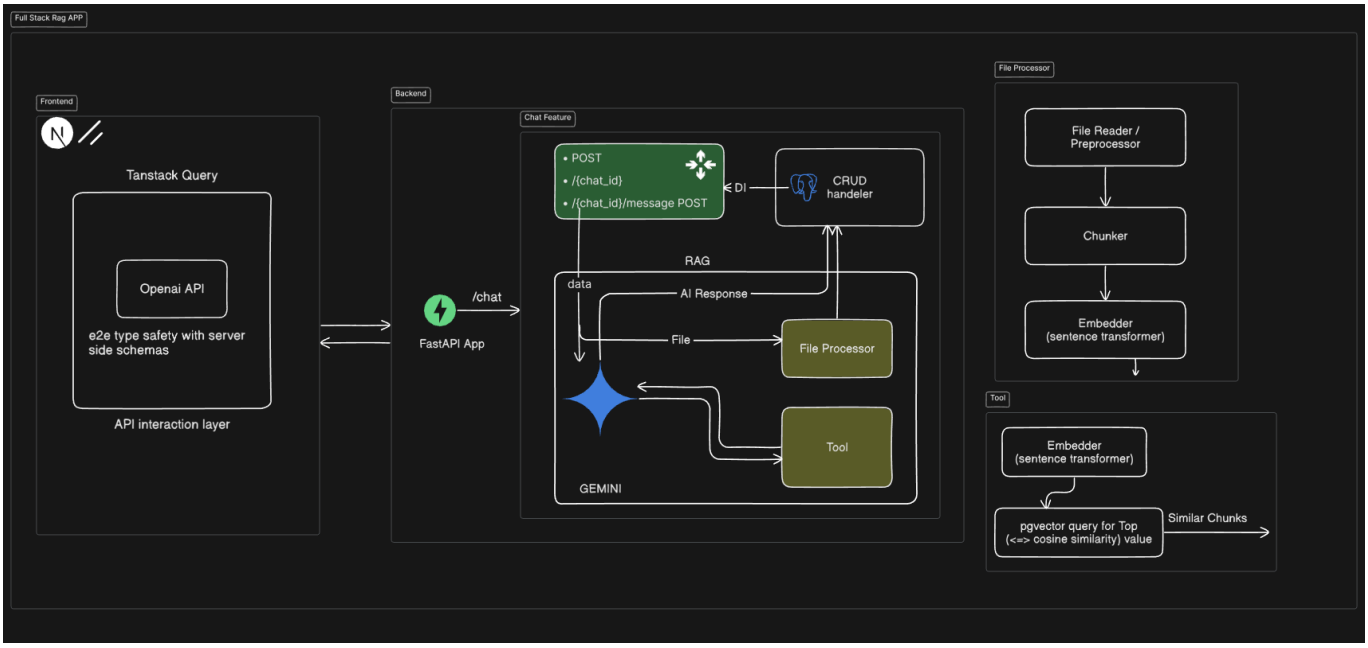
RawRAG: A Retrieval-Augmented Generation System

This project is a web application that implements a Retrieval-Augmented Generation (RAG) system using Python, FastAPI, Next.js, and TypeScript. It integrates with Google GenAI for generative AI capabilities and uses PostgreSQL for data storage along with pgvector extension for vector data.

In this project i haven't used any high level RAG frameworks and database ORM's in order to provide a clear understanding of the underlying mechanics of RAG systems.

Architecture



Key Decisions:

1. **psycopg instead of SQLAlchemy**: Lightweight and direct interaction with PostgreSQL without the overhead of an ORM. Perfect for understanding raw SQL operations as well as scope of the project.
2. **sentence-transformers**: Lightweight as Compared to other embedding models, 384-d vector embedding is sufficient for semantic similarity without burning a lot of compute resources.
3. **tool calling**: Gemini calls the readfile tool which accepts a list of queries, and similar chunks are retrieved based upon cosine similarity of query and chunk embeddings. this way, the system can handle ambiguity and provide a more accurate response.
4. **Chunking Strategy**: After experimenting with various chunk sizes and overlaps, I settled on a 2000-char chunks with 200-char overlap for optimal context retention and analysis.
5. **File validation:** Strict content_type/extension checks (PDF/TXT/MD) before processing.

