생성형 AI 기반 RAG 개발자 과정

Flask 기반 RAG 서비스 구조 설계

Session 17 / 33 — Day 3

스크립트에서 웹서비스로

박수현 · (사)한국정보공학기술사회 · 2026

LLM 이론 · LangChain · ChromaDB CRUD · Agent — 4일 핸즈온

이번 시간을 마치면

🏗️ 전체 구조

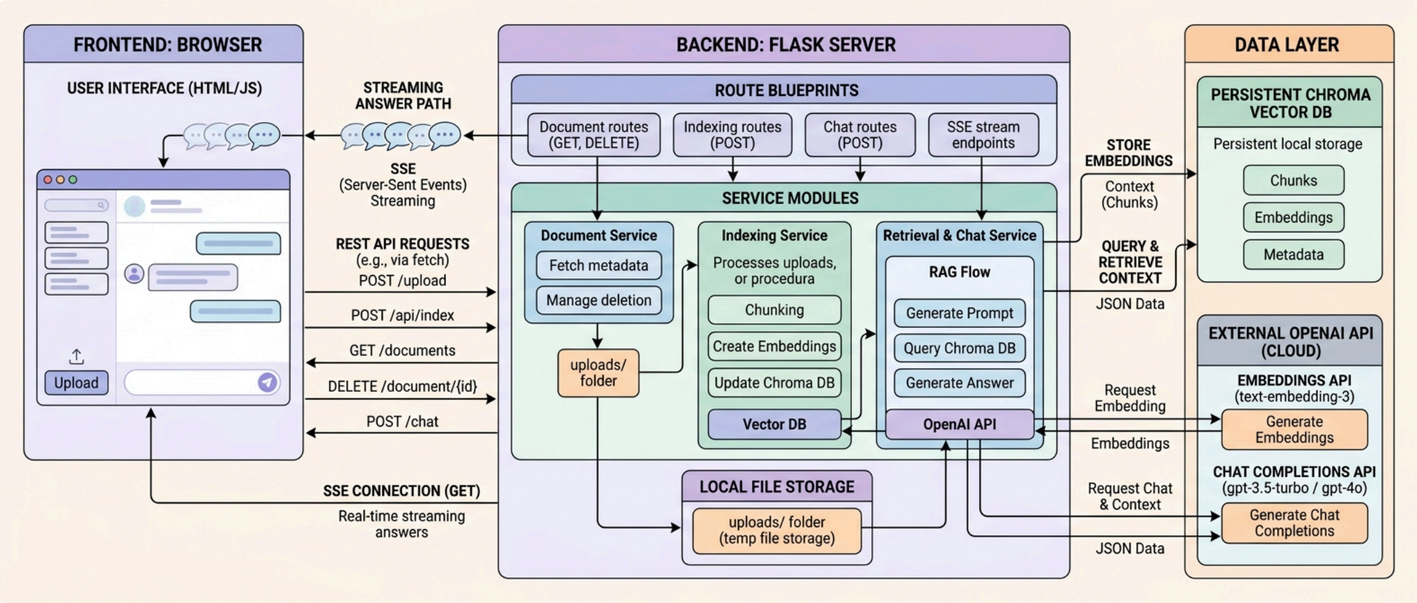
- **Backend / Frontend 분리**
- **REST API** 리소스 설계
- **서비스 계층** (라우트와 로직 분리)

💾 영속과 폴더

- **ChromaDB 영속** 저장
- **uploads / chroma_db** 디렉토리
- 프로젝트 **폴더 구조** 표준

> 🎯 목표: 코드 작성에 앞서 **각 구성 요소의 위치와 역할**을 정리한다.

브라우저 ⇄ Flask ⇄ 서비스 ⇄ ChromaDB



RAG 웹서비스 3계층 · 브라우저(fetch/SSE) ↔ Flask(/upload·/files·/ask) ↔ Chroma(영속)+OpenAI API

💡 핵심 원칙: **라우트(HTTP)는 얇게, 로직은 서비스 계층에**. 화면과는 JSON 으로만 통신한다.

문서(files)와 질문(ask) 두 리소스

화면(GET /) · 문서 리소스(/upload·/files) · 질문 리소스(/ask) 세 그룹으로 설계

메서드가 동작을 결정한다

메서드 · 경로	역할	CRUD
`GET /`	화면(HTML) 서빙	—
`POST /upload`	문서 업로드 + 색인	**C**
`GET /files`	문서 목록	**R**
`DELETE /files/<source>`	문서 삭제	**D**
`POST /ask`	질의응답	—

🔑 같은 `/files` 경로라도 **메서드가 동작을 결정**한다(GET 조회·DELETE 삭제). 응답을 JSON 으로 통일하면 curl·모바일 앱 등 **모든 클라이언트**에서 호출 가능하다.

