

AI 기반 개인 맞춤형 다이어트 관리 플랫폼

Diet Agent

(Full-Stack Web Application)

<에이전트>

김요원 정두균 이혜지 박종훈

목차

Table of contents

01. 프로젝트 개요 및 팀원 소개

02. 프로젝트 목표 및 개발 기간

03. 기술 스택 및 개발 환경

04. 시스템 아키텍처 개요

05. 주요 기능 상세 설명

06. 유틸리티 및 안정성 설계

07. 트러블 슈팅 경험

08. 배포 (Docker)

09. 프로젝트 성과 및 결론

10. 팀원별 소감

1. 프로젝트 개요 및 팀원 소개

1.1 프로젝트 개요

Diet Agent 프로젝트 개요

사용자의 신체 정보, 목표 체중, 목표 기간, 그리고 일상적인 활동 데이터를 기반으로
AI 챗봇과 자동화된 추천 시스템을 통해
식단, 운동, 일정, 진행률을 통합 관리하는 웹 플랫폼

본 프로젝트는 단순한 기능 구현을 넘어
실제 서비스 운영을 가정한 구조 설계와 배포까지를 목표로 개발됨



WHY

지속적인 기록, 분석, 동기 부여가
다이어트의 핵심 요소로 작용



VISION

AI와 자동화 추천 시스템을 통해
다이어트 계획을 통합 관리



VALUE

실제 서비스 운영을 가정한
구조 설계 및 배포

1. 프로젝트 개요 및 팀원 소개

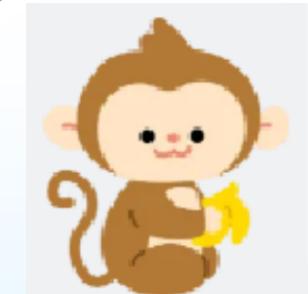
1.2 팀원 소개

이 프로젝트의 최고의 팀원입니다.



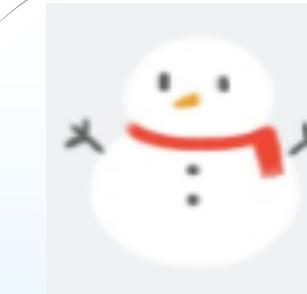
김요원 / PM

- 프로젝트 와이어프레임 설계 및 화면 흐름 기획
- DB 테이블·컬럼 공동 구현
- JWT 기반 인증 기능 구현 (React-Flask)
- 사용자 정보 관리 및 식단 기록 및 저장 기능 구현
- AI 챗봇 응답 로직 공동 구현
- 대시보드 주간 추세 시각화



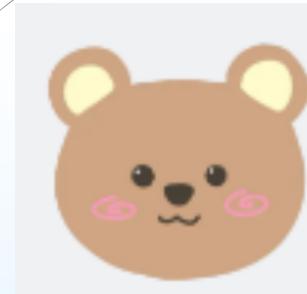
정두균 / developer

- 프론트엔드 전반 UI/UX 설계 및 구현
- React 기반 화면 개발
- 카드형 콘텐츠, 캘린더, 모달 중심의 사용자 친화적 인터페이스 구현
- 디자인, 컴포넌트 구조 설계
- 화면플로우 및 UX 구성
- DB 테이블, 컬럼 공동 구현
- AI 챗봇 응답 로직 공동 구현



이혜지 / developer

- 백엔드(Flask) 전체 디렉토리 구조 설계 및 파일 구현
- DB 관리 파일(sql, database.py) 구현
- 시드데이터 csv 파일생성
- React로 나의 정보 대시보드 초안 구현
- 오늘의 코칭 프로그래밍
- 문서작업 및 README 작성



박종훈 / developer

- 커뮤니티 개발 값 정규화, 금지어, 이상 입력 차단
- 추천 시간 필터 보조
- 몇 파일에 뭉쳐있던 것들을 fixed 기능별 분리
- DB성능/ 초기데이터 추가

2. 프로젝트 목표 및 개발 기간

2.1 프로젝트 목표



- 실제 체중 기록 기반한 목표 진행률 표현
- 캘린더 중심의 일정 관리

- 사용자 정보(DB)를 반영한 변화율 계산 및 저장
- JWT 기반 인증 구조 구현

- DB의 사용자 정보를 반영한 개인화 챗봇

프로젝트 목표

하드 코딩이 아닌 **실제 입력**하는 신체, 일정, 활동량, 목표 체중 및 기간 등에 대한 데이터를 기반으로 **목표 진행률을 계산**해주는 플랫폼 구현

사용자 정보를 **DB**에 저장하고, 이를 반영하여 개개인에게 맞춤 식단과 운동을 추천해주는 **챗봇기능**을 탑재한 플랫폼 구현

JWT를 기반한 인증 구조 구현 및 **Docker**를 활용한 배포 환경 구성

2. 프로젝트 목표 및 개발 기간

2.2 개발 기간

주차별 프로젝트 개발 과정

1주차: 25.12.08 ~
25.12.12

- React로 페이지 골격 완성
(인트로, 메인,
로그인/회원가입, 대쉬보드,
마이페이지, 다이어트관리,
스케줄 관리)
- 로그인/회원가입 기능 구현
-DB, 데이터 설계

2주차: 25.12.15
~ 25.12.19

- 페이지 UI 디자인
- Axios 사용하여
백엔드와 연결 확인 및 세팅
- DB 모델 및 라우트 작성
- jwt 토큰 기능 추가
- 챗봇 간단한 대화 및 추천 기능
추가

3주차: 25.12.22
~ 25.12.26

- 전체적인 기능 구현 점검
(대쉬보드, 정보 입력,
목표에 따른 진행률 계산,
추천 기능, 커뮤니티 및
공지 기능, 챗봇 등)

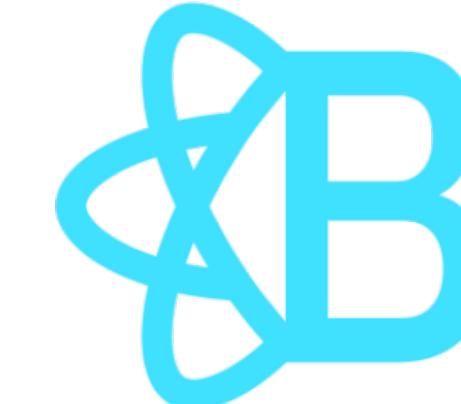
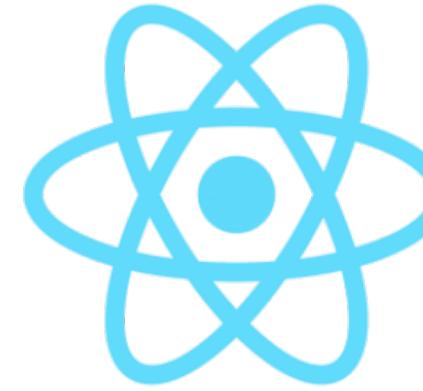
4주차: 25.12.29
~ 26.1.2 +a

