Coding Assignment

Hi,
As part of the AI Engineering Internship process, we have a simple but meaningful backend assignment for you. The goal is to understand your fundamentals in Python, API design, async programming, and how you structure logic. This is **pure backend** — no frontend required and no machine learning or RL.

Please attempt this only if you are comfortable writing clean Python and enjoy building systems.

Overview of the Assignment

You need to build a **small agent workflow engine**, something like a very simplified version of LangGraph. The idea is to create a system where we can define a sequence of steps (nodes), connect them, maintain a shared state, and run the workflow end-to-end via APIs.

This is not meant to be complicated. Even a small working version is totally fine — focus on correctness, clarity, and structure.

What You Need to Build

1. A Minimal Workflow / Graph Engine

This is the core part. It should support:

- **Nodes**: Each node is just a Python function that reads and modifies a shared state.
- **State**: A dictionary or Pydantic model that flows from one node to another.
- **Edges**: Define which node runs after which. A simple mapping like {"extract": "analyze"} is enough.
- **Branching**: Basic conditional routing is enough (e.g., if some value in the state is above a threshold, go to a different node).
- **Looping**: Support a simple loop (for example run a node repeatedly until a condition is met).

You do *not* need a fully dynamic graph language. A clean Python structure works.

2. Tool Registry (Simple Version)

Just maintain a dictionary of “tools” (Python functions) that nodes can call.

Example:

```
def detect_smells(code):
  return {"issues": 3}
```

You can pre-register tools or allow registration via an API — both are okay.

3. FastAPI Endpoints

Expose these via FastAPI:

- POST /graph/create
  Input: JSON describing nodes and edges.
  Output: graph_id.
- POST /graph/run
  Input: graph_id + initial state.
  Output: final state + a simple execution log.
- GET /graph/state/{run_id}
  Return the current state of an ongoing workflow.

You can choose to store graphs and runs in memory or in a small SQLite/Postgres database.

Optional (nice to have, not required):

- A WebSocket endpoint to stream logs step-by-step.
- Async execution of long-running steps.

Sample Workflow You Must Implement

To show that your engine works, implement **one example workflow**. Pick any one of the following (they are intentionally simple):

Option A: Code Review Mini-Agent

1. Extract functions
2. Check complexity
3. Detect basic issues

4. Suggest improvements
5. Loop until “quality_score >= threshold”

Option B: Summarization + Refinement

1. Split text into chunks
2. Generate summaries
3. Merge summaries
4. Refine final summary
5. Stop when summary length under a limit

Option C: Data Quality Pipeline

1. Profile data
2. Identify anomalies
3. Generate rules
4. Apply rules
5. Loop until anomaly count is small

These can be **fully rule-based**. No ML is expected.

What to Submit

Your GitHub repo should contain:

- The FastAPI project (/app folder is ideal)
- Code for the graph engine
- Code for your example agent workflow
- A short README:
    - How to run
    - What your workflow engine supports
    - What you would improve with more time

Clean structure matters more than features.

How You Will Be Evaluated

We mainly look for:

- How well you structure your Python code
- Clarity of the graph/engine logic
- Clean and easy-to-understand APIs
- Ability to think in terms of state → transitions → loops
- Basic async/code hygiene

Optional extras (background tasks, logging, WebSockets, etc.) will help you stand out, but they're not required.

If anything is unclear, feel free to ask. I encourage you to keep it simple but well-designed — that is what we look for.