



API TESTING VỚI POSTMAN

LỜI NÓI ĐẦU

Cuốn ebook này là thành quả trong việc tổng hợp kiến thức và tìm hiểu về test API của mình. Nó có thể đúng và chưa đúng, nhưng chắc sẽ phù hợp cho những người mới tiếp cận test API và sử dụng công cụ Postman. Read and enjoy it!

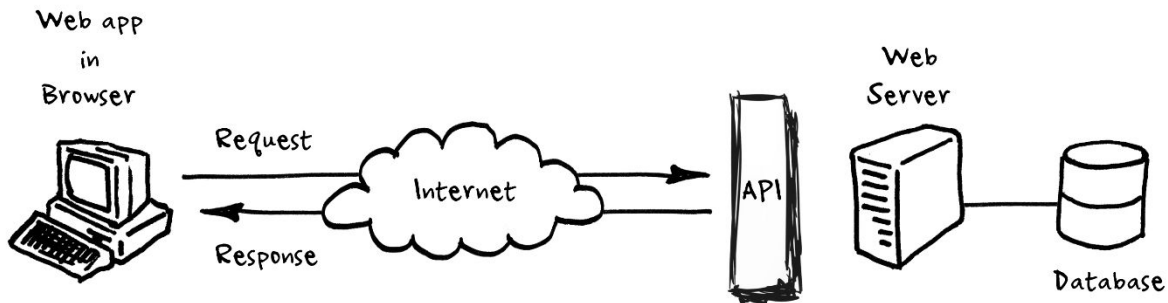
Giang Nguyen

CONTENT

I. API là gì? Và vì sao phải test API ?	2
II. Protocol dùng trong Restful API	4
III. Định dạng dữ liệu JSON và XML	7
IV. Giới thiệu chung về Postman	10
V. Cách tạo Request	14
VI. Collections trong Postman	17
VII. Cách sử dụng Environments	21
VIII. Test Response	25
IX. Sử dụng Pre-request Script	30
X. Xây dựng API Document	34
XI. Run Test Suites từ Runner	41
XII. Cách test API như thế nào?	46

I. API là gì? Và vì sao phải test API ?

1. API là gì?

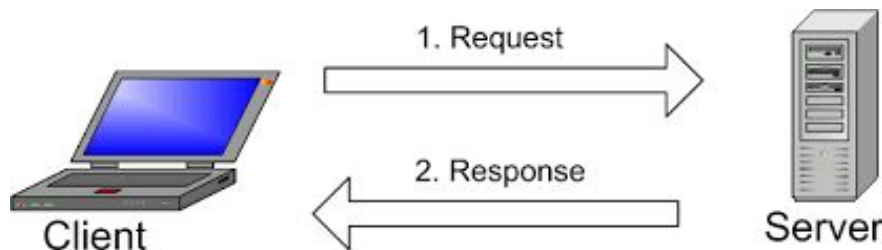


Nói đơn giản, API là cái cầu nối giữa client và server. Client ở đây có thể là máy tính, điện thoại sử dụng hệ điều hành khác nhau và được viết bằng những ngôn ngữ khác nhau, ví dụ như Swift, Objective-C, Java. Tương tự, server back-end cũng được viết bằng các ngôn ngữ khác nhau. Để 2 thằng này có thể nói chuyện được với nhau chúng phải nói cùng 1 ngôn ngữ. Ngôn ngữ ấy chính là API.

Chúng ta hãy lấy một ví dụ đơn giản cho vấn đề này :

Giả sử bạn là 1 người hướng dẫn viên du lịch, và quản lý 1 nhóm du lịch hợp chủng quốc. Trong nhóm có người Nga, Mỹ, Nhật, Thụy Điển, Đức, Việt Nam. Để có thể làm mọi việc một cách suôn sẻ, tất cả cái nhóm này phải cùng nói 1 ngôn ngữ, có thể là tiếng anh hoặc tiếng Việt. Ở đây người hướng dẫn viên sẽ đóng vai trò là Server, người du lịch sẽ đóng vai trò là client.

Khi đi trên đường hoặc đến thăm địa danh du lịch, những người khách có thể hỏi hướng dẫn viên rất nhiều câu hỏi khác nhau. “Cái kia là cái gì?”, “Quả này ăn như thế nào?”, “Bạn ơi, chỗ đi WC ở đâu nhỉ?”... Với mỗi một hành động hỏi như vậy, tương ứng với việc gửi 1 request được gửi lên server với những tham số đầu vào như “Cái kia” hay “quả này”. (Gửi request còn được gọi là Call API). Với mỗi câu hỏi, người hướng dẫn viên sẽ trả lời 1 cách khác nhau – cái này gọi là response. “Cái đó là cái để đập vào đầu những đứa nào hỏi nhiều”, “Quả này cứ cho vào mồm là xong”. :)))



Định dạng trong việc hỏi và trả lời ở trên có thể thông qua trò chuyện trực tiếp hoặc viết giấy. Ở trong API thì có 2 định dạng chính là xml và json. Hiện tại, mình chỉ có kinh nghiệm với json nên chỉ giới thiệu và lấy ví dụ ở những phần sau bằng json thôi.

2. Vì sao phải test API?

Trong quá trình triển khai dự án, phần server và client làm độc lập với nhau nên có nhiều chỗ client chưa làm xong, mình không thể chờ client làm xong để test được dữ liệu mà test API bằng công cụ khác luôn → Lúc này việc test hoàn toàn không phụ thuộc gì vào client.

Kể cả khi client làm xong rồi, nếu mình test trên client mà thấy lỗi liên quan đến logic và dữ liệu thì cũng cần test thêm cả API để biết chính xác là server sai hay client sai → fix lỗi sẽ nhanh hơn.

Khi làm hệ thống web services, dự án của mình chỉ viết API cho bên khác dùng, mình sẽ không có client để test giống như các dự án khác → phải test API hoàn toàn.

II. Protocol dùng trong Restful API

Ở phần I, mình đã giải thích vai trò của API đối với client và server, phần này sẽ nói về cách mà 2 thằng đó (sau đây gọi là 2 chiếc máy tính) nói chuyện với nhau.

1. Protocol là gì?

Giả sử: Có 2 người A và B nói chuyện với nhau qua điện thoại, nếu người A hỏi 1 câu rồi im lặng, người B sẽ biết rằng người A đang chờ đợi câu trả lời và đến lượt người B nói. Hai chiếc máy tính cũng giao tiếp 1 cách lịch sự như vậy và được mô tả với cái thuật ngữ "Protocol" – giao thức.

Giao thức chính là những luật lệ được chấp thuận để 2 cái máy tính có thể nói chuyện với nhau.

Tuy nhiên, luật lệ này chặt chẽ hơn rất nhiều so với giao tiếp giữa người với người. Máy tính sẽ không thông minh để có thể nhận biết 2 câu "A là chồng B" hay "B là vợ A" có cùng ý nghĩa. Để 2 máy tính giao tiếp hiệu quả, server phải biết chính xác cách mà client sắp xếp cái message nó gửi lên như thế nào.

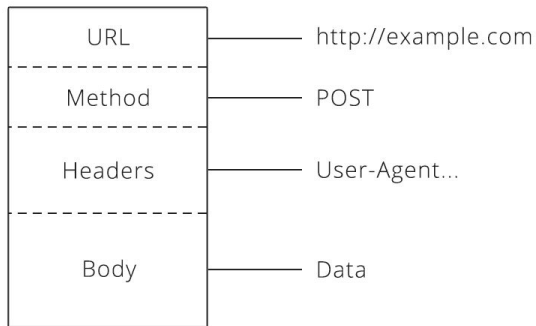
Chúng ta đã từng nghe đến những Protocol cho những mục đích khác nhau, ví dụ như Mail có POP hay IMAP, message có XMPP, Kết nối thiết bị: Bluetooth. Trong web thì Protocol chính là HTTP – HyperText Transfer Protocol, vì sự phổ biến của nó mà hầu hết các công ty chọn nó là giao thức cho các API.

Lưu ý: API có thể viết trên nền Protocol khác, ví dụ như SOAP. Trong phần này, mình chỉ giới thiệu về HTTP.

2. HTTP hoạt động như thế nào?

Cuộc sống của HTTP xoay quanh cái vòng luẩn quẩn: Request và Response. Client gửi request, server gửi lại response là liệu server có thể làm được cái client muốn hay không. Và API được xây dựng trên chính 2 thành phần: Request và Reponse. Trước tiên, ta phải hiểu cấu trúc của mỗi thành phần.

Request



Request

Một cái request đúng chuẩn cần có 4 thứ:

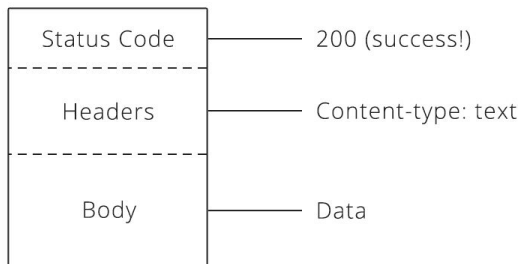
1. URL
2. Method
3. Header
4. Body

OK, bây giờ sẽ soi từng thứ một.

- **URL** là 1 cái địa chỉ duy nhất cho 1 thứ (dùng danh từ), có thể là web page, image, hoặc video. API mở rộng cái ý tưởng gốc của URL cho những thứ khác, ví dụ: customers, products. Và như thế client dễ dàng cho server biết cái nó muốn là cái gì, những cái này còn được gọi chung là “resources” – nguồn lực.
- **Method**: là cái hành động client muốn tác động lên “resources”, và nó thường là động từ. Có 4 loại Method hay được dùng:
 - GET: Yêu cầu server đưa lại resource: Hãy tưởng tượng ra cái cảnh vào fb, tay vuốt new feeds.
 - POST: Yêu cầu server cho tạo ra 1 resource mới. Ví dụ: đăng ký 1 chuyến đi ở GrabBike.
 - PUT: Yêu cầu server cho sửa / thêm vào resource đã có trên hệ thống. Ví dụ: Edit 1 post ở trên fb.
 - DELETE: Yêu cầu server cho xóa 1 resource. Cái này chắc chắn cần ví dụ.
- **Header**: nơi chứa các thông tin cần thiết của 1 request nhưng end-users không biết có sự tồn tại của nó. Ví dụ: độ dài của request body, thời gian gửi request, loại thiết bị đang sử dụng, loại định dạng cái response mà client có đọc được...
- **Body**: nơi chứa thông tin mà client sẽ điền. Giả sử bạn đặt 1 cái bánh pizza, thì thông tin ở phần body sẽ là: Loại bánh pizza, kích cỡ, số lượng đặt.

Response:

Sau khi nhận được request từ phía client, server sẽ xử lý cái request đó và gửi ngược lại cho client 1 cái response. Cấu trúc của 1 response tương đối giống phần request nhưng Status code sẽ thay thế cho URL và Method. Tóm lại, nó có cấu trúc 3 phần:



Response

1. Status code
2. Header
3. Body

- **Status code** là những con số có 3 chữ số và có duy nhất 1 ý nghĩa. Chắc các bạn cũng không còn lạ lẫm với những Error “404 Not Found” hoặc “503 Service Unavailable”. Full list có ở [đây](#).
- **Header** và **Body** tương đối giống với request.

III. Định dạng dữ liệu JSON và XML

Như phần 1 đã có giới thiệu, định dạng data trong API thường dùng 2 loại chính là JSON (JavaScript Object Notation) và XML (Extensible Markup Language). Hôm nay, mình sẽ nói kỹ hơn về từng loại định dạng.

1. Định dạng JSON

Ngày nay, JSON được sử dụng nhiều trong Restful API. Nó được xây dựng từ Javascript, ngôn ngữ mà được dùng nhiều, tương thích với cả front-end và back-end của cả web app và web service. JSON là 1 định dạng đơn giản với 2 thành phần: keys và values.

- Key thể hiện thuộc tính của Object
- Value thể hiện giá trị của từng Key

Ví dụ:

```
{  
  "id": "5",  
  "title": "KUNIKO METHOD",  
  "created": "2017-04-24 13:48:56"  
}
```

Trong ví dụ trên, keys nằm bên trái, values nằm bên phải.



key value

```
"title": "KUNIKO METHOD",
```

Có nhiều trường hợp, 1 Key sẽ có Value là 1 dãy key + value. Ví dụ như hình:


```
{
  "error_code": 0,
  "error_msg": "",
  "data": {
    "id": "26",
    "file": "files/59017ab52cb10.epub"
  }
}
```

Trong hình trên Key có tên là Data có Value là 2 cặp Key + value.

2. Định dạng XML

Trong JSON dùng { } và [] để đánh dấu dữ liệu. XML thì tương tự như HTML, dùng thẻ để đánh dấu và được gọi là nodes.

Lấy luôn ví dụ ở trên nhưng viết bằng xml, nó sẽ như thế này:

```
<error_code>0</error_code>
<error_msg></ error_msg >
<data>
  <id>26</id>
  <file> files/59017ab52cb10.epub </file>
</data>
```

node mở value node đóng

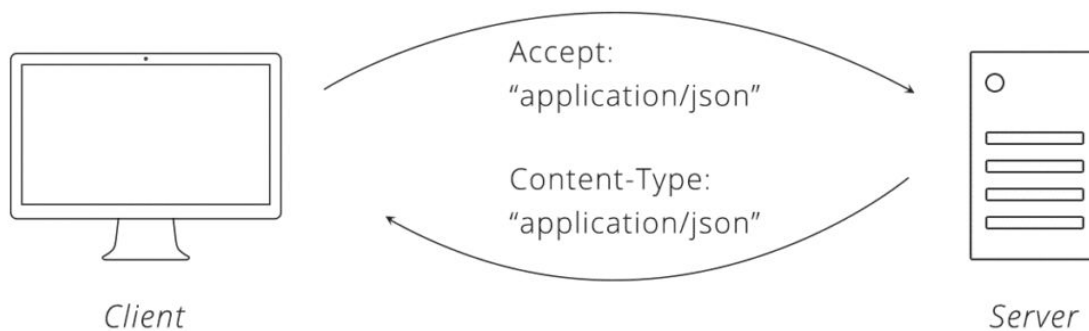
↓ ↓ ↓

```
<file> files/59017ab52cb10.epub </file>
```


3. Định dạng dữ liệu được sử dụng như thế nào trong HTTP.

Quay lại phần 2, phần header có chức năng lưu những thông tin mà người dùng không biết, trong đó có 1 thành phần xác định format của data: *Content-Type*

Khi client gửi *Content-Type* trong header của request, nó đang nói với server rằng dữ liệu trong phần body của request là được định dạng theo kiểu đó. Khi client muốn gửi JSON nó sẽ đặt *Content-Type* là "application/json". Khi bắt đầu nhận request, server sẽ check cái *Content-Type* đầu tiên và như thế nó biết cách đọc dữ liệu trong body. Ngược lại, khi server gửi lại client 1 response, nó cũng gửi lại *Content-Type* để cho client biết cách đọc body của response.



Đôi khi client chỉ đọc được 1 loại định dạng, ví dụ là JSON mà server lại trả về XML thì client sẽ bị lỗi. Do đó, 1 thành phần khác ở trong header là *Accept* sẽ giúp client xử lý vấn đề trên bằng cách nói luôn với server loại nó có thể đọc được. Ví dụ : *Accept* : "application/json" . Chốt lại: dựa vào 2 thành phần *Content-Type* và *Accept*, client và server có thể hiểu và làm việc một cách chính xác.

IV. Giới thiệu chung về Postman

1. Ưu, nhược điểm của Postman

Postman là 1 công cụ để test API của cty Postdot Technologies được bắt đầu phát triển từ năm 2012. Hiện tại Postman có 3 phiên bản: Postman, Postman Pro (2016) và Postman Enterprise (2017). Mình mới sử dụng Postman phiên bản free nên mình chỉ giới thiệu phần này.

Ưu điểm:

- Dễ sử dụng, hỗ trợ cả chạy bằng UI và non-UI.
- Hỗ trợ viết code cho assert tự động bằng Javascript.
- Hỗ trợ cả RESTful services và SOAP services.
- Có chức năng tạo API document.

Nhược điểm:

- Những bản tính phí mới hỗ trợ những tính năng advance: Làm việc theo team, support trực tiếp...