- 전체 디자인, 기능 점검 및
마무리
- README 작성 및 문서 작업
- Docker에 배포

3. 기술 스택 및 개발 환경

3.1 Frontend

- React (Vite)
- React Router DOM
- Axios
- CSS 기반 컴포넌트 스타일링



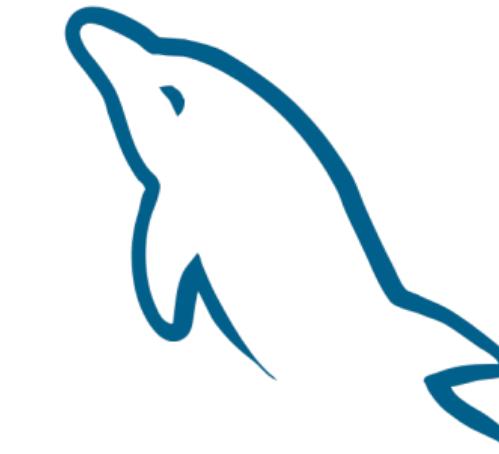
3.2 Backend

- Python Flask
- Flask Blueprint 구조
- PyJWT (JWT 인증)



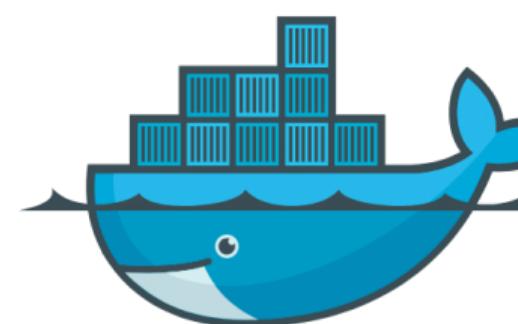
3.3 Database

- MySQL
- sqlite3.Row를 활용한 딕셔너리 조회



3.4 인증 및 보안

- JWT Access Token 방식
- LocalStorage 기반 인증 유지
- 보호된 라우트 처리 (RequireLogin)

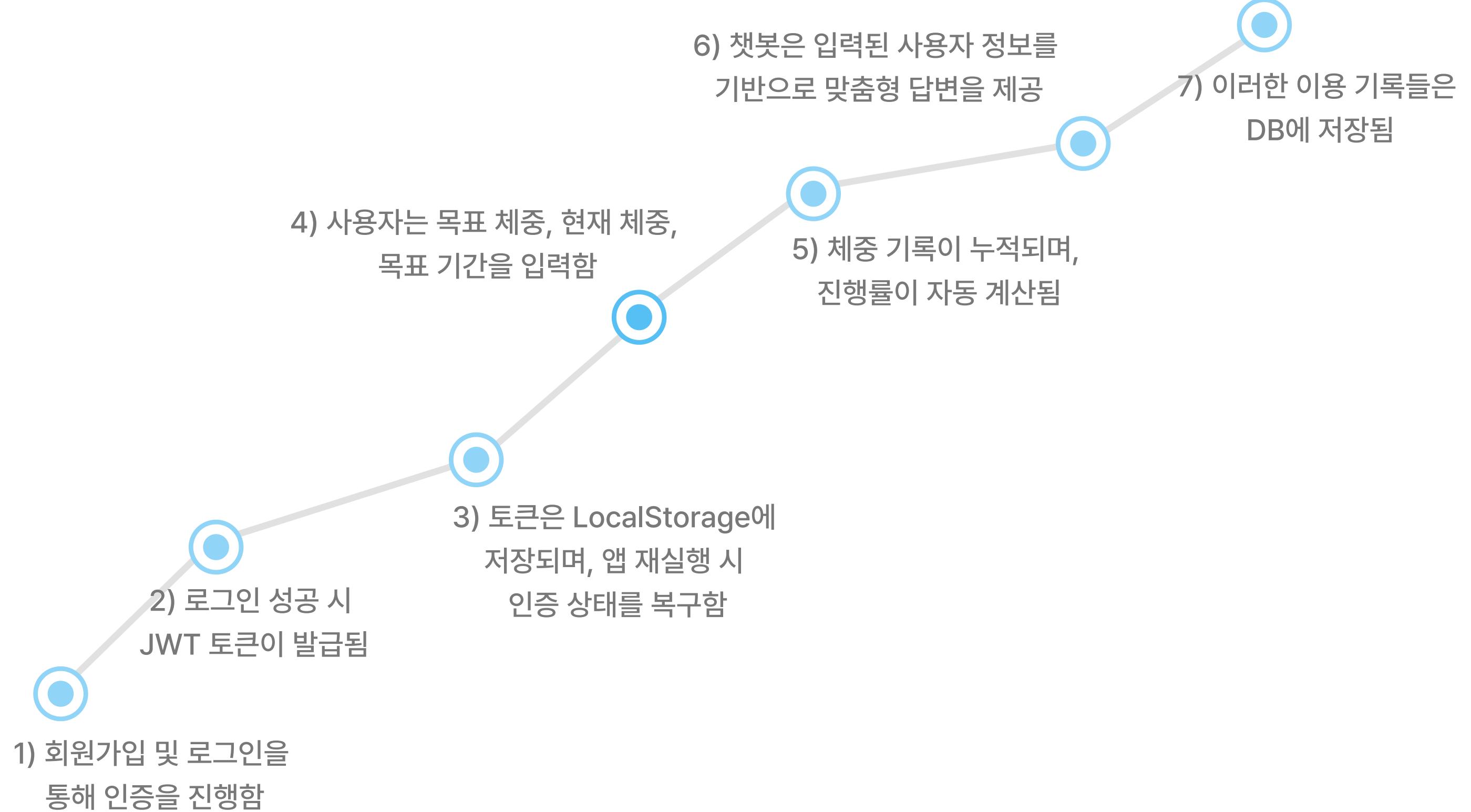


3.5 배포 환경

- Docker
- Dockerfile 기반 컨테이너화
- 환경 독립적 실행 구조

Diet Agent의 동작 흐름

4. 시스템 구조 개요



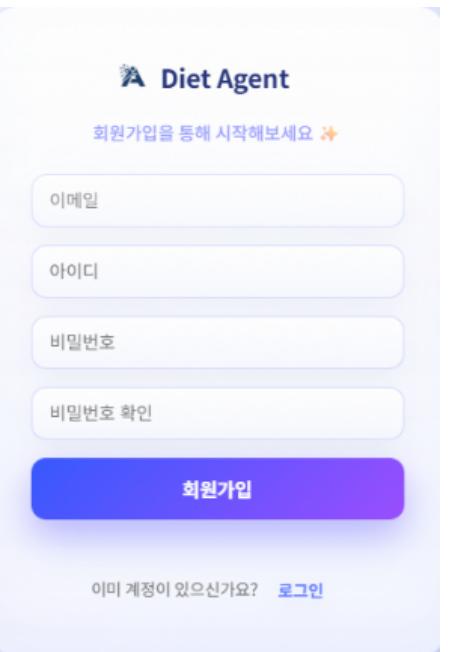
5. 주요 기능 상세 설명

5.1 회원가입 및 로그인

5.1 회원가입 및 로그인

- 이메일 및 비밀번호 기반 회원가입
- 로그인 시 JWT Access Token 발급
- 토큰을 이용한 사용자 인증 유지
- /api/user/me API를 통해 로그인 상태 복구

