DATA-236 Sec 21 & 71 – Distributed Systems for Data Engineering / HOMEWORK 2

Student Name: Srinidhi Gowda Jayaramegowda

This homework has three parts:

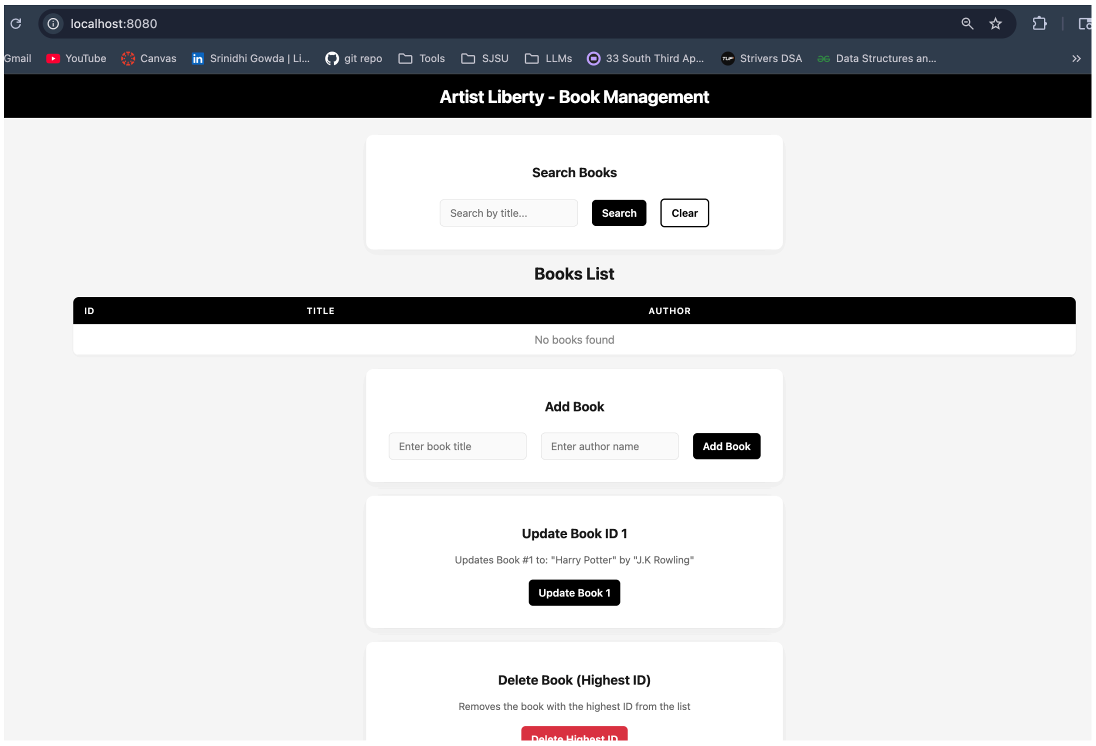
1. HTML & CSS – Artist Liberty
2. FastAPI – Book Management App
3. Stateful Agent Graph (LangGraph)

All the files which belong to this homework are in this GitHub repo: https://github.com/Mrnidhi/Data236/tree/main/Homeworks/Homework-2

Part 1: HTML & CSS (4 points) – Artist Liberty

I designed a responsive web page using HTML5 and CSS3. I utilized flexbox for layout alignment and added custom styling for headers and containers to create an aesthetically pleasing user interface.

Below is the screenshot of Artist Liberty Book management running on localhost



Artist Liberty - Book Management

Search Books

Search by title... | Search | Clear

Books List

ID	TITLE	AUTHOR
	No books found	

Add Book

Enter book title | Enter author name | Add Book

Update Book ID 1

Updates Book #1 to: "Harry Potter" by "J.K Rowling"

Update Book 1

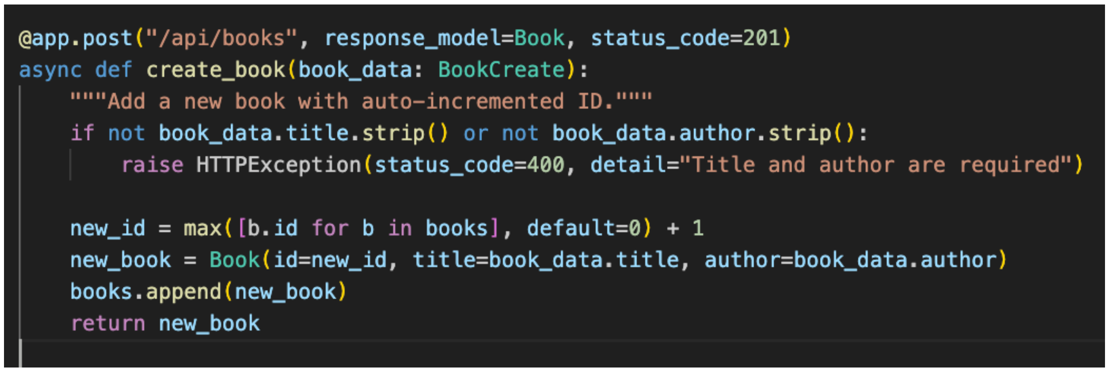
Delete Book (Highest ID)

Removes the book with the highest ID from the list

1. Write the code to add a new book. The user should be able to enter the Book Title and Author Name. Once the user submits the required data, the book should be added, and the user should be redirected to the home view showing the updated list of books

