

## Lab 15 – Backend API Development: Creating RESTful services with AI

### Task 1: Setup Flask Backend

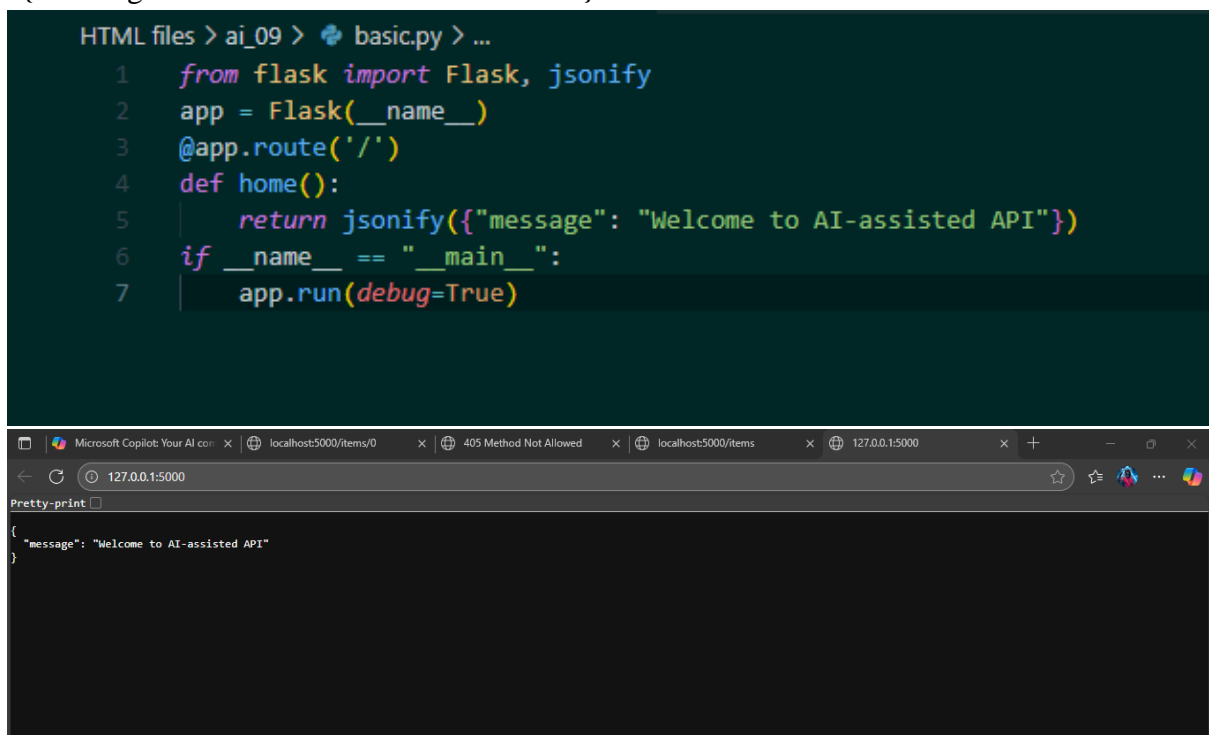
#### Instructions:

- Install Flask and set up a basic Python server.
- Use AI to generate starter code for a simple backend with a single endpoint.

**Prompt:** Generate a basic Flask backend with one endpoint that returns a welcome message in JSON.

#### Expected Output:

- Running the server should show:  
http://127.0.0.1:5000/
- Accessing it in the browser or Postman returns:  
{"message": "Welcome to AI-assisted API"}



The screenshot shows a code editor with a Python script for a Flask application. The script defines a Flask app, sets a route for the root path, and returns a JSON response. Below the code editor, a web browser window is open at the URL http://127.0.0.1:5000/. The browser's developer tools show the JSON response returned by the API.

```
HTML files > ai_09 > basic.py > ...
1  from flask import Flask, jsonify
2  app = Flask(__name__)
3  @app.route('/')
4  def home():
5      return jsonify({"message": "Welcome to AI-assisted API"})
6  if __name__ == "__main__":
7      app.run(debug=True)
```

Microsoft Copilot: Your AI code assistant

localhost:5000/items/0 405 Method Not Allowed localhost:5000/items 127.0.0.1:5000

127.0.0.1:5000

Pretty-print

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

### Task 2: Create a CRUD API – Read and Create

#### Instructions:

- Use AI to implement endpoints to **list all items** and **add a new item**.

Use an in-memory Python list to store items

**Prompt:** Create a Flask API with two endpoints: one to list all items (GET /items) and one to add a new item (POST /items). Use an in-memory Python list to store items. Return JSON responses.