2. Cài đặt Postman

Download tại địa chỉ:



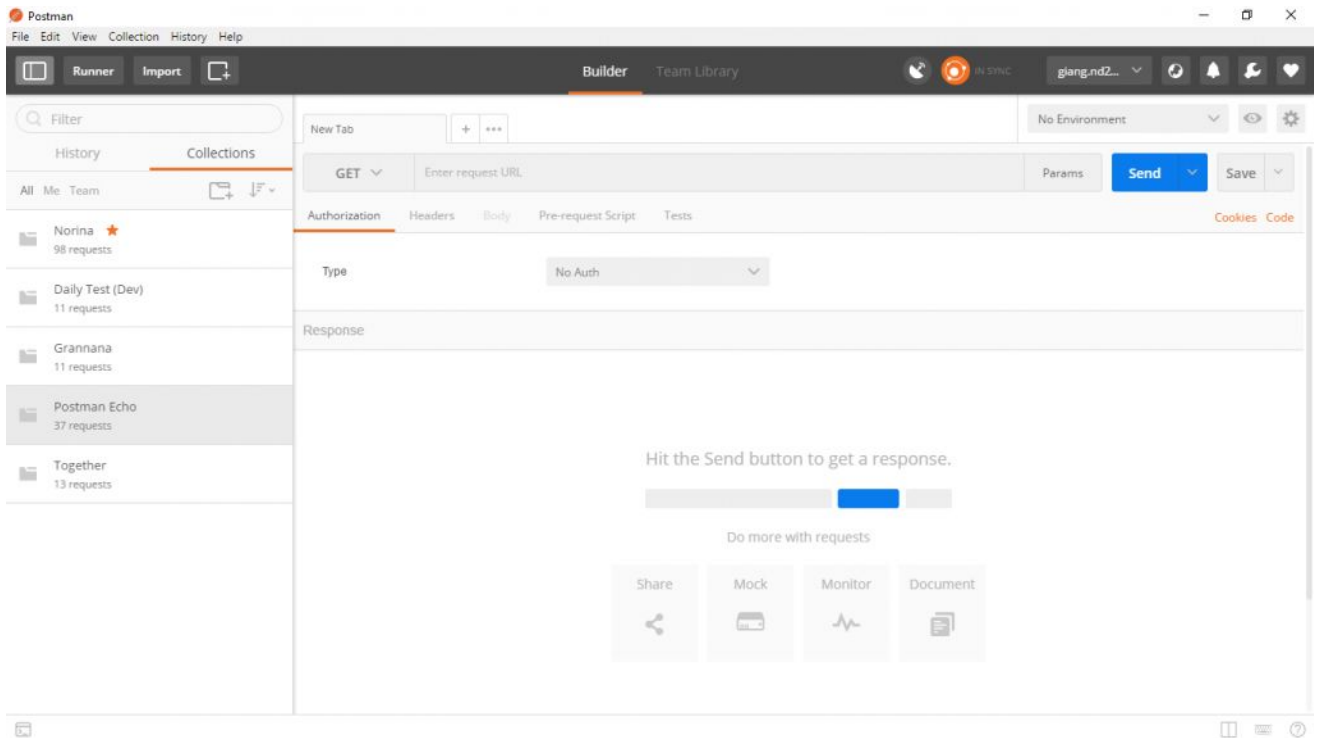
<https://www.getpostman.com/>



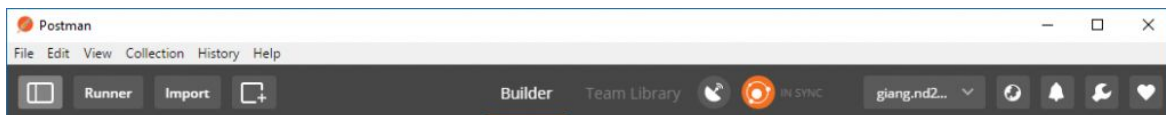
Sau đó, cài đặt như 1 phần mềm bình thường.

3. Các thành phần chính của Postman

Giao diện của Postman:

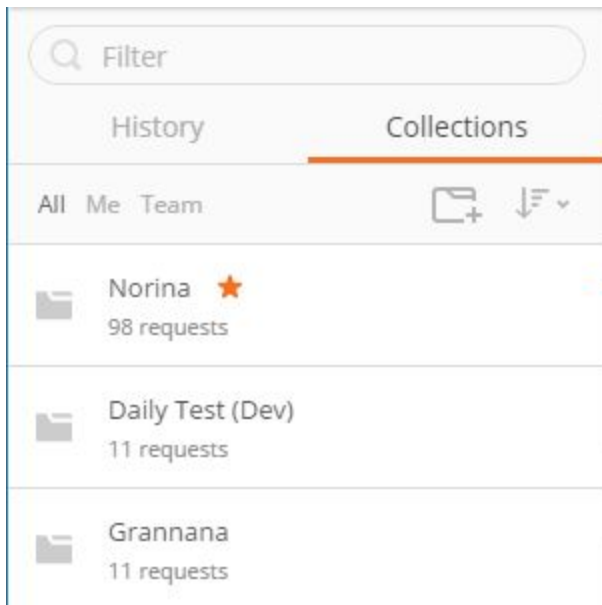


- **Settings:** chứa các thông tin về cài đặt chung.

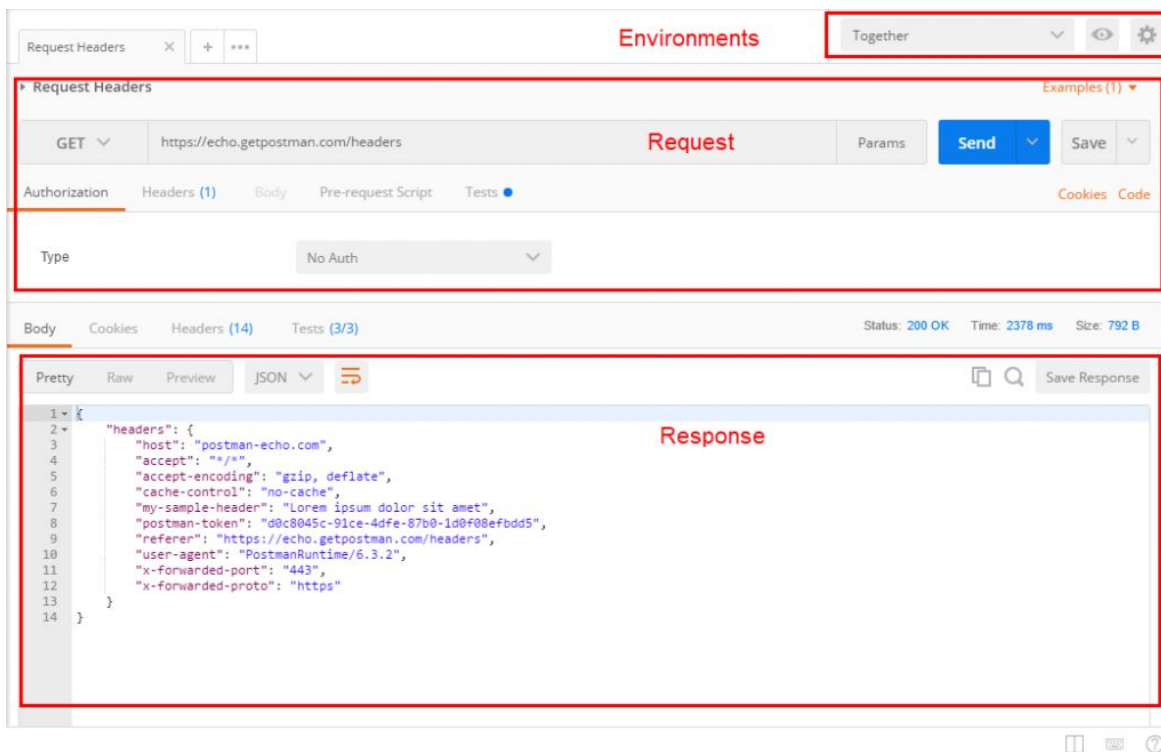


- Thông tin Account: dùng để Login, logout và sync data.
- Settings tùy chỉnh: themes, shortcut, format...
- Import data từ ngoài vào

- **Collections:** Lưu trữ thông tin của các API theo folder hoặc theo thời gian.



- **API content:** hiển thị nội dung chi tiết API và các phần hỗ trợ giúp thực hiện test API. Đây là phần mà tester phải làm việc nhiều nhất.



Trong phần này gồm có 3 thành phần chính:

- *Enviroments*: Chứa các thông tin môi trường. Ví dụ: mình làm 1 dự án nhưng có 3 môi trường khác nhau: dev, staging và product. Có phần này, mình có thể nhanh chóng đổi sang môi trường cần test mà không phải mất công đổi URL của từng request. (Cái này sẽ được nói rõ hơn ở những chương sau)
- *Request*: Phần chứa các thông tin chính của API, có thể đọc lại chương 2.
- *Reponse*: Chứa các thông tin trả về sau khi Send Request.

V. Cách tạo Request

Khi làm việc với API, chúng ta chỉ làm việc với 2 dạng API chính là GET và POST.

– *GET*: Yêu cầu server đưa lại resource: Hãy tưởng tượng ra cái cảnh vào fb, tay vuốt new feeds.

– *POST*: Yêu cầu server cho tạo ra 1 resource mới. Ví dụ: đăng ký 1 chuyến đi ở GrabBike.

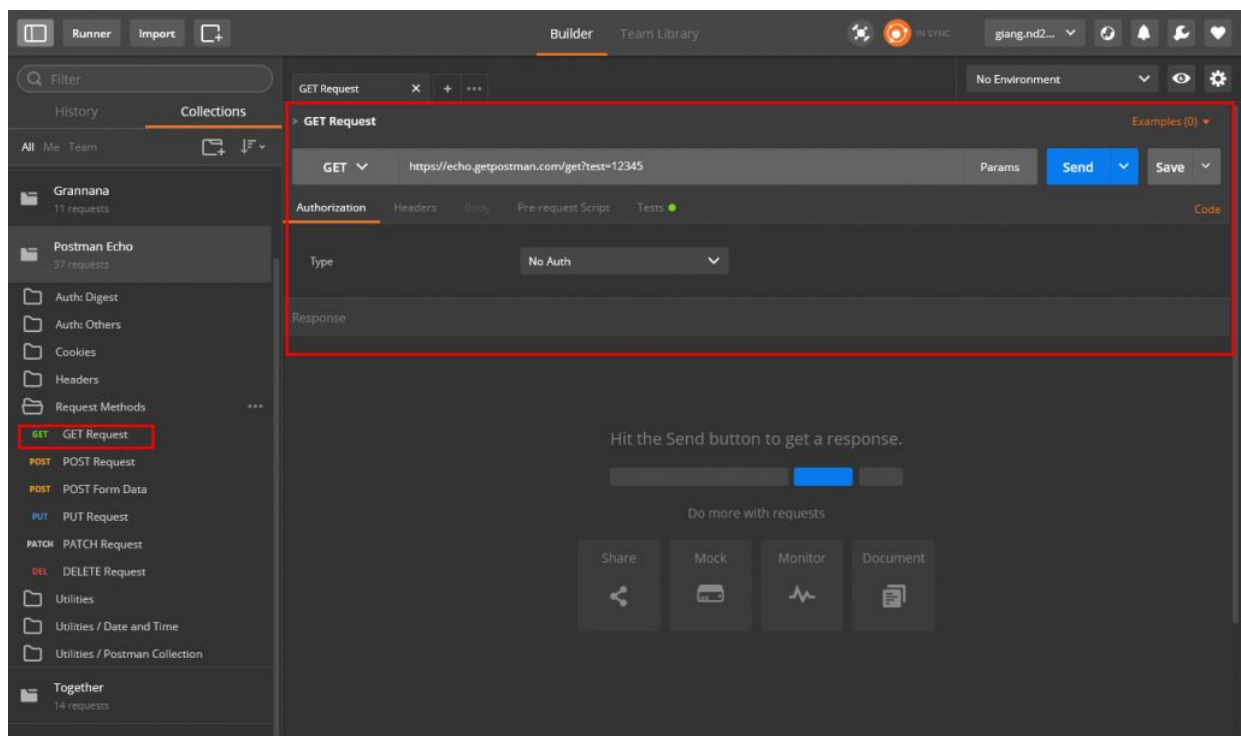
Và một request gồm có 4 thành phần:

1. URL
2. Method
3. Headers
4. Body

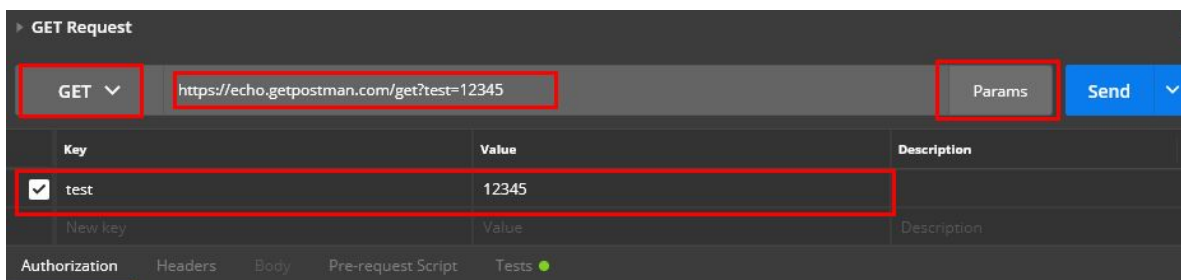
Khi vào dự án thật, những thông tin trên thì bạn lấy ở đâu, ở ông developer nhé. Muốn test được API thì phải có API documents. Cái này tùy công ty sẽ có chuẩn và mẫu riêng, nhưng mà nhìn chung thì phải cung cấp đủ các thông tin sau: Tên API, mục đích sử dụng, Method, URL, Params, Sample Request, Sample Response.

1. Tạo request GET

Mình xin phép dùng luôn API mẫu của Postman cung cấp..

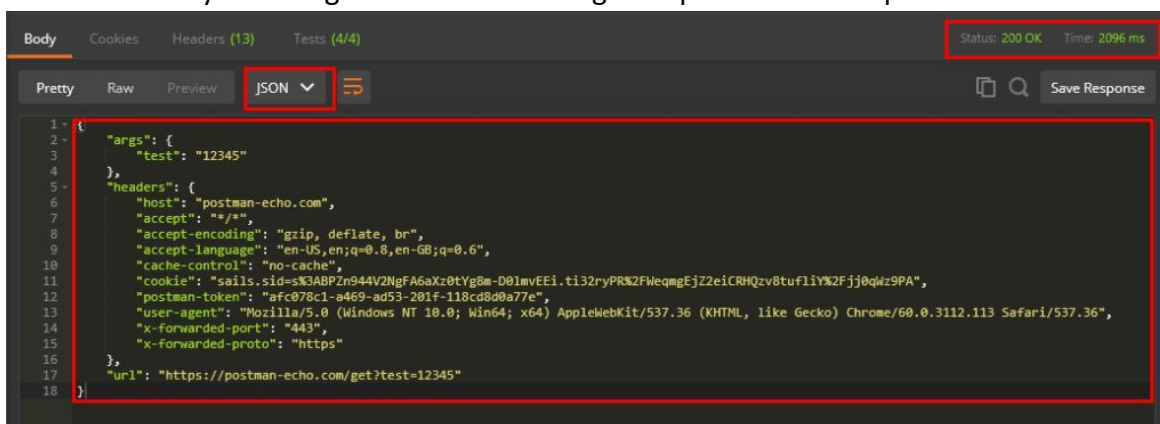


1. *URL*: <https://echo.getpostman.com/get>
2. *Method*: GET
3. *Headers*: Không cần điền gì cả
4. *Body*: Phương thức GET không có body, các bạn phải điền tham số vào **Params**



Lưu ý: Tất cả các Params truyền vào phải chính xác, không được để thừa 1 khoảng trống hay xuống dòng.

Sau khi điền đầy đủ thông tin thì ấn SEND để gửi request và chờ response trả về.



Thông tin trả về sẽ có mấy điểm cần quan tâm:

1. Định dạng dữ liệu trả về: thông thường là json và nên để chế độ Pretty để cho dễ nhìn.

2. Nội dung dữ liệu: Đây là phần bạn phải kiểm tra.

– Bạn so sánh với cái Sample Response ở API docs để xem cấu trúc trả về đã đúng hay chưa.

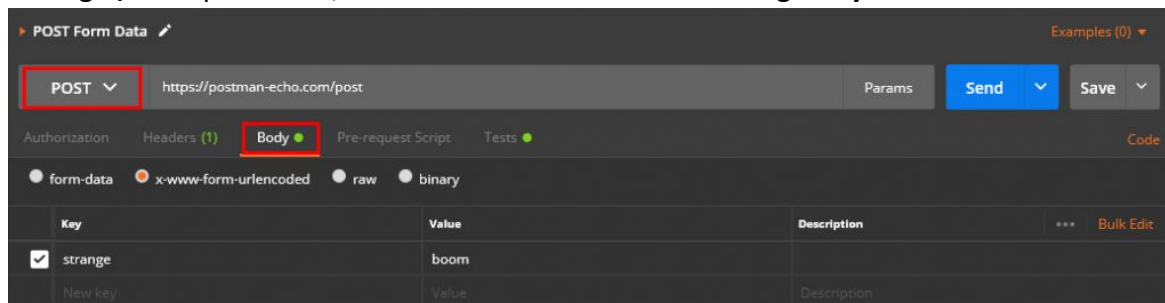
– Value của từng key đã đúng chưa, so sánh với nội dung trong DB. (không có DB là không làm được API testing đâu).

3. Trạng thái của API (status) và thời gian trả về.

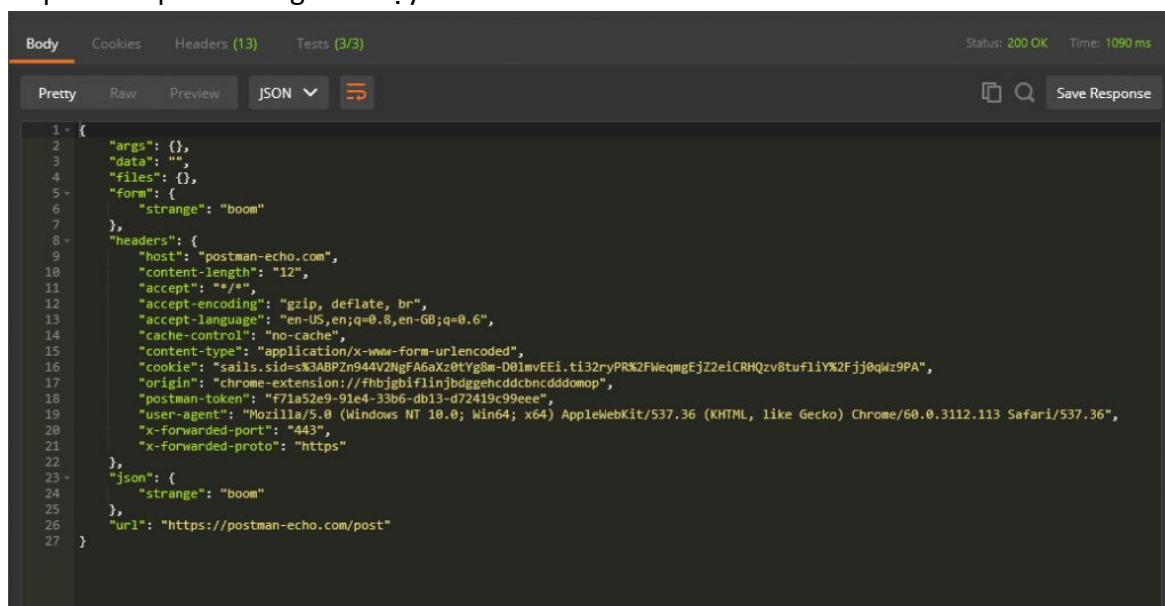
Xin được lưu ý, thời gian chạy API bằng Postman luôn ngắn hơn thời gian test trên giao diện Mobile vì nhiều lý do: đường truyền internet ở máy tính ổn định hơn wifi, và sau khi nhận response thì Mobile phải chạy code khởi tạo giao diện để hiển thị.

