 반응형 디자인
- 직관적인 UI



5주 개발 타임라인

1주차: 기반 구축

개발 환경 구축, API 키 발급, 기본 백엔드 구현, 크롤러 개발 시행

2주차: 핵심 기능

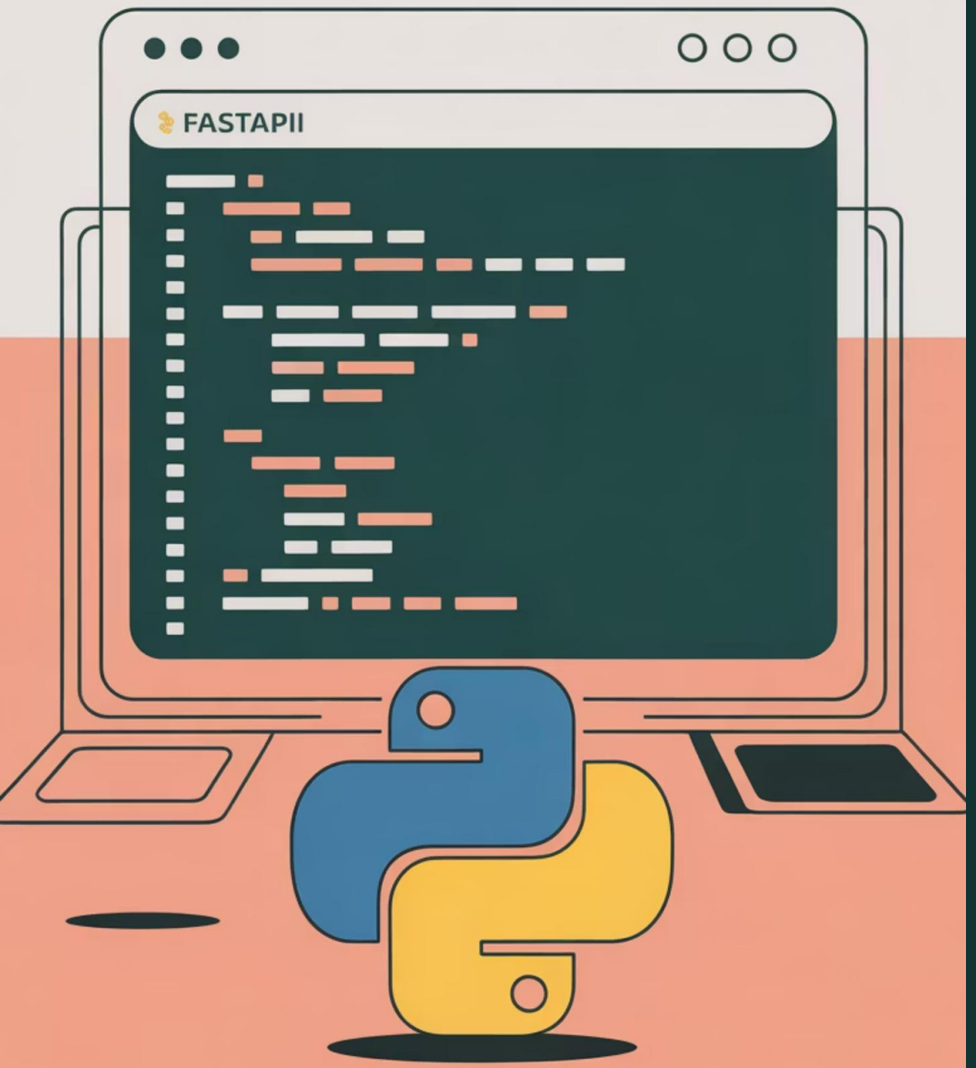
LLM 통합, DB 설계, 프론트엔드 구현, 추천 알고리즘 고도화

3주차: 통합 테스트

Docker 컨테이너화, AWS 배포 시작, 전체 시스템 통합 테스트

4-5주차: 완성

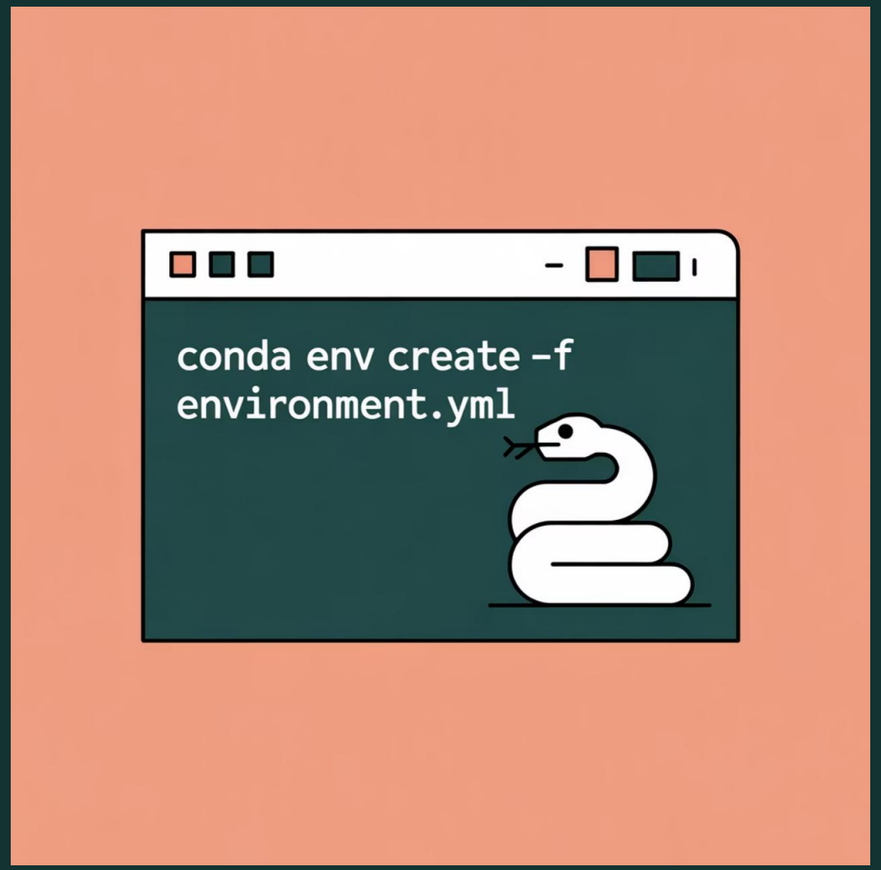
기능 고도화, 문서화 및 발표 준비



개발 진행 상황
Backend

Backend 개발 환경 구축

Conda, Git, FastAPI 기반 인프라



패키지 관리 및 가상환경

Conda를 활용하여 Python 패키지 의존성을 체계적으로 관리하고, 프로젝트별 독립된 개발 환경을 구성

- Python 3.10 환경 설정 및 Fast API 설치
- AI 라이브러리 (Transformers, CLIP) 통합
- 크롤링 도구 (BeautifulSoup, Selenium) 구성

버전 관리

Git으로 코드 변경 이력을 추적하고, 브랜치 전략을 통해 안정적인 협업 환경 구축

핵심 API 엔드포인트

POST /api/upload
이미지 파일 업로드 및 임시 저장 처리

POST /api/analyze
Vision API를 통한 이미지 특징 분석 및 벡터 추출

GET /api/recommend
유사도 기반 패션 아이템 추천 결과 반환

POST /api/chat
LLM 기반 대화형 스타일링 추천 생성

GET /api/price-compare
크롤링된 데이터 기반 최저가 비교 정보 제공

FastAPI 기본 구조

RESTful API 엔드포인트 설계

라우터 구조
모듈화된 라우터로 API 엔드포인트를 분리하여 유지보수성을 높임

서비스 레이어
비즈니스 로직을 서비스 모듈로 분리하여 코드 재사용성과 테스트 용이성을 확보

모델 정의
Pydantic을 활용한 데이터 검증 및 직렬화로 API 안정성을 보장합니다.

Vision API : Detection

Google Cloud Vision을 활용한 이미지 분석

구현 기능

- 의류 아이템 객체 감지
- 색상 팔레트 추출
- 패턴 및 텍스처 인식
- 브랜드 로고 탐지
- 이미지 품질 평가

처리 과정

1. 이미지 전처리 및 크기 최적화
2. Vision API 호출 및 응답 수신
3. 특징 벡터 추출 및 정규화
4. FAISS Vector DB에 저장
5. 유사도 검색 인덱스 구축

현재 무료 크레딧 확보 가능성을 검토 중이며, 대안으로 Yolo-fashion 모델과의 성능 비교를 진행하고 있습니다.

```python
class VisionAnalyzer:
    def __init__(self):
        credentials_path = os.getenv('GOOGLE_APPLICATION_CREDENTIALS')
        if credentials_path:
            os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = credentials_path
        self.client = vision.ImageAnnotatorClient()

    def analyze_fashion_image(self, image_path: str) -> Dict:
        """이미지에서 패션 아이템 감지"""
        try:
            with open(image_path, 'rb') as image_file:
                content = image_file.read()

            image = vision.Image(content=content)

            # 객체 감지
            objects = self.client.object_localization(image=image).localized_object_annotations

            # 색상 추출
            colors = self.client.image_properties(image=image).image_properties_annotation.dominant_colors.colors

            # 라벨 감지
            labels = self.client.label_detection(image=image).label_annotations

            # 의류 아이템 필터링
            fashion_items = []
            for obj in objects:
                fashion_keywords = [
                    'clothing', 'shirt', 'pants', 'dress', 'jacket', 'shoe',
                    'suit', 'tie', 'coat', 'blazer', 'top', 'bottom', 'footwear',
                    'sleeve', 'collar', 'outerwear', 'apparel', 'garment'
                ]
```

