**Trường Đại Học Công Nghiệp Hà Nội**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO MÔN HỌC PHÁT TRIỂN PHẦN MỀM HƯỚNG DỊCH VỤ**

**ĐỀ TÀI: NỀN TẢNG TƯ VẤN THÔNG SỐ KỸ THUẬT Ô TÔ THEO KIẾN TRÚC HƯỚNG DỊCH VỤ**

**GVDH: Hoàng Quang Huy**

**Nhóm: 16**

**Họ tên sinh viên: Nguyễn Đình Hội - 2022603030**

**Lê Thị Ngọc Lan - 2022602329**

**Lê Văn Hải - 2022603180**

**Đào Xuân Giang – 2022603320**

**Lớp: 20241IT6151001**

**Hà Nội, tháng 12 năm 2024**

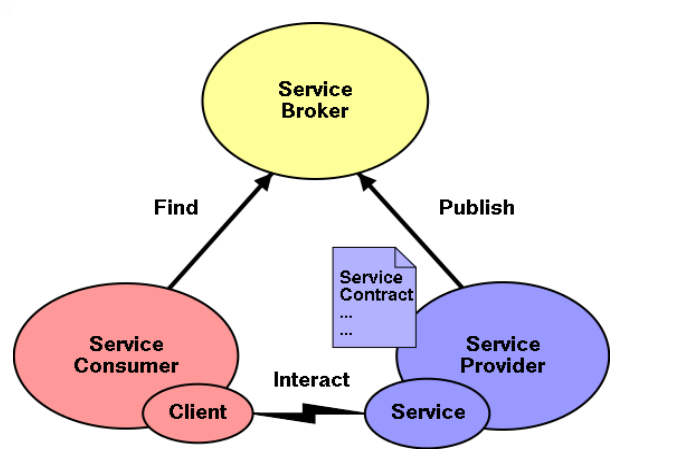
# **CHƯƠNG 1: KIẾN TRÚC HƯỚNG DỊCH VỤ**

# **Tìm kiểu về kiến trúc hướng dịch vụ (SOA)**

### **Kiến trúc hướng dịch vụ là gì?**

Kiến trúc hướng dịch vụ (SOA) là một phương pháp phát triển phần mềm sử dụng các thành phần của phần mềm được gọi là dịch vụ để tạo ra các ứng dụng dành cho doanh nghiệp. Mỗi dịch vụ cung cấp một tính năng doanh nghiệp, đồng thời các dịch vụ cũng có thể giao tiếp với nhau giữa nhiều nền tảng và ngôn ngữ. Nhà phát triển tận dụng SOA để tái sử dụng các dịch vụ trong nhiều hệ thống khác nhau hoặc kết hợp một số dịch vụ độc lập để thực hiện các tác vụ phức tạp.

Ví dụ: Một ví dụ về kiến ​​trúc hướng dịch vụ cho người tiêu dùng là ứng dụng dành cho người chạy bộ. Thay vì xây dựng các công cụ tùy chỉnh để theo dõi khoảng cách và bản đồ, ứng dụng có thể tích hợp các dịch vụ cho thiết bị GPS của điện thoại và dịch vụ bản đồ hiện có, chẳng hạn như Google Maps.



Sơ đồ cộng tác trong SOA

Có 4 yếu tố quan trọng cấu thành nên một hệ thống SOA:

* **Platform (Nền tảng):** Nền tảng của SOA bao gồm các công nghệ và hạ tầng cần thiết cho việc phát triển, triển khai, và vận hành các dịch vụ:
* **Middleware**: Là phần mềm trung gian hỗ trợ giao tiếp giữa các dịch vụ khác nhau. Middleware giúp kết nối các dịch vụ lại với nhau và quản lý các giao thức giao tiếp, truyền tải dữ liệu.
* **Enterprise Service Bus (ESB)**: ESB cung cấp khả năng điều phối dữ liệu và giao tiếp giữa các dịch vụ khác nhau. ESB có thể chuyển đổi dữ liệu từ định dạng của dịch vụ này sang định dạng của dịch vụ khác, cho phép các dịch vụ tương tác mà không cần phụ thuộc chặt chẽ vào nhau.
* **Web Services**: Đây là các dịch vụ dựa trên giao thức web như SOAP (Simple Object Access Protocol) và REST (Representational State Transfer) để các dịch vụ có thể giao tiếp với nhau qua mạng. SOAP thường được dùng cho các giao tiếp cần tính bảo mật và tiêu chuẩn cao, trong khi REST linh hoạt và nhẹ hơn, phổ biến trong các ứng dụng web hiện đại.
* **API Gateway**: Là cổng quản lý truy cập đến các dịch vụ. API Gateway điều phối lưu lượng truy cập, kiểm soát bảo mật, quản lý các phiên bản API, và cung cấp khả năng quản lý tập trung, giúp các dịch vụ bảo mật hơn và quản lý dễ dàng hơn.
* **People (Yếu tố con người):** Yếu tố con người trong SOA đóng vai trò quan trọng trong việc phát triển, vận hành, và quản lý các dịch vụ.
* **Developers (Nhà phát triển)**: Các nhà phát triển chịu trách nhiệm thiết kế và xây dựng các dịch vụ độc lập theo mô hình SOA.
* **Architects (Kiến trúc sư)**: Là người đảm bảo các dịch vụ được thiết kế tuân thủ các nguyên tắc SOA. Họ định hình cấu trúc tổng thể của hệ thống để các dịch vụ có thể tương tác hiệu quả và linh hoạt.
* **Operators (Người vận hành)**: Là người chịu trách nhiệm giám sát và quản lý các dịch vụ khi chúng được triển khai trong môi trường sản xuất. Các operator phải đảm bảo các dịch vụ hoạt động ổn định, giám sát các lỗi phát sinh, xử lý các sự cố và duy trì tính liên tục của hệ thống.
* **Business Analysts (Chuyên viên phân tích nghiệp vụ)**: Những người này định nghĩa các yêu cầu nghiệp vụ, đảm bảo các dịch vụ được phát triển đáp ứng mục tiêu của tổ chức.
* **Process (Quy trình):** SOA yêu cầu quy trình quản lý và vận hành để đảm bảo các dịch vụ hoạt động có hệ thống, nhất quán, và an toàn:
* **Service Lifecycle Management (Quản lý vòng đời dịch vụ)**: Đây là quy trình quản lý các dịch vụ từ giai đoạn thiết kế, phát triển, kiểm thử, triển khai, và bảo trì.
* **Governance (Quản trị)**: Quản trị SOA liên quan đến việc thiết lập các quy tắc, tiêu chuẩn, và chính sách cho các dịch vụ. Nó bao gồm quản lý quyền truy cập, bảo mật, đảm bảo tính nhất quán và tương thích giữa các dịch vụ.
* **Change Management (Quản lý thay đổi)**: Đây là quy trình để kiểm soát và quản lý các thay đổi trong dịch vụ. Điều này giúp các dịch vụ cập nhật mà không làm gián đoạn các dịch vụ đang hoạt động, từ đó hạn chế ảnh hưởng tiêu cực đến hệ thống.
* **Monitoring and Optimization (Giám sát và tối ưu hóa)**: Giám sát dịch vụ nhằm phát hiện sớm các vấn đề, từ đó tối ưu hóa hiệu suất và sử dụng tài nguyên hiệu quả. Việc giám sát và tối ưu hóa giúp duy trì tính ổn định và nâng cao hiệu quả của hệ thống.
* **Practice** **(Thực hành)**:
* **Designing for Reusability (Thiết kế để tái sử dụng)**: Các dịch vụ nên được xây dựng để có thể tái sử dụng trong nhiều ứng dụng mà không cần chỉnh sửa nhiều. Điều này tiết kiệm tài nguyên và giảm thời gian phát triển.
* **Loose Coupling (Liên kết lỏng lẻo)**: Các dịch vụ liên kết lỏng lẻo, không phụ thuộc quá nhiều vào nhau. Nhờ vậy, khi cập nhật hoặc thay thế dịch vụ, hệ thống vẫn hoạt động mà không cần thay đổi các dịch vụ khác.
* **Service Contracts (Hợp đồng dịch vụ)**: Mỗi dịch vụ có "hợp đồng" xác định đầu vào, đầu ra và cách thức hoạt động. Điều này giúp các bên liên quan hiểu và tích hợp dịch vụ dễ dàng hơn.
* **Security (Bảo mật)**: Bảo mật là yếu tố quan trọng trong SOA. Xác thực, mã hóa dữ liệu và quản lý truy cập giúp đảm bảo chỉ người được ủy quyền mới có thể sử dụng dịch vụ.
* **Monitoring and Management (Giám sát và Quản lý)**: Giám sát dịch vụ giúp phát hiện các vấn đề kịp thời, theo dõi hiệu suất, và tối ưu hóa dịch vụ để hoạt động hiệu quả hơn.
* **Versioning (Quản lý phiên bản)**: Khi dịch vụ có thay đổi hoặc nâng cấp, cần duy trì nhiều phiên bản dịch vụ để không ảnh hưởng đến các ứng dụng sử dụng phiên bản cũ.
* **Testing (Kiểm thử)**: Kiểm thử đơn vị, tích hợp và hiệu năng giúp đảm bảo các dịch vụ hoạt động chính xác, đặc biệt khi kết hợp với các dịch vụ khác trong hệ thống SOA.

