Generative AI & Chatbot Development: Python Assignment

Objective

Develop a smart **Retrieval-Augmented Generation (RAG)** API that can answer questions based on information extracted from **any document type** — including PDFs, Word files, images (OCR), .txt, and even small databases. Bonus points for supporting **image-based questions** (e.g., diagrams, scanned docs).

What It Should Do

Create a FastAPI application that:

- Accepts any document type as input (.pdf, .docx, .txt, .jpg, .png, .csv, SQLite .db, etc.)
- Extracts and preprocesses relevant content (text and/or image-based)
- Embeds content and stores in a vector store like FAISS
- Accepts **text or image-based questions**
- Performs similarity search and constructs a context prompt
- Sends the prompt to an LLM (OpenAI or similar)
- Returns a final answer via API response

Core Tasks

1. Document Ingestion

- Accept file uploads or a path/URL to a document.
- Handle:
  - .pdf via PyMuPDF or pdfplumber
  - .docx via python-docx
  - .txt directly
  - .jpg, .png, or scanned .pdf using OCR (pytesseract)
  - .csv or .db using pandas/sqlite3
- Convert all content into **clean, meaningful chunks of text** (with overlap).

2. Embeddings + Storage

- Use OpenAI embeddings (or SentenceTransformers) to generate embeddings.
- Store them in FAISS or ChromaDB.
- Save metadata (e.g., filename, page, chunk index).

3. Question Endpoint

Expose a POST /query endpoint like:

```
{
  "question": "What does the invoice say about payment terms?",
  "image_base64": "optional_base64_encoded_image"
}
```

- Perform OCR on image (if provided).
- Perform vector search based on the question.
- Construct context + question prompt.
- Send to LLM (OpenAI, Claude, etc.) and return a clean answer.

4. Bonus Features (Optional but Impressive)

- Image+text multimodal prompt support using GPT-4 Vision or Claude
- Handle multi-document querying
- Add `/upload` endpoint to upload files and return a `file_id`
- Use LangChain for chaining and orchestration
- Add file-type icons and metadata to response
- Containerize using Docker
- Minimal web frontend using Streamlit

🧪 Sample Workflow

1. **Upload File**: Uploads a `.pdf`, `.docx`, or `.jpg` via `/upload`
2. **Ask a Question**:
   - "What are the product specs mentioned in the attached PDF?"
   - "What is written in this image?" (image passed in `base64`)
3. **API Returns**:
   - Context
   - Final Answer
   - Source info (e.g., page 3 of invoice.pdf)

🚀 Technologies You May Use

- Python, FastAPI, async/await
- FAISS or ChromaDB
- OCR: `pytesseract`, `easyocr`
- Document Parsers: `pdfplumber`, `docx`, `pandas`, etc.
- Embeddings: OpenAI, HuggingFace (e.g., `all-MiniLM`)
- LLM API: OpenAI, Claude, HuggingFace Hub
- Docker (bonus)

📤 Submission

- GitHub repo or ZIP with:
  - Source code
  - Sample files
  - `README.md` with:
    - Instructions
    - API usage
    - Environment setup

        - Sample .env
- Deployed version (optional but bonus)

Evaluation Criteria

Criteria	Weight
File parsing & preprocessing	20%
Vector search + RAG flow	20%
Image OCR handling	15%
API design & FastAPI usage	15%
Prompt engineering & LLM response	15%
Bonus (Docker, LangChain, UI, etc.)	15%