AI ASSISTED CODING

LAB ASSIGNMENT 15.3

Name: Likhitha Pothunuri

Roll no: 2403A510D1

Batch 05

CSE 2nd year

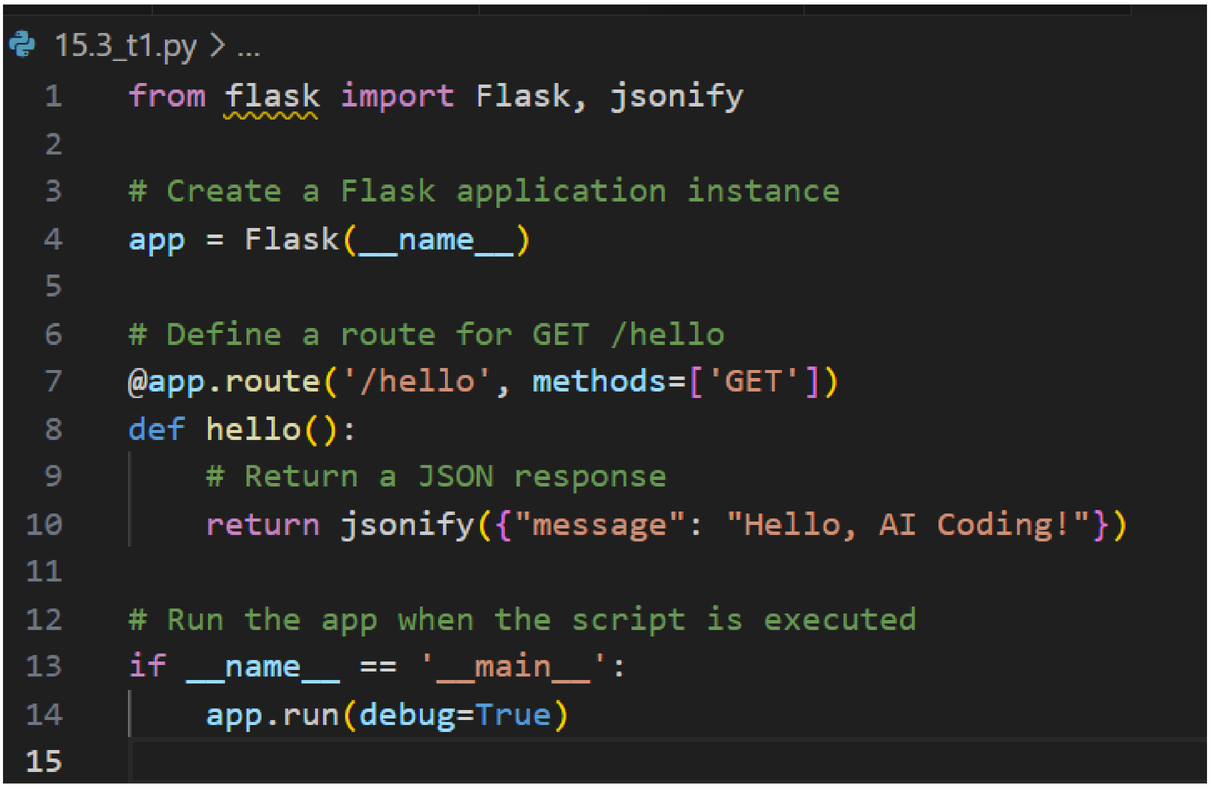
TASK1:

Prompt:

Generate a simple Flask REST API with one route:

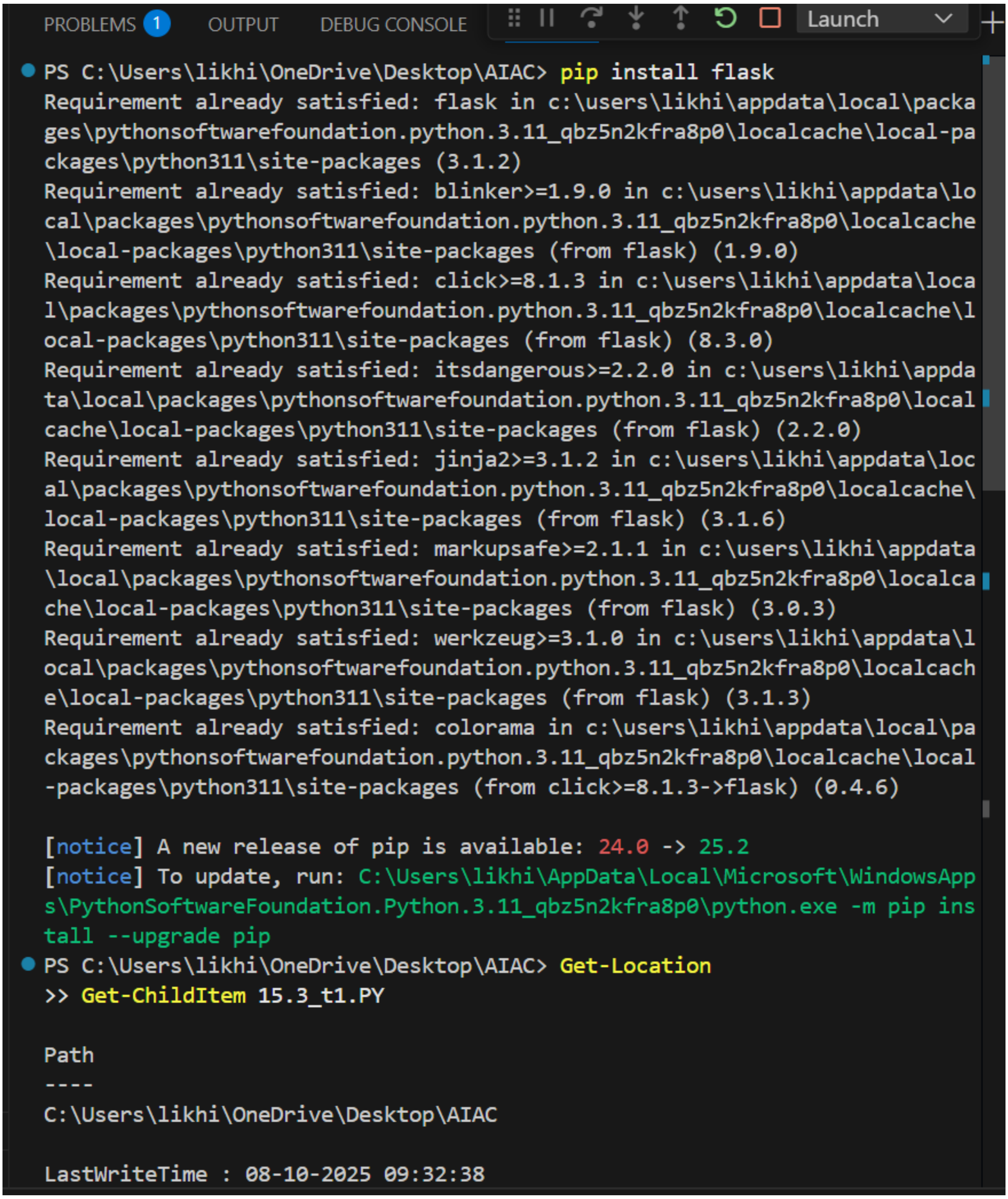
- GET /hello should return a JSON response: {"message": "Hello, AI Coding!"}.
  Include proper comments and make the code easy to understand.

