Project Title: Lamma – Development

Objective

To design and develop an open-source chatbot using the **Llama 3** model family, capable of answering both general and domain-specific queries through Retrieval-Augmented Generation (RAG).
The system will run **entirely on local machines** with a Python-based user interface.

Tasks

1. **Model & Environment Setup**
    - Study the **Llama 3** architecture, model variants, and license terms.
    - Install and configure **Ollama**, **vLLM**, or **llama.cpp** for local model serving.
    - Run sample prompts to verify local inference.
    - Document installation steps, dependencies, and environment configuration.
2. **API Layer Development**
    - Build a **FastAPI** backend to handle user queries and forward them to the model.
    - Create REST endpoints such as `/chat` and `/v1/chat/completions`.
    - Implement basic authentication (API key or local token).
    - Prepare small Python test scripts or Postman collections to validate API calls.
3. **Retrieval-Augmented Generation (RAG) Integration**
    - Set up a **local vector database** (pgvector / Qdrant / Chroma).
    - Develop a document ingestion pipeline (PDF → text → chunks → embeddings).
    - Integrate retrieval into the chatbot flow (retrieve → context → generate).

   - Test the chatbot’s ability to answer from uploaded documents.
   - Measure accuracy and latency in local setup.

4. **Python-Based Front-End Interface**
   - Build an interactive chatbot interface
   - Connect the interface to the FastAPI backend for real-time chat.
   - Add features such as “Clear Chat,” “Upload Document,” “Show References,” and “Toggle System Prompt.”
   - Ensure smooth interaction and proper error handling within the Python interface.

5. **Monitoring & Logging**
   - Implement **Python logging** to record queries, responses, and system errors.
   - Track response times and token usage locally.
   - Maintain version logs for model, embeddings, and configuration changes.
   - Visualize simple metrics (e.g., requests per session, response delay) using Matplotlib or Plotly if needed.

6. **Safety & Governance Layer**
   - Integrate **Llama Guard** or a rule-based filter for input/output moderation.
   - Block unsafe, irrelevant, or sensitive queries.
   - Display a disclaimer and usage policy within the Python interface.
   - Document moderation test cases and outcomes.

7. **Documentation & Reporting**
   - Prepare detailed technical documentation covering:
     - Architecture diagram
     - API structure

        - RAG workflow
        - Front-end interface overview
    - Include setup and execution instructions for local machines.
    - Prepare a short demo video showing the chatbot in action.
    - Publish source code and documentation in a local or GitHub repository.

Deliverables

- Locally running chatbot accessible via Python interface
- Functional API connected to Llama 3 model
- Document-aware (RAG-enabled) query handling
- Logging and moderation mechanisms
- Full technical documentation and demo video