2025-09-26-금요일

K K O K K A O T

# 꼬까옷

## 객체 인식 및  딥러닝 기반 패션 코디 추천 서비스

**팀 : 삼석사와아이들**

팀장 : 이문용
팀원 : 김재현 김현진 어지유 박재익

CONTENTS

# 목차

# 01

# 서비스 개요

# 서비스 개요


# 꼬까옷

객체 인식 및  딥러닝 기반
패션 코디 추천 서비스

# 서비스 흐름 1/2

## 실시간 인식 YOLOv11

이미지 내에서 옷이 어디에 있는지, 상의인지, 하의인지, 아우터인지를 신속하게 식별

## 의류 크롭 특징

사진이나 이미지에서 원하지 않는 외부의 특정 부분을 잘라내어 원하는 부분만 남기는 작업

즉 의류 자체의 시각적 특징을 완벽하게 확보

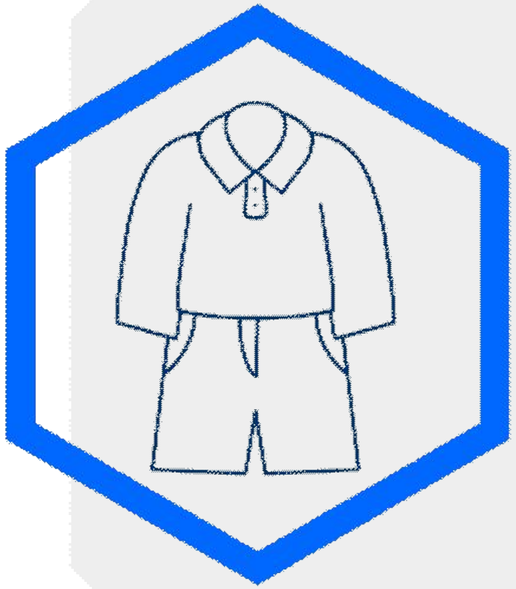
## 정밀 분할 (마스크orSAM2)

배경이나 다른 사물과 분리된 픽셀 단위의 정교한 의류 마스크(Mask)를 생성

# 서비스 흐름 2/2

## 가상 옷장

정밀하게 분리된 의류 이미지는 이제 '자산화' 단계에 들어감



## 코디 추천

카테고리, 색상, 계절과 같은 메타데이터가 사용자별 데이터베이스에 저장되어 비로소 '디지털 옷장'이 완성

# 02

# 데이터 수집

# 데이터 수집


## DeepFashion2

데이터 수집 단계에서, 저희는 대표적인 글로벌 패션 데이터셋인 DeepFashion2를 초기에 검토했습니다.

하지만 분석 결과, 이 데이터셋은 저희가 목표로 하는 'K-Fashion'의 고유한 트렌드와 스타일을 반영하기에는 한계가 있다고 판단했습니다.

따라서 저희는 범용 데이터셋 대신, 프로젝트의 목적에 더 정확하게 부합하는 국내 AI Hub의 K-Fashion 특화 데이터셋을 활용하기로 최종 결정했습니다.

# 데이터 수집

## K-Fashion 이미지

정교한 추천을 위한 데이터 기반 마련

- AI Hub의 K-Fashion 이미지 데이터를 활용하여
  의류의 핵심 속성을 태깅하고, 이를 통해 추천 모델의
  정확성과 품질을 향상시킬 예정

- 총 1,200,000건 분량의 K-Fashion 이미지 데이터셋 구축

- 대분류 10가지, 세부속성 186가지, 스타일 23가지 레이블링 정보
  전문영역에서의 활용성을 고려한 스타일 및 세부속성 분류 구성

*2021, AI hub*

# 03

# 모델 구현

# 모델 구현

저희 서비스는 세 가지 핵심 AI 모델이 순서대로 작동하여 구현됩니다.

첫째, '탐지' 모델 (YOLOv11)이 이미지 속에서 옷을 빠르고 정확하게 찾아냅니다.

둘째, '분할' 모델이 찾아낸 옷의 형태만 정교하게 분리하여 특징을 분석합니다.

마지막으로, '추천' 모델이 분석된 정보를 바탕으로 최신 트렌드에 맞는 코디를 제안합니다.

요약하자면, '찾고, 분석하고, 추천하는' 이 3단계 자동화 모델이 저희 서비스의 핵심 기술입니다.

