**BỘ CÔNG THƯƠNG**

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP.HCM**



**HỆ THỐNG QUẢN LÝ Lớp HỌC TÍN CHỈ**

***Nhóm 01 - Sinh viên thực hiện***

1. Võ Tấn Đạt –
2. Trần Thanh Đại – 20077931

**Giảng viên hướng dẫn: TS. Võ Văn Hải**

# Phần 1: Mô tả yêu cầu bài toán

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.

# Phần 2: Lựa chọn kiến trúc thực hiện bài toán

**2.1 Chọn kiến trúc**

***Kiến trúc được lựa chọn:*** Microservice

**2.2 Lý do chọn kiến trúc**

Việc lựa chọn kiến trúc microservice có vài lý do nhự sau:

**- Bản thân microservice đã có những lợi điểm khi sử dụng, các ưu điểm nổi bật như sau:**

**Tính mô-đun và linh hoạt:** Với microservices, ta có thể triển khai các dịch vụ riêng lẻ cho từng chức năng (đăng ký học phần, quản lý thông tin sinh viên, xử lý thông báo email, quản lý lớp học, v.v.). Điều này giúp hệ thống dễ dàng mở rộng và bảo trì hơn.

**Khả năng mở rộng:** Mỗi microservice có thể được mở rộng độc lập dựa trên nhu cầu và tải công việc của nó. Ví dụ, dịch vụ xử lý đăng ký học phần có thể cần mở rộng nhiều hơn vào đầu học kỳ khi sinh viên đăng ký học phần nhiều.

**Khả năng chịu lỗi tốt hơn:** Một lỗi trong một microservice sẽ không ảnh hưởng đến toàn bộ hệ thống, giúp đảm bảo sự ổn định và liên tục của dịch vụ.

**- Tương tác rất tốt với MQ apache Kafka, Kafka thường được sử dụng trong mô hình microservice để cung cấp khả năng xử lý sự kiện và truyền thông không đồng bộ giữa các dịch vụ và sử dụng Kafka đối với yêu cầu bài toán đăng ký học phần theo tín chỉ trên là rất hợp lý:**

**Quản lý đăng ký học phần:** Kafka có thể được sử dụng để xử lý các yêu cầu đăng ký học phần một cách bất đồng bộ. Khi sinh viên gửi yêu cầu đăng ký, yêu cầu này có thể được đưa vào một hàng đợi Kafka để xử lý sau đó. Điều này giúp giảm tải cho hệ thống và cải thiện trải nghiệm người dùng.

**Gửi thông báo email:** Khi sinh viên đăng ký thành công, một thông điệp có thể được gửi tới Kafka, sau đó một dịch vụ khác chịu trách nhiệm gửi email sẽ đọc thông điệp này và gửi email xác nhận. Điều này giúp tách biệt logic gửi email khỏi quá trình đăng ký, làm hệ thống linh hoạt và dễ bảo trì hơn.

# Phần 3: Cơ sở lý thuyết

**3.1 Kiến trúc microservice**



Table 1: Mô hình kiến trúc của Microservice

**3.1.1 Nguyên Lý Cơ Bản của Kiến Trúc Microservices**

**Single Responsibility Principle (SRP):** Mỗi dịch vụ trong kiến trúc microservices được thiết kế để thực hiện một nhiệm vụ cụ thể hoặc một phần nhỏ của ứng dụng, giúp dễ dàng quản lý và phát triển.

**Độc lập Triển khai (Independent Deployment):** Mỗi dịch vụ có thể được phát triển, triển khai và mở rộng độc lập mà không ảnh hưởng đến các dịch vụ khác, giúp tăng tính linh hoạt và tốc độ triển khai.

**Giao tiếp qua Giao diện Được Xác Định Rõ Ràng (Defined Interfaces):** Các dịch vụ giao tiếp với nhau thông qua các giao diện công khai, thường là API RESTful hoặc giao thức nhắn tin, giúp tách biệt và bảo vệ tính toàn vẹn của mỗi dịch vụ.

