**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THÀNH PHỐ HỒ CHÍ MINH**



TÀI LIỆU BÁO CÁO ĐỒ ÁN

Môn học: Kiến trúc và Thiết kế phần mềm

*Đề tài****:*** ***Hệ thống quản lý lớp học tín chỉ***

|  |  |
| --- | --- |
| Lớp học phần  Nhóm  GVHD | : DHKTPM16A  : 27  : Võ Văn Hải |

|  |  |  |
| --- | --- | --- |
| **STT** | **HỌ VÀ TÊN** | **MSSV** |
| 1 | Nguyễn Văn Thuận | 20069161 |
| 2 | Lê Thế Vinh | 20070671 |
| 3 | Nguyễn Mạnh Trí | 20067011 |

Thành phố Hồ Chí Minh, tháng 5 năm 2024

**Mục lục**

[LỜI MỞ ĐẦU 3](#_Toc167053366)

[CHƯƠNG I: CƠ SỞ LÝ THUYẾT 4](#_Toc167053367)

[1. Kiến trúc phần mềm 4](#_Toc167053368)

[1.1. Kiến trúc phần mềm là gì 4](#_Toc167053369)

[1.2. Các nguyên lý cơ bản của Kiến trúc Phần mềm 4](#_Toc167053370)

[1.3. Các đặc tính của Kiến trúc Phần mềm 4](#_Toc167053371)

[1.4. Nguyên tắc SOLID của Kiến trúc Phần mềm 5](#_Toc167053372)

[1.5. Tầm quan trọng của Kiến trúc Phần mềm 5](#_Toc167053373)

[1.6. Ưu điểm của Kiến trúc Phần mềm 6](#_Toc167053374)

[1.7. Nhược điểm của Kiến trúc Phần mềm 6](#_Toc167053375)

[2. Các kiểu kiến trúc phần mềm phổ biến 6](#_Toc167053376)

[2.1. Kiến trúc nguyên khối (Monolithic Architecture) 6](#_Toc167053377)

[2.2. Kiến trúc phân lớp (Layered Architecture) 6](#_Toc167053378)

[2.3. Kiến trúc Pipes and Filters (Pipes and Filters Architecture) 7](#_Toc167053379)

[2.4. Kiến trúc hướng dịch vụ (Service-Oriented Architecture - SOA) 7](#_Toc167053380)

[2.5. Kiến trúc Microservices (Microservices Architecture) 8](#_Toc167053381)

[2.6. Kiến trúc hướng sự kiện (Event-Driven Architecture) 8](#_Toc167053382)

[2.7. Kiến trúc bộ phận (Component-Based Architecture) 9](#_Toc167053383)

[CHƯƠNG II: HIỆN THỰC 10](#_Toc167053384)

[1. Lý do chọn Kiến trúc Microservices 10](#_Toc167053385)

[2. Các dịch vụ trong hệ thống 10](#_Toc167053386)

[3. Mô hình kiến trúc của hệ thống 11](#_Toc167053387)

[4. Mặt hạn chế của kiến trúc 11](#_Toc167053388)

[4.1. Độ phức tạp 11](#_Toc167053389)

[4.2. Giao tiếp giữa các dịch vụ 11](#_Toc167053390)

[4.3. Đảm bảo tính nhất quán của dữ liệu 11](#_Toc167053391)

[4.4. Bảo mật và quản lý truy cập 12](#_Toc167053392)

[4.5. Chi phí phát triển và vận hành 12](#_Toc167053393)

[4.6. Công cụ và kiến thức chuyên môn 12](#_Toc167053394)

# **LỜI MỞ ĐẦU**

Trong thế giới công nghệ, từ trẻ nhỏ, người trẻ, trung niên cho đến người già, mọi người đều sử dụng điện thoại thông minh, máy tính xách tay, máy tính, PDA, v.v. để giải quyết mọi tác vụ đơn giản cho đến phức tạp trong cuộc sống bằng cách sử dụng một số chương trình phần mềm, mọi thứ trông rất đơn giản đối với góc nhìn của người dùng. Đó cũng là mục đích của một phần mềm tốt nhằm cung cấp chất lượng dịch vụ tốt và môi trường thân thiện với người dùng. Ở đó, sự trừu tượng tổng thể của bất kỳ sản phẩm phần mềm nào làm cho những ứng dụng trông đơn giản và rất dễ sử dụng hơn cho người dùng. Nhưng ngược lại, việc xây dựng một ứng dụng phần mềm phức lại tạp hơn nhiều bao gồm những quy trình phức tạp.

