

**SK네트웍스 Family AI 과정 14기  
 모델 배포 개발된 LLM 연동 웹 애플리케이션**



| **산출물 단계** | 모델배포 |
| --- | --- |
| **평가 산출물** | LLM 연동 웹 애플리케이션 |
| **제출 일자** | 2025.09.25 |
| **깃허브 경로** | https://github.com/skn-ai14-250409/SKN14-Final-6Team-AI |
| **작성 팀원** | 김광령 |

### ***1. 프로젝트 개요***

* 프로젝트명 : Qook - AI 신선식품 챗봇 플랫폼
* 작성일 : 2025-09-24
* 작성자 : 김광령
* 버전 : v1.0

### ***2. 시스템 개요***

* 목표: 데이터베이스와 LLM을 연동하여 자연어 기반 신선식품 쇼핑과 레시피 검색을 제공하는 통합 AI 챗봇 플랫폼
* 기능:
  + **자연어 상품/레시피 검색**: 사용자가 자연어로 원하는 상품이나 레시피를 찾고 주문
  + **비전 AI 레시피 검색**: 음식 사진 업로드를 통한 레시피 추천 및 재료 매칭
  + **통합 고객서비스**: FAQ 자동 응답, 주문 조회, 환불 처리 서비스 제공
  + **개인화 추천**: 멤버십 등급별 맞춤 서비스 및 알레르기, 비건, 비선호 음식 필터링
  + **마이페이지**: 주문/채팅 관련 내역을 확인하거나 개인 정보를 수정할 수 있음

### ***3. 시스템 아키텍처***

* **백엔드**
  + 언어: Python
  + 프레임워크: FastAPI/ Django
  + LLM 연동: OpenAI GPT-4, GPT-4 Vision
  + 벡터 데이터베이스: Pinecone
  + 데이터베이스: MySQL
  + 기타 기술 스택:
    - 인증: JWT
    - 데이터 검증: Pydantic
* **프론트엔드**
  + 웹디자인: HTML, CSS
  + 클라이언트 로직: JavaScript
* **외부 API**
  + 벡터 검색: Pinecone API
  + 웹 검색: Tavily API
  + 주소 검색: Kakao Address API

### ***4. Llm 연동 및 벡터 데이터베이스 구현***

#### 4.1 벡터 데이터베이스와 llm 연동

* 연동 목적: 사용자 쿼리에 대해 벡터 데이터베이스에서 관련 문서를 검색하고, LLM이 이를 바탕으로 정확한 응답을 생성시키기 위함함
* 데이터 소스: 마켓컬리 FAQ, 이용약관 데이터를 수집하여 Qook 전용 맞춤 데이터셋 구축

**예시 코드**:

| **def** **faq\_policy\_rag**(state) -> Dict[str, Any]:  """FAQ RAG 검색 - 실제 동작하는 코드"""  **if** **not** (openai\_client **and** pinecone\_index):  **return** \_faq\_db\_fallback(state)    **try**:  *# 🔥 1단계: 사용자 질문을 벡터로 변환*  embedding\_response = openai\_client.embeddings.create(  model="text-embedding-3-small",   input=state.query  )  query\_embedding = embedding\_response.data[0].embedding    *# 🔥 2단계: Pinecone에서 유사한 문서 검색*   results = pinecone\_index.query(  vector=query\_embedding,   top\_k=5,   include\_metadata=**True**,  filter={"type": {"$in": ["faq", "terms"]}}  )    *# 🔥 3단계: 검색된 문서들을 컨텍스트로 활용*  docs, citations = [], []  threshold = 0.4 *# 유사도 임계값*    **for** match **in** results.matches:  **if** match.score >= threshold: *# 유사도가 높은 것만 선별*  content = match.metadata.get("text", "")  docs.append(content)    *# 🔥 4단계: LLM에게 컨텍스트와 함께 답변 요청*  context\_text = "\n\n".join(docs)  rag\_prompt = f"""  질문: {state.query}  참고 문서: {context\_text}    위 문서를 기반으로 답변을 작성하세요.  """   **except** Exception **as** e:  **return** {"error": str(e)} |
| --- |

* 프롬프트 최적화
  + 쿼리 개선: 사용자 입력을 의도에 맞게 표준화하여 검색 정확도 향상
  + 컨텍스트 압축: 벡터 검색 결과를 요약하여 토큰 사용량 최적화

### ***5. 예외 처리 및 보안***

#### 5.1 예외 처리

* 예상치 못한 상황에 대한 예외 처리:
  + API 연결 실패: OpenAI, Pinecone 등 외부 서비스 장애 대응
  + 데이터베이스 연결 오류: MySQL 연결 실패 시 재시도 로직

