

[하루핏] 최종보고서

팀명
고하이 (Go-High!)
팀장
구하영
팀원
고진석, 이가원
개발주제
AI 기반 운동·식단 개인 맞춤 관리 웹 서비스

목차

제1장 개요	4
1.1 프로젝트 개요	4
1.1.1. 배경	4
1.1.2. 목적 및 필요성	4
1.1.3. 기대효과	4
1.2 개발 주제 및 범위	4
1.2.1. 개발 주제	4
1.2.2. 주요 기능 요약	4
1.2.3. 사용 대상	4
1.2.4. 범위 및 특징	4
1.3 관련 기술 및 도구	5
1.3.1. 사용 기술 스택	4
1.3.2. 주요 라이브러리 및 API	4
1.4 업무분장 및 일정계획표	6
 제2장 분석 공정	 10
2.1 수요 분석	10
2.1.1. 요구사항 분석	10
2.2 유저 시나리오	10
2.2.1. 이용자	10
2.3 데이터 흐름 및 구조	13
2.3.1. 개체-관계 도표 ERD	13
2.3.2. 주요 데이터 구조 설명	13
2.4 프로세스 분석	20
2.4.1. 페이지 구조(화면 설계 요약)	20
2.4.2. 자료 흐름도(Data Flow Diagram)	20
2.4.3. 핵심 기능 목록	22
2.4.3. 기능 명세서	22

제3장 설계 공정	30
3.1 시스템 아키텍처	30
3.1.1. 전체 구성도	30
3.1.2. 기술 환경 요약	30
3.1.2. 기술 흐름도	30
3.2 자료 설계	31
3.2.1. 주요 데이터 항목	31
3.2.2. 테이블/컬렉션 구성	34
3.3 코드 설계	37
3.3.1. 구조 차트(Structure Chart)	37
3.3.2. 모듈 개요도	38
3.3.3. 모듈 설명서	39
3.3.4. 엔티티-프로세스 매트릭스	48
3.4 핵심 기능	50
3.5 인터페이스 설계	61
 제4장 구현 공정	 82
4.1 프로그램 작성	82
4.1.1. 코딩 컨벤션 (Back)	82
4.1.2. 코딩 컨벤션 (Front)	83
4.2 시 험	83
4.2.1. 기능 시험	84
4.3 프로그램 운영	84
4.3.1. 프로그램 사용법	84
 제5장 배포	 88
 제6장 결 론	 96
6.1 문제점 및 개선방안	96
6.2 결론	96

제1장 개요

1.1 목적 및 기대효과

1.1.1. 배경

- AI 활용에 대한 관심도 향상으로, 생활 전반의 영역에 걸쳐 AI에게 솔루션을 얻고 많은 부분을 자동화 하고자 하는 수요가 증가하고 있다. 이를 건강 관리에 접목 시켜, 챗봇 형태를 도입한 개인 맞춤형 운동/식단 AI 코치 서비스를 기획했다.

1.1.2. 목적 및 필요성

- 운동 및 식단을 통합 관리할 수 있는 개인 맞춤형 시스템을 개발한다.
- OpenAI 기반 자연어 인터페이스를 활용한 직관적이고 간편한 사용자 경험을 제공한다.
- 사용자 이력 기반의 연속성 있는 피드백 제공 및 히스토리를 관리한다.

1.1.3. 기대효과

- **손쉬운 접근성:** 복잡한 회원가입 절차가 필요 없이 닉네임/PIN번호를 기반으로 사용자를 구별하여 높은 접근성을 제공한다.
- **개인화된 경험:** LLM이 사용자의 과거 이력을 학습하고 기억함으로써, 개인의 성향과 라이프 스타일에 맞는 맞춤형 피드백을 제공한다. 이는 딱딱하고 관리 위주의 시스템이 아닌, 마치 친근한 개인 트레이너처럼 느껴지는 앱 이미지를 구축하여 사용자의 건강 관리 동기를 지속적으로 부여한다.
- **유연한 맞춤형 코칭:** 사용자의 건강 관리 목표와 참여도에 따라 다양한 모드(Mode)를 설정할 수 있다. 각 모드에 최적화된 LLM 프롬프트 세팅을 통해 사용자들은 자신의 상황에 맞는 심화된 코칭을 경험할 수 있다.
- **효율적인 데이터 관리:** Open API를 활용하여 운동 및 식단 관련 데이터를 효율적으로 관리하고 제공함으로써, 서비스의 확장성과 유연성을 높이고, 최신 건강 데이터를 실시간으로 반영하여 사용자에게 정확하고 시의적절한 정보를 제공할 수 있게 한다.

1.2 개발 주제 및 범위

1.2.1. 개발 주제

- [하루핏]은 AI 기반의 개인 맞춤형 운동 및 식단 코칭 서비스를 개발하여 사용자들이 건강한 라이프 스타일을 지속 가능하게 유지할 수 있도록 돕는 것을 목표로 한다. 특히 자연어 처리(NLP)와 거대 언어 모델(LLM)을 활용하여 사용자에게 최적화된 운동 루틴과 식단 계획을 제안하고, 이를 친근한 챗봇 형태로 제공하는 개인 맞춤형 건강 관리 서비스를 개발한다. 사용자의 목표, 신체 정보, 식습관, 운동 선호도 등을 종합적으로 분석하여, 단순히 정보를 나열하는 것을 넘어 실질적인 행동 변화를 유도하는 코칭을 제공하는 것이 핵심이다.

1.2.2. 사용 대상자

- 운동이나 식단 관리를 시작하고 싶지만 방법을 모르거나 꾸준히 실천하기 어려운 이용자
- 개인 맞춤형 코칭을 받기 원하는 이용자
- AI 기술 기반의 편리하고 혁신적인 서비스를 경험 하고 싶은 사용자

1.2.3. 범위 및 특징

- Windows 11 기반의 PC , mac OS
- 자체적으로 구축한 MongoDB를 통해 사용자 관련 데이터를 효율적으로 저장 및 관리할 뿐 아니라, OpenAI API를 활용하여 최신 정보의 신뢰성과 서비스 데이터의 다양성을 확보합니다. 추후에는 외부 공공/상업 Open API(예: 영양 성분 데이터, 운동 종류 데이터 등)를 적극적으로 활용하여 서비스의 지속적인 확장과 업데이트에 용이한 구조를 제공할 수 있습니다.

1.3 관련 기술 및 도구

1.3.1. 사용 기술 스택

- Frontend
- React
- JavaScript(ES6)
- Axios
- HTML5, CSS
- Backend
- Node.js
- Express.js
- 배포 및 인프라
- Render
- git, github

1.3.2. 주요 라이브러리 및 API

- OpenAI API
- Mongoose (MongoDB ORM for Node.js)
- Axios / Fetch API
- JWT (JSON Web Tokens)

1.4 업무분장 및 일정계획표

구 분		기간	2025년 7월 ~ 2025 7월		
업무명(모듈명)		담당자	2 주	3 주	4 주
1.0 기획	1.1 아이디어 구상	공동	→		
	1.2 기획서 작성	공동	→		
2.0 분석	2.1 분석_조사	공동	→		
	2.2. 데이터 베이스 설계	고진석		→	
3.0 Front	3.1 인덱스 페이지	3.1.1 - 공통 스타일 및 UI 컴포넌트 - 로그인 화면 - 모드 선택 뷰 이가원	→		
	3.2 메인 페이지	3.2.1 - 메인 페이지 대시보드 - 식단 관리 기본 페이지 - 운동 관리 기본 페이지 - 상태 관리 기본 페이지 - 달력 기본 페이지 팀	→		
		3.2.2 - 식단 관리 상세 페이지 - 운동 관리 상세 페이지 - 상태 관리 상세 페이지 - 달력 상세 페이지 - 'About Us' 페이지 이가원	→		
	3.3. 기능 연동	3.3.1. - 백엔드 API 연동 - 사용자 경험(UX) 개선 - 반응형 디자인 상세 적용 이가원	→		

구 분			기간	2024년 1월 ~ 2024 2월		
업무명(모듈명)			담당자	2 주	3 주	4 주
4.0.0. 백엔드 개발	4.1. 데이터베이스 구축	4.1.1. - 데이터베이스 스키마 설계 - 초기 데이터 및 더미 삽입	고진석		→	
	4.2.0. 핵심 LLM 및 API 기능 구현	4.2.1. - UUID 기반 사용자 식별/관리 기능 - AI 챗봇 인터페이스 API (LLM 프롬프트) - LLM 파싱 & 분류 로직 구현 - 모드별 맞춤형 응답 및 API 호출 연동 - 상세 탭별 데이터 처리 API - 대화 히스토리 관리 API - 공통 유틸리티 및 미들웨어	고진석		→	
5.0.0. 테스트 및 수정 / 보완	5.1.0. 기능 개선 및 추가	5.1.1 - LLM 응답 품질 최적화 - 사용자 경험 피드백 반영 - 데이터 시각 고도화	고진석		→	
	5.2.0. 성능 최적화 및 안정화	5.2.1. - 백엔드 API 성능 최적화 - 프론트엔드 렌더링 최적화 - 에러 처리 및 로깅 시스템 강화 - 취약점 점검 및 보완	고진석		→	
	5.3.0. 코드 리팩토링 및 문서화	5.3.1. - 백엔드 코드 리팩토링 - 프론트엔드 코드 리팩토링 - API 문서 업데이트	팀		→	
	5.4.0. 서버 테스트 및 모니터링	5.4.1. - 스테이징 서버 테스트 - 성능 및 부하 테스트 - 모니터링 시스템 구축	팀		→	

구 분			기간	2024년 1월 ~ 2024 2월		
업무명(모듈명)			담당자	2 주	3 주	4 주
6.0.0. 배포	6.1.0	백엔드 서비스 배포 준비	고진석			————→
	6.2.0	프론트엔드 서비스 배포 준비	이가원			————→
	6.3.0	최종 연동 확인 및 유저 테스트	팀			————→
7.0.0. 마무리	7.1.1.	최종 보고서 및 발표 준비	팀			————→

제2장 분석 공정

2.1 수요 분석

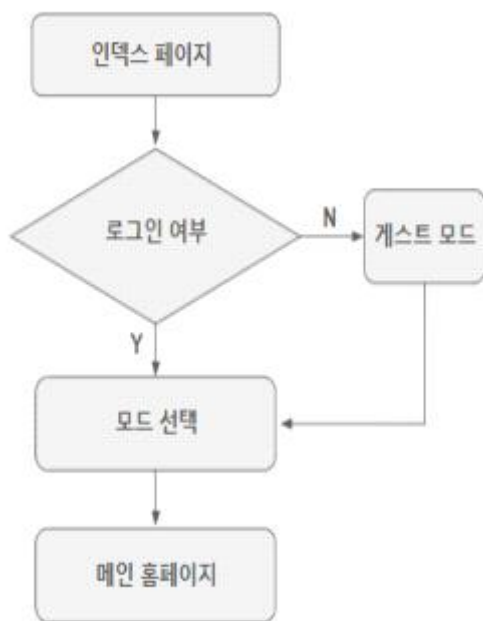
2.1.1. 요구사항 분석

- 1) PIN번호- 사용자 닉네임 입력시 PIN 필드 추가하여 사용자 구분
- 2) 자연어 입력시 식단/운동 등 기록 자동 입력 - LLM 모델의 파싱-분류로 입력 자동화
- 3) 루틴 리포트- 히스토리 데이터 시각화
- 4) 모드 설정- 목표와 상태에 맞는 단계별 모드
- 5) 격려 메시지 출력 - 감정 상태/기록 기반 긍정적 응원 문장 출력

2.2 유저 시나리오

2.2.1. 주요 사용자 시나리오

- 인덱스 페이지

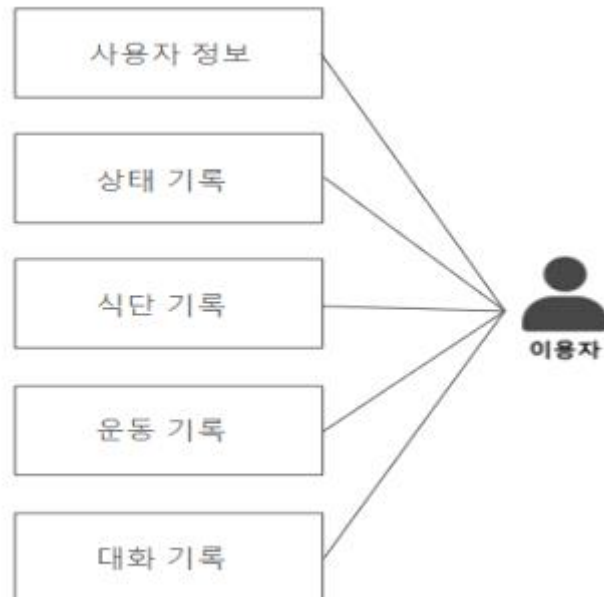


- 메인홈페이지



2.3 데이터 흐름 및 구조

2.3.1. 개체-관계 도표 ERD



2.3.2. 주요 데이터 구조 설명

DietEntry 컬렉션	
_id	ObjectId (Primary Key)
user	ObjectId (해당 상태 기록의 사용자 ID, User 컬렉션 참조, 필수)
date	Date (기록 날짜, 기본값:현재 시간, 필수)
mealType	String (식사 종류 : 'breakfast, lunch, dinner, other' 중 하나, 기본값: 'other')
foodItems	Array of Objects (섭취 음식 목록)
waterIntakeMI	NUmber (물 섭취량, 밀리리터, 기본값:0)
totalCalories	NUmber (해당 식단의 총 칼로리, 계산된 값, 기본값:0)
notes	String (기타 메모, 기본값: ' ')
createdAt	Date (문서 생성일시)
updatedAt	Date (마지막 정보 수정일시)

WorkoutEntry 컬렉션	
_id	ObjectId (Primary Key)
user	ObjectId (해당 상태 기록의 사용자 ID, User 컬렉션 참조, 필수)
date	Date (기록 날짜, 기본값:현재 시간, 필수)
workoutType	String (운동 종류, 기본값: 'other')
exercises	Array of Objects (운동 세부 목록)
totalDuration Minutes	Number (해당 운동 기록의 총 시간, 분, 계산된 값, 기본값:0)
totalCalories Burned	Number (해당 운동 기록의 총 칼로리, 소모량, 계산된 값, 기본값:0)
notes	String (기타 메모, 기본값:'')
createdAt	Date (문서 생성일시)
updatedAt	Date (마지막 정보 수정일시)

User 컬렉션	
_id	ObjectId (MongoDB 고유 ID, Primary Key)
nickname	String (사용자 닉네임, 필수, 고유값, 공백제거)
mode	String (현재 설정된 난이도 모드, 기본값: 'easy')
pin	Number (PIN 번호, 선택 사항)
isGuest	Boolean (게스트 사용자 여부, 기본값:false)
targetWeight	Number (목표 체중, kg 기본값:null)
targetCalories	Number (목표 칼로리, 기본값:null)
waterGoal	Number (개인별 물 목표량, 리터단위, 기본값: 2)
createdAt	Date (문서 생성일시)
updatedAt	Date (마지막 정보 수정일시)

Status 컬렉션	
_id	ObjectId (Primary Key)
user	ObjectId (해당 상태 기록의 사용자 ID, User 컬렉션 참조, 필수)
weight	Number (체중, kg, 필수)
bodyFatPercentage	Number (체지방률, %)
skeletalMuscleMass	Number (골격근량, kg)
notes	String (기타 메모, 기본값: ' ')
createdAt	Date (문서 생성일시)
updatedAt	Date (마지막 정보 수정일시)

