**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM**

A logo with a red and blue letter and a drop of water

Description automatically generated

HỆ THỐNG QUẢN LÍ ĐĂNG KÍ

HỌC PHẦN

**Giảng Viên: Võ Văn hải**

***Nhóm N- Sinh viên thực hiện***

1. Lê Nguyên Sinh - 20124551
2. Trần Đình Chương - 20002835

# MỤC LỤC

[MỤC LỤC 2](#_Toc166969809)

[CHƯƠNG 1 : GIỚI THIỆU 4](#_Toc166969810)

[1.1 Tổng quan 4](#_Toc166969811)

[1.2 Mục tiêu đề tài 5](#_Toc166969812)

[1.3 Phạm vi đề tài 5](#_Toc166969813)

[1.4 Mô tả yêu cầu chức năng 6](#_Toc166969814)

[CHƯƠNG 2 : CƠ SỞ LÝ THUYẾT](#_Toc166969815) 6

[2.1 Phân tích lựa chọn kiến trúc 6](#_Toc166969816)

[2.1.1 Chọn kiến trúc trúc nào? 6](#_Toc166969817)

[2.1.2 Lý luận lý do chọn 7](#_Toc166969818)

[2.2 Thuận lợi và khó khăn 8](#_Toc166969819)

[2.3 Các công nghệ sử dụng 10](#_Toc166969820)

[2.3.1 Java 11](#_Toc166969821)

[2.3.2 JWT (JSON Web Token) 11](#_Toc166969822)

[2.3.3 Spring boot 11](#_Toc166969823)

[2.3.4 Postman 11](#_Toc166969824)

[2.3.5 Apache Kafka 11](#_Toc166969825)

[2.3.6 Docker 11](#_Toc166969826)

[2.3.7 MariaDB 11](#_Toc166969828)

[2.3.8 Github 11](#_Toc166969829)

[CHƯƠNG 3 : PHÂN TÍCH VÀ THIẾT KẾ 12](#_Toc166969830)

[3.1 Sơ đồ Usecase tổng quát 13](#_Toc166969831)

[3.2 Sơ đồ lớp - Class diagram 13](#_Toc166969836)

[3.3 Áp dụng kiến trúc được chọn/ thiết kết 15](#_Toc166969845)

[CHƯƠNG 4 : HIỆN THỰC 16](#_Toc166969846)

[4.1 Cấu hình phần cứng, phần mềm 16](#_Toc166969847)

[4.2 Giao diện của hệ thống 17](#_Toc166969848)

[4.3 Kế hoạch và hiện thực kiểm thử hệ thống 18](#_Toc166969849)

[CHƯƠNG 5 : KẾT LUẬN 19](#_Toc166969850)

[5.1 Kết quả đạt được 19](#_Toc166969851)

[5.2 Những phần chưa đạt được. 19](#_Toc166969852)

[5.3 Hướng phát triển 20](#_Toc166969853)

[TÀI LIỆU THAM KHẢO 20](#_Toc166969854)

