API Testing

#1. Tìm hiểu về kiến trúc Client/Server

#1.1. Khái niệm

**Client server** là mô hình mạng máy tính gồm có 2 thành phần chính đó là máy khách (client) và máy chủ (server). Server chính là nơi giúp lưu trữ tài nguyên cũng như cài đặt các chương trình dịch vụ theo đúng như yêu cầu của client. Ngược lại, Client bao gồm máy tính cũng như các loại thiết bị điện tử nói chung sẽ tiến hành gửi yêu cầu đến server.

Mô hình mạng Client Server sẽ cho phép mạng tập trung các ứng dụng có cùng chức năng tại một hoặc nhiều dịch vụ file chuyên dụng. Chúng sẽ trở thành trung tâm của hệ thống. Hệ điều hành của mô hình Client server sẽ cho phép người dùng chia sẻ đồng thời cùng một loại tài nguyên mà không giới hạn vị trí địa lý.

**#1.2. Nguyên tắc hoạt động**

**Client**

Client chính là khách hàng sử dụng dịch vụ. Nó có thể là một tổ chức hay cá nhân cụ thể nào đó. Và khi khái niệm này được sử dụng trong lĩnh vực kỹ thuật số thì cũng mang ý nghĩa tương tự như vậy. Trong **Client Server** thì Client chính là một máy tính (Host). Chúng có khả năng nhận thông tin từ nhà cung cấp và sử dụng dịch vụ cụ thể (Server).

**Server**

Server là từ dùng để nói về một máy chủ hoặc một phương tiện được sử dụng để phục vụ các dịch vụ nào đó. Khi khái niệm này được sử dụng trong lĩnh vực công nghệ thì Server là một máy tính từ xa. Chúng có chức năng là cung cấp các thông tin (dữ liệu) cho một dịch vụ cụ thể nào đó hoặc quyền truy cập đối với dịch vụ.

Hay hiểu một cách đơn giản thì Client là nguồn yêu cầu và Server phải phục vụ theo nó.

## #1.3. Ưu, nhược điểm

**Ưu điểm:**

Tập trung

Ưu điểm đầu tiên của mô hình **Client Server** kiểu mạng khách chủ đó chính là khả năng kiểm soát tập trung (Centralization) đã được tích hợp sẵn. Theo như mô hình này thì tất cả mọi thông tin cần thiết đều sẽ được đặt ở một vị trí duy nhất. Đây là một ưu điểm vô cùng hữu ích được những người quản trị viên mạng yêu thích bởi vì họ có thể toàn quyền quản lý cũng như điều hành mọi việc.

Tính năng này giúp cho mọi sự cố trong mạng đều sẽ được giải quyết ở cùng một nơi thống nhất. Đồng thời, việc cập nhật cơ sở tài nguyên, dữ liệu cũng sẽ dễ dàng hơn rất nhiều.

Bảo mật

Trong mạng Client Server, tất cả các dữ liệu đều sẽ được bảo vệ một cách tối đa nhờ vào hệ thống kiến trúc tập trung của mạng. Thông qua đó, nó sẽ giúp người dùng kiểm soát truy cập để chỉ có những ai được cấp quyền truy cập thì mới được thực hiện các thao tác cần thiết.

Muốn làm như vậy, chúng ta cần phải áp đặt thông tin đăng nhập cũng như Username hay Password. Bên cạnh đó, nếu dữ liệu của chúng ta bị mất thì các file sẽ được khôi phục một cách vô cùng dễ dàng chỉ từ một bản sao lưu duy nhất mà thôi.

Khả năng mở rộng

Mô hình mạng kết nối **Client Server** có khả năng mở rộng vô cùng tốt. Chỉ cần người dùng cần sử dụng bất cứ lúc nào thì họ cũng có thể tăng được số lượng tài nguyên của mình. Ví dụ như số Client hoặc Server. Nhờ đó mà chúng ta có thể tăng kích thước của Server một cách dễ dàng mà không bị gián đoạn nhiều.

**Nhược điểm:**

Tắc nghẽn lưu lượng

Nói về nhược điểm lớn nhất của mô hình mạng **Client Server** đó chính là tắc nghẽn lưu lượng. Trong trường hợp có quá nhiều Client tạo request từ cùng một Server thì nó có thể sẽ làm cho kết nối chậm hơn. Trong trường hợp xấu nhất còn có thể xuất hiện hiện tượng crash. Khi một server bị quá tải thì sẽ tạo ra nhiều vấn đề khi truy cập thông tin.

Độ bền

Client Server là mạng tập trung chính vì thế, khi Server chính xảy ra sự cố hoặc bị nhiễu thì cũng đồng nghĩa với việc toàn bộ hệ thống mạng sẽ bị gián đoạn. Như vậy, bạn cần chú ý đó là mạng thiếu tính ổn định và độ bền. Bạn cần chú ý khi thực hiện.

Chi phí

Chi phí được sử dụng để thiết lập và bảo trì Server trong Client Server thường sẽ khá cao. Lý do là vì các hệ thống mạng có sức mạnh rất lớn cũng đồng nghĩa với việc giá để chi cho việc này là rất đắt. Chính vì vậy, không phải ai cũng có khả năng chỉ trả và sử dụng.

Bảo trì

Khi các Server thực hiện triển khai để làm việc thì nó cũng sẽ hoạt động một cách không ngừng nghỉ. Điều này đồng nghĩa với việc chúng ta cần phải quan tâm đến việc bảo trì hệ thống đúng mức. Khi xảy ra bất cứ vấn đề gì cũng cần phải giải quyết ngay lập tức. Vậy nên, cần phải có một nhà quản lý mạng chuyên biệt để tiến hành duy trì hoạt động của Server khi chúng được đưa vào và sử dụng.

Tài nguyên

Một điều mà chúng ta rất cần phải lưu ý đó chính là không phải tất cả tài nguyên hiện có trên Server đều sử dụng được. Ví dụ một cách đơn giản đó chính là chúng ta không thể in trực tiếp được tài liệu từ trên web cũng như tiến hành chỉnh sửa bất kỳ một thông tin nào trên ổ cứng của Client cả.

# #2. Khái niệm API

API là viết tắt của Application Programming Interface – phương thức trung gian kết nối các ứng dụng và thư viện khác nhau. Nó được hiểu là **giao diện lập trình ứng dụng**. API cung cấp khả năng truy xuất đến một tập các hàm hay dùng. Nó khiến cho việc trao đổi dữ liệu giữa các ứng dụng trở nên hiệu quả hơn.

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. 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.

