flask_14_Integration_2

Certainly! Let's break down the `app.py` and `consumer_app.py` files in detail to understand how they work.

1. `APP.PY` (SIMPLE BACKEND API APPLICATION)

THIS FILE SETS UP A FLASK APPLICATION THAT PERFORMS BASIC CRUD OPERATIONS ON A COLLECTION OF BOOKS STORED IN A JSON FILE.

Importing Libraries

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

- `Flask`: The Flask class is used to create an instance of a web application.
- `request`: The request object holds data related to the incoming HTTP request.
- `jsonify`: Converts Python dictionaries to JSON format.
- `abort`: Used to send HTTP error codes.
- `json`: Used to handle JSON file operations.

Setting Up the Flask App

```
app = Flask(__name__)
```

- `app`: The Flask application instance.

Helper Functions to Load and Save Books

```
def load_books():
    with open('books.json', 'r') as f:
        return json.load(f)
def save_books(books):
    with open('books.json', 'w') as f:
        json.dump(books, f, indent=4)
```

- `load_books()`: Reads the `books.json` file and returns the list of books.
- `save_books(books)`: Saves the list of books to the `books.json` file.

Routes for CRUD Operations

Get All Books

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

- `@app.route('/books', methods=['GET'])`: Defines a route for the GET method at `/books`.
- `get_books()`: Loads the books and returns them in JSON format.

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)
```

- `@app.route('/books/<int:book_id>', methods=['GET'])`: Defines a route for the GET method at `/books/<book_id>`.
- `get_book(book_id)`: Finds a book by ID and returns it in JSON format. If not found, it aborts with a 404 error.

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
```

- `@app.route('/books', methods=['POST'])`: Defines a route for the POST method at `/books`.
- `add_book()`: Adds a new book to the list, assigns a unique ID, saves it, and returns the new book with a 201 status code.

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)
```

- `@app.route('/books/<int:book_id>', methods=['PATCH'])`: Defines a route for the PATCH method at `/books/<book_id>`.
- `update_book(book_id)`: Updates the specified book with the provided data, saves it, and returns the updated book. If the book is not found, it aborts with a 404 error.

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
```

- `@app.route('/books/<int:book_id>', methods=['DELETE'])`: Defines a route for the DELETE method at `/books/<book_id>`.
- `delete_book(book_id)`: Deletes the specified book and returns a 204 status code. If the book is not found, it aborts with a 404 error.

Running the Application

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

- This block checks if the script is executed directly and runs the Flask application in debug mode on port 5000.

2. `CONSUMER_APP.PY` (CONSUMER API APPLICATION)

THIS FILE SETS UP A FLASK APPLICATION THAT CONSUMES THE BOOK MANAGEMENT API.

Importing Libraries

```
from flask import Flask, request, jsonifyimport requests
```

- `Flask`, `request`, `jsonify`: Same as in `app.py`.
- `requests`: A library for making HTTP requests.

Setting Up the Flask App

```
app = Flask(__name__)
```

- `app`: The Flask application instance.

Base URL of the Book Management API

```
BOOK_API_URL = 'http://127.0.0.1:5000/books'
```

- `BOOK_API_URL`: The base URL of the Book Management API.

Routes for Consuming the Book Management API

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
```

- `@app.route('/consumer/books', methods=['GET'])`: Defines a route for the GET method at `/consumer/books`.
- `get_books()`: Makes a GET request to the Book Management API and returns the response.

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
```

- `@app.route('/consumer/books/<int:book_id>', methods=['GET'])`: Defines a route for the GET method at `/consumer/books/<book_id>`.
- `get_book(book_id)`: Makes a GET request to the Book Management API for a specific book ID and returns the response.

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
```

- `@app.route('/consumer/books', methods=['POST'])`: Defines a route for the POST method at `/consumer/books`.
- `add_book()`: Makes a POST request to the Book Management API to add a new book and returns the response.

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
```

- `@app.route('/consumer/books/<int:book_id>', methods=['PATCH'])`: Defines a route for the PATCH method at `/consumer/books/<book_id>`.
- `update_book(book_id)`: Makes a PATCH request to the Book Management API to update a specific book and returns the response.

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
```

- `@app.route('/consumer/books/<int:book_id>', methods=['DELETE'])`: Defines a route for the DELETE method at `/consumer/books/<book_id>`.
- `delete_book(book_id)`: Makes a DELETE request to the Book Management API to delete a specific book and returns the response.

Running the Application

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

- This block checks if the script is executed directly and runs the Flask application in debug mode on port 5001.

SUMMARY

- `app.py` creates a simple CRUD API for managing books.
- `consumer_app.py` creates another Flask application that consumes the API provided by `app.py`, demonstrating how to integrate two Flask applications.