# Chương 1: GIỚI THIỆU

* 1. **Tổng quan**
* Trong bối cảnh giáo dục hiện đại, việc quản lý lớp học theo hệ thống tín chỉ đã trở thành một yêu cầu tất yếu của các trường đại học. Hệ thống quản lý lớp học tín chỉ giúp cải thiện quy trình đăng ký học phần, đảm bảo tính công bằng và minh bạch trong việc phân chia tài nguyên giảng dạy và theo dõi tiến độ học tập của sinh viên. Đề tài này sẽ tập trung vào việc xây dựng một hệ thống quản lý lớp học tín chỉ, đáp ứng các nhu cầu của trường đại học trong việc quản lý học phần, lớp học, và thông tin học tập của sinh viên.
  1. **Mục tiêu đề tài**
* Mục tiêu chính của đề tài là phát triển một hệ thống quản lý lớp học tín chỉ, giúp tự động hóa quy trình đăng ký học phần và quản lý học tập của sinh viên. Cụ thể, đề tài hướng đến:
* Xây dựng một hệ thống trực tuyến cho phép sinh viên đăng ký học phần dễ dàng và nhanh chóng.
* Đảm bảo tính chính xác và hiệu quả trong việc quản lý số lượng sinh viên trong mỗi lớp học.
* Cung cấp các công cụ để theo dõi tiến độ học tập của sinh viên, bao gồm số tín chỉ đã đạt, điểm môn học, và điểm trung bình tích lũy.
* Hỗ trợ quy trình xét tốt nghiệp và quản lý thông tin cựu sinh viên.
  1. **Phạm vi đề tài**
* Phạm vi của đề tài bao gồm:
  + Phát triển hệ thống đăng ký học phần trực tuyến cho sinh viên theo cơ chế tín chỉ.
  + Quản lý danh sách môn học, lớp học và thông tin giảng viên.
  + Xử lý các yêu cầu đăng ký môn học của sinh viên, bao gồm kiểm tra các điều kiện tiên quyết và giới hạn số tín chỉ.
  + Quản lý thông tin học tập của sinh viên, bao gồm số tín chỉ đã đạt, các môn đã học, và điểm số.
  + Hỗ trợ sinh viên trong quá trình xét tốt nghiệp và quản lý thông tin cựu sinh viên.
  + Gửi thông báo qua email về kết quả đăng ký và yêu cầu đóng học phí.
  1. **Mô tả yêu cầu chức năng**
* Hệ thống quản lý lớp học tín chỉ cần đảm bảo các yêu cầu chức năng chính sau:
* Services Đăng Ký Tài Khoản và Đăng Nhập:
  + Cho phép sinh viên tạo tài khoản mới và đăng nhập vào hệ thống.
  + Bảo mật thông tin tài khoản và hỗ trợ chức năng khôi phục mật khẩu.
* Services Quản Lý Sinh Viên:
  + Quản lý thông tin cá nhân và học tập của sinh viên.
  + Cập nhật hồ sơ sinh viên, bao gồm ngành học, khóa học, và các thông tin liên quan khác.
* Services Đăng Ký Môn Học:
  + Hiển thị danh sách các môn học khả dụng trong mỗi học kỳ.
  + Kiểm tra điều kiện tiên quyết trước khi cho phép đăng ký môn học.
  + Giới hạn số tín chỉ đăng ký tối đa cho mỗi sinh viên là 30 tín chỉ mỗi học kỳ.
  + Quản lý danh sách sinh viên đăng ký mỗi lớp và danh sách dự bị khi lớp đã đầy.
  + Yêu cầu sinh viên xác nhận trước khi hoàn tất đăng ký.
* Services Quản Lý Lớp Học:
  + Quản lý thông tin về các lớp học, bao gồm số lượng sinh viên, giảng viên, và thời gian học.
  + Điều chỉnh số lượng lớp học mở ra dựa trên danh sách dự bị và nhu cầu thực tế.
* Services Xác Nhận và Thanh Toán:
  + Gửi email xác nhận việc đăng ký thành công và thông báo quyết định đóng học phí cho sinh viên.
  + Hỗ trợ các phương thức thanh toán học phí trực tuyến.
* Services Gửi Thông Báo:
  + Gửi thông báo quan trọng đến sinh viên qua email hoặc hệ thống thông báo nội bộ.
  + Cập nhật các thông tin về thời khóa biểu, lịch thi, và các sự kiện quan trọng khác.
* Services Khảo sát:
  + Quản lý và phân phối khảo sát: Cho phép tạo, quản lý khảo sát và gửi thông báo mời sinh viên tham gia qua email hoặc hệ thống nội bộ.
  + Thu thập và phân tích dữ liệu: Thu thập phản hồi từ sinh viên, cung cấp công cụ phân tích dữ liệu và đảm bảo tính ẩn danh, bảo mật thông tin.
* Services Tốt Nghiệp:
  + Kiểm tra số tín chỉ đã đạt và các điều kiện tốt nghiệp khác.
  + Cho phép sinh viên đăng ký xét tốt nghiệp khi đủ điều kiện.
  + Quản lý và công khai thông tin bằng cấp trên trang web của nhà trường.
  + Lưu trữ thông tin của cựu sinh viên để theo dõi quá trình làm việc, thực hiện các khảo sát và các hoạt động khác, nếu sinh viên đồng ý.