**Decentralized Data Management:** Mỗi dịch vụ có thể quản lý dữ liệu của riêng mình và sử dụng công nghệ lưu trữ phù hợp với yêu cầu cụ thể của nó, giúp tối ưu hóa hiệu suất và độ tin cậy.

**Tự động hóa DevOps:** Các microservices thường được triển khai bằng cách sử dụng các công cụ DevOps để tự động hóa quy trình phát triển, kiểm thử và triển khai, giúp tăng tốc độ phát triển và giảm lỗi.

**3.1.2 Nguyên Tắc Thiết Kế Microservices**

**Domain-Driven Design (DDD):** Sử dụng thiết kế hướng miền để chia nhỏ ứng dụng thành các dịch vụ dựa trên các ngữ cảnh miền (bounded contexts).

**API Gateway:** Sử dụng API Gateway để quản lý các yêu cầu từ phía khách hàng đến các dịch vụ khác nhau, cung cấp một điểm truy cập duy nhất và giúp giảm tải cho các dịch vụ.

**Circuit Breaker Pattern:** Sử dụng mô hình cầu chì để phát hiện và xử lý lỗi trong các dịch vụ, giúp ngăn chặn lỗi lan truyền và tăng độ tin cậy.

**Event Sourcing và CQRS:** Sử dụng event sourcing để ghi lại lịch sử thay đổi của dữ liệu và CQRS để tách biệt các lệnh (command) và truy vấn (query), giúp tối ưu hóa hiệu suất và tính nhất quán.

**3.1.3 Công nghệ và công cụ sửa dụng trong microservice**

**Docker:** Sử dụng Docker để đóng gói và triển khai các dịch vụ microservices một cách nhất quán và di động.

**Kubernetes:** Sử dụng Kubernetes để điều phối và quản lý các container dịch vụ microservices, cung cấp khả năng mở rộng và tự phục hồi.

**Kafka / RabbitMQ:** Sử dụng Kafka hoặc RabbitMQ để quản lý giao tiếp không đồng bộ giữa các dịch vụ, giúp tăng hiệu suất và giảm độ phức tạp của hệ thống.

**Truyền thông điệp giữa các component trong microservice**: TCP, gRPC, NATS, Messenger Broken, Redis

**3.2 Apache kafka**

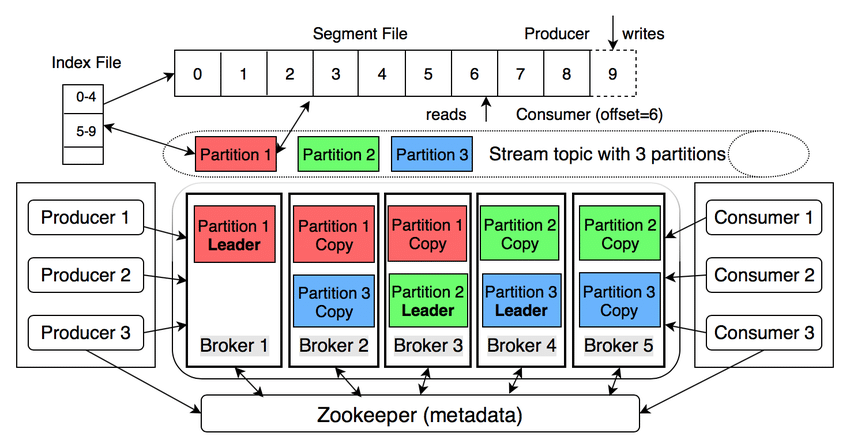


Table 2: Kiến trúc của Apache Kafka

**3.2.1 Kiến Trúc và Thành Phần Chính**

**Producer (Nhà sản xuất):** Là các ứng dụng hoặc dịch vụ gửi thông điệp (messages) đến các chủ đề (topics) trong Kafka. Producers có thể gửi thông điệp đến một hoặc nhiều chủ đề.

