**🧠 Project Title: Conversational Document QA with Gemini RAG**

**📌 Overview**

This project allows users to **upload documents** (.pdf, .txt, .docx) which are **processed and embedded** using **Google Gemini embeddings**. Users can then **query the content** of these documents using natural language, and receive context-aware answers, even across sessions. It also includes **OCR support** for scanned PDFs and images within Word documents.

**🧩 Core Components**

**1. Frontend:**

* **HTML Page** (index.html) for uploading documents and sending queries.

**2. Backend:**

* **Flask app** that handles:
  + File upload and validation
  + Document parsing (with fallback to OCR)
  + Chunking and embedding
  + Retrieval and contextual question answering
  + Session-based conversational history

**📁 File Handling**

**Supported formats:**

* .pdf (text-based or scanned)
* .txt
* .docx (text and images)

**Upload Details:**

* Saved under: ./uploads
* Metadata stored in document\_metadata.json
* Vectorstore saved in: ./chroma\_db6/{document\_id}

**🔍 Text Extraction Strategy**

| **File Type** | **Method** | **OCR Fallback** |
| --- | --- | --- |
| .pdf | PyPDFLoader | pdf2image → TrOCR |
| .docx | python-docx | Image extraction from Word → TrOCR |
| .txt | Direct load | ❌ N/A |

**🧠 Embedding & Vector DB**

* Embedding Model: **GoogleGenerativeAIEmbeddings** (models/embedding-001)
* Vector DB: **Chroma**
* Each document is chunked via RecursiveCharacterTextSplitter

**💬 RAG Pipeline (LangChain)**

**Chat Flow:**

1. Retrieve relevant chunks via filtered retriever (document\_id based).
2. Contextualize user query with session history.
3. Use Gemini Pro (gemini-1.5-pro) to generate answers.

**Components:**

* **Retriever:** Filters by document\_id, retrieves top k=3 matches.
* **Prompt Templates:**
  + Reformulate query for context-awareness.
  + Answer query concisely using retrieved chunks.
* **Memory:** ChatMessageHistory stored per session\_id.

**🔁 Session Management**

* Each conversation is tracked using a conversation\_id.
* Conversations are stateful via RunnableWithMessageHistory.

**🔒 Environment & Configuration**

* API Key managed via .env:

### 📌 ****APIs****

#### POST /api/embedding

* Uploads a document.
* Embeds and stores it with a unique document\_id.

#### POST /api/query

* Accepts a query, document\_id, and conversation\_id.
* Returns a context-aware answer.

### 📌 ****Important Paths & Files****

| **Path** | **Purpose** |
| --- | --- |
| /uploads/ | Uploaded document storage |
| /chroma\_db6/{id} | Vector DB for each doc |
| document\_metadata.json | Tracks metadata for all docs |
| .env | Contains secret keys |
| index.html | Basic upload UI |

### 📚 ****Technologies Used****

| **Category** | **Tech** |
| --- | --- |
| Backend | Flask |
| NLP | LangChain, Gemini Pro |
| OCR | TrOCR (Microsoft) |
| Vector DB | Chroma |
| Parsing | PyPDFLoader, docx, PIL |
| Embeddings | GoogleGenerativeAIEmbeddings |
| File Handling | Werkzeug, PDF2Image |
| Image Processing | Pillow |
| Prompt Engineering | LangChain Prompt Templates |
| Chat Memory | ChatMessageHistory from LangChain |
| Environment | .env via dotenv |

### 🧪 ****Future Enhancements****

* Add **user login and authentication**
* Support **multi-file retrieval** in a single chat
* Improve **frontend UX** (JS-based chat interface)
* Add **contextual citations** in the final answers
* Store user chat history in a **database (e.g., MongoDB/PostgreSQL)**