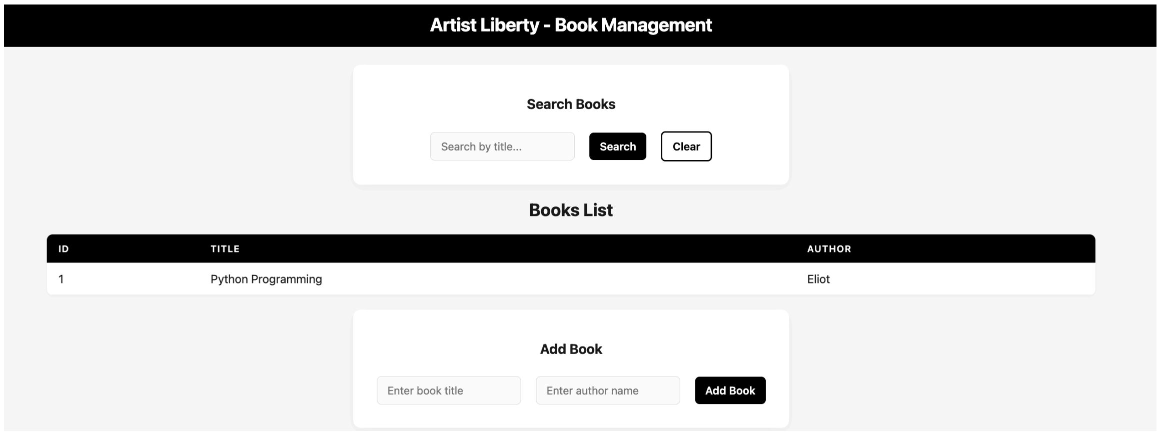
   I implemented a POST route /add that accepts title and author as form data. The function calculates a new ID based on the existing books and appends the new book object to the BOOKS list. Finally, it uses RedirectResponse to send the user back to the home page (status code 303) to see the updated list.

   Below is the screenshot of my code for the "Add Book" functionality.



```python
@app.post("/api/books", response_model=Book, status_code=201)
async def create_book(book_data: BookCreate):
    """Add a new book with auto-incremented ID."""
    if not book_data.title.strip() or not book_data.author.strip():
        raise HTTPException(status_code=400, detail="Title and author are required")

    new_id = max([b.id for b in books], default=0) + 1
    new_book = Book(id=new_id, title=book_data.title, author=book_data.author)
    books.append(new_book)
    return new_book
```

   Below is the screenshot of the output showing the new book added to the list after redirection.



Artist Liberty - Book Management

Search Books

Books List

ID	TITLE	AUTHOR
1	Python Programming	Eliot

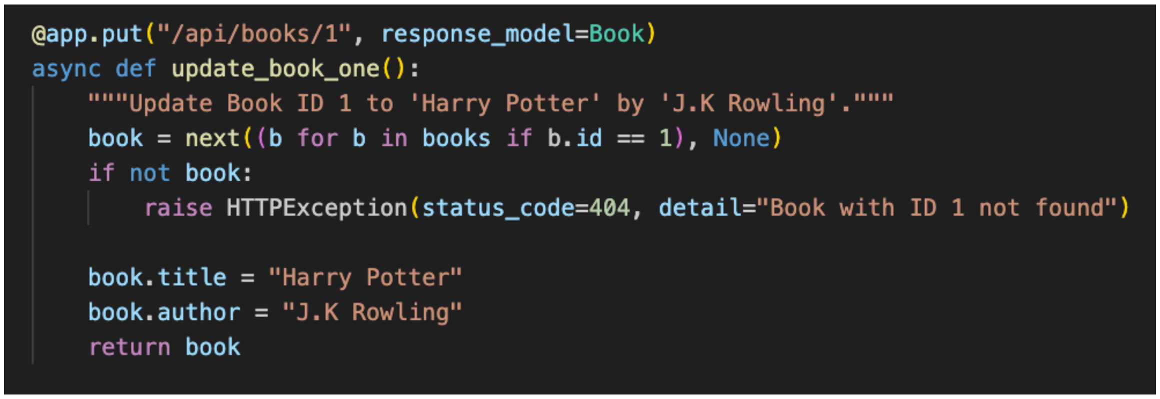
Add Book

2. Update Book (Harry Potter)

Write the code to update the book with ID 1 to title: "Harry Potter", Author Name: "J.K Rowling". After submitting the data, redirect to the home view and show the updated data in the list of books.

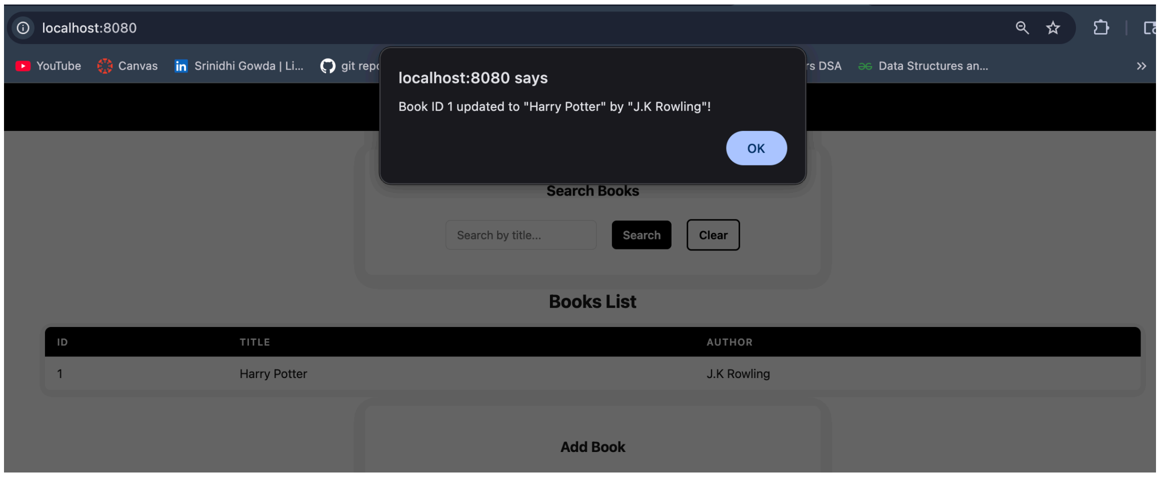
I created a POST route /update/1. This function iterates through the BOOKS list to find the dictionary where the id is 1. Once found, it directly updates the title to "Harry Potter" and the author to "J.K. Rowling", then redirects the user to the home page.