II. Tạo request POST

Tương tự như phần trên, chỉ khác là điền tham số vào trong **body**.



Và phần response cũng như vậy



Nếu các bạn chưa được tham gia dự án để có docs API và muốn thử nghiệm test các API thật. Try this: <https://any-api.com>

VI. Collections trong Postman

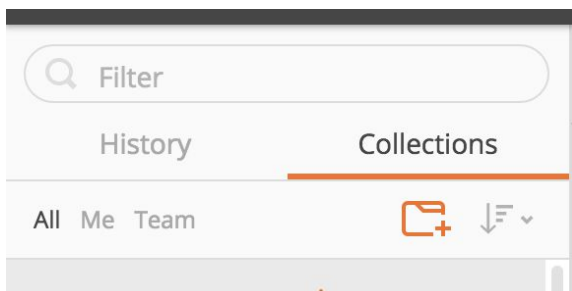
Lúc bắt đầu viết về postman, mình không nghĩ là sẽ viết về phần này nhưng mà sau khi nhìn thấy 1 vài đồng chí developers công ty mình để cái đồng request hỗn độn, mất 5p mới tìm ra cái request đã dùng hôm trước ở đâu. Mình phải nghĩ lại là “Không phải ai cũng để ý rằng có phần collections này ở trên đời” và quyết định viết vì biết đâu có ai đó không biết. :))

Hiểu nôm na, nó chính là Folder, giúp đóng gói những request vào chung 1 chỗ. Ở thế không dùng có được không? Câu trả lời là ĐƯỢC, tuy nhiên sẽ gặp phải 1 số vấn đề sau đây:

- Sẽ phải dùng History để tìm lại những request đã dùng, tương tự như bạn suốt ngày phải lục lọi phần History của Chrome, trong khi chỉ cần 1 động tác bookmark lại là xong.
- Không dùng được chức năng tạo API documents tự động mà Postman cung cấp
- Không thể dùng được chức năng Runner, giúp chạy liên tục các Request.

1. Cách tạo Collection.

- Click vào button [tạo collection] bên sidebar



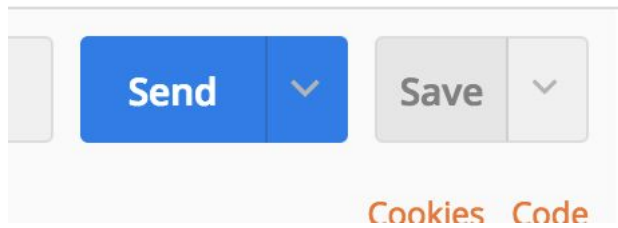
- Điền tên và mô tả (không bắt buộc) collection đó.



- Lưu request vào Collection.

BƯỚC 1. Tạo ra 1 new Request (Như phần trước)

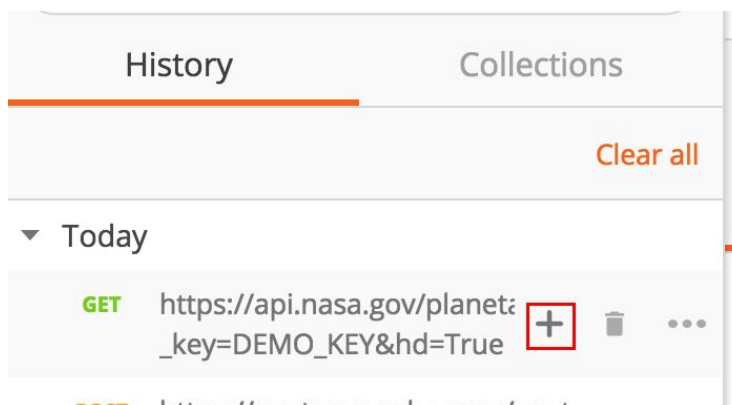
BƯỚC 2. Ấn nút Save



BƯỚC 3. Chọn Collection cần lưu và Save tiếp.

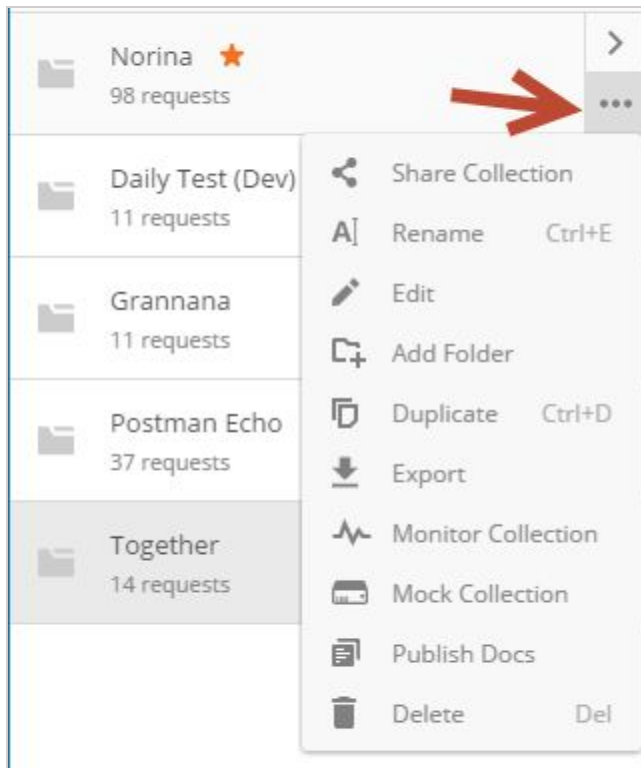
Note: Với những trường TH mà muốn add request từ bên History vào Collection.

BƯỚC 1. Click vào icon (+)



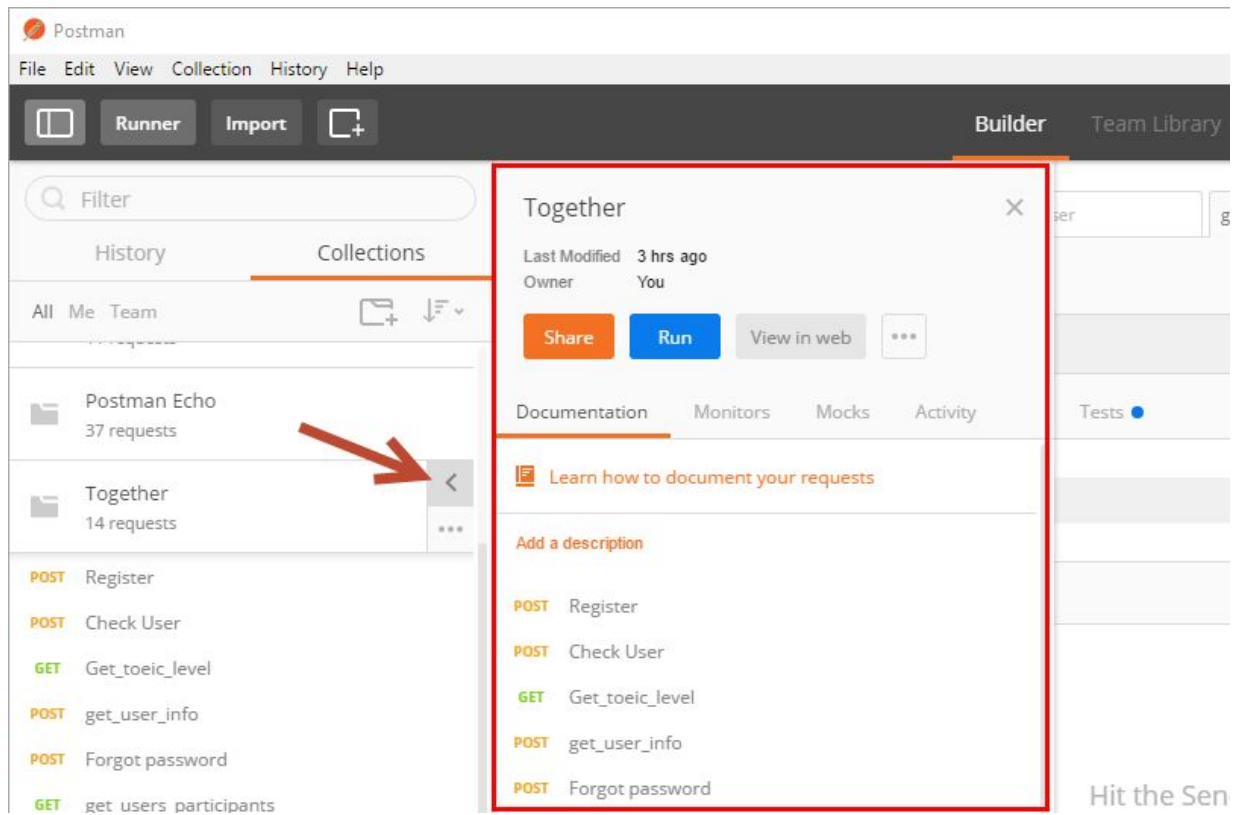
BƯỚC 2. Chọn Collection cần lưu và Save.

2. Các settings chính của 1 Collection.



- **Share collections:** tạo ra link để share với người khác collection (bị hạn chế bởi kiểu account).
- **Rename:** Đổi tên của collection.
- **Edit:** Sửa tên và mô tả của collection.
- **Add Folder:** tạo thêm collection mới bên trong Collection đó.
- **Duplicate:** nhân đôi collection đang có.
- **Export:** Xuất collection ra dạng file .json
- **Monitor Collection:** Dùng để test hiệu năng (bị hạn chế bởi kiểu account).
- **Mock Collection:** giúp giả lập các API sử dụng chức năng Example mà postman hỗ trợ. (bị hạn chế bởi kiểu account).
- **Publish Docs:** Tạo ra API Docs định dạng HTML.
- **Delete:** Xóa Collection.

Ngoài cách trên, có thể xem chi tiết Collection bằng cách click vào mũi tên [>] .



VII. Cách sử dụng Environments

Phần này đã có nhiều người viết bài hướng dẫn tiếng việt nhưng còn thiếu 1 số phần nên mình viết lại.

Chức năng chính của Environment là 1 chỗ lưu “biến” giống như “biến” trong code để mình có thể tái sử dụng ở nhiều nơi.

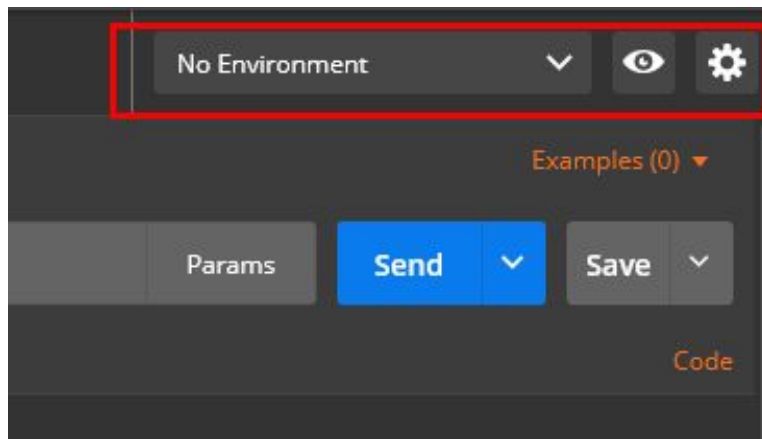
Ứng dụng:

- Nhanh chóng chuyển qua lại giữa các môi trường Dev và Product mà không cần tạo lại các request mới vì phải thay đổi lại URL.
- Giúp lưu lại giá trị của response API trước để điền vào API sau. (Phần này có kết hợp với phần Pre-request và Tests, sẽ được giới thiệu ở các phần tiếp theo).
- Không phải sửa giá trị của các tham số quá nhiều lần.

Ở Postman sẽ chia làm 2 loại Environments: Local và Global

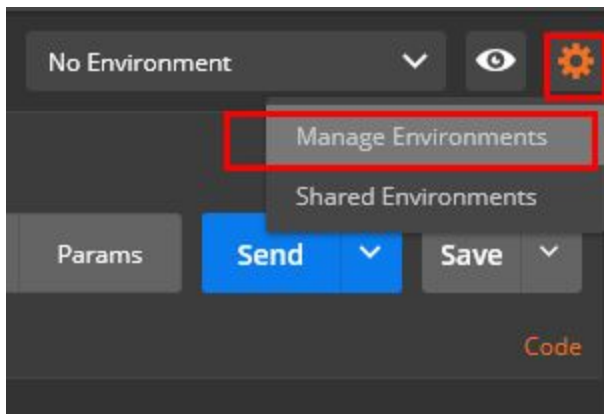
- Local: Phạm vi ảnh hưởng chỉ có khi chọn đúng Enviroments.
- Global: Phạm vi ảnh hưởng đến toàn bộ các project có trong Postman, nhưng nếu có 2 biến cùng tên ở Local và Global thì sẽ ưu tiên lấy Local.

Vị trí của Environment trong khung làm việc của postman.

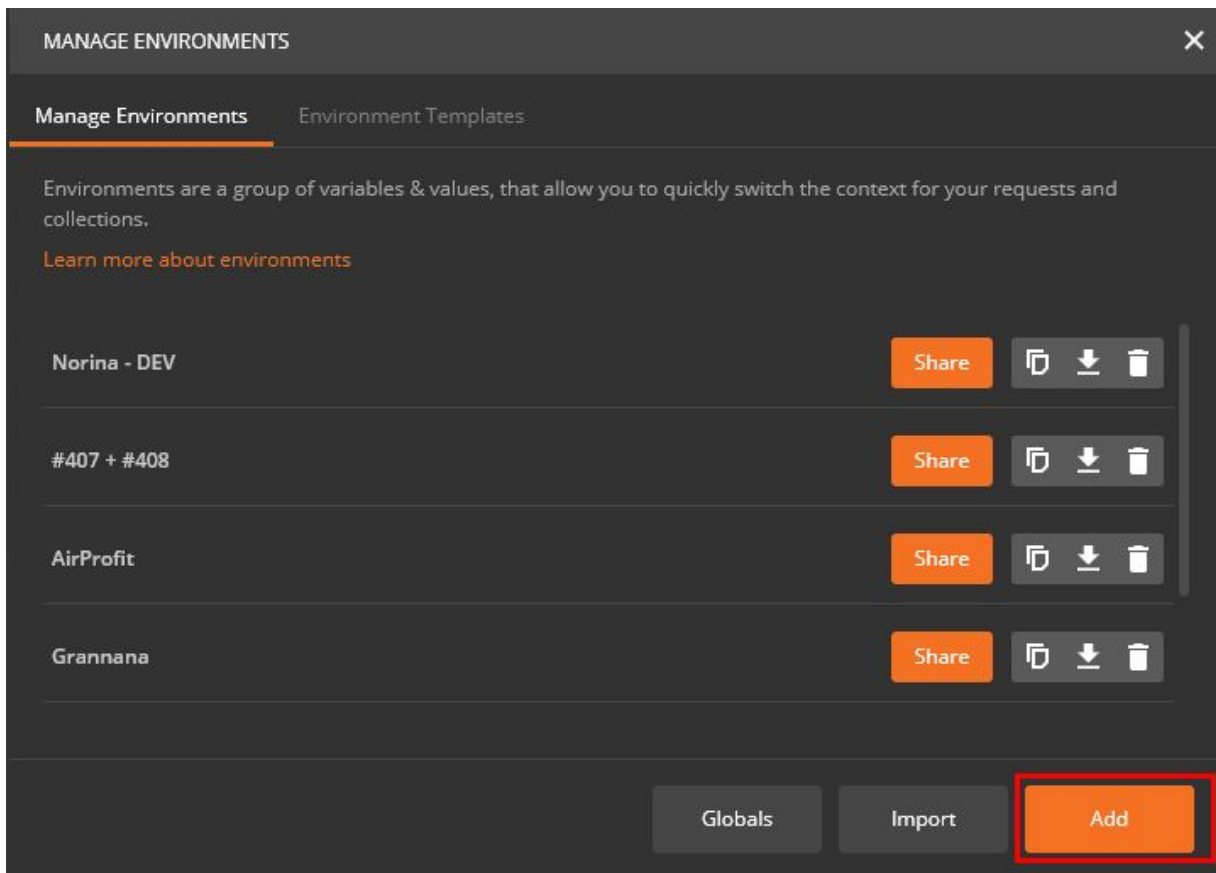


Tạo 1 Environment

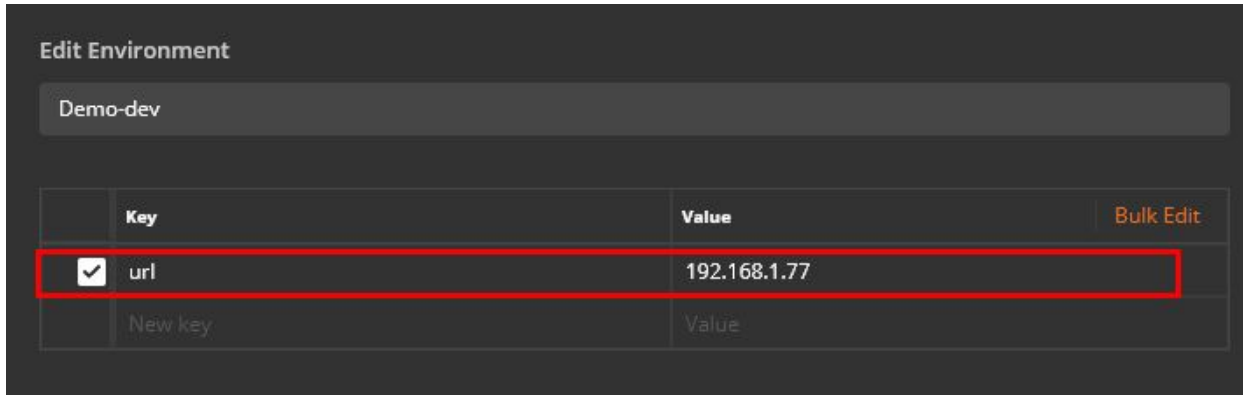
BƯỚC 1: MỞ Manage Environments



BƯỚC 2: Add thêm Environment mới



BƯỚC 3: Điền tên của Environment, tên và giá trị của biến.
Ở đây, mình lấy ví dụ với biến `url` có giá trị là `192.168.1.77`

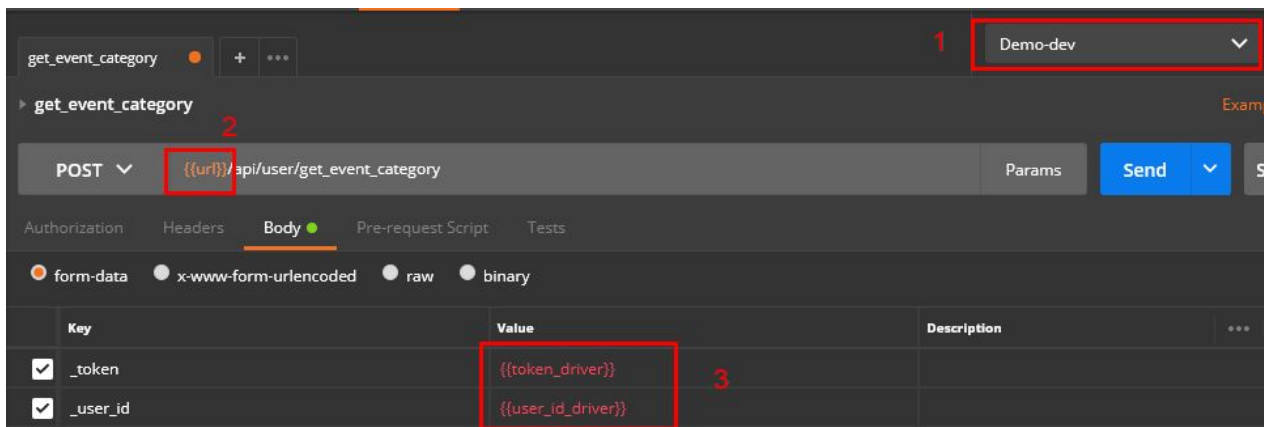


Dấu checkbox thể hiện rằng: có active cái biến đó hay không. Trong ví dụ trên: mình đang active biến url và có thể sử dụng biến này trong môi trường Demo-dev. Thế là bạn đã tạo được 1 môi trường cho việc test API rồi đấy. =))) đùa chút thôi. Dưới đây là cách sử dụng các biến.