<user_routes.py>

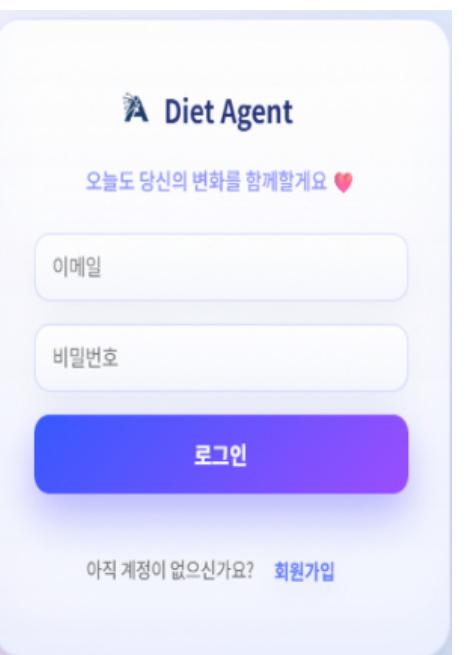


```
# 회원가입
@user_bp.route('/signup', methods=['POST'])
def signup():
    data = request.get_json() or {}
    email = (data.get('email') or '').strip()
    password = (data.get('password') or '').strip()

    if not email or not password:
        return jsonify({'message': '이메일과 비밀번호를 입력해주세요.'}), 400

    conn = get_connection()
    cur = conn.cursor()

    cur.execute("SELECT id FROM users WHERE email = ?", (email,))
    if cur.fetchone():
        conn.close()
        return jsonify({'message': '이미 가입된 이메일입니다.'}), 400
```



```
# jwt 발급 (identity = user_id) -> Access + Refresh
user_id = row['id']
role = row['role']
access_token = create_access_token(
    identity=str(user_id),
    expires_delta=timedelta(hours=1),
    additional_claims={"role": role}
)
refresh_token = create_refresh_token(
    identity=str(user_id),
    additional_claims={"role": role}
)

return jsonify({
    "accessToken": access_token,
    "refreshToken": refresh_token,
    "role": role
}), 200
```

```
# 내 정보 조회
@user_bp.route('/me', methods=['GET'])
@jwt_required()
def get_my_info():
    # identity는 문자열로 저장되어 있으므로 DB 조회 전 int로 변환
    try:
        user_id = int(get_jwt_identity())
    except Exception:
        return jsonify({'message': 'токен 정보가 올바르지 않습니다.'}), 401
```

로그인 시 JWT 에 저장한 사용자 정보 조회
-> 재로그인시 로그인 상태 복구

유효한 accessToken이 있으면 로그인된 상태로 간주
-> 사용자 인증 유지

5. 주요 기능 상세 설명

5.2 JWT 인증 구조

5.2 JWT 인증 구조

- Access Token 기반 인증(user_routes.py / Axios.jsx)
- Authorization Header에 Bearer Token 사용(Axios.jsx)
- 보호된 페이지 접근 시 토큰 검증(JWT 라이브러리 / user_routes.py)

backend

```
access_token = create_access_token(  
    identity=str(user_id),  
    expires_delta=timedelta(hours=1),  
    additional_claims={"role": role}  
)
```

user_routes.py에서 accessToken 생성
->로그인 후 localStorage에 저장(Axios.jsx)

frontend

```
import axios from "axios";  
  
/* ======  
| axios 기본 설정 -> 각 페이지에서 임포트 후 api.get or api.post 등으로 사용 가능  
===== */  
const api = axios.create({  
  baseURL: "http://127.0.0.1:5000",  
  withCredentials: true  
};  
  
// 요청 보낼 때마다 자동으로 토큰 붙임  
api.interceptors.request.use(config) => {  
  const token = localStorage.getItem("accessToken");  
  // 토큰이 "undefined"/"null" 문자열로 저장되는 케이스 방지  
  if (token && token !== "undefined" && token !== "null") {  
    // axios 버전에 따라 headers가 없을 수 있어 안전 처리  
    config.headers = config.headers || {};  
    config.headers.Authorization = `Bearer ${token}`;  
  }  
  return config;  
};  
  
export default api;
```

Axios.jsx: localStorage에 저장된 accessToken
-> 이후 모든 API 요청에서 재사용

```
# 내 정보 조회  
@user_bp.route('/me', methods=['GET'])  
@jwt_required()  
def get_my_info():  
  # identity는 문자열로 저장되어 있으므로 DB 조회 전 int로 변환  
  try:  
    user_id = int(get_jwt_identity())  
  except Exception:  
    return jsonify({'message': 'токن 정보가 올바르지 않습니다.'}), 401
```

user_routes.py에서

@jwt_required(): 보호되어야 하는 API 명시

flask_jwt_extended 라이브러리를 통해
Header 파싱, Bearer 타입 확인, JWT 디코딩, 서명 검증

Authorization Header에 Bearer Token 더해짐
->api.get("/api/user/me") 요청시
인증 헤더가 자동 포함됨

5. 주요 기능 상세 설명

5.3 체중 기록 기반 진행률 계산

5.3 체중 기록 기반 진행률 계산

현재 체중 및 목표 체중, 달성 기간 등을 입력하면 체중 기록 대비 진행률이 계산됨

시작, 목표, 현재 체중을 기반으로 감량/증량/유지 목표를 모두 지원하는 체중 변화 진행률 계산 로직

(추후 히스토리 기반 체중 변화 진행률 계산까지도 확장 가능)



```
const openGoalModal = () => {
  setGoalEditForm({
    type: effectiveGoal.type,
    startWeight: effectiveGoal.startWeight,
    targetWeight: effectiveGoal.targetWeight,
    calories: effectiveGoal.calories,
    protein: effectiveGoal.protein,
    activityKcal: effectiveGoal.activityKcal,
    startDate: effectiveGoal.startDate,
    endDate: effectiveGoal.endDate,
  });
  setIsGoalModalOpen(true);
};

const saveGoalFromModal = async () => {
  try {
    const payload = {
      type: goalEditForm.type,
      startWeight: Number(goalEditForm.startWeight),
      targetWeight: Number(goalEditForm.targetWeight),
      targetCalories: Number(goalEditForm.calories),
      targetProtein: Number(goalEditForm.protein),
      targetActivityKcal: Number(goalEditForm.activityKcal),
      startDate: goalEditForm.startDate,
      endDate: goalEditForm.endDate,
    };
    const res = await api.put("/api/diet/goal", payload);
    // PUT 응답은 {goal:{...}}
    const g = res?.data?.goal || payload;
  } catch (err) {
    console.error(err);
  }
};
```