**4 đặc điểm nổi bật của API**

* API sử dụng mã nguồn mở, dùng được với mọi client hỗ trợ XML, JSON.
* API có khả năng đáp ứng đầy đủ các thành phần HTTP: URI, request/response headers, caching, versioning, content forma…. Bạn có thể sử dụng các host nằm trong phần ứng dụng hoặc trên IIS.
* Mô hình web API dùng để hỗ trợ MVC như: unit test, injection, ioc container, model binder, action result, filter, routing, controller. Ngoài ra, nó cũng hỗ trợ RESTful đầy đủ các phương thức như: GET, POST, PUT, DELETE các dữ liệu.
* Được đánh giá là một trong những kiểu kiến trúc hỗ trợ tốt nhất với các thiết bị có lượng băng thông bị giới hạn như smartphone, tablet…

**#3. API Testing là gì? Mục đích của API Testing**

**#3.1. API Testing là gì?**

API Testing là một phương pháp kiểm thử phần mềm, bao gồm kiểm tra trực tiếp giao diện lập trình ứng dụng (API) mà không tác động tới client. Test API còn là một phần của kiểm thử tích hợp nhằm xác định các API có đáp ứng kỳ vọng về chức năng, độ tin cậy, hiệu suất và bảo mật hay không.

**#3.2. Mục đích của API Testing**

**Kiểm thử ứng dụng sớm mà không cần giao diện người dùng**

Nếu bạn tìm thấy lỗi càng muộn thì bạn càng mất nhiều thời gian và công sức để sửa nó. API Testing sẽ giúp người kiểm thử tham gia sớm vào vòng đời phát triển của sản phẩm. Với API Testing, bạn hoàn toàn có thể bắt đầu kiểm thử ứng dụng sớm mà không cần đến giao diện người dùng. Điều này sẽ giúp bạn sớm khắc phục được các vấn đề trong vòng đời phát triển, nếu không thì sẽ mất nhiều chi phí để khắc phục khi lỗi được xác định ở quá trình kiểm thử GUI. Ưu điểm của API Testing là có thể kiểm tra rất nhiều logic mà không bị phụ thuộc vào GUI.

**Tạo ra một chiến lược kiểm thử**

Đây là hình ảnh của “Kim tự tháp tự động hóa” (Automation pyramid). Nếu chúng ta nắm được, chúng ta có thể tạo ra một chiến lược tự động hoá hiệu quả.

Đi từ tầng dưới của kim tự tháp, các chi phí liên quan đến việc tạo ra và duy trì các phương pháp, thời gian thực hiện, phạm vi kiểm thử sẽ dần tăng lên. Kim tự tháp chỉ ra rằng chúng ta cần làm nhiều kiểm thử tự động thông qua Uni Test và API Testing hơn là thực hiện kiểm thử dựa trên GUI.

Trên thực tế, việc liên tục tích hợp, thời gian để kiểm thử hồi quy GUI mấy quá nhiều thời gian để nhận lại phản hồi. Các chi phí liên quan đến việc thực hiện và duy trì các phương pháp kiểm thử sẽ dần tăng lên.

**#4. JSON và Cách tổ chức dữ liệu JSON**

**JavaScript Object Notation** (thường được viết tắt là **JSON**) là một kiểu dữ liệu mở trong Javascript. Kiểu dữ liệu này bao gồm chủ yếu là văn bản, có thể đọc được theo dạng cặp "thuộc tính - giá trị". Về cấu trúc, nó mô tả một vật thể bằng cách bọc những vật thể con trong vật thể lớn hơn trong dấu ngoặc nhọn (**{ }**). JSON là một kiểu dữ liệu trung gian, chủ yếu được dùng để vận chuyển thông tin giữa các thành phần của một chương trình.

**#Cấu trúc chuỗi JSON**

Ngày nay, JSON được sử dụng nhiều trong 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

Values có 6 loại:

● Object - {}

● Array - []

● String - ""

● Number - 20

● Boolean - true/false

● Null



**Object là gì?**

Object trong JSON được thể hiện bằng dấu ngoặc nhọn {}. Object trong JSON cũng khá giống với Object trong JavaScript nhưng có một số điểm giới hạn như:

* Key (thuộc tính): Phải luôn được đặt trong dấu ngoặc kép “”.
* Value: Chỉ được sử dụng các kiểu dữ liệu cơ bản như number, string, arrays. objects, null và không được sử dụng funtion, date, hay undefined.
* Không đặt dấu “,” cuối cùng như bên Object của JavaScript.

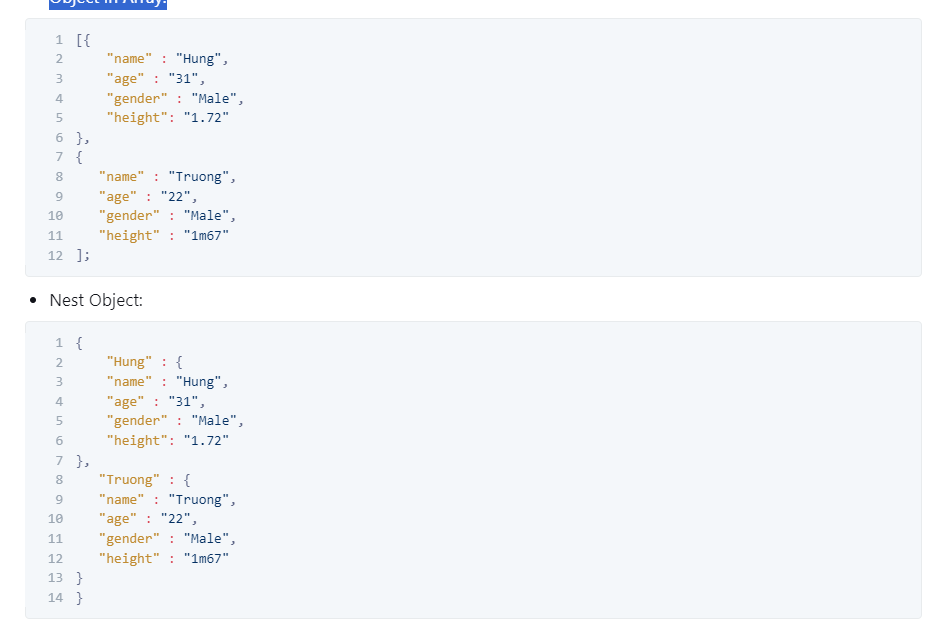
**Các kiểu Object**

Object có 2 kiểu phổ biến là:

* Object in Array.
* Nest Object.

