**DocQA — Document Question Answering with RAG, ChromaDB, and Gemini**

This project is a **document question-answering system** built using **FastAPI**, **Streamlit**, **ChromaDB**, and **Google Gemini (LLM)**.

It allows you to:

1. Upload documents (PDF, DOCX, TXT).
2. Extract and split text into chunks.
3. Convert chunks into embeddings and store them in a **vector database (ChromaDB)**.
4. Ask natural language questions.
5. Retrieve the most relevant chunks using **vector similarity search**.
6. Pass those chunks as context to **Gemini LLM**, which generates accurate answers.

**🚀 Project Structure**

docqa/

├── backend/

│ ├── main.py # FastAPI backend (API endpoints)

│ ├── embeddings.py # Handles embedding generation and ChromaDB

│ ├── file\_utils.py # Handles file reading + text chunking

│ └── requirements.txt # Backend dependencies

│

├── frontend/

│ ├── streamlit\_app.py # Streamlit UI

│ └── requirements.txt # Frontend dependencies

│

└── README.md # Documentation (this file)

**⚙️ Setup Instructions**

**1. Clone repository**

git clone https://github.com/yourname/docqa.git

cd docqa

**2. Create virtual environment**

python -m venv venv\_docqa

source venv\_docqa/bin/activate # Mac/Linux

venv\_docqa\Scripts\activate # Windows

**🛠️ Backend Setup (FastAPI + ChromaDB + Gemini)**

**Install dependencies**

cd backend

pip install -r requirements.txt

requirements.txt should include:

fastapi

uvicorn

chromadb

pymupdf

python-docx

google-generativeai

**Start backend**

uvicorn main:app --reload --port 8000

The backend will be available at:  
👉 <http://127.0.0.1:8000>

**🎨 Frontend Setup (Streamlit)**

**Install dependencies**

cd ../frontend

pip install -r requirements.txt

requirements.txt should include:

streamlit

requests

**Start frontend**

streamlit run streamlit\_app.py

The frontend will be available at:  
👉 <http://localhost:8501>

**🔑 Google Gemini API Key**

1. Go to Google AI Studio.
2. Generate an API key.
3. Save it as an environment variable:

**Windows (PowerShell):**

setx GEMINI\_API\_KEY "your\_api\_key\_here"

**Linux / Mac (bash/zsh):**

export GEMINI\_API\_KEY="your\_api\_key\_here"

Alternatively, you can hardcode the key in main.py (not recommended for production).

**🧩 How It Works**

**Upload Phase**

* User uploads a PDF/DOCX/TXT file.
* Text is extracted:
  + **PDF** → PyMuPDF (fitz)
  + **DOCX** → python-docx
  + **TXT** → plain text read
* Text is split into **chunks** (800 characters with 100 overlap).
* Each chunk is converted into a **vector embedding**.
* Embeddings + metadata are stored in **ChromaDB**.

**Query Phase**

* User types a question.
* Question is embedded into a vector.
* Vector similarity search in ChromaDB retrieves top **K chunks**.
* Retrieved chunks are passed to **Gemini (gemini-1.5-flash)** as context.
* Gemini generates a **well-formatted, natural language answer**.

**📡 API Endpoints (Backend)**

**POST /upload**

Upload and index a document.  
**Request:** Multipart file upload.  
**Response:**

{

"status": "ok",

"filename": "document.pdf",

"num\_chunks": 15

}

**POST /query**

Ask a question and get an answer from Gemini.  
**Request:**

{

"query": "What are the core values?",

"top\_k": 4

}

**Response:**

{

"query": "What are the core values?",

"answer": "The core values are Integrity, Respect, Innovation, and Excellence.",

"sources": [

{"source\_filename": "company.pdf", "ord": 0, "text\_snippet": "Our core values are ..."}

]

}

**💡 Tips for Better Accuracy**

* Adjust **chunk size & overlap** in file\_utils.py:
* chunk\_text(text, max\_chars=1000, overlap=150)
* Increase **top\_k** when querying (e.g., 6–8 instead of 4).
* Use **gemini-1.5-pro** for more complex reasoning (slower, more accurate).
* Clean documents before upload (remove images, tables if possible).

**🔍 Example Workflow**

1. Upload company\_values.pdf.
2. Ask: *"What are the 10 company values?"*
3. System:
   * Extracts + chunks document.
   * Finds the chunk containing all 10 values.
   * Passes context to Gemini.
4. Gemini answers:
5. ✅ The company values are:
6. 1. Integrity
7. 2. Innovation
8. 3. Teamwork
9. ...
10. 10. Sustainability

**🧑‍💻 Future Improvements**

* Support for **images + OCR** (PDFs with scanned text).
* Multi-document ranking & merging.
* Caching answers for repeated queries.
* Deploy to **Docker / Cloud Run**.