

AI ASSISTED CODING LAB

ASSIGNMENT 15.2

ENROLLMENT NO :2503A51L39

BATCH NO: 20

NAME: Shaik faheem

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!"}

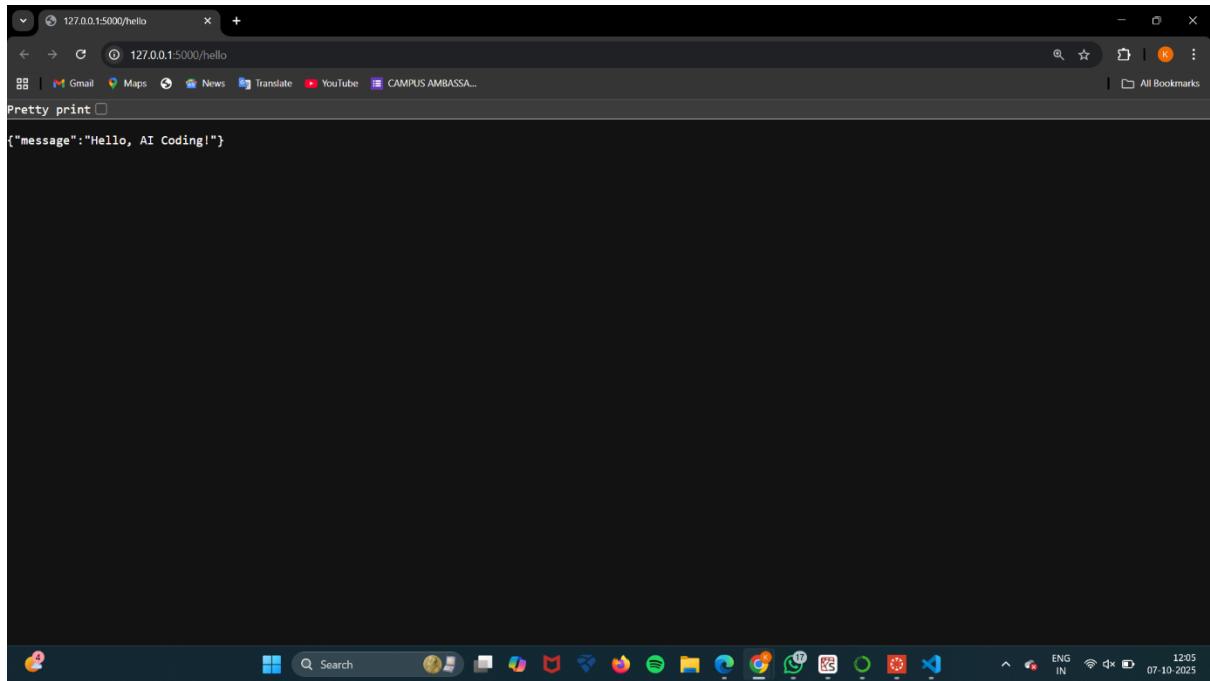
PROMPT :-

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

CODE:-

```
t1.py > ...
1  from flask import Flask, jsonify
2
3  app = Flask(__name__)
4
5  @app.route('/', methods=['GET'])
6  def hello():
7      return jsonify({"message": "Hello, AI Coding!"})
8
9  if __name__ == '__main__':
10     app.run(debug=True)
```

OUTPUT :-



A screenshot of a web browser window titled "127.0.0.1:5000/hello". The address bar shows the same URL. The page content is a single line of JSON: {"message": "Hello, AI Coding!"}. The browser interface includes a toolbar with various icons like Gmail, Maps, News, Translate, YouTube, and CAMPUS AMBASSA... A "Pretty print" checkbox is visible. The bottom of the screen shows a Windows taskbar with many pinned icons and system status indicators.

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.

