**Báo cáo thu hoạch về phần tìm hiểu Rest API**

1. **Giới thiệu về REST API**

**REST API là gì?**

**REST**, hay REpresentational State Transfer, là một kiểu kiến ​​trúc để cung cấp các tiêu chuẩn giữa các hệ thống máy tính trên web, giúp các hệ thống giao tiếp với nhau dễ dàng hơn. Các hệ thống tuân thủ REST, thường được gọi là hệ thống RESTful, được đặc trưng bởi cách chúng không trạng thái và tách biệt các mối quan tâm của máy khách và máy chủ. Chúng ta sẽ đi sâu vào ý nghĩa của những thuật ngữ này và tại sao chúng là những đặc điểm có lợi cho các dịch vụ trên Web.

**API** (Application Programming Interface) là giao diện lập trình ứng dụng giúp tạo ra các phương thức kết nối với các thư viện và ứng dụng khác nhau. REST API là một ứng dụng chuyển đổi cấu trúc dữ liệu có các phương thức để kết nối với các thư viện và ứng dụng khác. REST API không được xem là một công nghệ, nó là một giải pháp để tạo ra các ứng dụng web services thay thế cho các kiểu khác như SOAP, WSDL (Web Service Definition Language), …

## ****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…

## ****Ưu và nhược điểm của API****

### **Ưu điểm**

1. Giao tiếp hai chiều phải được xác nhận trong các giao dịch sử dụng API. Cũng chính vì vậy mà các thông tin rất đáng tin cậy.
2. API là công cụ mã nguồn mở, có thể kết nối mọi lúc nhờ vào Internet.
3. Hỗ trợ chức năng RESTful một cách đầy đủ.
4. Cấu hình đơn giản khi được so sánh với WCF (Window Communication Foundation). Cung cấp cấp trải nghiệm thân thiện với người dùng.

### **Nhược điểm**

* Tốn nhiều chi phí phát triển, vận hành, chỉnh sửa.
* Đòi hỏi kiến thức chuyên sâu.
* Có thể gặp vấn đề bảo mật khi bị tấn công hệ thống.

**Ứng dụng của API**

* **Web API:** Là hệ thống API được sử dụng trong các hệ thống website, chẳng hạn: Google, Facebook… Hầu hết các website đều cung cấp hệ thống API cho phép bạn kết nối, lấy dữ liệu hoặc cập nhật cơ sở dữ liệu. Đa số Web API được thiết kế theo tiêu chuẩn [RESTful](https://thietbitudong.com.vn/restful-api-la-gi/).
* **API trên hệ điều hành:** Windows hay Linux có rất nhiều API. Họ cung cấp các tài liệu API là đặc tả các hàm, phương thức cũng như các giao thức kết nối. Nó giúp lập trình viên có thể tạo ra các phần mềm ứng dụng có thể tương tác trực tiếp với hệ điều hành.
* **API của thư viện phần mềm (framework):** API mô tả và quy định các hành động mong muốn mà các thư viện cung cấp. Một API có thể có nhiều cách triển khai khác nhau, giúp cho một chương trình viết bằng ngôn ngữ này có thể sử dụng được thư viện viết bằng ngôn ngữ khác.

**RESTful API** là một tiêu chuẩn dùng trong việc thiết kế các API cho các ứng dụng web để quản lý các resource. RESTful là một trong những kiểu thiết kế API được sử dụng phổ biến ngày nay để cho các ứng dụng (web, mobile…) khác nhau giao tiếp với nhau. Chức năng quan trọng nhất của REST là quy định cách sử dụng các HTTP method (như GET, POST, PUT, DELETE…) và cách định dạng các URL cho ứng dụng web để quản các resource. RESTful không quy định logic code ứng dụng và không giới hạn bởi ngôn ngữ lập trình ứng dụng, bất kỳ ngôn ngữ hoặc framework nào cũng có thể sử dụng để thiết kế một

**RESTful hoạt động như thế nào?**



**Ràng buộc của REST**

**Client - Server**: Hoạt động theo mô hình Client - Server, việc tách biệt này nhằm đơn giản hóa việc thực hiện các thành phần (kiểu chia để trị), giảm sự phức tạp của ngữ nghĩa kết nối, nâng cao hiệu quả của việc điều chỉnh hiệu năng, tăng khả năng mở rộng của máy chủ.

**Stateless**: Server và Client không lưu trạng thái của nhau. Với mỗi một request được gửi đi đều phải được đóng gói đầy đủ thông tin để server có thể nhận và hiểu được. Điều này giúp hệ thống dễ phát triển, bảo trì, mở rộng vì không tốn công CRUD trạng thái của Client. Tuy nhiên có mặt hạn chế là làm tăng lưu lượng thông tin cần truyền tải giữa client và server.

**Cacheable**: Mọi thứ trên web đều có thể được lưu trữ, do đó response phải xác định rõ ràng liệu chúng có thể lưu trữ được hay không, tránh lưu trữ bộ nhớ đệm không phù hợp hoặc lưu trữ các thông tin cũ, không cần thiết.

**Layered system:** Giảm mức độ phức tạp của hệ thống, giúp các thành phần tách biệt nhau từ đó dễ dàng mở rộng. Với mỗi một lớp chỉ trao đổi trực tiếp với lớp ngay dưới và trên nó.

