AI ASSISTED CODING

LAB:15.4

ROLLNO:2403A52084

BATCH:04

TASK1:

PROMPT: Generate Python code to set up a basic Flask backend with one endpoint. The endpoint should be / and return a JSON response: {"message": "Welcome to AI-assisted API"}. The server should run locally on port 5000 with debug mode enabled.

CODE:

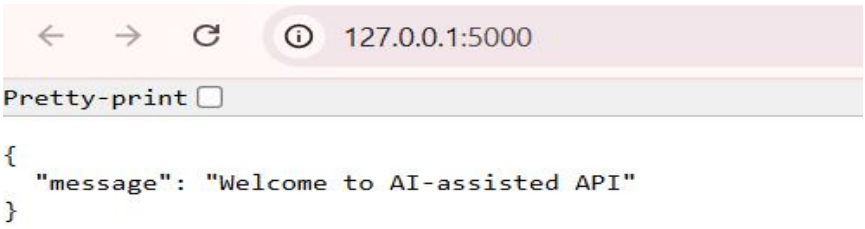
task1

lab15_4 > task1 > ...

```python
from flask import Flask, jsonify

app = Flask(__name__)

@app.route('/')
def hello_world():
    return jsonify({"message": "Welcome to AI-assisted API"})

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

OUTPUT:

```
PS C:\Users\User\OneDrive\Desktop\ai assisted coding> pip install flask
hon.exe -m pip install --upgrade pip
PS C:\Users\User\OneDrive\Desktop\ai assisted coding> & C:/Users/User/AppData/Local/Microsoft/WindowsApps/python3.11.exe
 * Serving Flask app 'task1'
 * 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: 270-575-033
 * Detected change in 'c:\\Users\\User\\OneDrive\\Desktop\\ai assisted coding\\lab15_4\\task1', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 270-575-033
```



OBSERVATION:

Defines a minimal Flask app with GET / returning {"message":"Welcome to AI-assisted API"}. Runs locally on 127.0.0.1:5000 with debug=True.Requires Flask (pip install flask). Note: disable debug in production and change host to 0.0.0.0 to expose external.

TASK2:

PROMPT: Create a Flask backend with two endpoints:

1. GET /items → returns a JSON list of items (initially empty).2. POST /items → accepts JSON payload like {"name":"Book","price":200}, adds it to the list, and returns {"message": "Item added", "item": {...}} with status code 201.Use request.get_json () to parse incoming data and jsonify() for responses. Include a global items list to store data in memory. Enable debug mode and run on localhost.

CODE:

task2.py

lab15_4 > task2.py > delete_item

```python
from flask import Flask, jsonify, request
app = Flask(__name__)
# In-memory data store (replace with a database for production)
items = []
# GET all items
@app.route('/items', methods=['GET'])
def get_items():
    return jsonify(items) # Returns the list of items as a JSON response
# POST a new item
@app.route('/items', methods=['POST'])
def add_item():
    try:
        data = request.get_json()  # Get JSON data from the request body
        if data is None:
            return jsonify({"error": "Request body must contain valid JSON"}), 400
        if not isinstance(data, dict):
            return jsonify({"error": "Request body must be a JSON object"}), 400

        items.append(data)  # Add the new item to the list
        return jsonify({"message": "Item added", "item": data}), 201  # Return success message and the
    except Exception as e:
        # Handle potential errors (e.g., invalid JSON)
        return jsonify({"error": str(e)}), 400
# PUT (Update) an existing item by its index
@app.route('/items/<int:index>', methods=['PUT'])
def update_item(index):
    if 0 <= index < len(items):
        data = request.get_json()
        if data is None:
            return jsonify({"error": "Request body must contain valid JSON"}), 400
        items[index] = data
        return jsonify({"message": "Item updated", "item": data})
    return jsonify({"error": "Item not found"}), 404
# DELETE an item by its index
```

