ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

# BÀI TẬP NHÓM SỐ 11

## BỘ MÔN: KIẾN TRÚC HƯỚNG DỊCH VỤ

### LỚP MÔN HỌC: INT3505 21

# CHỦ ĐỀ:

Tìm hiểu về Thiết kế API

Version 1.0 Rev. 03

Ngày nộp: 16 tháng 05, năm 2021.

# DANH SÁCH THÀNH VIÊN NHÓM:

- LÊ ĐÌNH HOÀNG (MSSV: 17021254)

- NGUYỄN ĐÌNH NHẬT MINH (MSSV: 17021298)

## I. Giới thiệu:

- Như chúng ta đã biết, API (Application Programming Interface) là tập những định nghĩa và giao thức để xây dựng và tích hợp hệ thống phần mềm. Như vậy, API đóng vai trò trung gian liên kết giao tiếp các dịch vụ khác nhau, cho phép nhà phát triển tích hợp các dịch vụ khác nhau vào hệ thống của mình, cũng như đảm nhiệm việc giao tiếp giữa các dịch vụ trong hệ thống với nhau. Do đó, thiết kế API cũng mang vai trò quan trọng trong việc phát triển hệ thống phần mềm.

## II. Thiết kế API là gì? Tại sao cần thiết kế API?

- Thiết kế API là quá trình phát triển các API để đưa dữ liệu và chức năng của chương trình tới người dùng và các nhà phát triển. Việc thiết kế API về cơ bản dựa trên yếu tố quan trọng nhất là chiến lược kinh doanh và mục tiêu dành cho API đó  của doanh nghiệp.
- Thông thường, phát triển API ngoài bản thân các API ra thì còn phải xét tới giá trị cốt lõi của việc phát triển này mang tới cho doanh nghiệp. Những gì những API đó mang lại, kết quả nó làm được là điều quan trọng nhất trong việc phát triển API, chứ không phải bên trong API được lập trình tỉ mỉ tới đâu,... .
- Về thực tiễn, API cần được hiểu bởi người phát triển sử dụng API đó, cho dù hệ thống kia sử dụng API nhưng cài đặt để sử dụng nó lại là do nhà phát triển.

## III. Các yếu tố cơ bản trong thiết kế API:

**1. Xác định người sử dụng API:**

- Để có thể xây dựng các API hữu ích, đầu tiên chúng ta cần xác định nó phục vụ cho ai. Có thể có nhiều hơn một tác nhân sử dụng chính. Như đã nói ở mục trước, các API sẽ được sử dụng bởi các nhà phát triển, vì vậy cần xác định đối tượng nhà phát triển đó là ai, xác định dạng đối tượng đó có những nhu cầu gì, và tại sao họ cần sử dụng API đang được xây dựng.
- Xác định được đúng tác nhân sử dụng đã tạo tiền đề để phân tích các yêu cầu thiết kế, từ đó ngăn chặn việc API được làm ra không đúng mục đích sử dụng của bất kì ai.
- Thay đổi thiết kế API sau khi đã hoàn thiện rất khó khăn và tốn nhiều chi phí. Vì vậy, cần phải đặc tả yêu cầu và đánh giá cẩn thận trước khi tích hợp API vào các hệ thống. Để làm vậy, trước tiên cần phải xác định chính xác các đối tượng sử dụng API.

