Objective

Assess the candidate's experience in:

- Building LLM-based agentic bots (multi-agent pipeline)
- Implementing a RAG (Retrieval-Augmented Generation) flow
- Modular architecture and clean code
- Integration with external systems (webhooks/API)
- Prompt design—even if stubbed—and testing/documentation

"Product-Query Bot via RAG Pipeline"

Scenario

Zubale needs a microservice that receives user questions about products (simulated WhatsApp messages), retrieves relevant information from a document corpus, and responds via a webhook callback.

Requirements (Time-boxed, max 3 hours)

1. **Incoming Endpoint**
    - **POST /query** accepting JSON

```JSON
{ "user_id": "string", "query": "string" }
```

    - Validate input and enqueue the request (e.g., using Python asyncio.Queue).

2. **RAG Pipeline**
    - Index ~5–10 product description documents into an embedded vector store (in-memory or file-based).
    - On receiving a query:
        - Retrieve top-k documents semantically similar to the query.
        - Send a prompt to a stubbed LLM function (e.g., ai_generate(context, query)) that returns a generated answer grounded in retrieved context.

3. **Multi-Agent Structure**
    - Implement at least two agents:
        - **Retriever Agent**: handles semantic retrieval.
        - **Responder Agent**: compiles generation logic.

- Either use a framework like LangGraph or a clean custom orchestration to emphasize modularity and “multi-agent” separation .

4. **Callback Integration**
   - After generation, send the answer via POST to a configurable CALLBACK_URL (log output is acceptable if external endpoint isn’t available).

5. **Code Quality**
   - Include unit tests (e.g., pytest or jest), at least for the retrieval logic or prompt flow.
   - Use environment variables for configuration (e.g., CALLBACK_URL, top-k).
   - Provide clear documentation: setup, run instructions, time spent.
   - Containerization with Docker is a plus but not required.

Evaluation Criteria

Dimension	**What We Assess**
RAG Pipeline	Understanding of retrieval + generation, avoiding hallucination
Agent Design	Clean separation of concerns or use of agent frameworks
Integration	Webhook/API flow simulating real product use
Prompt & LLM Mock	Evidence of thoughtful prompt structure
Testing & Documentation	Clarity, readability, minimal automated coverage
Engineering Practices	Modularity, configuration, Docker usage, code style

---

Optional Enhancements (if time permits)

- Real API call to an LLM (e.g., OpenAI, Claude).
- A memory or context agent to track conversation history.
- Use of Docker Compose to wire up the service.
- A frontend script to simulate POST requests.
- Basic evaluation with automated unit scoring or vetting.

Deliverables

1. Git repo with:
   a. Source code and test suite
   b. README with: instructions to run/index, how to test flow, time taken (indicative of 1–2 h)
2. (Optional) Dockerfile and/or docker-compose.yml
3. Example POST commands or scripts demonstrating the exercise
4. **Video sharing the screen and explaining the outputs and a demo, ideally with a duration of less than 6 minutes**