# AI API 연동 Application 구현방안 레포트 표준 프레임워크

## 📋 개요

본 프레임워크는 {Application} 구현방안을 간소하고 실용적으로 작성하기 위한 표준 템플릿입니다.

**적용 변수**:

* {Application}: 개발할 애플리케이션명
* {Database}: 사용할 데이터베이스 유형
* {External System}: AI API외에 연동할 외부 시스템/API
* {AI API}: AI API 명

## 🏗️ 시스템 아키텍처 개요

### 전체 구성도

[사용자] → [Frontend] → [Backend API] → [{External System} API]

↓ ↓

[{Database}] [외부 데이터]

↓ ↓

[Python AI Service] → [Vector DB]

↓

[AI API]

### 핵심 기술 스택

* **Frontend**: React 18 + TypeScript + UI Framework
* **Backend**: Spring Boot 3.x / Node.js + REST API
* **Database**: {Database} (PostgreSQL/MongoDB/MySQL)
* **AI Service**: Python + FastAPI + LangChain
* **External API**: {External System} API
* **AI API:** {AI API}
* **Infrastructure**: Cloud Platform + Container + CI/CD

## 🔬 Vector DB 구축 및 RAG 파이프라인 설계

### Vector Database 선택

**추천 옵션**:

* **Chroma**: 오픈소스, 빠른 프로토타이핑
* **Pinecone**: 관리형, 대규모 확장
* **Weaviate**: 하이브리드 검색 지원

### Vector 스토어 구성

# 데이터 임베딩 스키마 예시

{

"id": "unique\_id",

"content": "원본 텍스트 또는 데이터",

"embedding": [0.1, 0.2, ...], # 768차원 벡터

"metadata": {

"category": "분류",

"source": "{External System}",

"timestamp": "2024-03-15",

# 도메인별 추가 메타데이터

}

}

### RAG 파이프라인 구조

# 의사코드

def rag\_pipeline(user\_query):

# 1. 쿼리 임베딩

query\_embedding = embedding\_model.encode(user\_query)

# 2. 유사도 검색

relevant\_docs = vector\_db.similarity\_search(

query\_embedding,

k=10,

filters=metadata\_filters

)

# 3. 컨텍스트 구성

context = format\_context(relevant\_docs)

# 4. LLM 생성

response = llm.generate(

prompt=prompt\_template,

context=context

)

return response

## 🔌 {External System} API 연동 파이프라인

### API 연동 구조

# 의사코드

class ExternalAPIClient:

def \_\_init\_\_(self):

self.api\_key = config.EXTERNAL\_API\_KEY

self.base\_url = config.EXTERNAL\_API\_URL

async def fetch\_data(self, params):

# 1. API 호출

response = await http\_client.get(

url=f"{self.base\_url}/endpoint",

headers={"Authorization": f"Bearer {self.api\_key}"},

params=params

)

# 2. 데이터 정규화

normalized\_data = self.normalize(response.data)

# 3. 벡터화 (필요시)

if requires\_embedding:

embedding = embedding\_model.encode(normalized\_data)

await vector\_db.store(embedding, metadata)

return normalized\_data

### 데이터 동기화 전략

* **실시간 동기화**: Webhook, WebSocket
* **배치 동기화**: Cron Job, Scheduled Task
* **캐싱 전략**: Redis, In-Memory Cache

## 💾 {Database} 데이터베이스 구축

### 데이터베이스 설계 원칙

-- 핵심 테이블 구조 예시

-- 1. 사용자 테이블

CREATE TABLE users (

id BIGSERIAL PRIMARY KEY,

email VARCHAR(255) UNIQUE NOT NULL,

-- 도메인별 필수 필드

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- 2. 외부 시스템 연동 테이블

CREATE TABLE external\_system\_data (

id BIGSERIAL PRIMARY KEY,

user\_id BIGINT REFERENCES users(id),

external\_id VARCHAR(100),

sync\_status VARCHAR(20),

raw\_data JSONB,

processed\_data JSONB,

synced\_at TIMESTAMP

);

-- 3. AI 처리 결과 테이블

CREATE TABLE ai\_results (

id BIGSERIAL PRIMARY KEY,

user\_id BIGINT REFERENCES users(id),

input\_data JSONB,

result\_data JSONB,

vector\_id VARCHAR(100),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- 인덱스 생성

CREATE INDEX idx\_sync\_status ON external\_system\_data(sync\_status);

CREATE INDEX idx\_user\_created ON users(created\_at DESC);

### 데이터 초기화 전략

* **시드 데이터**: 개발/테스트용 기본 데이터
* **마이그레이션**: Flyway, Liquibase 활용
* **백업 정책**: 일일 백업, 3-2-1 규칙

## 🐍 Python AI 서비스 개발

### FastAPI 기반 서비스 구조

# main.py 의사코드

from fastapi import FastAPI

from langchain import LLMChain

app = FastAPI()

# AI 서비스 엔드포인트

@app.post("/ai/generate")

async def generate\_response(request: AIRequest):

# 1. 컨텍스트 조회

context = await get\_relevant\_context(request.query)

# 2. 프롬프트 구성

prompt = build\_prompt(

template=PROMPT\_TEMPLATE,

context=context,

user\_data=request.user\_data

)

# 3. LLM 호출

response = await llm\_chain.arun(prompt)

# 4. 후처리

processed\_response = post\_process(response)

return {"result": processed\_response}

### {AI API} 프롬프트 템플릿

PROMPT\_TEMPLATE = """

당신은 {Application}의 AI 어시스턴트입니다.

## 사용자 정보

{user\_context}

## 관련 데이터

{relevant\_data}

## 요청사항

{user\_query}

위 정보를 바탕으로 다음 형식으로 응답하세요:

1. 핵심 답변

2. 구체적 실행 방안

3. 추가 권장사항

"""

## 컨테이너화 Dockerfile

FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

## 프롬프팅 예시

| o: AI 연동 Application 구현 방안을 작성해 주세요. [요구사항] - Application 목적: 건강검진 결과와 직업군 정보를 기준으로 건강 개선을 위해 AI로 부터 최대 5개의 추천 미션을 받음  - 실시간 API 호출을 줄이고 Vector 검색으로 통일하여 응답 속도 개선되도록 Vector DB 구성 - 환경변수  - Application: suggest\_mission  - Database: PostGreSQL  - External System: 건강관리공단  - AI API: Claude API [참고자료] AI API 연동 Application 구현방안 레포트 표준 프레임워크 [결과형식] 아티팩트 |
| --- |