AI-Assisted Coding

Lab Assignment-15.3

Name: Neha Shazneen
Roll no: 2403A510A1
Batch no:05
CSE 2nd Year

Task Description #1 – Basic REST API Setup

Task: Ask AI to generate a Flask REST API with one route:

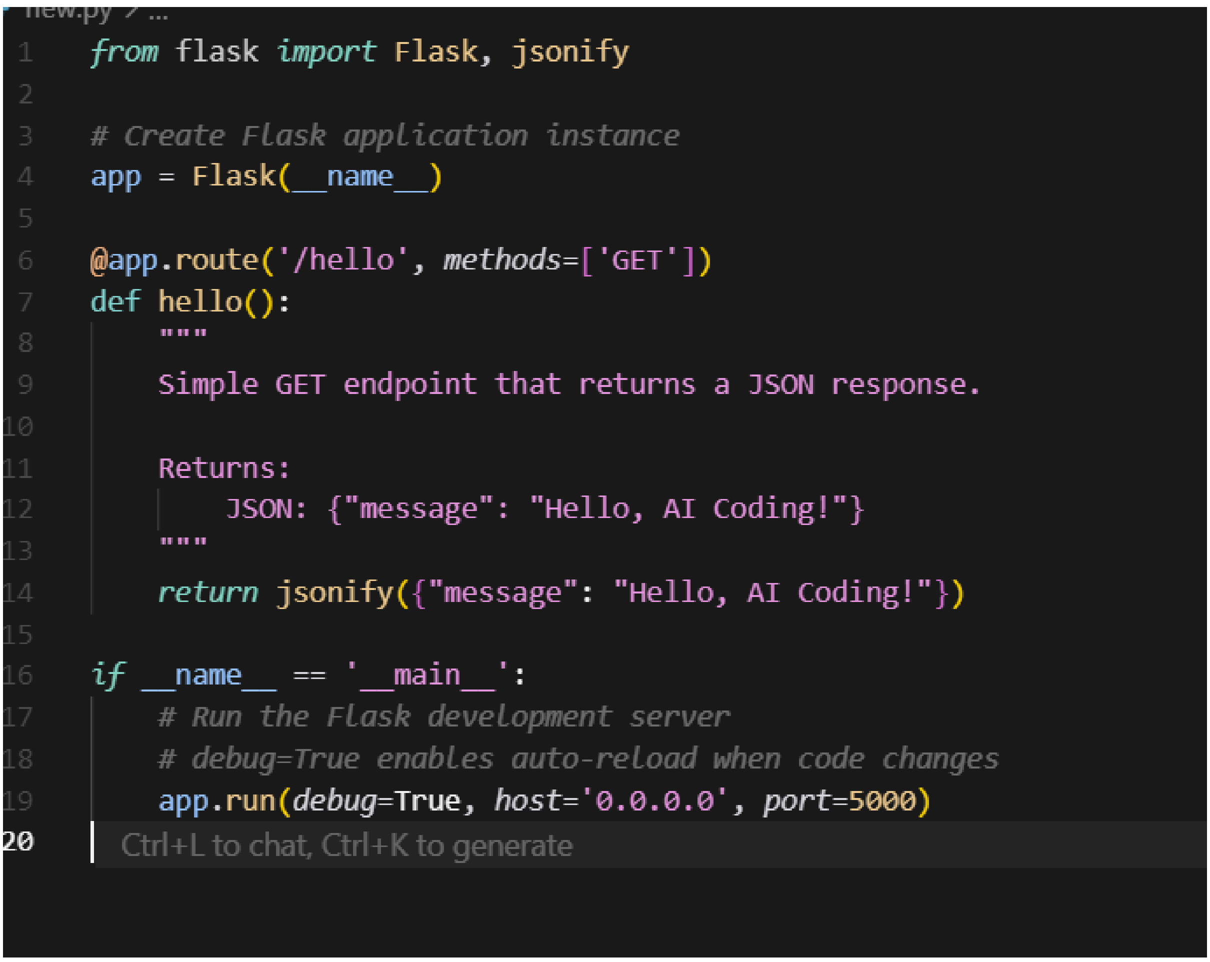
GET /hello → returns {"message": "Hello, AI Coding!"}

Prompt:

Create a simple REST API using Flask with a single route:

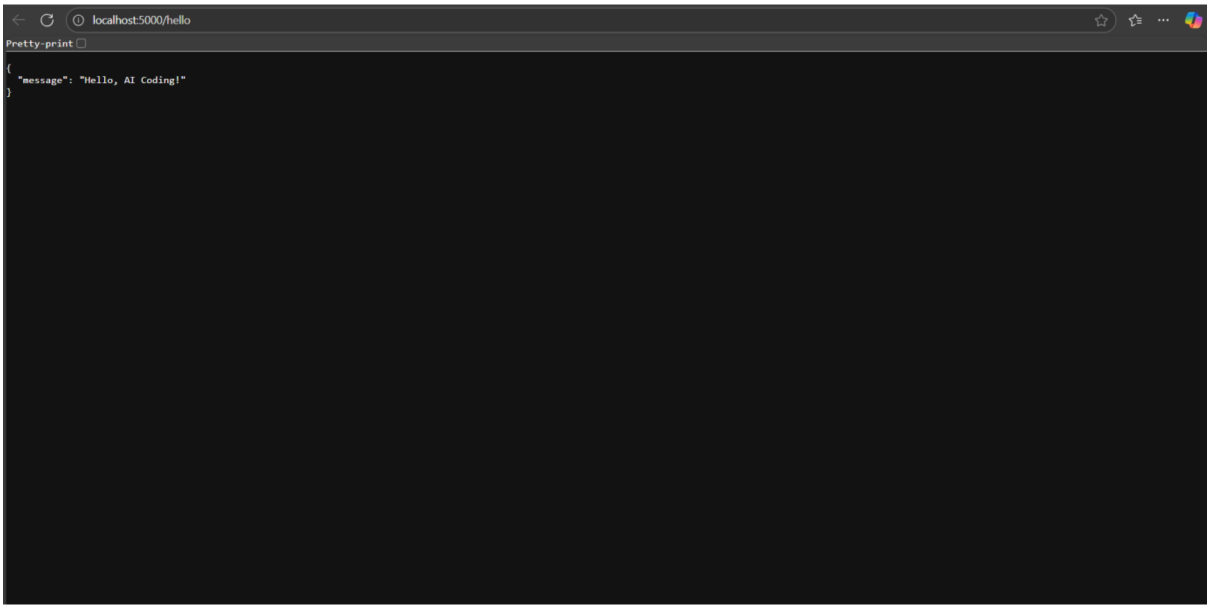
- GET /hello → returns a JSON response: {"message": "Hello, AI Coding!"}.
  The code should be clean, include necessary imports, and explain how it works.

