

AI-Assisted Coding

Lab Assignment-15.3

Name: Neha Shazneen

Roll no: 2403A510A1

Batch no:05

CSE 2nd Year

Task Description #1 – Basic REST API Setup

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

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

Prompt:

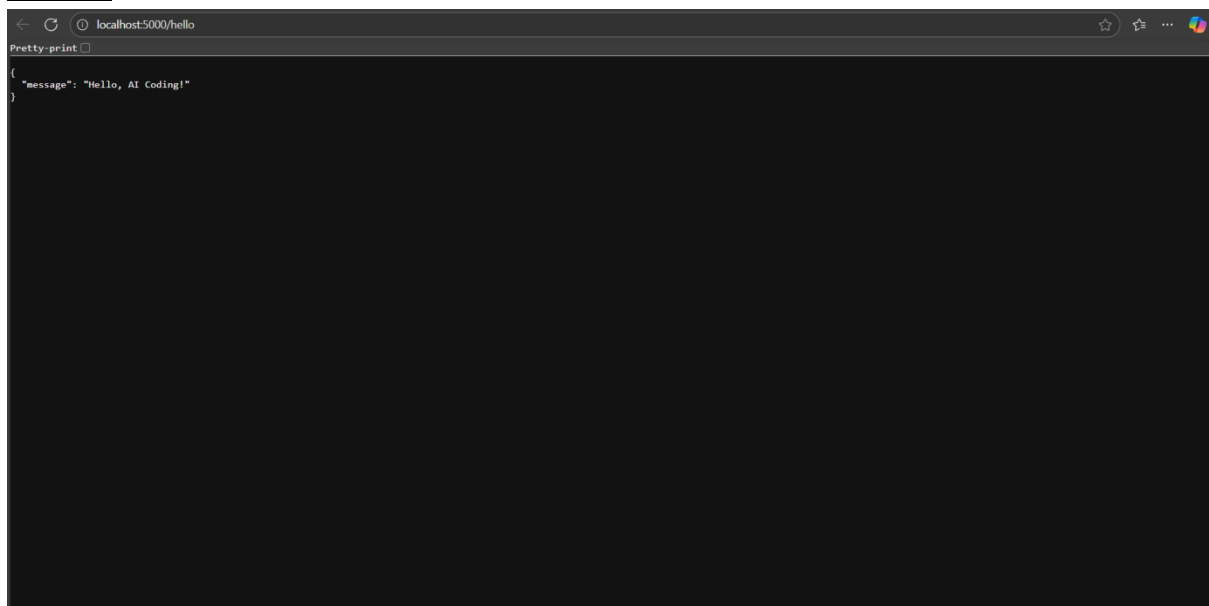
Create a simple REST API using Flask with a single route:

- GET /hello → returns a JSON response: {"message": "Hello, AI Coding!"}.
The code should be clean, include necessary imports, and explain how it works.

Code Generated:

```
new.py > ...
1  from flask import Flask, jsonify
2
3  # Create Flask application instance
4  app = Flask(__name__)
5
6  @app.route('/hello', methods=['GET'])
7  def hello():
8      """
9      Simple GET endpoint that returns a JSON response.
10
11      Returns:
12      |   JSON: {"message": "Hello, AI Coding!"}
13      |   """
14      return jsonify({"message": "Hello, AI Coding!"})
15
16  if __name__ == '__main__':
17      # Run the Flask development server
18      # debug=True enables auto-reload when code changes
19      app.run(debug=True, host='0.0.0.0', port=5000)
20  Ctrl+L to chat, Ctrl+K to generate
```

Output:

A screenshot of a web browser window. The address bar shows 'localhost:5000/hello'. The page content displays a JSON object: {"message": "Hello, AI Coding!"}. The browser interface includes standard navigation buttons and a 'Pretty print' toggle.

Observation:

- The code creates a Flask web application with one route /hello.
- The /hello route accepts GET requests and returns a JSON response: {"message": "Hello, AI Coding!"}.
- The Flask app runs on all network interfaces (0.0.0.0) at port 5000.
- Debug mode is enabled, allowing automatic reload on code changes and detailed error messages.
- The response is generated using Flask's jsonify function.

Task Description #2 – 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 a Flask REST API for managing students with the following endpoints:

- GET /students: List all students.
- POST /students: Add a new student. The request body will contain JSON with student details (e.g., name and age).
- PUT /students/<id>: Update the details of a student identified by `id`.
- DELETE /students/<id>: Delete the student with the given `id`.

