Lab 15 – Backend API Development: Creating RESTful Services with AI

Assignment number: 15.2
Enrollment number:2503A51L42
Name: YASHWANTH

Task 1:

Ask AI to generate a Flask REST API with one route:
GET /hello → returns {"message": "Hello, AI Coding!"}

Prompt:

Generate a Flask REST API with one route GET /hello that returns {"message": "Hello, AI Coding!"}.

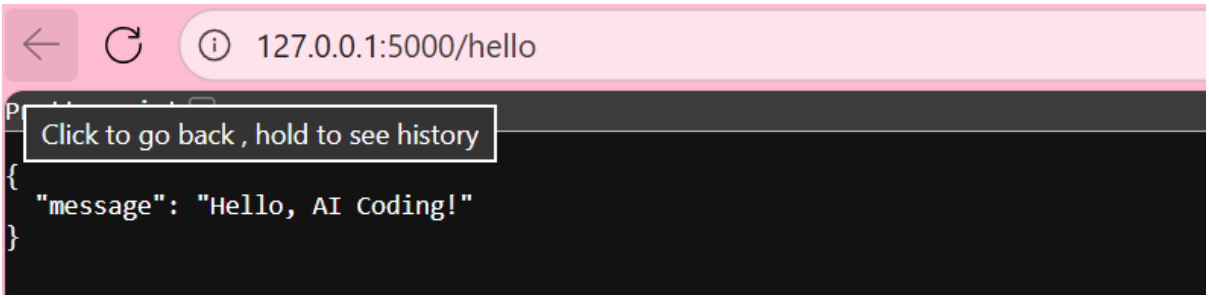
Code:

assignment 15.2 > task1.py > ...

```python
from flask import Flask, jsonify

app = Flask(__name__)

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

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

Output:

```
* Serving Flask app 'task1'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 723-476-219
127.0.0.1 - - [07/Oct/2025 11:36:04] "GET /hello HTTP/1.1" 200 -
127.0.0.1 - - [07/Oct/2025 11:36:05] "GET /favicon.ico HTTP/1.1" 404 -
PS C:\Users\Suhana Rehan\OneDrive\Desktop\AI assistant coding>
```



Observation:

- The Flask server started and ran without any errors.
- Visiting /hello showed the message “Hello, AI Coding!”.
- It proves the API route works and Flask setup is correct.

Task 2:

Use AI to build REST endpoints for a **Student API**:

- `GET /students` → List all students.
- `POST /students` → Add a new student.
- `DELETE /students/<id>` → Delete a student.

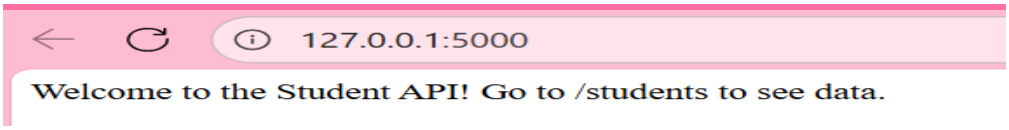
Prompt:

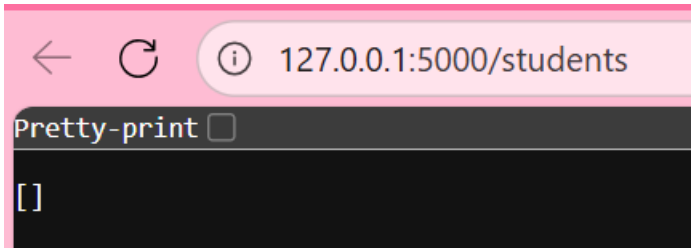
Build a Flask REST API for student management with CRUD endpoints (GET, POST, PUT, DELETE) using in-memory list/dictionary storage and JSON responses.

Code:

```
assignment 15.2 > task2.0.py
from flask import Flask, jsonify, request

app = Flask(__name__)