Một yếu tố quan trọng trong quá trình xây dựng phần mềm là kiến trúc phần mềm. Đây là yếu tố quyết định cách các thành phần trong hệ thống phần mềm tương tác với nhau và định hình tổng thể cấu trúc của ứng dụng. Kiến trúc phần mềm không chỉ ảnh hưởng đến khả năng mở rộng, bảo trì và hiệu suất của hệ thống mà còn đảm bảo rằng các yêu cầu kinh doanh và kỹ thuật được đáp ứng một cách hiệu quả.

Trong bối cảnh đó, bài tập này sẽ đi sâu vào việc lựa chọn kiến trúc phù hợp nhất cho hệ thống quản lý lớp học tín chỉ tại một trường đại học. Hệ thống này cần đảm bảo khả năng quản lý đăng ký học phần, kiểm soát số lượng tín chỉ, hỗ trợ các môn học tiên quyết và cung cấp một giao diện thân thiện với người dùng. Chúng ta sẽ xem xét các yêu cầu cụ thể của hệ thống và lựa chọn kiến trúc phần mềm phù hợp để đáp ứng các yêu cầu đó một cách tốt nhất.

# **CHƯƠNG I: CƠ SỞ LÝ THUYẾT**

## **Kiến trúc phần mềm**

### **Kiến trúc phần mềm là gì**

Kiến trúc phần mềm định nghĩa tổ chức cơ bản của một hệ thống và đơn giản hơn là giải pháp có cấu trúc. Nó xác định cách các thành phần của một hệ thống phần mềm được lắp ráp, mối quan hệ và giao tiếp giữa chúng. Nó phục vụ như một bản thiết kế cho ứng dụng phần mềm và là cơ sở phát triển cho nhóm phát triển.

### **Các nguyên lý cơ bản của Kiến trúc Phần mềm**

* Cấu trúc hệ thống: Kiến trúc phần mềm định nghĩa cấu trúc của hệ thống, xác định cách các thành phần được sắp xếp và tổ chức.
* Hành vi hệ thống: Nó mô tả cách các thành phần của hệ thống tương tác và hoạt động cùng nhau để đáp ứng các yêu cầu nghiệp vụ.
* Mối quan hệ giữa các thành phần: Kiến trúc xác định cách các thành phần trong hệ thống liên kết và tương tác với nhau.
* Cấu trúc giao tiếp: Nó thiết lập các giao diện và cơ chế giao tiếp giữa các thành phần phần mềm.
* Cân bằng nhu cầu của các bên liên quan: Kiến trúc phần mềm giúp cân bằng và đáp ứng các nhu cầu và yêu cầu khác nhau của các bên liên quan.
* Ảnh hưởng đến cấu trúc nhóm: Kiến trúc phần mềm thường phản ánh và ảnh hưởng đến cách tổ chức nhóm phát triển.
* Tập trung vào các yếu tố quan trọng: Kiến trúc nhấn mạnh các yếu tố và quyết định thiết kế quan trọng sớm trong quá trình phát triển.
* Nắm bắt các quyết định thiết kế ban đầu: Nó giúp ghi lại và truyền đạt các quyết định thiết kế chính từ giai đoạn đầu của dự án.

### **Các đặc tính của Kiến trúc Phần mềm**

Kiến trúc phần mềm bao gồm các đặc tính được phân thành các danh mục rộng dựa trên hoạt động, yêu cầu hiếm khi xuất hiện, cấu trúc, v.v. Dưới đây là một số đặc tính quan trọng thường được xem xét:

* Đặc tính Hoạt động:
  + Sẵn có (Availability)
  + Hiệu suất (Performance)
  + Độ tin cậy (Reliability)
  + Khả năng chịu lỗi thấp (Low Fault Tolerance)
  + Khả năng mở rộng (Scalability)
* Đặc tính Cấu trúc:
  + Khả năng cấu hình (Configurability)
  + Khả năng mở rộng (Extensibility)
  + Khả năng hỗ trợ (Supportability)
  + Khả năng di động (Portability)
  + Khả năng bảo trì (Maintainability)
* Đặc tính Xuyên suốt:
  + Khả năng tiếp cận (Accessibility)
  + Bảo mật (Security)
  + Khả năng sử dụng (Usability)
  + Quyền riêng tư (Privacy)
  + Khả năng thực thi (Feasibility)

### **Nguyên tắc SOLID của Kiến trúc Phần mềm**

Mỗi ký tự của từ SOLID đại diện cho một nguyên tắc của kiến trúc phần mềm. Các nguyên tắc này được tuân thủ để tránh các lỗi chiến lược trong sản phẩm. Một kiến trúc phần mềm phải tuân thủ nguyên tắc SOLID để tránh các thất bại kiến trúc hoặc phát triển.

