DATA-236 Sec 21 & 71 - Distributed Systems for Data Engineering

HOMEWORK 2

Vimalanandhan Sivanandham

017596436

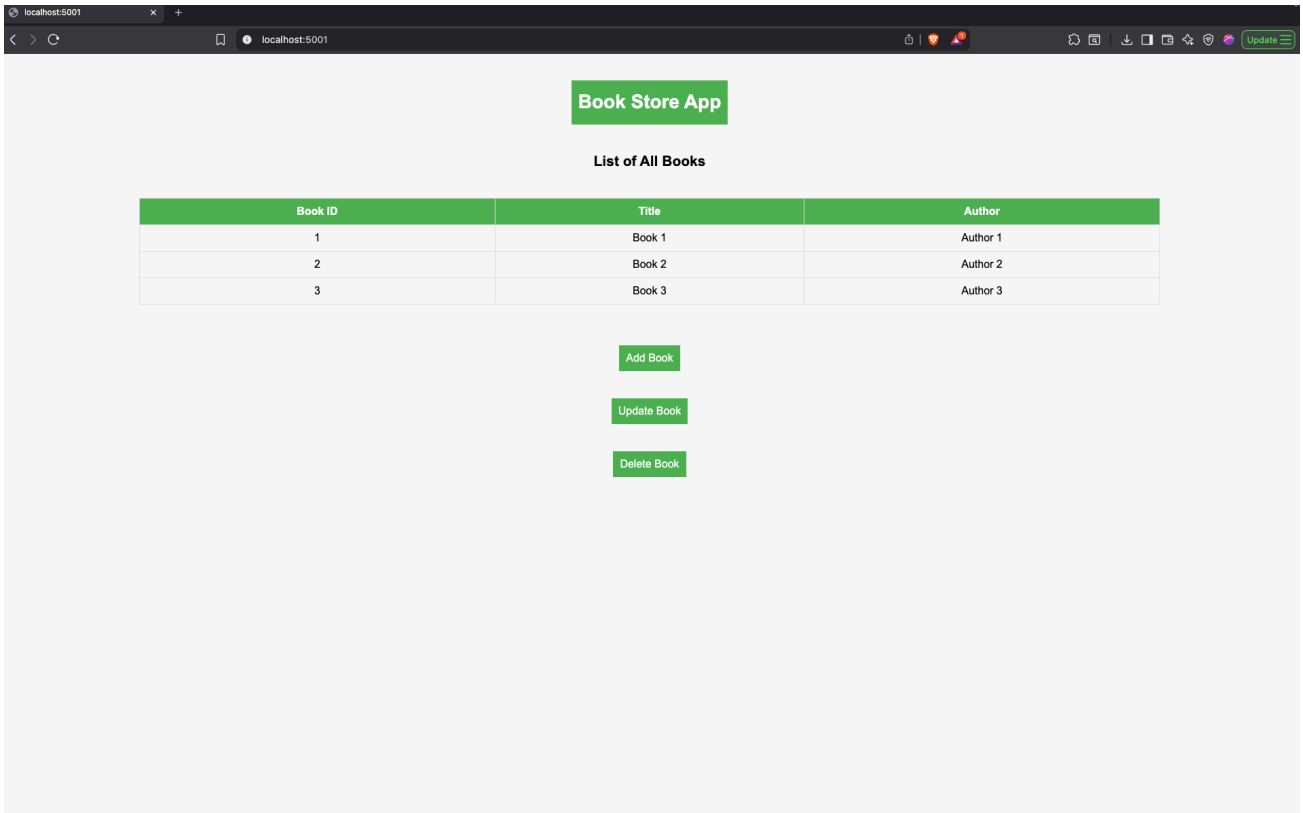
GitHub link for full code artifacts - https://github.com/Vimalanandhan/DATA-236---Distributed-Systems-for-Data-Engineering/tree/main/Assignements/Assignement2

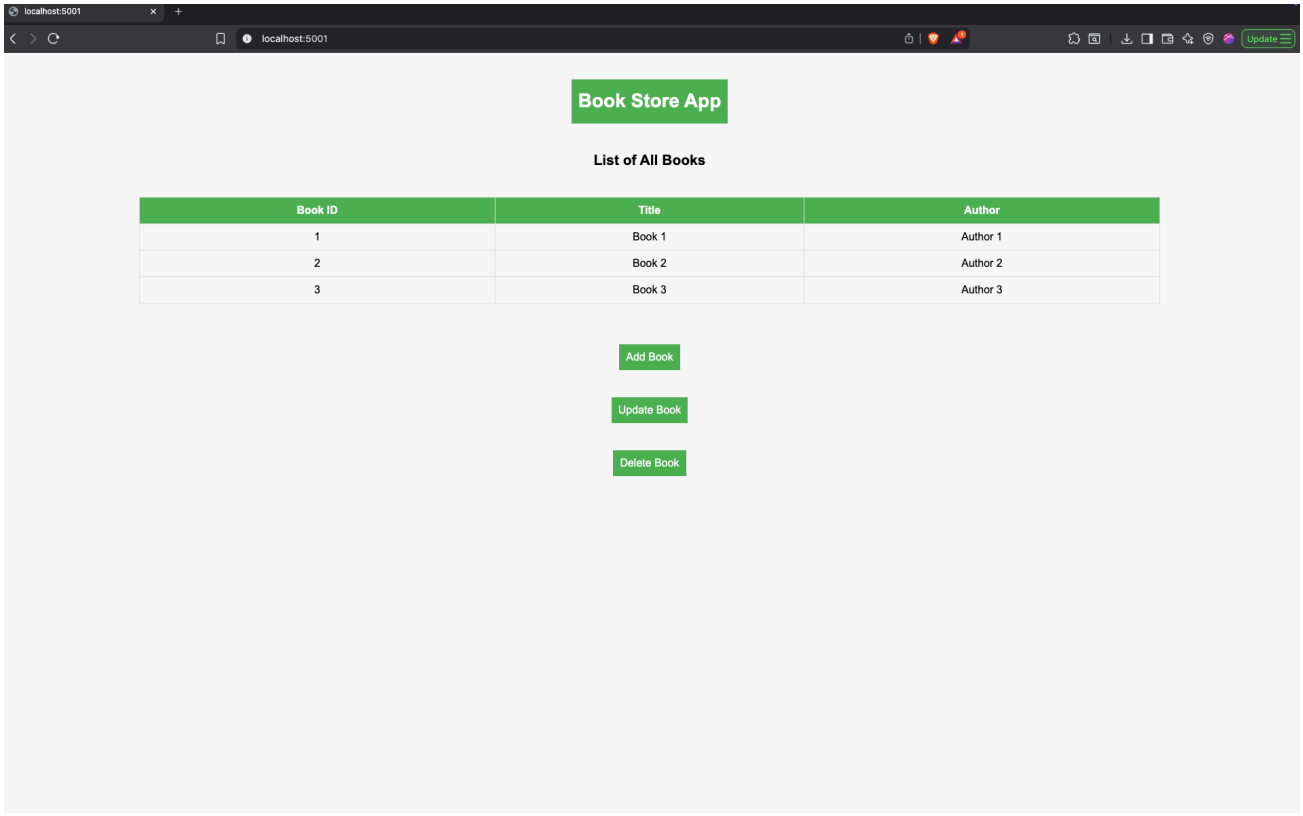
Instructions:

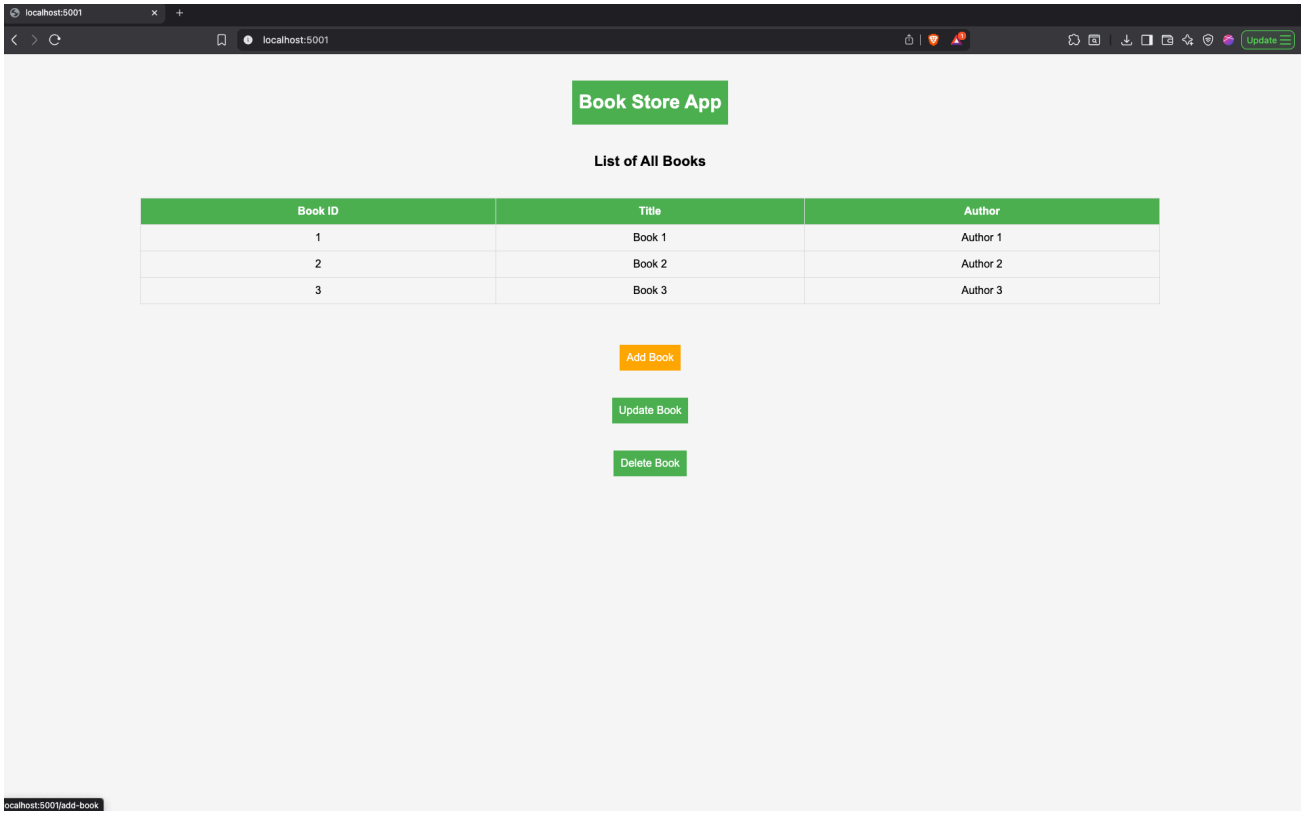
- Please provide the screenshots of the code solution for each question along with its intended output. Ensure that the code and corresponding output screenshots are placed together, one below the other.
- Submission should be in PDF Format.
- Please name your submission file as {last_name}_HW2.pdf

