Agentic RAG Automation Hackathon

Detailed Problem Description & Example Question Pattern

1. Introduction & Motivation

Traditional chatbots answer questions in a single step and often hallucinate when reliable data is missing. Modern AI systems instead use **Retrieval-Augmented Generation (RAG)** and **agentic workflows** to reason, retrieve knowledge, call tools, and execute tasks step by step.

In this hackathon, participants will build a **multi-step automation agent** that combines: Document retrieval using vector databases Decision-making using agent graphs Backend tool execution via APIs The goal is not to build a chatbot, but to design an **intelligent system** that can plan and act.

2. Problem Objective

Your task is to design and implement an **Agentic RAG Automation System** that can: Understand a user request Decide what steps are required Retrieve information from private documents Call external tools via backend APIs Generate a final, grounded response The system must operate using **multiple sequential steps** rather than a single prompt-response interaction.

3. System Overview (High-Level Flow)

A typical execution of your system should follow this pattern: User submits a query The agent plans the required steps Relevant documents are retrieved Backend tools are called if needed Intermediate results are stored in agent state A final verified response is produced Each step must depend on the previous step's output.

4. Module 1: Document Ingestion & Vector Store

In this module, participants will prepare private knowledge for retrieval.

Example:
You are given a PDF containing university rules. The system should split the document into chunks, generate embeddings, and store them in ChromaDB.

Key Requirements: Document loading (PDF or text) Chunking with configurable size and overlap Embedding generation using *sentence-transformers/all-MiniLM-L6-v2* Persistent ChromaDB storage

5. Module 2: Retrieval-Augmented Generation (RAG)

This module ensures that the system answers questions using retrieved knowledge.

Example Question:
“What is the attendance requirement mentioned in the policy document?”

Expected Behavior: Retrieve relevant chunks from ChromaDB Pass retrieved text as context to the LLM

Generate an answer only from that context If the information is missing, respond with:
"I don't know based on the provided documents."

6. Module 3: Backend Tool Server

Agents often require capabilities beyond language generation. In this module, participants must expose tools via a backend API.

Example Tools: Wikipedia search Calculator Text statistics analyzer **Important Rule:**
Tools must be accessed via HTTP APIs. Direct local function calls inside the agent are not allowed.

7. Module 4: Agent Design with LangGraph

Participants must design an agent using **LangGraph**.

Minimum Required Nodes: Planner / Router: decides next action Retriever: fetches document context Tool Executor: calls backend APIs Answer Generator / Refusal Node The agent must use conditional transitions and shared state.

8. Module 5: Multi-Step Automation Agent (MANDATORY)

This is the core of the challenge. Your agent must execute a task using multiple dependent steps.

Example Automation Task:
“From the document, identify total participants and calculate the percentage of first-year students.”

Expected Steps: Retrieve relevant document sections Extract numerical data Call calculation tool via backend API Generate final explanation

9. Module 6: Final Output & Validation

The final output must be: Grounded in retrieved context Verified using tool outputs Free from hallucination If the system lacks sufficient data, it must explicitly refuse to answer.

10. Bonus Module: MCP Server Integration (Optional)

Participants may integrate a **Model Context Protocol (MCP) server**.

Bonus Examples: Fetching structured context from MCP Delegating a sub-task to MCP Combining MCP output with RAG context Successful integration will receive bonus points.

11. Evaluation Criteria

Scoring Breakdown: Multi-step automation logic – 30% RAG correctness – 20% Backend tool usage – 20% LangGraph design – 15% Code clarity & robustness – 15% **Bonus:** MCP integration (+10%)

12. Final Notes

This hackathon focuses on **thinking in systems**. You are encouraged to design clean workflows, clear agent states, and reliable decision logic. Complexity is less important than correctness and clarity.