AI ASSISTED CODING LAB 15.3

Name:BANDI SRISHANTH

Batch:06

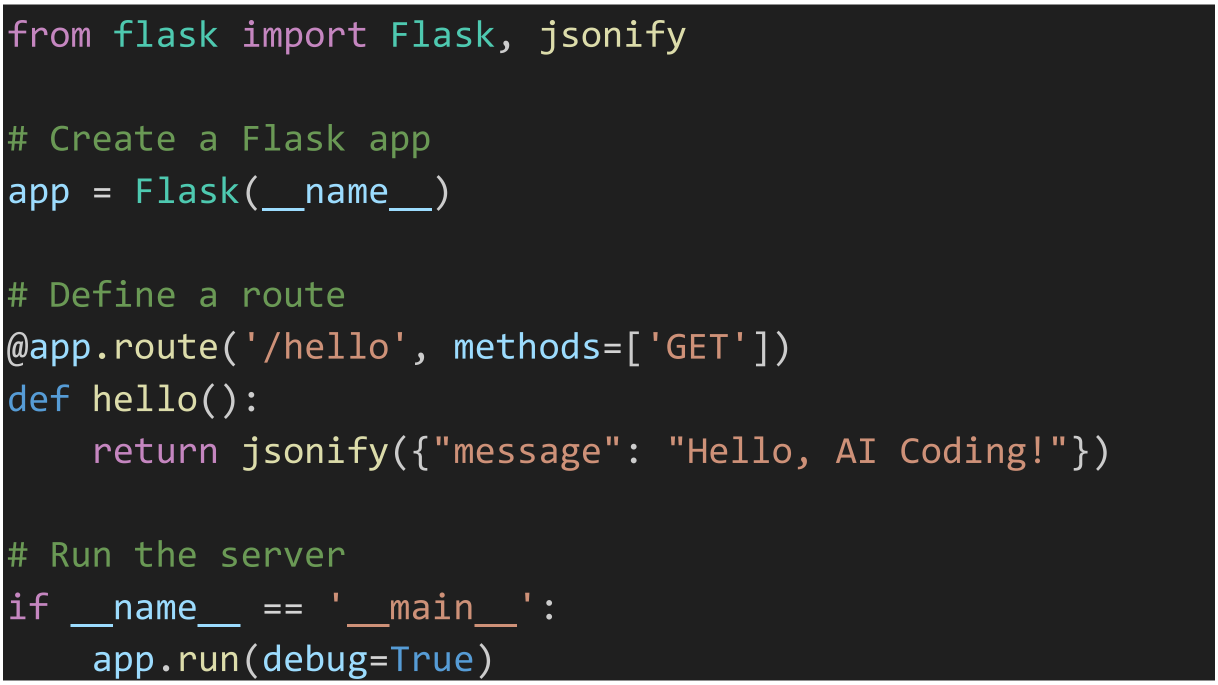
Hall Tno.:2403A510G3

Task Description #1 – Basic REST API Setup

Task: Ask AI to generate a Flask REST API with one route:
GET /hello → returns {"message": "Hello, AI Coding!"}

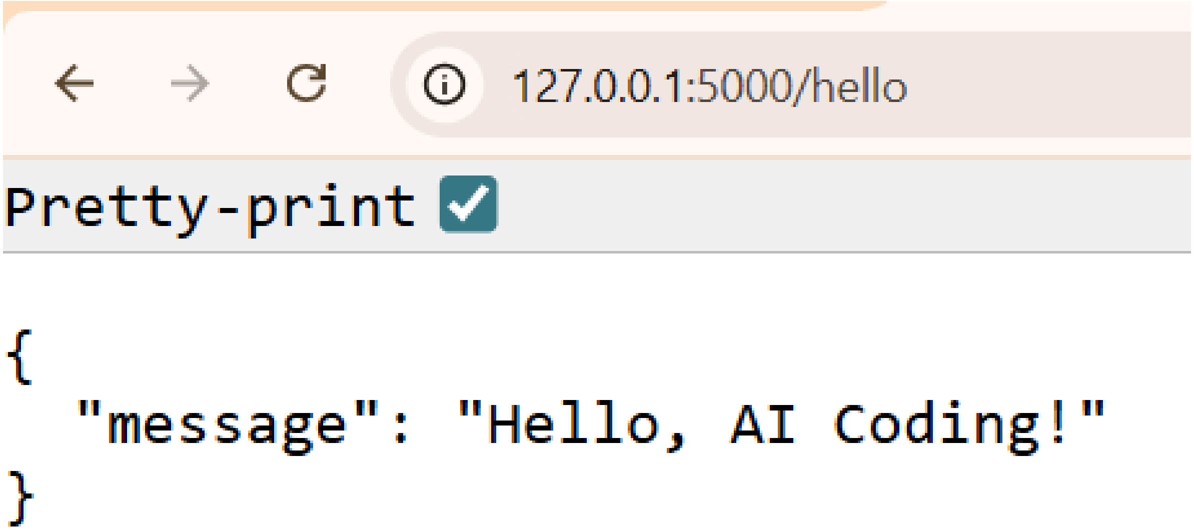
PROMPT : to generate a Flask REST API with one route:
GET /hello → returns {"message": "Hello, AI Coding!"}

