

ASSIGNMENT -15.3

NAME :S SAI RUCHITHA

HALL TICKET 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

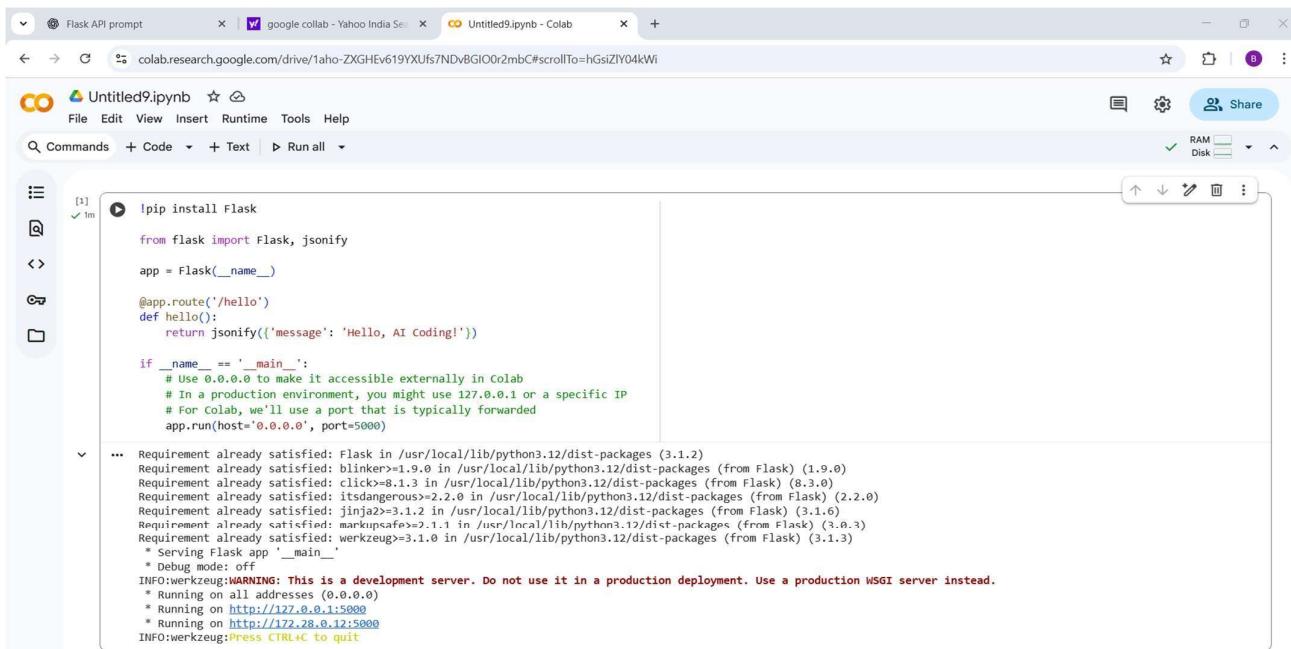
/hello → returns {"message": "Hello, AI Coding!"}

PROMPT :

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

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

CODE :



