

# AI ASSISTED CODING

## LAB-15.4

NAME : R.SURYANARAYANA

ENROLL,NO : 2403A52038

BATCH : 03

### 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

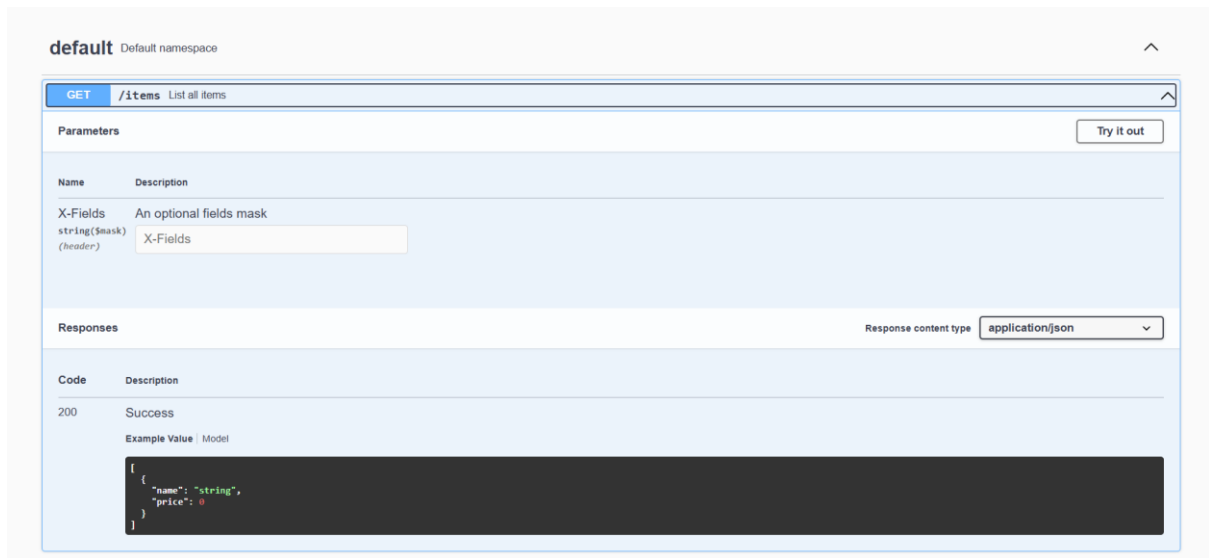
## PROMPT:

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)
18
```

## OUTPUT:



The image shows a Swagger UI interface for a REST API. At the top, it says "default" and "Default namespace". Below that, the endpoint is "GET /items" with a description "List all items". There is a "Try it out" button. Under the "Parameters" section, there is a table with two columns: "Name" and "Description". The first row shows "X-Fields" with the description "An optional fields mask". Below this, there is a text input field containing "X-Fields". Under the "Responses" section, there is a table with two columns: "Code" and "Description". The first row shows "200" with the description "Success". Below this, there is a section for "Example Value" and "Model". The "Example Value" is a JSON object: 

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

. The "Model" is a dropdown menu with "application/json" selected.

## 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 REST client interface with a PUT request to `/items/{index}`. The request has a JSON payload `{ "name": "string", "price": 0 }` and a path parameter `index`. The response is a 200 status code with the same JSON payload.

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

Code	Description
200	Success Example Value   Model <pre>{   "name": "string",   "price": 0 }</pre>
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
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 # DELETE /items/<int:index>
26 @app.route('/items/<int:index>', methods=['DELETE'])
27 def delete_item(index):
28     if index < 0 or index >= len(items):
29         return jsonify({"error": "Item not found"}), 404
30     removed_item = items.pop(index)
31     return jsonify({"message": "Item deleted", "item": removed_item})
32
33 if __name__ == "__main__":
34     app.run(debug=True)
35
```

## OUTPUT:

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** /items/{index} Delete an item given its identifier
- Parameters:**
  - index** (required): The item identifier. Type: integer (path). Input field contains "index".
- Responses:**
  - Code:** 204, 404
  - Description:** Item deleted, 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"""
```

```

35 class Item(Resource):
36     def get(self, index):
37
38         if 0 <= index < len(items):
39             return items[index]
40         api.abort(404)
41
42     @api.doc('update_item')
43     @api.expect(item_model)
44     @api.marshal_with(item_model)
45     def put(self, index):
46         """Update an item given its identifier"""
47         if 0 <= index < len(items):
48             items[index] = api.payload
49             return items[index]
50         api.abort(404)
51
52     @api.doc('delete_item')
53     @api.response(204, 'Item deleted')
54     def delete(self, index):
55         """Delete an item given its identifier"""
56         if 0 <= index < len(items):
57             items.pop(index)
58             return '', 204
59         api.abort(404)
60
61 if __name__ == '__main__':
62     app.run(debug=True)
63
64
65

```

## OUTPUT:

**Item API** <sup>1.0</sup>  
 [ 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.