



# API TESTING INTRODUCTION

2020

For Vietnamese testers

Giang Nguyen  
Nguyenduygiang2508@gmail.com

# Lời nói đầu

Xin chào!

Mình là Giang Nguyễn, một tester bình thường, có hứng thú với testing và technology.

Có thể bạn đã đọc cuốn “API Testing với Postman” trước của mình, nhưng cảm thấy chưa thỏa mãn, bạn muốn 1 cuốn sách với nhiều thông tin hơn, đầy đủ hơn. Phải nói thật là chính mình cũng muốn vậy, do đó mình quyết định viết lại và update nhiều nhiều thứ hơn nữa. Khác với cuốn trước, cuốn này mình chỉ nói về API Testing, sẽ không có phần hướng dẫn về Postman hay bất cứ 1 tool nào cả. Mình sẽ viết từng tool ở các cuốn ebook khác nhau (nếu có thời gian: D).

Cuốn sách này là kết quả của nghiên cứu, thực hành của mình về API trong mấy năm đi làm vừa qua. Tất nhiên là sẽ còn những thiếu sót và điểm chưa đúng lắm, nhưng nó sẽ phù hợp cho những bạn tester beginner tiếp cận và làm việc với API Testing. Tuy nhiên, beginner không có nghĩa là bạn tester nào mới cũng có thể hiểu, bạn nên biết một số khái niệm về Software Development để có thể hiểu hoàn toàn những thứ mình viết.

Cuốn sách này mình free cho testers dùng nó để self-study, vui lòng không sử dụng nó cho mục đích thương mại, ví dụ dùng nó làm giáo trình, hoặc quảng cáo cho khóa học của bạn. Tất nhiên, chắc chắn có ai làm như vậy, nhưng nếu bạn có sử dụng 1 phần nào đó của sách (ở bất kỳ hình thức nào) thì vui lòng reference về sách của mình. Cảm ơn!

Cuối cùng, hi vọng cuốn sách có ích cho bạn và giúp đỡ bạn khi test API.

Have fun and enjoy!

P/s: Xin lỗi vì văn phong lung tung và cách viết xen lẫn tiếng anh và tiếng việt, mình cảm thấy dễ thể hiện ý kiến của mình nhất khi viết như vậy.

# CONTENT

<b>Chapter I: API Testing high-level</b>	4
1. Where is API Testing in test pyramid?	4
2. Why, What, When to test?	5
2.1 Why API Testing is important?	5
2.2 What to test	5
2.3 When to test	5
3. API Testing in CI/CD pipeline	6
4. Structure của 1 test API	6
4.1 Arrange-Act-Assert: AAA pattern	7
4.2 Given-When-Then: BDD style	7
<b>Chapter II: API Introduction</b>	8
5. What is API?	8
5.1 Interface in real life?	8
5.2 Application programming interface (API)	8
6. API Structure	9
6.1. What is protocol?	9
6.2. How does HTTP work?	9
6.3 API - Under the GUI	12
7. Data format	13
7.1. JSON format	13
7.2. XML format	13
7.3. How to use data format in HTTP	14
8. Authentication	15
8.1 Basic Authentication	15
8.2 API Key Authentication	15
8.3 OAuth2	15
<b>Chapter III: API Automation Test</b>	17

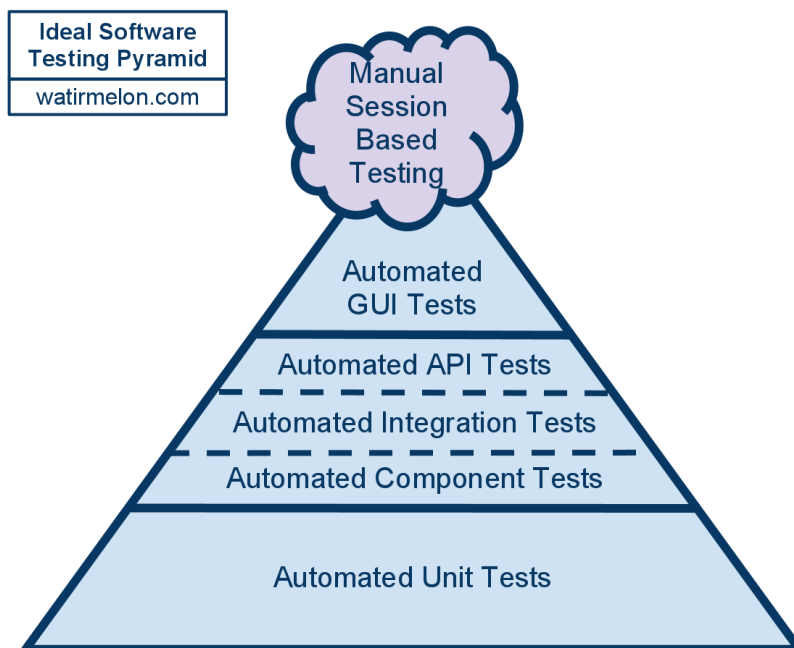
9. How to test an/many API(s)?	17
9.1 Syntax Testing	17
9.2 Functional Testing	18
9.3 Flow Testing	18
10. Challenges	20
10.1 Types of API	20
10.2 Choose Error code to expect	22
10.3 Prepare test data	23
10.4 Assert Value	23
<b>Reference</b>	25

# Chapter I: API Testing high-level

## 1. Where is API Testing in test pyramid?

Test pyramid là 1 khái niệm được sử dụng khi nói về các level của automation test với 3 levels chính: Unit test, Integration/Service test, UI test. Trong đó, API Test nằm trong Integration test. Để nói về pyramid testing thì các bạn có thể đọc thêm ở [đây](#). Khi Unit Test và UI Test rất rõ ràng để phân biệt thì Integration Test lại rất khó hiểu và dẫn đến nhiều hiểu nhầm. Rất may mắn là Martin Fowler cũng có 1 [bài](#) để phân biệt, và ta có thể xếp API Test vào loại **broad integration tests** với những đặc điểm:

- Yêu cầu live version của tất cả các services, test environment và network access.
- Test code paths đi qua tất cả các services, chứ không chỉ test connection giữa chúng.



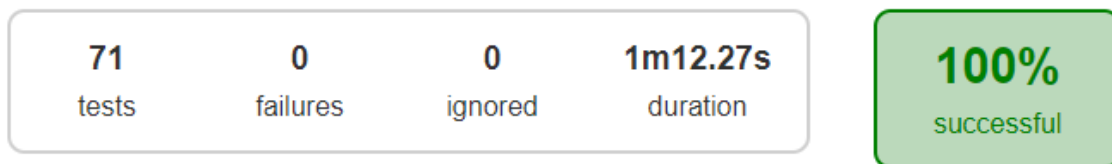
Có thể bạn sẽ nói là, bạn sẽ có thể mock service nào đó và vẫn thực hiện test API được mà, đâu cần phải live version đâu. Oh...well...Actually, you're right. Thực tế thì ta viết test API khi có API specification rồi, nên lúc ta run test, có thể đang dùng mock service chứ không phải là live service. Định nghĩa ở trên, đó là điều kiện lý tưởng nhưng không phải lúc nào ta cũng có thể đáp ứng được, nên đoạn này bạn tự linh động cho nhu cầu của mình.

