Hands-on Lab - CRUD operations with Python

Skills Network

Estimated Time Needed: 60 mins

In this lab, you will learn how to create a **Products's list** using a Flask server. Your application should allow you to add a product, retrieve the products, retrieve a specific product with its id, update a specific product with its id, and delete a product with its id. All these operations will be achieved through the REST API endpoints in your Flask server.

You will create an application with API endpoints to perform Create, Retrieve, Update, and Delete operations on the above data using a Flask server.

You will use cURL and POSTMAN to test the implemented endpoints.

Objectives:

After completing this lab you will be able to:

- Create API endpoints to perform Create, Retrieve, Update, and Delete operations on transient data with a Flask server.
- Create REST API endpoints, and use POSTMAN to test your REST APIs.

Set-up : Create application

1. Open a terminal window by using the menu in the editor: **Terminal > New Terminal**.
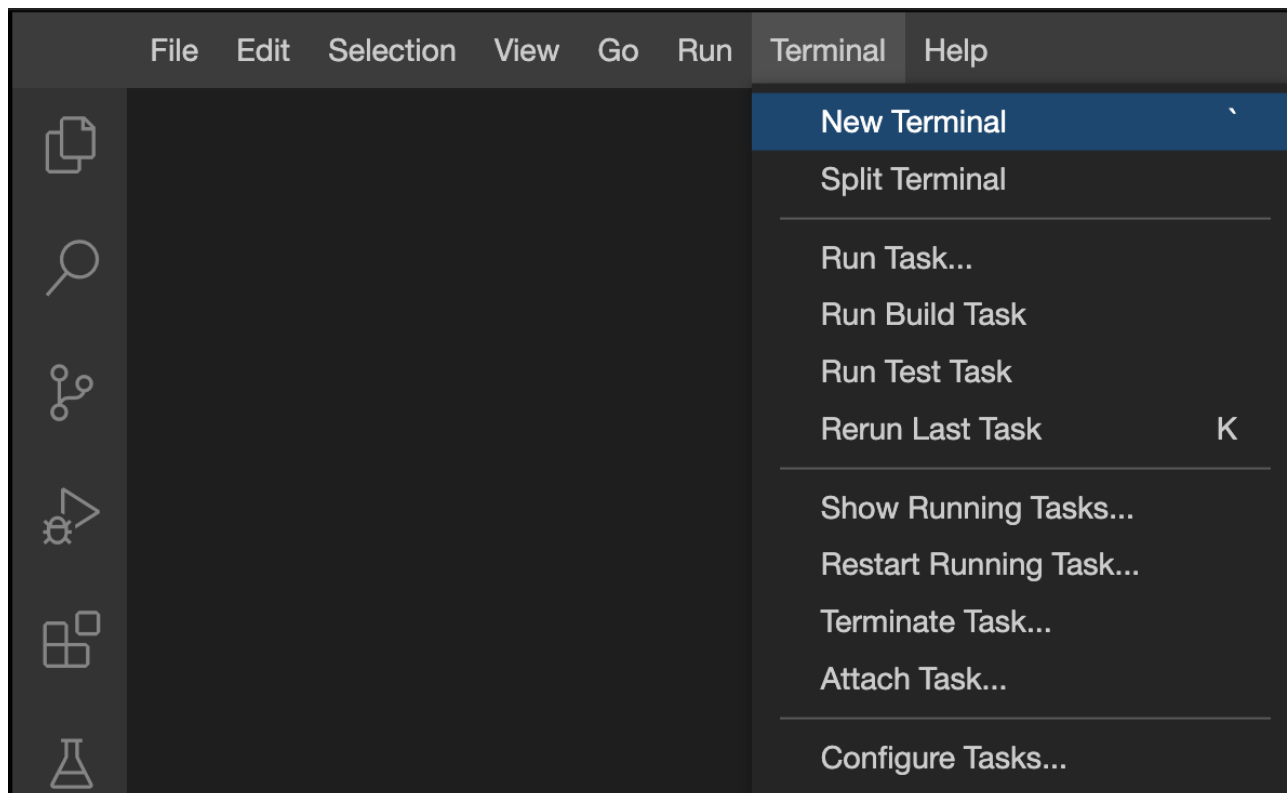
2. Change to your project folder, if you are not in the project folder already.

1. 1

1. `cd /home/project`

Copied!

3. Run the following command to clone the Git repository that contains the starter code needed for this lab, if it doesn't already exist.

1. 1

1. `[ ! -d 'jmgdo-microservices' ] && git clone https://github.com/ibm-developer-skills-network/jmgdo-microservices.git`

Copied!

5. Change to the directory **jmgdo-microservices/CRUD** to start working on the lab.

1. 1

1. `cd jmgdo-microservices/CRUD`

Copied!

6. List the contents of this directory to see the artifacts for this lab.

1. 1

1. `ls`

Copied!

7. Run the following command on the terminal to install the packages that are required.

1. 1

1. `python3 -m pip install flask flask_cors`

Copied!

