Full Stack Development with AI
Lab 12.2 – Creating RESTful API Endpoints with Flask and Connexion

Lab Overview

In this lab, you will learn how to create RESTful API endpoints using Flask and Connexion. The endpoints are designed to support a client-side rendering web application, and are similar to those that you have created in Lab 12.1.

You will be using the same synthetic product data as Lab 10.2.

Exercise 1 – Creating a new Flask and Connexion Application

1. Use the file explorer or terminal on your computer to create a new folder `app`.
2. Start Visual Studio Code (VS Code) and open the newly created `app` folder.
3. Create a new subfolder `db` within the `app` folder.
4. Start a terminal in VS Code.
5. Run the following commands to install the required Python libraries using `pip`:

```
python -m pip install flask
python -m pip install connexion[flask,uvicorn,swagger-ui]
```

Exercise 2 – Cloning the SQLite Database

For this lab, we will be reusing the "products.db" SQLite database that you have created in Lab 10.2.

Copy the file "products.db" into the `app\db` folder of your new Flask and Connexion application.

Exercise 3 – Creating New Endpoints

In this exercise, you will be creating a series of endpoints using the PUT, GET, POST and DELETE HTTP methods to implement the create, retrieve, update and delete use cases for product records.

1. Create a new file "app.py" in the root of the web application or the `app` folder itself.

2. Paste the following code fragment representing the basic structure of a Flask and Connexion application into "app.py"

```
from connexion import FlaskApp

app = FlaskApp(__name__)
app.add_api("openapi.yaml")

if __name__ == "__main__":
    app.run(port=8000)
```

3. Create a new file "openapi.yaml" in the root of the web application or the `app` folder itself.

4. Paste the following configuration into "openapi.yaml"

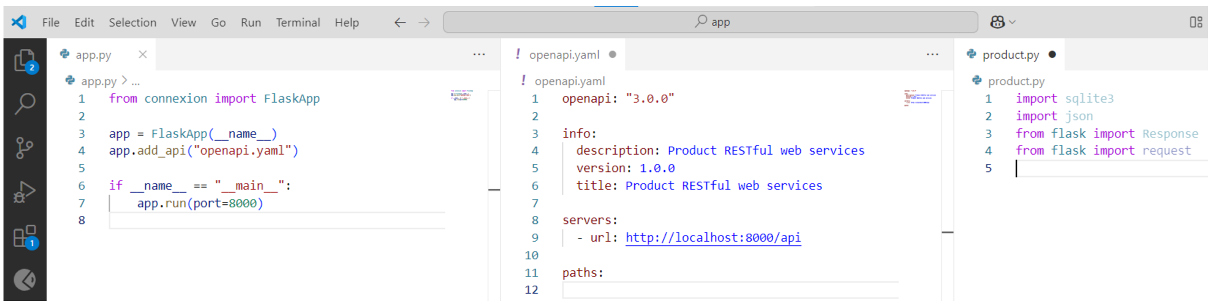
```
openapi: "3.0.0"

info:
  description: Product RESTful web services
  version: 1.0.0
  title: Product RESTful web services

servers:
  - url: http://localhost:8000/api

paths:
```

5. Create a new file "product.py" in the root of the web application or the `app` folder itself. This is the Python source file containing the endpoint hander functions.

6. Paste the following code fragment into "product.py"

```
import sqlite3
import json
from flask import Response
from flask import request
```

7. Your VS Code should resemble the following figure:



```python
from connexion import FlaskApp

app = FlaskApp(__name__)
app.add_api("openapi.yaml")

if __name__ == "__main__":
    app.run(port=8000)
```

```yaml
openapi: "3.0.0"

info:
  description: Product RESTful web services
  version: 1.0.0
  title: Product RESTful web services

servers:
  - url: http://localhost:8000/api

paths:
```

```python
import sqlite3
import json
from flask import Response
from flask import request
```

8. In "product.py", define a Python handler function `retrieveAllProducts` that performs the following two tasks:

    a. Retrieves all the product records from the `products` table using a `SELECT` query.
    b. Return the product records as an array of objects in JSON format together with the HTTP 200 status code.

9. In "openapi.yaml", define a `GET` endpoint to the path `/product` that maps to the Python handler function `retrieveAllProducts` defined in (8).

