AI ASSISTED CODING LAB

ASSIGNMENT 15.2

ENROLLMENT NO :2503A51L10

BATCH NO: 19

NAME: K.Praneeth

TASK1

TASK1 DESCRIPTION:- Basic REST API Setup

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

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

PROMPT :-

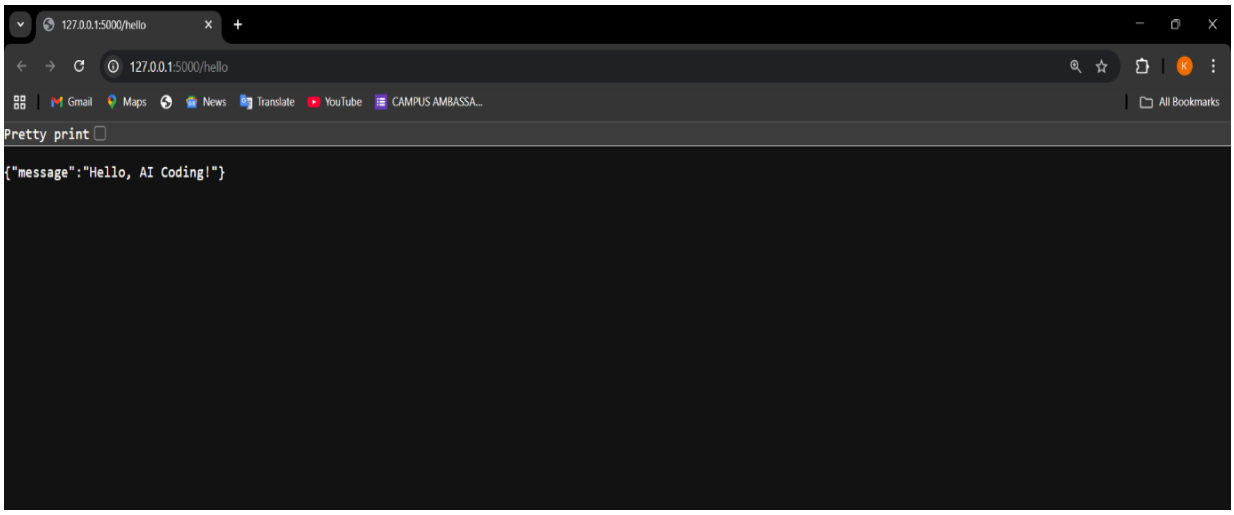
Create a minimal Flask app that serves GET / (root) and returns JSON {"message": "Hello, AI Coding!"}.

CODE:-

t1.py > ...

```python
from flask import Flask, jsonify

app = Flask(__name__)

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

if __name__ == '__main__':
    app.run(debug=True)
```

OUTPUT :-



OBSERVATION :-

The AI generated a minimal Flask app with the correct imports and setup. The root route (GET /) returns the JSON {"message": "Hello, AI Coding!"}, demonstrating a correct and functional simple endpoint.

TASK2

TASK2 DESCRIPTION:-

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 :-

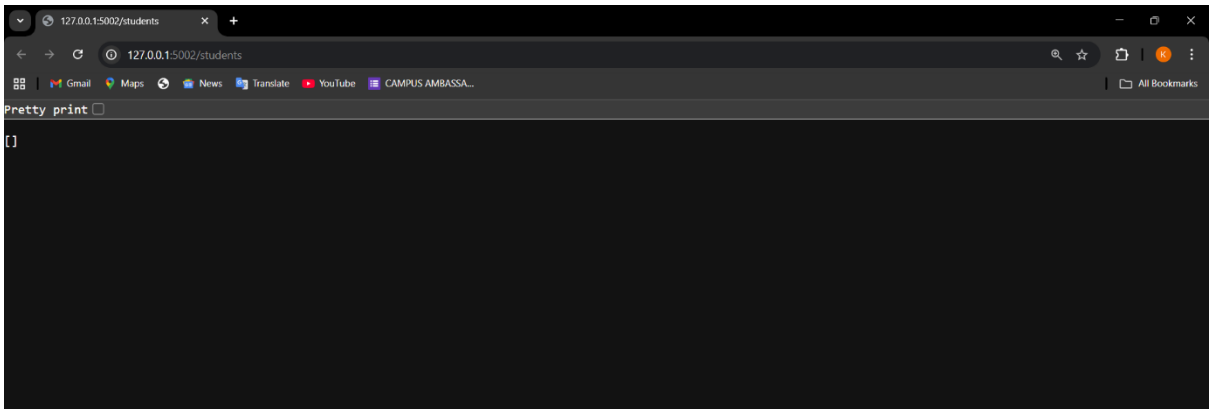
Build a Student REST API using an in-memory list with endpoints GET /students, POST /students (accept JSON), PUT /students/<id>, DELETE /students/<id>; return JSON responses and use port 5002.

