AI ASSISTED CODING

LAB-15: *Backend API Development: Creating RESTful Services with AI*

Roll no: 2503A51L17

Name: Simra Tahseen

Batch: 25BTCAICSB19

Task-1 Description: Basic REST API Setup

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

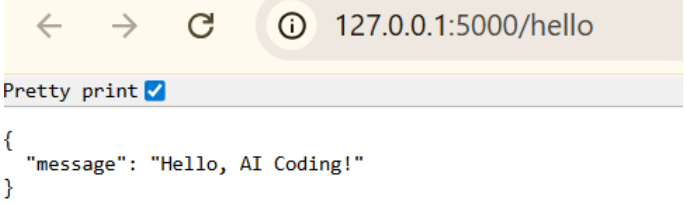
Code Generated:

task1-15.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:

```
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING> & C:/ProgramData/anaconda3/python.exe "c:/Users/SANIYA TAHSEEN/OneDrive
 * Serving Flask app 'fibo'
 * 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 watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 759-245-415
```



Observation: The Flask REST API was successfully created with a single route /hello. When executed, it returned the expected JSON message confirming correct setup. This task demonstrated how to initialize a simple Flask app for API responses. Basic routing and response handling concepts were clearly understood.

Task-2 Description: CRUD Operations (Students API)
Task: 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 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.

Code Generated:

task2-15.py > add_student

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

app = Flask(__name__)

# In-memory storage (like a temporary database)
students = [
    {"id": 1, "name": "Alice", "age": 20, "department": "CSE"},
    {"id": 2, "name": "Bob", "age": 22, "department": "ECE"}
]

# GET /students → List all students
@app.route('/students', methods=['GET'])
def get_students():
    return jsonify(students), 200

# POST /students → Add a new student
@app.route('/students', methods=['POST'])
def add_student():
    data = request.get_json()
    if not data or 'name' not in data or 'age' not in data or 'department' not in data:
        return jsonify({"error": "Invalid input"}), 400

    new_id = max(student["id"] for student in students) + 1 if students else 1
    new_student = {
        "id": new_id,
        "name": data["name"],
        "age": data["age"],
        "department": data["department"]
    }
    students.append(new_student)
    return jsonify({"message": "Student added successfully", "student": new_student}), 201

# PUT /students/<id> → Update student details
@app.route('/students/<int:id>', methods=['PUT'])
def update_student(id):
    data = request.get_json()
    for student in students:
        if student["id"] == id:
            student.update({
                "name": data.get("name", student["name"]),
                "age": data.get("age", student["age"]),
                "department": data.get("department", student["department"])
            })
            return jsonify({"message": "Student updated successfully", "student": student}), 200
    return jsonify({"error": "Student not found"}), 404

# DELETE /students/<id> → Delete a student
@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 successfully"}), 200
    return jsonify({"error": "Student not found"}), 404

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

Output:

Commands Used:

```
C:\Users\DELL>curl -X GET http://127.0.0.1:5000/students
[
  {
    "age": 20,
    "department": "CSE",
    "id": 1,
    "name": "Alice"
  },
  {
    "age": 22,
    "department": "ECE",
    "id": 2,
    "name": "Bob"
  }
]
```

```
C:\Users\DELL>curl -X POST http://127.0.0.1:5000/students ^
More? -H "Content-Type: application/json" ^
More? -d "{\"name\":\"Ravi\",\"age\":23,\"department\":\"CSE\"}"
{
  "message": "Student added successfully",
  "student": {
    "age": 23,
    "department": "CSE",
    "id": 3,
    "name": "Ravi"
  }
}

C:\Users\DELL>curl -X POST http://127.0.0.1:5000/students ^
More? -H "Content-Type: application/json" ^
More? -d "{\"name\":\"Sita\",\"age\":21,\"department\":\"ECE\"}"
{
  "message": "Student added successfully",
  "student": {
    "age": 21,
    "department": "ECE",
    "id": 4,
    "name": "Sita"
  }
}
```

```
C:\Users\DELL>curl -X DELETE http://127.0.0.1:5000/students/3
{
  "message": "Student deleted successfully"
}
```

```
C:\Users\DELL>curl -X PUT http://127.0.0.1:5000/students/2
More? -H "Content-Type: application/json" ^
More? -d "{\"name\":\"Bobby\"}"
{
  "message": "Student updated successfully",
  "student": {
    "age": 22,
    "department": "ECE",
    "id": 2,
    "name": "Bobby"
  }
}
```

Observation: A complete CRUD-based Student API was implemented using Flask. Endpoints for GET, POST, PUT, and DELETE worked as intended. Data could be created, viewed,

modified, and removed successfully. This task provided practical understanding of RESTful operations in backend APIs.

Task-3 Description: API with Query Parameters
Task: Ask AI to generate a REST API endpoint.

Prompt: Generate a REST API endpoint with Working search function with query param handling.