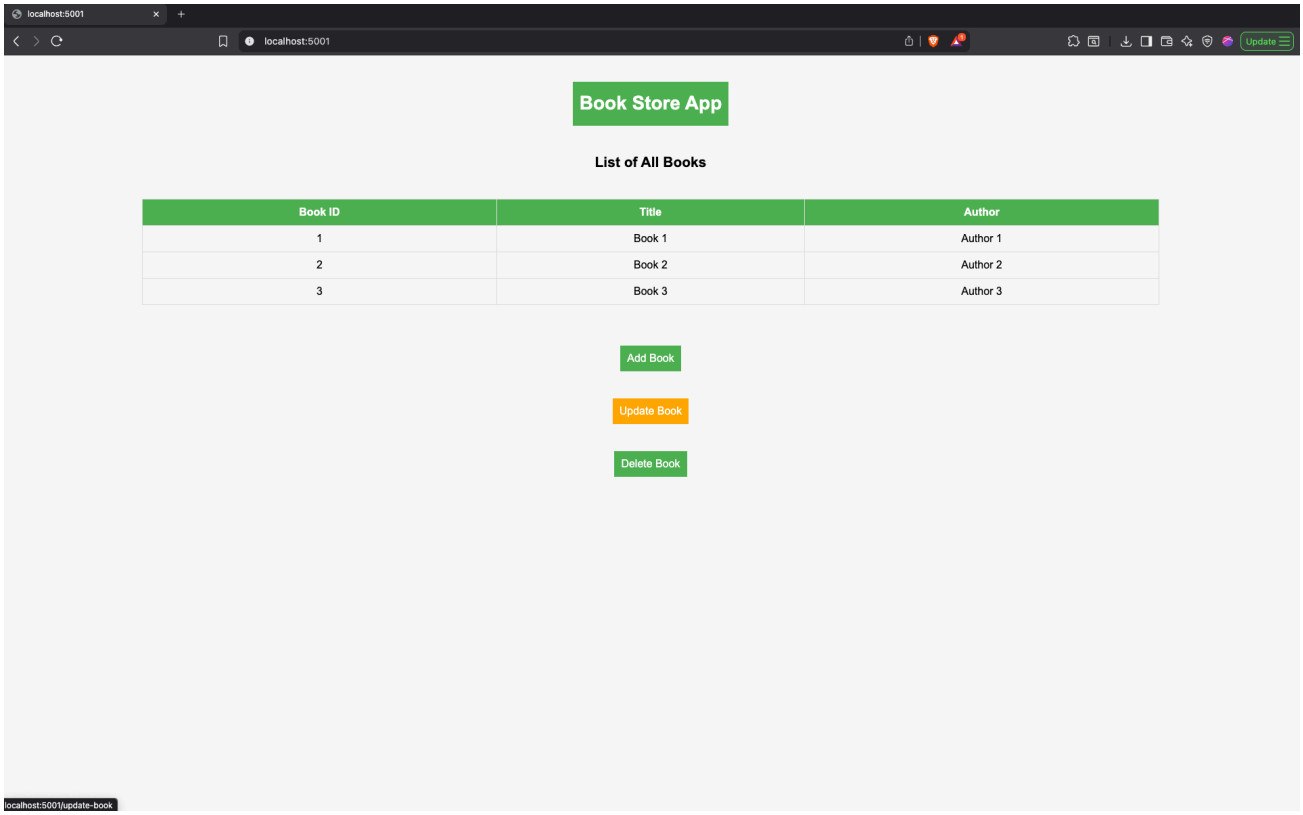
Part 1. HTML & CSS (4 points) - Artist Liberty.
Part 2. HTTP, Express, NodeJS (6 points)

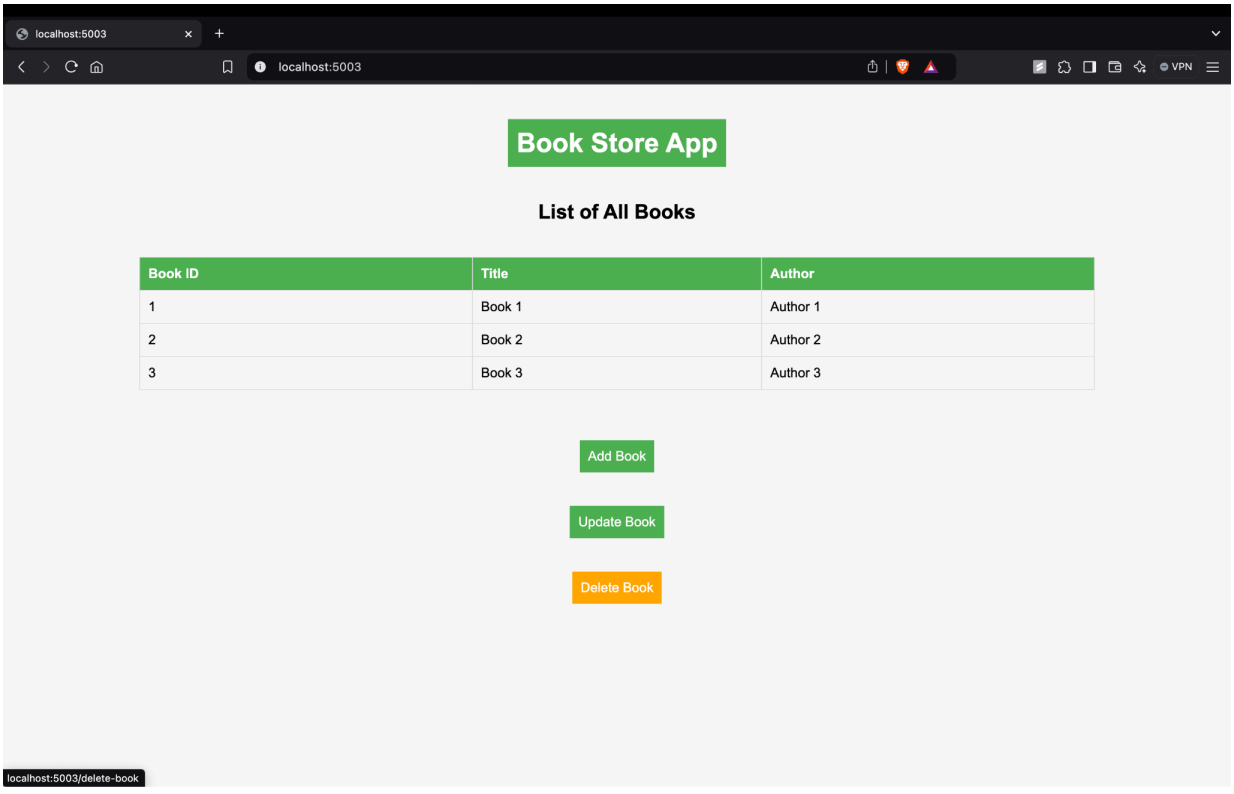
1. Write the code to add a new book. The user should be able to enter the Book Title and Author Name. Once the user submits the required data, the book should be added, and the user should be redirected to the home view showing the updated list of books. (2 points)

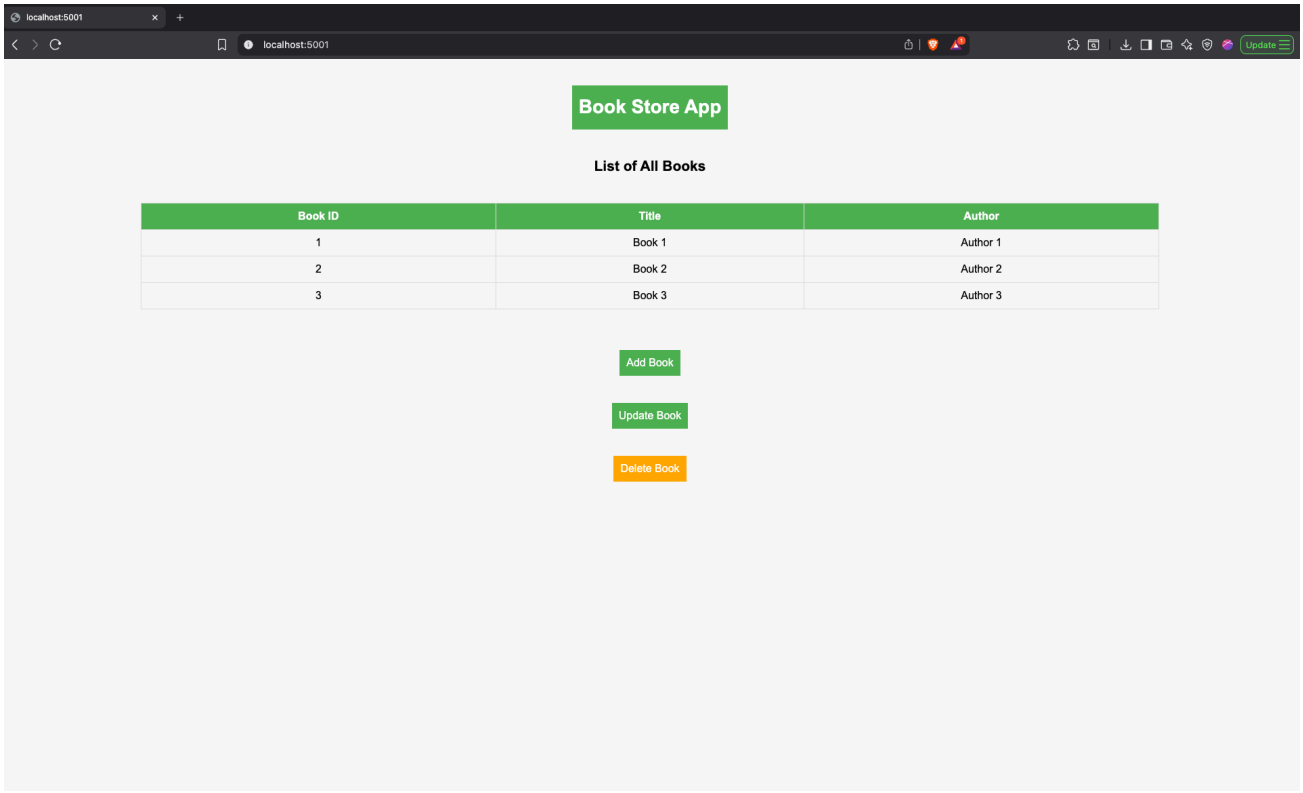


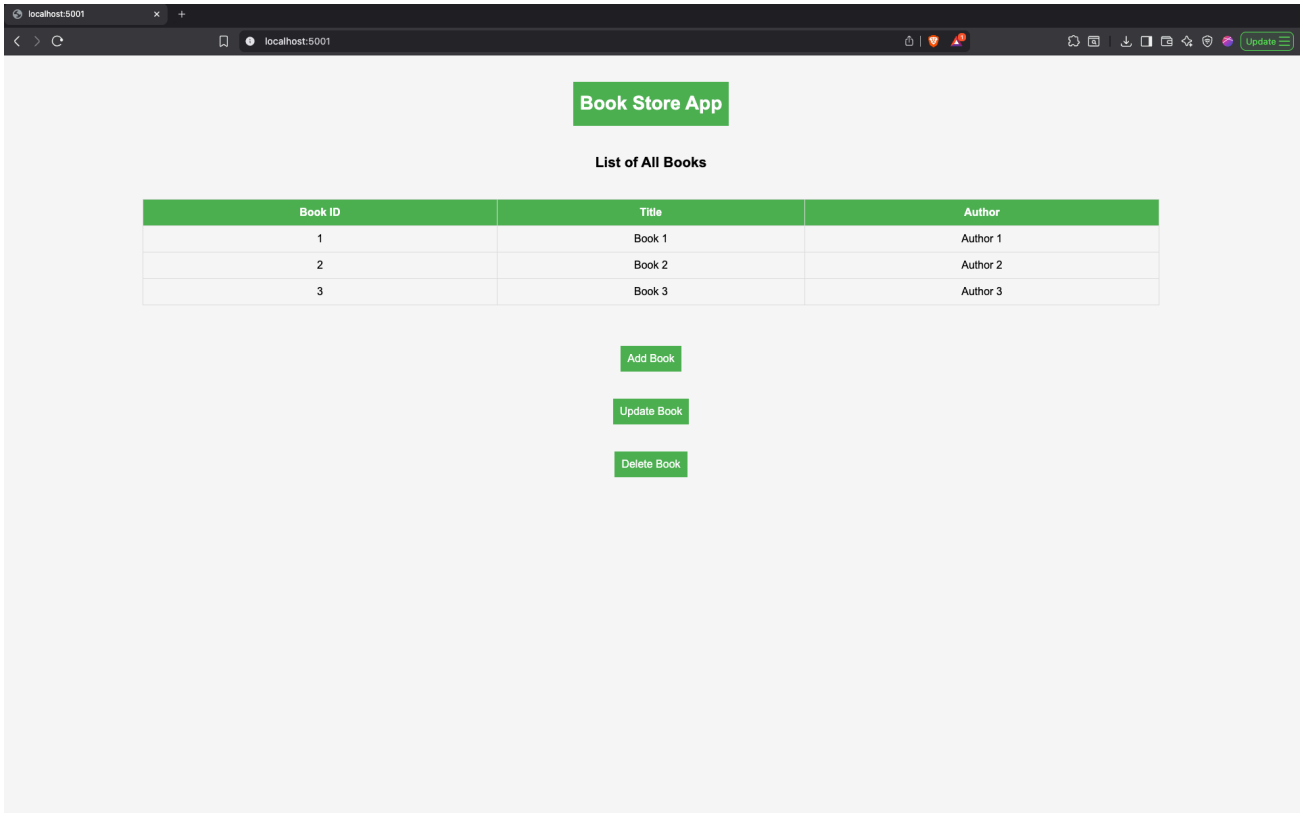

localhost:5001
localhost:5001
Update
Book Store App
List of All Books
Book ID
Title
Author
1
Book 1
Author 1
2
Book 2
Author 2
3
Book 3
Author 3
Add Book
Update Book
Delete Book


