

# Assignment 15.4

**Name :**Nagarjuna.

**Ht no :** 2403a52064

**Batch no :** 03

## **TASK- 1: Prompt:**

"Using AI assistance, generate a basic Flask backend application.

**Requirements:**

- Install Flask.
- Create a Python server with a single endpoint /.
- The endpoint should return a JSON response:  
{ "message": "Welcome to AI-assisted API" }
- The app should run in debug mode on http://127.0.0.1:5000/.

## **CODE :**

```
# Import the required libraries

from flask import Flask, jsonify

# Create a Flask application instance

app = Flask(__name__)

# Define a single route (endpoint)

@app.route('/')

def home():

    # Return a JSON response

    return jsonify({"message": "Welcome to AI-assisted API"})

#Run the Flask application

if __name__ == "__main__":

    app.run(debug=True)
```

## **OUTPUT :**

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

## EXPLANATION:

- ❓ **Import Flask and jsonify** – brings in the tools needed to create a web app and send JSON data.
- ❓ **Create an app** using `app = Flask(__name__)` – this starts your Flask application.
- ❓ **Define a route** with `@app.route('/')` – this tells Flask what to do when someone visits the home URL (/).
- ❓ **Create a function home()** – it runs when you open the URL.
- ❓ **Return a JSON message** using `jsonify({"message": "Welcome to AI-assisted API"})`.
- ❓ **Start the server** with `app.run(debug=True)` – this runs the app on your computer.
- ❓ **Open the browser or Postman** at `http://127.0.0.1:5000/` – you'll see the message:

## TASK-2

### PROMPT:

Use AI to create a simple Flask backend with two endpoints.

Requirements:

- Use a Python list to store items.
- Create a GET endpoint `/items` to show all items.
- Create a POST endpoint `/items` to add a new item.

When a new item is added, return a message and the item details.

### CODE :

```
from flask import Flask, jsonify, request

# Initialize Flask app
app = Flask(__name__)

# In-memory list to store items
items = []

# GET all items (READ)
@app.route('/items', methods=['GET'])
```

```

def get_items():
    return jsonify(items)

# -----

# POST a new item (CREATE)

# -----

@app.route('/items', methods=['POST'])
def add_item():

    data = request.get_json() # Get JSON data from request

    items.append(data)      # Add new item to the list

    return jsonify({"message": "Item added", "item": data}), 201

# Run the Flask app

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

```

OUTPUT :

REQUEST:

\_\_\_\_\_

GET <http://127.0.0.1:5000/items>

RESPONSE:

\_\_\_\_\_ []

EXPLANATION :

1. **Create app & list** → app = Flask(\_\_name\_\_), items = []
2. **GET /items** → returns all items in JSON ([] initially)
3. **POST /items** → adds new item from JSON request, returns success message
4. **Run server** → app.run(debug=True) → access at <http://127.0.0.1:5000/items>

TASK-3 :

PrOmPT:

"Create a Flask PUT endpoint to update an existing item in a Python list.

Requirements:

- The endpoint URL should be /items/<int:index> where index is the item's position in the list.
- If the index is invalid, return a 404 error with JSON {"error": "Item not found"}.

- If valid, replace the item at that index with JSON data from the request.
- Return a success message in JSON: {"message": "Item updated", "item": data}."

## CODE :

```
from flask import Flask, jsonify, request

# Initialize Flask app
app = Flask(__name__)

# In-memory list to store items
items = []

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

    return jsonify(items)

# -----
# POST a new item (CREATE)
# -----
@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> (UPDATE)
# -----
@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})

#Run the Flask app

if __name__ == "__main__":

    app.run(debug=True)
```

## OUTPUT :

INITIAL GET /ITEMs

### rEqUEST:

GET http://127.0.0.1:5000/items

### rESPONSE:

[]

POST /ITEMs (ADD AN ITEM)

### rEqUEST:

POST http://127.0.0.1:5000/items

Content-Type: application/json

Body:

```
{
  "name": "Book",
  "price": 200
}
```

### rESPONSE:

```
{
  "message": "Item added",
  "item": {
    "name": "Book",
    "price": 200
  }
}
```

GET /ITEMs (AfTeR ADDING ITEM)

### rEqUEST:

GET http://127.0.0.1:5000/itemS

rESPONSE:

```
[  
  
  {  
    "name": "Book",  
    "price": 200  
  }  
]  
PUT /ITEMS/0 (UPDATE THE ITEM AT INDEX 0)
```

rEqUEST:

PUT http://127.0.0.1:5000/items/0

Content-Type: application/json

Body:

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

rESPONSE:

```
{  
  
  "message": "Item updated",  
  "item": {  
    "name": "Notebook",  
    "price": 250  
  }  
}
```

GET /ITEMS (AFTER UPDATE)

rEqUEST:

GET http://127.0.0.1:5000/items

rESPONSE:

```
[  
  
  {  
    "name": "Notebook",
```

```
"price": 250
}
]
```

PUT /ITEMS/5 (INVALID INDEX)

rEQUEST:

PUT http://127.0.0.1:5000/items/5

Content-Type: application/json

Body:

```
{
  "name": "Pen",
  "price": 50
}
```

Response:

```
{
  "error": "Item not found"
}
```

### EXPLANATION :

🔗 Endpoint: /items/<int:index> → updates item at given index.

🔗 Check & update:

- Invalid index → {"error": "Item not found"}
- Valid index → replace item with new JSON data.