The screenshot shows a Google Colab interface with a code cell containing Python code for a Flask application. The code imports Flask and jsonify, creates a Flask app, defines a route '/hello' that returns a JSON response with the message 'Hello, AI Coding!', and runs the app on port 5000. The output shows the Flask application is running on multiple addresses (0.0.0.0, 127.0.0.1, and 172.28.0.12) on port 5000. A warning message from werkzeug states: "INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead." It also lists three running instances.

```
!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 <http://127.0.0.1:5000>

* Running on <http://172.28.0.12:5000>

INFO:werkzeug:Press CTRL+C to quit

TASK 2 :

Use AI to build REST endpoints for a Student API:

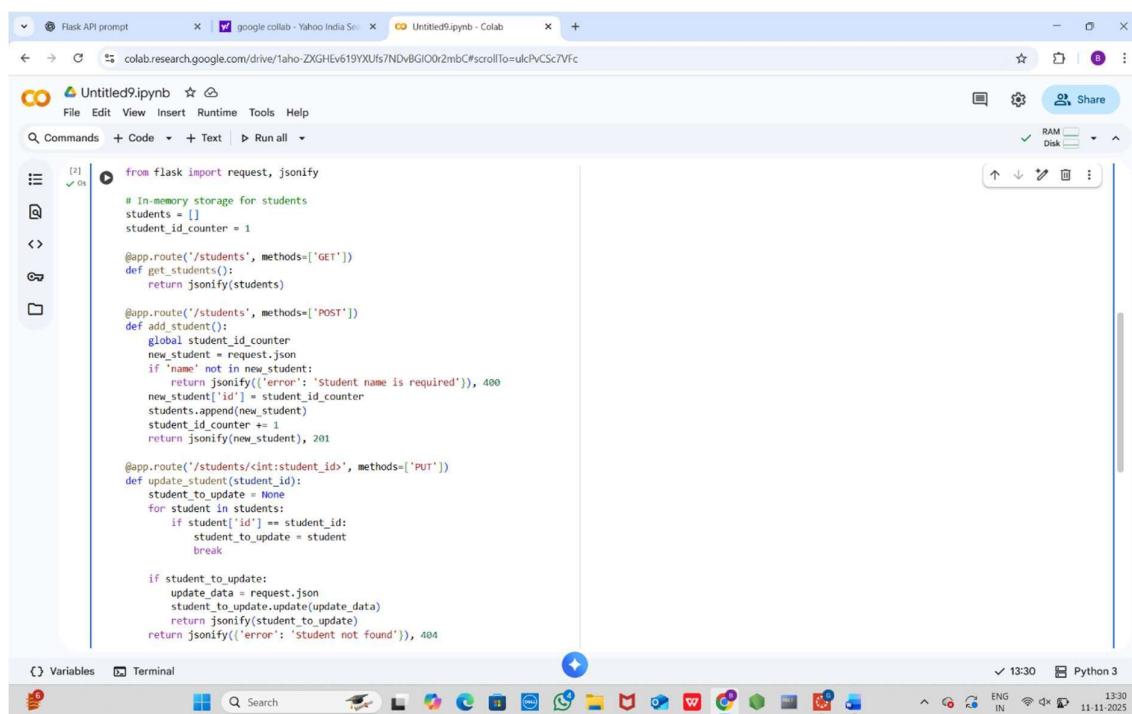
- GET /students → List all students.
- POST /students → Add a new student.
- PUT /students/<id> → Update student details.

- 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 :



```

Flask API prompt      google colab - Yahoo India Se... Untitled9.ipynb - Colab
← → ⌂ colab.research.google.com/drive/1aho-ZXGHEv619YXUfs7NDvBGIOr2mbC#scrollTo=ulcPvCSc7VFc
Untitled9.ipynb ★ ⓘ
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▶ Run all ▾
[2] ✓ 0s
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

```

The screenshot shows a Google Colab notebook titled "Untitled9.ipynb". The code implements a simple Flask REST API for managing students. It uses an in-memory list to store student data. The API includes endpoints for getting all students (GET /students), adding a new student (POST /students), and updating a specific student (PUT /students/<id>). The student ID is generated sequentially starting from 1. The code uses the Flask framework and the jsonify function from the flask library to handle JSON data.

```

[1] !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>2.0.0 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://127.0.0.1:5000
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)

```

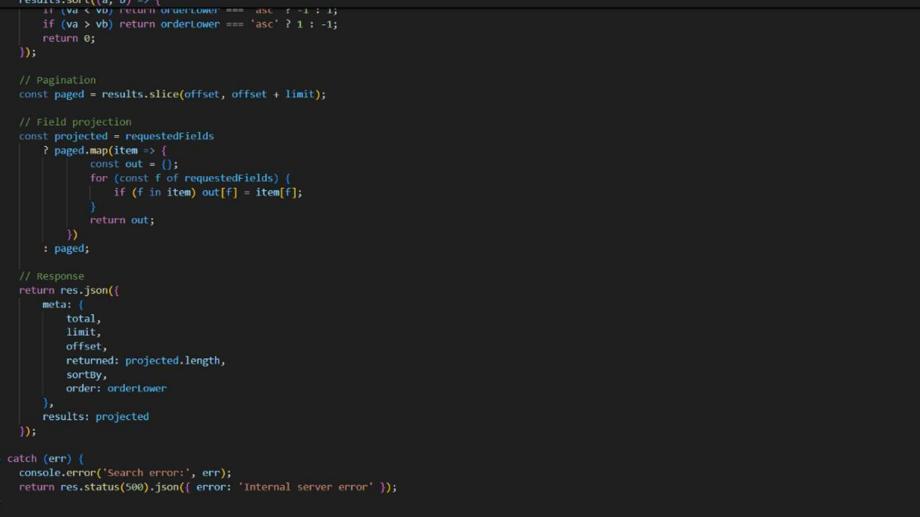
TASK 3 :

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 :



A screenshot of a code editor showing a Node.js file named `ai1.py`. The code implements a search endpoint that sorts results based on query parameters and paginates them. It also handles errors and logs the port number when the application starts.