ChatHistory 컬렉션	
_id	ObjectId (Primary Key)
user	ObjectId (해당 상태 기록의 사용자 ID, User 컬렉션 참조, 필수)
messages	Array of Objects (대화 메시지 배열)
createdAt	Date (문서 생성일시)
updatedAt	Date (마지막 정보 수정일시)

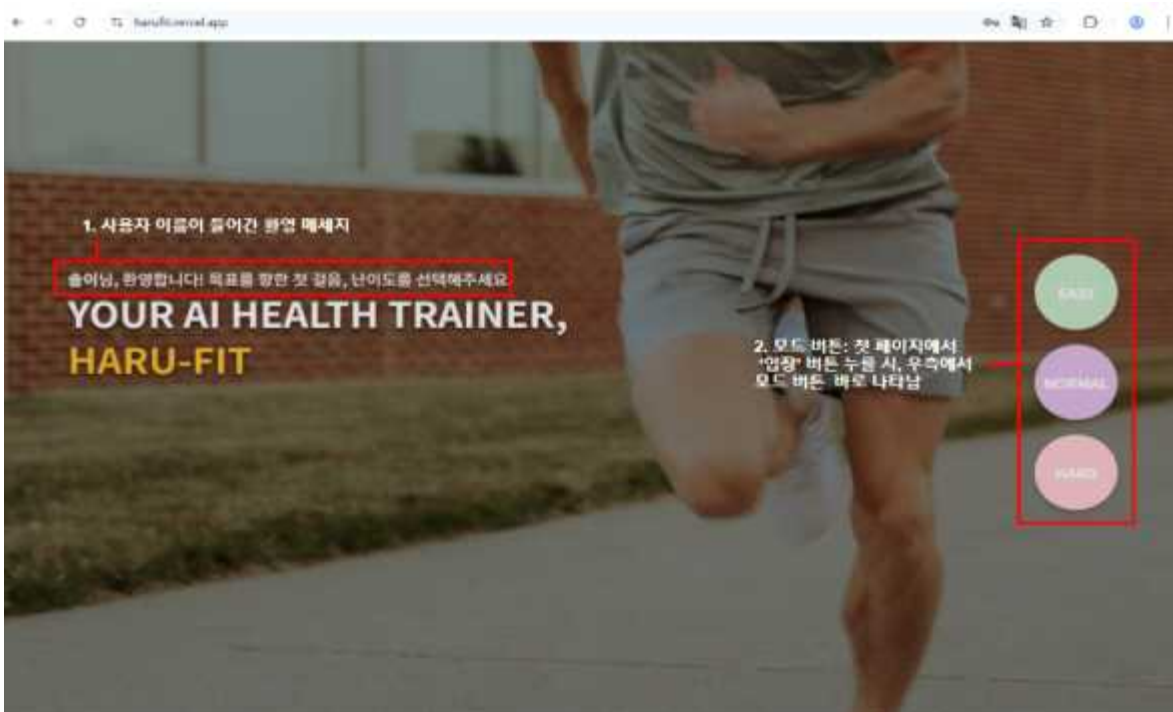
2.4 프로세스 분석

2.4.1. 페이지 구조(화면 설계 요약)

- Index page 인덱스 페이지: 앱 진입 첫 화면



- Index page 인덱스 페이지: 로그인 이후 모드 버튼 출현



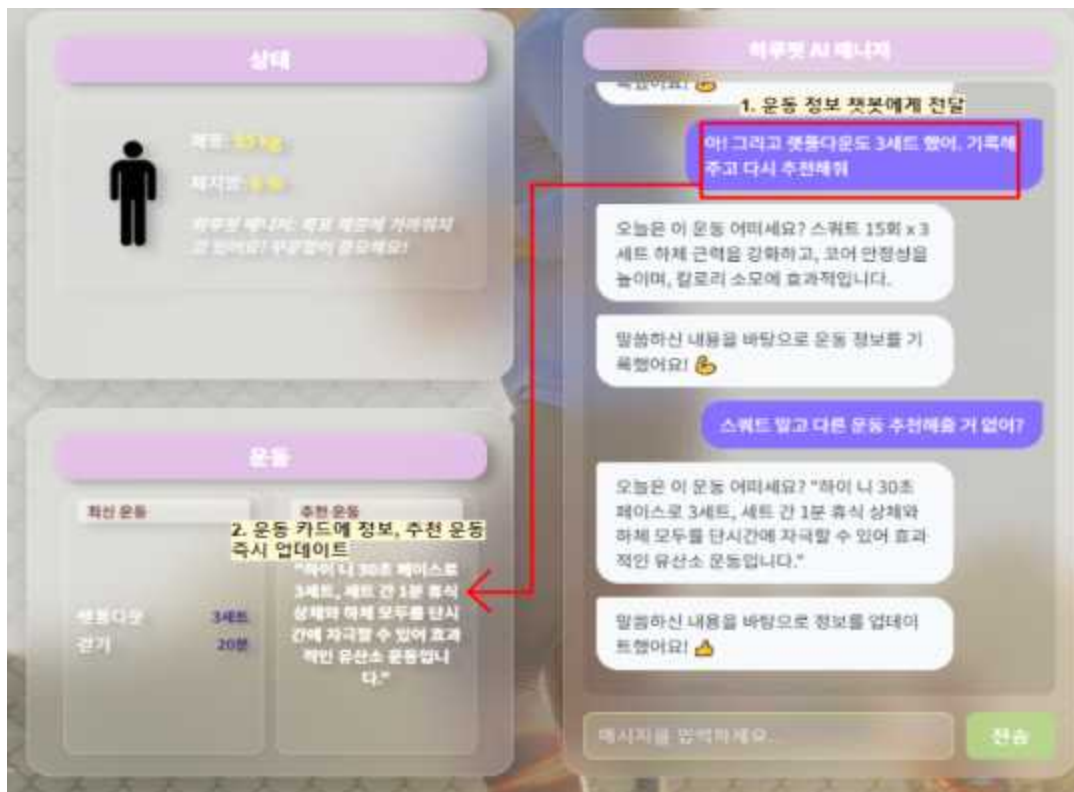
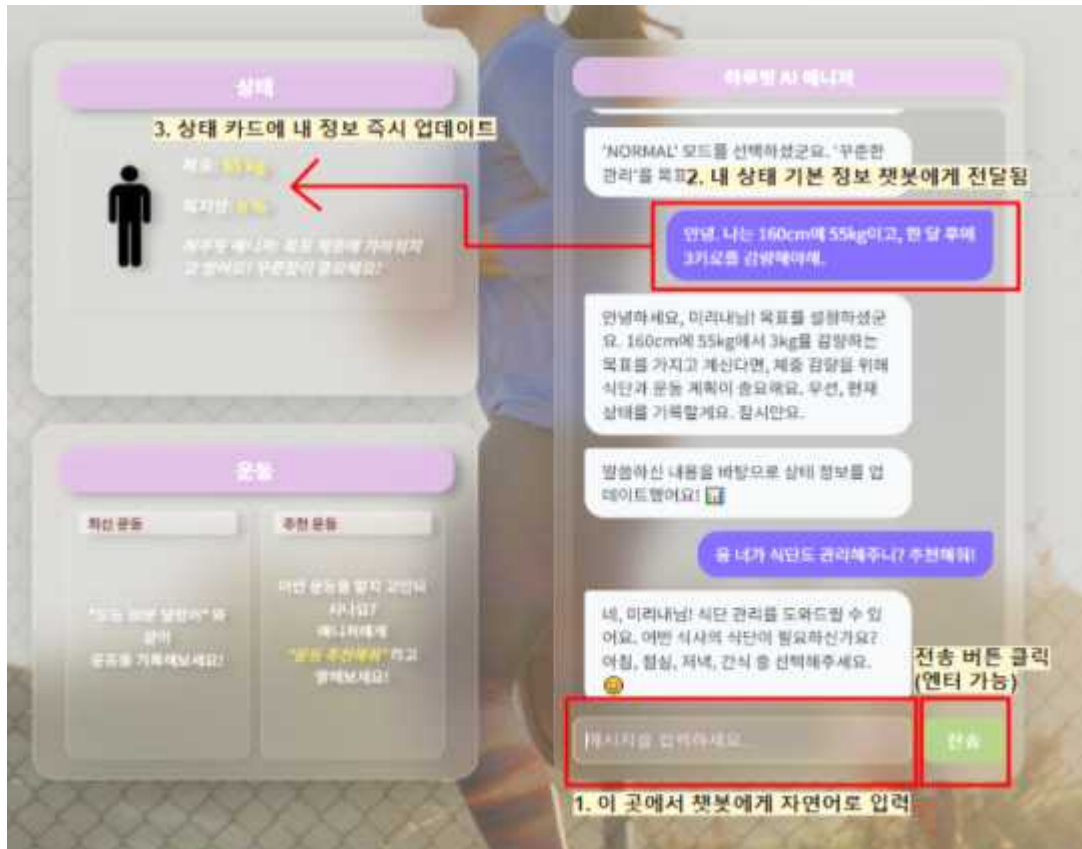
- Index page 인덱스 페이지: 모드 버튼 호버 시 모드 설명 출현



- HomePage 메인홈페이지: 인덱스페이지에서 로그인->모드 선택 이후 진입 첫 화면



- HomePage 메인홈페이지: 챗봇에게 상태 입력 -> 카드(상태 statusCard, 운동 workoutCard 등) 즉각 반영



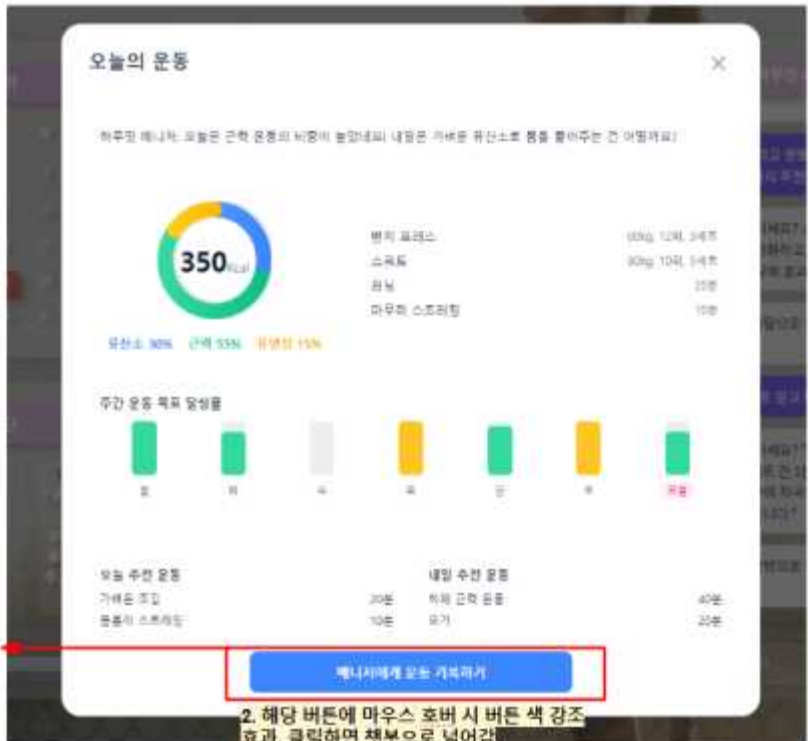
- HomePage 메인홈페이지-카드 탭별 상세화면: 해당 화면은 페이지 이동의 개념이 아닌, 모달뷰로써 클릭시에 확장되며 나타난다.



1. 각 카드 탭에 마우스 호버, 시 헤더 부분 우측에 화살표 출현 -> 화살표 클릭 -> 상세 뷰 화면 확장됨.



3. 상세 뷰에서 넘어온 채팅창에는 기본적으로 해당 카테고리의 기본 입력 폼(form)이 세팅되어있음



2. 해당 버튼에 마우스 호버 시 버튼 색 강조 효과, 클릭하면 채팅으로 넘어감

오늘의 식단

하루의 에너지 사용자인 '오늘' 탄수화물 섭취가 조금 부족해 보여요. 통곡물이나 채소를 더 추가해 보는 건 어떨까요?

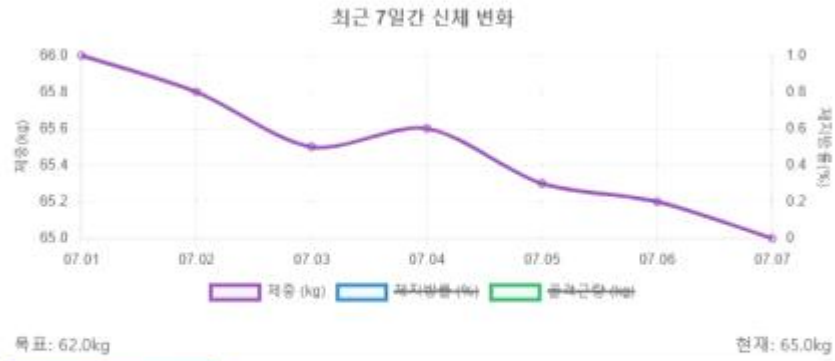


▶ 식단 모달뷰

나의 몸 상태



하루씩 매니저: 최근 체중 변화가 꾸준히 나타나고 있네요! 그래프에 마우스를 올려 상세 수치를 확인해보세요.



현재 체중
65.0 kg
▼ 0.2kg

체지방률
22.3 %
▼ 0.2%

골격근량
28.2 kg
▲ 0.1kg

매니저에게 상태 기록하기

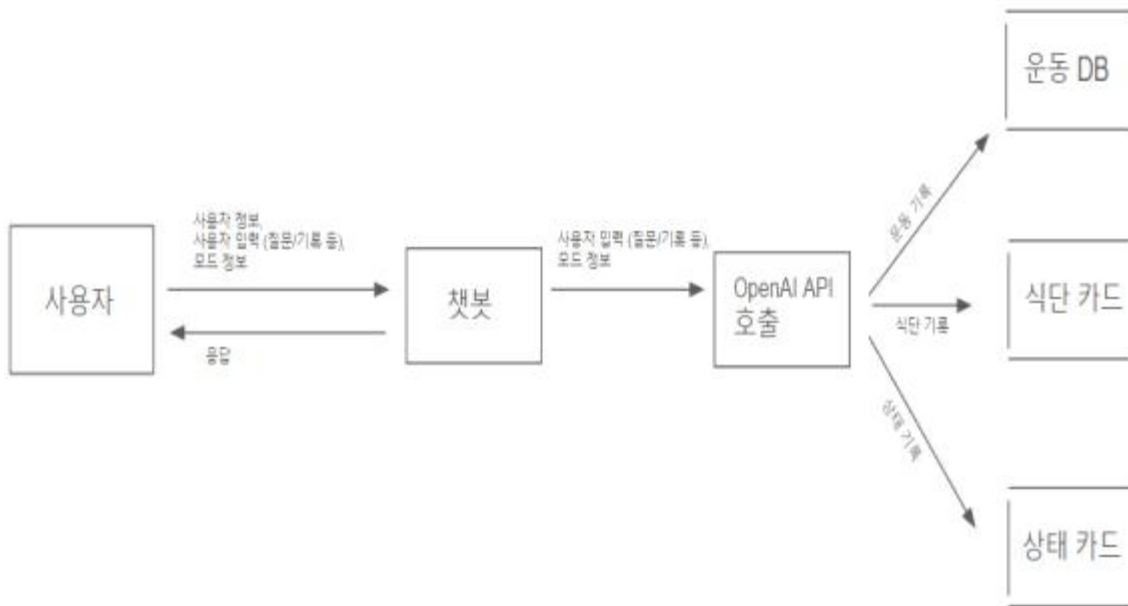
▶ 상태 모달뷰



▶ 달력 모달뷰

- 참고: 모달뷰 내부 데이터들은 MVP 단계에서는 MockData로 처리.