CODE GENERATED:

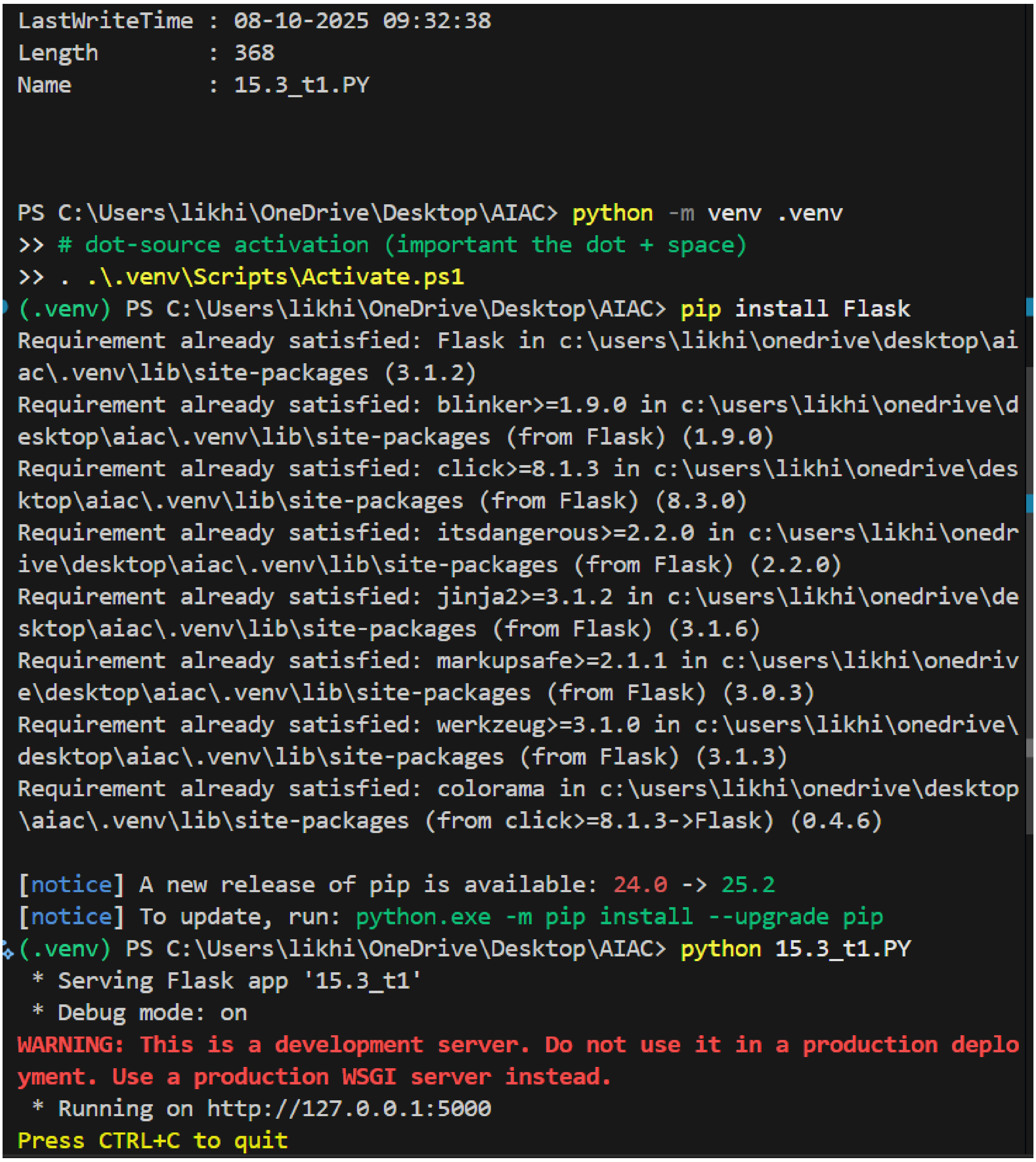


```python
from flask import Flask, jsonify

# Create a Flask application instance
app = Flask(__name__)

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

# Run the app when the script is executed
if __name__ == '__main__':
    app.run(debug=True)