# Chương 2: CƠ SỞ LÝ THUYẾT

* 1. **Phân tích lựa chọn kiến trúc**
     1. Chọn / thiết kế một mẫu kiến trúc theo yêu cầu đề bài.
* Chọn / thiết kế theo mô hình kiến trúc **Microservices**.
  + 1. Lý luận lý do chọn.
* Chọn / thiết kế theo mô hình kiến trúc **Microservices** vì:
* Trước tiên: Vì sao chọn **Microservices** thay vì mô hình **Monolithic**. Ở mô hình **Monolithic** thì các chức năng được đóng gói thành một khối duy nhất trước khi được triển khai lên server. Nó bộc lộ một vài khuyết điểm như:
  + Cồng kềnh:
    - Hệ thông trên thì không phải do một người làm, mà là cả một team làm. Mỗi người sẽ đảm nhận một vài chức năng nhất định, nên mỗi người sau khi phát triển xong chức năng của mình thì không chỉ build và test code của mình mà còn phải build và test code của cả chương trình => Khá là cồng kềnh, mất thời gian.
    - Khi phát triển chức năng của mình thì cũng phải cần thận để tránh làm ảnh hưởng đến chức năng của người khác.
  + Tốc độ phát triển càng ngày càng chậm:
    - Nếu hệ thồng nhỏ chỉ có 3 đến 4 chức năng thì việc thêm mới chức năng vẫn dễ dàng. Tuy nhiên ở bài toán trên có khá nhiều chức năng (Như: Quản Lý Sinh Viên, Đăng Ký Môn Học, Quản Lý Lớp Học, Xác Nhận và Thanh Toán,...). Các chức năng đó đang chạy và rằng buộc chặt chẽ với nhau khi đó nếu thêm một chức năng mới chúng ta phải xem xét sự rằng buộc của nó đến từng các chức năng => Rất mất thời gian và độ nguy hiểm đến hệ thống rất cao.
  + Vì các chức năng bị đóng gói thành một khối nên sẽ cứng nhắc, thiếu linh hoạt. Nếu nâng cấp tài nguyên thì phải nâng cấp cho toàn bộ hệ thống => gây lãng phí.
  + Độ tin tưởng thấp: Nếu một chức năng nào đó bị chết thì khả năng rất lớn các chức khắc sẽ bị chết theo.
* Còn đối với mô hình kiến trúc **Microservices** thì:
  + Các khối phần mềm sẽ được chia nhỏ thành các service khác nhau, như bài trên sẽ có các services:
    - Services Đăng Ký Tài Khoản và Đăng Nhập.
    - Services Quản Lý Sinh Viên.
    - Services Đăng Ký Môn Học.
    - Services Quản Lý Lớp Học.
    - Services Xác Nhận và Thanh Toán.
    - Services Gửi Thông Báo.
    - Services Khảo Sát.
    - Services Tốt Nghiệp.
  + Các service trên không bị phụ thuộc vào nhau, nên việc vận hành và phát triển phần mềm của các lập trình viên cũng đơn giản hơn.
  + Các service phát triển độc lập, nên mô hình này linh hoạt hơn **Monolithic**, tự do lựa chọn ngôn ngữ cho từng service, và nếu nâng cấp tài nguyên thì chỉ cần nâng cấp cho cái service mà mình cần nâng cấp, không cần nâng cấp cả hệ thống 🡪 không gây lãng phí.
* Về độ tin tưởng cao: Còn đối với **Microservices** vì các service triền khai độc lập và phân tán nên nếu chẳng may có một service bị chết nó sẽ không ảnh hưởng đến các service khác.
  1. Thuận lợi và khó khăn.
* **Thuận lợi:** Nhờ vào việc các service không bị phụ thuộc vào nhau khiến cho việc vận hành và phát triển phần mềm của các lập trình viên cũng đơn giản hơn:
  + Các team có thể tự do phát triển các chức năng của mình mà không sợ bị ảnh hưởng đến chức năng của team khác, không mất thời gian để build hoặc test lại toàn bộ chương trình: Mình làm service **Quản Lý Sinh Viên** thì mình cần build chức năng của mình thôi 🡪 Tăng hiệu quả và thời gian làm việc.
  + Khi một Service mới được thêm vào, thì nó chỉ cần đảm bảo luồng hoạt động đúng với một vài service, chứ không cần cả hệ thống 🡪 test đơn giản và nhanh chóng hơn.