**Consumer (Người tiêu thụ):** Là các ứng dụng hoặc dịch vụ đọc và xử lý thông điệp từ các chủ đề trong Kafka. Consumers thường là một phần của nhóm tiêu thụ (consumer group), trong đó mỗi thông điệp trong một chủ đề chỉ được xử lý bởi một consumer duy nhất trong nhóm.

**Broker (Máy chủ môi giới):** Là các máy chủ trong cụm Kafka (Kafka cluster) lưu trữ và quản lý các chủ đề và phân vùng (partitions) của chúng. Mỗi cụm Kafka có thể chứa một hoặc nhiều broker.

**Topic (Chủ đề):** Là các danh mục (categories) trong Kafka mà các thông điệp được gửi tới. Mỗi chủ đề có thể có nhiều phân vùng để hỗ trợ khả năng mở rộng và phân phối tải.

**Partition (Phân vùng):** Là các phần của một chủ đề. Mỗi phân vùng là một log tuần tự, bất biến và chỉ chứa một chuỗi các bản ghi. Các phân vùng cho phép chủ đề mở rộng vượt quá giới hạn của một broker duy nhất và cung cấp song song trong xử lý thông điệp.

**Zookeeper:** Là một dịch vụ quản lý cấu hình và đồng bộ hóa phân tán được sử dụng bởi Kafka để quản lý thông tin metadata về các broker, chủ đề, và phân vùng.

**3.2.2 Nguyên Lý Hoạt Động**

**Log-Based Storage:**Kafka sử dụng mô hình lưu trữ dựa trên log, trong đó mỗi phân vùng của một chủ đề là một log tuần tự của các bản ghi. Điều này cho phép Kafka xử lý các luồng dữ liệu với độ trễ thấp và hiệu suất cao.

**Replication (Sao lưu dữ liệu):** Mỗi phân vùng có thể có nhiều bản sao (replicas) để đảm bảo độ tin cậy và khả năng chịu lỗi. Một trong các bản sao này được chỉ định làm bản sao lãnh đạo (leader) và các bản sao còn lại là các bản sao theo sau (followers).

**Offset (Chỉ số):** Mỗi thông điệp trong một phân vùng được gán một chỉ số (offset) duy nhất và bất biến. Consumers sử dụng chỉ số này để theo dõi vị trí của chúng trong log và quản lý quá trình đọc thông điệp.

**Consumer Groups:** Các consumer có thể được tổ chức thành các nhóm tiêu thụ. Mỗi thông điệp trong một phân vùng được xử lý bởi một consumer duy nhất trong nhóm, cho phép xử lý thông điệp song song và cân bằng tải giữa các consumer.

**At-Least-Once Delivery:** Kafka đảm bảo rằng mỗi thông điệp được giao ít nhất một lần. Điều này có nghĩa là có thể có các thông điệp trùng lặp, và consumers cần phải xử lý điều này nếu cần thiết.

**3.3 gRPC (Remote Procedure Calls)**

**3.3.1 gRPC là gì?**

gRPC là một framework RPC mã nguồn mở, hiện đại và hiệu năng cao mà có thể chạy trên bất kỳ môi trường nào. Framework này được Google khởi công phát triển vào năm 2015, đến 08/2016 thì được phát hành chính thức. Đây được cho là một thế hệ tiếp theo của RPC (Remote Procedure Calls) đặc biệt là trong mô hình Microservices.

**3.3.2 Các tính năng chính của gRPC**

**Hiệu Suất Cao và Tiết Kiệm Băng Thông:**

gRPC sử dụng HTTP/2, cho phép truyền tải dữ liệu song song, nén header và multiplexing streams.

Sử dụng Protocol Buffers (protobuf), một phương thức serial hóa dữ liệu hiệu quả và nhỏ gọn.

