ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

KIẾN TRÚC PHẦN MỀM

ĐỀ TÀI:
HỆ THỐNG GIA SƯ THÔNG MINH (ITS)

Giảng viên hướng dẫn: Trần Trương Tuấn Phát

Sinh viên thực hiện:

Nguyễn Anh Khoa	2211614
Phạm Quang Minh	2212075
Đoàn Ngọc Hoàng Sơn	2212935
Nguyễn Văn Sơn	2212949
Trần Nam Sơn	2212956
Nguyễn Hiệp Tài	2212985
Huỳnh Cảnh Thịnh	2213272

TP. HỒ CHÍ MINH, THÁNG 12 NĂM 2025

Danh sách thành viên & đóng góp

STT	Họ và Tên	MSSV	Công việc được giao	Mức độ hoàn thành
1	Nguyễn Anh Khoa	2211614	BE	100%
2	Phạm Quang Minh	2212075	Deploy & demo	100%
3	Đoàn Ngọc Hoàng Sơn	2212935	BE, thiết kế kiến trúc	100%
4	Nguyễn Văn Sơn	2212949	FE	100%
5	Trần Nam Sơn	2212956	API Gateway, thiết kế kiến trúc	100%
6	Nguyễn Hiệp Tài	2212985	ERD, use-case, requirement	100%
7	Huỳnh Cảnh Thịnh	2213272	FE	100%

Mục lục

1 Giới thiệu **1**
- 1.1 Đặt vấn đề . . . 1
- 1.2 Mô tả ý tưởng . . . 2

2 Cơ sở lý thuyết và công nghệ **3**
- 2.1 Công nghệ Lõi và Backend . . . 3
- 2.2 Công nghệ Frontend . . . 3
- 2.3 Nền tảng Cloud: Amazon Web Services (AWS) . . . 4
- 2.4 Docker và Amazon Elastic Container Service (ECS) . . . 4
- 2.5 Spring Cloud Gateway . . . 5
- 2.6 Spring Cloud Discovery và Eureka . . . 5
  - 2.6.1 Eureka Server – Service Registry . . . 5
  - 2.6.2 Eureka Client – Service Discovery . . . 6
- 2.7 Giao tiếp giữa các Service: REST API trên mạng nội bộ ECS . . . 6
- 2.8 AWS Amplify . . . 6

3 Yêu cầu về hệ thống **7**
- 3.1 Xác định người dùng . . . 7
- 3.2 Yêu cầu chức năng . . . 7
- 3.3 Yêu cầu phi chức năng . . . 8

4 Đặc tả chi tiết các use-case **10**
- 4.1 Quản lý hệ thống . . . 10
  - 4.1.1 Tổng quan . . . 10
  - 4.1.2 Đặc tả chi tiết . . . 10
- 4.2 Quản lý tài khoản . . . 12
  - 4.2.1 Tổng quan . . . 12
  - 4.2.2 Đặc tả chi tiết . . . 12
- 4.3 Quản lý quyền truy cập . . . 15
  - 4.3.1 Tổng quan . . . 15
  - 4.3.2 Đặc tả chi tiết . . . 15
- 4.4 Quản lý khóa học . . . 18
  - 4.4.1 Tổng quan . . . 18
  - 4.4.2 Đặc tả chi tiết . . . 18
- 4.5 Học khóa học . . . 20
  - 4.5.1 Tổng quan . . . 20
  - 4.5.2 Đặc tả chi tiết . . . 20

5 Phân tích & Thiết kế hệ thống **24**
- 5.1 Entity Relationship Diagram (ERD) . . . 24
  - 5.1.1 Tổng quan . . . 24
  - 5.1.2 Mapping . . . 27
- 5.2 Lựa chọn kiến trúc hệ thống . . . 27
  - 5.2.1 Đặc tính kiến trúc (Architecture characteristics) . . . 27
  - 5.2.2 Đánh giá sự đánh đổi . . . 28
  - 5.2.3 Đề Xuất Các Phong Cách Kiến Trúc . . . 29
    - 5.2.3.a Kiến trúc Modular Monolith . . . 29
    - 5.2.3.b Kiến trúc Microservices . . . 29
    - 5.2.3.c Kiến trúc Service-Based . . . 29
  - 5.2.4 Lựa Chọn Phong Cách Kiến Trúc . . . 30
- 5.3 Thiết kế kiến trúc . . . 31
  - 5.3.1 Module view . . . 31
  - 5.3.2 Component & Connector View . . . 33
  - 5.3.3 Allocation view . . . 35