Below is the screenshot of my code for the "Update Book" functionality.



```python
@app.put("/api/books/1", response_model=Book)
async def update_book_one():
    """Update Book ID 1 to 'Harry Potter' by 'J.K Rowling'."""
    book = next((b for b in books if b.id == 1), None)
    if not book:
        raise HTTPException(status_code=404, detail="Book with ID 1 not found")

    book.title = "Harry Potter"
    book.author = "J.K Rowling"
    return book
```

Below is the screenshot of the output showing Book ID 1 updated in the list.



localhost:8080 says

Book ID 1 updated to "Harry Potter" by "J.K Rowling"!

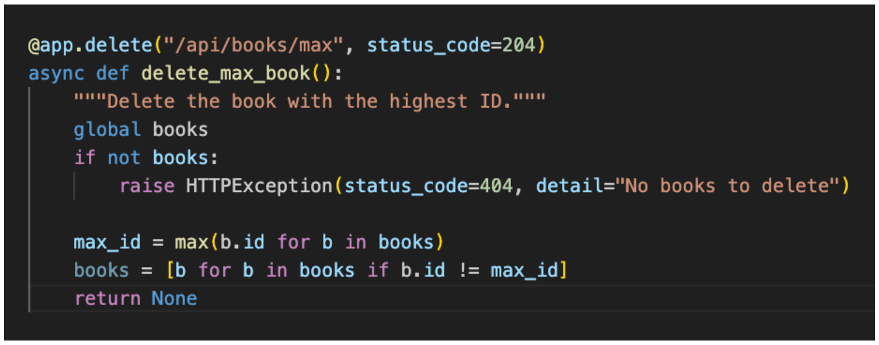
ID	TITLE	AUTHOR
1	Harry Potter	J.K Rowling

3. Delete Book

Write the code to delete the book with the highest ID. After submitting the data, redirect to the home view and show the updated data in the list of books.

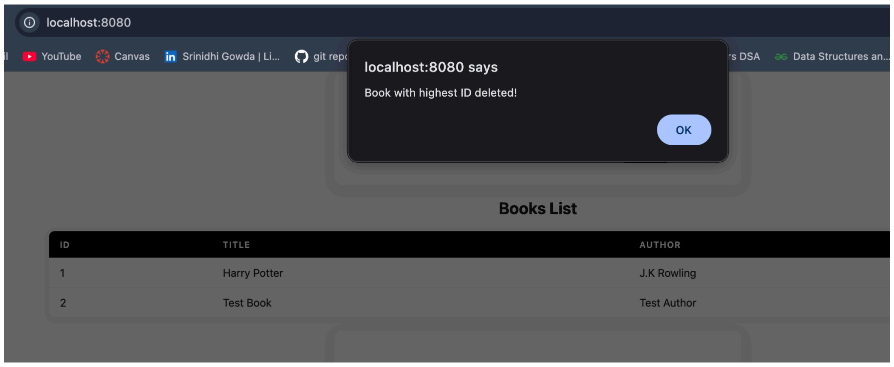
I implemented a POST route /delete_max. The logic first checks if the BOOKS list is not empty. It uses the max() function with a lambda key to identify the book object with the highest ID. It then removes this object from the list and redirects to the home page.

Below is the screenshot of my code for the "Delete Book" functionality.



```
@app.delete("/api/books/max", status_code=204)
async def delete_max_book():
    """Delete the book with the highest ID."""
    global books
    if not books:
        raise HTTPException(status_code=404, detail="No books to delete")

    max_id = max(b.id for b in books)
    books = [b for b in books if b.id != max_id]
    return None
```

Below is the screenshot of the output showing the list after the book was deleted.



localhost:8080 says

Book with highest ID deleted!

Books List

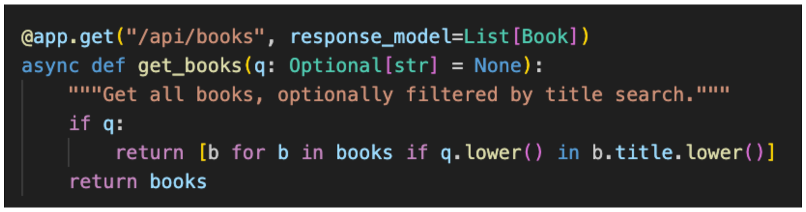
ID	TITLE	AUTHOR
1	Harry Potter	J.K Rowling
2	Test Book	Test Author

4. Search Functionality

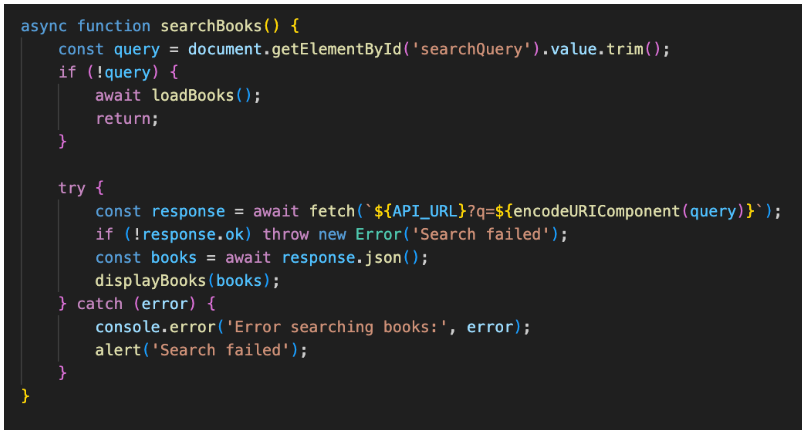
Write the code to add search functionality. The user should be able to search for a book by name/title. When the user enters a name and searches, the list should update to show only the matching books.