Code Generated:



```python
from flask import Flask, jsonify

# Create Flask application instance
app = Flask(__name__)

@app.route('/hello', methods=['GET'])
def hello():
    """
    Simple GET endpoint that returns a JSON response.

    Returns:
        JSON: {"message": "Hello, AI Coding!"}
    """
    return jsonify({"message": "Hello, AI Coding!"})

if __name__ == '__main__':
    # Run the Flask development server
    # debug=True enables auto-reload when code changes
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Output:



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

Observation:

- The code creates a Flask web application with one route /hello.
- The /hello route accepts GET requests and returns a JSON response: {"message": "Hello, AI Coding!"}.
- The Flask app runs on all network interfaces (0.0.0.0) at port 5000.
- Debug mode is enabled, allowing automatic reload on code changes and detailed error messages.
- The response is generated using Flask’s jsonify function.

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.

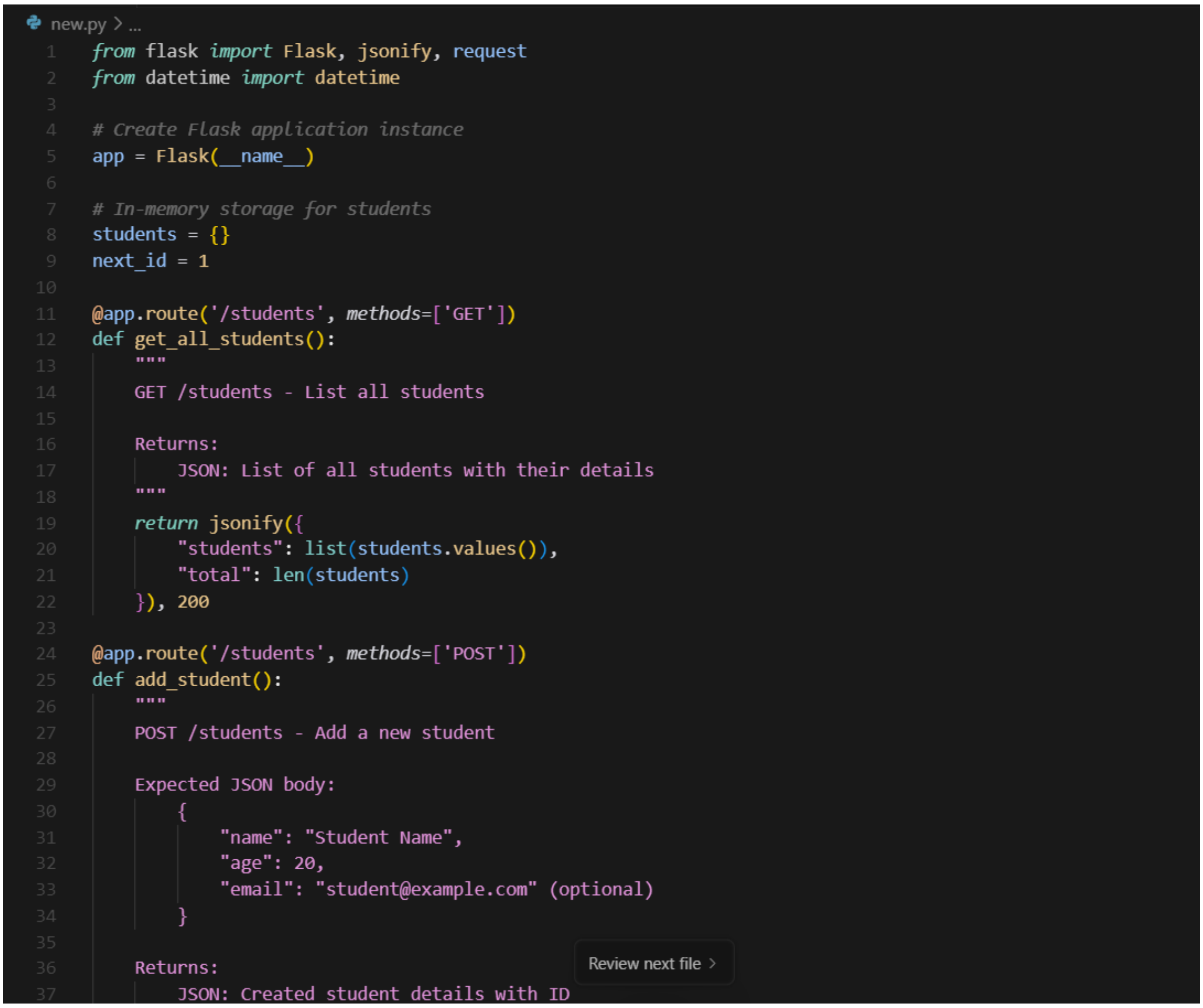
Prompt:

Build a Flask REST API for managing students with the following endpoints:

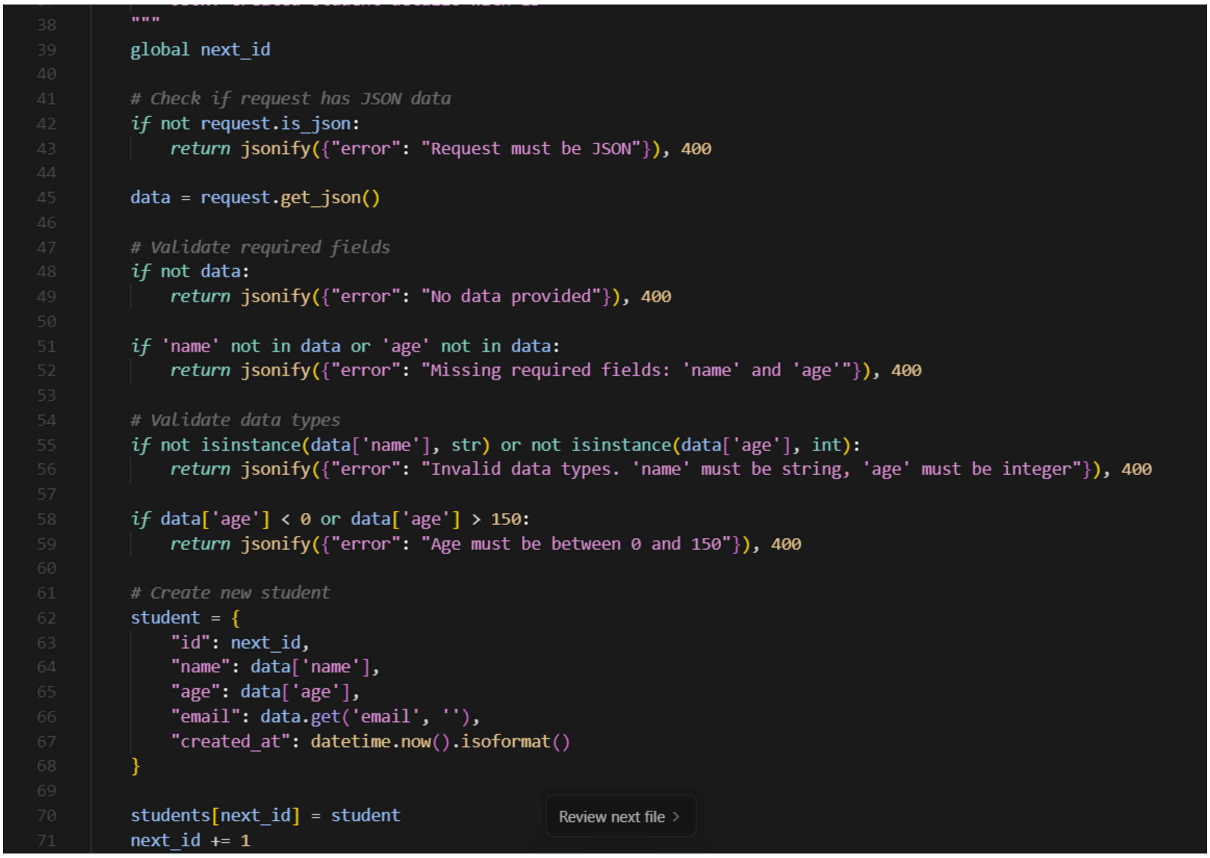
- GET /students: List all students.
- POST /students: Add a new student. The request body will contain JSON with student details (e.g., name and age).
- PUT /students/<id>: Update the details of a student identified by `id`.
- DELETE /students/<id>: Delete the student with the given `id`.

Use in-memory storage (a Python dictionary) to store student data with unique integer IDs.
Return appropriate JSON responses and HTTP status codes.
Handle errors like missing data or invalid student IDs.

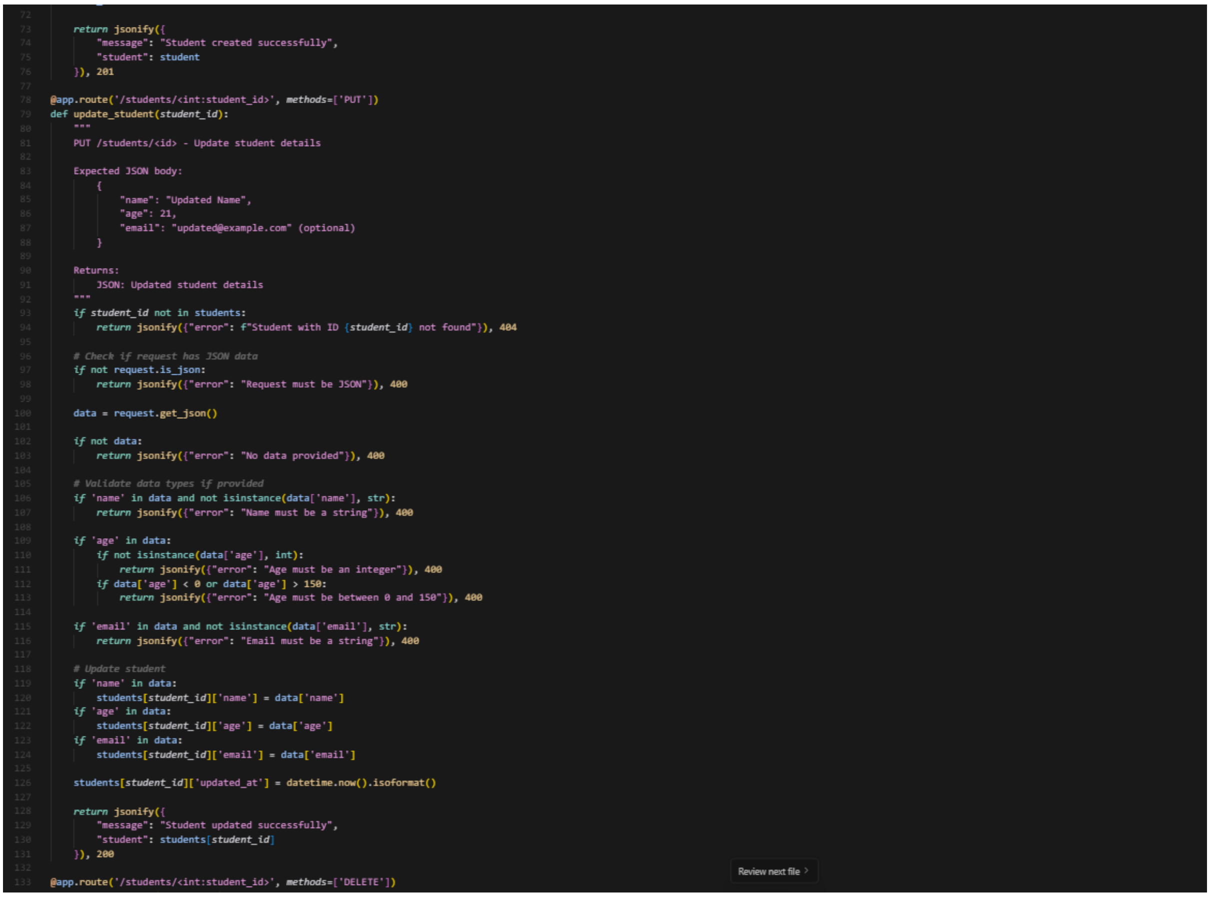
Code Generated:



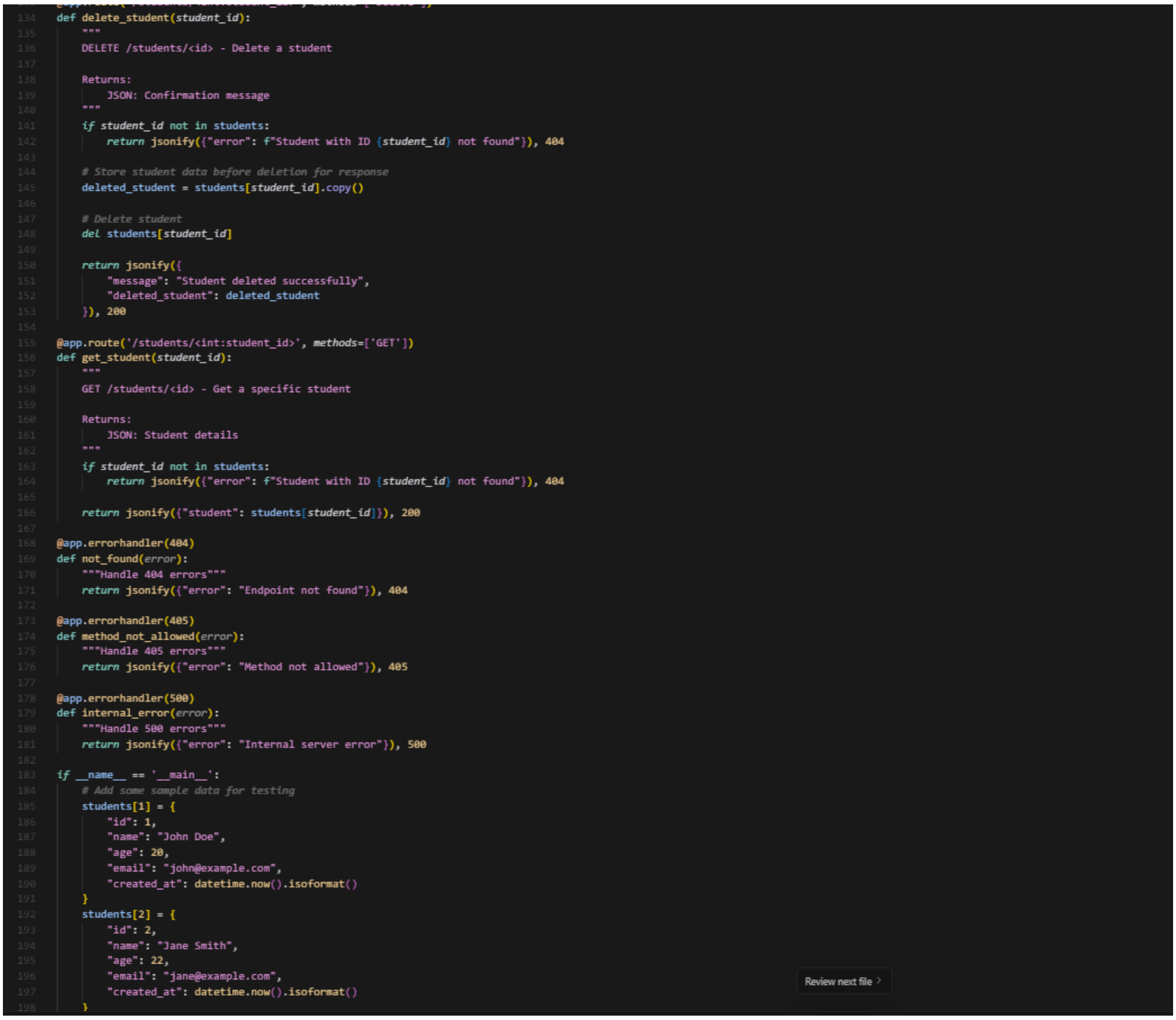
```python
from flask import Flask, jsonify, request
from datetime import datetime