# YOLOv11

File Edit View Option

MM.YYYY

Your company.com

```
TRAIN_RATIO = 0.9
SEED = 42
COPY_MODE = "copy"  # "copy" | "link"
IMG_EXTS = {".jpg", ".jpeg", ".png", ".bmp", ".webp"}

MAJOR_ORDER = ["상의","하의","아우터","원피스","신발","가방","도
MAJOR_EN = {"상의":"top","하의":"bottom","아우터":"outer","원피
    "신발":"shoes","가방":"bag","모자":"hat","액세서리"

# ========== 내부 설정 ==========
random.seed(SEED)
XPAT_NUM = re.compile(r"^X좌표(\d+)$")
YPAT_NUM = re.compile(r"^Y좌표(\d+)$")
POLY_ARRAY_KEYS = ["좌표","points","polygon"]
RECT_KEYSETS = [
    (("X좌표","x","x1","left","left_x"), ("Y좌표","y","y1","to
    ("가로","width","가로(px)","w"),     ("세로","height","세
    (("x1","좌상단X","left_x","left"), ("y1","좌상단Y","left_y
    ("x2","우하단X","right_x","right"), ("y2","우하단Y","righ
]
NAME_KEYS = [
    ("이미지 정보","이미지 파일명"), ("파일 이름",), ("이미지 정
    ("파일이름",), ("file_name",), ("filename",), ("image",),
]
SIZE_KEYS = [
    ("이미지 정보","이미지 너비","이미지 높이"), ("width","heigh
]
```

```
# 사용자 설정
YOLO_VERSION = "./yolo11n.pt"  # yolov11s.pt, yolov11m.pt, yol
PROJECT_NAME = "kfashion_yolov11"
EPOCHS = 100
IMGSZ = 640
BATCH = 32

# 데이터셋 위치
DATA_DIR = Path("./_data/project/Fashion/Data/03AiHub_K-fashic
DATA_YAML = Path("./_data/project/Fashion/Data/03AiHub_K-fashi

# 결과 디렉토리
RESULTS_DIR = Path("./_data/project/Fashion/Code/runs/detect")

# 학습 명령어 생성
train_cmd = f"""
yolo detect train \
  model={YOLO_VERSION} \
  data={DATA_YAML} \
  epochs={EPOCHS} \
  imgsz={IMGSZ} \
  batch={BATCH} \
  name={PROJECT_NAME}
"""

# 실행
print("🔧 YOLOv11 학습 시작...")
print(train_cmd)
subprocess.run(train_cmd, shell=True, check=True)
print(f"\n✅ 학습 완료! 결과: {RESULTS_DIR}/weights/best.pt")
```


```
Image sizes 640 train, 640 val
Using 8 dataloader workers
Logging results to /home/jaeik/Project/Study25/runs/detect/kfashion_yolov112
Starting training for 100 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/100      4.59G      1.397      3.872      1.747        209        640: 2% ──────────── 39/1572 8.3it/s 10.5s<3:06
```

# YOLOv11

File Edit View Option

MM.YYYY

Your company.com


```python
def fail(msg: str):
    print(f"[ERR] {msg}", file=sys.stderr)
    sys.exit(1)

def main():
    # 존재 확인
    if not YOLO_MODEL.exists():
        fail(f"모델 파일이 없습니다: {YOLO_MODEL.resolve()}")
    if not DATA_YAML.exists():
        fail(f"data.yaml 이 없습니다: {DATA_YAML.resolve()}")

    # runs/detect/{PROJECT_NAME}로 저장되게 하려면 project와 name을 함께 지정
    project_arg = f"project={str(PROJECT_DIR.resolve())}"
    name_arg = f"name={PROJECT_NAME}"

    # yolo 명령을 리스트 인자로 전달 (공백 경로 안전)
    train_cmd = [
        "yolo", "detect", "train",
        f"model={str(YOLO_MODEL.resolve())}",
        f"data={str(DATA_YAML.resolve())}",
        f"epochs={EPOCHS}",
        f"imgsz={IMGSZ}",
        f"batch={BATCH}",
        f"device={DEVICE}",
        f"workers={WORKERS}",
        project_arg,
        name_arg,
    ]
```

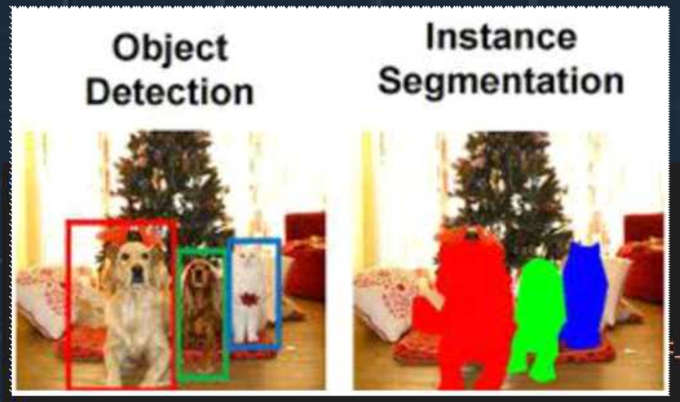
```
YOLO11n summary (fused): 100 layers, 2,584,102 parameters, 0 gradients, 6.3 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 90/90 2.2it/s 40.4s
                   all      11513      33945      0.874      0.467      0.552      0.466
                   top       6976      13952      0.876      0.471      0.563      0.477
                bottom       6054      12188      0.913       0.47      0.558      0.449
                 outer       1761       3521      0.805      0.444      0.515      0.448
                 dress       2182       4364      0.903      0.484      0.572      0.491
```

