**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM**



**Hệ thống quản lý lớp học tín chỉ**

***Nhóm 20 - Sinh viên thực hiện***

1. Đặng Thị Quyền Cơ – 20043001
2. Đặng Thị Quyền Trân – 20042961

# MỤC LỤC

[MỤC LỤC 1](#_Toc14693502)

[DANH MỤC CÁC HÌNH VẼ 1](#_Toc14693503)

[DANH MỤC CÁC BẢNG BIỂU 1](#_Toc14693504)

[CHƯƠNG 1 : GIỚI THIỆU 1](#_Toc14693505)

[1.1 Tổng quan 1](#_Toc14693506)

[1.2 Mục tiêu đề tài 1](#_Toc14693507)

[1.3 Phạm vi đề tài 1](#_Toc14693508)

[1.4 Mô tả yêu cầu chức năng 1](#_Toc14693509)

[CHƯƠNG 2 : CƠ SỞ LÝ THUYẾT 1](#_Toc14693510)

[2.1 Cloud Computing với AWS 1](#_Toc14693511)

[2.2 Node.js 1](#_Toc14693512)

[2.3 MongoDB / DynamoDB 1](#_Toc14693513)

[CHƯƠNG 3 : PHÂN TÍCH VÀ THIẾT KẾ 1](#_Toc14693514)

[3.1 Phân tích yêu cầu bằng UML 1](#_Toc14693515)

[3.1.1 Usecase tổng quát 1](#_Toc14693516)

[3.1.2 Danh sách tác nhân và mô tả 1](#_Toc14693517)

[3.1.3 Danh sách các tình huống hoạt động (Use cases) 1](#_Toc14693518)

[3.1.4 Tình huống hoạt động 1](#_Toc14693519)

[3.2 Class diagram 1](#_Toc14693520)

[3.3 Deployment diagram 1](#_Toc14693521)

[CHƯƠNG 4 : HIỆN THỰC 1](#_Toc14693522)

[4.1 Cấu hình phần cứng, phần mềm 1](#_Toc14693523)

[4.2 Giao diện của hệ thống 1](#_Toc14693524)

[CHƯƠNG 5 : KẾT LUẬN 1](#_Toc14693525)

[5.1 Kết quả đạt được 1](#_Toc14693526)

[5.2 Hạn chế của đồ án 1](#_Toc14693527)

[5.3 Hướng phát triển 1](#_Toc14693528)

[TÀI LIỆU THAM KHẢO 1](#_Toc14693529)

# : GIỚI THIỆU

## Tổng quan

Một trường đại học cần quản lý hệ thống đăng ký học phần của sinh viên theo theo cơ chế tín chỉ. Các sinh viên sẽ được biên chế ở các khoa theo mỗi ngành học của mình đăng ký.

Trong mỗi học kỳ, nhà trường sẽ chọn một số môn học của từng ngành học cho sinh viên đăng ký học. Sinh viên đăng ký theo học vào một lớp nhất định (lớp tín chỉ) với số sinh viên được đăng ký tối đa được qui định cho từng lớp.

Môn học được mở có thể là môn học tự chọn

Mỗi môn học có một hoặc nhiều môn học tiên quyết. Sinh viên sẽ không được đăng ký môn học mà có môn tiên quyết mình chưa học.

Mỗi học kỳ sinh viên được đăng ký tối đa 30 tín chỉ. Nếu quá số này, hệ thống sẽ không cho đăng ký.

Để phòng ngừa trường hợp sinh viên đăng ký xong rồi hủy bỏ, nhà trường yêu cầu sinh viên phải xác nhận trước khi đăng ký.

Vào ngày mở đăng ký, sinh viên sẽ đăng nhập vào hệ thống và sẽ nhìn thấy danh sách các môn học mà mình có khả năng đăng ký. Sinh viên chọn các môn học và tiến hành đăng ký. Trường hợp các lớp đã đầy, sinh viên sẽ được đưa vào một danh sách dự bị để nhà trường cân nhắc có mở thêm lớp hay không. Nếu không mở thêm lớp, sinh viên sẽ bị hủy đăng ký môn đó.

Sau khi đăng ký thành công, một email thông báo sẽ được gửi cho sinh viên xác nhận việc đăng ký và nhận quyết định đóng học phí.

Ngoài ra, hệ thống còn có khả năng cho phép sinh viên xem thông tin học tập của mình (số tín chỉ đã đạt, số môn đã học, điểm môn học, điểm trung bình tích lũy…), thời khóa biểu theo tuần và các tiện ích khác.

Sau khi số tín chỉ đã đạt theo từng ngành, sinh viên có quyền đăng ký xét tốt nghiệp. Nếu mọi tiêu chuẩn đều thỏa mãn, sinh viên sẽ được cấp bằng tốt nghiệp và sẽ được đưa vào danh sách các cựu sinh viên. Thông tin về bằng cấp sẽ được công khai trên trang web của nhà trường. Thông tin của cựu sinh viên sẽ được lưu giữ để theo dõi quá trình làm việc (nếu sinh viên đồng ý), làm các cuộc survey, cũng như nhiều hoạt động khác.

## Mục tiêu đề tài

1.2.1 Quản Lý Đăng Ký Học Phần:

* Hệ thống sẽ cung cấp giao diện đăng ký học phần cho sinh viên, dựa trên cơ chế tín chỉ và ngành học của họ.
* Sinh viên chỉ được đăng ký các môn học mà họ đủ điều kiện.
* Hệ thống sẽ giới hạn số tín chỉ tối đa mỗi học kỳ theo quy định của nhà trường và không cho đăng ký nếu vượt quá giới hạn này.