2.4.2. 자료 흐름도(Data Flow Diagram)



2.4.3. 핵심 기능 목록

- 1) **AI챗봇 기반 기록/코칭**: 사용자의 자연어 입력을 OpenAI LLM model로 분석하여 '기록', '추천', '설정 변경', '되묻기' 등 다양한 의도를 명확히 구분하고 처리하여, 정해진 버튼이나 폼 없이 대화만으로 앱의 모든 핵심 기능을 사용할 수 있다.
- 2) **히스토리 기반 개인화 및 고도화된 코칭**: 사용자의 기록을 기억하고, 이 히스토리를 기반으로 현재 상태와 상황을 이해하고 조합하여 초개인화된 코칭을 제공한다. 또한 llm의 프롬프트는 단순한 정보 조회를 넘어, AI에게 '다이어트 전문가', '퍼스널 트레이너'와 같은 구체적인 역할을 부여하여 친밀한 느낌을 제공한다.
- 3) **모드별 서비스**: 사용자 선택 모드(Easy/Normal/Hard)에 따른 콘텐츠 및 코칭을 제공한다.
- 4) **실시간 UI 피드백**: 사용자가 채팅으로 정보 기록/ 추천 받기를 행하면, 즉시 해당 UI 카드가 업데이트되어 사용자로 하여금 행동이 시스템에 잘 반영되었음을 시각적으로 확인 시켜준다.
- 5) **개인화 리포트 및 통계**: 기록 데이터 기반의 시각화된 분석 리포트를 제공한다.

2.4.4. 기능 명세서

기능명	사용자 인증 및 관리
기능 정의	사용자의 로그인, 개인 정보(닉네임, PIN, 모드, 목표)를 관리하는 기능
관련 엔티티	사용자 (User), User_DB
입력 데이터	닉네임, PIN, 모드 정보 (easy/normal/hard), 목표 (체중, 칼로리, 물 섭취량)
출력 데이터	인증 토큰, 사용자 정보 (닉네임, 모드, 목표 등), 로그인/회원가입 성공/실패 메시지
상세 설명	
<p>사용자는 닉네임과 PIN 번호를 입력하여 회원가입 및 로그인을 한다. 로그인 성공 시 사용자 세션이 유지되며, 메인 페이지로 이동한다. 사용자는 로그인 후 모드를 선택하고, 필요에 따라 목표(체중, 칼로리, 물 섭취량)를 설정하거나 변경할 수 있다.</p>	
요구사항 및 제약조건	
<p>이용자가 입력한 아이디가 이미 존재하는 아이디일 경우 이용자에게 이미 존재하는 아이디임을 알린다. 입력 정보가 형식에 맞지 않을 경우 이용자에게 올바르게 입력하지 않은 형식임을 알린다.</p>	

기능명	AI 챗봇 기반 기록 및 코칭
기능 정의	사용자의 자연어 입력에 기반하여 식단, 운동, 신체 상태를 기록하고, 개인 맞춤형 코칭 및 피드백을 제공하는 핵심 기능
관련 엔티티	사용자 (User), 챗봇 (AI 챗봇), OpenAI API 호출, WorkoutEntry DB, DietEntry DB, Status DB, ChatHistory_DB
입력 데이터	사용자 자연어 메시지 (식단, 운동, 상태, 질의 등)
출력 데이터	AI 챗봇 응답 메시지, 기록 저장 결과, 개인화된 코칭/피드백
상세 설명	
<p>사용자는 챗봇 인터페이스에 자연어로 메시지를 입력한다. 챗봇은 입력 문장의 의도(식단, 운동, 기타)를 파악하고, OpenAI API를 호출하여 응답을 생성하거나 데이터를 추출한다. 추출된 데이터는 해당하는 DB(운동, 식단, 상태)에 저장되고 화면 각 카드에 띄워진다. 챗봇은 사용자의 기록 및 과거 대화를 기반으로 맞춤형 코칭과 격려 메시지를 제공한다.</p>	
요구사항 및 제약조건	
<p>자연어 입력시 식단/운동 등 기록 자동 입력 - LLM 모델의 파싱-분류로 입력 자동화, 격려 메시지 출력 - 감정 상태/기록 기반 긍정적 응원 문장 출력</p>	

기능명	식단 기록 관리
기능 정의	사용자의 식단 정보를 기록하고, 기록된 식단 데이터를 조회하며, 물 섭취량을 관리하는 기능
관련 엔티티	챗봇 (AI 챗봇), 식단 카드 (DietEntry DB)
입력 데이터	챗봇을 통해 파싱된 식단 정보 (식사 종류, 음식 목록, 칼로리, 영양소, 물 섭취량 등)
출력 데이터	기록 성공 여부, 조회된 식단 목록, 총 칼로리 등
상세 설명	
<p>사용자가 챗봇에 식단 관련 내용을 입력하면, AI 챗봇이 이를 파싱하여 DietEntry 스키마에 따라 식단 정보를 식단 카드 DB에 저장한다. 기록된 식단은 날짜별, 식사 종류별로 조회할 수 있으며, 총 칼로리 및 물 섭취량이 계산되어 표시된다.</p>	
요구사항 및 제약조건	
자연어 입력시 식단/운동 등 기록 자동 입력 - LLM 모델의 파싱-분류로 입력 자동화	

기능명	운동 기록 관리
기능 정의	사용자의 운동 정보를 기록하고, 기록된 운동 데이터를 조회하는 기능
관련 엔티티	챗봇 (AI 챗봇), 운동 카드 (WorkoutEntry DB)
입력 데이터	챗봇을 통해 파싱된 운동 정보 (운동 종류, 운동 목록, 시간, 세트, 칼로리 등)
출력 데이터	기록 성공 여부, 조회된 운동 목록, 총 운동 시간, 총 소모 칼로리 등
상세 설명	
<p>사용자가 챗봇에 운동 관련 내용을 입력하면, AI 챗봇이 이를 파싱하여 WorkoutEntry 스키마에 따라 운동 정보를 운동 카드 DB에 저장한다. 기록된 운동은 날짜별, 운동 종류별로 조회할 수 있으며, 총 운동 시간 및 총 소모 칼로리가 계산되어 표시된다.</p>	
요구사항 및 제약조건	
자연어 입력시 식단/운동 등 기록 자동 입력 - LLM 모델의 파싱-분류로 입력 자동화	

기능명	대화 기록 관리
기능 정의	사용자(user)와 AI 어시스턴트(assistant) 간의 대화 내용을 시간 순서대로 저장하고 조회하는 기능
관련 엔티티	챗봇 (AI 챗봇), 운동 카드 (ChatHistory DB)
입력 데이터	사용자 메시지, AI 응답 메시지 (role, content, timestamp, extractedData)
출력 데이터	특정 사용자의 대화 기록 목록
상세 설명	
<p>사용자와 챗봇의 모든 대화 내용은 ChatHistory_DB에 저장된다. 이는 챗봇이 이전 대화의 맥락을 이해하고 더 자연스러운 상호작용 및 맞춤형 코칭을 제공하는 데 활용된다. 사용자는 자신의 과거 대화 기록을 조회할 수 있다.</p>	
요구사항 및 제약조건	
없음 (내부 시스템 요구사항)	

기능명	메인페이지
기능 정의	사용자가 개인별 건강 목표(예: 체중, 칼로리, 물 섭취량)를 설정하고, 이를 조회 및 수정하는 기능.
관련 엔티티	사용자 (User), User_DB, CalendarGoal_DB (추가 예정)
입력 데이터	목표 체중, 목표 칼로리, 목표 물 섭취량 등
출력 데이터	설정된 목표 값, 목표 설정 성공/실패 여부
상세 설명	
<p>사용자는 앱 내에서 자신의 targetWeight, targetCalories, waterGoal 등의 목표를 설정하고 관리할 수 있다. 설정된 목표는 개인화된 코칭 및 통계 리포트의 기준으로 활용된다. calendarGoalRoutes를 통해 캘린더 기반의 목표 관리 기능이 추가될 수 있으며, 이는 별도의 CalendarGoal_DB에 저장된다.</p>	
요구사항 및 제약조건	
모드 설정- 목표와 상태에 맞는 단계별 모드	

기능명	모드 설정
기능 정의	사용자가 자신의 목표와 현재 상태에 맞춰 'easy', 'normal', 'hard' 중 하나의 코칭 모드를 선택하고 변경하는 기능
관련 엔티티	사용자 (User), User_DB
입력 데이터	선택된 모드 (easy, normal, hard)
출력 데이터	설정된 모드, 모드 변경 성공 여부
상세 설명	
<p>사용자는 [하루핏] 앱에서 제공하는 세 가지 난이도(easy, normal, hard) 중 하나를 선택하여 자신의 운동 및 식단 코칭의 강도를 조절할 수 있다. 이 설정은 AI 챗봇의 응답 및 추천 내용에 영향을 미쳐 개인화된 경험을 제공한다.</p>	
요구사항 및 제약조건	
<p>모드 설정- 목표와 상태에 맞는 단계별 모드</p>	

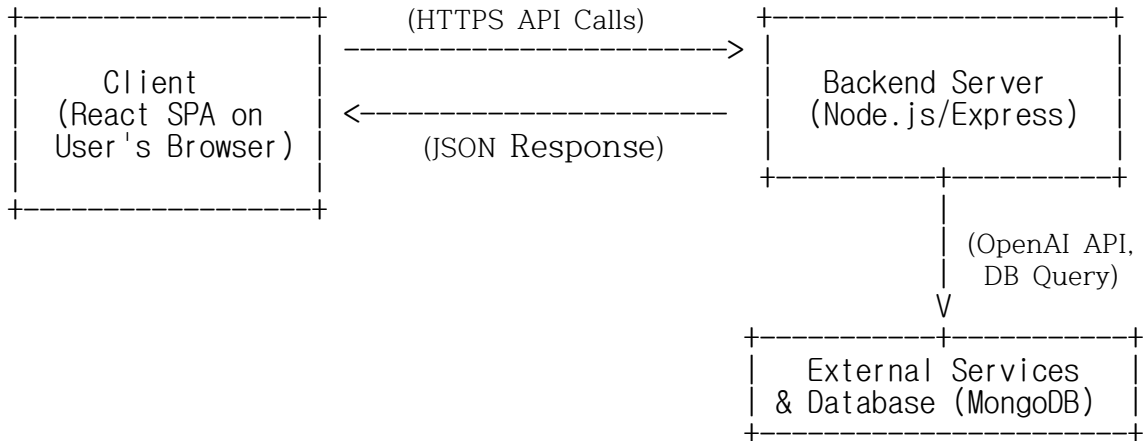
기능명	루틴 및 리포트 제공
기능 정의	사용자의 기록된 운동, 식단, 신체 상태 데이터를 분석하여 개인화된 루틴 및 시각화된 리포트를 제공하는 기능
관련 엔티티	운동 카드 (WorkoutEntry DB), 식단 카드 (DietEntry DB), 상태 카드 (Status DB), 통계 및 분석 (프로세스)
입력 데이터	WorkoutEntry, DietEntry, Status 의 기록 데이터
출력 데이터	주/월간 운동 요약, 영양 섭취 분석, 체중 변화 그래프 등 시각화된 리포트, 루틴 추천
상세 설명	
<p>시스템은 사용자의 누적된 기록 데이터를 분석하여 운동 및 식단 루틴의 패턴을 파악한다. 이를 바탕으로 사용자가 자신의 건강 상태를 직관적으로 이해하고 목표 달성에 도움이 될 수 있도록 다양한 형태의 통계 및 그래프(루틴 리포트)를 제공한다.</p>	
요구사항 및 제약조건	
<p>루틴 리포트- 히스토리 데이터 시각화</p>	

제3장 개요

3.1 시스템 아키텍처

3.1.1. 전체 구성도

Harufit은 최신 웹 기술 스택을 기반으로 한 클라이언트-서버 아키텍처로 구성된다.



▶ 하루핏 시스템의 전체 흐름을 네트워크/데이터 흐름 관점에서 보여주는 아키텍처 다이어그램

3.1.2. 기술 환경 요약

구분	기술	버전/설명	선택사유
Frontend	React	18.x	컴포넌트 기반 아키텍처로 UI의 재사용성과 유지보수성을 높임
	Axios	1.x	Promise 기반의 비동기 통신을 통해 서버와 효율적으로 데이터를 교환
Backend	Node.js	18.x LTS	비동기 I/O 처리에 강점을 보여, 채팅 기반의 실시간 상호 작용에 적합
	Express	4.x	Node.js를 위한 경량 웹프레임워크로, RESTful API를 빠르고 유연하게 구축
Database	MongoDB		유연한 스키마 구조로, 건강 데이터처럼 항목이 자주 바뀔 수 있는 데이터 관리에 용이
	Mongoose		MongoDB 데이터를 객체 지향적으로 모델링하고, 데이터 유효성 검증을 편리하게 처리
AI	OpenAI API	gpt-4o	뛰어난 자연어 이해(NLU) 능력과 tools 기능을 통해 사용자의 복잡한 의도를 정확히 파악

3.1.3. 기술 흐름도 (Sequence Diagram)

- 사용자의 채팅 입력부터 AI 분석, 데이터 저장, 그리고 여러 UI 카드(식단, 달력 등)가 동시에 업데이트되기까지의 핵심 데이터 흐름을 순서도로 나타낸다. 특히 AI가 사용자의 입력이 불안전하다고 판단할 경우 '되묻기(Clarification)'를 통해 데이터의 정확성을 보장하는 과정이 포함된 것이 핵심이다.

<코드 스니펫>

```
participant User as 🧑🏻💻 사용자(Browser)
participant FE as 🏠 Frontend (React)
participant BE as 🖥️ Backend (Node.js)
participant AI as 🤖 OpenAI API
participant DB as 🗄️ MongoDB
```

User->>FE: 채팅 메시지 입력("저녁으로 닭가슴살200g 먹었어")

FE->>BE: POST /api/ai/parse-and-log (메시지, 대화 이력 전송)

BE->>AI: 1. 대화 분석 요청(systemPrompt + history + message)

alt AI가 정보 부족을 감지할 경우(e.g., 식사 시간 누락)

AI-->>BE: 2a. '되묻기' 요청(tool_calls: request_clarification)

BE-->>FE: '되묻기' 질문 반환("언제 드셨나요?")