A diagram of a service oriented architecture

Description automatically generated

## **Kiến trúc hướng dịch vụ mang lại những lợi ích gì?**

* + - Rút ngắn thời gian đưa ra thị trường :Nhà phát triển tái sử dụng các dịch vụ trên những quy trình kinh doanh khác nhau để tiết kiệm thời gian và chi phí. Họ có thể hợp dịch các ứng dụng nhanh hơn bằng SOA so với việc lập trình và thực hiện tích hợp từ đầu.
    - Bảo trì hiệu quả : Các dịch vụ nhỏ dễ tạo dựng, cập nhật và khắc phục lỗi hơn những đoạn mã lớn trong ứng dụng đơn khối. Việc sửa đổi bất kỳ dịch vụ nào trong SOA cũng không làm ảnh hưởng đến chức năng tổng thể của quy trình kinh doanh.
    - Khả năng thích ứng cao hơn: SOA dễ thích ứng hơn với những cải tiến về công nghệ. Bạn có thể hiện đại hóa các ứng dụng của mình một cách hiệu quả và tiết kiệm.

## **Những nguyên tắc cơ bản của kiến trúc hướng dịch vụ là gì?**

* Khả năng tương tác: Mỗi dịch vụ trong SOA bao gồm các tài liệu mô tả chỉ rõ chức năng của dịch vụ cùng các điều khoản và điều kiện liên quan. Mọi hệ thống máy khách đều có thể vận hành dịch vụ, dù dùng nền tảng cơ sở hay ngôn ngữ lập trình gì.
* Liên kết ít phụ thuộc: Các dịch vụ trong SOA cần được liên kết ít phụ thuộc, có càng ít sự phụ thuộc vào các tài nguyên bên ngoài như mô hình dữ liệu hay hệ thống thông tin thì càng tốt. Các dịch vụ này cũng nên ở tình trạng vô trạng thái mà không giữ lại bất kỳ thông tin nào từ các phiên làm việc hay giao dịch trước. Bằng cách này, khi bạn sửa đổi một dịch vụ, các ứng dụng máy khách và dịch vụ khác đang sử dụng dịch vụ này sẽ không bị ảnh hưởng đáng kể.
* Tính trừu trượng: Khách hàng hoặc người sử dụng dịch vụ trong SOA không cần biết logic lập trình hay chi tiết triển khai của dịch vụ. Đối với họ, dịch vụ nên như một chiếc hộp đen. Khách hàng nhận được thông tin cần thiết về tính năng và cách sử dụng dịch vụ thông qua hợp đồng dịch vụ và các tài liệu mô tả dịch vụ khác.
* Độ chi tiết: Các dịch vụ trong SOA nên có kích thước và phạm vi phù hợp, lý tưởng là tính đóng gói một chức năng kinh doanh riêng biệt mỗi dịch vụ. Sau đó, nhà phát triển có thể sử dụng nhiều dịch vụ để tạo ra một dịch vụ tổng hợp phục vụ việc thực hiện những thao tác phức tạp.

## **Kiến trúc hướng dịch vụ gồm những thành phần nào?**

### **Service (Dịch vụ)**

Dịch vụ là những khối dựng cơ bản của SOA. Chúng có thể là dịch vụ tư nhân – chỉ dành cho người dùng nội bộ của một tổ chức – hoặc công cộng – tất cả mọi người đều có thể truy cập dịch vụ đó qua Internet. Cụ thể, mỗi dịch vụ có ba đặc điểm chính.

