Báo cáo bài 6

1. **API:** là viết tắt của Application Programming Interface (Giao diện lập trình ứng dụng) là một tập hợp các định nghĩa, giao thức, và công cụ chương trình con để xây dựng để xây dựng phần mềm và ứng dụng. Hiểu đơn giản API là một trung gian phần mềm cho phép hai ứng dụng giao tiếp với nhau. API có thể trả về dữ liệu mà bạn cần cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như JSON hay XML. Với API, bạn có thể tiếp cận, truy xuất dữ liệu từ máy chủ thể hiện chúng trên ứng dụng phần mềm hoặc website của mình một cách dễ dàng hơn. Tính tới nay, API đã phát triển với nhiều loại ứng dụng và phần mềm khác nhau. Thế hệ mới nhất của web/ app API có thể ứng dụng được ở mọi hệ thống từ cơ sở dữ liệu, hệ điều hành, hệ thống nền web, thư viện hay thậm chí là phần cứng máy tính
2. **REST:** (**RE**presentational **S**tate **T**ransfer)  là một dạng chuyển đổi cấu trúc dữ liệu được đề xuất bởi nhà khoa học máy tính Roy Thomas Fielding vào năm 2012.

REST là một kiểu kiến trúc được sử dụng trong việc giao tiếp giữa các máy tính (máy tính cá nhân và máy chủ của trang web) trong việc quản lý các tài nguyên trên internet.

A diagram of a api

Description automatically generated

REST được sử dụng rất nhiều trong việc phát triển các ứng dụng Web Services sử dụng giao thức HTTP trong giao tiếp thông qua mạng internet. Các ứng dụng sử dụng kiến trúc REST này thì sẽ được gọi là ứng dụng phát triển theo kiểu RESTful. Một RESTful API đơn giản là một API tuân thủ các nguyên tắc và ràng buộc của REST.

REST bao gồm 3 bộ phận cơ bản như: bộ máy chủ ngoài chứa dữ liệu (external server), máy chủ API và máy chủ khách (client). Trong đó, máy khách có thể là bất cứ thứ gì, ứng dụng trên nền web, thư viện hoặc thậm chí là các phần mềm khác nhau của một phần mềm máy tính.

REST cho phép các máy khách truy cập máy chủ API và thực hiện các lệnh lấy về, chỉnh sửa hay xóa dữ liệu từ external server. Các lập trình viên có thể thoải mái truy xuất, chỉnh sửa dữ liệu từ máy chủ mà không cần biết hệ thống hoạt động như thế nào.

**Cách hoạt động của REST**

**Khi làm việc với server sẽ gồm 4 hoạt động thiết yếu là:**

* Lấy dữ liệu ở một định dạng nào đó (JSON)
* Tạo mới dữ liệu
* Cập nhật dữ liệu
* Xóa dữ liệu

REST hoạt động chủ yếu dựa vào giao thức HTTP. Mỗi trong 4 hoạt động cơ bản trên sẽ sử dụng những phương thức HTTP riêng (HTTP method):

* POST (CREATE) : Tạo mới một Resource.
* GET (READ) : Trả về một Resource hoặc một danh sách Resource.
* PUT (UPDATE) : Cập nhật, thay thế thông tin cho Resource.
* DELETE (DELETE) : Xoá một Resource.

REST là một kiến trúc thống nhất giúp thiết kế các website để có thể dễ dàng quản lý các tài nguyên. Nó không phải là một quy luật buộc bạn phải tuân theo mà đơn giản là một kiến trúc được đề xuất ra và kiến trúc này hiện đang được sử dụng rất phổ biến vì tính đơn giản, dễ hiểu và rất ưu việt của nó. Với các ứng dụng web được thiết kế sử dụng RESTful, ban có thể dễ dàng biết được URL và HTTP method để quản lý một resource

