

AI ASSISTED CODING

LAB-15.4

NAME:CH.RAMCHARAN

ENROLL,NO:2403A52069

BATCH:04

TASK-01:

Setup Flask Backend

PROMPT:

Create a basic Flask server in Python with one endpoint. The endpoint / should return a JSON message saying: Welcome to AI-assisted API. Include instructions for installing Flask and running the server

CODE:

```
15.4.1.py  X
lab15.4 > 15.4.1.py > add_item
1   from flask import Flask, jsonify, request
2
3   app = Flask(__name__)
4
5   # In-memory store and id generator
6   items = []
7   _next_id = 1
8
9   # GET all items
10  @app.route('/items', methods=['GET'])
11  def get_items():
12      return jsonify(items), 200
13
14  # POST a new item
15  @app.route('/items', methods=['POST'])
16  def add_item():
17      global _next_id
18      data = request.get_json(silent=True)
19      if not data or not isinstance(data, dict):
20          return jsonify({"error": "Invalid JSON body; expected an object"}), 400
21
22      # Attach a server-generated id to each item
23      item = {"id": _next_id}
24      item.update(data)
25      _next_id += 1
26
27      items.append(item)
28      return jsonify(item), 201
29
30  if __name__ == '__main__':
31      app.run(host='0.0.0.0', port=5000, debug=True)
```

OUTPUT:

```
PS C:\Users\ramch\OneDrive\Desktop\ai> PS C:\Users\ramch\OneDrive\Desktop\ai> python app.py
>> * Serving Flask app 'app'
>> * Debug mode: on
>> WARNING: This is a development server. Do not use it in a production deployment.
>> * Running on http://127.0.0.1:5000
>> Press CTRL+C to quit
>> * Restarting with stat
>> * Debugger is active!
>> * Debugger PIN: 432-889-112
```

OBSERVATION

The AI made building a Flask server very easy. It generated clear code, fixed errors quickly, and helped me run the backend without confusion. It saves time and makes learning much simpler.

TASK-02:

Create a CRUD API – Read and Create

PROMPT:

Create a Flask API with two endpoints: one to list all items using a GET request, and one to add a new item using a POST request. Store the items in a Python list.

CODE:

```
lab15.4 > 15.4.2.py > ...
1
2  from flask import Flask, jsonify, request
3  app = Flask(__name__)
4  items = []
5  # GET all items
6  @app.route('/items', methods=['GET'])
7  def get_items():
8      return jsonify(items)
9  # POST a new item
10 @app.route('/items', methods=['POST'])
11 def add_item():
12     data = request.get_json()
13     items.append(data)
14     return jsonify({"message": "Item added", "item": data}), 201
15
16 if __name__ == "__main__":
17     app.run(debug=True)
```

OUTPUT:

The screenshot shows a Swagger UI interface for a 'default' namespace. At the top, it displays a 'GET /items' endpoint with a description 'List all items'. Below this, the 'Parameters' section includes a parameter 'X-Fields' with a type 'string(\$mask)' and a description 'An optional fields mask'. The 'Responses' section shows a successful response (200) with a status of 'Success'. An example value is provided as a JSON object:

```
[ { "name": "string", "price": 0 } ]
```

OBSERVATION:

The AI helped me build the API easily. It explained the code clearly and made the whole process faster and simpler

TASK-03:

Update Item

PROMPT:

Create a Flask PUT endpoint that updates an item in a list using its index. If the index is invalid, return an error message.

CODE:

```

lab15.4 > 15.4.3.py > get_items
1
2  from flask import Flask, jsonify, request
3  app = Flask(__name__)
4  items = []
5  # GET all items
6  @app.route('/items', methods=['GET'])
7  def get_items():
8      | return jsonify(items)
9  # POST a new item
10  @app.route('/items', methods=['POST'])
11  def add_item():
12      | data = request.get_json()
13      | items.append(data)
14      | return jsonify({"message": "Item added", "item": data}), 201
15
16  # PUT /items/<int:index>
17  @app.route('/items/<int:index>', methods=['PUT'])
18  def update_item(index):
19      | if index < 0 or index >= len(items):
20      |     return jsonify({"error": "Item not found"}), 404
21      | data = request.get_json()
22      | items[index] = data
23      | return jsonify({"message": "Item updated", "item": data})
24
25  if __name__ == "__main__":
26      | app.run(debug=True)
27

```

OUTPUT:

The screenshot shows a Swagger UI interface for a REST API. The endpoint is `PUT /items/{index}`. The `Try it out` button is visible in the top right.

Parameters

Name	Description
<code>payload</code> * required	Example Value Model Object (body) <pre>{ "name": "string", "price": 0 }</pre>
Parameter content type <code>application/json</code>	
<code>X-Fields</code>	An optional fields mask <code>string(\$mask)</code> (header) <code>X-Fields</code>
<code>index</code> * required	The item identifier <code>integer</code> (path) <code>index</code>

Responses

Code	Description	Response content type
200	Success Example Value Model <pre>{ "name": "string", "price": 0 }</pre>	<code>application/json</code>
404	Item not found	

OBSERVATION:

The AI helped me easily add the update feature. It made the code simple and clear, and the task became much easier to understand.