* Triển khai dịch vụ: Triển khai dịch vụ là phần mã xây dựng logic để thực hiện chức năng dịch vụ cụ thể.
* Hợp đồng dịch vụ: Hợp đồng dịch vụ xác định bản chất của dịch vụ cùng các điều khoản và điều kiện liên quan, chẳng hạn như những điều kiện tiên quyết để sử dụng dịch vụ, chi phí dịch vụ, và chất lượng của dịch vụ được cung cấp.
* Giao diện dịch vụ: Trong SOA, các dịch vụ hoặc hệ thống khác giao tiếp với một dịch vụ thông qua giao diện của dịch vụ đó. Giao diện xác định cách bạn có thể gọi dịch vụ để thực hiện các hoạt động hoặc trao đổi dữ liệu. Giao diện giúp làm giảm sự phụ thuộc giữa dịch vụ và trình yêu cầu dịch vụ.

### **Service provider (Nhà cung cấp dịch vụ)**

Nhà cung cấp dịch vụ tạo dựng, duy trì và cung cấp một hoặc nhiều dịch vụ mà người khác có thể sử dụng. Các tổ chức có thể tự tạo dịch vụ của riêng hoặc mua từ một nhà cung cấp dịch vụ bên thứ ba.

### **Service customer (Người tiêu dùng dịch vụ)**

Người sử dụng dịch vụ yêu cầu nhà cung cấp dịch vụ vận hành một dịch vụ cụ thể. Đó có thể là cả một hệ thống, một ứng dụng, hay một dịch vụ khác. Hợp đồng dịch vụ nêu rõ các quy tắc nhà cung cấp dịch vụ và người sử dụng dịch vụ phải tuân theo khi tương tác với nhau. Nhà cung cấp dịch vụ và người sử dụng dịch vụ có thể đến từ các bộ phận, tổ chức và thậm chí là các ngành khác nhau.

### **Service registry (Sổ đăng ký dịch vụ)**

Sổ đăng ký dịch vụ, hay kho dịch vụ, là một danh mục các dịch vụ có sẵn, có thể truy cập qua mạng. Sổ đăng ký dịch vụ này lưu trữ các tài liệu mô tả dịch vụ từ các nhà cung cấp dịch vụ. Tài liệu mô tả dịch vụ chứa thông tin và cách giao tiếp với dịch vụ. Người sử dụng dịch vụ có thể dễ dàng tìm được những dịch vụ họ cần bằng cách dùng sổ đăng ký dịch vụ.

## **Kiến trúc hướng dịch vụ hoạt động như thế nào?**

Các dịch vụ giao tiếp bằng những quy tắc được thiết lập sẵn, quyết định việc trao đổi thông tin qua một mạng lưới. Những quy tắc này được gọi là giao thức giao tiếp. Sau đây là một vài giao thức tiêu chuẩn để triển khai SOA:

* Giao thức truy cập đối tượng đơn giản( SOAP)
* HTTP RESTful
* Apache Thrift
* Apache ActiveMQ
* Dịch vụ thông báo Java (JMS)



# **So sánh và đánh giá kiến trúc SOA với kiến trúc Monolithic**

## **Kiến trúc hướng dịch vụ (SOA)**

A computer with icons and words

Description automatically generated

### **Ưu điểm của SOA**

#### **Khả năng tái sử dụng các dịch vụ**

Do tính chất độc lập và không liên kết chặt chẽ của các thành phần chức năng trong các ứng dụng hướng dịch vụ, các thành phần này có thể được tái sử dụng trong nhiều ứng dụng mà không ảnh hưởng đến các dịch vụ khác.

#### **Khả năng bảo trì tốt hơn**

Vì mỗi dịch vụ phần mềm là một đơn vị độc lập nên dễ dàng cập nhật và bảo trì mà không làm ảnh hưởng đến các dịch vụ khác. Ví dụ, các ứng dụng doanh nghiệp lớn có thể được quản lý dễ dàng hơn khi được chia thành các dịch vụ.

#### **Độ tin cậy cao hơn**

Dịch vụ dễ gỡ lỗi và kiểm tra hơn là các khối mã lớn như trong phương pháp tiếp cận đơn khối. Điều này, đến lượt nó, làm cho các sản phẩm dựa trên SOA đáng tin cậy hơn.

