[Bài đọc] Nguyên tắc thiết kế API

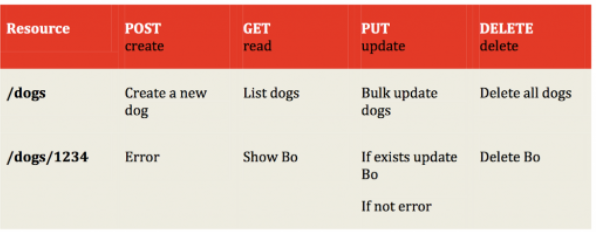
Những yếu tố quyết định chất lượng API tốt

* Self-documenting: nhìn vào đường dẫn của API là chúng ta có thể đoán dược mục đích sử dụng của nó.
* Flexible: Tính mở rộng của API.
* Unified structure and attribute names: Thống nhất về mặt cấu trúc cho resource cũng như cách đặt tên các thuộc tính.
* Clear error message: Khi hệ thống xảy ra lỗi thì phải có hiển thị thông điệp rõ ràng để tiện cho việc fix bug.

Một số nguyên tắc thiết kế API

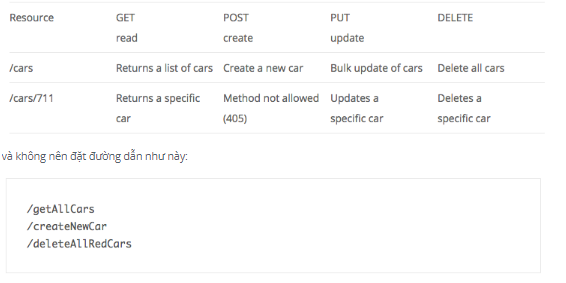
1.Sử dụng HTTP method để mô tả về chức năng của resource:

Chúng ta có 4 HTTP method cơ bản bao gồm POST, GET, PUT, DELETE. Với mỗi method sẽ ứng với một chức năng tương ứng của API là **CRUD**(tạo, đọc, sửa và xoá).



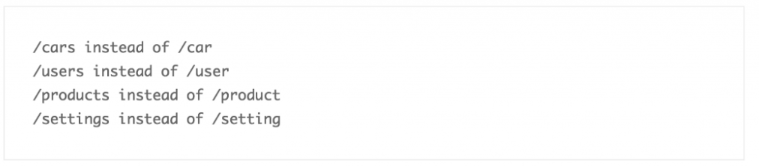
### 2. Sử dụng danh từ số nhiều và không sử dụng động từ

Chúng ta nên đặt tên đường dẫn như này:



### 3.Chỉ sử dụng danh từ số nhiều

Mọi người hay có thói quen sử dụng cả danh từ số ít và danh từ số nhiều. Và lời khuyên của mình cho mọi người là hãy  chỉ sử dụng danh từ số nhiều cho tất cả các resource.



4. Liên kết trong resource

Thông thường chúng ta sẽ có rất nhiều resource có quan hệ đến nhau và việc thiết kế liên kết cho những resource đó là việc hết sức đau đầu đối với các lập trình viên.

Giả sử chúng ta có 2 resource là cars và users. Bây giờ chúng ta cần lấy tất cả các xe của 1 user cụ thể, ta sẽ có API sau:

* GET /users/123/cars

Bây giời ta cần xem chi tiết thông tin của một xe cụ thể của user 123, ta sẽ có 2 cách như sau:

* GET /users/123/cars/5 (lấy thông tin của car 5 của user 123)
* GET /cars/5 (lấy thông tin của car 5)

Tuỳ theo nghiệp vụ mà 2 resource sẽ có response khác hoặc giống nhau.

Một lưu ý nữa trong vấn đề liên kết resource là là một resource chỉ được ***liên kết tối đa 2 đối tương*** (object), việc liên kết quá nhiều đối tượng sẽ làm cho resource trở nên rối mặt.

* GET /users/1/posts/5/comments/10

Như ví dụ ở trên, chúng ta sẽ lấy comment 10 thuộc post 5 của user 1 API này sẽ trở nên phức tạp cho chúng ta lúc sử dụng.