Google Vision API 구현 코드

Yolo-fashion 모델 구현

```python
# 1. 객체 탐지 모델 (YOLOS)
def load_detection_model(model_name="valentinafeve/yolos-fashionpedia"):
    print("객체 탐지 모델 불러오는 중...")
    processor = AutoImageProcessor.from_pretrained(model_name)
    model = AutoModelForObjectDetection.from_pretrained(model_name)
    return processor, model


# 2. FashionCLIP 모델 (임베딩용)
def load_embedding_model(model_name="patrickjohncyh/fashion-clip"):
    print("FashionCLIP 임베딩 모델 불러오는 중...")
    processor = AutoProcessor.from_pretrained(model_name)
    model = AutoModelForZeroShotImageClassification.from_pretrained(model_n
    return processor, model


# 3. 이미지 로드
def load_image(image_path):
    print(f"이미지 로드: {image_path}")
    image = Image.open(image_path).convert("RGB")
    return image


# 4. 객체 탐지
def detect_objects(image, processor, model, threshold=0.5):
    print("객체 탐지 중...")
    inp
    wi
        outputs = model(**inputs)

    target_sizes = torch.tensor([image.size[::-1]])  # (height, width)
```

```
탐지된 객체들:
shoe: 0.942 | 박스: [217.33212280273438, 548.708251953125, 295.17288208
00781, 590.6530151367188]
pocket: 0.692 | 박스: [220.04920959472656, 263.9134521484375, 248.05813
598632812, 296.4765625]
shoe: 0.975 | 박스: [313.1114196777344, 567.0408935546875, 375.64489746
09375, 606.5457153320312]
collar: 0.675 | 박스: [274.0464782714844, 0.4754805564880371, 351.42044
06738281, 18.65863037109375]
collar: 0.934 | 박스: [259.9573059082031, -0.11159539222717285, 334.823
2421875, 30.98740005493164]
collar: 0.86 | 박스: [262.7142333984375, 7.045397758483887, 336.6693725
5859375, 62.75390625]
                                                          427.7333
                                                          0.0307006
shirt, blouse: 0.524 | 박스: [214.70172119140625, 9.7613525390625, 384.
65374755859375, 291.5111083984375]
sleeve: 0.975 | 박스: [189.73037719726562, 48.462188720703125, 255.6159
```

현재 Google Vision API 무료 크레딧 확보 가능성을 검토 중이며, 대안으로 Yolo-fashion 모델과의 성능 비교를 진행하고 있습니다.

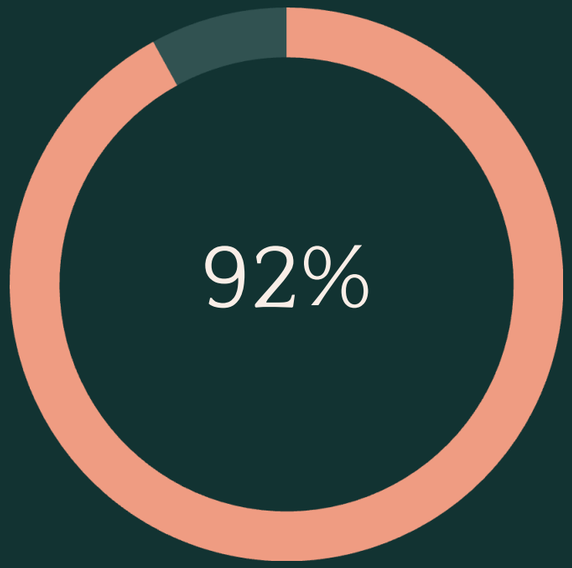
Yolo-fashion 모델 구현

```python
# 6. 결과 시각화
def visualize_results(image, results, model, save_path="output_with_boxes.png"):
    print(f"탐지 결과 시각화 → {save_path}")
    plt.figure(figsize=(10, 10))
    plt.imshow(image)
    ax = plt.gca()

    for score, label, box in zip(results["scores"], results["labels"], results["boxes"]):
        xmin, ymin, xmax, ymax = box.tolist()
        width, height = xmax - xmin, ymax - ymin
        ax.add_patch(
            patches.Rectangle(
                (xmin, ymin), width, height, linewidth=2, edgecolor="red", fa
            )
        )
        label_name = model.config.id2label[label.item()]
        ax.text(
            xmin,
            ymin,
            f"{label_name}: {round(score.item(), 2)}",
            bbox=dict(facecolor="yellow", alpha=0.5),
            fontsize=10,
            color="black",
        )

    plt
    plt.title("Fashion Object Detection")
    plt.savefig(save_path)
```

현재 Google Vision API 무료 크레딧 확보 가능성을 검토 중이며, 대안으로 Yolo-fashion 모델과의 성능 비교를 진행하고 있습니다.

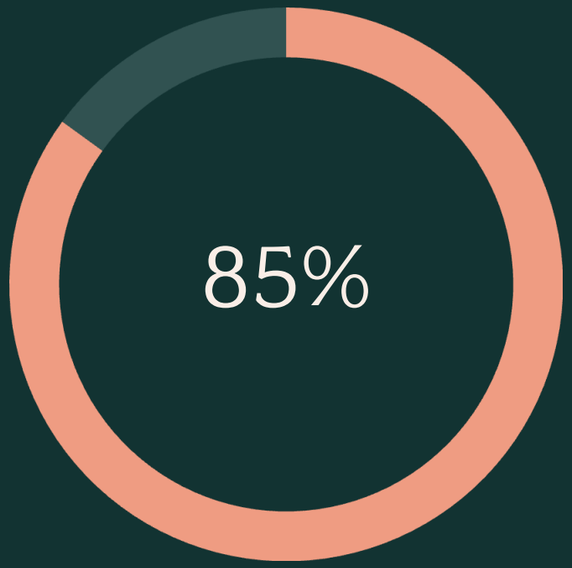
Yolo-fashion 분석 현황

Yolo-fashion 모델은 이미지에서 의류 아이템을 정확히 감지하고 카테고리를 분류
상의, 하의, 원피스, 아우터 등을 구분하며, 각 아이템의 위치와 크기 정보도 제공



의류 감지 정확도

테스트 데이터셋 기준 F1 스코어



카테고리 분류 정확도

15개 의류 카테고리 분류 성능

평균 처리 시간

GPU 환경에서 이미지 1장 분석 소요 시간

Google Vision vs Yolo-fashion 비교

이미지 분석 모델 성능 평가

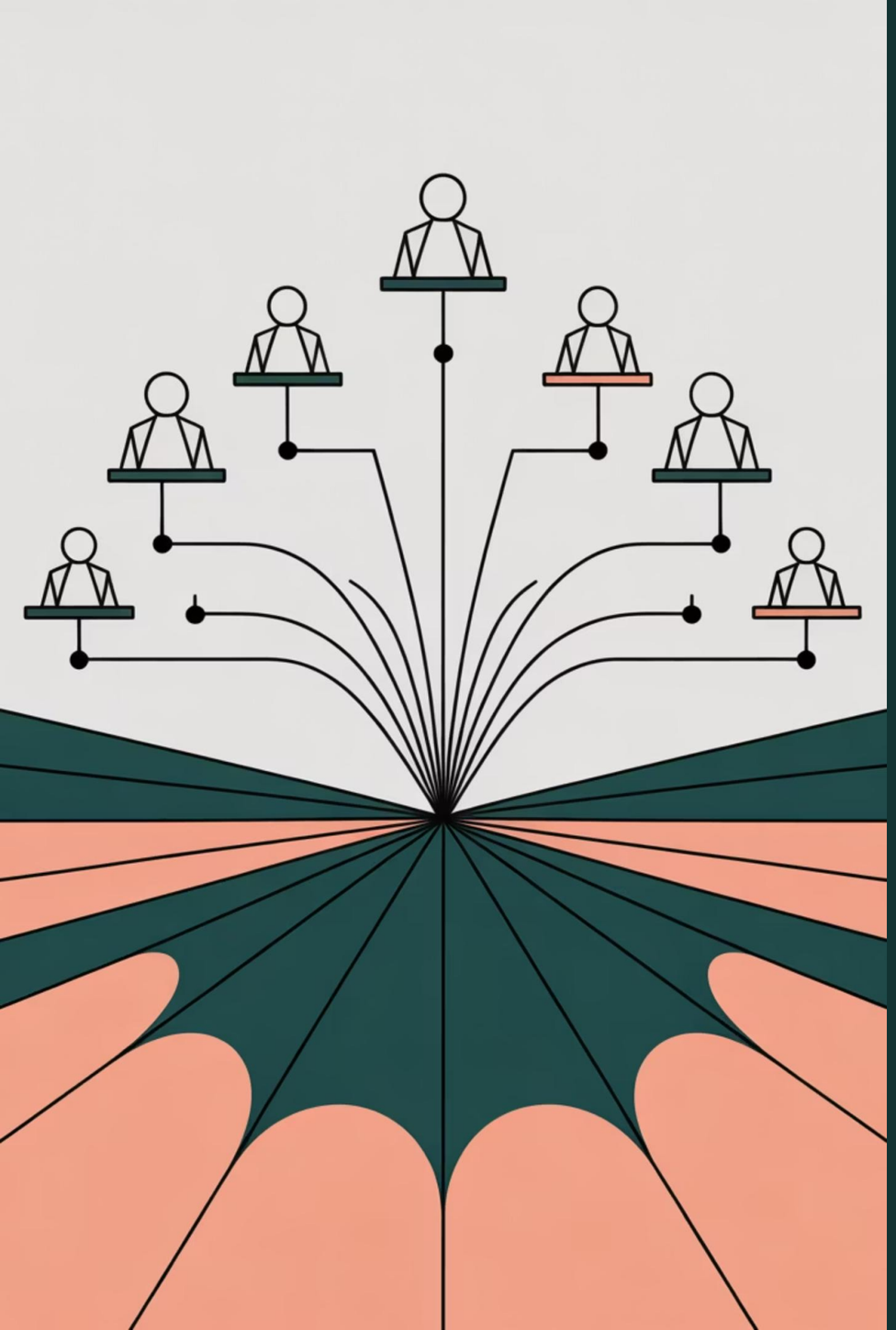
Google Vision API

- **장점**
  - 높은 범용성과 정확도
  - 다양한 객체 감지 가능
  - 색상 및 텍스처 분석 우수
  - 간편한 API 호출
- **단점**
  - 사용량에 따른 비용 발생
  - 커스터마이징 어려움

Yolo-fashion

- **장점**
  - 패션 도메인 특화 모델
  - 의류 아이템 세분화 인식
  - 무료 오픈소스
- **단점**
  - GPU 리소스 필요
  - 모델 튜닝 및 유지보수 필요

현재 두 모델의 **정확도, 처리 속도, 비용 효율성**을 종합적으로 비교 평가 중입니다.