#### **Phát triển song song**

Vì kiến ​​trúc hướng dịch vụ bao gồm nhiều lớp, nên nó ủng hộ tính song song trong quá trình phát triển. Các dịch vụ độc lập có thể được phát triển song song và hoàn thành cùng lúc. Dưới đây, bạn có thể thấy cách phát triển ứng dụng SOA được thực hiện bởi nhiều nhà phát triển song song

### **Nhược điểm của SOA**

#### **Quản lý phức tạp**

Nhược điểm chính của kiến ​​trúc hướng dịch vụ là tính phức tạp của nó. Mỗi dịch vụ phải đảm bảo rằng các thông điệp được gửi đi đúng thời hạn. Số lượng các thông điệp này có thể lên tới hơn một triệu cùng một lúc, khiến việc quản lý tất cả các dịch vụ trở thành một thách thức lớn.

#### **Chi phí đầu tư cao**

Phát triển SOA đòi hỏi sự đầu tư ban đầu lớn về nguồn nhân lực, công nghệ và phát triển.

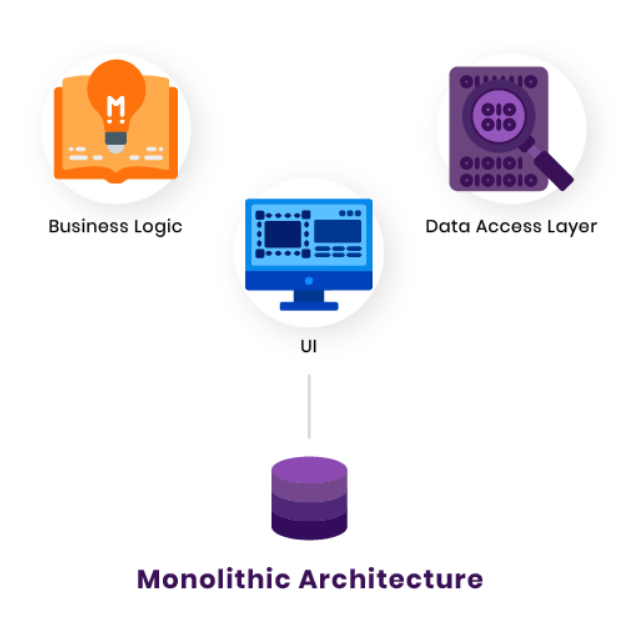
#### **Quá tải thêm**

Trong SOA, tất cả các đầu vào đều được xác thực trước khi một dịch vụ tương tác với một dịch vụ khác. Khi sử dụng nhiều dịch vụ, điều này làm tăng thời gian phản hồi và giảm hiệu suất tổng thể.

### **Đánh giá**

Phương pháp SOA phù hợp nhất với các hệ thống doanh nghiệp phức tạp như hệ thống ngân hàng. Hệ thống ngân hàng cực kỳ khó để chia thành các dịch vụ vi mô. Nhưng phương pháp monolithic cũng không tốt cho hệ thống ngân hàng vì một phần có thể gây hại cho toàn bộ ứng dụng. Giải pháp tốt nhất là sử dụng phương pháp SOA và tổ chức các ứng dụng phức tạp thành các dịch vụ độc lập riêng biệt.

## **Kiến trúc monolithic**



### **Ưu điểm của kiến trúc monolithic**

#### **Phát triển và triển khai đơn giản hơn**

Có rất nhiều công cụ bạn có thể tích hợp để tạo điều kiện phát triển. Ngoài ra, tất cả các hành động được thực hiện với một thư mục, giúp triển khai dễ dàng hơn. Với lõi đơn khối, các nhà phát triển không cần triển khai các thay đổi hoặc cập nhật riêng biệt, vì họ có thể thực hiện cùng một lúc và tiết kiệm rất nhiều thời gian.

* **Tốc độ phát triển**
  + Vì mọi phần của ứng dụng đều được tích hợp chặt chẽ nên việc phát triển các tính năng mới sẽ nhanh hơn.
  + Các nhà phát triển có thể thực hiện thay đổi đối với cơ sở mã mà không phải lo lắng về việc làm hỏng các phần khác của ứng dụng.
  + Điều này có thể giúp rút ngắn chu kỳ phát triển và thời gian đưa các tính năng mới ra thị trường.
* **Gỡ lỗi**
  + Việc gỡ lỗi và theo dõi sự cố trong một ứng dụng độc lập thường dễ dàng hơn vì mọi thứ đều được kết nối và ở cùng một nơi.
  + Các nhà phát triển có thể sử dụng các công cụ để theo dõi luồng thực thi thông qua ứng dụng, giúp xác định và sửa lỗi dễ dàng hơn

### **Nhược điểm của kiến trúc Monolithic**

#### **Cơ sở mã trở nên cồng kềnh theo thời gian**

Theo thời gian cơ sở mã bắt đầu trông thực sự đồ sộ và trở nên khó hiểu và khó sửa đổi. Với cơ sở mã ngày càng phát triển, chất lượng giảm và môi trường phát triển tích hợp (IDE) bị quá tải.