    - Ví dụ: Chúng ta thêm một service **Đánh Giá Giảng Viên** vào hệ thống để cho phép sinh viên đánh giá giảng viên sau mỗi học kỳ. Service này cần đảm bảo luồng hoạt động đúng với một vài service, nhưng không phải toàn bộ hệ thống. Các service đó bao gồm:
      * Services **Quản Lý Sinh Viên**: Khi sinh viên muốn đánh giá giảng viên, hệ thống sẽ kiểm tra thông tin sinh viên để đảm bảo rằng sinh viên đủ điều kiện đánh giá.
      * Services **Đăng Ký Môn Học**: Khi sinh viên hoàn thành một môn học, dịch vụ này sẽ cập nhật trạng thái môn học đã hoàn thành, cho phép sinh viên đánh giá giảng viên.
  + Do các service phát triển độc lập nên ta có thể tự do lựa chọn ngôn ngữ cho từng service 🡪 các lập trình viên có thể dụng các ngôn ngữ yêu thích của mình để làm việc hiệu quả hơn, hệ thống sẽ trở nên linh hoạt và đa dạng hơn, các chức năng có thể được tối ưu nhất có thể(sức mạnh của các framework):
    - Services Đăng Ký Tài Khoản và Đăng Nhập:
      * Ngôn ngữ: Java, Framework: Spring Boot
      * Lý do: Spring Boot có tính năng bảo mật mạnh mẽ và hỗ trợ tốt cho xác thực và quản lý tài khoản người dùng, dễ dàng tích hợp với các dịch vụ khác trong hệ thống.
    - Services Quản Lý Sinh Viên:
      * Ngôn ngữ: Java, Framework: Spring Boot
      * Lý do: Spring Boot có ORM mạnh mẽ giúp quản lý dữ liệu sinh viên phức tạp hiệu quả, đảm bảo hiệu suất cao và khả năng mở rộng tốt.
    - Services Đăng Ký Môn Học:
      * Ngôn ngữ: Node.js, Framework: Express.js
      * Lý do: Node.js với Express.js phù hợp cho xử lý nhiều yêu cầu đồng thời và có khả năng xử lý I/O hiệu quả, lý tưởng cho việc đăng ký môn học.
    - Services Quản Lý Lớp Học:
      * Ngôn ngữ: C#, Framework: .NET Core
      * Lý do: .NET Core mạnh mẽ trong quản lý tài nguyên và tương tác với cơ sở dữ liệu, cung cấp công cụ mạnh mẽ cho phát triển ứng dụng doanh nghiệp.
    - Services Xác Nhận và Thanh Toán:
      * Ngôn ngữ: Java, Framework: Spring Boot
      * Lý do: Spring Boot cung cấp các module bảo mật mạnh mẽ và dễ dàng tích hợp với dịch vụ thanh toán, đảm bảo các giao dịch tài chính được bảo mật chặt chẽ.
    - Services Gửi Thông Báo:
      * Ngôn ngữ: Node.js, Framework: NestJS
      * Lý do: NestJS hỗ trợ tốt cho hệ thống thời gian thực và xử lý nhiều yêu cầu đồng thời hiệu quả, phù hợp cho việc gửi thông báo hàng loạt.
    - Services Khảo sát:
      * Ngôn ngữ: Python, Framework: Flask
      * Lý do: Flask là micro-framework nhẹ và linh hoạt, dễ tùy chỉnh và tích hợp, phù hợp cho việc phát triển các dịch vụ khảo sát nhanh chóng..
    - Services Tốt Nghiệp:
      * Ngôn ngữ: Python, Framework: Django
      * Lý do: Django có ORM mạnh mẽ giúp quản lý quy trình tốt nghiệp phức tạp và phát triển ứng dụng web nhanh chóng với công cụ quản trị mạnh mẽ.
  + Trong hệ thống quản lý lớp học tín chỉ, mỗi service được triển khai lên các môi trường server khác nhau. Nên khi cần nâng cấp tài nguyên thì:
    - Tăng tài nguyên cho các service chịu tải cao: Service **Đăng Ký Môn Học** có thể gặp lượng truy cập cao vào đầu mỗi học kỳ. Chúng ta có thể tăng cường tài nguyên (RAM, CPU) cho service này mà không cần nâng cấp toàn bộ hệ thống.
