Purpose

This PDF is included as a demo document for the RAG MVP project. It is designed to test:

- PDF text extraction
- chunking
- embedding creation
- vector retrieval using cosine distance
- answer generation grounded only in retrieved sources

What this project does

The RAG MVP allows users to ingest documents and ask questions about them. The system works as a pipeline:

1. **Ingest**
- The document is stored as a Document record.
- The text is split into multiple chunks.
- Each chunk is embedded into a vector.
2. **Retrieve**
- A user question is embedded into a vector.
- The system compares the question vector to stored chunk vectors.
- The system selects the top-K most similar chunks.
3. **Generate**
- The model receives the question and the retrieved chunks as “Sources.”
- The model is instructed to answer using only those sources.
- If the sources do not contain the answer, the model should respond: “I don’t know.”

Key technologies

- The backend is built with Django (Python).
- Vectors are stored in Postgres using the pgvector extension.
- Vector retrieval uses cosine distance (smaller distance means a closer match).
- Embeddings are generated using an embeddings model.
- Answers are generated using a chat model.

Important definitions

- **Document:** A stored file or text entry you ingest (PDF, pasted text, or .txt file).
- **Chunk:** A piece of the document text stored separately for retrieval.
- **Embedding:** A numeric vector representation of text used for similarity search.
- **Cosine distance:** A measure of how similar two embeddings are. Lower distance is better.

Why chunking matters

Chunking improves retrieval quality. Instead of searching one huge document, the system searches smaller chunks to find the specific section that best matches the question. Chunk overlap can help prevent losing context when an important sentence is near the boundary between two chunks.

Why a vector database (pgvector) is useful

Traditional keyword search fails when the user uses different wording than the document. Vector search is semantic:

- “car engine” can match “vehicles use motors”
- “summarize the document” can match the sections that describe overall purpose

Using Postgres + pgvector keeps things simple because the project can store:

- normal relational data (documents, logs)
- vector data (chunk embeddings)
  in one database.

Guardrails (“I don’t know”)

The app can return “I don’t know” when:

- no chunks exist
- retrieved chunks are too far away (distance is above a configured max_distance)
  This prevents the model from confidently hallucinating unsupported answers.

Observability (logging)

For each question, a log entry may store:

- the question
- k (how many chunks were retrieved)
- the document ID used
- best distance score
- latency (time the request took)
- errors (if any)

This makes it easier to debug retrieval quality and performance.