FE-->>User: AI 응답으로 질문 표시

else AI가 정보를 성공적으로 구조화한 경우

AI-->>BE: 2b. 구조화된JSON 데이터 반환(tool_calls: log_health_data)

opt 식단 기록일 경우

BE->>AI: 3. 영양 정보 분석 요청(getNutritionEstimate)

AI-->>BE: 칼로리 등 영양 정보 반환

end

BE->>DB: 4. 새 데이터 저장(new DietEntry().save())

DB-->>BE: 저장 성공

BE-->>FE: 5. 데이터 처리 완료 응답 전송

Note over FE: onDataRefresh 함수 호출(refetch 신호)

FE->>BE: 6a. 식단 카드 데이터 다시 요청(GET /api/diet/today)

BE->>DB: 데이터 집계(Aggregation)

DB-->>BE: 일일 요약 데이터 반환

BE-->>FE: 최신 식단 요약 데이터 전송

FE->>BE: 6b. 달력 데이터 다시 요청(GET /api/calendar/summary)

Note over BE, DB: DietEntry, WorkoutEntry를 동시에 조회하여 날짜별 기록 유무 집계

DB-->>BE: 월간 활동 요약 데이터 반환

BE-->>FE: 최신 달력 요약 데이터 전송

FE-->>User: 7. 식단 카드와 달력 카드 동시 업데이트

end

3.2 자료 설계

3.2.1. 주요 데이터 항목 (자료 사전)

컬렉션/ DTO	필드명	내역	자료형	제약조건/비고
users	_id	사용자고유 ID	ObjectId	Primary Key
	nickname	닉네임	String	Required, Unique
	password	해시된비밀번호	String	Required
	waterGoal	일일물섭취목표	Number	Default: 2 (단위: L)
dietentries	_id	식단기록 ID	ObjectId	Primary Key
	user	사용자 ID	ObjectId	Foreign Key (users 참조)
	date	기록날짜(KST)	Date	Required
	mealType	식사종류	String	Enum: 'breakfast', 'lunch', 'dinner', 'snack'
	foodItems	음식목록	Array of Objects	{ name, quantity, calories }
	waterIntakeMl	물섭취량	Number	단위: mL
workoutentries	_id	운동기록 ID	ObjectId	Primary Key
	user	사용자 ID	ObjectId	Foreign Key (users 참조)
	date	기록날짜(KST)	Date	Required
	exercises	운동목록	Array of Objects	{ name, reps, sets, durationMinutes, weightKg }
statuses	_id	상태기록 ID	ObjectId	Primary Key
	user	사용자 ID	ObjectId	Foreign Key (users 참조)
	date	기록날짜(KST)	Date	Required
	weight	체중	Number	단위: kg
	bodyFatPercentage	체지방률	Number	단위: %
	(CalendarSummary)	(월간활동요약)		DB 에저장되지않는, API 응답용데이터구조
	date	날짜	String	YYYY-MM-DD' 형식
	hasDiet	식단기록유무	Boolean	
	hasWorkout	운동기록유무	Boolean	

3.2.2. 컬렉션 구성(ERD)

- Haru-Fit의 데이터베이스는 MongoDB를 기반으로 하며, 각 데이터는 컬렉션(Collection) 단위로 구성된다. 핵심 설계는 사용자(USER) 한 명이 여러 개의 상태(STATUS), 식단(DIET_ENTRY), 운동(WORKOUT_ENTRY) 기록을 가질 수 있는 1:N 관계를 중심으로 이루어진다.

- 달력 데이터에 대한 설계원칙

UI에는 '달력 카드'가 있지만, 아래 ERD(개체-관계 다이어그램)에는 별도의 달력 엔티티가 없다. 이는 달력에 표시되는 활동 마크(분홍/파란 점)가 DIET_ENTRY와 WORKOUT_ENTRY에 저장된 date 필드를 백엔드에서 실시간으로 집계(Aggregation)하여 동적으로 생성하는 '뷰(View)'의 개념이기 때문이다. 이 설계는 데이터베이스의 중복을 방지하고 항상 정확한 활동 요약を保장한다.

- Entity-Relationship Diagram (ERD)

아래 다이어그램은 실제 데이터베이스에 영구적으로 저장되는 핵심 컬렉션들과, 동적으로 생성되는 CALENDAR_VIEW 간의 데이터 흐름 관계를 시각적으로 표현한다. DIET_ENTRY와 WORKOUT_ENTRY에서 점선 화살표가 CALENDAR_VIEW로 향하는 것은, 이 두 컬렉션의 데이터를 조합해서 달력의 내용을 만든다는 의미이다.

<코드 스니펫>

```
USER ||--o{ STATUS : "records"
USER ||--o{ DIET_ENTRY : "records"
USER ||--o{ WORKOUT_ENTRY : "records"

DIET_ENTRY }..>o CALENDAR_VIEW : "provides data for"
WORKOUT_ENTRY }..>o CALENDAR_VIEW : "provides data for"

USER {
    ObjectId _id PK
    string nickname "Unique"
    string password
    number waterGoal
}
STATUS {
    ObjectId _id PK
    ObjectId user "FK"
    date date
    number weight
    number bodyFatPercentage
}
DIET_ENTRY {
    ObjectId _id PK
    ObjectId user "FK"
    date date
    string mealType
    array foodItems
```

```

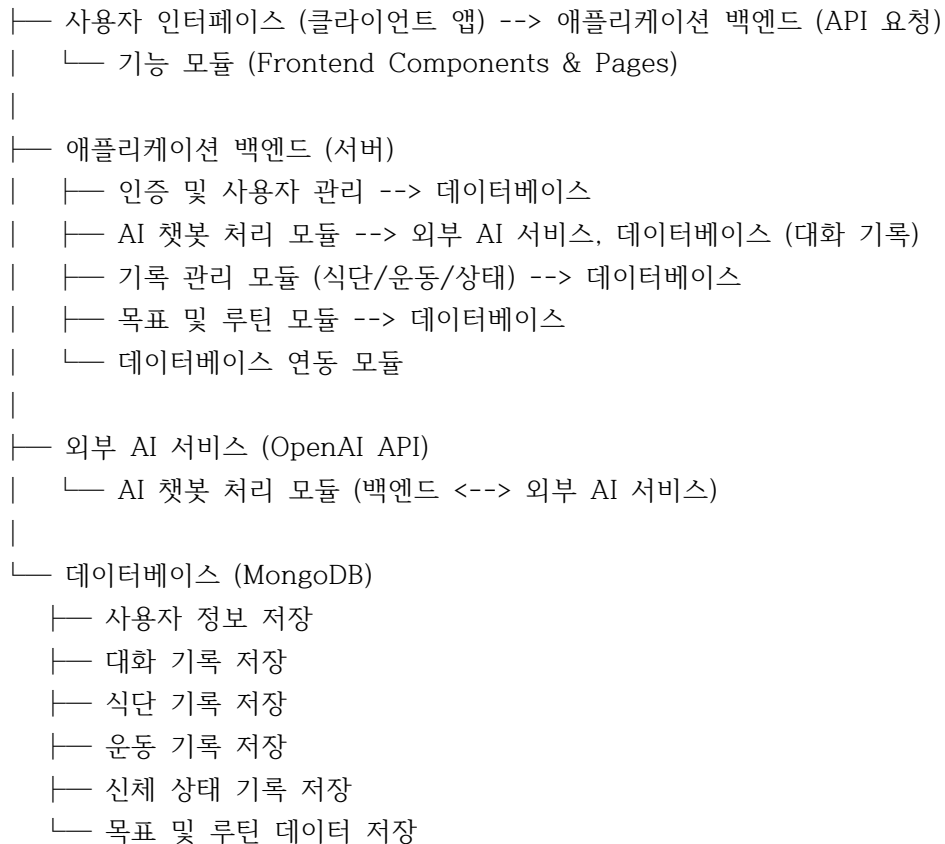
        number waterIntakeMl
    }
    WORKOUT_ENTRY {
        ObjectId _id PK
        ObjectId user "FK"
        date date
        array exercises
    }
    CALENDAR_VIEW {
        string date "YYYY-MM-DD"
        boolean hasDiet
        boolean hasWorkout
        array allRecordsForDate
        note "실제 DB 테이블이 아닌, 동적으로 생성되는 데이터 뷰"
    }

```

3.3 코드 설계

3.3.1. 구조 차트 (Structure Chart)

[하루핏] 시스템



3.3.2. 모듈 개요도

모듈번호	보충설명
1	코어시스템 및 UI
1.1	HomePage UI 및 상태관리
1.2	ManagerChat AI 채팅 인터페이스
2	핵심 데이터 처리(Backend)
2.1	AI 채팅 처리(aiController), AI 서비스 로직(aiService)
2.2	요약 데이터 API (diet/workout/status/calendar Controllers)
3	주요 기능: 데이터 기록
3.1	AI 기반 식단 및 물기록
3.2	AI 기반 운동 기록
3.3	AI 기반 신체 상태 기록
4	주요 기능: AI 추천
4.1	개인화된 식단 추천
4.2	개인화된 운동 추천
5	데이터 시각화(UI Cards)
5.1	DietCard (식단/물/추천)
5.2	WorkoutCard (운동/추천)
5.3	StatusCard (상태)
5.4	Calendar (활동 요약)

3.3.3. 모듈 설명서

1) 코어 시스템 및 UI

모듈명	HomePage UI 및 상태관리	모듈 ID	1.1
설명	모든 기능 카드와 채팅 UI 를 포함하는 메인 대시보드. 앱의 전체 데이터 흐름과 상태 관리를 총괄.		
관련 클래스	HomePage.jsx, useApi.js, DietCard.jsx, WorkoutCard.jsx, StatusCard.jsx, Calendar.jsx		
입력데이터	각 API 엔드포인트로부터 받은 요약 데이터(상태, 식단, 운동, 달력), localStorage 의 추천 데이터		
출력데이터	전체화면 UI, 각 카드 컴포넌트로 전달되는 props		
단계	처리내역	비고	
1	컴포넌트 마운트 시 useApi 훅을 통해 각 카드의 데이터를 비동기적으로 로드.	단방향 데이터 흐름	
2	localStorage 에서 오늘 날짜의 추천 데이터(식단, 운동)를 로드하여 상태를 초기화. (새로고침 시 유지)		
3	ManagerChat 으로부터 onDataRefresh 신호를 수신하면, 데이터 종류에 따라 refetch 를 호출하거나 추천 데이터 상태를 직접 업데이트하여 UI 를 리렌더링.		

모듈명	ManagerChat AI 채팅인터페이스	모듈 ID	1.2
설명	사용자와 AI 의 모든 상호작용이 이루어지는 채팅 컴포넌트.		
관련 클래스	ManagerChat.jsx, apiClient.js		
입력데이터	사용자 키보드 입력		
출력데이터	백엔드 API 서버로 전송하는 POST 요청(메시지, 대화이력)		
단계	처리내역	비고	
1	사용자가 메시지를 입력하고, '전송' 버튼을 클릭.	Axios 사용	
2	현재까지의 대화이력(messages state)과 새 메시지를 포함하여 백엔드/api/ai/parse-and-log 로 전송.		
3	백엔드로부터 받은 응답(AI 답변, UI 업데이트 신호)을 처리. (AI 답변은 채팅창에 표시, 업데이트 신호는 onDataRefresh 콜백호출)		

2) 핵심 데이터 처리 (Backend)

모듈명	AI 채팅처리	모듈 ID	2.1
설명	사용자의 자연어 채팅을 받아 A 로 분석하고, 그 결과를 바탕으로 데이터 기록/추천/되문기 등 적절한 작업을 수행하는 핵심 백엔드 로직.		
관련 클래스	aiController.js, aiService.js		
입력데이터	프론트엔드로부터 받은 POST 요청(사용자 메시지, 대화 이력)		
출력데이터	프론트엔드로 보내는 JSON 응답(채팅답변, UI 업데이트용 데이터)		
단계	처리내역	비고	
1	aiController 가/api/ai/parse-and-log 요청을 수신.		
2	aiService 를 통해 대화 이력과 함께 OpenAI API 에 분석 요청.	systemPrompt, tools 전달	
3	AI 응답(tool_calls)을 분석하여 log_health_data, get_recommendation, request_clarification 등으로 분기.	핵심로직	
4	Mongoose 모델을 사용해 DB 에 데이터를 저장하거나, 추천 내용을 생성.		
5	최종 결과를 프론트엔드에 reply 와 savedData 가 포함된 JSON 으로 반환.		

모듈명	요약 데이터 API	모듈 ID	2.2
설명	각 카드에 표시될 요약 데이터를 생성하여 제공하는 API.		
관련클래스	dietController.js, workoutController.js, statusController.js, calendarController.js		
입력데이터	GET 요청(사용자 ID 는 JWT 토큰에서추출)		
출력데이터	각 카드에 맞는 요약된 JSON 데이터		
단계	처리내역	비고	
1	프론트엔드의 useApi 혹은 각 엔드포인트(예:/api/diet/today)로 GET 요청.		
2	해당 컨트롤러는 Mongoose 모델을 사용하여 오늘 날짜의 사용자 데이터를 조회.		
3	MongoDB Aggregation Pipeline 을 사용하여 DB 단에서 데이터를 효율적으로 집계 및 계산(예: 총칼로리합산).	데이터 정합성 확보	
4	카드 UI 에 최적화된 형태로 데이터를 가공해 JSON 으로 응답.		

3) 데이터 기록

모듈명	사용자 데이터 기록	모듈 ID	3
설명	사용자의 건강 관련 데이터를 통합적으로 기록하고 관리하는 핵심 모듈. 요청에 따라 데이터를 생성, 조회, 수정 및 삭제 (추후 추가).		
관련 클래스	[BE] dietController.js, workoutController.js, 등 Controller 파일 [Model] DietEntry.js, WorkoutEntry.js, Status.js, ChatHistory.js, user.js		
입력데이터	각 기록 유형에 따른 JSON 형식의 요청 본문, 사용자 ID, AI 챗 메시지 및 대화 히스토리		
출력데이터	생성되거나 업데이트된 기록 객체 데이터 (JSON)		
단계	처리 내역	비고	
1	사용자의 API 요청 또는 AI 챗 메시지를 통해 기록 데이터를 수신.		
2	수신된 데이터의 유효성을 검사하고, 필요한 경우 추가적인 데이터 처리(예:칼로리 자동 계산, 체중 변화 분석)를 수행.		
3	처리된 데이터를 기반으로 DietEntry, Status, User 모델 등을 사용하여 DB에 새로운 기록을 생성 또는 업데이트		
4	데이터 처리 결과를 담은 적절한 HTTP 상태 코드와 JSON 응답을 클라이언트에게 반환		

4) AI 추천

모듈명	개인화된식단/운동추천	모듈 ID	4
설명	사용자의 추천 요청 및 대화 맥락을 파악하여 개인화되고 상세한식단/운동을추천.		
관련클래스	aiService.js, aiController.js		
입력데이터	사용자 추천 요청 메시지, 전체 대화 이력(history)		
출력데이터	상세 추천 내용(카드 UI 표시용), 채팅 답변		
단계	처리 내역	비고	
1	AI가 사용자의 "운동추천해줘"와 같은 요청을 인식하고 get_workout_recommendation 도구 사용을 결정.		
2	aiController가 aiService의 getSimpleWorkoutRecommendation 함수를 대화이력(history)과 함께 호출.	개인화의 핵심	
3	aiService는 '전문트레이너' 역할을 부여하는 상세한 프롬프트를 구성하여 OpenAI API 호출.	프롬프트 엔지니어링	
4	AI가 생성한 추천내용(예: "스쿼트\n3 세트\n하체강화에좋습니다")을 받아 프론트엔드로 전달.		

5) 데이터 시각화(UI Cards)

모듈명	사용자 데이터 기록	모듈 ID	5
설명	사용자의 식단, 운동, 신체 상태 데이터를 메인 대시보드에서 카드 형태로 요약하여 시각적으로 제공하는 모듈		
관련 클래스	[FE] StatusCard.jsx, DietCard.jsx, WorkoutCard.jsx [BE] statusController.js, dietController.js, workoutController.js, aiController.js [Model] Status.js, DietEntry.js, WorkoutEntry.js, User.js		
입력데이터	각 카드 유형에 따른 JSON 형식의 요약 데이터, 모드 (mode) 및 확장 버튼 콜백 (onExpand)		
출력데이터	사용자에게 시각적으로 표시되는 카드 형태의 대시보드 UI.		
단계	처리 내역	비고	
1	각 UI 카드 컴포넌트가 백엔드 API (해당 컨트롤러를 통해) 또는 상위 컴포넌트로부터 시각화할 데이터를 props 로 수신		
2	수신된 데이터를 사용자 친화적인 형식으로 가공하고, 이를 바탕으로 카드 UI 를 구성하여 화면에 렌더링		
3	'확장'(onExpand) 버튼 클릭 등 사용자 인터랙션을 감지하여 상위 컴포넌트로 전달		

3.3.4. 엔티티 프로세스 매트릭스

Haru-Fit의 핵심 프로세스(사용자 기능)가 어떤 엔티티에 대해 생성(C), 조회(R), 수정(U), 삭제(D) 작업을 수행하는지 나타낸 표.