**Hỗ Trợ Đa Ngôn Ngữ:**

gRPC hỗ trợ nhiều ngôn ngữ lập trình, bao gồm C++, Java, Python, Go, Ruby, C#, Node.js, PHP và nhiều ngôn ngữ khác. Điều này giúp dễ dàng tích hợp gRPC vào các hệ thống đã có sẵn hoặc các dự án mới.

**Đa Dạng Mô Hình Gọi:**

**Unary RPC**: Client gửi một yêu cầu và nhận một phản hồi.

**Server Streaming RPC**: Client gửi một yêu cầu và nhận nhiều phản hồi từ server.

**Client Streaming RPC**: Client gửi nhiều yêu cầu và nhận một phản hồi từ server.

**Bidirectional Streaming RPC**: Cả client và server gửi nhiều yêu cầu và phản hồi lẫn nhau trong cùng một kết nối.

**IDL (Interface Definition Language):**

gRPC sử dụng protobuf để định nghĩa các dịch vụ và các method của chúng, giúp tự động tạo ra các client và server stub cho các ngôn ngữ khác nhau.

**Tính Bảo Mật Cao:**

Hỗ trợ TLS để mã hóa truyền thông giữa client và server.

Dễ dàng tích hợp với các hệ thống xác thực và ủy quyền hiện có.

**3.3.3 Nhược Điểm của gRPC**

**Phức Tạp Hơn REST:**

gRPC có độ phức tạp cao hơn RESTful API, đặc biệt là khi thiết lập và cấu hình. Việc học cách sử dụng protobuf và gRPC có thể đòi hỏi thời gian và công sức hơn so với REST.

**Hỗ Trợ HTTP/1.1 Hạn Chế:**

gRPC sử dụng HTTP/2, nên không tương thích tốt với các môi trường hoặc hệ thống chỉ hỗ trợ HTTP/1.1, gây khó khăn khi tích hợp vào các hệ thống cũ.

**Hỗ Trợ Trình Duyệt Hạn Chế:**

gRPC không được hỗ trợ trực tiếp bởi trình duyệt do hạn chế của HTTP/2, vì vậy, việc xây dựng ứng dụng web cần thông qua gRPC-Web hoặc các proxy khác.

**Khả Năng Debug và Quan Sát Thấp Hơn:**

Việc debug và quan sát gRPC phức tạp hơn do sử dụng HTTP/2 và protobuf, khó đọc hơn so với JSON trong REST. Các công cụ debug cũng chưa phổ biến và dễ sử dụng như REST.

**Tương Thích Hạn Chế với Một Số Môi Trường:**

Một số hệ sinh thái không có hỗ trợ tốt cho gRPC, đặc biệt là trong môi trường nhẹ nhàng như các thiết bị IoT hoặc các ngôn ngữ lập trình ít phổ biến.

**3.5 Công nghệ sử dụng**

**3.5.1 NestJS**

A red and black logo

Description automatically generated

Nest (NestJS) là một framework được thiết kế để xây dựng các ứng dụng máy chủ Node.js hiệu quả và có khả năng mở rộng. Nó sử dụng JavaScript tiến triển, được xây dựng và hoàn toàn hỗ trợ TypeScript (nhưng vẫn cho phép các nhà phát triển viết mã trong JavaScript thuần) và kết hợp các yếu tố của OOP (Lập trình hướng đối tượng), FP (Lập trình hàm) và FRP (Lập trình hàm phản ứng).

Nest cung cấp một mức độ trừu tượng hóa trên các framework Node.js thông thường này (Express/Fastify), nhưng cũng tiếp tục tiếp cận các API của chúng trực tiếp cho nhà phát triển. Điều này mang lại sự tự do cho nhà phát triển để sử dụng các module của bên thứ ba mà có sẵn cho nền tảng cơ bản.

