Simple Flask API Application Documentation

Overview

This documentation provides an overview of a simple Flask API application that supports CRUD operations (GET, POST, PATCH, DELETE) and a CLI-based client application that interacts with the API.

`MAIN.PY` - FLASK API APPLICATION

THIS APPLICATION HANDLES THE CRUD OPERATIONS FOR ITEMS.

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

app = Flask(__name__)

# Mock database
items = {}

# GET method to fetch items
@app.route('/items', methods=['GET'])
def get_items():
    return jsonify(items), 200

# POST method to add an item
@app.route('/items', methods=['POST'])
def add_item():
    item_id = request.json.get('id')
    item_name = request.json.get('name')
    if not item_id or not item_name:
        return jsonify({"message": "Invalid data"}), 400
    items[item_id] = item_name
    return jsonify({"message": "Item added"}), 201

# PATCH method to update an item
@app.route('/items/<item_id>', methods=['PATCH'])
def update_item(item_id):
    if item_id not in items:
        return jsonify({"message": "Item not found"}), 404
    item_name = request.json.get('name')
    if not item_name:
        return jsonify({"message": "Invalid data"}), 400
    items[item_id] = item_name
    return jsonify({"message": "Item updated"}), 200

# DELETE method to delete an item
@app.route('/items/<item_id>', methods=['DELETE'])
def delete_item(item_id):
    if item_id not in items:
        return jsonify({"message": "Item not found"}), 404
```

```
    del items[item_id]
    return jsonify({"message": "Item deleted"}), 200

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

ENDPOINTS

1. **GET /items**
    - Description: Fetch all items.
    - Response: JSON object containing all items.
    - Status Code: 200 OK
1. **POST /items**
    - Description: Add a new item.
    - Request Body: JSON object containing `id` and `name`.
    - Response: JSON object with a success message.
    - Status Code: 201 Created
1. **PATCH /items/<item_id>**
    - Description: Update an existing item.
    - Request Body: JSON object containing `name`.
    - Response: JSON object with a success message.
    - Status Code: 200 OK
1. **DELETE /items/<item_id>**
    - Description: Delete an item.
    - Response: JSON object with a success message.
    - Status Code: 200 OK

`CLIENT.PY` - CLI-BASED CLIENT APPLICATION

THIS APPLICATION INTERACTS WITH THE FLASK API APPLICATION (`APP1.PY`) THROUGH HTTP REQUESTS.

```
import requests
import json

BASE_URL = 'http://localhost:5000'

def get_items():
    response = requests.get(f'{BASE_URL}/items')
    print(response.json())

def add_item(item_id, item_name):
    response = requests.post(f'{BASE_URL}/items', json={"id": item_id, "name": item_na
```

```
    print(response.json())

def update_item(item_id, item_name):
    response = requests.patch(f'{BASE_URL}/items/{item_id}', json={"name": item_name})
    print(response.json())

def delete_item(item_id):
    response = requests.delete(f'{BASE_URL}/items/{item_id}')
    print(response.json())

def main():
    while True:
        print("\nOptions:")
        print("1. Get Items")
        print("2. Add Item")
        print("3. Update Item")
        print("4. Delete Item")
        print("5. Exit")

        choice = input("Enter choice: ")
        if choice == '1':
            get_items()
        elif choice == '2':
            item_id = input("Enter item ID: ")
            item_name = input("Enter item name: ")
            add_item(item_id, item_name)
        elif choice == '3':
            item_id = input("Enter item ID: ")
            item_name = input("Enter new item name: ")
            update_item(item_id, item_name)
        elif choice == '4':
            item_id = input("Enter item ID: ")
            delete_item(item_id)
        elif choice == '5':
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == '__main__':
    main()
```

TESTING WITH POSTMAN

TO TEST `APP.PY` USING POSTMAN:

1. **GET Items**:
    - Method: GET
    - URL: `http://localhost:5000/items`
1. **POST Item**:

- Method: POST
- URL: `http://localhost:5000/items`
- Body: (raw JSON)

```
{
"id": "1",
"name": "Item 1"
}
```

1. **PATCH Item**:
    - Method: PATCH
    - URL: `http://localhost:5000/items/1`
    - Body: (raw JSON)

```
{
"name": "Updated Item 1"
}
```

1. **DELETE Item**:
    - Method: DELETE
    - URL: `http://localhost:5000/items/1`

This setup allows you to test the API using Postman and interact with the API using a simple CLI-based client without exposing the routes in integrate`.py`.