

Flask API trong Python

1.Cấu trúc HTTP Request và HTTP Response

1.1 HTTP là gì ?

HTTP (Hypertext Transfer Protocol) là giao thức được thiết kế theo kiểu client – server, giao tiếp giữa client và server dựa vào một cặp request – response, client đưa ra các request và server trả lời các request này.

1.2 HTTP Request

- HTTP Request hay còn gọi là lời yêu cầu từ phía client đến serverserver
- Để bắt đầu Request phía client khởi tạo một HTTP session bằng cách mở một kết nối TCP(để gửi dữ liệu giữa các thiết bị trong một mạng máy tính) đến HTTP server sau đó gửi request đến server này. Request có thể được tạo bằng nhiều cách, trực tiếp khi người dùng nhấp vào một liên kết trên trình duyệt hoặc gián tiếp, ví dụ như một video được đính kèm trong một website và việc request đến website này sẽ dẫn đến một request tới video ấy.
- Bắt đầu của HTTP Request sẽ là dòng Request-Line bao gồm 3 thông tin đó là:
 1. Method: là phương thức mà HTTP Request này sử dụng, thường là GET, POST, ngoài ra còn một số phương thức khác như HEAD, PUT, DELETE, OPTION, CONNECT. Trong ví dụ trên là GET
 2. URI: là địa chỉ định danh của tài nguyên. Trong trường hợp này URI là / - tức request cho tài nguyên gốc, nếu request không yêu cầu một tài nguyên cụ thể, URI có thể là dấu *.
 3. HTTP version: là phiên bản HTTP đang sử dụng, ở đây là HTTP 1.1.

Request Header	Value
(Request-Line)	GET / HTTP/1.1
Host	www.stdio.vn
User-Agent	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language	vi-VN,vi;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding	gzip, deflate
Cookie	_ga=GA1.2.529740874.1426574792; PHPSESSID=0e61i2eb0mff46hejl7r0kfms2; _gat=1; ci_session=a%3...
Connection	keep-alive

Tiếp theo là các trường request-header, cho phép client gửi thêm các thông tin bổ sung về thông điệp HTTP request và về chính client. Một số trường thông dụng như:

- Accept: loại nội dung có thể nhận được từ thông điệp response. Ví dụ: text/plain, text/html...
- Accept-Encoding: các kiểu nén được chấp nhận. Ví dụ: gzip, deflate, xz, exi...
- Connection: tùy chọn điều khiển cho kết nối hiện thời. Ví dụ: keep-alive, Upgrade...
- Cookie: thông tin HTTP Cookie từ server.
- User-Agent: thông tin về user agent của người dùng.

Để hiểu hơn về HTTP request chúng ta hãy tìm hiểu thêm về GET và POST đây là 2 phương thức chúng ta hay sử dụng trong HTTP Request

1.2.1 Phương thức GET

Phương thức GET là yêu cầu từ khách hàng (client) đến máy chủ (server). Khi một yêu cầu GET được gửi, máy chủ sẽ trả về dữ liệu mà không có thay đổi nào về trạng thái của tài nguyên trên máy chủ.

Với GET, câu truy vấn sẽ được đính kèm vào đường dẫn của HTTP request. Ví dụ: `/?username="abc"&password="def"`. Một số đặc điểm của phương thức GET:

- GET request có thể được **cached**, **bookmark** và lưu trong lịch sử của trình duyệt.

1. **cached** : Caching là quá trình lưu trữ tạm thời dữ liệu để có thể truy cập nhanh hơn trong tương lai. Dữ liệu được lưu trữ trong bộ nhớ đệm (cache) giúp cải thiện hiệu suất của hệ thống bằng cách giảm thiểu thời gian truy cập vào các nguồn dữ liệu gốc.
 2. **bookmark** : là quá trình lưu lại liên kết đến một trang web để có thể truy cập lại sau này một cách dễ dàng.
- GET request bị giới hạn về chiều dài, do chiều dài của URL là có hạn.
 - GET request không nên dùng với dữ liệu quan trọng, chỉ dùng để nhận dữ liệu.

1.2.2 Phương Thức POST

Phương thức POST là yêu cầu từ khách hàng (client) đến máy chủ (server). Phương thức này được sử dụng để gửi dữ liệu đến máy chủ để tạo mới hoặc cập nhật tài nguyên

Với POST thì câu truy vấn sẽ được gửi trong phần message body của HTTP request. Một số đặc điểm của POST như:

- POST không thể, cached, bookmark hay lưu trong lịch sử trình duyệt, và lý do là:
 1. **cached** : POST dùng để tạo hoặc cập nhật tài nguyên. Nếu yêu cầu POST được cache, trình duyệt có thể gửi lại yêu cầu cũ mà không có sự thay đổi nào, dẫn đến tình trạng không nhất quán trong dữ liệu trên máy chủ.
 2. **Bookmark** : Bookmark thường lưu một URL nhưng POST không có URL định nghĩa rõ ràng để lưu lại
 3. **Lịch sử trình duyệt** : POST thường được sử dụng để thay đổi các thông tin của người dùng, đây là những thông tin nhạy cảm nên không thể lưu trên lịch sử trình duyệt
- POST không bị giới hạn về độ dài.

1.2.3 Các phương thức khác

Ngoài GET và POST, HTTP request còn có thể có một số phương thức khác như:

- HEAD: giống như GET nhưng chỉ gửi về HTTP header.
- PUT: tải lên một mô tả về URI định trước.
- DELETE: xóa một tài nguyên định trước.