1.2.2 Quản Lý Lớp Học:

* Hệ thống sẽ hiển thị số lượng sinh viên tối đa được đăng ký cho mỗi lớp và ngăn không cho đăng ký khi lớp đã đầy.

1.2.3 Xác Nhận Đăng Ký:

* Sinh viên sẽ phải xác nhận đăng ký trước khi hệ thống chấp nhận.
* Hệ thống sẽ gửi email thông báo xác nhận và quyết định đóng học phí sau khi đăng ký thành công.

1.2.4 Quản Lý Danh Sách Dự Bị:

* Hệ thống sẽ tự động đưa sinh viên vào danh sách dự bị khi lớp học đã đầy.
* Nhà trường sẽ xem xét mở thêm lớp dựa trên danh sách này.

1.2.5 Quản Lý Thông Tin Học Tập:

* Hệ thống sẽ cung cấp thông tin về số tín chỉ, số môn học, điểm môn học, điểm trung bình tích lũy và thời khóa biểu theo tuần cho sinh viên.

1.2.6 Quản Lý Tốt Nghiệp và Cựu Sinh Viên:

* Sau khi đạt đủ số tín chỉ theo ngành, sinh viên có thể đăng ký xét tốt nghiệp.
* Thông tin về bằng cấp của cựu sinh viên sẽ được công khai trên trang web của nhà trường.
* Hệ thống sẽ lưu giữ thông tin cựu sinh viên để theo dõi quá trình làm việc và tham gia các hoạt động sau tốt nghiệp (nếu có sự đồng ý của sinh viên).

1.2.7 Bảo mật và Quản lý Người Dùng:

* Hệ thống sẽ cung cấp các biện pháp bảo mật để bảo vệ thông tin cá nhân của sinh viên và người dùng khác.
* Hệ thống sẽ quản lý quyền truy cập dựa trên vai trò của người dùng, đảm bảo chỉ có quyền truy cập cần thiết được cấp.

1.2.8 Tính Mở Rộng và Hiệu Suất:

* Hệ thống sẽ được thiết kế để dễ dàng mở rộng và có hiệu suất cao, có khả năng xử lý hàng ngàn đăng ký cùng một lúc.
* Hệ thống sẽ sử dụng công nghệ và cơ sở hạ tầng linh hoạt để đảm bảo sự ổn định và đáng tin cậy.

## Phạm vi đề tài

Phạm vi dự án quản lý lớp học tín chỉ bao gồm quản lý sinh viên, môn học, đăng ký môn học, quản lý tài khoản và xác thực, gửi thông báo và email, giao diện người dùng, bảo mật, tích hợp hệ thống và cập nhật liên tục.

Top of Form

# : CƠ SỞ LÝ THUYẾT

## Kiến trúc phần mềm

### Kiến trúc phần mềm là gì?

Theo Wikipedia, kiến trúc phần mềm (Software architecture) đề cập đến cấu trúc cơ bản của một hệ thống phần mềm và quy tắc của việc tạo ra những cấu trúc và hệ thống như vậy. Mỗi cấu trúc bao gồm sự sắp xếp của các yếu tố phần mềm, mối quan hệ giữa các yếu tố, và tính chất của các yếu tố đó.

*“Software architecture is the set of structures needed to reason about a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations.”*

Thiết kế kiến trúc phần mềm (software architecture design) ẩn dụ cho việc thiết kế một toà nhà. Nó được coi như bản thiết kế của hệ thống phần mềm. Từ đó có thể hình dung ra lượng công việc cụ thể phải làm, dẫn tới, việc quản lý dự án trở nên dễ dàng hơn.

Cho tới giờ vẫn chưa có một định nghĩa phản ánh đầy đủ nhất về kiến trúc phần mềm bởi hai lý do:

* Sự phát triển nhanh chóng của công nghệ, môi trường phát triển, các loại mô hình kiến trúc phần mềm khác nhau khiến cho các định nghĩa trước đó nhanh chóng trở nên lỗi thời.
* Sự khác biệt về quan điểm giữa các trường phái. Ví dụ: một số software architect coi kiến trúc phần mềm là bản thiết kế chi tiết của hệ thống, trong khi những người khác định nghĩa nó là lộ trình phát triển hệ thống.

### Định nghĩa kiến trúc phần mềm

Có 3 định nghĩa về kiến trúc phần mềm từ các nguồn khác nhau:

* *Đầu tiên là từ IEEE*, định nghĩa kiến trúc là tổ chức cơ bản của hệ thống, bao gồm các thành phần, mối quan hệ và nguyên tắc thiết kế.
* *Sau đó, Bass, Clements và Kazman* mô tả kiến trúc là cấu trúc của hệ thống, bao gồm các yếu tố phần mềm, tính chất bên ngoài của các yếu tố đó và mối quan hệ giữa chúng.
* *Cuối cùng, Garlan và Shaw* đề cập đến kiến trúc như việc thiết kế và chỉ định cấu trúc tổng thể của hệ thống, vượt ra ngoài các thuật toán và cấu trúc dữ liệu.

Điểm chung của ba định nghĩa này là việc kiến trúc tập trung vào cấu trúc, mối quan hệ giữa các thành phần và cách mà hệ thống có thể thay đổi.

Đồng thời, để có cái nhìn tổng thể rõ ràng hơn về kiến trúc phần mềm, chúng ta cũng có thể mở rộng phần định nghĩa bằng cách xem xét các khía cạnh quan trọng khác của nó, bao gồm:

+ *Kiến trúc chỉ định giao tiếp thành phần*: Mô tả cách các thành phần của hệ thống phần mềm giao tiếp với nhau, bao gồm cách truyền dữ liệu và quản lý thông tin điều khiển.