역할별 폴더 분리

```
project/
├ app.py            ← create_app() 앱 팩토리
├ routes/           ← Flask Blueprint
│   ├ page_bp.py    ← GET / (화면 서빙)
│   ├ files_bp.py   ← 업로드·목록·삭제
│   └ qa_bp.py      ← 질의응답
├ services/
│   ├ vectorstore.py ← 색인·목록·삭제
│   └ qa_service.py  ← 검색·답변
├ uploads/          ← 원본 파일
├ chroma_db/        ← 벡터 영속
└ templates/        ← 프론트(index.html)
```

분리의 근거

- **app.py** = `create_app()` 으로 Blueprint 조립 전담
- **routes** = HTTP 입출력 전담 (Blueprint)
- **services** = 핵심 로직 (재사용·테스트 용이)
- **uploads / chroma_db** = 데이터 영속
- 단일 파일에 집중하면 **유지보수 비용이 급증**

> 📁 참고: `8.web_app/4.refactor_with_score/` (routes·services 분리 버전)

색인 1회, 영구 재사용

영속이 필요한 이유

- 서버 재시작마다 재색인하면 **시간·비용 낭비**
- `persist_directory` 로 디스크에 저장
- 다음 기동 시 **로드 후 즉시 검색**

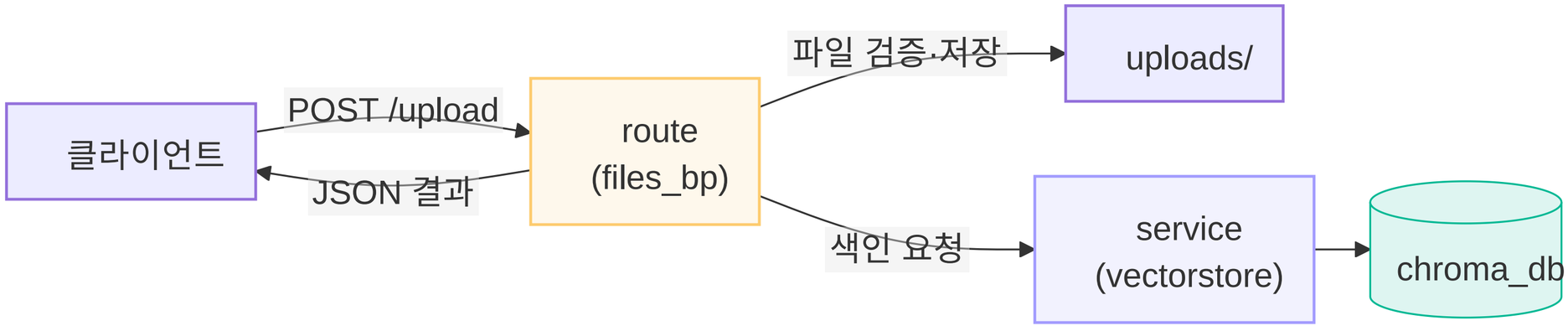
```
Chroma(
  persist_directory="./chroma_db",
  embedding_function=emb,
)
```

두 저장소의 역할

폴더	저장 내용
`uploads/`	**원본 파일** (재다운로드·재색인용)
`chroma_db/`	**벡터 + 메타데이터**

> 🔑 원본과 벡터를 **함께** 보관해야 삭제·재색인이 가능하다.

업로드 요청의 처리 순서



💡 route 는 수신·위임·응답만 담당한다. 무거운 작업(파싱·임베딩)은 **서비스 계층**이 처리한다.

핵심 5가지

1. ✅ 구조 = **브라우저 ⇄ Flask(routes) ⇄ Service ⇄ Chroma + OpenAI API**
2. ✅ REST 리소스 — `/upload` · `/files` **(CRUD)** · `/ask` **(질의)** · `/` **(화면)**
3. ✅ **routes 얇게 / services 로 로직 분리** → 재사용·테스트 용이
4. ✅ **uploads/(원본) + chroma_db/(벡터)** 둘 다 영속
5. ✅ 폴더 구조를 먼저 확정하면 이후 기능 추가가 수월