localhost:5001
localhost:5001
Update
Book Store App
List of All Books
Book ID
Title
Author
1
Book 1
Author 1
2
Book 2
Author 2
3
Book 3
Author 3
Add Book
Update Book
Delete Book
ocalhost:5001/add-book

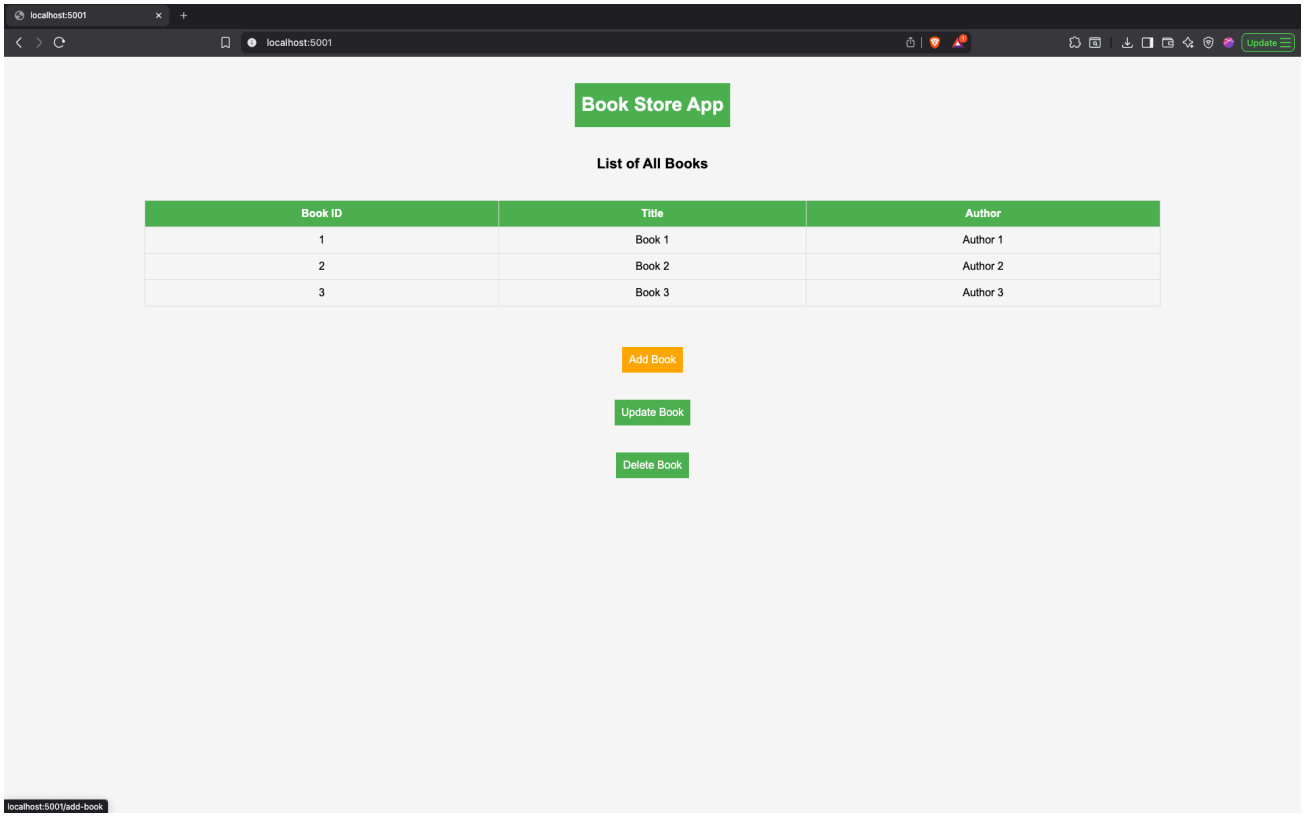

localhost:5001
localhost:5001
Update
Book Store App
List of All Books
Book ID
Title
Author
1
Book 1
Author 1
2
Book 2
Author 2
3
Book 3
Author 3
Add Book
Update Book
Delete Book
localhost:5001/update-book

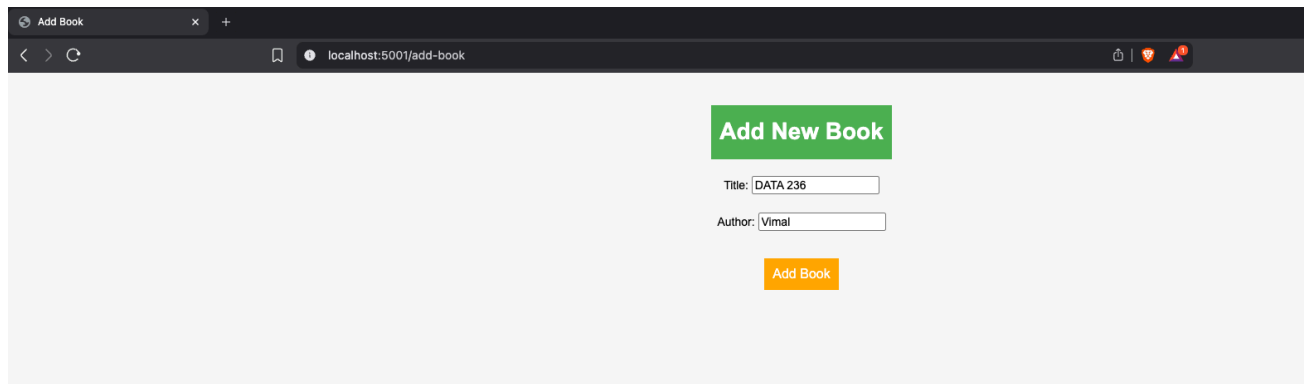

localhost:5003
localhost:5003
VPN
Book Store App
List of All Books
Book ID
Title
Author
1
Book 1
Author 1
2
Book 2
Author 2
3
Book 3
Author 3
Add Book
Update Book
Delete Book
localhost:5003/delete-book

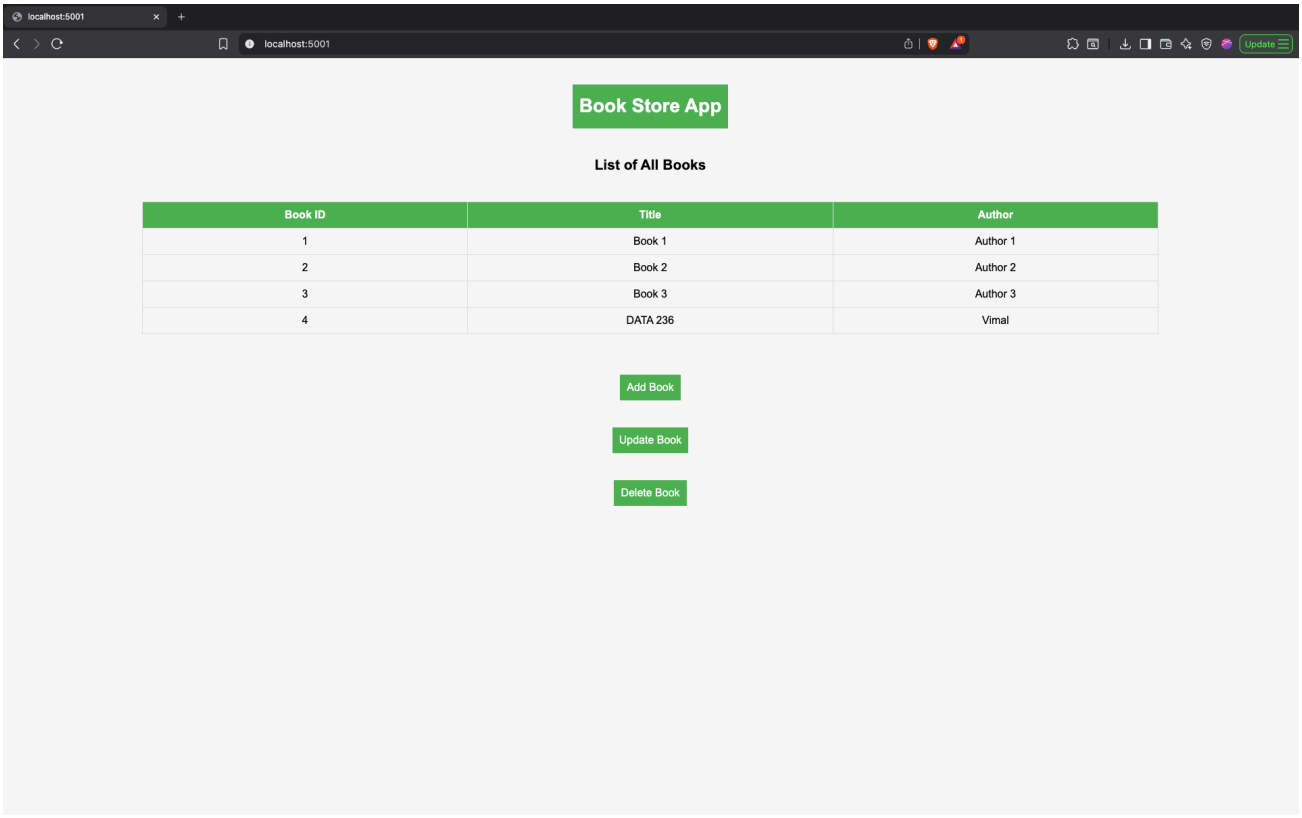

localhost:5001
localhost:5001
Update
Book Store App
List of All Books
Book ID
Title
Author
1
Book 1
Author 1
2
Book 2
Author 2
3
Book 3
Author 3
Add Book
Update Book
Delete Book


localhost:5001
localhost:5001
Update
Book Store App
List of All Books
Book ID
Title
Author
1
Book 1
Author 1
2
Book 2
Author 2
3
Book 3
Author 3
Add Book
Update Book
Delete Book

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. (2 points)






localhost:5001
localhost:5001
Update
Book Store App
List of All Books
Book ID
Title
Author
1
Book 1
Author 1
2
Book 2
Author 2
3
Book 3
Author 3
4
DATA 236
Vimal
Add Book
Update Book
Delete Book