* Nguyên tắc Trách nhiệm Đơn lẻ (Single Responsibility Principle):

Mỗi dịch vụ nên có một mục tiêu duy nhất.

* Nguyên tắc Mở-Đóng (Open-Closed Principle):

Các module phần mềm nên độc lập và có khả năng mở rộng.

* Nguyên tắc Thay thế Liskov (Liskov Substitution Principle):

Các dịch vụ độc lập nên có khả năng giao tiếp và thay thế lẫn nhau.

* Nguyên tắc Phân tách Giao diện (Interface Segregation Principle):

Phần mềm nên được chia thành các dịch vụ nhỏ để tránh sự dư thừa.

* Nguyên tắc Đảo ngược Phụ thuộc (Dependency Inversion Principle):

Các module cấp cao không nên phụ thuộc vào các module cấp thấp; sự thay đổi ở cấp cao sẽ không ảnh hưởng đến cấp thấp.

### **Tầm quan trọng của Kiến trúc Phần mềm**

Kiến trúc phần mềm thuộc giai đoạn thiết kế của vòng đời phát triển phần mềm. Nó là một trong những bước đầu tiên của toàn bộ quá trình phát triển phần mềm. Không có kiến trúc phần mềm, tiến hành phát triển phần mềm giống như xây dựng một ngôi nhà mà không có bản thiết kế kiến trúc.

Do đó, kiến trúc phần mềm là một phần quan trọng trong phát triển ứng dụng phần mềm. Dưới đây là một số lý do kiến trúc phần mềm quan trọng từ quan điểm kỹ thuật và phát triển:

* Chọn các thuộc tính chất lượng để tối ưu hóa hệ thống.
* Tạo điều kiện cho việc tạo mẫu sớm.
* Cho phép xây dựng hệ thống theo từng thành phần.
* Giúp quản lý các thay đổi trong hệ thống.

Ngoài ra, kiến trúc phần mềm còn quan trọng đối với nhiều yếu tố khác như chất lượng phần mềm, độ tin cậy của phần mềm, khả năng bảo trì, khả năng hỗ trợ và hiệu suất của phần mềm.

### **Ưu điểm của Kiến trúc Phần mềm**

Cung cấp nền tảng vững chắc cho dự án phần mềm.

Giúp cải thiện hiệu suất.

Giảm chi phí phát triển.

### **Nhược điểm của Kiến trúc Phần mềm**

Đôi khi khó khăn trong việc tìm kiếm các công cụ và tiêu chuẩn hóa cho kiến trúc phần mềm.

Dự đoán ban đầu về thành công của dự án dựa trên kiến trúc không phải lúc nào cũng chính xác.

## **Các kiểu kiến trúc phần mềm phổ biến**

Các kiểu kiến trúc phần mềm, còn được gọi là mẫu kiến trúc, là các phương pháp đã được thiết kế để thiết kế hệ thống phần mềm. Các loại này cung cấp hướng dẫn cho việc cấu trúc một hệ thống, xác định các thành phần của nó và mô tả cách chúng tương tác với nhau. Một số kiểu kiến trúc phổ biến nhất được liệt kê dưới đây, cùng với các đặc điểm chính, ưu và nhược điểm của chúng:

### **Kiến trúc nguyên khối (Monolithic Architecture)**

Đặc điểm:

* Tất cả các thành phần của ứng dụng được gói gọn trong một khối duy nhất.
* Gồm giao diện người dùng, logic nghiệp vụ, và cơ sở dữ liệu trong một ứng dụng duy nhất.

Ưu điểm:

* Dễ phát triển và thử nghiệm trong giai đoạn ban đầu.
* Đơn giản trong triển khai và quản lý.
* Hiệu suất tốt khi ứng dụng còn nhỏ.

