# 1. 프로젝트 개요

- **프로젝트명**: CLICKA (Click + Assistant)
- **작성일**: 2025.09.04
- **작성자**: 남궁건우
- **버전**: v1.0

# 2. 시스템 개요

- **목표**: 단순 검색을 넘어 문서, 회의, 일정까지 통합 관리하는 올인원 AI 비서 구축. RAG + LLM 기반의 조직 내 정보 통합·자동화 시스템 제공.
- **기능**:
  - **문서 검색 및 분석**: 내부 문서 벡터 검색 및 RAG 기반 질의응답
  - **문서 편집**: AI 에이전트를 통한 문서 내용 수정 및 편집
  - **일정 관리**: 캘린더 연동 및 이벤트 자동 생성
  - **멀티 에이전트 시스템**: 질문 유형별 전문 에이전트 라우팅
  - **실시간 채팅**: Electron 기반 데스크톱 애플리케이션

# 3. 시스템 아키텍처

- **백엔드 (FastAPI + LangGraph)**:
  - **언어**: Python 3.11+
  - **웹 프레임워크**: FastAPI (비동기 처리, 자동 API 문서화)
  - **AI 프레임워크**: LangGraph (멀티 에이전트 시스템)
  - **LLM**: OpenAI GPT-4o (스트리밍 응답 지원)
  - **벡터 데이터베이스**: ChromaDB (문서 임베딩 저장)
  - **관계형 데이터베이스**: MySQL (사용자, 세션, 문서 메타데이터)
  - **문서 처리**: Docling (PDF/HWP → Markdown 변환)
- **프론트엔드 (Electron + React)**:
  - **데스크톱 프레임워크**: Electron (크로스 플랫폼)
  - **UI 프레임워크**: React 18 + Vite
  - **스타일링**: TailwindCSS
  - **상태 관리**: React Hooks + Context API
  - **실시간 통신**: Server-Sent Events (SSE)
- **주요 화면 구성**:
  - **로그인/인증**: JWT 기반 사용자 인증
  - **채팅 인터페이스**: 실시간 AI 대화 및 문서 검색
  - **문서 편집기**: Rich Text Editor (TipTap 기반)
  - **캘린더**: 일정 관리 및 이벤트 생성
  - **관리자 패널**: 사용자 및 시스템 관리

# 4. LLM 연동 및 벡터 데이터베이스 구현

- **연동 목적**: 사용자의 질의에 대해 내부 문서·회의록·일정 데이터를 벡터 검색으로 불러오고, LLM이 이를 요약·응답.
- **구현 방식**:
  - **문서 처리 파이프라인**:
    - PDF/HWP → Docling을 통한 Markdown 변환
    - 텍스트 청킹 및 임베딩 생성 (OpenAI Embeddings)
    - ChromaDB에 벡터 저장 및 메타데이터 관리
  - **멀티 에이전트 시스템**:
    - **RoutingAgent**: 사용자 질문을 분석하여 적절한 전문 에이전트로 라우팅
    - **DocumentSearchAgent**: 내부 문서 검색 및 RAG 기반 질의응답
    - **DocumentEditorAgent**: 문서 편집 및 수정 작업 처리
    - **GeneralChatAgent**: 일반 대화 및 문서 내용 질문 처리
  - **실시간 스트리밍**: FastAPI StreamingResponse를 통한 실시간 응답 전송
- **실제 구현된 프롬프트 시스템**:
  - **라우팅 에이전트 프롬프트**:

```
system_prompt = f"""당신은 사용자의 질문을 가장 적절한 전문가에게 전달하는 라우팅 전문가입니다.
정확한 라우팅 결정을 위해 전체 대화 기록을 반드시 고려해야 합니다.

세 명의 전문가가 있습니다:
1. DocumentSearchAgent: 재무 보고서, 감사 결과, 내부 규정 등 내부 문서 검색이 필요한 질문
2. DocumentEditorAgent: 사용자가 현재 작업중인 문서의 내용을 수정, 변경, 추가, 삭제하는 명령
3. GeneralChatAgent: 일반적인 대화, 인사, 혹은 위 두 전문가의 역할을 제외한 모든 질문

오직 'DocumentSearchAgent', 'DocumentEditorAgent', 'GeneralChatAgent' 중 하나로만 대답하십시오.
"""
```

