ASSIGNMENT -15.3

NAME :S SAI RUCHITHA

HALLTICKECT NO : 2403A52316

BATCH NUMBER : 01

COURSE CODE : 24CS002PC215

PROGRAM NAME : B.TECH

YEAR/SEM : 2ND AND 3RD

TASK 1 :

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

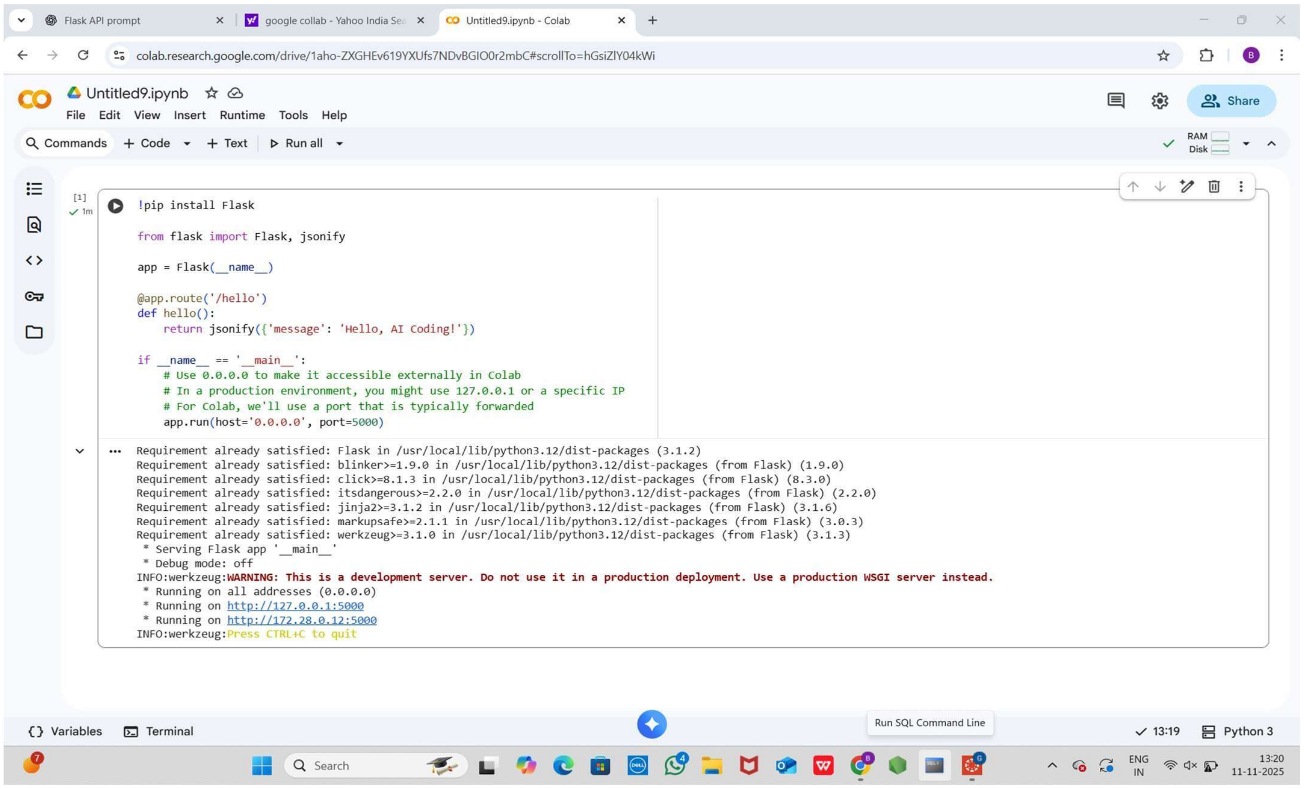
/hello → returns {"message": "Hello, AI Coding!"}

PROMPT :

“Create a Flask REST API with one route: GET /hello → returns

JSON { 'message': 'Hello, AI Coding!' }"