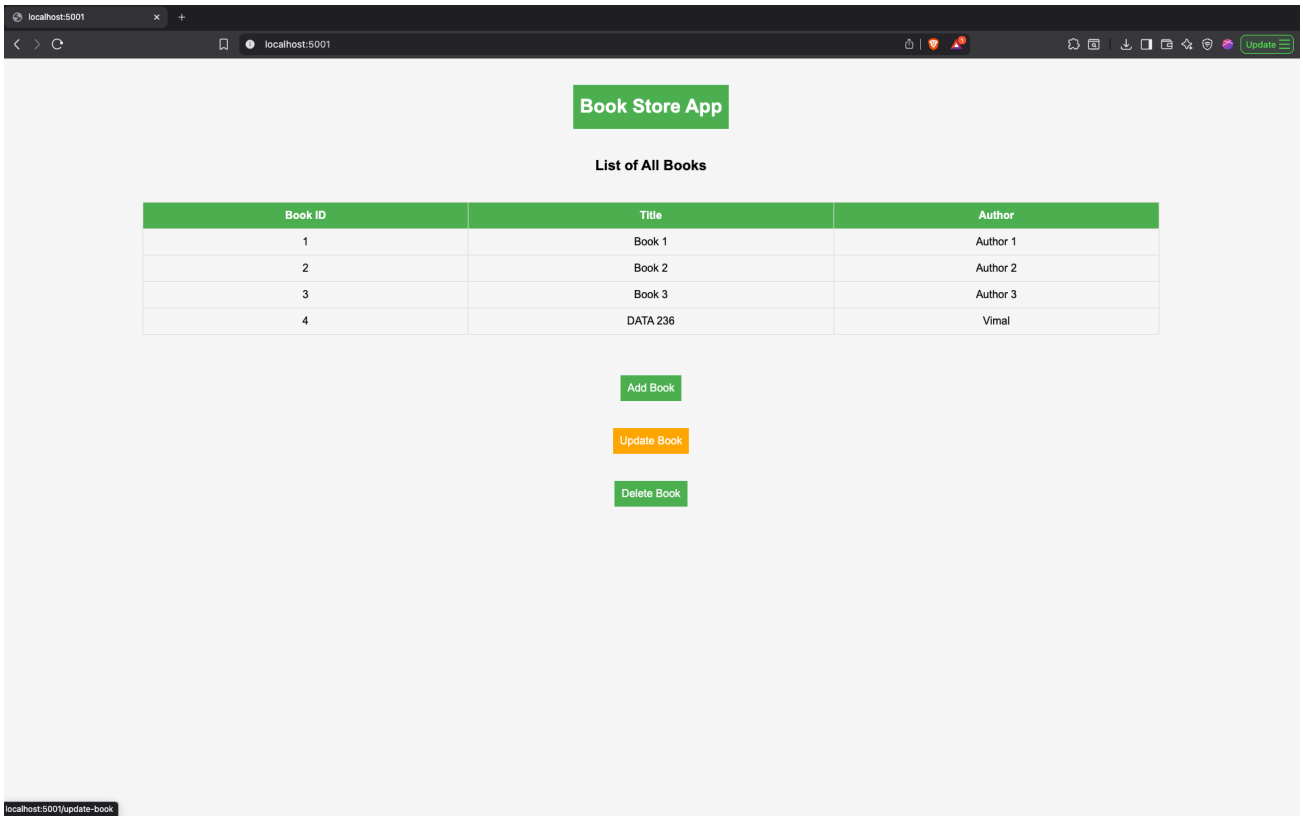
JS index.js M | .gitignore | home.ejs | create.ejs M

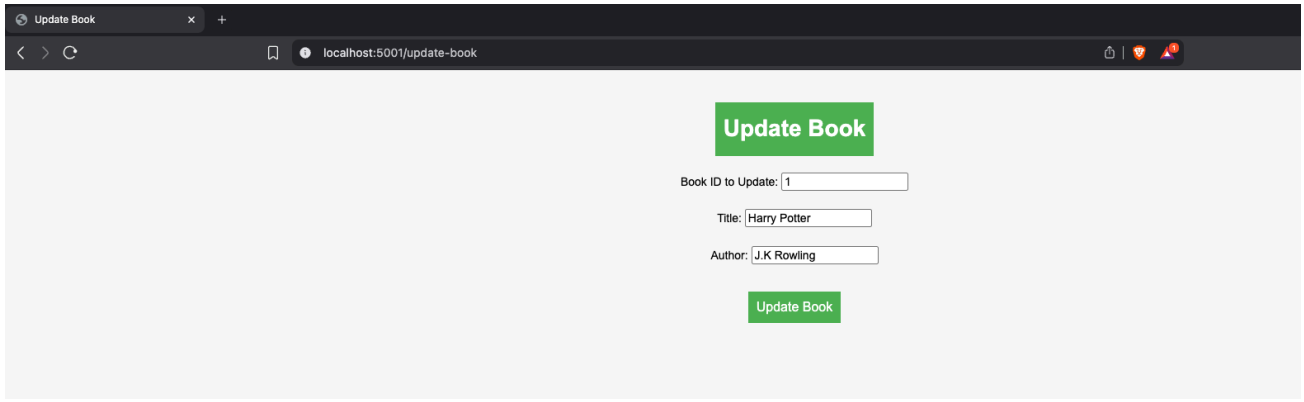
JS index.js > app.post('/update-book') callback

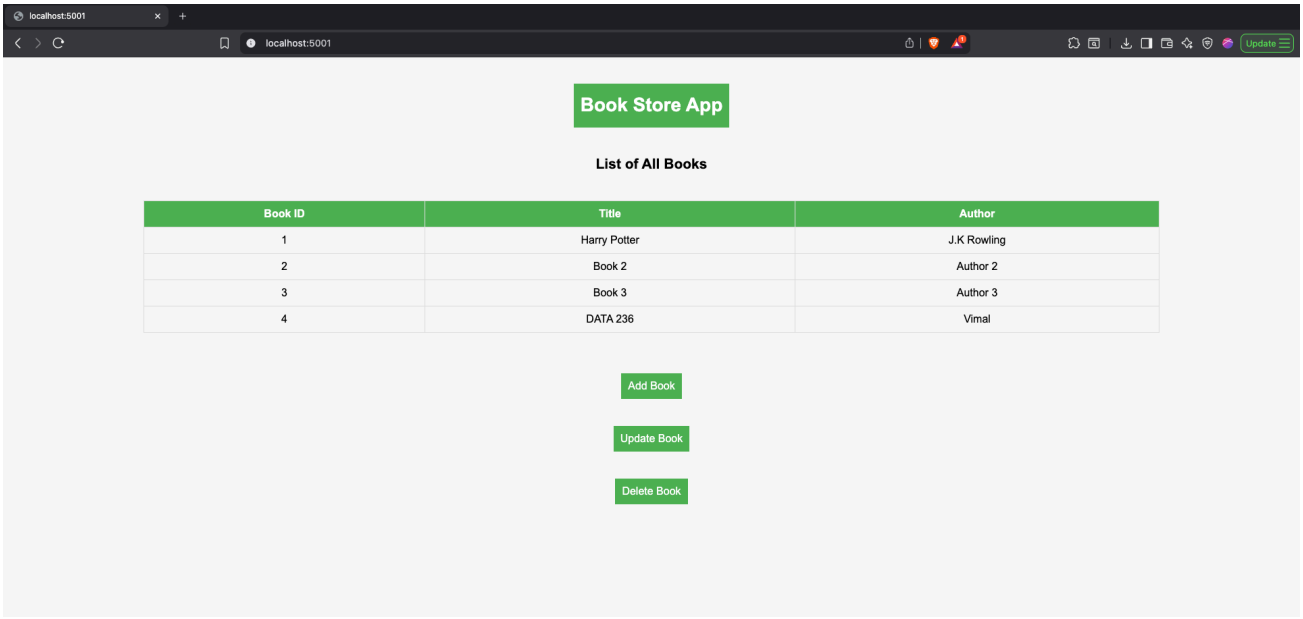
```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');

app.set('view engine', 'ejs');
app.set('views', './views');
app.use(express.static(__dirname + '/public'));

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

let books = [
    { "BookID": "1", "Title": "Book 1", "Author": "Author 1" },
    { "BookID": "2", "Title": "Book 2", "Author": "Author 2" },
    { "BookID": "3", "Title": "Book 3", "Author": "Author 3" }
];

app.get('/', function (req, res) {
    res.render('home', {
        books: books
    });
});

// Add Book
app.get('/add-book', function (req, res) {
    res.render('create');
});

app.post('/add-book', function (req, res) {
    const newBook = {
        "BookID": (books.length + 1).toString(),
        "Title": req.body.title,
        "Author": req.body.author
    };
    books.push(newBook);
    res.redirect('/');
});


app.get('/update-book', function (req, res) {
    res.render('update-book', { book: null });
});
```

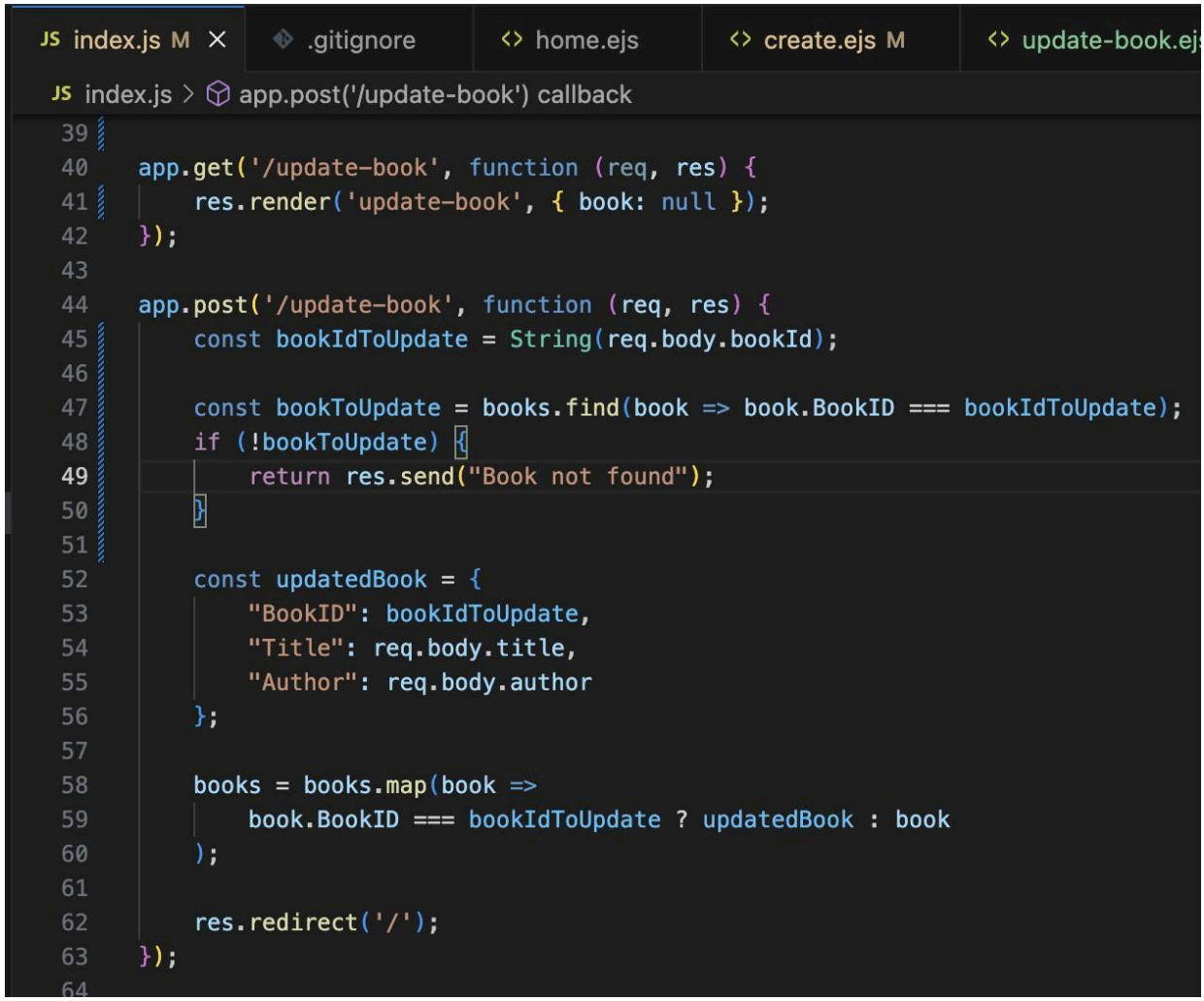
```
43
44  app.post('/update-book', function (req, res) {
45      const bookIdToUpdate = String(req.body.bookId);
46
47      const bookToUpdate = books.find(book => book.BookID === bookIdToUpdate);
48      if (!bookToUpdate) {
49          return res.send("Book not found");
50      }
51
52      const updatedBook = {
53          "BookID": bookIdToUpdate,
54          "Title": req.body.title,
55          "Author": req.body.author
56      };
57
58      books = books.map(book =>
59          book.BookID === bookIdToUpdate ? updatedBook : book
60      );
61
62      res.redirect('/');
63  });
64
65  // Delete Book
66  app.get('/delete-book', function (req, res) {
67      res.render('delete');
68  });
69
70  app.post('/delete-book', function (req, res) {
71      const maxId = Math.max(...books.map(book => parseInt(book.BookID, 10)));
72      books = books.filter(book => parseInt(book.BookID, 10) !== maxId);
73      res.redirect('/');
74  });
75
76
77  app.listen(5001, function () {
78      console.log("Server listening on port 5001");
79  });
```