# Segmentation

File Edit View Option

MM.YYYY

Your company.com




```
_dress","sling_dress"

def build_cfg(train_name, val_name, out_dir, num_classes=13, img_min=640, base_lr=2
    cfg = get_cfg()
    cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_r
    cfg.MODEL.MASK_ON = True
    cfg.DATASETS.TRAIN = (train_name,)
    cfg.DATASETS.TEST = (val_name,)
    cfg.DATALOADER.NUM_WORKERS = 2
    cfg.SOLVER.IMS_PER_BATCH = batch_size
    cfg.SOLVER.BASE_LR = base_lr
    cfg.SOLVER.MAX_ITER = max_iter
    cfg.SOLVER.STEPS = []
    cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
    cfg.MODEL.ROI_HEADS.NUM_CLASSES = num_classes
    cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = score_thresh
    cfg.INPUT.MIN_SIZE_TRAIN = (img_min,)
    cfg.INPUT.MIN_SIZE_TEST = img_min
    cfg.MODEL.ROI_BOX_HEAD.CLS_AGNOSTIC_BBOX_REG = bool(int(class_agnostic))
    cfg.OUTPUT_DIR = out_dir
    os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
    return cfg
```

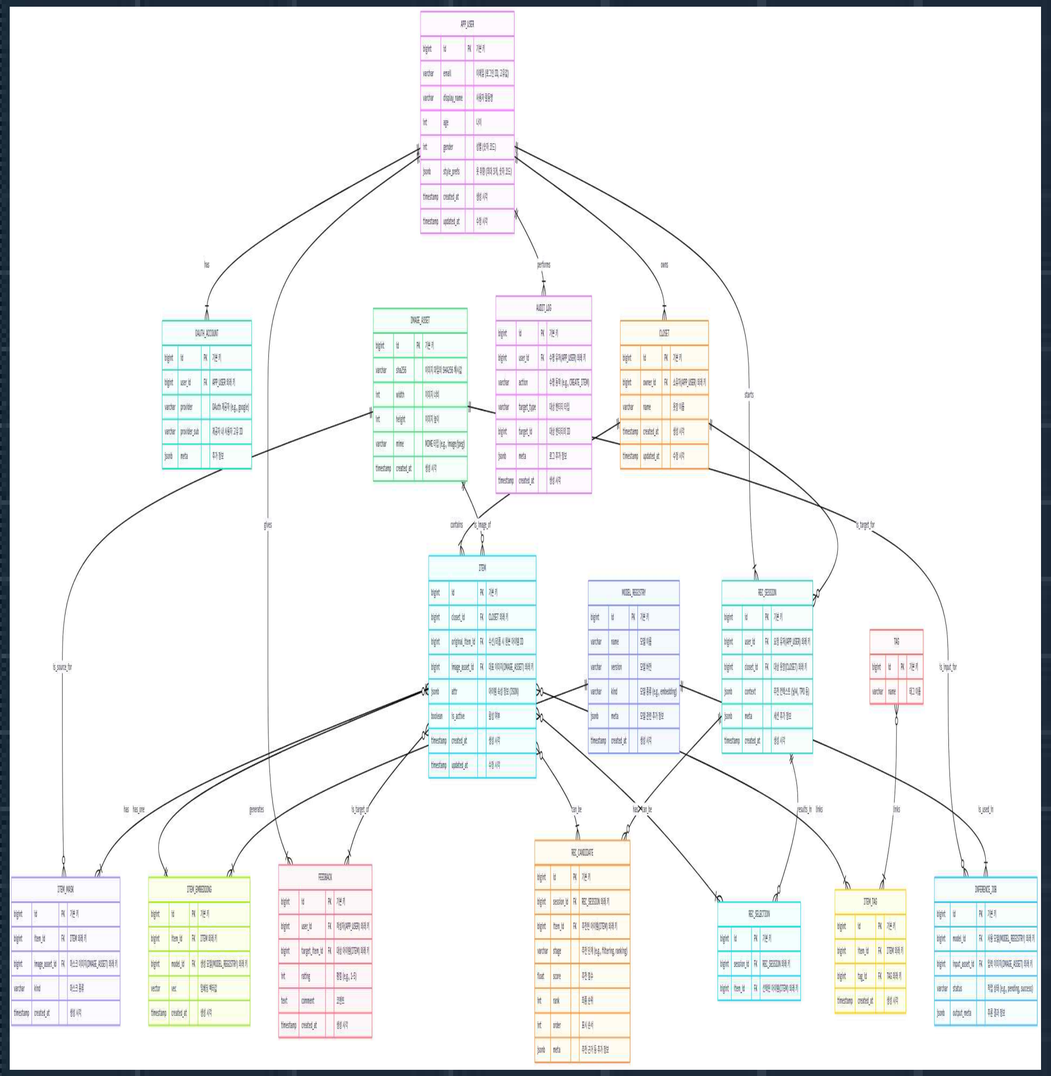
```
class Trainer(DefaultTrainer):
    @classmethod
    def build_evaluator(cls, cfg, dataset_name, output_folder=None):
        if output_folder is None:
            output_folder = os.path.join(cfg.OUTPUT_DIR, "eval")
        os.makedirs(output_folder, exist_ok=True)
        return COCOEvaluator(dataset_name, cfg, True, output_folder)

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--df2_root", required=True)
    ap.add_argument("--out_dir", required=True)
    ap.add_argument("--num_classes", type=int, default=13)
    ap.add_argument("--img_min", type=int, default=640)
    ap.add_argument("--batch", type=int, default=2)
    ap.add_argument("--lr", type=float, default=2.5e-4)
    ap.add_argument("--max_iter", type=int, default=2000)
    ap.add_argument("--score_thresh", type=float, default=0.5)
    ap.add_argument("--class_agnostic", type=int, default=0)
    a = ap.parse_args()

    train_json = os.path.join(a.df2_root, "annotations", "train.json")
    val_json = os.path.join(a.df2_root, "annotations", "validation.json"
    train_img_root = os.path.join(a.df2_root, "train", "image")
    val_img_root = os.path.join(a.df2_root, "validation", "image")
    assert os.path.exists(train_json) and os.path.exists(val_json), "Run

    train_name = "df2_train"
    val_name = "df2_val"
    register_coco_instances(train_name, {}, train_json, train_img_root)
    register_coco_instances(val_name, {}, val_json, val_img_root)
    MetadataCatalog.get(train_name).thing_classes = NAMES
    MetadataCatalog.get(val_name).thing_classes = NAMES

    cfg = build_cfg(train_name, val_name, a.out_dir, a.num_classes, a.im
    trainer = Trainer(cfg)
    trainer.resume_or_load(resume=False)
    trainer.train()

if __name__ == "__main__":
    main()
```