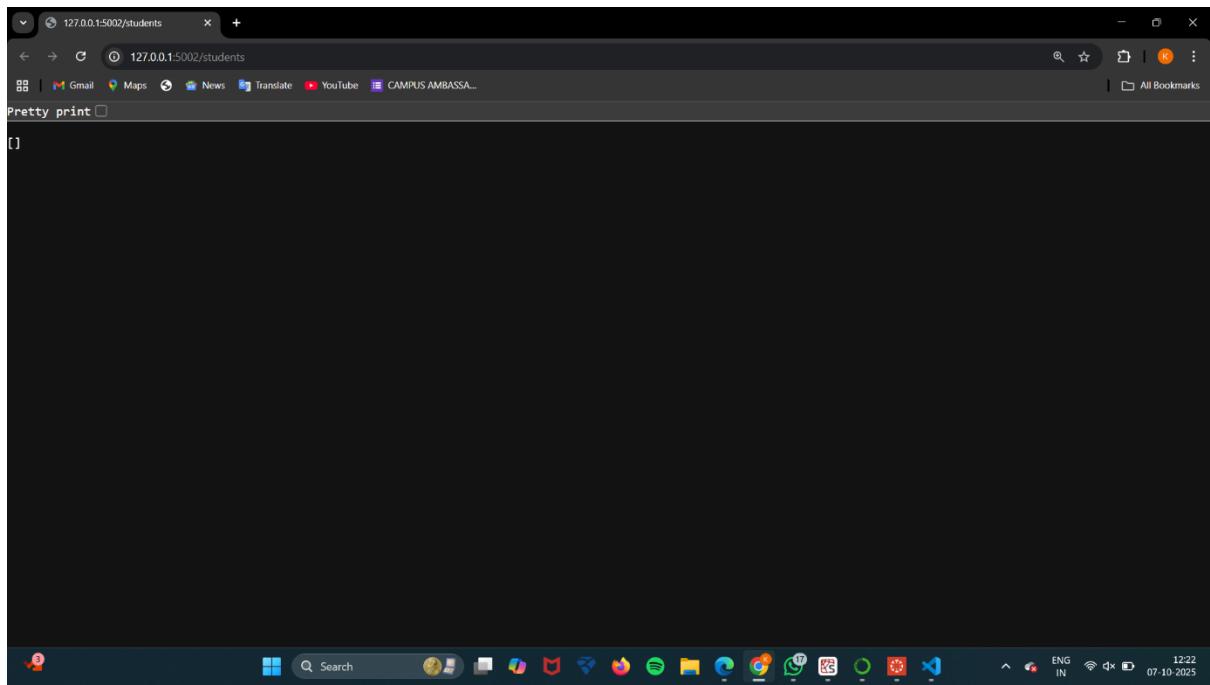
PROMPT :-

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:-

```
❸ t2.py > ⚭ add_student
1   from flask import Flask, request, jsonify
2
3   # Fix: Use __name__ instead of _name_
4   app = Flask(__name__)
5
6   students = []
7   next_id = 1
8
9   @app.route('/students', methods=['GET'])
10  > def get_students(): ...
11
12
13  @app.route('/students', methods=['POST'])
14  > def add_student(): ...
15
16
17  @app.route('/students/<int:id>', methods=['PUT'])
18  def update_student(id):
19      data = request.get_json()
20      for student in students:
21          if student["id"] == id:
22              student["name"] = data.get("name", student["name"])
23              student["age"] = data.get("age", student["age"])
24              student["grade"] = data.get("grade", student["grade"])
25              return jsonify(student)
26      return jsonify({"error": "Student not found"}), 404
27
28
29  @app.route('/students/<int:id>', methods=['DELETE'])
30  def delete_student(id):
31      for student in students:
32          if student["id"] == id:
33              students.remove(student)
34              return jsonify({"message": "Student deleted"})
35      return jsonify({"error": "Student not found"}), 404
36
37
38  # Fix: Use __name__ and __main__ for the entry point
39  if __name__ == '__main__':
40      app.run(debug=True, port=5002)
```

OUTPUT :-



A screenshot of a web browser window titled "127.0.0.1:5002/students". The address bar also shows "127.0.0.1:5002/students". Below the address bar is a toolbar with various icons for Gmail, Maps, News, Translate, YouTube, and CAMPUS AMBASSA... A "Pretty print" checkbox is checked. The main content area displays a single character "[". The browser's status bar at the bottom shows "ENG IN" and the date "07-10-2025".