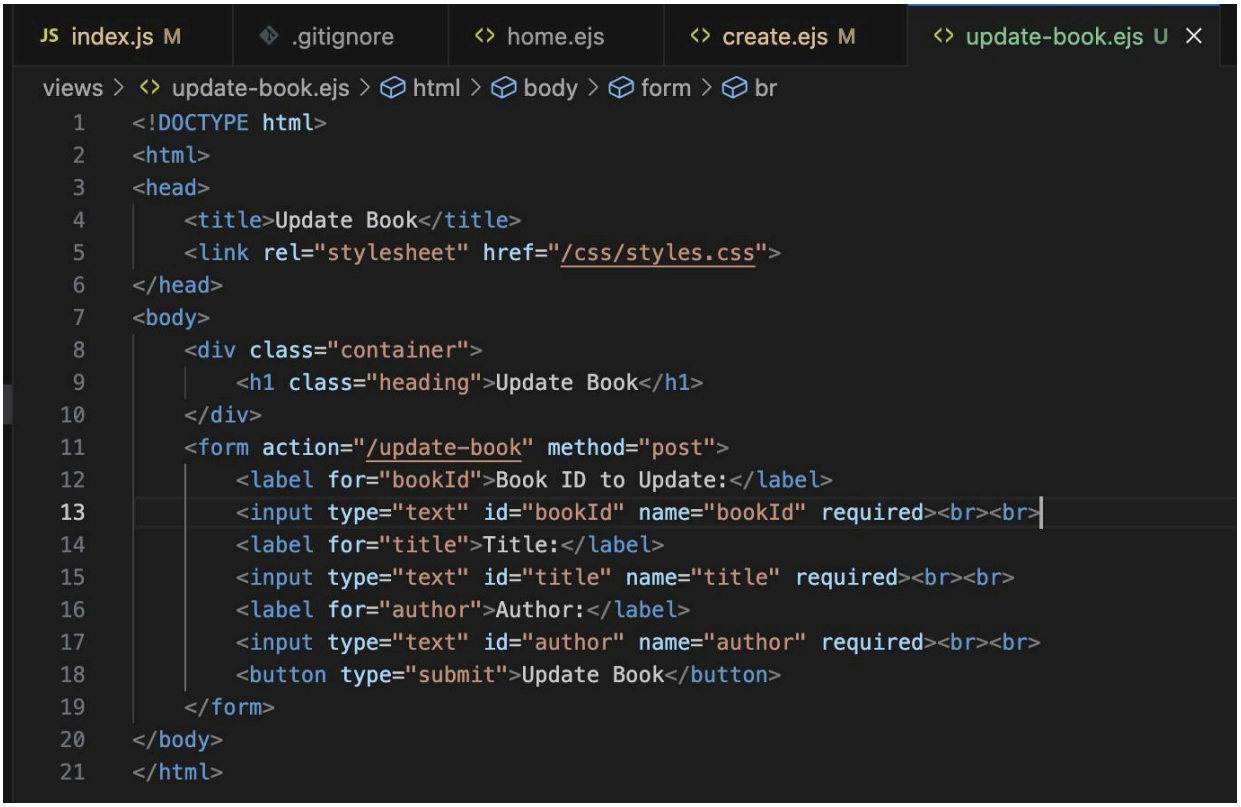
2.Write the code to update book with id 1 to title:”Harry Potter”, Author Name: “J.K Rowling”. After submitting the data, redirect to the home view and show the updated data in the list of books. (2 points)



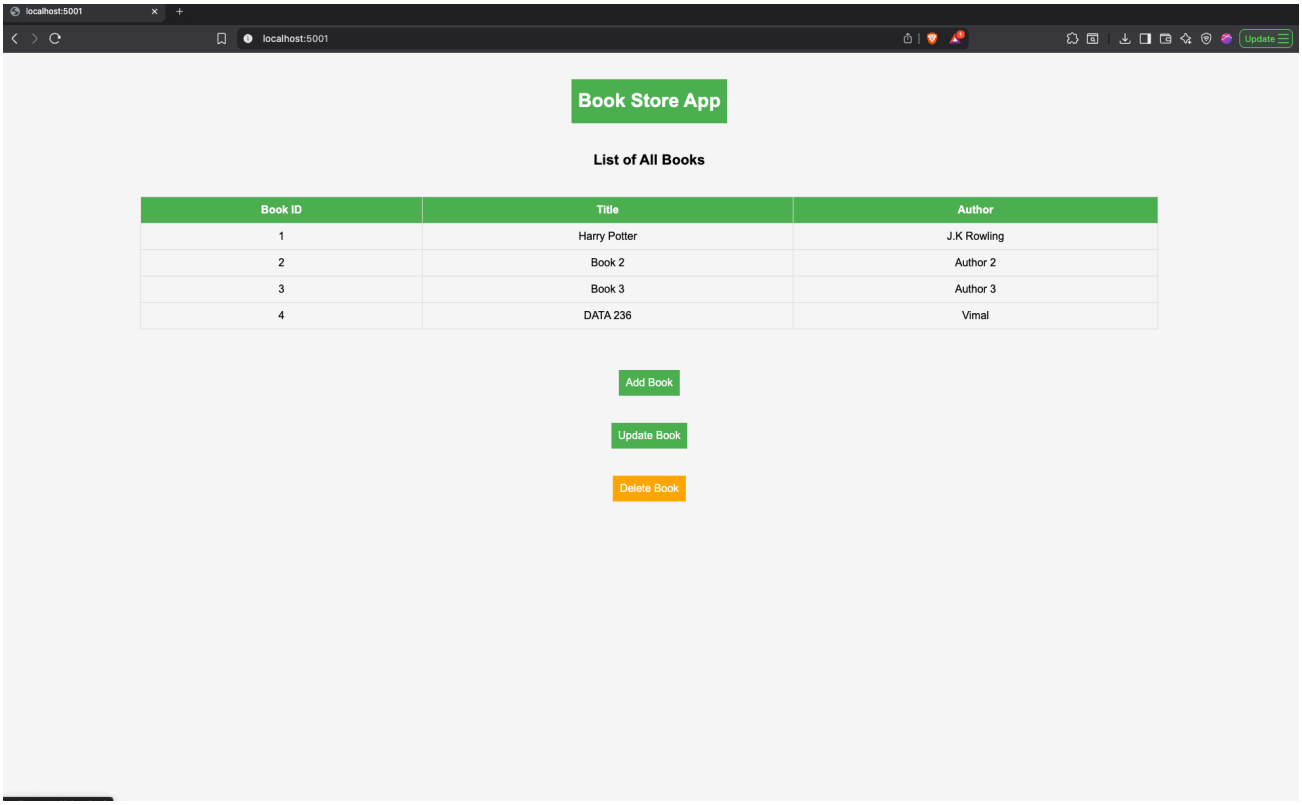


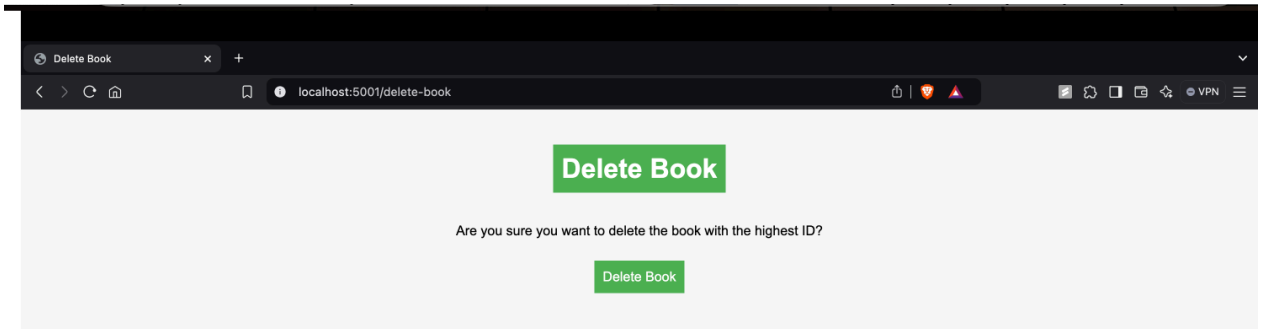

localhost:5001
localhost:5001
Update
Book Store App
List of All Books
Book ID
Title
Author
1
Harry Potter
J.K Rowling
2
Book 2
Author 2
3
Book 3
Author 3
4
DATA 236
Vimal
Add Book
Update Book
Delete Book

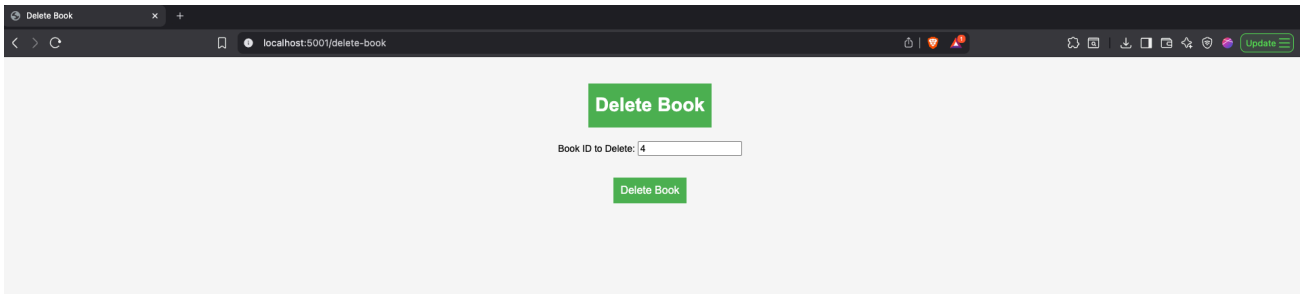

index.js M
.gitignore
home.ejs
create.ejs M
update-book.ej
index.js > app.post('/update-book') callback
39
40 app.get('/update-book', function (req, res) {
41 res.render('update-book', { book: null });
42 });
43
44 app.post('/update-book', function (req, res) {
45 const bookIdToUpdate = String(req.body.bookId);
46
47 const bookToUpdate = books.find(book => book.BookID === bookIdToUpdate);
48 if (!bookToUpdate) {
49 return res.send("Book not found");
50 }
51
52 const updatedBook = {
53 "BookID": bookIdToUpdate,
54 "Title": req.body.title,
55 "Author": req.body.author
56 };
57
58 books = books.map(book =>
59 book.BookID === bookIdToUpdate ? updatedBook : book
60);
61
62 res.redirect('/');
63 });
64