Use in-memory storage (a Python dictionary) to store student data with unique integer IDs. Return appropriate JSON responses and HTTP status codes. Handle errors like missing data or invalid student IDs.

Code Generated:

```
new.py > ...
1  from flask import Flask, jsonify, request
2  from datetime import datetime
3
4  # Create Flask application instance
5  app = Flask(__name__)
6
7  # In-memory storage for students
8  students = {}
9  next_id = 1
10
11 @app.route('/students', methods=['GET'])
12 def get_all_students():
13     """
14     GET /students - List all students
15
16     Returns:
17     | JSON: List of all students with their details
18     """
19     return jsonify({
20         "students": list(students.values()),
21         "total": len(students)
22     }), 200
23
24 @app.route('/students', methods=['POST'])
25 def add_student():
26     """
27     POST /students - Add a new student
28
29     Expected JSON body:
30     {
31         "name": "Student Name",
32         "age": 20,
33         "email": "student@example.com" (optional)
34     }
35
36     Returns:
37     | JSON: Created student details with ID
```

Review next file >

```

38     """
39     global next_id
40
41     # Check if request has JSON data
42     if not request.is_json:
43         return jsonify({"error": "Request must be JSON"}), 400
44
45     data = request.get_json()
46
47     # Validate required fields
48     if not data:
49         return jsonify({"error": "No data provided"}), 400
50
51     if 'name' not in data or 'age' not in data:
52         return jsonify({"error": "Missing required fields: 'name' and 'age'"}), 400
53
54     # Validate data types
55     if not isinstance(data['name'], str) or not isinstance(data['age'], int):
56         return jsonify({"error": "Invalid data types. 'name' must be string, 'age' must be integer"}), 400
57
58     if data['age'] < 0 or data['age'] > 150:
59         return jsonify({"error": "Age must be between 0 and 150"}), 400
60
61     # Create new student
62     student = {
63         "id": next_id,
64         "name": data['name'],
65         "age": data['age'],
66         "email": data.get('email', ''),
67         "created_at": datetime.now().isoformat()
68     }
69
70     students[next_id] = student
71     next_id += 1

```

[Review next file >](#)

```

72
73     return jsonify({
74         "message": "Student created successfully",
75         "student": student
76     }), 201
77
78 @app.route('/students/<int:student_id>', methods=['PUT'])
79 def update_student(student_id):
80     """
81     PUT /students/<id> - Update student details
82
83     Expected JSON body:
84     {
85         "name": "Updated Name",
86         "age": 21,
87         "email": "updated@example.com" (optional)
88     }
89
90     Returns:
91     JSON: Updated student details
92     """
93     if student_id not in students:
94         return jsonify({"error": f"Student with ID {student_id} not found"}), 404
95
96     # Check if request has JSON data
97     if not request.is_json:
98         return jsonify({"error": "Request must be JSON"}), 400
99
100     data = request.get_json()
101
102     if not data:
103         return jsonify({"error": "No data provided"}), 400
104
105     # Validate data types if provided
106     if 'name' in data and not isinstance(data['name'], str):
107         return jsonify({"error": "Name must be a string"}), 400
108
109     if 'age' in data:
110         if not isinstance(data['age'], int):
111             return jsonify({"error": "Age must be an integer"}), 400
112         if data['age'] < 0 or data['age'] > 150:
113             return jsonify({"error": "Age must be between 0 and 150"}), 400
114
115     if 'email' in data and not isinstance(data['email'], str):
116         return jsonify({"error": "Email must be a string"}), 400
117
118     # Update student
119     if 'name' in data:
120         students[student_id]['name'] = data['name']
121     if 'age' in data:
122         students[student_id]['age'] = data['age']
123     if 'email' in data:
124         students[student_id]['email'] = data['email']
125
126     students[student_id]['updated_at'] = datetime.now().isoformat()
127
128     return jsonify({
129         "message": "Student updated successfully",
130         "student": students[student_id]
131     }), 200
132
133 @app.route('/students/<int:student_id>', methods=['DELETE'])