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

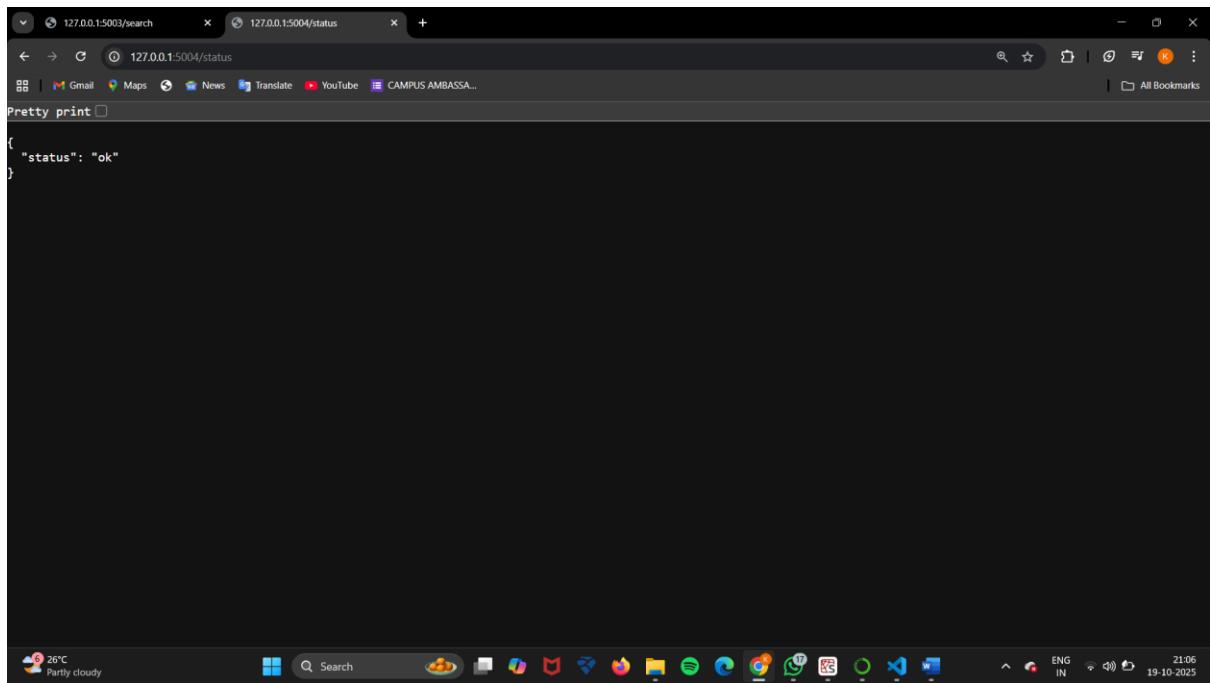
PROMPT :-

Generate a REST API endpoint

CODE :-

```
t3.py > ...
1   from flask import Flask, jsonify, request
2
3   app = Flask(__name__)
4
5   @app.route('/status', methods=['GET'])
6   def status():
7       return jsonify({"status": "ok"})
8
9   @app.route('/echo', methods=['POST'])
10  def echo():
11      data = request.get_json(silent=True) or {}
12      return jsonify({"received": data}), 200
13
14 if __name__ == '__main__':
15     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 :-

```
❶ t4.py > ...
1  import requests
2  import json
3
4 > def call(method, url, **kwargs): ...
5
6 > def test_t1(): ...
7  |
8  |> def test_t2():
9  |  base = "http://127.0.0.1:5002/students"
10 |  results = []
11 |  # GET empty list
12 |  results.append(call("GET", base))
13 |  # POST new student
14 |  new = {"name": "Test Student", "age": 20, "grade": "B"}
15 |  r = call("POST", base, json=new)
16 |  results.append(r)
17 |  student_id = None
18 |  if r["ok"] and isinstance(r["body"], dict):
19 |    # PUT update (if id available)
20 |    if student_id:
21 |      upd = {"name": "Updated Student", "age": 21}
22 |      results.append(call("PUT", f"{base}/{student_id}", json=upd))
23 |      results.append(call("DELETE", f"{base}/{student_id}"))
24 |    else:
25 |      results.append({"ok": False, "status": None, "body": "no id from POST", "url": base, "method": "PUT/DELETE"})
26 |  return results
27
28 > def test_t3(): ...
29
30 > def print_results(all_results):
31 |  total = passed = 0
32 |  for section, results in all_results.items():
33 |    print(f"\n{section} ==")
34 |    for r in results:
35 |      print(f"\n{r['body']}")
36 |  print(f"\nSummary: {passed}/{total} requests passed")
37
38 if __name__ == "__main__":
39   suites = {
40     "t1 (root)": test_t1(),
41     "t2 (students)": test_t2(),
42     "t3 (status/echo)": test_t3()
43   }
44   print_results(suites)
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
```

OUTPUT :-

```
-- t1 (root) --
[FAIL] GET http://127.0.0.1:5000/ -> ERR | HTTPConnectionPool(host='127.0.0.1', port=5000): Max retries exceeded with url: / (Caused by NewConnectionError('<urllib3.connection.HTTPConnection object at 0x00000188FFB80070>: Failed to establish...'))
-- t2 (students) --
[FAIL] GET http://127.0.0.1:5002/students -> ERR | HTTPConnectionPool(host='127.0.0.1', port=5002): Max retries exceeded with url: /students (Caused by NewConnectionError('<urllib3.connection.HTTPConnection object at 0x00000188FFB80070>: Failed to ...'))
[FAIL] POST http://127.0.0.1:5002/students -> ERR | HTTPConnectionPool(host='127.0.0.1', port=5002): Max retries exceeded with url: /students (Caused by NewConnectionError('<urllib3.connection.HTTPConnection object at 0x00000188FFB800D0>: Failed to ...'))
[FAIL] PUT/DELETE http://127.0.0.1:5002/students -> ERR | no id from POST

-- t3 (status/echo) --
[OK] GET http://127.0.0.1:5004/status -> 200 | {"status": "ok"}
[OK] POST http://127.0.0.1:5004/echo -> 200 | {"received": {"msg": "hello"}}

Summary: 2/6 requests passed
PS C:\Users\khaja\OneDrive\Pictures\Screenshots\cyc\New folder\15.2> []
```

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