+ *Kiến trúc giải quyết các yêu cầu phi chức năng*: tóm tắt cách mà kiến trúc phần mềm đáp ứng các yêu cầu không liên quan đến chức năng, bao gồm khả năng mở rộng, khả dụng, hiệu suất, bảo mật, và sự di động.

### Kiến trúc giải quyết các yêu cầu phi chức năng

Trong quá trình phát triển phần mềm, kiến trúc không chỉ tập trung vào việc xây dựng cấu trúc và chức năng của hệ thống, mà còn chú trọng vào việc đáp ứng các yêu cầu phi chức năng. Các yêu cầu này bao gồm những điều như hiệu suất, bảo mật, sẵn sàng, di động, và chúng đóng vai trò quan trọng trong việc đảm bảo rằng hệ thống hoạt động một cách hiệu quả và đáng tin cậy. Các yêu cầu này bao gồm:

+ Yêu cầu về *ràng buộc kỹ thuật* liên quan đến các yêu cầu đặc biệt về công nghệ hoặc giới hạn kỹ thuật mà hệ thống phải tuân thủ. Ví dụ, việc quy định ngôn ngữ lập trình hoặc hệ điều hành cụ thể có thể là một ràng buộc kỹ thuật.

+ Yêu cầu về *ràng buộc kinh doanh* là những yêu cầu được xác định dựa trên lợi ích kinh doanh của tổ chức. Ví dụ, một yêu cầu kinh doanh có thể là việc đảm bảo giao diện người dùng cuối thân thiện để tăng cường trải nghiệm người dùng và tăng cường doanh số bán hàng.

+ *Thuộc tính chất lượng* liên quan đến các yêu cầu về khả năng mở rộng, sẵn sàng, dễ bảo trì, di động, hiệu suất và bảo mật của ứng dụng. Ví dụ, một ứng dụng di động cần có khả năng sẵn sàng để người dùng có thể truy cập vào nó mọi lúc, mọi nơi.

Quan điểm kiến trúc nhấn mạnh vào việc xác định và định hình cách mà hệ thống sẽ được xây dựng để đáp ứng các yêu cầu phi chức năng. Điều này bao gồm việc lựa chọn các mô hình thiết kế, công nghệ và các phương pháp phát triển phù hợp để đảm bảo rằng hệ thống có thể mở rộng, dễ bảo trì, an toàn và đáp ứng được yêu cầu về hiệu suất.

## Một số mô hình kiến trúc phần mềm

Hiện nay, có rất nhiều mô hình kiến trúc phần mềm khác nhau, mỗi mô hình đều có những ưu điểm, nhược điểm riêng phù hợp cho từng loại ứng dụng. Dưới đây là một số mô hình kiến trúc phổ biến.

### Layered Architecture

#### Mô hình lớp (Layered Architecture) là gì?

Mô hình lớp (Layered Architecture) là một kiểu kiến trúc phần mềm trong đó hệ thống được chia thành các lớp (layer) theo chức năng. Mỗi lớp thực hiện một vai trò cụ thể và giao tiếp với các lớp lân cận theo một cách có tổ chức và có thứ tự. Các lớp thường được thiết kế theo mô hình phân cấp, trong đó mỗi lớp chỉ phụ thuộc vào lớp bên dưới nó và cung cấp các dịch vụ cho lớp trên.

#### Nguyên tắc hoạt động của mô hình lớp

**Phân chia lớp**: Hệ thống được chia thành các lớp riêng biệt dựa trên chức năng, ví dụ như:

* Lớp trình bày (Presentation Layer): Quản lý giao diện người dùng và xử lý đầu vào từ người dùng.
* Lớp nghiệp vụ (Business Logic Layer): Xử lý logic nghiệp vụ và các quy tắc kinh doanh.
* Lớp dữ liệu (Data Access Layer): Quản lý việc truy cập và lưu trữ dữ liệu.
* Lớp cơ sở hạ tầng (Infrastructure Layer): Cung cấp các dịch vụ chung như logging, giao tiếp mạng, v.v.

**Giao tiếp giữa các lớp:** Các lớp chỉ giao tiếp với lớp ngay bên dưới và bên trên nó. Dữ liệu và yêu cầu di chuyển từ lớp này sang lớp khác theo một hướng cụ thể, thường từ lớp trên xuống dưới (top-down) hoặc từ dưới lên trên (bottom-up).

**Đóng gói và trừu tượng hóa:** Mỗi lớp đóng gói chức năng của nó và cung cấp các giao diện rõ ràng để các lớp khác tương tác mà không cần biết chi tiết triển khai bên trong.

#### Ưu điểm của mô hình lớp

**Dễ hiểu và dễ quản lý:**

Cấu trúc phân lớp giúp các nhà phát triển dễ dàng hiểu và quản lý hệ thống.

Mỗi lớp có thể được phát triển và duy trì độc lập.

**Tăng khả năng tái sử dụng:**

Các lớp có thể được tái sử dụng trong các hệ thống khác nếu chúng cung cấp các chức năng chung.

Logic nghiệp vụ có thể được sử dụng lại trong nhiều ứng dụng khác nhau.

**Tăng khả năng bảo trì:**

Khi có thay đổi trong một lớp, chỉ cần thay đổi lớp đó mà không ảnh hưởng đến các lớp khác.

Giảm sự phụ thuộc và tăng tính mô-đun của hệ thống.

**Tách biệt các mối quan tâm:**

Mỗi lớp giải quyết một khía cạnh cụ thể của hệ thống, giúp tách biệt rõ ràng các mối quan tâm (separation of concerns).

#### Nhược điểm của mô hình lớp

**Hiệu suất**:

Hệ thống có thể bị chậm do phải đi qua nhiều lớp để hoàn thành một tác vụ.