최종 모델 선정 방향

하이브리드 최적화

분석 테스트 및 토큰 제한 발생 또는 인터넷 불가 상황에서
Google Vision 운용 불가의 경우 등을 대비하여,
Yolo-fashion으로 보완하는 전략을 검토 중

성능 모니터링

A/B 테스트를 통해 두 모델의 실제 사용자 만족도와
추천 정확도를 지속적으로 비교 평가

취향 분석 및 추천

LLM : 대화형 스타일링

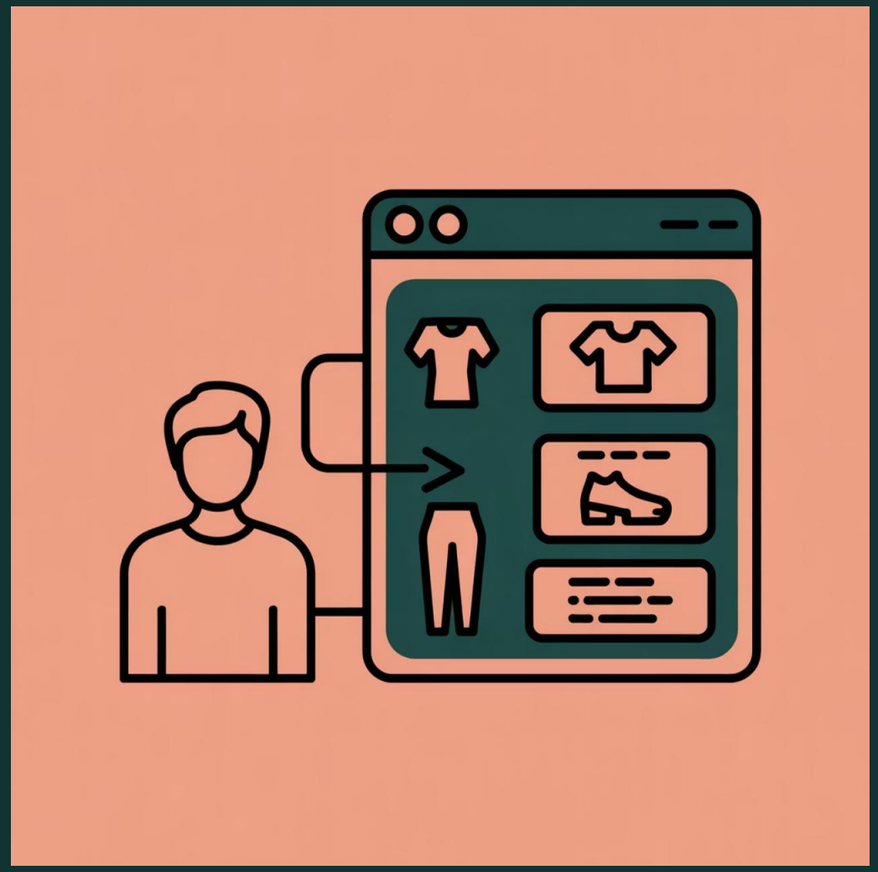
자연어 처리 기반 맞춤형 추천

주요 기능

- **사용자 취향 분석 및 페르소나 생성**
- **트렌드 정보 제공**
- 상황별 코디 제안 (출근, 데이트, 캐주얼)
- 스타일링 팁 및 조합 추천
- 다국어 지원 (한국어, 영어)

API 연동

OpenAI GPT-4 및 Claude API를 사용하여
고품질의 자연어 생성을 구현



※ 현재 구현 고도화 중

크롤링 데이터 활용 방안

상품 데이터베이스 구축

수집한 상품 정보 정규화, 통합 데이터베이스를 구축

- 상품명, 브랜드, 카테고리
- 가격 정보 및 할인율
- 이미지 URL 및 상세 설명
- 판매처 링크 및 재고 상태

트렌드 분석

인기 상품과 급상승 키워드를 분석, 트렌드 인사이트 제공

- 주간/월간 베스트 아이템
- 색상 및 스타일 트렌드
- 브랜드별 인기도 분석
- 시즌별 패션 변화 추이

최저가 비교 서비스

동일 상품의 여러 판매처 가격을 실시간으로 비교하여 사용자에게 제공

추천 알고리즘 학습

크롤링 데이터를 AI 모델 학습에 활용, 추천 정확도 고도화

크롤링 기술 스택 및 전략

기술 도구

- **BeautifulSoup**: HTML 파싱 및 데이터 추출
- **Selenium**: 동적 페이지 렌더링 및 자동화
- **Scrapy**: 대규모 크롤링 프레임워크

데이터 처리

- 중복 제거 및 데이터 정규화
- 이미지 다운로드 및 압축
- 가격 정보 파싱 및 통화 표준화
- PostgreSQL 데이터베이스 저장

크롤링 전략

- **주기적 수집**: Celery Beat로 스케줄링
- **Rate Limiting**: 서버 부하 방지
- **User-Agent 로테이션**: 차단 회피
- **프록시 활용**: IP 분산

법적 고려사항

- API 우선 사용 원칙 준수
- 공개된 정보만 수집
- 저작권 및 개인정보 보호법 준수
- GDPR 준수

멀티 데이터 수집

크롤링 대상

무신사

국내 최대 패션 플랫폼,
트렌디한 브랜드와 스타일
정보가 풍부,
주기적인 신상품 수집 가능

INSTAGRAM

인플루언서와 브랜드 계정
에서 최신 패션 트렌드와 스
타일링 정보 수집 가능

NAVER

국내 최대 포털 플랫폼.
다양한 쇼핑몰과 브랜드의
상품 정보 수집 가능.
가격 비교 가능.

크롤링 구현

무신사

musinsa.py | mask_rcnn_windows.py C:\...\Desktop | mask_rcnn_windows.py tf114 | mask_rcnn_windows_face.py tf114

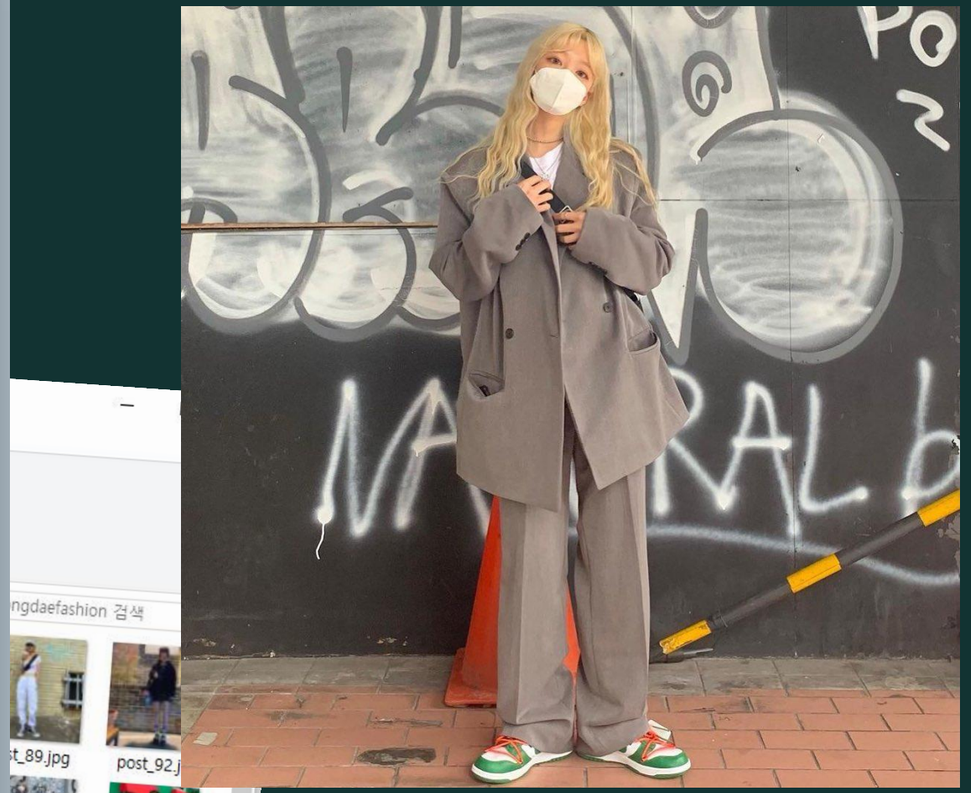
tf114 > musinsa.py > ...