Nhược điểm:

* Khó mở rộng và duy trì khi ứng dụng phát triển.
* Tạo ra điểm lỗi duy nhất (single point of failure).
* Khó khăn trong việc sử dụng các công nghệ khác nhau cho các phần khác nhau của ứng dụng.
* Triển khai thay đổi đòi hỏi phải xây dựng lại và triển khai toàn bộ ứng dụng.

### **Kiến trúc phân lớp (Layered Architecture)**

Đặc điểm:

* Ứng dụng được chia thành các tầng (layers), mỗi tầng chịu trách nhiệm một phần của ứng dụng.
* Thường có các tầng như: Presentation (Giao diện người dùng), Business Logic (Logic nghiệp vụ), Data Access (Truy cập dữ liệu), và Database (Cơ sở dữ liệu).

Ưu điểm:

* Tính module cao, dễ bảo trì và mở rộng.
* Dễ dàng thay thế hoặc nâng cấp từng tầng mà không ảnh hưởng đến các tầng khác.
* Rõ ràng và có cấu trúc tốt, dễ hiểu và quản lý.

Nhược điểm:

* Có thể gặp vấn đề hiệu suất do phải qua nhiều lớp trung gian.
* Sự phụ thuộc giữa các lớp có thể làm giảm tính linh hoạt.
* Khó tối ưu hóa toàn bộ hệ thống do các lớp có thể bị cô lập về mặt hiệu suất.

### **Kiến trúc Pipes and Filters (Pipes and Filters Architecture)**

Đặc điểm:

Ứng dụng được chia thành các bộ lọc (filters) thực hiện các thao tác dữ liệu, và các ống dẫn (pipes) để truyền dữ liệu giữa các bộ lọc.

Ưu điểm:

* Tính linh hoạt cao, dễ dàng thêm hoặc thay đổi các bộ lọc.
* Tăng khả năng tái sử dụng các bộ lọc.
* Dễ dàng kiểm tra và gỡ lỗi từng bộ lọc riêng lẻ.

Nhược điểm:

* Có thể gặp vấn đề về hiệu suất do phải chuyển đổi dữ liệu nhiều lần.
* Khó khăn trong việc theo dõi và gỡ lỗi luồng dữ liệu phức tạp.
* Sự phối hợp giữa các bộ lọc có thể làm tăng độ phức tạp của hệ thống.

### **Kiến trúc hướng dịch vụ (Service-Oriented Architecture - SOA)**

Đặc điểm:

* Chia ứng dụng thành các dịch vụ độc lập, mỗi dịch vụ thực hiện một chức năng nghiệp vụ cụ thể.
* Các dịch vụ giao tiếp với nhau qua các giao thức như HTTP, SOAP, REST.

Ưu điểm:

* Tính module cao, dễ bảo trì và mở rộng.
* Tái sử dụng các dịch vụ dễ dàng.
* Hỗ trợ tính phân tán, có thể triển khai trên nhiều máy chủ khác nhau.

Nhược điểm:

* Độ phức tạp cao trong việc quản lý giao tiếp giữa các dịch vụ.
* Hiệu suất có thể bị ảnh hưởng do độ trễ mạng.
* Đòi hỏi quản lý chặt chẽ về bảo mật và xác thực dịch vụ.

### **Kiến trúc Microservices (Microservices Architecture)**

Đặc điểm:

* Ứng dụng được chia thành các dịch vụ nhỏ, độc lập, mỗi dịch vụ chịu trách nhiệm một phần chức năng cụ thể.
* Mỗi microservice có thể được phát triển, triển khai và mở rộng độc lập.

Ưu điểm:

* Tính module và linh hoạt cao, dễ mở rộng và bảo trì.
* Dễ dàng triển khai các công nghệ khác nhau cho từng microservice.
* Tăng khả năng chịu lỗi và khả năng mở rộng hệ thống.

Nhược điểm:

* Độ phức tạp cao trong việc quản lý và triển khai.
* Đòi hỏi nhiều công cụ và hạ tầng hỗ trợ.
* Khó khăn trong việc đảm bảo tính nhất quán dữ liệu và điều phối giao dịch giữa các dịch vụ.
* Giao tiếp giữa các dịch vụ có thể gây ra độ trễ và ảnh hưởng đến hiệu suất tổng thể.

### **Kiến trúc hướng sự kiện (Event-Driven Architecture)**

Đặc điểm:

* Ứng dụng được chia thành các thành phần sản xuất và tiêu thụ sự kiện.
* Các thành phần giao tiếp với nhau thông qua các sự kiện.