Ví dụ:

* Object in Array:



**#Cấu trúc mảng JSON**

**Mảng là gì?**

Mảng trong JSON đại diện cho danh sách thứ tự các giá trị. Mảng trong JSON có thể lưu trữ nhiều giá trị. Nó có thể lưu trữ chuỗi, số, boolean hoặc đối tượng trong mảng JSON.

Trong mảng JSON, các giá trị phải được phân tách bằng dấu phẩy.

[] ( Dấu ngoặc vuông) đại diện cho mảng JSON.

**Mảng dưới dạng đối tượng JSON**

* Mảng JSON chuỗi

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]

* Mảng JSON của các chuỗi

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

[1, 3, 5, 7, 9]

* Mảng JSON của các giá trị boolean

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

[true, true, false, false, true]

* Mảng JSON của các đối tượng

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

{"name": "John",

"age": 37,

"car": ["Ford", "BMW", "Fiat"]

}

* Mảng đa chiều trong JSON

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

[

[ "a" , "b" , "c" ],

[ "m" , "n" , "o" ],

[ "x" , "y" , "z" ]

]

**Truy cập giá trị mảng**

Truy cập các giá trị mảnh bằng cách sử dụng chỉ số: x = myObj.car[0];

**#5. Cấu trúc của 1 gói tin Request/Response**

**#HTTP là gì?**

**HTTP (HyperText Transfer Protocol)** là *giao thức truyền tải siêu văn bản*, giao thức HTTP dựa trên giao thức TCP/IP nó cho phép lấy về các tài nguyên ví dụ như văn bản HTML, text, video, ảnh ...

**HTTP** là nền tảng dùng để trao đổi dữ liệu cho các ứng dụng Web với mô hình Client/Server. Các yêu cầu (request hay HTTP Request) được tạo bởi client (*thường là các Web browser như chrome, firefox ...*) được gửi đến server http, server sẽ gửi lại dữ liệu (response hay HTTP Response) để client nhận được. Các request, response là các message có cấu trúc đơn giản gọi chung nó là các HTTP Message.

Sự đơn giản của HTTP Message thể hiện ở chỗ nó có thể đọc được mà không nhất thiết phải có công cụ phân tích riêng biệt, nó giảm đi sự phức tạp khi phát triển, kiểm tra ứng dụng.

**#Sơ đồ làm việc HTTP truyền tải dữ liệu Client/Server**

Khi client (*như web browser, các Http API trong ngôn ngữ lập trình ...*) kết nối đến server để gửi yêu cầu (request) và lấy về dữ liệu trả về (response) từ server thì nó trải qua các bước:

* **Bước 1) Mở kết nối TCP** - Kết nối TCP (*Giao thức HTTP dựa trên TPC*) trên địa chỉ xác định bởi URL (Uniform Resource Locator) và cổng thường là 80, kết nối này được dùng để gửi các yêu cầu (request) và nhận phản hồi (response). Client có thể mở ra kết nối TCP mới hoặc sử dụng kết nối đang có, thậm chí nó tạo ra nhiều kết nối TCP cùng lúc đến server.
* **Bước 2) Gửi HTTP Message (request)** - HTTP Message (request) chính là nội dung yêu cầu được client tạo ra và gửi đến server. HTTP Message có nội dung text (plain text) mà chúng ta có thể đọc được (người đọc được). Với phiên bản HTTP/2 thì nội dung HTTP Message được bao bọc trong các frame, nó làm cho người không đọc được một cách trực tiếp - tuy nhiên về mặt ý nghĩa nội dung không đổi so với HTTP/1.1  
  Ví dụ đây là nội dung một HTTP Message (request do client tạo ra)

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

GET /lap-trinh-php/ HTTP/1.0

Host: xuanthulab.net

Accept: text/html, application/xhtml+xml, application/xml;q=0.9, image/webp, \*/\*;q=0.8

* **Bước 3) Đọc HTTP Message nhận được từ servcer (response)** - Http Message (response) trả về từ server có cấu trúc tương tự Http Message (request), ví dụ đây là một response:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

HTTP/1.0 200 OK

Date: Sat, 13 Feb 2021 09:59:11 GMT

Server: nginx

Accept-Ranges: bytes

Content-Length: 54545

Content-Type: text/html; charset=UTF-8

<!DOCTYPE html><html lang="vi">...

* **Bước 4) Đóng kết nối hoặc sử dụng lại cho các truy vấn khác**

**#HTTP Message**

**HTTP Message** (*Thông điệp HTTP*) là các thông tin được trao đổi qua lại giữa client và server. Có hai kiểu HTTP Message: HTTP Request Message (Request) được gửi đi bởi client tới server - khi server nhận được nó biết phải thực hiện nhiệm vụ nào đó, HTTP Response Message (Response) là trả lời từ server về cho client.

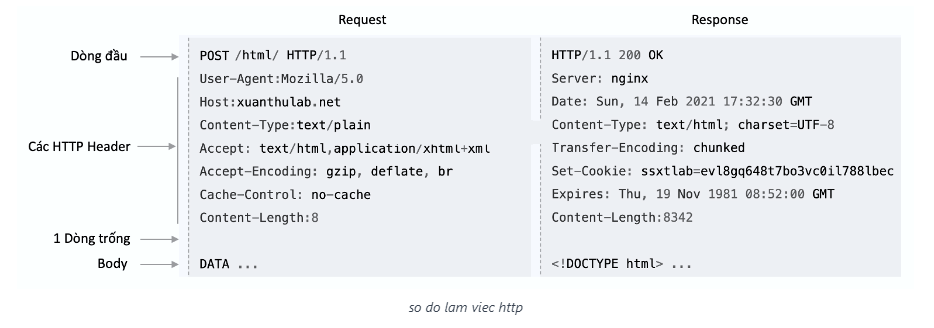
**HTTP Message** trong phiên bản HTTP/1.1 có các thành phần dữ liệu trình bày trong định dạng text (plain text) mà người có thể đọc hiểu. Trong bản HTTP/2 thì các thành phần đưa vào định dạng nhị phân (binary) là các frame làm cho người không đọc được trực tiếp nữa. Tuy vậy ý nghĩa của các thành phần trong HTTP Message giữa hai phiên bản không đổi. Do vậy, ở đây trình bày về HTTP/1.1 nhưng vẫn có thể áp dụng cho HTTP/2

**HTTP/1.1** (chuẩn hóa năm 1997) là phiên bản giao thức HTTP mặc định trong các trình duyệt, server http ngày nay. **HTTP/2** (chuẩn hóa năm 2015) là phiên bản tiếp theo đang dần phổ biến (đặc biệt là tốc độ, an toàn, nén và ghép kênh), nếu server http của bạn hỗ trợ HTTP/2 thì hầu hết các browser sẽ tự động sử dụng được luôn.