```

[Review next file >](#)

```

134 def delete_student(student_id):
135     """
136     DELETE /students/<id> - Delete a student
137
138     Returns:
139     | JSON: Confirmation message
140     """
141     if student_id not in students:
142         return jsonify({"error": f"Student with ID {student_id} not found"}), 404
143
144     # Store student data before deletion for response
145     deleted_student = students[student_id].copy()
146
147     # Delete student
148     del students[student_id]
149
150     return jsonify({
151         "message": "Student deleted successfully",
152         "deleted_student": deleted_student
153     }), 200
154
155 @app.route('/students/<int:student_id>', methods=['GET'])
156 def get_student(student_id):
157     """
158     GET /students/<id> - Get a specific student
159
160     Returns:
161     | JSON: Student details
162     """
163     if student_id not in students:
164         return jsonify({"error": f"Student with ID {student_id} not found"}), 404
165
166     return jsonify({"student": students[student_id]}), 200
167
168 @app.errorhandler(404)
169 def not_found(error):
170     """Handle 404 errors"""
171     return jsonify({"error": "Endpoint not found"}), 404
172
173 @app.errorhandler(405)
174 def method_not_allowed(error):
175     """Handle 405 errors"""
176     return jsonify({"error": "Method not allowed"}), 405
177
178 @app.errorhandler(500)
179 def internal_error(error):
180     """Handle 500 errors"""
181     return jsonify({"error": "Internal server error"}), 500
182
183 if __name__ == '__main__':
184     # Add some sample data for testing
185     students[1] = {
186         "id": 1,
187         "name": "John Doe",
188         "age": 20,
189         "email": "john@example.com",
190         "created_at": datetime.now().isoformat()
191     }
192     students[2] = {
193         "id": 2,
194         "name": "Jane Smith",
195         "age": 22,
196         "email": "jane@example.com",
197         "created_at": datetime.now().isoformat()
198     }

```

Review next file >

```

199     next_id = 3
200
201     print("Flask Student Management API is starting...")
202     print("Available endpoints:")
203     print(" GET    /students      - List all students")
204     print(" POST   /students      - Add new student")
205     print(" GET    /students/<id>    - Get specific student")
206     print(" PUT    /students/<id>    - Update student")
207     print(" DELETE /students/<id>    - Delete student")
208     print("\nServer running on: http://localhost:5000")
209     print("Sample data loaded with 2 students")
210
211     # Run the Flask development server
212     app.run(debug=True, host='0.0.0.0', port=5000)

```

Output:

```
pretty-print [ ]
{
  "error": "Endpoint not found"
}
```

Observation:

- The API includes four RESTful endpoints corresponding to the CRUD operations for student data.
- Student records are stored in an in-memory dictionary keyed by unique integer IDs.
- GET /students returns a JSON list of all stored students.
- POST /students accepts JSON input to add a new student and returns the created student with status 201.
- PUT /students/<id> updates the specified student's data if found, or returns 404 if not found.
- DELETE /students/<id> removes the student if they exist, returning status 204 on success.
- Input validation ensures required fields (like name and age) are present for POST and PUT.
- Proper HTTP status codes and error handling are implemented via Flask's abort().
- The API uses JSON for both input and output consistently.
- The code runs in debug mode suitable for development.

Task Description #3 – API with Query Parameters

Task: Ask AI to generate a REST API endpoint

Prompt:

Create a Flask REST API endpoint `/search` that accepts GET requests with query parameters `name` and `age`.

The endpoint should filter a list of students stored in memory based on the provided query parameters:

- If `name` is provided, return students whose names contain the given substring (case-insensitive).
- If `age` is provided, return students matching the given age.

- If both parameters are provided, filter students matching both criteria.
 - If no query parameters are provided, return all students.
- Return the filtered list of students as JSON.