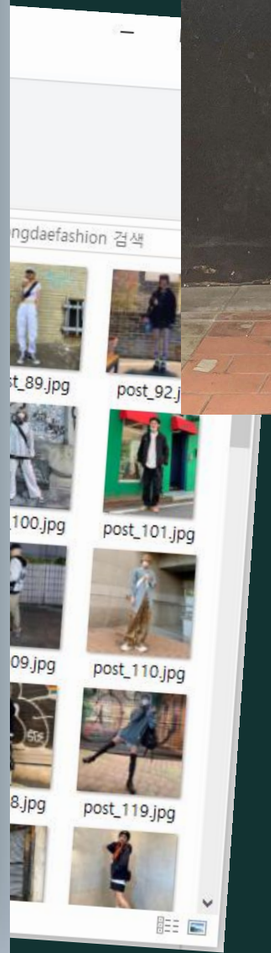
```python
time.sleep(3)

# ==================== 스크롤 + 상품 수집 ====================
SCROLL_STEP = 500           # 한 번에 스크롤할 픽셀
SCROLL_PAUSE_TIME = 1.5     # 스크롤 후 대기 시간
collected_ids = set()
data_list = []

last_scroll = 0
while True:
    # 화면에 보이는 모든 상품 요소
    items = driver.find_elements(By.CSS_SELECTOR, 'div[data-item-id]')

    for item in items:
        try:
            item_id = item.get_attribute("data-item-id")
            if item_id in collected_ids:
                continue
            collected_ids.add(item_id)

            # StaleElement 대비 재시도
            for _ in range(3):
                try:
                    rank = item.get_attribute("data-item-list-index")
                    brand = item.get_attribute("data-item-brand")
                    name = item.find_element(By.CSS_SELECTOR, 'p[class*="line-clamp-2"]').text
                    price = item.get_attribute("data-price")
                    img_url = item.find_element(By.CSS_SELECTOR, 'img').get_attribute("src")
                    break
                except:
                    time.sleep(0.5)

            # 이미지 저장
            img_filename = f"{rank}_{name.replace('/', '_').replace(' ', '_')}.jpg"
            img_path = os.path.join(IMAGE_DIR, img_filename)
            try:
                img_data = requests.get(img_url).content
```

musinsa_ranking.csv > data

```
rank,brand,name,price,img_path
109,bittercells,오버사이즈드 샤샤 스트라이프 셔츠-그레이,55250,musinsa_images\109_오버사이즈드_샤샤_스트라이프_셔츠-그레이.jpg
110,teket,Noise Zip-Up Sweatshirt Khaki,109000,musinsa_images\110_Noise_Zip-Up_Sweatshirt_Khaki.jpg
111,massimodutti2,루즈핏 100% 코튼 포플린 셔츠,109000,musinsa_images\111_루즈핏_100%_코튼_포플린_셔츠.jpg
112,spao,플란넬 오버핏 체크 셔츠_SPYCF4TC54,35910,musinsa_images\112_플란넬_오버핏_체크_셔츠_SPYCF4TC54.jpg
113,fabregat,[SEEMCITY] 에테르 체크 셔츠 (블루),53900,musinsa_images\113_[SEEMCITY]_에테르_체크_셔츠_(블루).jpg
114,signature,크롭 긴팔 티셔츠[스트라이프],20890,musinsa_images\114_크롭_긴팔_티셔츠[스트라이프].jpg
115,aeae,DOT SMALL LETTER L/S [NAVY],53100,musinsa_images\115_DOT_SMALL_LETTER_L_S_[NAVY].jpg
116,huey,Check Rollup Tee Navy,52200,musinsa_images\116_Check_Rollup_Tee_Navy.jpg
117,aeae,DOT SMALL LETTER L/S [WHITE],53100,musinsa_images\117_DOT_SMALL_LETTER_L_S_[WHITE].jpg
118,descente,코튼 기모 맨투맨 SP323XHT93,76300,musinsa_images\118_코튼_기모_맨투맨_SP323XHT93.jpg
119,ghostrepublic,래글런 트랙 후드배색 헨리넥 후드티 GHT-483_네이비,39900,musinsa_images\119_래글런_트랙_후드배색_헨리넥_후드티_GHT-483_네이비.jpg
120,madden,[7COLORS] 퍼니 풀오버 후디,62290,musinsa_images\120_[7COLORS]_퍼니_풀오버_후디.jpg
121,kashiko,Double Collar Slim Shirts Black,130000,musinsa_images\121_Double_Collar_Slim_Shirts_Black.jpg
122,divein,UNIFORM CROP LONG SLEEVE (MELANGE GRAY),35100,musinsa_images\122_UNIFORM_CROP_LONG_SLEEVE_(MELANGE_GRAY).jpg
123,marithefrancoisgirbaud,(UNISEX / WOMEN) 3PACK SMALL REGULAR LOGO TEE MIX,39200,musinsa_images\123_(UNISEX___WOMEN)_3PACK_SMALL_REGULAR_LOGO_TEE_MIX.jpg
124,topten,남성) 옥스포드 버튼다운 셔츠_MSF5WC1911,19900,musinsa_images\124_남성)_옥스포드_버튼다운_셔츠_MSF5WC1911.jpg
125,noirer,로우 게이지 헤비 크롭 니트 (블랙),99000,musinsa_images\125_로우_게이지_헤비_크롭_니트_(블랙).jpg
126,musinsastandard,헤비웨이트 오버사이즈 데님 셔츠 [다크 블루],48890,musinsa_images\126_헤비웨이트_오버사이즈_데님_셔츠_[다크_블루].jpg
127,thecoldestmoment,TCM mini logo stripe long sleeve (blue/navy),48600,musinsa_images\127_TCM_mini_logo_stripe_long_sleeve_(blue_navy).jpg
128,thecoldestmoment,TCM mini logo stripe long sleeve (yellow/grey),48600,musinsa_images\128_TCM_mini_logo_stripe_long_sleeve_(yellow_grey).jpg
129,leire,에센셜 크롭 티셔츠 [블랙],23900,musinsa_images\129_에센셜_크롭_티셔츠_[블랙].jpg
130,vivalavida,[기모선택] Gleam 헤비테리 후드,48490,musinsa_images\130_[기모선택]_Gleam_헤비테리_후드.jpg
131,ambler,Star 스트라이프 롱슬리브 AJS203(블루),31410,musinsa_images\131_Star_스트라이프_롱슬리브_AJS203(블루).jpg
132,musinsastandardwoman,우먼즈 멀티 스트라이프 칼라드 니트 [그레이],56900,musinsa_images\132_우먼즈_멀티_스트라이프_칼라드_니트_[그레이].jpg
133,lee,부클 트위치 로고 기모 후드 티셔츠 네이비,94050,musinsa_images\133_부클_트위치_로고_기모_후드_티셔츠_네이비.jpg
```

```python
import json, os, requests, mimetypes
from urllib.parse import urlparse

# === 1. JSON 불러오기 ===
path = r"C:\Study25"
json_file = os.path.join(path, "dataset_instagram-post-scraper_2025-09-24_06-13-09-878.json")  # 새로 받은 JSON 경로
with open(json_file, "r", encoding="utf-8") as f:
    data = json.load(f)

# === 2. 게시물 데이터만 추출 ===
# (url 필드가 /p/ 로 시작하는 게시물만)
posts = [item for item in data if "url" in item and "/p/" in item["url"]]

print(f"총 게시물 수: {len(posts)}")

# === 3. 이미지 URL 컬럼명 찾기 ===
# (실제 JSON 구조 확인 필요. 보통 displayUrl, imageUrl, thumbnail_src 중 하나)
possible_keys = ["displayUrl", "display_url", "imageUrl", "thumbnail_src"]

# === 4. 이미지 저장할 폴더 ===
save_dir = os.path.join(path, "downloaded_posts")
os.makedirs(save_dir, exist_ok=True)

# === 5. 다운로드 ===
for i, post in enumerate(posts, start=1):
    url = None
    for key in possible_keys:
        if key in post:
            url = post[key]
            break
    if not url:
        print(f"[스킵] 게시물 {i}: 이미지 URL 없음")
        continue

    try:
        r = requests.get(url, stream=True, timeout=15, headers={"User-Agent":"Mozilla/5.0"})
        if r.status_code == 200:
            mime = r.headers.get("Content-Type", "")
            ext = mimetypes.guess_extension(mime.split(";")[0].strip()) or os.path.splitext(urlparse(url).path)[1] or ".jpg"
            if not ext.startswith("."): ext = "." + ext
            file_path = os.path.join(save_dir, f"post_{i}{ext}")
            with open(file_path, "wb") as out_file:
                for chunk in r.iter_content(8192):
                    if chunk:
                        out_file.write(chunk)
            print(f"[저장] {file_path}")
        else:
            print(f"[실패] {url} (status {r.status_code})")
    except Exception as e:
        print(f"[에러] {url}: {e}")

```





인스타

네이버API

```python
class FashionCrawler:
    def __init__(self):
        self.client_id = os.getenv('NAVER_CLIENT_ID')
        self.client_secret = os.getenv('NAVER_CLIENT_SECRET')

        print(f"=== 네이버 API 키 확인 ===")
        print(f"Client ID: {self.client_id}")
        print(f"Client Secret: {self.client_secret}")
        print(f"========================")

        if not self.client_id or not self.client_secret:
            raise ValueError("네이버 API 키가 설정되지 않았습니다")

    def crawl_musinsa(self, keyword: str, max_items: int = 20) -> List[Dict]:
        """네이버 쇼핑 검색 (무신사 포함 전체 쇼핑몰)"""
        products = []

        url = "https://openapi.naver.com/v1/search/shop.json"
        headers = {
            "X-Naver-Client-Id": self.client_id,
            "X-Naver-Client-Secret": self.client_secret
        }
        params = {
            "query": keyword,
            "display": min(max_items, 100),
            "sort": "sim"  # 정확도순
        }

        try:
            response = requests.get(url, headers=headers, params=params, timeout=5)
            response.raise_for_status()
            data = response.json()

            for item in data.get('items', []):
```

CLIP Embedding 및 Vector DB

의미론적 유사도 기반 검색 시스템

CLIP (Contrastive Language-Image Pre-training) 모델을 활용하여 이미지와 텍스트를 동일한 벡터 공간에 임베딩
"캐주얼한 데님 재킷"과 같은 자연어 질의에 시각적으로 유사한 상품 검색 가능