입력한 목표 및 달성 기간을 저장하는 변수
(MyInfoPage.jsx)

```
def _calc_time_progress(start_date: str, end_date: str) -> int:
  try:
    s = datetime.fromisoformat(start_date)
    e = datetime.fromisoformat(end_date)
  except Exception:
    return 0
  if s >= e:
    return 0
  now = datetime.utcnow()
  total = (e - s).total_seconds()
  passed = min(max((now - s).total_seconds(), 0.0), total)
  return int(round((passed / total) * 100)) if total else 0

def _calc_weight_progress(goal_type: str, start_w, target_w, current_w) -> int:
  try:
    s = float(start_w)
    t = float(target_w)
    c = float(current_w)
  except Exception:
    return 0

  if goal_type == "loss":
    if s <= t:
      return 0
    denom = s - t
    numer = s - c
    return max(0, min(100, int(round((numer / denom) * 100)))))

  if goal_type == "gain":
    if t <= s:
      return 0
    denom = t - s
    numer = c - s
    return max(0, min(100, int(round((numer / denom) * 100))))
```

체중 변화 진행률 계산 로직(기간/실제 체중 분리하여 고려)
(diet_routes.py)

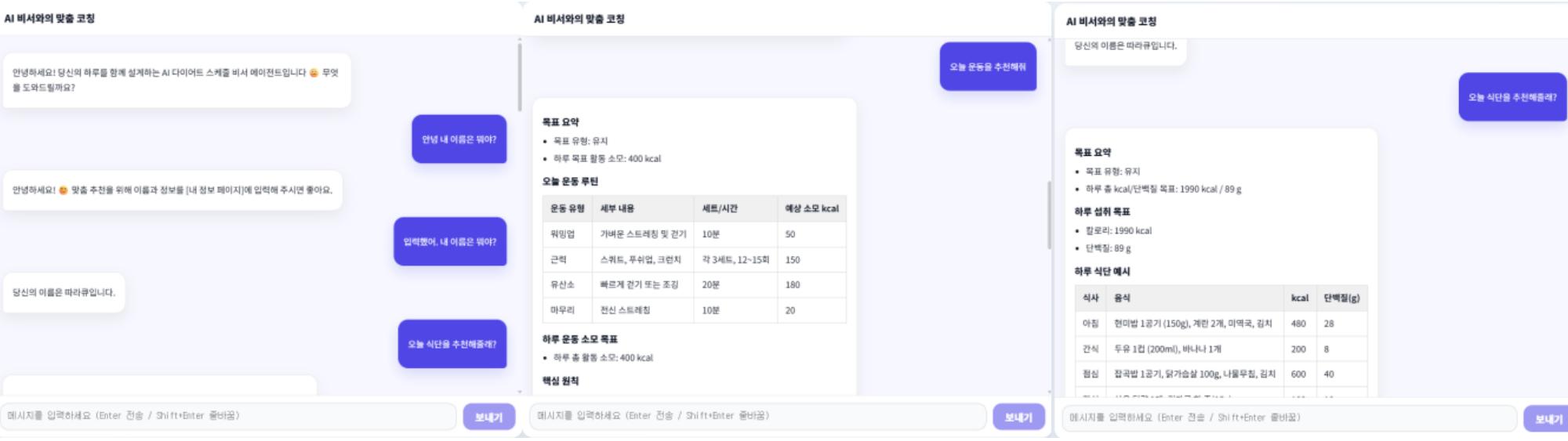
5. 주요 기능 상세 설명

5.4 AI 챗봇

5.4 AI 챗봇

최근 체중 변화, 식단/운동 선호 및 비선호, 일정 정보, 이전 대화 기록 등을 종합하여 상황에 맞는 답변을 제공

(KEY 인증을 위한 .env 파일은 보안상의 이유로 별도 관리가 필요하며,
추후 DB에 주간 기록을 저장하여 보다 더 개인화 할 수 있도록 확장이 가능)



저장된 정보를 로드하여 식단 및 운동을 추천하며,
사용자의 키와 몸무게, 연령과 성별 등을 고려한 목표 섭취 칼로리 및 단백질도 계산

```
# 사용자의 대화 의도 판단 함수
def ai_engine.py
def detect_intent(message: str) -> str:
    text = message.lower().strip()

    # 내 정보 질문 (최우선)
    if any(k in text for k in [
        "내 키", "키는", "신장은",
        "내 몸무게", "몸무게는", "체중은",
        "내 이름", "이름은",
        "내 목표", "목표는",
        "내 칼로리", "섭취 칼로리", "섭취칼로리",
        "단백질은", "단백질 목표",
        "활동 소모", "운동 소모", "소모 칼로리"
    ]):
        return "info_request"

    # 음식 영양 정보
    if any(k in text for k in ["칼로리", "단백질"]) and "식단" not in text:
        return "food_nutrition"

    if any(k in text for k in ["포함", "넣어서", "같이", "먹을건데"]) and "식단" in text:
        return "diet_with_food"

    if any(k in text for k in ["대신", "알고", "다른"]):
        return "diet_food_specific"

    if "식단" in text:
        return "diet"

    if any(k in text for k in ["운동", "루틴"]):
        return "exercise"
```

프롬프트, 키워드 로직 등 설계
(ai_engine.py)