**Request** và **Response** có những dữ liệu khác nhau nhưng chúng chia sẻ chung một cấu trúc, ví dụ dưới đây là một request và response (bạn có thể dùng Developer Tool trong trình duyệt Chrome hoặc ứng dụng Postman hoặc ứng dụng Fiddler để có thể đọc request/respone)



**Request** và **Response** có những dữ liệu khác nhau nhưng chúng chia sẻ chung một cấu trúc, ví dụ dưới đây là một request và response (bạn có thể dùng Developer Tool trong trình duyệt Chrome hoặc ứng dụng Postman hoặc ứng dụng Fiddler để có thể đọc request/respone)



Cấu trúc HTTP Message

Cấu trúc chung của Request/Response có 4 thành phần

* **Dòng đầu tiên** (một dòng): Cho biết thông tin yêu cầu hoặc trạng thái (thành công hay lỗi) của response.
* **Phần header** (các dòng sau dòng đầu tiên trước một dòng trắng): Nó là một tập hợp các dòng chứa thông tin về HTTP Message, thông tin về phần body
* **Một dòng trắng** cho biết phần thông tin (dòng đầu và header) đã gửi hết
* **Phần body** chứa dữ liệu đính kèm với request (như HTML Form) hoặc nội dung văn bản đính kèm cùng response. Kích cỡ dữ liệu này (size) có được xác định bởi thông tin trong header.

Mặc dù có cấu trúc chung nhưng về chi chi tiết Request và Response có những đặc điểm riêng.

**#HTTP Request Message**

* **Dòng đầu tiên**: Dòng này chứa thông tin để gửi tới server, dựa vào thông tin này mà server thực thi hành động phù hợp. Dòng này nó có chứa ba thông tin cách nhau bởi khoảng trắng, ví dụ như:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

POST /html/ HTTP/1.1

**HTTP Method** (*Phương thức HTTP*) là thành phần thứ nhất (ví dụ trên là POST), nó có giá trị như POST, GET, PUT. Nó cho biết yêu cầu cần được thực hiện trên server đối với một tài nguyên nào đó, cụ thể như các phương thức:

* + **GET** cho biết Request chỉ cần lấy dữ liệu (tài nguyên).
  + **POST** được sử dụng để cho biết Request là chuyển dữ liệu lên server (thương HTTP Message trường hợp này có kèm dữ liệu như HTML Form ...)
  + **PUT** ghi đè (thay thế) tài nguyên nào đó bằng dữ liệu trong Request.
  + **DELETE** xóa tài nguyên

**Địa chỉ URL** - là thành phần thứ hai (ví dụ trên là /html/). Địa chỉ tài nguyên truy vấn, có thể là URL tương đối - tuyệt đối (kể cả cổng, nếu có cổng thì viết cổng sau ký hiệu :)

**HTTP Version** Thành phần thứ ba cho biết phiên bản HTTP (thường là HTTP/1.1)

* **Header của Request**: Các header có cấu trúc đó là một chuỗi là tên header tiếp theo là dấu : và giá trị cho header. Mỗi header được viết trên một dòng. Có rất nhiều loại header (đã chuẩn hóa hoặc header do bạn tự đặt), có thể tham khảo danh sách các header tại [HTTP Request Header](https://en.wikipedia.org/wiki/List_of_HTTP_header_fields#Request_fields), ví dụ vài header như:**Host** là header chỉ ra host (domain, IP) và cổng của server mà Request gửi đến. Nếu không chỉ rõ port thì mặc định là 80 với http và 443 với https. Ví dụ:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

Host: xuanthulab.net

**Accept** trong Request cho biết kiểu nội dung trả về mà client có thể hiểu. (các kiểu cách nhau bởi dấu phẩy (,) có độ ưu tiên mặc định 1, nếu muốn xác định độ ưu tiên cho kiểu nào thì cho thêm ;q=value) Ví dụ:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

Accept: text/html, application/xhtml+xml, application/xml;q=0.9, image/webp, \*/\*;q=0.8

**User-Agent** là header của Request cho phép server xác định được ứng dụng nào, hệ điều hành gì gửi Request. Ví dụ:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:12.0) Gecko/20100101 Firefox/12.0

**Content-Length** cho biết độ dài (byte) của phần body (nếu có đính kèm trong request). Ví dụ:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

Content-Length: 12345

**Content-Type** trong Request, header này cho server biết kiểu dữ liệu được đính kèm trong phần body. Ví dụ:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

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

**Cookie** header này gửi giá trị cookie lấy được từ truy vấn trước để server phục hồi phiên làm việc. Có thể gửi nhiều cookie cách nhau bởi ;

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

Cookie: name=value; name2=value2; name3=value3

Trong ứng dụng PHP mặc định tên Cookie là PHPSESSID, vậy tại server php (backend) đọc được giá trị này - nó sẽ phục hồi Session.

* **Body của Request**: Chứa dữ liệu của Request (dữ liệu này có kiểu xác định ở header Content-Type và độ dài Content-Length), không phải mọi Request đều có body, dữ liệu body thường là HTML Form hay JSON.

**#HTTP Response Message**

* **Dòng đầu tiên**: còn gọi là dòng trạng thái, nó chứa ba mẩu tin gồm
  + Phiên bản HTTP, ví dụ

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

HTTP/1.1

* + Mã trả về như 200, 404, 301, 302 ...
  + Đoạn text ngắn giải thích mã ví dụ OK, Not Found, Moved Permanently ...

Đây là một dòng trạng thái

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

HTTP/1.1 400 Bad Request

Tham khảo một số Status code ở bảng này:

* **Header của Response**: tương tự như header của Request, các header thông dụng có thể tham khảo tại: [Một số header của Response](https://en.wikipedia.org/wiki/List_of_HTTP_header_fields#Response_fields) , ví dụ như: **Date** chứa thông tin ngày tháng thông tin (response) được phát sinh

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

Date: Wed, 21 Oct 2015 07:28:00 GMT

**Set-Cookie** header này chứa cookie của server gửi cho client. Client lưu lại để gửi lại cho server để phục hồi phiên làm việc.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

Set-Cookie: CookieID=94583405kfds; Max-Age=3600;

Một số giá trị thiết lập như:

* + Max-Age=number thời gian hết hạn cookie
  + HttpOnly không cho JS truy cập Cookie

Có những header giống header của Request như: Content-Type, Content-Length ...