## 2. Why, What, When to test?

### 2.1 Why API Testing is important?

- Tăng độ bao phủ của test, cover những cases mà UI test không chạm tới được hoặc rất tốn công để viết test, như validation input của phía server hoặc là những test liên quan đến filter hoặc search. Ví dụ: field email có validation cả front-end và back-end, nếu sử dụng UI test thì bạn chỉ chạm được vào validation của front-end, bạn sẽ phải dùng API Test để check validation của back-end.
- API Test được thực hiện trong thời gian ngắn và sẽ có feedback rất sớm, thay vì 30p hay vài tiếng như UI test. Ví dụ như hình dưới đây: 71 API test cases (request) có thời gian run là 1m12s

### Test Summary



### 2.2 What to test

API Test có 3 đối tượng chính, cũng có thể coi là 3 categories để chúng ta gom và nhóm các test cases lại.

- Kiểm tra validation input ở phía server
- Kiểm tra logic của từng chức năng
- Đảm bảo các flow của ứng dụng hoạt động tốt.

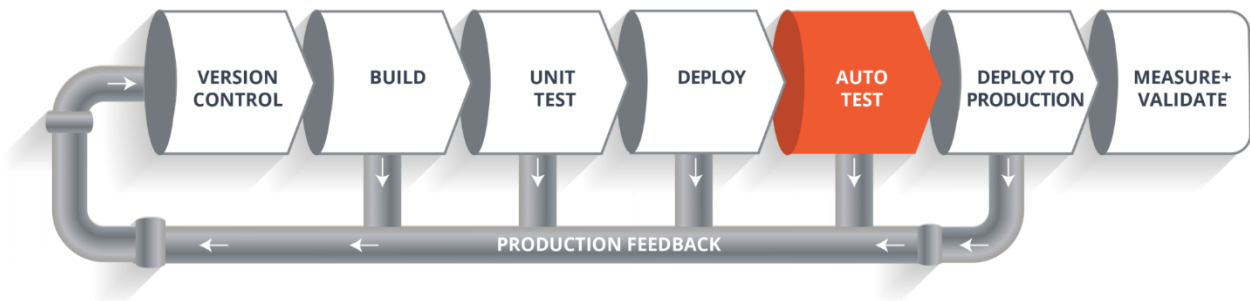
Ngoài ra, bạn cũng có thể add thêm các testcase cho các test khác ví dụ như security. Với mình thì API test chỉ tập trung vào functional test nên mình không add thêm các test cases của phần khác vào.

### 2.3 When to test

Bạn nên làm API Test trong những trường hợp sau:

- Được yêu cầu làm API Test từ khách hàng hoặc PM
- Được yêu cầu làm Automation Test nhưng lại không có đủ thời gian để làm cả API lẫn UI.
- Khi muốn test sớm các chức năng nhưng UI chưa làm xong.
- Khi dự án của bạn là web service, bạn không có UI để test.

### 3. API Testing in CI/CD pipeline



Trên pipeline ta có thể nhận thấy API Test sẽ nằm ở phần Auto Test, sau bước Deploy nhưng nó sẽ run trước Automation UI Test. Vì như thế nó vừa đảm bảo điều kiện: test khi tất cả các service đã run và sẽ cho kết quả trong vòng 5-10p sau khi deploy. Nếu tất cả các test pass thì mới chuyển đến phần run Automation UI test, nếu fail thì không tiếp tục UI test.

Do API Test cũng là 1 dạng black-box test và không hề cần sử dụng code của ứng dụng nên ta có thể chọn bất kỳ tools nào mà ta muốn. Ta có thể kể đến 1 số tools thường dùng như Postman, Soap UI, Jmeter, Rest-Assured, Karate, Robot Framework, Katalon Studio... Và script của API Test, ta không nhất thiết phải đặt vào trong repo của ứng dụng, ta có thể đặt ở 1 repo riêng rẽ, độc lập.

API Test nên được tự động run bằng việc sử dụng CI tools như Jenkins, Travis CI...

### 4. Structure của 1 test API

Một Automation Test thường bao gồm 3 phần:

- **Setup data test:** chuẩn bị pre-condition cần thiết như mock data, login vào hệ thống, đọc file csv data test...
- **Do actions:** Trong Unit là gọi method/function, trong API test là call API, còn trong UI test là điền thông tin, click button...
- **Check result:** So sánh cái kết quả mong muốn và kết quả thực tế.

API test cũng có 3 phần như trên, và bạn có thể chọn 1 trong 2 cách dưới đây để thể hiện:

- [AAA pattern](#): thường được sử dụng cho Unit Test
- BDD style: thường được sử dụng cho User Acceptance Test, theo phong cách viết của [Gherkin](#)

## 4.1 Arrange-Act-Assert: AAA pattern

Ý nghĩa của AAA pattern khá đơn giản:

- Arrange = Setup data test
- Act = Do actions
- Assert = Check result

```
void registerFailWithMissingParams(String fullName, String emailAddress, String companyName,
                                   String jobTitle, String phoneNumber, String requestRoles) {  
  
    reg.setFullName(fullName);  
    reg.setEmailAddress(emailAddress);  
    reg.setCompanyName(companyName);  
    reg.setJobTitle(jobTitle);  
    reg.setPhoneNumber(phoneNumber);  
    reg.setRequestRoles(requestRoles);  
  
    Response res = callAPI(reg);  
  
    res.prettyPrint();  
    res.then().statusCode(400);  
}
```

Arrange

Act

Assert

## 4.2 Given-When-Then: BDD style

Ví dụ dưới đây sử dụng Rest-Assured, trong đó:

- Given = Setup Pre-condition/ Setup data
- When = Do actions
- Then = Compare expected value with actual value

```
@Test public void  
getHelloWithParam() {  
    given().  
        param("name", "coder").  
    when().  
        get(HELLO).  
    then().  
        statusCode(HttpStatus.SC_OK).  
        contentType("application/json").  
        body(equalTo("Hello coder!"));  
}
```



## Chapter II: API Introduction

### 5. What is API?

#### 5.1 Interface in real life?

Interface dịch ra tiếng Việt là “bề mặt, giao diện”, nhưng nó không phải cái mọi người hay dùng với nghĩa “mặt mũi đẹp hay xấu”. =))) Interface được sử dụng để giao tiếp giữa 2 đối tượng, đồng thời làm cho việc giao tiếp này đơn giản nhất có thể bằng việc giấu kín việc xử lý chi tiết của mỗi đối tượng.



