24-Hour GenAI Intern Assignment – AI Operations Assistant

Objective

Build an AI Operations Assistant that accepts a natural-language task, plans steps, calls tools (APIs), and returns a final structured answer. The system must run locally and demonstrate agent-based reasoning, LLM usage, and real API integrations.

Core Capabilities

• Multi-agent architecture (Planner, Executor, Verifier)
• LLM-powered reasoning and structured outputs
• Integration with at least two real third-party APIs
• Runnable locally on localhost via API, UI, or CLI

Mandatory Architecture

Planner Agent: Converts user input into a step-by-step plan and selects tools.
Executor Agent: Executes steps and calls APIs.
Verifier Agent: Validates results and fixes missing or incorrect output.

API & LLM Requirements

Candidates must integrate at least two real APIs (e.g., GitHub, Weather, News). Any major LLM provider may be used. Monolithic prompts are not allowed.

Project Structure

ai_ops_assistant/
■■■ agents/
■■■ tools/
■■■ llm/
■■■ main.py
■■■ requirements.txt
■■■ .env.example
■■■ README.md

Evaluation Criteria

Agent design (25), LLM usage (20), API integration (20), code clarity (15), working demo (10), documentation (10). Pass score: 70.

Sample Solution Outline

High-Level Flow:
1. User submits a natural language task via API/UI.
2. Planner Agent uses LLM to produce a JSON plan with steps and required tools.
3. Executor Agent iterates through steps, calling tools (GitHub API, Weather API, etc.).
4. Verifier Agent checks completeness, retries missing data, and formats final output.

Example Tools:
• GitHubTool: Search repositories, fetch stars and descriptions.
• WeatherTool: Fetch current weather by city.

LLM Usage:
• Planner prompt constrained to JSON schema.
• Verifier prompt ensures output quality and schema compliance.

Error Handling:
• Retry on API failure.
• Graceful fallback when partial data is available.

Improvements With More Time:
• Caching API responses
• Cost tracking per request
• Parallel tool execution