MEINRAG Technical Architecture Document

MEINRAG is a Retrieval-Augmented Generation (RAG) system designed for document question-answering. The system uses a FastAPI backend with LangChain to orchestrate the RAG pipeline. Users upload documents in various formats (PDF, DOCX, TXT, Markdown, HTML, Excel, PowerPoint) which are then processed, chunked, and stored as vector embeddings.

The core architecture consists of three layers:
1. Document Ingestion Layer - handles file upload, format detection, text extraction, and chunking using RecursiveCharacterTextSplitter.
2. Vector Storage Layer - stores embeddings with a switchable backend supporting both ChromaDB (default) and FAISS.
3. Query Layer - retrieves relevant chunks via similarity search, constructs a context-augmented prompt, and generates answers using an LLM.

The system supports two LLM providers: OpenAI (direct) and OpenRouter (proxy). Embeddings are always generated using OpenAI's text-embedding-3-small model to ensure consistency across the vector store, regardless of which chat model is used.

Vector Store Comparison

ChromaDB is the default vector store. It provides automatic persistence to disk, built-in metadata filtering, and requires no additional infrastructure. Data is stored in a local SQLite database with parquet files. ChromaDB is ideal for datasets up to 500,000 documents and provides sub-100ms query times.

FAISS (Facebook AI Similarity Search) is the alternative backend optimized for raw search speed. It uses optimized index structures and can leverage GPU acceleration. FAISS excels with datasets exceeding 500,000 documents but requires manual persistence (save_local/load_local calls) and lacks native metadata filtering.

The system uses an abstract VectorStoreManager interface that both implementations conform to. Switching between stores requires only changing the VECTOR_STORE environment variable from 'chroma' to 'faiss' (or vice versa). The factory pattern in vectorstore/factory.py handles instantiation.

API Endpoints

The REST API provides five endpoints:

POST /documents/upload - Accepts a file upload via multipart form data. The file is saved to disk, processed through the document loader pipeline, split into chunks, embedded, and stored in the vector store. Returns a document ID and chunk count.

GET /documents - Lists all uploaded documents with metadata including filename, file type, chunk count, and upload timestamp.

DELETE /documents/{doc_id} - Removes a document from the vector store and metadata registry. Also deletes the original uploaded file from disk.

POST /query - Accepts a question and optional top_k parameter. Retrieves the most relevant chunks from the vector store, constructs a context-augmented prompt, sends it to the LLM, and returns the answer along with source chunks.

GET /health - Returns system status including the active LLM provider, vector store type, and total document count.