```python
# DELETE an item by its index
@app.route('/items/<int:index>', methods=['DELETE'])
def delete_item(index):
    if 0 <= index < len(items):
        removed_item = items.pop(index)
        return jsonify({"message": "Item deleted", "item": removed_item})
    return jsonify({"error": "Item not found"}), 404
if __name__ == '__main__':
    app.run(debug=True)  # Enable debug mode for development
```

OUTPUT:


```
[
  {
    "name": "Notebook",
    "price": 250
  }
]
```

OBSERVATION:

This is a functional, straightforward Flask application that correctly implements a basic CRUD API.

The main weakness is using the list index to identify items for updates and deletions (/items/<int:index>). This is not a stable approach for an API, because if you delete an item, the indices of all subsequent items will change, breaking any client references to them.

A more robust solution would be to assign a unique, persistent ID (like a UUID) to each item when it's created and use that ID in the URL to reference it (e.g., /items/<string:item_id>).

TASK3:

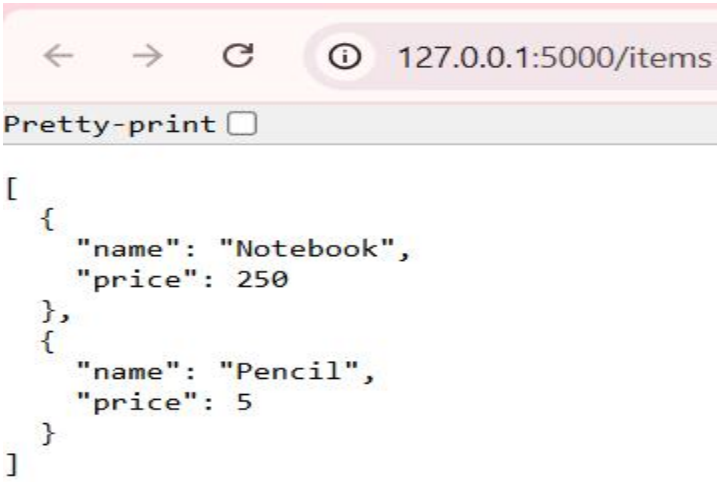
- PROMPT: "Create a minimal Python Flask server with one endpoint / that returns JSON {"message":"Welcome to AI-assisted API"}; run locally on port 5000 with debug=True."
- "Add a PUT endpoint PUT /items/<int:index> that updates an existing item by index (404 if not found) and returns {"message":"Item updated","item":...}."

CODE:

ab15_4 > task3.py

```python
from flask import Flask, jsonify, request
app = Flask(__name__)
# Simple in-memory items list for demo purposes
items = [
    {"name": "Pen", "price": 10},
    {"name": "Pencil", "price": 5}
]
@app.route('/items', methods=['GET'])
def list_items():
    return jsonify(items)
# PUT /items/<int:index> - update item by 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()
    if not data or not isinstance(data, dict):
        return jsonify({"error": "Invalid payload"}), 400
    # Update only provided fields to preserve other fields
    items[index].update(data)
    return jsonify({"message": "Item updated", "item": items[index]})
if __name__ == '__main__':
    app.run(host='127.0.0.1', port=5000, debug=True)
```

OUTPUT:



OBSERVATION:

- Created
  files: lab15_4/task1/app.py, lab15_4/task1/requirements.txt, lab15_4/task 1/README.md.
- Added task3.py with an in-memory items list, GET /items, and PUT /items/<int:index> that updates fields and returns the updated item.
- Expected runtime: server at http://127.0.0.1:5000/ and PUT /items/0 with {"name":"Notebook","price":250} returns {"message":"Item updated","item":{"name":"Notebook","price":250}}.
- Note: dependency install was not completed in-session (pip install was cancelled) and the last local run of task3.py exited with code 1 — you may need to install Flask (pip install -r lab15_4/task1/requirements.txt) or fix local edits before running.

TASK4:

PROMPT: Start the server: run Python on task4.py using the full path (quote the script path because of spaces).
Observe: Flask prints "Running on http://127.0.0.1:5000" and the process stays running.

Test endpoint: send an HTTP GET to http://127.0.0.1:5000/items. Observe: you receive JSON with the Notebook item (and the Flask terminal logs a 200 GET).

CODE:

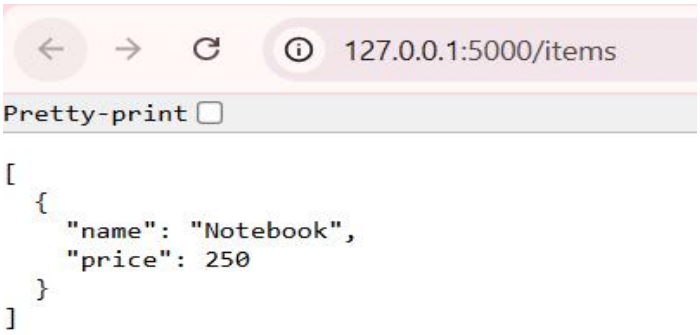
task4.py

lab15_4 > task4.py > ...