Exercise 1: Understand the server application

1. In the Files Explorer open the **jmgdo-microservices/CRUD** folder and view **products.py**.

2. You first need to import the packages required to create REST APIs with Flask.

1. 1

```
2. 2
```

```
1. from flask import Flask, jsonify, request
2. import json
```

Copied!

3. You can then create the Flask application, which will service all the REST APIs for adding, retrieving, updating, and deleting products.

```
1. 1
```

```
1. app = Flask("Product Server")
```

Copied!

4. The code has precreated products added to the list. These are defined in the following code.

```
1. 1
2. 2
3. 3
4. 4
```

```
1. products = [
2.     {'id': 143, 'name': 'Notebook', 'price': 5.49},
3.     {'id': 144, 'name': 'Black Marker', 'price': 1.99}
4. ]
```

Copied!

5. Now that the server is defined, you will create the REST API endpoints and define the routes or paths, one for each of the following operations.

- Retrieve all the products - `GET` Request Method
- Retrieve a product by its id - `GET` Request Method
- Add a product - `POST` Request Method
- Update a product by its id - `PUT` Request Method
- Delete a product by its id - `DELETE` Request Method

Add the following code to products.py in the space provided.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
```

```
1.  # Example request – http://localhost:5000/products
2.  @app.route('/products', methods=['GET'])
3.  def get_products():
4.      return jsonify(products)
5.
6.  # Example request – http://localhost:5000/products/144 – with method GET
7.  @app.route('/products/<id>', methods=['GET'])
8.  def get_product(id):
9.      id = int(id)
10.     product = [x for x in products if x["id"] == id][0]
11.     return jsonify(product)
12.
13. # Example request – http://localhost:5000/products – with method POST
14. @app.route('/products', methods=['POST'])
15. def add_product():
16.     products.append(request.get_json())
17.     return '', 201
18.
19. # Example request – http://localhost:5000/products/144 – with method PUT
20. @app.route('/products/<id>', methods=['PUT'])
21. def update_product(id):
22.     id = int(id)
23.     updated_product = json.loads(request.data)
24.     product = [x for x in products if x["id"] == id][0]
25.     for key, value in updated_product.items():
26.         product[key] = value
27.     return '', 204
28.
29. # Example request – http://localhost:5000/products/144 – with method DELETE
30. @app.route('/products/<id>', methods=['DELETE'])
31. def remove_product(id):
32.     id = int(id)
33.     product = [x for x in products if x["id"] == id][0]
34.     products.remove(product)
35.     return '', 204
```

Copied!

Run and test the server with cURL

1. In the terminal, run the python server using the following command.

```
1. 1
```

```
1. python3 products.py
```

Copied! Executed!

The server starts running and listening at port 5000.

```
 * Serving Flask app 'Product Server' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 145-923-897
```

2. Open another `Terminal` by clicking the Terminal menu and selecting **New Terminal**.

3. In the new terminal, run the following command to access the **http://localhost:5000/products** API endpoint. `curl` command stands for `Client URL` and is used as command line interfacing with the server serving REST API endpoints. It is, by default, a `GET` request.

1. 1

1. `curl http://localhost:5000/products`

[Copied!] [Executed!]

This returns a JSON with the products that have been preloaded.

```
theia@theia-lavanyas:/home/project$ curl http://localhost:5000/products
[
  {
    "id": 143,
    "name": "Notebook",
    "price": 5.49
  },
  {
    "id": 144,
    "name": "Black Marker",
    "price": 1.99
  }
]
```

4. In the terminal, run the following command to add a product to the list. This will be a POST request to which you will pass the product parameter as a JSON.

1. 1
2. 2
3. 3

1. `curl -X POST -H "Content-Type: application/json" \`
2. `   -d '{"id": 145, "name": "Pen", "price": 2.5}' \`
3. `   http://localhost:5000/products`

[Copied!]

This command will not return any output. It will add the product to the list of products.

5. Verify if the product is added by running the following command.

1. 1

1. `curl http://localhost:5000/products/145`

[Copied!]

This should return the details of the product you just added with a POST request.

