API Documentation

This documentation provides a comprehensive overview of the REST API for the RAG-based chatbot, as implemented in app.py. The API is built using FastAPI and serves as the backend for document management and conversational intelligence.

---

1. Overview

The backend provides endpoints for managing the lifecycle of government documents (upload, indexing, deletion) and interacting with the RAG (Retrieval-Augmented Generation) graph for grounded question answering.

Base URL: http://localhost:8000

---

2. Core Endpoints

A. Conversational Chat

Interacts with the LangGraph-based RAG workflow to retrieve relevant context and generate a cited response.

- **Endpoint:** POST /chat
- **Request Body (ChatRequest):**

JSON

```
{
  "question": "What are the current tax filing requirements?",
  "thread_id": "unique-session-id-123"
}
```

- **Success Response (ChatResponse):**
    - **answer:** A JSON string containing synthesized_answer, detailed_analysis, and confidence_score.
    - **sources:** A list of unique filenames used in the retrieval process.
    - **confidence_score:** A float (0.0–1.0) indicating the LLM's certainty based on context availability.
    - **latency_ms:** Time taken to process the request in milliseconds.

B. Document Management

1. Upload Document

Uploads a file for background ingestion and indexing. Supported formats include PDF, CSV, TXT, and MD.

- **Endpoint:** POST /upload
- **Payload:** multipart/form-data with a file field.
- **Behavior:** The file is saved to a temporary directory, and an asynchronous background task (run_ingestion) is triggered to process the file into the vector store.
- **Response:**

JSON

```
{
  "status": "queued",
  "filename": "policy_manual.pdf"
}
```

2. List Documents

Retrieves a unique list of all filenames currently indexed in the vector database.

- **Endpoint:** GET /documents
- **Response:**

JSON

```
{
  "documents": ["manual.pdf", "data.csv"],
  "count": 2
}
```

3. Delete Document

Removes all indexed chunks associated with a specific filename from the vector store.

- **Endpoint:** POST /delete
- **Request Body:**

JSON

```
{
```

"filename": "policy_manual.pdf"

}

- **Response:**

JSON

{

"status": "success",

"message": "Deleted policy_manual.pdf"

}

---

3. Data Models

GraphState (Internal State Management)

Used by the orchestration layer to track data across the RAG nodes:

- **question:** The user's input string.
- **documents:** List of retrieved LangChain Document objects.
- **metadata_manifest:** List of source filenames.
- **confidence_score:** Calculated grounding score.

---

4. Processing Pipelines

Ingestion Pipeline (run_ingestion)

When a file is uploaded, the backend performs the following:

1. **Parsing:** Extracts text based on file type (e.g., pdfplumber for PDFs).
2. **Heading Awareness:** Uses regex to identify document sections (e.g., "Section 1") for better metadata tagging.
3. **Summarization:** Generates a one-sentence summary for every text chunk to assist in "Smart Retrieval".
4. **Vectorization:** Converts text to embeddings using all-MiniLM-L6-v2 and stores them in ChromaDB.

Retrieval Pipeline (SmartRetriever)

The invoke method in retriever.py executes:

1. **Query Analysis:** LLM-driven intent extraction to determine if specific metadata filters (like header or source) are needed.
2. **Hybrid Retrieval:** Performs semantic search followed by a FlashRank reranking step to prioritize the most relevant chunks.
3. **Caching:** Results are cached via DiskCache for 3600 seconds to reduce LLM costs and latency.

---

5. System Health

- **Endpoint:** GET /health
- **Response:** {"status": "healthy"}
- **Logging:** All requests are logged with method, path, status code, and duration via middleware.