CODE :



```
!pip install Flask

from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/hello')
def hello():
    return jsonify({'message': 'Hello, AI Coding!'})

if __name__ == '__main__':
    # Use 0.0.0.0 to make it accessible externally in Colab
    # In a production environment, you might use 127.0.0.1 or a specific IP
    # For Colab, we'll use a port that is typically forwarded
    app.run(host='0.0.0.0', port=5000)
```

```
Requirement already satisfied: Flask in /usr/local/lib/python3.12/dist-packages (3.1.2)
Requirement already satisfied: blinker>=1.9.0 in /usr/local/lib/python3.12/dist-packages (from Flask) (1.9.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.12/dist-packages (from Flask) (8.3.0)
Requirement already satisfied: itsdangerous>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from Flask) (2.2.0)
Requirement already satisfied: jinja2>=3.1.2 in /usr/local/lib/python3.12/dist-packages (from Flask) (3.1.6)
Requirement already satisfied: markupsafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from Flask) (3.0.3)
Requirement already satisfied: werkzeug>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from Flask) (3.1.3)
 * Serving Flask app '__main__'
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.28.0.12:5000
INFO:werkzeug:Press CTRL+C to quit
```

OUTPUT :

* Running on all addresses (0.0.0.0)

* Running on h p://127.0.0.1:5000

* Running on h p://172.28.0.12:5000

INFO:werkzeug:Press CTRL+C to quit

TASK 2 :

Use AI to build REST endpoints for a Student API:

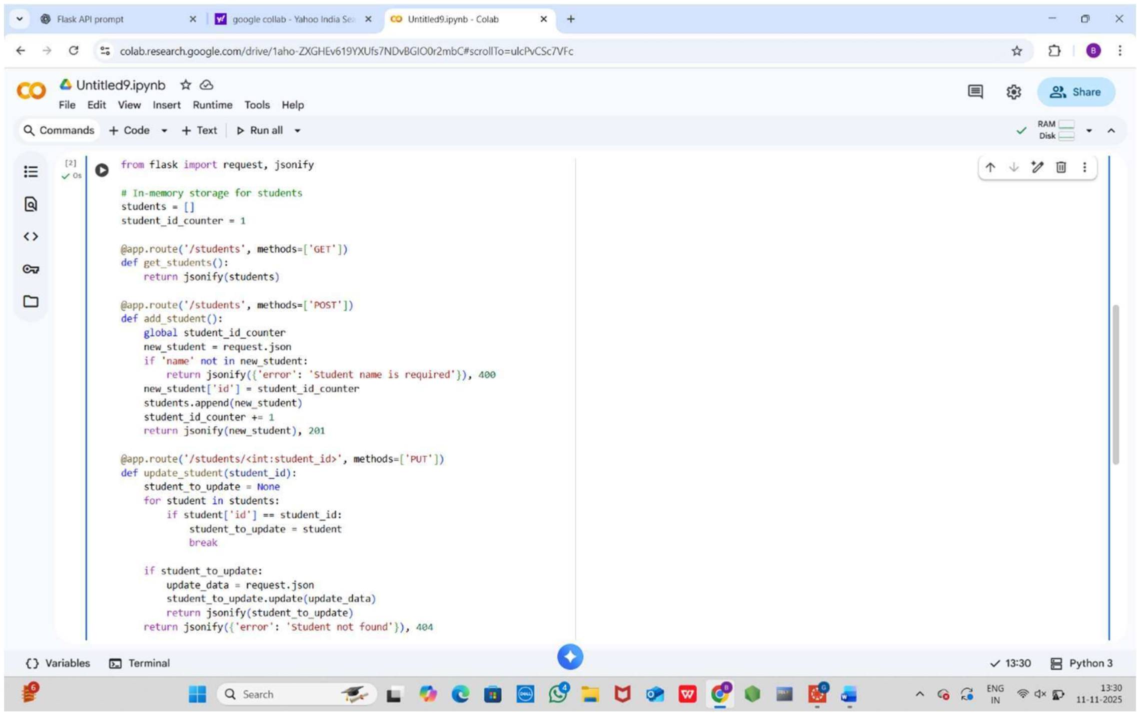
- GET /students → List all students.
- POST /students → Add a new student.
- PUT /students/<id> → Update student details.

- DELETE /students/<id> → Delete a student.