Quá trình truyền tải dữ liệu giữa các lớp có thể tạo ra độ trễ.

**Khó khăn trong việc quản lý thay đổi lớn:**

Khi có thay đổi lớn trong yêu cầu nghiệp vụ, có thể phải thay đổi nhiều lớp, gây khó khăn và tốn kém.

**Cấu trúc cứng nhắc:**

Các lớp cố định và phụ thuộc lẫn nhau có thể làm hệ thống trở nên cứng nhắc, khó thích ứng với các thay đổi không mong đợi.

**Tiềm năng lãng phí tài nguyên:**

Mỗi lớp có thể lặp lại các thao tác đã được thực hiện ở lớp khác, dẫn đến lãng phí tài nguyên và thời gian.

### Microservices

#### Microservices là gì?

Mô hình Microservices là một kiến trúc phần mềm trong đó mọt ứng dụng được xây dựng như một tập hợp các dịch vụ nhỏ, độc lập. Mỗi dịch vụ (microservices) thực hiện một nhiệm vụ cụ thể và có thể triển khai, mở rộng và quản lý một cách độc lập. Các microservices giao tiếp với nhau thông qua các giao thức nhẹ như HTTP/HTTPS, REST hoặc gRPC.

#### Nguyên tắc hoạt động

**Độc lập và tự trị:**

Mỗi microservice là một đơn vị độc lập có thể phát triển, triển khai và mở rộng riêng rẽ mà không ảnh hưởng đến các dịch vụ khác.

**Chức năng đơn lẻ:**

Mỗi microservice thực hiện một chức năng cụ thể và rõ ràng, thường được xác định bởi các quy tắc nghiệp vụ cụ thể.

**Giao tiếp thông qua API:**

Các microservices giao tiếp với nhau thông qua các API, thường sử dụng giao thức HTTP/HTTPS với RESTful API hoặc gRPC. Giao tiếp thường không đồng bộ để tăng hiệu quả và khả năng chịu lỗi.

**Triển khai độc lập:**

Các microservices có thể được triển khai và nâng cấp riêng lẻ mà không cần phải dừng toàn bộ hệ thống.

**Tách biệt lưu trữ dữ liệu:**

Mỗi microservice quản lý dữ liệu của riêng nó và có cơ sở dữ liệu riêng. Điều này giúp giảm thiểu sự phụ thuộc giữa các dịch vụ.

**Tự động hóa DevOps:**

Sử dụng các công cụ CI/CD (Continuous Integration/Continuous Deployment) để tự động hóa quá trình kiểm tra, xây dựng và triển khai các microservices.

#### Ưu điểm của mô hình Microservices

**Tính linh hoạt và khả năng mở rộng:**

Mỗi microservice có thể được mở rộng độc lập, giúp hệ thống dễ dàng mở rộng theo nhu cầu.

Khả năng triển khai các dịch vụ mới hoặc nâng cấp các dịch vụ hiện có mà không làm gián đoạn toàn bộ hệ thống.

**Dễ dàng triển khai và bảo trì:**

Các đội phát triển có thể làm việc độc lập trên các microservice khác nhau, giúp tăng tốc độ phát triển và triển khai.

Dễ dàng bảo trì và nâng cấp các phần của hệ thống mà không ảnh hưởng đến các phần khác.

**Tăng khả năng chịu lỗi:**

Nếu một microservice gặp sự cố, các microservice khác vẫn có thể hoạt động bình thường, giúp hệ thống tổng thể trở nên đáng tin cậy hơn.

**Sử dụng công nghệ phù hợp nhất:**

Các đội phát triển có thể chọn ngôn ngữ lập trình và công nghệ phù hợp nhất cho từng microservice, không bị ràng buộc bởi một công nghệ chung cho toàn bộ hệ thống.

#### Nhược điểm của mô hình Microservices

**Phức tạp trong quản lý và giám sát:**

Với số lượng lớn các microservice, việc giám sát và quản lý chúng trở nên phức tạp hơn.

Yêu cầu các công cụ và kỹ thuật giám sát tiên tiến để theo dõi và quản lý trạng thái của các dịch vụ.

**Giao tiếp giữa các dịch vụ:**

Giao tiếp giữa các microservice qua mạng có thể tạo ra độ trễ và ảnh hưởng đến hiệu suất.

Cần phải xử lý các vấn đề như định tuyến, cân bằng tải, và đảm bảo tính toàn vẹn của dữ liệu trong giao tiếp giữa các dịch vụ.

**Phức tạp trong kiểm tra:**

Việc kiểm tra toàn bộ hệ thống trở nên phức tạp hơn do phải kiểm tra tích hợp giữa nhiều microservice.

Cần phải phát triển các chiến lược kiểm tra tự động và kiểm tra tích hợp phức tạp hơn.

**Quản lý dữ liệu phân tán:**

Việc quản lý dữ liệu phân tán giữa các microservice có thể gây khó khăn, đặc biệt là khi cần đảm bảo tính nhất quán của dữ liệu.

Cần áp dụng các mẫu thiết kế như SAGA hoặc Event Sourcing để quản lý giao dịch phân tán.

### Microkernel

#### Mô hình Microkernel Architecture là gì?

Mô hình Microkernel Architecture (hay còn gọi là kiến trúc nhân nhỏ) là một kiểu kiến trúc phần mềm trong đó hệ thống được chia thành một "microkernel" nhỏ gọn và các phần mở rộng hoặc module hoạt động bên ngoài nhân này. Microkernel chứa các chức năng cốt lõi cần thiết để hệ thống hoạt động, như quản lý bộ nhớ, xử lý luồng và giao tiếp cơ bản. Các chức năng khác như dịch vụ hệ điều hành, giao diện người dùng, và các dịch vụ cao cấp khác được triển khai dưới dạng các module độc lập có thể thêm vào hoặc thay đổi mà không cần sửa đổi microkernel.