    You can use the Swagger Editor to help you to validate the correctness of the yaml configuration – https://editor-next.swagger.io/

10. In "product.py", define a Python handler function `retrieveProductById` that performs the following four tasks:

    a. Accepts a path parameter for the product id of the product to be retrieved.
    b. Retrieves the product record from the products table using a `SELECT` query with an appropriate `WHERE` clause.
    c. Return the product record as an object in JSON format together with the HTTP 200 status code.
    d. If the required product does not exist, return an appropriate error message together with the HTTP 404 status code.

11. In "openapi.yaml", define a `GET` endpoint to the path `/product/{product_id}` that maps to the Python handler function `retrieveProductById` defined in (10).

12. In "product.py", define a Python handler function `createNewProduct` that performs the following four tasks:

    a. Retrieves the data for a new product record from the body of the request in JSON object format.
    b. Insert the data for the new product record into the products table using an `INSERT` statement.
    c. Return an information message together with the HTTP 200 status code.
    d. If an error occurs, return an appropriate error message together with the HTTP 400 (Bad Request) or 409 (Conflict) status code depending on the nature of the error.

13. In “openapi.yaml”, define a `PUT` endpoint to the path `/product` that maps to the Python handler function `createNewProduct` defined in (12).

14. In “product.py”, define a Python handler function `updateProduct` that performs the following five tasks:

    a. Accepts a path parameter for the product id of the product to be updated.
    b. Retrieves the data for an existing product record from the body of the request in JSON object format.
    c. Update the data for the existing product record into the `products` table using an `UPDATE` statement with an appropriate `WHERE` clause.
    d. Return an information message together with the HTTP 200 status code.
    e. If an error occurs, return an appropriate error message together with the HTTP 400 (Bad Request) or 404 (Not Found) status code depending on the nature of the error.

15. In “openapi.yaml”, define a `POST` endpoint to the path `/product/{product_id}` that maps to the Python handler function `updateProduct` defined in (14).

16. In “product.py”, define a Python handler function `deleteProduct` that performs the following four tasks:

    a. Accepts a path parameter for the product id of the product to be deleted.
    b. Delete the data for the existing product record into the `products` table using a `DELETE` statement with an appropriate `WHERE` clause.
    c. Return an information message together with the HTTP 200 status code.
    d. If an error occurs, return an appropriate error message together with the HTTP 404 status code.

17. In “openapi.yaml”, define a `DELETE` endpoint to the path `/product/{product_id}` that maps to the Python handler function `deleteProduct` defined in (16).

Exercise 4 – Running and Testing the Endpoints

Follow the instructions below to run the web application:

1. Start a new terminal in VS Code.

2. Run the following command to run the web application:

```
python app.py
```

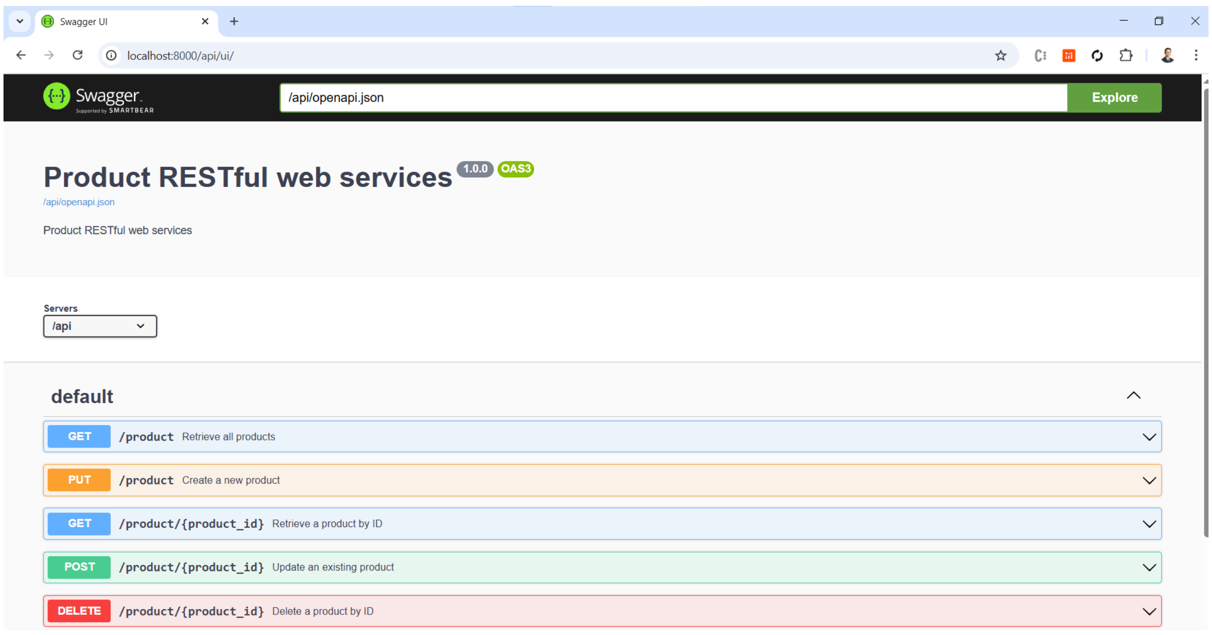
3. Use Postman to test each of the following endpoints in a similar manner as what you have done in Lab 12.1:

   - `GET` endpoint to retrieve all product records
   - `GET` endpoint to retrieve one product record
   - `PUT` endpoint to create a new product record
   - `POST` endpoint to update an existing product record
   - `DELETE` endpoint to update an existing product record

   Remember to use the `GET` endpoint to retrieve all products and verify that the product records have been manipulated correctly.

4. Repeat the testing of the endpoints by introducing some error conditions such as retrieving a product record using a product id that does not exist.

5. Last but not least, navigate to http://localhost:5000/api/ui to view the Swagger UI dashboard. This dashboard shows all the endpoints in your application together with comprehensive documentation.



Product RESTful web services 1.0.0 OAS3

/api/openapi.json

Product RESTful web services

Servers: /api

default

GET /product Retrieve all products

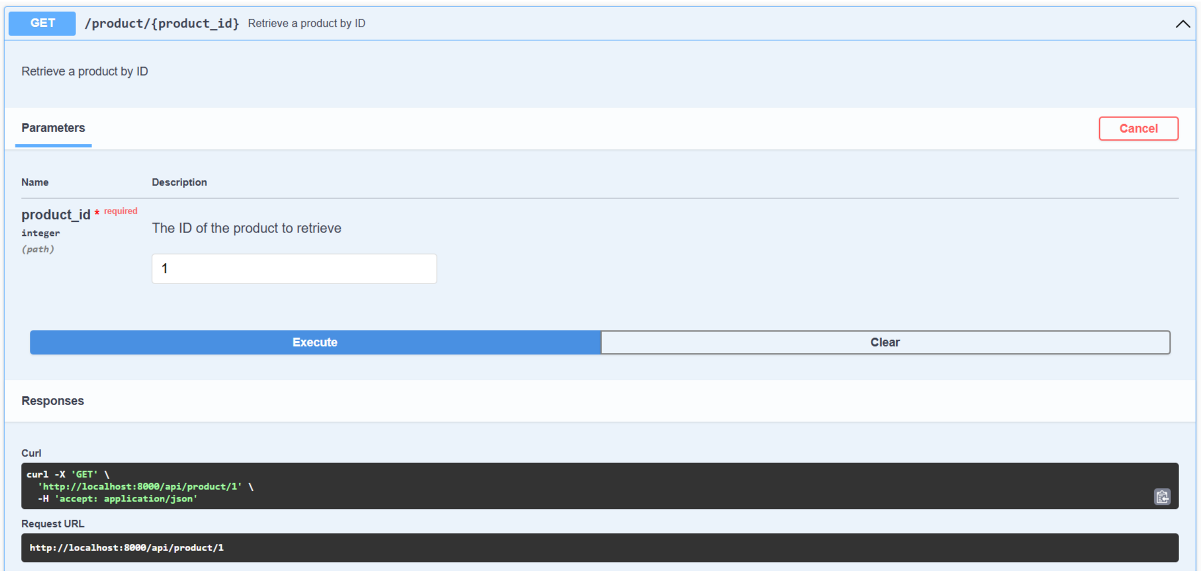
PUT /product Create a new product

GET /product/{product_id} Retrieve a product by ID

POST /product/{product_id} Update an existing product

DELETE /product/{product_id} Delete a product by ID

In fact, you can even test the endpoints within the dashboard without the use of Postman. Go ahead and give it a try.



Retrieve a product by ID

product_id * required
integer
(path)

The ID of the product to retrieve

1

Curl

```
curl -X 'GET' \
  'http://localhost:8000/api/product/1' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:8000/api/product/1
```

-- End of Lab --