Code Review Workflow Graph

Name: Shaik Rukhiya Masthani
Reg.No: RA2211026020184
Degree: B.Tech
Course: Computer Science and Engineering with specialization in Artificial Intelligence and Machine Learning
CGPA - 9.07

Step 1: Creating the Graph

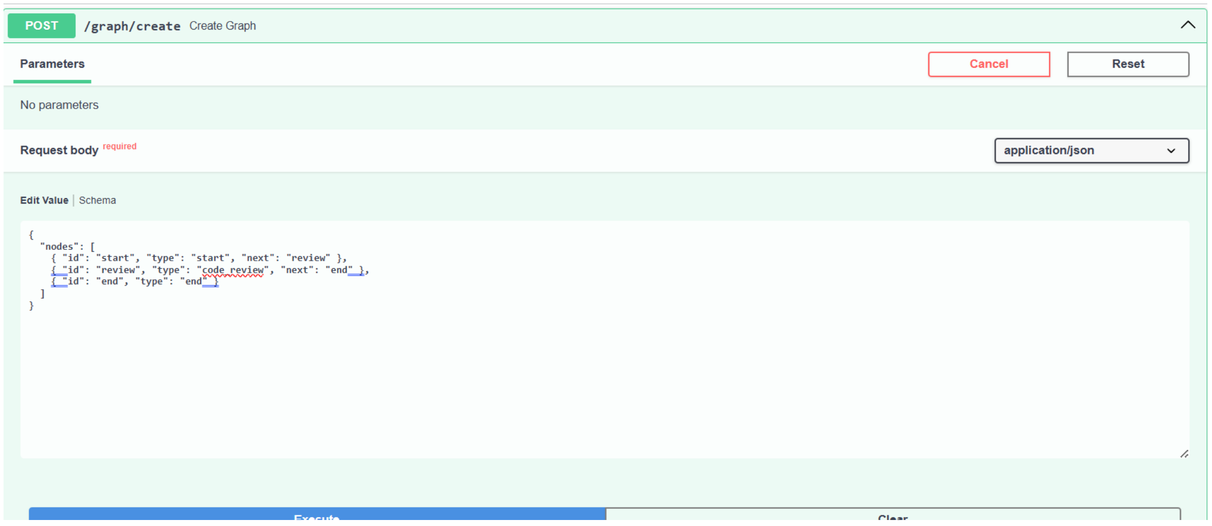
For the first step, I created a workflow graph using the FastAPI endpoint POST /graph/create. The graph defines the sequence of tasks in a simple code review workflow.

I defined three nodes in the JSON request:

- start – The starting point of the workflow.
- review – Represents the code review step.
- end – Marks the completion of the workflow.

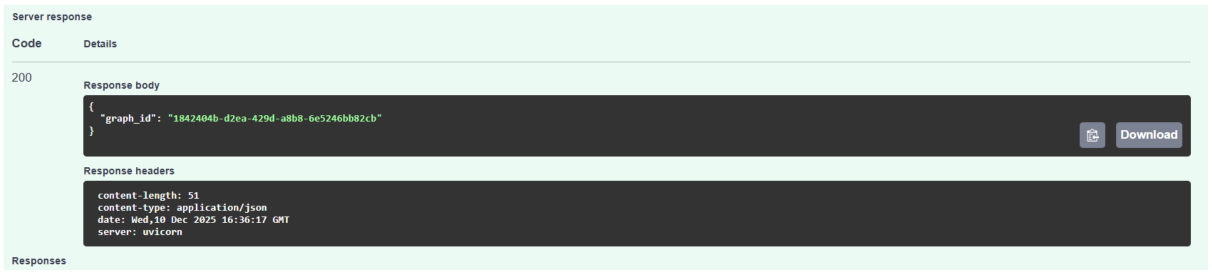
Each node has a type and a next field to indicate the next step in the workflow.

Request JSON used:



```
{
  "nodes": [
    { "id": "start", "type": "start", "next": "review" },
    { "id": "review", "type": "code_review", "next": "end" },
    { "id": "end", "type": "end" }
  ]
}
```

After executing the request, the server returned a graph_id to uniquely identify this workflow:



```
{
  "graph_id": "1842404b-d2ea-429d-a8b8-6e5246bb82cb"
}
```