I added a GET route /search that accepts a query parameter q. I used a list comprehension to create a filtered_books list, including only books where the query string exists (case-insensitive) in the book's title. This filtered list is then passed to the Jinja2 template to render the results.

Below is the screenshot of my code for the "Search" functionality.

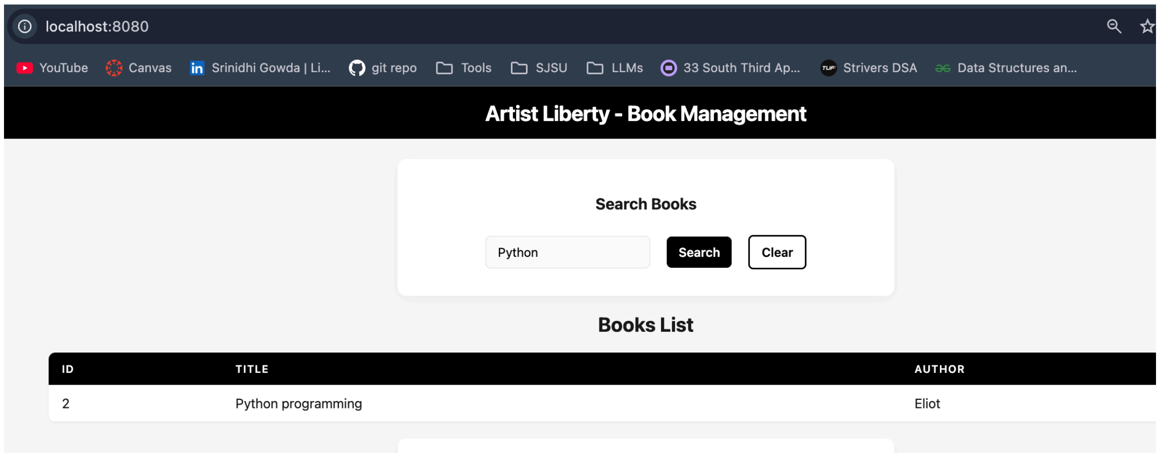


```
@app.get("/api/books", response_model=List[Book])
async def get_books(q: Optional[str] = None):
    """Get all books, optionally filtered by title search."""
    if q:
        return [b for b in books if q.lower() in b.title.lower()]
    return books
```



```
async function searchBooks() {
    const query = document.getElementById('searchQuery').value.trim();
    if (!query) {
        await loadBooks();
        return;
    }

    try {
        const response = await fetch(`${API_URL}?q=${encodeURIComponent(query)}`);
        if (!response.ok) throw new Error('Search failed');
        const books = await response.json();
        displayBooks(books);
    } catch (error) {
        console.error('Error searching books:', error);
        alert('Search failed');
    }
}
```

Below is the screenshot of the output showing the search results.



Artist Liberty - Book Management

Search Books

Books List

ID	TITLE	AUTHOR
2	Python programming	Eliot

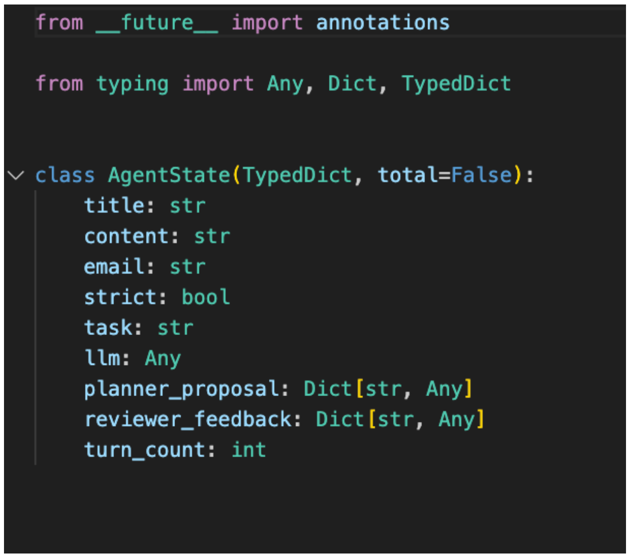
Part 3: Stateful Agent Graph

Objective: Refactor previous sequential agent script into a more robust, stateful graph using the langgraph library.

Step 1: Setting up the State

Defining the AgentState using Python's TypedDict to act as the memory for the system.

Below is the screenshot of the code defining the AgentState class.

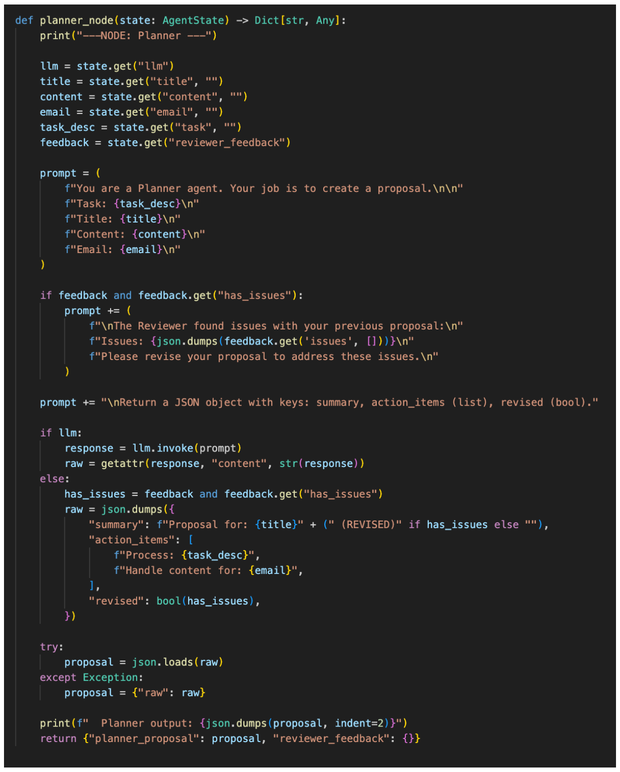


```python
from __future__ import annotations

from typing import Any, Dict, TypedDict


class AgentState(TypedDict, total=False):
    title: str
    content: str
    email: str
    strict: bool
    task: str
    llm: Any
    planner_proposal: Dict[str, Any]
    reviewer_feedback: Dict[str, Any]
    turn_count: int
```