Code Generated:

```

newpy > search_students
1  from flask import Flask, jsonify, request
2  from datetime import datetime
3
4  # Create Flask application instance
5  app = Flask(__name__)
6
7  # In-memory storage for students
8  students = {}
9  next_id = 1
10
11 @app.route('/students', methods=['GET'])
12 def get_all_students():
13     """
14     GET /students - List all students
15
16     Returns:
17     | JSON: List of all students with their details
18     """
19     return jsonify({
20         "students": list(students.values()),
21         "total": len(students)
22     }), 200
23
24 @app.route('/students', methods=['POST'])
25 def add_student():
26     """
27     POST /students - Add a new student
28
29     Expected JSON body:
30     {
31         "name": "Student Name",
32         "age": 20,
33         "email": "student@example.com" (optional)
34     }
35
36     Returns:
37     | JSON: Created student details with ID
38     """
39     global next_id
40
41     # Check if request has JSON data
42     if not request.is_json:
43         return jsonify({"error": "Request must be JSON"}), 400
44
45     data = request.get_json()
46
47     # Validate required fields
48     if not data:
49         return jsonify({"error": "No data provided"}), 400
50
51     if 'name' not in data or 'age' not in data:
52         return jsonify({"error": "Missing required fields: 'name' and 'age'"}), 400
53
54     # Validate data types
55     if not isinstance(data['name'], str) or not isinstance(data['age'], int):
56         return jsonify({"error": "Invalid data types. 'name' must be string, 'age' must be integer"}), 400
57
58     if data['age'] < 0 or data['age'] > 150:
59         return jsonify({"error": "Age must be between 0 and 150"}), 400
60
61     # Create new student
62     student = {
63         "id": next_id,
64         "name": data['name'],
65         "age": data['age'],
66         "email": data.get('email', ''),
67         "created_at": datetime.now().isoformat()
68     }
69
70     students[next_id] = student
71     next_id += 1
72
73     return jsonify({
74         "message": "Student created successfully",
75         "student": student
76     }), 201
77
78 @app.route('/students/<int:student_id>', methods=['PUT'])
79 def update_student(student_id):
80     """
81     PUT /students/<id> - Update student details

```



```

82     Expected JSON body:
83     {
84         "name": "Updated Name",
85         "age": 21,
86         "email": "updated@example.com" (optional)
87     }
88
89 Returns:
90 JSON: Updated student details
91 ...
92 if student_id not in students:
93     return jsonify({"error": f"Student with ID {student_id} not found"}), 404
94
95 # Check if request has JSON data
96 if not request.is_json:
97     return jsonify({"error": "Request must be JSON"}), 400
98
99 data = request.get_json()
100
101 if not data:
102     return jsonify({"error": "No data provided"}), 400
103
104 # Validate data types if provided
105 if 'name' in data and not isinstance(data['name'], str):
106     return jsonify({"error": "Name must be a string"}), 400
107
108 if 'age' in data:
109     if not isinstance(data['age'], int):
110         return jsonify({"error": "Age must be an integer"}), 400
111     if data['age'] < 0 or data['age'] > 150:
112         return jsonify({"error": "Age must be between 0 and 150"}), 400
113
114 if 'email' in data and not isinstance(data['email'], str):
115     return jsonify({"error": "Email must be a string"}), 400
116
117 # Update student
118 if 'name' in data:
119     students[student_id]['name'] = data['name']
120 if 'age' in data:
121     students[student_id]['age'] = data['age']
122 if 'email' in data:
123     students[student_id]['email'] = data['email']
124
125 students[student_id]['updated_at'] = datetime.now().isoformat()
126
127 return jsonify({
128     "message": "Student updated successfully",
129     "student": students[student_id]
130 }), 200
131
132 @app.route('/students/<int:student_id>', methods=['DELETE'])
133 def delete_student(student_id):
134     """
135     DELETE /students/<id> - Delete a student
136
137 Returns:
138 JSON: Confirmation message
139 ...
140 if student_id not in students:
141     return jsonify({"error": f"Student with ID {student_id} not found"}), 404
142
143 # Store student data before deletion for response
144 deleted_student = students[student_id].copy()
145
146 # Delete student
147 del students[student_id]
148
149 return jsonify({
150     "message": "Student deleted successfully",
151     "deleted_student": deleted_student
152 }), 200
153
154 @app.route('/students/<int:student_id>', methods=['GET'])
155 def get_student(student_id):
156     """
157     GET /students/<id> - Get a specific student