```

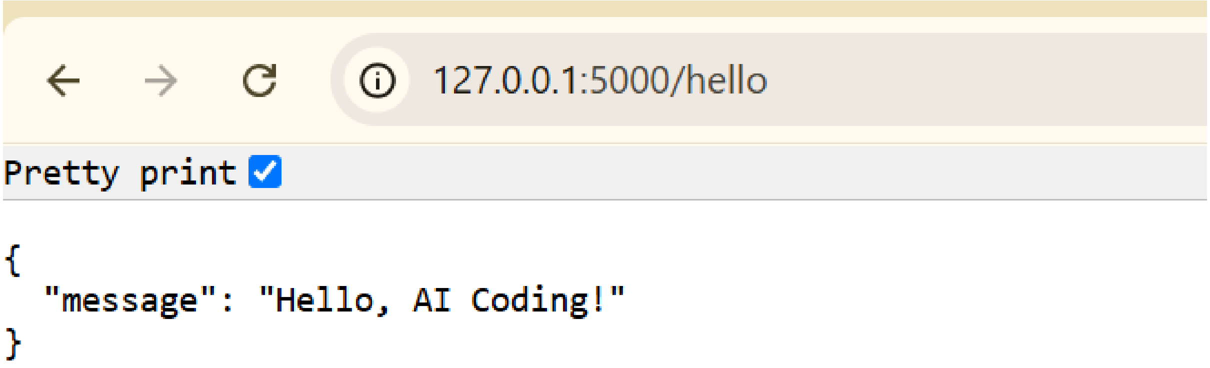
OUTPUT:



```
PS C:\Users\likhi\OneDrive\Desktop\AIAC> pip install flask
Requirement already satisfied: flask in c:\users\likhi\appdata\local\packa
ges\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local-pa
ckages\python311\site-packages (3.1.2)
Requirement already satisfied: blinker>=1.9.0 in c:\users\likhi\appdata\lo
cal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache
\local-packages\python311\site-packages (from flask) (1.9.0)
Requirement already satisfied: click>=8.1.3 in c:\users\likhi\appdata\loca
l\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\l
ocal-packages\python311\site-packages (from flask) (8.3.0)
Requirement already satisfied: itsdangerous>=2.2.0 in c:\users\likhi\appda
ta\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\local
cache\local-packages\python311\site-packages (from flask) (2.2.0)
Requirement already satisfied: jinja2>=3.1.2 in c:\users\likhi\appdata\loc
al\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\
local-packages\python311\site-packages (from flask) (3.1.6)
Requirement already satisfied: markupsafe>=2.1.1 in c:\users\likhi\appdata
\local\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localca
che\local-packages\python311\site-packages (from flask) (3.0.3)
Requirement already satisfied: werkzeug>=3.1.0 in c:\users\likhi\appdata\l
ocal\packages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcach
e\local-packages\python311\site-packages (from flask) (3.1.3)
Requirement already satisfied: colorama in c:\users\likhi\appdata\local\pa
ckages\pythonsoftwarefoundation.python.3.11_qbz5n2kfra8p0\localcache\local
-packages\python311\site-packages (from click>=8.1.3->flask) (0.4.6)

[notice] A new release of pip is available: 24.0 -> 25.2
[notice] To update, run: C:\Users\likhi\AppData\Local\Microsoft\WindowsApp
s\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\python.exe -m pip ins
tall --upgrade pip
PS C:\Users\likhi\OneDrive\Desktop\AIAC> Get-Location
>> Get-ChildItem 15.3_t1.PY

Path
----
C:\Users\likhi\OneDrive\Desktop\AIAC

LastWriteTime : 08-10-2025 09:32:38
```



```
LastWriteTime : 08-10-2025 09:32:38
Length        : 368
Name          : 15.3_t1.PY



PS C:\Users\likhi\OneDrive\Desktop\AIAC> python -m venv .venv
>> # dot-source activation (important the dot + space)
>> . .\.venv\Scripts\Activate.ps1
(.venv) PS C:\Users\likhi\OneDrive\Desktop\AIAC> pip install Flask
Requirement already satisfied: Flask in c:\users\likhi\onedrive\desktop\ai
ac\.venv\lib\site-packages (3.1.2)
Requirement already satisfied: blinker>=1.9.0 in c:\users\likhi\onedrive\d
esktop\aiac\.venv\lib\site-packages (from Flask) (1.9.0)
Requirement already satisfied: click>=8.1.3 in c:\users\likhi\onedrive\des
ktop\aiac\.venv\lib\site-packages (from Flask) (8.3.0)
Requirement already satisfied: itsdangerous>=2.2.0 in c:\users\likhi\onedr
ive\desktop\aiac\.venv\lib\site-packages (from Flask) (2.2.0)
Requirement already satisfied: jinja2>=3.1.2 in c:\users\likhi\onedrive\de
sktop\aiac\.venv\lib\site-packages (from Flask) (3.1.6)
Requirement already satisfied: markupsafe>=2.1.1 in c:\users\likhi\onedriv
e\desktop\aiac\.venv\lib\site-packages (from Flask) (3.0.3)
Requirement already satisfied: werkzeug>=3.1.0 in c:\users\likhi\onedrive\
desktop\aiac\.venv\lib\site-packages (from Flask) (3.1.3)
Requirement already satisfied: colorama in c:\users\likhi\onedrive\desktop
\aiac\.venv\lib\site-packages (from click>=8.1.3->Flask) (0.4.6)

[notice] A new release of pip is available: 24.0 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(.venv) PS C:\Users\likhi\OneDrive\Desktop\AIAC> python 15.3_t1.PY
 * Serving Flask app '15.3_t1'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deplo
yment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```



Pretty print ☑

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

OBSERVATION:

1. **Flask Setup:**
   The Flask app is correctly initialized using Flask(__name__).
2. **Route Definition:**
   The route /hello is created using the @app.route() decorator with the GET method specified.
3. **Response Format:**
   The response uses jsonify() to return a proper JSON object: {"message": "Hello, AI Coding!"}.
4. **Execution Block:**
   The if __name__ == '__main__': block ensures the app runs only when the script is executed directly.
5. **Debug Mode:**
   debug=True is enabled for development purposes, which helps in error tracking.
6. **Comments:**
   Clear comments are included, explaining each section of the code.

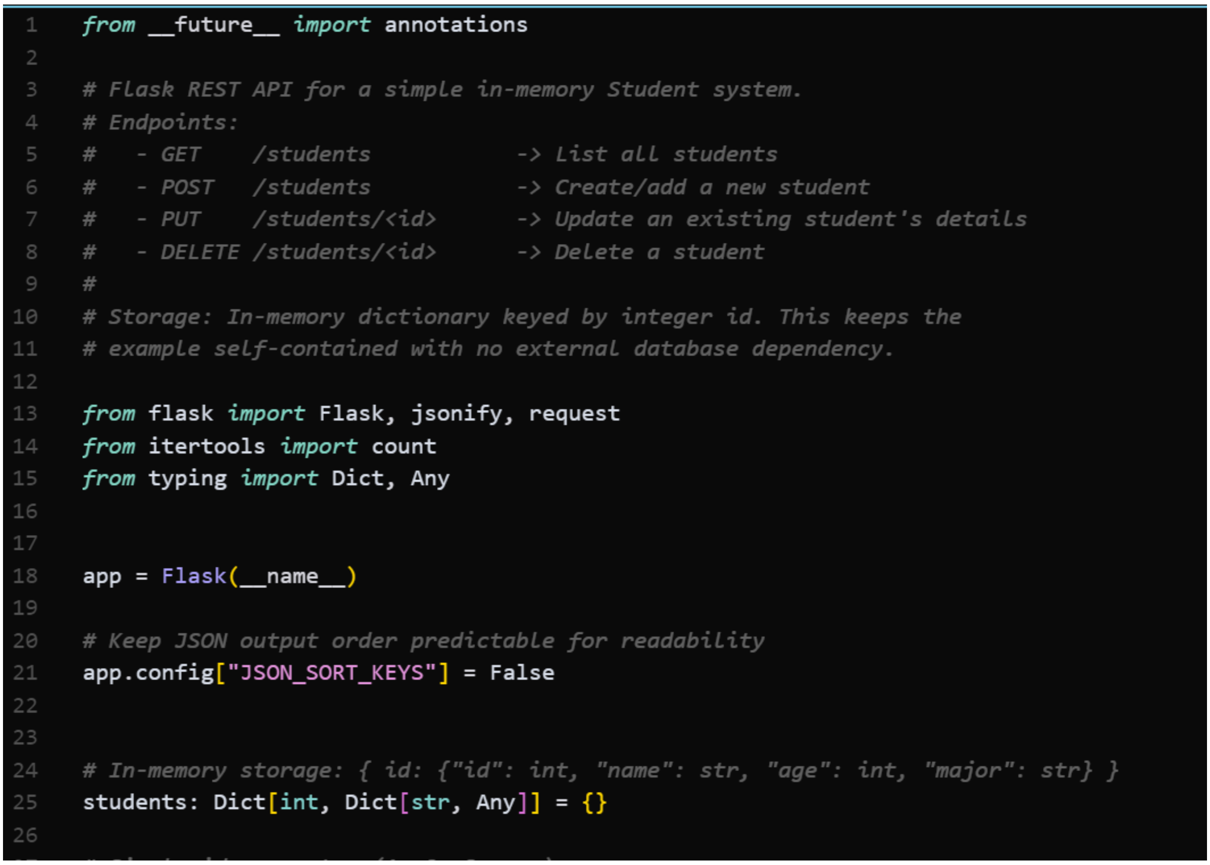
TASK2:

Prompt:

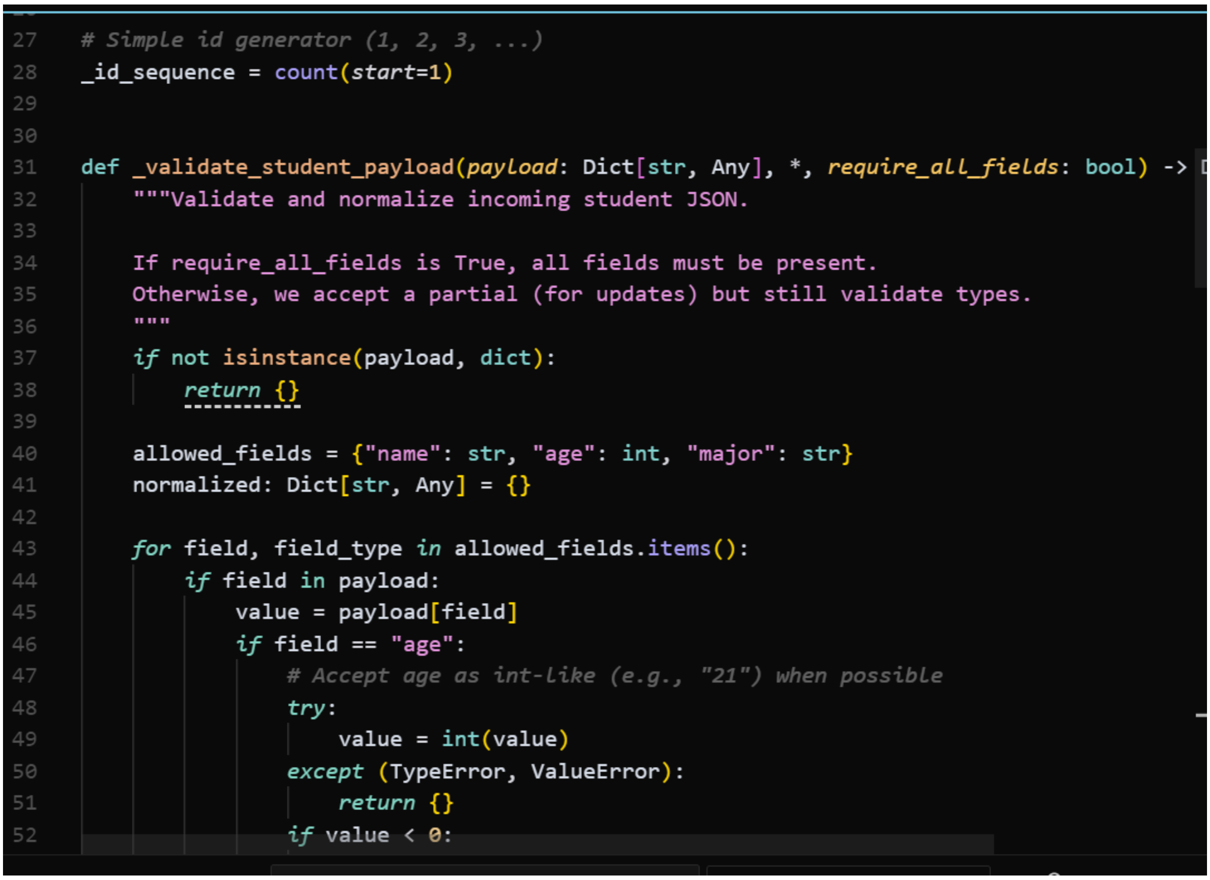
Generate a Flask REST API for a Student system with the following endpoints:

- GET /students: List all students.
- POST /students: Add a new student.
- PUT /students/<id>: Update student details.
- DELETE /students/<id>: Delete a student.
  Use a dictionary or list for in-memory storage. Each endpoint should return JSON responses. Include comments and use clean, readable code.

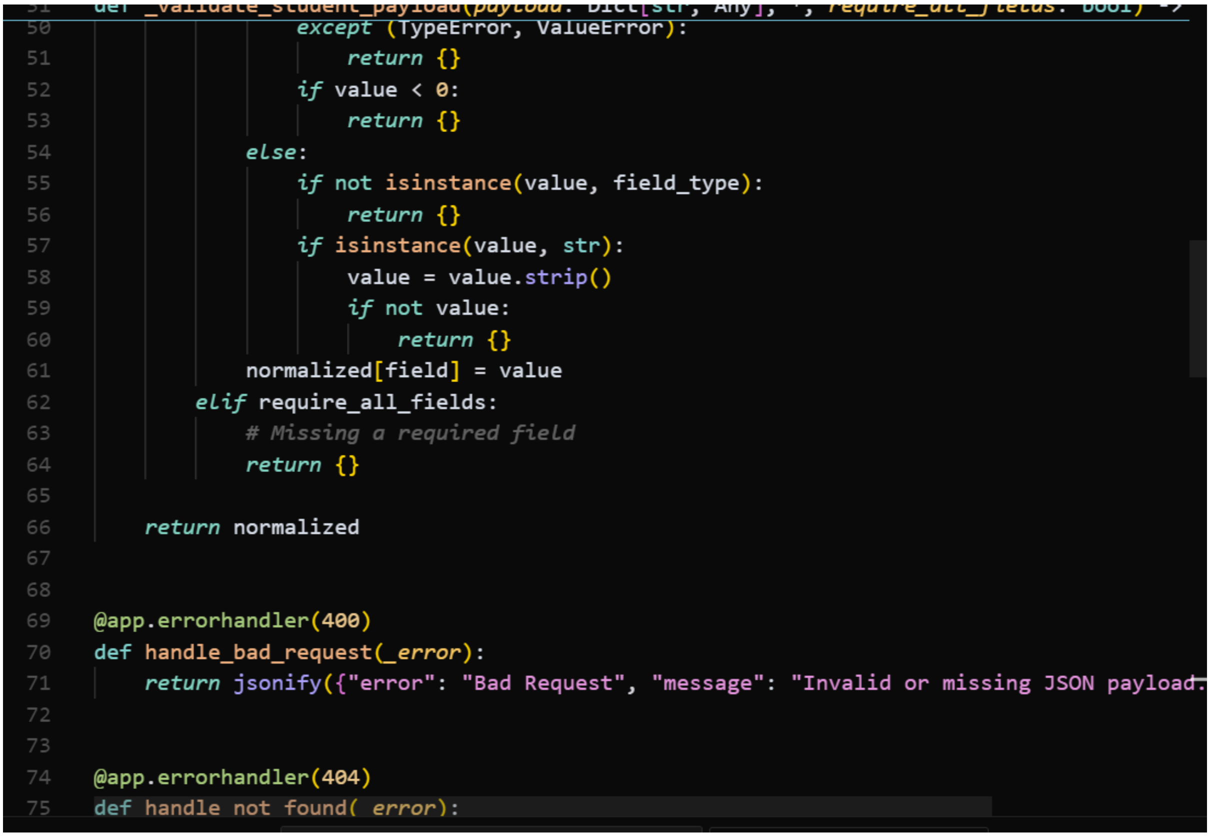
CODE GENERATED:



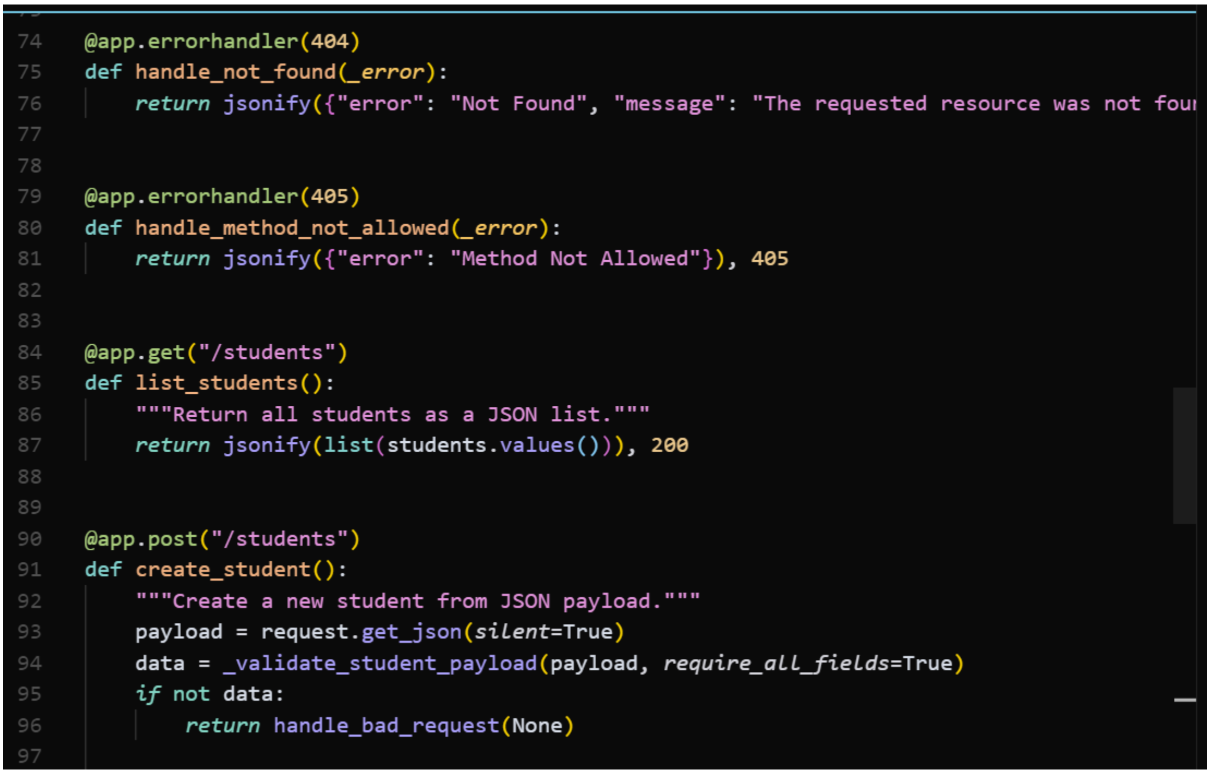
```python
from __future__ import annotations