**2. Xác định và đạc tả các yêu cầu/ca sử dụng:**

- Sau khi xác định người sử dụng API, ta cần phải dặc tả những yêu cầu cho từng đối tượng sử dụng. Thay vì đoán mò các yêu cầu của từng đối tượng, chúng ta có thể chủ động liên hệ những đối tượng khách hàng tiềm năng sẽ sử dụng API của chúng ta, và làm việc với những khách hàng này ngay từ những khâu đầu tiên của công việc thiết kế API.
- Sau khi thu thập được các yêu cầu cụ thể, chúng ta cần phân loại chúng vào các nhóm đơn giản, có thể tuỳ thuộc theo chức năng hay nhóm người sử dụng. Ví dụ như nhóm chức năng cho tài khoản người dùng, nhóm chức năng cho quản lý sản phẩm,... . Việc phân nhóm yêu cầu/ca sử dụng không phải lúc nào cũng dễ dàng, có những ca sử dụng sẽ bị lai tạp tính năng giữa các nhóm. Do đó, việc phân nhóm ca sử dụng cũng giúp chúng ta hình dung được luồng hoạt động của API đó, cũng như phát hiện các vấn đề khó để giải quyết sơm từ đầu.

**3. Đặc tả sự tương tác với các dịch vụ khác của hệ thống:**

- Như chúng ta đã tìm hiểu, hệ thống Vi dịch vụ được tạo bởi các dịch vụ hoạt động đồng thời trong một hệ thống thay vì một hệ thống đơn khối đảm nhiệm nhiều chức năng; hay các hệ thống đơn khối cần mở rộng chức năng qua dịch vụ đều cần phải sử dụng API để giao tiếp giữa các dịch vụ. Theo đó, việc đặc tả giao tiếp liên dịch vụ là rất cần thiết trong sự hoạt động của API.
- Nhà phát triển cần phải cố gắng làm cho API có thể sử dụng được lâu nhất có thể mà không cần thay đổi (khả dụng lâu), cho dù hệ thống và các dịch vụ có thể thay đổi rất nhiều. Để làm được vậy, cần phải tối thiểu hoá sự ghép nối giữa các dịch vụ, về thực tiễn đối với thiết kế API là giới hạn hết sức số lượng dịch vụ hay hệ thống có thể tương tác tới API của chúng ta, đồng thời cũng cần giới hạn sự ghép nối với các dịch vụ cần tương tác đó.

**4. Lập phương án bảo trì API:**

- Giống như các hệ thống phần mềm, API cũng sẽ có lỗi, dù ít hơn và kém nghiêm trọng hơn so với các hệ thống hay dịch vụ. Vì vậy, cũng cần phải định kỳ gỡ lỗi và bổ sung tính măng mới dựa trên các ca sử dụng mới của từng khách hàng tuỳ thuộc theo bối cảnh doanh nghiệp hoạt động thực tế tại thời điểm triển khai đó.
- API cần phải hoạt động được trong thời gian dài mà không cần phải chỉnh sửa những dịch vụ cài đặt để sử dụng API đó, vì vậy nên tránh việc đánh phiên bản cho API để đảm bảo sự tương thích này, và chỉ sử dụng khi thực sự cần thiết phải ngăn chặn không cho một phiên bản API cũ hoạt động.

**5. Làm văn bản cho API:**

- Cũng giống như với các hệ thống phần mềm, API cũng cần phải có văn bản để đặc tả cách hoạt động, đặc tả kỹ thuật,...với tất cả những nhà phát triển làm việc với API. Do đó, văn bản của API cũng cần được cập nhật mỗi khi API có sự thay đổi.

**6. Xác định cách nhà phát triển tương tác với API:**

- Dựa trên các yêu cầu đã được xác định ở bước trên, chúng ta có thể xác định được cần phải thiết kế tương tác như thế nào. Một số kiểu tương tác điển hình: tương tác mở (không cần key), tương tác bằng token (VD như OAuth),... và các cơ chế bảo vệ dữ liệu nhà phát triển và người dùng đã được đề cập đến trong bài tập về nhà tuần 8: tìm hiểu về an ninh cho API.

**7. Lên phương án hỗ trợ người dùng:**

- Trong quá trình vận hành, những lỗi phát sinh hay tính năng mới cần được đưa phản hồi lại tới doanh nghiệp làm API nhằm cải thiện và cải tiến API theo từng giai đoạn, vì vậy, phương án hỗ trợ người dùng là rất quan trọng.
- Có nhiều phương án, tuỳ thuộc và quy mô và yêu cầu hỗ trợ của API, một số phương án phổ biến: sử dụng đội hỗ trợ riêng, sử dụng nền tảng bên thứ ba làm nền tảng hỗ trợ, thương mại hoá dịch vụ hỗ trợ,... .

