Application Analysis RAG Service

📋 서비스 개요

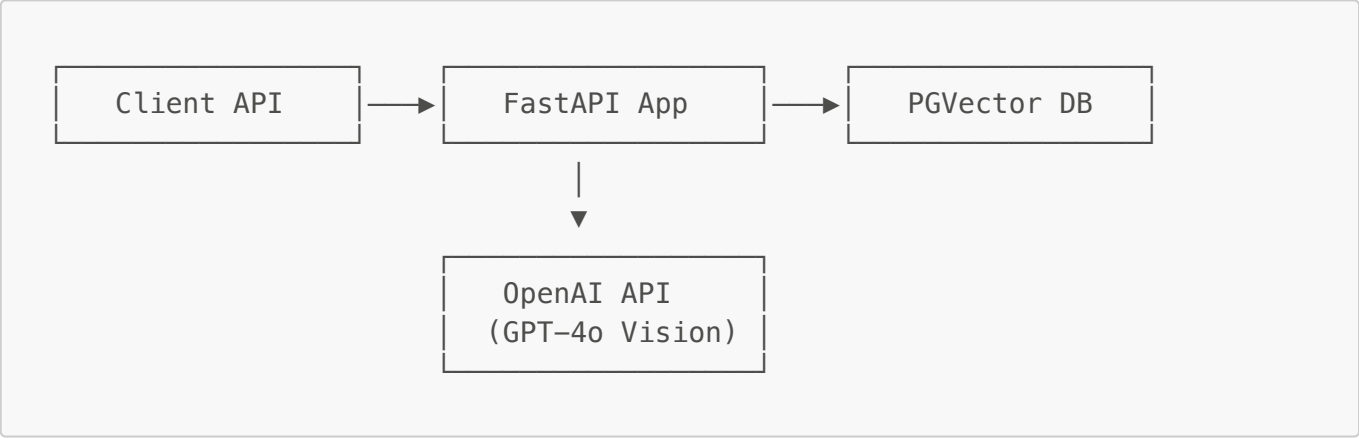
Application Analysis RAG Service는 FastAPI 기반의 RAG(Retrieval-Augmented Generation) 구축 서비스입니다. 애플리케이션 화면 캡처 이미지를 분석하여 UI 구성요소, 기능, 사용자 플로우를 체계화하고, 이를 벡터 데이터베이스에 저장하여 향후 소스코드 변경 시 테스트 케이스 생성을 위한 기반 데이터를 제공합니다.

🎯 주요 목적

- 애플리케이션 화면 이미지를 AI로 분석하여 구조화된 정보 추출
- 추출된 정보를 벡터 임베딩으로 변환하여 PGVector에 저장
- 소스코드 변경 시 영향 받을 화면을 빠르게 검색할 수 있는 RAG 시스템 구축
- 테스트 케이스 생성을 위한 기반 데이터 제공

🏗️ 아키텍처

시스템 구성



기술 스택

- **Backend**: FastAPI, Python
- **Vector Database**: PostgreSQL + PGVector Extension
- **AI/ML**: OpenAI GPT-4o (Vision), text-embedding-3-small
- **Framework**: LangChain
- **Database ORM**: SQLAlchemy
- **Dependency Injection**: Custom DI Container

🔄 시퀀스 다이어그램

🚀 핵심 기능

1. 이미지 분석 및 구조화

```
# 화면 캡처 이미지를 GPT-4o Vision으로 분석
- UI 컴포넌트 식별 (버튼, 폼, 테이블 등)
- 텍스트 요소 추출 (라벨, 메뉴, 버튼 텍스트)
- 기능적 특성 분석 (CRUD 작업, 사용자 액션)
- 화면 유형 분류 (입력폼, 목록, 상세보기 등)
```

2. 벡터 임베딩 저장

```
# OpenAI text-embedding-3-small 모델 사용
- 분석된 텍스트를 벡터로 변환
- PGVector에 메타데이터와 함께 저장
- 컬렉션별 관리 (서비스/버전별 분리 가능)
```