# Create Flask application instance
app = Flask(__name__)

# In-memory storage for students
students = {}
next_id = 1

@app.route('/students', methods=['GET'])
def get_all_students():
    """
    GET /students - List all students

    Returns:
        JSON: List of all students with their details
    """
    return jsonify({
        "students": list(students.values()),
        "total": len(students)
    }), 200

@app.route('/students', methods=['POST'])
def add_student():
    """
    POST /students - Add a new student

    Expected JSON body:
        {
            "name": "Student Name",
            "age": 20,
            "email": "student@example.com" (optional)
        }

    Returns:
        JSON: Created student details with ID
```



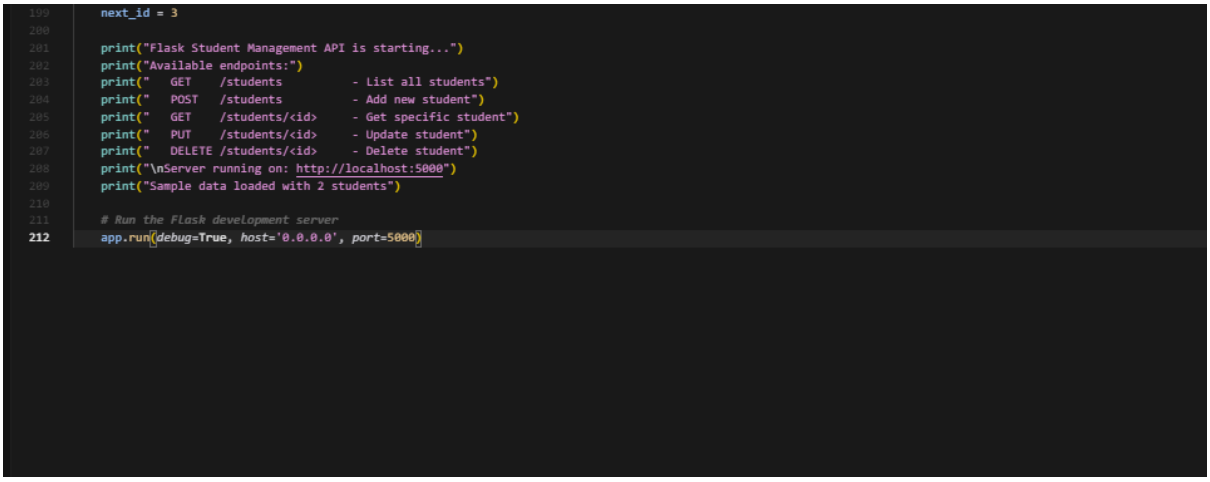
```
    """
    global next_id

    # Check if request has JSON data
    if not request.is_json:
        return jsonify({"error": "Request must be JSON"}), 400

    data = request.get_json()

    # Validate required fields
    if not data:
        return jsonify({"error": "No data provided"}), 400

    if 'name' not in data or 'age' not in data:
        return jsonify({"error": "Missing required fields: 'name' and 'age'"}), 400

    # Validate data types
    if not isinstance(data['name'], str) or not isinstance(data['age'], int):
        return jsonify({"error": "Invalid data types. 'name' must be string, 'age' must be integer"}), 400

    if data['age'] < 0 or data['age'] > 150:
        return jsonify({"error": "Age must be between 0 and 150"}), 400

    # Create new student
    student = {
        "id": next_id,
        "name": data['name'],
        "age": data['age'],
        "email": data.get('email', ''),
        "created_at": datetime.now().isoformat()
    }

    students[next_id] = student
    next_id += 1
```



```python

    return jsonify({
        "message": "Student created successfully",
        "student": student
    }), 201

@app.route('/students/<int:student_id>', methods=['PUT'])
def update_student(student_id):
    """
    PUT /students/<id> - Update student details

    Expected JSON body:
        {
            "name": "Updated Name",
            "age": 21,
            "email": "updated@example.com" (optional)
        }

    Returns:
        JSON: Updated student details
    """
    if student_id not in students:
        return jsonify({"error": f"Student with ID {student_id} not found"}), 404