**8. Lựa chọn mô hình phù hợp:**

- Lựa chọn mô hình API phù hợp là rất quan trọng, bởi nó xác định cách dữ liệu được đưa tới các dịch vụ cần sử dụng API như thế nào. Lựa chọn mô hình thích hợp, với các igao thức và mẫu thiết kế thích hợp có thể giúp tiết kiệm tiền bạc, công sức, thời gian và khiến cho API không còn quá nhạy cảm với những sự thay đổi trong tương lai.

## IV. Các mô hình API:

### 1. API yêu cầu-phản hồi (Request-Response API):

- Các API yêu cầu-phản hồi thường hiển thị giao diện thông qua máy chủ web dựa trên HTTP. API xác định một tập hợp các điểm cuối. Máy khách thực hiện các yêu cầu HTTP đối với dữ liệu đến các điểm cuối đó và máy chủ trả về các phản hồi. Phản hồi thường được gửi lại dưới dạng JSON hoặc XML. Có ba mô hình phổ biến: REST, RPC và GraphQL.

*1.1. REST:*

- REST là lựa chọn phổ biến nhất để phát triển API gần đây. Mô hình REST được sử dụng bởi các nhà cung cấp như Google, Stripe, Twitter và GitHub. REST là tất cả về tài nguyên. Tài nguyên là một thực thể có thể được xác định, đặt tên, địa chỉ hoặc xử lý trên web. Các API REST hiển thị dữ liệu dưới dạng nguồn cung cấp và sử dụng các phương thức HTTP tiêu chuẩn để thể hiện các giao dịch Tạo, Đọc, Cập nhật và Xóa (CRUD) dựa trên các tài nguyên này.

*1.2. RPC:*

- (RPC) là một trong những mô hình API đơn giản nhất, trong đó máy khách thực thi một khối mã trên máy chủ khác. Trong khi REST là về tài nguyên, RPC là về các hành động. Khách hàng thường chuyển một tên phương thức và các đối số đến một máy chủ và nhận lại JSON hoặc XML. Các API RPC thường tuân theo hai quy tắc đơn giản:

+ Các điểm cuối chứa tên của hoạt động sẽ được thực hiện.
+ Các lệnh gọi API được thực hiện với động từ HTTP thích hợp nhất: GET cho các yêu cầu chỉ đọc và POST cho những người khác.

*1.3. GraphQL:*

- GraphQL là một ngôn ngữ truy vấn cho các API đã đạt được sức hút đáng kể trong thời gian gần đây. Nó được Facebook phát triển nội bộ vào năm 2012 trước khi phát hành công khai vào năm 2015 và đã được các nhà cung cấp API như GitHub, Yelp và Pinterest chấp nhận. GraphQL cho phép máy khách xác định cấu trúc của dữ liệu được yêu cầu và máy chủ trả về chính xác cấu trúc đó.

### 2. API hướng sự kiện (Event-Driven API):

- Với các API yêu cầu-phản hồi, đối với các dịch vụ có dữ liệu thay đổi liên tục, phản hồi có thể nhanh chóng trở nên cũ. Các nhà phát triển muốn cập nhật những thay đổi trong dữ liệu thường đi đến quyết định thăm dò API (polling API). Với tính năng thăm dò, các nhà phát triển liên tục truy vấn các điểm cuối API theo tần suất xác định trước và tìm kiếm dữ liệu mới. Có ba cơ chế phổ biến: WebHooks, WebSockets và HTTP Streaming.