Đây là ví dụ Interface của xe máy, người lái xe bấm vào còi, chỉ cần biết là còi sẽ kêu, còn hoạt động bên trong xe máy thế nào để còi kêu thì người lái xe không cần biết.

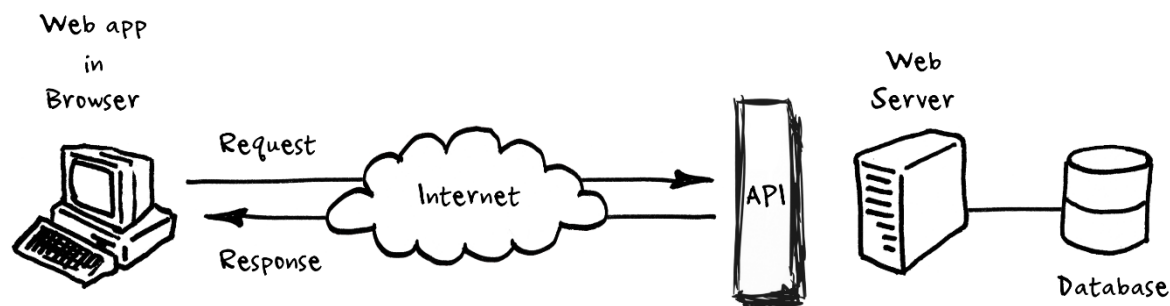


Hoặc bạn có 1 cái case máy tính và 1 màn hình, 2 cái đó muốn kết nối với nhau thì bắt buộc phải thông qua 1 dây nối với 2 đầu tiếp xúc. Hai cái đầu tiếp xúc đó chính là API. Điểm tiếp xúc đó là cố định, nếu 1 màn hình khác muốn sử dụng thì cũng phải có đầu tiếp giống như thế, nếu không là không kết nối được.

**Chốt lại, Interface là 1 bộ các quy tắc để 2 đối tượng có thể giao tiếp được với nhau mà không cần biết chi tiết là mỗi đối tượng sẽ làm gì với mỗi yêu cầu.**

#### 5.2 Application programming interface (API)

API chính là Interface của các đối tượng phần mềm. Nó được sử dụng rộng rãi khi nói về mô hình Client-Server và khi nói về cách sử dụng các library (các public method cũng được coi là API).



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, Nodejs, Java. Tương tự, server back-end cũng được viết bằng các ngôn ngữ khác nhau. Để 2 thành phần này có thể nói chuyện được với nhau chúng phải nói cùng 1 “ngôn ngữ”. Ngôn ngữ ấy chính là API.

## 6. API Structure

### 6.1. What is protocol?

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.

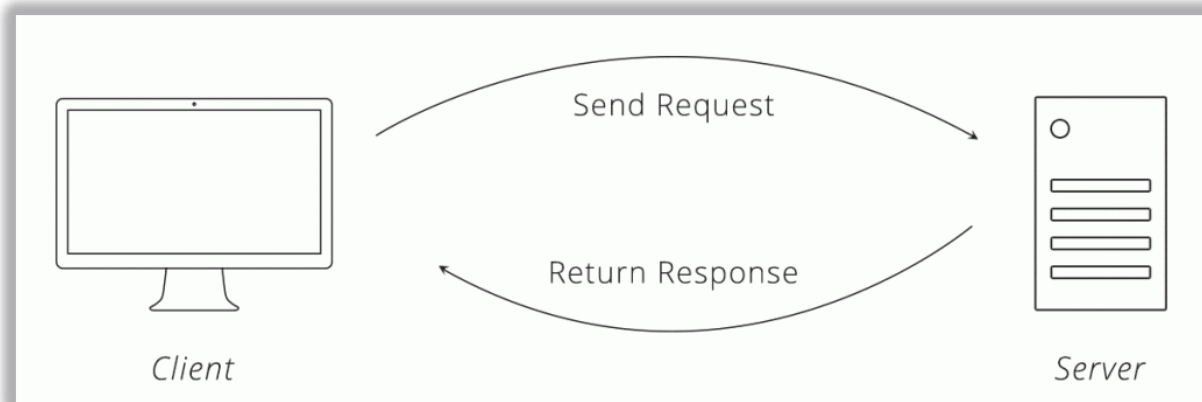
*Giao thức (Protocol) là những luật lệ được chấp thuận để 2 cái máy tính có thể nói chuyện với nhau.*

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.

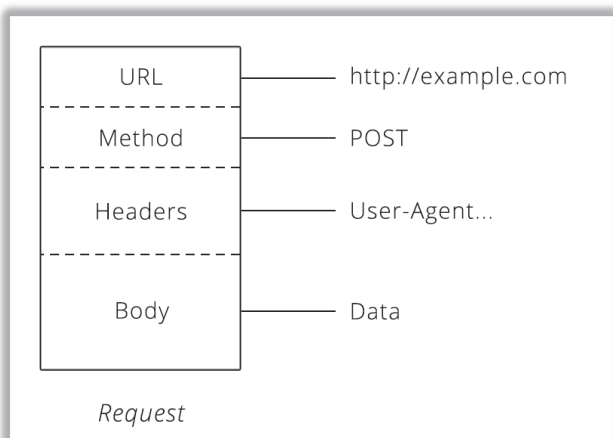
Chúng ta đã từng nghe đến những Protocol cho những mục đích khác nhau, ví dụ như Mail có POP hay IMAP, message có XMPP, Kết nối thiết bị: Bluetooth. Trong web thì Protocol chính là HTTP – HyperText Transfer Protocol, vì sự phổ biến của nó mà hầu hết các công ty chọn nó là giao thức cho các API.

### 6.2. How does HTTP work?

Cuộc sống của HTTP xoay quanh cái vòng luân quần: 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 Response.