프로세스/ 엔티티	USER	STATUS	DIET_ENTRY	WORKOUT_ENTRY
회원가입/ 로그인	CR			
사용자설정변경	U			
AI 채팅으로기록	R	CRU	C	C
카드데이터조회	R	R	R	R
달력데이터생성			R	R

C:생성(Create), R:조회(Read), U:수정(Update), D:삭제(Delete)

- 매트릭스 상세 설명

- 회원가입/ 로그인(USER: CR):사용자가 회원가입 시 엔티티가**생성(C)**되고, 로그인 시에는 인증을

위해**조회(R)**된다.

- **사용자 설정 변경(USER: U):**사용자가 채팅으로 '물 목표량 변경' 등을 요청하면 엔티티의 같은 필드가 **수정(U)**된다.
- **AI 채팅으로 기록(R, CRU, C, C):**
 - 기록을 저장하기 위해 먼저 사용자 정보를**조회(R)**한다.
 - : 그날의 상태 기록이 없으면**생성(C)**하고, 이미 있다면 기존 기록을 **조회(R)**하여**수정(U)**한다.
 - , : 식단과 운동은 기록할 때마다 새로운 데이터로**생성(C)**된다.
- **카드 데이터 조회(R, R, R, R):**프론트엔드의 각 카드를 표시하기 위해 모든 관련 엔티티를**조회(R)**하여 요약 정보를 만든다.
- **달력 데이터 생성(R, R):**달력의 활동 마크를 표시하기 위해 와에서특정월의데이터를**조회(R)**하여 동적으로 계산한다.

3.4 핵심 기능 설계 상세

- 1) AI 채팅 기반 데이터 처리

사용자의 자연어 입력을 OpenAI TOOLS 기능으로 분석하여 '기록', '추천', '설정 변경', '되문기' 등 다양한 의도를 명확히 구분하고 처리한다. 이는 정해진 버튼이나 폼 없이 대화만으로 앱의 모든 핵심 기능을 사용할 수 있게 한다.

- 2) 히스토리 기반 개인화 및 고도화된 코칭

aiService의 프롬프트는 단순한 정보 조회를 넘어, AI에게 '다이어트 전문가', '퍼스널 트레이너'와 같은 구체적인 역할을 부여한다. 또한 사용자의 이전 대화 이력(history)을 참고하여 "달리기를 좋아하신다니, 오늘은 인터벌 러닝을 추천합니다" 와 같이 개인의 선호도를 반영한 맞춤형 추천을 제공한다.

- 3) 모드 선택 기능

사용자의 mode 설정에 따라 AI 매니저의 코칭의 강도, 추천 방식, 프롬프트 내용 등을 동적으로 조절하여 개인화된 코칭 경험 제공하며, mode 별로 다른 UI를 적용하여, 사용자에게 모드 정보를 시각적으로 제공한다.

- 4) 실시간 UI 피드백

사용자가 채팅으로 정보를 기록하거나 추천을 받으면, 프론트엔드의 콜백과 useApi의 메커니즘을 통해 즉시 해당 UI 카드가 업데이트된다. 이는 사용자에게 자신의 행동이 시스템에 잘 반영되었음을 시각적으로 확인시켜 주는 중요한 역할을 한다.

1) AI 채팅 기반 데이터 처리

파일명	[Controller] aiController.js, chatHistoryController.js [Service] aiService.js [Model] ChatHistory.js
요약	사용자의 자연어 채팅을 OpenAI LLM(GPT-4o)으로 분석하여 의도(기록, 추천, 설정 등)를 파악하고, 그에 맞는 시스템 기능을 자동으로 호출 및 처리

[Controller] : aiController.js는 AI 기능의 메인 관제탑(Control Tower) 역할

1. 클라이언트로부터 메시지(message)와 대화기록(history) 수신

```

exports.parseAndLogChat = async (req, res) => {
  const user = req.user;
  const { message, history } = req.body;

  if (!message) {
    return res.status(400).json({ message: '메시지를 입력해주세요.' });
  }

  try {
    const aiResponseMessage = await aiService.getAiChatResponse(message, history || [], user);

    const toolCalls = aiResponseMessage.tool_calls;
    let savedData = [];
    let clarificationQuestion = null;
    let customReply = null;

    3. AI가 결정한 함수(toolCalls)가 있으면 실행
    if (toolCalls) {
      for (const toolCall of toolCalls) {
        const functionName = toolCall.function.name;
        const parsedArgs = JSON.parse(toolCall.function.arguments);

        4. 함수 이름에 따라 분기 처리
        if (functionName === 'log_health_data') {
          for (const item of parsedArgs.logs) {
            // ...
          }
        } else if (functionName === 'get_diet_recommendation') {
          const mealType = parsedArgs.mealType;
          // ...
        } else if (functionName === 'request_clarification') {
          clarificationQuestion = parsedArgs.question;
          // ...
        }
      }

      5. 최종 응답을 클라이언트에 전송
      res.status(200).json({
        reply: customReply || aiResponseMessage.content ||
        (savedData.length > 0 ? "기록되었습니다." : "네, 확인했습니다."),
        savedData: savedData,
        clarification: clarificationQuestion,
      });
    }
  } catch (error) {
    // ...
  }
}

```

[Service] Controller로부터 받은 사용자 메시지를 바탕으로, OpenAI API와 직접 통신하여 사용자의 의도를 분석하고 함수 호출을 결정하는 핵심 로직을 수행

1. Controller로부터 사용자 메시지, 대화 기록 수신

```

exports.getAiChatResponse = async (userMessage, history, user) => {
  try {
    const messages = [{ role: 'system', content: 'systemPrompt' }, ...history, { role: 'user', content: 'prompt: 내 식재량은 ${user.nickname}이며 ${userMessage}' }];

    const response = await openai.chat.completions.create({
      model: 'gpt-4o', messages: messages, tools: tools,
      tool_choices: 'auto' });

    return response.choices[0].message;
  } catch (error) {
    // ...
  }
}

```

2. OpenAI API에 요청 전송 (프롬프트, 대화기록, 사용 가능 함수 목록 포함)

4. AI의 분석 결과를 Controller에 반환

- tools 배열에 log_health_data, get_diet_recommendation 등 애플리케이션과 연동될 함수의 명세서를 정의
- 이 응답(tool_calls가 포함된 메시지)이 aiController.js로 반환되어 실제 기능이 실행

2) 히스토리 기반 개인화 및 고도화된 코칭

파일명	[Controller] aiController.js, chatHistoryController.js [Service] aiService.js [Model] ChatHistory.js
요약	사용자의 대화 기록 (ChatHistory)과 AI 서비스 로직을 활용하여 개인화된 건강 코칭 및 추천을 제공하고, 데이터 기록의 정확도를 높이며, 필요한 경우 사용자에게 추가 정보를 요청하여 고도화된 인터랙션을 구현.

[Schema]

1.ChatHistorySchema는 user와 messages 배열을 포함하여 사용자별 대화 기록을 저장하고 관리.

```
const ChatHistorySchema = mongoose.Schema({
  user: { // 어떤 사용자의 대화 기록인지 연결
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true,
    unique: true, // 한 사용자당 하나의 대화 히스토리 문서만 가짐
  },
  messages: [ChatMessageSchema], // 대화 메시지 배열
}, {
  timestamps: true,
});

const ChatHistory = mongoose.model('ChatHistory', ChatHistorySchema);
```

- ▶ ChatHistory.js
 - unique: true 설정을 통해 한 사용자당 하나의 히스토리 문서만 존재하도록 보장한다. 이 스키마를 통해 저장된 messages가 aiController와 aiService의 history 매개변수로 전달되어 AI의 개인화된 코칭 및 대화 흐름 유지에 활용된다.

[Controller]

1. 클라이언트로부터 받은 message와 "history(과거 대화 기록)"를 aiService.getAiChatResponse 함수로 전달하여 AI 응답을 생성

```
exports.parseAndLogChat = async (req, res) => {
  const user = req.user;
  const { message, history } = req.body;

  if (!message) {
    return res.status(400).json({ message: "메시지를 입력해주세요." });
  }

  try {
    const aiResponseMessage = await aiService.getAiChatResponse(
      message, history || [], user);
  } catch (error) {
    console.error('AI Chat Response Error:', error);
    return res.status(500).json({ message: "AI 응답 생성에 실패했습니다." });
  }
}
```

2. 응답에 tool_calls가 포함되어 있을 경우, 해당 함수를 실행. 특히 get_workout_recommendation 호출 시, 과거 history를 AI 모델에 전달 -> 사용자 이전 대화 기반 개인화된 운동/추천 생성

```
} else if (functionName === 'get_workout_recommendation') {
  const recommendation = await aiService.getSimpleWorkoutRecommendation(history);
  customReply = "오늘은 이 운동 어떠세요?";
  savedData.push({
    type: 'workout_recommendation',
    data: { name: recommendation }
  });
} else if (functionName === 'request_clarification') {
  clarificationQuestion = parsedArg.question;
}
```

3. 함수명 request_clarification 안 붙어 호출되면, 사용자에게 질문할 내용을 추출 -> clarificationQuestions 설정 -> 클라이언트에 반환함으로써 누락된 정보를 보완

- ▶ aiController.js,

파일명	[Controller] aiController.js, chatHistoryController.js [Service] aiService.js [Model] ChatHistory.js
[Service]	<div data-bbox="316 524 1279 1037" data-label="Text"> <pre>const systemPrompt = `당신은 사용자의 건강을 책임지는 AI 코치 '하루핏 매니저'입니다. 항상 친절하고 증기를 부여하는 말투로 대화하세요. 사용자가 '식단 추천' 또는 '운동 추천'을 요청하면, 각각 'get_diet_recommendation', 'get_workout_recommendation' 함수를 호출하세요. 사용자의 메시지에서 '물 목표' 또는 '목표 음수량'과 같은 키워드를 발견하면 'set_user_preferences' 함수를 호출하여 사용자의 설정을 변경하세요. 그 외 식단, 운동, 상태, 물 섭취 정보는 'log_health_data' 함수를 호출하여 데이터를 기록하세요. **[가장 중요한 원칙]** 1. **최신 메시지 우선**: 항상 사용자의 '가장 마지막 메시지'를 기준으로 목표의 종류(식단/운동/상태/설정/추천)를 판단하세요. 이전 대화는 참고만 합니다. 2. **결합된 데이터 구조 생성**: 함수를 호출할 때, 각 타입에 맞는 데이터 구조를 반드시 지켜야 합니다. - **식단 기록 시**: '사과 2개'는 \items 배열에 \[{ "name": "사과", "quantity": 2 }]\`로 저장합니다. - **운동 기록 시**: '스쿼트 10회 3세트'는 \items 배열에 \[{ "name": "스쿼트", "reps": 10, "sets": 3 }]\` 형태로, '30분 달리기'는 \[{ "name": "달리기", "durationMinutes": 30 }]\` 형태로 최대한 상세하게 저장해야 합니다. - **상태 기록 시**: '70kg', '체지방 15%' 등의 값을 정확히 추출해야 합니다. **[기록 규칙]** - 사용자가 '오거했어'처럼 '저항' 운동 내용을 말했지만 세부 정보(시간, 횟수 등)가 없다면, 기록하기 전에 'request_clarification'을 사용해 "요거는 몇 분 하셨나요?"와 같이 반드시 뒤에서 상세 정보를 물어내야 합니다. - '식단' 기록 시 '식사 시간' 정보가 없는 경우라면, 기록 자체가 불가능한 필수 정보가 누락되었을 때만 'request_clarification'으로 질문하세요. - 당신이 질문한 후 사용자가 답변하면, 그 답변과 이전 맥락을 종합하여 'log_health_data'를 다시 호출하세요. `;</pre> </div> <div data-bbox="347 1055 1086 1120" data-label="Text"> <p>1.systemPrompt를 통해 AI의 역할을 '하루핏 매니저'로 정의하고, '최신 메시지 우선' 원칙을 강조하여 대화 맥락을 파악하도록 지시</p> </div> <div data-bbox="316 1135 1279 1536" data-label="Text"> <pre>exports.getAiChatResponse = async (userMessage, history, user) => { try { const messages = [{ role: "system", content: systemPrompt }, ...history, { role: "user", content: `(참고: 내 이름은 \${user.nickname})이야` } {userMessage}]]; const response = await openai.chat.completions.create({ model: "gpt-4o", messages: messages, tools: tools, tool_choice: "auto" }); return response.choices[0].message; } catch (error) { console.error('AI 응답 생성 중 오류 발생:', error); throw new Error('하루핏 AI 매니저의 응답 생성에 실패했습니다.');</pre> </div> <div data-bbox="347 1547 1136 1632" data-label="Text"> <p>2. 사용자의 history (과거 대화 기록)를 messages 배열에 포함하여 gpt-4o 모델에 전달함으로써 대화의 연속성을 유지하고 AI가 이전 대화를 참고하여 답변 및 툴 호출을 할 수 있음</p> </div> <div data-bbox="308 1666 1347 1924" data-label="List-Group"> <p>▶ aiService.js</p> <ul style="list-style-type: none"> - 정의된 tools (함수 목록: log_health_data, set_user_preferences, get_diet_recommendation, get_workout_recommendation, request_clarification)를 AI 모델에 제공하여 사용자의 의도에 따라 적절한 기능을 호출하도록 합니다. 특히 log_health_data 함수에 대한 상세 규칙 (식단, 운동 기록 시 데이터 구조, 필수 정보 누락 시 request_clarification 호출)을 명시하여 정확하고 고도화된 데이터 기록을 유도 </div>

3) 모드별 서비스

파일명	[Controller] aiController.js, [Service] aiService.js [Model] user.js
요약	사용자의 mode 설정에 따라 AI 코칭의 강도, 추천 방식, 프롬프트 내용 등을 동적으로 조절하여 개인화된 코칭 경험 제공.