*2.1 WebHooks:*

- WebHook chỉ là một URL chấp nhận HTTP POST (hoặc GET, PUT, hoặc XÓA). Một nhà cung cấp API triển khai WebHooks sẽ đơn giản ĐĂNG thông báo lên URL đã định cấu hình khi có điều gì đó xảy ra. Không giống như các API yêu cầu-phản hồi, với WebHooks, bạn có thể nhận thông tin cập nhật trong thời gian thực. Một số nhà cung cấp API, như Slack, Stripe, GitHub và Zapier, hỗ trợ WebHooks.

*2.2 Websockets:*

- WebSocket là một giao thức được sử dụng để thiết lập kênh truyền thông hai chiều qua một kết nối Giao thức điều khiển truyền tải (TCP) duy nhất. Mặc dù giao thức thường được sử dụng giữa

máy khách web (ví dụ: trình duyệt) và máy chủ, nhưng đôi khi nó cũng được sử dụng cho giao tiếp giữa máy chủ với máy chủ

*2.3 HTTP Streamming:*

- Với các API phản hồi yêu cầu HTTP, máy khách gửi một yêu cầu HTTP và máy chủ trả về phản hồi HTTP có độ dài hữu hạn. Giờ đây, có thể đặt thời lượng của phản hồi này là vô thời hạn. Với HTTP Streaming, máy chủ có thể tiếp tục đẩy dữ liệu mới trong một kết nối lâu dài duy nhất do máy khách mở.

## V. Lưu văn bản cho API:

- Vì mục tiêu của bất kỳ API nào là triển khai cho nhà phát triển, nên điều quan trọng là không được quên một trong những nội dung quan trọng nhất của bạn — một nội dung mà bạn nên tham khảo cả trong thông báo lỗi và có thể ngay cả trong phản hồi siêu phương tiện của bạn: tài liệu.
- Tài liệu là một trong những yếu tố quan trọng nhất trong việc xác định thành công của API, vì tài liệu mạnh mẽ, dễ hiểu giúp cho việc triển khai API trở nên dễ dàng. Thật không may, tài liệu có thể là một trong những thách thức lớn nhất, vì cho đến nay, người viết API vẫn phải viết tài liệu khi họ cố gắng lấy nó từ các bình luận mã hoặc ghép nó lại với nhau sau khi API đã được phát triển.
- Thách thức là không chỉ tài liệu của API phải nhất quán về hình thức mà còn phải nhất quán với chức năng của API và đồng bộ với những thay đổi mới nhất. Tài liệu đó cũng phải dễ hiểu và được viết cho các nhà phát triển (thường là bởi một nhóm tài liệu có kinh nghiệm).
- Các giải pháp cho tài liệu đã bao gồm các hệ thống của bên thứ ba, việc sử dụng CMS (Hệ thống quản lý nội dung) hiện có hoặc thậm chí là CMS chuyên dụng dựa trên phần mềm nguồn mở như Drupal / WordPress. Nhưng chúng ta cũng gặp phải một thách thức nữa là trong khi các giải pháp dành riêng cho tài liệu API đắt tiền có thể cung cấp tính nhất quán về giao diện API (thứ khó duy trì hơn với CMS), chúng vẫn dựa vào nỗ lực thủ công của nhà phát triển (nếu bắt nguồn từ mã) hoặc một nhóm tài liệu để giữ cho chúng được đồng bộ hóa.
- Với sự mở rộng của các thông số kỹ thuật mở như RAML, việc lập tài liệu trở nên dễ dàng hơn rất nhiều. Thay vì cố gắng phân tích cú pháp mã nhận xét và có các mô tả nội tuyến được viết (thường là) bởi các nhà phát triển, nhóm tài liệu vẫn có thể cung cấp tài liệu mô tả trong thông số kỹ thuật và tất cả các tham số / ví dụ mã đã được bao gồm, giúp việc chuyển đổi sang tài liệu trở nên nhanh chóng.
- Một tài liệu tốt phải rõ ràng và ngắn gọn, nhưng cũng phải trực quan, cung cấp những điều sau:

+ Giải thích rõ ràng về chức năng của phương pháp / tài nguyên.
+ Gọi điện chia sẻ thông tin quan trọng với nhà phát triển, bao gồm cả các cảnh báo và lỗi
+ Một cuộc gọi mẫu với nội dung loại phương tiện tương quan.
+ Danh sách các tham số được sử dụng trên tài nguyên / phương thức này, cũng như các loại c của chúng, định dạng đặc biệt, quy tắc và liệu chúng có được yêu cầu hay không.
+ Phản hồi mẫu, bao gồm nội dung loại phương tiện.
+ Ví dụ về mã cho nhiều ngôn ngữ bao gồm tất cả các mã cần thiết (VD: Curl với PHP, cũng như các ví dụ cho Java, .Net, Ruby, v.v.).
+ Ví dụ về SDK (nếu SDK được cung cấp) cho biết cách truy cập tài nguyên / phương pháp sử dụng SDK cho ngôn ngữ của họ.

+ Trải nghiệm tương tác để thử / kiểm tra lệnh gọi API (Bảng điều khiển API, Sổ tay API)
+ Các câu hỏi / tình huống thường gặp với các ví dụ về mã.
+ Liên kết đến các tài nguyên bổ sung (ví dụ khác, blog, v.v.).
+ Phần nhận xét nơi người dùng có thể chia sẻ / thảo luận về mã.
+ Các nguồn hỗ trợ khác (diễn đàn, biểu mẫu liên hệ, v.v.).

## VI. Những phương pháp tốt nhất khi thiết kế API (best practices):

- Chúng ta nên thiết kế API:
+ Cho các trường hợp sử dụng trong đời thực.
+ Cho trải nghiệm tốt nhất cho nhà phát triển.
+ Dễ hiểu và dễ nắm bắt.
+ Hướng tới sự nhất quán.
+ Giúp cho việc khắc phục sự cố trở nên dễ dàng.
+ Làm cho API của có thể mở rộng.

## VII. Đánh giá API:

- Về cơ bản. bất kỳ API nào thoả mãn tất cả các yêu cầu được đưa ra đều có thể coi là một API tốt, các yêu cầu thường được đưa ra dựa trên tính khả dụng, hiệu năng, yeu cầu văn bản, và tính mở rộng, cũng như các yêu cầu phi chức năng khác của các bên liên quan. Nhưng không phải lúc nào tất cả các yêu cầu đều có thể được thoả mãn, ví dụ như với chi phí có hạn thì không thể có hiệu năng vượt trội; khi đó, đội phát triển và các bên liên quan cần thoả thuận lại các yêu cầu của API, nhằm cân bằng giữa các yêu cầu về các khía cạnh khác nhau cảu việc phát triển. Ngoài ra, một API tốt là API có thể chịu được sự thử thách của thời gian mà không cần thay đổi, cho dù hệ thống có đổi thay.
- Các thuộc tính đánh giá về mặt tổng quan phổ biến khác cho API:
+ Tính đúng đắn và hoàn thiện: API có thể thực hiện được tất cả mọi chức năng nó cần phải làm trong yêu cầu.
+ Tính độc nhất: các chức năng của API phải tối giản và không trùng lặp.
+ Tính phi lệ thuộc: người sử dụng API không lệ thuộc và không quan tâm cách bên trong API được cài đặt như thế nào.
+ Tính linh hoạt: tất cả các hoạt động được đặc tả trong yêu cầu đều phải thực hiện được mà không cần cài đặt thêm trường hợp đặc biệt.
+ Tính mở rộng: hệ thống có thể được mở rộng và cải tiến mà không cần thay đổi thiết kế API.
+ Tính an ninh: hệ thống cần phải có cơ chế ngăn chặn những hoạt động trái phép. Xem thêm trong bài tập về nhà số 8: Tìm hiểu về an ninh API.