**Họ và tên:** Nguyễn Quốc Hùng

**BÁO CÁO EX6 – REST API**

1. Các khái niệm
2. **API:**

* **API là gì?**

API (Application and Programming Interface) hay còn gọi là Giao diện lập trình ứng dụng, là cơ chế cho phép 2 thành phần phần mềm giao tiếp với nhau bằng một tập hợp các định nghĩa và giao thức.

Có rất nhiều loại API khác nhau, ví dụ: API Facebook, API Twitter, v.v. Tuy nhiên, khi mọi người nói về các API này, hầu hết họ đều đề cập đến API REST.

* **Cơ chế hoạt động của API:**

Kiến trúc API thường được giải thích dưới dạng máy chủ và máy khách. Ứng dụng gửi yêu cầu được gọi là máy khách, còn ứng dụng gửi phản hồi được gọi là máy chủ.

API hoạt động theo 4 cách khác nhau, tùy vào thời điểm và lý do chúng được tạo ra:

* **API SOAP**

Các API này sử dụng Giao thức truy cập đối tượng đơn giản. Máy chủ và máy khách trao đổi thông đệp bằng XML. Đây là loại API kém linh hoạt được dùng phổ biến trước đây.

* **API RPC**

Những API này được gọi là Lệnh gọi thủ tục từ xa. Máy khách hoàn thành một hàm (hoặc thủ tục) trên máy chủ còn máy chủ gửi kết quả về cho máy khách.

* **API Websocket**

API Websocket là một bản phát triển API web hiện đại khác sử dụng các đối tượng JSON để chuyển dữ liệu. API WebSocket hỗ trợ hoạt động giao tiếp hai chiều giữa ứng dụng máy khách và máy chủ. Máy chủ có thể gửi thông điệp gọi lại cho các máy khách được kết nối, điều này khiến loại API này hiệu quả hơn API REST.

* **API REST**

Đây là loại API phổ biến và linh hoạt nhất trên web hiện nay. Máy khách gửi yêu cầu đến máy chủ dưới dạng dữ liệu. Máy chủ dùng dữ liệu đầu vào từ máy khách này để bắt đầu các hàm nội bộ và trả lại dữ liệu đầu ra cho máy khách.

1. **REST:**

* **REST là gì?**

REST (Representation State Transfer) hay còn được gọi là Chuyển trạng thái đại diện, xác định một tập hợp các hàm như GET, PUT, DELETE, v.v. mà máy khách có thể dùng để truy cập vào dữ liệu của máy chủ. Máy khách và máy chủ trao đổi dữ liệu qua giao thức HTTP.

* **Tính năng chính của REST:**

Tính năng chính của API REST là tính không trạng thái. Tính không trạng trái nghĩa là máy chủ không lưu dữ liệu của máy khách giữa các yêu cầu. Các yêu cầu mà máy khách gửi cho máy chủ tương tự như URL mà bạn nhập vào trình duyệt để truy cập vào trang web. Phản hồi từ máy chủ là dữ liệu thuần chứ không được kết xuất thành đồ họa như thường thấy trên trang web.

* **Một số thuật ngữ quan trọng của REST:**
* **API** - Giao diện lập trình ứng dụng.
* **REST** - Chuyển trạng thái đại diện.
* **Client** – Thực thể đang sử dụng API, bằng cách thực hiện các lệnh gọi HTTP(s) đối với “điểm cuối” API.
* **Server** – Thực thể phục vụ API, bằng cách cung cấp phản hồi cho lệnh gọi HTTP(s) nhận được trên “điểm cuối”
* **HTTP** – Giao thức truyền siêu văn bản là một giao thức web xác định cách định dạng và truyền tải tin nhắn. Hơn nữa, HTTP mô tả các hành động mà máy chủ web và trình duyệt phải thực hiện để đáp ứng các yêu cầu khác nhau.
* **Endpoints** – các liên kết đại diện cho các chức năng của API mà các client có thể sử dụng.
* **HTTP verbs** – các hành động khác nhau mà có thể được thực thi đối với các điểm cuối:

|  |  |
| --- | --- |
| **METHOD** | **ACTION** |
| POST | (**C**REATE) Tạo mới một Resource. |
| GET | (**R**EAD) Trả về một Resource hoặc một danh sách Resource. |
| PUT | (**U**PDATE) Cập nhật, thay thế thông tin cho Resource. |
| DELETE | (**D**ELETE) Xoá một Resource. |
| HEAD | Giống với GET nhưng response trả về không có body, chỉ có header. |
| PATCH | Ghi đè một hoặc 1 vài thông tin được thay đổi của đối tượng. |

