Flask_14_Integration

STEP 1: CREATE A SIMPLE BACKEND FLASK API APPLICATION

```
simple_backend_api/
│
├── app.py
├── books.json
└── requirements.txt
```

Step 1.1: Create the `books.json` File

```
[
    {
        "id": 1,
        "title": "title_1",
        "author": "author_1"
    },
    {
        "id": 2,
        "title": "title_2",
        "author": "author_2"
    }
]
```

Step 1.2: Create the Flask Application (main`.py`)

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

app = Flask(__name__)


# Load books from a JSON file
def load_books():
    with open('books.json', 'r') as f:
        return json.load(f)


# Save books to a JSON file
def save_books(books):
    with open('books.json', 'w') as f:
        json.dump(books, f, indent=4)


# Get all books
@app.route('/books', methods=['GET'])
def get_books():
    books = load_books()
```

```python
    return jsonify(books)


# Get a single book by ID
@app.route('/books/<int:book_id>', methods=['GET'])
def get_book(book_id):
    books = load_books()
    book = next((book for book in books if book['id'] == book_id), None)
    if book is None:
        abort(404)
    return jsonify(book)


# Add a new book
@app.route('/books', methods=['POST'])
def add_book():
    new_book = request.json
    books = load_books()
    new_book['id'] = books[-1]['id'] + 1 if books else 1
    books.append(new_book)
    save_books(books)
    return jsonify(new_book), 201


# Update an existing book
@app.route('/books/<int:book_id>', methods=['PATCH'])
def update_book(book_id):
    books = load_books()
    book = next((book for book in books if book['id'] == book_id), None)
    if book is None:
        abort(404)

    data = request.json
    book.update(data)
    save_books(books)
    return jsonify(book)


# Delete a book
@app.route('/books/<int:book_id>', methods=['DELETE'])
def delete_book(book_id):
    books = load_books()
    book = next((book for book in books if book['id'] == book_id), None)
    if book is None:
        abort(404)

    books.remove(book)
    save_books(books)
    return '', 204


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

STEP 2: CREATE ANOTHER FLASK APPLICATION TO INTEGRATE WITH THE FIRST ONE

Directory Structure

```
consumer_api/
│
├── consumer_app.py
└── requirements.txt
```

Step 2.1: Create the Second Flask Application (`consumer_app.py`)

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

app = Flask(__name__)

# Base URL of the Book Management API
BOOK_API_URL = 'http://127.0.0.1:5000/books'

# Route to get all books
@app.route('/consumer/books', methods=['GET'])
def get_books():
    response = requests.get(BOOK_API_URL)
    return jsonify(response.json()), response.status_code

# Route to get a single book by ID
@app.route('/consumer/books/<int:book_id>', methods=['GET'])
def get_book(book_id):
    response = requests.get(f"{BOOK_API_URL}/{book_id}")
    return jsonify(response.json()), response.status_code

# Route to add a new book
@app.route('/consumer/books', methods=['POST'])
def add_book():
    new_book = request.json
    response = requests.post(BOOK_API_URL, json=new_book)
    return jsonify(response.json()), response.status_code

# Route to update an existing book
@app.route('/consumer/books/<int:book_id>', methods=['PATCH'])
def update_book(book_id):
    update_data = request.json
    response = requests.patch(f"{BOOK_API_URL}/{book_id}", json=update_data)
    return jsonify(response.json()), response.status_code

# Route to delete a book
@app.route('/consumer/books/<int:book_id>', methods=['DELETE'])
def delete_book(book_id):
    response = requests.delete(f"{BOOK_API_URL}/{book_id}")
    return '', response.status_code

if __name__ == '__main__':
    app.run(debug=True, port=5001)
```

Get all records

Get single records

TESTING THE CONSUMER APPLICATION

YOU CAN USE TOOLS LIKE POSTMAN OR `CURL` TO TEST THE CONSUMER APPLICATION.

- **GET** all books: `GET /consumer/books`
- **GET** a single book: `GET /consumer/books/<id>`

POST a new book: `POST /consumer/books`

```
{
"title": "title_3",
"id":3,
"author": "author_3"
}
```

PATCH an existing book: `PATCH /consumer/books/<id>`

http://127.0.0.1:5001/consumer/books/3

```
{
"title": "updated_title_3"
}
```

DELETE a book: `DELETE /consumer/books/<id>`

http://127.0.0.1:5001/consumer/books/3

This setup creates a secondary Flask application that interacts with your original Simple Backend API. It demonstrates how one service can consume the API of another service, allowing you to build more complex systems by integrating multiple services.