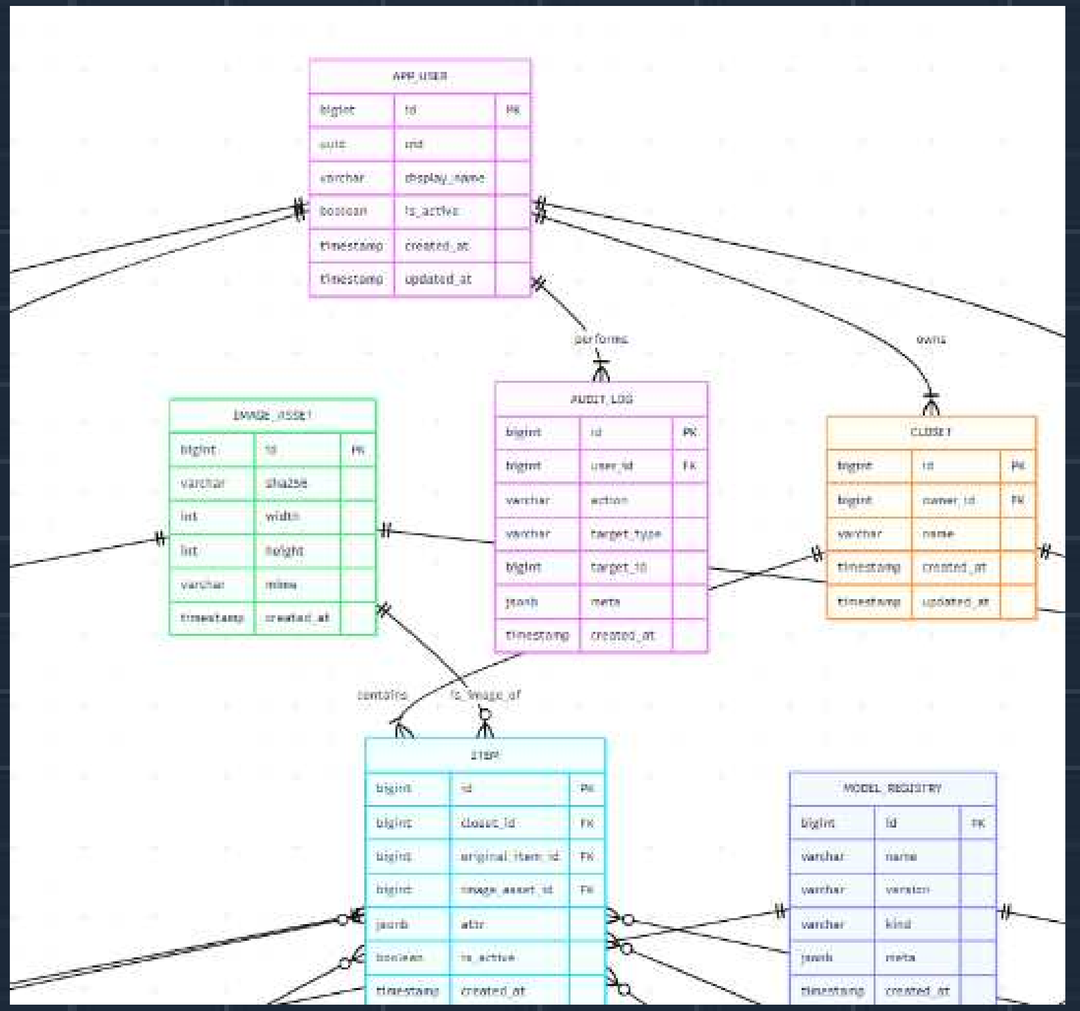
# ERD 구성

File Edit View Option

MM.YYYY

Your company.com





# Table_정의서

File Edit View Option

MM.YYYY

Your company.com

| 테이블명 | 컬럼명 | 데이터 타입 | Y | NULL 여부 | 설명 |
|---|---|---|---|---|---|
| APP_USER | id | bigint | PK | NOT NULL | 기본 키 |
| | email | varchar | Unique | NOT NULL | 이메일(로그인 ID, 고유값) |
| | display_name | varchar | | NULL | 사용자 표시명 |
| | age | int | | NULL | 나이 |
| | gender | int | | NULL | 성별 (숫자 코드) |
| | style_prefs | jsonb | | NULL | 옷 취향 (최대 3개, 숫자 코드) |
| | created_at | timestamp | | NOT NULL | 생성 시각 |
| | updated_at | timestamp | | NOT NULL | 수정 시각 |
| OAUTH_ACCOUNT | id | bigint | PK | NOT NULL | 기본 키 |
| | user_id | bigint | FK | NOT NULL | APP_USER 외래 키 |
| | provider | varchar | | NOT NULL | OAuth 제공자 (e.g. google) |
| | provider_sub | varchar | | NOT NULL | 제공자 내 사용자 고유 ID |
| | meta | jsonb | | NULL | 추가 정보 |
| CLOSET | id | bigint | PK | NOT NULL | 기본 키 |
| | owner_id | bigint | FK | NOT NULL | 소유자(APP_USER) 외래 키 |
| | name | varchar | | NOT NULL | 옷장 이름 |
| | created_at | timestamp | | NOT NULL | 생성 시각 |
| | updated_at | timestamp | | NOT NULL | 수정 시각 |
| ITEM | id | bigint | PK | NOT NULL | 기본 키 |
| | closet_id | bigint | FK | NOT NULL | CLOSET 외래 키 |
| | original_item_id | bigint | FK | NULL | 수선/리폼 시 원본 아이템 ID |
| | image_asset_id | bigint | FK | NOT NULL | 대표 이미지(IMAGE_ASSET) 외래 키 |
| | attr | jsonb | | NULL | 아이템 속성 정보 (JSON) |
| | is_active | boolean | | NOT NULL | 활성 여부 |
| | created_at | timestamp | | NOT NULL | 생성 시각 |