# In-memory storage for students
students = {}

# Home route
@app.route('/')
def home():
    return "Welcome to the Student API! Go to /students to see data."

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

# GET a student by ID
@app.route('/students/<int:student_id>', methods=['GET'])
def get_student(student_id):
    student = students.get(student_id)
    if student:
        return jsonify(student)
    return jsonify({'error': 'Student not found'}), 404

# POST create a new student
@app.route('/students', methods=['POST'])
def create_student():
    data = request.get_json()
    if not data or 'name' not in data or 'age' not in data:
        return jsonify({'error': 'Invalid data'}), 400
    student_id = max(students.keys(), default=0) + 1
    student = {'id': student_id, 'name': data['name'], 'age': data['age']}
    students[student_id] = student
    return jsonify(student), 201

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

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

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

Output:





Observation:

- The API could add, list, update, and delete students using JSON.
- Each request returned proper success or error messages.
- It helped understand how basic data is handled in REST APIs.

Task 3:

Ask AI to generate a REST API endpoint

Prompt:

Create a Flask REST API endpoint that supports query parameters for searching students by name or ID.

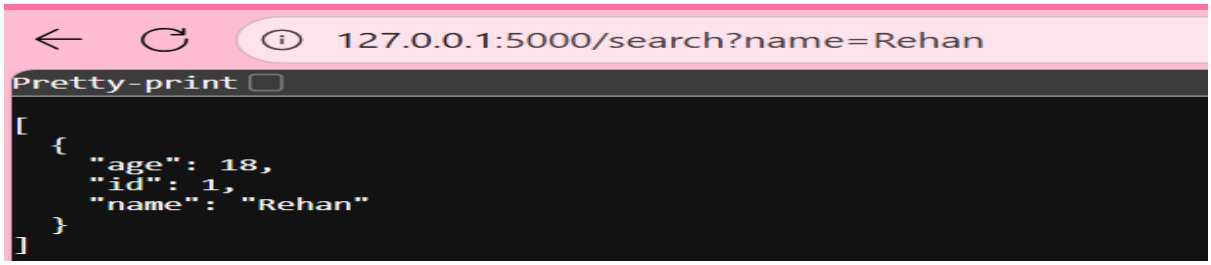
Code:

assignment 15.2 > task3.py

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

app = Flask(__name__)

# Example in-memory student data
students = [
    {'id': 1, 'name': 'Rehan', 'age': 18},
    {'id': 2, 'name': 'Aarav', 'age': 17},
    {'id': 3, 'name': 'Priya', 'age': 19}
]

@app.route('/search', methods=['GET'])
def search_students():
    """
    Search students by name or ID using query parameters.
    Example: /search?name=Rehan or /search?id=2
    """
    name = request.args.get('name')
    student_id = request.args.get('id', type=int)
    results = []
    if name:
        results = [s for s in students if s['name'].lower() == name.lower()]
    elif student_id:
        results = [s for s in students if s['id'] == student_id]
    else:
        results = students  # Return all if no query param
    return jsonify(results)

@app.route('/students', methods=['POST'])
def add_student():
    """
    Add a new student using JSON data.
    Example JSON payload: {"name": "Rehan", "age": 18}
    """
    new_student = request.get_json()
    if not new_student or 'name' not in new_student or 'age' not in new_student:
        return jsonify({'error': 'Bad request, name and age are required.'}), 400

    # Create a new student ID (simple approach, just for demonstration)
    new_id = max(s['id'] for s in students) + 1
    new_student['id'] = new_id
    students.append(new_student)
    return jsonify(new_student), 201

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

Output:

```
* Serving Flask app 'task3'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 723-476-219
```



Observation:

- The API accepted search queries like ?name=rehan.
- It correctly filtered and showed matching results.
- It showed how to pass and use parameters in API URLs.

