Overview

Your objective is to build a **small AI-driven application** showcasing key elements of **LangChain v0.3** and **LangGraph v0.3**:

1. **Document ingestion and retrieval** (with embeddings and a local vector store).
2. A **multi-step or agentic flow** orchestrated by **LangGraph**.
3. **Memory** for multi-turn context.
4. **Streaming** or partial/step-by-step output.
5. **Basic reliability/error-handling** to demonstrate real-world readiness.

Note: You can **choose** any LLM or additional external APIs and tools, as needed. If your solution depends on third-party services (e.g., external data or special APIs), just provide **clear instructions** (e.g., how to obtain credentials or keys). There is **no requirement** that the chosen LLM or external services be free to use, only that your instructions make it feasible for a reviewer to replicate or test your solution if they also have the same LLM account (e.g., an OpenAI API key).

High-Level Requirements

1. **Document Ingestion & Retrieval (RAG)**

    - Ingest a small local dataset (a handful of text/Markdown files).
    - Split these documents and create embeddings.
    - Store them in a local vector store (e.g., ChromaDB) so it's easy to spin up and test.
    - Illustrate how you retrieve relevant chunks to ground the LLM's outputs.

2. **LangChain Components**

    - Implement at least one **Prompt Template** or a chain using Runnables.
    - Show usage of a **tool** (function calling, or any other accessible external function) if you aim for agentic behavior.

3. **LangGraph Flow**

    - Define a **graph**.
    - **If you claim it's an agent** (like ReAct/CodeAct), ensure it genuinely demonstrates "Think → Act → Observe" or a comparable agentic loop.
    - Otherwise, a standard multi-step workflow is also fine, so long as your graph includes some meaningful branching or multi-step transitions.

4. **Memory**

    - Maintain conversation or contextual state across multiple user turns.
    - This can be done with either the state or via **LangGraph's** more advanced memory/persistence.

5. **Streaming**

    - Demonstrate partial or incremental output – e.g., token-by-token streaming from the LLM or real-time updates about the agent's steps.

6. **Reliability / Error Handling**

    - Show a small mechanism to handle or log errors (invalid LLM output, timeouts, etc.).
    - Optionally, use a callback or a mini fallback node if the system fails to produce expected results.

7. **User Interface**

    - Provide a **simple** but testable UI. You can pick **any** approach (CLI, Streamlit, Gradio, minimal web server, etc.) so reviewers can quickly see the system in action.
    - Let the user run multiple queries that demonstrate memory across sessions, retrieval from your vector DB, and your multi-step logic.

8. **Documentation & Setup**

    - **README** with steps to install, run ingestion (if separate), and start the interface.
    - List any environment variables or secrets needed (e.g., an `.env.example`).
    - Show how a reviewer can test it (e.g., "Run `python main.py` and ask about X. You'll see retrieval from the local docs.").

Deliverables

1. **Source Code** in a well-structured format (repo or archive).
2. **README** detailing installation, usage, plus any setup for external APIs if applicable.
3. **Scripts or Functions** for data ingestion and app execution.
4. **Optional Tests** if you have time (simple integration or unit tests).

Review Criteria

- **Correct Usage of LangChain & LangGraph**: Appropriate use of models, memory, Runnables, compiled graphs, etc.
- **RAG Implementation**: Are you effectively retrieving relevant data from the local DB to ground LLM answers?
- **Agentic / Multi-Step Flow**: If you present an agent, does it genuinely show agentic steps? If not an agent, is your multi-step logic structured, branching, or otherwise meaningful beyond a single LLM call?
- **Memory & Streaming**: Are conversation details retained, and is partial output shown?
- **Reliability**: Basic logging, callbacks, or error handling in place.
- **Documentation**: Clear instructions so others can replicate or test with their own LLM keys or service credentials.

Feel free to add extra flair or features. The goal is to demonstrate a **small but realistic** LLM-based application using the latest **LangChain** and **LangGraph** capabilities in a way that's straightforward for others to run and review. Good luck!