[흐름]

User 모델에 저장된 mode 값 -> aiController -> aiService 내부 AI 프롬프트 구성에 활용

[Model]

```
const UserSchema = mongoose.Schema({
  nickname: {
    type: String,
    required: [true, '닉네임은 필수입니다.'],
    unique: true,
    trim: true,
  },
  mode: {
    type: String,
    enum: ['easy', 'normal', 'hard'],
    default: 'easy',
  },
  pin: {
```

- 사용자의 mode 정보를 저장하는 스키마 필드를 정의,
- 사용자가 선택한 코칭 모드를 문자열('easy', 'normal', 'hard')로 저장

[Frontend]

프론트엔드의 useAuth.js 혹은 사용하여 사용자 모드 상태를 관리하고, ManagerChat.jsx 컴포넌트가 이 모드 값을 받아서 apiClient를 통해 백엔드로 전달

1. 로컬 스토리지에서 mode 값 초기화

```
export default function useAuth() {
  const [nickname, setNickname] = useState(() => localStorage.getItem('userNickname') || '');
  const [pin, setPin] = useState(() => localStorage.getItem('userPin') || '');
  const [mode, setMode] = useState(() => localStorage.getItem('userMode') || 'normal');
```

2. 로그인/회원가입 시 선택된 mode를 백엔드로 전송

```
const login = useCallback(async (inputNickname, inputPin, inputMode = 'normal') => {
  try {
    const response = await apiClient.post('/users', {
      nickname: inputNickname,
      pin: inputPin,
      mode: inputMode
    });
  } catch {
    // ...
  }
});
```

3. 응답받은 mode 값을 로컬 스토리지에 저장

```
const [ nickname: resNickname, mode: resMode ] = response.data;
localStorage.setItem('userNickname', resNickname);
localStorage.setItem('userPin', inputPin); // PIN은 보안상 서버에서 직접 반환하지 않는 경우가 많으므로 입력값 사용
localStorage.setItem('userMode', resMode);
```


파일명	[Controller] aiController.js, [Service] aiService.js [Model] user.js
	<div data-bbox="422 427 877 456">1. useAuth 훅을 통해 현재 사용자의 모드 가져오기</div> <div data-bbox="405 465 1158 651"> <pre>export default function HomePage() { const location = useLocation() const queryParams = new URLSearchParams(location.search) const initialMode = localStorage.getItem('userMode') queryParams.get('mode') 'normal' const [selectedMode, setSelectedMode] = useState(initialMode); }</pre> </div> <div data-bbox="422 669 951 698">2. ManagerChat 컴포넌트에 selectedMode prop으로 전달</div> <div data-bbox="405 707 1158 873"> <pre><div className="chat-area"> <ManagerChat mode={selectedMode} shouldFocusInput={triggerChatFocus} triggerSource={chatTriggerSource} onDataRefresh={handleDataUpdate} systemMessage={chatSystemMessage} /> </div></pre> </div> <div data-bbox="422 882 1129 934">2. ManagerChat 컴포넌트 내부 useEffect 함수로 로컬스토리지에 저장된 유저 정보와 모드에 따라 초기 메시지를 설정</div> <div data-bbox="405 943 1158 1198"> <pre>useEffect(() => { const initChat = () => { const userNickname = localStorage.getItem('userNickname') '게스트'; const userMode = localStorage.getItem('userMode'); const modeDescriptions = { easy: "편의한 시뮬", normal: "무궁한 권력", hard: "공격의 본능" }; let initialMessages = [{ sender: 'ai', text: "반갑습니다! \${userNickname}님, 허우토큰 함께 건강해줄 준비 되셨나요?" }]; if (userMode) { initialMessages.push({ sender: 'ai', text: `\${userMode.toUpperCase()} 모드에 선택하셨습니다. \${modeDescriptions[userMode]}를 목표로 함께 나아가요!` }); } setMessages(initialMessages); }; }, []);</pre> </div> <div data-bbox="414 1207 1102 1261">3. 백엔드 AI 채팅 API 호출 시 history와 함께 mode 정보는 user 인증 정보에 포함되어 자동으로 전달됨</div> <div data-bbox="405 1267 970 1541"> <pre>const handleSendMessage = async (e) => { e.preventDefault(); if (!input.trim() isLoading) return; const userMessage = { sender: 'user', text: input }; const historyForApi = messages.map(msg => ({ role: msg.sender === 'ai' ? 'assistant' : 'user', content: msg.text })); setMessages(prev => [...prev, userMessage]); setInput(''); setIsLoading(true); try { const response = await axios.post('/api/ai/chat', { message: userMessage.text, history: historyForApi, mode: selectedMode }); } catch (error) { console.error('AI Chat Error:', error); } }</pre> </div> <div data-bbox="248 1615 1356 1765"> <p>▶ 프론트엔드의 useAuth.js ,HomePage.jsx, ManagerChat.jsx 에서의 mode 값 사용 흐름</p> <ul style="list-style-type: none"> -handleSendMessage 함수에서 백엔드 /api/ai/chat 엔드포인트로 메시지를 전송할 때, 명시적으로 req.body에 mode를 포함하지 않는다. 이는 인증 미들웨어(authMiddleware)를 통해 req.user 객체에 사용자의 mode 정보가 이미 포함되어 있기 때문 </div>

파일명	[Controller] aiController.js, [Service] aiService.js [Model] user.js
[Backend]	<div data-bbox="368 427 1195 524" data-label="Text"> <pre>// PUT /api/users/mode - 사용자 모드 변경 router.put('/mode', authMiddleware, userController.updateUserMode);</pre> </div> <div data-bbox="368 524 536 551" data-label="Text"> <p>userRoutes.js</p> </div> <div data-bbox="363 573 1214 1088" data-label="Text"> <pre>exports.updateUserMode = async (req, res) => { try { const { mode } = req.body; const validModes = ['easy', 'normal', 'hard']; if (!mode !validModes.includes(mode)) { return res.status(400).json({ message: '올바른 모드를 선택해주세요.' }); } const updatedUser = await User.findByIdAndUpdate(req.user._id, { mode: mode }, { new: true }); if (!updatedUser) { return res.status(404).json({ message: '사용자를 찾을 수 없습니다.' }); } res.status(200).json({ message: '모드가 성공적으로 변경되었습니다.', user: updatedUser }); } catch (error) { console.error('모드 업데이트 중 오류:', error); res.status(500).json({ message: '모드 업데이트 중 서버 오류가 발생했습니다.' }); } };</pre> </div> <div data-bbox="363 1088 566 1115" data-label="Text"> <p>userController.js</p> </div> <div data-bbox="242 1162 1367 1270" data-label="List-Group"> <ul style="list-style-type: none"> ▶ userRoutes.js와 userController.js를 통해 사용자 모드 정보를 업데이트 <ul style="list-style-type: none"> - req.user._id를 사용하여 현재 로그인된 사용자를 식별하고, 해당 사용자의 mode 필드를 받은 값으로 업데이트 </div> <div data-bbox="213 1314 469 1346" data-label="Section-Header"> <h4>[Backend Controller]</h4> </div> <div data-bbox="352 1350 1243 1720" data-label="Text"> <pre>exports.parseAndLogChat = async (req, res) => { const user = req.user; const { message, history } = req.body; if (!message) { return res.status(400).json({ message: '메시지를 입력해주세요.' }); } try { const aiResponseMessage = await aiService.getAiChatResponse(message, history [], user);</pre> </div> <div data-bbox="360 1727 537 1758" data-label="Text"> <p>aiController.js</p> </div> <div data-bbox="296 1765 1355 1874" data-label="List-Group"> <ul style="list-style-type: none"> - 클라이언트로부터 받은 message와 history 외에 현재 로그인된 user 객체를 aiService.getAiChatResponse 함수로 전달한다. 이 user 객체에는 user.js 모델에 정의된 mode 정보가 포함되어 있어, AI 서비스 계층에서 사용자의 모드에 따른 분기 처리가 가능하다. </div>

4) 실시간 UI 피드백

파일명	[Trigger] ManagerChat.jsx [Orchestrator] HomePage.jsx [Data Access] useApi.js [Presentation] 카드 컴포넌트들
요약	AI 챗봇이 사용자의 정보를 기록/업데이트하게될 시 콜백과 API의 메커니즘을 통해 해당 UI 카드가 즉각 업데이트된다.

[Trigger]: 사용자 액션을 감지하고 데이터 새로고침을 요청하는 시작점

1. 백엔드 API 호출 성공 시 onDataRefresh 콜백 호출

```
const handleMessage = async (e) => {
  e.preventDefault();
  if (!input.trim() || isLoading) return;
  const userMessage = { sender: 'user', text: input };
  const historyForApi = messages.map(msg => ({ role: msg.sender === 'ai' ?
    'assistant' : 'user', content: msg.text }));
  setMessages(prev => [...prev, userMessage]);
  setInput('');
  setIsLoading(true);
  try {
    const response = await apiClient.post('/ai/parse-and-log', { message: userMessage,
      text: historyForApi });
    // 1. AI의 답변 일련을 먼저 표시합니다.
    const aiReplyMessage = { sender: 'ai', text: response.data.reply };
    setMessages(prev => [...prev, aiReplyMessage]);
    const savedData = response.data.savedData;
    // 2. 백엔드로부터 받은 데이터(기록 또는 추천)가 있다면, 무조건 HomePage로
    전달하여 피드를 업데이트 합니다.
    if (savedData && savedData.length > 0) {
      // onDataRefresh
      onDataRefresh(savedData);
    }
  } catch (error) {
    console.error('API 호출 실패:', error);
  }
}
```

부트인 HomePage.jsx에서 prop으로 받은 함수

ManagerChat.jsx

[Orchestrator]: 트리거로부터 신호를 받아 필요한 데이터를 재요청하도록 지시하는 중간 관리자

1. useApi 훅을 사용해 각 데이터 타입별로 데이터 상태 및 refetch 함수 가져옴

```
const { data: statusData, error: statusError, refetch: refetchStatus } = useApi(
  API_ENDPOINTS.STATUS_TODAY);
const { data: dietData, error: dietError, refetch: refetchDiet } = useApi(
  API_ENDPOINTS.DIET_TODAY);
const { data: workoutData, error: workoutError, refetch: refetchWorkout } =
  useApi(API_ENDPOINTS.WORKOUT_TODAY);
const { data: calendarData, error: calendarError, refetch: refetchCalendar } =
  useApi(API_ENDPOINTS.CALENDAR_SUMMARY);
```

2. 데이터 새로고침을 요청하는 콜백 함수 정의 (ManagerChat에 전달될 것)

```
const handleDataUpdate = useCallback((savedDataArray) => {
  if (!savedDataArray || savedDataArray.length === 0) {
    refetchStatus();
    refetchDiet();
    refetchWorkout();
    refetchCalendar();
  }
  return;
}, []);
```

HomePage.jsx

- handleDataUpdate 함수: 각 useApi 훅에서 반환된 refetch 함수들(refetchStatusData, 등)을 호출해 해당 UI 카드들의 데이터를 최신으로 업데이트하도록 지시, 이 기능(데이터 새로고침)을 onDataRefresh라는 prop의 이름으로 ManagerChat에 전달한다.

파일명	[Trigger] ManagerChat.jsx	[Orchestrator] HomePage.jsx
	[Data Access] useApi.js	[Presentation] 카드 컴포넌트들

[Data Access]: 백엔드 API를 호출하여 데이터를 가져오고, 로딩/에러 상태를 관리하며, 데이터 재요청 메커니즘을 제공한다. 프론트엔드에서 데이터를 최신 상태로 유지하고, 사용자의 행동에 따라 UI가 즉각적으로 반응하도록 하는 데 필수적인 역할.

3. API 응답 데이터를 data 상태에 저장한다. (UI 업데이트 트리거) setData를 호출하면 useApi 훅을 사용하는 컴포넌트들이 변경된 data를 감지하여 리렌더링한다.

```
const newData = response.data;
setData(prevData => {
  if (typeof newData === 'object' && newData !== null && !Array.isArray(newData)) {
    return { ...prevData, ...newData };
  }
  if (Array.isArray(newData)) {
    return [...prevData, ...newData];
  }
  return newData;
});
```

4. 컴포넌트 마운트 시 즉시 데이터 fetching

```
useEffect(() => {
  if (fetchImmediately) {
    request();
  }
}, [fetchImmediately, request]);
```

5. 외부에서 데이터 재요청을 가능하게 하는 'refetch' 함수 제공

```
const refetch = useCallback(() => {
  return request();
}, [request]);
```

- setData가 호출되면 React는 data 상태가 변경되었음을 인지하고, 이 훅을 사용하는 모든 View 컴포넌트들의 리렌더링을 스케줄링하여 즉각적으로 UI가 업데이트되게 한다.
- refetch 함수: useApi 훅의 외부(예: HomePage.jsx)에서 request 함수를 수동으로 다시 호출하도록 트리거할 수 있게 한다. ManagerChat.jsx에서 데이터가 기록/추천 되었을 때 HomePage.jsx의 onDataRefresh 콜백이 이 refetch를 호출함으로써, 사용자의 행동에 대한 즉각적인 UI 업데이트가 시작된다.

[Presentation]: 재요청된 최신 데이터를 받아 화면에 시각적으로 반영한다.

HomePage.jsx로부터 useApi.js를 통해 업데이트된 data prop을 전달받아 해당 데이터를 기반으로 UI를 다시 렌더링한다.

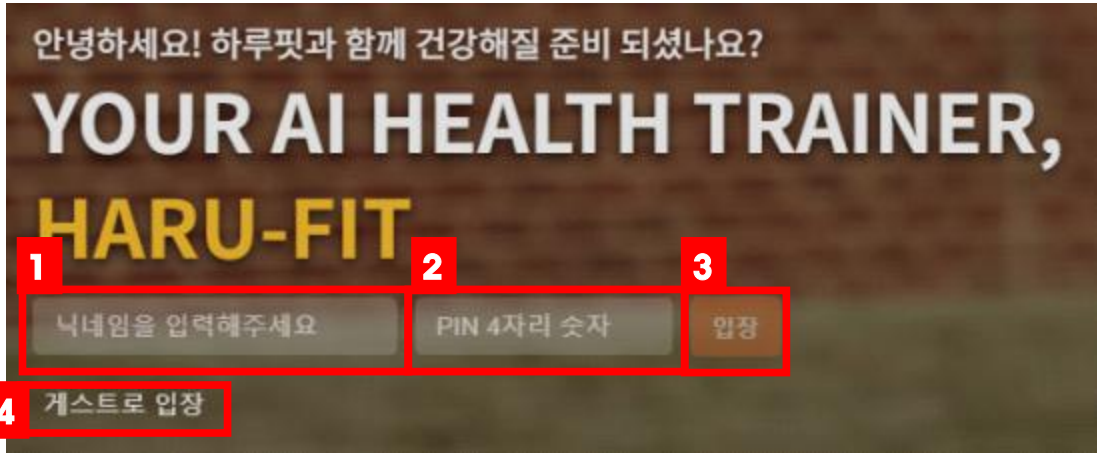
```
export default function DietCard({ onExpand, mode, data }) {
  const eatenMeals = data?.eatenMeals || [];
  const waterIntake = data?.waterIntake || { current: 0, goal: 2 };
  const recommendedMeal = data?.recommendedMeal;

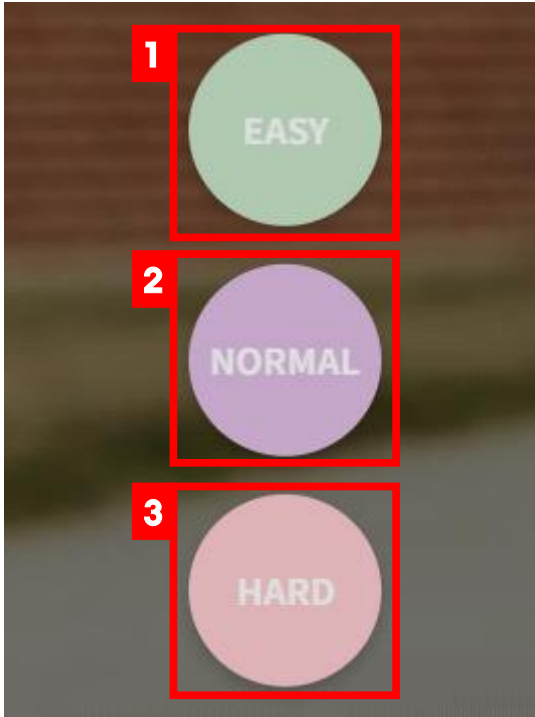
  const breakfast = eatenMeals.find(m => m.type === 'breakfast') || {
    menu: '기록 없음', kcal: 0 };

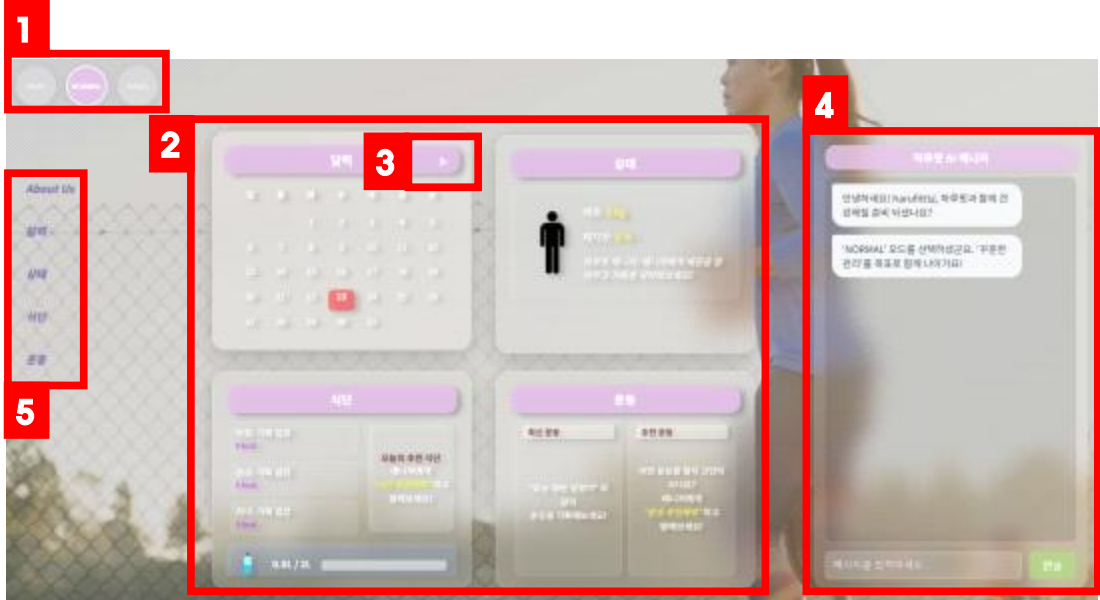
  <div className="diet-item">
    <p>아침: {breakfast.menu}</p> <span className="diet-kcal">
      {breakfast.kcal}</span>
    </div>
```