**Ưu điểm:**

* Khả năng mở rộng và bảo trì: Kiến trúc modul giúp dễ dàng tổ chức và bảo trì mã, cho phép mở rộng ứng dụng một cách dễ dàng.
* Hỗ trợ Microservices: Hỗ trợ tự nhiên cho kiến trúc Microservices, gói tiện ích @nestjs/microservices hữu ích cho giao tiếp dịch vụ.
* TypeScript là trung tâm: TypeScript giúp mã nguồn mạnh mẽ và phát hiện lỗi sớm.
* Hỗ trợ GraphQL: Module @nestjs/graphql giúp tích hợp GraphQL dễ dàng.
* Middleware và Guards: Cho phép can thiệp vào các yêu cầu và áp dụng logic xử lý, hỗ trợ xác thực và phân quyền.
* Cộng đồng năng động và hệ sinh thái phong phú

**Nhược điểm:**

* Khó học đối với những người mới
* Phức tạp cho dự án nhỏ:Cấu trúc modular có thể quá phức tạp cho các ứng dụng đơn giản, tốn thời gian thiết lập và bảo trì.
* Gánh nặng hiệu suất: Có thể nặng hơn các framework khác, gây lo ngại về hiệu suất cho các ứng dụng yêu cầu tài nguyên thấp.
* Hạn chế trong dự án cổ điển: Không tương thích hoàn toàn với JavaScript hoặc các framework cũ, chuyển đổi sang NestJS có thể tốn nhiều công sức.

**3.5.2 AWS Web Service**

A group of logos in a cloud

Description automatically generated

Theo cách đơn giản nhất, điện toán đám mây là việc cung cấp các dịch vụ điện toán như máy chủ, lưu trữ, cơ sở dữ liệu, mạng, phần mềm, phân tích, và trí tuệ qua Internet để cung cấp tài nguyên linh hoạt và kinh tế theo quy mô. Thay vì phải mua và duy trì các thiết bị và phần mềm, bạn chỉ trả tiền cho các dịch vụ đám mây mà bạn sử dụng. Điều này giúp giảm chi phí hoạt động, tăng hiệu quả vận hành cơ sở hạ tầng, và dễ dàng mở rộng quy mô khi nhu cầu kinh doanh thay đổi.

**Ưu điểm:**

* Cung cấp môi trường ảo để sử dụng 1 máy tính ảo cho mục đích chạy các phần mềm mà không cần đến phần cứng
* Có sẵn các cấu hình bảo mật truy cập mạng, port, ssh, tcp,...
* Hỗ trợ nhiều tính năng, monitoring system, load balancer,...

**Nhược điểm:**

* Chi phí khá cao đối với khu vực Việt Nam cho các dự án doanh nghiệp hay cá nhân
* Thường xuyên phải kiểm tra tránh việc phát sinh chi phí không đáng có
* Là một eco-system lớn và phức tạp nên sẽ khó khăn đối với người mới

**3.5.3 ReactJS**

A logo with blue and white text

Description automatically generated

ReactJS, một thư viện JavaScript dựa trên các thành phần, được tạo ra ban đầu bởi Facebook và được phát hành chính thức cách đây đúng mười năm. Thư viện này đơn giản hóa việc tạo ra giao diện người dùng tương tác cho các nhà phát triển, đồng thời quản lý trạng thái của các thành phần một cách hiệu quả. Khả năng của nó để kết hợp nhiều thành phần cho các ứng dụng phức tạp mà không mất trạng thái của chúng trong DOM (Mô hình Đối tượng Tài liệu) của trình duyệt là một ưu điểm đáng kể đối với nhiều nhà phát triển.

**Ưu điểm**:

Sử dụng kiến trúc dựa trên thành phần, giúp tạo ra các thành phần tái sử dụng và dễ bảo trì.

Linh hoạt trong việc xây dựng các ứng dụng tùy chỉnh.

Sẵn sàng thích ứng với các xu hướng công nghệ mới.