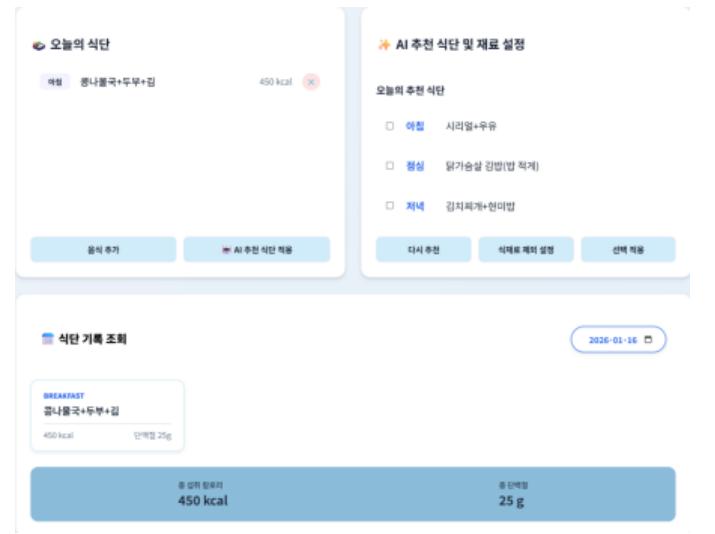


사용자의 정보가 없는 경우의 답변

식단 및 운동 외 자유 주제로도 대화 가능

5. 주요 기능 상세 설명

5.5 식단 및 운동 추천 시스템



```
# AI 추천 생성
@diet_bp.route("/ai/recommend", methods=["POST"])
@jwt_required()
def create_ai_recommendation():
    user_id = get_jwt_identity()

    # 여기서:
    # - user 정보 조회
    # - AI 호출
    # - ai_recommended_meals + items 저장

    return jsonify({"message": "ai recommendation created"})

# AI 추천 - 오늘의 식단 적용
@diet_bp.route("/ai/apply", methods=["POST"])
@jwt_required()
def apply_ai_meal():
    user_id = get_jwt_identity()
    data = request.json # ai_meal_id, date, mode
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    conn = get_connection()
    cur = conn.cursor()
```

```
# GET /api/preferences/food/
@pref_bp.route("/food", methods=["GET"])
@jwt_required()
def get_food_pref():
    user_id = get_jwt_identity()
    row = get_food_preferences(user_id)
    if not row:
        return jsonify({"likes": "", "dislikes": "", "allergies": ""})
    return jsonify([
        "likes": row["likes"] or "",
        "dislikes": row["dislikes"] or "",
        "allergies": row["allergies"] or ""
    ])

# PUT /api/preferences/food/
@pref_bp.route("/food", methods=["PUT"])
@jwt_required()
def put_food_pref():
    user_id = get_jwt_identity()
    data = request.get_json() or {}
    likes = data.get("likes") or ""
    dislikes = data.get("dislikes") or ""
    allergies = data.get("allergies") or ""
    save_food_preferences(user_id, likes, dislikes, allergies)
    return jsonify({"message": "음식 선호가 저장되었습니다."})
```

```
@diet_bp.route("/excluded", methods=["PUT"])
@jwt_required()
def upsert_diet_excluded():
    user_id = get_jwt_identity()
    data = request.get_json() or []
    items = (data.get("items") or "").strip()

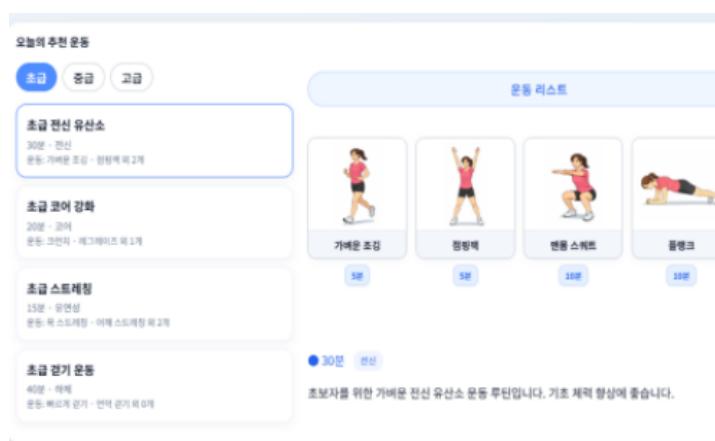
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        """
        INSERT INTO diet_exclusions (user_id, items, updated_at)
        VALUES (?, ?, ?)
        ON CONFLICT(user_id) DO UPDATE SET
            items=excluded.items,
            updated_at=excluded.updated_at
        """,
        (user_id, items, datetime.utcnow().isoformat())
    )
    conn.commit()
    conn.close()

    return jsonify({"message": "제외 식재료가 저장되었습니다.", "items": items})
```

자동식단추천(diet_routes.py)

선호/비선호 메뉴(preference_routes.py), 제외 메뉴(diet_routes.py) 저장

(제외 메뉴: 식단 생성 시 강제 필터링 조건이기 때문에 다이어트 목표 도메인과 함께 분리해 관리)



```
@workout_bp.route("/workouts", methods=["GET"])
def get_workouts():
    level = request.args.get('level') # 'beginner', 'intermediate', 'advanced'
    all_workouts = get_all_workouts() # This will come from a service/data source

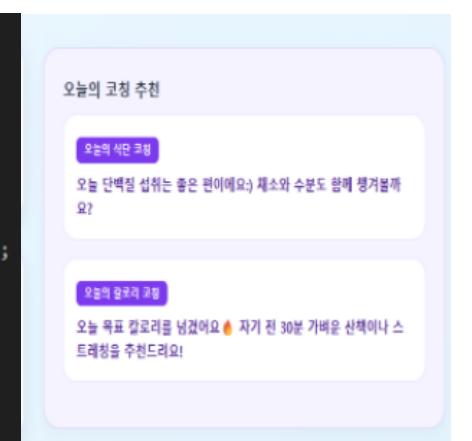
    if level:
        filtered_workouts = [w for w in all_workouts if w['level'].lower() == level.lower()]
    else:
        filtered_workouts = all_workouts

    formatted_workouts = []
    for w in filtered_workouts:
        formatted_workouts.append({
            "id": w['id'],
            "level": w['level'],
            "title": w['title'],
            "duration": w['duration'],
            "part": w['part'],
            "exercises": w['exercises'], # Assuming this is already a list of strings
            "videourl": w['videourl'], # Assuming this is the full YouTube URL
            "description": w['description']
        })

    return jsonify(formatted_workouts)
```

운동추천(workout_routes.py)

```
const calorieCoachMessage = useMemo(() => {
  if (!token) {
    return "로그인 후 칼로리 기반 코칭을 받을 수 있어요.";
  }
  if (!calorieSummary) {
    return "오늘 섭취한 식단을 기록하면 칼로리 코칭이 제공돼요 😊";
  }
  if (intakeKcal >= targetKcal) {
    return "오늘 목표 칼로리를 넘겼어요! 자기 전 30분 가벼운 산책이나 스트레칭을 추천드려요!";
  }
  if (percentKcal >= 80) {
    return "오늘 칼로리 섭취는 목표에 잘 맞고 있어요 :)";
  }
  return "아직 칼로리에 여유가 있어요! 균형 잡힌 간식 정도는 괜찮아요 😊";
}, [token, calorieSummary, intakeKcal, targetKcal, percentKcal]);
```



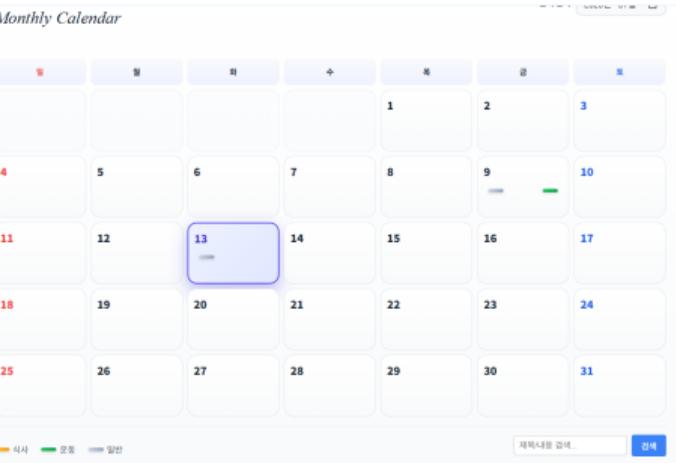
오늘의 코칭 시리즈(MainPage.jsx)