- OPTIONS: trả về phương thức HTTP mà server hỗ trợ.
- CONNECT: chuyển kết nối của HTTP request thành một kết nối HTTP tunnel.

1.3 HTTP Response

- HTTP Response là lời trả lời từ server
- Cấu trúc HTTP response gần giống với HTTP request, chỉ khác nhau là thay vì Request-Line, thì HTTP có response có Status-Line. Và giống như Request-Line, Status-Line cũng có ba phần như sau:

1. HTTP-version: phiên bản HTTP cao nhất mà server hỗ trợ.
2. Status-Code: mã kết quả trả về.
3. Reason-Phrase: mô tả về Status-Code.

Response Header	Value
(Status-Line)	HTTP/1.1 304 Not Modified
Date	Tue, 17 Mar 2015 14:15:52 GMT
Server	Apache/2
Connection	Keep-Alive
Keep-Alive	timeout=1, max=100
Etag	"b8c1a-2275-5115f173d1e40"
Vary	Accept-Encoding,User-Agent

HTTP Status Codes

Một số loại Status-Code thông dụng mà server trả về cho client như sau:

1xx: information Message: các status code này chỉ có tính chất tạm thời, client có thể không quan tâm.

2xx Successful: khi đã xử lý thành công request của client, server trả về status dạng này:

- 200 OK: request thành công.

- 202 Accepted: request đã được nhận, nhưng không có kết quả nào trả về, thông báo cho client tiếp tục chờ đợi.
- 204 No Content: request đã được xử lý nhưng không có thành phần nào được trả về.
- 205 Reset: giống như 204 nhưng mã này còn yêu cầu client reset lại document view.
- 206 Partial Content: server chỉ gửi về một phần dữ liệu, phụ thuộc vào giá trị range header của client đã gửi.

3xx Redirection: server thông báo cho client phải thực hiện thêm thao tác để hoàn tất request:

- 301 Moved Permanently: tài nguyên đã được chuyển hoàn toàn tới địa chỉ Location trong HTTP response.
- 303 See other: tài nguyên đã được chuyển tạm thời tới địa chỉ Location trong HTTP response.
- 304 Not Modified: tài nguyên không thay đổi từ lần cuối client request, nên client có thể sử dụng đã lưu trong cache.

4xx Client error: lỗi của client:

- 400 Bad Request: request không đúng dạng, cú pháp.
- 401 Unauthorized: client chưa xác thực.
- 403 Forbidden: client không có quyền truy cập.
- 404 Not Found: không tìm thấy tài nguyên.
- 405 Method Not Allowed: phương thức không được server hỗ trợ.

5xx Server Error: lỗi của server:

- 500 Internal Server Error: có lỗi trong quá trình xử lý của server.
- 501 Not Implemented: server không hỗ trợ chức năng client yêu cầu.
- 503: Service Unavailable: Server bị quá tải, hoặc bị lỗi xử lý.

2.HTTP Request với Postman

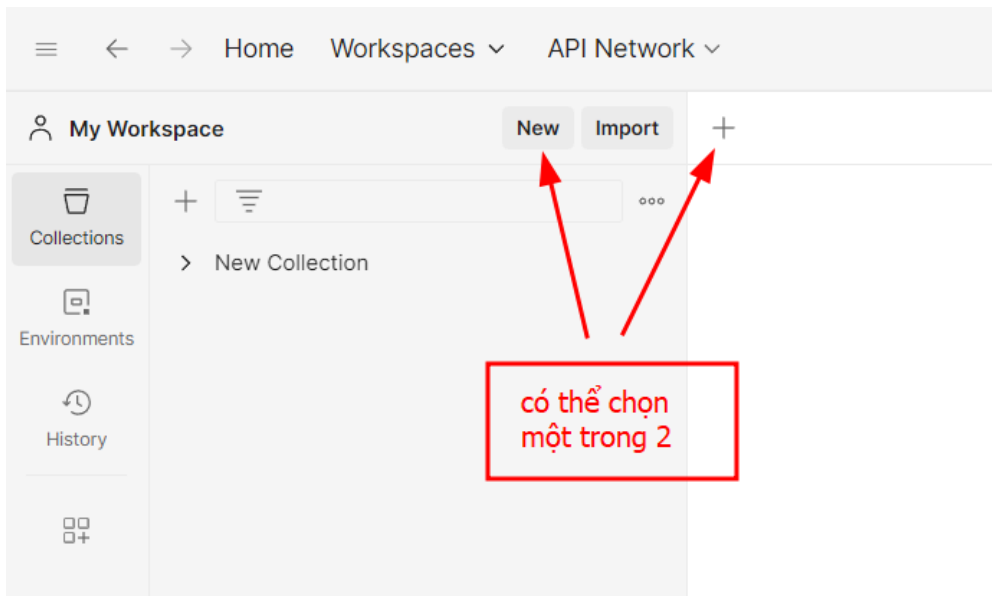
2.1 Postman

- Postman là một công cụ mạnh mẽ được sử dụng để phát triển, thử nghiệm và quản lý API (Application Programming Interface)

- Postman cho phép người dùng gửi các yêu cầu HTTP đến một server và nhận phản hồi. Bạn có thể gửi các loại yêu cầu khác nhau như GET, POST, PUT, DELETE, PATCH, v.v