CLIP 모델 선택

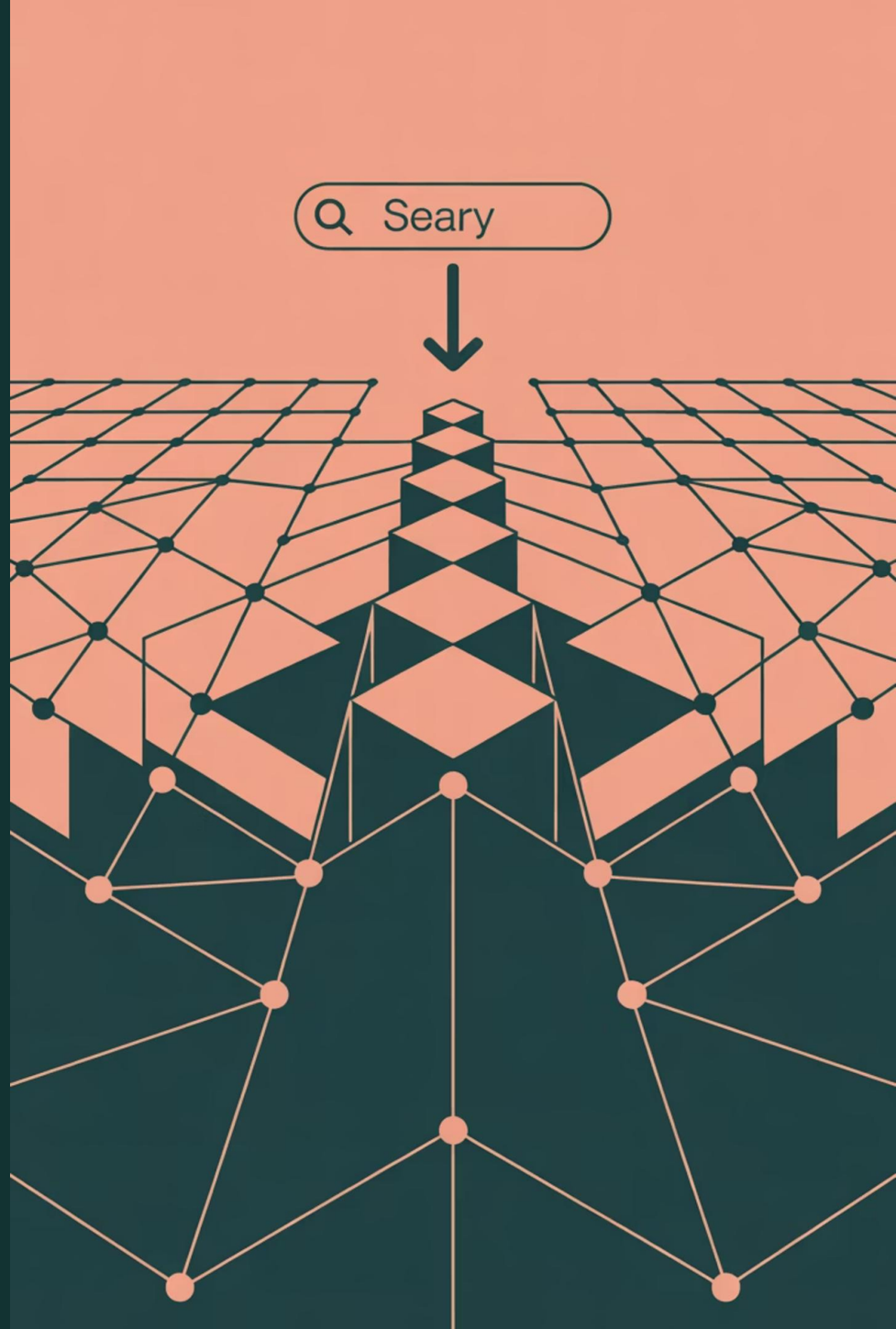
ViT-B/32 모델을 사용하여
512차원 벡터로 인코딩

FAISS 인덱싱

Facebook의 FAISS 라이브러리로
고속 유사도 검색 구현

유사도 측정

코사인 유사도를 사용하여 상위 K개 결과 반환



데이터베이스 설계 PostgreSQL 스키마 구조

Users 테이블

사용자 정보, 선호 스타일, 검색 히스토리 저장

Products 테이블

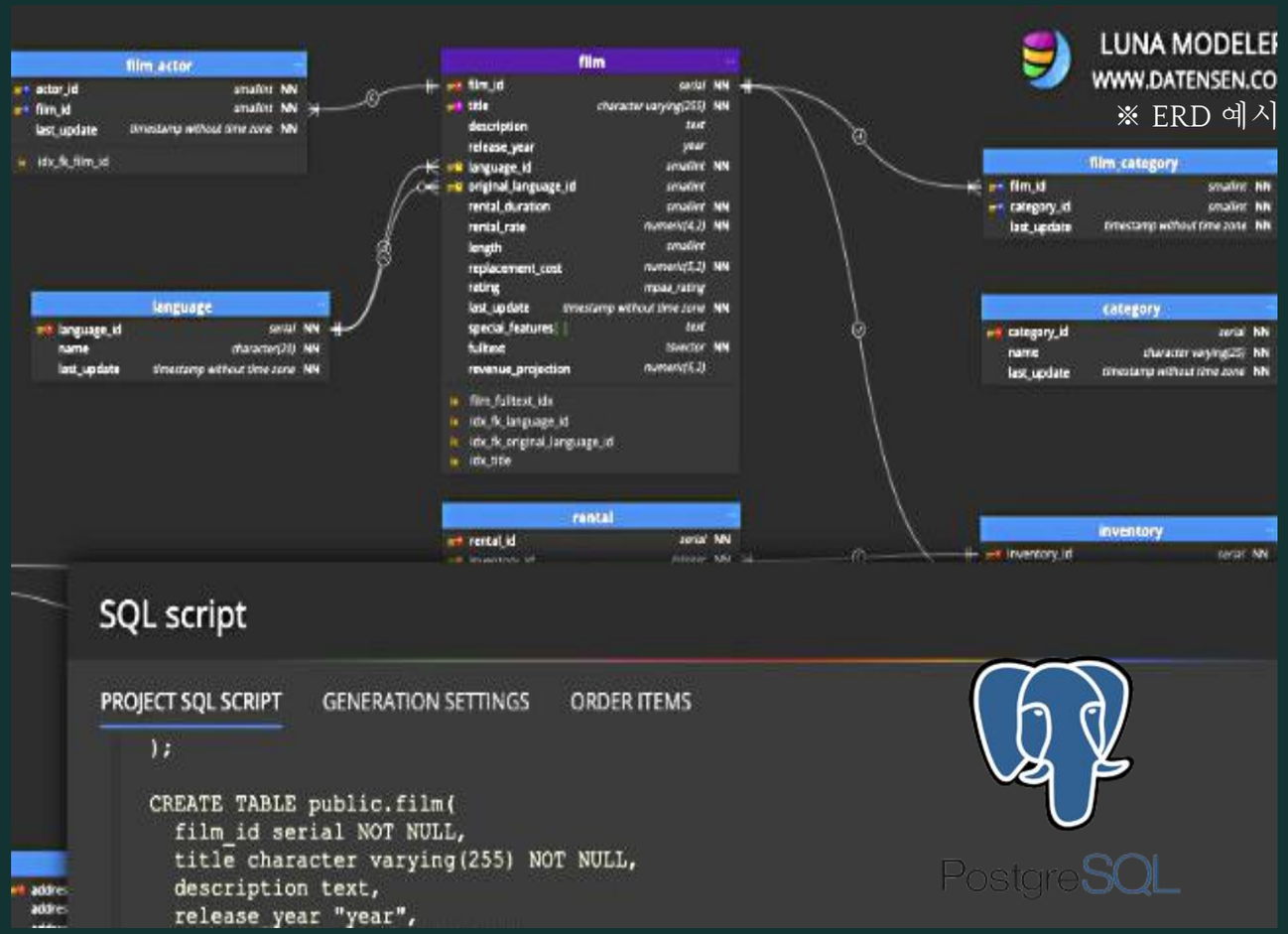
상품 정보, 가격, 이미지 URL, 브랜드, 카테고리

Embeddings 테이블

CLIP 벡터, Vision API 특징, 메타데이터

Analytics 테이블

사용자 행동 분석, 추천 성과



가상피팅

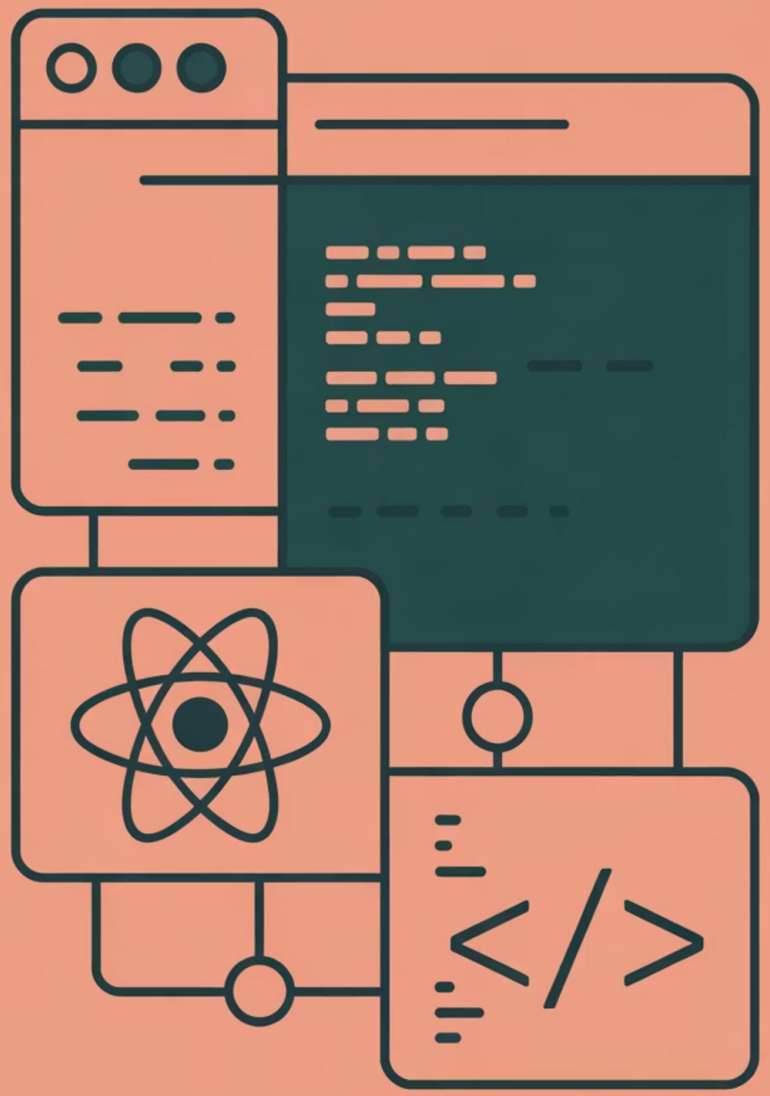
OOTDiffusion - Hugging Face API

웹 기반 가상 피팅 서비스 – **Gradio UI**

가상피팅 구현

```python
class KFashionVirtualTryOn:
    def process_user_image(
        # 3. 각 추천 아이템에 대해 가상 피팅 생성
        print("가상 피팅 이미지 생성 중...")
        tryon_results = []

        for item in recommendations:
            # 데모용 가먼트 이미지 생성 (실제로는 DB에서 로드)
            garment_image = self.load_or_generate_garment(item)

            # 가상 피팅 적용
            fitted_image = self.tryon_model.generate_tryon(
                person_image=user_image,
                garment_image=garment_image,
                garment_description=item.description
            )

            tryon_results.append(fitted_image)