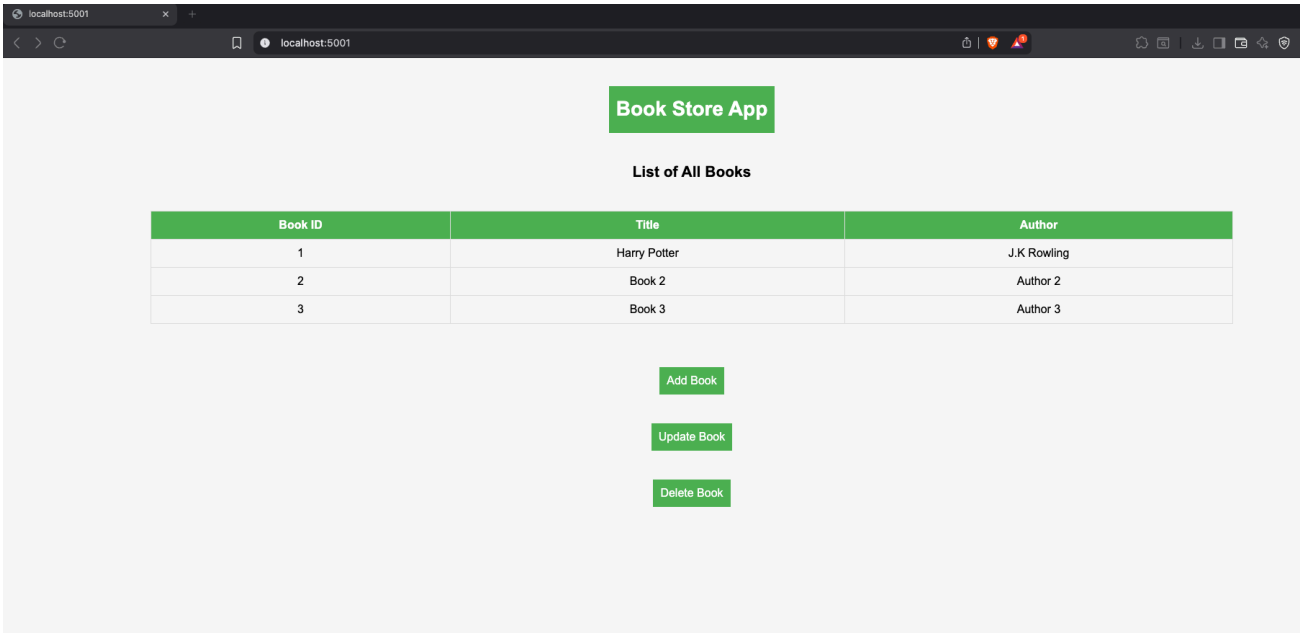


3.Write the code to delete the book with the highest id. After submitting the data, redirect to the home view and show the updated data in the list of books. (2 points)

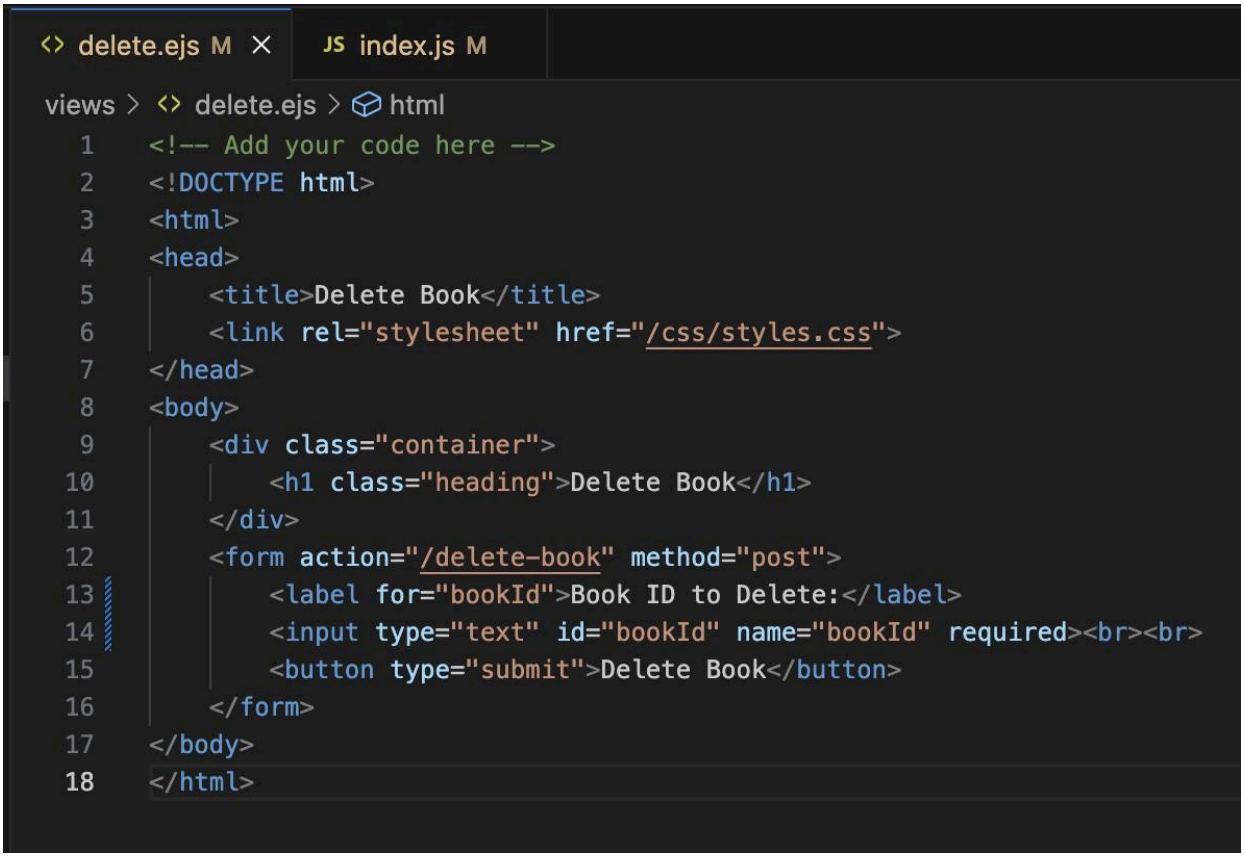

localhost:5001
Update
Book Store App
List of All Books
Book ID
Title
Author
1
Harry Potter
J.K Rowling
2
Book 2
Author 2
3
Book 3
Author 3
4
DATA 236
Vimal
Add Book
Update Book
Delete Book


Delete Book
localhost:5001/delete-book
VPN
Delete Book
Are you sure you want to delete the book with the highest ID?
Delete Book


Delete Book
localhost:5001/delete-book
Update
Delete Book
Book ID to Delete: 4
Delete Book




```
app.get('/delete-book', function (req, res) {
    res.render('delete');
});

app.post('/delete-book', function (req, res) {
    const bookIdToDelete = req.body.bookId;

    if (!bookIdToDelete) {
        return res.send("Book ID is required");
    }

    books = books.filter(book => book.BookID !== bookIdToDelete);

    res.redirect('/');
});
```



Part 3: Stateful Agent Graph

Objective: The goal of this assignment is to refactor your previous sequential agent script into a more robust, stateful graph using the langgraph library. This will implement the supervisor pattern from the lecture, creating a system that can dynamically route tasks and even loop back for self-correction.

Step 1: Understanding the Core Concepts

In your last assignment, you created a simple "waterfall" process: the Planner ran, then the Reviewer, then it ended. This is rigid. A graph-based approach allows for more complex flows, like loops and conditional paths.

We will use langgraph to build this. Key concepts are:

- **AgentState**: A shared dictionary that acts as the "memory" for all agents. Every agent can read from and write to this central state.
- **Nodes**: These are the workers. Each of our agents (Planner, Reviewer) will become a node. A node is just a Python function that takes the current AgentState and returns a dictionary with updates.
- **Edges**: These are the arrows that connect the nodes, defining the flow of control. We'll use conditional edges to let a supervisor decide which path to take.

Step 2: Setting up the State
First, define the shared AgentState using Python's TypedDict. This class will represent the memory of our system. It needs to hold the initial inputs, the outputs from each agent, and a turn counter to prevent infinite loops.

Step 3: Creating the Agent Nodes
Next, convert your Planner and Reviewer logic into standalone functions. Each function must:
1. 2. 3. Accept state: AgentState as its only argument.
Perform its task (e.g., call the LLM).
Return a dictionary containing only the keys of the AgentState it wants to update.

Step 4: Building the Supervisor (The Router)
The supervisor is the "brain" of the operation. It doesn't do the work; it directs it. We split its logic into two parts:
1. **A State-Updating Node (supervisor_node):** This node's only job is to modify the state, like incrementing the turn counter.
2. **A Routing Function (router_logic):** This function reads the state and decides where to go next by returning a string (e.g., "planner", "reviewer", or END).

Step 5: Assembling the Graph
Now, wire everything together in your main function.

Step 6: Running and Testing
Finally, invoke your graph with the initial state and use the .stream() method to see the output from each step.

To test your correction loop, temporarily modify your reviewer_node to always return an issue, and watch the graph route the task back to the planner.

Code:
This below code consists of all the steps from Step 1 to Step 6. Please find the below code for steps

```python
import argparse
import json
import time
import re
from typing import Dict, List, Any, TypedDict
from langchain_ollama import ChatOllama
from langchain_core.messages import HumanMessage, SystemMessage
from langgraph.graph import StateGraph, END
from langgraph.graph.message import add_messages


class AgentState(TypedDict):
    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
    messages: List[Any]


class StatefulAgentGraph:

    def __init__(self, model: str, base_url: str = "http://localhost:11434"):
        """Initialize the stateful agent graph."""
        self.llm = ChatOllama(
            model=model,
            temperature=0.2,
            base_url=base_url,
            num_ctx=2048,
            format="json",
        )
        self.graph = self._build_graph()

    def extract_json_from_response(self, response_content: str) -> Dict[str, Any]:
        """Extract JSON from model response, handling various formats."""
        content = response_content.strip()

        json_patterns = [
            r'\{.*\}',
            r'```json\s*(\{.*?\})\s*```',
            r'```\s*(\{.*?\})\s*```',
        ]

        for pattern in json_patterns:
            matches = re.findall(pattern, content, re.DOTALL)
            if matches:
                try:
                    return json.loads(matches[0])
                except json.JSONDecodeError:
                    continue

        try:
            return json.loads(content)
        except json.JSONDecodeError:
            return None

    def supervisor_node(self, state: AgentState) -> Dict[str, Any]:
        print("---NODE: Supervisor ---")

        turn_count = state.get("turn_count", 0) + 1
        print(f"Turn count: {turn_count}")

        has_proposal = state.get("planner_proposal") is not None and state.get("planner_proposal") != {}

        if not has_proposal:
            print("No proposal found, routing to Planner")
            return {
                "turn_count": turn_count,
                "task": "planner"
            }
        else:
            print("Proposal found, routing to Reviewer")
            return {
                "turn_count": turn_count,
                "task": "reviewer"
            }

    def planner_node(self, state: AgentState) -> Dict[str, Any]:
        """Planner node that analyzes content and generates initial proposal."""
        print("---NODE: Planner ---")

        title = state["title"]
        content = state["content"]
        llm = state["llm"]
```