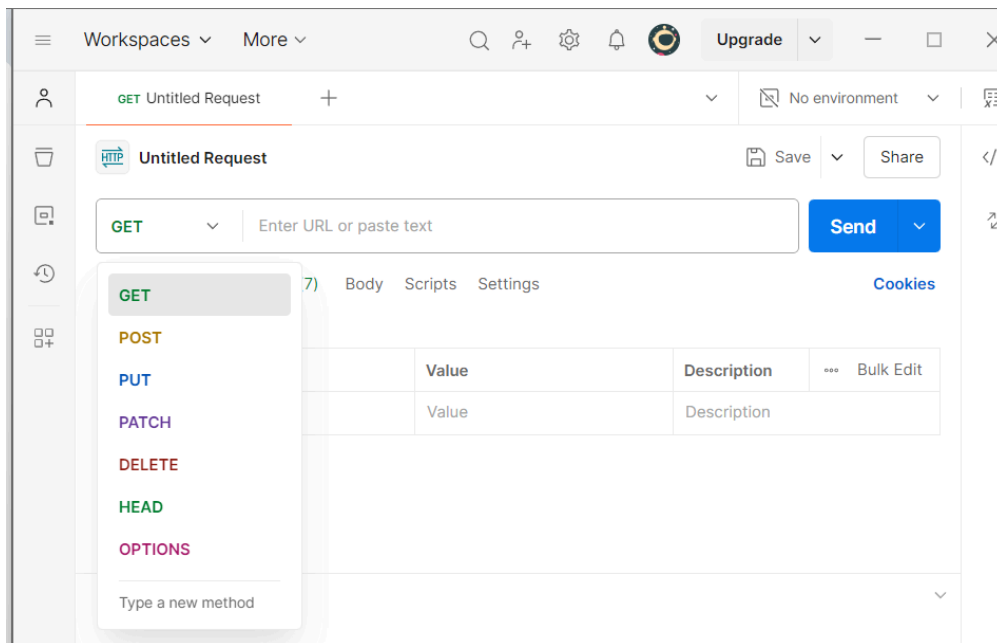
2.2 HTTP Request với Postman

1. Đầu tiên hãy mở Postman mà tạo một yêu cầu mới bằng cách nhấn vào chữ **new** hoặc dấu cộng



2. Ở phần phương thức HTTP hãy chọn các phương thức mà bạn muốn sử dụng :

- **GET**: Lấy dữ liệu từ server.
- **POST**: Gửi dữ liệu đến server.
- **PUT**: Cập nhật dữ liệu trên server.
- **DELETE**: Xóa dữ liệu trên server.