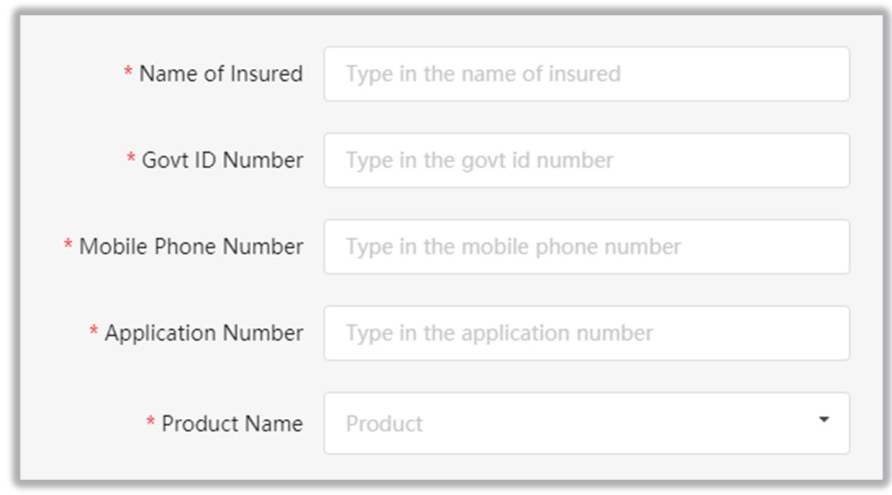
* **Body của Response**: Chứa dữ liệu của Respone (dữ liệu này có kiểu xác định ở header Content-Type và độ dài Content-Length), không phải mọi Response đều có body.

# #6. Cách thực hiện kiểm thử API

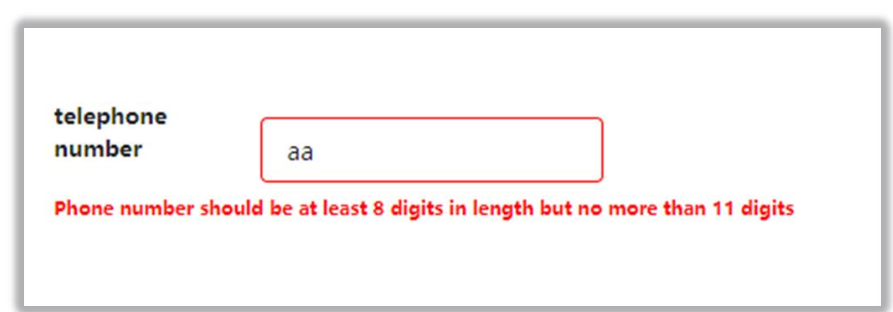
## #6.1. Syntax Testing

Để kiểm tra validation input của phía server thì ta có 1 số thứ cần phải check:

* **Require fields**: những field bắt buộc nhưng không gửi cùng request. Cái này sẽ trả về error code 400



**Validation rules**: với mỗi field thường có những điều kiện khác nhau, ta hãy cố nghĩ ra nhiều testcases phá vỡ cái rule đấy. Error code 400.



**Token**: nhiều API sẽ yêu cầu token, ta thử với trường hợp không có token hoặc token sai. Error code 401. Đừng nhầm lẫn với Error code 403

**#6.2. Functional Testing**

Đây là nơi check chức năng của mỗi API, bao gồm cả 2 loại testcases: positive và negative. Mỗi API có những chức năng khác nhau và phục vụ cho rất nhiều loại business khác nhau. Ví dụ:

* Caculation / counting
* Search / Filter
* States of process
* Access rights / Authorization
* …

Mình lấy trường hợp cụ thể của 1 API Filter Student theo 2 fileds: name, enrolled date from - to. Yêu cầu chỉ cán bộ nhà trường được quyền gọi API này, ta có 1 số testcases như sau:

* Filter Student đã có trong hệ thống theo name / enrolled date from / enrolled date to
* Filter Student không có trong hệ thống theo name
* Filter Student khi chọn enrolled date from > enrolled date to (có thể Error code 400 hoặc Error code 200 và trả về rỗng, tùy cách xử lý của team)
* Filter Student khi chọn enrolled date from < enrolled date to
* Cán bộ nhà trường xem toàn bộ Student
* Sinh viên xem toàn bộ Student (Error 403 code)
* ….(còn rất nhiều)

Functional Testing này có 2 câu hỏi:

1. **Khi test API có so sánh với DB không?** Mình nên làm việc với DB khi mình làm Manual API testing hoặc Manual UI Testing, còn Automation API testing thì mình không làm vì thực hiện query đến DB từ code sẽ khiến cho code test trở nên phức tạp; bị phụ thuộc vào DB → nhiều exception hơn.
2. **Có kiểm tra các trường hợp lỗi 5xx không?** Mình không expect và test các cases trả về Error code 5xx vì công sức tạo nên cái case đấy quá lớn (ví dụ như database error, phải tắt DB) hoặc tất cả các case đó là do handle exception không tốt → phải fix và return Error code 4xx.

**#6.3. Test Scenarios (Flow Testing)**

Hai loại Syntax Testing và Functional mục tiêu là test chức năng của từng API riêng lẻ, còn Flow Testing là nhằm mục đích để test việc gọi liên tiếp nhiều API theo trình tự tạo thành những flow cơ bản của App. Flow này được tạo theo dựa theo trình tự flow làm việc của End-user với UI.

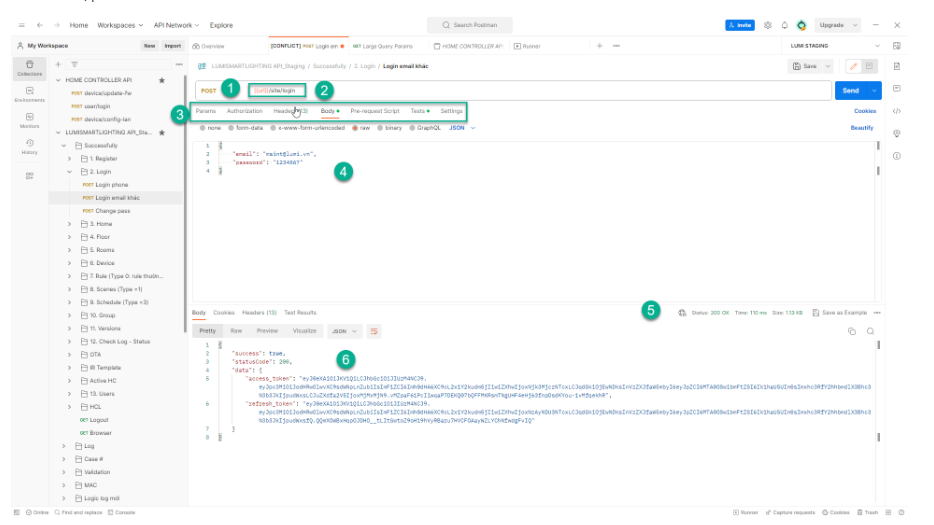
hia Flow Testing này thành 2 loại:

* Short flow: Chứa tất cả các flow của app, chia dựa theo chức năng.
* Long flow: Chứa 1 flow dài của App, chính là phần cốt lõi của App.