5. 주요 기능 상세 설명

5.6 캘린더 일정 연동

5.6 캘린더 일정 연동

- 날짜별 식단 및 운동 일정 관리
- 운동과 소모 칼로리의 정밀한 기록을 위해 일정 메모와도 연동



```
# 캘린더 조회
@calendar_bp.route("", methods=["GET"])
@jwt_required()
def get_calendar():
    user_id = get_jwt_identity()
    month = request.args.get("month")

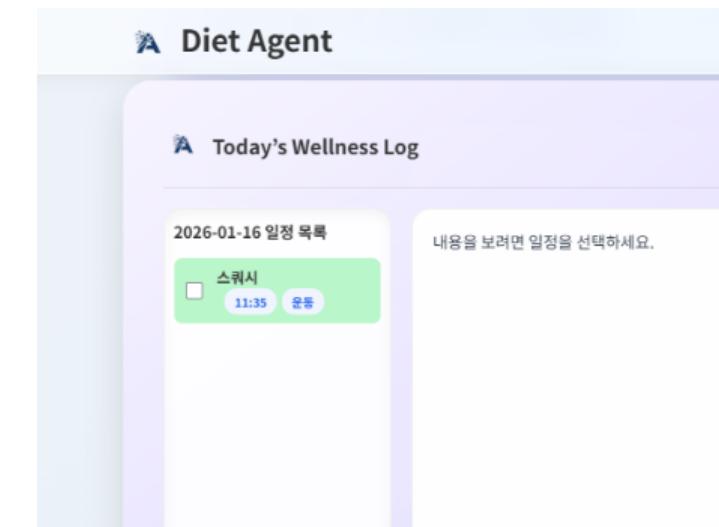
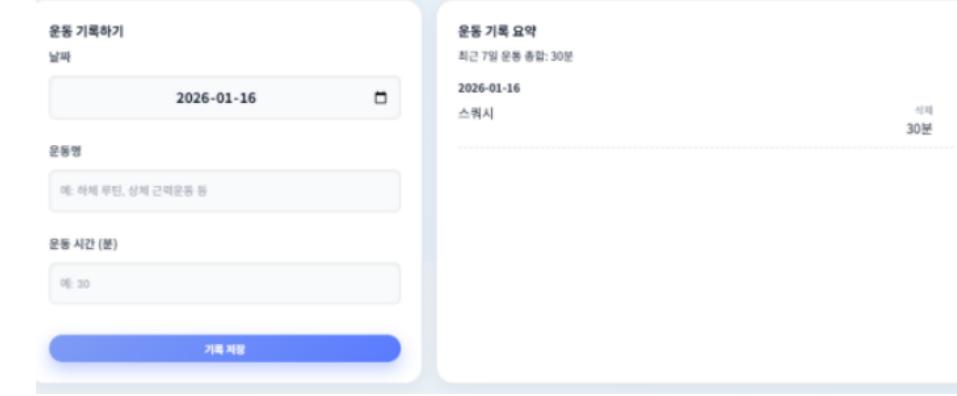
    if not month:
        return jsonify({"message": "month가 필요합니다."}), 400

    conn = get_connection()
    cur = conn.cursor()

    # 식단
    cur.execute(
        """
        SELECT date, menu, calories
        FROM diet_recommendations
        WHERE user_id = ? AND date LIKE ?
        """,
        (user_id, f"%{month}%")
    )
    diets = cur.fetchall()

    # 운동
    cur.execute(
        """
        SELECT date, workout, calories
        FROM workout_recommendations
        WHERE user_id = ? AND date LIKE ?
        """,
        (user_id, f"%{month}%")
    )
    workouts = cur.fetchall()
```

캘린더 일정 구현(calendar_routes.py)



```
useEffect(() => {
  if (!userId) return;

  const fetchWorkoutLogs = async () => {
    try {
      const response = await api.get("/api/schedule/items", {
        params: { kind: "운동" } // Fetch only workout type schedules
      });
      // Transform schedule items to workout log format
      const fetchedLogs = (response.data.items || []).map(item => ({
        id: item.id,
        date: item.date,
        name: item.title,
        minutes: parseInt(item.memo) || 0, // Assuming memo contains minutes as a number string
      }));
      setLogs(fetchedLogs);
    } catch (e) {
      console.error("Failed to fetch workout logs from backend:", e);
      setLogs([]);
    }
  };

  fetchWorkoutLogs();
}, [userId]); // Depend on userId
```

운동 일정 연동(WorkOutPage.jsx)

5. 주요 기능 상세 설명

5.7 커뮤니티

5.7 커뮤니티

-관리자의 공지 게시 기능

-사용자의 문의 사항 답변 기능(관리자)

<community_routes.py>

```
# 공지사항 API
# -----
@community_bp.route("/notices", methods=["GET"])
def get_notices():
    """공지사항 리스트 조회"""
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        """
        SELECT id, title, created_at, view_count
        FROM notices
        ORDER BY datetime(created_at) DESC
        """
    )
    rows = cur.fetchall()
    conn.close()

    notices = [
        {
            "id": row["id"],
            "title": row["title"],
            "createdAt": row["created_at"],
            "viewCount": row["view_count"]
        }
        for row in rows
    ]
    return jsonify({"notices": notices})
```

공지사항 API

관리자: 공지 등록·수정·삭제

사용자: 조회

문의하기

```
# 문의 API
# -----
@community_bp.route("/inquiries", methods=["POST"])
@jwt_required()
def create_inquiry():
    """문의 작성"""
    user_id = _jwt_user_id_int()
    if user_id is None:
        return jsonify({"message": "인증 정보를 확인할 수 없습니다."}), 401

    data = request.get_json() or {}
    title = (data.get("title") or "").strip()
    content = (data.get("content") or "").strip()

    if not title or not content:
        return jsonify({"message": "제목과 내용을 모두 입력해주세요."}), 400
```

문의 API

JWT 인증을 거쳐 확인된 사용자가 문의를 작성

```
@community_bp.route("/admin/inquiries/<int:inquiry_id>", methods=["GET"])
@admin_required
def admin_get_inquiry_detail(inquiry_id):
    """관리자: 문의 상세"""
    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        """
        SELECT i.id, i.user_id, u.email as user_email, i.title, i.content, i.status, i.answer, i.created_at, i.
        answered_at
        FROM inquiries i
        LEFT JOIN users u ON u.id = i.user_id
        WHERE i.id = ?
        """,
        (inquiry_id,),
    )
    row = cur.fetchone()
    conn.close()
    if not row:
        return jsonify({"message": "해당 문의를 찾을 수 없습니다."}), 404
```

관리자: DB에서 문의를 조회

```
@community_bp.route("/admin/inquiries/<int:inquiry_id>/answer", methods=["POST"])
@admin_required
def admin_answer_inquiry(inquiry_id):
    """관리자: 문의 답변"""
    data = request.get_json() or {}
    answer = (data.get("answer") or "").strip()
    if not answer:
        return jsonify({"message": "답변 내용을 입력해주세요."}), 400

    conn = get_connection()
    cur = conn.cursor()
    cur.execute(
        """
        UPDATE inquiries
        SET answer = ?, status = '답변완료', answered_at = ?
        WHERE id = ?
        """,
        (answer, datetime.now().isoformat(timespec="seconds"), inquiry_id),
    )
    conn.commit()
    updated = cur.rowcount
    conn.close()
    if updated == 0:
        return jsonify({"message": "해당 문의를 찾을 수 없습니다."}), 404
    return jsonify({"message": "답변이 등록되었습니다."})
```