CODE :



```python
from flask import Flask, jsonify

# Create a Flask app
app = Flask(__name__)

# Define a route
@app.route('/hello', methods=['GET'])
def hello():
    return jsonify({"message": "Hello, AI Coding!"})

# Run the server
if __name__ == '__main__':
    app.run(debug=True)
```

OUTPUT :



```
{
  "message": "Hello, AI Coding!"
}
```

OBSERVATION :

The jsonify() function is used to convert the data into proper JSON. When the program runs, it starts a local server on port 5000, and the message can be viewed using a web browser, Postman, or Curl. This code demonstrates the basic working of a RESTful API using Flask in a clear and easy way.

Task Description #2 – CRUD Operations (Students API)

Task:

Use AI to build REST endpoints for a Student API:

- GET /students → List all students.
- POST /students → Add a new student.
- PUT /students/<id> → Update student details.
- DELETE /students/<id> → Delete a student.

Expected Output:

- Flask API with dictionary/list storage.
- JSON responses for each operation.

PROMPT : build REST endpoints for a Student API:

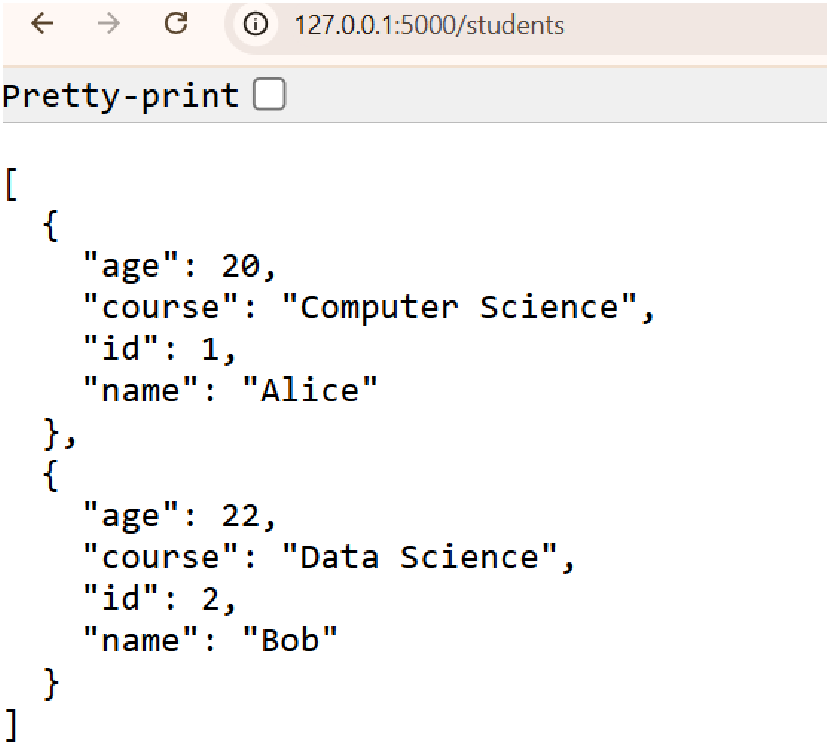
CODE :

```
from flask import Flask, jsonify, request

app = Flask(__name__)

# Sample data (acts as a simple in-memory database)
students = [
    {"id": 1, "name": "Alice", "age": 20, "course": "Computer Science"},
    {"id": 2, "name": "Bob", "age": 22, "course": "Data Science"}
]
next_id = 3

# -------------------------
# 1️⃣GET all students
# -------------------------
@app.route('/students', methods=['GET'])
```

```
def get_students():
    """Returns a list of all students."""
    return jsonify(students)

# -------------------------
# 2️⃣GET a student by ID
# -------------------------
@app.route('/students/<int:id>', methods=['GET'])
def get_student(id):
    """Returns a single student by their ID."""
    student = next((s for s in students if s["id"] ==
id), None)
    if student:
        return jsonify(student)
    return jsonify({"message": "Student not found"}),
404

# -------------------------
# 3️⃣POST (Add new student)
# -------------------------
@app.route('/students', methods=['POST'])
def add_student():
    """Adds a new student to the list."""
    global next_id