    - Nâng cấp tài nguyên cho các service cần lưu trữ lớn: Service như **Quản Lý Sinh Viên** và **Quản Lý Lớp Học** đòi hỏi lưu trữ nhiều dữ liệu. Chúng ta chỉ cần nâng cấp dung lượng lưu trữ (ROM) cho các service này thay vì nâng cấp toàn bộ hệ thống, giúp tiết kiệm chi phí và tránh lãng phí tài nguyên.
    - Nâng cấp tài nguyên cho các service cần bảo mật: Service như **Xác Nhận và Thanh Toán** cần có tài nguyên bảo mật cao để xử lý thông tin tài chính của sinh viên. Cần tăng cường bảo mật và cung cấp các tài nguyên phù hợp để đảm bảo an toàn dữ liệu và giao dịch.
  + Về độ tin tưởng: Trong **Microservices** vì các service triền khai độc lập và phân tán nên nếu chẳng may có một service bị chết nó sẽ không ảnh hưởng đến các service khác 🡪 dễ khoanh vùng nguyên nhân và kịp thời khác phục tránh để các lỗi này lan ra khiến cho toàn bộ hệ thống bị chết 🡪 Độ tin tưởng sẽ cao:
    - Cụ thể: Nếu Service **Xác Nhận và Thanh Toán** gặp sự cố, như mất kết nối đến cơ sở dữ liệu thanh toán, thì hệ thống vẫn có thể tiếp tục hoạt động bình thường. Các Service khác như **Quản lý Sinh viên** và **Đăng ký Môn học** không bị ảnh hưởng và vẫn có thể hoạt động độc lập. Sinh viên vẫn có thể đăng ký môn học và quản lý tài khoản của mình mà không gặp phải sự cố từ Service **Xác Nhận và Thanh Toán**. Điều này giúp tăng tính ổn định và tin cậy của hệ thống.
* **Khó khăn(Cái giá phải trả):**
* Phức tạp hóa: Mô hình Microservices đòi hỏi nhiều phần nhỏ hoạt động cùng nhau. Việc quản lý và triển khai nhiều dịch vụ có thể phức tạp hơn so với kiến trúc monolithic.
  + Việc quản lý và triển khai nhiều dịch vụ như Services **Đăng Ký Tài Khoản và Đăng Nhập**, **Quản Lý Sinh Viên**, và **Đăng Ký Môn Học**,... Có thể trở nên phức tạp. Ví dụ, mỗi service này có thể yêu cầu cơ sở dữ liệu riêng, logic xử lý riêng, và cần được triển khai trên các môi trường riêng biệt. Điều này đòi hỏi sự phối hợp chặt chẽ giữa các đội ngũ phát triển và vận hành, cũng như cần có hệ thống giám sát và quản lý để đảm bảo tính nhất quán và ổn định.
* Chi phí cao: khi dùng mô hình này thì bắt buộc ta phải tách service, nên vậy chúng ta phải mua thêm server 🡪 chi phi sẽ tăng lên nhiều hơn khi chúng ta scale up.
  + Mỗi Services như **Đăng Ký Môn Học**, **Quản Lý Sinh Viên**, và **Xác Nhận và Thanh Toán** sẽ cần phải có môi trường server riêng biệt. Điều này đồng nghĩa với việc phải mua thêm server để triển khai các dịch vụ này, dẫn đến tăng chi phí. Đặc biệt, khi hệ thống phải scale up để đáp ứng nhu cầu của hàng ngàn sinh viên, chi phí cơ sở hạ tầng có thể tăng lên khá nhiều.
* Debug khó khăn: Khi hệ thống lớn, việc theo dõi và debug các dịch vụ gọi nhau có thể khá phức tạp(các service gọi nhau loạn xạ).
  + Ví dụ, nếu xảy ra sự cố trong quá trình đăng ký môn học, cần phải xác định rõ ràng là service nào gây ra vấn đề - có thể là Services Đăng Ký Môn Học, Services Quản Lý Lớp Học, hoặc Services Xác Nhận và Thanh Toán - điều này đòi hỏi nhiều thời gian và công sức để phân tích và sửa chữa.
* Code trùng lặp: Mỗi dịch vụ có mã nguồn riêng, dẫn đến khả năng code trùng lặp.
  + Mỗi service có mã nguồn riêng và có thể chứa các chức năng tương tự nhau, dẫn đến việc trùng lặp code. Ví dụ, Services **Quản Lý Sinh Viên** và **Quản Lý Lớp Học** có thể cần truy cập và xử lý thông tin sinh viên, và do đó, có thể có đoạn code tương tự để thực hiện nhiệm vụ này.