#### **Khó khăn trong việc áp dụng công nghệ mới**

Nếu cần thêm một số công nghệ mới vào ứng dụng của bạn, các nhà phát triển có thể gặp phải rào cản khi áp dụng. Thêm công nghệ mới có nghĩa là phải viết lại toàn bộ ứng dụng, tốn kém và mất thời gian.

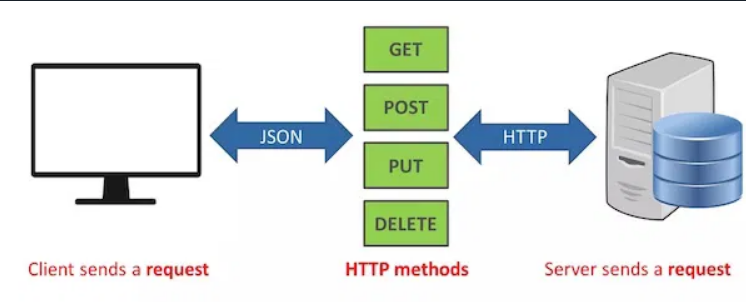
#### **Tính linh hoạt hạn chế**

Trong các ứng dụng đơn khối, mọi bản cập nhật nhỏ đều yêu cầu triển khai lại toàn bộ. Do đó, tất cả các nhà phát triển phải đợi cho đến khi hoàn tất. Khi nhiều nhóm làm việc trên cùng một dự án, tính linh hoạt có thể giảm đáng kể.

### **Đánh giá**

Kiến trúc monolithic rất dễ làm việc với các nhóm nhỏ, đó là lý do tại sao nhiều công ty khởi nghiệp chọn cách tiếp cận này khi xây dựng ứng dụng. Các thành phần của phần mềm monolithic được kết nối và phụ thuộc lẫn nhau, giúp phần mềm trở nên độc lập.

# **Mô hình triển khai SOA**



## **RESTful API là gì?**

RESTful API là giao diện mà hai hệ thống máy tính sử dụng để trao đổi thông tin một cách an toàn qua internet. Hầu hết các ứng dụng kinh doanh phải giao tiếp với các ứng dụng nội bộ và bên thứ ba khác để thực hiện nhiều tác vụ khác nhau. Nó chú trọng vào tài nguyên hệ thống (tệp văn bản, ảnh, âm thanh, video, hoặc dữ liệu động…), bao gồm các trạng thái tài nguyên được định dạng và được truyền tải qua HTTP.

## **Lợi ích của RESTful API?**

### Khả năng mở rộng

Các hệ thống triển khai REST API có thể mở rộng hiệu quả vì REST tối ưu hóa tương tác giữa máy khách và máy chủ. Tính không trạng thái loại bỏ tải máy chủ vì máy chủ không phải lưu giữ thông tin yêu cầu của máy khách trước đó. Bộ nhớ đệm được quản lý tốt loại bỏ một phần hoặc toàn bộ một số tương tác giữa máy khách và máy chủ. Tất cả các tính năng này hỗ trợ khả năng mở rộng mà không gây ra tình trạng tắc nghẽn giao tiếp làm giảm hiệu suất.

### Tính linh hoạt

Dịch vụ web RESTful hỗ trợ tách biệt hoàn toàn giữa máy khách và máy chủ. Chúng đơn giản hóa và tách rời các thành phần máy chủ khác nhau để mỗi phần có thể phát triển độc lập. Các thay đổi về nền tảng hoặc công nghệ tại ứng dụng máy chủ không ảnh hưởng đến ứng dụng máy khách. Khả năng phân lớp các chức năng ứng dụng làm tăng tính linh hoạt hơn nữa. Ví dụ, các nhà phát triển có thể thực hiện các thay đổi đối với lớp cơ sở dữ liệu mà không cần viết lại logic ứng dụng.

### Độc lập

REST API độc lập với công nghệ được sử dụng. Bạn có thể viết cả ứng dụng máy khách và máy chủ bằng nhiều ngôn ngữ lập trình khác nhau mà không ảnh hưởng đến thiết kế API. Bạn cũng có thể thay đổi công nghệ cơ bản ở cả hai bên mà không ảnh hưởng đến giao tiếp.