🔗 Response: {"message": "Item updated", "item": data}

### TASK- 4 :

#### PrOmPT:

– Create a Flask DELETE endpoint to remove an item from a Python list by its index.

rEQUIrEmENTS:

- The endpoint URL should be /items/<int:index> where index is the position of the item in the list.
- If the index is invalid, return a 404 error with JSON {"error": "Item not found"}.

- If valid, remove the item from the list and return a JSON response: {"message": "Item deleted", "item": removed item}."

## CODE :

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

```
#Initialize Flask app
```

```
app = Flask(__name__)
```

```
#In-memory list to store items
```

```
items = []
```

```
#-----
```

```
#GET all items (READ)
```

```
#-----
```

```
@app.route('/items', methods=['GET'])
```

```
def get_items():
```

```
    return jsonify(items)
```

```
#-----
```

```
#POST a new item (CREATE)
```

```
#-----
```

```
@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> (UPDATE)
```

```
#-----
```

```
@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> (DELETE)

#-----

@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) # Remove item from list

    return jsonify({"message": "Item deleted", "item": removed_item})

#Run the Flask app

if __name__ == "__main__":

    app.run(debug=True)

```

## OUTPUT :

ADD AN ITEM (POST /ITEMS)

REQUEST BODY:

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

RESPONSE:

```
{
  "message": "Item added",
  "item": {"name": "Notebook", "price": 250}
}
```

---

DELETE THE ITEM (DELETE /ITEMS/0)

RESPONSE:

```
{
  "message": "Item deleted",
  "item": {"name": "Notebook", "price": 250}
}
```

---

ChECK ITEmS LIST (GET /ITEmS)

rESPONSE:

```
[]
```

---

DELETE INvALID INDEX (DELETE /ITEmS/5)

rESPONSE:

```
{
  "error": "Item not found"
}
```

EXPLANATION :

1. Endpoint: /items/<int:index> → deletes the item at the given index in the list.
2. Check index:
  - o Invalid index → return {"error": "Item not found"} with 404 status
  - o Valid index → proceed to remove the item.
3. Remove item: Use items.pop(index) to delete the item from the list.
4. Return response: {"message": "Item deleted", "item": removed\_item} showing the deleted item.

TASK- 5: PrOmPT:

"Add inline comments and docstrings to all Flask API endpoints.

Requirements:

- Each endpoint should have a docstring explaining:
  - The URL and HTTP method
  - The purpose of the endpoint
  - Expected request and response (if applicable)
- Add inline comments inside each function to explain key steps.
- Optionally, integrate Swagger or Flask-RESTX to auto-generate API documentation at /docs."

## CODE:

```
from flask import Flask, jsonify, request

from flask_restx import Api, Resource, fields


#Initialize Flask app

app = Flask(__name__)

api = Api(app, doc="/docs", title="Item Store API", description="Simple CRUD API with Swagger documentation")


#In-memory list to store items

items = []


#Model for Swagger documentation
item_model = api.model('Item', {

    'name': fields.String(required=True, description='Name of the item'),

    'price': fields.Float(required=True, description='Price of the item')

})


#-----

#GET all items

#-----

@api.route('/items')

classItemList(Resource):

    @api.doc(description="Get all items in the store")

    defget(self):

        """

        GET /items

        Returns a list of all items in the store.

        """

        return jsonify(items)


@api.expect(item_model)

@api.doc(description="Add a new item to the store")

defpost(self):

    """

    POST /items
```

Adds a new item to the store.

Expects JSON payload with 'name' and 'price'.

```
"""
```

```
data = request.get_json()
```

```
items.append(data)
```

```
return {"message": "Item added", "item": data}, 201
```

```
#-----
```

```
#PUT /items/<int:index>
```

```
#-----
```

```
@api.route('/items/<int:index>')
```

```
classItem(Resource):
```

```
    @api.expect(item_model)
```

```
    @api.doc(description="Update an existing item by index")
```

```
    defput(self, index):
```

```
        """
```

```
        PUT/items/<index>
```

```
        Updates an item at the given index.
```

```
        Expects JSON payload with 'name' and 'price'.
```

```
        """
```

```
        if index < 0 or index >= len(items):
```

```
            return {"error": "Item not found"}, 404
```

```
        data = request.get_json()
```

```
        items[index] = data
```

```
        return {"message": "Item updated", "item": data}
```

```
    @api.doc(description="Delete an existing item by index")
```

```
    defdelete(self, index):
```

```
        """
```

```
        DELETE /items/<index>
```

```
        Deletes an item at the given index.
```

```
        """
```

```
        if index < 0 or index >= len(items):
```

```
            return {"error": "Item not found"}, 404
```

```
        removed_item = items.pop(index)
```

```
return {"message": "Item deleted", "item": removed_item}
```

```
#Run the Flask app
```

```
if __name__ == "__main__":
```

```
    app.run(debug=True)
```

## OUTPUT :

GET /items (initially empty)

[]

POST /items

Request Body:

```
{"name": "Book", "price": 200}
```

Response:

```
{  
  "message": "Item added",  
  "item": {"name": "Book", "price": 200}  
}
```

PUT /items/0

Request Body:

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

Response:

```
{  
  "message": "Item updated",  
  "item": {"name": "Notebook", "price": 250}  
}
```

DELETE /items/0

Response:

```
{  
  "message": "Item deleted",  
  "item": {"name": "Notebook", "price": 250}  
}
```

### EXPLANATION :

- ❑ Docstrings & Comments: Explain URL, method, purpose, and payload for each endpoint.
- ❑ Swagger (/docs): Auto-generates interactive API docs with endpoint details and sample requests.
- ❑ CRUD Endpoints Covered: GET, POST, PUT, DELETE — all visible and testable via Swagger.