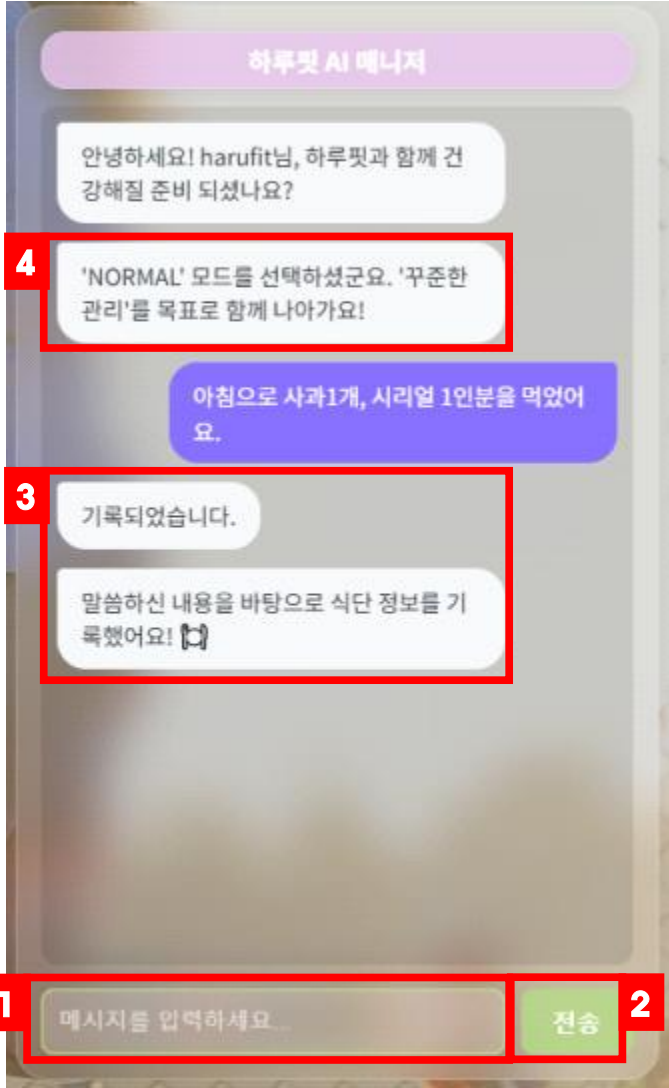
▶ 예시: DietCard.jsx (식단 카드맵)

3.5 인터페이스 설계

인터페이스명	로그인 및 시작 페이지 (IndexPage.jsx)
설명	닉네임과 비밀번호를 이용한 간편 로그인 및 회원가입 ※향후 개선 사항: bcrypt 비밀번호 암호화, 소셜 로그인(Google/Kakao) 도입
	
이벤트	액션
1. 닉네임 입력창	서비스에서 사용할 닉네임 입력
2. 비밀번호 입력창	로그인에 사용할 4자리 비밀번호 입력
3. 입장 버튼	비밀번호 유효성 검사(오류 시 메시지 표시), 통과 시 로그인/가입 처리. 신규/기존 사용자에게 따라 모드 선택/메인 대시보드로 이동
4. 게스트로 입장 버튼	'Guest' 계정으로 서비스 진입 후, 모드 선택 페이지로 이동

인터페이스명	모드 선택 (IndexPage.jsx)
설명	로그인 후 UI 테마 및 AI 매니저 모드를 설정하는 초기 화면
	
이벤트	액션
1. EASY 버튼 클릭	'EASY' 모드(편안한 시작) 설정 및 메인 대시보드로 이동
2. NORMAL 버튼 클릭	'NORMAL' 모드(꾸준한 관리) 설정 및 메인 대시보드로 이동
3. HARD 버튼 클릭	'HARD' 모드(강력한 변화) 설정 및 메인 대시보드로 이동

인터페이스명	메인 대시보드 (HomePage.jsx)
설명	사용자의 모든 건강 정보를 요약한 4개의 카드와 AI 코치로 구성된 핵심 화면
	
이벤트	액션
1. 모드 변경 버튼	클릭 시 해당 모드로 UI 테마 및 AI 코칭 강도 변경
2. 달력/상태/식단/운동 카드	AI 채팅으로 기록된 해당 날짜의 건강 정보를 요약하여 표시
3. 상세보기 버튼 (▶)	클릭 시 해당 카드의 상세 정보가 담긴 모달(Modal) 창 열림
4. AI 코치 채팅창	자연어를 입력하여 건강 정보 기록 및 AI와 상호작용
5. 사이드바 메뉴 (About Us 등)	클릭 시, 해당 기능의 상세 정보가 담긴 모달(Modal) 창 열림

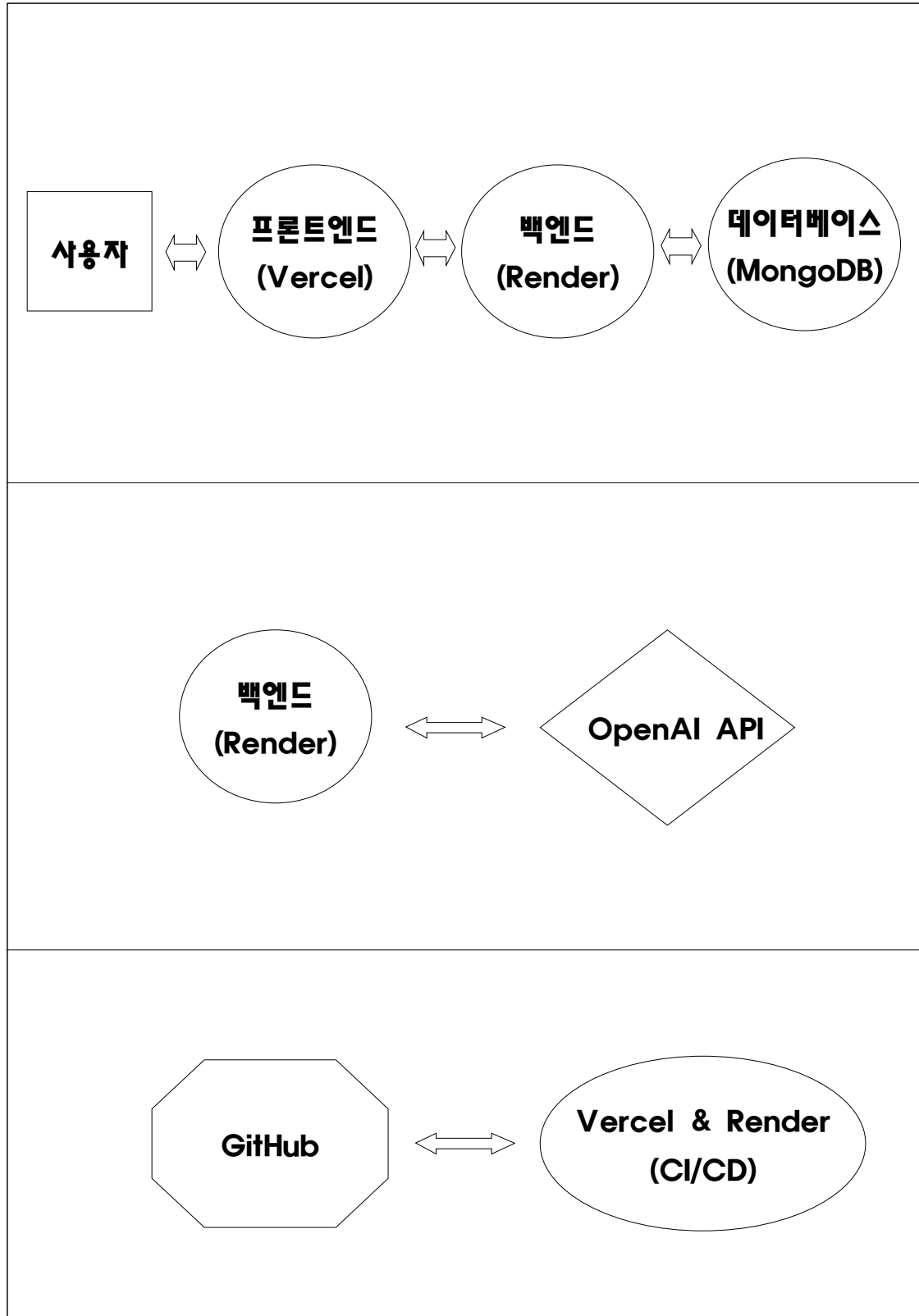
인터페이스명	AI 코치 채팅창 (ManagerChat.jsx)
설명	사용자와 AI 코치 간의 모든 상호작용이 이루어지는 핵심 인터페이스
 <p>The screenshot shows a mobile app interface for an AI coach. At the top, a pink header reads '하루핏 AI 매니저'. Below it, a white chat bubble says '안녕하세요! harufit님, 하루핏과 함께 건강해질 준비 되셨나요?'. A red box labeled '4' highlights a white system message bubble: ''NORMAL' 모드를 선택하셨군요. '꾸준한 관리'를 목표로 함께 나아가요!'. Below that, a blue chat bubble says '아침으로 사과1개, 시리얼 1인분을 먹었어요.'. A red box labeled '3' highlights a white system response bubble: '기록되었습니다. 말씀하신 내용을 바탕으로 식단 정보를 기록했어요! 📅'. At the bottom, a red box labeled '1' highlights the text input field with the placeholder '메시지를 입력하세요...'. To its right, a green button labeled '전송' is highlighted with a red box labeled '2'.</p>	
이벤트	액션
1. 메시지 입력창	식단, 운동 등 기록할 내용을 자연어로 입력
2. 전송 버튼	AI 코치에게 메시지 전송 및 분석 요청
3. AI 응답 메시지	AI의 분석 결과(기록 완료, 추천 등)를 대화 형태로 표시
4. 시스템 메시지	모드 변경 등 채팅 외부 이벤트에 대한 시스템 알림 표시

인터페이스명	요약 정보 카드 (대표 예시: DietCard.jsx)
설명	메인 대시보드에 표시되는 4개 카드 중 하나로, 특정 주제(식단, 운동 등)의 핵심 정보를 요약하여 보여줍니다.
<div></div>	
이벤트	액션
1. 식사별 요약	채팅으로 기록된 아침/점심/저녁 식사 메뉴와 칼로리 요약 표시
2. 수분 섭취 바	목표량 대비 현재 수분 섭취량을 진행률 바로 시각화
3. 추천 안내	AI 식단 추천 기능 사용을 유도하는 안내 메시지 표시
4. 상세보기 버튼 (▶)	클릭 시 해당 카드의 상세 정보가 담긴 모달 창 열림

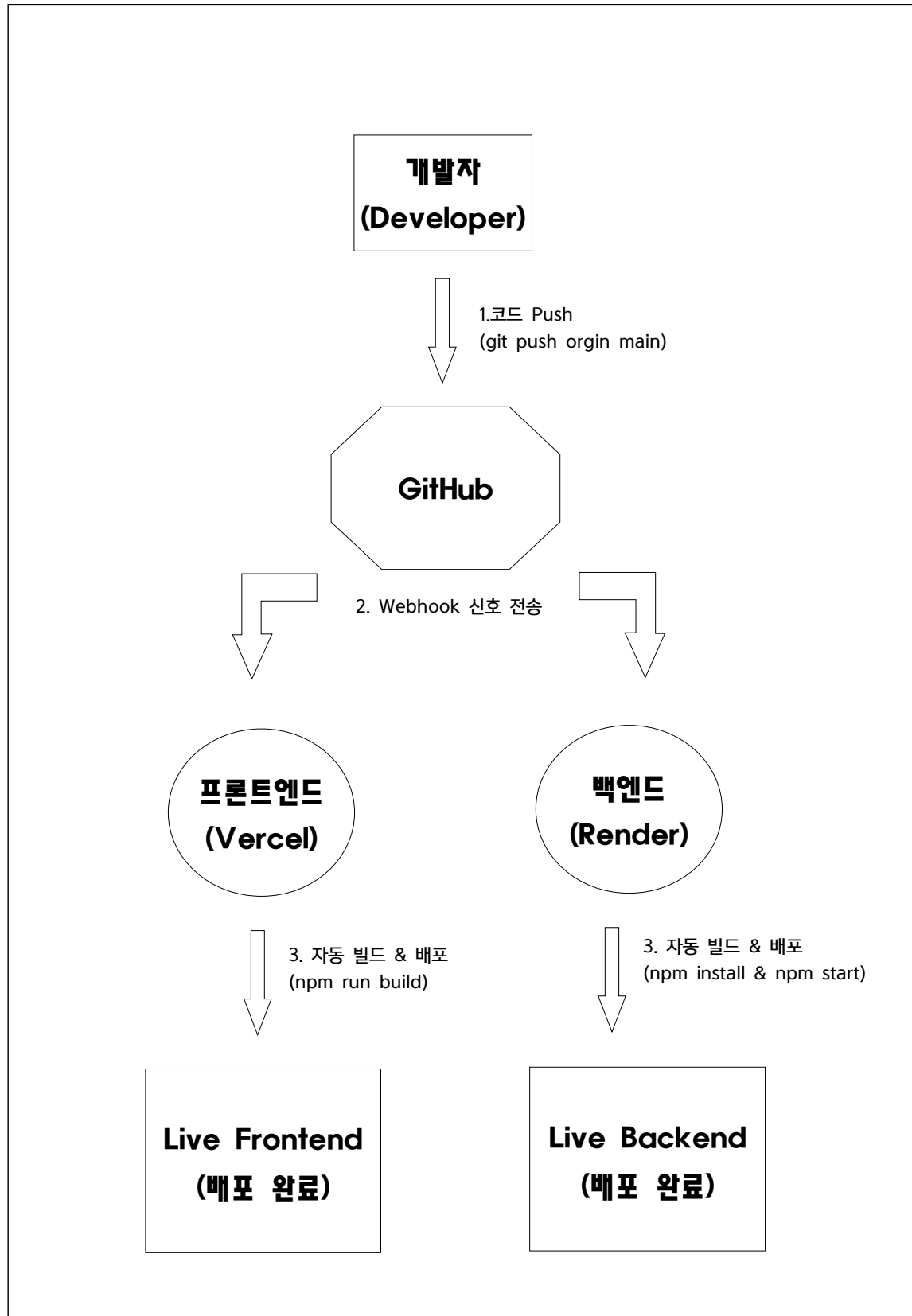
인터페이스명	식단 상세 모달 (DietExpanded.jsx)
설명	선택한 날짜의 식단 및 수분 섭취에 대한 모든 기록과 통계를 보여주는 상세 정보창
이벤트	액션
1. AI 매니저 조언	AI가 사용자의 식단 기록을 바탕으로 제공하는 일일 조언 표시
2. 영양소 총합	해당일 섭취한 총 칼로리, 탄수화물, 단백질, 지방 정보 요약 표시
3. 수분 섭취 현황	목표량 대비 현재 섭취량을 원형 그래프로 시각화하여 표시
4. 식사별 기록 목록	아침/점심/저녁/간식별로 기록된 메뉴와 칼로리 상세 표시
5. 식단 목표 현황	일일 목표 칼로리 대비 현재 섭취량 달성률을 진행률 바로 표시
6. 주간 달성률 그래프	요일별 칼로리 목표 달성률을 막대그래프로 시각화하여 표시
7. 추천 식단 목록	AI 코치가 제안하는 아침/점심/저녁 추천 메뉴와 칼로리 표시
8. '식단 기록하기' 버튼	클릭 시, AI 채팅창으로 이동하여 추가적인 식단 기록을 유도
9. 닫기 버튼 (X)	모달 창을 닫고 메인 대시보드로 복귀