        return user_profile, recommendations, tryon_result
```

```python
# Gradio UI 구성
def create_gradio_interface():
    """Gradio 웹 인터페이스 생성"""

    # 서비스 인스턴스
    service = KFashionVirtualTryOn()

    def process_image(image):
        """이미지 처리 함수"""
        if image is None:
            return None, "이미지를 업로드해주세요.", []

        try:
            # PIL 이미지로 변환
            if not isinstance(image, Image.Image):
                image = Image.fromarray(image)

            # 처리
            profile, recommendations, results = service.process_user_image(image, num
```



개발 진행 상황
Frontend

Frontend 기술 스택

React

최신 React 기능 활용 검토 중

Hooks, Suspense, Server Components

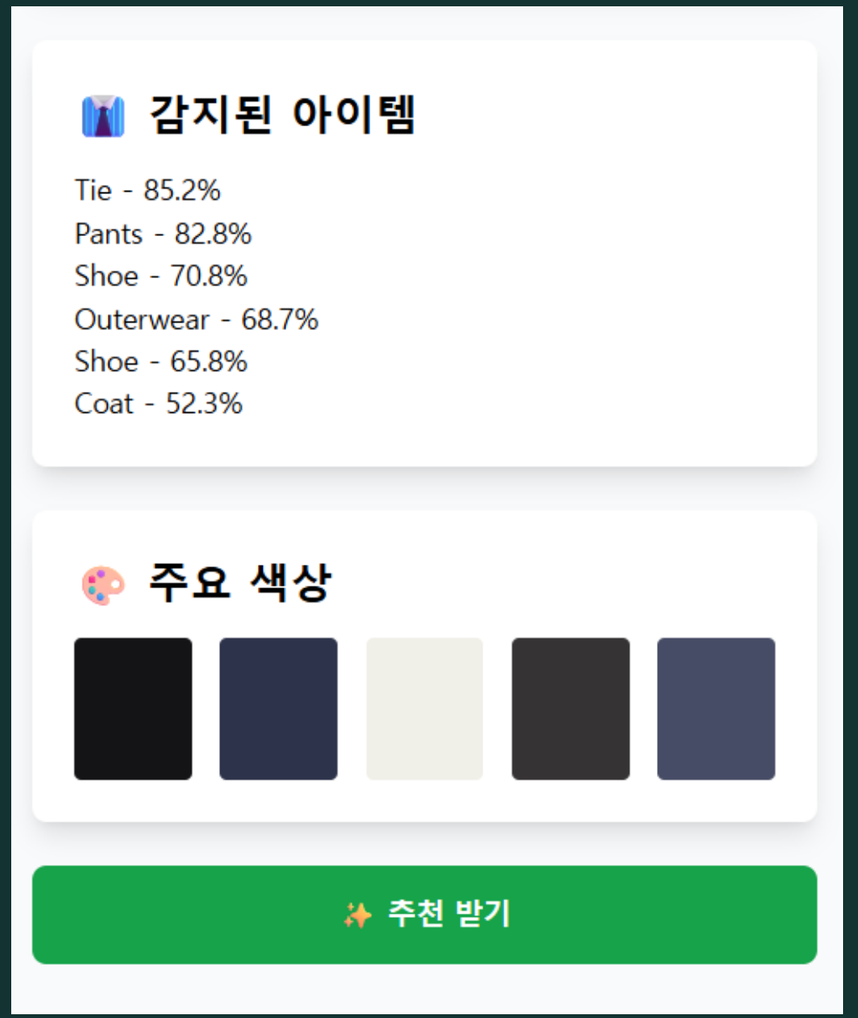
TypeScript

타입 안정성 확보로 런타임 에러 최소화

및 개발 생산성 향상

모던 웹 기술 스택을 활용하여 빠르고 반응성 높은 사용자 인터페이스를 구현 중

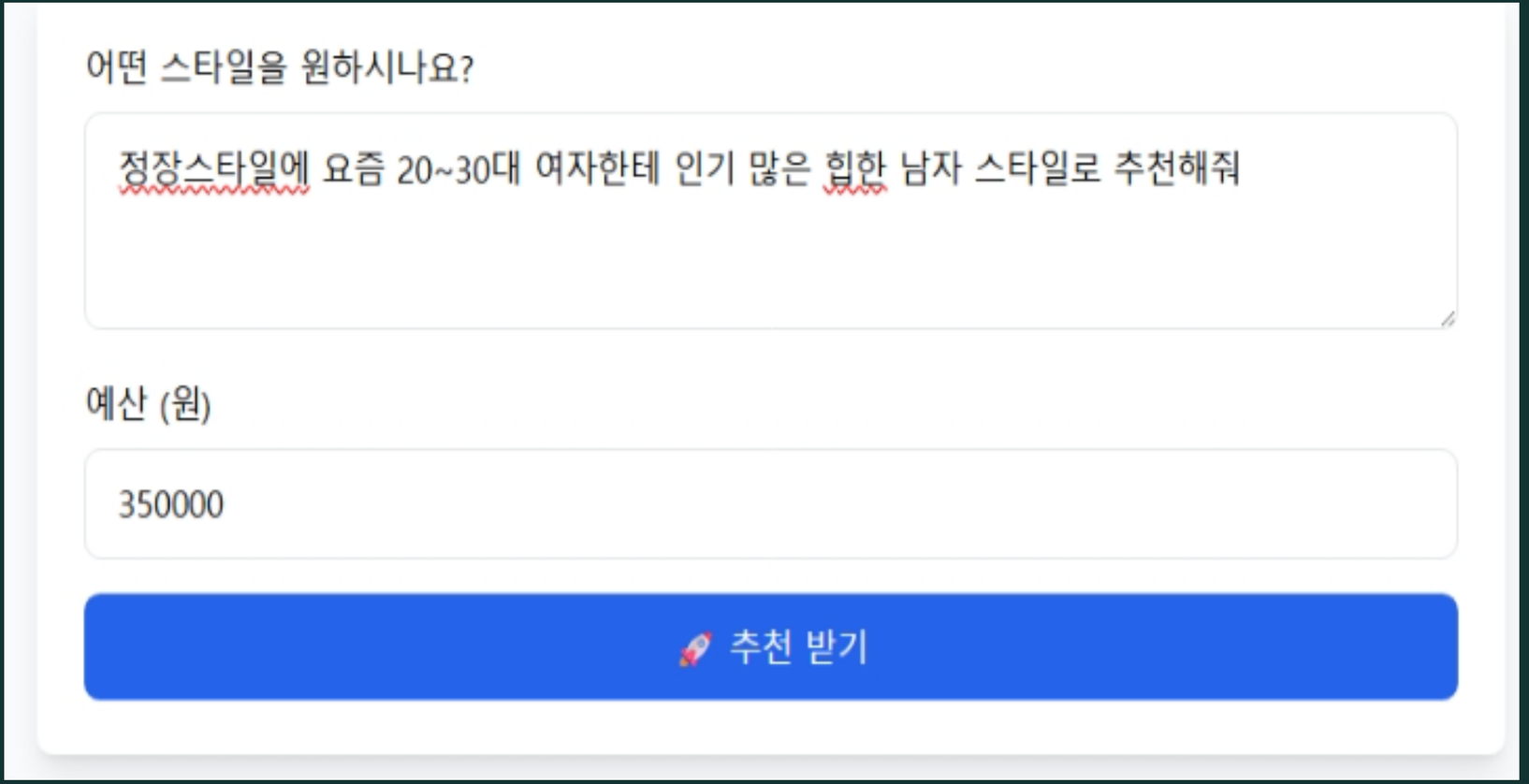
※ UI 변경 예정!



이미지 업로드 및 Google Vision API 분석 화면

※ UI 변경 예정!

대화형 스타일링 인터페이스



확장 플랜

AI 챗봇 기능

자연어로 스타일 선호도 입력,
AI가 맞춤형 추천 생성

- 실시간 대화형 인터페이스
- 마크다운 렌더링 지원
- 이미지 첨부 가능
- 대화 히스토리 저장

※ UI 변경 예정!

입력 및 요청 스타일 분석, 추천 상품 상세 및 최저가 비교

스타일 분석

20~30대 여성들 사이에서 힙한 남자 정장 스타일이 각광받고 있습니다. 사복에 포멀한 요소를 섞어 현대적이고 세련된 룩을 완성하는 추세입니다.