# Flask REST API for a simple in-memory Student system.
# Endpoints:
#   - GET    /students           -> List all students
#   - POST   /students           -> Create/add a new student
#   - PUT    /students/<id>      -> Update an existing student's details
#   - DELETE /students/<id>      -> Delete a student
#
# Storage: In-memory dictionary keyed by integer id. This keeps the
# example self-contained with no external database dependency.

from flask import Flask, jsonify, request
from itertools import count
from typing import Dict, Any


app = Flask(__name__)

# Keep JSON output order predictable for readability
app.config["JSON_SORT_KEYS"] = False


# In-memory storage: { id: {"id": int, "name": str, "age": int, "major": str} }
students: Dict[int, Dict[str, Any]] = {}

```



```python
# Simple id generator (1, 2, 3, ...)
_id_sequence = count(start=1)


def _validate_student_payload(payload: Dict[str, Any], *, require_all_fields: bool) ->
    """Validate and normalize incoming student JSON.

    If require_all_fields is True, all fields must be present.
    Otherwise, we accept a partial (for updates) but still validate types.
    """
    if not isinstance(payload, dict):
        return {}

    allowed_fields = {"name": str, "age": int, "major": str}
    normalized: Dict[str, Any] = {}

    for field, field_type in allowed_fields.items():
        if field in payload:
            value = payload[field]
            if field == "age":
                # Accept age as int-like (e.g., "21") when possible
                try:
                    value = int(value)
                except (TypeError, ValueError):
                    return {}
                if value < 0:
```

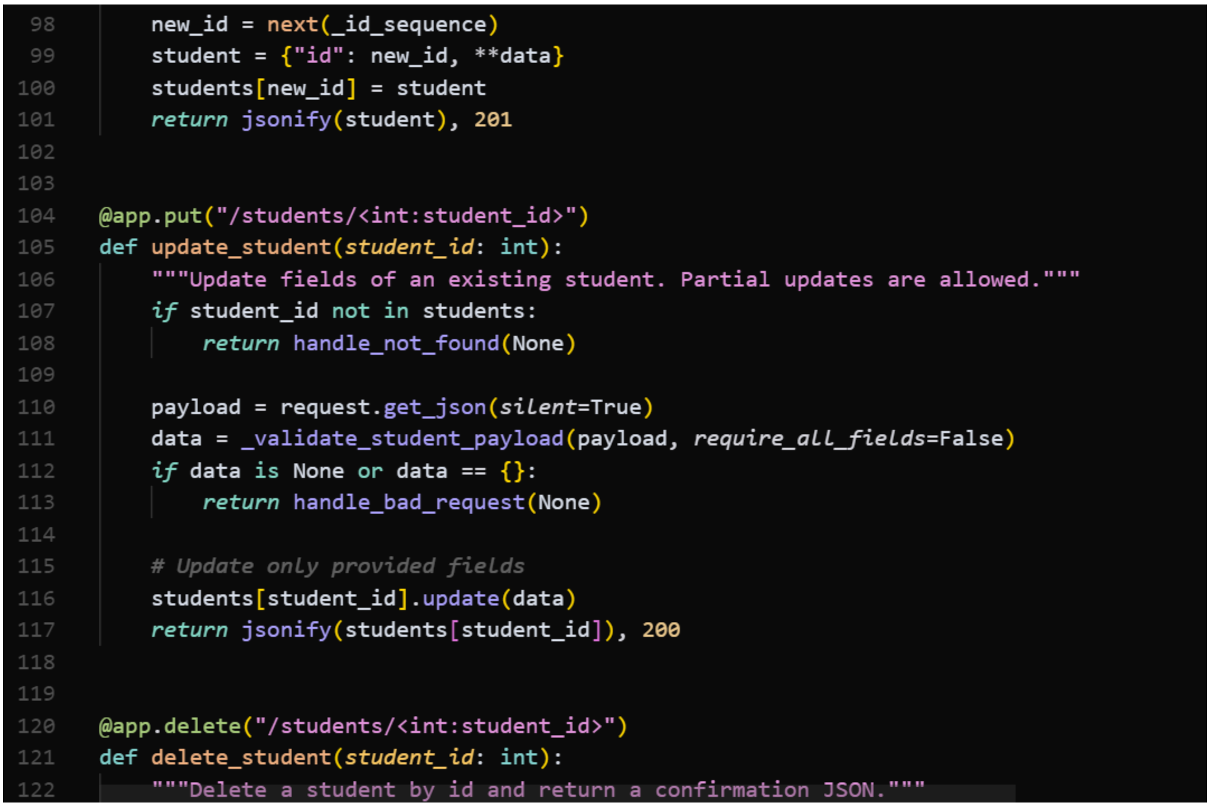