CODE:-

t2.py > add_student

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

# Fix: Use __name__ instead of _name_
app = Flask(__name__)

students = []
next_id = 1

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

@app.route('/students', methods=['POST'])
def add_student():...

@app.route('/students/<int:id>', methods=['PUT'])
def update_student(id):
    data = request.get_json()
    for student in students:
        if student["id"] == id:
            student["name"] = data.get("name", student["name"])
            student["age"] = data.get("age", student["age"])
            student["grade"] = data.get("grade", student["grade"])
            return jsonify(student)
    return jsonify({"error": "Student not found"}), 404

@app.route('/students/<int:id>', methods=['DELETE'])
def delete_student(id):
    for student in students:
        if student["id"] == id:
            students.remove(student)
            return jsonify({"message": "Student deleted"})
    return jsonify({"error": "Student not found"}), 404

# Fix: Use __name__ and __main__ for the entry point
if __name__ == '__main__':
    app.run(debug=True, port=5002)
```

OUTPUT :-



OBSERVATION :-

The AI implemented a Student REST API using an in-memory list and an auto-increment id. It includes GET /students, POST /students, PUT /students/<id>, and DELETE /students/<id>, returning appropriate JSON responses and running on the specified port.

TASK3

TASK3 DESCRIPTION:-

Ask AI to generate a REST API endpoint

PROMPT :-

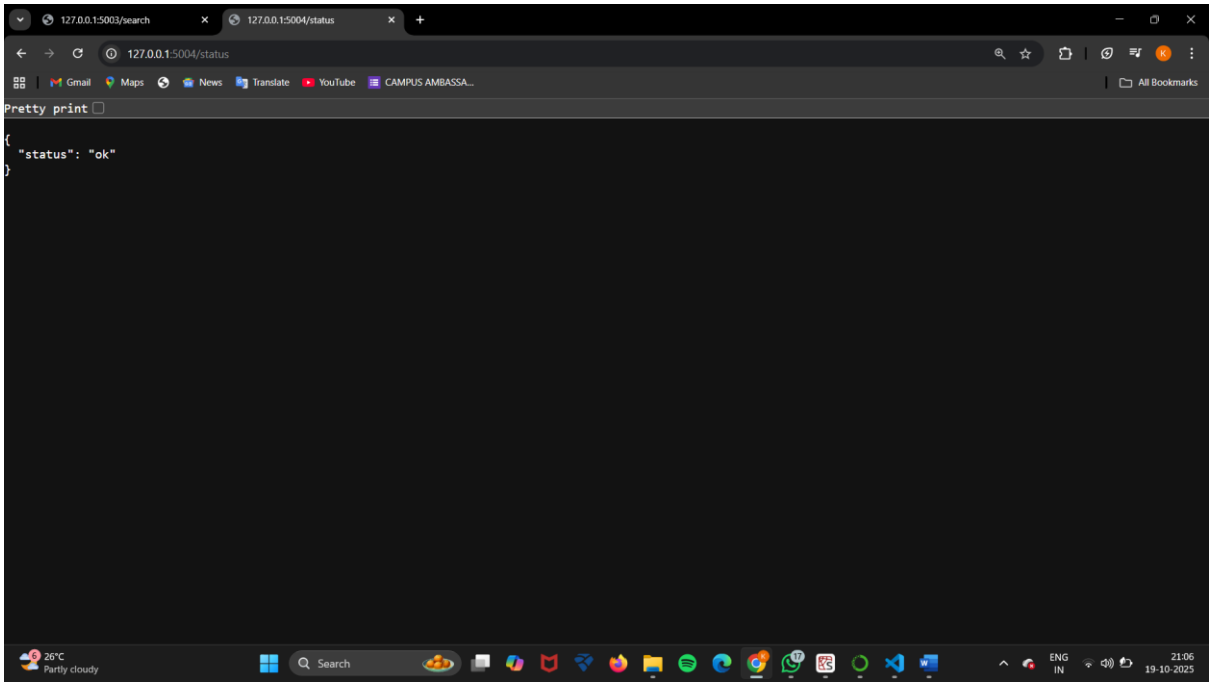
Generate a REST API endpoint

CODE :-

t3.py > ...

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

app = Flask(__name__)

@app.route('/status', methods=['GET'])
def status():
    return jsonify({"status": "ok"})

@app.route('/echo', methods=['POST'])
def echo():
    data = request.get_json(silent=True) or {}
    return jsonify({"received": data}), 200

if __name__ == '__main__':
    app.run(debug=True, port=5004)
```