```
File Edit Selection View Go Run Terminal Help ← → Search

package.json 1 Untitled-1 ai1.py ai3.py

38 app.get('/search', (req, res) => {
39   results.sort((a, b) => {
40     if (va < vb) return orderLower === 'desc' ? -1 : 1;
41     if (va > vb) return orderLower === 'asc' ? 1 : -1;
42     return 0;
43   });
44
45   // Pagination
46   const paged = results.slice(offset, offset + limit);
47
48   // Field projection
49   const projected = requestedFields.map(item => {
50     const out = {};
51     for (const f of requestedFields) {
52       if (f in item) out[f] = item[f];
53     }
54     return out;
55   });
56   const paged = paged.map(item => {
57     const out = {};
58     for (const f of requestedFields) {
59       if (f in item) out[f] = item[f];
60     }
61     return out;
62   });
63
64   // Response
65   return res.json({
66     meta: {
67       total,
68       limit,
69       offset,
70       returned: projected.length,
71       sortBy,
72       order: orderLower
73     },
74     results: projected
75   });
76
77 } catch (err) {
78   console.error('Search error:', err);
79   return res.status(500).json({ error: 'Internal server error' });
80 }
81
82 app.listen(PORT, () => {
83   console.log(`Search API listening on http://localhost:${PORT}`);
84 });

Ln 9, Col 63 (7 selected) Spaces: 4 UTF-8 CRLF {} JavaScript ⚡ Go Live
```

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 :

```
1 # test_student_api.py
2 # Simple script using requests to exercise a student REST API (GET, POST, PUT, DELETE)
3 # Configure base URL with STUDENT_API_URL environment variable, e.g.:
4 # export STUDENT_API_URL="http://localhost:5000/api/students"
5
6 import os
7 import json
8 import requests
9
10 BASE_URL = os.getenv("STUDENT_API_URL", "http://localhost:5000/api/students")
11 HEADERS = {"Content-Type": "application/json"}
12
13 def _print_resp(resp):
14     print(f"HTTP {resp.status_code} - {resp.url}")
15     try:
16         print(json.dumps(resp.json(), indent=2))
17     except ValueError:
18         print(resp.text)
19
20
21 def get_students(params=None):
22     resp = requests.get(BASE_URL, params=params)
23     _print_resp(resp)
24     resp.raise_for_status()
25     return resp.json()
26
27
28 def get_student(student_id):
29     url = f"{BASE_URL.rstrip('/')}/{student_id}"
30     resp = requests.get(url)
31     _print_resp(resp)
32     resp.raise_for_status()
33     return resp.json()
34
35
36 def create_student(payload):
37     resp = requests.post(BASE_URL, headers=HEADERS, json=payload)
38     _print_resp(resp)
39     resp.raise_for_status()
40     return resp.json()
41
42
43
```

```
85     "lastName": "Example",
86     "email": "alice.example@example.com",
87     "age": 21
88 }
89 print("\nPOST /students")
90 created = None
91 try:
92     created = create_student(new_student)
93 except Exception as e:
94     print("Create failed:", e)
95
96 student_id = extract_id(created) if created else None
97 if not student_id:
98     print("Could not determine created student id; adjust payload or API response parsing.")
99 else:
100     # 3) GET by id
101     print("\nGET /students/{student_id}")
102     try:
103         get_student(student_id)
104     except Exception as e:
105         print("Get by id failed:", e)
106
107     # 4) PUT update
108     updated_payload = {"firstName": "Alice", "lastName": "Updated", "age": 22}
109     print("\nPUT /students/{student_id}")
110     try:
111         update_student(student_id, updated_payload)
112     except Exception as e:
113         print("Update failed:", e)
114
115     # 5) DELETE
116     print("\nDELETE /students/{student_id}")
117     try:
118         delete_student(student_id)
119     except Exception as e:
120         print("Delete failed:", e)
121
122     # Final list
123     print("\nGET /students (final)")
124     try:
125         get_students()
126     except Exception as e:
127         print("Final GET failed:", e)
```

OUTPUT :

- * Serving Flask app '__main__'
- * Debug mode: off