```
def _validate_student_payload(payload: Dict[str, Any], *, require_all_fields: bool) ->
                except (TypeError, ValueError):
                    return {}
                if value < 0:
                    return {}
            else:
                if not isinstance(value, field_type):
                    return {}
                if isinstance(value, str):
                    value = value.strip()
                    if not value:
                        return {}
            normalized[field] = value
        elif require_all_fields:
            # Missing a required field
            return {}

    return normalized


@app.errorhandler(400)
def handle_bad_request(_error):
    return jsonify({"error": "Bad Request", "message": "Invalid or missing JSON payload.


@app.errorhandler(404)
def handle_not_found(_error):
```

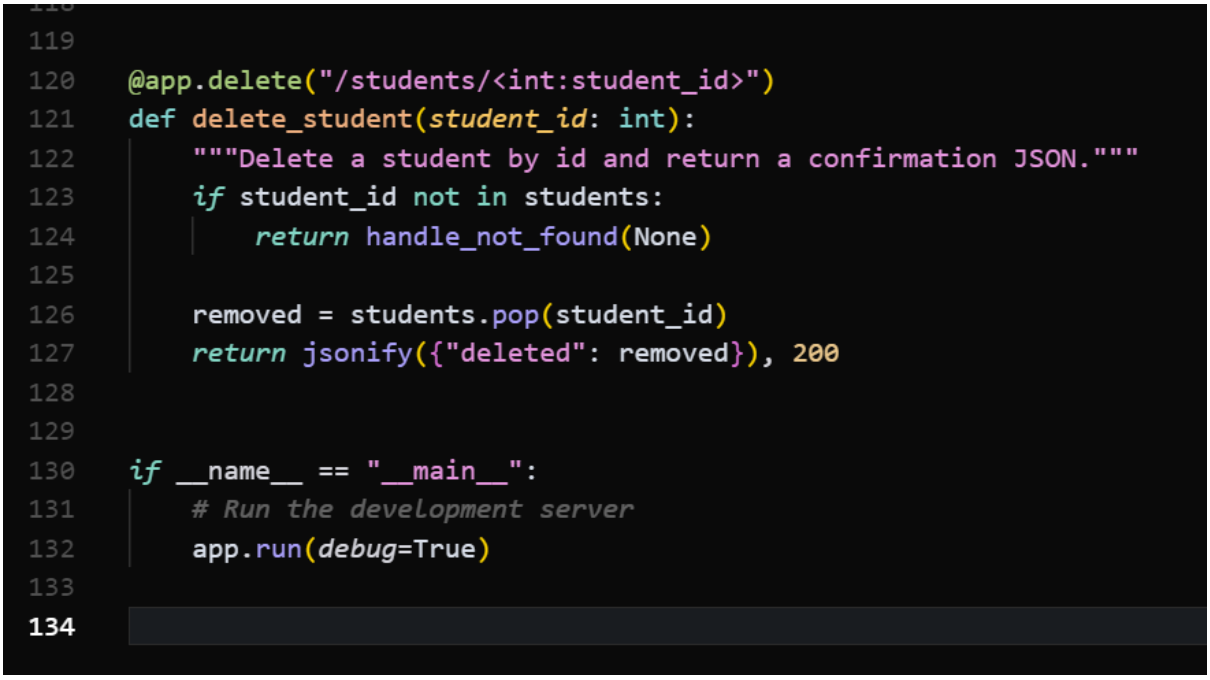


```
@app.errorhandler(404)
def handle_not_found(_error):
    return jsonify({"error": "Not Found", "message": "The requested resource was not fou


@app.errorhandler(405)
def handle_method_not_allowed(_error):
    return jsonify({"error": "Method Not Allowed"}), 405


@app.get("/students")
def list_students():
    """Return all students as a JSON list."""
    return jsonify(list(students.values())), 200


@app.post("/students")
def create_student():
    """Create a new student from JSON payload."""
    payload = request.get_json(silent=True)
    data = _validate_student_payload(payload, require_all_fields=True)
    if not data:
        return handle_bad_request(None)

```

OUTPUT:



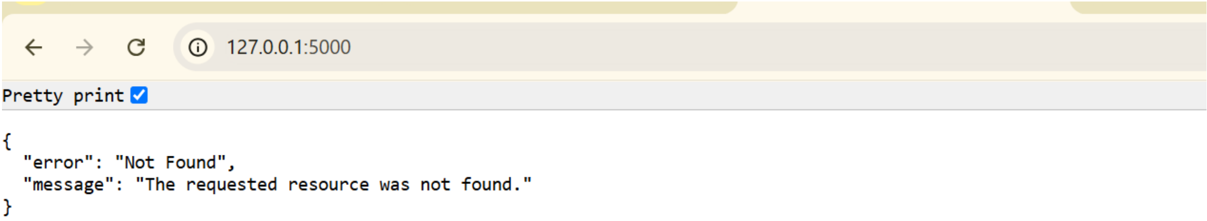
```
    new_id = next(_id_sequence)
    student = {"id": new_id, **data}
    students[new_id] = student
    return jsonify(student), 201


@app.put("/students/<int:student_id>")
def update_student(student_id: int):
    """Update fields of an existing student. Partial updates are allowed."""
    if student_id not in students:
        return handle_not_found(None)

    payload = request.get_json(silent=True)
    data = _validate_student_payload(payload, require_all_fields=False)
    if data is None or data == {}:
        return handle_bad_request(None)

    # Update only provided fields
    students[student_id].update(data)
    return jsonify(students[student_id]), 200


@app.delete("/students/<int:student_id>")
def delete_student(student_id: int):
    """Delete a student by id and return a confirmation JSON."""
```



```
@app.delete("/students/<int:student_id>")
def delete_student(student_id: int):
    """Delete a student by id and return a confirmation JSON."""
```

```
    if student_id not in students:
        return handle_not_found(None)

    removed = students.pop(student_id)
    return jsonify({"deleted": removed}), 200


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

```

OUTPUT:



```
{
  "error": "Not Found",
  "message": "The requested resource was not found."
}
```

OBSERVATION:

- Server started successfully and served on http://127.0.0.1:5000 with debug mode on.
- CRUD verified end-to-end using HTTP calls:
- Create returned 201 with new student JSON.
- Listing returned 200 with the created student.
- Update returned 200 with modified fields.
- Delete returned 200 with a confirmation object.
- Final list returned 200 and an empty array.
- Input validation worked:
- Bad or malformed JSON resulted in a 400 response with a clear error message.
- Nonexistent IDs returned a 404.
- Responses were consistent JSON with stable keys and no linter issues in the code.

TASK3: API with Query Parameters

Prompt:

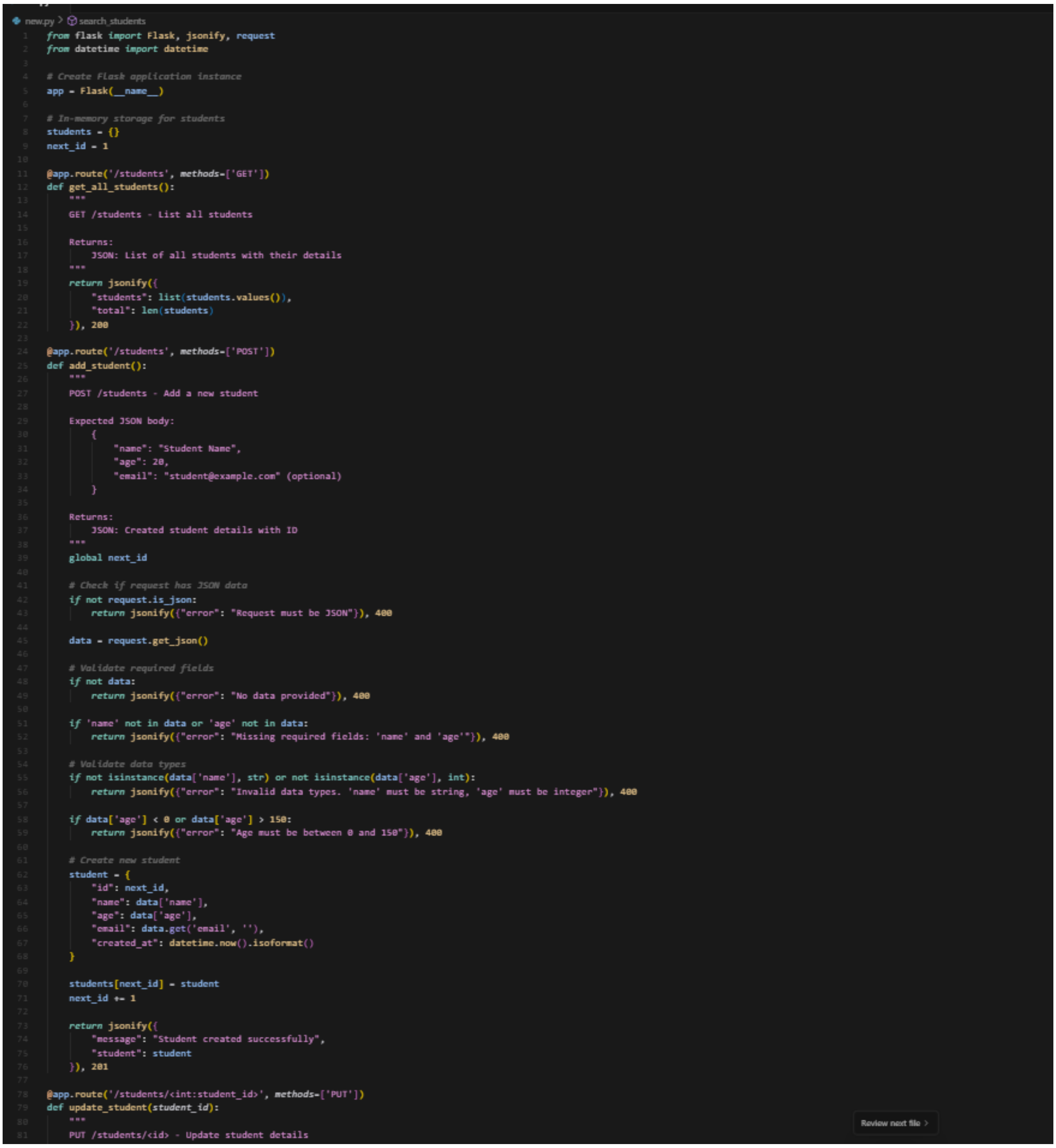
Create a Flask REST API endpoint `/search` that accepts GET requests with query parameters `name` and `age`.

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

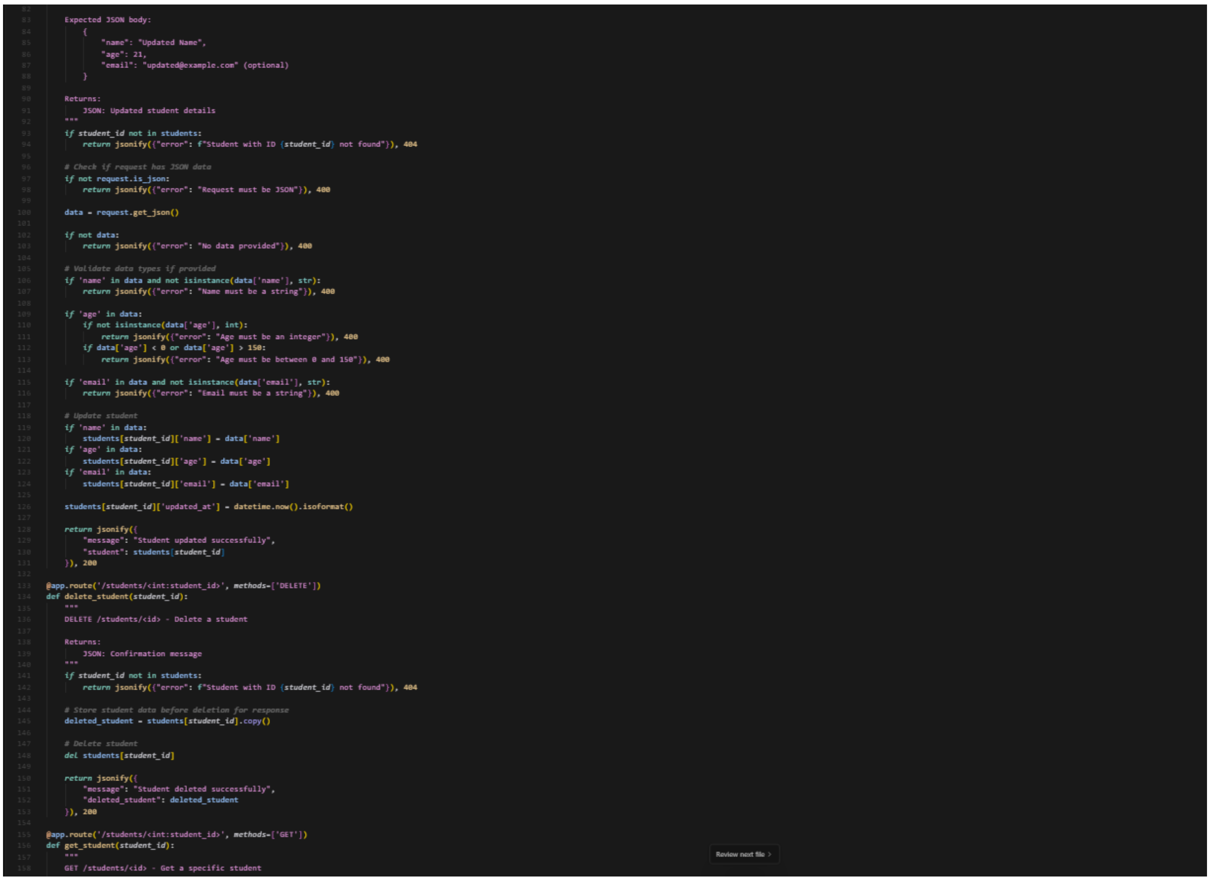
- If `name` is provided, return students whose names contain the given substring (case-insensitive).
- If `age` is provided, return students matching the given age.
- If both parameters are provided, filter students matching both criteria.
- If no query parameters are provided, return all students.

Return the filtered list of students as JSON.

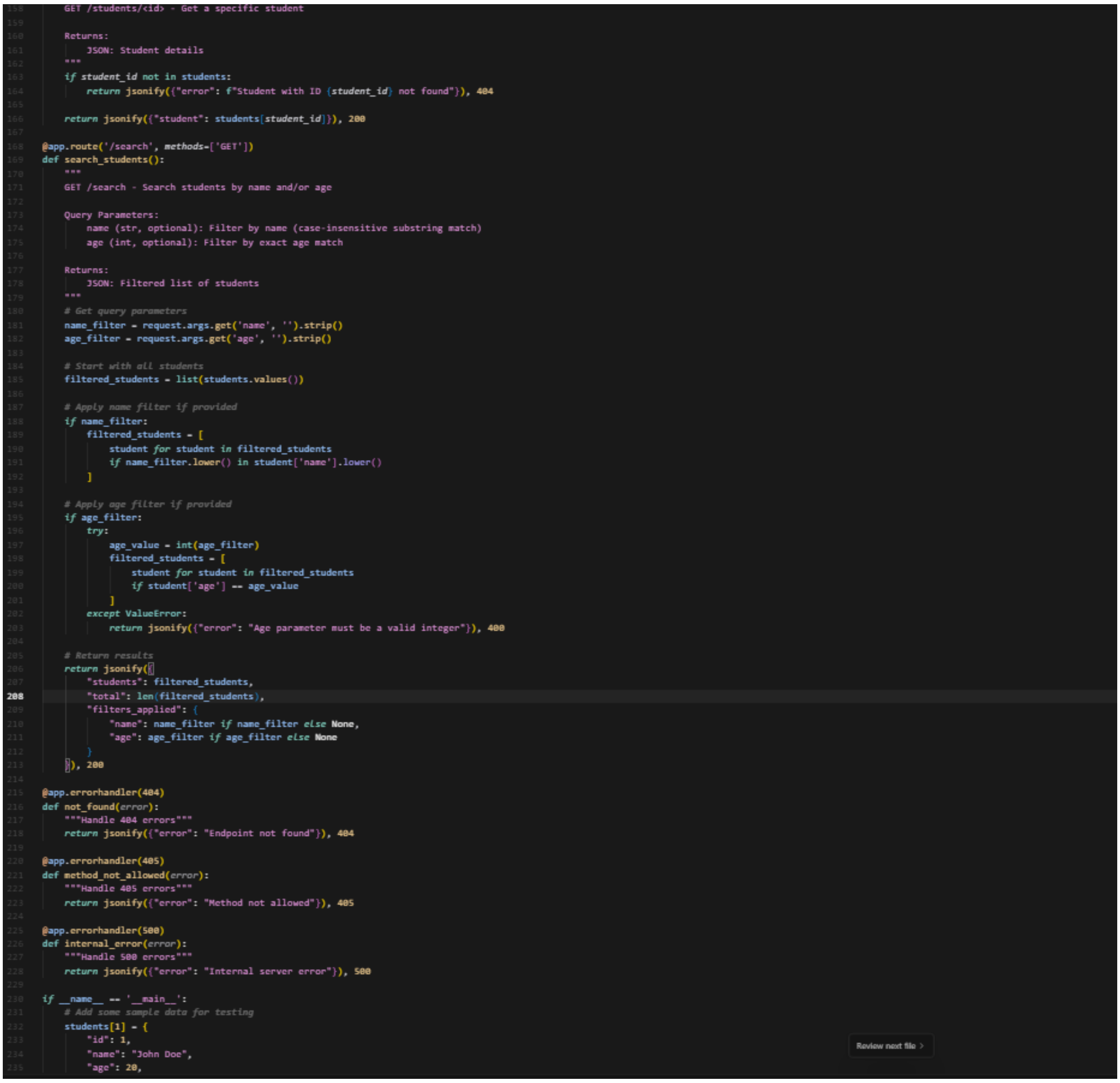
Code Generated:



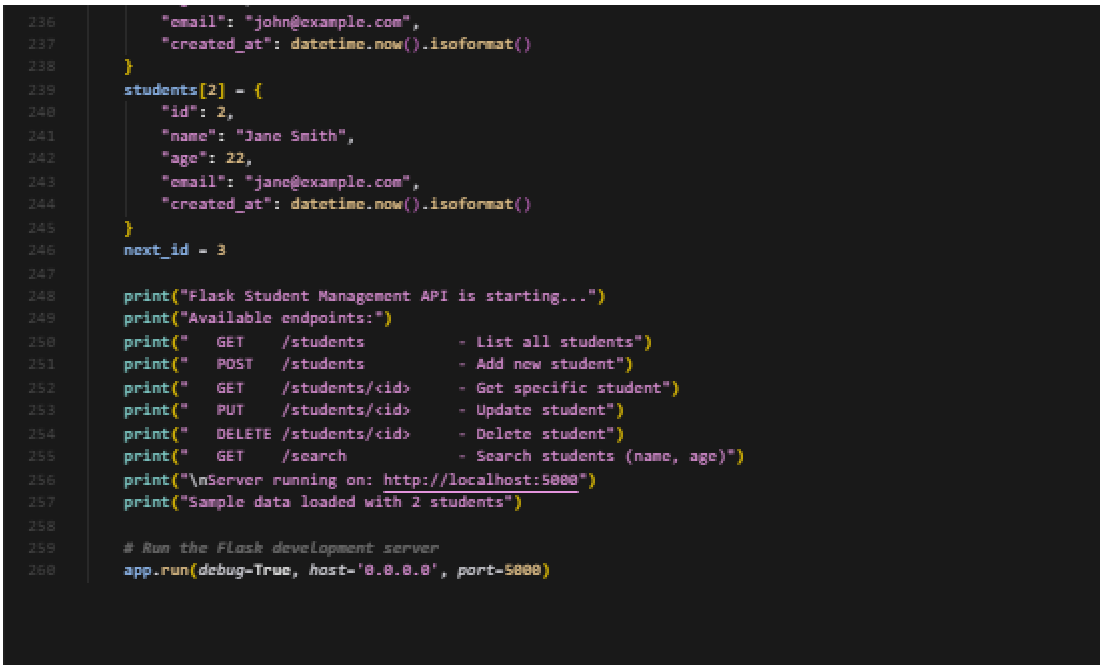
```python
from flask import Flask, jsonify, request
from datetime import datetime