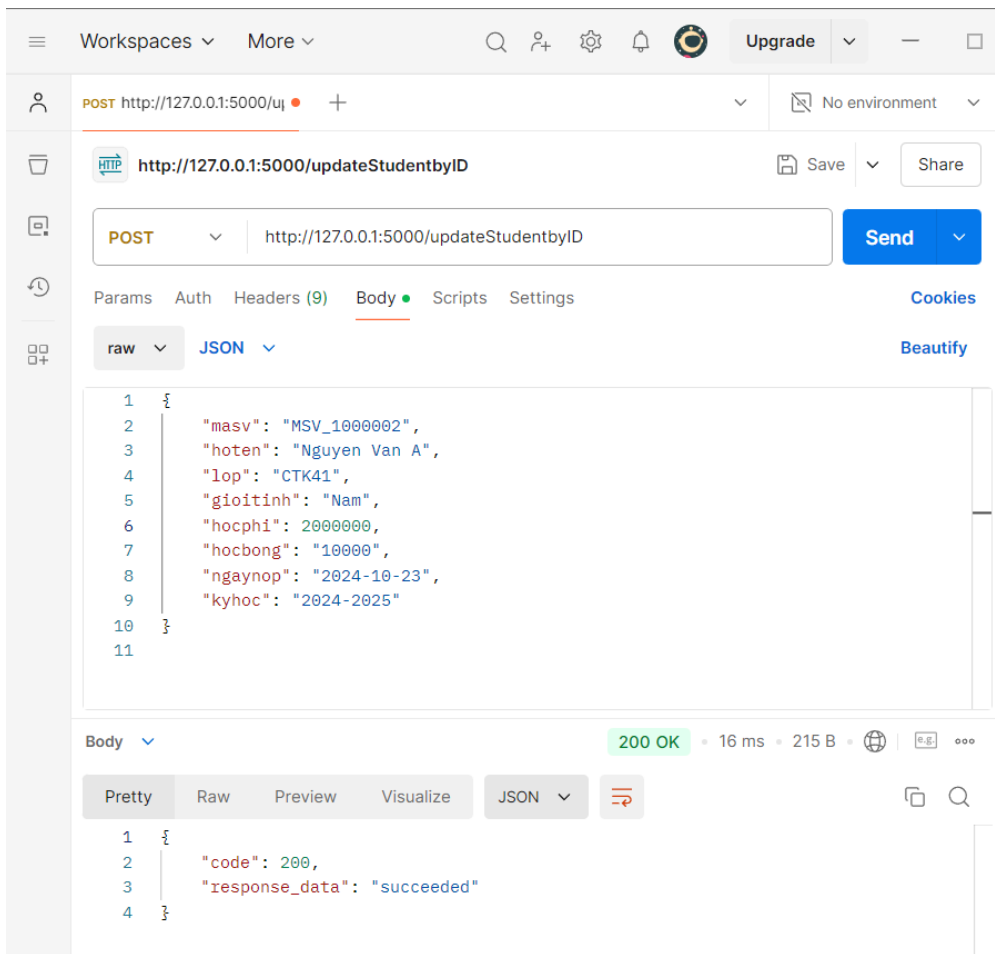


3. Nhập URL của API mà bạn muốn gửi yêu cầu sau đó nhấn chọn **Send**, kết quả sẽ được trả về dưới dạng Json

The screenshot shows the Postman application interface. At the top, there's a header with 'Workspaces' and 'More' dropdowns, a search bar, and a 'Upgrade' button. Below the header, the main area is divided into several sections. The top section shows the request details: a green 'GET' method and the URL 'http://127.0.0.1:5000/ping'. To the right of the URL are 'Save' and 'Share' buttons. Below this is a 'Send' button. The next section is for parameters, with tabs for 'Params', 'Auth', 'Headers (7)', 'Body', 'Scripts', and 'Settings'. The 'Params' tab is active, showing a table for 'Query Params' with columns 'Key', 'Value', and 'Description'. The table has one row with 'Key' and 'Value' fields. Below the parameters section is the 'Body' section, which is currently empty. At the bottom, the response section shows a status of '200 OK' with a response time of '4 ms' and a size of '210 B'. The response is displayed in 'Pretty' format, showing a JSON object:

```
{  "code": 200,  "response_data": "pong"}
```

4. Nếu bạn sử dụng các phương thức POST hay PUT thì hãy chuyển sang phần **body** và chọn vào **raw** sau đó chọn **Json**, nếu hiện ra kết quả thành công thì chương trình của bạn đã được sửa hay đã được cập nhật

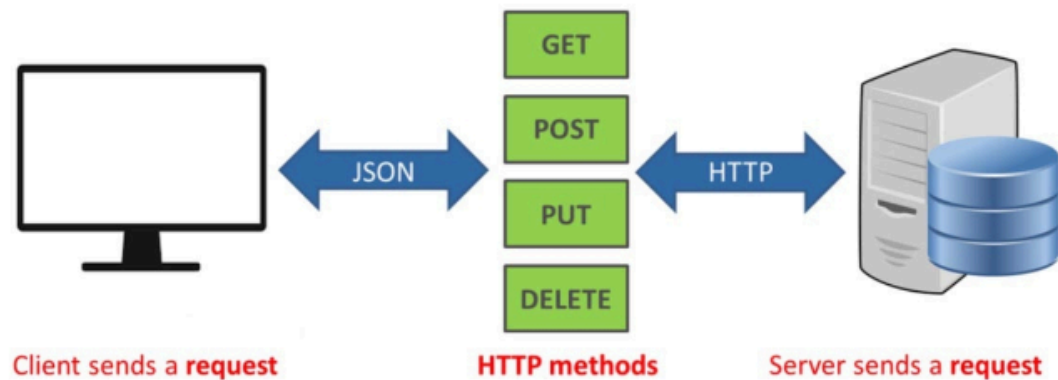


3.Tìm hiểu về web API

- Web API là phương thức cho phép các ứng dụng web riêng biệt có thể giao tiếp, trao đổi dữ liệu qua lại với nhau. Nhờ đó, người dùng có thể kết nối và cập nhật cơ sở dữ liệu một cách đơn giản nhanh chóng.
- Web API sẽ tự động hóa việc quản lý, cập nhật và truy suất dữ liệu. Nhờ đó, nó giúp tăng năng suất cũng như tạo hiệu quả công việc tốt hơn. Nó khá linh động, cho phép lấy dữ liệu từ bất kỳ website/ứng dụng nào (miễn là có hỗ trợ) một cách dễ dàng.
- Để hiểu rõ hơn về Web API là gì thì bạn cần hiểu phương thức hoạt động của nó. Có thể hình dung qua 3 bước đơn giản:
Bước 1: Cơ sở dữ liệu sẽ được cập nhật trên một web server để bất kỳ ai cũng có thể truy cập.

Bước 2: Thông qua giao thức HTTP/HTTPS, bên thứ 3 có thể gửi yêu cầu lấy dữ liệu đến server. Tại đây sẽ kiểm tra/xác minh đồng thời tìm kiếm các dữ liệu tương ứng với yêu cầu.

Bước 3: Dữ liệu sẽ được trả lại (thường ở dạng JSON hoặc XML) cho bên yêu cầu để phục vụ các mục đích khác nhau.



4. cấu trúc JSON data

4.1 Dữ liệu JSON cơ bản

- **JSON** (JavaScript Object Notation) là một định dạng nhẹ để trao đổi dữ liệu, rất phổ biến trong các ứng dụng web và API. Nó dễ đọc cho con người và cũng dễ xử lý cho máy móc.
- **JSON** được biểu diễn dưới dạng **KEY** và **VALUE**. Mỗi cặp này sẽ cách nhau bởi một dấu phẩy `,`, và toàn bộ **JSON** được nằm trong dấu ngoặc nhọn `{ }`
- **KEY** là một chuỗi (string) và phải được đặt trong dấu ngoặc kép `" "`
- **VALUE** Có thể là một trong các kiểu dữ liệu như chuỗi, số, đối tượng, mảng, giá trị boolean, hoặc `null`