추천 상품

무신사스탠다드
[무신사 스탠다드] 오버 사이즈 크롭 블레이저 [블랙] MMAJK008-BK
51,890원

무신사스탠다드
무신사스탠다드 오버사이즈 블레이저 MMJJK101
57,050원

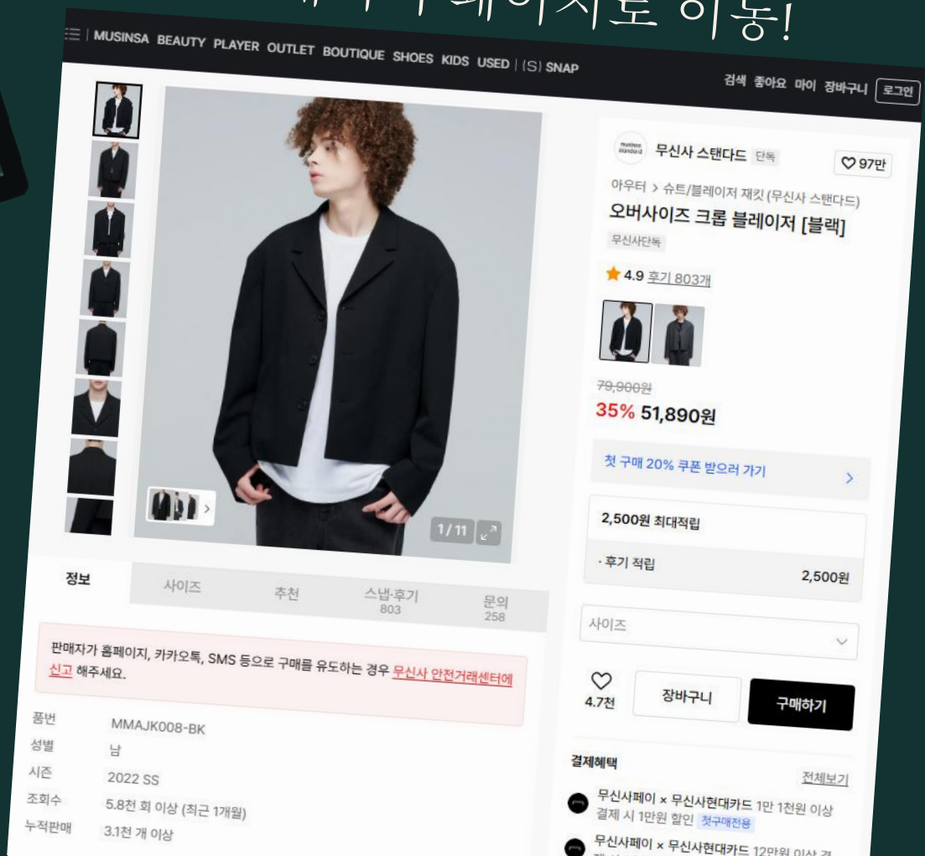
무신사스탠다드
무신사스탠다드 빅 대디 오버사이즈 블레이저 자켓 MMAJK009
60,160원

[단독진행]베른 오버핏 트렌치 코트 베이지 그레이 블랙 무료배송
65,000원

남자 오버핏 블레이저 자켓 봄 가을 투버튼 마이 블랙 그레이
65,900원

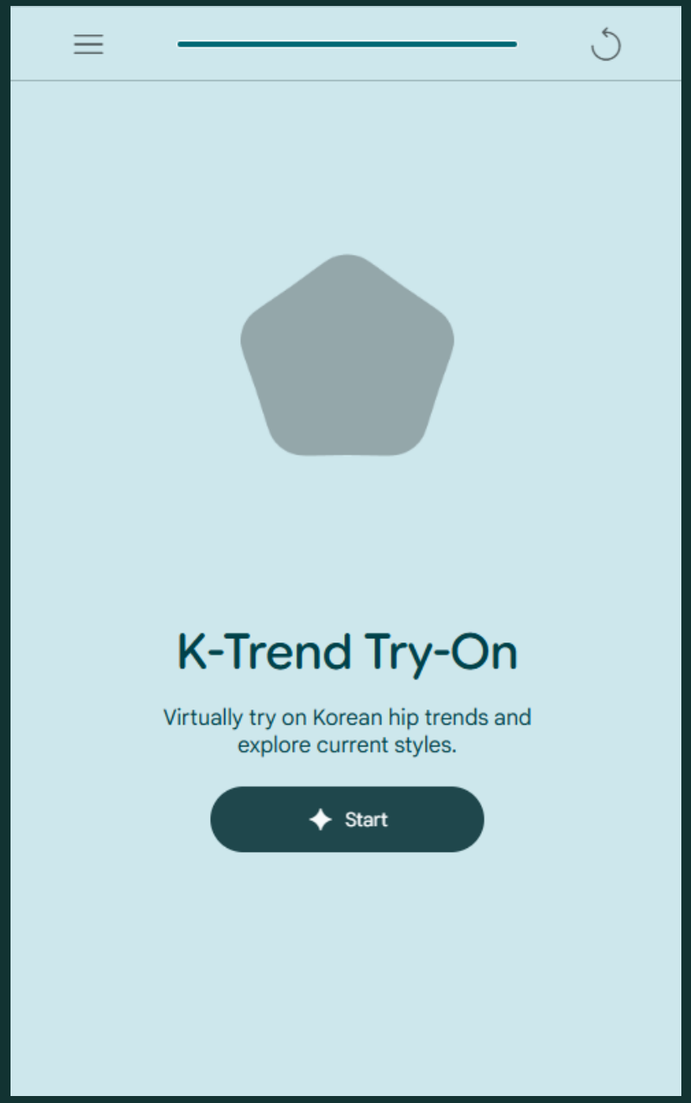
리버클래시
리버클래시 릴렉스핏 트윌 싱글 자켓 LJW24202
69,000원

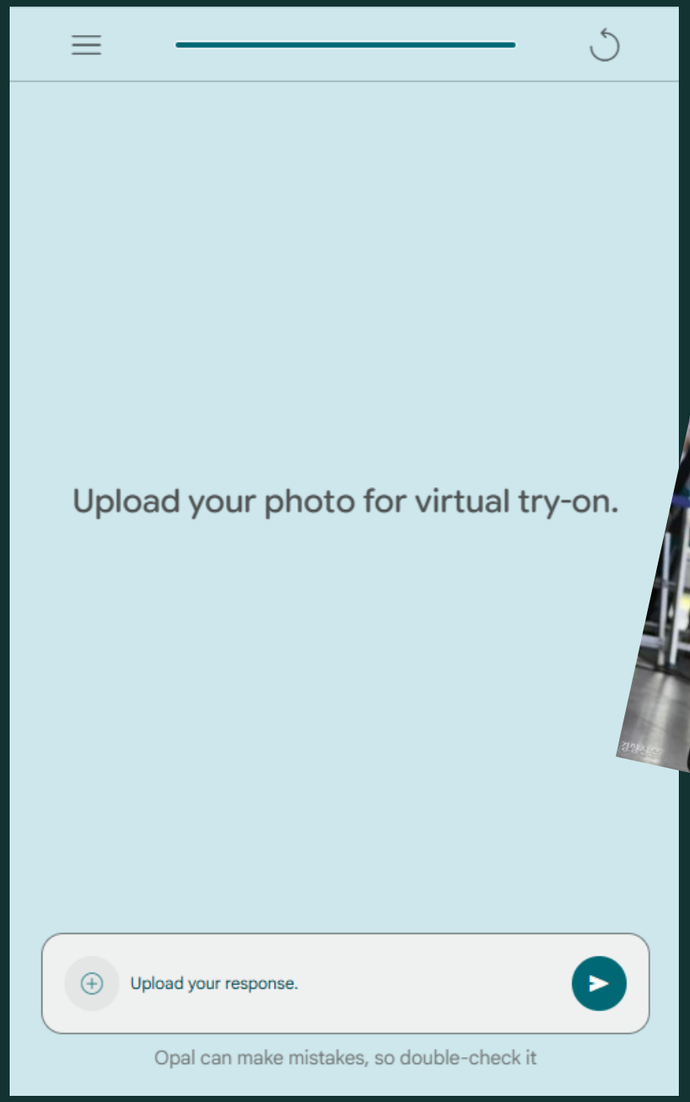
네이버 페이지로 이동!



※ UI 변경 예정!