## Request



Một cái request đúng chuẩn cần có 4 thứ:

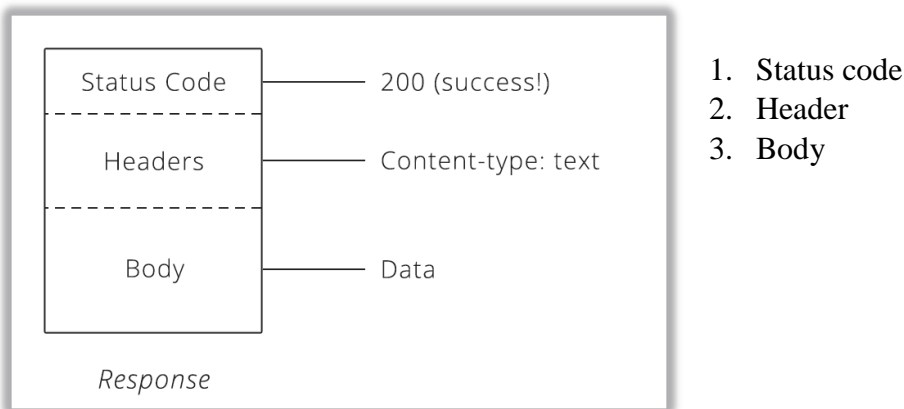
1. URL
2. Method
3. Header
4. Body(except GET)

- **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 thông tin của 1 resource. Bạn hãy tưởng tượng ra cái cảnh bạn vào facebook, tay vuốt new feeds.
  - POST: Yêu cầu server cho tạo ra 1 resource mới. Ví dụ: đăng ký 1 chuyến đi ở GrabBike. Hoặc đôi khi nó dùng để gửi thông tin lên server khi mà GET method không đáp ứng được (do hạn chế bảo độ dài của URL khi có sử dụng nhiều parameters) và POST sẽ an toàn hơn khi thông tin được đặt vào body chứ không phải show trên URL.
  - PUT: Yêu cầu server cho sửa vào 1 resource đã có trên hệ thống. Ví dụ: Edit 1 bài viết trên facebook.
  - DELETE: Yêu cầu server xóa 1 resource.

- **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 response. Cấu trúc của 1 response tương đối giống phần request nhưng Status code sẽ thay thế cho URL và Method.



- **Status code** là những con số có 3 chữ số và có duy nhất 1 ý nghĩa. Nó có thể coi là short answer cho request, mình sẽ còn quay lại status code ở chapter 3.

1xx: Informational responses

2xx: Successful responses (200, 201)

3xx: Redirects

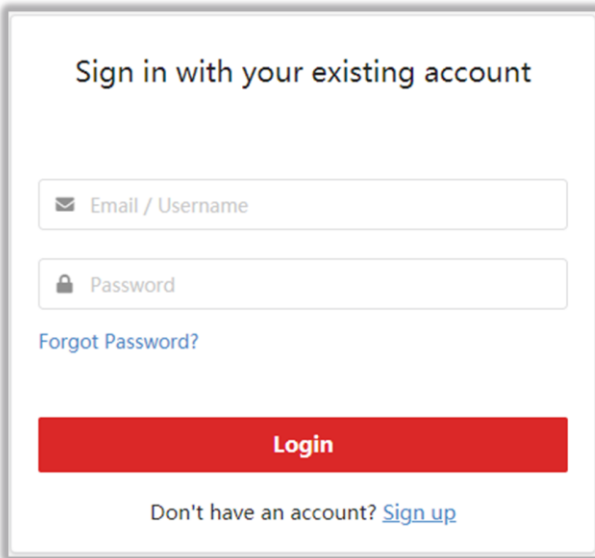
4xx: Client errors (400, 401, 403)

5xx: Server errors

- **Header & Body** thì tương đối giống với request.

### 6.3 API - Under the GUI

Câu hỏi mình hay được nhận nhất kể từ khi viết blog đến giờ “Anh ơi, em đọc rất nhiều posts về API của anh, cũng hiểu hiểu, nhưng em vẫn chưa biết phải test những cái gì?”. Lý do có thể là bạn chưa móc nối được liên hệ giữa API và UI. Thực ra thì API và UI “gần như” là một, API là bản thể còn UI là biểu hiện.



POST /api/authentication/login HTTP/1.1

Host: xxx.com

Connection: keep-alive

Content-Length: 41

Accept: application/json, text/plain, \*/\*

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)

Content-Type: application/json; charset=UTF-8

```
{
  "username": "",
  "password": ""
}
```

Trên UI, bạn nhìn thấy 2 fields là username và password, trên API username & password cũng xuất hiện ở phần body. Bạn có thể viết testcase cho form login, thì rất đơn giản bạn cũng viết testcase tương tự cho API login. Tất nhiên là sẽ có vô vàn trường hợp UI và API không thể mapping 1:1, API chứa nhiều thông tin hơn UI, do đó, cũng cần viết thêm nhiều testcase hơn. Chi tiết các loại testcases sẽ được viết ở chương mục 9.

## 7. Data format

Định dạng data trong API thường dùng 2 loại chính là JSON (JavaScript Object Notation) và XML (Extensible Markup Language).

### 7.1. JSON format

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:

```
1 {  
2   "firstName": "John",  
3   "lastName" : "doe",  
4   "age"      : 26,  
5   "address"  : {  
6     "streetAddress": "naist street",  
7     "city"         : "Nara",  
8     "postalCode"  : "630-0192"  
9   },  
10  "phoneNumbers": [  
11    {  
12      "type" : "iPhone",  
13      "number": "0123-4567-8888"  
14    },  
15    {  
16      "type" : "home",  
17      "number": "0123-4567-8910"  
18    }  
19  ]  
20 }
```



The diagram shows a JSON object with several nested structures. Two orange arrows point to specific parts of the JSON: one points to the key "type" in the first phone number object, and the other points to the value "0123-4567-8910" for the same key. Below the arrows, the words "Key" and "Value" are written in red text.

- Object - { }
- Array - [ ]
- String - ""
- Number - 20
- Boolean - true/false
- Null

## 7.2. XML format

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

Ví dụ:

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

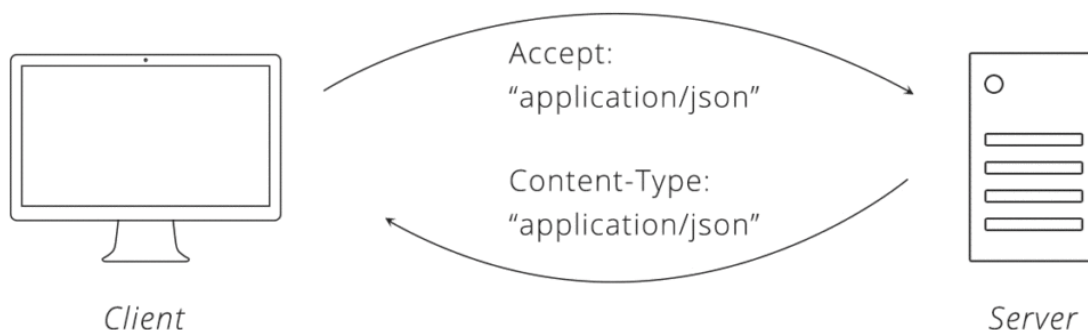
node mở      value      node đóng

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

## 7.3. How to use data format in 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.

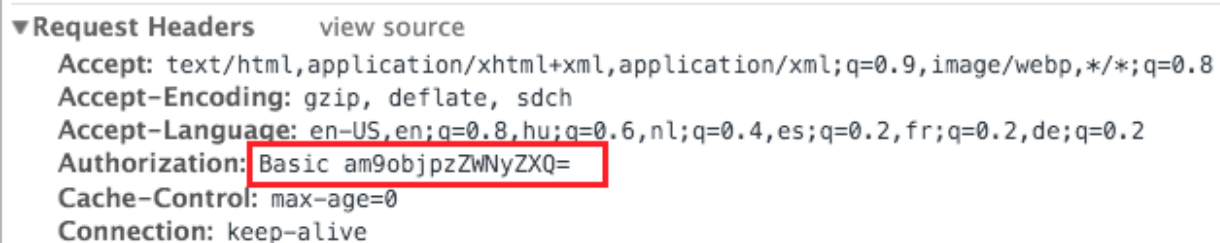
## 8. Authentication

Xin nói trước, đây không phải Security test nên mình không giải thích nhiều, bạn muốn biết thêm thì tự tìm hiểu tiếp. Mục đích của phần này là giới thiệu và hướng dẫn làm việc với API mà có sử dụng các chuẩn Authentication thông thường, bao gồm:

- Basic Authentication
- API Key Authentication
- OAuth2

### 8.1 Basic Authentication

Basic Auth thì chỉ yêu cầu username và password thôi. Client nhập 2 thông tin trên rồi gửi chúng qua HTTP header cho server. Khi server nhận được 1 request, nó sẽ soi vào Authorization header và so sánh thông tin đó với thông tin Credential mà chúng cất giữ ở DB. Nếu đúng, server sẽ chấp thuận request của client và trả thêm các thông mà client yêu cầu ở phần Body. Nếu không đúng, server sẽ trả lại mã code 401, báo hiệu rằng quá trình xác thực fail và yêu cầu bị từ chối.



The screenshot shows a list of HTTP request headers. The 'Authorization' header is highlighted with a red box and contains the value 'Basic am9objpzZWNyZXQ='. Other headers include 'Accept', 'Accept-Encoding', 'Accept-Language', 'Cache-Control', and 'Connection'.

```
▼ Request Headers    view source
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,hu;q=0.6,nl;q=0.4,es;q=0.2,fr;q=0.2,de;q=0.2
Authorization: Basic am9objpzZWNyZXQ=
Cache-Control: max-age=0
Connection: keep-alive
```

Username và password sẽ được encode dưới dạng Base64. Bạn có thể dùng [site này](#) để decode/encode Base64.

### 8.2 API Key Authentication

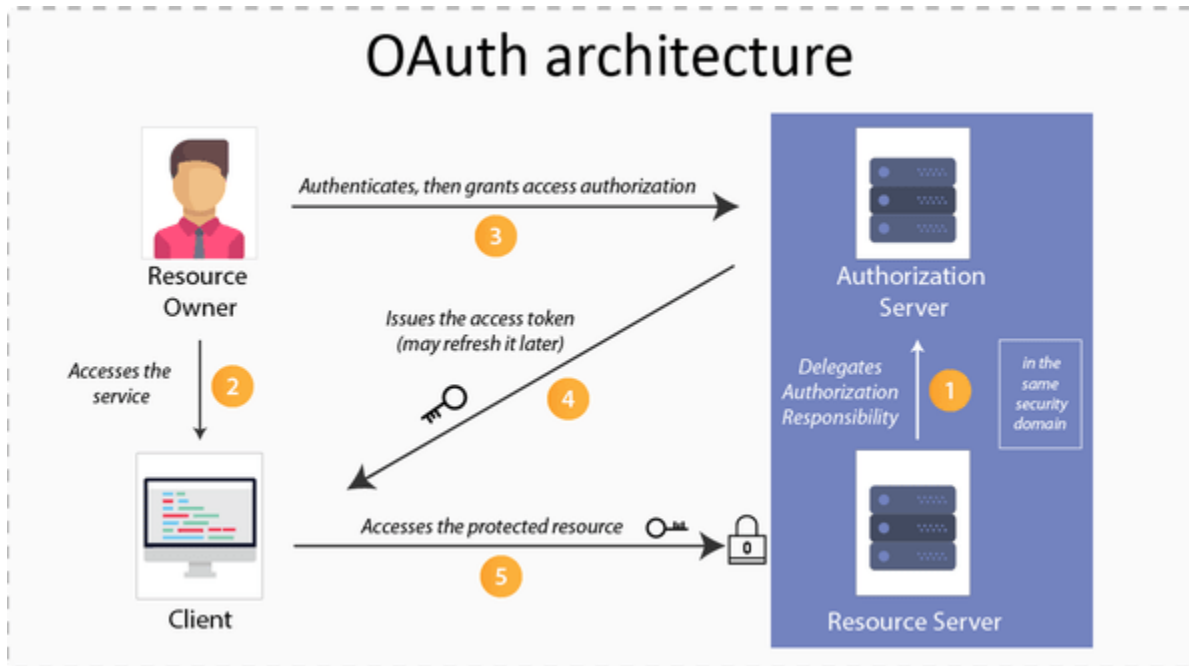
API Key Authentication là 1 kỹ thuật giúp xử lý điểm yếu (luôn gửi username:password) của mô hình Basic Auth ở phía trên. Khi Client xác thực với API Key, server sẽ biết để đồng ý cho client truy cập tới resource. API key nằm ở vị trí mà người lập trình mong muốn vì không có chuẩn nào cả. Có thể đặt nó trên header, trên URL ([http://example.com?api\\_key=my\\_secret\\_key](http://example.com?api_key=my_secret_key)), hoặc là ở Body. Và cho dù có đặt chúng ở đâu đi chăng nữa, chúng cũng sẽ có cùng 1 tác dụng.



### 8.3 OAuth2

OAuth là 1 framework, định nghĩa giao thức xác định cách mà access\_token được trao đổi giữa các bên. Nó cung cấp 4 flows làm việc phục vụ cho những đối tượng khác nhau:

- Authorization Code
- Resource Owner Password Credentials
- Implicit
- Client Credentials



Token (access\_token) là cái key mà sẽ giúp ta có thể access đến resource. Trong API test thì hầu như API nào cũng cần đến token. Ngoài ra thì còn 1 loại token nữa: refresh\_token, nhằm mục đích lấy access\_token mới khi access\_token bị expired, mà không cần nhập lại username:password.

Trong OAuth2 không chỉ định format của token mà để cho developers tự quyết định. Rất nhiều developers chọn sử dụng JWT (Json Web Token) format. JWT gồm 3 phần:

- Header: loại token và thuật toán mã hóa mà nó sử dụng
- Payload: chứa data
- Signature: để verify token

Chốt lại, để làm việc với API có sử dụng OAuth2 thì ta phải lấy được Access\_token, mà để lấy được thì ta phải biết là Application của mình đang sử dụng flow nào trong 4 flows ở trên.