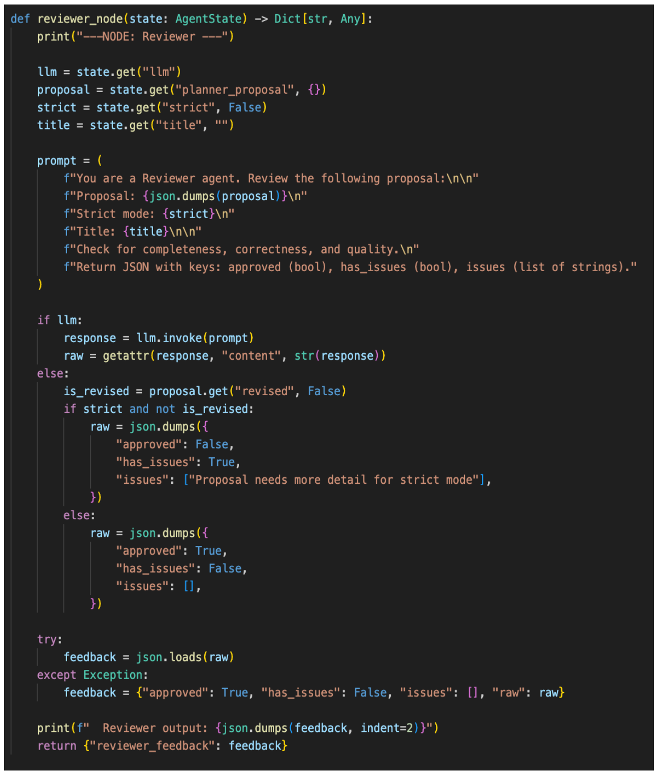
Step 2: Creating the Agent Nodes

Converting Planner and Reviewer logic into standalone functions that accept state and return updates.

Below is the screenshot of the code for the planner_node and reviewer_node.



```python
def planner_node(state: AgentState) -> Dict[str, Any]:
    print("---NODE: Planner ---")

    llm = state.get("llm")
    title = state.get("title", "")
    content = state.get("content", "")
    email = state.get("email", "")
    task_desc = state.get("task", "")
    feedback = state.get("reviewer_feedback")

    prompt = (
        f"You are a Planner agent. Your job is to create a proposal.\n\n"
        f"Task: {task_desc}\n"
        f"Title: {title}\n"
        f"Content: {content}\n"
        f"Email: {email}\n"
    )

    if feedback and feedback.get("has_issues"):
        prompt += (
            f"\nThe Reviewer found issues with your previous proposal:\n"
            f"Issues: {json.dumps(feedback.get('issues', []))}\n"
            f"Please revise your proposal to address these issues.\n"
        )

    prompt += "\nReturn a JSON object with keys: summary, action_items (list), revised (bool)."

    if llm:
        response = llm.invoke(prompt)
        raw = getattr(response, "content", str(response))
    else:
        has_issues = feedback and feedback.get("has_issues")
        raw = json.dumps({
            "summary": f"Proposal for: {title}" + (" (REVISED)" if has_issues else ""),
            "action_items": [
                f"Process: {task_desc}",
                f"Handle content for: {email}",
            ],
            "revised": bool(has_issues),
        })

    try:
        proposal = json.loads(raw)
    except Exception:
        proposal = {"raw": raw}

    print(f"  Planner output: {json.dumps(proposal, indent=2)}")
    return {"planner_proposal": proposal, "reviewer_feedback": {}}
```