* **Parameters** – các thông số của request. Các tham số này có thể là tham số header hoặc body.
* **Permissions** – tập hợp các quyền mà khách hàng có liên quan đến một số API. Trong hầu hết các trường hợp, những quyền này đều liên quan đến token.
* **Response Code**: Phản hồi từ máy chủ chứa mã trạng thái để cảnh báo client về thông tin về sự thành công của hoạt động. Một số mã thường gặp:

|  |  |
| --- | --- |
| **Status code** | **Ý nghĩa** |
| 200 (OK) | Đây là phản hồi tiêu chuẩn cho các yêu cầu HTTP thành công. |
| 201 (CREATED) | Đây là phản hồi tiêu chuẩn cho yêu cầu HTTP dẫn đến việc tạo thành công một mục. |
| 204 (NO CONTENT) | Đây là phản hồi tiêu chuẩn cho các yêu cầu HTTP thành công, trong đó không có gì được trả về trong nội dung phản hồi. |
| 400 (BAD REQUEST) | Yêu cầu không thể được xử lý do cú pháp yêu cầu không đúng, kích thước quá lớn hoặc lỗi client khác. |
| 403 (FORBIDDEN) | Client không có quyền truy cập tài nguyên này. |
| 404 (NOT FOUND) | Tài nguyên không thể được tìm thấy tại thời điểm này. Có thể nó đã bị xóa hoặc chưa tồn tại. |
| 500 (INTERNAL SERVER ERROR) | Câu trả lời chung cho một lỗi không mong muốn nếu không có thông tin cụ thể hơn. |

* Limitations – Tập hợp các giới hạn mà khách hàng có liên quan đến một số API.
* Token – Cơ chế ủy quyền, một phiên bản an toàn hơn của sự kết hợp tên người dùng/mật khẩu.
* Grand Token Type – Các cách khác nhau để nhận token từ một số API và sử dụng nó.
* OAuth – Giao thức ủy quyền mở.
* JSON – Định dạng trao đổi dữ liệu dạng gọn nhẹ.
* cURL – Thư viện để thực thi các giao thức khác nhau từ dòng lệnh. Nó có thể được sử dụng để thực hiện các yêu cầu HTTP từ dòng lệnh.
* Postman – Phần mềm miễn phí để thực hiện các yêu cầu API (từ bảng điều khiển đồ họa). Một công cụ rất hữu ích và cần phải có dành cho các nhà phát triển API.
* **Ưu điểm của API REST:**
* **Hiệu năng:** các thành phần đảm bảo được việc giao tiếp theo đúng một quy ước giúp hệ thống có thể vận hành tốt hơn.
* **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 chuyển 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.
* **Đơn giản:** xây dựng rất đơn giản, ví dụ như việc khai báo resource trong rails đã giúp xây dựng các uri cho resource.
* **Tính tin cậy:** khó để xảy ra lỗi trong giao tiếp giữa các thành phần gây sụp đổ hệ thống.
* **Nhược điểm của API REST:**
* Chỉ hoạt động trên các giao thức HTTP.
* Việc bảo mật và xác thực có thể không đảm bảo bằng SOAP (stateless).
* **Làm thế nào để bảo mật API REST?**

Mọi API đều phải được bảo mật bằng phương thức xác thực và giám sát đầy đủ. Có 2 cách chính để bảo mật cho API REST:

* + **Token xác thực:**

Những token này được sử dụng để cho phép người dùng thực hiện lệnh gọi API. Token xác thực kiểm tra xem thông tin nhận dạng người dùng nhập có chính xác không và họ có quyền truy cập lệnh gọi API cụ thể đó không. Ví dụ: khi bạn đăng nhập vào máy chủ email, máy khách email của bạn sẽ dùng token xác thực để bảo mật hoạt động truy cập.

* + **Khóa API:**

Khóa API xác thực chương trình hoặc ứng dụng thực hiện lệnh gọi API. Các khóa này nhận dạng ứng dụng và đảm bảo khóa có quyền truy cập cần thiết để thực hiện lệnh gọi API cụ thể. Khóa API không bảo mật như token nhưng chúng cho phép giám sát API để thu thập dữ liệu về việc sử dụng. Bạn có thể nhận thấy những chuỗi ký tự và chữ số dài trong URL trình duyệt khi bạn truy cập các trang web khác nhau. Chuỗi này là một khóa API mà trang web sử dụng để thực hiện lệnh gọi API nội bộ.