## Chapter III: API Automation Test

### 9. How to test an/many API(s)?

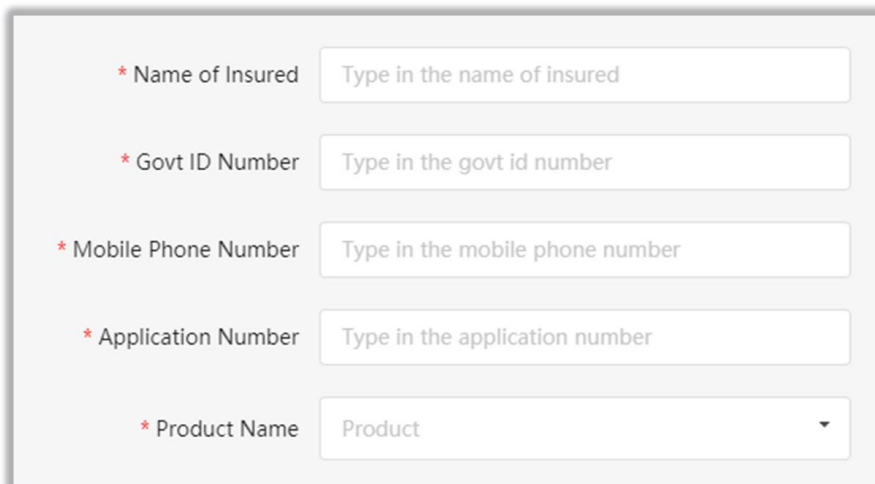
Mục 2.2 đã nói về đối tượng của API Testing, đó là:

- **Syntax Testing:** Kiểm tra validation input ở phía server
- **Functional Testing:** Kiểm tra logic của từng chức năng
- **Flow Testing:** Đảm bảo các flow của ứng dụng hoạt động tốt.

#### 9.1 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



\* Name of Insured Type in the name of insured

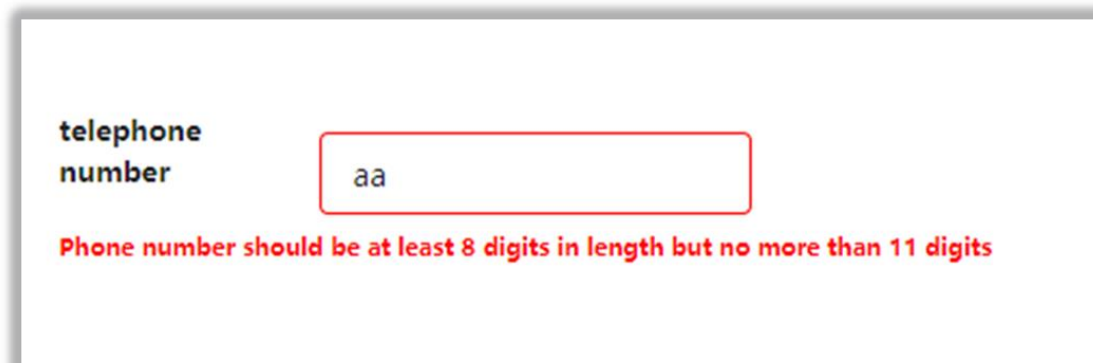
\* Govt ID Number Type in the govt id number

\* Mobile Phone Number Type in the mobile phone number

\* Application Number Type in the application number

\* Product Name Product

- **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.



telephone number aa

Phone number should be at least 8 digits in length but no more than 11 digits

- **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.

## 9.2 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 filed: name, enrolled date from - to. Yêu cầu chỉ cán bộ nhà trường được quyền gọi API này, ta có 1 số testcases như sau:

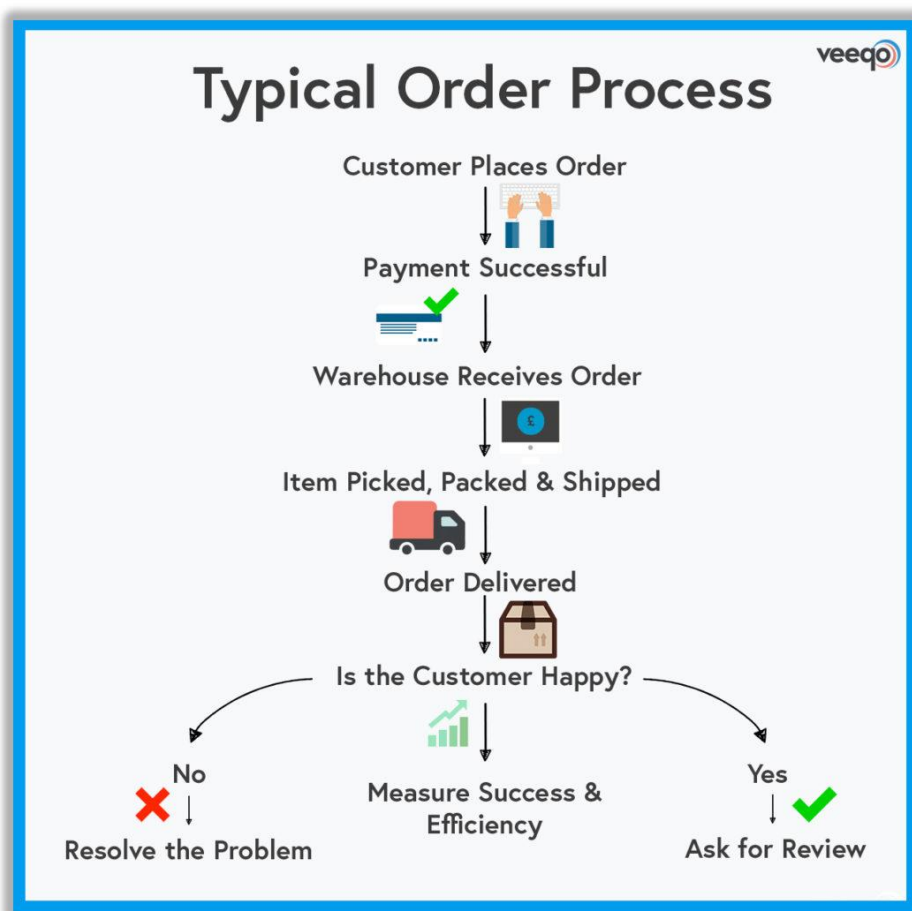
- Filter Student đã có trong hệ thống theo name / enrolled date from / enrolled date to
- Filter Student không có trong hệ thống theo name
- Filter Student khi chọn enrolled date from > enrolled date to (có thể Error code 400 hoặc Error code 200 và trả về rỗng, tùy cách xử lý của team)
- Filter Student 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.