Có mấy câu hỏi như sau:

1. Lấy giá trị của các biến trên như thế nào trong request?

Trả lời: Chỉ cần viết theo cú pháp {{tên_biến}}: ví dụ: {{url}}



Như trong hình:

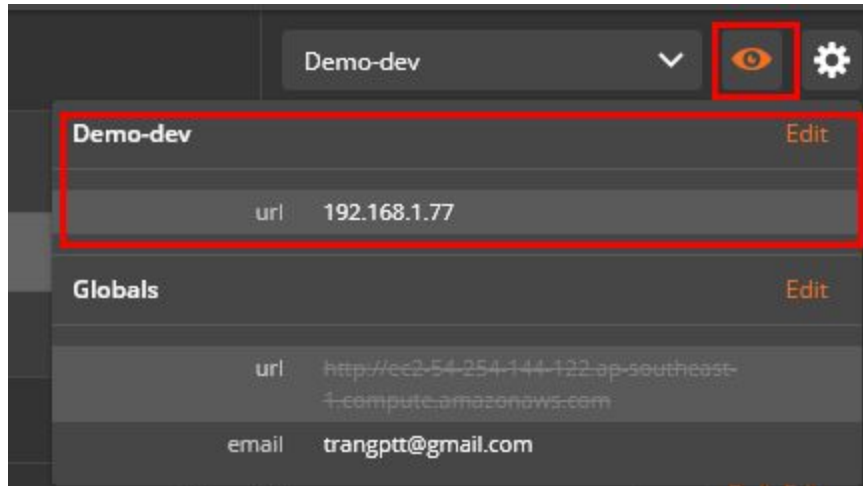
1. Chỗ chuyển đổi qua lại giữa các Environment
2. Cách lấy giá trị biến. Lấy đúng sẽ có màu da cam, đưa chuột hover vào thì hiển thị giá trị của biến.
3. Nếu tên biến có màu đỏ có nghĩa là không có biến này trong Environment, chuyện này thường xảy ra khi chuyển đổi qua lại giữa các môi trường của các dự án khác nhau, hoặc đã inactive cái biến đó.

2. Làm thế nào để biết được giá trị của các biến trong 1 environment nào đó.

Trả lời:

Cách 1: vào lại Manage Environment giống như việc Add Environment

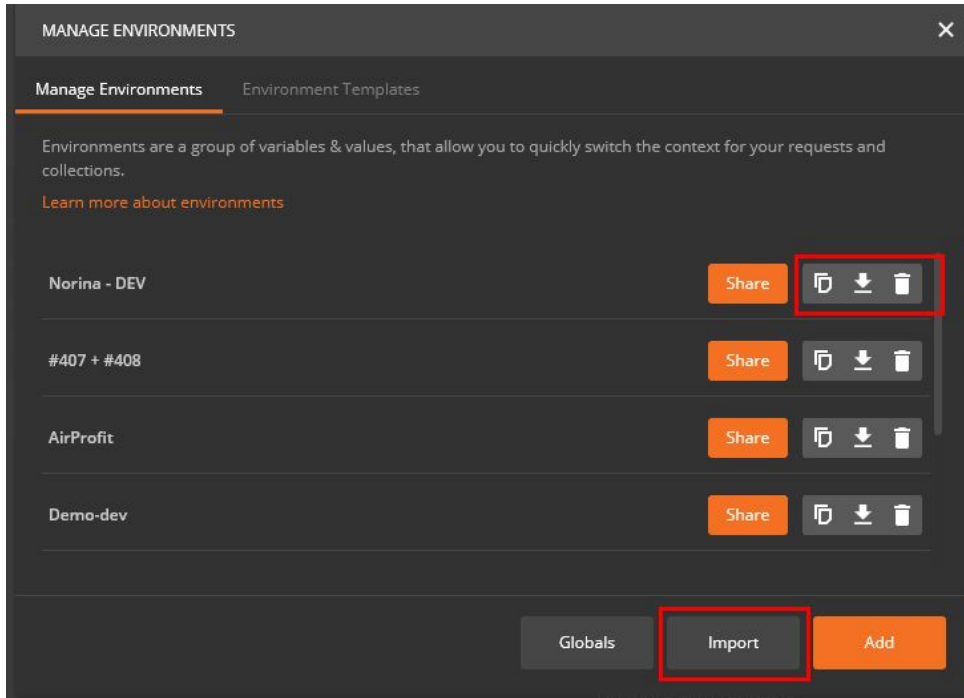
Cách 2. Click vào icon con mắt



Muốn sửa thông tin của Environment bạn có thể click vào nút *Edit* để sửa.

3. Có thể làm những gì với 1 Environment?

Trả lời: Postman cung cấp những chức năng đơn giản cho 1 Environment như: import – export, duplicate, add, edit, delete.

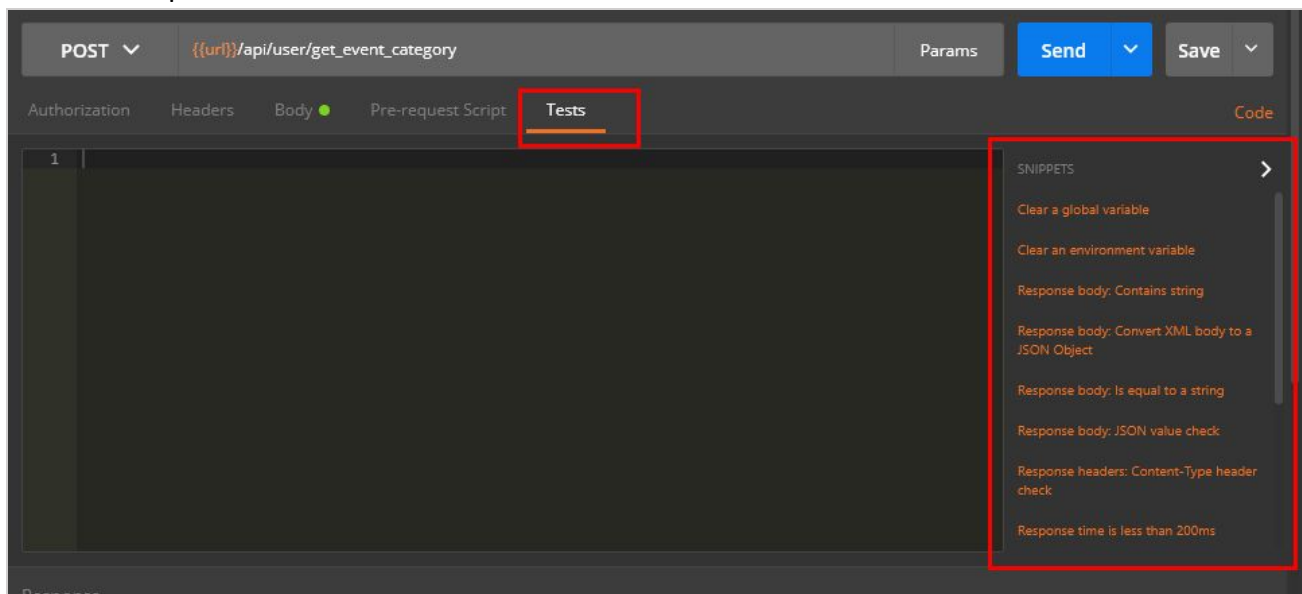


VIII. Test Response

Test response là tính năng đặc biệt quan trọng với những người test API. Làm sao có thể suốt ngày run từng cái request rồi check từng kết quả trả về một cách thủ công được, phải có cách gì nhanh hơn chứ.

Phần này cung cấp 2 tính năng cực hay giúp người test đẩy nhanh được tốc độ test API.

1. Check tự động kết quả trả về của từng field với 1, 2 dòng code, rất dễ, không cần biết code cũng làm được.
2. Lưu giá trị của Response thành biến trong Environment để tiếp tục truyền vào param của API tiếp theo.



Postman cung cấp một khung làm việc để ta có thể làm việc, chỉ hỗ trợ ngôn ngữ Javascript thuần, không hỗ trợ jquery hay các thứ khác nhé.

→ Nếu muốn biết chắc code mình chạy đúng, hãy viết trước ở ngoài nhé.

Phần ở bên phải là tập hợp những cú pháp Postman cung cấp sẵn cho người dùng, khỏi cần phải nghĩ. Ok vào bài toán cụ thể nhé.

Bài toán 1:

Tôi có 1 API login, tôi muốn biết là sau khi login vào xong, user_id của tôi trả về có đúng hay không.

Bước 1: Chạy thử API 1 lần để lấy được cấu trúc Response của API.

```
1 {
2   "error_code": 0,
3   "error_msg": "",
4   "data": {
5     "token": "02ec06d1519164d16f82d7d2642ebf70",
6     "user": {
7       "coupons": [],
8       "coupons_seen": 0,
9       "profile": {
10        "id": "401",
11        "email": "norina.hn2017@gmail.com",
12        "name": "dzung",
13        "user_type": "2",
14        "phone_number": "+84989410526",
15        "password": "156daf415327ed5c5372cec353314cda",
```

Ta có thể thấy user_id nằm ở vị trí: *root > data > user > profile > id* và trong trường hợp này id của user này là 401.

Bước 2: Viết Test

```
1 var jsonData = JSON.parse(responseBody);
2
3
4 tests["Check user id"] = jsonData.data.user.profile.id === 401;
```

Code:

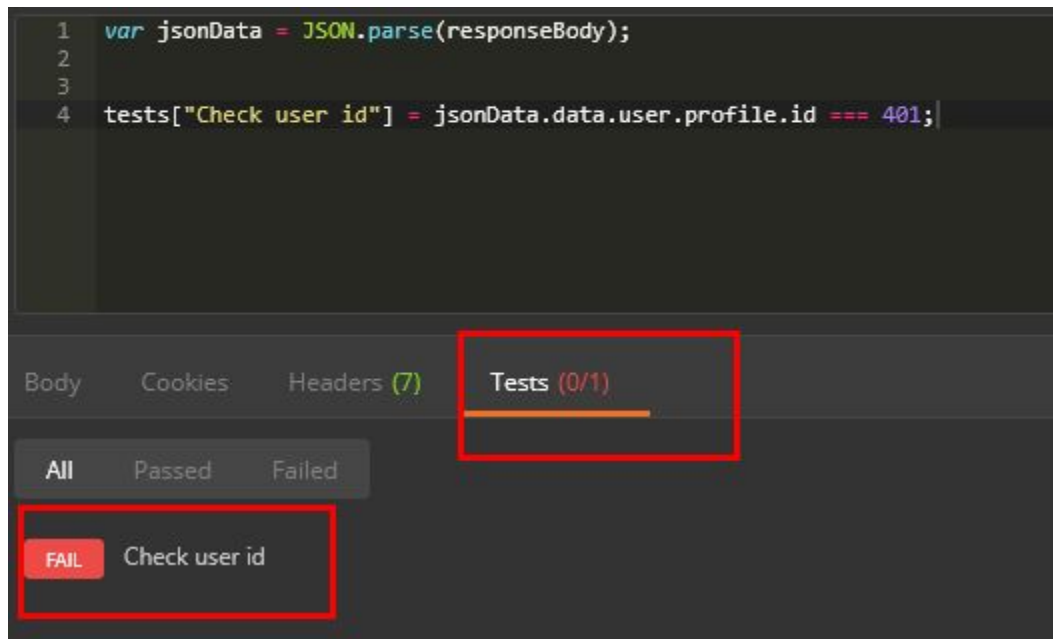
```
var jsonData = JSON.parse(responseBody);
tests["Check user id"] = jsonData.data.user.profile.id === 401;
```

1: Parse cái Response trả về và lưu vào biến "jsonData" → cái này chính là root đã viết ở trên.

2: Test xem user_id có bằng 401 không.

Cách lấy giá trị hoàn toàn giống như trong Javascript thôi. Bạn có thể đọc thêm ở đây về cách xử lý Object của Json. <http://goessner.net/articles/JsonPath/>

Bước 3: Sau khi viết xong Test thì các bạn run Request lại rồi ngó xem phần Test của mình có đúng không.



Theo kết quả thì Test của mình đang Fail, á ù, mình viết đúng rồi mà nhỉ.

Đây là điểm mà các bạn mới làm sẽ hay gặp, đó là vấn đề ngay tại cái công cụ giúp mình test. Trong trường hợp này, đó là giá trị trả về nó là 1 String, mình không thể so sánh String và Int được. Hãy ngó lại cái ảnh phía trên, bạn sẽ thấy id là "401", ta chỉ cần thêm dấu nháy kép vào lại phần test của mình là xong.

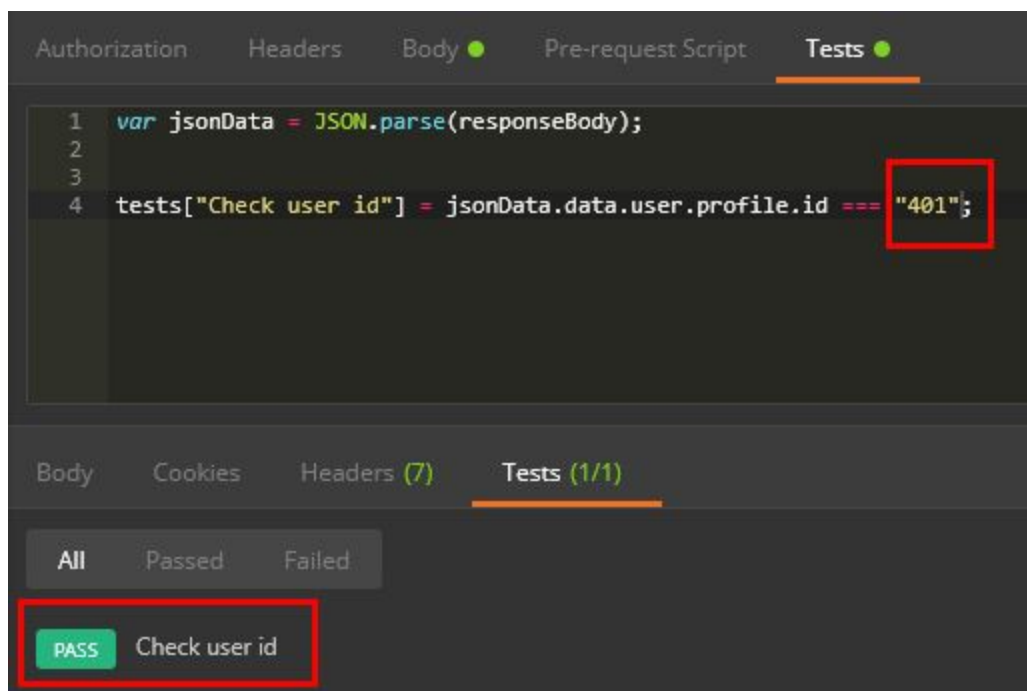
Code đúng sẽ là:

```
var jsonData = JSON.parse(responseBody);
tests["Check user id"] = jsonData.data.user.profile.id === "401";
```

UPDATE:

Postman thay đổi 1 số built-in functions nhưng các functions cũ vẫn chạy ngon, nên nếu bạn muốn dùng các functions mới thì đây:

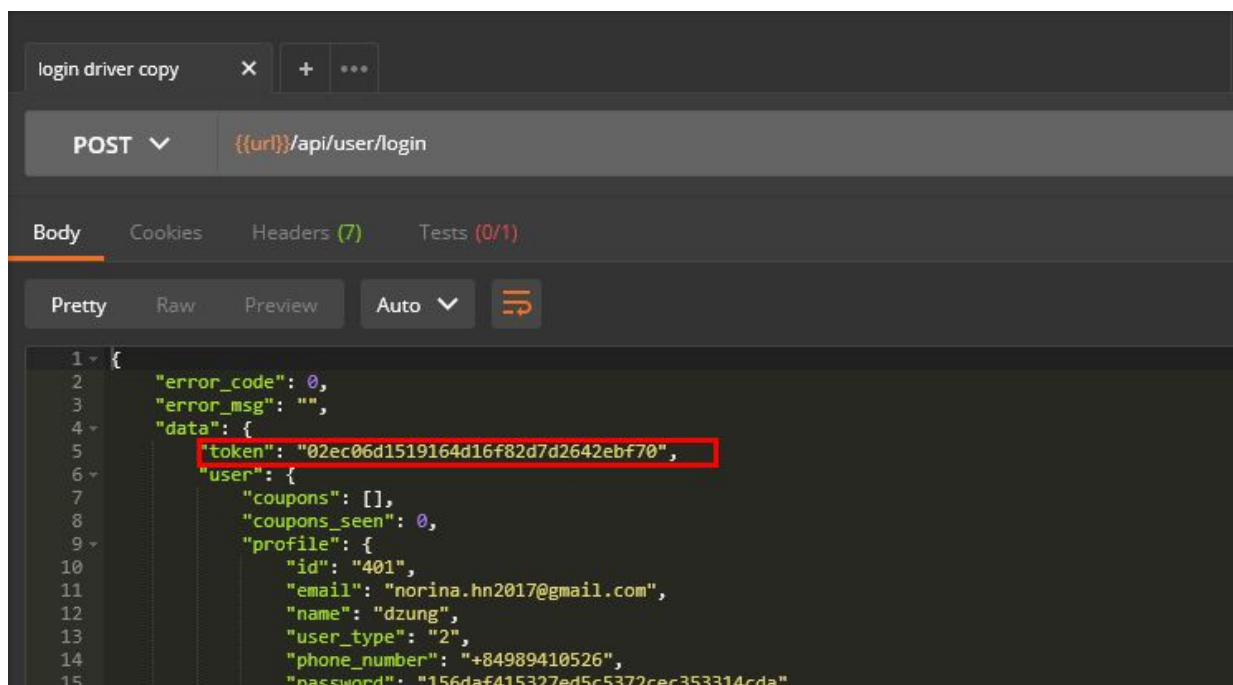
```
var jsonData = pm.response.json();
pm.test("Check user_id", function () {
  pm.expect(jsonData.data.user.profile.id).to.eql("401");
});
```



Bài toán 2:

Tôi có 1 API login, sau khi login tôi muốn lưu lại giá trị của token để làm data cho những API tiếp theo.