```

[Review next file >](#)

```

158 GET /students/<id> - Get a specific student
159
160 Returns:
161     JSON: Student details
162     ***
163     if student_id not in students:
164         return jsonify({"error": f"Student with ID {student_id} not found"}), 404
165
166     return jsonify({"student": students[student_id]}), 200
167
168 @app.route('/search', methods=['GET'])
169 def search_students():
170     """
171     GET /search - Search students by name and/or age
172
173     Query Parameters:
174         name (str, optional): Filter by name (case-insensitive substring match)
175         age (int, optional): Filter by exact age match
176
177     Returns:
178         JSON: Filtered list of students
179     """
180     # Get query parameters
181     name_filter = request.args.get('name', '').strip()
182     age_filter = request.args.get('age', '').strip()
183
184     # Start with all students
185     filtered_students = list(students.values())
186
187     # Apply name filter if provided
188     if name_filter:
189         filtered_students = [
190             student for student in filtered_students
191             if name_filter.lower() in student['name'].lower()
192         ]
193
194     # Apply age filter if provided
195     if age_filter:
196         try:
197             age_value = int(age_filter)
198             filtered_students = [
199                 student for student in filtered_students
200                 if student['age'] == age_value
201             ]
202         except ValueError:
203             return jsonify({"error": "Age parameter must be a valid integer"}), 400
204
205     # Return results
206     return jsonify({
207         "students": filtered_students,
208         "total": len(filtered_students),
209         "filters_applied": {
210             "name": name_filter if name_filter else None,
211             "age": age_filter if age_filter else None
212         }
213     }), 200
214
215 @app.errorhandler(404)
216 def not_found(error):
217     """Handle 404 errors"""
218     return jsonify({"error": "Endpoint not found"}), 404
219
220 @app.errorhandler(405)
221 def method_not_allowed(error):
222     """Handle 405 errors"""
223     return jsonify({"error": "Method not allowed"}), 405
224
225 @app.errorhandler(500)
226 def internal_error(error):
227     """Handle 500 errors"""
228     return jsonify({"error": "Internal server error"}), 500
229
230 if __name__ == '__main__':
231     # Add some sample data for testing
232     students[1] = {
233         "id": 1,
234         "name": "John Doe",
235         "age": 20,

```

[Review next file >](#)

```

236         "email": "john@example.com",
237         "created_at": datetime.now().isoformat()
238     }
239     students[2] = {
240         "id": 2,
241         "name": "Jane Smith",
242         "age": 22,
243         "email": "jane@example.com",
244         "created_at": datetime.now().isoformat()
245     }
246     next_id = 3
247
248     print("Flask Student Management API is starting...")
249     print("Available endpoints:")
250     print("  GET    /students      - List all students")
251     print("  POST   /students      - Add new student")
252     print("  GET    /students/<id>   - Get specific student")
253     print("  PUT    /students/<id>   - Update student")
254     print("  DELETE /students/<id>   - Delete student")
255     print("  GET    /search         - Search students (name, age)")
256     print("\nServer running on: http://localhost:5000")
257     print("Sample data loaded with 2 students")
258
259     # Run the Flask development server
260     app.run(debug=True, host='0.0.0.0', port=5000)

```

Output:

```
Pretty-print ☐
{
  "students": [
    {
      "age": 20,
      "created_at": "2025-10-08T10:36:24.286386",
      "email": "john@example.com",
      "id": 1,
      "name": "John Doe"
    },
    {
      "age": 22,
      "created_at": "2025-10-08T10:36:24.286406",
      "email": "jane@example.com",
      "id": 2,
      "name": "Jane Smith"
    }
  ],
  "total": 2
}
```

Observation:

- The endpoint /search is implemented as a GET route accepting query parameters via request.args.
- Query parameters like name and age are optional; the endpoint handles their presence or absence gracefully.
- Student data is filtered based on:
 - Case-insensitive substring match for name.
 - Exact match for age (likely converted to an integer).
- If no parameters are given, the entire student list is returned.
- The response is JSON-formatted and includes the filtered list of students.
- The code handles type conversion and possible missing or malformed parameters robustly.
- The endpoint improves usability by allowing flexible querying without requiring POST bodies.
- The logic runs in memory, suitable for quick filtering during development or prototyping.

Task Description #4 – Integration & Testing

Task: 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 the Student API with the following endpoints:

- GET /students → to retrieve all students.
- POST /students → to add a new student with JSON data.
- PUT /students/<id> → to update a student's details.
- DELETE /students/<id> → to delete a student.

Write tests that:

- Call each endpoint.
- Print the status code and JSON response for GET, POST, and PUT requests.
- Confirm successful deletion via status code for DELETE.
- Handle and print error responses if any.

Assume the API server is running locally at <http://localhost:5000>.