RAG Pipeline

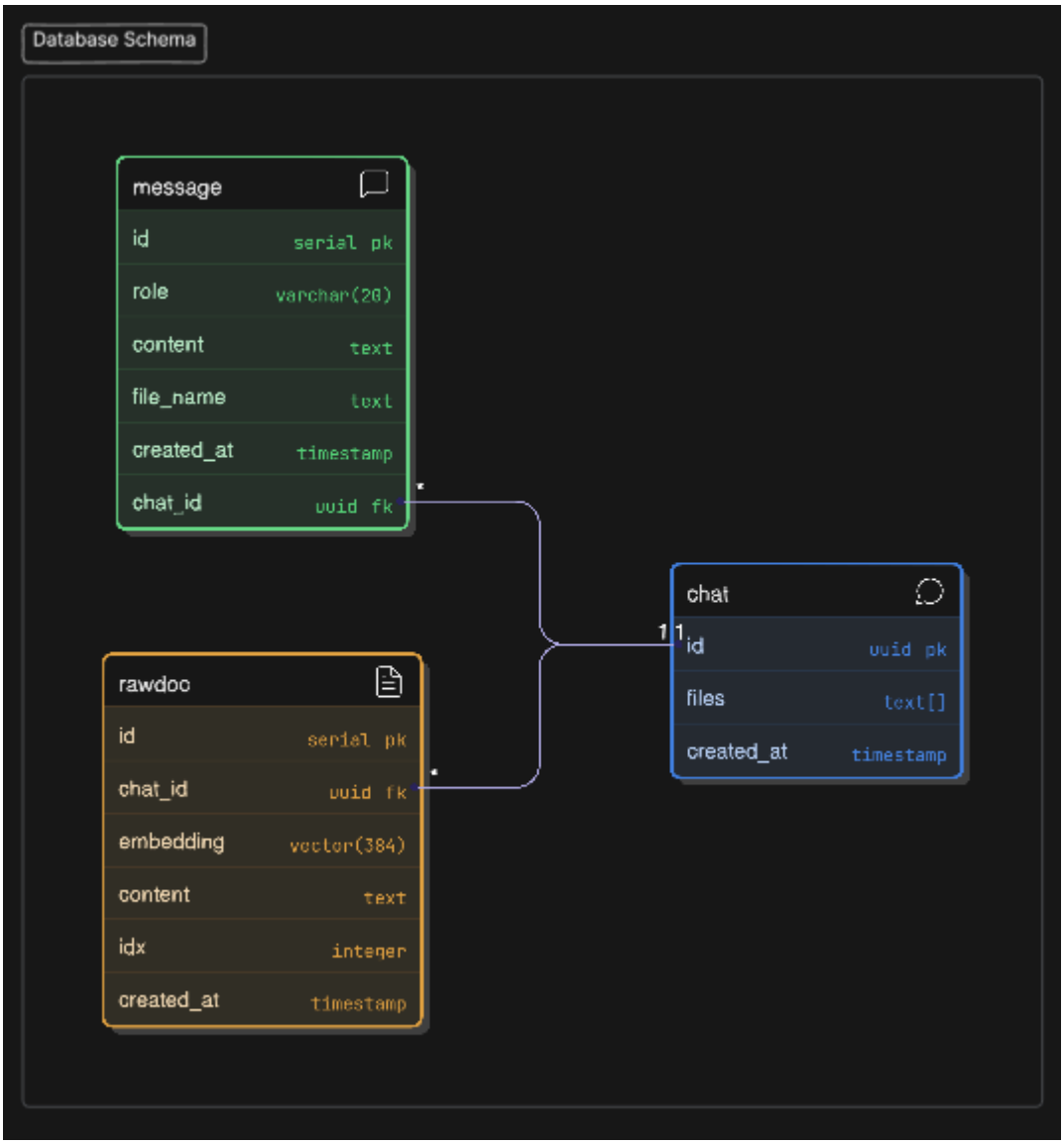
The RAG pipeline in this project consists of the following 2 main components:

1. **Document Preprocessing**: PDF documents are parsed using PyPDF to extract text content, null bytes are removed, and for the files other than PDFs, text is read directly using normal file read operations.

2. **Chunking**: The text content is chunked into smaller chunks (default 2000) with overlapping characters (default 200) to ensure that each chunk contains a sufficient amount of context for GEMINI to generate a response.
3. **Embedding**: The chunked text content is embedded sentence transformers model using the all-MiniLM-L6 vector model which generates 384-dimensional vectors.
4. **Storage**: The generated embeddings along with the original text chunks and index of chunk are stored in a PostgreSQL database using the pgvector extension.
5. **Retrieval**: The embeddings are used to retrieve the similar chunks based on cosine similarity value to the embeddings of the user's question. The retrieved chunks are then used to generate a response using GEMINI.

Database Schema

The database schema Can be found in the `api/migration.py` file. It includes tables for storing rawdocument data, chat session and chat messages.



Tables:

1. `chat`: Stores the chat sessions and their associated files.

2. `message`: Stores the chat messages, i.e user queries and AI responses.

3. `rawdoc`: Stores the raw document data along with the embeddings and the index of the chunk.

Note: Output schemas can be found in the `api/feature/schemas.py` file.