```python
from flask import Flask, jsonify, request
app = Flask(__name__)
# Sample data
items = [{"name": "Notebook", "price": 250}]
# 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})
# GET /items
@app.route('/items', methods=['GET'])
def get_items():
    return jsonify(items)
# POST /items
@app.route('/items', methods=['POST'])
def add_item():
    data = request.get_json()
    if not data or 'name' not in data or 'price' not in data:
        return jsonify({"error": "Invalid item format"}), 400
    items.append(data)
    return jsonify({"message": "Item added", "item": data}), 201

if __name__ == '__main__':
    # Disable the reloader on Windows to avoid socket.fromfd WinError in
    # some terminal environments. Keep debug on for helpful errors.
    app.run(debug=True, use_reloader=False)
```

OUTPUT:

```
PS C:\Users\User\OneDrive\Desktop\ai assisted coding> Invoke-RestMethod -Uri http://127.0.0.1:5000/items -Metho
PS C:\Users\User\OneDrive\Desktop\ai assisted coding> ^C
d Get

name     price
----     -----
Notebook   250
```



OBSERVATION:

- Start server: shows "Running on http://127.0.0.1:5000" and stays running.
- GET /items: returns JSON array with the Notebook item (HTTP 200).
- POST /items: returns HTTP 201 with the added item.
- DELETE /items/<index>: returns HTTP 200 with deleted item (or 404 if index invalid).
- Failure sign: a stack trace with "socket.fromfd" / "dup" / "WinError 10038" (reloader/socket issue).

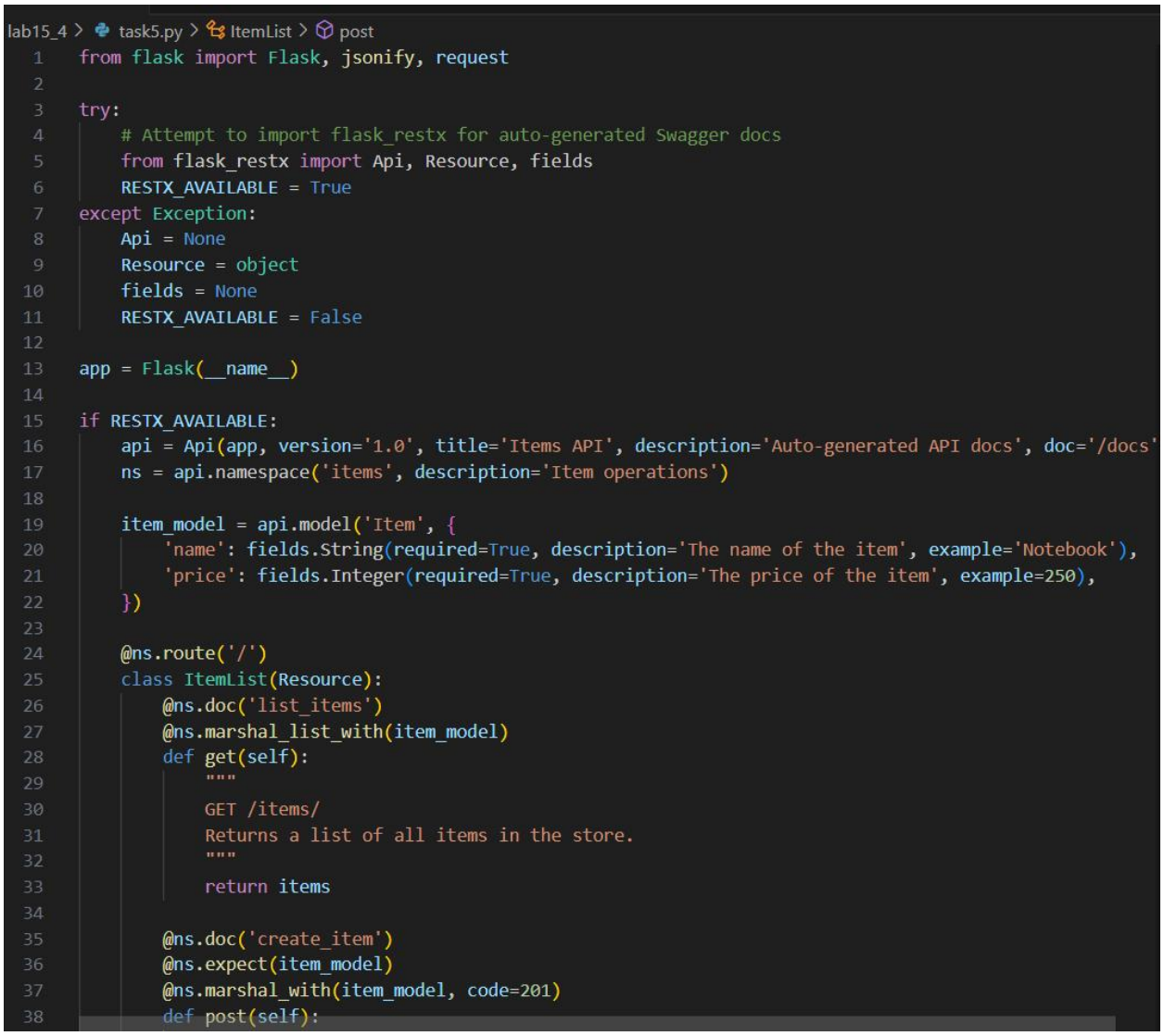
TASK5:

PROMPT:

- Start server:
  python3.11.exe "c:/Users/User/OneDrive/Desktop/ai assisted coding/lab15_4/task5.py"
- GET items:
  Invoke-RestMethod -Uri http://127.0.0.1:5000/items -Method Get
- POST item:
  Invoke-RestMethod -Uri http://127.0.0.1:5000/items -Method Post -ContentType 'application/json' -Body '{"name":"Pen","price":5}'
- DELETE item (index 0):
  Invoke-RestMethod -Uri http://127.0.0.1:5000/items/0 -Method Delete

- Docs (browser or PowerShell):
  Open http://127.0.0.1:5000/docs (or Invoke-RestMethod -Uri http://127.0.0.1:5000/docs -Method Get)

CODE:



```
        def post(self):
            """
            POST /items/
            Add a new item. Expects JSON with 'name' and 'price'.
            """
            data = request.get_json()
            items.append(data)
            return data, 201

    @ns.route('/<int:index>')
    @ns.response(404, 'Item not found')
    @ns.param('index', 'The item index')
    class Item(Resource):
        @ns.doc('delete_item')
        def delete(self, index):
            """
            DELETE /items/<index>
            Delete the item at the given index and return it.
            """
            if index < 0 or index >= len(items):
                api.abort(404, "Item not found")
            removed = items.pop(index)
            return {'message': 'Item deleted', 'item': removed}

else:
    # Fallback simple Flask routes with docstrings and inline comments
    items = [{'name': 'Notebook', 'price': 250}]

    @app.route('/items', methods=['GET'])
    def get_items():
        """
        GET /items
        Returns a list of all items in the store.