Reactjs giúp việc viết các đoạn code js dễ dàng hơn với cú pháp đặc biệt là jsx cho phép tích hợp code html và js trong cùng 1 file

**Nhược điểm:**

* Khái niệm và mẫu phức tạp có thể gây khó khăn cho người mới.
* Phức tạp khi tích hợp với các công nghệ khác.
* Không phải là một framework hoàn chỉnh.
* Kích thước mã lớn và hiệu suất giảm sút trên các thiết bị cũ và mạng yếu.

**3.5.4 Docker**

A diagram of a software application

Description automatically generated with medium confidence

**Docker** là một nền tảng để cung cấp cách để building, deploying và running ứng dụng dễ dàng hơn bằng cách sử dụng các containers (trên nền tảng ảo hóa) cho phép bạn dựng, kiểm thử và triển khai ứng dụng một cách nhanh chóng.

**Ưu điểm:**

* docker hỗ trợ thiết lập nhanh chóng với nhiều lựa chọn điển hình là docker compose
* docker setup có thể quick check vô cùng nhanh. Ví dụ bạn có thể cài đặt redis trên local với thời gian dưới 10s và có thể xoá hoặc tắt nó đi đảm bảo không chạy ngầm
* Hỗ trợ mạng ảo để giao tiếp internal giữa các service
* Hỗ trợ đa kiến trúc, AMD, ARM, x86,...

**Nhược điểm:**

Khó tiếp cận đối với những người lập trình

Quá nhiều sự lựa chọn triển khai cho cùng một mục đích

Phải cần am hiểu nhiều về networking, development application, programing language cho việc config đóng gói phần mềm

**3.5.5 Redis**

A red and grey logo

Description automatically generated

Redis là một cơ sở dữ liệu mã nguồn mở và bộ nhớ cache được sử dụng để lưu trữ dữ liệu dạng key-value trong bộ nhớ trong. Nó cũng hoạt động như một trình thông điệp và hỗ trợ các tính năng như sao chép, kịch bản Lua, việc xóa các phần tử ít được truy cập nhất, giao dịch và bền vững trên đĩa. Redis hỗ trợ nhiều loại cấu trúc dữ liệu trừu tượng và cung cấp tính sẵn có cao thông qua Redis Sentinel và Redis Cluster. Nó có thể chạy trên nhiều nền tảng, bao gồm Linux, Windows và macOS.

**Ưu điểm:**

* Hiệu suất cao: Redis được thiết kế để hoạt động với bộ nhớ trong, giúp cung cấp thời gian đáp ứng nhanh chóng và xử lý dữ liệu nhanh chóng.
* Hỗ trợ nhiều cấu trúc dữ liệu: Redis hỗ trợ nhiều loại cấu trúc dữ liệu trừu tượng như chuỗi, danh sách, bản đồ, tập hợp, và nhiều hơn nữa, giúp phù hợp với nhiều loại ứng dụng khác nhau.
* Hỗ trợ sao chép và bền vững: Redis cung cấp tính năng sao chép dữ liệu và khả năng bền vững trên đĩa, giúp đảm bảo an toàn và sẵn sàng của dữ liệu.
* Linh hoạt và đa nền tảng: Redis có thể chạy trên nhiều nền tảng khác nhau như Linux, Windows và macOS, và có thể tích hợp dễ dàng vào các ứng dụng hiện có.
* Cộng đồng phát triển lớn: Redis có một cộng đồng phát triển mạnh mẽ, cung cấp hỗ trợ và tài liệu phong phú cho người dùng.

**Nhược điểm:**

* Hạn chế bộ nhớ: Do Redis lưu trữ dữ liệu trong bộ nhớ, nên có thể gặp phải hạn chế về dung lượng bộ nhớ khi xử lý các tập dữ liệu lớn.
* Không phù hợp cho các tác vụ CPU-bound: Redis chủ yếu được thiết kế cho các tác vụ I/O-bound và không phù hợp cho các tác vụ yêu cầu xử lý nhiều CPU-bound.
* Cần quản lý tài nguyên: Vì Redis lưu trữ dữ liệu trong bộ nhớ, nên cần phải quản lý tài nguyên để đảm bảo hiệu suất và sự ổn định của hệ thống.