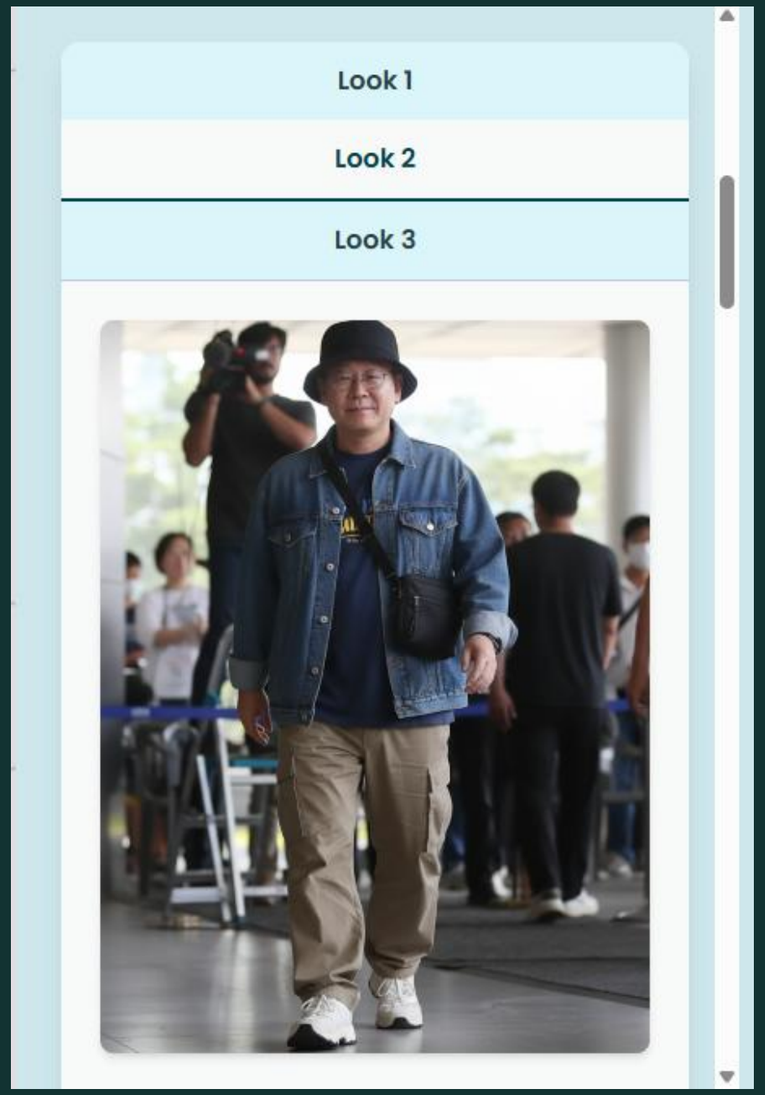
가상 피팅 추천 시스템

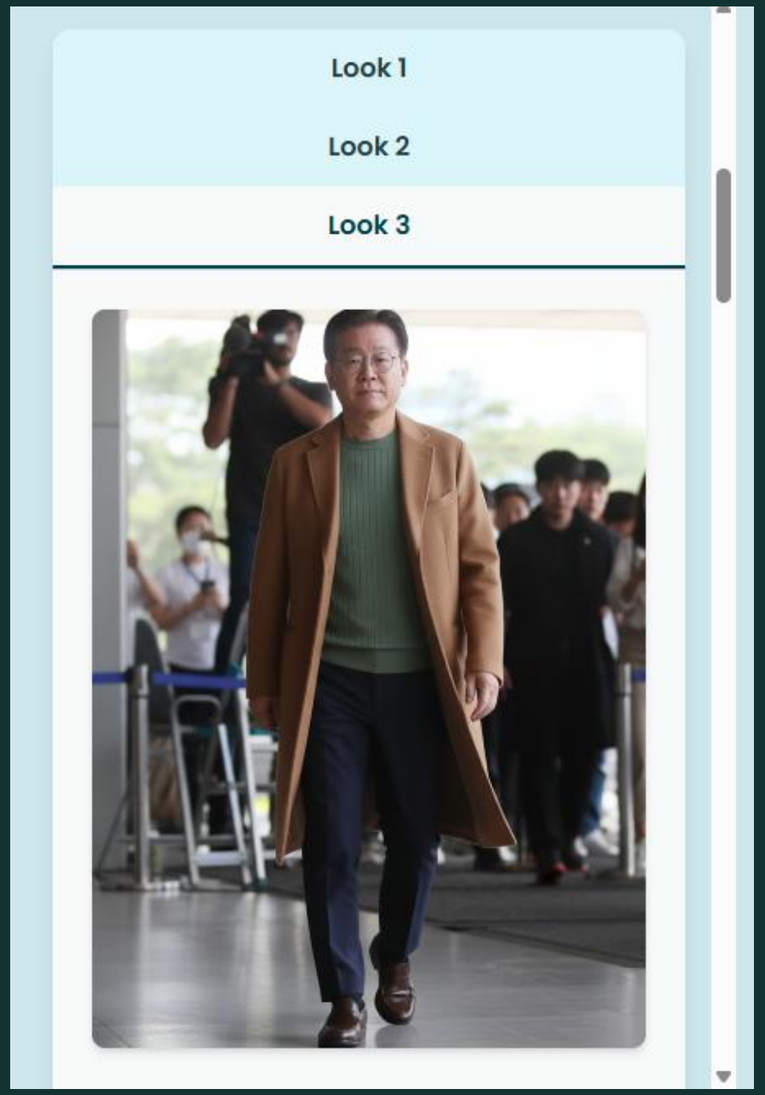




※ UI 변경 예정!

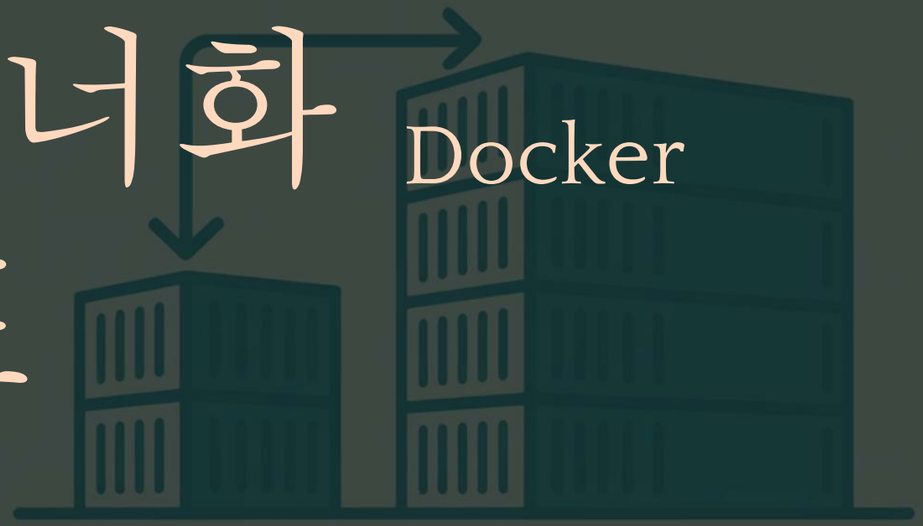
가상 피팅 추천 시스템

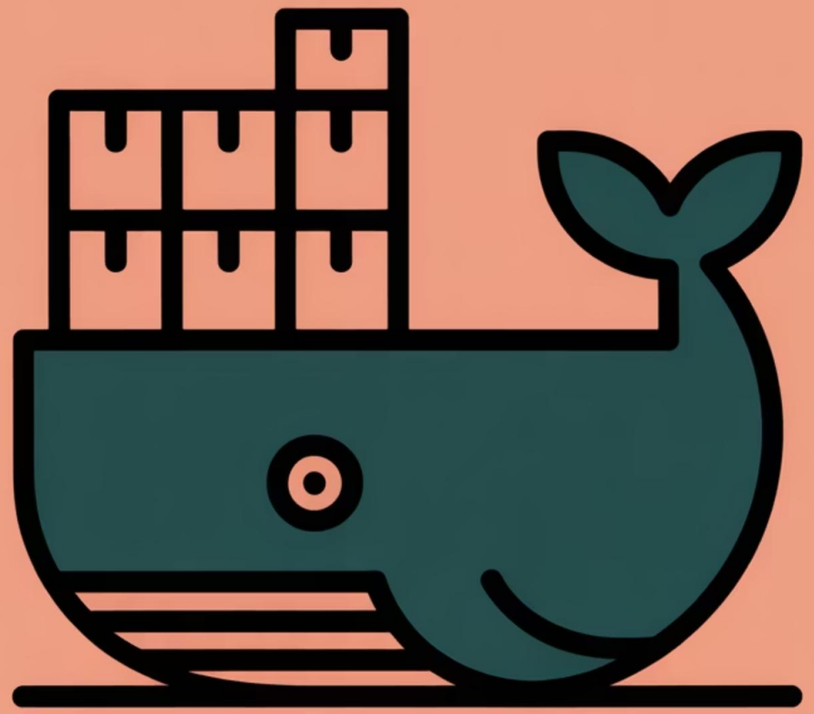




컨테이너화 및 배포

Docker





Docker 구성 전략

멀티 컨테이너 아키텍처

Backend 컨테이너

FastAPI 애플리케이션과 Python 런타임 환경

Frontend 컨테이너

React 빌드 결과물을 서빙하는 Nginx

PostgreSQL 컨테이너

관계형 데이터베이스 서비스

Redis 컨테이너

캐싱 및 세션 관리

각 서비스를 독립된 컨테이너로 분리하여 개발, 테스트, 배포 환경의 일관성을 확보했습니다.

Dockerfile 구성

멀티 스테이지 빌드 최적화

Backend Dockerfile

```
FROM python:3.10-slim as baseWORKDIR /appCOPY requir
ements.txt .RUN pip install --no-cache-dir -r requir
ements.txtFROM base as productionCOPY . .CMD ["uvico
rn", "main:app", "--host", "0.0.0.0"]
```

경량 베이스 이미지 사용과 레이어 캐싱으로 빌드 시간을 단축했습니다.

Frontend Dockerfile

```
FROM node:18-alpine as buildWORKDIR /appCOPY package
*.json ./RUN npm ciCOPY . .RUN npm run buildFROM ngi
nx:alpineCOPY --from=build /app/dist /usr/share/ngin
x/html
```

빌드 단계와 런타임 단계를 분리하여 최종 이미지 크기를 최소화했습니다.

향후 계획

개발 및 고도화 로드맵

알고리즘 고도화 및 기능 추가

추천 알고리즘 개선

- 컨텍스트 인식: 날씨, 장소 고려
- 강화: 사용자 피드백 기반 학습
- A/B 테스트: 추천 로직 최적화
- Rcommend : 추천 로직 최적화

성능 최적화 및 고도화

- LLM & UI 고도화
- Redis 캐싱 전략 고도화
- 데이터베이스 쿼리 최적화
- 비동기 처리 및 백그라운드 작업

신규 기능

- 소셜 기능:
  코디 공유 및 팔로우
- 위시리스트:
  관심 상품 관리

References

You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detection, Yuxin Fang *, Bencheng Liao

A lightweight and accurate YOLO-like network for small target detection in Aerial Imagery, Alessandro Bettia

An Image is Worth 16X16 Words: Transformers for Image Recognition at Scale, Al exey Dosovitskiy

Q&A

감사합니다