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

lab15.4 > 15.4.1.py > add_item

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

app = Flask(__name__)

# In-memory store and id generator
items = []
_next_id = 1

# GET all items
@app.route('/items', methods=['GET'])
def get_items():
    return jsonify(items), 200

# POST a new item
@app.route('/items', methods=['POST'])
def add_item():
    global _next_id
    data = request.get_json(silent=True)
    if not data or not isinstance(data, dict):
        return jsonify({"error": "Invalid JSON body; expected an object"}), 400

    # Attach a server-generated id to each item
    item = {"id": _next_id}
    item.update(data)
    _next_id += 1

    items.append(item)
    return jsonify(item), 201

if __name__ == '__main__':
    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:

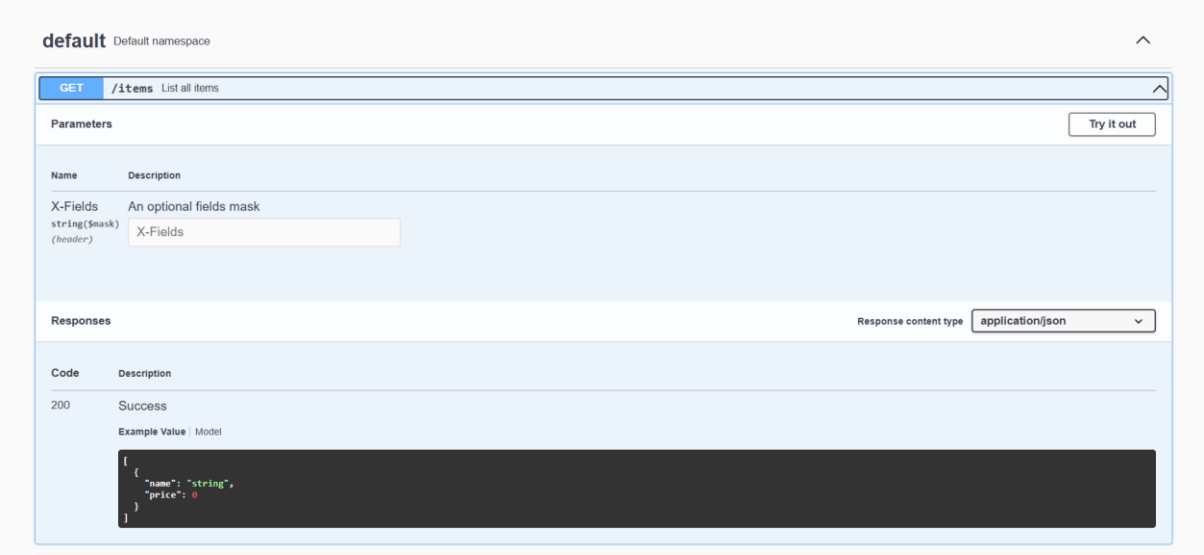
Create a Flask API with two endpoints: one to list all items using a GET request, and one to add a new item using a POST request. Store the items in a Python list.

CODE:

lab15.4 > 15.4.2.py > ...

```python

from flask import Flask, jsonify, request
app = Flask(__name__)
items = []
# GET all items
@app.route('/items', methods=['GET'])
def get_items():
    return jsonify(items)
# POST a new item
@app.route('/items', methods=['POST'])
def add_item():
    data = request.get_json()
    items.append(data)
    return jsonify({"message": "Item added", "item": data}), 201

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

```

OUTPUT:



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

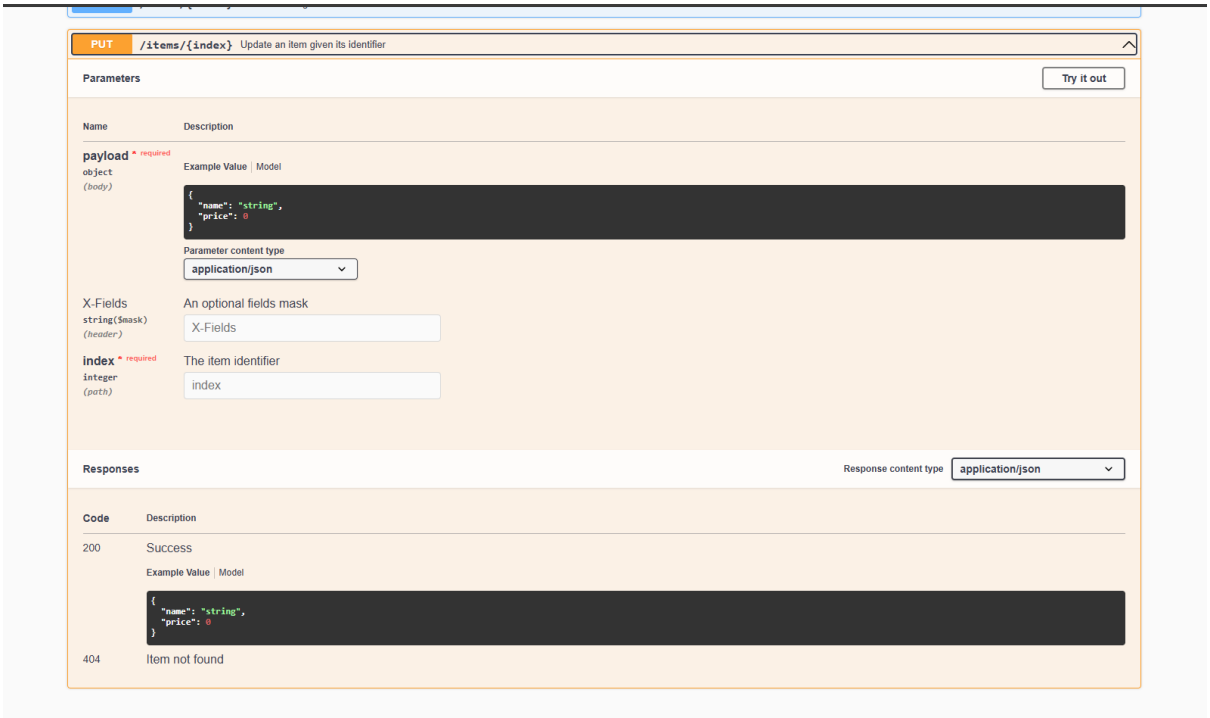
PROMPT:

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:

```python
lab15.4 > 15.4.3.py > get_items

from flask import Flask, jsonify, request
app = Flask(__name__)
items = []
# GET all items
@app.route('/items', methods=['GET'])
def get_items():
    return jsonify(items)
# POST a new item
@app.route('/items', methods=['POST'])
def add_item():
    data = request.get_json()
    items.append(data)
    return jsonify({"message": "Item added", "item": data}), 201

# PUT /items/<int:index>
@app.route('/items/<int:index>', methods=['PUT'])
def update_item(index):
    if index < 0 or index >= len(items):
        return jsonify({"error": "Item not found"}), 404
    data = request.get_json()
    items[index] = data
    return jsonify({"message": "Item updated", "item": data})

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

OUTPUT:



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:

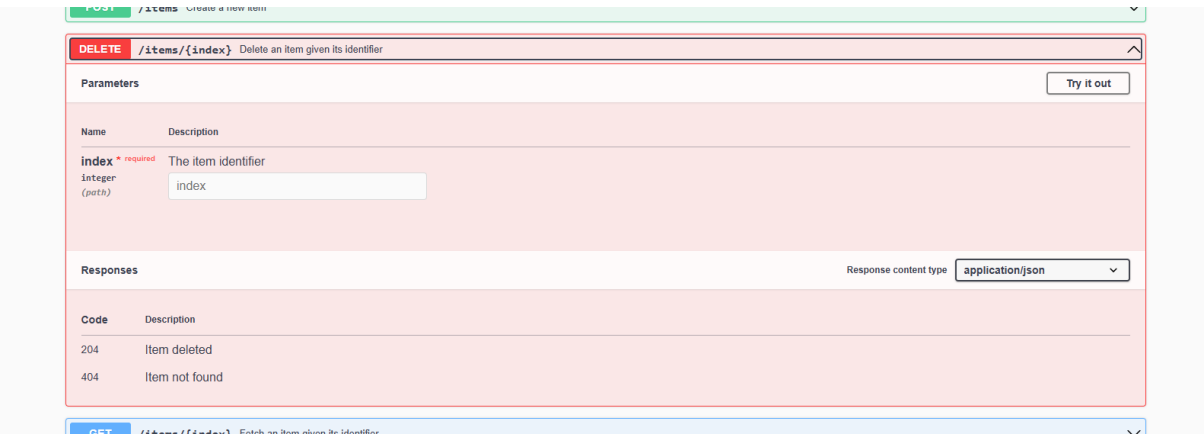
Create a Flask DELETE endpoint that removes an item from a list using its index. If the index is invalid, return an error message.

CODE:

lab15.4 > 15.4.4.py > add_item