```
theia@theia-lavanyas:/home/project$ curl http://localhost:5000/products/145
{
  "id": 145,
  "name": "Pen",
  "price": 2.5
}
```

Using POSTMAN to test the REST API endpoints

While the `GET` endpoints are easy to test with `curl` using a command line interface, the `POST`, `PUT` and `DELETE` commands can be cumbersome. To circumvent this problem, you can use Postman, which is a software that is available as a service.

Postman will need remote access to the server as it is an external entity. To get the URL click the following button.

[Launch Application]

Copy the URL into a notepad or other text editors. Go to https://www.postman.com/ and sign up before you start using POSTMAN.

▶ Click here for instructions to sign up for Postman services.

1. Click **Create New** to start creating a request to the REST API endpoint.

Get started with Postman

Start with something new	Import an existing file
Create a new request, collection, or API in a workspace	Import any API schema file from your local drive or Github
Create New →	Import file →

2. Choose **HTTP Request** from the options that you are presented with.

Create New

Building Blocks

HTTP Request	**WebSocket Request** BETA	**gRPC Request**
Create a basic HTTP request	Test and debug your WebSocket connections	Test and debug y request

Collection	**Environment**	**Workspace**
Save your requests in a collection for reuse and sharing	Save values you frequently use in an environment	Create a workspa independently or

Advanced

API Documentation	**Mock Server**	**Monitor**
Create and publish beautiful documentation for your APIs	Create a mock server for your in-development APIs	Schedule automa check performanc

API	**Flows** BETA
Manage all aspects of API design, development, and testing	Create API workflows by connecting series of requests through a drag-and-drop UI.

Not sure where to start? Explore featured APIs, collections, and workspaces published by the Postman community.

3. Paste the URL that you have copied into the address bar. Change the request type to `POST`. To send input as JSON, choose `raw->body->JSON`.

| POST ⌄ | https://lavanyas-5000.theia-0-labs-prod-misc-tools-us-east-0.proxy.cognitiveclass.ai/products |

Params   Authorization   Headers (7)   **Body** ●   Pre-request Script   Tests   Settings

○ none   ○ form-data   ○ x-www-form-urlencoded   ● raw   ○ binary   ○ GraphQL   **JSON** ⌄

```
1  {
2      "id":146,
3      "name":"Laptop Bag",
4      "price":45.00
5  }
```

4. Enter the following JSON object and click the Send button. This will add the product to your previous list of products.

```
1. 1
2. 2
3. 3
4. 4
5. 5
```

```
1. {
2.     "id":146,
3.     "name":"Laptop Bag",
4.     "price":45.00
5. }
```

Copied!

4. Verify the list of products to ensure that the request has gone through and the product is added. Change the request type to GET and remove the JSON object from the request body. Click Send and observe the output in the response window.

Params   Authorization   Headers (6)   **Body**   Pre-request Script   Tests   Settings

⚪ none   ⚪ form-data   ⚪ x-www-form-urlencoded   🔴 raw   ⚪ binary   ⚪ GraphQL   **JSON** $\vee$

```
1
```

Body   Cookies (1)   Headers (10)   Test Results   🌐 Status: **200 OK**

Pretty   Raw   Preview   Visualize   JSON $\vee$ ⇄

```
 7    {
 8        "id": 144,
 9        "name": "Black Marker",
10        "price": 1.99
11    },
12    {
13        "id": 145,
14        "name": "Pen",
15        "price": 2.5
16    },
17    {
18        "id": 146,
19        "name": "Laptop Bag",
20        "price": 45.0
21    }
22 ]
```

5. Test the PUT endpoint to update product details by changing the price of the item with id 146 to 42.00. Set the request type to PUT and add the product id to the end of the URL and add the value to be changed as a JSON body and click Send.

```
1. 1
2. 2
3. 3
```

```
1. {
2.     "price":42.00
3. }
```

Copied!

| PUT | ∨ | https://lavanyas-5000.theia-0-labs-prod-misc-tools-us-east-0.proxy.cognitiveclass.ai/products/146 |

Params   Authorization   Headers (8)   Body ●   Pre-request Script   Tests   Settings

○ none   ○ form-data   ○ x-www-form-urlencoded   ● raw   ○ binary   ○ GraphQL   **JSON** ∨

```
1   {
2           "price": 42.00
3   }
```

6. Verify the product with id 146 to ensure that the request has gone through and the product is updated. Change the request type to GET and remove the JSON object from the request body. Click Send and observe the output in the response window.

| GET | ∨ | https://lavanyas-5000.theia-0-labs-prod-misc-tools-us-east-0.proxy.cognitiveclass.ai/products/146 |

Params   Authorization   Headers (6)   Body   Pre-request Script   Tests   Settings

○ none   ○ form-data   ○ x-www-form-urlencoded   ● raw   ○ binary   ○ GraphQL   **JSON** ∨

```
1
```

Body   Cookies (1)   Headers (10)   Test Results   🌐 Status: 200 OK

Pretty   Raw   Preview   Visualize   JSON ∨   ⇄

```
1   {
2       "id": 146,
3       "name": "Laptop Bag",
4       "price": 42.0
5   }
```

Practice labs

1. Update product details of product with id 144 by changing its price to 2.50.

▶ Click here for a hint
▶ Click here for the solution

2. Add the following product using POST method.

```
1. 1
2. 2
3. 3
4. 4
5. 5
```

```
1. {
2.     "id":142,
3.     "name":"Eraser",
4.     "price":1.50
5. }
```

Copied!

▶ Click here for a hint

3. Delete the product with id 142.

▶ Click here for a hint
▶ Click here for the solution

Congratulations! You have completed the lab for CRUD operations with Python.

Summary:

In this lab, you have performed CRUD Operations like GET, POST, PUT, and DELETE on a Python server App and tested the above methods using POSTMAN.

Author(s)

Lavanya T S

Changelog

Date	Version	Changed by	Change Description
01-11-2022	1.0	Lavanya T S	Initial version created
2022-11-25	1.1	Steve Hord	QA pass edits