# #7. Cách viết testcase API

Để viết testcase cho API, cần đảm bảo các tiêu chí sau:

* Kiểm tra cú pháp: dựa trên điều kiện đầu vào
  + Kiểm tra trường bắt buộc/tùy chọn
  + Nhập sai định dạng/độ dài
  + Nhập sai logic (ngày bắt đầu > ngày kết thúc…)
  + Nhập bằng chữ hoa/chữ thường
  + Dữ liệu nhập vào không hợp lệ
  + Method sai
  + Chạy API mà không có Anthentication, không có Authorization
  + Chạy API với url sai
* Kiểm tra chức năng: dựa trên yêu cầu nghiệp vụ
  + Xác minh phản hồi cho từng trường trả về với ánh xạ dữ liệu từ máy chủ/hoặc máy tính khác chính xác
  + Xác minh trong trường hợp không có dữ liệu trả về từ Backend
  + Xác minh xử lý mã lỗi
  + Xác minh thêm/cập nhật bản ghi trên DB
* Xác minh cấu trúc
* Tích hợp API

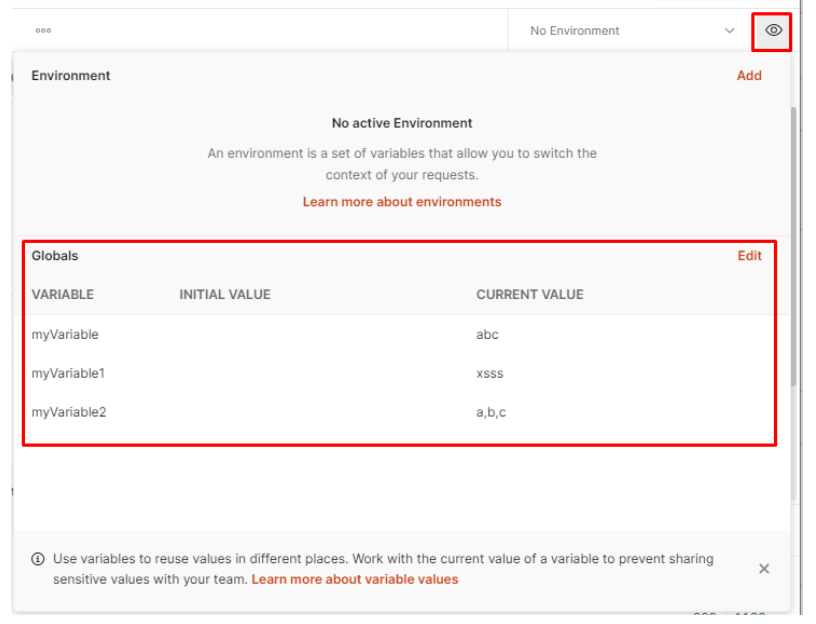


# #8. Sử dụng các biến để viết scripts test

## #****8.1. Cách sử dụng các loại biến trong postman script****

Trong postman có nhiều loại biến với các mục đích khác nhau, bạn nên biết những function cơ bản với những loại biến này để viết script được nhanh và pro hơn.

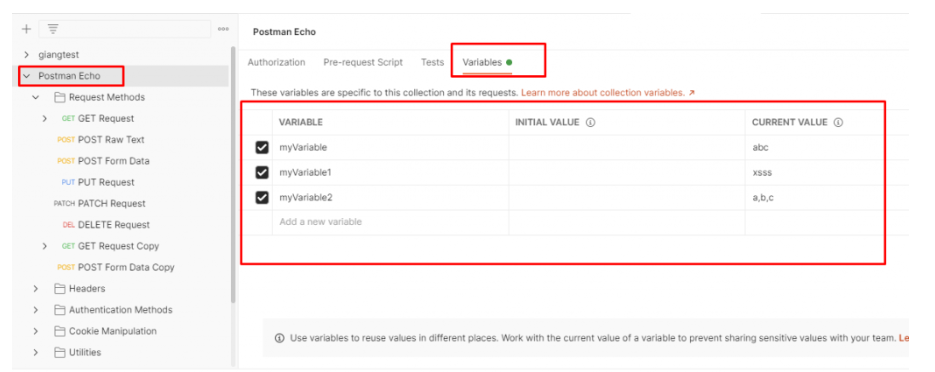
### #a. Biến Global



**Cách dùng:**

|  |  |
| --- | --- |
| **Goals** | Nhiều mục đích, chủ yếu để quick test, và tất cả các request đều có thể sử dụng. |
| **Set** | pm.globals.set(‘myVariable’, MY\_VALUE); |
| **Get** | pm.globals.get(‘myVariable’); |
| **Remove** | pm.globals.unset(‘myVariable’); |
| **Clear** | pm.globals.clear(); |

### #b. Biến Collection



**Cách dùng:**

|  |  |
| --- | --- |
| **Goals** | Chỉ có sử dụng với các request nằm trong 1 collection, dùng để lưu trữ những biến mà sẽ sử dụng trong quá trình run test mà không khác nhau giữa các môi trường với nhau: dev, test, staging. |
| **Set** | pm.collectionVariables.set(‘myVariable’, MY\_VALUE); |
| **Get** | pm.collectionVariables.get(‘myVariable’); |
| **Remove** | pm.collectionVariables.unset(‘myVariable’); |
| **Clear** | pm.collectionVariables.clear(); |

### #c. Biến Envionment

**Cách dùng:**

|  |  |
| --- | --- |
| **Goals** | Dùng riêng cho các môi trường khác nhau của app, có thể lưu các biến mà khác nhau ở mỗi môi trường như URL, username/password, hoặc các data mà dùng cho các request tiếp theo. |
| **Set** | pm.environment.set(‘myVariable’, MY\_VALUE); |
| **Get** | pm.environment.get(‘myVariable’); |
| **Remove** | pm.environment.unset(‘myVariable’); |
| **Clear** | pm.environment.clear(); |
| **Show name** | pm.environment.name |

### #d. Biến Local

Biến này chỉ khởi tạo khi ấn request bắt đầu và clear toàn bộ khi finish khi request kết thúc.