```python

from flask import Flask, jsonify, request
app = Flask(__name__)
items = []
# GET all items
@app.route('/items', methods=['GET'])
def get_items():
    return jsonify(items)
# POST a new item
@app.route('/items', methods=['POST'])
def add_item():
    data = request.get_json()
    items.append(data)
    return jsonify({"message": "Item added", "item": data}), 201

# PUT /items/<int:index>
@app.route('/items/<int:index>', methods=['PUT'])
def update_item(index):
    if index < 0 or index >= len(items):
        return jsonify({"error": "Item not found"}), 404
    data = request.get_json()
    items[index] = data
    return jsonify({"message": "Item updated", "item": data})

# DELETE /items/<int:index>
@app.route('/items/<int:index>', methods=['DELETE'])
def delete_item(index):
    if index < 0 or index >= len(items):
        return jsonify({"error": "Item not found"}), 404
    removed_item = items.pop(index)
    return jsonify({"message": "Item deleted", "item": removed_item})

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

```

OUTPUT:



OBSERVATION:

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

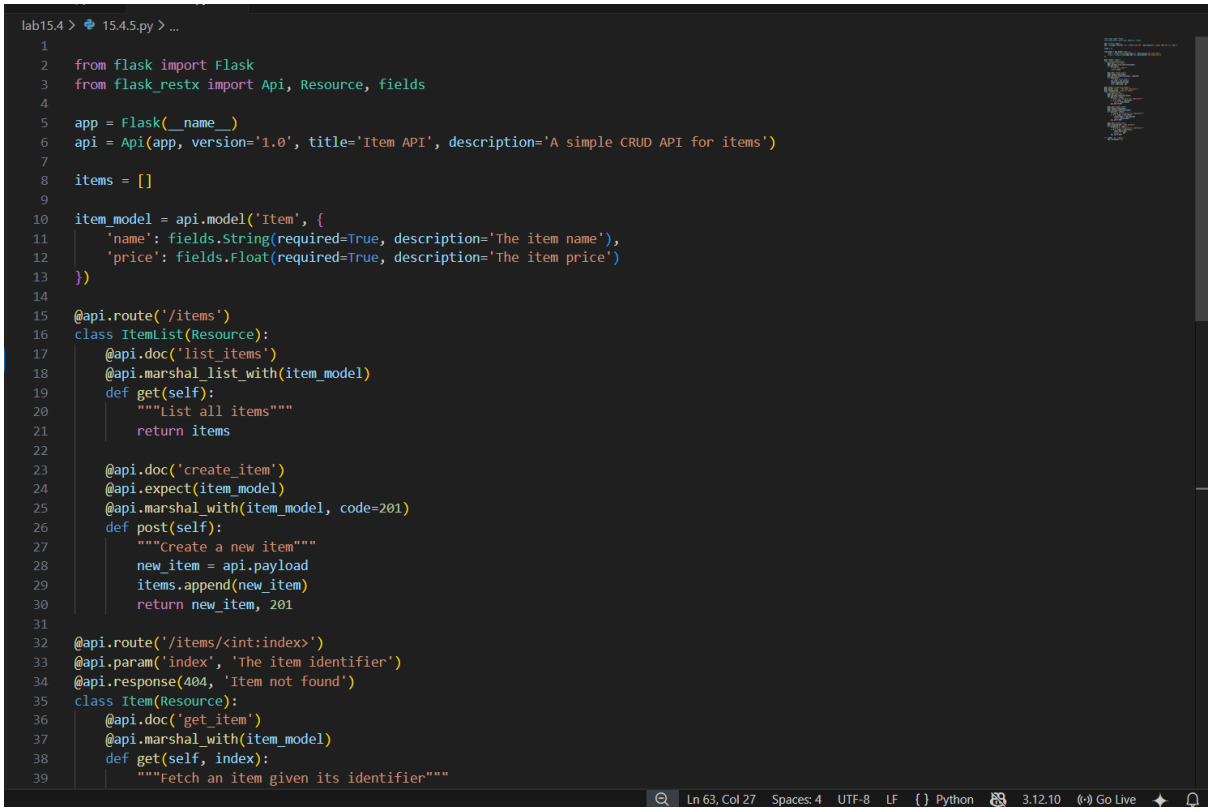
TASK-05:

Add Auto-Generated Documentation

PROMPT:

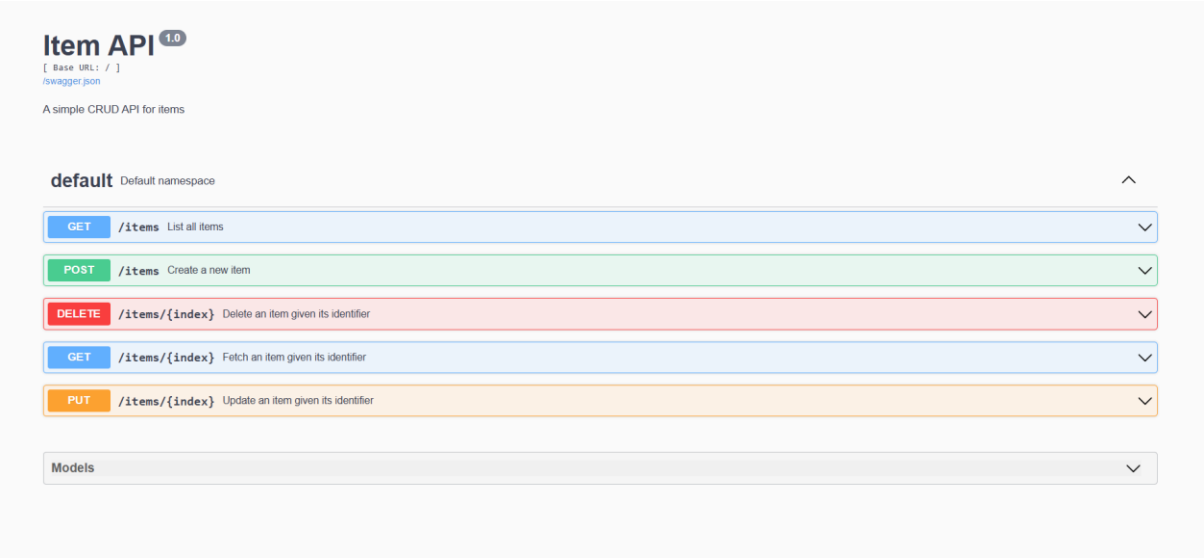
Add docstrings and inline comments to all Flask endpoints. Optionally use Swagger or Flask-RESTX to auto-generate API documentation so the endpoints can be viewed clearly.

CODE:



```python
class Item(Resource):
    def get(self, index):
        if 0 <= index < len(items):
            return items[index]
        api.abort(404)

    @api.doc('update_item')
    @api.expect(item_model)
    @api.marshal_with(item_model)
    def put(self, index):
        """Update an item given its identifier"""
        if 0 <= index < len(items):
            items[index] = api.payload
            return items[index]
        api.abort(404)

    @api.doc('delete_item')
    @api.response(204, 'Item deleted')
    def delete(self, index):
        """Delete an item given its identifier"""
        if 0 <= index < len(items):
            items.pop(index)
            return '', 204
        api.abort(404)

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

OUTPUT:



OBSERVATION:

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