Quản lý nhiều service đòi hỏi một đội ngũ lớn với chuyên môn cao và khả năng giải quyết vấn đề nhanh chóng. Sự chia sẻ trách nhiệm và tương tác chặt chẽ giữa các thành viên trong nhóm cũng là yếu tố quan trọng để đảm bảo sự ổn định của hệ thống.

* 1. Các công nghệ sử dụng
     1. Java
* Java là một ngôn ngữ lập trình mạnh mẽ, đa nền tảng và được sử dụng rộng rãi, phù hợp cho việc xây dựng các ứng dụng web phức tạp. Java cung cấp khả năng mở rộng và bảo mật cao, làm cho nó trở thành lựa chọn lý tưởng cho hệ thống quản lý đăng ký học phần.
* Lý do sử dụng: Java cung cấp khả năng mở rộng, bảo mật cao và hỗ trợ xây dựng các ứng dụng quản lý đăng ký học phần.
  + 1. JWT (JSON Web Token)
* JWT là một tiêu chuẩn mở (RFC 7519) cho việc truyền thông tin an toàn giữa hai bên dưới dạng một đối tượng JSON. JWT thường được sử dụng để xác thực và trao đổi thông tin, giúp đảm bảo tính toàn vẹn và bảo mật của dữ liệu.
* Lý do sử dụng: JWT giúp đảm bảo tính toàn vẹn và bảo mật của dữ liệu trong hệ thống.
  + 1. Spring Boot
* Spring Boot là một dự án của Spring Framework, giúp tạo ra các ứng dụng Spring dễ dàng và nhanh chóng. Spring Boot cung cấp khả năng tự động cấu hình và quản lý các phụ thuộc, giúp giảm thiểu công sức cần thiết để thiết lập và cấu hình ứng dụng.
* Lý do sử dụng: Spring Boot tự động cấu hình và quản lý phụ thuộc, giúp giảm thiểu công sức cần thiết cho thiết lập ứng dụng.
  + 1. Postman
* Postman là một công cụ phổ biến để kiểm thử API. Nó cho phép các nhà phát triển kiểm tra các yêu cầu API, xem phản hồi và gỡ lỗi, giúp tăng tốc độ phát triển và đảm bảo chất lượng của API.
* Lý do sử dụng: Postman giúp tăng tốc độ phát triển và đảm bảo chất lượng của API.
  + 1. Apache Kafka
* Apache Kafka là một nền tảng phân tán xử lý luồng dữ liệu, cho phép xây dựng các hệ thống xử lý dữ liệu thời gian thực. Kafka có khả năng xử lý lượng lớn dữ liệu, đảm bảo tính sẵn sàng cao và khả năng mở rộng.
* Lý do sử dụng: Kafka có khả năng xử lý lượng lớn dữ liệu và đảm bảo tính sẵn sàng cao.
  + 1. Docker
* Docker là một nền tảng mở cho việc phát triển, vận chuyển và chạy ứng dụng. Docker sử dụng công nghệ container để giúp ứng dụng chạy một cách nhất quán trên mọi môi trường, từ phát triển đến sản xuất.
* Lý do sử dụng: Docker giúp ứng dụng chạy nhất quán trên mọi môi trường.
  + 1. MariaDB
* MariaDB là một hệ quản trị cơ sở dữ liệu quan hệ, là một nhánh của MySQL. MariaDB được thiết kế để có khả năng tương thích cao với MySQL, đồng thời cung cấp nhiều tính năng mới và cải tiến hiệu suất.
* Lý do sử dụng: MariaDB cung cấp tính năng mới và hiệu suất tốt.
  + 1. Github
* GitHub là một dịch vụ lưu trữ mã nguồn dựa trên web, sử dụng Git làm hệ thống kiểm soát phiên bản. GitHub không chỉ là nơi lưu trữ mã nguồn mà còn là một nền tảng hợp tác, giúp các nhà phát triển cùng nhau làm việc hiệu quả hơn.
* Lý do sử dụng: GitHub giúp hợp tác và quản lý mã nguồn hiệu quả.

# Chương 3: PHÂN TÍCH VÀ THIẾT KẾ

3.1. Sơ đồ Usecase tổng quát.

A diagram of a network

Description automatically generated

3.2. Sơ đồ lớp - Class diagram

A diagram of a computer

Description automatically generated

3.3. Áp dụng kiến trúc được chọn/ thiết kết

A diagram of a company

Description automatically generated

**Cơ Sở lý thuyết**

**JWT(JSON Web Token )** :Khi triển khai API Gateway trong môi trường microservices, một trong những nhiệm vụ quan trọng là bảo đảm rằng các yêu cầu từ client đến các dịch vụ backend được bảo vệ và xác thực. JSON Web Token (JWT) là một phương pháp phổ biến để thực hiện xác thực và ủy quyền. Trong cấu hình này, trước khi các yêu cầu API đến được dịch vụ đích, chúng sẽ phải qua một lớp lọc (filter) để kiểm tra và xác thực JWT.