3. 유사도 검색

```
# 구현된 검색 기능
- similarity_search(): 기본 유사도 검색
- similarity_search_with_score(): 점수 포함 검색 (테스트용)
- 검색 키워드 최적화를 위한 다양한 메타데이터 활용
```

📁 프로젝트 구조

```
app/
├── api/                          # API 레이어
│   ├── rag_controller.py         # RAG 관련 엔드포인트
│   └── model/                    # Request/Response 모델
├── core/                         # 비즈니스 로직
│   ├── interface/                # 인터페이스 정의
│   └── service/                  # 서비스 구현
│       ├── rag_generation_service.py
│       └── data_extractor.py
├── infra/                        # 인프라 레이어
│   ├── database/                 # 데이터베이스 연결
│   ├── repository/               # 저장소 구현
│   └── external/                 # 외부 서비스 연동
│       ├── llm/                  # LLM 클라이언트
│       └── embedding/            # 임베딩 클라이언트
├── config/                       # 설정 파일
│   ├── database_config.py        # DB 설정
│   └── prompt.py                 # LLM 프롬프트
├── di_container.py               # 의존성 주입 컨테이너
└── main.py                       # FastAPI 애플리케이션 엔트리포인트
```

📡 API 엔드포인트

1. RAG 벡터 데이터 추가

```
POST /api/rag/add/vector
Content-Type: multipart/form-data

Parameters:
- collection_name: string (컬렉션 이름)
- service_name: string[] (서비스명 배열)
- screen_name: string[] (화면명 배열)
- version: string[] (버전 배열)
- access_level: string[] (접근 레벨 배열)
- images: file[] (이미지 파일 배열)
```

2. 벡터 생성 (개발용)

```
GET /api/rag/generation/vector?collection_name=test_collection
```

3. 헬스 체크

```
GET /api/rag/health
```

💡 사용 예시

1. 화면 이미지 분석 및 벡터 저장

```
# 멀티파트 폼으로 이미지와 메타데이터 전송
form_data = {
    'collection_name': 'mobile_app_v1',
    'service_name': ['쇼핑앱'],
    'screen_name': ['상품 목록 화면'],
    'version': ['1.2.0'],
    'access_level': ['user'],
    'images': [image_file]
}

response = requests.post('/api/rag/add/vector', files=form_data)
```

2. 저장된 데이터 구조 예시

```
{
    "input_metadata": {
        "service_name": "쇼핑앱",
        "screen_name": "상품 목록 화면",
```

```
        "version": "1.2.0",
        "access_level": "user"
    },
    "screen_analysis": {
        "visible_title": "상품 목록",
        "screen_type": "목록",
        "layout_description": "그리드 형태의 상품 목록과 상단 검색바",
        "primary_purpose": "사용자가 상품을 탐색하고 검색할 수 있는 화면"
    },
    "extracted_elements": {
        "all_visible_text": ["상품 목록", "검색", "필터", "정렬"],
        "button_texts": ["검색", "필터", "장바구니"],
        "field_labels": ["검색어 입력"],
        "menu_items": ["홈", "카테고리", "마이페이지"]
    }
}
```

🔍 검색 및 활용

유사도 검색 예시

```
# 서비스에서 유사한 화면 검색
similar_screens = rag_service.search_similar_screens(
    collection_name="mobile_app_v1",
    query="상품 목록 화면",
    k=5
)

# 결과: 유사한 UI 패턴을 가진 화면들의 정보
```

🎯 향후 확장 계획

1. **더 정교한 검색**: 메타데이터 필터링, 복합 조건 검색
2. **배치 처리**: 대량 이미지 일괄 처리 기능
3. **웹 화면 분석**: 모바일 앱 외 웹 애플리케이션 지원
4. **rag데이터 추가** : 텍스트 설명외에도 소스 코드 전체를 api단위로 나누고, 이를 화면과 매핑한 데이터를 rag로 구축