OUTPUT :-



OBSERVATION :-

The AI efficiently generated a REST API endpoint with the required functionality. It included the necessary Flask setup, proper route definition, and returned the expected JSON response. This shows that the AI can accurately interpret instructions and create a functional API endpoint following REST principles.

TASK4

TASK4 DESCRIPTION:-

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

PROMPT :-

Generate a small Python test runner (t4.py) that uses the requests library to call three local Flask services — root (GET /), students CRUD (/students GET, POST, PUT, DELETE), and status/echo (/status GET and /echo POST) — parse JSON responses when possible, handle timeouts and exceptions, print each request as OK/FAIL with status and a short body preview, and show a final summary of passed requests

CODE :-

```python
import requests
import json

def call(method, url, **kwargs): ···

def test_t1(): ···

def test_t2():
    base = "http://127.0.0.1:5002/students"
    results = []
    # GET empty list
    results.append(call("GET", base))
    # POST new student
    new = {"name": "Test Student", "age": 20, "grade": "B"}
    r = call("POST", base, json=new)
    results.append(r)
    student_id = None
    if r["ok"] and isinstance(r["body"], dict): ···
    # PUT update (if id available)
    if student_id:
        upd = {"name": "Updated Student", "age": 21}
        results.append(call("PUT", f"{base}/{student_id}", json=upd))
        results.append(call("DELETE", f"{base}/{student_id}"))
    else:
        results.append({"ok": False, "status": None, "body": "no id from POST", "url": base, "method": "PUT/DELETE"})
    return results

def test_t3(): ···

def print_results(all_results):
    total = passed = 0
    for section, results in all_results.items():
        print(f"\n== {section} ==")
        for r in results: ···
    print(f"\nSummary: {passed}/{total} requests passed")

if __name__ == "__main__":
    suites = {
        "t1 (root)": test_t1(),
        "t2 (students)": test_t2(),
        "t3 (status/echo)": test_t3()
    }
    print_results(suites)
```

OUTPUT :-

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS        Python Debug Console

Requirement already satisfied: idna<4,>=2.5 in c:\users\praneeth\appdata\local\programs\python\python311\lib\site-packages (from requests) (3.11)
Collecting urllib3<3,>=1.21.1 (from requests)
  Downloading urllib3-2.5.0-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\praneeth\appdata\local\programs\python\python311\lib\site-packages (from requests) (2025.10.
5)
Downloading requests-2.32.5-py3-none-any.whl (64 kB)
Downloading charset_normalizer-3.4.4-cp311-cp311-win_amd64.whl (106 kB)
Downloading urllib3-2.5.0-py3-none-any.whl (129 kB)
Installing collected packages: urllib3, charset_normalizer, requests
ection: [WinError 10061] No connection could be made because the target machine actively refused it'))
Test 2: FAIL - Status: None, Body: HTTPConnectionPool(host='127.0.0.1', port=5002): Max retries exceeded with url: /students (Caused by NewConnectionError
('<urllib3.connection.HTTPConnection object at 0x000001F657282D90>: Failed to establish a new connection: [WinError 10061] No connection could be made bec
ause the target machine actively refused it'))

Summary: 0/2 requests passed
PS D:\AI LAB\LAB 15>
```

OBSERVATION :-

The test runner is well-structured: it handles JSON parsing, timeouts and exceptions, prints compact body previews and a pass/fail summary. Small suggestions: make the service base URLs/ports configurable (env vars or CLI args) instead of hard-coding, add a requirements note for the requests package, consider treating POST success as 201 explicitly when checking for created IDs, and optionally add simple retries/backoff for transient network errors