| | updated_at | timestamp | | NOT NULL | 수정 시각 |
| IMAGE_ASSET | id | bigint | PK | NOT NULL | 기본 키 |
| | sha256 | varchar | | NOT NULL | 이미지 파일의 SHA256 해시값 |
| | width | int | | NULL | 이미지 너비 |
| | height | int | | NULL | 이미지 높이 |
| | mime | varchar | | NOT NULL | MIME 타입 (e.g. image/jpeg) |
| | created_at | timestamp | | NOT NULL | 생성 시각 |
| ITEM_MASK | id | bigint | PK | NOT NULL | 기본 키 |
| | item_id | bigint | FK | NOT NULL | ITEM 외래 키 |
| | image_asset_id | bigint | FK | NOT NULL | 마스크 이미지(IMAGE_ASSET) 외래 키 |
| | kind | varchar | | NULL | 마스크 종류 |
| | created_at | timestamp | | NOT NULL | 생성 시각 |
| TAG | id | bigint | PK | NOT NULL | 기본 키 |
| | name | varchar | Unique | NOT NULL | 태그 이름 |
| ITEM_TAG | id | bigint | PK | NOT NULL | 기본 키 |

Query Query History

```sql
-- APP_USER 테이블 생성
CREATE TABLE APP_USER (
    id BIGINT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    display_name VARCHAR(100),
    age INT,
    gender INT,
    style_prefs JSONB,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

-- 테이블 및 각 컬럼에 대한 설명(Comment) 추가
COMMENT ON TABLE APP_USER IS '애플리케이션 사용자 정보 테이블';
COMMENT ON COLUMN APP_USER.id IS '기본 키';
COMMENT ON COLUMN APP_USER.email IS '이메일 (로그인 ID, 고유값)';
COMMENT ON COLUMN APP_USER.display_name IS '사용자 활동명';
COMMENT ON COLUMN APP_USER.age IS '나이';
COMMENT ON COLUMN APP_USER.gender IS '성별 (숫자 코드)';
COMMENT ON COLUMN APP_USER.style_prefs IS '옷 취향 (최대 3개, 숫자 코드)';
COMMENT ON COLUMN APP_USER.created_at IS '생성 시각';
COMMENT ON COLUMN APP_USER.updated_at IS '수정 시각';
```

```sql
select * from app_user
```