    # Check if request has JSON data
    if not request.is_json:
        return jsonify({"error": "Request must be JSON"}), 400

    data = request.get_json()

    if not data:
        return jsonify({"error": "No data provided"}), 400

    # Validate data types if provided
    if 'name' in data and not isinstance(data['name'], str):
        return jsonify({"error": "Name must be a string"}), 400

    if 'age' in data:
        if not isinstance(data['age'], int):
            return jsonify({"error": "Age must be an integer"}), 400
        if data['age'] < 0 or data['age'] > 150:
            return jsonify({"error": "Age must be between 0 and 150"}), 400

    if 'email' in data and not isinstance(data['email'], str):
        return jsonify({"error": "Email must be a string"}), 400

    # Update student
    if 'name' in data:
        students[student_id]['name'] = data['name']
    if 'age' in data:
        students[student_id]['age'] = data['age']
    if 'email' in data:
        students[student_id]['email'] = data['email']

    students[student_id]['updated_at'] = datetime.now().isoformat()

    return jsonify({
        "message": "Student updated successfully",
        "student": students[student_id]
    }), 200

@app.route('/students/<int:student_id>', methods=['DELETE'])
```

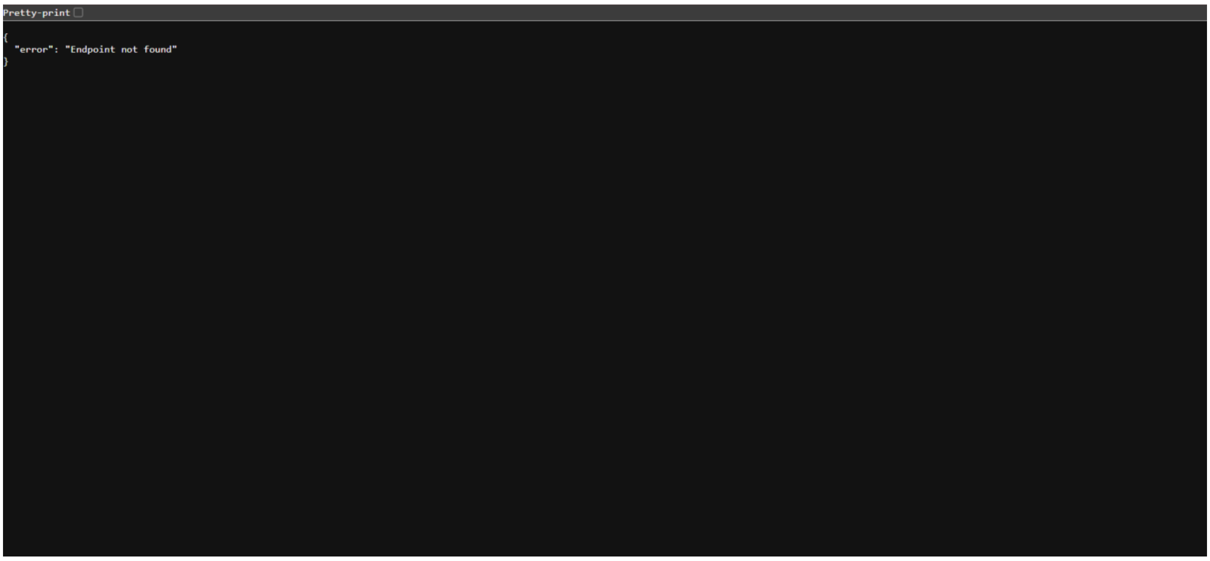


```python
def delete_student(student_id):
    """
    DELETE /students/<id> - Delete a student

    Returns:
        JSON: Confirmation message
    """
    if student_id not in students:
        return jsonify({"error": f"Student with ID {student_id} not found"}), 404

    # Store student data before deletion for response
    deleted_student = students[student_id].copy()

    # Delete student
    del students[student_id]

    return jsonify({
        "message": "Student deleted successfully",
        "deleted_student": deleted_student
    }), 200

@app.route('/students/<int:student_id>', methods=['GET'])
def get_student(student_id):
    """
    GET /students/<id> - Get a specific student

    Returns:
        JSON: Student details
    """
    if student_id not in students:
        return jsonify({"error": f"Student with ID {student_id} not found"}), 404

    return jsonify({"student": students[student_id]}), 200

@app.errorhandler(404)
def not_found(error):
    """Handle 404 errors"""
    return jsonify({"error": "Endpoint not found"}), 404

@app.errorhandler(405)
def method_not_allowed(error):
    """Handle 405 errors"""
    return jsonify({"error": "Method not allowed"}), 405

@app.errorhandler(500)
def internal_error(error):
    """Handle 500 errors"""
    return jsonify({"error": "Internal server error"}), 500

if __name__ == '__main__':
    # Add some sample data for testing
    students[1] = {
        "id": 1,
        "name": "John Doe",
        "age": 20,
        "email": "john@example.com",
        "created_at": datetime.now().isoformat()
    }
    students[2] = {
        "id": 2,
        "name": "Jane Smith",
        "age": 22,
        "email": "jane@example.com",
        "created_at": datetime.now().isoformat()
    }
```



```python
    next_id = 3

    print("Flask Student Management API is starting...")
    print("Available endpoints:")
    print("   GET    /students          - List all students")
    print("   POST   /students          - Add new student")
    print("   GET    /students/<id>     - Get specific student")
    print("   PUT    /students/<id>     - Update student")
    print("   DELETE /students/<id>     - Delete student")
    print("\nServer running on: http://localhost:5000")
    print("Sample data loaded with 2 students")

    # Run the Flask development server
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Output:



```
{
  "error": "Endpoint not found"
}
```

Observation:

- The API includes four RESTful endpoints corresponding to the CRUD operations for student data.
- Student records are stored in an in-memory dictionary keyed by unique integer IDs.
- GET /students returns a JSON list of all stored students.
- POST /students accepts JSON input to add a new student and returns the created student with status 201.
- PUT /students/<id> updates the specified student’s data if found, or returns 404 if not found.
- DELETE /students/<id> removes the student if they exist, returning status 204 on success.
- Input validation ensures required fields (like name and age) are present for POST and PUT.
- Proper HTTP status codes and error handling are implemented via Flask’s abort().
- The API uses JSON for both input and output consistently.
- The code runs in debug mode suitable for development.

Task Description #3 – API with Query Parameters

Task: Ask AI to generate a REST API endpoint

Prompt:

Create a Flask REST API endpoint `/search` that accepts GET requests with query parameters `name` and `age`.

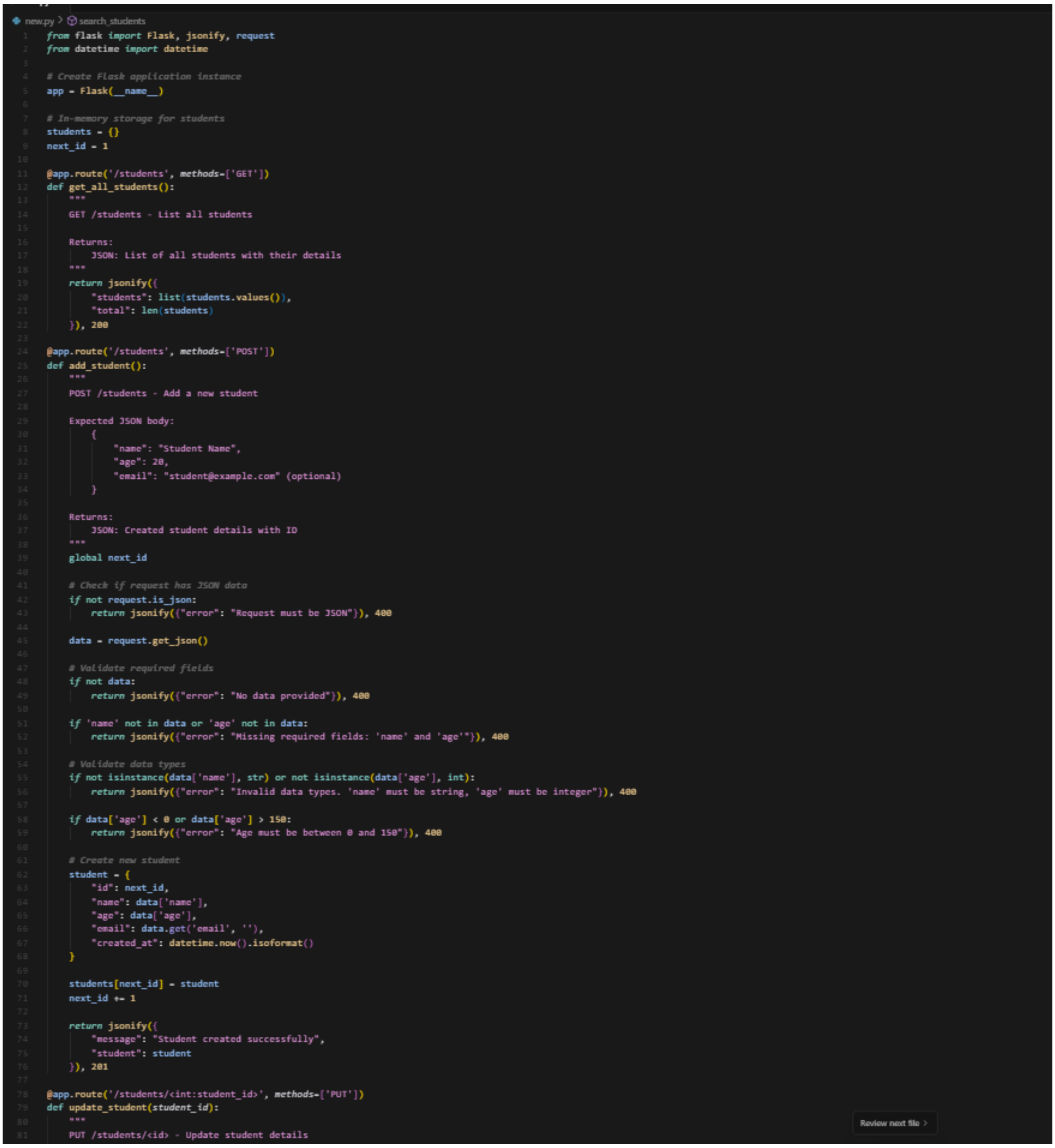
The endpoint should filter a list of students stored in memory based on the provided query parameters:

- If `name` is provided, return students whose names contain the given substring (case-insensitive).
- If `age` is provided, return students matching the given age.

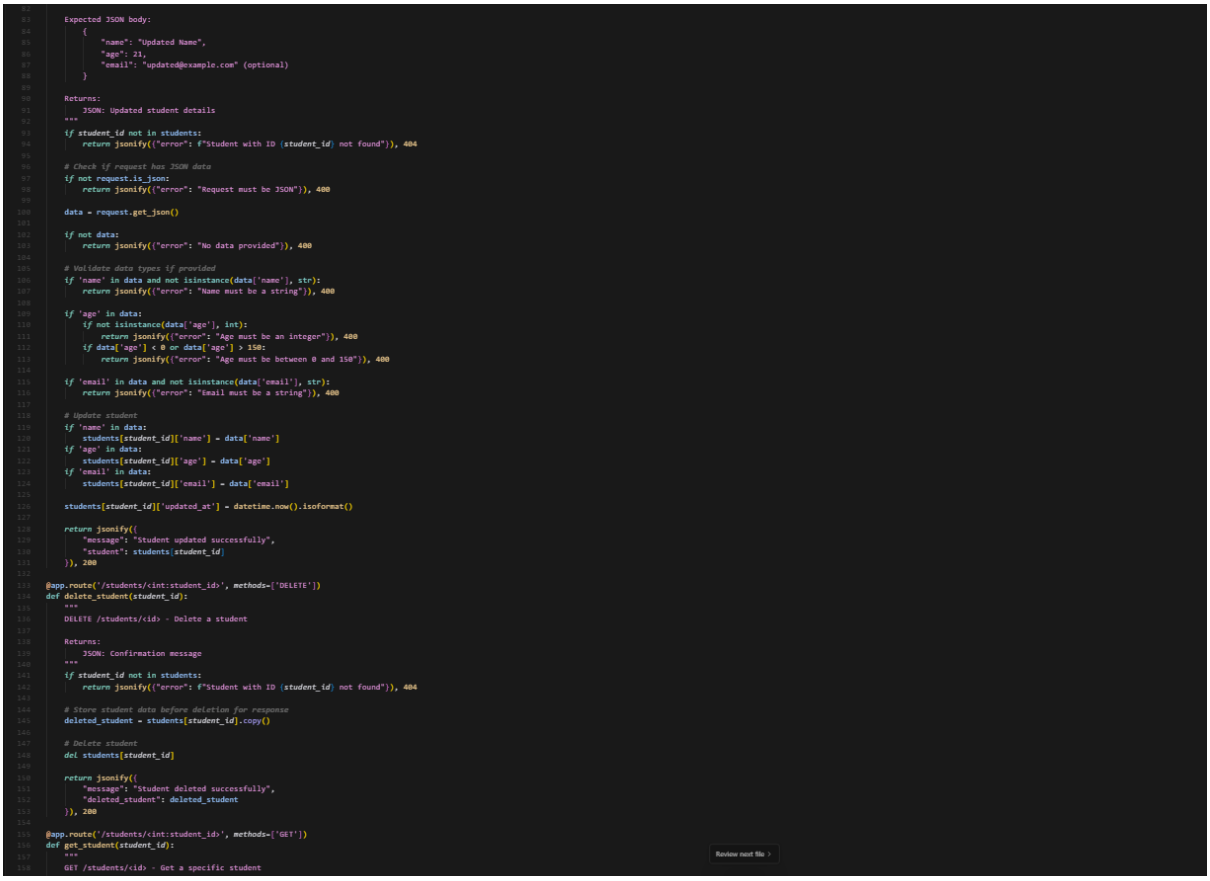
- If both parameters are provided, filter students matching both criteria.
- If no query parameters are provided, return all students.

Return the filtered list of students as JSON.

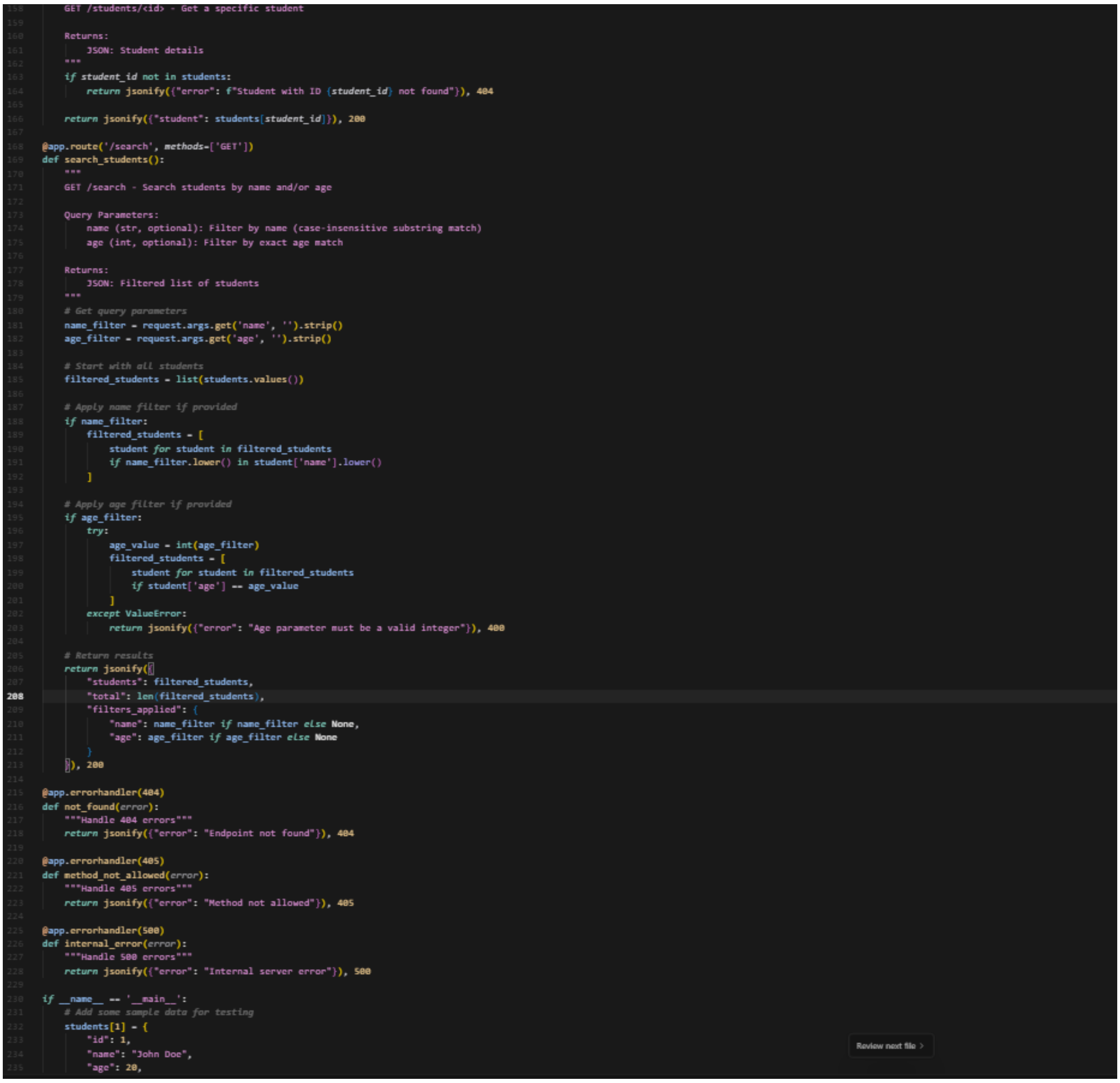
Code Generated:



