SK네트웍스 **Family AI** 과정 **15**기

모델배포 개발된 LLM 연동 웹 애플리케이션

산출물 단계	모델배포
평가 산출물	LLM 연동 웹 애플리케이션
제출 일자	2026.02.05
깃허브 경로	https://github.com/SKN19-Final-1team/backend
작성 팀원	안수이

1. 프로젝트 개요

- 프로젝트명: 카드사 상담원의 업무 효율성 향상을 위한 AI 상담 업무 지원 서비스
- 작성일: 2026.02.05
- 작성자: 안수이
- 버전: ver.1
- 승인관리자:

2. 시스템 개요

- 목표: 이 애플리케이션은 벡터 데이터베이스와 LLM을 연동하여 실시간으로 효율적인 검색 및 응답을 제공하여 상담 중 상담원이 빠르고 정확하게 상담 정보를 찾을 수 있도록 지원하는 것을 목표로 합니다.
- 기능:
    - STT 기반 실시간 음성 인식과 키워드 추출 연동
    - 추출한 키워드를 이용한 문서 자동 검색
    - 벡터 데이터베이스와 LLM 연동
    - 연동한 LLM 기반 맞춤형 응대 가이드 생성
    - 효율적인 프롬프트 최적화
    - 안전한 API 키 관리 및 예외 처리
    - 실시간성 보장을 위한 속도 개선

3. 시스템 아키텍처

- 백엔드:
    - 언어: Python
    - 프레임워크: FastAPI
    - LLM 연동: OpenAI GPT-4.1-mini
    - 벡터 데이터베이스: PostgreSQL PGVector
    - 데이터베이스: PostgreSQL

- 프론트엔드:
    - 언어: TypeScript
    - 프레임워크: React
    - 상호작용: Fetch 요청을 통한 비동기 통신
    - UI: TailwindCSS

4. LLM 연동 및 벡터 데이터베이스 구현

4.1 벡터 데이터베이스와 LLM 연동

- 연동 목적 : LLM이 벡터 데이터베이스에 저장된 정보를 바탕으로 상담 중 상담원의 업무를 지원하는 칸반보드에 표시할 적절한 응답 및 추천 멘트를 생성하도록 연동합니다.

- 프롬프트:

```python
def _build_messages(query: str, docs: List[Dict[str, Any]], consult_docs: List[Dict[str, Any]]) -> List[Dict[str,

    system_prompt = (
        "당신은 카드사 콜센터 상담원을 돕는 내부 안내 스크립트를 작성하는 AI입니다. "
        "고객에게 바로 읽어줄 수 있는 '완성된 안내 문장'만 작성하세요.\n\n"

        "[작성 원칙]\n"
        "1. 반드시 제공된 Documents와 Consultation cases에 포함된 정보만 사용하세요.\n"
        "2. 문서에 없는 내용, 추측, 일반 상식, 약관 문장 그대로 인용은 절대 금지합니다.\n"
        "3. 법조문·약관 문장은 그대로 옮기지 말고, 상담원이 말하듯 쉽게 풀어서 설명하세요.\n"
        "4. 전화번호, URL, 이메일, 개인정보는 절대 포함하지 마세요.\n\n"

        "[출력 형식]\n"
        "- 전체는 최대 3문장\n"
        "- 문단, 번호, 불릿, 따옴표 사용 금지.\n\n"

        "[문장별 역할]\n"
        "첫 번째 문장: 고객 상황을 한 줄로 정리하며 공감 표현을 합니다.\n"
        "두 번째 문장: 지금 바로 안내해야 할 핵심 처리 방법 또는 절차를 명확하게 설명합니다.\n"
        "세 번째 문장: 안내를 마친 뒤 확인해야 할 핵심 한 가지를 질문합니다.\n\n"

        "[중요 제한 사항]\n"
        "- 이미 문서에 답이 충분한 경우, 불필요한 추가 질문을 하지 마세요.\n"
        "- '어떤 단계에서 막히셨는지', '확인 후 안내드리겠습니다' 같은 모호한 문장은 사용하지 마세요.\n"
        "- '손님:', '고객:', '상담사:' 같은 화자 표기는 절대 쓰지 마세요.\n"
        "- [날짜#], [금액#], [비율#], [카드사명#] 같은 대괄호 플레이스홀더는 절대 쓰지 마세요.\n"
        "- 문서 제목, 파일명, 조항 번호, 조문 표기는 고객에게 절대 말하지 마세요.\n"
        "- '잠시만 기다려 주세요', '확인 후 안내드리겠습니다', '기다려주셔서 감사합니다' 같은 관용구는 절대 쓰지 마세요.\n"
        "- 예방 수칙, 일반 주의사항, 배경 설명은 포함하지 마세요.\n"
        "- 답을 모를 경우에만 한 문장으로 정보 추가 요청을 하세요.\n\n"
        "[근거 사용]\n"
        "- 반드시 Documents 내용에 근거한 문장만 작성하세요.\n"
        "- Documents에 없는 절차/정책/요금/기간/조건은 절대 만들지 마세요.\n\n"
        "[필수 디테일]\n"
        "- Documents에 포함된 구체적 디테일을 최소 1개는 반드시 포함하세요.\n\n"

        "항상 상담원이 고객에게 바로 읽어주는 상황을 가정하고, 간결하고 단정하게 작성하세요."
    )
```