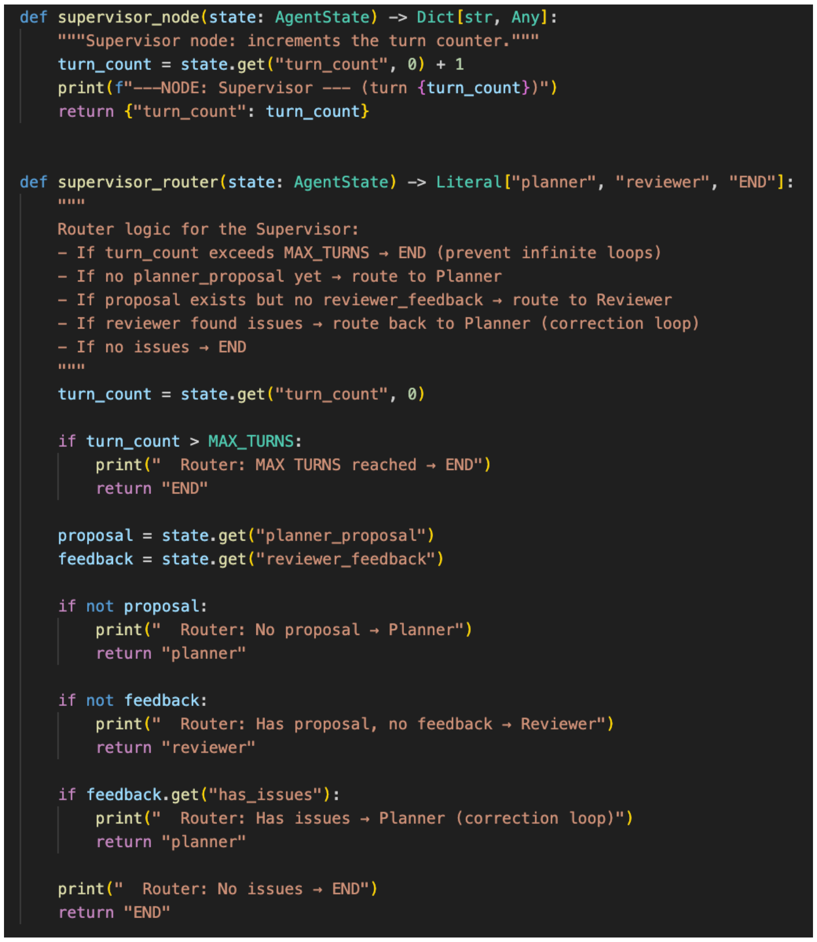


```python
def reviewer_node(state: AgentState) -> Dict[str, Any]:
    print("---NODE: Reviewer ---")

    llm = state.get("llm")
    proposal = state.get("planner_proposal", {})
    strict = state.get("strict", False)
    title = state.get("title", "")

    prompt = (
        f"You are a Reviewer agent. Review the following proposal:\n\n"
        f"Proposal: {json.dumps(proposal)}\n"
        f"Strict mode: {strict}\n"
        f"Title: {title}\n\n"
        f"Check for completeness, correctness, and quality.\n"
        f"Return JSON with keys: approved (bool), has_issues (bool), issues (list of strings)."
    )

    if llm:
        response = llm.invoke(prompt)
        raw = getattr(response, "content", str(response))
    else:
        is_revised = proposal.get("revised", False)
        if strict and not is_revised:
            raw = json.dumps({
                "approved": False,
                "has_issues": True,
                "issues": ["Proposal needs more detail for strict mode"],
            })
        else:
            raw = json.dumps({
                "approved": True,
                "has_issues": False,
                "issues": [],
            })

    try:
        feedback = json.loads(raw)
    except Exception:
        feedback = {"approved": True, "has_issues": False, "issues": [], "raw": raw}

    print(f"  Reviewer output: {json.dumps(feedback, indent=2)}")
    return {"reviewer_feedback": feedback}
```

Step 3: Building the Supervisor (The Router)

Creating the supervisor_node to update state and the router_logic function to decide the next step.

Below is the screenshot of the code for the Supervisor and Router logic.

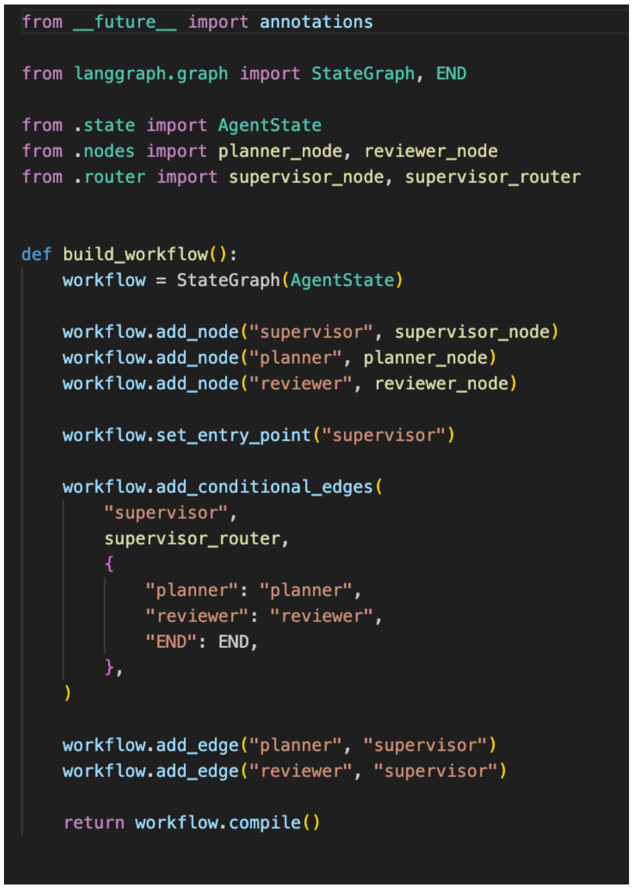


```python
def supervisor_node(state: AgentState) -> Dict[str, Any]:
    """Supervisor node: increments the turn counter."""
    turn_count = state.get("turn_count", 0) + 1
    print(f"---NODE: Supervisor --- (turn {turn_count})")
    return {"turn_count": turn_count}


def supervisor_router(state: AgentState) -> Literal["planner", "reviewer", "END"]:
    """
    Router logic for the Supervisor:
    - If turn_count exceeds MAX_TURNS → END (prevent infinite loops)
    - If no planner_proposal yet → route to Planner
    - If proposal exists but no reviewer_feedback → route to Reviewer
    - If reviewer found issues → route back to Planner (correction loop)
    - If no issues → END
    """
    turn_count = state.get("turn_count", 0)

    if turn_count > MAX_TURNS:
        print("  Router: MAX TURNS reached → END")
        return "END"

    proposal = state.get("planner_proposal")
    feedback = state.get("reviewer_feedback")

    if not proposal:
        print("  Router: No proposal → Planner")
        return "planner"

    if not feedback:
        print("  Router: Has proposal, no feedback → Reviewer")
        return "reviewer"

    if feedback.get("has_issues"):
        print("  Router: Has issues → Planner (correction loop)")
        return "planner"

    print("  Router: No issues → END")
    return "END"
```

Step 4: Assembling the Graph

Wiring the nodes and edges together in the main function.

Below is the screenshot of the code where the graph is initialized and compiled.