Vẫn dùng ví dụ ở trên, ta sẽ thấy vị trí của token là: root > data > token.



Ta sẽ viết thêm 1 code vào phía dưới 2 dòng code mình đã viết ở trên:

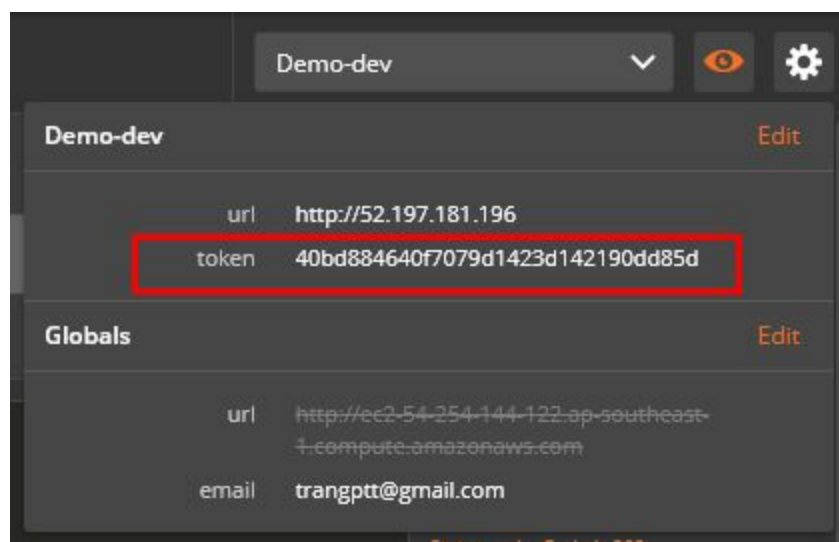
```
var jsonData = JSON.parse(responseBody);
tests["Check user id"] = jsonData.data.user.profile.id === "401";
postman.setEnvironmentVariable("token", jsonData.data.token);
```

UPDATE:

Postman thay đổi 1 số built-in functions nhưng các functions cũ vẫn chạy ngon, nên nếu bạn muốn dùng các functions mới thì đây:

```
var jsonData = pm.response.json();
pm.test("Check user_id", function () {
  pm.expect(jsonData.data.user.profile.id).to.eql("401");
});
pm.environment.set("token", jsonData.data.token);
```

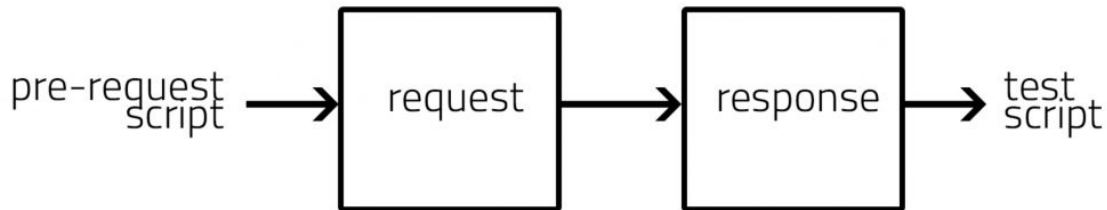
Sau đó, ta chỉ run lại rồi kiểm tra trong Environment thôi.



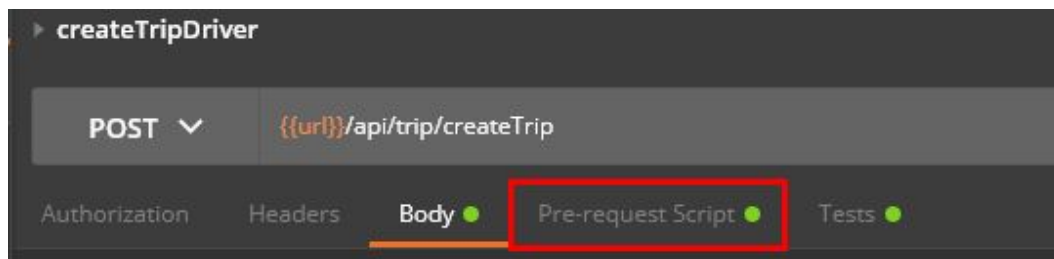
Khi đã lưu được biến vào trong Environment rồi thì phần gán biến vào request sẽ giống với phần mình đã hướng dẫn ở phần trước.

IX. Sử dụng Pre-request Script

Phần trước mình đã viết về test Response, nay mình sẽ tiếp tục viết về Pre-request. Dưới đây là các bước khi gửi 1 request. Phần Pre-request sẽ là phần Postman sẽ xử lý trước khi thực hiện gửi request, và phần test script để xử lý response được trả về.



Vậy Pre-request có thể làm được việc gì? Nó xử lý có mỗi 1 phần thôi đó là tạo dữ liệu (biến) để truyền vào param trong request.



Có 1 chức năng thôi nhưng cực kỳ hữu ích trong các trường hợp khác nhau.

Ví dụ: Mình có 1 API tạo ra 1 chuyến đi giống kiểu grab và uber, gồm các param dưới đây.

- `_user_id`
- `_token`
- `Source`
- `Destination`
- `Departure_datetime`

Thêm 1 chút requirement:

`_user_id` : có được sau khi login

`_token` : có được sau khi login

`Source`: Địa điểm xuất phát

`Destination`: Địa điểm đến

`Departure_datetime`: Thời gian xuất phát của chuyến đi luôn lớn hơn thời điểm hiện tại. (Lưu ý: thời gian của Nhật sớm hơn VN 2 tiếng, nếu VN là 15h thì ở Nhật là 17h).

Thời gian đầu khi mình test API này, thì mình chỉ tạo được 2 biến `_user_id` và `_token` sau khi run API login rồi sử dụng phần Test Script để lưu response lại. Các bạn có thể đọc lại phần

trước để hiểu thêm.

	Key	Value
<input checked="" type="checkbox"/>	_user_id	{{user_id_driver}}
<input checked="" type="checkbox"/>	_token	{{token_driver}}

Còn 3 biến *Source*, *Destination*, *Departure_datetime* mình phải tự nhập lại bằng tay mỗi lần run API → rất mất công sức, nếu không sửa các param trên thì API của mình sẽ sai.

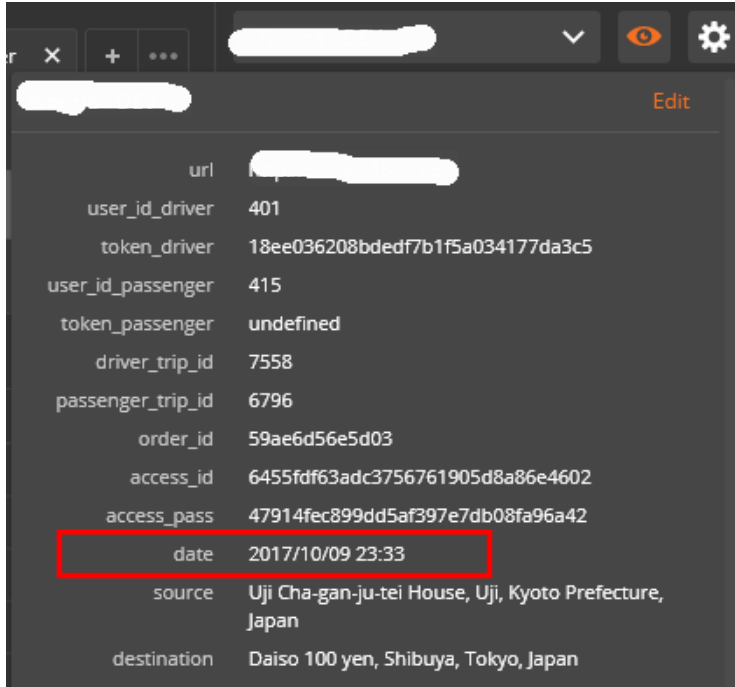
Giả sử hôm nay ngày 09/10/2017, mình chạy API đúng, nhưng sang ngày mai (10/10/2017) mình lại phải đi sửa 1 đồng API vì sai ngày. T_T Mình đã bị như thế trong 1 thời gian, cho đến khi mình biết cách sử dụng Pre-Request.

Xử lý *Departure_datetime*

Mình viết ra 1 hàm `getToday` để đảm bảo lúc nào run Testcase cũng trả về ngày hôm nay mà không cần phải maintain sửa lại API.

```
1 //create date today
2 var today = new Date();
3 var dd = today.getDate();
4 var mm = today.getMonth()+1; //January is 0!
5 var yyyy = today.getFullYear();
6
7 - if(dd<10) {
8     dd = '0'+dd;
9 }
10 - if(mm<10) {
11     mm = '0'+mm;
12 }
13 var second = Math.floor((Math.random() * 60) + 1);
14 - if (second < 10){
15     second = '0'+second;
16 }
17 today = yyyy + '/' + mm + '/' + dd + ' 23:' +second;
18
19 postman.setEnvironmentVariable("date", today);
20
```

Các bạn có thể sẽ nghĩ là mình giỏi tự nghĩ ra cái hàm đấy :))) Không phải đâu, các bạn chỉ cần google “Method `getToday` javascript”. Sau đó, mình chỉ sửa code 1 chút để có thể lấy được giờ random mà mình muốn. Cuối cùng là lưu biến đó vào Environment là xong.



Xử lý Source, Destination

Về bản chất, cả 2 param này đều điền giá trị là địa điểm, nhưng mình không thể chỉ test với 2 địa điểm được, nó sẽ tạo ra 1 đồng các chuyến đi giống hết điểm đến và điểm đi. Mình cần 1 lượng lớn các địa điểm rồi mình sẽ lấy random cho phần Source, Destination.

Mình tạo ra 1 cái array chứa toàn bộ địa điểm (mình export từ trong DB ra) rồi lấy random và lưu thành 2 biến trong Environment.

Ví dụ:

```
var myArray = [ {
  "source" : "Tokyo, Japan"
},
{
  "source" : "Gifu Prefecture, Japan"
},
{
  "source" : "Aichi Prefecture, Japan"
},
{
  "source" : "Tokyo Disney Resort, Urayasu, Chiba Prefecture, Japan"
}
]
var source = myArray[Math.floor(Math.random() * myArray.length)]['source'];
var destination = myArray[Math.floor(Math.random() * myArray.length)]['source'];
```

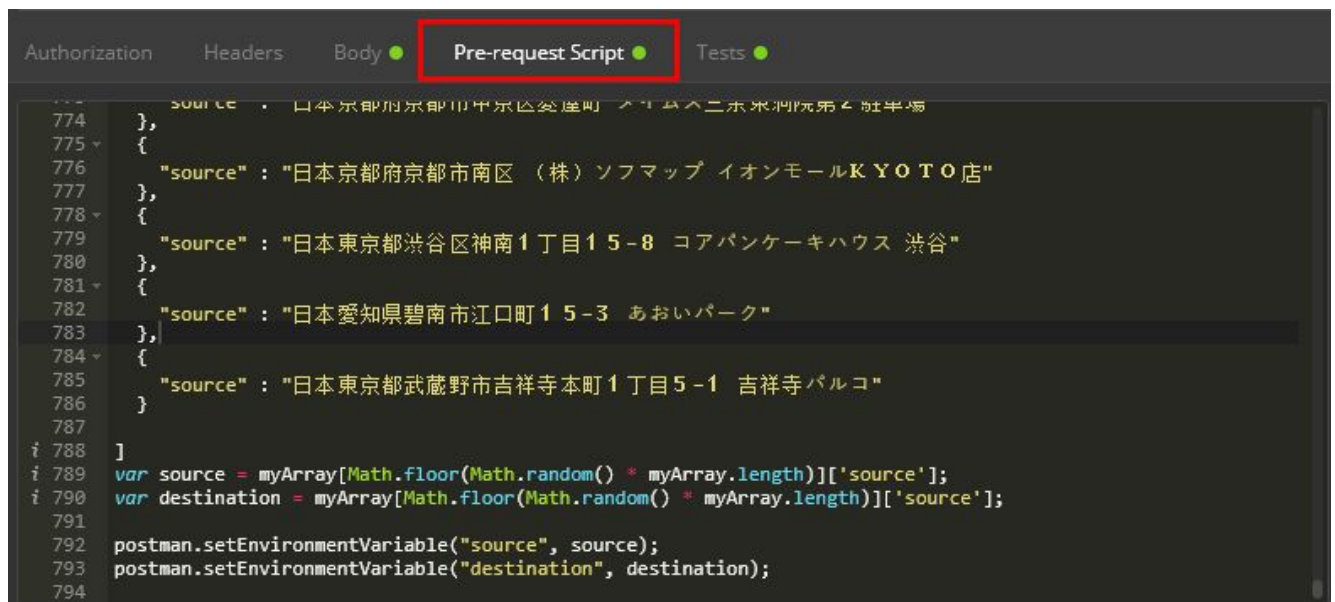
```
postman.setEnvironmentVariable("source", source);
postman.setEnvironmentVariable("destination", destination);
```

UPDATE:

Postman thay đổi 1 số built-in functions nhưng các functions cũ vẫn chạy ngon, nên nếu bạn muốn dùng các functions mới thì đây:

```
pm.environment.set("source", source);
pm.environment.set("destination", destination);
```

Do cái array của mình rất nhiều nên mình chỉ chụp hình minh họa thôi.



```
774     },
775     {
776       "source" : "日本京都府京都市南区 (株) ソフマップ イオンモールK Y O T O 店"
777     },
778     {
779       "source" : "日本東京都渋谷区神南1丁目15-8 コアパンケーキハウス 渋谷"
780     },
781     {
782       "source" : "日本愛知県碧南市江口町15-3 あおいパーク"
783     },
784     {
785       "source" : "日本東京都武蔵野市吉祥寺本町1丁目5-1 吉祥寺パルコ"
786     }
787   ]
788 ]
789 var source = myArray[Math.floor(Math.random() * myArray.length)]['source'];
790 var destination = myArray[Math.floor(Math.random() * myArray.length)]['source'];
791
792 postman.setEnvironmentVariable("source", source);
793 postman.setEnvironmentVariable("destination", destination);
794
```

Sau khi hoàn thành hết mình sẽ có 1 API với các tham số động, linh hoạt, chứ không phải là những API với dữ liệu cố định.

Authorization Headers **Body** Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

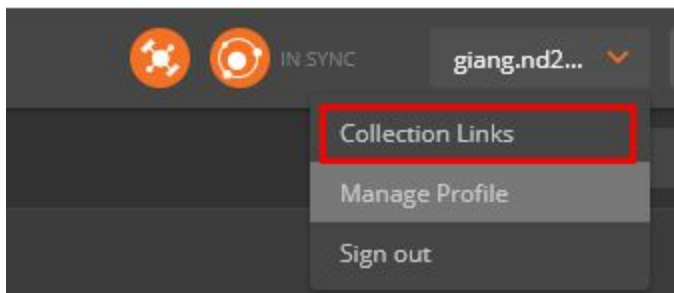
Key	Value
<input checked="" type="checkbox"/> _user_id	{{user_id_driver}}
<input checked="" type="checkbox"/> _token	{{token_driver}}
<input checked="" type="checkbox"/> source	{{source}}
<input checked="" type="checkbox"/> destination	{{destination}}
<input checked="" type="checkbox"/> departure_datetime	{{date}}

X. Xây dựng API Document

Postman ngoài việc cung cấp 1 công cụ test API còn hỗ trợ chúng ta làm API document cực kỳ chuyên nghiệp và dễ dàng. API document này có thể dùng chung cho cả team và khách hàng. Thông thường, API thường do Dev viết trên sheets nhưng đến 1 giai đoạn phát triển nào đó dev sẽ lười viết docs API hoặc API sửa nhiều, họ sẽ không nhớ để update những sửa chữa đấy.