## **RESTful API hoạt động như thế nào**

 Máy khách liên hệ với máy chủ bằng cách sử dụng API khi cần một tài nguyên.

* Máy khách gửi yêu cầu đến máy chủ. Máy khách làm theo tài liệu API để định dạng yêu cầu theo cách mà máy chủ có thể hiểu được.
* Máy chủ xác thực máy khách và xác nhận rằng máy khách có quyền thực hiện yêu cầu đó.
* Máy chủ nhận được yêu cầu và xử lý nội bộ.
* Máy chủ trả về phản hồi cho máy khách. Phản hồi chứa thông tin cho máy khách biết yêu cầu có thành công hay không. Phản hồi cũng bao gồm bất kỳ thông tin nào mà máy khách yêu cầu.

## **Thành phần của máy khách RESTful API**

### **Mã định danh tài nguyên duy nhất**

Máy chủ xác định từng tài nguyên bằng các mã định danh tài nguyên duy nhất. Máy chủ thực hiện xác định tài nguyên bằng cách sử dụng Uniform Resource Locator (URL). URL chỉ định đường dẫn đến tài nguyên.

### **Methods**

Các nhà phát triển thường triển khai API RESTful bằng cách sử dụng Giao thức truyền siêu văn bản (HTTP). Một phương thức HTTP cho máy chủ biết những gì cần làm với tài nguyên:

* **GET**

GET để truy cập tài nguyên nằm tại URL được chỉ định trên máy chủ. Có thể lưu trữ các yêu cầu GET và gửi tham số trong yêu cầu API RESTful để hướng dẫn máy chủ lọc dữ liệu trước khi gửi.

* **POST**

Máy khách sử dụng POST để gửi dữ liệu đến máy chủ. Chúng bao gồm biểu diễn dữ liệu với yêu cầu. Gửi cùng một yêu cầu POST nhiều lần có tác dụng phụ là tạo cùng một tài nguyên nhiều lần.

* **PUT**

Khách hàng sử dụng PUT để cập nhật tài nguyên hiện có trên máy chủ. Không giống như POST, việc gửi cùng một yêu cầu PUT nhiều lần trong dịch vụ web RESTful sẽ mang lại cùng một kết quả.

* **DELETE**

Khách hàng sử dụng yêu cầu DELETE để xóa tài nguyên. Yêu cầu DELETE có thể thay đổi trạng thái máy chủ. Tuy nhiên, nếu người dùng không có xác thực phù hợp, yêu cầu sẽ không thành công.

### **HTTP headers**

* Dữ liệu

Các yêu cầu API REST có thể bao gồm dữ liệu cho POST, PUT và các phương thức HTTP khác để hoạt động thành công.

* Các tham số

Yêu cầu API RESTful có thể bao gồm các tham số cung cấp cho máy chủ nhiều thông tin chi tiết hơn về những gì cần thực hiện.

* Tham số đường dẫn chỉ định chi tiết URL.
* Tham số truy vấn yêu cầu thêm thông tin về tài nguyên.
* Các tham số cookie xác thực khách hàng một cách nhanh chóng.

## **Phương pháp xác thực RESTful API**

### **Khóa API**

Khóa API là một tùy chọn khác để xác thực REST API. Trong phương pháp này, máy chủ gán một giá trị được tạo duy nhất cho máy khách lần đầu. Bất cứ khi nào máy khách cố gắng truy cập tài nguyên, nó sẽ sử dụng khóa API duy nhất để xác minh chính nó.

## **Phản hồi của máy chủ**

### **Dòng trạng thái**

Dòng trạng thái chứa mã trạng thái ba chữ số thông báo yêu cầu thành công hay thất bại. Ví dụ, mã 2XX biểu thị thành công, nhưng mã 4XX và 5XX biểu thị lỗi. Mã 3XX biểu thị chuyển hướng URL.

# **CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HƯỚNG DỊCH VỤ**

# **CHƯƠNG 3: THIẾT KẾ VÀ CÀI ĐẶT CHƯƠNG TRÌNH**

# **CHƯƠNG 4: SỬ DỤNG CÁC CÔNG CỤ CHẠY CHƯƠNG TRÌNH VÀ KIỂM THỬ**

# **CHƯƠNG 5: ĐÁNH GIÁ HƯỚNG PHÁT TRIỂN**