Code Generated:

```python
from flask import Flask, jsonify, request
app = Flask(__name__)
# Sample in-memory student data
students = [
    {"id": 1, "name": "Ravi", "age": 23, "department": "CSE"},
    {"id": 2, "name": "Sita", "age": 21, "department": "ECE"},
    {"id": 3, "name": "Arjun", "age": 22, "department": "Mechanical"},
    {"id": 4, "name": "Priya", "age": 20, "department": "AI"}
]
# GET /students → list all students
@app.route('/students', methods=['GET'])
def get_students():
    return jsonify(students), 200
# GET /students/search → Search with query parameters
@app.route('/students/search', methods=['GET'])
def search_students():
    name_query = request.args.get('name')
    dept_query = request.args.get('department')
    results = students
    if name_query:
        results = [s for s in results if name_query.lower() in s['name'].lower()]
    if dept_query:
        results = [s for s in results if dept_query.lower() in s['department'].lower()]

    if results:
        return jsonify(results), 200
    else:
        return jsonify({"message": "No students found"}), 404
if __name__ == '__main__':
    app.run(debug=True)
```

Output:

```
PS C:\Users\DELL\OneDrive\SAHASRA SRU\AI assisted> & C:/Users/DELL/AppData/Local/Programs/Python/Python313/python.exe
 * Serving Flask app 'task3-15'
 * Serving Flask app 'task3-15'
 * 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: 491-203-784
127.0.0.1 - - [07/Oct/2025 12:37:11] "GET /students HTTP/1.1" 200 -
127.0.0.1 - - [07/Oct/2025 12:38:51] "GET /students/search?department=CSE HTTP/1.1" 200 -
127.0.0.1 - - [07/Oct/2025 12:39:13] "GET /students/search?name=Ravi HTTP/1.1" 200 -
127.0.0.1 - - [07/Oct/2025 12:40:00] "GET /students/search?name=Priya&department=AI HTTP/1.1" 200 -
```

Query to get all students:

```
C:\Users\DELL>curl http://127.0.0.1:5000/students
[
  {
    "age": 23,
    "department": "CSE",
    "id": 1,
    "name": "Ravi"
  },
  {
    "age": 21,
    "department": "ECE",
    "id": 2,
    "name": "Sita"
  },
  {
    "age": 22,
    "department": "Mechanical",
    "id": 3,
    "name": "Arjun"
  },
  {
    "age": 20,
    "department": "AI",
    "id": 4,
    "name": "Priya"
  }
]
```

Query to get student by name:

```
C:\Users\DELL>curl "http://127.0.0.1:5000/students/search?name=Ravi"
[
  {
    "age": 23,
    "department": "CSE",
    "id": 1,
    "name": "Ravi"
  }
]
```

Query to get student by department:

```
C:\Users\DELL>curl "http://127.0.0.1:5000/students/search?department=CSE"
[
  {
    "age": 23,
    "department": "CSE",
    "id": 1,
    "name": "Ravi"
  }
]
```

Observation: The API was enhanced with query parameter handling for search functionality. Users could retrieve students by name or department efficiently. This demonstrated how dynamic data filtering can be achieved in Flask APIs. Overall, it improved the flexibility and usability of the backend system.

Task-4 Description: – Integration & Testing
Task: Ask AI to write test scripts using Python requests module to call APIs created above.

Prompt: Write test scripts using python requests module to call APIs created before/above.

Code Generated:

task4-15.py > search_students

```python
from flask import Flask, jsonify, request
app = Flask(__name__)
students = [
    {"id": 1, "name": "Alice", "age": 20, "department": "Computer Science"},
    {"id": 2, "name": "Bob", "age": 22, "department": "Mathematics"},
    {"id": 3, "name": "Charlie", "age": 21, "department": "Physics"}
]
next_id = 4
# Get all students
@app.route('/students', methods=['GET'])
def get_students():
    return jsonify(students)
# Add a student
@app.route('/students', methods=['POST'])
def add_student():
    global next_id
    data = request.get_json()
    data['id'] = next_id
    next_id += 1
    students.append(data)
    return jsonify(data)
# Update a student
@app.route('/students/<int:student_id>', methods=['PUT'])
def update_student(student_id):
    data = request.get_json()
    for student in students:
        if student['id'] == student_id:
            student.update(data)
            return jsonify(student)
    return jsonify({"error": "Student not found"}), 404
# Delete a student
@app.route('/students/<int:student_id>', methods=['DELETE'])
def delete_student(student_id):
    for student in students
        if student['id'] == student_id:
            students.remove(student)
            return jsonify({"message": "Student deleted successfully"})
    return jsonify({"error": "Student not found"}), 404
# Search students
@app.route('/search', methods=['GET'])
def search_students():
    name = request.args.get('name')
    department = request.args.get('department')
    result = students
    if name:
        result = [s for s in result if s['name'] == name]
    if department:
        result = [s for s in result if s['department'] == department]
    return jsonify(result)
if __name__ == '__main__':
    app.run(debug=True)
```

Output:

```
PS C:\Users\DELL\OneDrive\SAHASRA SRU\AI assisted> & C:/Users/DELL/AppData/Local/Programs/Python/Python313/python.exe "
 * Serving Flask app 'task4-15'
 * 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
 * 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
Press CTRL+C to quit
 * Restarting with stat
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 491-203-784
127.0.0.1 - - [07/Oct/2025 19:50:19] "GET /students HTTP/1.1" 200 -
 * Debugger PIN: 491-203-784
127.0.0.1 - - [07/Oct/2025 19:50:19] "GET /students HTTP/1.1" 200 -
127.0.0.1 - - [07/Oct/2025 19:50:19] "GET /students HTTP/1.1" 200 -
127.0.0.1 - - [07/Oct/2025 19:50:19] "GET /favicon.ico HTTP/1.1" 404 -
```

127.0.0.1:5000/students

Pretty print


```
[
  {
    "age": 20,
    "department": "Computer Science",
    "id": 1,
    "name": "Alice"
  },
  {
    "age": 22,
    "department": "Mathematics",
    "id": 2,
    "name": "Bob"
  },
  {
    "age": 21,
    "department": "Physics",
    "id": 3,
    "name": "Charlie"
  }
]
```


Observation: Python requests module was used to test all the API endpoints. Each request returned appropriate JSON responses confirming functionality. The testing scripts validated both data integrity and route accuracy. This task reinforced API testing and integration verification skills.