Data Output Messages Notifications

| id PK bigint | email character varying (255) | display_name character varying (100) | age integer | gender integer | style_prefs jsonb | created_at timestamp without time zone | updated_at timestamp without time zone |
|---|---|---|---|---|---|---|---|

# 환경 설정

File Edit View Option

MM.YYYY

Your company.com

tsconfig.json server.js U

project > project_api > server.js > ...

```
// server.js

// 1. 필요한 라이브러리들을 불러옵니다.
const express = require('express');
const cors = require('cors');
const { Pool } = require('pg');

// 2. Express 앱을 생성하고 기본 설정을 합니다.
const app = express();
const port = 4000; // 서버를 실행할 포트 번호

app.use(cors()); // 다른 주소(React Native 앱)에서의 요청을 허용
app.use(express.json()); // JSON 형태의 데이터를 서버가 이해할 수 있도록 설정

// 3. 🐘 PostgreSQL 데이터베이스 연결 설정을 합니다.
// (이 정보는 서버에만 저장되어 안전합니다!)
const pool = new Pool({
                          // PostgreSQL 사용자 이름
                        // 데이터베이스 주소
                  // 데이터베이스 이름
                  ostgreSQL 비밀번호
                          // PostgreSQL 기본 포트
});

// 4. ⚙️ API 주소(엔드포인트)를 만듭니다.
// 테스트용 주소: GET 요청이 http://localhost:4000/ 로 오면 "API 서버 정상 작동" 메시지를 보냅니다.
app.get('/', (req, res) => {
  res.send('✅ API 서버 정상 작동');
});
```

tsconfig.json api.ts

project > lib > api.ts > ...

```
// PROJECT01/lib/api.ts
import axios from 'axios';

// 🚨 1단계에서 찾은 내 PC의 IP 주소를 여기에 입력하세요!
const API_URL = 'http://192.168.56.1:4000';

const apiClient = axios.create({
  baseURL: API_URL,
});

// '/api/users' 주소로 데이터를 요청하는 함수
export const getUsers = () => {
  return apiClient.get('/api/users');
};
```

```
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
🚀 백엔드 API 서버가 http://localhost:4000 에서 실행 중입니다.
```

# 04

# 서비스 구현

# 서비스 구현


KKOKKAOT
꼬까옷에 오신 것을
환영합니다
AI가 당신의 체형과 취향을
분석하여 완벽한 스타일링을
제안해드립니다
• 실시간 옷장 분류 • 맞춤 코디 추천
시작하기


꼬까옷
새 계정을 만들어 보세요
로그인
회원가입
???
?@?
회원가입


프로필 설정
1 / 5
사용하실 이름을 알려주세요
개인화된 서비스를 위해 필요합니다
진지전능하신선생님
다음


프로필 설정
2 / 5
성별을 선택해주세요
맞춤 추천을 위해 필요합니다
여성
남성
기타
이전
다음


체형 분석
정확한 코디 추천을 위해 체형 분석이 필요합니다
최적의 분석을 위한 촬영 가이드
• 정면을 바라보고 서서 촬영
• 체형이 잘 드러나는 옷 착용
• 밝은 곳에서 선명하게 촬영
• 전신이 모두 나오도록 촬영
체형 분석을 위한 사진을 업로드해주세요
AI가 자동으로 체형을 분석합니다
카메라 촬영
사진 선택