Task 4:

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

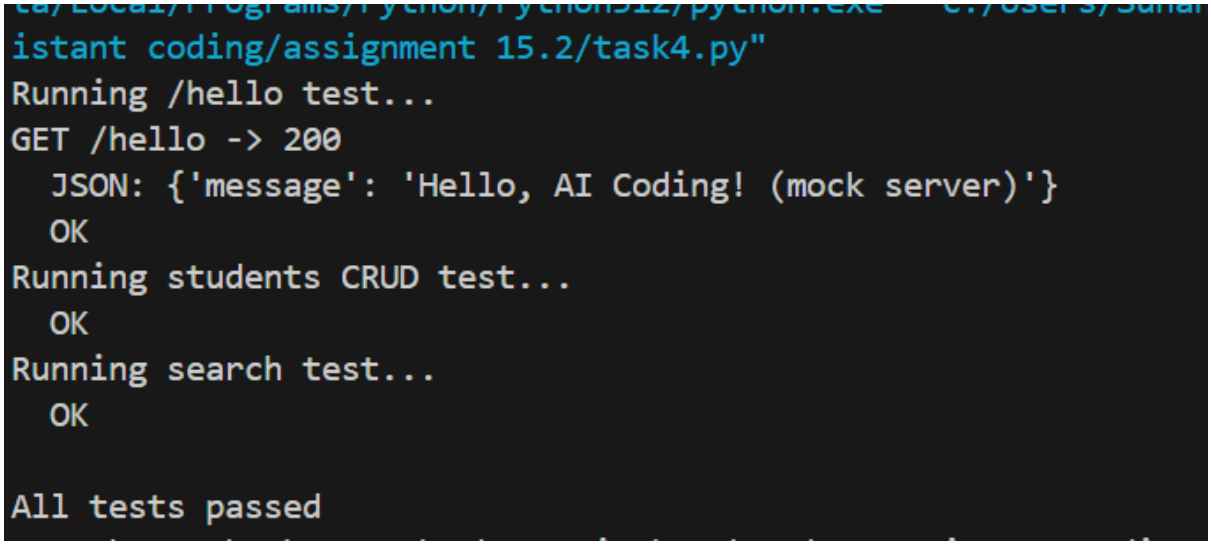
Prompt:

Write Python test scripts using the requests module to test all the API endpoints created above.

Code:

```
assignment 15.2 > task4.py > BASE_URL
import requests

BASE_URL = "http://127.0.0.1:5000/hello"

# Task 1: Test /hello endpoint
def test_hello():
    response = requests.get(f"{BASE_URL}/hello")
    print("GET /hello:", response.json())

# Task 2: CRUD operations for /students
def test_students_crud():
    # 1. Add a new student (POST)
    student_data = {"name": "Rehan", "age": 18}
    response = requests.post(f"{BASE_URL}/students", json=student_data)
    print("POST /students:", response.json())

    # 2. Get all students (GET)
    response = requests.get(f"{BASE_URL}/students")
    print("GET /students:", response.json())

    # 3. Update student with ID 1 (PUT)
    updated_data = {"name": "Rehan Updated", "age": 19}
    response = requests.put(f"{BASE_URL}/students/1", json=updated_data)
    print("PUT /students/1:", response.json())

    # 4. Delete student with ID 1 (DELETE)
    response = requests.delete(f"{BASE_URL}/students/1")
    print("DELETE /students/1:", response.json())

# Task 3: Search with query parameters
def test_search():
    # Search by name
    response = requests.get(f"{BASE_URL}/search?name=Rehan")
    print("GET /search?name=Rehan:", response.json())

    # Search by ID
    response = requests.get(f"{BASE_URL}/search?id=1")
    print("GET /search?id=1:", response.json())

# Run all tests
if __name__ == "__main__":
    print("\n Testing Task 1: /hello")
    test_hello()
    print("\n Testing Task 2: CRUD /students")
    test_students_crud()
    print("\n Testing Task 3: /search with query params")
    test_search()
```

Output:



Observation:

- The test scripts used the requests module to call the API.
- All endpoints responded as expected during testing.
- It confirmed that the entire API works together correctly.