agents_graph.py > AgentState

```python
class StatefulAgentGraph:
    def planner_node(self, state: AgentState) -> Dict[str, Any]:

        system_prompt = """You are a content analysis expert. Given a blog title and content, you must:

1. Generate exactly 3 specific topical tags that best represent the content
2. Write a concise summary in 25 words or less
3. Identify any potential issues

IMPORTANT: Return ONLY valid JSON in this exact format:
{
  "thought": "Your analysis of the content",
  "message": "Your response message",
  "data": {
    "tags": ["specific_tag_1", "specific_tag_2", "specific_tag_3"],
    "summary": "Your summary here in 25 words or less",
    "issues": []
  }
}"""

        user_prompt = f"""Title: {title}
Content: {content}

Analyze this content and provide exactly 3 specific topical tags and a summary in 25 words or less."""

        messages = [
            SystemMessage(content=system_prompt),
            HumanMessage(content=user_prompt)
        ]

        start_time = time.time()
        response = llm.invoke(messages)
        end_time = time.time()

        # Extract JSON from response
        proposal = self.extract_json_from_response(response.content)

        if proposal is None:
            proposal = {
                "thought": "Content analysis completed",
                "message": "Analyzed the blog content for planning",
                "data": {
                    "tags": ["machine learning", "artificial intelligence", "data analysis"],
                    "summary": "Introduction to machine learning concepts and applications",
                    "issues": []
                }
            }

        proposal["execution_time_ms"] = int((end_time - start_time) * 1000)

        print(f"Planner proposal: {json.dumps(proposal, indent=2)}")

        return {
            "planner_proposal": proposal,
            "task": "supervisor"
        }

    def reviewer_node(self, state: AgentState) -> Dict[str, Any]:
        """Reviewer node that reviews the planner's proposal and provides feedback."""
        print("---NODE: Reviewer ---")

        title = state["title"]
        content = state["content"]
        planner_proposal = state["planner_proposal"]
        llm = state["llm"]

        system_prompt = """You are a content review expert. Review the Planner's analysis and suggest improvements.

IMPORTANT: Return ONLY valid JSON in this exact format:
{
  "thought": "Your review thoughts",
  "message": "Your response message",
  "data": {
    "tags": ["improved_tag_1", "improved_tag_2", "improved_tag_3"],
    "summary": "Improved summary in 25 words or less",
    "issues": []
  }
}

Focus on making the tags more specific and the summary more concise."""

        user_prompt = f"""Title: {title}
Content: {content}
```

agents_graph.py > AgentState

```python
class StatefulAgentGraph:
    def reviewer_node(self, state: AgentState) -> Dict[str, Any]:
Planner's output:
{json.dumps(planner_proposal, indent=2)}

Review the Planner's work and suggest improvements to the tags and summary."""

        messages = [
            SystemMessage(content=system_prompt),
            HumanMessage(content=user_prompt)
        ]

        start_time = time.time()
        response = llm.invoke(messages)
        end_time = time.time()

        # Extract JSON from response
        feedback = self.extract_json_from_response(response.content)

        if feedback is None:
            feedback = {
                "thought": "Review completed",
                "message": "Reviewed the planner's output",
                "data": {
                    "tags": planner_proposal.get("data", {}).get("tags", ["review", "content", "analysis"]),
                    "summary": planner_proposal.get("data", {}).get("summary", "Content reviewed"),
                    "issues": []
                }
            }

        feedback["execution_time_ms"] = int((end_time - start_time) * 1000)

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

        return {
            "reviewer_feedback": feedback,
            "task": "supervisor"
        }

    def should_continue(self, state: AgentState) -> str:
        """Conditional edge function to determine next step."""
        turn_count = state.get("turn_count", 0)
        reviewer_feedback = state.get("reviewer_feedback", {})

        if turn_count > 5:
            print("Maximum turns reached, ending")
            return "end"

        issues = reviewer_feedback.get("data", {}).get("issues", [])
        has_issues = len(issues) > 0

        if has_issues:
            print("Issues found, routing back to Planner")
            return "planner"
        else:
            print("No issues found, ending")
            return "end"

    def _build_graph(self) -> StateGraph:
        """Build the stateful agent graph."""
        # Create the graph
        workflow = StateGraph(AgentState)

        # Add nodes
        workflow.add_node("supervisor", self.supervisor_node)
        workflow.add_node("planner", self.planner_node)
        workflow.add_node("reviewer", self.reviewer_node)

        workflow.add_conditional_edges(
            "supervisor",
            lambda state: state.get("task", "planner"),
            {
                "planner": "planner",
                "reviewer": "reviewer"
            }
        )

        workflow.add_conditional_edges(
            "reviewer",
            self.should_continue,
            {
                "planner": "planner",
                "end": END
            }
        )
```

agents_graph.py > StatefulAgentGraph > test_correction_loop

```python
class StatefulAgentGraph:
    def _build_graph(self) -> StateGraph:

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

        workflow.set_entry_point("supervisor")

        return workflow.compile()

    def run(self, title: str, content: str, email: str, strict: bool = False) -> Dict[str, Any]:
        """Run the stateful agent graph using .stream() method as required by Step 6."""
        print("Starting Stateful Agent Graph...")
        print(f"Processing: {title}")
        print("-" * 50)

        initial_state = {
            "title": title,
            "content": content,
            "email": email,
            "strict": strict,
            "task": "planner",
            "llm": self.llm,
            "planner_proposal": {},
            "reviewer_feedback": {},
            "turn_count": 0,
            "messages": []
        }

        print("Streaming graph execution step by step:")
        print("-" * 50)

        final_state = initial_state
        step_count = 0
        for step_output in self.graph.stream(initial_state):
            step_count += 1
            print(f"Step {step_count}: {step_output}")
            print("-" * 30)
            for node_name, node_output in step_output.items():
                final_state.update(node_output)

        final_output = self._create_final_output(final_state)

        print("\n" + "="*50)
        print("Stateful Agent Graph completed successfully!")
        print("="*50)

        return final_output

    def test_correction_loop(self, title: str, content: str, email: str, strict: bool = False) -> Dict[str, Any]:
        """Test the correction loop by modifying reviewer to always return issues as required by Step 6."""
        print("Testing correction loop - Reviewer will always find issues...")
        print("=" * 60)

        # Store original reviewer node
        original_reviewer = self.reviewer_node

        def test_reviewer_node(state: AgentState) -> Dict[str, Any]:
            """Modified reviewer that always finds issues for testing."""
            print("---NODE: Reviewer (TEST MODE - finds issues) ---")

            # Call original reviewer
            result = original_reviewer(state)

            # Force issues to be present, To test correction loop, temporarily modified reviewer node to always return an issue, and watch the graph route the task back to the planner.
            if "data" in result:
                result["data"]["issues"] = [{"test": "Forced issue for testing correction loop"}]

            print(f"Test Reviewer feedback (with forced issues): {json.dumps(result, indent=2)}")
            return result

        self.graph = self._build_graph_with_custom_reviewer(test_reviewer_node)

        print("Streaming correction loop test step by step:")
        print("-" * 50)

        initial_state = {
            "title": title,
            "content": content,
            "email": email,
            "strict": strict,
            "task": "planner",
            "llm": self.llm,
            "planner_proposal": {},
            "reviewer_feedback": {},
            "turn_count": 0,
            "messages": []
        }

        final_state = initial_state
        step_count = 0
        for step_output in self.graph.stream(initial_state):
            step_count += 1
            print(f"Test Step {step_count}: {step_output}")
            print("-" * 30)
            for node_name, node_output in step_output.items():
                final_state.update(node_output)

        self.graph = self._build_graph()

        final_output = self._create_final_output(final_state)

        print("\n" + "="*50)
        print("Correction Loop Test completed!")
        print("="*50)
```

```python
agents_graph.py > StatefulAgentGraph > test_correction_loop
class StatefulAgentGraph:
    def test_correction_loop(self, title: str, content: str, email: str, strict: bool = False) -> Dict[str, Any]:
        return final_output

    def _build_graph_with_custom_reviewer(self, custom_reviewer_node):
        """Build graph with custom reviewer node for testing."""
        workflow = StateGraph(AgentState)

        # Add nodes
        workflow.add_node("supervisor", self.supervisor_node)
        workflow.add_node("planner", self.planner_node)
        workflow.add_node("reviewer", custom_reviewer_node)

        workflow.add_conditional_edges(
            "supervisor",
            lambda state: state.get("task", "planner"),
            {
                "planner": "planner",
                "reviewer": "reviewer"
            }
        )

        workflow.add_conditional_edges(
            "reviewer",
            self.should_continue,
            {
                "planner": "planner",
                "end": END
            }
        )

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

        workflow.set_entry_point("supervisor")

        return workflow.compile()

    def _create_final_output(self, state: AgentState) -> Dict[str, Any]:
        """Create the final output from the state."""
        planner_proposal = state.get("planner_proposal", {})
        reviewer_feedback = state.get("reviewer_feedback", {})

        final_data = reviewer_feedback.get("data", planner_proposal.get("data", {}))

        return {
            "title": state["title"],
            "email": state["email"],
            "content": state["content"],
            "agents": [
                {
                    "role": "Planner",
                    "content": planner_proposal.get("message", ""),
                    "execution_time_ms": planner_proposal.get("execution_time_ms", 0)
                },
                {
                    "role": "Reviewer",
                    "content": reviewer_feedback.get("message", ""),
                    "execution_time_ms": reviewer_feedback.get("execution_time_ms", 0)
                }
            ],
            "final": {
                "tags": final_data.get("tags", []),
                "summary": final_data.get("summary", ""),
                "issues": final_data.get("issues", [])
            },
            "submissionDate": time.strftime("%Y-%m-%dT%H:%M:%SZ", time.gmtime()),
            "turn_count": state.get("turn_count", 0)
        }


def main():
    """Main function to run the stateful agent graph."""
    parser = argparse.ArgumentParser(description="Stateful Agent Graph with Supervisor Pattern")
    parser.add_argument("--model", default="smollm:1.7b", help="Ollama model to use")
    parser.add_argument("--title", required=True, help="Blog title")
    parser.add_argument("--content", required=True, help="Blog content")
    parser.add_argument("--email", required=True, help="Email address")
    parser.add_argument("--base_url", default="http://localhost:11434", help="Ollama base URL")
    parser.add_argument("--strict", action="store_true", help="Enable strict mode")
    parser.add_argument("--test_loop", action="store_true", help="Test correction loop by forcing issues")

    args = parser.parse_args()

    try:
        # Initialize the stateful agent graph
        graph = StatefulAgentGraph(args.model, args.base_url)

        # Run the graph or test correction loop
        if args.test_loop:
            result = graph.test_correction_loop(args.title, args.content, args.email, args.strict)
        else:
            result = graph.run(args.title, args.content, args.email, args.strict)

        print("\nFinal Result:")
        print(json.dumps(result, indent=2))

    except Exception as e:
        print(f"Error: {e}")
        print("Make sure Ollama is running and the model is available.")
        return 1

    return 0


if __name__ == "__main__":
    exit(main())
```