TASK-04:

Delete Item

PROMPT:

Create a Flask DELETE endpoint that removes an item from a list using its index. If the index is invalid, return an error message.

CODE:

```
lab15.4 > 15.4.4.py > add_item
1  from flask import Flask, jsonify, request
2  app = Flask(__name__)
3  items = []
4  # GET all items
5  @app.route('/items', methods=['GET'])
6  def get_items():
7      return jsonify(items)
8  # POST a new item
9  @app.route('/items', methods=['POST'])
10 def add_item():
11     data = request.get_json()
12     items.append(data)
13     return jsonify({"message": "Item added", "item": data}), 201
14
15 # PUT /items/<int:index>
16 @app.route('/items/<int:index>', methods=['PUT'])
17 def update_item(index):
18     if index < 0 or index >= len(items):
19         return jsonify({"error": "Item not found"}), 404
20     data = request.get_json()
21     items[index] = data
22     return jsonify({"message": "Item updated", "item": data})
23
24 # DELETE /items/<int:index>
25 @app.route('/items/<int:index>', methods=['DELETE'])
26 def delete_item(index):
27     if index < 0 or index >= len(items):
28         return jsonify({"error": "Item not found"}), 404
29     removed_item = items.pop(index)
30     return jsonify({"message": "Item deleted", "item": removed_item})
31
32 if __name__ == "__main__":
33     app.run(debug=True)
34
```

OUTPUT:

The screenshot shows a Swagger UI interface for a REST API. The URL is `/items/{index}` and the method is `DELETE`. The description is "Delete an item given its identifier".

Parameters

Name	Description
index * required	The item identifier (path)
index	index

Responses

Code	Description
204	Item deleted
404	Item not found

Response content type: application/json

OBSERVATION:

The AI made the delete feature easy to implement. It provided clear code and helped me complete the task quickly and simply.

TASK-05:

Add Auto-Generated Documentation

PROMPT:

Add docstrings and inline comments to all Flask endpoints. Optionally use Swagger or Flask-RESTX to auto-generate API documentation so the endpoints can be viewed clearly.

CODE:

```
lab15.4 > 15.4.5.py > ...
1
2  from flask import Flask
3  from flask_restx import Api, Resource, fields
4
5  app = Flask(__name__)
6  api = Api(app, version='1.0', title='Item API', description='A simple CRUD API for items')
7
8  items = []
9
10 item_model = api.model('Item', {
11     'name': fields.String(required=True, description='The item name'),
12     'price': fields.Float(required=True, description='The item price')
13 })
14
15 @api.route('/items')
16 class ItemList(Resource):
17     @api.doc('list_items')
18     @api.marshal_list_with(item_model)
19     def get(self):
20         """List all items"""
21         return items
22
23     @api.doc('create_item')
24     @api.expect(item_model)
25     @api.marshal_with(item_model, code=201)
26     def post(self):
27         """Create a new item"""
28         new_item = api.payload
29         items.append(new_item)
30         return new_item, 201
31
32     @api.route('/items/<int:index>')
33     @api.param('index', 'The item identifier')
34     @api.response(404, 'Item not found')
35     class Item(Resource):
36         @api.doc('get_item')
37         @api.marshal_with(item_model)
38         def get(self, index):
39             """Fetch an item given its identifier"""

Q | Ln 63, Col 27  Spaces: 4  UTF-8  LF  [] Python  3.12.10  ⓘ Go Live  ◆  Q
```

```

35     class Item(Resource):
36         def get(self, index):
37             if 0 <= index < len(items):
38                 return items[index]
39             api.abort(404)
40
41         @api.doc('update_item')
42         @api.expect(item_model)
43         @api.marshal_with(item_model)
44         def put(self, index):
45             """Update an item given its identifier"""
46             if 0 <= index < len(items):
47                 items[index] = api.payload
48                 return items[index]
49             api.abort(404)
50
51         @api.doc('delete_item')
52         @api.response(204, 'Item deleted')
53         def delete(self, index):
54             """Delete an item given its identifier"""
55             if 0 <= index < len(items):
56                 items.pop(index)
57                 return '', 204
58             api.abort(404)
59
60         if __name__ == '__main__':
61             app.run(debug=True)
62
63

```

OUTPUT:

Item API 1.0

[Base URL: /]
/swagger.json

A simple CRUD API for items

default Default namespace

- GET /items List all items
- POST /items Create a new item
- DELETE /items/{index}** Delete an item given its identifier
- GET /items/{index} Fetch an item given its identifier
- PUT /items/{index} Update an item given its identifier

Models

OBSERVATION:

The AI made documenting the API easy. It clearly explained each endpoint and helped create clean, readable documentation.