    if not request.json or not 'name' in request.json or
not 'course' in request.json:
        return jsonify({"message": "Bad request, 'name'
and 'course' are required"}), 400

    new_student = {
        "id": next_id,
        "name": request.json['name'],
        "age": request.json.get('age', None),
        "course": request.json['course']
    }
```

```python
    students.append(new_student)
    next_id += 1
    return jsonify(new_student), 201

# -------------------------
# 4️⃣ PUT (Update student)
# -------------------------
@app.route('/students/<int:id>', methods=['PUT'])
def update_student(id):
    """Updates an existing student's details."""
    student = next((s for s in students if s["id"] == id), None)
    if not student:
        return jsonify({"message": "Student not found"}), 404

    if not request.json:
        return jsonify({"message": "Bad request"}), 400

    student['name'] = request.json.get('name', student['name'])
    student['age'] = request.json.get('age', student['age'])
    student['course'] = request.json.get('course', student['course'])

    return jsonify(student)

# -------------------------
# 5️⃣ DELETE (Remove student)
# -------------------------
@app.route('/students/<int:id>', methods=['DELETE'])
def delete_student(id):
    """Deletes a student from the list."""
    global students
    student = next((s for s in students if s["id"] == id), None)
    if not student:
        return jsonify({"message": "Student not found"}), 404

    students = [s for s in students if s["id"] != id]
    return jsonify({"message": "Student deleted successfully"})

# -------------------------
# Run the app
# -------------------------
if __name__ == '__main__':
    app.run(debug=True)
```

OUTPUT :



```
[
  {
    "age": 20,
    "course": "Computer Science",
    "id": 1,
    "name": "Alice"
  },
  {
    "age": 22,
    "course": "Data Science",
    "id": 2,
    "name": "Bob"
  }
]
```

OBSERVATION :

The API returns data in JSON format, which can be tested using a browser or Postman. When the code is run, a local server starts on port 5000, allowing users to interact with student data easily. This program demonstrates the basic concept of backend API development