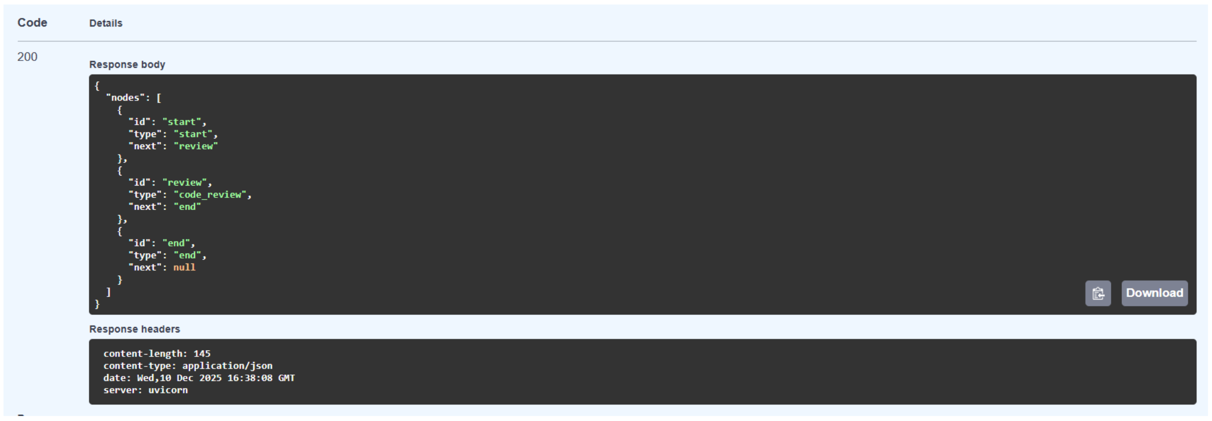
```
content-length: 51
content-type: application/json
date: Wed,10 Dec 2025 16:36:17 GMT
server: uvicorn
```

This confirmed that the workflow graph was successfully created on the server.

Step 2: Retrieving the Graph

Next, I used the **GET /graph/{graph_id}** endpoint to retrieve the graph I just created. I provided the graph_id obtained in Step 1.

The server returned all the nodes with their types and connections, verifying that the graph was stored correctly.


Code Details

200

Response body

```
{
  "nodes": [
    {
      "id": "start",
      "type": "start",
      "next": "review"
    },
    {
      "id": "review",
      "type": "code_review",
      "next": "end"
    },
    {
      "id": "end",
      "type": "end",
      "next": null
    }
  ]
}
```

Response headers

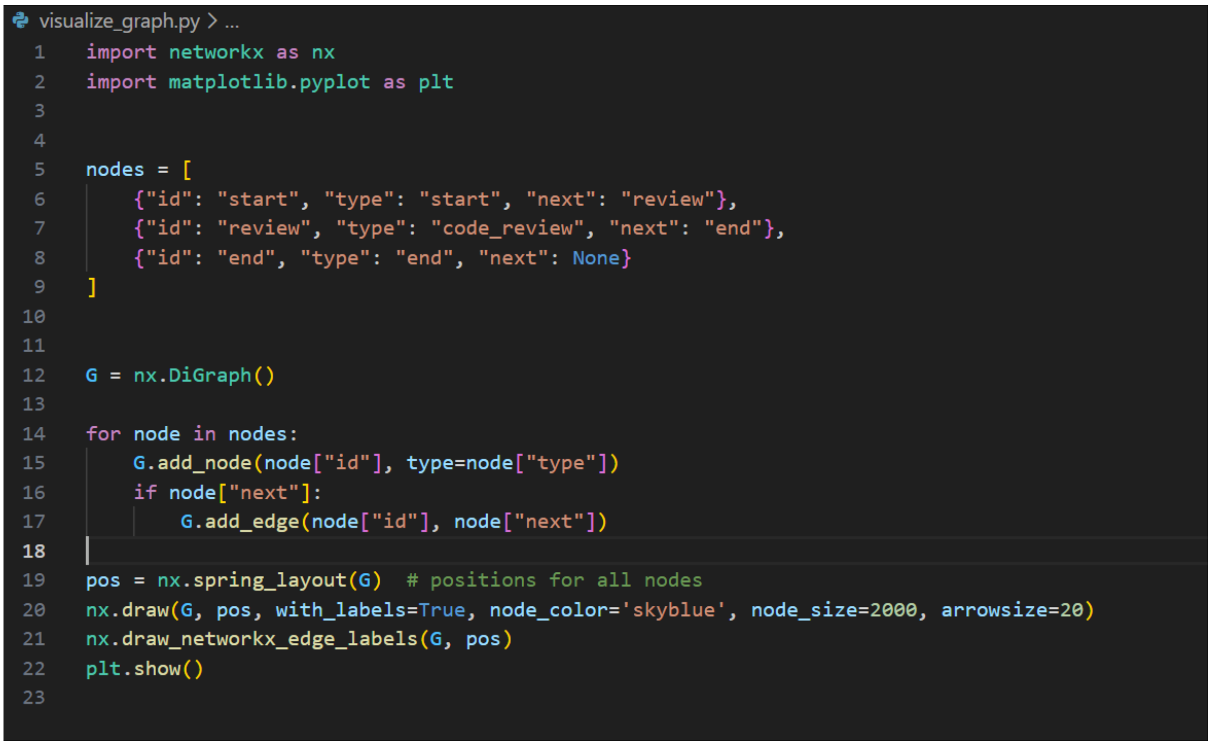
```
content-length: 145
content-type: application/json
date: Wed,10 Dec 2025 16:38:08 GMT
server: uvicorn
```

This step ensures that the workflow graph exists and can be retrieved at any time using the graph ID.