#### Expected Output:

- **GET /items** → [] initially
- **POST /items** with payload {"name":"Book","price":200} → {"message": "Item added", "item": {"name":"Book","price":200}}
- **GET /items** now returns:  
[{"name":"Book","price":200}]

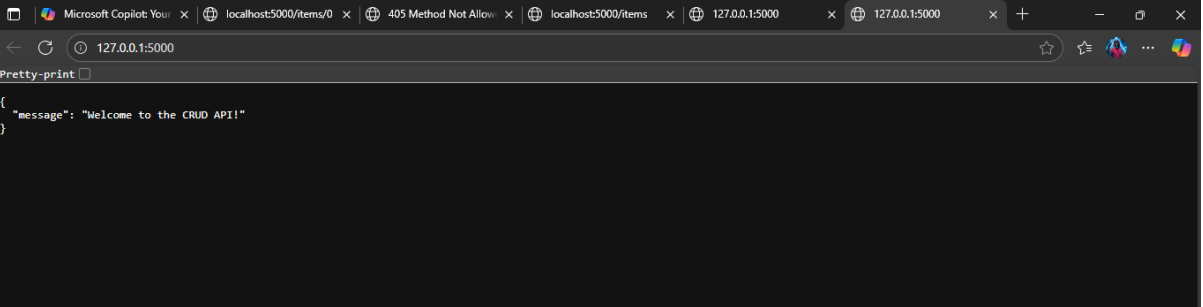
HTML files > ai\_09 > crud.api.py > ...

```
1  from flask import Flask, jsonify, request
2  import threading
3  import time
4  import requests
5
6  app = Flask(__name__)
7
8  # In-memory list to store items
9  items = []
10
11 # GET /items → List all items
12 @app.route('/items', methods=['GET'])
13 def get_items():
14     return jsonify(items), 200
15
16 # POST /items → Add a new item
17 @app.route('/items', methods=['POST'])
18 def add_item():
19     data = request.get_json()
20
21     # Validate input
22     if not data or 'name' not in data or 'price' not in data:
23         return jsonify({"error": "Missing 'name' or 'price'"}), 400
24
25     try:
26         data['price'] = float(data['price'])
27     except (ValueError, TypeError):
28         return jsonify({"error": "'price' must be a number"}), 400
```

HTML files > ai\_09 > crud.api.py > ...

```
18 def add_item():
28     return jsonify({"error": "'price' must be a number"}), 400
29
30     items.append(data)
31     return jsonify({"message": "Item added", "item": data}), 201
32
33 # Optional root route
34 @app.route('/')
35 def index():
36     return jsonify({"message": "Welcome to the CRUD API!"}), 200
37
38 # Function to test POST request after server starts
39 def test_post():
40     time.sleep(1) # Wait for server to start
41     url = "http://127.0.0.1:5000/items"
42     data = {"name": "Book", "price": 200}
43     response = requests.post(url, json=data)
44     print("POST Response:", response.json())
45
46 if __name__ == "__main__":
47     threading.Thread(target=test_post).start()
48     app.run(debug=True)
49
```

```
PS C:\c codes> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/c codes/HTML files/ai_09/crud.api.py"
* Serving Flask app 'crud.api'
* 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: 121-704-370
127.0.0.1 - - [23/Oct/2025 14:22:03] "POST /items HTTP/1.1" 201 -
POST Response: {'item': {'name': 'Book', 'price': 200.0}, 'message': 'Item added'}
127.0.0.1 - - [23/Oct/2025 14:22:03] "POST /items HTTP/1.1" 201 -
POST Response: {'item': {'name': 'Book', 'price': 200.0}, 'message': 'Item added'}
POST Response: {'item': {'name': 'Book', 'price': 200.0}, 'message': 'Item added'}
127.0.0.1 - - [23/Oct/2025 14:22:04] "POST /items HTTP/1.1" 201 -
POST Response: {'item': {'name': 'Book', 'price': 200.0}, 'message': 'Item added'}
POST Response: {'item': {'name': 'Book', 'price': 200.0}, 'message': 'Item added'}
127.0.0.1 - - [23/Oct/2025 14:22:07] "GET / HTTP/1.1" 200 -
```



### Task 3: Update Item

#### Instructions:

- Use AI to create a **PUT endpoint** to update an existing item based on its index or ID.

**Prompt:** Build a Flask API with two endpoints: • GET /items to return all items from an in-memory list • POST /items to add a new item to the list using JSON payload Return responses in JSON format.