**Các ràng buộc trong REST**

1. **Kiến trúc client-server:** Server sẽ là tập hợp các services nhỏ xử lý các request từ client, việc tách biệt giúp tăng tính linh hoạt của client cũng như khả năng mở rộng của server mà vẫn đảm bảo được giao tiếp.
2. **Stateless (phi trạng thái):** Server và client sẽ không lưu trạng thái của nhau, mỗi request và response sẽ chứa đầy đủ thông tin, tách biệt với nhau, điều này giúp hệ thống dễ phát triển và mở rộng.
3. **Cacheability (có khả năng cache):** Các response có thể được lấy ra từ cache của server giúp giảm thiểu thời gian xử lý, các request phải đảm bảo tính duy nhất để response không bị nhầm lẫn với các request khác.
4. **Layered system (phân lớp hệ thống):** Hệ thống được chia làm nhiều lớp, việc giao tiếp của 1 lớp chỉ được tiến hành với lớp ở trên và lớp ở dưới của nó, khả năng cân bằng tải (load balancing) và cache dữ liệu trong hệ thống cũng sẽ được cải thiện.
5. **Uniform interface (chuẩn hoá các interface):** Đây là ràng buộc quan trọng nhất của hệ thống REST. Để đảm bảo ràng buộc này, hệ thống tập trung vào việc xử lý các tài nguyên (resource). Mỗi một resource sẽ được xác định (định danh) bằng một URI (Uniform Resource Identifier) riêng biệt.

**Ưu điểm của REST**

* REST cũng có ưu điểm khi sử dụng giao thức stateless (không trạng thái). Hệ thống này không sử dụng session, cookie, không cần biết những thông tin đó trong mỗi lần request đến máy chủ ngoài. Điều này giúp REST giảm tải cho máy chủ ngoài, nâng cao hiệu suất làm việc.
* Tính khả biến: với các hệ thống cần thay đổi các tài nguyên liên tục, sử dụng REST với việc tạo request đơn giản sẽ giúp mọi chuyện trở nên đơn giản hơn.
* Tính mở rộng: các hệ thống REST có khả năng mở rộng rất cao nhờ sự tách biệt giữa các thành phần và các quy ước giao tiếp được quy định sẵn.
* Tính linh hoạt: việc chuẩn hoá interface giúp hệ thống trở nên linh hoạt hơn, có thể sử dụng cho cho nhiều nền tảng khác nhau, mobile, web,...
* Trong sáng: trong giao tiếp giữa các thành phần, các request trở nên rất rõ ràng, dễ hiểu.

**Nhược điểm của REST**

* REST Chỉ hoạt động trên các giao thức HTTP.
* Ưu điểm của việc sử dụng các cấu trúc HTTP lại cũng chính là một hạn chế. Bởi các hạn chế của HTTP cũng trở thành các hạn chế của kiến trúc REST. Ví dụ: HTTP không lưu trữ thông tin về trạng thái giữa các chu kỳ khi phản hồi yêu cầu, và như vậy các ứng dụng dựa trên REST rơi vào tình trạng không rõ trạng thái và mọi tác vụ quản lý trạng thái phải được máy khách thực hiện.

**Thiết kế RESTful API**

1. GET (SELECT): Trả về một Resource hoặc Những hành động CRUD sử dụng những phương thức HTTP

* một danh sách Resource.
* POST (CREATE): Tạo mới một Resource.
* PUT (UPDATE): Cập nhật thông tin cho Resource.
* PATCH (UPDATE): Cập nhật một thành phần, thuộc tính của Resouce.
* DELETE (DELETE): Xoá một Resource.
* HEAD – Trả về thông tin chung của một hoặc danh sách Resource.
* OPTIONS – Trả về thông tin mà người dùng được phép với Resource.

1. Sử dụng danh từ số nhiều, không dùng động từ

Danh từ số nhiều: Mặc dù là sẽ có những ngữ cảnh mô tả một trường hợp trong những Resource nhưng trong thực tế để giữ cho định dạng URL nhất quán thì sẽ nên luôn luôn sử dụng số nhiều. Không phải đối phó với những danh từ (person/people, goose/geese) sẽ làm những người sử dụng API thấy dễ chịu hơn và cũng sẽ cho bên cung cấp API dễ thực thi nó (/tickets và tickets/12 thường được viết trong một controller chung)

GET /tickets – Trả về danh sách những ticket

GET /tickets/12 – Trả về một ticket được định danh

POST /tickets – Tạo mới một ticket

PUT /tickets/12 – Cập nhật thông tin cho ticket #12

PATCH /tickets/12 – Cập nhật một thuộc tính cho ticket #12

DELETE /tickets/12 – Xóa ticket #12

**Nếu tồn tại một quan hệ duy nhất với Resource khác, RESTful cung cấp những hướng dẫn có ích**

Ví dụ:

**GET /tickets/12/messages** – Trả về danh sách message của ticket #12

**GET /tickets/12/messages/5** – Trả về message #5 của ticket #12

**POST /tickets/12/messages** – Tạo mới một message trong ticket #12

**PUT /tickets/12/messages/5** – Cập nhật message #5 của ticket #12

**PATCH /tickets/12/messages/5** – Cập nhật một số thuộc tính của message #5 cho ticket #12

**DELETE /tickets/12/messages/5** – Xoá message #5 for ticket #12

**Không sử dụng động từ**

GET /getAllCars

POST /createNewCar

DELETE /deleteAllRedCars

1. **Version**

Hãy luôn sử dụng version trong API của bạn. Version sẽ giúp bạn lặp lại nhanh hơn và ngăn ngừa được những request không hợp lệ. Nó cũng sẽ giúp bạn suôn sẻ dễ dàng khi chuyên đổi những version API hay như là bạn có thể tiếp tục cung cấp các những version cũ trong một khoảng thời gian.

https://example.org/api/v1/\*

https://api.example.com/v1/\*

Nếu ứng dụng của bạn lớn, sự lựa chọn tốt nhất nên đặt API ở subdomain (api). Nó có thể giúp làm giảm được sự tải trang.

1. **Lọc kết quả, sắp xếp, tìm kiếm, phân trang**

**Lọc kết quả:** Sử dụng một biến duy nhất cho mỗi trường được lọc.

* GET /tickets?state=open – Lấy duy nhất trạng thái đang open từ danh sách ticket
* GET /cars?seats<=2 Trả về một danh sách những oto với số ghế tối đa là 2

**Sắp xếp:** Sử dụng tham biến "sort" để mô tả luật sắp xếp.

* GET /tickets?sort=-priority – Trả về một danh sách những ticket được sắp xếp theo priority giảm dần.
* GET /tickets?sort=-priority,created\_at – Trả về một danh sách những ticket được sắp xếp theo priority giảm dần. Với một ticket cùng priority, những ticket cũ sẽ được sắp xếp trước.

**Tìm kiếm:** Đôi khi những điều kiện lọc cơ bản không đủ và bạn cần đến full text search. Sử dụng tham biến “q”, một tham biến được dùng trong [ElasticSearch](https://www.elastic.co/) hoặc [Lucene](http://lucene.apache.org/).

* GET /tickets?q=keyword&state=open&sort=-priority,created\_at

**Giới hạn dữ liệu được trả về**: Sử dụng tham biến "fields"

* GET /tickets?fields=id,subject,customer\_name,updated\_at&state=open&sort=-updated\_at

**Phân trang:**

Sử dụng page và per\_page

* GET /ticket?page=1&per\_page=100

Hoặc có thể sử dụng mặc định limit và offset:

* GET /ticket?offset=10&limit=100

1. **Snake\_case hay camelCase**

Nhiều nghiên cứu chỉ ra rằng snake\_case dễ đọc hơn camelCase khoảng 20% và rất nhiều những API phổ biến đều sử dụng snake\_case.

1. **Pretty print và gzip cần được hỗ trợ**

Một API mà những khoảng trắng, ký tự space đã được nén lại thì thật không vui khi mà phải quan sát từ Browser. Và khi sử dụng Gzip có thể giúp tiết kiệm đến 60% băng thông.

1. **Authentication**

RESTful API không sử dụng session và cookie, bởi vậy nên sử dụng một mã bí mật access\_toket với mỗi request. Luôn sử dụng SSL <https://example.com/users?access_token=xxxxxxxx>