**예외 처리 코드 예시**:

| **try**:  **import** openai  openai\_api\_key = os.getenv("OPENAI\_API\_KEY")  openai\_client = openai.OpenAI(api\_key=openai\_api\_key) **if** openai\_api\_key **else** **None**  **if** **not** openai\_api\_key:  logger.warning("OpenAI API key not found. Using mock responses.") **except** Exception:  openai\_client = **None**  logger.warning("OpenAI package not available.")  **try**:  **from** pinecone **import** Pinecone  pinecone\_api\_key = os.getenv("PINECONE\_API\_KEY")  **if** pinecone\_api\_key:  pinecone\_client = Pinecone(api\_key=pinecone\_api\_key)  pinecone\_index = pinecone\_client.Index("qook")  **else**:  pinecone\_client = **None**  pinecone\_index = **None**  logger.warning("Pinecone API key not found. Using mock responses.") **except** Exception:  pinecone\_client = **None**  pinecone\_index = **None**  logger.warning("Pinecone package not available.") |
| --- |

**5.2 보안**

* **보안 코드**
  + 환경 변수 관리: 모든 민감한 정보를 환경 변수로 분리
  + JWT 토큰 기반 인증: 세션 관리 및 사용자 권한 검증
  + HTTPS 통신: 모든 API 통신에서 암호화 적용
  + 입력 검증: SQL Injection, XSS 공격 방지

**보안 코드 예시**:

| **def** **\_runtime\_secret**() -> str:  """런타임마다 다른 비밀키 생성으로 보안 강화"""  **return** f"{SECRET\_KEY}:{\_RUNTIME\_SALT}"  ACCESS\_TOKEN\_EXPIRE\_HOURS = 24  **def** **create\_access\_token**(data: dict, expires\_delta: Optional[timedelta] = None):  """JWT 액세스 토큰 생성 - 만료시간 포함"""  to\_encode = data.copy()  **if** expires\_delta:  expire = datetime.utcnow() + expires\_delta  **else**:  expire = datetime.utcnow() + timedelta(hours=ACCESS\_TOKEN\_EXPIRE\_HOURS)    *# 🔥 만료시간을 토큰에 포함시켜 보안 강화*  to\_encode.update({"exp": expire})  encoded\_jwt = jwt.encode(to\_encode, \_runtime\_secret(), algorithm=ALGORITHM)  **return** encoded\_jwt  **def** **verify\_token**(credentials: HTTPAuthorizationCredentials = Depends(security)):  """JWT 토큰 검증 - 만료/변조 검사"""  **try**:  *# 🔥 런타임 시크릿으로 토큰 검증*  payload = jwt.decode(credentials.credentials, \_runtime\_secret(), algorithms=[ALGORITHM])  user\_id: str = payload.get("sub")  **if** user\_id **is** **None**:  **raise** HTTPException(status\_code=401, detail="Invalid authentication credentials")    **except** jwt.PyJWTError: *# 🔥 만료되거나 잘못된 토큰 처리*  **raise** HTTPException(status\_code=401, detail="Invalid authentication credentials")    **return** user\_id |
| --- |

**환경 변수 예시** (.env 파일)

| """ OPENAI\_API\_KEY=your-openai-key PINECONE\_API\_KEY=your-pinecone-key  SECRET\_KEY=your-jwt-secret-key """ |
| --- |

### 

### 

### ***6. 코드 모듈화 및 주석***

#### 6.1 모듈화

* 프로젝트는 기능별로 명확하게 폴더를 분리
* [app.py](http://app.py) -> [workflow.py](http://workflow.py) -> nodes/ -> utils/ 순으로 의존성 계층 구조 형성
* 각 파일이 단일 책임을 갖고 특정 파일에서 필요한 함수를 가져와 실행

#### 6.2 주석

* **주석 예시**

| **def** **handoff**(state: ChatState) -> Dict[str, Any]:  """  상담사 이관  - 저신뢰/예외 케이스에서 인간 상담사/CRM으로 이관합니다.  - 대화 요약/근거/사용자 메타를 함께 전달합니다.  """  **class** **ConversationSummarizer**:  """대화 요약 및 개인정보 필터링"""    **@staticmethod**  **def** **filter\_sensitive\_info**(text: str) -> str:  """개인정보 마스킹"""  **import** re    *# 🔥 실제 개인정보 보호 로직들*  text = re.sub(r'010-\d{4}-(\d{4})', r'010-\*\*\*\*-\1', text) *# 전화번호*  text = re.sub(r'(\w)[\w\.-]\*@', r'\1\*\*\*@', text) *# 이메일*  text = re.sub(r'\b(\d{4})-(\d{4})-(\d{4})-(\d{4})\b', r'\*\*\*\*-\*\*\*\*-\*\*\*\*-\4', text) *# 카드번호*    **return** text |
| --- |