**Cách dùng:**

|  |  |
| --- | --- |
| **Goals** | Chỉ dùng trong vòng đời của 1 request |
| **Set** | pm.variables.set(‘myVariable’, MY\_VALUE); |
| **Get** | pm.variables.get(‘myVariable’); |
| **Remove** | pm.variables.unset(‘myVariable’); |
| **Clear** | pm.variables.clear(); |

### #e. Biến Interation

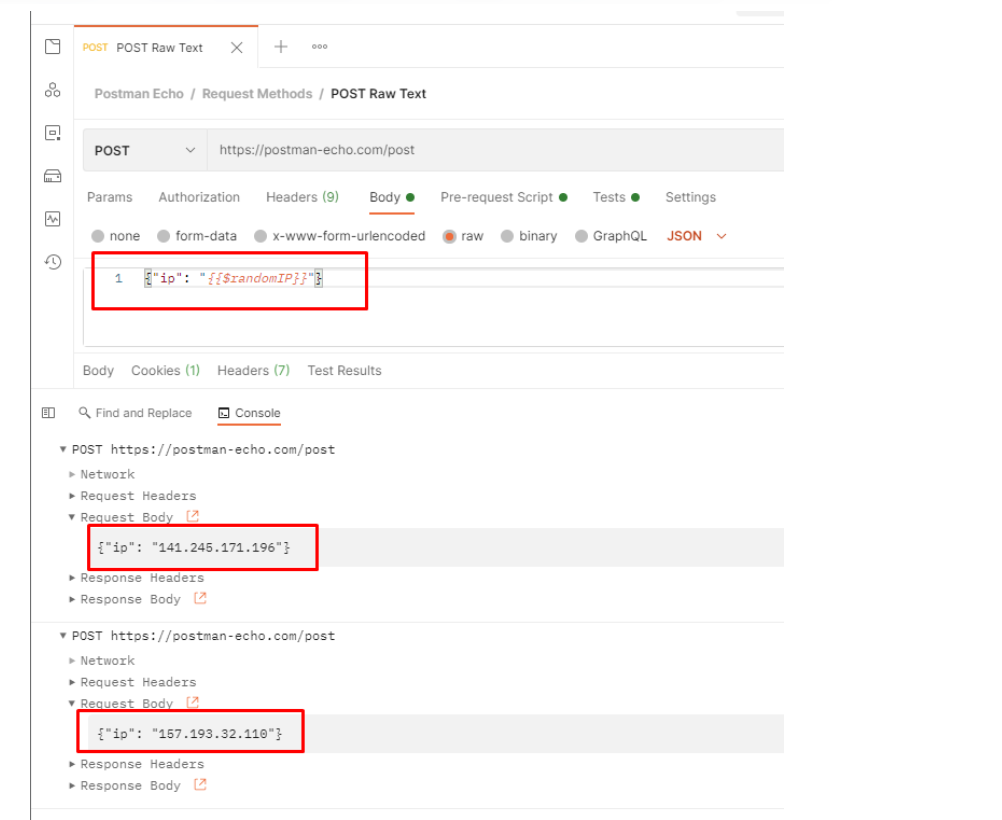
Đọc thông tin từ file data dạng csv hoặc json.

**Cách dùng:**

|  |  |
| --- | --- |
| **Goals** | Chỉ work khi dùng với chức năng đọc file data từ CSV hoặc JSON file từ Runner hoặc Newman |
| **Get** | pm.iterationData.get(‘myVariable’); |

## #8.2. Cách lấy giá trị các biến Dynamic

Một thời gian dài trước đây, team postman nhận rất nhiều phản hồi của người dùng về việc thêm thư viện để fake data vào trong postman. Và cuối cùng, team postman đã thỏa mãn người dùng bằng việc cho thêm vào postman rất nhiều biến dynamic, có nghĩa là dummy data mỗi lần run. Ví dụ mỗi lần run, mình lại có 1 random ip.



### #a. Cách gọi biến ngoài script

Sử dụng cú pháp:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

{{$ten\_bien}} //Ví dụ: {{$randomIP}}

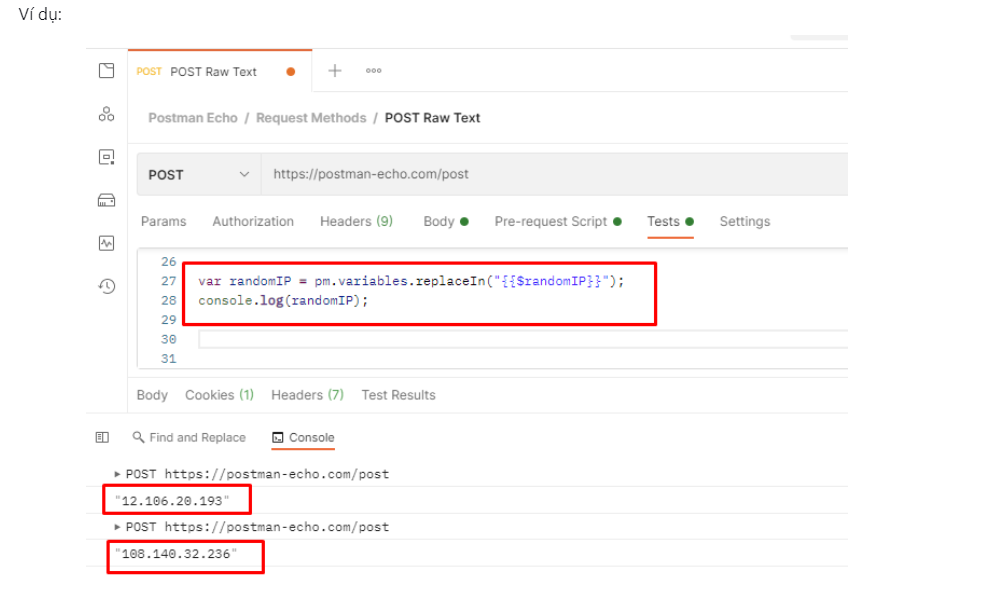
Full list các biến dạng dynamic có ở [đây](https://learning.postman.com/docs/writing-scripts/script-references/variables-list/).