Ưu điểm:

* Tính module và linh hoạt cao.
* Dễ dàng mở rộng và xử lý các sự kiện theo thời gian thực.
* Tăng khả năng phản ứng nhanh chóng với các thay đổi và sự kiện xảy ra.

Nhược điểm:

* Khó khăn trong việc theo dõi và gỡ lỗi luồng sự kiện phức tạp.
* Cần các công cụ và hạ tầng để quản lý sự kiện hiệu quả.
* Có thể gặp khó khăn trong việc đảm bảo tính nhất quán của dữ liệu.

### **Kiến trúc bộ phận (Component-Based Architecture)**

Đặc điểm:

* Ứng dụng được chia thành các bộ phận độc lập, mỗi bộ phận thực hiện một chức năng cụ thể.
* Các bộ phận có thể được phát triển và triển khai độc lập.

Ưu điểm:

* Tính module và linh hoạt cao.
* Dễ bảo trì và mở rộng.
* Tăng khả năng tái sử dụng các bộ phận và giảm thiểu sự trùng lặp.

Nhược điểm:

* Cần các công cụ và hạ tầng để quản lý và tích hợp các bộ phận hiệu quả.
* Có thể gặp vấn đề về hiệu suất khi tích hợp nhiều bộ phận phức tạp.
* Đòi hỏi sự phối hợp chặt chẽ giữa các nhóm phát triển các bộ phận khác nhau.

# **CHƯƠNG II: HIỆN THỰC**

Kiến trúc phù hợp cho ứng dụng đăng ký tín chỉ: **Kiến trúc microservices**

## **Lý do chọn Kiến trúc Microservices**

Kiến trúc Microservices được cho là tối ưu cho hệ thống quản lý lớp học tín chỉ là vì các lý do sau:

* Phân chia ứng dụng theo các chức năng: Microservices cho phép chia hệ thống thành các dịch vụ nhỏ, mỗi dịch vụ đảm nhận một nhiệm vụ cụ thể. Ví dụ, quản lý sinh viên, môn học, đăng ký, xác nhận đăng ký, quản lý tín chỉ, gửi email, thông tin học tập và xử lý xét tốt nghiệp. Điều này giúp đơn giản hóa việc phát triển và bảo trì hệ thống.
* Phát triển và triển khai các dịch vụ độc lập: Các dịch vụ có thể được phát triển, triển khai và mở rộng độc lập. Điều này rất quan trọng trong hệ thống giáo dục, nơi các quy định và yêu cầu có thể thay đổi liên tục.
* Khả Năng Mở Rộng: Khi số lượng sinh viên và khối lượng đăng ký tín chỉ tăng lên, các microservices có thể được mở rộng độc lập để đáp ứng nhu cầu mà không ảnh hưởng đến toàn bộ hệ thống. Ví dụ, khi đến thời điểm đăng ký tín chỉ, nhà trường hoàn toàn có thể mở rộng dịch vụ này để nó có thể đáp ứng số lượng lớn sinh viên đăng ký mà không làm gián đoạn hay mở rộng các dịch vụ khác giúp tiết kiệm chi phí đáng kể trong quá trình vận hành hệ thống.
* Khả Năng Chịu Lỗi Cao: Nếu một microservice gặp lỗi, nó sẽ không ảnh hưởng đến toàn bộ hệ thống. Ví dụ, nếu dịch vụ gửi email gặp sự cố, các dịch vụ khác như quản lý đăng ký và thông tin học tập vẫn hoạt động bình thường.
* Độc Lập Về Công Nghệ: Mỗi microservice có thể được phát triển bằng ngôn ngữ lập trình và công nghệ phù hợp nhất với chức năng của nó, giúp tối ưu hóa hiệu suất và năng suất phát triển.

## **Các dịch vụ trong hệ thống**

* Student Service: Quản lý thông tin sinh viên và hồ sơ cá nhân.
* Authentication Service: Quản lý xác thực và phân quyền người dùng
* Course Service: Quản lý môn học, điều kiện tiên quyết và danh sách môn học mỗi học kỳ.
* Grade Service: Quản lý điểm số và tín chỉ của sinh viên
* Enrollment Service: Xử lý đăng ký học phần, kiểm tra giới hạn tín chỉ và điều kiện tiên quyết.
* Confirmation Service: Quản lý quá trình xác nhận đăng ký của sinh viên.
* Notification Service: Gửi email xác nhận và thông báo đóng học phí.
* Schedule Service: Quản lý thời khóa biểu và các tiện ích liên quan.
* Graduation Service: Xử lý yêu cầu xét tốt nghiệp và quản lý thông tin cựu sinh viên.