  - **문서 검색 에이전트 프롬프트**:

```
def get_document_search_system_prompt():
    return """당신은 한국방송광고진흥공사의 내부 문서 검색 전문가입니다.
    사용자의 질문에 대해 관련 문서를 검색하고, 정확하고 유용한 정보를 제공해야 합니다.
    검색된 문서의 내용을 바탕으로 답변하며, 가능한 경우 원문 링크나 출처를 제공하세요."""
```

  - **문서 편집 에이전트 프롬프트**:

```
def get_document_editor_system_prompt():
    return """당신은 문서 편집 전문가입니다.
    사용자가 제공한 문서 내용을 바탕으로 요청된 편집 작업을 수행합니다.
    문서의 구조와 형식을 유지하면서 정확하게 편집하세요."""
```

# 5. 예외 처리 및 보안

- **예외 처리**:
  - 파일 업로드시, 파일목록에서 전송실패시 안내 메시지 반환
  - 챗봇에서 기존 채팅 세션 삭제 실패시 안내메시지 반환
- **보안 관리**:

- API 키 및 DB 접속 정보는 환경 변수로 관리
- JWT 인증, HTTPS 통신 적용
- 사내 내부문서 접근 제한 고려 → 한국광고진흥공사 공공데이터 활용

# 6. 코드 모듈화 및 주석

- **실제 모듈화 구조**:
  - **백엔드 (FastAPI + LangGraph)**:
    - `main.py` → FastAPI 애플리케이션 진입점
    - `ChatBot/agents/` → AI 에이전트 모듈
      - `RoutingAgent.py` → 질문 분류 및 라우팅
      - `DocumentSearchAgent.py` → 문서 검색 전용 에이전트
      - `DocumentEditorAgent.py` → 문서 편집 전용 에이전트
      - `chat_agent.py` → 일반 대화 에이전트
    - `ChatBot/tools/` → 에이전트 도구 모듈
      - `retriever_tool.py` → 벡터 검색 도구
      - `editor_tool.py` → 문서 편집 도구
      - `agent_logic.py` → 에이전트 로직 도구
    - `routers/` → API 라우터 모듈
      - `users_routes.py` → 사원관리용 API
      - `chat_routes.py` → 채팅 API 엔드포인트
      - `document_routes.py` → 문서 관리 API
      - `auth_routes.py` → 인증 API
      - `calendar_routes.py` → 캘린더 API
    - `database/` → SQLAlchemy 데이터베이스 모델
  - **프론트엔드 (React + Electron)**:
    - `src/components/` → React 컴포넌트
      - `ChatWindow/` → 채팅 UI 컴포넌트
      - `FeatureWindow/` → 기능별 윈도우 컴포넌트
      - `services/` → API 통신 서비스
    - `main.js` → Electron 메인 프로세스
    - `preload.js` → Electron 프리로드 스크립트
- **실제 주석 예시**:

```python
# 라우팅 에이전트 - 질문 분류 함수
def route_question(state: AgentState) -> Literal["document_search", "general_chat", "document_edit"]:
    """
    사용자의 질문을 분석하여 가장 적절한 전문가 에이전트로 라우팅한다.

    Args:
        state: 현재 에이전트 상태 (메시지 히스토리, 문서 내용 등)

    Returns:
        Literal: 라우팅할 에이전트 타입
            - "document_search": 내부 문서 검색이 필요한 경우
            - "document_edit": 문서 편집/수정이 필요한 경우
            - "general_chat": 일반 대화 또는 문서 내용 질문
    """
```

```javascript
// 채팅 API 서비스 - 메시지 저장 함수
async function saveMessage({ sessionId, role, content }) {
  /**
   * 채팅 메시지를 백엔드에 저장한다.
   *
   * @param {Object} params - 메시지 저장 파라미터
   * @param {string} params.sessionId - 채팅 세션 ID
   * @param {string} params.role - 메시지 역할 (user/assistant)
   * @param {string} params.content - 메시지 내용
   * @returns {Promise<Object>} 저장 결과
   */
}
```