API Document của Postman có điểm ưu việt hơn khi nó update luôn những gì mình thao tác trên khung làm việc vào API docs. Thực ra là cứ mỗi khi ta tạo ra 1 Collection thì Postman đã tạo Document cho Collection đó rồi, nên việc của chúng ta tương đối nhàn hạ. :)))

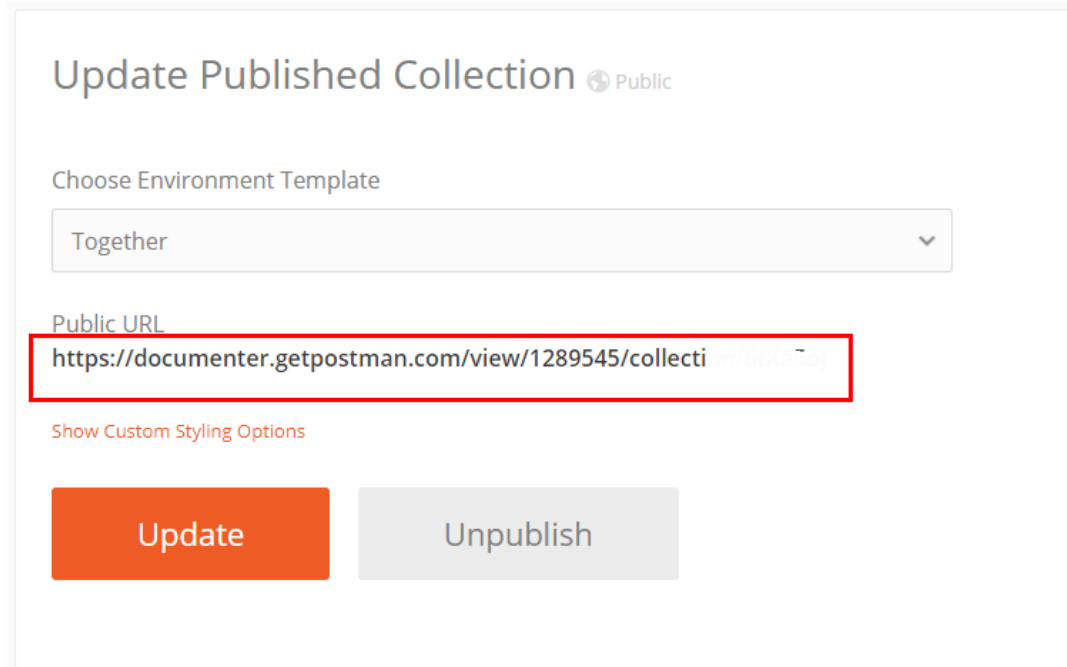
Muốn biết tài khoản của chúng ta có bao nhiêu Collections. Click chọn [Collection Links]



Toàn bộ Docs API của những Collections sẽ được show ra.

Team	Private	Links	Type to filter
Together 14 requests	# Introduction What does your API do? # Overview Things that the developers should know about # Authentication What is the preferred way of using the API?	View Docs	Publish
Daily Test (Dev) 11 requests	## what is Markdown? see [Wikipedia](http://en.wikipedia.org/wiki/Markdown) Markdown is a *lightweight markup language*, originally created by John Gruber	View Docs	Publish
AirProfits Documents 6 requests	Server Host: http://airprofits.codelovers.vn/ Default Response message: * success > { "status": "success", "data": {...} } * error: > { "status": "error", "code":	View Docs	Publish
AirProfits 18 requests	Server Host: http://airprofits.codelovers.vn/ Default Response message: * success > { "status": "success", "data": {...} } * error: > { "status": "error", "code":	View Docs	Publish
Grannana 11 requests	Add description for improved documentation Last modified 12 Oct, 2017	View Docs	Publish
Norina 106 requests	You can use the Sharing Debugger to see the information that is used when your website content is shared on Facebook, Messenger and other places. The Batch	View Docs	Publish
Postman Echo 38 requests	Postman Echo is service you can use to test your REST clients and make sample API calls. It provides endpoints for 'GET', 'POST', 'PUT', various auth	View Docs	Publish

- Click vào View Docs để xem API Docs (đây là link của trạng thái Private, link này không share được).
- Click vào Publish để tạo Link share Public (Link này thì ai cũng nhìn được nội dung). Bạn chỉ cần copy cái link rồi gửi cho người khác.



Update Published Collection Public

Choose Environment Template

Together

Public URL

<https://documenter.getpostman.com/view/1289545/collecti>

[Show Custom Styling Options](#)

Update Unpublish

Vấn đề là 1 API Docs gồm những thành phần gì và thể hiện những phần đấy trong Postman như thế nào?

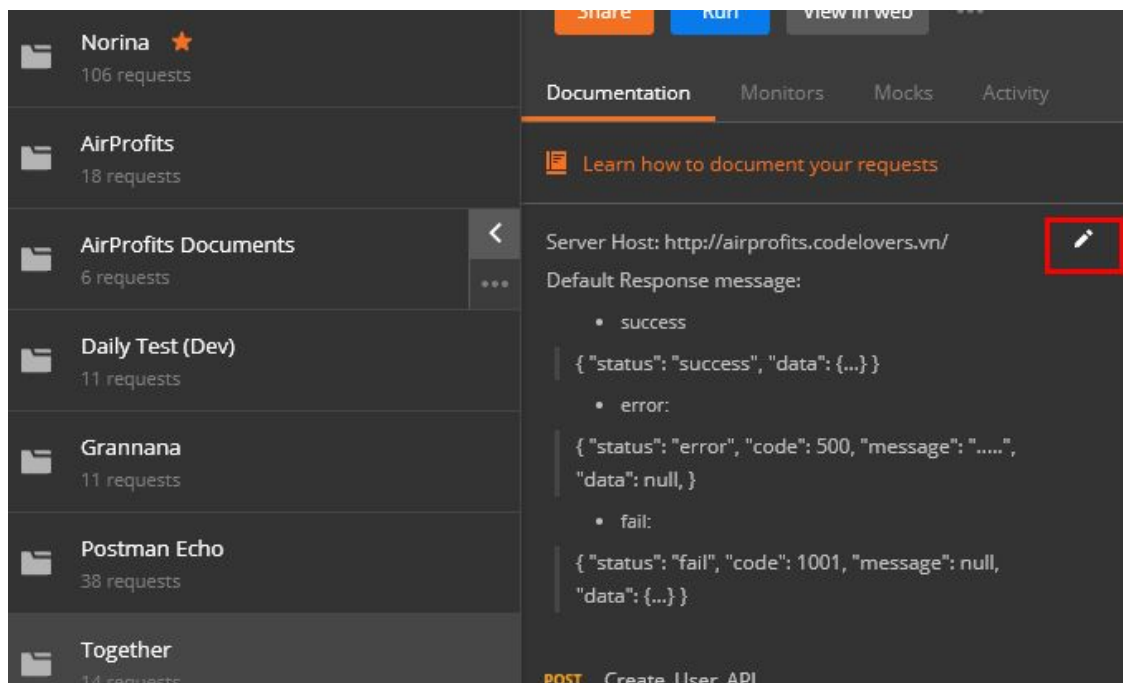
Các thành phần của 1 API Document:

- Tên của API Document
- Mô tả của API Docs
- Tên của từng API
- Mô tả của cả API: Mục đích của API, note lại các mục cần lưu ý
- Params + mô tả của params
- Sample Request
- Sample Response

Tên của API Document

Chính là tên của Collection, cách edit đơn giản, xem lại phần 7.

Mô tả của API Docs

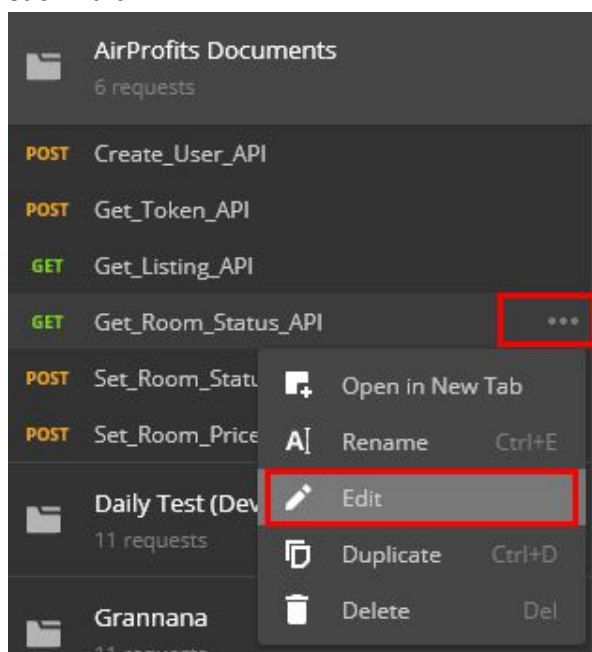


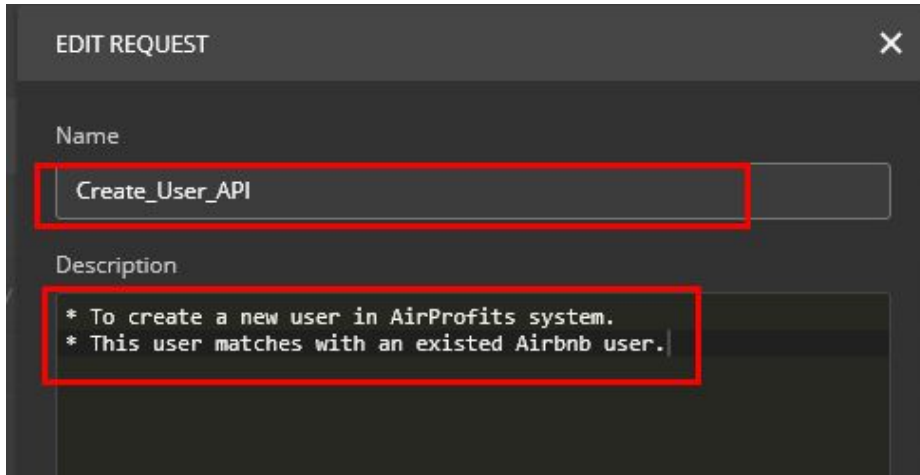
Cách Format đoạn Text này dùng Markdown. Bạn có thể xem thử mẫu ở đây <http://markdownlivepreview.com/>

Tên của từng API & Mô tả của cả API

Tên của API chính là tên chúng ta đặt cho từng Request.

Cách Edit

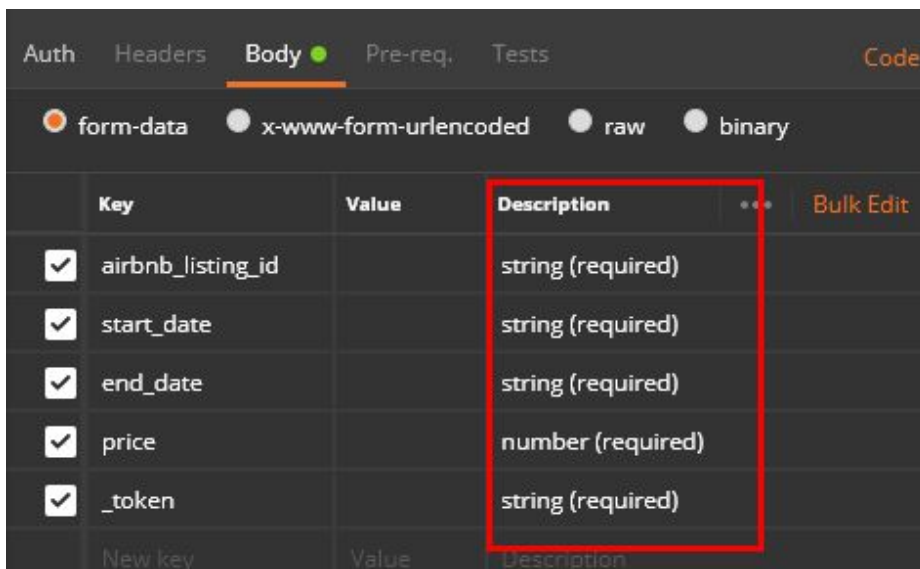




Chỗ này cũng hỗ trợ Markdown để format nhé, nói chung là chỗ nào điền text thì Postman cũng hỗ trợ Markdown nhé. :))))))

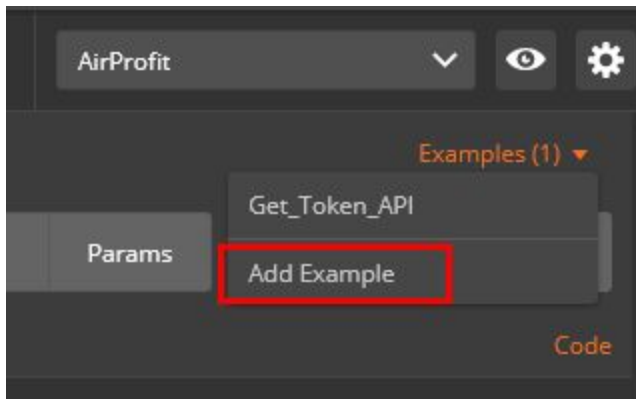
Params + mô tả của params

Các bạn điền vào phần Param hoặc Body giống như điền API vẫn điền trước đây nhưng khi viết API Docs, lưu ý: KHÔNG điền phần Value. Xem hình



Sample Request & Sample Response

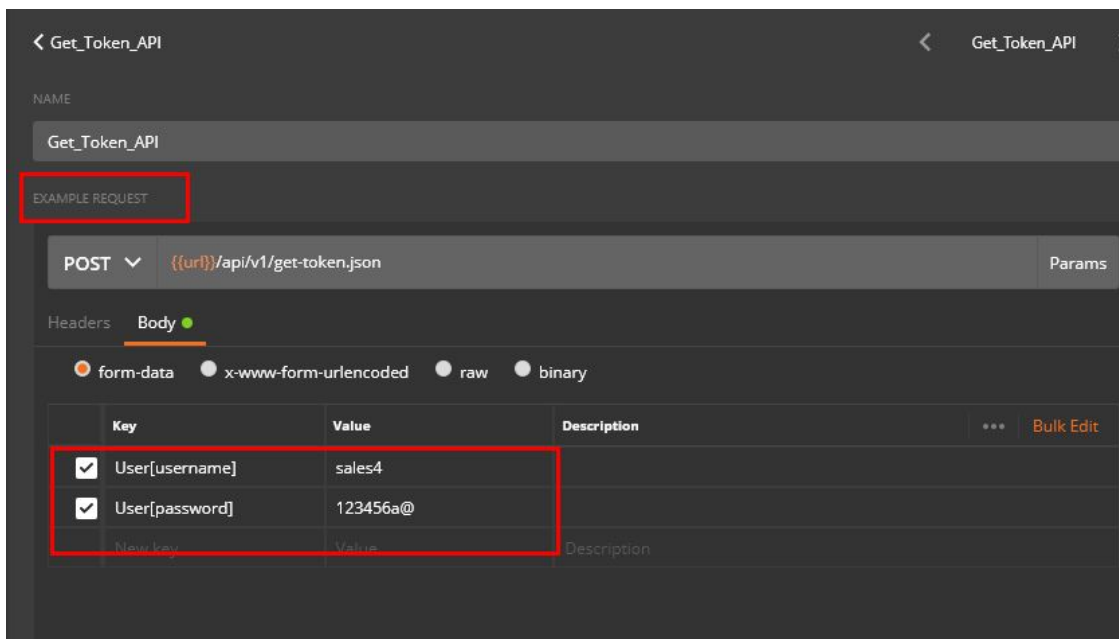
Đây là tính năng Example mà Postman cung cấp.



Bạn Add 1 cái Example rồi điền thông tin giống như 1 API thật sự là xong.

Sample Request

Lúc này thì bạn phải điền Value cho từng Param, không được bỏ trống.



Sau đó, bạn xem lại thành quả của mình ở 1 trong 2 link Public và Private ở phía đầu mình đã nói. Nó sẽ có dạng như sau:

AIRPROFITS DOCUMENTS

Introduction

- POST Create_User_API
- POST Get_Token_API
- GET Get_Listing_API
- GET Get_Room_Status_API
- POST Set_Room_Status_API
- POST Set_Room_Price_API
- POST Pre_Approve_API
- POST Withdraw_Pre_Approve_API
- POST Withdraw_Special_Offer_API
- POST Send_Special_Offer_API
- POST Send_Alteration_Request_API
- POST Decline_Inquiry_API
- POST Cancel_Alteration_Request_API
- POST Accept_Alteration_Request_API
- POST Decline_Alteration_Request_API
- GET Get_Setting_Auto_Messages_API
- GET List_Setting_Auto_Messages_API
- POST Create_Setting_Auto_Messages_API
- POST Edit_Setting_Auto_Messages_API
- GET Delete_Setting_Auto_Messages_API

AirProfits Documents

Server Host [http://localhost:3000](#)

Default Response message:

- success
 - `{ "status": "success", "data": {...} }`
- error:
 - `{ "status": "error", "code": 500, "message": ".....", "data": null, }`
- fail:
 - `{ "status": "fail", "code": 1001, "message": null, "data": {...} }`

POST Create_User_API

`/api/v1/create-user.json`

- To create a new user in AirProfits system.
- This user matches with an existed Airbnb user.