**Lời khuyên**: Với các api truy vấn dữ liệu dạng filter với nhiều param thì có thể dùng cú pháp như sau:

/users/1/filter?properties.post=5&properties.comment=10

Xây dựng sẵn bộ điều kiện truy vấn đơn giản gồm các thành phần như sau:

* neq: không bằng
* gt: lớn hơn
* gte: lớn hơn bằng
* lt: nhỏ hơn
* lte: nhỏ hơn bằng
* in: có trong
* not\_in: không trong

5. Versioning

Versioning là một điều bắt buộc với tất cả resource, việc đánh version cho resource tuân thủ 2 nguyên tắc sau:

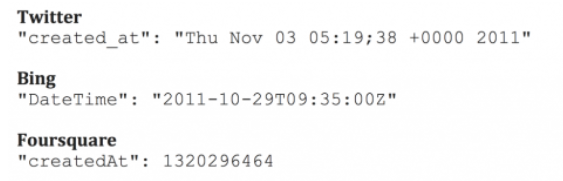
* Bắt đầu bằng “v” và kết thúc bằng một số nguyên dương , tránh dùng số thập phân (dùng v1 thay vì v1.5)
* Versioning sẽ được đặt ở vị trí đầu tiên của resource

Ví dụ:

* GET /v1/users/1 thay vì GET /users/v1.5/1

6. Đặt tên cho các attribute

Chúng ta sẽ xem sét 3 ví dụ sau:



Ta có thể nhìn thấy 3 dạng khác nhau. Ở đây chúng ta sẽ không nói đến cách nào đúng cách nào sai mà mỗi team sẽ tự chọn cho mình một cách và tuân thủ cách viết đó cho toàn project.

Ngoài ra, chúng ta cũng nên chú ý đến việc thống nhất một danh từ cho một attribute cụ thể. Có rất nhiều bạn mặc lỗi là dùng nhiều danh từ khác nhau để nói về cùng 1 loại attribute (“user\_id” và “user” cùng được sử dụng khi muốn lấy “user\_id” hay “from\_date” và “from” cùng được sử dụng khi muốn lấy “from\_date”)

Tóm lại tuân thủ 2 nguyên tắc sau:

* Thống nhất 1 loại convention.
* Thống nhất cách đặt tên cho cùng 1 loại attribute.

7. Phân trang

Xem sét cách lấy 25 phần tử từ vị trí thứ 50 của 3 hệ thống sau:

* Facebook: offset 50 and limit 25
* Twitter: page 3 and rpp 25 (records per page)
* LinkedIn: start 50 and count 25

Trong trường hợp này, mình thiên về cách sử dung của facebook vì nó trực quan và dễ hiểu hơn. Ngoài ra, những bạn nào đã code database thì khái niệm “limit” và “offset” cũng không phải là một khái niệm gì mới.

8. Tìm kiếm

Quy tắc: attribute tên là “q”(query)

Global search:

* GET /search?q=fluffy+fur

Scope search:

* GET /users/123/cars?q=fluffy+fur

9. Lựa chọn field trả về

Trong một số trường hợp, client sẽ không cần đầy đủ thông tin của một object. Ví dụ như phiên bản mobile sẽ không lấy hết thông tin của user như phiên bản web. Chính vì vậy chúng ta cần resource có khả năng tuỳ chỉnh những field trả về. Việc này làm tăng performance cho phía client.

* GET /users?fields=id,name,address

10. Định dạng dữ liệu trả về

Đối với những resource hỗ trợ nhiều định dạng dữ liệu trả về, HTTP-Header sẽ là nơi để xác định dịnh dạng đó.

* Content-Type: Khai báo request format
* Accept: Khai báo response format

11. HTTP status code và error message

Chuẩn HTTP cung cấp cho ta rất nhiều status code. Chúng ta sẽ không cần biết hết tất cả nhưng ít nhất nên biết đến những status code:

* **200 OK**— This is most commonly used HTTP code to show that the operation performed is successful.
* **201 CREATED** — This can be used when you use POST method to create a new resource.
* **202 ACCEPTED** — This can be used to acknowledge the request sent to the server.
* **400 BAD REQUEST**— This can be used when client side input validation fails.
* **401 UNAUTHORIZED / 403 FORBIDDEN** — This can be used if the user or the system is not authorised to perform certain operation.
* **404 NOT FOUND** — This can be used if you are looking for certain resource and it is not available in the system.
* **500 INTERNAL SERVER ERROR**— This should never be thrown explicitly but might occur if the system fails.
* **502 BAD GATEWAY**— This can be used if server received an invalid response from the upstream server.

Đối với error message trả về khi có lỗi xảy ra cần đảm báo dễ hiểu cho cả user, client developer và backend developer. Về cơ bản chúng ta sẽ có 3 message trả về phục vụ cho 3 đối tượng khác nhau:

* user message (message trả về cho user)
* internal message (message trả vể cho backend developer)
* code (message trả về cho client developer)

Một message đầy đủ yêu cầu sẽ có dạng như sau:

{

    "error":{

        "userMessage": "Sorry, the requested resource does not exist",

        "internalMessage": "No car found in the database",

        "code":34

    }

}

Last modified: Tuesday, 26 January 2021, 6:31 PM