4.2 Các kiểu dữ liệu trong JSON

- **String (Chuỗi):** Chuỗi ký tự nằm trong dấu ngoặc kép. Ví dụ: `"Hello World"`
- **Number (Số):** Cả số nguyên và số thực. Ví dụ: `123` , `45.67`
- **Boolean:** Chỉ có hai giá trị là `true` hoặc `false` .
- **Array (Mảng):** Một danh sách các giá trị, có thể chứa bất kỳ kiểu dữ liệu nào. Ví dụ: `[1, "apple", true]`
- **Object (Đối tượng):** Một tập hợp các cặp key-value. Ví dụ: `{"name": "John", "age": 30}`
- **Null:** Giá trị rỗng, biểu thị giá trị không xác định. Ví dụ: `null`

4.3 Ứng dụng của JSON

- **API:** JSON thường được sử dụng để trao đổi dữ liệu giữa máy chủ và ứng dụng
- **Lưu trữ cấu hình:** JSON cũng được sử dụng để lưu trữ dữ liệu cấu hình trong các ứng dụng.

5. Flask API

5.1 Kiến trúc của Flask API

1. **Flask Application:** Tạo một instance của ứng dụng Flask là bước đầu tiên. Instance này sẽ quản lý tất cả các route, middleware, và cấu hình của ứng dụng.
2. **Routes:** Flask cho phép bạn định nghĩa các route (đường dẫn) bằng cách sử dụng decorators. Mỗi route tương ứng với một URL và một hàm xử lý yêu cầu.
3. **Request Object:** Chứa thông tin về yêu cầu mà client gửi đến server, bao gồm headers, form data, và JSON payload.
4. **Response Object:** Đối tượng trả về cho client, có thể là các nội dung HTML, JSON, hoặc các mã trạng thái HTTP khác.
5. **Models:** Thể hiện dữ liệu và logic truy xuất dữ liệu.
6. **Templates:** Nếu ứng dụng cần render HTML, Flask sử dụng Jinja2 để tạo các template HTML.

5.2 Các thư viện chính của Flask API

1. Flask

- **Chức năng:** Là framework chính, cung cấp các công cụ và cấu trúc cơ bản để xây dựng ứng dụng web.
- **Hoạt động:** Cho phép bạn định nghĩa các route (đường dẫn), xử lý các yêu cầu HTTP, và trả về phản hồi.

2. Flask-RESTful

- **Chức năng:** Mở rộng Flask để dễ dàng xây dựng API RESTful.
- **Hoạt động:** Cung cấp các lớp và phương thức để tạo tài nguyên (resources) và xử lý các phương thức HTTP như GET, POST, PUT, DELETE.

3. Flask-SQLAlchemy

- **Chức năng:** Tích hợp SQLAlchemy vào Flask để dễ dàng quản lý cơ sở dữ liệu.
- **Hoạt động:** Cho phép bạn sử dụng ORM (Object-Relational Mapping) để tương tác với cơ sở dữ liệu thông qua các lớp Python thay vì sử dụng SQL thuần túy.

4. Flask-Migrate

- **Chức năng:** Quản lý di chuyển (migration) cơ sở dữ liệu cho ứng dụng Flask.
- **Hoạt động:** Dùng để tạo, áp dụng, và quay lại các thay đổi trong cấu trúc cơ sở dữ liệu.

5. Flask-JWT-Extended

- **Chức năng:** Cung cấp phương thức xác thực JWT (JSON Web Token) cho ứng dụng Flask.
- **Hoạt động:** Cho phép bảo vệ các route bằng cách yêu cầu token xác thực cho các yêu cầu nhất định.

6. Flask-CORS

- **Chức năng:** Cho phép Cross-Origin Resource Sharing (CORS) cho ứng dụng Flask.
- **Hoạt động:** Giúp cho các yêu cầu từ miền khác có thể truy cập vào API của bạn.

7. Flask-Login

- **Chức năng:** Quản lý phiên đăng nhập người dùng.
- **Hoạt động:** Cung cấp các tiện ích để xác thực người dùng và duy trì trạng thái đăng nhập.

8. Flask-Email

- **Chức năng:** Gửi email từ ứng dụng Flask.
- **Hoạt động:** Cung cấp một giao diện đơn giản để gửi email thông qua các dịch vụ SMTP.

5.3 Cách hoạt động tổng thể của Flask API

1. Khởi động ứng dụng

- Khi bạn khởi động ứng dụng Flask, nó sẽ chạy một server web (mặc định là Flask Development Server) để lắng nghe các yêu cầu đến một cổng cụ thể (thường là cổng 5000).

2. Định nghĩa các route (đường dẫn)

- Sử dụng decorators để định nghĩa các route cho API. Mỗi route tương ứng với một URL cụ thể và có thể xử lý các phương thức HTTP như GET, POST, PUT, DELETE.

3. Xử lý yêu cầu

- Khi server nhận một yêu cầu từ client (trình duyệt hoặc ứng dụng), Flask sẽ xác định route nào phù hợp với yêu cầu dựa trên URL và phương thức HTTP.
- Flask sau đó sẽ gọi hàm xử lý tương ứng cho route đó.

4. Tương tác với cơ sở dữ liệu (nếu cần)

- Nếu API cần truy cập hoặc thay đổi dữ liệu, nó có thể tương tác với cơ sở dữ liệu thông qua ORM như SQLAlchemy hoặc thông qua các truy vấn SQL thuần túy.

5. Xử lý xác thực và phân quyền (nếu cần)

- Nếu API yêu cầu xác thực người dùng, có thể sử dụng các thư viện như Flask-JWT-Extended hoặc Flask-Login để kiểm tra token hoặc phiên đăng nhập trước khi xử lý yêu cầu.