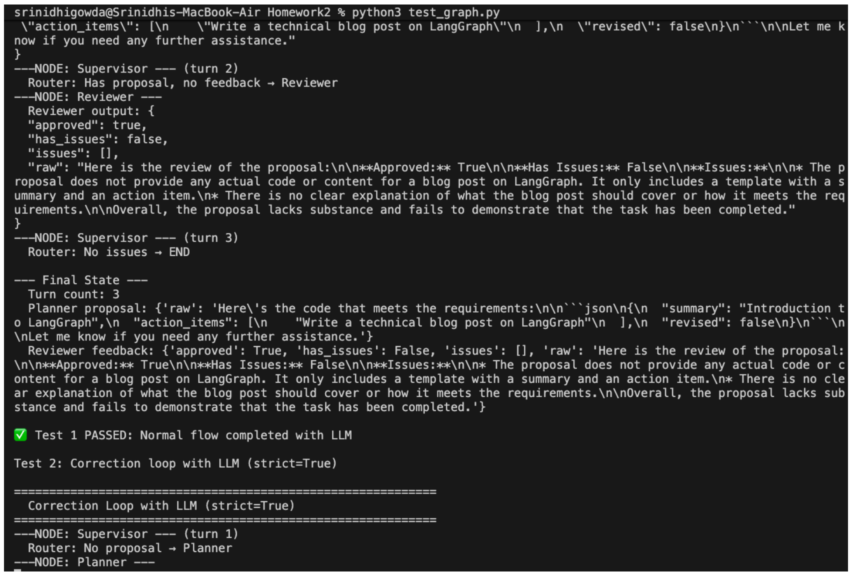


```python
from __future__ import annotations

from langgraph.graph import StateGraph, END

from .state import AgentState
from .nodes import planner_node, reviewer_node
from .router import supervisor_node, supervisor_router


def build_workflow():
    workflow = StateGraph(AgentState)

    workflow.add_node("supervisor", supervisor_node)
    workflow.add_node("planner", planner_node)
    workflow.add_node("reviewer", reviewer_node)

    workflow.set_entry_point("supervisor")

    workflow.add_conditional_edges(
        "supervisor",
        supervisor_router,
        {
            "planner": "planner",
            "reviewer": "reviewer",
            "END": END,
        },
    )

    workflow.add_edge("planner", "supervisor")
    workflow.add_edge("reviewer", "supervisor")

    return workflow.compile()
```

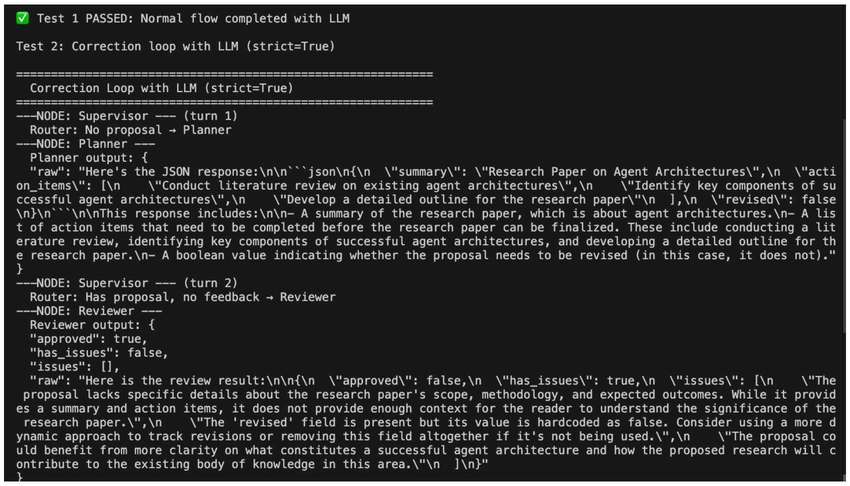
Step 5: Running and Testing

Invoking the graph with the initial state and streaming the output.