Task Description #3 – API with Query Parameters

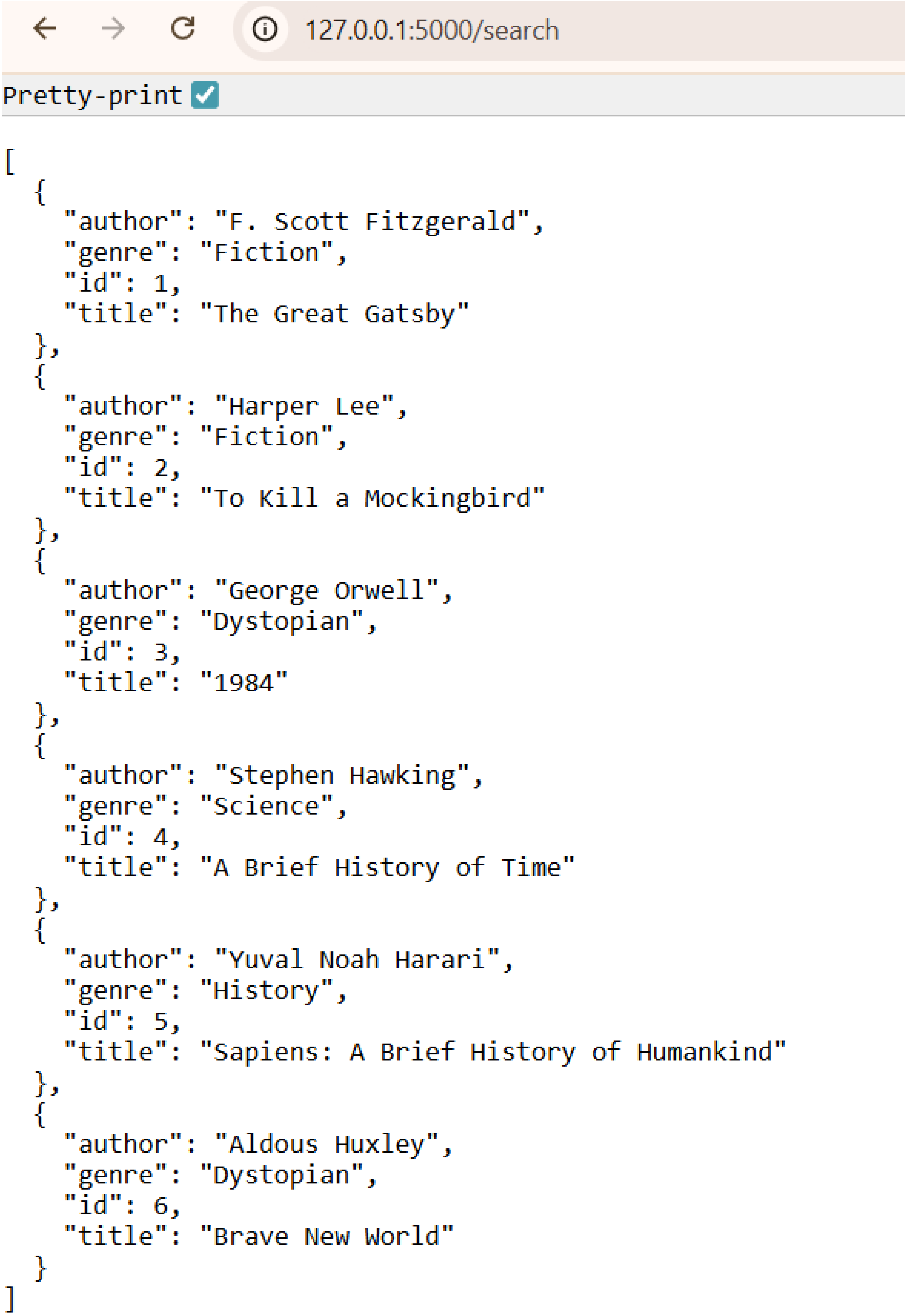
Task: Ask AI to generate a REST API endpoint
Expected Output:
Working search function with query param handling.

PROMPT : generate a REST API endpoint

CODE :

```
from flask import Flask, jsonify, request

app = Flask(__name__)

# Sample data: A list of books to search through
books = [
    {"id": 1, "title": "The Great Gatsby", "author": "F. Scott Fitzgerald", "genre": "Fiction"},
    {"id": 2, "title": "To Kill a Mockingbird", "author": "Harper Lee", "genre": "Fiction"},
    {"id": 3, "title": "1984", "author": "George Orwell", "genre": "Dystopian"},
    {"id": 4, "title": "A Brief History of Time",
"author": "Stephen Hawking", "genre": "Science"},
    {"id": 5, "title": "Sapiens: A Brief History of
Humankind", "author": "Yuval Noah Harari", "genre":
"History"},
    {"id": 6, "title": "Brave New World", "author":
"Aldous Huxley", "genre": "Dystopian"}
]

@app.route('/search', methods=['GET'])
def search_books():
    """
    Searches for books based on query parameters.
    Example Usage:
    - /search?author=George Orwell
    - /search?genre=Fiction
    - /search?title=History
    - /search?genre=Dystopian&author=Aldous Huxley
    """
    # Get query parameters from the request URL
    query_author = request.args.get('author')
    query_genre = request.args.get('genre')
    query_title = request.args.get('title')

    # Start with the full list of books
    results = books

    # Filter by author if the parameter is provided
    if query_author:
        results = [book for book in results if
book['author'].lower() == query_author.lower()]

    # Filter by genre if the parameter is provided
    if query_genre:
        results = [book for book in results if
book['genre'].lower() == query_genre.lower()]

    # Filter by title (contains) if the parameter is
provided
    if query_title:
        results = [book for book in results if
query_title.lower() in book['title'].lower()]

    # Return the filtered list as JSON
    return jsonify(results)

if __name__ == '__main__':
    # To run:
    # 1. pip install Flask
    # 2. python ai_lab_15_3.py
    # 3. Open your browser and go to an address like:
    #    http://127.0.0.1:5000/search?genre=Dystopian
    app.run(debug=True)
```