관리자의 답변이 DB에 저장되고 UI에서도 확인됨

6. 유ти리티 및 안정성 설계

6.1 입력값 안전 처리(실제 서비스 환경을 고려한 안정성 강화)

- 빈 값, 문자열, 숫자 입력에 대한 방어 처리(500 예방)
- SQL Injection 방어
- DB 조회 결과 None 처리

```
data = request.get_json() or {}
```

auth_routes.py,
diet_routes.py,
preferences_routes.py,
community_routes.py,
chatbot_routes.py 등의
모든 POST / PUT API

JSON Body가 없을 때 None 반환

->빈 요청이어도 None으로 처리하여 500 예러 예방

```
cur.execute(  
    "SELECT * FROM today_meals WHERE user_id=? AND date=?",
    (user_id, today),
)
```

DB 접근 코드(models) 전반

: 문자열을 직접 삽입하지 않고 파라미터로 바인딩

-> 데이터 유출 및 변조 위험을 근본적으로 차단하여
SQL Injection(사용자 입력값이 SQL 문으로 실행되도록
조작하는 공격) 방어

```
row = get_food_preferences(user_id)
if not row:
    return jsonify({"likes": "", "dislikes": "", "allergies": ""})
return jsonify({
    "likes": row["likes"] or "",
    "dislikes": row["dislikes"] or "",
    "allergies": row["allergies"] or ""
})
```

preference_routes.py

: 데이터 없을 때, 프론트에서 row.likes 접근 시 오류 방지

```
def _calc_weight_progress(goal_type: str, start_w, target_w, current_w) -> int:
    try:
        s = float(start_w)
        t = float(target_w)
        c = float(current_w)
    except Exception:
        return 0
```

체중·진행률 계산(diet_routes.py)

: None / 빈 값 / 프론트 오류 전송 등의 경우로
계산 불가능 시 안전하게 0% 반환하여 500 오류 예방

7. 트러블 슈팅 경험

7.1 로그인 성공 후 화면 이동 실패 문제

원인: 인증 로직과 네비게이션이 여러 컴포넌트에 분산되어 있음

해결: App.jsx와 RequireLogin 역할 분리

<App.jsx>

```
<Route path="/login" element={<LoginPage onLoginSuccess={handleLoginSuccess} />} />
```

```
const handleLoginSuccess = (accessToken) => {
  // 1) 토큰 저장
  localStorage.setItem("accessToken", accessToken);

  // 2) ✓ 바로 메인으로 이동(폴리로드 없이) - 유저가 체감상 "로그인 성공 후 안 넘어감" 방지
  //     /me 조회는 백그라운드에서 처리하고, 실패하면 다시 로그인으로 보냄
  setLoggedIn(true);
  navigate("/main", { replace: true });

  // 3) 유저 정보 로드(/me)
  api
    .get("/api/user/me", {
      headers: { Authorization: `Bearer ${accessToken}` },
    })
    .then((res) => {
      setUser(res.data);
      if (res.data?.id) localStorage.setItem("userId", String(res.data.id));
    })
    .catch(() => {
      // 토큰이 있어도 /me 실패 시 안전하게 로그아웃 처리
      localStorage.removeItem("accessToken");
      localStorage.removeItem("userId");
      localStorage.removeItem("role");
      setUser(null);
      setLoggedIn(false);
      navigate("/login", { replace: true });
    });
};
```

로그인 성공 이후의 모든 인증 흐름과 라우팅을
App.jsx의 handleLoginSuccess로 일원화

```
<Route path="/main" element={<RequireLogin><MainPage /></RequireLogin>} />
```

```
useEffect(() => {
  const id = localStorage.getItem("accessToken");
  if (!id) {
    navigate("/login", { replace: true });
  } else {
    setAuthed(true);
  }
  setChecked(true);
}, [navigate]);

if (!checked) return null;
return authed ? children : null;
}
```

보호된 라우트(RequireLogin)에서는 토큰 존재 여부만
검사하도록 역할을 분리

7. 트러블 슈팅 경험

7.2 챗봇 응답 Error 발생 및 부자연스러운 언어 처리

원인: 언어 처리 모델의 순서 및 구조 이상

해결: 로직 구성 재정렬 및 프롬프트 구체화

<ai_engine.py>

```
# ai_engine.py
def generate_reply(user_id: int, message: str) -> str:
    # ✓ 인사 처리 (최우선)
    # ✓ 인사 처리 (최우선)
    if is_greeting(message):
        # 사용자 상태 조회
        state = _get_state(user_id)
        name = state.get("name")

        if name:
            return f"안녕하세요, {name}님! 😊 오늘도 활기차게 하루를 시작해볼까요?"
        else:
            return (
                "안녕하세요! 😊\n"
                "맞춤 추천을 위해 이름과 정보를 [내 정보 페이지]에 입력해 주시면 좋아요."
            )

    intent = detect_intent(message)

    meta_intent = detect_meta_intent(message)

    # ✓ META_INTENT 처리 (intent보다 우선)
    if meta_intent == "CONFIRM_MEMORY":
        return (
            "현재는 정보를 불러오는 기능만 제공하고 있고,\n"
            "대화 내용을 자동으로 저장하는 기능은 아직 준비 중이에요."
        )
```

응답 로직

```
# 프롬프트 선택
# -----
# 1) 특정 음식 대체 추천
if intent == "diet_food_specific":
    food_to_replace = message # 예: "그릭 요거트"
    system_prompt = """
너는 헬스·식단 전문가다.
사용자가 말한 음식을 대신할 건강한 간식을 한 가지 추천해줘.
추천 이유나 칼로리/단백질도 함께 알려줘.
출력은 Markdown이나 표 필요 없고, 간단히 한 문장으로.
"""
    user_prompt = f"{food_to_replace} 대신 먹을 간식 추천해줘."

# -----
# 2) 특정 음식 칼로리/단백질 확인
elif intent == "food_nutrition":
    food_to_check = extract_food_name(message)
    system_prompt = """
너는 헬스·영양 전문가다.
사용자가 말한 음식의 일반적인 1인분 기준
칼로리(kcal)와 단백질(g)을 알려줘.
목표 칼로리와는 절대 연결하지 마라.
출력은 한 문장.
"""
    user_prompt = f"{food_to_check}의 칼로리와 단백질을 알려줘.

# -----
# 3) 기존 식단 추천
```

