This system is a **Retrieval-Augmented Generation (RAG)** application that allows users to chat with their PDF documents. It uses a Gradio web interface, processes documents and queries using Gemini AI models, and is deployed as a scalable serverless application on Google Cloud Run.

User Interface Layer (Gradio) 🖥️

The user interface is built with **Gradio**, a Python library for creating simple web apps for machine learning models. This layer is the primary point of interaction for the user.

- **Functionality:**
    - **PDF Upload:** Users can upload a PDF document directly through the web browser.
    - **Question Input:** A text box allows users to ask questions about the uploaded document.
    - **Answer Display:** The system displays the AI-generated answers in the interface.
    - **API Key Configuration:** It provides a field for users to securely enter their Google AI API key, which is required to access the Gemini models.
- **Documentation:** Gradio Docs

RAG Pipeline Process ⚙️

This is the core logic of the application, handling everything from document ingestion to answer generation. The process ensures that the AI's answers are grounded in the content of the provided PDF.

1. Document Processing

When a user uploads a PDF, it goes through several preprocessing steps to become searchable.

- **PDF Extraction:** The system uses the **PyMuPDF** library to extract raw text content from each page of the uploaded PDF.
- **Text Chunking:** The extracted text is too long to be fed into the model at once. It's broken down into smaller, overlapping **chunks** (e.g., 1000 characters). This makes the search process more precise and manageable. Each chunk retains a reference to its original page number.
- **Document Embeddings:** Each text chunk is converted into a numerical representation called an **embedding**. This is done using Google's text-embedding-004 model. An embedding is a vector (in this case, a list of 768 numbers) that captures the semantic meaning of the text. All these vectors are stored for later comparison.

2. Query Processing & Retrieval

When a user asks a question, the system finds the most relevant information from the processed document.

- **Query Embedding:** The user's question is also converted into a 768-dimensional vector using the same text-embedding-004 model.
- **Similarity Search:** The system calculates the **cosine similarity** between the question's vector and the vector of every document chunk. A higher similarity score means the chunk's content is semantically closer to the question's intent.
- **Retrieval:** The top-K (e.g., the top 3) most similar document chunks are selected. These chunks form the **context** that will be used to answer the question.

3. Answer Generation

The final step involves using a powerful language model to synthesize an answer.

- **Context Assembly:** The retrieved chunks are combined with the original user question into a carefully crafted prompt. The prompt essentially instructs the LLM: "Using only the following information, answer this question."
- **LLM Generation:** This combined prompt is sent to a Gemini model, such as gemini-1.5-flash. The model reads the provided context and the question to generate a coherent, contextually accurate answer. This prevents the model from hallucinating or using information not present in the document.
- **Documentation:** Google AI Gemini API

Cloud Deployment (Google Cloud Run) ☁️

To make the application accessible and scalable, it's deployed on **Google Cloud Run**, a serverless platform.

- **Containerization:** The Python application, along with all its dependencies (Gradio, PyMuPDF, Google's AI library), is packaged into a **Docker container**. This ensures it runs consistently anywhere.
- **Serverless Deployment:** Cloud Run automatically starts and stops instances of the container based on incoming traffic. This means you only pay for the compute resources used while handling requests, and it can automatically scale up to handle many users or scale down to zero when inactive.
- **Accessibility:** Once deployed, the application is assigned a public URL and listens for requests on a specified port (e.g., 8080).
- **Configuration:** Sensitive information like the Google AI API key can be managed securely as **environment variables** in the Cloud Run service, rather than being hardcoded in the application.
- **Documentation:** Google Cloud Run Docs

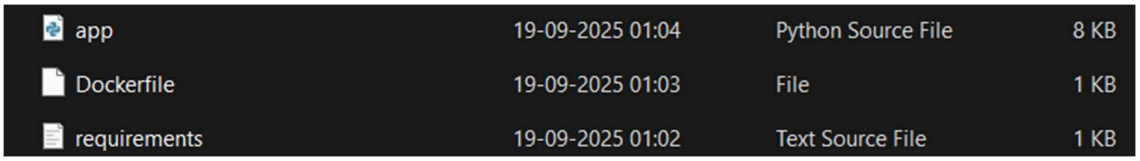
External Dependencies 🔗

The system relies on Google's powerful AI services to function.

- **Google AI Studio API:** This is the endpoint the application communicates with to access Google's AI models for both embedding text and generating answers.
- **Gemini Models:**
    - **Embedding Model (text-embedding-004):** Used for converting text (both document chunks and user queries) into semantic vectors.
    - **Generative Model (gemini-1.5-flash):** The large language model that understands the context and question to generate the final human-like answer.
- **Documentation:** Gemini

Screenshots:

Folder Structure:



Rest was coding part.

For cloud run:

```
# Authenticate
gcloud auth login
gcloud config set project YOUR_PROJECT_ID

# Build Docker image
gcloud builds submit --tag gcr.io/YOUR_PROJECT_ID/gemini-rag

# Deploy to Cloud Run
gcloud run deploy gemini-rag \
  --image gcr.io/YOUR_PROJECT_ID/gemini-rag \
  --platform managed \
  --region asia-south1 \
  --allow-unauthenticated
```