# 서비스 구현

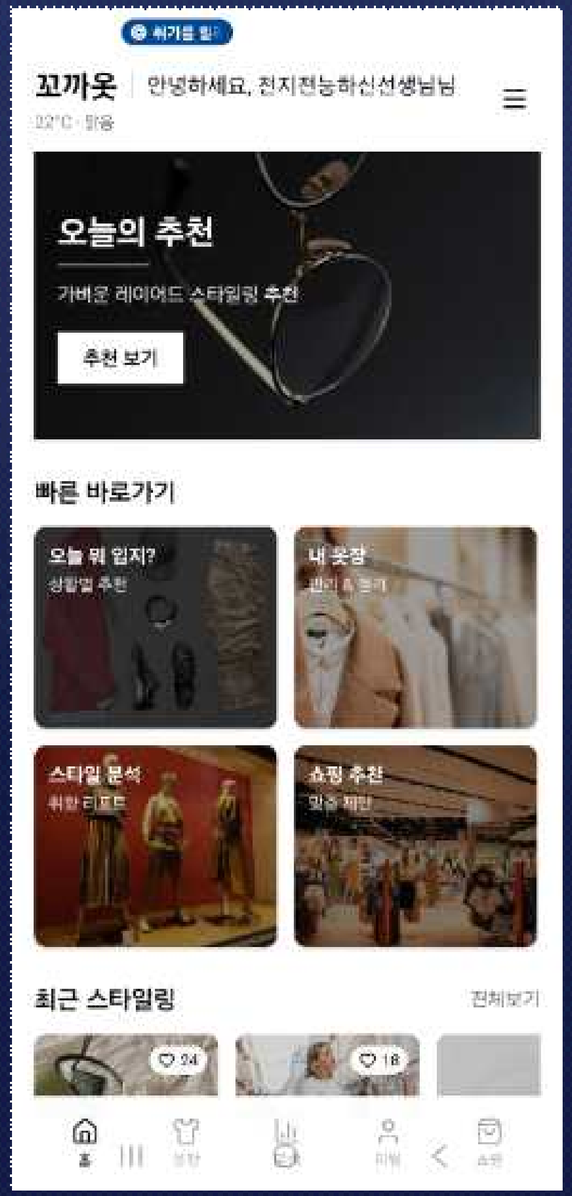

프로필 설정
3 / 5
나이대를 선택해주세요
연령에 맞는 스타일을 제안해드립니다
10대
20대
30대
40대
50대 이상
이전
다음


프로필 설정
4 / 5
본인의 정확한 피부톤을 선택해주세요
정확한 분석을 위해 필요합니다
쿨 페어
웜 페어
쿨 미디움
웜 미디움
쿨 탄
웜 탄
이전
다음

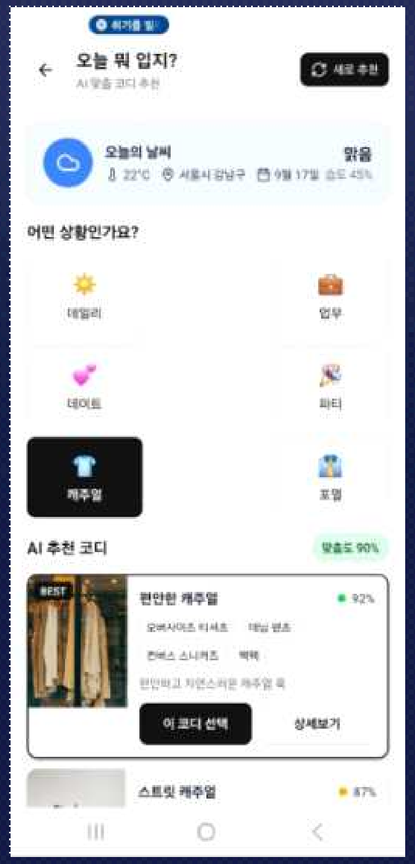

보헤미안
프레피
빈티지
신체 정보 (선택사항)
키 (cm)
몸무게 (kg)
이전
완료


가상 옷장
옷 등록하기
0개
옷을 촬영하여 AI가 자동으로 분류합니다
옷 촬영하기
앨범에서 선택
촬영 가이드
나중에 등록하기

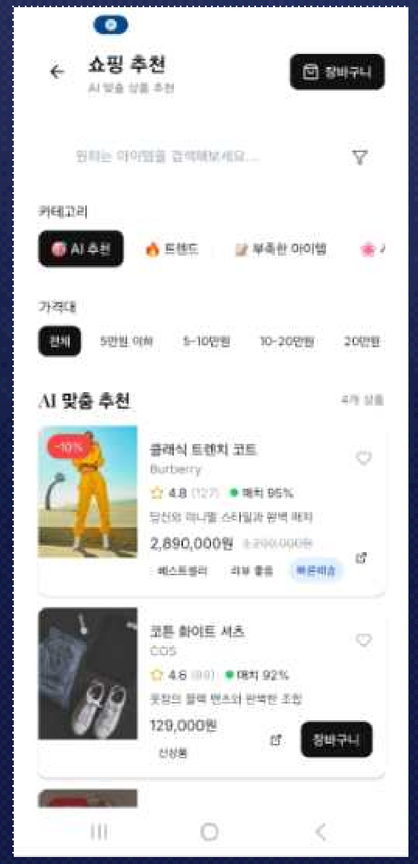
# 서비스 구현


꼬까옷
안녕하세요, 천지전능하신선생님님
오늘의 추천
가벼운 레이어드 스타일링 추천
추천 보기
빠른 바로가기
오늘 뭐 입지?
내 옷장
스타일 분석
쇼핑 추천
최근 스타일링
전체보기