**3.5.6 MongoDB**



MongoDB là một cơ sở dữ liệu không theo mô hình bảng (NoSQL) được thiết kế dưới dạng tài liệu (document-oriented). Thay vì sử dụng cấu trúc dựa trên bảng như các cơ sở dữ liệu quan hệ, MongoDB sử dụng một schema linh hoạt gọi là BSON để lưu trữ dữ liệu dưới dạng tài liệu như JSON. Mỗi bộ sưu tập (collection) trong MongoDB có thể chứa các tài liệu với kích thước và cấu trúc khác nhau. Với dữ liệu được lưu trữ dưới dạng tài liệu JSON, việc truy vấn trở nên nhanh chóng và hiệu quả.

**Ưu điểm:**

* Được phát triển từ nền tảng C++ nên tốc độ đọc và ghi rất nhanh chóng
* Khả năng mở rộng cao
* Dữ liệu được caching (ghi đệm) lên RAM nên việc truy vấn dữ liệu đáp ứng gần như tức thì
* Dữ liệu phi cấu trúc, linh hoạt với các lược đồ(schema) không cần định nghĩa trước hay liên kết các bảng như những hệ QTCSDL khác.

**Nhược điểm:**

* Không đáp ứng được những mô hình giao dịch với lược lưu trữ cao cùng với độ chính xác do không sự ràng buộc dẫn đến sự trùng lặp. Dẫn tới việc không gian bộ nhớ gia tăng không cần thiết
* Xử lý đối với các transaction còn nhiều khó khăn và không ổn định
* Việc lưu trữ dữ liệu vào các ổ cứng ngay lập tức có thể bị mất nếu có các sự cố vật lý.

**3.5.7 MySQL**



MySQL là một hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) mã nguồn mở, được phát triển và hỗ trợ bởi Oracle Corporation. MySQL sử dụng ngôn ngữ SQL (Structured Query Language) để quản lý và truy vấn dữ liệu. Đây là một trong những hệ quản trị cơ sở dữ liệu phổ biến nhất trên thế giới, được sử dụng rộng rãi trong các ứng dụng web, doanh nghiệp, và nhiều loại ứng dụng khác.

**Ưu điểm:**

* Hiệu suất cao: Xử lý truy vấn nhanh chóng, tối ưu cho ứng dụng web.
* Khả năng mở rộng: Hỗ trợ hệ thống lớn với khả năng mở rộng ngang và dọc.
* Bảo mật mạnh mẽ: Cung cấp các tính năng bảo mật như SSL, mã hóa dữ liệu và quyền truy cập chi tiết.
* Hỗ trợ ACID: Đảm bảo tính toàn vẹn của dữ liệu với các giao dịch ACID.
* Cộng đồng lớn: Có nhiều tài liệu và hỗ trợ kỹ thuật từ cộng đồng.

**Nhược điểm**

* Hiệu suất hạn chế trong một số tình huống: Không tối ưu cho các hệ thống có khối lượng dữ liệu rất lớn hoặc yêu cầu xử lý phức tạp.
* Linh hoạt lược đồ kém: Khó thay đổi cấu trúc dữ liệu do yêu cầu lược đồ phải được xác định trước.
* Khả năng mở rộng hạn chế so với NoSQL: Mở rộng ngang (sharding) phức tạp hơn.
* Xử lý JSON và dữ liệu phi cấu trúc hạn chế: Không linh hoạt và mạnh mẽ như các cơ sở dữ liệu NoSQL.