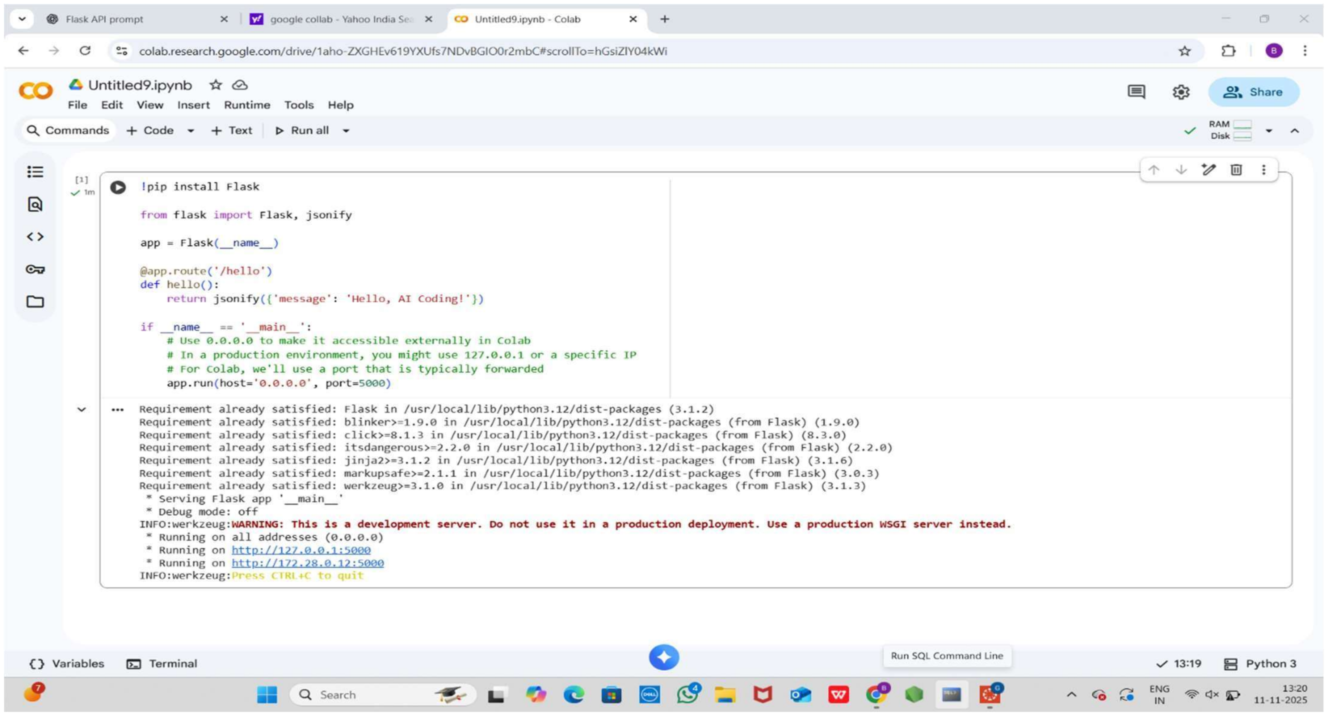
PROMPT :

“Create a Flask REST API for Students with CRUD: GET /students, POST /students, PUT /students/<id>, DELETE /students/<id>. Use list/dic onary storage and return JSON.”

CODE :



```
from flask import request, jsonify

# In-memory storage for students
students = []
student_id_counter = 1

@app.route('/students', methods=['GET'])
def get_students():
    return jsonify(students)

@app.route('/students', methods=['POST'])
def add_student():
    global student_id_counter
    new_student = request.json
    if 'name' not in new_student:
        return jsonify({'error': 'Student name is required'}), 400
    new_student['id'] = student_id_counter
    students.append(new_student)
    student_id_counter += 1
    return jsonify(new_student), 201

@app.route('/students/<int:student_id>', methods=['PUT'])
def update_student(student_id):
    student_to_update = None
    for student in students:
        if student['id'] == student_id:
            student_to_update = student
            break

    if student_to_update:
        update_data = request.json
        student_to_update.update(update_data)
        return jsonify(student_to_update)
    return jsonify({'error': 'Student not found'}), 404
```



```
!pip install Flask

from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/hello')
def hello():
    return jsonify({'message': 'Hello, AI Coding!'})

if __name__ == '__main__':
    # Use 0.0.0.0 to make it accessible externally in Colab
    # In a production environment, you might use 127.0.0.1 or a specific IP
    # For Colab, we'll use a port that is typically forwarded
    app.run(host='0.0.0.0', port=5000)
```

```
Requirement already satisfied: Flask in /usr/local/lib/python3.12/dist-packages (3.1.2)
Requirement already satisfied: blinker>=1.9.0 in /usr/local/lib/python3.12/dist-packages (from Flask) (1.9.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.12/dist-packages (from Flask) (8.3.0)
Requirement already satisfied: itsdangerous>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from Flask) (2.2.0)
Requirement already satisfied: jinja2>=3.1.2 in /usr/local/lib/python3.12/dist-packages (from Flask) (3.1.6)
Requirement already satisfied: markupsafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from Flask) (3.0.3)
Requirement already satisfied: werkzeug>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from Flask) (3.1.3)
 * Serving Flask app '__main__'
 * Debug mode: off
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.28.0.12:5000
INFO:werkzeug:Press CTRL+C to quit
```