HEADERS

Content-Type	application/x-www-form-urlencoded
---------------------	-----------------------------------

BODY

username	string (required, unique)
-----------------	---------------------------

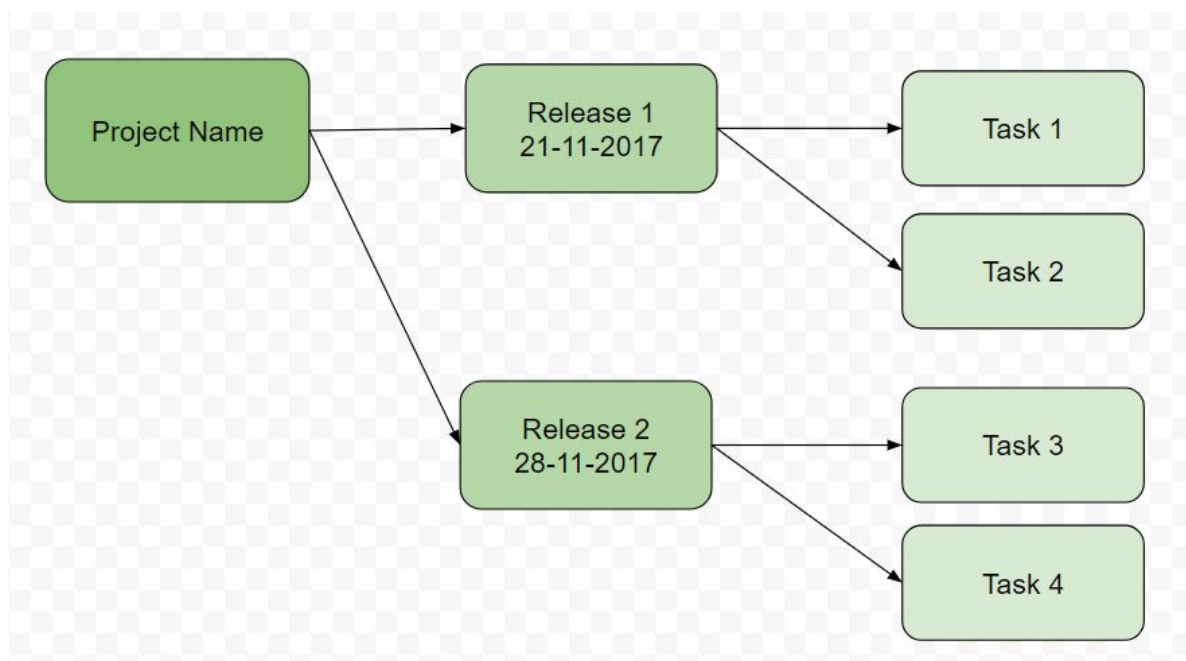
Phần bên trái sẽ là List API có trong dự án, click vào từng API để xem chi tiết.

XI. Run Test Suites từ Runner

Theo phần trước các bạn đã biết cách tạo ra những test đơn giản cho từng API, nhưng mà 1 dự án thì có quá nhiều API và quá nhiều task khác nhau, mỗi task là 1 tập hợp của 1 vài API thì phải giải quyết như thế nào. Cùng với đó là cách quản lý mà bạn nghĩ có áp dụng được cho Postman hay không? Và làm thế nào để run test đỡ tốn công sức nhất.

I. Quản lý Test Suites

Trong Postman quản lý các API theo dạng Collections và tùy vào dự án mình sẽ có cách quản lý khác nhau. Vì đặc thù dự án cứ 1 tuần lại release 1 lần nên đây là cách mình quản lý.

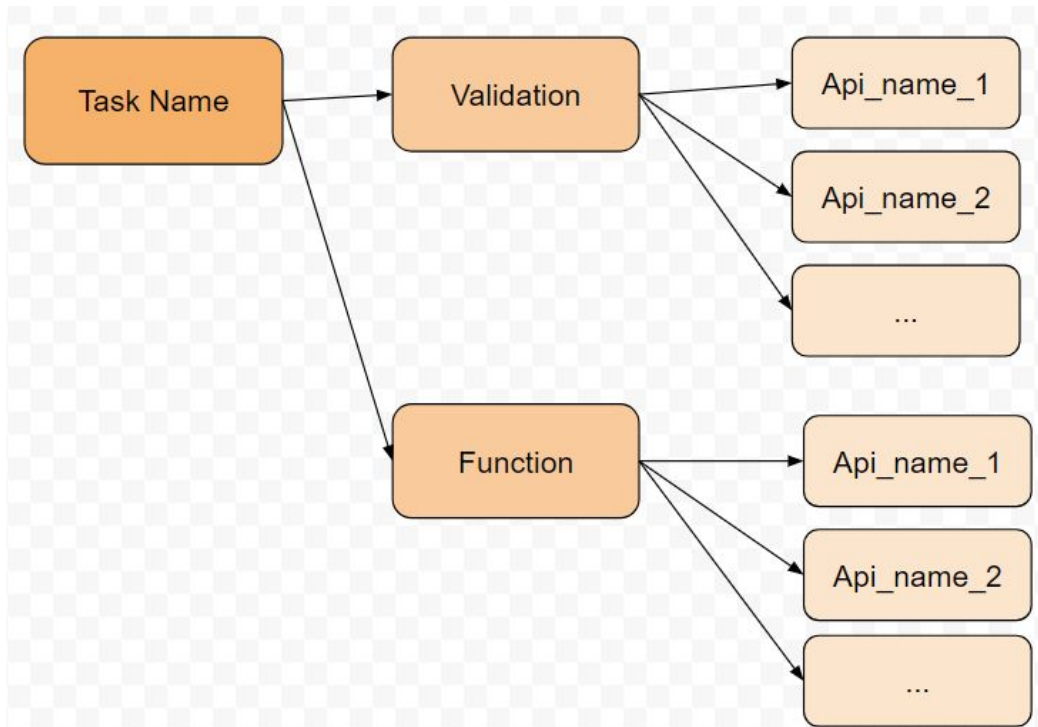


Vấn đề là từng task nhỏ thì mình sẽ sắp xếp như thế nào???

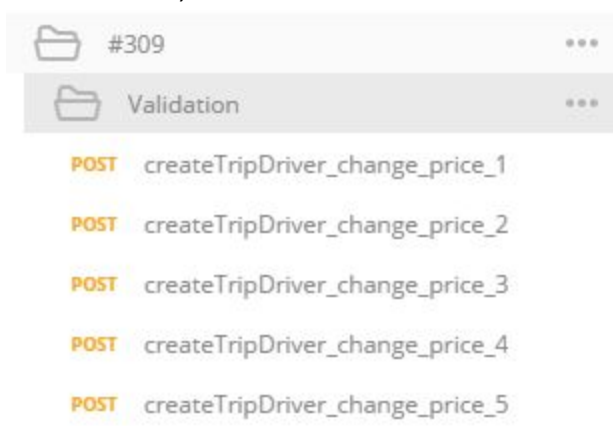
Phần trước, mình đã hướng dẫn với 1 API đơn lẻ có 2 kiểu test tương ứng:

1. Syntax Testing (Validation)
2. Functional Testing

Theo gợi ý từ các [Bloggers Postman](#), thì bạn không nên chỉ dùng 1 request cho tất cả các loại test của mình, mà nên với mỗi trường hợp bạn sẽ tạo ra 1 request, chỉ khác nhau phần Description và Test thôi. Do đó đây là cách mình cấu trúc API cho từng task.



Trên Postman, nó sẽ có hình thù như sau:



Các API trong hình từ 1 đến 5 sẽ có phần Description và Test khác nhau.

API_1:

▼ createTripDriver_change_price_1

- Không nhập Price

POST ▾ `{{url}}/apiv/1.0/create_trip`

Authorization Headers Body ● Pre-request Script ● Tests ●

```
1 var jsonData = JSON.parse(responseBody);
2 postman.setEnvironmentVariable("driver_trip_id", jsonData.data.id);
3 postman.setEnvironmentVariable("source", jsonData.data.source);
4 postman.setEnvironmentVariable("destination", jsonData.data.destination);
5
6
7
8
9 tests["Check price = null"] = jsonData.data.price === null;
```

API_2:

▼ createTripDriver_change_price_2

- đặt price = 0 ✎

POST ▾ `{{url}}/apiv/1.0/create_trip`

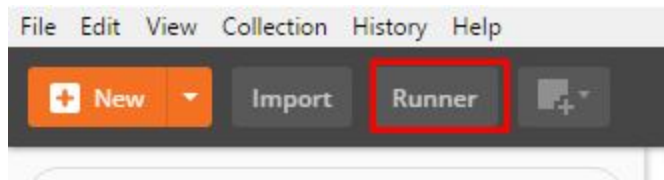
Authorization Headers Body ● Pre-request Script ● Tests ●

```
1 var jsonData = JSON.parse(responseBody);
2 postman.setEnvironmentVariable("driver_trip_id", jsonData.data.id);
3 postman.setEnvironmentVariable("source", jsonData.data.source);
4 postman.setEnvironmentVariable("destination", jsonData.data.destination);
5
6
7
8 tests["price = 0"] = jsonData.data.price === "0";
```

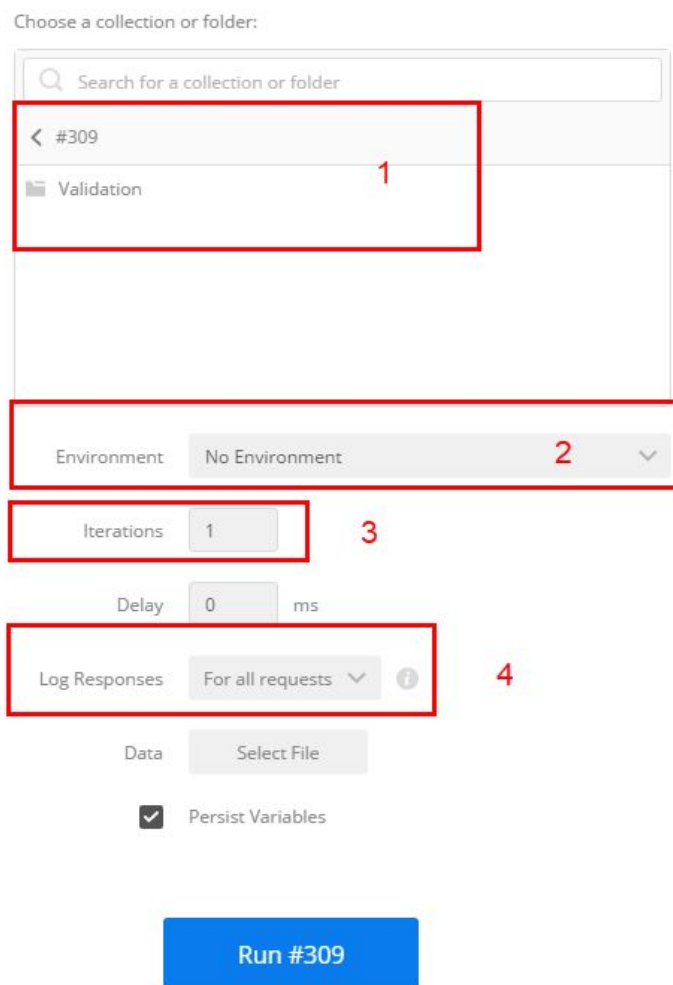
Tương tự bạn sẽ làm như vậy với các trường hợp khác.

II. Run Test Suites bằng chức năng Runner

Từ đầu series đến giờ các bạn mới biết cách test API theo cách “thủ công”, ấn SEND check từng API. Postman cung cấp tính năng run List API theo Folder bằng tính năng Runner.

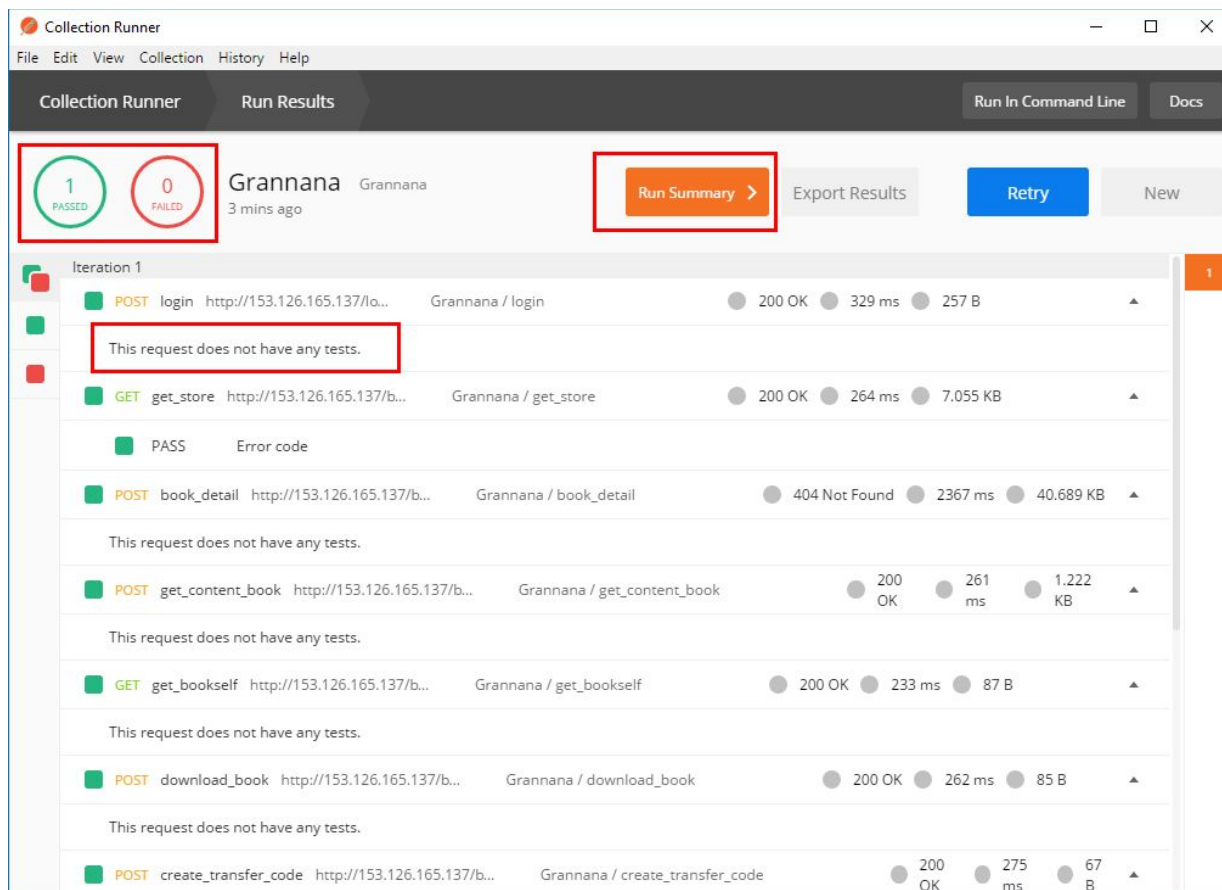


Màn hình của Runner sẽ hiện ra, ở đây bạn sẽ quan tâm đến những thành phần sau đây.



1. Folder sẽ run.
2. Environment theo dự án
3. Số lần lặp lại
4. Option cho bạn xem lại Log những request bạn muốn.

Sau khi run xong, bạn sẽ nhìn thấy report như sau:



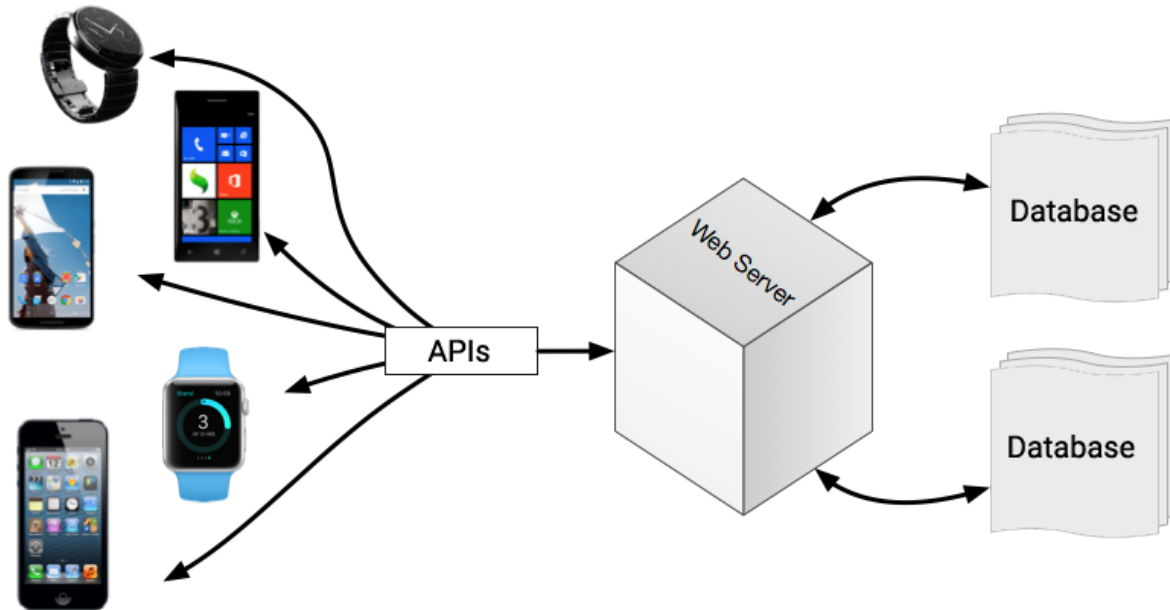
Lưu ý: Số Pass or Fail được tính trên số **Test** bạn viết, chứ không tính trên số Request bạn run.

Bạn có xem short version của report ở Summary và Lưu Log ở Export Result.
Nếu bạn muốn xem chi tiết từng thông số của request bạn click vào tên của Request đó.



XII. Cách test API như thế nào?

Sau khi đọc xong các phần “test API với Postman” của mình, các bạn có thể nắm được cái kiến thức cơ bản của API và các chức năng của Postman đem lại. Nhưng cách sắp xếp test và viết Testcase cho API như thế nào thì vẫn có vẻ chưa thông lắm, nên đây là phần cách test API như thế nào cho hợp lý.



Nhắc lại kiến thức 1 chút: API chỉ là cầu nối nói chuyện giữa Client và Server. API không thực hiện 1 business logic nào cả, đơn thuần chỉ là thùng đi giao thư, chuyển thông tin thôi. Thế test API là test thùng giao thư ah? Hay là test cái gì? Xin được phép trả lời luôn: mình **dùng API để test business logic ở phía server**. Hãy xem ví dụ dưới đây để hiểu rõ hơn.