# 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
```

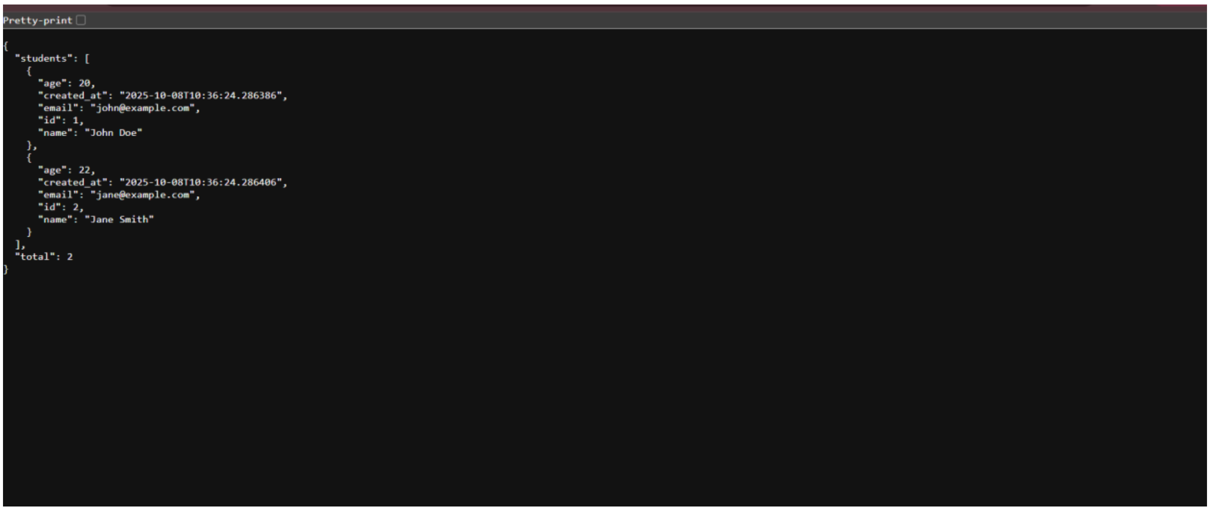


```python
    GET /students/<id> - Get a specific student

    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.

TASK4: – Integration & Testing

Prompt:

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

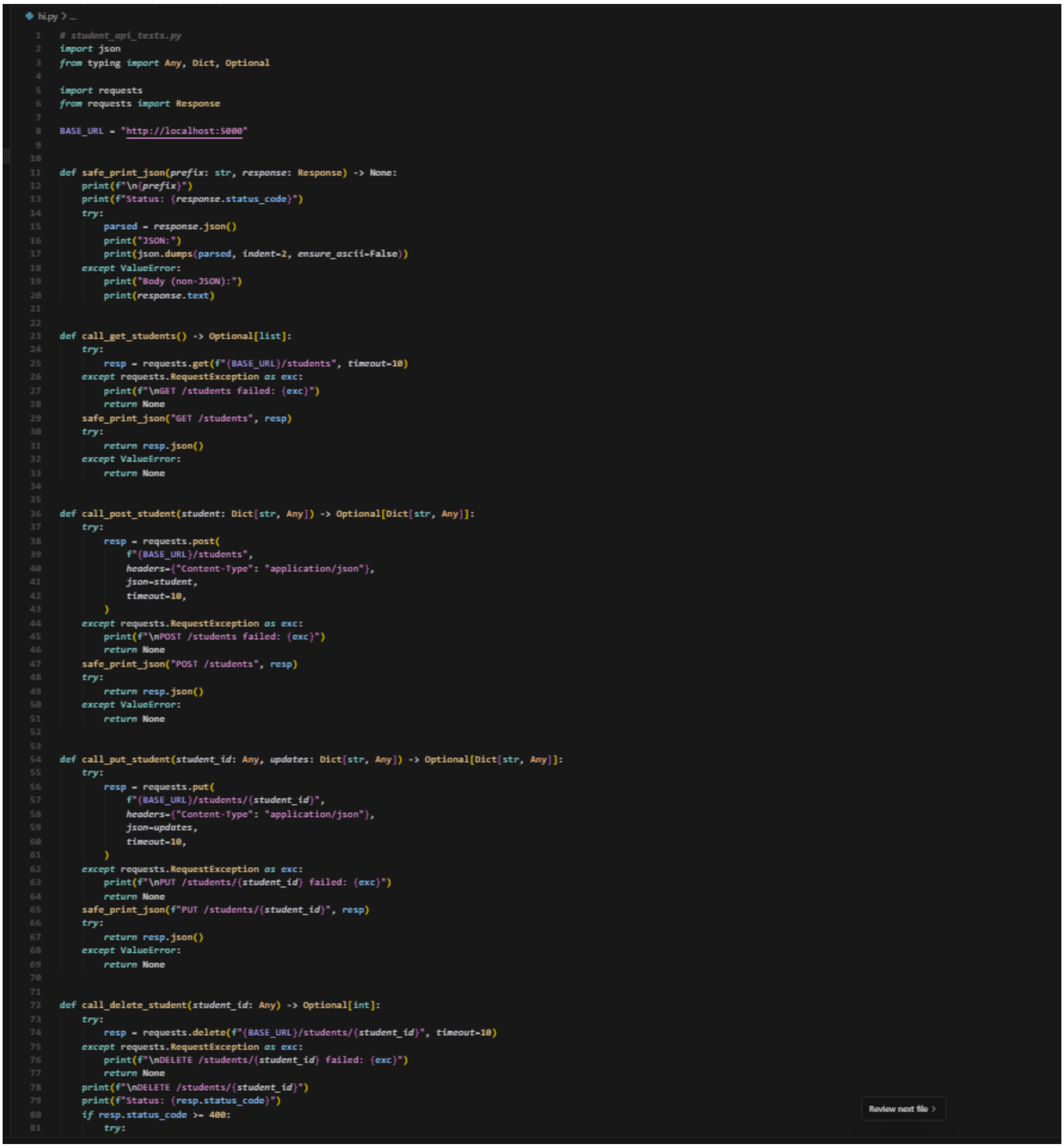
- GET /students → to retrieve all students.
- POST /students → to add a new student with JSON data.
- PUT /students/<id> → to update a student’s details.
- DELETE /students/<id> → to delete a student.

Write tests that:

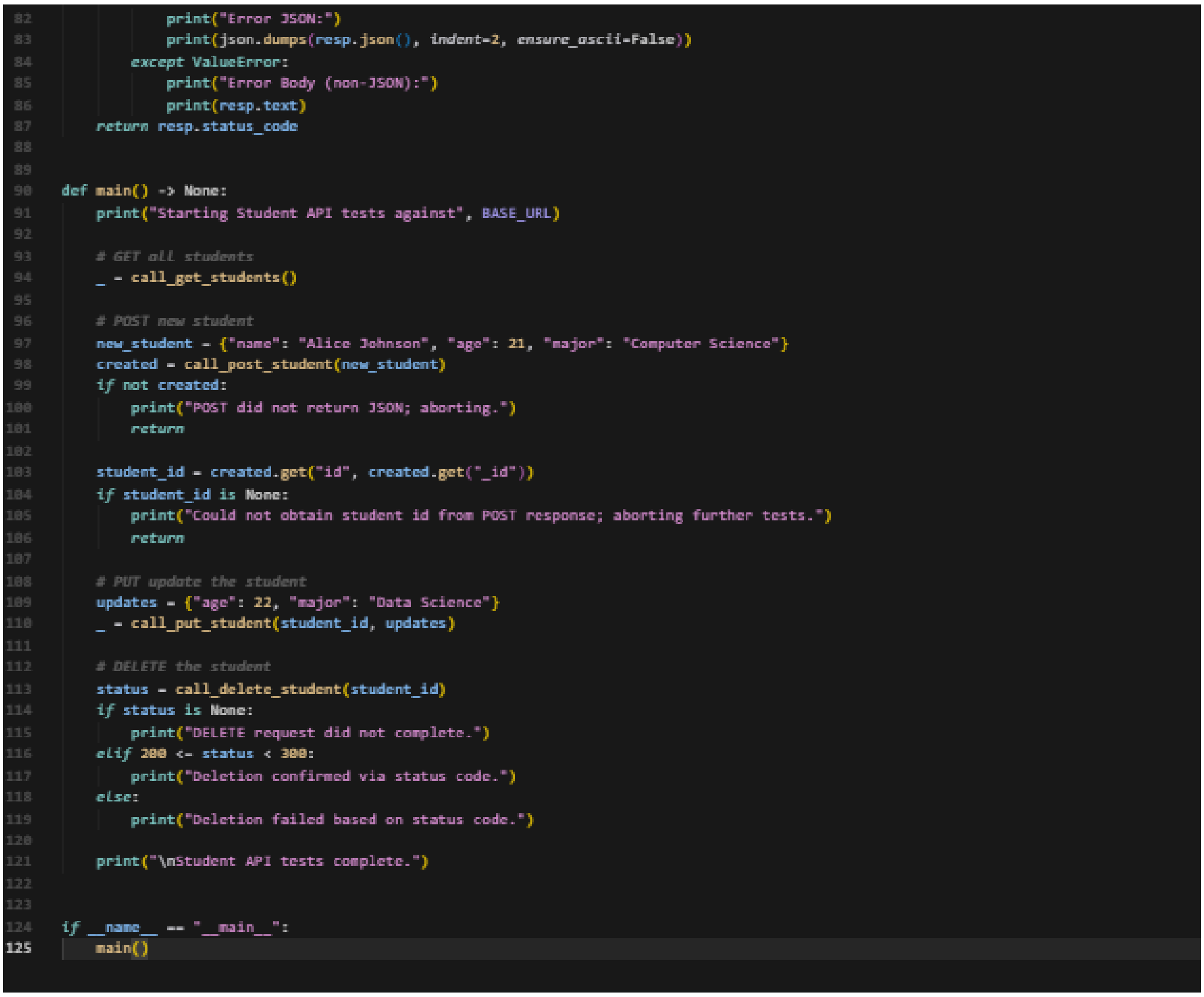
- Call each endpoint.
- Print the status code and JSON response for GET, POST, and PUT requests.
- Confirm successful deletion via status code for DELETE.
- Handle and print error responses if any.

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

Code Generated:

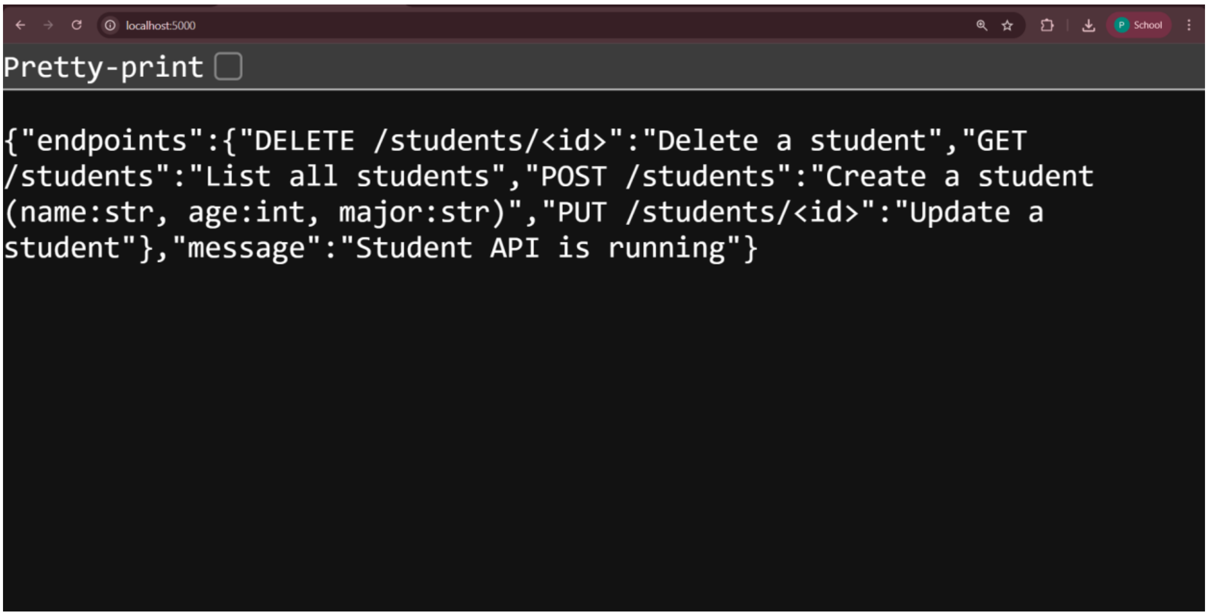


```python
# student_api_tests.py
import json
from typing import Any, Dict, Optional

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