프롬프트

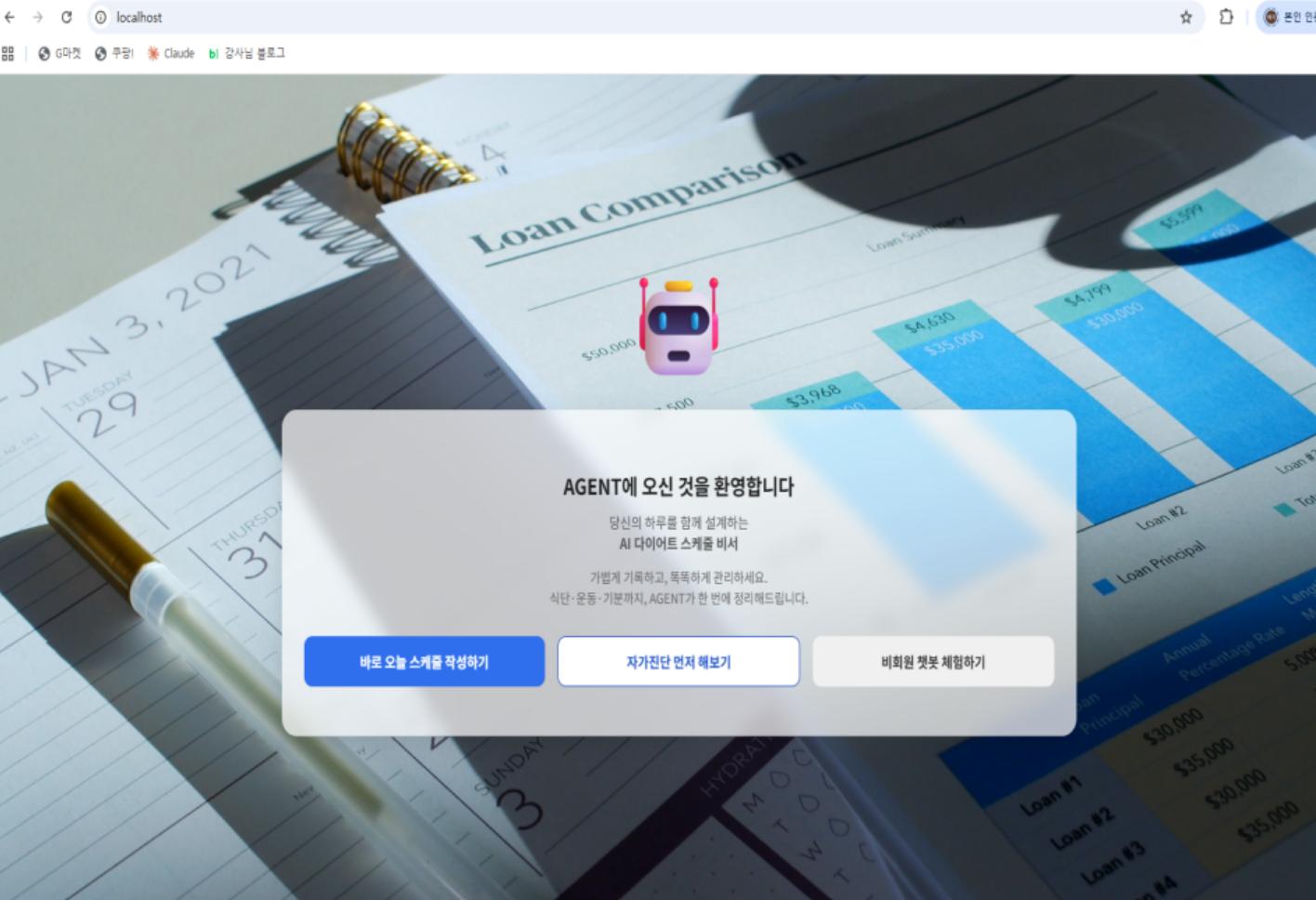
8. 배포 (Docker)

Docker를 활용하여 서버를 컨테이너화

-환경 의존성 문제 해결

-배포 및 실행 단순화

(단, 실행시 .env 파일에 사용자의 KEY를 별도로 입력해야 함)



Name	Tag	ID
sprout-app	latest	1
lhj8/flask-sprout	1.0	3
lhj8/flask-pybo	1.0	2
lhj8/ai_agent_diet_scheduler_backend	latest	8
lhj8/ai_agent_diet_scheduler_frontend	latest	0

9. 프로젝트 성과 및 결론

Diet Agent 프로젝트 성과 및 결론

9.1

프로젝트 성과

- React + Flask 기반 풀스택 개발 경험
- JWT 인증 구조 직접 설계 및 구현
- DB 중심 서비스 아키텍처 이해
- 실제 서비스 관점의 에러 처리 및 안정성 설계
- Docker 기반 배포 경험

9.2

결론

Diet Agent는 단순한 기능 나열이 아닌, “사용자 데이터를 이해하고 함께 관리하는 서비스”를 목표로 개발된 프로젝트임

9.3

의의

기획 -> 설계
-> 개발 -> 배포까지 전 과정을 직접 수행하며, 실제 서비스 개발에 필요한 역량을 종합적으로 경험할 수 있었음

10. 팀원별 소감

*김요원

-이번 프로젝트를 통해 React, Flask, DB가 연동되어 하나의 기능이 완성되는 전체 흐름을 코드 단위로 이해할 수 있었습니다. 특히 프론트엔드와 백엔드가 유기적으로 연결되어 동작하는 과정을 직접 경험하며, 서비스 구조를 고려한 개발의 중요성을 느꼈습니다. 또한, 팀원들과의 역할 분담과 커뮤니케이션을 통해 문제를 해결해 나가는 과정에서 협업의 중요성을 경험할 수 있었습니다. 비록 AI 챗봇 기능은 기본적인 구현에 그쳐 아쉬움이 남지만, 향후 추가 학습을 통해 응답 로직을 더 개선하고 싶습니다.

*정두균

-Diet Agent 프로젝트를 통해 React 기반 프론트엔드 화면 구현뿐만 아니라, 사용자 흐름을 고려한 UI/UX 설계의 중요성을 경험했습니다. 로그인부터 운동·일정 관리까지 전체 화면을 직접 구성하며, 컴포넌트 재사용과 디자인 일관성을 유지하는 방법을 익혔고, DB 테이블과 컬럼 설계에 참여하며 화면과 데이터 구조의 연결을 이해할 수 있었습니다.

*이혜지

-React, Flask, MySQL, JSON 등을 종합하여 하나의 플랫폼을 구현해본 경험이 많은 공부가 되었습니다. 그리고 제 자신의 분량 뿐 아니라 프로젝트의 전체를 볼 수 있으면서 의견 및 해결 방안을 제시하는 자세가 정말 필요하겠구나 하고 생각이 들었습니다. 그리고 계속해서 작업을 하면 할수록 보완할 점들이 많이 보여서 틈틈히 업그레이드를 해야겠다고 느꼈습니다. 이후에 실제 업무에서는 이 프로젝트보다 훨씬 복잡한 구조를 가질 것이고, 팀원들끼리 역할 분담도 쉽지 않겠지만, 어떤 상황에서도 보다 능동적으로 임하겠습니다.

*박종훈

-팀원분들께서 넘 출중하셔서 사실 제가 한일이 많이 없긴 하지만 뭔가 많이 배우고 보람찬 시간이였던것 같습니다.

Thank You