6. Trả về phản hồi

- Sau khi xử lý yêu cầu, API sẽ trả về một phản hồi cho client. Phản hồi này thường ở định dạng JSON, nhưng cũng có thể là HTML, XML hoặc một định dạng khác.

7. Xử lý lỗi

- Bạn có thể định nghĩa các hàm xử lý lỗi để trả về thông điệp lỗi cụ thể cho các lỗi phổ biến (như 404 Not Found, 400 Bad Request, 500 Internal Server Error).

8. Logging và Monitoring

- Trong quá trình hoạt động, bạn có thể thêm logging để ghi lại thông tin về các yêu cầu và phản hồi. Điều này hữu ích để theo dõi và gỡ lỗi ứng dụng.
- Có thể tích hợp các công cụ giám sát để theo dõi hiệu suất của API và phát hiện sự cố.

6. Cách sử dụng cơ bản của Flask API

6.1 Cách bắt đầu đơn giản

1. Đầu tiên chúng ta nhập `Flask` lớp. Một thể hiện của lớp này sẽ là ứng dụng WSGI của chúng ta.
2. Tiếp theo, chúng ta tạo một thể hiện của lớp này. Đối số đầu tiên là tên của mô-đun hoặc gói của ứng dụng. `__name__` là một phím tắt tiện lợi cho việc này, phù hợp với hầu hết các trường hợp. Điều này là cần thiết để Flask biết nơi tìm kiếm các tài nguyên như mẫu và tệp tĩnh.
3. Sau đó, chúng ta sử dụng `route()` trình trang trí để cho Flask biết URL nào sẽ kích hoạt hàm của chúng ta.
4. Hàm trả về thông điệp mà chúng ta muốn hiển thị trên trình duyệt của người dùng. Kiểu nội dung mặc định là HTML, do đó HTML trong chuỗi sẽ được trình duyệt hiển thị.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

Lưu dưới dạng `hello.py` hoặc tên tương tự. Đảm bảo không gọi ứng dụng của bạn `flask.py` vì điều này sẽ xung đột với chính Flask. Để chạy ứng dụng, hãy sử dụng `flask` lệnh hoặc `.`. Bạn cần cho Flask biết ứng dụng của bạn ở đâu bằng tùy chọn này. `python -m flask --app`

```
$ flask --app hello run
* Serving Flask app 'hello'
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```

Sau khi đã sử dụng câu lệnh với Flask bây giờ hãy truy cập <http://127.0.0.1:5000/> và bạn sẽ thấy **Hello world**

6.2 Chế độ gỡ lỗi

Lệnh này có thể làm nhiều hơn là chỉ khởi động máy chủ phát triển. Bằng cách bật chế độ gỡ lỗi, máy chủ sẽ tự động tải lại nếu mã thay đổi và sẽ hiển thị trình gỡ lỗi tương tác trong trình duyệt nếu xảy ra lỗi trong khi yêu cầu. `flask run`

TypeError

TypeError: cannot concatenate 'str' and 'NoneType' objects

Traceback (most recent call last)

```
File "/Users/mitsuhiko/Development/flask/flask.py", line 650, in __call__
    return self.wsgi_app(environ, start_response)

File "/Users/mitsuhiko/Development/werkzeug-main/werkzeug/wsgi.py", line 406, in __call__
    return self.app(environ, start_response)

File "/Users/mitsuhiko/Development/flask/flask.py", line 616, in wsgi_app
    rv = self.dispatch_request()

File "/Users/mitsuhiko/Development/flask/flask.py", line 535, in dispatch_request
    return self.view_functions[endpoint](**values)

File "/Users/mitsuhiko/Development/flask/test.py", line 8, in index
    return 'Hello ' + name

[console ready]
>>> type(name)
<type 'NoneType'>
>>>
```

Để bật chế độ gỡ lỗi, hãy sử dụng `--debug` tùy chọn. Khi thực hiện chạy hãy sử dụng `debug=True`

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

Và kết quả sẽ được hiện ra là :

```
$ flask --app hello run --debug
* Serving Flask app 'hello'
* Debug mode: on
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
```


- * Restarting with stat
- * Debugger is active!
- * Debugger PIN: nnn-xxx-xxx

6.3 Thoát HTML

- **HTML (HyperText Markup Language)** là ngôn ngữ đánh dấu tiêu chuẩn được sử dụng để tạo ra các trang web. HTML cung cấp cấu trúc cho nội dung trang web bằng cách sử dụng các thẻ (tags) để định nghĩa các phần tử khác nhau như tiêu đề, đoạn văn, hình ảnh, liên kết, bảng, biểu mẫu, và nhiều nội dung khác. HTML không phải là ngôn ngữ lập trình mà là ngôn ngữ đánh dấu, dùng các thẻ (tags) để định nghĩa cấu trúc của nội dung trên trang web.
- **Thẻ HTML (HTML Tags):** HTML sử dụng các thẻ để đánh dấu các phần tử trên trang. Mỗi thẻ thường có một thẻ mở và một thẻ đóng. Thẻ mở là `<p>`, thẻ đóng là `</p>`, và nội dung của đoạn văn nằm giữa các thẻ này.
- Khi trả về HTML (kiểu phản hồi mặc định trong Flask), bất kỳ giá trị nào do người dùng cung cấp được hiển thị trong đầu ra phải được thoát để bảo vệ khỏi các cuộc tấn công tiêm nhiễm. Các mẫu HTML được hiển thị bằng Jinja, được giới thiệu sau, sẽ tự động thực hiện việc này. `escape()`, được hiển thị ở đây, có thể được sử dụng theo cách thủ công. Nó được bỏ qua trong hầu hết các ví dụ để ngắn gọn, nhưng bạn nên luôn lưu ý cách bạn đang sử dụng dữ liệu không đáng tin cậy.