OUTPUT :



127.0.0.1:5000/search

Pretty-print ☑

```
[
  {
    "author": "F. Scott Fitzgerald",
    "genre": "Fiction",
    "id": 1,
    "title": "The Great Gatsby"
  },
  {
    "author": "Harper Lee",
    "genre": "Fiction",
    "id": 2,
    "title": "To Kill a Mockingbird"
  },
  {
    "author": "George Orwell",
    "genre": "Dystopian",
    "id": 3,
    "title": "1984"
  },
  {
    "author": "Stephen Hawking",
    "genre": "Science",
    "id": 4,
    "title": "A Brief History of Time"
  },
  {
    "author": "Yuval Noah Harari",
    "genre": "History",
    "id": 5,
    "title": "Sapiens: A Brief History of Humankind"
  },
  {
    "author": "Aldous Huxley",
    "genre": "Dystopian",
    "id": 6,
    "title": "Brave New World"
  }
]
```

OBSERVATION :

The code efficiently handles these optional parameters by starting with the full list of books and progressively filtering it down if a parameter is present. The final list of matching books is then returned in JSON format, a standard for web APIs. The script includes sample data and comments explaining how to run the server and test the search functionality.

Task Description #4 – Integration & Testing

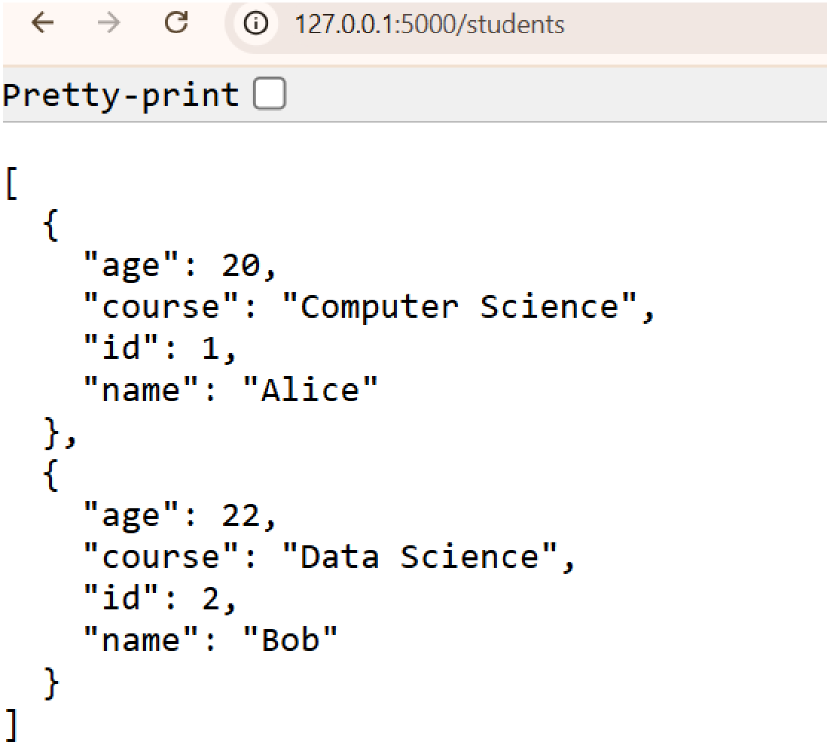
Task: Ask AI to write test scripts using Python requests module to call APIs created above.

PROMPT : write test scripts using Python requests module to call APIs created above.