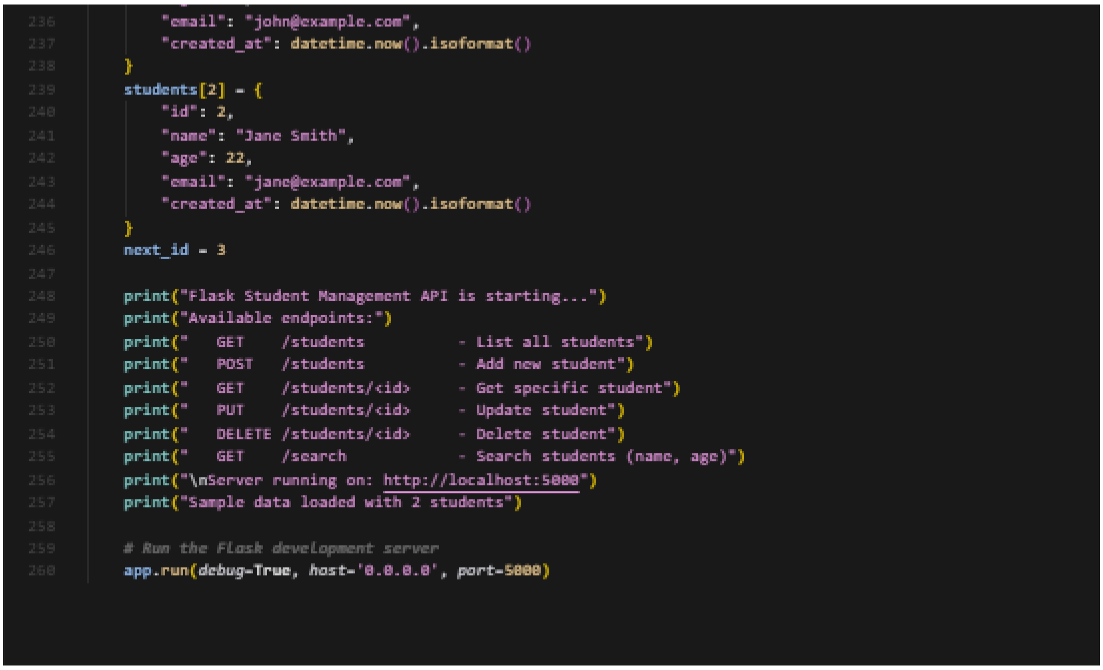
```python
from flask import Flask, jsonify, request
from datetime import datetime

# Create Flask application instance
app = Flask(__name__)

# In-memory storage for students
students = {}
next_id = 1

@app.route('/students', methods=['GET'])
def get_all_students():
    """
    GET /students - List all students

    Returns:
        JSON: List of all students with their details
    """
    return jsonify({
        "students": list(students.values()),
        "total": len(students)
    }), 200

@app.route('/students', methods=['POST'])
def add_student():
    """
    POST /students - Add a new student

    Expected JSON body:
        {
            "name": "Student Name",
            "age": 20,
            "email": "student@example.com" (optional)
        }

    Returns:
        JSON: Created student details with ID
    """
    global next_id

    # Check if request has JSON data
    if not request.is_json:
        return jsonify({"error": "Request must be JSON"}), 400

    data = request.get_json()

    # Validate required fields
    if not data:
        return jsonify({"error": "No data provided"}), 400

    if 'name' not in data or 'age' not in data:
        return jsonify({"error": "Missing required fields: 'name' and 'age'"}), 400

    # Validate data types
    if not isinstance(data['name'], str) or not isinstance(data['age'], int):
        return jsonify({"error": "Invalid data types. 'name' must be string, 'age' must be integer"}), 400

    if data['age'] < 0 or data['age'] > 150:
        return jsonify({"error": "Age must be between 0 and 150"}), 400

    # Create new student
    student = {
        "id": next_id,
        "name": data['name'],
        "age": data['age'],
        "email": data.get('email', ''),
        "created_at": datetime.now().isoformat()
    }

    students[next_id] = student
    next_id += 1

    return jsonify({
        "message": "Student created successfully",
        "student": student
    }), 201

@app.route('/students/<int:student_id>', methods=['PUT'])
def update_student(student_id):
    """
    PUT /students/<id> - Update student details
```



```
    Expected JSON body:
        {
            "name": "Updated Name",
            "age": 21,
            "email": "updated@example.com" (optional)
        }

    Returns:
        JSON: Updated student details
    """
    if student_id not in students:
        return jsonify({"error": f"Student with ID {student_id} not found"}), 404

    # Check if request has JSON data
    if not request.is_json:
        return jsonify({"error": "Request must be JSON"}), 400

    data = request.get_json()

    if not data:
        return jsonify({"error": "No data provided"}), 400

    # Validate data types if provided
    if 'name' in data and not isinstance(data['name'], str):
        return jsonify({"error": "Name must be a string"}), 400

    if 'age' in data:
        if not isinstance(data['age'], int):
            return jsonify({"error": "Age must be an integer"}), 400
        if data['age'] < 0 or data['age'] > 150:
            return jsonify({"error": "Age must be between 0 and 150"}), 400

    if 'email' in data and not isinstance(data['email'], str):
        return jsonify({"error": "Email must be a string"}), 400

    # Update student
    if 'name' in data:
        students[student_id]['name'] = data['name']
    if 'age' in data:
        students[student_id]['age'] = data['age']
    if 'email' in data:
        students[student_id]['email'] = data['email']

    students[student_id]['updated_at'] = datetime.now().isoformat()

    return jsonify({
        "message": "Student updated successfully",
        "student": students[student_id]
    }), 200

@app.route('/students/<int:student_id>', methods=['DELETE'])
def delete_student(student_id):
    """
    DELETE /students/<id> - Delete a student

    Returns:
        JSON: Confirmation message
    """
    if student_id not in students:
        return jsonify({"error": f"Student with ID {student_id} not found"}), 404

    # Store student data before deletion for response
    deleted_student = students[student_id].copy()

    # Delete student
    del students[student_id]

    return jsonify({
        "message": "Student deleted successfully",
        "deleted_student": deleted_student
    }), 200

@app.route('/students/<int:student_id>', methods=['GET'])
def get_student(student_id):
    """
    GET /students/<id> - Get a specific student
```

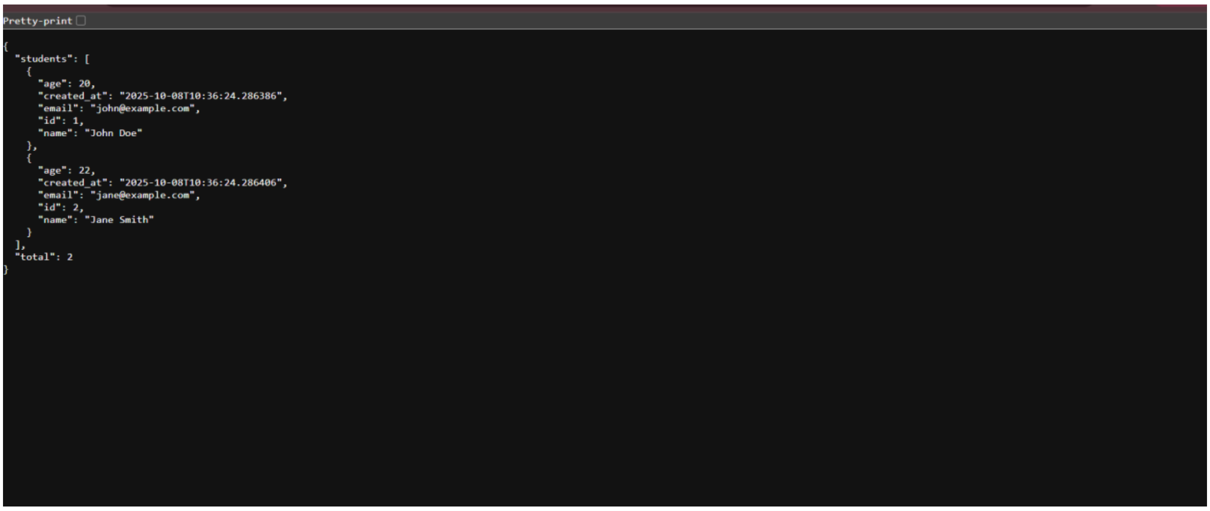


```python
    GET /students/<id> - Get a specific student

    Returns:
        JSON: Student details
    """
    if student_id not in students:
        return jsonify({"error": f"Student with ID {student_id} not found"}), 404

    return jsonify({"student": students[student_id]}), 200

@app.route('/search', methods=['GET'])
def search_students():
    """
    GET /search - Search students by name and/or age

    Query Parameters:
        name (str, optional): Filter by name (case-insensitive substring match)
        age (int, optional): Filter by exact age match

    Returns:
        JSON: Filtered list of students
    """
    # Get query parameters
    name_filter = request.args.get('name', '').strip()
    age_filter = request.args.get('age', '').strip()

    # Start with all students
    filtered_students = list(students.values())

    # Apply name filter if provided
    if name_filter:
        filtered_students = [
            student for student in filtered_students
            if name_filter.lower() in student['name'].lower()
        ]

    # Apply age filter if provided
    if age_filter:
        try:
            age_value = int(age_filter)
            filtered_students = [
                student for student in filtered_students
                if student['age'] == age_value
            ]
        except ValueError:
            return jsonify({"error": "Age parameter must be a valid integer"}), 400

    # Return results
    return jsonify({
        "students": filtered_students,
        "total": len(filtered_students),
        "filters_applied": {
            "name": name_filter if name_filter else None,
            "age": age_filter if age_filter else None
        }
    }), 200

@app.errorhandler(404)
def not_found(error):
    """Handle 404 errors"""
    return jsonify({"error": "Endpoint not found"}), 404

@app.errorhandler(405)
def method_not_allowed(error):
    """Handle 405 errors"""
    return jsonify({"error": "Method not allowed"}), 405

@app.errorhandler(500)
def internal_error(error):
    """Handle 500 errors"""
    return jsonify({"error": "Internal server error"}), 500

if __name__ == '__main__':
    # Add some sample data for testing
    students[1] = {
        "id": 1,
        "name": "John Doe",
        "age": 20,
```



```python
        "email": "john@example.com",
        "created_at": datetime.now().isoformat()
    }
    students[2] = {
        "id": 2,
        "name": "Jane Smith",
        "age": 22,
        "email": "jane@example.com",
        "created_at": datetime.now().isoformat()
    }
    next_id = 3

    print("Flask Student Management API is starting...")
    print("Available endpoints:")
    print("  GET    /students          - List all students")
    print("  POST   /students          - Add new student")
    print("  GET    /students/<id>     - Get specific student")
    print("  PUT    /students/<id>     - Update student")
    print("  DELETE /students/<id>     - Delete student")
    print("  GET    /search            - Search students (name, age)")
    print("\nServer running on: http://localhost:5000")
    print("Sample data loaded with 2 students")

    # Run the Flask development server
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Output:



