** RAG Integration with a System Prompt**

1. Overview & Goal

We want to enhance our existing Llama-based AI solution by integrating Retrieval Augmented Generation (RAG) with ChromaDB **and** a dedicated system prompt. The system prompt establishes the AI's “role” and global rules, while RAG ensures the AI can fetch and use relevant information from our document repository.

2. Current Setup

- **Model**: Llama (running on a VPS, served via Flask)

- **Vector Database**: ChromaDB, already set up for embedding and retrieving relevant document chunks

- **RAG**: We have an established pipeline that retrieves the top-k documents from ChromaDB based on user queries

3. Objective

- Implement a **global system prompt** that dictates the AI’s style and behavior (e.g., “the AI is an expert in X and should respond with Y style”).

- Ensure every user query includes both the system prompt (role definition) **and** the retrieved context from ChromaDB.

- Maintain the existing RAG flow so that relevant content is always included for the final answer.

---

4. Scope & Tasks

1. **System Prompt Implementation**

  - Create a **System Prompt** or “System Message” that establishes the AI’s role. This might look like:

    \[

\text{[INST] <<SYS>> <System Instructions> <</SYS>> User Query [/INST]}

\]

or a “prefixed prompt” in a non-chat environment.

- Ensure this prompt is prepended (or inserted in the designated “system” section) every time the model is called.

2. **Context Retrieval & Prompt Assembly**

- **Context Retrieval**: Use our existing embedding-based search to fetch the most relevant document chunks from ChromaDB.

- **Prompt Assembly**: Combine:

1. The **system prompt** (global role & rules)

2. The **retrieved chunks** (as relevant context)

3. The **user’s query**

- Finalize a structured format that the Llama model can parse and respond to (either the Llama-Chat format or a custom text format).

3. **API/Flask Integration**

- Adapt our existing Flask routes (API endpoints) so that each user request triggers:

1. Embedding calculation and retrieval from ChromaDB (top-k chunks).

2. Prompt construction (system prompt + context + user query).

3. Model inference.

4. Return the final answer to the user.

4. **Testing & Validation**

- Implement a few sample queries to confirm:

- The system prompt is correctly enforcing the AI’s style/role.

- The retrieved chunks are relevant and properly integrated into the final response.

- Check if the AI answers align with our requirements (e.g., language, formatting, accuracy, disclaimers, etc.).

5. **Documentation**

  - Provide clear documentation on how to modify the system prompt (for administrators).

  - Explain where to add or remove documents in ChromaDB, plus any indexing steps.

---

5. Acceptance Criteria

- **System Prompt** is consistently applied to all queries.

- The AI's responses reflect the specified role (e.g., uses domain-specific knowledge, keeps the desired tone, etc.).

- RAG retrieval works seamlessly, injecting relevant context into the prompt with minimal latency.

- Clear instructions exist for maintaining and adjusting the system prompt and ChromaDB embeddings as the knowledge base evolves.