OUTPUT :

# The app.run() call is in the first cell (hGsiZlY04kWi), so we don't need it here.

# if __name__ == '__main__':

# app.run(host='0.0.0.0', port=5000)

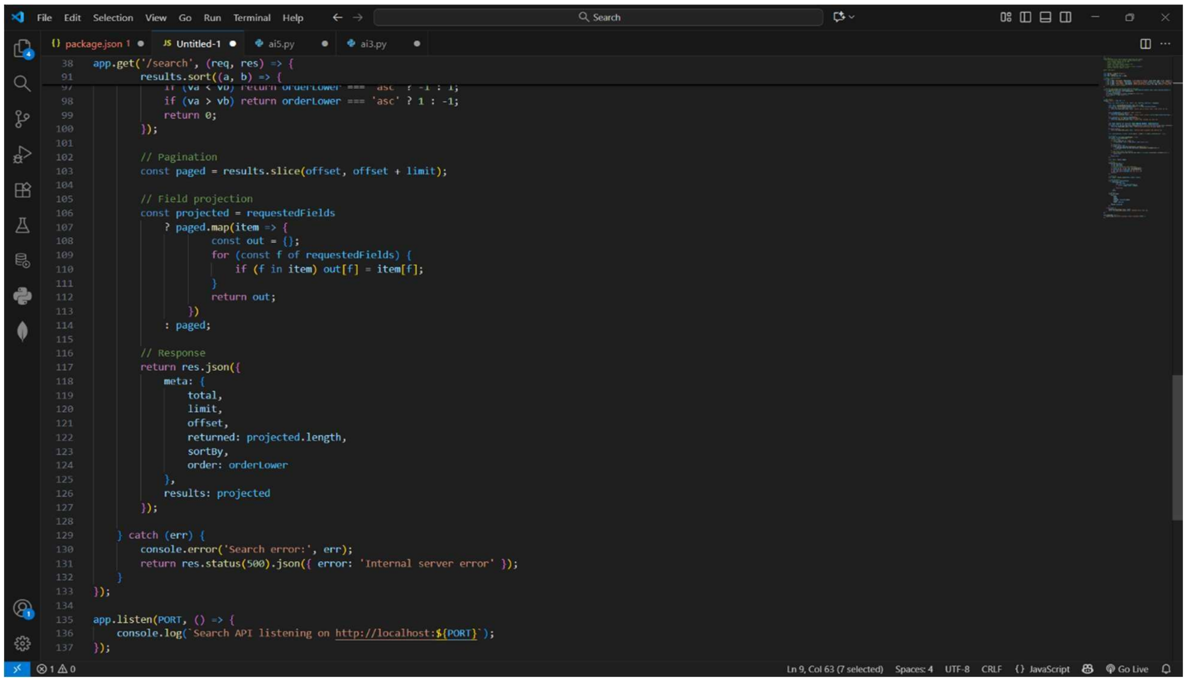
TASK 3 :

Ask AI to generate a REST API endpoint

PROMT :

"Create a REST API endpoint with query parameters for searching data. Include proper query param handling and return results in JSON."