## 9.3 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.



Mình chia Flow Testing này thành 2 loại:

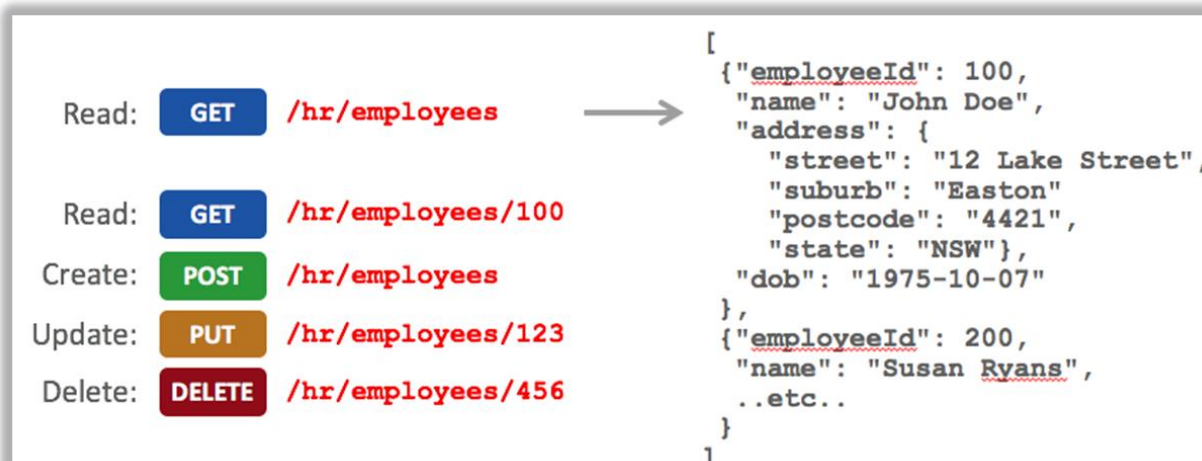
- Short flow: Chứa tất cả các flow của app, chia dựa theo chức năng.
- Long flow: Chứa 1 flow dài của App, chính là phần cốt lõi của App.

**Lưu ý:** ở đây mục tiêu là flow được hoàn thành và tất cả các API trả về kết quả đúng, nên sẽ không có các cases dạng negative để tạo ra exception. Valid input → right output.

## 10. Challenges

### 10.1 Types of API

Để quản lý resource, chỉ cần 4 loại API Get, Post, Put, Delete là đã đủ, nhưng trong thực tế thì sẽ có 5 APIs.



### GET

Thông thường sẽ có 2 APIs Get cho 1 resource:

- Get all: kèm theo nhiều parameters để filter (ko bắt buộc)
- Get one: lấy data của đúng 1 đối tượng của resource theo id

Goal: Retrieve data

Method: GET

URI: `${domain}/api/v1/${resource}/${id}`

Header

- Authorization

Parameters

<b>GET</b> /api/company	
<b>Parameters</b>	
Name	Description
name	Default value :
string	
(query)	
	<input type="text" value="name"/>

<b>GET</b> /api/company/{id}
------------------------------

## POST

Goal: Create a resource

Method: POST

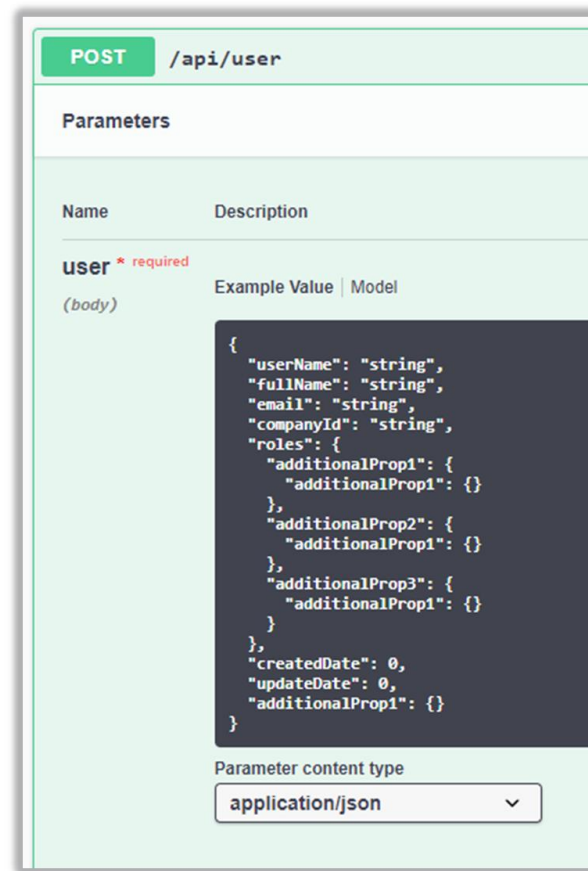
URI: `${domain}/api/v1/${resource}`

Header

- Content-type
- Authorization

Body

- Raw
- Form-data
- x-www-form-urlencoded



Swagger UI for the POST endpoint `/api/user`. The interface shows the method **POST** and the endpoint `/api/user`. Under the **Parameters** section, there is a table with columns **Name** and **Description**. A parameter named **user** is listed as **\* required** and its location is **(body)**. To the right of the parameter name are tabs for **Example Value** and **Model**. The **Example Value** tab is active, displaying a JSON object: 

```
{  "userName": "string",  "fullName": "string",  "email": "string",  "companyId": "string",  "roles": {    "additionalProp1": {      "additionalProp1": {}    },    "additionalProp2": {      "additionalProp1": {}    },    "additionalProp3": {      "additionalProp1": {}    }  },  "createdDate": 0,  "updateDate": 0,  "additionalProp1": {}}
```

 Below the JSON, there is a dropdown menu for **Parameter content type** with `application/json` selected.

## PUT

Goal: Update a resource

Method: POST

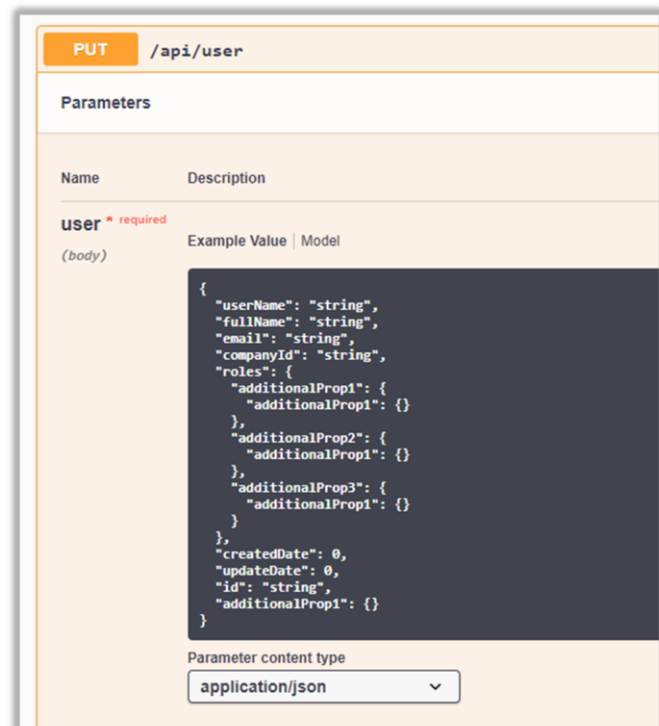
URI: `${domain}/api/v1/${resource}/${id}`