#### Expected Output:

- **PUT /items/0** with payload {"name":"Notebook","price":250} → {"message": "Item updated", "item": {"name":"Notebook","price":250}}

HTML files > ai\_09 > 3\_ai.py > ...

```
1  from flask import Flask, request, jsonify
2
3  app = Flask(__name__)
4
5  # Sample items list
6  items = [
7      {"name": "Pen", "price": 20},
8      {"name": "Book", "price": 100}
9  ]
10
11 # PUT /items/<int:index> → Update item
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
17     data = request.get_json()
18     if not data or not isinstance(data, dict):
19         return jsonify({"error": "Invalid input"}), 400
20
21     items[index].update(data)
22     return jsonify({"message": "Item updated", "item": items[index]})
23
24 # GET /items/<int:index> → View item (for browser testing)
25 @app.route('/items/<int:index>', methods=['GET'])
26 def get_item(index):
27     if index < 0 or index >= len(items):
28         return jsonify({"error": "Item not found"}), 404
29
30     return jsonify(items[index])
31
32 if __name__ == '__main__':
33     app.run(debug=True)
```

```

PS C:\c codes> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/c codes/HTML files/ai_09/3_ai.py"
* Serving Flask app '3_ai'
* 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: 121-704-370
127.0.0.1 - - [23/Oct/2025 14:24:52] "GET / HTTP/1.1" 404 -
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: 121-704-370
127.0.0.1 - - [23/Oct/2025 14:24:52] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [23/Oct/2025 14:25:16] "GET /items HTTP/1.1" 404 -
127.0.0.1 - - [23/Oct/2025 14:26:07] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [23/Oct/2025 14:26:19] "GET /items/0 HTTP/1.1" 200 -

```

The screenshot shows a web browser window with the address bar set to `localhost:5000/items/0`. The page content displays a JSON object: `{ "name": "Pen", "price": 20 }`. The browser's developer tools are open, showing the response in the console.

## Task 4: Delete Item

### Instructions:

- Use AI to create a **DELETE endpoint** to remove an item by index or ID.

**Prompt :** Add a DELETE endpoint to a Flask API that removes an item from an in-memory list by index. If the index is invalid, return a 404 error. Respond with JSON.

### Expected Output:

- DELETE /items/0** → `{"message": "Item deleted", "item": {"name": "Notebook", "price": 250}}`
- GET /items** → `[]`

HTML files > ai\_09 > 4\_AI.py > ...

```
1  from flask import Flask, request, jsonify
2
3  app = Flask(__name__)
4
5  # Sample items list
6  items = [
7      {"name": "Notebook", "price": 250}
8  ]
9
10 # DELETE /items/<int:index> → Delete item by index
11 @app.route('/items/<int:index>', methods=['DELETE'])
12 def delete_item(index):
13     if index < 0 or index >= len(items):
14         return jsonify({"error": "Item not found"}), 404
15     removed_item = items.pop(index)
16     return jsonify({"message": "Item deleted", "item": removed_item})
17
18 # GET /items → View all items
19 @app.route('/items', methods=['GET'])
20 def get_items():
21     return jsonify(items)
22
23 if __name__ == '__main__':
24     app.run(debug=True)
25
```

```
PS C:\c codes> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/c codes/HTML files/ai_09/4_AI.py"
* Serving Flask app '4_AI'
* 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!
* Restarting with stat
* Debugger is active!
* Debugger is active!
* Debugger PIN: 121-704-370
127.0.0.1 - - [23/Oct/2025 14:17:05] "DELETE /items/0 HTTP/1.1" 200 -
127.0.0.1 - - [23/Oct/2025 14:19:07] "DELETE /items/0 HTTP/1.1" 404 -

```

```

PS C:\c codes> curl.exe -X DELETE http://localhost:5000/items/0
>>
{
  "item": {
    "name": "Notebook",
    "price": 250
  },
  "price": 250
},
},
},
"message": "Item deleted"
}
PS C:\c codes> curl.exe -X DELETE http://localhost:5000/items/0
{
  "error": "Item not found"
}
PS C:\c codes> 

```

## Task 5: Add Auto-Generated Documentation

### Instructions:

- Use AI to add inline comments and docstrings for all endpoints.
- Optionally, integrate Swagger / Flask-RESTX to auto-generate API documentation.

Starter Comment Example:

```

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

```

### Expected Output:

- Clear documentation for all endpoints, e.g., via /docs if using Swagger.
- Students should be able to see endpoint description, methods, and sample payloads.

### Prompt :

Add inline comments and docstrings to all endpoints in a Flask API that manages an in-memory list of items. Each endpoint should include a clear description, HTTP method, and expected input/output. Optionally, integrate Swagger or Flask-RESTX to auto-generate API documentation accessible via /docs.