Terminal output:

```
spartan@MLK-SCS-M7J3NJ9HTV 236 % python3 agents_graph.py --model phi3:mini --title "Machine Learning Fundamentals" --content "Machine learning
 is a subset of artificial intelligence that enables computers to learn and make decisions from data without being explicitly programmed." --e
mail "student@sjsu.edu" --strict
/Users/spartan/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+,
 currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
Starting Stateful Agent Graph...
Processing: Machine Learning Fundamentals
--------------------------------------------------
Streaming graph execution step by step:
--------------------------------------------------
---NODE: Supervisor ---
Turn count: 1
No proposal found, routing to Planner
Step 1: {'supervisor': {'turn_count': 1, 'task': 'planner'}}
------------------------------
---NODE: Planner ---
Planner proposal: {
  "thought": "The blog post introduces the concept of machine learning as part of AI, focusing on its ability to learn from data.",
  "message": "Machine Learning: Subset of AI for decision making through self-learning without explicit programming",
  "data": {
    "tags": [
      "Machine Learning Basics",
      "Artifics Intelligence",
      "Data Driven Decisions"
    ],
    "summary": "Blog explores machine learning as a data-driven subset of AI, enabling decision making through self-learning without explicit
programming.",
    "issues": []
  },
  "execution_time_ms": 5082
}
Step 2: {'planner': {'planner_proposal': {'thought': 'The blog post introduces the concept of machine learning as part of AI, focusing on its
ability to learn from data.', 'message': 'Machine Learning: Subset of AI for decision making through self-learning without explicit programmin
g', 'data': {'tags': ['Machine Learning Basics', 'Artifics Intelligence', 'Data Driven Decisions'], 'summary': 'Blog explores machine learning
 as a data-driven subset of AI, enabling decision making through self-learning without explicit programming.', 'issues': []}, 'execution_time_
ms': 5082}, 'task': 'supervisor'}}
------------------------------
---NODE: Supervisor ---
Turn count: 2
Proposal found, routing to Reviewer
Step 3: {'supervisor': {'turn_count': 2, 'task': 'reviewer'}}
------------------------------
---NODE: Reviewer ---
Reviewer feedback: {
  "thought": "The blog post provides a general overview of machine learning within AI, but it lacks specificity in its approach towards improv
ing understanding for beginners.",
  "message": "Machine Learning: Essential concepts & applications as part of Artificial Intelligence",
  "data": {
    "tags": [
      "Intro to Machine Learning",
      "AI Fundamentals",
      "Practical Applications"
    ],
    "summary": "Blog delves into machine learning, its core concepts and practical applications in AI for beginners.",
    "issues": []
  },
  "execution_time_ms": 6817
}
No issues found, ending
Step 4: {'reviewer': {'reviewer_feedback': {'thought': 'The blog post provides a general overview of machine learning within AI, but it lacks
specificity in its approach towards improving understanding for beginners.', 'message': 'Machine Learning: Essential concepts & applications a
s part of Artificial Intelligence', 'data': {'tags': ['Intro to Machine Learning', 'AI Fundamentals', 'Practical Applications'], 'summary': 'B
log delves into machine learning, its core concepts and practical applications in AI for beginners.', 'issues': []}, 'execution_time_ms': 6817
}, 'task': 'supervisor'}}
------------------------------

==================================================
Stateful Agent Graph completed successfully!
==================================================

Final Result:
{
  "title": "Machine Learning Fundamentals",
```

```
==================================================
Stateful Agent Graph completed successfully!
==================================================

Final Result:
{
  "title": "Machine Learning Fundamentals",
  "email": "student@sjsu.edu",
  "content": "Machine learning is a subset of artificial intelligence that enables computers to learn and make decisions from data without bei
ng explicitly programmed.",
  "agents": [
    {
      "role": "Planner",
      "content": "Machine Learning: Subset of AI for decision making through self-learning without explicit programming",
      "execution_time_ms": 5082
    },
    {
      "role": "Reviewer",
      "content": "Machine Learning: Essential concepts & applications as part of Artificial Intelligence",
      "execution_time_ms": 6817
    }
  ],
  "final": {
    "tags": [
      "Intro to Machine Learning",
      "AI Fundamentals",
      "Practical Applications"
    ],
    "summary": "Blog delves into machine learning, its core concepts and practical applications in AI for beginners.",
    "issues": []
  },
  "submissionDate": "2025-09-13T00:56:42Z",
  "turn_count": 2
}
```

Step 6: Running and Testing

Finally, invoke your graph with the initial state and use the .stream() method to see the output from each step.

To test your correction loop, temporarily modify your reviewer_node to always return an issue, and watch the graph route the task back to the planner.

I have changed accordingly as part of testing step 6.

```
spartan@MLK-SCS-M7J3NJ9HTV 236 % python3 agents_graph.py --model phi3:mini --title "Test Content" --content "This is a test of the correction
loop functionality." --email "test@sjsu.edu" --strict --test_loop
/Users/spartan/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1+,
 currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
Testing correction loop - Reviewer will always find issues...
============================================================
Streaming correction loop test step by step:
------------------------------------------------------------
---NODE: Supervisor ---
Turn count: 1
No proposal found, routing to Planner
Test Step 1: {'supervisor': {'turn_count': 1, 'task': 'planner'}}
------------------------------
---NODE: Planner ---
Planner proposal: {
  "thought": "The blog post seems to be testing something related to technology, specifically focusing on feedback mechanisms.",
  "message": "Test Content: Blog about evaluating correction loop functionality for system improvements",
  "data": {
    "tags": [
      "technology_testing",
      "correction_loop",
      "system_feedback"
    ],
    "summary": "Blog post tests the effectiveness of a feedback mechanism designed to correct errors within systems.",
    "issues": [
      "The blog lacks technical depth which might be needed for readers interested in system engineering."
    ]
  },
  "execution_time_ms": 4486
}
Test Step 2: {'planner': {'planner_proposal': {'thought': 'The blog post seems to be testing something related to technology, specifically foc
using on feedback mechanisms.', 'message': 'Test Content: Blog about evaluating correction loop functionality for system improvements', 'data'
: {'tags': ['technology_testing', 'correction_loop', 'system_feedback'], 'summary': 'Blog post tests the effectiveness of a feedback mechanism
 designed to correct errors within systems.', 'issues': ['The blog lacks technical depth which might be needed for readers interested in syste
m engineering.']}, 'execution_time_ms': 4486}, 'task': 'supervisor'}}
------------------------------
---NODE: Supervisor ---
Turn count: 2
Proposal found, routing to Reviewer
Test Step 3: {'supervisor': {'turn_count': 2, 'task': 'reviewer'}}
------------------------------
---NODE: Reviewer (TEST MODE - Always finds issues) ---
---NODE: Reviewer ---
Reviewer feedback: {
  "thought": "The planner has identified a key aspect of content but could refine their focus on specificity for technology enthusiasts.",
  "message": "Test Content: In-depth blog post evaluating correction loop functionality in system engineering",
  "data": {
    "tags": [
      "system_engineering",
      "error_correction",
      "feedback_mechanisms"
    ],
    "summary": "Blog delves into the intricacury of correction loops, assessing their role in system error rectification.",
    "issues": [
      "The summary could be more concise and directly reflect content specificity."
    ]
  },
  "execution_time_ms": 6348
}
Test Reviewer feedback (with forced issues): {
  "reviewer_feedback": {
    "thought": "The planner has identified a key aspect of content but could refine their focus on specificity for technology enthusiasts.",
    "message": "Test Content: In-depth blog post evaluating correction loop functionality in system engineering",
    "data": {
      "tags": [
        "system_engineering",
        "error_correction",
        "feedback_mechanisms"
      ],
      "summary": "Blog delves into the intricacury of correction loops, assessing their role in system error rectification.",
      "issues": [
        "The summary could be more concise and directly reflect content specificity."
      ]
```

```
        "The summary could be more concise and directly reflect content specificity."
      ]
    },
    "execution_time_ms": 6348
  },
  "task": "supervisor"
}
Issues found, routing back to Planner
Test Step 4: {'reviewer': {'reviewer_feedback': {'thought': 'The planner has identified a key aspect of content but could refine their focus o
n specificity for technology enthusiasts.', 'message': 'Test Content: In-depth blog post evaluating correction loop functionality in system en
gineering', 'data': {'tags': ['system_engineering', 'error_correction', 'feedback_mechanisms'], 'summary': 'Blog delves into the intricacury o
f correction loops, assessing their role in system error rectification.', 'issues': ['The summary could be more concise and directly reflect c
ontent specificity.']}, 'execution_time_ms': 6348}, 'task': 'supervisor'}}
------------------------------
---NODE: Planner ---
Planner proposal: {
  "thought": "The blog post seems to be testing something related to technology, specifically focusing on feedback mechanisms.",
  "message": "Test Content: A test of the correction loop functionality",
  "data": {
    "tags": [
      "Technology Testing",
      "Feedback Mechanism Evaluation",
      "Correction Loop"
    ],
    "summary": "Blog post tests a system's ability to self-correct through feedback.",
    "issues": []
  },
  "execution_time_ms": 3853
}
Test Step 5: {'planner': {'planner_proposal': {'thought': 'The blog post seems to be testing something related to technology, specifically foc
using on feedback mechanisms.', 'message': 'Test Content: A test of the correction loop functionality', 'data': {'tags': ['Technology Testing'
, 'Feedback Mechanism Evaluation', 'Correction Loop'], 'summary': "Blog post tests a system's ability to self-correct through feedback.", 'iss
ues': []}, 'execution_time_ms': 3853}, 'task': 'supervisor'}}
------------------------------
---NODE: Supervisor ---
Turn count: 3
Proposal found, routing to Reviewer
Test Step 6: {'supervisor': {'turn_count': 3, 'task': 'reviewer'}}
------------------------------
---NODE: Reviewer (TEST MODE - Always finds issues) ---
---NODE: Reviewer ---
Reviewer feedback: {
  "thought": "The blog post title is too generic, lacks context on what 'correction loop functionality' refers to.",
  "message": "Test Content: A test of our new correction loop in user interface design",
  "data": {
    "tags": [
      "User Interface Design Testing",
      "Correction Loop Implementation Review",
      "Feedback System Evaluation"
    ],
    "summary": "Blog post evaluates a newly implemented correction loop in user interface design, focusing on its feedback system.",
    "issues": []
  },
  "execution_time_ms": 5481
}
Test Reviewer feedback (with forced issues): {
  "reviewer_feedback": {
    "thought": "The blog post title is too generic, lacks context on what 'correction loop functionality' refers to.",
    "message": "Test Content: A test of our new correction loop in user interface design",
    "data": {
      "tags": [
        "User Interface Design Testing",
        "Correction Loop Implementation Review",
        "Feedback System Evaluation"
      ],
      "summary": "Blog post evaluates a newly implemented correction loop in user interface design, focusing on its feedback system.",
      "issues": []
    },
    "execution_time_ms": 5481
  },
  "task": "supervisor"
}
No issues found, ending
```

```
}
No issues found, ending
Test Step 7: {'reviewer': {'reviewer_feedback': {'thought': "The blog post title is too generic, lacks context on what 'correction loop functi
onality' refers to.", 'message': 'Test Content: A test of our new correction loop in user interface design', 'data': {'tags': ['User Interface
 Design Testing', 'Correction Loop Implementation Review', 'Feedback System Evaluation'], 'summary': 'Blog post evaluates a newly implemented
correction loop in user interface design, focusing on its feedback system.', 'issues': []}, 'execution_time_ms': 5481}, 'task': 'supervisor'}}
----------------------------

================================================
Correction Loop Test completed!
================================================

Final Result:
{
  "title": "Test Content",
  "email": "test@sjsu.edu",
  "content": "This is a test of the correction loop functionality.",
  "agents": [
    {
      "role": "Planner",
      "content": "Test Content: A test of the correction loop functionality",
      "execution_time_ms": 3853
    },
    {
      "role": "Reviewer",
      "content": "Test Content: A test of our new correction loop in user interface design",
      "execution_time_ms": 5481
    }
  ],
  "final": {
    "tags": [
      "User Interface Design Testing",
      "Correction Loop Implementation Review",
      "Feedback System Evaluation"
    ],
    "summary": "Blog post evaluates a newly implemented correction loop in user interface design, focusing on its feedback system.",
    "issues": []
  },
  "submissionDate": "2025-09-13T00:57:19Z",
  "turn_count": 3
}
```