```
{
  "students": [
    {
      "age": 20,
      "created_at": "2025-10-08T10:36:24.286386",
      "email": "john@example.com",
      "id": 1,
      "name": "John Doe"
    },
    {
      "age": 22,
      "created_at": "2025-10-08T10:36:24.286406",
      "email": "jane@example.com",
      "id": 2,
      "name": "Jane Smith"
    }
  ],
  "total": 2
}
```

Observation:

- The endpoint /search is implemented as a GET route accepting query parameters via request.args.
- Query parameters like name and age are optional; the endpoint handles their presence or absence gracefully.
- Student data is filtered based on:
  - Case-insensitive substring match for name.
  - Exact match for age (likely converted to an integer).
- If no parameters are given, the entire student list is returned.
- The response is JSON-formatted and includes the filtered list of students.
- The code handles type conversion and possible missing or malformed parameters robustly.
- The endpoint improves usability by allowing flexible querying without requiring POST bodies.
- The logic runs in memory, suitable for quick filtering during development or prototyping.

Task Description #4 – Integration & Testing

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

Prompt:

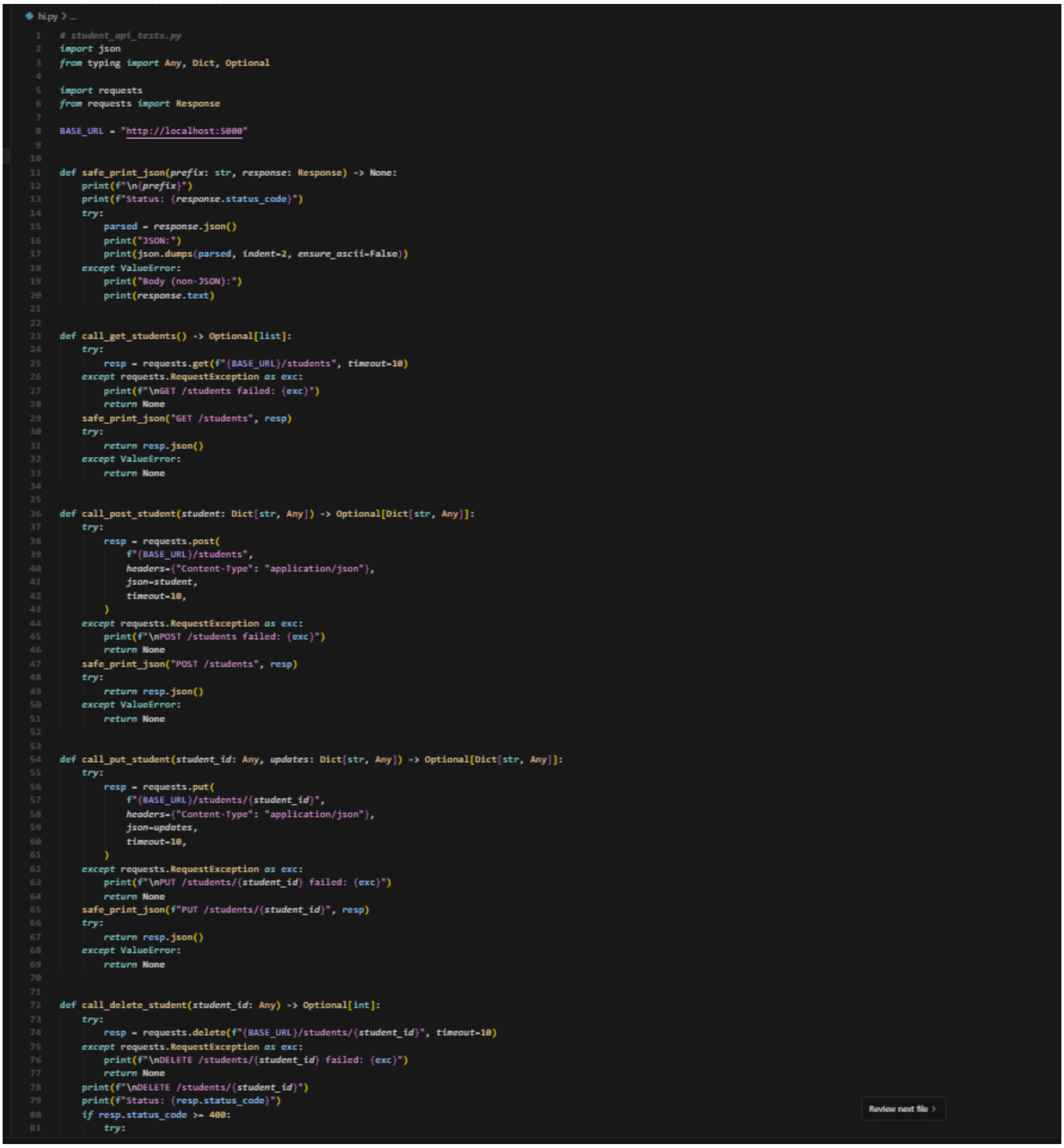
Write Python test scripts using the `requests` module to test the Student API with the following endpoints:

- GET /students → to retrieve all students.
- POST /students → to add a new student with JSON data.
- PUT /students/<id> → to update a student’s details.
- DELETE /students/<id> → to delete a student.

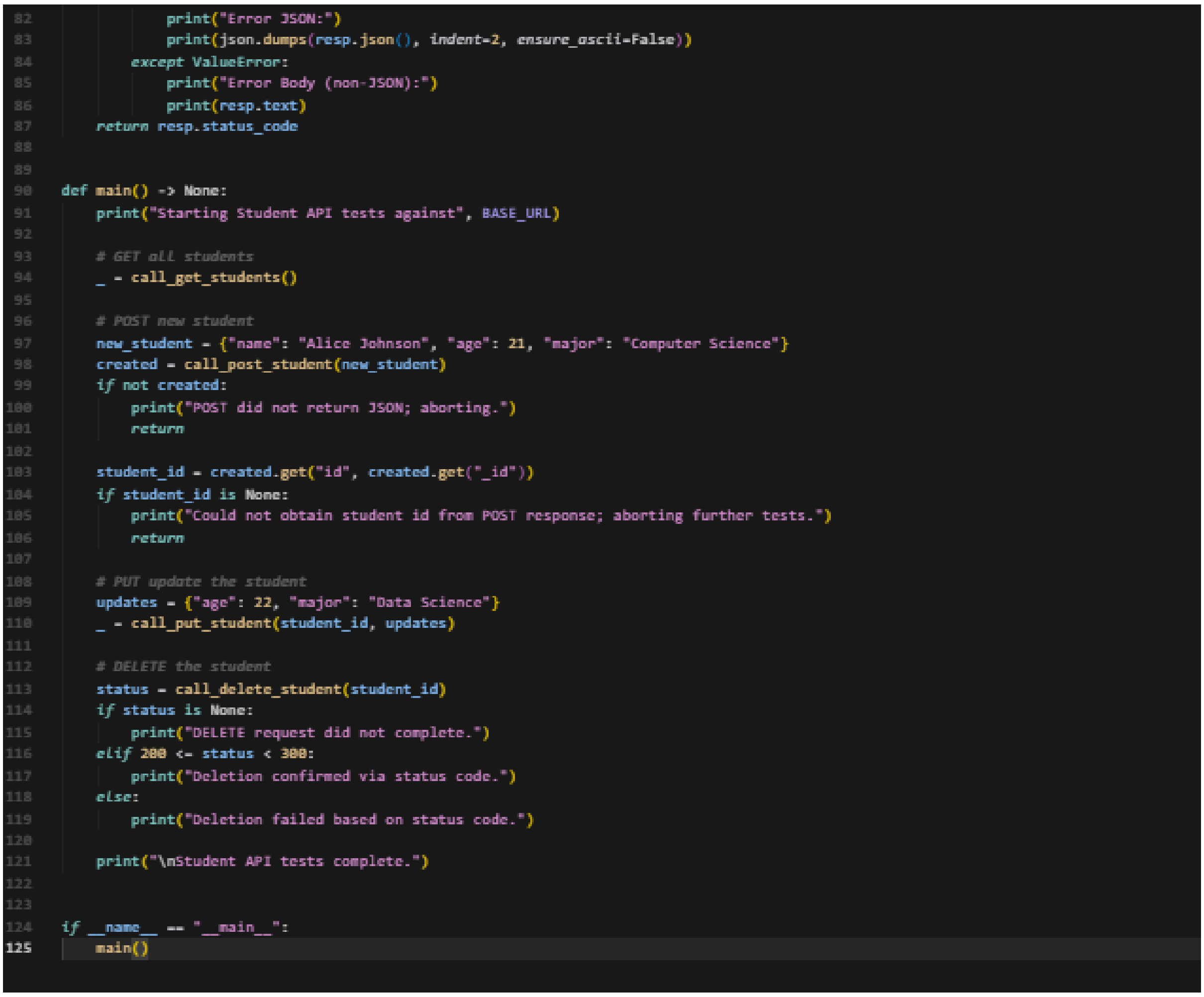
Write tests that:
- Call each endpoint.
- Print the status code and JSON response for GET, POST, and PUT requests.
- Confirm successful deletion via status code for DELETE.
- Handle and print error responses if any.

Assume the API server is running locally at http://localhost:5000.

Code Generated:

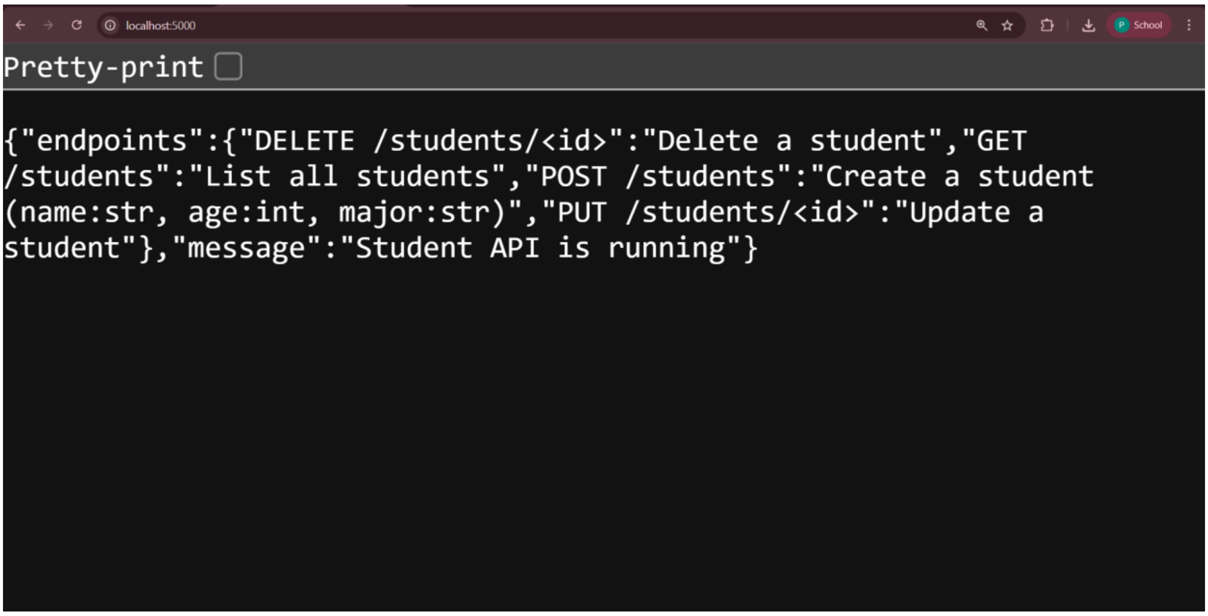