Ví dụ:

Tôi muốn check API update_profile gồm 2 trường Name và Birthday. Trong đó trường Name là bắt buộc và phải lớn hơn 4 ký tự. Trường Birthday thì không bắt buộc nhập.

Cách xử lý của Server và Client (có thể không giống với cty bạn):

1. User vào màn hình Profile, sửa lại 2 trường Name và Birthday.
2. User ấn vào nút Update Profile (Code ở client sẽ check điều kiện của trường Name, nếu đúng thì submit gửi API, gọi là request, nếu sai sẽ hiện thông báo tương ứng).
3. Thông tin mới gồm Name và Birthday theo phong bì thư của API cập bến Server.
4. Server đọc thư và check điều kiện lại 1 lần nữa.
5. Nếu các thông tin Name và Birthday đều Valid thì 2 thông tin đó được cập nhật vào Database.
6. Server trả lại thông tin, gọi là response, về lại cho client thông báo rằng nó đã cập nhật thành công.
7. User nhìn thấy Name và Birthday của mình đã được thay đổi ở màn hình Profile.

Khi thực hiện test API, chính là việc chúng ta test các bước 4, 5 và 6. Đó đó, với 1 API đơn lẻ, chúng ta sẽ check 2 phần chính:

- tạm gọi là **Syntax Testing** (Validate dữ liệu – bước 4 + bước 6)
- và **Functional Testing** (Test business logic – bước 5 và 6).

Syntax Testing

Loại này sẽ tập trung vào cái Method check điều kiện: Accept với data đúng và Reject với data sai hay không. Một vài ví dụ:

- Bỏ trống trường bắt buộc → Trong Response sẽ phải có thông báo lỗi, các thông tin khác không được cập nhật. Server không thực hiện 1 business logic nào cả.
- Bỏ trống trường không bắt buộc → Không có lỗi gì cả, Server vẫn thực hiện business logic.
- Điền các thông tin sai kiểu định dạng, ví dụ trường thời gian lại điền chữ → Trong Response sẽ phải có thông báo lỗi...

Chốt lại: Cái này giống hệt như những trường hợp Validate dữ liệu, chúng ta vẫn hay làm hàng ngày.

Functional Testing

Loại này check các Method xử lý dữ liệu và thực hiện 1 chức năng có đúng hay không. Ví dụ:

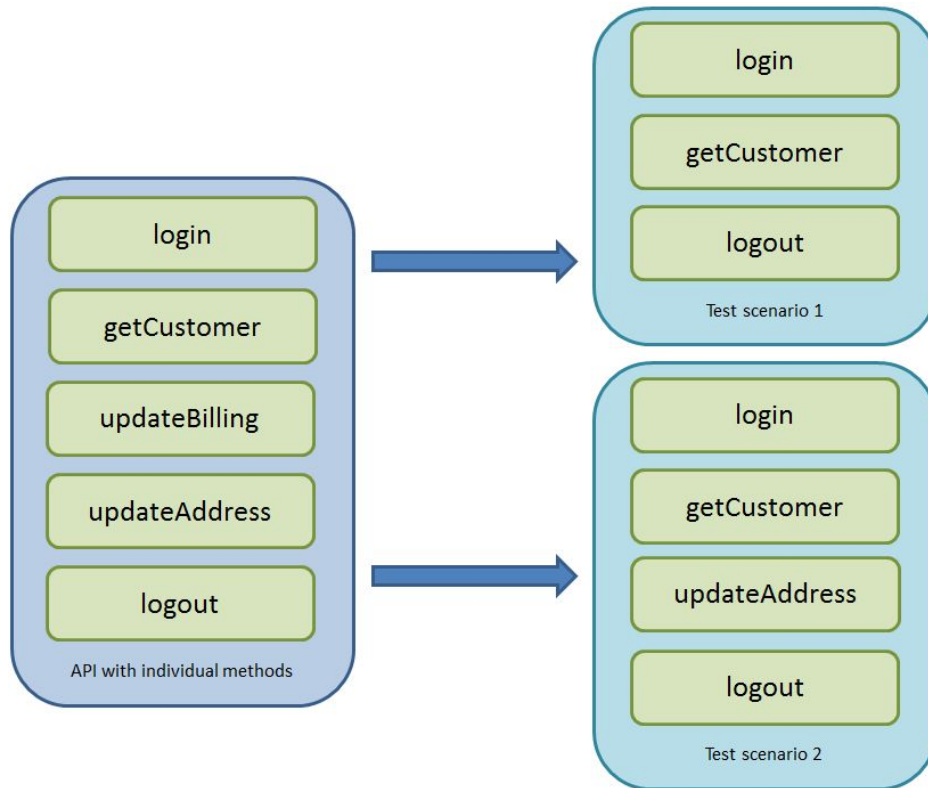
- Giá là X và số phần trăm discount là Y thì số tiền phải trả là $X*(1-Y)$ hay không → Nó chính là việc test Method tính toán với các tham số X và Y mà thôi. Việc thực hiện business logic có thể không lưu kết quả vào DB.
- Việc Update trường Name ở ví dụ ban đầu có được lưu vào DB hay không? → mở DB ra và check kết quả.
- Yêu cầu trả về thông tin của những user có tên là "Nam" → Vào DB thực hiện câu Query và so sánh với Response xem 2 kết quả có khớp nhau hay không...

Ấy ấy, chưa hết nhé. =))) 2 cái loại test trên phục vụ cho test các API đơn lẻ thôi. Còn nữa

Test scenarios

Cuối cùng là ta ghép các API lại với nhau sẽ nó có bị lỗi ở đâu không? Chỗ này chính là những cái Test Suite, gộp nhiều Test Case lại.

Ví dụ như hình:



Phần này chắc mình cũng không phải nói nhiều, các bạn cũng đã quá quen thuộc với nó.

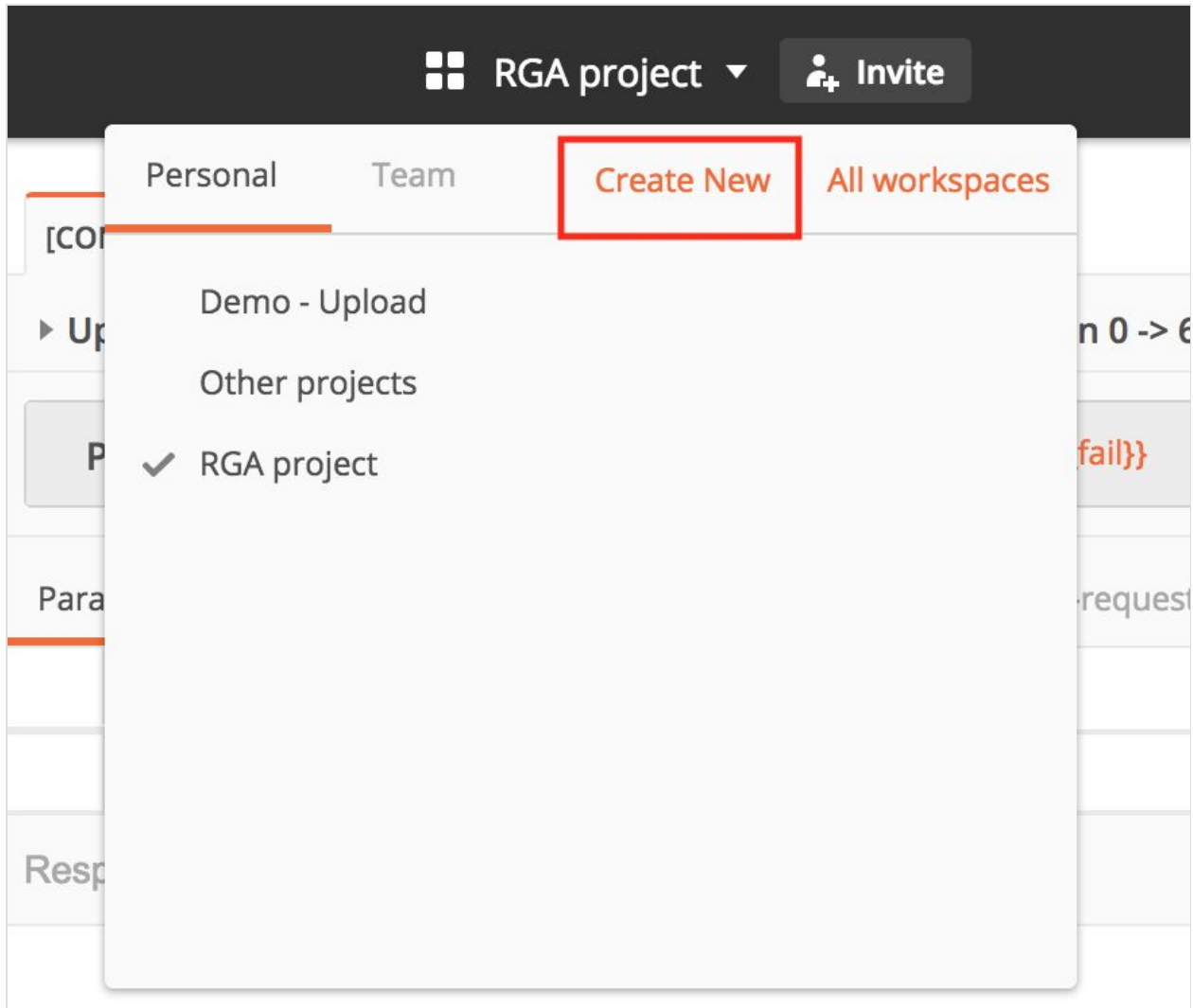
Lưu ý của test API:

1. Khi sử dụng Postman, hãy để mỗi trường hợp là 1 API riêng biệt, không test đề lên nhau, sau khó kiểm soát và không tạo được test case cho automation.
2. Để không phải căng mắt check từng response của các trường hợp đơn lẻ, hãy đọc lại phần 9 Test Response.

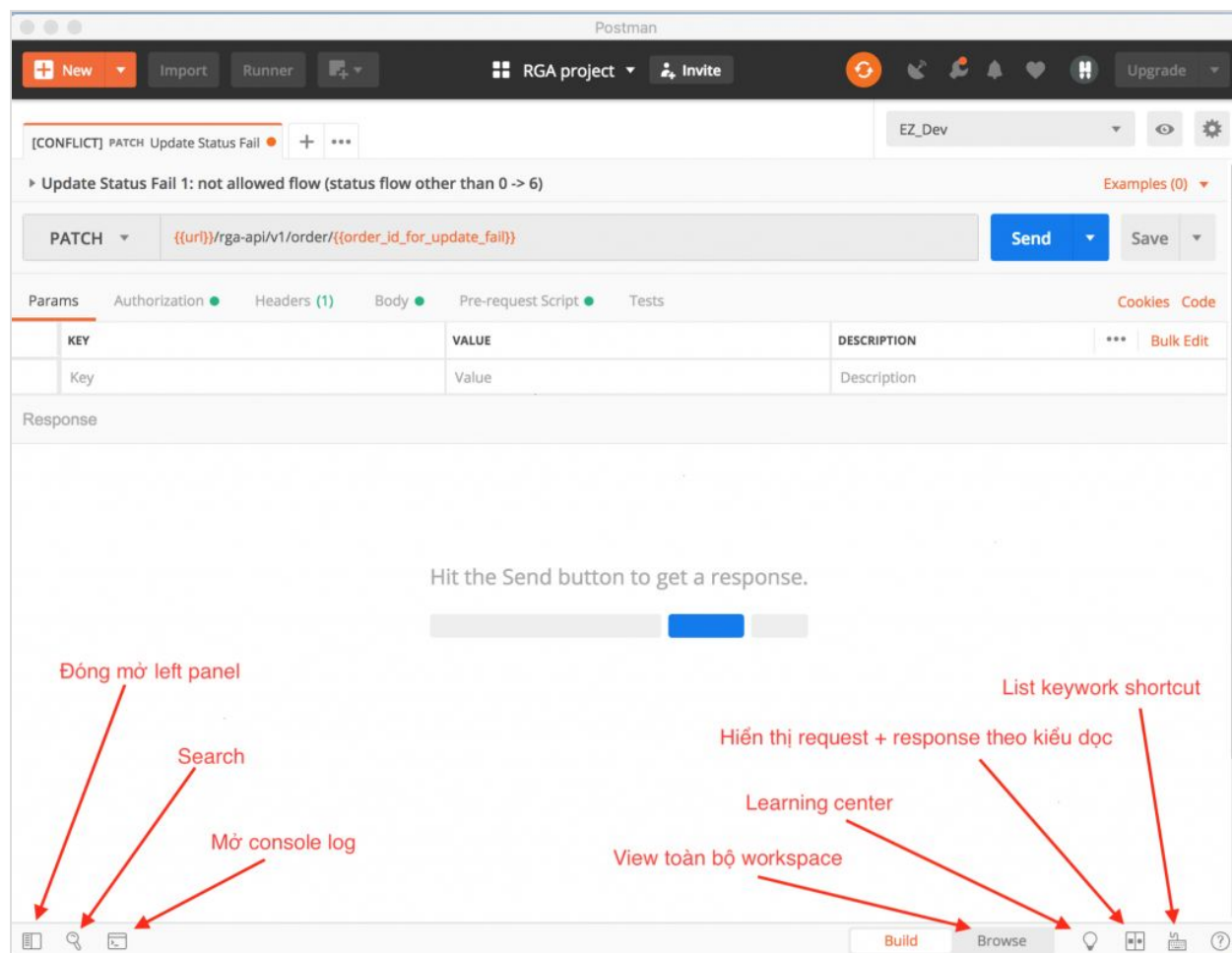
XIII. Những lưu ý khi code API test

1. Postman – Workspace

Trước kia, ta chỉ có 1 workspace và làm nhiều project trên đó nhưng đã có tính năng tách workspace thì ta nên tách mỗi project là 1 workspace, sẽ dễ nhìn hơn rất nhiều, tránh được những nhầm lẫn khi kéo thả các request từ folder này sang folder khác.



Ngoài ra, còn có nhiều tiện ích mà chúng ta ít sử dụng:



Trong số này thì mình hay sử dụng console log nhất vì console log sẽ show toàn bộ thông tin request và response, giúp ích cho việc debug nhiều. Nếu bạn thắc mắc là mỗi khi run Send API thì response sẽ trả về được hiển thị ở phía dưới rồi, cần gì phải console log? Có 2 trường hợp bạn sẽ cần phải sử dụng console log:

- Bạn test theo Runner. Runner dashboard chỉ lưu full thông tin của 10 request đầu tiên, những request phía sau, nó chỉ lưu thông tin cơ bản, bạn ko xem được full thông tin nếu có request nào đó bị lỗi.
- Nếu bạn viết các tham số của 1 API dưới dạng biến thì bạn cần có console log thì mới view được giá trị thực các tham số ở mỗi lần run.

2. Test

Ở phần test, mình có nói ở [bài 9](#) rồi nhưng có 1 vài điểm mọi người cần chú ý:

Các error code cần được check nếu các error code này được sử dụng đúng như định nghĩa của REST API ví dụ:

- 200 – success: Dùng cho happy case: nhận request và trả response có data đúng.
- 400 – bad request: Dùng trong các TH lỗi ở phía client ví dụ: sai data type, missing data parameter.
- 401 – Unauthorized: Dùng trong việc thiếu token authen
- 403 – Forbidden: Dùng cho việc access vào những resource không có quyền hạn
- 500 – Server error: Dùng cho các vấn đề error của server: Thiếu library, disconnect DB...

Ngoài error code thì có thể sử dụng response data để verify thêm. Đọc lại [bài này](#) để biết cách test API như thế nào. Ngoài ra, các built-in function của Postman đã được thay đổi khá nhiều, các bạn xem ở đây. [Link docs](#)

3. Environment

Phần này thì cũng không có nhiều sự thay đổi, nhưng có 1 số lưu ý nhỏ

Nên đặt tên biến meaningful. Ví dụ: ta thực hiện test 1 field là Phone, thì có 2 cases: phone sai và phone đúng, ta không nên đặt tên biến là phone và dùng chung cho cả hai trường hợp, ta nên tách ra thành wrongPhone và rightPhone.

Environment có thể lưu được array. Ví dụ: Khi nhận response trả về 1 list các giá trị customerId và bạn muốn lưu list customerId đây về thành 1 array để request khác bạn có thể lấy random các giá trị đây

category_id	3	3
province_id	1	1
city_list	133,187,105	133,187,105
city_id	187	187
district_id	3	3

4. Run script

Để đảm bảo rằng test API lúc nào chạy cũng đúng, mình có 2 cách làm:

- Với TH các API có đủ, “đủ” ở đây có nghĩa là với bất cứ cái dữ liệu, action nào trên UI thì cũng có API tương ứng. → Mình sẽ tạo test data bằng cách dùng API cần thiết, sau đó mới run API test. Cách làm này sẽ giúp cho các Test case không bị phụ thuộc vào test data ở chỗ nào cả.

Ví dụ: Bạn cần test API update_info_user, bạn sẽ sử dụng API create_user trước, thay vì sử dụng 1 user đã có sẵn trong hệ thống

- Với TH các API không có đủ, ta nên chuẩn bị sẵn các data test của mình và cả DB test nữa. Cách làm:
 1. Bạn tạo ra đủ các master data bằng cách thủ công: các data cần thiết để có thể run được API, ví dụ bạn tạo ra list các sản phẩm trước khi thực hiện mua hàng, đặt số lượng của sản phẩm ở mức cao tương đối, ví dụ đặt số lượng = 100.
 2. Chạy thử 1 lượt với tất cả các API bạn có với Runner, nếu có lỗi thì fix.
 3. Chạy lại lần 2 để đảm bảo đã hết lỗi. (Lưu ý, nên dynamic parameter nhiều nhất có thể, đừng hard code data)
 4. Dump DB ra thành 1 file sql để khi nào mình run lại script thì mình sẽ run lại file sql này trước.

Reference

1. An Introduction to APIs - Brian Cooksey <https://zapier.com/learn/apis/>
2. Postman Document <https://www.getpostman.com/docs/>
3. API testing best practices - Bas Dijkstra
<https://www.ontestautomation.com/api-testingbest-practices/>