- 프롬프트 최적화:
    - 작성 원칙, 출력 형식, 문장별 역할, 중요 제한 사항로 지시를 구분화하였습니다.
    - 내부 안내 스크립트를 제공하는 AI라는 명확한 역할 부여하였습니다.
    - LLM이 요구 사항에 맞는 최적의 결과를 도출할 수 있도록 주기적으로 테스트하고 최적화하였습니다.

예시 코드: LLM과 벡터 데이터베이스 연동

```
40
41
42  def generate_guide_text(
43      messages: List[Dict[str, str]],
44      temperature: float = 0.2,
45      max_tokens: int = 320,
46      top_p: float = 0.9,
47      timeout_sec: int = 30,
48  ) -> str:
49      client = _get_openai_client()
50      if not client:
51          return ""
52      try:
53          resp = client.chat.completions.create(
54              model=GUIDE_MODEL_NAME,
55              messages=messages,
56              temperature=temperature,
57              max_tokens=max_tokens,
58              top_p=top_p,
59              stop=["손님:", "상담사:", "고객:"],
60              timeout=timeout_sec,
61          )
62          return (resp.choices[0].message.content or "").strip()
63      except Exception as exc:
64          return ""
65
```

5. 예외 처리 및 보안

5.1 예외 처리

- 예상치 못한 상황에 대한 예외 처리:
    - API 요청이 실패하거나, 벡터 데이터베이스가 응답하지 않을 때 빈 문자열을 return 합니다.

5.2 보안 관리

- 환경 변수를 사용하여 코드에 민감한 정보를 노출시키지 않도록 관리합니다.

```
_load_env()

GUIDE_MODEL_NAME = "gpt-4.1-mini"
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")

_openai_client = None
```

6. 코드 모듈화 및 주석

6.1 모듈화

코드는 각 기능을 별도의 모듈로 분리하여 유지보수를 용이하게 합니다.

guide_client.py의 generate_guide_text로 모듈화하여 필요한 부분만 호출하여 사용합니다.

```python
from app.guide.guide_client import generate_guide_text
```

```python
def generate_guide_message(
    query: str,
    docs: List[Dict[str, Any]],
    consult_docs: List[Dict[str, Any]],
) -> str:
    if not docs:
        return ""
    intent = detect_intent(query)
    messages = _build_messages(query, docs, consult_docs)
    output = generate_guide_text(messages)
    normalized = normalize_output(output, intent)
    normalized = apply_question_policy(normalized, query)
```