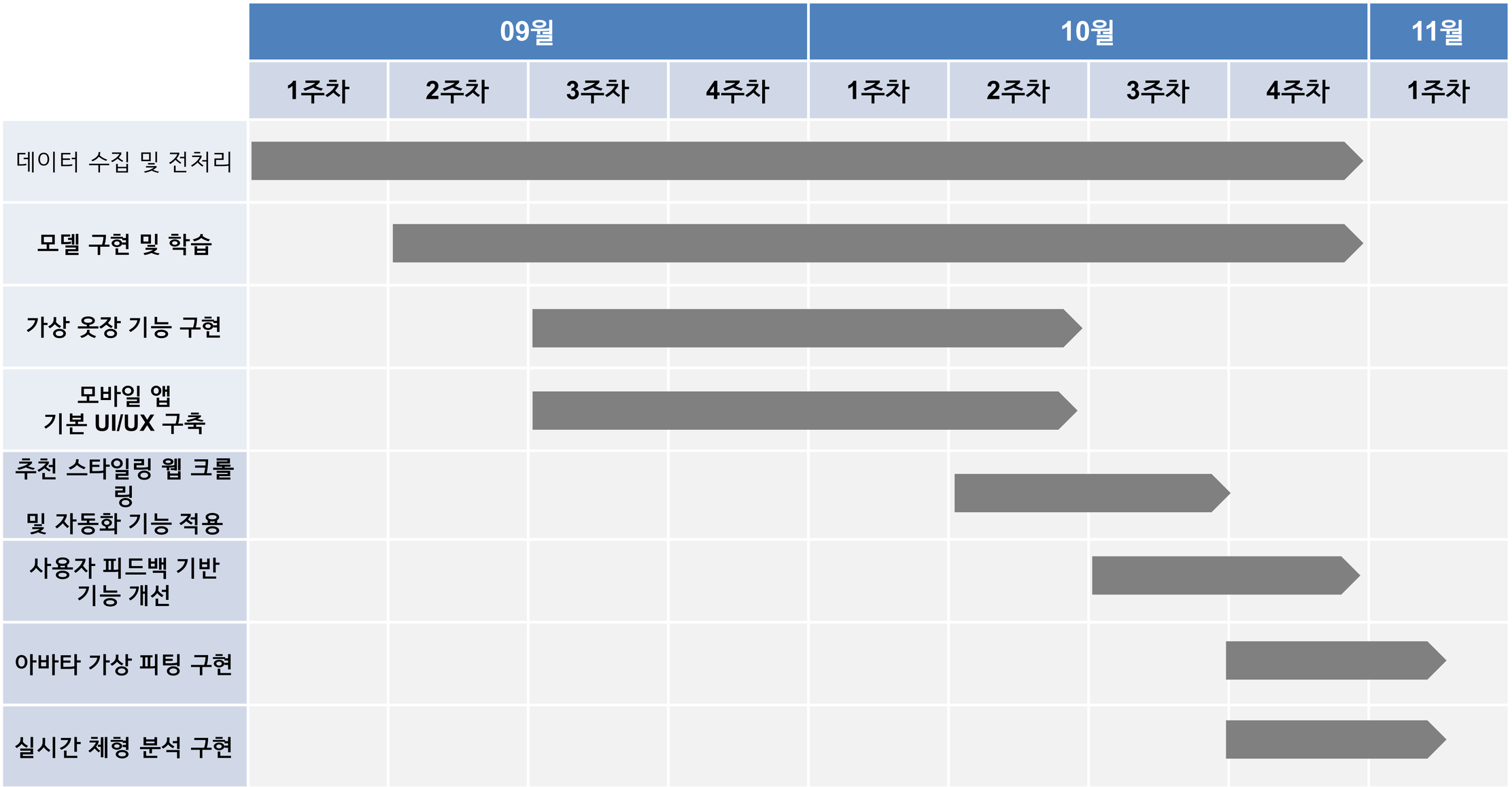

내 옷장
추가
전체
상의
하의
러닝 스니커즈


오늘 뭐 입지?
새로 추천
오늘의 날씨
맑음
어떤 상황인가요?
데일리
업무
데이트
파티
캐주얼
포멀
AI 추천 코디
편안한 캐주얼
이 코디 선택
상세보기
스트릿 캐주얼


스타일 분석
이번 주
이번 달
이번 시즌
올해
당신의 주요 스타일
미니멀
89%
스타일 일관성
스타일 구성
미니멀
캐주얼
클래식
모던
선호 컬러 팔레트


쇼핑 추천
장바구니
카테고리
AI 추천
트렌드
가격대
전체
AI 맞춤 추천
클래식 트렌치 코트
Burberry
2,890,000원
코튼 화이트 셔츠
COS
129,000원
장바구니

# 프로젝트 추진 일정


09월
10월
11월
1주차
2주차
3주차
4주차
1주차
2주차
3주차
4주차
1주차
데이터 수집 및 전처리
모델 구현 및 학습
가상 옷장 기능 구현
모바일 앱
기본 UI/UX 구축
추천 스타일링 웹 크롤링
및 자동화 기능 적용
사용자 피드백 기반
기능 개선
아바타 가상 피팅 구현
실시간 체형 분석 구현

# THANK YOU

## Q&A