HTML files > ai\_09 > 5\_ai.py > ...

```
1  from flask import Flask, request
2  from flask_restx import Api, Resource, fields
3
4  # Initialize Flask and Flask-RESTX
5  app = Flask(__name__)
6  api = Api(
7      app,
8      version='1.0',
9      title='Item Store API',
10     description='Manage items with CRUD operations',
11     doc='/docs' # Swagger UI available at /docs
12 )
13
14 # In-memory item list
15 items = [
16     {"name": "Pen", "price": 20},
17     {"name": "Book", "price": 100}
18 ]
19
20 # Define the item model for Swagger documentation
21 item_model = api.model('Item', {
22     'name': fields.String(required=True, description='Name of the item', example='Notebook'),
23     'price': fields.Integer(required=True, description='Price of the item', example=250)
24 })
25
26 @api.route('/items')
27 class ItemList(Resource):
28     def get(self):
```

HTML files > ai\_09 > 5\_ai.py > ...

```
26 @api.route('/items')
27 class ItemList(Resource):
28     def get(self):
29         """
30         GET /items
31         Returns a list of all items.
32         """
33         return items
34
35     @api.expect(item_model)
36     def post(self):
37         """
38         POST /items
39         Adds a new item to the list.
40         Payload: { "name": "Notebook", "price": 250 }
41         """
42         data = request.json
43         items.append(data)
44         return {"message": "Item added", "item": data}, 201
45
46 @api.route('/items/<int:index>')
47 @api.doc(params={'index': 'Index of the item in the list'})
48 class Item(Resource):
49     def get(self, index):
50         """
51         GET /items/<index>
52         Returns a single item by index.
53         """
```



HTML files > ai\_09 > 5\_ai.py > ItemList > get

```
48 class Item(Resource):
49     def get(self, index):
54         if index < 0 or index >= len(items):
55             api.abort(404, "Item not found")
56         return items[index]
57
58     @api.expect(item_model)
59     def put(self, index):
60         """
61         PUT /items/<index>
62         Updates an existing item by index.
63         Payload: { "name": "Notebook", "price": 250 }
64         """
65         if index < 0 or index >= len(items):
66             api.abort(404, "Item not found")
67         data = request.json
68         items[index].update(data)
69         return {"message": "Item updated", "item": items[index]}
70
71     def delete(self, index):
72         """
73         DELETE /items/<index>
74         Deletes an item by index.
75         """
76         if index < 0 or index >= len(items):
77             api.abort(404, "Item not found")
78         removed_item = items.pop(index)
79         return {"message": "Item deleted", "item": removed_item}
80
81 if __name__ == '__main__':
82     app.run(debug=True)
83
```

```
PS C:\c codes> & C:/Users/hp/AppData/Local/Programs/Python/Python313/python.exe "c:/c codes/HTML files/ai_09/5_ai.py"
* Serving Flask app '5_ai'
* 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: 121-704-370
127.0.0.1 - - [23/Oct/2025 15:10:20] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [23/Oct/2025 15:10:20] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [23/Oct/2025 15:10:36] "GET /doc HTTP/1.1" 404 -
127.0.0.1 - - [23/Oct/2025 15:10:39] "GET /doc HTTP/1.1" 404 -
127.0.0.1 - - [23/Oct/2025 15:11:05] "GET /docs HTTP/1.1" 200 -
127.0.0.1 - - [23/Oct/2025 15:11:05] "GET /swaggerui/swagger-ui-standalone-preset.js HTTP/1.1" 200 -
127.0.0.1 - - [23/Oct/2025 15:11:05] "GET /swaggerui/droid-sans.css HTTP/1.1" 200 -
127.0.0.1 - - [23/Oct/2025 15:11:05] "GET /swaggerui/swagger-ui.css HTTP/1.1" 200 -
127.0.0.1 - - [23/Oct/2025 15:11:05] "GET /swaggerui/swagger-ui-bundle.js HTTP/1.1" 200 -
127.0.0.1 - - [23/Oct/2025 15:11:06] "GET /swagger.json HTTP/1.1" 200 -
* Restarting with stat
* Debugger is active!
* Debugger PIN: 121-704-370
127.0.0.1 - - [23/Oct/2025 15:10:20] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [23/Oct/2025 15:10:20] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [23/Oct/2025 15:10:36] "GET /doc HTTP/1.1" 404 -
127.0.0.1 - - [23/Oct/2025 15:10:39] "GET /doc HTTP/1.1" 404 -
127.0.0.1 - - [23/Oct/2025 15:11:05] "GET /docs HTTP/1.1" 200 -
127.0.0.1 - - [23/Oct/2025 15:11:05] "GET /swaggerui/swagger-ui-standalone-preset.js HTTP/1.1" 200 -
```

Item Store API

[ Base URL: / ]  
/swagger.json

Manage items with CRUD operations

default

Default namespace

POST

/items

POST /items

GET

/items

GET /items

PUT

/items/{index}

PUT /items/<index>

DELETE

/items/{index}

DELETE /items/<index>

GET

/items/{index}

GET /items/<index>

Models

Activate Windows  
Go to Settings to activate Windows.