

AI ASSISTED CODING

LAB:15.4

ROLLNO:2403A52084

BATCH:04

TASK1:

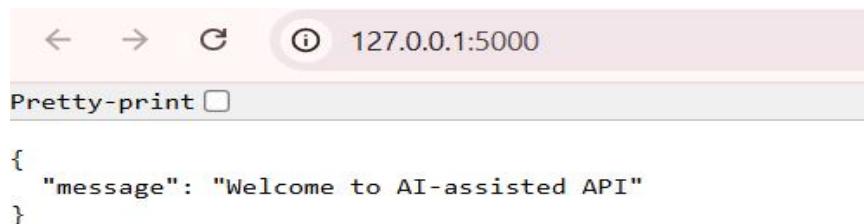
PROMPT: Generate Python code to set up a basic Flask backend with one endpoint. The endpoint should be / and return a JSON response: {"message": "Welcome to AI-assisted API"}. The server should run locally on port 5000 with debug mode enabled.

CODE:

```
task1      X
lab15_4 > task1 > ...
1  from flask import Flask, jsonify
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def hello_world():
7      return jsonify({"message": "Welcome to AI-assisted API"})
8
9  if __name__ == '__main__':
10     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
```



```
{  
  "message": "Welcome to AI-assisted API"  
}
```

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.

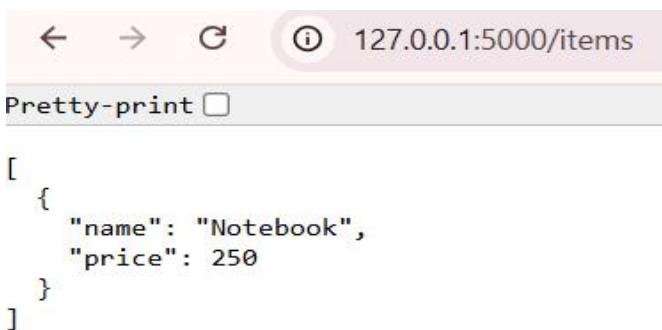
CODE:

```

task2.py  X
lab15_4 > task2.py > delete_item
 1  from flask import Flask, jsonify, request
 2  app = Flask(__name__)
 3  # In-memory data store (replace with a database for production)
 4  items = []
 5  # GET all items
 6  @app.route('/items', methods=['GET'])
 7  def get_items():
 8      return jsonify(items) # Returns the list of items as a JSON response
 9  # POST a new item
10  @app.route('/items', methods=['POST'])
11  def add_item():
12      try:
13          data = request.get_json() # Get JSON data from the request body
14          if data is None:
15              return jsonify({"error": "Request body must contain valid JSON"}), 400
16          if not isinstance(data, dict):
17              return jsonify({"error": "Request body must be a JSON object"}), 400
18
19          items.append(data) # Add the new item to the list
20          return jsonify({"message": "Item added", "item": data}), 201 # Return success message and the
21      except Exception as e:
22          # Handle potential errors (e.g., invalid JSON)
23          return jsonify({"error": str(e)}), 400
24  # PUT (Update) an existing item by its index
25  @app.route('/items/<int:index>', methods=['PUT'])
26  def update_item(index):
27      if 0 <= index < len(items):
28          data = request.get_json()
29          if data is None:
30              return jsonify({"error": "Request body must contain valid JSON"}), 400
31          items[index] = data
32          return jsonify({"message": "Item updated", "item": data})
33      return jsonify({"error": "Item not found"}), 404
34  # DELETE an item by its index
35
36  @app.route('/items/<int:index>', methods=['DELETE'])
37  def delete_item(index):
38      if 0 <= index < len(items):
39          removed_item = items.pop(index)
40          return jsonify({"message": "Item deleted", "item": removed_item})
41      return jsonify({"error": "Item not found"}), 404
42  if __name__ == '__main__':
43      app.run(debug=True) # Enable debug mode for development

```

OUTPUT:



The screenshot shows a browser window with the URL 127.0.0.1:5000/items. The page displays a single JSON array element: [{}]. There are navigation buttons at the top left and a 'Pretty-print' checkbox at the bottom left.

```
[{"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>).

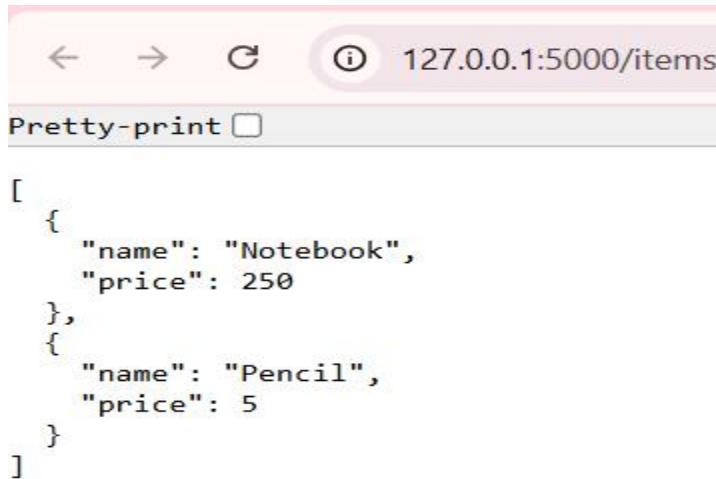
TASK3:

- **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
 1  from flask import Flask, jsonify, request
 2  app = Flask(__name__)
 3  # Simple in-memory items list for demo purposes
 4  items = [
 5      {"name": "Pen", "price": 10},
 6      {"name": "Pencil", "price": 5}
 7  ]
 8  @app.route('/items', methods=['GET'])
 9  def list_items():
10      return jsonify(items)
11  # PUT /items/<int:index> - update item by index
12  @app.route('/items/<int:index>', methods=['PUT'])
13  def update_item(index):
14      if index < 0 or index >= len(items):
15          return jsonify({"error": "Item not found"}), 404
16      data = request.get_json()
17      if not data or not isinstance(data, dict):
18          return jsonify({"error": "Invalid payload"}), 400
19      # Update only provided fields to preserve other fields
20      items[index].update(data)
21      return jsonify({"message": "Item updated", "item": items[index]})
22  if __name__ == '__main__':
23      app.run(host='127.0.0.1', port=5000, debug=True)
```

OUTPUT:



```
[{"name": "Notebook", "price": 250}, {"name": "Pencil", "price": 5}]
```

OBSERVATION:

- Created files: lab15_4/task1/app.py, lab15_4/task1/requirements.txt, lab15_4/task1/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.

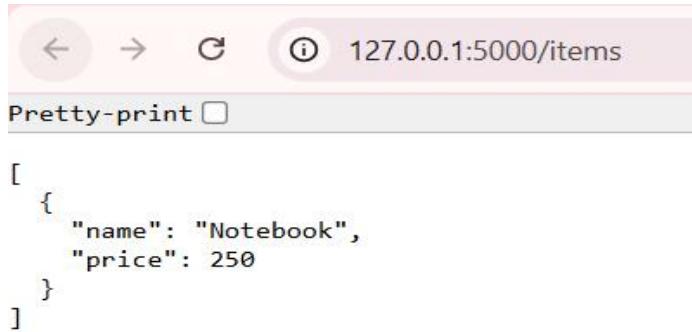
Test endpoint: send an HTTP GET to `http://127.0.0.1:5000/items`.
Observe: you receive JSON with the Notebook item (and the Flask terminal logs a 200 GET).

CODE:

```
task4.py  x
lab15_4 > task4.py > ...
1  from flask import Flask, jsonify, request
2  app = Flask(__name__)
3  # Sample data
4  items = [{"name": "Notebook", "price": 250}]
5  # DELETE /items/<int:index>
6  @app.route('/items/<int:index>', methods=['DELETE'])
7  def delete_item(index):
8      if index < 0 or index >= len(items):
9          return jsonify({"error": "Item not found"}), 404
10     removed_item = items.pop(index)
11     return jsonify({"message": "Item deleted", "item": removed_item})
12 # GET /items
13 @app.route('/items', methods=['GET'])
14 def get_items():
15     return jsonify(items)
16 # POST /items
17 @app.route('/items', methods=['POST'])
18 def add_item():
19     data = request.get_json()
20     if not data or 'name' not in data or 'price' not in data:
21         return jsonify({"error": "Invalid item format"}), 400
22     items.append(data)
23     return jsonify({"message": "Item added", "item": data}), 201
24
25 if __name__ == '__main__':
26     # Disable the reloader on Windows to avoid socket.fromfd WinError in
27     # some terminal environments. Keep debug on for helpful errors.
28     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 -Method Get
name      price
----      -----
Notebook   250
```



```
[  
  {  
    "name": "Notebook",  
    "price": 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](http://127.0.0.1:5000) "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:

```
lab15_4 > task5.py > ItemList > post
1   from flask import Flask, jsonify, request
2
3   try:
4       # Attempt to import flask_restx for auto-generated Swagger docs
5       from flask_restx import Api, Resource, fields
6       RESTX_AVAILABLE = True
7   except Exception:
8       Api = None
9       Resource = object
10      fields = None
11      RESTX_AVAILABLE = False
12
13  app = Flask(__name__)
14
15  if RESTX_AVAILABLE:
16      api = Api(app, version='1.0', title='Items API', description='Auto-generated API docs', doc='/docs')
17      ns = api.namespace('items', description='Item operations')
18
19      item_model = api.model('Item', {
20          'name': fields.String(required=True, description='The name of the item', example='Notebook'),
21          'price': fields.Integer(required=True, description='The price of the item', example=250),
22      })
23
24      @ns.route('/')
25      class ItemList(Resource):
26          @ns.doc('list_items')
27          @ns.marshal_list_with(item_model)
28          def get(self):
29              """
30                  GET /items/
31                  Returns a list of all items in the store.
32                  """
33              return items
34
35          @ns.doc('create_item')
36          @ns.expect(item_model)
37          @ns.marshal_with(item_model, code=201)
38          def post(self):
```

```
38     def post(self):
39         """
40             POST /items/
41             Add a new item. Expects JSON with 'name' and 'price'.
42             """
43         data = request.get_json()
44         items.append(data)
45         return data, 201
46
47     @ns.route('/<int:index>')
48     @ns.response(404, 'Item not found')
49     @ns.param('index', 'The item index')
50     class Item(Resource):
51         @ns.doc('delete_item')
52         def delete(self, index):
53             """
54                 DELETE /items/<index>
55                 Delete the item at the given index and return it.
56                 """
57         if index < 0 or index >= len(items):
58             api.abort(404, "Item not found")
59         removed = items.pop(index)
60         return {'message': 'Item deleted', 'item': removed}
61
62     else:
63         # Fallback simple Flask routes with docstrings and inline comments
64         items = [{'name': 'Notebook', 'price': 250}]
65
66     @app.route('/items', methods=['GET'])
67     def get_items():
68         """
69             GET /items
70             Returns a list of all items in the store.
71
72             Response (200):
73             [
74                 {"name": "Notebook", "price": 250}
```

```

75     ]
76     """
77     # Return current in-memory list of items as JSON
78     return jsonify(items)
79
80     @app.route('/items', methods=['POST'])
81     def add_item():
82         """
83             POST /items
84             Add a new item. Request JSON must include 'name' (string) and 'price' (int).
85
86             Sample request body:
87             {"name": "Pen", "price": 5}
88
89             Responses:
90             201 - Item created and returned
91             400 - Invalid item format
92             """
93             data = request.get_json()
94             # Validate the payload structure
95             if not data or 'name' not in data or 'price' not in data:
96                 return jsonify({'error': 'Invalid item format'}), 400
97             items.append(data)
98             return jsonify({'message': 'Item added', 'item': data}), 201
99
100    @app.route('/items/<int:index>', methods=['DELETE'])
101    def delete_item(index):
102        """
103            DELETE /items/<index>
104            Delete the item at the specified index. Returns the deleted item.
105
106            Responses:
107            200 - Item deleted
108            404 - Item not found
109            """
110            if index < 0 or index >= len(items):
111                return jsonify({'error': 'Item not found'}), 404
112
113            removed = items.pop(index)
114            return jsonify({'message': 'Item deleted', 'item': removed})
115
116            # Optionally provide a /docs hint when RESTX is not available
117            @app.route('/docs')
118            def docs_hint():
119                """
120                    /docs (hint)
121                    When flask_restx is not installed, this endpoint hints how to enable
122                    Swagger documentation by installing flask-restx. It does not serve
123                    documentation itself in this fallback mode.
124                    """
125
126            if __name__ == '__main__':
127                # Run without the reloader on Windows to avoid socket.fromfd WinError in some terminals
128                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 instead.
  * 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 -Method 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 -Method 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 -Method Delete | ConvertTo-Json -Depth 5
{
    "item": {

        "item": {
            "name": "Pen",
            "price": 5
        },
        "message": "Item deleted"
    }
}
```

- 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.