GenAI Assignment Documentation

Multi-Agent AI Ops Assistant

Candidate: Manas Kumar Rout
Company: TrulyMadly
Role: GenAI Intern

---

1. Project Overview

This project implements a **Multi-Agent AI Operations Assistant** that can understand a user task, plan execution steps, call real third-party APIs, validate results, and return a complete final response.

The system is designed to demonstrate:

- Multi-agent reasoning
- Structured LLM outputs
- Real API integrations
- End-to-end execution
- Local execution with a single command

The project satisfies all **mandatory pass/fail criteria** specified in the assignment email.

---

2. High-Level Architecture

Architecture Flow

User Input

↓

Planner Agent (LLM → Structured JSON Plan)

↓

Executor Agent (Tool/API Calls)

↓

Verifier Agent (Validation + Retry)

↓

Final Answer

Key Design Principles

- Separation of concerns via agents
- Deterministic structured planning
- Tool-based execution (no hardcoded responses)
- Fault tolerance via verifier retry

---

3. Multi-Agent Design

3.1 Planner Agent

Purpose:
Transforms a natural-language task into a **structured JSON execution plan**.

Responsibilities:

- Understand user intent
- Decide which tools/APIs are required
- Output a machine-readable plan

Example Output:

```
{
  "task": "Get current weather in Mumbai",
  "steps": [
    {
      "tool": "weather.current_by_city",
      "tool_input": { "city": "Mumbai", "units": "metric" },
      "save_as": "mumbai_weather"
    }
  ]
}
```

3.2 Executor Agent

Purpose:
Executes each step from the planner by invoking real APIs.

Responsibilities:

- Call registered tools
- Handle API responses
- Store outputs in a structured format

3.3 Verifier Agent

Purpose:
Ensures the task has been fully satisfied.

Responsibilities:

- Validate tool outputs against the task
- Detect missing or incomplete results
- Retry missing tool calls **once**
- Produce final user-ready output

---

4. Integrated Third-Party APIs

4.1 Groq LLM API

- Used for Planner agent
- Model: llama-3.3-70b-versatile
- Provides reasoning and structured output

4.2 OpenWeatherMap API

- Fetches real-time weather data
- Endpoint: /data/2.5/weather
- Used by weather.current_by_city tool

4.3 GitHub Search API

- Searches public GitHub repositories
- Used by github.search_repos tool
- Token optional (avoids rate limits)

---

5. Project Structure

```
trulymadly-genai-assignment/
|
├── agents/
|   ├── planner.py
|   ├── executor.py
|   ├── verifier.py
|   ├── schemas.py
|
├── tools/
|   ├── github_tool.py
|   ├── weather_tool.py
|   ├── tool_registry.py
|
├── llm/
|   ├── client.py
|   ├── prompts.py
|
├── main.py
├── requirements.txt
├── .env.example
```

├── README.md

---

6. Environment Configuration

Required Environment Variables

.env (not committed):

OPENAI_API_KEY=gsk_XXXXXXXX

OPENAI_BASE_URL=https://api.groq.com/openai/v1

OPENAI_MODEL=llama-3.3-70b-versatile

OPENWEATHER_API_KEY=XXXXXXXX

GITHUB_TOKEN=XXXXXXXX

.env.example (committed):

OPENAI_API_KEY=

OPENAI_BASE_URL=https://api.groq.com/openai/v1

OPENAI_MODEL=llama-3.3-70b-versatile

OPENWEATHER_API_KEY=

GITHUB_TOKEN=

---

7. Setup Instructions (Localhost)

7.1 Prerequisites

- Python 3.10+
- Internet connection

7.2 Installation

python -m venv .venv

.venv\Scripts\activate # Windows

pip install -r requirements.txt

---

8. Running the Project

The project runs locally using **one command**, as required:

uvicorn main:app --reload

Open in browser:

http://localhost:8000

---

9. Example Prompts for Evaluation

1. **Weather Query**

What's the weather in Mumbai right now?

2. **GitHub Repository Search**

Find top 5 GitHub repos for "fastapi jwt auth" and summarize best choice.

3. **Multi-Tool Query**

Find top 5 repos for "react markdown editor" and also check weather in Bangalore.

4. **Another Weather Query**

What's the current weather in Delhi?

---

10. Sample Successful Output

{

"task": "What's the weather in Mumbai right now?",

"plan": {...},

"tool_outputs": {...},

"final_answer": {

"city": "Mumbai",

"weather": "smoke",

```
  "temperature": 30.99,
  "humidity": 40
 },
 "verifier_notes": [
  "Current weather in Mumbai is provided as requested."
 ]
}
```


---

11. Known Limitations & Trade-offs

- Single-turn execution (no conversational memory)
- Verifier retries only once (prevents infinite loops)
- No user authentication (out of scope)
- Minimal UI (focus on backend GenAI logic)

---

12. Security Considerations

- .env is excluded from GitHub via .gitignore
- No API keys are hardcoded
- Push protection enabled by GitHub
- .env.example contains placeholders only

---

13. Assignment Requirements Checklist

Requirement	Status
Multi-agent design	✅
Structured LLM output	✅
≥2 real APIs	✅

Requirement	Status
End-to-end execution	✅
One-command local run	✅
No hardcoded responses	✅

---

14. Conclusion

This project demonstrates a complete **GenAI multi-agent system** with real-world API integrations, structured reasoning, fault tolerance, and evaluator-friendly execution.

The system fulfills all requirements outlined by TrulyMadly and is ready for technical evaluation.

GitHub Repository:
https://github.com/Manas1024/Trulymadly-genai-assignment