5.배포

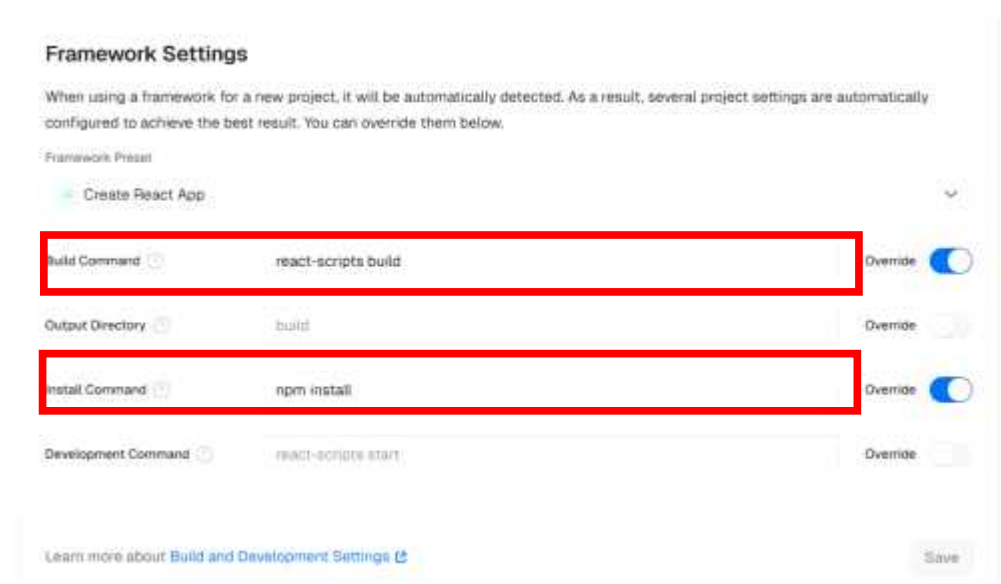
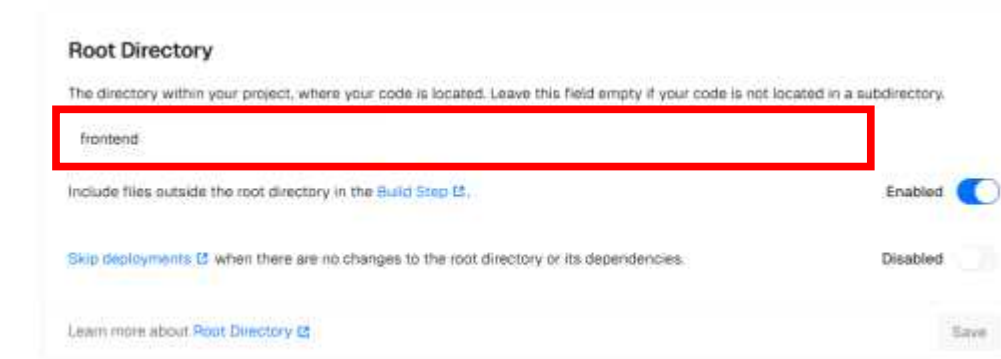
1) 전체 시스템 구성도



2) CI/CD 파이프라인 흐름도



3) 프론트엔드 배포(Vercel) - GitHub에 있는 frontend 폴더를 Vercel 계정과 연결

화면명	Vercel 대시보드의 Project Settings
 <p>Framework Settings</p> <p>When using a framework for a new project, it will be automatically detected. As a result, several project settings are automatically configured to achieve the best result. You can override them below.</p> <p>Framework Preset: Create React App</p> <p>Build Command: react-scripts build (Override: On)</p> <p>Output Directory: build (Override: Off)</p> <p>Install Command: npm install (Override: On)</p> <p>Development Command: react-scripts start (Override: Off)</p> <p>Learn more about Build and Development Settings</p> <p>Save</p>	
 <p>Root Directory</p> <p>The directory within your project, where your code is located. Leave this field empty if your code is not located in a subdirectory.</p> <p>Root Directory: frontend</p> <p>Include files outside the root directory in the Build Step: Enabled</p> <p>Skip deployments when there are no changes to the root directory or its dependencies: Disabled</p> <p>Learn more about Root Directory</p> <p>Save</p>	
<ul style="list-style-type: none"> ● GitHub 저장소의 frontend 폴더를 Root Directory로 지정하고, React 프레임워크의 빌드 설정을 구성 ● 환경 변수(VITE_API_URL)에 백엔드 서버 주소를 등록하여 프론트엔드와 백엔드를 연결 ● main 브랜치에 푸시할 때마다 자동으로 배포가 진행되도록 CI/CD를 설정 	

4) 백엔드 배포 (Render)- GitHub에 있는 backend 폴더를 연결.

화면명	Render 대시보드의 Web Service 설정												
<div><div><div><div>Build Command</div><div>Render runs this command to build your app before each deploy</div><div>backend/ \$ npm install</div><div>Edit</div></div><div><div>Pre-Deploy Command <small>Optional</small></div><div>Render runs this command before the start command. Useful for database migrations and static asset uploads.</div><div>backend/ \$</div><div>Edit</div></div><div><div>Start Command</div><div>Render runs this command to start your app with each deploy</div><div>backend/ \$ npm start</div><div>Edit</div></div><div><div>Auto-Deploy</div><div>By default, Render automatically deploys your service whenever you update its code or configuration. Disable to handle deploys manually. Learn more</div><div>On Commit</div><div>Edit</div></div><div><div>Deploy Hook</div><div>Your private URL, to trigger a deploy for this server. Remember to keep this a secret.</div><div><div>Generate hook</div></div></div></div></div>													
<div><div><div><div>Root Directory <small>Optional</small></div><div>If set, Render runs commands from this directory instead of the repository root. Additionally, code changes outside of this directory do not trigger an auto-deploy. Most commonly used with a monorepo.</div><div>backend</div><div>Edit</div></div></div></div>													
<div><div><div><div>Environment Variables</div><div>Set environment-specific config and secrets (such as API keys), then read those values from your code. Learn more</div><div><div>Export</div><div>Edit</div></div><table><thead><tr><th>Key</th><th>Value</th><th></th></tr></thead><tbody><tr><td>JWT_SECRET</td><td>*****</td><td>Generate</td></tr><tr><td>MONGO_URI</td><td>*****</td><td>Generate</td></tr><tr><td>OPENAI_API_KEY</td><td>*****</td><td>Generate</td></tr></tbody></table></div></div></div>		Key	Value		JWT_SECRET	*****	Generate	MONGO_URI	*****	Generate	OPENAI_API_KEY	*****	Generate
Key	Value												
JWT_SECRET	*****	Generate											
MONGO_URI	*****	Generate											
OPENAI_API_KEY	*****	Generate											
<div><div><div><div>● GitHub 저장소의 backend 폴더를 연결하고, npm install 및 node server.js를 빌드/시작 명령어로 지정함.</div><div>● DB 접속 정보(MONGO_URI)와 API 키 등 민감 정보는 환경 변수로 안전하게 설정함.</div></div></div></div>													

5) 데이터베이스 설정 (MongoDB Atlas)

화면명	MongoDB Atlas 대시보드의 Network Access & Connect 설정																		
<div><div>Network Access</div><div><div>IP Address List</div><div>Firewall</div><div>Whitelist Endpoints</div></div><div><div>ADD IP ADDRESS</div></div><div><div>You will only be able to connect to your cluster from the following list of IP Addresses.</div><div><table><thead><tr><th>IP Address</th><th>Comments</th><th>Status</th><th>Actions</th></tr></thead><tbody><tr><td>203.146.234.20/32</td><td>(Created as part of the Auto Setup process)</td><td>Active</td><td>EDIT DELETE</td></tr><tr><td>18.55.142.14/32 (includes your current IP address)</td><td></td><td>Active</td><td>EDIT DELETE</td></tr><tr><td>0.0.0.0/0 (includes your current IP address)</td><td></td><td>Active</td><td>EDIT DELETE</td></tr></tbody></table></div></div><div><div>Clusters</div><div><div>Cluster0</div><div>Connect</div><div>View Monitoring</div><div>Browse Collections</div></div><div><div>Enhance Your Experience</div><div>For productive throughput and richer metrics, upgrade to a dedicated cluster tier.</div><div>Upgrade</div></div><div><div>Connectivity</div><div>44.8</div><div>100.00 R/s</div><div>CL2 143.95 R/s</div><div>Cluster Size</div><div>118.28 MB / 512.00 MB (23%)</div></div><div><div>Version</div><div>3.0.8</div><div>API / Shell (ip-northeast-0)</div><div>PM</div><div>Replica Set - 3 nodes</div><div>Backup</div><div>Private</div><div>Direct API Access</div><div>Atlas RL</div><div>Atlas Search</div></div><div><div>Connect</div><div>Cluster0</div><div>Cluster0</div></div><div><div>ADD TAG</div></div></div></div> <div><div>Connecting with MongoDB Driver</div><div><div>1. Select your driver and version</div><div>We recommend installing and using the latest driver version.</div><div><div>Driver</div><div>Node.js</div><div>Version</div><div>6.7 or later</div></div></div><div><div>2. Install your driver</div><div>Run the following on the command line</div><div><div>npm install mongodb</div></div><div><div>View MongoDB Node.js Driver installation instructions.</div></div></div><div><div>3. Add your connection string into your application code</div><div>Use this connection string in your application</div><div><div>View full code sample</div></div><div><div>mongodb+srv://harufit:<db_password>@cluster0.o1ob3gd.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0</div></div><div><div>Replace <db_password> with the password for the harufit database user. Ensure any option params are URL encoded.</div></div></div></div> <tr><td colspan="2"><div><div>클라우드 데이터베이스 서비스인 MongoDB Atlas에 무료 등급 클러스터를 생성.</div><div>Render 백엔드 서버만 접속을 허용하도록 네트워크 접근을 제어하고, 생성된 연결 문자열(Connection String)을 백엔드의 환경 변수로 전달하여 연동을 완료.</div></div></td></tr>		IP Address	Comments	Status	Actions	203.146.234.20/32	(Created as part of the Auto Setup process)	Active	EDIT DELETE	18.55.142.14/32 (includes your current IP address)		Active	EDIT DELETE	0.0.0.0/0 (includes your current IP address)		Active	EDIT DELETE	<div><div>클라우드 데이터베이스 서비스인 MongoDB Atlas에 무료 등급 클러스터를 생성.</div><div>Render 백엔드 서버만 접속을 허용하도록 네트워크 접근을 제어하고, 생성된 연결 문자열(Connection String)을 백엔드의 환경 변수로 전달하여 연동을 완료.</div></div>	
IP Address	Comments	Status	Actions																
203.146.234.20/32	(Created as part of the Auto Setup process)	Active	EDIT DELETE																
18.55.142.14/32 (includes your current IP address)		Active	EDIT DELETE																
0.0.0.0/0 (includes your current IP address)		Active	EDIT DELETE																
<div><div>클라우드 데이터베이스 서비스인 MongoDB Atlas에 무료 등급 클러스터를 생성.</div><div>Render 백엔드 서버만 접속을 허용하도록 네트워크 접근을 제어하고, 생성된 연결 문자열(Connection String)을 백엔드의 환경 변수로 전달하여 연동을 완료.</div></div>																			

6.1 문제점 및 개선방안

현재 하루핏의 핵심 기능은 외부 고성능 AI 모델(GPT-4o)에 의존하고 있어, 다음과 같은 명확한 한계점과 개선 과제를 가지고 있다.

- 문제점 1: 범용 LLM 기반 자연어 처리(NLP)의 한계

현재의 GPT-4o 모델은 훌륭하지만, 헬스케어라는 특수 도메인에 100% 최적화되지 않는다. 이로 인해 가끔 사용자의 미묘한 맥락(감정, 핑계 등)을 놓치거나, 부정확한 영양 정보를 제공할 가능성이 존재한다.

개선 방안 → 도메인 특화 NLP 모델 고도화:

이 문제를 해결하기 위해, 하루핏의 실제 대화 데이터를 활용하여 **언어 모델을 미세조정(Fine-tuning)**하고, 검증된 자체 식품 영양 DB를 연동할 계획이다. 이를 통해 AI의 맥락 및 감정 이해 능력을 강화하고, 정보의 정확성을 확보하여 더욱 정교하고 신뢰도 높은 코칭을 제공하게 된다.

- 문제점 2: 높은 운영 비용 및 외부 서비스 의존성

모든 대화 처리 시 외부 API를 호출하므로, 사용자 수가 증가할수록 운영 비용이 크게 증가하며 외부 서비스의 안정성에 따라 우리 서비스가 직접적인 영향을 받는다.

개선 방안 → 능동적 관리 시스템 및 경량 모델 도입:

이를 해결하기 위해, AI가 기록된 데이터 패턴을 스스로 분석하여 먼저 사용자에게 개입하는 능동적 관리 시스템을 구축하여 불필요한 API 호출을 줄일 예정이다. 장기적으로는 하루핏의 대화 데이터로 미세조정한(Fine-tuned) 경량화 언어 모델(sLLM)을 도입하여 비용 효율성과 서비스 독립성을 확보하는 것을 목표로 한다.

6.2 결론

기존 헬스케어 앱이 사용자에게 주었던 ‘관리의 피로감’을 해결하고자, [하루핏]은 ‘대화’라는 가장 인간적인 소통 방식을 기술의 중심에 두었다. 이번 프로젝트를 통해, 사용자의 자연스러운 대화를 이해하고 반응하는 ‘AI 웰니스 코치’의 기술적 가능성을 성공적으로 검증했으며, 안정적인 서비스를 배포하며 그 첫걸음을 내딛게 되었다.

하지만 [하루핏]이 궁극적으로 추구하는 목표는 단순히 편리한 기록 도구가 아니다. [하루핏]의 최종 비전은, AI가 사용자의 과거와 현재를 기억하며 ‘기능’을 넘어 ‘관계’를 맺고, ‘목소리’라는 가장 인간적인 입력과 ‘카드’라는 가장 직관적인 출력을 결합하여, 건강 관리를 하고 있다는 사실조차 잊게 만드는 진정한 ‘AI 웰니스 파트너’로 진화하는 것이다.