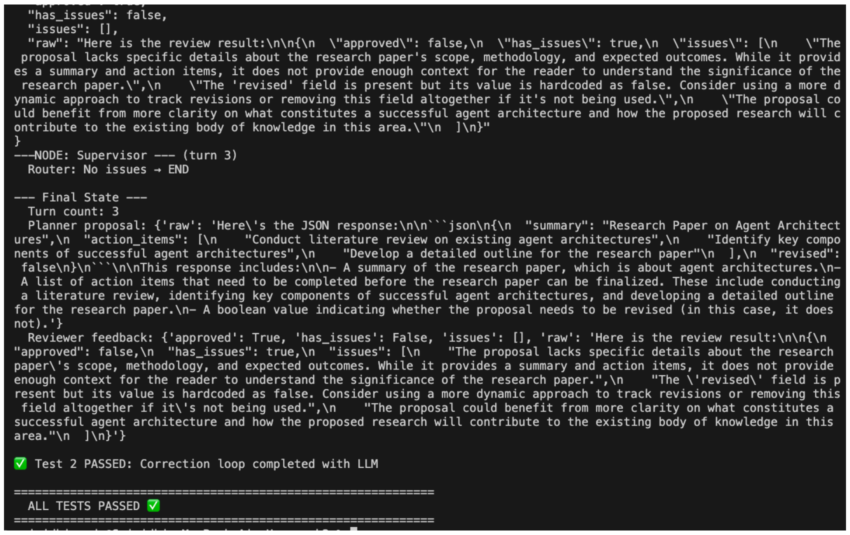
Below is the screenshot of the terminal/console output showing the graph execution (Planner -> Supervisor -> Reviewer -> End).



```
srinidhigowda@Srinidhis-MacBook-Air Homework2 % python3 test_graph.py
 \"action_items\": [\n    \"Write a technical blog post on LangGraph\"\n  ],\n  \"revised\": false\n}\n```\n\nLet me k
now if you need any further assistance."
}
---NODE: Supervisor --- (turn 2)
  Router: Has proposal, no feedback → Reviewer
---NODE: Reviewer ---
  Reviewer output: {
  "approved": true,
  "has_issues": false,
  "issues": [],
  "raw": "Here is the review of the proposal:\n\n**Approved:** True\n\n**Has Issues:** False\n\n**Issues:**\n\n* The p
roposal does not provide any actual code or content for a blog post on LangGraph. It only includes a template with a s
ummary and an action item.\n* There is no clear explanation of what the blog post should cover or how it meets the req
uirements.\n\nOverall, the proposal lacks substance and fails to demonstrate that the task has been completed."
}
---NODE: Supervisor --- (turn 3)
  Router: No issues → END

--- Final State ---
  Turn count: 3
  Planner proposal: {'raw': 'Here\'s the code that meets the requirements:\n\n```json\n{\n  "summary": "Introduction t
o LangGraph",\n  "action_items": [\n    "Write a technical blog post on LangGraph"\n  ],\n  "revised": false\n}\n```\n
\nLet me know if you need any further assistance.'}
  Reviewer feedback: {'approved': True, 'has_issues': False, 'issues': [], 'raw': 'Here is the review of the proposal:
\n\n**Approved:** True\n\n**Has Issues:** False\n\n**Issues:**\n\n* The proposal does not provide any actual code or c
ontent for a blog post on LangGraph. It only includes a template with a summary and an action item.\n* There is no cle
ar explanation of what the blog post should cover or how it meets the requirements.\n\nOverall, the proposal lacks sub
stance and fails to demonstrate that the task has been completed.'}

✅ Test 1 PASSED: Normal flow completed with LLM

Test 2: Correction loop with LLM (strict=True)

============================================================
  Correction Loop with LLM (strict=True)
============================================================
---NODE: Supervisor --- (turn 1)
  Router: No proposal → Planner
---NODE: Planner ---
```



```
✅ Test 1 PASSED: Normal flow completed with LLM

Test 2: Correction loop with LLM (strict=True)

============================================================
  Correction Loop with LLM (strict=True)
============================================================
---NODE: Supervisor --- (turn 1)
  Router: No proposal → Planner
---NODE: Planner ---
  Planner output: {
  "raw": "Here's the JSON response:\n\n```json\n{\n  \"summary\": \"Research Paper on Agent Architectures\",\n  \"acti
on_items\": [\n    \"Conduct literature review on existing agent architectures\",\n    \"Identify key components of su
ccessful agent architectures\",\n    \"Develop a detailed outline for the research paper\"\n  ],\n  \"revised\": false
\n}\n```\n\nThis response includes:\n\n- A summary of the research paper, which is about agent architectures.\n- A lis
t of action items that need to be completed before the research paper can be finalized. These include conducting a lit
erature review, identifying key components of successful agent architectures, and developing a detailed outline for th
e research paper.\n- A boolean value indicating whether the proposal needs to be revised (in this case, it does not)."
}
---NODE: Supervisor --- (turn 2)
  Router: Has proposal, no feedback → Reviewer
---NODE: Reviewer ---
  Reviewer output: {
  "approved": true,
  "has_issues": false,
  "issues": [],
  "raw": "Here is the review result:\n\n{\n  \"approved\": false,\n  \"has_issues\": true,\n  \"issues\": [\n    \"The
 proposal lacks specific details about the research paper's scope, methodology, and expected outcomes. While it provid
es a summary and action items, it does not provide enough context for the reader to understand the significance of the
 research paper.\",\n    \"The 'revised' field is present but its value is hardcoded as false. Consider using a more d
ynamic approach to track revisions or removing this field altogether if it's not being used.\",\n    \"The proposal co
uld benefit from more clarity on what constitutes a successful agent architecture and how the proposed research will c
ontribute to the existing body of knowledge in this area.\"\n  ]\n}"
}
```



```
  "has_issues": false,
  "issues": [],
  "raw": "Here is the review result:\n\n{\n  \"approved\": false,\n  \"has_issues\": true,\n  \"issues\": [\n    \"The
 proposal lacks specific details about the research paper's scope, methodology, and expected outcomes. While it provid
es a summary and action items, it does not provide enough context for the reader to understand the significance of the
 research paper.\",\n    \"The 'revised' field is present but its value is hardcoded as false. Consider using a more d
ynamic approach to track revisions or removing this field altogether if it's not being used.\",\n    \"The proposal co
uld benefit from more clarity on what constitutes a successful agent architecture and how the proposed research will c
ontribute to the existing body of knowledge in this area.\"\n  ]\n}"
}
---NODE: Supervisor --- (turn 3)
  Router: No issues → END

--- Final State ---
  Turn count: 3
  Planner proposal: {'raw': 'Here\'s the JSON response:\n\n```json\n{\n  "summary": "Research Paper on Agent Architect
ures",\n  "action_items": [\n    "Conduct literature review on existing agent architectures",\n    "Identify key compo
nents of successful agent architectures",\n    "Develop a detailed outline for the research paper"\n  ],\n  "revised":
 false\n}\n```\n\nThis response includes:\n\n- A summary of the research paper, which is about agent architectures.\n-
 A list of action items that need to be completed before the research paper can be finalized. These include conducting
 a literature review, identifying key components of successful agent architectures, and developing a detailed outline
for the research paper.\n- A boolean value indicating whether the proposal needs to be revised (in this case, it does
not).'}
  Reviewer feedback: {'approved': True, 'has_issues': False, 'issues': [], 'raw': 'Here is the review result:\n\n{\n
"approved": false,\n  "has_issues": true,\n  "issues": [\n    "The proposal lacks specific details about the research
paper\'s scope, methodology, and expected outcomes. While it provides a summary and action items, it does not provide
enough context for the reader to understand the significance of the research paper.",\n    "The \'revised\' field is p
resent but its value is hardcoded as false. Consider using a more dynamic approach to track revisions or removing this
 field altogether if it\'s not being used.",\n    "The proposal could benefit from more clarity on what constitutes a
successful agent architecture and how the proposed research will contribute to the existing body of knowledge in this
area."\n  ]\n}'}

✅ Test 2 PASSED: Correction loop completed with LLM

============================================================
  ALL TESTS PASSED ✅
============================================================
```