## **RAG Question Answer Application Technical Documentation**

### **Overview**

This is a Retrieval-Augmented Generation (RAG) Question Answering application built using **Python**, **Streamlit**, **ChromaDB**, **Ollama**, and **LangChain**. The application allows users to process JSON documents, create a vector store, and ask questions against the stored documents.

## **Key Components**

### **Dependencies**

* **os**: File and path operations
* **json**: JSON data handling
* **chromadb**: Vector database for document storage and retrieval
* **ollama**: Local language model inference
* **streamlit**: Web application framework
* **langchain\_community.document\_loaders**: JSON document loading
* **sentence\_transformers**: Cross-encoder for document re-ranking

## **System Architecture**

### **1. Document Processing**

* **Function**: process\_json\_file(file\_path)
  + Processes JSON files using LangChain's JSONLoader.
  + Extracts documents with customizable metadata.
  + Supports flexible JSON structures.

### **2. Vector Store Management**

* **Function**: get\_vector\_collection()
  + Uses ChromaDB for vector storage.
  + Employs Ollama's embedding function with the nomic-embed-text model.
  + Configures a cosine similarity space for embeddings.

### **3. Document Indexing**

* **Function**: add\_to\_vector\_collection()
  + Clears the existing collection before adding new documents.
  + Generates unique IDs for each document split.
  + Supports batch document insertion.

### **4. Semantic Search**

* **Function**: query\_collection()
  + Retrieves relevant documents based on the query.
  + Configurable number of results.
  + Uses vector similarity search.

### **5. Re-ranking Mechanism**

* **Function**: re\_rank\_cross\_encoders()
  + Utilizes a cross-encoder model for more accurate relevance scoring.
  + Selects the top 3 most relevant documents.
  + Improves the initial semantic search results.

### **6. Language Model Interaction**

* **Function**: call\_llm()
  + Uses Ollama's llama3.2:3b model.
  + Implements a detailed system prompt for context-aware responses.
  + Streams model responses.

## **User Interface (Streamlit)**

* **Document Processing**: Accessed via the sidebar.
* **Question Input**: Text area for user queries.
* **Response Display**: Streaming responses displayed interactively.
* **Expandable Sections**: View retrieved documents and their IDs.

## **System Prompt Design**

The system prompt guides the language model to:

* Analyze context thoroughly.
* Organize thoughts systematically.
* Generate comprehensive answers.
* Use clear, structured formatting.
* Rely exclusively on provided context.

## **Key Workflows**

### **1. Document Processing Workflow**

1. Load JSON file.
2. Split documents.
3. Generate embeddings.
4. Store in the vector database.

### **2. Question Answering Workflow**

1. User submits a query.
2. Retrieve semantically similar documents.
3. Re-rank documents.
4. Generate a context-aware response.
5. Stream the response to the user.

## **Configuration and Customization**

* Modify embedding models easily.
* Adjust document processing parameters.
* Configure vector store settings.

## **Potential Improvements**

* Support multiple file formats.
* Add advanced filtering options.
* Implement more sophisticated re-ranking methods.
* Support multiple language models.
* Add caching mechanisms.

## **Dependencies Installation**

bash

Copy code

pip install chromadb ollama streamlit langchain sentence-transformers

## **Usage**

1. Prepare a JSON document.
2. Run the Streamlit application.
3. Process the document.
4. Ask questions.

## **Error Handling**

* Checks for file existence.
* Handles processing and query exceptions.
* Provides user-friendly error messages.

## **Performance Considerations**

* Uses efficient embedding and search techniques.
* Streams model responses for faster feedback.
* Minimizes computational overhead.

## **Security Notes**

* Relies on a local language model.
* No external API dependencies.
* Processes documents locally.