CODE :



```
app.get('/search', (req, res) => {
        results.sort((a, b) => {
            if (va < vb) return orderLower === 'asc' ? -1 : 1;
            if (va > vb) return orderLower === 'asc' ? 1 : -1;
            return 0;
        });

        // Pagination
        const paged = results.slice(offset, offset + limit);

        // Field projection
        const projected = requestedFields
            ? paged.map(item => {
                const out = {};
                for (const f of requestedFields) {
                    if (f in item) out[f] = item[f];
                }
                return out;
            })
            : paged;

        // Response
        return res.json({
            meta: {
                total,
                limit,
                offset,
                returned: projected.length,
                sortBy,
                order: orderLower
            },
            results: projected
        });

    } catch (err) {
        console.error('Search error:', err);
        return res.status(500).json({ error: 'Internal server error' });
    }
});

app.listen(PORT, () => {
    console.log(`Search API listening on http://localhost:${PORT}`);
});
```

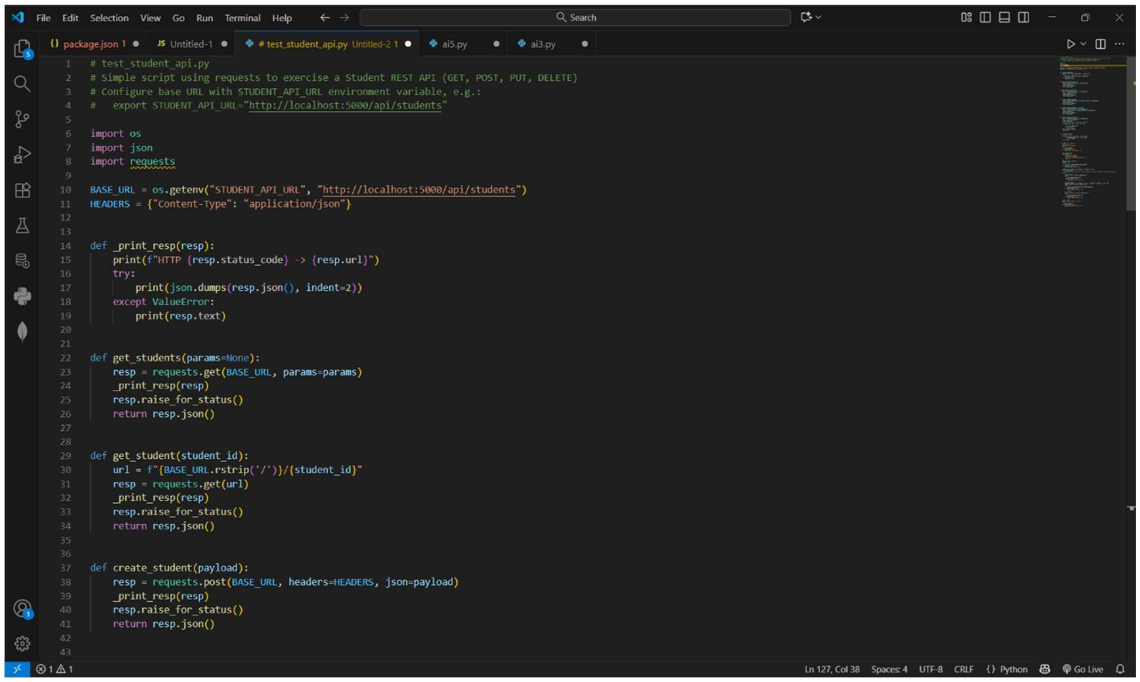
TASK 4 :