```
from markupsafe import escape

@app.route("/<name>")
def hello(name):
    return f"Hello, {escape(name)}!"
```

6.4 Quy tắc biến đổi

Bạn có thể thêm các phần biến vào URL bằng cách đánh dấu các phần bằng `<variable_name>`. Sau đó, hàm của bạn nhận được `<variable_name>` dưới dạng đối số từ khóa. Tùy chọn, bạn có thể sử dụng trình chuyển đổi để chỉ định loại đối số như `<converter:variable_name>`. Các loại bộ chuyển đổi mà chúng ta có thể dùng là **"string, int, float, path, uuid"**

```

from markupsafe import escape

@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return f'User {escape(username)}'

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return f'Post {post_id}'

@app.route('/path/<path:subpath>')
def show_subpath(subpath):
    # show the subpath after /path/
    return f'Subpath {escape(subpath)}'

```

6.5 Xây dựng URL

- Để xây dựng URL cho một hàm cụ thể, hãy sử dụng `url_for()` hàm. Nó chấp nhận tên của hàm làm đối số đầu tiên và bất kỳ số lượng đối số từ khóa nào, mỗi đối số tương ứng với một phần biến của quy tắc URL. Các phần biến không xác định được thêm vào URL làm tham số truy vấn.

Ví dụ, ở đây chúng ta sử dụng `test_request_context()` phương thức để thử `url_for()`. `test_request_context()` yêu cầu Flask hoạt động như thể nó đang xử lý một yêu cầu ngay cả khi chúng ta sử dụng shell Python.

```

from flask import url_for

@app.route('/')
def index():
    return 'index'

@app.route('/login')

```

```
def login():
    return 'login'

@app.route('/user/<username>')
def profile(username):
    return f'{username}'s profile'

with app.test_request_context():
    print(url_for('index'))
    print(url_for('login'))
    print(url_for('login', next='/'))
    print(url_for('profile', username='John Doe'))
```

Kết quả sẽ hiện ra và chings ta chỉ cần sử dụng đường dẫn này để truy cập một cách nhanh chóng

```
/
/login
/login?next=/
/user/John%20Doe
```

6.6 Phương pháp HTTP

Các ứng dụng web sử dụng các phương thức HTTP khác nhau khi truy cập URL. Bạn nên làm quen với các phương thức HTTP khi làm việc với Flask. Theo mặc định, một tuyến đường chỉ trả lời `GET` các yêu cầu. Bạn có thể sử dụng `methods` đối số của `route()` trình trang trí để xử lý các phương thức HTTP khác nhau.

```
from flask import request

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return do_the_login()
```

```
else:  
    return show_the_login_form()
```

7. Một số các Flask API cơ bản

1. Viết API /ping trả về: {"code": 200, "response_data": "pong"}

```
from flask import Flask, jsonify  
# Khởi tạo ứng dụng Flask  
app = Flask(__name__)  
@app.route('/')  
@app.route('/ping', methods=['GET'])  
def ping():  
    return jsonify({  
        "code": 200,  
        "response_data": "pong"  
    })  
# Chạy ứng dụng nếu file này được chạy trực tiếp  
if __name__ == '__main__':  
    app.run(debug=True)
```

Kết quả nhận được là :

GET http://127.0.0.1:5000/ping

Params Authorization Headers (9) Body ● Scripts Settings

Query Params

	Key	Value
	Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ↕

```
1 {
2   "code": 200,
3   "response_data": "pong"
4 }
```

**2. Viết API GET /findbyID?id=xxx lấy thông tin sinh viên theo ID từ Database truyền vào.
{"code": 200, "response_data": {<thông tin sinh viên dạng json>}}**

```
from flask import Flask, jsonify, request
import mysql.connector

app = Flask(__name__)

# Hàm để tạo kết nối mới đến MySQL cho mỗi request
def get_db_connection():
    return mysql.connector.connect(
        host="localhost",
        user="root",
        password="Phuc00000@",
        database="dachsachsv"
    )
```

```

@app.route('/findbyID', methods=['GET'])
def lay_id():
    # Lấy ID từ query string
    sinhvien_id = request.args.get('masv')
    print(f"ID nhận được: {sinhvien_id}")
    # Tạo kết nối đến MySQL
    connection = get_db_connection()
    mycursor = connection.cursor(dictionary=True)
    try:
        # Thực hiện truy vấn
        mycursor.execute("SELECT * FROM thongtinsv WHERE masv = %s", (sinhvien_id,))
        sinhvien = mycursor.fetchone() # Lấy một bản ghi