#### Nguyên tắc hoạt động của mô hình Microkernel

**Microkernel:**

Chứa các thành phần cốt lõi cần thiết để hệ thống hoạt động, như quản lý bộ nhớ, xử lý luồng, giao tiếp giữa các thành phần, và cơ chế liên lạc liên quy trình (IPC).

**Các dịch vụ người dùng (User Services):**

Các dịch vụ bổ sung như hệ thống tập tin, giao diện người dùng, và các dịch vụ ứng dụng được triển khai dưới dạng các module riêng biệt, hoạt động bên ngoài microkernel. Các dịch vụ này giao tiếp với microkernel thông qua IPC.

**Module:**

Các thành phần hoặc dịch vụ bổ sung có thể được thêm vào hoặc loại bỏ một cách linh hoạt. Các module này có thể được phát triển và triển khai độc lập với microkernel.

**Giao tiếp thông qua IPC:**

Các module giao tiếp với microkernel và với nhau thông qua các cơ chế IPC. Điều này cho phép các thành phần hệ thống hoạt động độc lập và giảm sự phụ thuộc giữa chúng.

#### Ưu điểm của mô hình Microkernel

**Tính mô-đun cao:**

Microkernel giúp tách biệt các thành phần hệ thống thành các module riêng biệt, dễ dàng phát triển, bảo trì và nâng cấp mà không ảnh hưởng đến toàn bộ hệ thống.

**Tính linh hoạt và khả năng mở rộng:**

Các module mới có thể được thêm vào hoặc loại bỏ mà không cần thay đổi microkernel, giúp hệ thống dễ dàng mở rộng và thích ứng với các yêu cầu mới.

**Tăng cường bảo mật và độ tin cậy:**

Việc tách biệt các dịch vụ vào các module riêng biệt giúp giảm thiểu rủi ro khi một phần của hệ thống bị lỗi hoặc bị tấn công. Các lỗi trong một module không ảnh hưởng đến các module khác hoặc microkernel.

**Dễ dàng di động:**

Microkernel có thể được triển khai trên nhiều nền tảng phần cứng và hệ điều hành khác nhau mà không cần thay đổi lớn.

#### Nhược điểm của mô hình Microkernel

**Hiệu suất:**

Việc giao tiếp giữa các module và microkernel thông qua IPC có thể tạo ra độ trễ và ảnh hưởng đến hiệu suất tổng thể của hệ thống. IPC thường chậm hơn so với các cuộc gọi hệ thống trực tiếp.

**Phức tạp trong thiết kế và triển khai:**

Thiết kế và triển khai một microkernel và các module có thể phức tạp và đòi hỏi kiến thức chuyên sâu về hệ thống.

**Tốn kém tài nguyên:**

Các cơ chế IPC và việc quản lý các module riêng biệt có thể tiêu tốn nhiều tài nguyên hệ thống hơn so với các kiến trúc khác.

### Kiến trúc hướng sự kiện

#### Mô hình hướng sự kiện (Event-Driven Architecture) là gì?

Mô hình hướng sự kiện (Event-Driven Architecture, EDA) là một kiểu kiến trúc phần mềm trong đó các thành phần của hệ thống giao tiếp và tương tác với nhau thông qua việc phát sinh và xử lý các sự kiện (events). Một sự kiện có thể là bất kỳ thay đổi trạng thái hoặc một hành động xảy ra trong hệ thống, và nó sẽ kích hoạt các phản hồi (handlers) để thực hiện một số tác vụ cụ thể.

#### Nguyên tắc hoạt động

**Sự kiện (Events):**

Sự kiện là thông tin về một hành động hoặc thay đổi trạng thái trong hệ thống. Mỗi sự kiện thường chứa dữ liệu cần thiết để mô tả chi tiết về sự kiện đó.

**Phát sinh sự kiện (Event Emitters):**

Các thành phần trong hệ thống (event emitters) phát sinh sự kiện khi có một hành động hoặc thay đổi trạng thái xảy ra. Các sự kiện này được gửi đến một hệ thống xử lý sự kiện trung tâm hoặc trực tiếp đến các bộ xử lý sự kiện (event handlers).

**Xử lý sự kiện (Event Handlers):**

Các thành phần chịu trách nhiệm xử lý các sự kiện nhận được. Mỗi bộ xử lý sự kiện có thể thực hiện một hoặc nhiều tác vụ cụ thể dựa trên thông tin sự kiện.

Bus hoặc Broker sự kiện (Event Bus/Broker):

Là một cơ chế trung gian giúp chuyển tiếp các sự kiện từ các emitters đến các handlers. Event bus hoặc broker có thể thực hiện các nhiệm vụ như lưu trữ, định tuyến, và đảm bảo tính nhất quán của sự kiện.

**Mô hình Pub/Sub (Publish/Subscribe):**

Trong mô hình này, các emitters "xuất bản" sự kiện đến một chủ đề (topic), và các handlers "đăng ký" với các chủ đề mà họ quan tâm. Khi một sự kiện được xuất bản, tất cả các handlers đã đăng ký với chủ đề tương ứng sẽ nhận và xử lý sự kiện đó.

#### Ưu điểm của mô hình hướng sự kiện

**Khả năng mở rộng cao:**

Hệ thống có thể dễ dàng mở rộng bằng cách thêm các event handlers mới mà không ảnh hưởng đến các thành phần hiện có.

Tăng khả năng xử lý song song và phân tán, giúp hệ thống đáp ứng tốt hơn với tải lớn.

