**Resources:**

- Book 1: Designing Web APIs, link:

https://www.dropbox.com/s/c5majx6nmr5uvzt/refs.zip?dl=0&file\_subpath=%2FDesigning+Web+APIs+Building+APIs+That+Developers+Love.pdf

- Book 2: Undistributed REST, link:

https://www.dropbox.com/s/c5majx6nmr5uvzt/refs.zip?dl=0&file\_subpath=%2FUndisturbedREST\_v1.pdf

**Rules:**

- Chỉ dịch tóm tắt ý, không dịch mọi thứ, nhưng phải đủ đề mục.

- Không cần dịch "closing thoughts", hay các phần đóng dề/tóm tắt cuốic ác chương.

- Không cần dịch các tình huống ví dụ thực tiễn dài dòng trong các mục (ví dụ ngắn thì tuỳ ý, thích thì dịch không thích thì thôi).

**Unfinished:**

- Mục IV, V, VI.

API Design:

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 thiét 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,... .

2. 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:

**(Book 1, Chapter 2: API Paradigms, page 9).**

V. Lưu văn bản cho API:

**(Book 2, Chapter 12: Documenting and Sharing Your API, page 153)**

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

**(Book 1, Chapter 4: Design Best Practice, page 47)**

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.