1. **Caching**

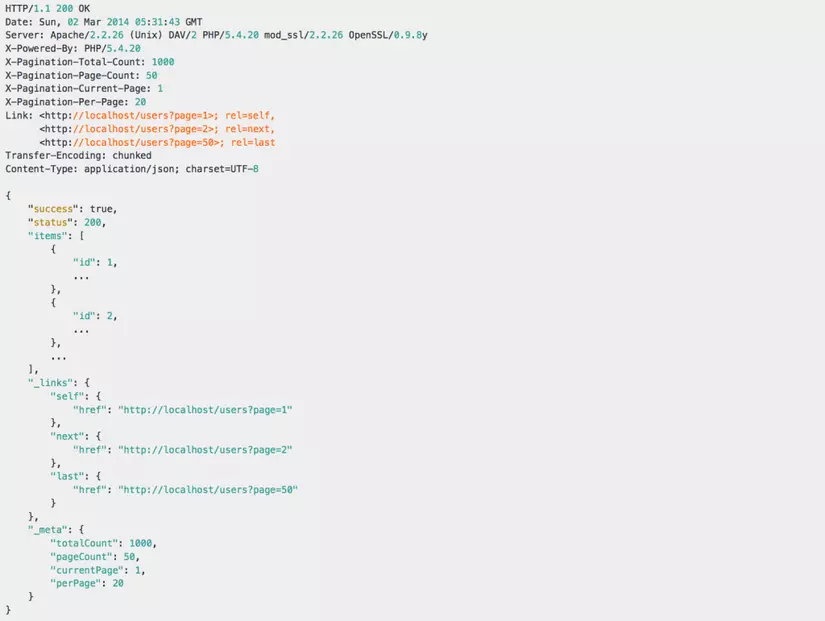
Có 2 phương pháp nên được áp dụng là [Etag](https://en.wikipedia.org/wiki/HTTP_ETag) và [Last-Moified](https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.29).

9. Dữ liệu trả về

Rất nhiều API được bao bọc như:

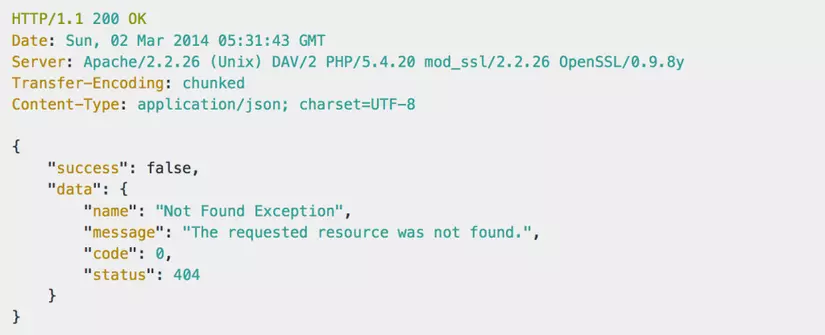


Dữ liệu trả về nhiều khi bao bọc là không thật sự cần thiết theo CORS và Link header from RFC 5988. Một vài REST client không cho phép truy cập tới HTTP header và cũng như JSONP không có quyền truy cập vậy, do vậy có thể bao bọc dữ liệu trả về trong những trường hợp cần thiết.



1. **Lỗi**

Có những lúc bạn muốn tuỳ biến định dạng lỗi mặc định được trả về, ví dụ như sử dụng những trạng thái, mã HTTP khác để chỉ ra những lỗi khác nữa, một lời khuyên bạn nên luôn sử dụng mã 200 và những mã lỗi thực sẽ là một thành phần của cấu trúc JSON được trả về giống như là sau:



1. **Những mã HTTP thường được sử dụng**

* 200 OK – Trả về thành công cho những phương thức GET, PUT, PATCH hoặc DELETE.
* 201 Created – Trả về khi một Resouce vừa được tạo thành công.
* 204 No Content – Trả về khi Resource xoá thành công.
* 304 Not Modified – Client có thể sử dụng dữ liệu cache.
* 400 Bad Request – Request không hợp lệ
* 401 Unauthorized – Request cần có sự authentication.
* 403 Forbidden – Server hiểu request nhưng bị từ chối không cho phép.
* 404 Not Found – Không tìm thấy rource từ URI
* 405 Method Not Allowed – Phương thức không cho phép với user hiện tại.
* 410 Gone – Resource không còn tồn tại, Version cũ đã không còn hỗ trợ.
* 415 Unsupported Media Type
* 422 Unprocessable Entity – Dữ liệu không được kiểm chứng
* 429 Too Many Requests – Request bị từ chối do bị giới hạn