**Tính linh hoạt:**

Các thành phần có thể phát triển và triển khai một cách độc lập, giúp dễ dàng thêm hoặc thay đổi chức năng mà không ảnh hưởng đến toàn bộ hệ thống.

Dễ dàng tích hợp với các hệ thống khác và thêm các tính năng mới dựa trên các sự kiện hiện có.

**Phản ứng nhanh với thay đổi:**

Hệ thống có thể phản ứng ngay lập tức với các sự kiện mới, giúp cải thiện khả năng đáp ứng và thời gian phản hồi.

**Khả năng chịu lỗi cao:**

Sự phân tách giữa các emitters và handlers giúp hệ thống có khả năng chịu lỗi tốt hơn. Nếu một handler bị lỗi, các handlers khác vẫn có thể tiếp tục hoạt động mà không bị ảnh hưởng.

#### Nhược điểm của mô hình hướng sự kiện

**Phức tạp trong quản lý và giám sát:**

Hệ thống hướng sự kiện có thể trở nên rất phức tạp khi số lượng sự kiện và handlers tăng lên. Việc theo dõi và gỡ lỗi trở nên khó khăn hơn.

Cần các công cụ giám sát và quản lý sự kiện hiệu quả để đảm bảo hệ thống hoạt động trơn tru.

**Đảm bảo tính nhất quán:**

Đảm bảo tính nhất quán của dữ liệu trong một hệ thống phân tán có thể là một thách thức lớn. Các sự kiện cần được xử lý theo đúng thứ tự và tránh bị mất.

Các mẫu thiết kế như Event Sourcing và CQRS (Command Query Responsibility Segregation) thường được sử dụng để quản lý tính nhất quán, nhưng cũng làm tăng độ phức tạp của hệ thống.

**Độ trễ trong xử lý sự kiện:**

Mặc dù hệ thống có thể phản ứng nhanh với các sự kiện, nhưng nếu không được thiết kế cẩn thận, độ trễ trong việc chuyển tiếp và xử lý sự kiện có thể ảnh hưởng đến hiệu suất của hệ thống.

**Tốn kém tài nguyên:**

Hệ thống hướng sự kiện thường yêu cầu nhiều tài nguyên để duy trì bus hoặc broker sự kiện và các cơ chế lưu trữ, định tuyến sự kiện.

### Kiến trúc pipeline

#### Mô hình Microkernel Architecture là gì?

Mô hình Pipeline Architecture là một kiểu kiến trúc phần mềm trong đó quá trình xử lý dữ liệu hoặc yêu cầu được chia thành các bước đơn giản và tuần tự, mỗi bước thực hiện một phần nhỏ của công việc và chuyển kết quả đến bước tiếp theo. Các bước này thường được tổ chức theo dạng "ống dẫn" (pipeline), trong đó dữ liệu hoặc yêu cầu được đưa qua từng bước một cho đến khi hoàn thành.

#### Nguyên tắc hoạt động của mô hình Pipeline

**Phân chia công việc:**

Quá trình xử lý được chia thành các bước nhỏ, mỗi bước thực hiện một phần cụ thể của công việc.

**Tuần tự:**

Các bước xử lý được thực hiện theo tuần tự, mỗi bước chờ kết quả từ bước trước đó trước khi thực hiện công việc của mình.

**Truyền dẫn dữ liệu:**

Dữ liệu hoặc kết quả từ bước trước được chuyển giao cho bước tiếp theo, thường qua cơ chế nào đó như tham số hàm hoặc cấu trúc dữ liệu chia sẻ.

**Xử lý song song (tùy chọn):**

Một số kiến trúc Pipeline cho phép xử lý song song của các bước, điều này có thể giúp cải thiện hiệu suất của hệ thống.

#### Ưu điểm của mô hình Pipeline

**Dễ hiểu và dễ triển khai:**

Việc chia công việc thành các bước nhỏ và tuần tự giúp kiến trúc này dễ hiểu và dễ triển khai hơn so với các kiến trúc phức tạp hơn.

**Tính linh hoạt:**

Các bước xử lý có thể được thay đổi hoặc thêm mới mà không ảnh hưởng đến các bước khác trong pipeline, giúp dễ dàng mở rộng và thích ứng với các yêu cầu mới.

Dễ dàng quản lý và bảo trì:

Mỗi bước trong pipeline thực hiện một chức năng cụ thể, điều này giúp dễ dàng quản lý, bảo trì và kiểm thử.

**Tăng hiệu suất:**

Pipeline có thể được tối ưu hóa để xử lý song song các bước, giúp cải thiện hiệu suất của hệ thống.

#### Nhược điểm của mô hình Pipeline

**Phụ thuộc vào bước trước:**

Một bước không thể bắt đầu cho đến khi bước trước hoàn thành, điều này có thể dẫn đến độ trễ và giảm hiệu suất nếu một bước mất quá nhiều thời gian để hoàn thành.

**Khó khăn trong xử lý các ngoại lệ:**

Việc xử lý các ngoại lệ trong pipeline có thể phức tạp và đòi hỏi sự quản lý cẩn thận để đảm bảo tính nhất quán của dữ liệu.

**Khó khăn trong xử lý dữ liệu không đồng bộ:**

Nếu có nhu cầu xử lý dữ liệu không đồng bộ hoặc phản hồi từ các bước trong quá trình xử lý, việc quản lý và đồng bộ hóa dữ liệu có thể trở nên phức tạp.

**Khó khăn trong xử lý các tác vụ phức tạp:**

Pipeline thích hợp cho các quy trình xử lý tuyến tính và đơn giản, nhưng có thể trở nên phức tạp khi cần xử lý các tác vụ phức tạp hoặc có mối quan hệ phức tạp giữa các bước xử lý.