Step 3: Visualizing the Graph

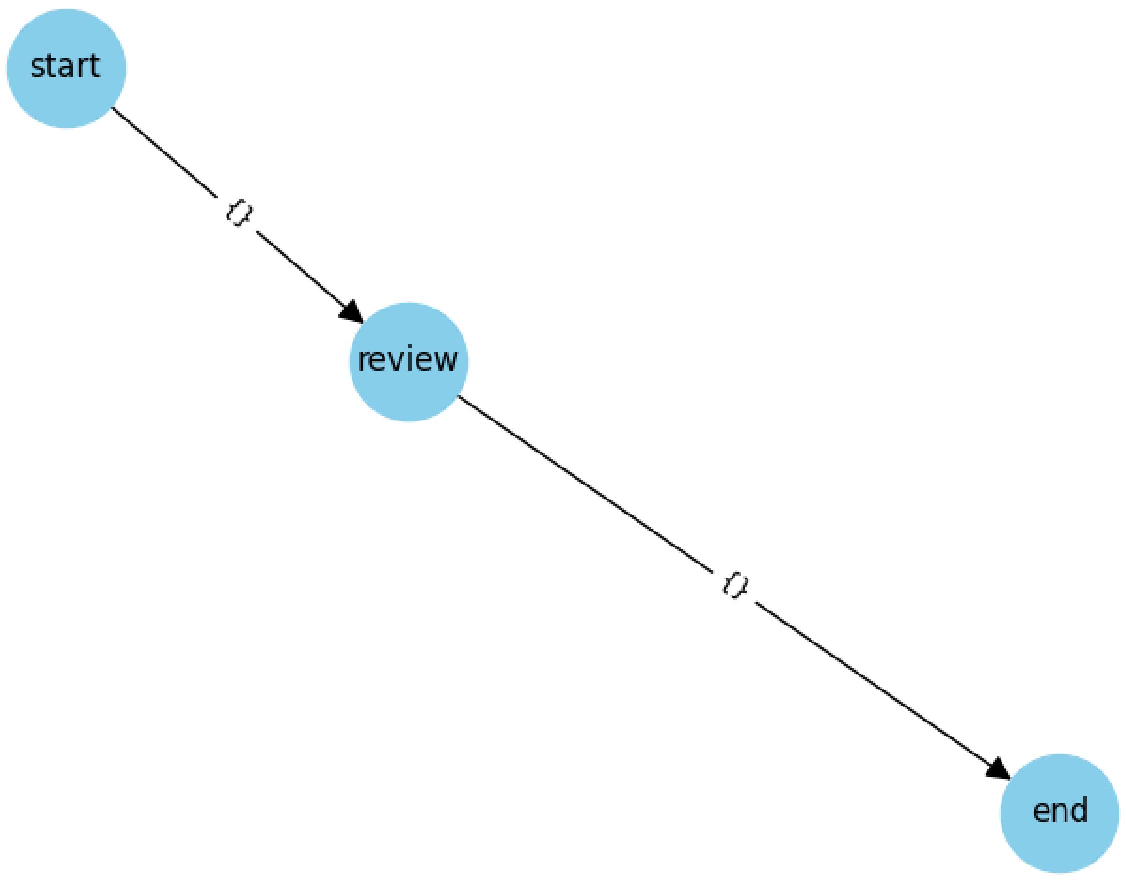
Finally, I visualized the workflow graph using **Python** with the networkx and matplotlib libraries.

- I created a directed graph where each node represents a step in the workflow.
- Arrows indicate the flow from one node to the next (next field).
- This allows a clear visual understanding of the workflow sequence.


visualize_graph.py > ...

```python
import networkx as nx
import matplotlib.pyplot as plt


nodes = [
    {"id": "start", "type": "start", "next": "review"},
    {"id": "review", "type": "code_review", "next": "end"},
    {"id": "end", "type": "end", "next": None}
]


G = nx.DiGraph()

for node in nodes:
    G.add_node(node["id"], type=node["type"])
    if node["next"]:
        G.add_edge(node["id"], node["next"])

pos = nx.spring_layout(G)  # positions for all nodes
nx.draw(G, pos, with_labels=True, node_color='skyblue', node_size=2000, arrowsize=20)
nx.draw_networkx_edge_labels(G, pos)
plt.show()

```

This confirmed that the workflow starts at start, goes to review, and ends at end.



Conclusion

All steps of the assignment were completed successfully:

1. Created a workflow graph with nodes and connections.
2. Retrieved the graph using its unique ID to verify correctness.
3. Visualized the graph in Python for better understanding of workflow flow.

The workflow graph has been tested and verified to match the intended sequence.