## **Mô hình kiến trúc của hệ thống**

A screenshot of a chat

Description automatically generated

## **Mặt hạn chế của kiến trúc**

### **Độ phức tạp**

* Quản lý nhiều dịch vụ: Microservices có thể biến mỗi chức năng chính thành một dịch vụ riêng lẻ. Do đó, việc quản lý và điều phối nhiều dịch vụ trở nên phức tạp hơn. Đặc biệt là trong môi tường giáo dục, nơi có nhiều quy trình phức tạp liên quan đến quản lý danh sách chờ, xác nhận môn tiên quyết và đăng ký học phần.
* Triển khai và bảo trì: Việc triển khai, cập nhật và bảo trì một số dịch vụ riêng lẻ có thể gây ra nhiều vấn đề. Điều này có thể giúp giảm chi phí và thời gian phát triển và đòi hỏi các kỹ năng và công cụ DevOps mạnh mẽ.

### **Giao tiếp giữa các dịch vụ**

* Độ trễ và hiệu suất: Giao tiếp giữa các dịch vụ thông qua internet có thể gây ra độ trễ và ảnh hưởng đến hiệu suất chung của hệ thống, đặc biệt là khi xử lý các yêu cầu thời gian thực như đăng ký lớp học.
* Quản lý phiên bản và tương thích: Khi mỗi dịch vụ có thể được phát triển và sử dụng độc lập, việc đảm bảo rằng các phiên bản dịch vụ khác nhau tương thích với nhau là một thách thức lớn.

### **Đảm bảo tính nhất quán của dữ liệu**

* Tính nhất quán cuối cùng: Trong Microservices, dữ liệu thường được phân tán giữa nhiều dịch vụ, dẫn đến khó khăn trong việc đảm bảo tính nhất quán tức thời. Điều này có thể gây ra vấn đề trong các quy trình đòi hỏi tính nhất quán cao, chẳng hạn như xác nhận đăng ký học phần và kiểm tra môn học tiên quyết.
* Quản lý giao dịch phân tán: Đối với các thao tác đòi hỏi tính nguyên tử (atomic transactions) như đăng ký môn học, việc quản lý giao dịch phân tán giữa nhiều dịch vụ là một thách thức phức tạp và dễ dẫn đến lỗi.

### **Bảo mật và quản lý truy cập**

* Bảo mật: Nhiều dịch vụ nhỏ cần có các cơ chế bảo mật riêng biệt, làm phức tạp việc quản lý bảo mật tổng thể của hệ thống.
* Quản lý khả năng truy cập: Điều khó khăn là đảm bảo rằng chỉ những người dùng có quyền hợp lệ mới có thể truy cập các dịch vụ khác nhau, đặc biệt khi phân quyền chi tiết cho nhiều vai trò khác nhau, chẳng hạn như quản trị viên, sinh viên, giảng viên.

### **Chi phí phát triển và vận hành**

* Chi phí: Thiết kế, phát triển, triển khai và vận hành một hệ thống Microservices thường đòi hỏi nhiều nguồn lực hơn so với một kiến trúc nguyên khối. Điều này bao gồm chi phí cho các công cụ DevOps, quản lý dịch vụ, và các chuyên gia có kỹ năng phù hợp.
* Tài nguyên: Việc chạy nhiều dịch vụ có thể yêu cầu nhiều tài nguyên hơn về mặt hạ tầng (máy chủ, lưu trữ, băng thông), dẫn đến chi phí vận hành cao hơn.

### **Công cụ và kiến thức chuyên môn**

* Công cụ hỗ trợ: Để quản lý hiệu quả hệ thống Microservices, cần các công cụ chuyên dụng như Kubernetes, Docker, hệ thống giám sát và logging phức tạp. Các công cụ này không phải lúc nào cũng dễ tiếp cận và sử dụng.
* Kiến thức và kỹ năng: Đội ngũ phát triển cần có kiến thức và kỹ năng chuyên sâu về Microservices, DevOps, quản lý giao dịch phân tán và các công nghệ liên quan. Điều này có thể đòi hỏi đào tạo bổ sung và tuyển dụng chuyên gia.