### Client server

#### Mô hình Client-Server là gì?

Mô hình Client-Server là một kiểu kiến trúc phần mềm trong đó các chức năng của hệ thống được phân chia giữa các máy khách (clients) và máy chủ (servers). Máy khách là các thiết bị hoặc ứng dụng người dùng cuối, trong khi máy chủ là các hệ thống mạnh mẽ cung cấp dịch vụ hoặc tài nguyên cho các máy khách. Máy khách gửi yêu cầu đến máy chủ và máy chủ phản hồi bằng cách cung cấp dữ liệu hoặc thực hiện các hành động theo yêu cầu.

#### Nguyên tắc hoạt động của mô hình Client-Server

**Phân chia nhiệm vụ:**

Máy khách (Client): Thực hiện các yêu cầu từ người dùng, gửi các yêu cầu này đến máy chủ, và hiển thị kết quả nhận được từ máy chủ. Các máy khách thường bao gồm các giao diện người dùng và xử lý đầu vào từ người dùng.

Máy chủ (Server): Xử lý các yêu cầu từ máy khách, thực hiện các tác vụ cần thiết (như truy vấn cơ sở dữ liệu, xử lý logic nghiệp vụ) và gửi phản hồi lại cho máy khách.

**Giao tiếp thông qua mạng:**

Máy khách và máy chủ giao tiếp với nhau qua một mạng (LAN, WAN, hoặc Internet). Giao tiếp này thường được thực hiện qua các giao thức mạng như HTTP, TCP/IP.

**Yêu cầu và phản hồi:**

Máy khách gửi yêu cầu đến máy chủ thông qua một giao thức xác định. Máy chủ nhận yêu cầu, xử lý nó và trả lại phản hồi cho máy khách.

**Đa dạng máy khách:**

Một máy chủ có thể phục vụ nhiều máy khách cùng lúc, và các máy khách có thể là các thiết bị khác nhau như máy tính cá nhân, điện thoại di động, hoặc các thiết bị IoT.

#### Ưu điểm của mô hình Client-Server

**Quản lý tập trung:**

Máy chủ quản lý tập trung dữ liệu và tài nguyên, giúp dễ dàng bảo trì, cập nhật và bảo mật. Điều này giúp đảm bảo tính toàn vẹn và an ninh của dữ liệu.

**Khả năng mở rộng:**

Hệ thống có thể mở rộng bằng cách thêm nhiều máy chủ để xử lý tải cao hơn hoặc thêm nhiều máy khách để mở rộng dịch vụ.

**Tài nguyên tập trung:**

Máy chủ có thể được thiết kế để sử dụng tài nguyên mạnh mẽ hơn và cung cấp hiệu suất tốt hơn cho các ứng dụng yêu cầu cao về tài nguyên.

**Phân chia công việc:**

Máy khách tập trung vào giao diện người dùng và trải nghiệm người dùng, trong khi máy chủ tập trung vào xử lý logic và quản lý dữ liệu. Điều này giúp tối ưu hóa hiệu suất của cả hai.

#### Nhược điểm của mô hình Client-Server

**Phụ thuộc vào máy chủ:**

Nếu máy chủ gặp sự cố hoặc ngừng hoạt động, các máy khách sẽ không thể thực hiện các tác vụ yêu cầu. Điều này làm cho hệ thống phụ thuộc rất nhiều vào độ tin cậy của máy chủ.

**Chi phí hạ tầng cao:**

Việc thiết lập và duy trì máy chủ mạnh mẽ và đáng tin cậy có thể tốn kém, đặc biệt là khi yêu cầu tính sẵn sàng cao và khả năng xử lý lớn.

**Tắc nghẽn mạng:**

Khi nhiều máy khách gửi yêu cầu đến máy chủ cùng lúc, điều này có thể dẫn đến tắc nghẽn và giảm hiệu suất của hệ thống. Máy chủ cần phải được thiết kế để xử lý tải cao hoặc phải sử dụng các phương pháp cân bằng tải.

**Bảo mật:**

Mặc dù dữ liệu được tập trung và dễ quản lý, nhưng nếu máy chủ bị tấn công, toàn bộ hệ thống có thể bị ảnh hưởng. Do đó, bảo mật máy chủ là một yêu cầu rất quan trọng.

# : CHỌN KIẾN TRÚC PHẦN MỀM VÀ THIẾT KẾ

## Lựa chọn kiến trúc phần mềm

Đối với yêu cầu quản lý lớp học tín chỉ ở trên thì lựa chọn mô hình kiến trúc layer khá phù hợp.

### Lý do lựa chọn

**Tách biệt rõ ràng các chức năng:**

Mô hình Layer giúp tách biệt các chức năng và trách nhiệm của từng phần trong hệ thống. Mỗi layer đảm nhận một vai trò cụ thể, chẳng hạn như giao diện người dùng, logic nghiệp vụ, truy cập dữ liệu, và tích hợp với các hệ thống bên ngoài. Điều này giúp tăng tính modular và dễ bảo trì cho hệ thống.

**Dễ dàng bảo trì và mở rộng:**

Khi có thay đổi hoặc cần thêm chức năng mới, chỉ cần cập nhật trong layer tương ứng mà không ảnh hưởng đến các layer khác. Ví dụ, nếu cần thay đổi giao diện người dùng, chỉ cần chỉnh sửa tầng Presentation mà không ảnh hưởng đến tầng Business Logic hay Data Access.

**Tăng tính linh hoạt:**

Các tầng được thiết kế để độc lập với nhau, cho phép sử dụng lại các thành phần ở các dự án khác hoặc thay thế chúng mà không cần phải thay đổi toàn bộ hệ thống. Ví dụ, có thể thay thế hệ thống gửi email mà không cần thay đổi tầng Business Logic.