Header

- Content-type
- Authorization

Body

- Raw
- Form-data
- x-www-form-urlencoded



Swagger UI for the PUT endpoint `/api/user`. The interface shows the method **PUT** and the endpoint `/api/user`. Under the **Parameters** section, there is a table with columns **Name** and **Description**. A parameter named **user** is listed as **\* required** and its location is **(body)**. To the right of the parameter name are tabs for **Example Value** and **Model**. The **Example Value** tab is active, displaying a JSON object: 

```
{  "userName": "string",  "fullName": "string",  "email": "string",  "companyId": "string",  "roles": {    "additionalProp1": {      "additionalProp1": {}    },    "additionalProp2": {      "additionalProp1": {}    },    "additionalProp3": {      "additionalProp1": {}    }  },  "createdDate": 0,  "updateDate": 0,  "id": "string",  "additionalProp1": {}}
```

 Below the JSON, there is a dropdown menu for **Parameter content type** with `application/json` selected.

## **DELETE**

Goal: Delete a resource

Method: DELETE

URI: `${domain}/api/v1/${resource}/${id}`

Header

- Authorization

The screenshot shows a REST client interface for a DELETE request to the endpoint `/api/user`. The request method is set to DELETE. Under the 'Parameters' section, there is a table with two columns: 'Name' and 'Description'. A single parameter is listed: 'userId', which is marked as 'required' (indicated by a red asterisk) and is a 'string' type used as a 'query' parameter. The input field for 'userId' contains the text 'userId'.

### 10.2 Choose Error code to expect

1	1xx (Informational)	Response tạm thời để thông báo về trạng thái kết nối
2	<b>2xx (Successful)</b>	Server đã nhận được request, đã hiểu và chấp nhận yêu cầu từ client
3	3xx (Redirection)	Cần thêm những actions của client để có thể hoàn thành được request
4	<b>4xx (Client Error)</b>	Đã có lỗi của client khi gửi request
5	5xx (Server Error)	Server gặp vấn đề và không thể thực hiện được yêu cầu từ client

Trong mấy loại trên, 2xx và 4xx là những cái mà test API cần phải quan tâm, cần phải hiểu để có thể sử dụng đúng khi assert kết quả. Một số loại thường gặp:

200: Request thành công

201: Request thành công và 1 resource mới đã được tạo ra (thường dùng cho Post)

400: Server không thể “hiểu” được request vì lỗi cú pháp

401: Mặc dù là “unauthorized” nhưng phải hiểu là “unauthenticated” → Cần authenticate

403: Client không có quyền để access vào resource, Client ở đây đã được authenticated.

### 10.3 Prepare test data

Chuẩn bị test data là việc mà testers nào cũng phải làm, gần như là công việc hàng ngày. Trong API test, data có thể đến từ 3 nguồn:

- Tự tạo bằng script
- Đọc từ nguồn bên ngoài: csv, excel, json file
- Từ response của 1 API khác

Data muôn hình vạn trạng, nhưng có 1 số loại data ta thường xuyên gặp:

- Date time: có nhiều dạng format khác nhau và mốc thời gian khác nhau (hiện tại, quá khứ, tương lai)
- Username: cần tính unique, không dùng lại được
- Phone: có nhiều rules, ví dụ như 8-12 chữ số, bắt đầu bằng số 0...
- Email: có tính unique
- Random number
- ...

#### Data test from script

- Sử dụng những chức năng mà tools test API cấp (gọi là built-in functions), ví dụ:
  - Postman: dùng những function và library có ở phần [Postman Sandbox API](#)
  - Jmeter: dùng phần [Function Helpers](#)
- Sử dụng thư viện:
  - Java: [java-faker](#)
  - Javascript: [faker.js](#)
- Tự mình viết code

#### Data test from external resources

- Sử dụng built-in functions:
  - Postman: dùng phần Test Runner để làm việc với [data files](#)
  - Jmeter: [CSV Data Set Config](#)
- Sử dụng thư viện: [JUnit5](#), [Apache POI](#)

#### Data test from response of other APIs

1. Extract value
  2. Save value into a variable
  3. Get value from the variable
- Sử dụng built-in functions để extract value:
    - Postman: [lodash](#) library + [variables](#)
    - Jmeter: Json Extractor, Regex Extractor ...
  - Sử dụng thư viện: [JsonPath](#)

### 10.4 Assert Value

Kiểm tra kết quả là phần mà tool nào cũng support đầy đủ. Tất nhiên, tùy API mà có rất nhiều yêu cầu assertion khác nhau nhưng tựu chung lại cũng chỉ là: **extract value** → **compare**.



Với mình, có 2 loại assertion chính:

- **Assert one:** Ví dụ - kiểm tra id của object Student dưới đây có bằng 1 không.

```
{
  "id": 1,
  "firstName": "Vernon",
  "lastName": "Harper",
  "email": "egestas.rhonus.Proin@massaQuisqueporttitor.org",
  "programme": "Financial Analysis",
  "courses": [
    "Accounting",
    "Statistics"
  ]
},
```

- **Assert all:** Kiểm tra 1 tập hợp các phần tử có cùng kết quả. Ví dụ - API trả về list những Student có cùng programme, ta phải kiểm tra từng Student xem có đúng như vậy hay ko.

### Bài tập thực hành:

Example: <https://gist.github.com/giang-nguyen-niteco/b85c8bf9a5da543550cb12e6f8be2858>

Hãy kiểm tra:

- Cuốn sách đầu tiên có tác giả (author) là “Nigel Rees”
- Tất cả các cuốn sách đều có giá (price) trên 5.
- Tất cả các cuốn sách thuộc loại (category) “fiction” thì đều có giá trên 7

## Reference

1. An Introduction to APIs - Brian Cooksey <https://zapier.com/learn/apis/>
2. API testing best practices - Bas Dijkstra <https://www.ontestautomation.com/api-testingbest-practices/>
3. HTTP response code - Developer.mozilla.org <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>
4. OAuth2 - google-cloud medium <https://medium.com/google-cloud/understanding-oauth2-and-building-a-basic-authorization-server-of-your-own-a-beginners-guide-cf7451a16f66>