**Phần 4: Cài đặt**

**4.1 Mô hình kiến trúc**

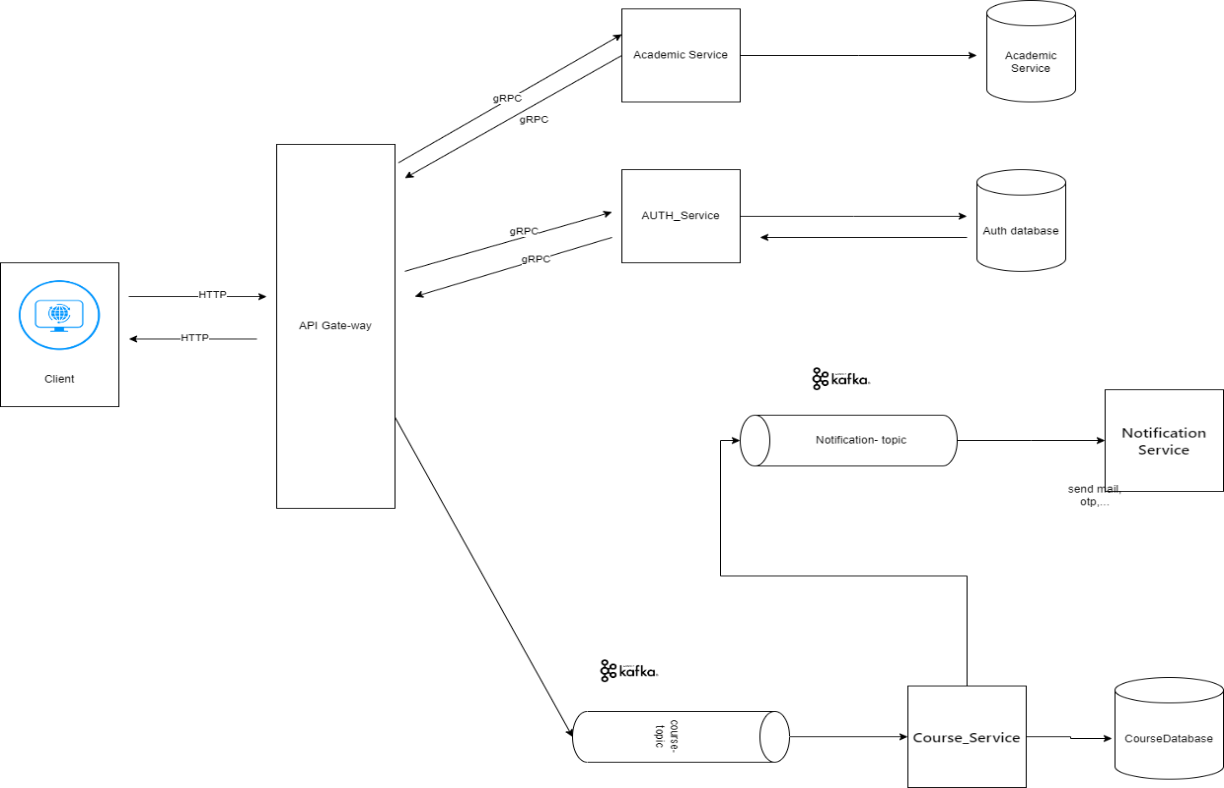


Table 3: Xây dựng kiến trúc của dự án

**Mô tả kiến trúc**

***Chia service theo chắc năng của hệ thống:***

**AuthSerivce:** Đảm nhiệm chức năng xác thực người dùng

**CourseService:** Đảm nhiệm chức năng liên quan đến chương trình khung và đăng kí học phần

**AcademicService:** Đảm nhiệm chức năng liên quan đến lịch học, học phần,…

**NotificationService:** Đảm nhiệm chức năng thông báo: SentOTP, gửi mail,…

**Giải quyết vấn đề sync data:** Dùng kiến trúc EDA theo mô hình pubsub để lắng nghe và cập nhật database

**4.2 Mô hình lớp**

**4.3 Màn hình người dùng và demo một số chức năng của hệ thống**

# TÀI LIỆU THAM KHẢO

Các tài liệu Tiếng Anh

Các tài liệu từ Internet