**Code on demande**: Đây là "optional" duy nhất trong ràng buộc của REST. Máy chủ có thể tạm thời mở rộng hoặc tùy chỉnh các chức năng của một máy khách bằng cách chuyển mã thực thi, như javascript.

**REST bao gồm những gì?**

Nếu là một lập trình viên không thể không biết đến CRUD, và nó cũng chính là 4 method được sử dụng bở REST, được thể hiện qua các method như sau:

* POST (Create): Thêm dữ liệu vào database
* GET (Read): Xem dữ liệu từ database
* PUT (Update): Cập nhật dữ liệu vào database
* DELETE (Delete): Xóa dữ liệu khỏi database
* PATCH (Thực chất là một phần của PUT)

**Tạo sao nên dùng REST?**

**Các ưu điểm của REST**

* Giúp cho ứng dụng trở nên rõ ràng hơn.
* REST URL đại diện cho resource chứ không phải là hành động.
* Dữ liệu được trả về với nhiều định dạng khác nhau như: xml, html, rss, json …
* Code đơn giản và ngắn gọn.
* REST chú trọng vào tài nguyên hệ thống

**Nhược điểm**:

* Bảo mật kém hơn so với SOAP
* Do không lưu trạng thái giữa Client - Server nên khối lượng thông tin truyền tải lớn -> tốn tài nguyên
* Chỉ hoạt động trên giao thức HTTP

**Các URI liên quan tới CRUD (Create, Read, Update, Delete) bao gồm:**

|  |  |
| --- | --- |
| **Action** | **URI + Data (XML hoặc JSON)** |
| Create (@POST) | **/contextPath/servletPath/employees**  { "empNo":"E05", "empName":"Martin", "position":"Salesman" }  <employee>   <empNo>E05</empNo>   <empName>Martin</empName>   <position>Salesman</position> </employee> |
| Read (@GET) | **/contextPath/servletPath/employees**  **/contextPath/servletPath/employees/{empNo}** |
| Update (@PUT) | **/contextPath/servletPath/employees**  { "empNo":"E01", "empName":"Smith 2", "position":"Cleck" }  <employee>   <empNo>E01</empNo>   <empName>Smith 2</empName>   <position>Clerk</position> </employee> |
| Delete (@DELETE) | **/contextPath/servletPath/employees/{empNo}** |

**Nguyên tắc thiết kế API**

**Nguyên tắc thứ nhất:** Sử dụng HTTP Method để mô tả chức năng của resource

Chúng ta thường sử dụng 4 chức năng như: GET, POST, PUT, DELETE tương đương với các chức năng là đọc, tạo, sửa, xóa.

**Nguyên tắc thứ hai**: Sử dụng danh từ số nhiều, không sử dụng động từ

Ví dụ: GET

/homes: sử dụng ĐÚNG

/getHomes: sử dụng SAI

**Nguyên tắc thứ ba:** Chỉ sử dụng danh từ số nhiều

Chúng ta sẽ dùng các REST API như /cars thay vì cho /car

**Nguyên tắc thứ tư:** Liên kết trong resource

Trong các resource có rất nhiều các quan hệ, vì thế thiết kế REST API sẽ rất đau đầu.

Chúng ta cần phải chia các quan hệ theo các cấp độ của quan hệ.

Ví dụ: chúng ta cần lấy thông tin của comment trong 1 bài đăng của 1 user.

Như vậy, chúng ta thấy user là đối tượng lớn nhất, sau đó đến bài đăng và cuối cùng là comment.

Ta sẽ có 1 API như sau: GET/users/1/posts/3/comments/10

**Nguyên tắc thứ năm:** Tìm kiếm

Sử dụng attribute là “q” (query)

Ví dụ: GET/cars?q=mercedes

Tìm tất cả ô tô có tên là mercedes

**Nguyên tắc thứ sáu:** Lựa chọn trường trả về

Sử dụng attribute là “field”

Ví dụ: GET/user?field=id, name, address

Lấy danh sách tất cả các user với các thông tin bao gồm id, name, address

**Một số http status code**

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** — Đây là mã HTTP được sử dụng phổ biến nhất để cho thấy rằng hoạt động được thực hiện là thành công.**201 CREATED** — This can be used when you use POST method to create a new resource.
* **202 ACCEPTED** — Điều này có thể được sử dụng để xác nhận yêu cầu được gửi đến máy chủ.
* **400 BAD REQUEST** — Điều này có thể được sử dụng khi xác nhận đầu vào phía khách hàng không thành công.
* **401 UNAUTHORIZED / 403 FORBIDDEN** — Điều này có thể được sử dụng nếu người dùng hoặc hệ thống không được phép thực hiện một số thao tác nhất định.
* **404 NOT FOUND** — Điều này có thể được sử dụng nếu bạn đang tìm kiếm một số tài nguyên nhất định và nó không có sẵn trong hệ thống.
* **500 INTERNAL SERVER ERROR** — Điều này không bao giờ nên được ném một cách rõ ràng nhưng có thể xảy ra nếu hệ thống bị lỗi.
* **502 BAD GATEWAY** — Điều này có thể được sử dụng nếu máy chủ nhận được phản hồi không hợp lệ từ máy chủ ngược dòng.