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

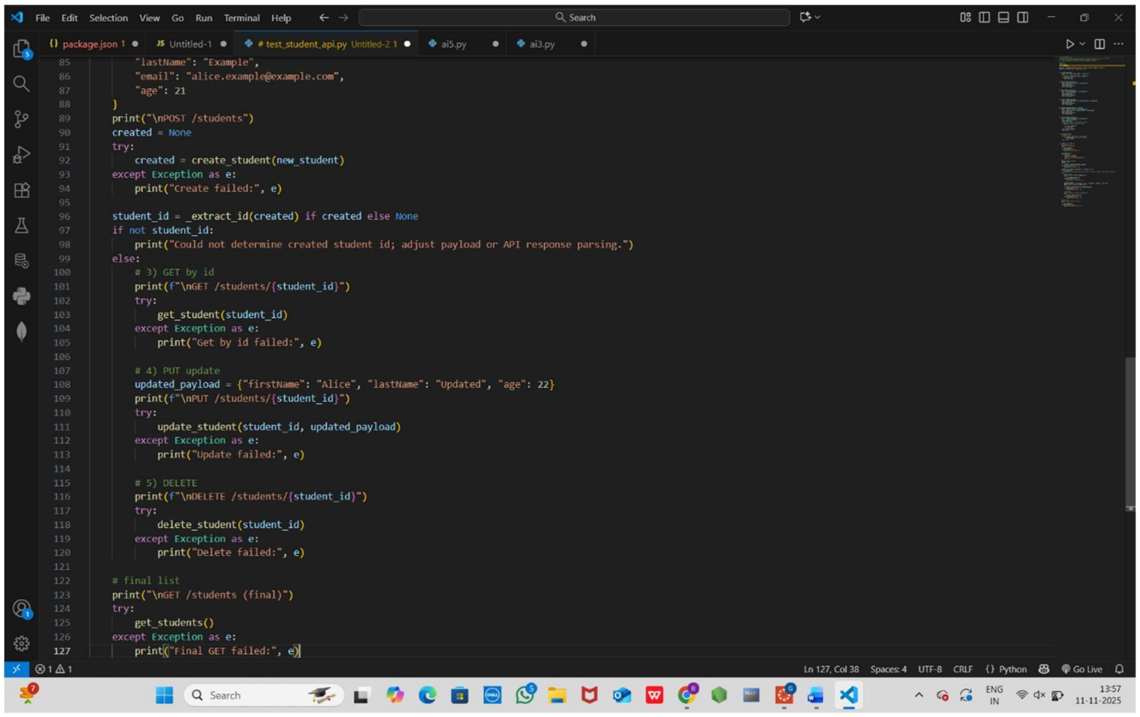
PROMPT :

"Write Python test scripts using the requests module to call REST API endpoints (GET, POST, PUT, DELETE) for the Student
API, and print the JSON responses."

CODE :



```python
# test_student_api.py
# Simple script using requests to exercise a Student REST API (GET, POST, PUT, DELETE)
# Configure base URL with STUDENT_API_URL environment variable, e.g.:
#   export STUDENT_API_URL="http://localhost:5000/api/students"

import os
import json
import requests

BASE_URL = os.getenv("STUDENT_API_URL", "http://localhost:5000/api/students")
HEADERS = {"Content-Type": "application/json"}


def _print_resp(resp):
    print(f"HTTP {resp.status_code} -> {resp.url}")
    try:
        print(json.dumps(resp.json(), indent=2))
    except ValueError:
        print(resp.text)


def get_students(params=None):
    resp = requests.get(BASE_URL, params=params)
    _print_resp(resp)
    resp.raise_for_status()
    return resp.json()


def get_student(student_id):
    url = f"{BASE_URL.rstrip('/')}/{student_id}"
    resp = requests.get(url)
    _print_resp(resp)
    resp.raise_for_status()
    return resp.json()


def create_student(payload):
    resp = requests.post(BASE_URL, headers=HEADERS, json=payload)
    _print_resp(resp)
    resp.raise_for_status()
    return resp.json()


```



```python
        "lastName": "Example",
        "email": "alice.example@example.com",
        "age": 21
    }
    print("\nPOST /students")
    created = None
    try:
        created = create_student(new_student)
    except Exception as e:
        print("Create failed:", e)

    student_id = _extract_id(created) if created else None
    if not student_id:
        print("Could not determine created student id; adjust payload or API response parsing.")
    else:
        # 3) GET by id
        print(f"\nGET /students/{student_id}")
        try:
            get_student(student_id)
        except Exception as e:
            print("Get by id failed:", e)

        # 4) PUT update
        updated_payload = {"firstName": "Alice", "lastName": "Updated", "age": 22}
        print(f"\nPUT /students/{student_id}")
        try:
            update_student(student_id, updated_payload)
        except Exception as e:
            print("Update failed:", e)

        # 5) DELETE
        print(f"\nDELETE /students/{student_id}")
        try:
            delete_student(student_id)
        except Exception as e:
            print("Delete failed:", e)

    # final list
    print("\nGET /students (final)")
    try:
        get_students()
    except Exception as e:
        print("Final GET failed:", e)
```

OUTPUT :

* Serving Flask app '__main__'

* Debug mode: off