Architecture Overview

This architecture overview document outlines the development of a Retrieval-Augmented Generation (RAG) chatbot tailored for Government Technology (GovTech). The architecture is designed for transparency, security, and factual grounding, drawing from the provided implementation files and industry best practices for the public sector.

Architecture Overview: RAG-Based Chatbot for GovTech

1. System Intent and Objectives

The primary goal is to provide citizens and government employees with a reliable AI assistant that delivers accurate information grounded in verified official documents (e.g., tax laws, policy manuals, or service guidelines).

- **Factual Grounding:** Mitigates "AI hallucinations" by providing specific facts as part of the LLM input prompt.
- **Freshness:** Accesses up-to-date information without the need for frequent model retraining.
- **Source Attribution:** Ensures transparency by citing specific documents used to generate an answer.

---

2. High-Level Architecture

The system follows a modular 5-part structure to ensure scalability and maintainability:

1. **Ingestion Pipeline:** Pre-processes and indexes government documents.
2. **Vector Database:** Stores document embeddings for semantic search.
3. **Retrieval Engine:** Identifies and ranks relevant information based on user intent.
4. **Orchestration Layer (Graph-Based):** Manages the workflow between retrieval and generation.
5. **User Interface:** A conversational front-end for user interaction.

---

3. Core Components

A. Data Ingestion & Indexing (ingestion.py)

This component handles the lifecycle of government documents before they are searchable.

- **Multi-Format Support:** Processes PDFs, CSVs, TXT, and Markdown files.
- **Semantic Chunking:** Uses a RecursiveCharacterTextSplitter with a chunk size of 1000 and overlap of 100 to preserve context between segments.
- **Vectorization:** Transforms text chunks into high-dimensional vectors using sentence-transformers/all-MiniLM-L6-v2.

B. Storage Layer

- **Vector Store:** ChromaDB is used to persist embeddings locally, allowing for fast semantic similarity searches.
- **Metadata Enrichment:** Every chunk is tagged with its source file and header to enable precise filtering and citation.

C. Smart Retrieval Engine (retriever.py)

Rather than a simple keyword search, this layer employs "Smart Retrieval":

- **Intent Analysis:** Uses an LLM to clean queries and determine if specific metadata filters (e.g., searching only "Tax" documents) should be applied.
- **Hybrid Search & Reranking:** After initial vector search, a Ranker (e.g., ms-marco-MiniLM-L-12-v2) re-scores results to ensure the most relevant context is prioritized.

D. Orchestration & Graph Logic (graph.py)

The system uses a state-graph workflow to manage complex logic:

- **Conditional Routing:** If the retriever finds no relevant documents, the system routes to a "no data" node instead of allowing the LLM to guess.
- **Deterministic Synthesis:** The synthesize_node uses a temperature of 0 for consistent, grounded responses and requires the model to output a confidence score.

E. Backend API (app.py)

Built with FastAPI, this layer serves as the bridge between the UI and the RAG logic.

- **Background Tasks:** Handles document ingestion as background processes to prevent UI blocking.
- **State Management:** Tracks unique thread_id sessions for conversational continuity.

---

4. Government-Specific Considerations

Feature	GovTech Requirement	Implementation Strategy
Data Security	Protection of PII and sensitive records	Local vector storage (Chroma) and zero-trust principles.
Auditability	Transparency in how answers were derived	Mandatory sources_cited in every synthesized response.
Reliability	Consistent and trustworthy public info	Low-temperature LLM settings and high-relevance reranking.
Cost Control	Efficient use of public funds	RAG avoids expensive model fine-tuning or retraining.

---

5. Technology Stack

- **LLM:** Llama-3.3-70b-versatile (via Groq).
- **Orchestration:** LangGraph / LangChain.
- **Embeddings:** HuggingFace Sentence Transformers.
- **Database:** ChromaDB (Vector) and DiskCache (Query Cache).
- **Frontend:** Streamlit for rapid internal prototyping.