* **API Gateway**
* API Gateway là một điểm truy cập duy nhất cho các client tương tác với các dịch vụ microservice. Nó đóng vai trò làm cầu nối giữa client và các dịch vụ backend, cung cấp các chức năng như:
* Routing :Định tuyến các yêu cầu từ client đến các dịch vụ thích hợp.
* Load Balancing : Phân phối tải công bằng giữa các dịch vụ.
* Security : Áp dụng các chính sách bảo mật, xác thực và ủy quyền.
* Rate Limiting: Hạn chế số lượng yêu cầu từ client để bảo vệ các dịch vụ backend.
* Caching: Lưu trữ tạm thời các phản hồi để tăng hiệu suất.
* **Discovery Server**
* Discovery Server là một dịch vụ quan trọng trong môi trường microservices, cho phép các dịch vụ tự động phát hiện nhau. Nó duy trì một danh sách các dịch vụ khả dụng và vị trí của chúng.
* Eureka Server từ Netflix là một giải pháp phổ biến được sử dụng cho mục đích này. Nó cung cấp khả năng đăng ký và phát hiện dịch vụ, cho phép các dịch vụ microservice tìm thấy và giao tiếp với nhau một cách linh hoạt.
* **Eureka Server** là một thành phần của Netflix OSS, cung cấp các tính năng chính sau:
* Service Registration: Các dịch vụ microservice đăng ký với Eureka Server khi khởi động và thông báo trạng thái của chúng định kỳ.
* Service Discovery: Các dịch vụ client có thể truy vấn Eureka Server để tìm địa chỉ của các dịch vụ khác để giao tiếp.
* **Private Docker Registry**
* Docker Registry là một kho lưu trữ riêng cho các Docker image, giúp các tổ chức quản lý và phân phối các image Docker một cách an toàn và hiệu quả.
* Docker Hub là một registry công cộng phổ biến, nhưng các tổ chức có thể thiết lập registry riêng của mình bằng cách sử dụng Docker Registry hoặc các dịch vụ như **Harbor**. Điều này giúp bảo mật và kiểm soát các image của riêng tổ chức.
* **Config Server**
* Config Server là một thành phần quan trọng trong Spring Cloud, cho phép quản lý cấu hình của các ứng dụng microservice một cách tập trung.
* Spring Cloud Config Server cung cấp khả năng lưu trữ cấu hình trong các kho lưu trữ như **Git** và cung cấp chúng cho các dịch vụ thông qua một API REST. Điều này giúp dễ dàng quản lý và cập nhật cấu hình mà không cần phải triển khai lại các dịch vụ.
* **Distributed Tracing**
* Distributed Tracing là một kỹ thuật theo dõi và giám sát các yêu cầu qua nhiều dịch vụ trong một hệ thống phân tán.
* Spring Cloud Sleuth là một công cụ trong hệ sinh thái Spring hỗ trợ việc theo dõi này, kết hợp với các hệ thống như **Zipkin** hoặc **Jaeger** để thu thập và hiển thị thông tin về các truy vấn, giúp phát hiện và khắc phục sự cố nhanh chóng.
* **Http clients**
* OpenFeign là một thư viện Java cho phép các dịch vụ microservice giao tiếp với nhau thông qua các API HTTP một cách dễ dàng và rõ ràng. Được tích hợp trong hệ sinh thái Spring Cloud, OpenFeign cung cấp một cách thức khai báo các API REST dưới dạng các interface Java, giúp đơn giản hóa việc gọi các dịch vụ từ xa mà không cần viết nhiều mã lệnh phức tạp.
* **Resilient Microservices**
* Circuit Breaker là một mẫu thiết kế giúp ngăn chặn các yêu cầu đến một dịch vụ bị lỗi hoặc hoạt động không ổn định, nhằm tránh làm quá tải hệ thống và lan truyền lỗi. Khi phát hiện một dịch vụ không phản hồi hoặc gặp lỗi quá nhiều lần liên tiếp, circuit breaker sẽ "ngắt mạch", ngăn chặn các yêu cầu tiếp theo cho đến khi dịch vụ đó phục hồi.
* **Mổ ta sơ lược luồng Hoạt động**

**1. Client gửi yêu cầu:**

- Client gửi yêu cầu đến API Gateway, yêu cầu có thể bao gồm các thông tin như địa chỉ URL, phương thức HTTP, và dữ liệu liên quan.

**2. API Gateway xử lý yêu cầu:**

- API Gateway nhận và xử lý yêu cầu từ client.

- Nếu cần, API Gateway có thể thực hiện các chức năng như định tuyến yêu cầu đến dịch vụ backend phù hợp, phân phối tải, kiểm tra bảo mật, và hạn chế tỷ lệ yêu cầu.

**3. Xác thực và ủy quyền:**

- Trước khi yêu cầu từ client được chuyển đến dịch vụ backend, API Gateway thường sẽ xác thực và ủy quyền bằng cách kiểm tra JWT (JSON Web Token) được gửi kèm theo yêu cầu.

- JWT chứa các thông tin về quyền hạn của người dùng và được ký bởi máy chủ xác thực.

**4. Discovery Server:**

- API Gateway sử dụng Discovery Server để tìm kiếm và phát hiện các dịch vụ backend.

- Discovery Server duy trì một danh sách các dịch vụ khả dụng và vị trí của chúng, giúp API Gateway định tuyến yêu cầu đến các dịch vụ này.

**5. Giao tiếp giữa các dịch vụ:**

- API Gateway sẽ chuyển tiếp yêu cầu từ client đến dịch vụ backend phù hợp thông qua một cơ chế giao tiếp như HTTP.