1. **RESTful API:**

* **RESTful API là gì?**

RESTful API là là một kiểu kiến trúc cho 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, web service…) khác nhau giao tiếp với nhau.

1. Các yêu cầu

* **Các ràng buộc của REST:**

Để được coi là một hế thống REST, hệ thống cần đảm bảo được 5 ràng buộc sau:

* 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.

1. Các quy tắc

* **6 quy tắc của REST:**

1. **Truyền thông phi trạng thái:**

Điều này có nghĩa là mỗi yêu cầu từ máy khách đến máy chủ phải chứa tất cả thông tin cần thiết để máy chủ thực hiện thao tác được yêu cầu mà không dựa vào thông tin được lưu trữ từ các tương tác trước đó.

Để đảm bảo liên lạc không trạng thái trong API REST của bạn, hãy làm theo các nguyên tắc sau:

* Bao gồm tất cả thông tin cần thiết trong mỗi yêu cầu API, chẳng hạn như mã thông báo xác thực, số nhận dạng và tải trọng dữ liệu, để máy chủ có thể xử lý yêu cầu một cách độc lập.
* Tránh lưu trữ trạng thái cụ thể của khách hàng trên máy chủ; sử dụng bộ nhớ phía máy khách cho bất kỳ yêu cầu quản lý phiên nào.
* Giảm thiểu sự phụ thuộc giữa các yêu cầu để cải thiện khả năng chịu lỗi và đơn giản hóa việc triển khai ứng dụng khách.

1. **Khả năng lưu trữ và hệ thống phân lớp:**

* **Khả năng lưu vào bộ nhớ đệm**

API REST phải tạo điều kiện thuận lợi cho việc lưu vào bộ nhớ đệm các phản hồi để cải thiện hiệu suất. Bằng cách lưu trữ dữ liệu phản hồi vào bộ nhớ đệm, máy khách có thể giảm độ trễ của các yêu cầu tiếp theo, giảm thiểu tải trên máy chủ và giảm lưu lượng truy cập trên mạng.

* **Hệ thống phân lớp**

Kiến trúc hệ thống phân lớp chia các mối quan tâm thành các lớp khác nhau, chẳng hạn như giao diện người dùng, logic nghiệp vụ và các lớp truy cập dữ liệu trong một ứng dụng web n tầng điển hình. Trong API REST, việc triển khai hệ thống phân lớp có thể nâng cao khả năng lưu trữ, bảo mật và khả năng quản lý.

1. **Sử dụng các phương pháp tiêu chuẩn và giao diện thống nhất**

Điều này liên quan đến việc sử dụng các quy ước nhất quán và các phương thức HTTP tiêu chuẩn để xử lý các yêu cầu API. Bằng cách tuân thủ các tiêu chuẩn này, nhà phát triển có thể giảm đáng kể độ phức tạp của việc triển khai và duy trì API.

1. **HATEOAS - Hypermedia là công cụ của trạng thái ứng dụng**

HATEOAS (Hypermedia as the Engine of Application State) là một hạn chế chính trong thiết kế API RESTful và đảm bảo rằng các tài nguyên được kết nối với nhau thông qua các liên kết hypermedia. Bằng cách cho phép khách hàng điều hướng API bằng cách đi theo các liên kết này, việc hiểu và khám phá các tài nguyên và hành động có sẵn sẽ trở nên dễ dàng hơn.

Để kết hợp HATEOAS trong API REST, hãy đưa các liên kết siêu phương tiện có liên quan vào biểu diễn tài nguyên và sử dụng các loại phương tiện được tiêu chuẩn hóa để truyền tải các mối quan hệ liên kết.

1. **Hỗ trợ mã theo yêu cầu**

Mã theo yêu cầu là một ràng buộc tùy chọn của API REST, cho phép máy chủ cung cấp logic ứng dụng để thực hiện các hành động cụ thể trên tài nguyên. Mặc dù không phải lúc nào cũng có thể áp dụng được nhưng nó cho phép linh hoạt và mở rộng hơn trong một số trường hợp nhất định. Lợi ích chính của Code-on-Demand là khả năng chuyển mã thực thi từ máy chủ sang máy khách, cho phép máy khách chạy mã đó và thực hiện các hành động được yêu cầu. Điều này có thể giảm số lượng mã hóa cứng cần thiết ở phía máy khách, cũng như hỗ trợ mở rộng chức năng của API mà không yêu cầu cập nhật đáng kể cho máy khách.