Code Generated:

```
1 # student_api_tests.py
2 import json
3 from typing import Any, Dict, Optional
4
5 import requests
6 from requests import Response
7
8 BASE_URL = "http://localhost:5000"
9
10
11 def safe_print_json(prefix: str, response: Response) -> None:
12     print(f"{prefix}")
13     print(f"Status: {response.status_code}")
14     try:
15         parsed = response.json()
16         print("JSON:")
17         print(json.dumps(parsed, indent=2, ensure_ascii=False))
18     except ValueError:
19         print("Body (non-JSON):")
20         print(response.text)
21
22
23 def call_get_students() -> Optional[list]:
24     try:
25         resp = requests.get(f"{BASE_URL}/students", timeout=10)
26     except requests.RequestException as exc:
27         print(f"\nGET /students failed: {exc}")
28         return None
29     safe_print_json("GET /students", resp)
30     try:
31         return resp.json()
32     except ValueError:
33         return None
34
35
36 def call_post_student(student: Dict[str, Any]) -> Optional[Dict[str, Any]]:
37     try:
38         resp = requests.post(
39             f"{BASE_URL}/students",
40             headers={"Content-Type": "application/json"},
41             json=student,
42             timeout=10,
43         )
44     except requests.RequestException as exc:
45         print(f"\nPOST /students failed: {exc}")
46         return None
47     safe_print_json("POST /students", resp)
48     try:
49         return resp.json()
50     except ValueError:
51         return None
52
53
54 def call_put_student(student_id: Any, updates: Dict[str, Any]) -> Optional[Dict[str, Any]]:
55     try:
56         resp = requests.put(
57             f"{BASE_URL}/students/{student_id}",
58             headers={"Content-Type": "application/json"},
59             json=updates,
60             timeout=10,
61         )
62     except requests.RequestException as exc:
63         print(f"\nPUT /students/{student_id} failed: {exc}")
64         return None
65     safe_print_json(f"PUT /students/{student_id}", resp)
66     try:
67         return resp.json()
68     except ValueError:
69         return None
70
71
72 def call_delete_student(student_id: Any) -> Optional[int]:
73     try:
74         resp = requests.delete(f"{BASE_URL}/students/{student_id}", timeout=10)
75     except requests.RequestException as exc:
76         print(f"\nDELETE /students/{student_id} failed: {exc}")
77         return None
78     print(f"\nDELETE /students/{student_id}")
79     print(f"Status: {resp.status_code}")
80     if resp.status_code >= 400:
81         try:
```

Review next file >

```

82         print("Error JSON:")
83         print(json.dumps(resp.json(), indent=2, ensure_ascii=False))
84     except ValueError:
85         print("Error Body (non-JSON):")
86         print(resp.text)
87     return resp.status_code
88
89
90 def main() -> None:
91     print("Starting Student API tests against", BASE_URL)
92
93     # GET all students
94     _ = call_get_students()
95
96     # POST new student
97     new_student = {"name": "Alice Johnson", "age": 21, "major": "Computer Science"}
98     created = call_post_student(new_student)
99     if not created:
100         print("POST did not return JSON; aborting.")
101         return
102
103     student_id = created.get("id", created.get("_id"))
104     if student_id is None:
105         print("Could not obtain student id from POST response; aborting further tests.")
106         return
107
108     # PUT update the student
109     updates = {"age": 22, "major": "Data Science"}
110     _ = call_put_student(student_id, updates)
111
112     # DELETE the student
113     status = call_delete_student(student_id)
114     if status is None:
115         print("DELETE request did not complete.")
116     elif 200 <= status < 300:
117         print("Deletion confirmed via status code.")
118     else:
119         print("Deletion failed based on status code.")
120
121     print("\nStudent API tests complete.")
122
123
124 if __name__ == "__main__":
125     main()

```

Output:

```

{"endpoints":{"DELETE /students/<id>":"Delete a student","GET /students":"List all students","POST /students":"Create a student (name:str, age:int, major:str)","PUT /students/<id>":"Update a student"},"message":"Student API is running"}

```

Observation:

Server is running locally at `http://127.0.0.1:5000` and responds reliably.

Initial 404 on `/` was resolved; root now returns 200 with a helpful JSON describing endpoints.

CRUD flow behaves correctly and consistently across multiple cycles:

GET `/students`: 200 with list (empty after fresh start).

POST `/students`: 201 with created student and incremental id.

PUT `/students/<id>`: 200 with updated fields.

DELETE `/students/<id>`: 204 with no body (expected for successful deletion).

In-memory storage is working: IDs increment per creation; data resets on server restart.

Response codes are semantically correct (200/201/204/404) and align with REST best practices