        # Kiểm tra xem sinh viên có tồn tại không
        if sinhvien:
            return jsonify({"code": 200, "response_data": sinhvien}) # Trả về thông tin sinh viên
        else:
            return jsonify({"code": 404, "message": "Không tìm thấy sinh viên"}), 404 # Trả về lỗi không tìm thấy
    except mysql.connector.Error as err:
        return jsonify({"code": 500, "message": str(err)}), 500 # Trả về lỗi nếu có
    finally:
        # Đóng cursor và kết nối
        mycursor.close()
        connection.close()
if __name__ == '__main__':
    app.run(debug=True) # Chạy ứng dụng trong chế độ debug

```

Ở đây chúng ta sử dụng phương thức GET để gửi đi yêu cầu gọi mã sinh viên là **"MSV_1000000"** và kết quả sẽ trả về là :

HTTP http://127.0.0.1:5000/findbyID?masv=MSV_1000000

GET http://127.0.0.1:5000/findbyID?masv=MSV_1000000

Params • Authorization Headers (9) Body • Scripts Settings

Key	Value
masv	MSV_1000000
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1  {
2    "code": 200,
3    "response_data": {
4      "gioitinh": "Nữ",
5      "hocbong": 0.0,
6      "hocphi": 131783.0,
7      "hoten": "Thành Trần",
8      "kyhoc": "2024-1",
9      "lop": "DL2",
10     "masv": "MSV_1000000",
11     "ngaynop": "Thu, 01 Feb 2024 00:00:00 GMT"
12   }
13 }

```

3. Viết API GET /countStudents lấy thông tin số lượng sinh viên trong bảng Database.
{"code": 200, "response_data": {<thông tin tổng số lượng sinh viên>}}

```

import mysql.connector
from flask import Flask, jsonify
value = Flask(__name__)
connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Phuc00000@",


```


```

database="dachsachsv"
)
@value.route("/countStudents", methods=['GET'])
def count_students():
    mycursor = connection.cursor()
    mycursor.execute("SELECT COUNT(*) FROM thongtinsv")
    result=mycursor.fetchone()
    return jsonify({"code": 200, "response_data": {"total_students": result} })
    mycursor.close()
if __name__=='__main__':
    value.run(debug=True)

```

Tương tự như ở câu 2 chúng ta sử dụng phương thức GET để gửi đi yêu cầu xem tổng số sinh viên

 http://127.0.0.1:5000/countStudents



GET  http://127.0.0.1:5000/countStudents

Params Authorization Headers (9) Body ● Scripts Settings

Query Params

	Key	Value
	Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON  

```

1  {
2    "code": 200,
3    "response_data": {
4      "total_students": [
5        5602929
6      ]
7    }
8  }

```


4. Viết API POST /updateStudentbyID để update thông tin của 1 sinh viên theo ID. {"code": 200, "response_data": "succeeded"}

```
from flask import Flask, jsonify, request
import mysql.connector
app = Flask(__name__)
# Kết nối đến cơ sở dữ liệu MySQL
connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="Phuc00000@",
    database="dachsachsv"
)
@app.route('/updateStudentbyID', methods=['POST'])
def update_student_by_id():
    # Lấy dữ liệu từ yêu cầu JSON
    data = request.get_json()

    # Lấy masv và các thông tin cần cập nhật
    masv = data.get('masv')
    hoten = data.get('hoten')
    lop = data.get('lop')
    gioitinh = data.get('gioitinh')
    hocphi = data.get('hocphi')
    hocbong = data.get('hocbong')
    ngaynop = data.get('ngaynop')
    kyhoc = data.get('kyhoc')
    # Kiểm tra xem ID có được cung cấp hay không
    if not masv:
        return jsonify({"code": 400, "message": "ID sinh viên không được cung cấp"}), 400
    # Khởi tạo cursor
    mycursor = connection.cursor()
```

```

try:
    # Cập nhật thông tin sinh viên trong cơ sở dữ liệu
    sql_update_query = """
    UPDATE thongtinsv      SET hoten = %s, lop = %s, gioitinh = %s, hocphi = %s, hocbong = %s, ngaynop = %s, kyhoc = %s      WHERE masv = %s
    """

    mycursor.execute(sql_update_query, (hoten, lop, gioitinh, hocphi, hocbong, ngaynop, kyhoc, masv))
    connection.commit() # Lưu thay đổi vào cơ sở dữ liệu

    if mycursor.rowcount > 0: # Kiểm tra xem có dòng nào được cập nhật không
        return jsonify({"code": 200, "response_data": "succeeded"}), 200
    else:
        return jsonify({"code": 404, "message": "Không tìm thấy sinh viên với ID cung cấp"}), 404

except mysql.connector.Error as err:
    return jsonify({"code": 500, "message": str(err)}), 500 # Trả về lỗi nếu có
finally:
    mycursor.close() # Đóng cursor

if __name__ == '__main__':
    app.run(debug=True) # Chạy ứng dụng trong chế độ debug

```

Ở đây chúng ta sử dụng phương thức POST để cập nhật lại thông tin cho sinh viên có mã sinh viên là **"MSV_1000008"**



http://127.0.0.1:5000/updateStudentbyID

Save



Share

POST



http://127.0.0.1:5000/updateStudentbyID

Send



Params Auth Headers (9) **Body** Scripts Settings

Cookies

raw

JSON

Beautify

```
1 {
2   "masv": "MSV_1000000",
3   "hoten": "Trần Phương Nam",
4   "lop": "QTANM",
5   "gioitinh": "Nam",
6   "hocphi": 104968.0,
7   "hocbong": 0.0,
8   "ngaynop": "2024-10-24",
9   "kyhoc": "2024-1"
10 }
```

Body

200 OK

18 ms

215 B



Pretty

Raw

Preview

Visualize

JSON



```
1 {
2   "code": 200,
3   "response_data": "succeeded"
4 }
```