**Quản lý tốt hơn các phụ thuộc:**

Mô hình Layer giúp quản lý các phụ thuộc giữa các thành phần của hệ thống một cách hiệu quả. Các thành phần ở tầng cao hơn chỉ phụ thuộc vào các thành phần ở tầng thấp hơn, giúp giảm thiểu sự phức tạp trong quản lý phụ thuộc.

**Tăng tính bảo mật:**

Bằng cách tách biệt các tầng, hệ thống có thể áp dụng các biện pháp bảo mật khác nhau cho từng tầng. Ví dụ, tầng Data Access có thể có các biện pháp bảo mật để đảm bảo dữ liệu không bị truy cập trái phép, trong khi tầng Presentation có thể áp dụng các biện pháp để ngăn chặn các cuộc tấn công từ phía người dùng.

**Đơn giản hóa việc kiểm thử:**

Từng tầng có thể được kiểm thử một cách độc lập, giúp phát hiện lỗi sớm và dễ dàng hơn. Unit test có thể được thực hiện cho tầng Business Logic mà không cần phụ thuộc vào tầng Presentation hoặc Data Access.

**Tăng hiệu suất làm việc nhóm:**

Khi phát triển một hệ thống lớn, mô hình Layer cho phép phân chia công việc một cách hiệu quả giữa các nhóm phát triển. Mỗi nhóm có thể tập trung vào một tầng cụ thể mà không cần lo lắng về các tầng khác, từ đó tăng năng suất làm việc và giảm xung đột trong quá trình phát triển.

### Trade off

1. Hiệu suất

Overhead từ việc tách lớp: Việc tách biệt các tầng có thể dẫn đến việc gọi hàm qua lại nhiều lần giữa các tầng, tạo ra overhead và có thể làm giảm hiệu suất của hệ thống. Điều này đặc biệt đáng chú ý trong các ứng dụng yêu cầu hiệu suất cao hoặc thời gian phản hồi nhanh.

Latency tăng lên: Khi dữ liệu phải di chuyển qua nhiều tầng, độ trễ trong quá trình xử lý và phản hồi yêu cầu của người dùng có thể tăng lên.

2. Phức tạp trong thiết kế và triển khai

Tăng độ phức tạp ban đầu: Thiết kế và triển khai một hệ thống theo kiến trúc Layer có thể phức tạp hơn so với các mô hình đơn giản khác. Đòi hỏi phải có sự hiểu biết tốt về các nguyên tắc kiến trúc và quản lý tầng.

Tăng cường phối hợp nhóm: Việc phân chia hệ thống thành nhiều tầng đòi hỏi sự phối hợp chặt chẽ giữa các nhóm phát triển, đặc biệt trong các dự án lớn.

3. Cứng nhắc

Giới hạn tính linh hoạt trong một số trường hợp: Mô hình Layer có thể trở nên cứng nhắc khi cần thay đổi hoặc mở rộng một số chức năng mà không ảnh hưởng đến các tầng khác. Điều này có thể làm cho việc điều chỉnh hệ thống trở nên khó khăn và tốn thời gian.

Quy tắc phụ thuộc nghiêm ngặt: Các tầng phải tuân thủ quy tắc phụ thuộc nghiêm ngặt, tầng trên không được phép gọi trực tiếp tầng dưới. Điều này có thể làm tăng sự phức tạp khi có yêu cầu đặc biệt hoặc khẩn cấp.

4. Khó kiểm thử tích hợp

Kiểm thử tích hợp phức tạp: Mặc dù các tầng có thể được kiểm thử độc lập, nhưng kiểm thử tích hợp giữa các tầng có thể trở nên phức tạp và tốn nhiều công sức. Việc đảm bảo các tầng hoạt động chính xác và tương tác tốt với nhau có thể đòi hỏi nhiều thời gian và tài nguyên.

5. Duy trì và cập nhật

Khó khăn khi duy trì tính nhất quán: Khi hệ thống phát triển, việc duy trì tính nhất quán giữa các tầng có thể trở nên khó khăn. Các thay đổi nhỏ ở tầng dưới có thể yêu cầu cập nhật và điều chỉnh ở các tầng trên.

Phụ thuộc vào kiến trúc sư phần mềm: Thiết kế và bảo trì kiến trúc Layer yêu cầu các kiến trúc sư phần mềm có kinh nghiệm và hiểu biết sâu về kiến trúc phần mềm, điều này có thể không phải lúc nào cũng có sẵn.

## Kiến trúc layer

# : HIỆN THỰC MÔ HÌNH HÓA CHO HỆ THỐNG

## Cấu hình phần cứng, phần mềm

Ghi rõ thông tin sử dụng các dịch vụ nào của AWS để thực hiện đồ án.

Cấu hình phần cứng, phần mềm của Server (nếu có).

Cấu hình phần cứng, phần mềm của Client trong hệ thống (nếu có).

## Giao diện của hệ thống

Màn hình chức năng đã thực hiện.

## Kế hoạch và hiện thực kiểm thử hệ thống

### Kế hoạch kiểm thử

### Kiểm thử hệ thống

# : KẾT LUẬN

## Kết quả đạt được

## Hạn chế của đồ án

## Hướng phát triển

# TÀI LIỆU THAM KHẢO

Các tài liệu Tiếng Anh

1. Kevin Faaborg & Sandro Pasquali - Mastering Node.js Second Edition – December 2017
2. David Herron, Diogo Resende & Valentin Bojinov - Node.js Complete Reference Guide – December 2018

Các tài liệu từ Internet

1. <https://nodejs.org/en/docs/>
2. <https://expressjs.com/en/guide/routing.html>