1. **Quy ước đặt tên rõ ràng và nhất quán**

Việc áp dụng các quy ước đặt tên rõ ràng và nhất quán là rất quan trọng để làm cho các API REST dễ hiểu và dễ điều hướng đối với các nhà phát triển. Quy ước đặt tên không nhất quán có thể gây nhầm lẫn cho khách hàng và làm tăng thời gian học cách sử dụng API. Việc tuân thủ các quy tắc và mẫu đã thiết lập giúp API RESTful có thể dự đoán được, dẫn đến phát triển nhanh hơn và áp dụng rộng rãi

* **Các nguyên tắc của RESTful API:**

1. **Sự tương tác giữa Client và Server trong Restful API:**

Restful API dựa trên kiểu tương tác Client-Server, trong đó các tài nguyên (resource) của server được truy cập và sử dụng bởi client thông qua các yêu cầu HTTP.

1. **Các phương thức HTTP trong Restful API:**

Restful API sử dụng các phương thức HTTP tiêu chuẩn như GET, POST, PUT và DELETE để truy cập và quản lý các tài nguyên của server. Các phương thức này được sử dụng để định nghĩa các hành động như lấy thông tin, tạo mới, cập nhật và xóa các tài nguyên trên server.

1. **Resource:**

Restful API định nghĩa tài nguyên (resource) là một đối tượng được quản lý trên server, ví dụ như một bài viết trên blog, một người dùng, hoặc một sản phẩm trong cửa hàng. Mỗi tài nguyên được đại diện bằng một URI (Uniform Resource Identifier).

1. **URI:**

URI (Uniform Resource Identifier) được sử dụng để định danh các tài nguyên trên server. URI được sử dụng để chỉ định địa chỉ của tài nguyên trên server, cũng như các yêu cầu để thực hiện các thao tác trên tài nguyên đó.

1. **Representation:**

Mỗi tài nguyên trong Restful API được đại diện bằng một định dạng cụ thể, ví dụ như HTML, JSON, XML hoặc YAML. Định dạng này được sử dụng để truyền thông tin giữa client và server.

1. **Metadata:**

Restful API sử dụng metadata để cung cấp thông tin về các tài nguyên như định dạng, ngôn ngữ, thời gian tạo và sửa đổi của tài nguyên. Metadata có thể được sử dụng để quản lý các tài nguyên trên server.

1. Các quy chuẩn

* **Quy chuẩn đặt tên các nguồn data:**
* Trong thiết kế REST API, việc đặt tên đại diện cho các nguồn data mà chúng ta sử dụng (hay còn gọi là Resouces) thực sự rất rất quan trọng. Việc đặt tên chuẩn và clear nhất sẽ có tác dụng lâu dài trong việc thiết kế đối với các project mà độ scalable cực kì lớn.
* Một Resource có thể là một chủ thể đơn lẻ (singleton) hoặc có thể là một tập hợp (collection). Ví dụ: Trong domain của 1 ngân hàng, “customer” là một singleton resource, “customers” là một collection resource.
* Đối với collection resource, ta có thể nhận định nó bằng URI **"/customers"**.
* Đối với singleton resource, chúng ta sử dụng:  **“/customers/{customerId}”**
* Một Resource cũng có thể chứa nhiều các tập hợp khác (sub-collection). Ví dụ: trong ngân hàng, một chủ thể customer nắm giữ thông tin của các tài khoản accounts khác nhau. Chúng ta có thể định danh API này bằng URN:

**“/customers/{customerId}/accounts”** để lấy toàn bộ thông tin sub-collection tài khoản của 1 user bất kì. Tương tự nếu muốn lấy 1 singleton resource account của accounts : **/customers/{customerId}/accounts/{accountId}**

* **Quy chuẩn đặt tên các URI:**
* **Sử dụng danh từ để đại diện cho Resource.**

RESTful API nên liên quan tới resource là một danh từ chỉ đối tượng thay vì một động từ chỉ hành động vì danh từ thường có các thuộc tính (properties) mà đông từ không có, cũng giống như resource thì thường có các attributes đi kèm

* **Nhất quán URI theo một chuẩn chung**
* Sử dụng "/" để ngăn cách quan hệ trong resource
* Không sử dụng "/" cuối API
* Sử dụng "-" thay vì “\_”
* Sử dụng chữ cái thường
* Không sử dụng định dạng file
* **Không phân định CRUD trên URI**

Vì vấn đề bảo mật, URI không nên được khai báo để cho biết rằng hành động CRUD đang được gọi.

* **Sử dụng query để filter các URI collection**

Thay vì viết API mới, mất rất nhiều thời gian. Chúng ta nên sử dụng các API cũ với thêm các query truyền vào