### #b. Cách gọi biến trong script (Pre-request và Tests)

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

pm.variables.replaceIn("{{$randomIP}}");

Ví dụ:



# #9. Work flow , scenarios và Authentication/Authorization

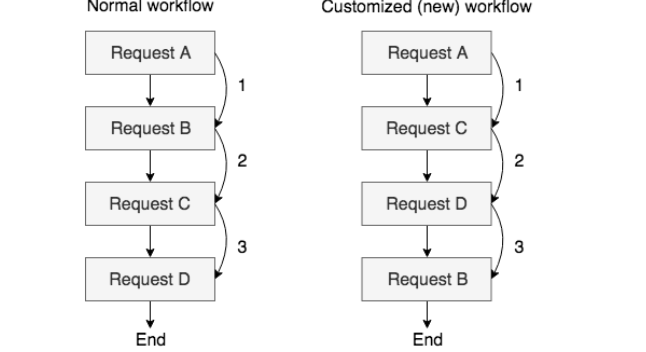
## #9.1. Workflow điều khiển thứ tự chạy request

**Workflow** là thứ tự của các request được sắp xếp để thực hiện một luồng chức năng nào đó như mong muốn. Chúng ta có thể điều khiển thứ tự các request trên Postman thay vì để chạy mặc định từ trên xuống dưới thông qua hàm setNextRequest.

### #Next Request

**Normal workflow:**  
Request A > Request B > Request C > Request D

**Customized (new) workflow**  
Request A > Request C > Request D > Request B và kết thúc.



Để thực hiện một workflow mới theo thứ tự này chúng ta sẽ chèn **postman.setNextRequest(’Ten Request’);** vào phần Tests như ảnh:

Thứ tự của các request và Tests script sẽ như sau.

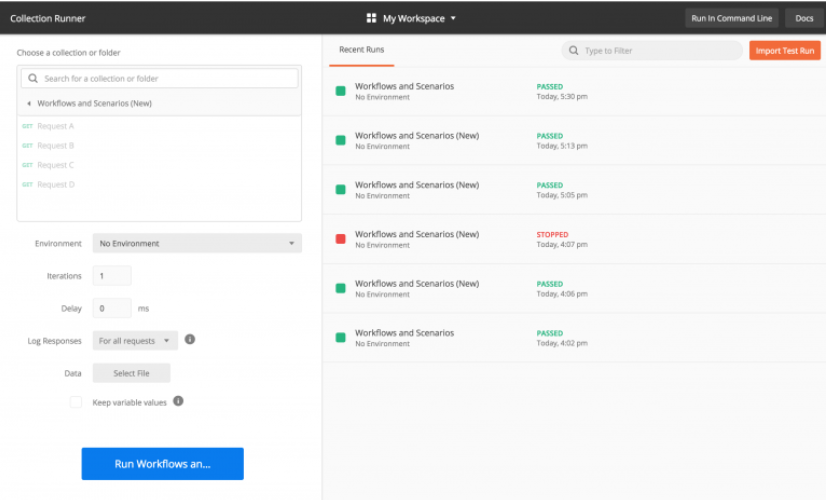
**Request A:** postman.setNextRequest(‘Request C’);  
(chạy request A đầu tiên sau đó nhảy đến Request C)

**Request C:** postman.setNextRequest(‘Request D’);  
(Sau khi chạy C sẽ chuyển qua D)

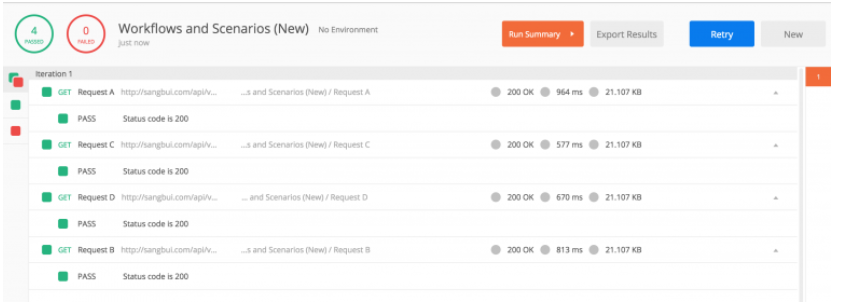
**Request D:** postman.setNextRequest(‘Request B’);  
(Sau khi chạy D sẽ chuyển qua B)

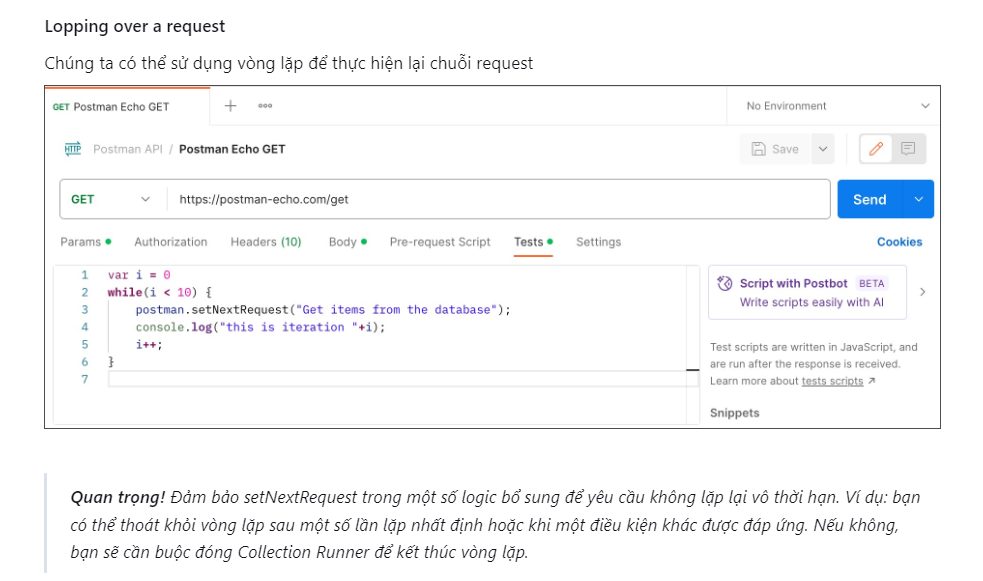
**Request B:** postman.setNextRequest(null);  
(Sau khi thực hiện request B sẽ kết thúc test (next request = null))

Để chạy thử chúng ta sẽ chọn nút **Runner**, chọn collection bạn muốn chạy và cuối cùng là chọn **Run**



Kết quả chạy theo thứ tự như mong muốn: Request A > Request C > Request D > Request B và kết thúc.





### #Dừng workflow

Để dừng một workflow, thêm câu lệnh bên dưới vào tab Tests của request

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

postman.setNextRequest(null);

Collection đang chạy sẽ dừng sau khi Postman hoàn thành yêu cầu hiện tại