CODE :

```
import requests
import json

# The base URL of your running Flask application
BASE_URL = "http://127.0.0.1:5000"

def test_student_api():
    """
    Tests the CRUD operations for the /students
endpoint.
    """
    print("--- □ Testing Student API ---")
    # 1. GET all students
    print("\n1. GET /students (Initial List)")
    try:
        response = requests.get(f"{BASE_URL}/students")
        print(f"Status Code: {response.status_code}")
        print("Response JSON:", response.json())
    except requests.exceptions.ConnectionError as e:
        print(f"Connection Error: Please ensure the
Flask API server is running. Details: {e}")
        return # Stop testing if the server is not
running

    # 2. POST a new student
    print("\n2. POST /students (Add a new student)")
    new_student_data = {"name": "Charlie", "age": 21,
"course": "Physics"}
    response = requests.post(f"{BASE_URL}/students",
json=new_student_data)
    print(f"Status Code: {response.status_code}")
    added_student = response.json()
    print("Response JSON:", added_student)
    student_id = added_student.get("id")

    # 3. PUT (update) the new student's details
    print(f"\n3. PUT /students/{student_id} (Update
student's course)")
    update_data = {"course": "Astrophysics"}
    response =
requests.put(f"{BASE_URL}/students/{student_id}",
json=update_data)
    print(f"Status Code: {response.status_code}")
    print("Response JSON:", response.json())

    # 4. GET the updated student to verify the change
    print(f"\n4. GET /students/{student_id} (Verify
update)")
    response =
requests.get(f"{BASE_URL}/students/{student_id}")
    print(f"Status Code: {response.status_code}")
    print("Response JSON:", response.json())

    # 5. DELETE the student
    print(f"\n5. DELETE /students/{student_id} (Remove
student)")
    response =
requests.delete(f"{BASE_URL}/students/{student_id}")
    print(f"Status Code: {response.status_code}")
    print("Response JSON:", response.json())

    # 6. GET all students again to confirm deletion
    print("\n6. GET /students (Final List)")
    response = requests.get(f"{BASE_URL}/students")
    print(f"Status Code: {response.status_code}")
    print("Response JSON:", response.json())
    print("-" * 30)

def test_book_search_api():
    """
    Tests the query parameter functionality of the
/search endpoint.
    """
    print("\n--- □ Testing Book Search API ---")

    # 1. Search by genre
    print("\n1. GET /search?genre=Dystopian")
    try:
        response = requests.get(f"{BASE_URL}/search",
params={"genre": "Dystopian"})
        print(f"Status Code: {response.status_code}")
        print("Response JSON:", response.json())
    except requests.exceptions.ConnectionError as e:
        print(f"Connection Error: Please ensure the Flask API server is running. Details: {e}")
        return

    # 2. Search by author
    print("\n2. GET /search?author=Harper Lee")
    response = requests.get(f"{BASE_URL}/search", params={"author": "Harper Lee"})
    print(f"Status Code: {response.status_code}")
    print("Response JSON:", response.json())

    # 3. Search with no results
    print("\n3. GET /search?genre=Comedy (No results expected)")
    response = requests.get(f"{BASE_URL}/search", params={"genre": "Comedy"})
    print(f"Status Code: {response.status_code}")
    print("Response JSON:", response.json())
    print("-" * 30)


if __name__ == "__main__":
    # --- How to Run ---
    # 1. Make sure your Flask API (e.g., ai_lab_15_2.py or ai_lab_15_3.py) is running in a separate terminal.
    # 2. Run this script in another terminal: python ai_lab_15_4.py
    # 3. You will need to install the 'requests' library first: pip install requests

    test_student_api()
    test_book_search_api()
```

OUTPUT :



Pretty-print ☐

```
[
  {
    "age": 20,
    "course": "Computer Science",
    "id": 1,
    "name": "Alice"
  },
  {
    "age": 22,
    "course": "Data Science",
    "id": 2,
    "name": "Bob"
  }
]
```

OBSERVATION :

this Python script is a simple web API built using the Flask framework. It combines two separate functionalities into a single server:

A Student API: Provides full CRUD (Create, Read, Update, Delete) operations for managing a list of students.

The script uses simple Python lists to store book data, acting as an in-memory database. When run, starts a local web server, making these API endpoints accessible.