- Dịch vụ backend có thể sử dụng OpenFeign để giao tiếp với các dịch vụ khác trong hệ thống microservices.

**6. Dịch vụ backend xử lý yêu cầu gửi email:**

Dịch vụ backend, được thiết kế để xử lý các yêu cầu gửi email, sẽ lắng nghe (listen) cho các thông điệp từ topic Kafka "send-email" thông qua Kafka Listener. Khi nhận được thông điệp, nó sẽ trích xuất các thông tin cần thiết từ thông điệp (như địa chỉ email, nội dung email, chủ đề, vv.) và sử dụng các dịch vụ gửi email để gửi email tới người nhận

**7. Resilient Microservices:**

- Trong quá trình giao tiếp giữa các dịch vụ, các mẫu thiết kế như Circuit Breaker có thể được triển khai để đảm bảo tính ổn định của hệ thống.

- Nếu một dịch vụ không phản hồi hoặc gặp lỗi, circuit breaker sẽ ngắt mạch để ngăn chặn các yêu cầu tiếp theo và tránh lan truyền lỗi.

**8. Quản lý cấu hình:**

- Cấu hình cho các dịch vụ có thể được quản lý tập trung thông qua Config Server. Config Server lưu trữ cấu hình trong kho lưu trữ như Git và cung cấp chúng cho các dịch vụ thông qua một API REST.

**8. Ghi nhận và giám sát:**

- Distributed Tracing giúp ghi nhận và giám sát các yêu cầu khi chúng đi qua nhiều dịch vụ trong hệ thống microservices.

- Thông tin về các truy vấn có thể được thu thập và hiển thị thông qua các hệ thống như Zipkin hoặc Jaeger, giúp phát hiện và khắc phục sự cố nhanh chóng.

Tóm lại, các thành phần trong hệ thống microservices cùng hoạt động cùng nhau để cung cấp một kiến trúc linh hoạt, mạnh mẽ và dễ quản lý.

* **Tổng Kết**
* Các thành phần như API Gateway, Discovery Server, Private Docker Registry, Eureka Server, Config Server và Distributed Tracing đóng vai trò quan trọng trong việc xây dựng và quản lý môi trường microservices. Chúng giúp cải thiện khả năng mở rộng, quản lý cấu hình, bảo mật, và giám sát hệ thống, tạo nên một kiến trúc microservices mạnh mẽ và hiệu quả.

Mổ ta sơ lược luồng Hoạt động

# Chương 4: HIỆN THỰC

4.1. Cấu hình phần cứng, phần mềm.

4.2. Giao diện hệ thống

* Màn Hình đăng: Nhà trường sẽ cấp cho sinh viên tài khoản đăng nhập với Mssv và Pass:

A screenshot of a computer

Description automatically generated

* Trang Thông tin Cá nhân

A screenshot of a computer

Description automatically generated

* Trang Ký Lớp Học theo môn học

A screenshot of a computer

Description automatically generated

* Trang Danh sách môn Học theo ngành học:

A screenshot of a computer

Description automatically generated

* Trang Danh sách lớp Học đã đăng ký:

A screenshot of a computer

Description automatically generated

* Nhận thông tin từ kaka và gữi email khi đăng ký khóa học thành công

A screenshot of a computer

Description automatically generated

# Chương 5: KẾT LUẬN

**5.1. Kết quả đạt được**

* Làm giao diện thân thiện & dễ sử dụng.
* Làm các chức năng của hệ thống như:
  + Đăng nhập
  + Xem thông tin cá nhân
  + Hiển thị danh sách môn Học theo ngành học
  + Đăng ký lớp học theo môn học
  + Xem danh sách lớp học đã đăng ký
  + Gửi và nhận thông báo về Email

5.2. Những phần chưa đạt được.

* Chưa làm chức năng xem lịch học, xem điểm.
* Chưa làm được chức năng thanh toán.
* Chưa làm được chức năng khảo sát.
* Chưa làm được chức năng tốt nghiệp.

5.3. Hướng phát triển.

* Hoàn thành các chức năng chưa làm được.
* Tăng cường bảo mật và bảo vệ dữ liệu cho hệ thống.
* Thêm quyền đăng nhập cho phụ huynh.
* Thêm chức năng hỗ trọ tư vấn học tập.
* Thêm chức năng quản lý sự kiện và hoạt động ngoại khóa.
* Làm giao diện đẹp và bắt mắt hơn (thêm các biểu đồ thống kê,...)

# 

# TÀI LIỆU THAM KHẢO