        Response (200):
        [
          {"name": "Notebook", "price": 250}
```

```
        ]
        """
        # Return current in-memory list of items as JSON
        return jsonify(items)

    @app.route('/items', methods=['POST'])
    def add_item():
        """
        POST /items
        Add a new item. Request JSON must include 'name' (string) and 'price' (int).

        Sample request body:
        {"name": "Pen", "price": 5}

        Responses:
        201 - Item created and returned
        400 - Invalid item format
        """
        data = request.get_json()
        # Validate the payload structure
        if not data or 'name' not in data or 'price' not in data:
            return jsonify({'error': 'Invalid item format'}), 400
        items.append(data)
        return jsonify({'message': 'Item added', 'item': data}), 201

    @app.route('/items/<int:index>', methods=['DELETE'])
    def delete_item(index):
        """
        DELETE /items/<index>
        Delete the item at the specified index. Returns the deleted item.

        Responses:
        200 - Item deleted
        404 - Item not found
        """
        if index < 0 or index >= len(items):
            return jsonify({'error': 'Item not found'}), 404
```

```
        removed = items.pop(index)
        return jsonify({'message': 'Item deleted', 'item': removed})

    # Optionally provide a /docs hint when RESTX is not available
    @app.route('/docs')
    def docs_hint():
        """
        /docs (hint)
        When flask_restx is not installed, this endpoint hints how to enable
        Swagger documentation by installing flask-restx. It does not serve
        documentation itself in this fallback mode.
        """
        return jsonify({'info': 'Install flask-restx to enable /docs UI: pip install flask-restx'})

if __name__ == '__main__':
    # Run without the reloader on Windows to avoid socket.fromfd WinError in some terminals
    app.run(debug=True, use_reloader=False)
```

OUTPUT:

```
PS C:\Users\User\OneDrive\Desktop\ai assisted coding> C:/Users/User/AppData/Local/Microsoft/WindowsApps/python3
.11.exe "c:/Users/User/OneDrive/Desktop/ai assisted coding/lab15_4/task5.py"
 * Serving Flask app 'task5'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server i
nstead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
PS C:\Users\User\OneDrive\Desktop\ai assisted coding> Invoke-RestMethod -Uri http://127.0.0.1:5000/docs -Method
 Get

info
----
Install flask-restx to enable /docs UI: pip install flask-restx


PS C:\Users\User\OneDrive\Desktop\ai assisted coding> Invoke-RestMethod -Uri http://127.0.0.1:5000/items -Metho
d Get | ConvertTo-Json -Depth 5
{
    "value":  [
                  {
                      "name":  "Notebook",
                      "price":  250
                  }
              ],
    "Count":  1
}
PS C:\Users\User\OneDrive\Desktop\ai assisted coding> ^C
PS C:\Users\User\OneDrive\Desktop\ai assisted coding> Invoke-RestMethod -Uri http://127.0.0.1:5000/items -Metho
d Post -ContentType 'application/json' -Body '{"name":"Pen","price":5}' | ConvertTo-Json -Depth 5
{
    "item":  {
                 "name":  "Pen",
                 "price":  5
             },
    "message":  "Item added"
}
PS C:\Users\User\OneDrive\Desktop\ai assisted coding> ^C
PS C:\Users\User\OneDrive\Desktop\ai assisted coding> Invoke-RestMethod -Uri http://127.0.0.1:5000/items/1 -Met
hod Delete | ConvertTo-Json -Depth 5
{
    "item":  {
```

- OBSERVATION: Start server: prints "Serving Flask app 'task5'", "Running on http://127.0.0.1:5000" and stays running.
- GET /items: returns JSON array (initially contains Notebook) and server logs GET 200.
- POST /items: returns 201 with {"message":"Item added","item":{...}}; subsequent GET includes the new item.
- DELETE /items/0: returns 200 with {"message":"Item deleted","item":{...}} (404 if index invalid).

- /docs: shows Swagger UI if flask-restx is installed; otherwise returns a JSON hint telling how to install it.