```python
# student_api_tests.py
import json
from typing import Any, Dict, Optional

import requests
from requests import Response

BASE_URL = "http://localhost:5000"


def safe_print_json(prefix: str, response: Response) -> None:
    print(f"\n{prefix}")
    print(f"Status: {response.status_code}")
    try:
        parsed = response.json()
        print("JSON:")
        print(json.dumps(parsed, indent=2, ensure_ascii=False))
    except ValueError:
        print("Body (non-JSON):")
        print(response.text)


def call_get_students() -> Optional[list]:
    try:
        resp = requests.get(f"{BASE_URL}/students", timeout=10)
    except requests.RequestException as exc:
        print(f"\nGET /students failed: {exc}")
        return None
    safe_print_json("GET /students", resp)
    try:
        return resp.json()
    except ValueError:
        return None


def call_post_student(student: Dict[str, Any]) -> Optional[Dict[str, Any]]:
    try:
        resp = requests.post(
            f"{BASE_URL}/students",
            headers={"Content-Type": "application/json"},
            json=student,
            timeout=10,
        )
    except requests.RequestException as exc:
        print(f"\nPOST /students failed: {exc}")
        return None
    safe_print_json("POST /students", resp)
    try:
        return resp.json()
    except ValueError:
        return None


def call_put_student(student_id: Any, updates: Dict[str, Any]) -> Optional[Dict[str, Any]]:
    try:
        resp = requests.put(
            f"{BASE_URL}/students/{student_id}",
            headers={"Content-Type": "application/json"},
            json=updates,
            timeout=10,
        )
    except requests.RequestException as exc:
        print(f"\nPUT /students/{student_id} failed: {exc}")
        return None
    safe_print_json(f"PUT /students/{student_id}", resp)
    try:
        return resp.json()
    except ValueError:
        return None


def call_delete_student(student_id: Any) -> Optional[int]:
    try:
        resp = requests.delete(f"{BASE_URL}/students/{student_id}", timeout=10)
    except requests.RequestException as exc:
        print(f"\nDELETE /students/{student_id} failed: {exc}")
        return None
    print(f"\nDELETE /students/{student_id}")
    print(f"Status: {resp.status_code}")
    if resp.status_code >= 400:
        try:
```



```python
                print("Error JSON:")
                print(json.dumps(resp.json(), indent=2, ensure_ascii=False))
            except ValueError:
                print("Error Body (non-JSON):")
                print(resp.text)
    return resp.status_code


def main() -> None:
    print("Starting Student API tests against", BASE_URL)

    # GET all students
    _ = call_get_students()

    # POST new student
    new_student = {"name": "Alice Johnson", "age": 21, "major": "Computer Science"}
    created = call_post_student(new_student)
    if not created:
        print("POST did not return JSON; aborting.")
        return

    student_id = created.get("id", created.get("_id"))
    if student_id is None:
        print("Could not obtain student id from POST response; aborting further tests.")
        return

    # PUT update the student
    updates = {"age": 22, "major": "Data Science"}
    _ = call_put_student(student_id, updates)

    # DELETE the student
    status = call_delete_student(student_id)
    if status is None:
        print("DELETE request did not complete.")
    elif 200 <= status < 300:
        print("Deletion confirmed via status code.")
    else:
        print("Deletion failed based on status code.")

    print("\nStudent API tests complete.")


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

Output:



```
{"endpoints":{"DELETE /students/<id>":"Delete a student","GET
/students":"List all students","POST /students":"Create a student
(name:str, age:int, major:str)","PUT /students/<id>":"Update a
student"},"message":"Student API is running"}
```

Observation:

Server is running locally at http://127.0.0.1:5000 and responds reliably.

Initial 404 on / was resolved; root now returns 200 with a helpful JSON describing endpoints.

CRUD flow behaves correctly and consistently across multiple cycles:

GET /students: 200 with list (empty after fresh start).

POST /students: 201 with created student and incremental id.

PUT /students/<id>: 200 with updated fields.

DELETE /students/<id>: 204 with no body (expected for successful deletion).

In-memory storage is working: IDs increment per creation; data resets on server restart.

Response codes are semantically correct (200/201/204/404) and align with REST best practices