### 

### ***7. 주요 기능별 구현 세부 사항***

#### 7.1 자연어 상품 검색

* **의도 분석**: 사용자 입력을 상품명, 카테고리, 수량, 가격대 등으로 파싱
* **벡터 검색**: 상품 설명을 임베딩하여 의미적 유사도 기반 검색
* **필터링**: 유기농, 원산지, 가격대 등 다중 조건 적용

| **def** **\_llm\_enhance\_all**(query: str) -> Optional[Dict[str, Any]]:  """🔥 전체 쿼리 보강 - 사용자 입력을 구조화된 데이터로 변환"""  system\_prompt = """신선식품 쇼핑몰의 전문 쿼리 분석가입니다. 사용자 입력을 분석하여 상품명, 카테고리, 수량, 가격대 등으로 파싱합니다.  # 출력 JSON 구조: - rewrite: {"text": "표준화된 쿼리", "keywords": [...], "confidence": 0.9} - slots: {  "product": "사과",  "category": "과일",   "quantity": 2,  "price\_cap": 10000,  "organic": true,  "origin": "국내산" } """   response = openai\_client.chat.completions.create(  model=Config.OPENAI\_MODEL,  messages=[  {"role": "system", "content": system\_prompt},  {"role": "user", "content": query}  ],  response\_format={"type": "json\_object"}  )    **return** json.loads(response.choices[0].message.content) |
| --- |

#### 7.2 비전 ai 레시피 검색

* **이미지 분석**: GPT-4 Vision으로 음식 사진에서 재료 및 요리명 추출
* **레시피 매칭**: 분석 결과를 바탕으로 관련 레시피 검색
* **재료 연결**: 레시피 재료와 쇼핑몰 상품을 자동으로 매핑

| **def** **\_analyze\_food\_image**(base64\_image\_data: str) -> Optional[Dict[str, Any]]:  """🔥 GPT-4 Vision으로 음식 사진에서 재료 및 요리명 추출"""    system\_prompt = """음식 이미지 분석 전문가입니다. 이미지를 분석하여 다음 정보를 JSON으로 제공하세요:  1. is\_food: 요리된 음식인지 판단 (true/false) 2. food\_name: 음식의 정확한 한국어 이름 ("김치찌개", "불고기") 3. confidence: 분석 신뢰도 (0.0-1.0) 4. description: 음식에 대한 간단한 설명 5. ingredients: 보이는 주요 재료들 (배열)  규칙: - 요리된 음식(완성된 요리)만 is\_food: true - 생재료만 있는 경우는 is\_food: false """   response = openai\_client.chat.completions.create(  model="gpt-4o", *# GPT-4 Vision 모델*  messages=[  {"role": "system", "content": system\_prompt},  {  "role": "user",  "content": [  {"type": "text", "text": "이 이미지에서 음식을 분석해주세요."},  {  "type": "image\_url",  "image\_url": {  "url": f"data:image/jpeg;base64,{base64\_image\_data}"  }  }  ]  }  ],  response\_format={"type": "json\_object"}  )    **return** json.loads(response.choices[0].message.content) |
| --- |

#### 7.3 통합 고객서비스

* **FAQ 자동 응답**: 벡터 검색을 통한 관련 답변 제공
* **상담사 연결**: 복잡한 문의 시 휴먼 핸드오프
* **주문 조회**: 실시간 배송 상태 추적

| **def** **faq\_policy\_rag**(state) -> Dict[str, Any]:  """🔥 벡터 검색을 통한 관련 답변 제공"""  **try**:  *# 1. 사용자 질문을 벡터로 변환*  embedding\_response = openai\_client.embeddings.create(  model="text-embedding-3-small",   input=state.query  )  query\_embedding = embedding\_response.data[0].embedding    *# 2. Pinecone에서 유사한 FAQ/정책 문서 검색*  results = pinecone\_index.query(  vector=query\_embedding,   top\_k=5,   include\_metadata=**True**,  filter={"type": {"$in": ["faq", "terms"]}} *# FAQ와 이용약관만*  )    *# 3. 유사도 임계값으로 필터링*  docs, citations = [], []  threshold = 0.4    **for** match **in** results.matches:  **if** match.score >= threshold:  content = match.metadata.get("text", "")  docs.append(content)  citations.append(f"{match.metadata.get('type')}:{match.metadata.get('category')}")   *# 4. LLM에게 컨텍스트와 함께 답변 요청*  context\_text = "\n\n".join(docs)  rag\_prompt = f"""  질문: {state.query}  참고 문서: {context\_text}    위 문서를 기반으로 친절하고 정확한 답변을 200자 이내로 작성하세요.  """    response = openai\_client.chat.completions.create(  model="gpt-4",  messages=[{"role": "user", "content": rag\_prompt}]  )    **return** {  "answer": response.choices[0].message.content,  "confidence": len(docs) \* 0.2, *# 찾은 문서 수에 따라 신뢰도 조정*  "sources": citations  }    **except** Exception **as** e:  **return** \_faq\_db\_fallback(state) *# DB 폴백* |
| --- |

### 

### ***8. 결론 및 향후 개선사항***

#### 8.1 프로젝트 성과

* **통합 경험**: 상품 검색부터 주문까지 챗봇 하나로 완성되는 사용자 경험
* **AI 활용**: 최신 LLM과 벡터 검색 기술을 활용한 정확한 응답
* **확장 가능한 아키텍처**: 모듈화된 구조로 기능 추가 및 확장 용이

#### 8.2 향후 개선 계획

* **다국어 지원**: 영어, 중국어 등 다양한 언어 지원
* **개인화 강화**: 사용자 구매 패턴 기반 추천 시스템 고도화