6 Hiện thực **37**
6.1 API Gateway . . . 37
6.2 Class diagram cho Media Service . . . 38
6.3 Class diagram cho Module quản lý tài khoản . . . 42
6.4 Class diagram cho Module quản lý khóa học . . . 45
6.5 Class diagram cho Module học tập . . . 47
6.6 Class diagram cho Module quiz . . . 49
6.7 Class diagram cho Module AI . . . 51
6.8 Class diagram cho Module Logging . . . 53

7 Tổng kết **56**
7.1 Nhận xét về Thiết kế và Hiện thực . . . 56
7.1.1 Ưu điểm của Thiết kế . . . 56
7.1.2 Hạn chế và Rủi ro . . . 56
7.2 Hướng phát triển và Chiến lược Tối ưu hóa Kiến trúc (SBA) . . . 56

8 Phụ lục **58**
8.1 ADR 00: Lựa chọn kiến trúc ban đầu: Microservices Architecture (MSA) . . . 58
8.2 ADR 01: Chuyển đổi kiến trúc từ Microservices (MSA) sang Service-Based Architecture (SBA) . . . 58

Danh sách hình vẽ

1 Tam giác học tập . . . 1
2 Các công nghệ Back-end . . . 3
3 React . . . 3
4 Amazon Web Services . . . 4
5 Docker và Amazon Elastic Container Service . . . 4
6 AWS Amplify . . . 6
7 Tổng quan các Use-case . . . 10
8 Các use-case về Quản lý hệ thống . . . 10
9 Các use-case về Quản lý tài khoản . . . 12
10 Các use-case về Quản lý quyền chỉnh sửa khóa học . . . 15
11 Các use-case về Quản lý khóa học . . . 18
12 Các use-case về Học khóa học . . . 20
13 Entity Relationship Diagram . . . 24
14 Relational Mapping Diagram . . . 27
15 Module view - Nguyên tắc tổ chức . . . 31
16 Module view - Package diagram . . . 32
17 Component & Connector view . . . 33
18 Component & Connector view . . . 35
19 Class diagram cho Media Service . . . 38
20 Class diagram cho Module quản lý tài khoản . . . 42
21 Class diagram cho Module quản lý khóa học . . . 45
22 Class diagram cho Module học tập . . . 47
23 Class diagram cho Module quiz . . . 49
24 Class diagram cho Module AI . . . 51
25 Class diagram cho Module Logging . . . 53

Danh sách bảng

1 Bảng User Stories 8
2 Danh sách yêu cầu phi chức năng (NFR) 9
3 Usecase «Cấu hình hệ thống» 10
4 Usecase «Xem nhật ký hoạt động AI» 11
5 Usecase «Xem nhật ký kiểm toán (Audit log)» 11
6 Usecase «Tạo hoặc Xóa tài khoản» 12
7 Usecase «Đặt lại mật khẩu (Reset password)» 13
8 Usecase «Đăng ký tài khoản» 13
9 Usecase «Đăng nhập» 14
10 Usecase «Yêu cầu đặt lại mật khẩu» 14
11 Usecase «Cấp và thu hồi quyền Trưởng khoa» 15
12 Usecase «Phê duyệt yêu cầu quyền truy cập» 16
13 Usecase «Thu hồi quyền truy cập» 16
14 Usecase «Yêu cầu quyền truy cập» 17
15 Usecase «Đề xuất chỉnh sửa khóa học» 18
16 Usecase «Tạo khóa học» 18
17 Usecase «Sửa nội dung khóa học» 19
18 Usecase «Ghi danh khóa học» 19
19 Usecase «Xem đề cương khóa học» 20
20 Usecase «Xem tài liệu học tập» 21
21 Usecase «Xem phản hồi thống kê» 21
22 Usecase «Làm bài kiểm tra (quiz)» 22
23 Usecase «Tạo báo cáo thống kê» 22
24 Usecase «Tham gia diễn đàn khóa học» 23

1 Giới thiệu

1.1 Đặt vấn đề

Trong môi trường giáo dục truyền thống, việc giảng dạy chủ yếu diễn ra theo phương pháp đồng bộ, nơi giáo viên truyền đạt kiến thức chung cho cả lớp. Điều này tạo ra một số thách thức lớn:

Giới hạn trong học tập cá nhân hóa

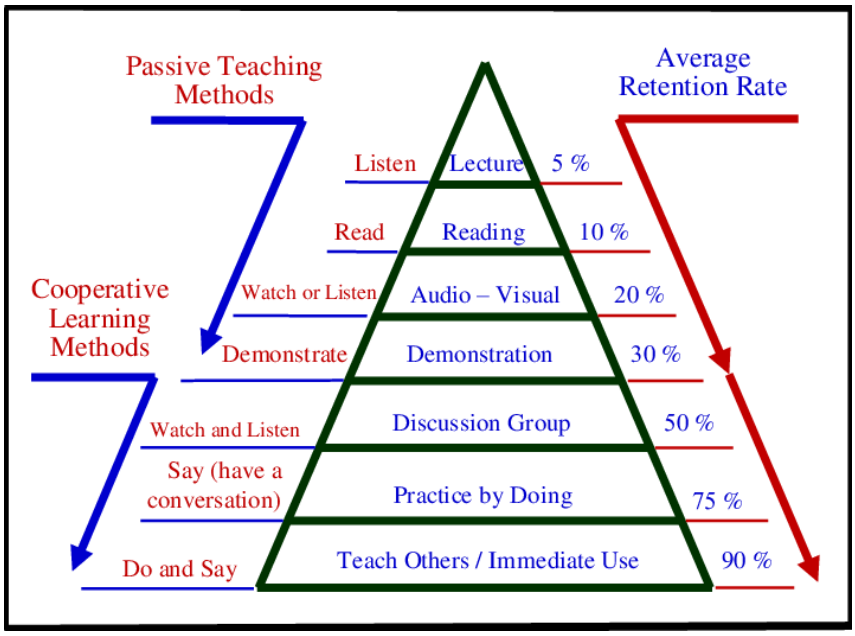
Việc giảng dạy truyền thống khiến giáo viên gặp khó khăn trong việc đáp ứng nhu cầu, tốc độ và phong cách học tập đa dạng của từng học viên.

- Tốc độ và nhu cầu: Mỗi học viên có tốc độ học khác nhau, nhưng thời gian và nguồn lực hạn chế khiến việc cung cấp trải nghiệm học tập cá nhân hóa, phản hồi tức thì và hỗ trợ thích ứng theo thời gian thực trở thành một thách thức lớn.
- Thiếu hiệu quả: Điều này dẫn đến sự thiếu hiệu quả trong việc phát huy tối đa tiềm năng của mỗi người học.

Sự thống trị của phương pháp giảng dạy thụ động

Các phương pháp giảng dạy truyền thống, nơi thông tin chảy một chiều từ người hướng dẫn đến học viên, thường dựa trên các hình thức thụ động.

- Tỷ lệ lưu giữ kiến thức thấp: Theo nguyên lý của Tam giác Học tập (Learning Pyramid)[1], các phương pháp thụ động như Listen hay Read có tỷ lệ lưu giữ trung bình rất thấp (chỉ 5% - 10%).



Hình 1: *Tam giác học tập*

- Thiếu kỹ năng mềm: Các phương pháp này không tạo điều kiện cho học viên tham gia vào các hoạt động học tập chủ động như giải quyết vấn đề, làm việc nhóm, hoặc giao tiếp , vốn là những kỹ năng mềm thiết yếu mà các chương trình kỹ thuật cần phải nâng cao.

Hạn chế trong đánh giá và theo dõi tiến độ học tập

Hệ thống giáo dục truyền thống chủ yếu dựa vào các bài kiểm tra định kỳ như giữa kỳ và cuối kỳ, dẫn đến việc đánh giá không phản ánh đầy đủ quá trình học tập của học viên.

- Thiếu dữ liệu thời gian thực: Giáo viên không thể theo dõi tiến độ của từng học viên theo từng buổi học, dẫn đến khó phát hiện sớm những lỗ hổng kiến thức.
- Quá trình bị gián đoạn: Học viên chỉ nhận được phản hồi sau khi hoàn thành bài kiểm tra, nên khó điều chỉnh chiến lược học tập ngay lập tức.
- Không khuyến khích sự phát triển liên tục: Khi đánh giá chỉ tập trung vào điểm số, việc hình thành năng lực lâu dài và tư duy phản biện bị xem nhẹ.

[1] https://www.educationcorner.com/the-learning-pyramid/

Thiếu đa dạng trong nội dung học tập và phương tiện giảng dạy

Phần lớn nội dung trong mô hình truyền thống được thiết kế theo chuẩn chương trình chung, không phân tầng theo trình độ và không linh hoạt theo nhu cầu cá nhân.

- Ít mức độ truy cập: Học viên không có cơ hội lựa chọn nội dung phù hợp với khả năng—ví dụ: nâng cao, bổ sung cơ bản hoặc nội dung mở rộng.
- Thiếu phương tiện tương tác: Việc phụ thuộc vào sách giáo khoa và bài giảng trên lớp hạn chế việc tiếp cận tài liệu multimedia hoặc công cụ mô phỏng trực quan.
- Cản trở mô hình học tập tự định hướng: Học viên khó tiếp cận tài liệu ở mọi lúc, mọi nơi, dẫn đến sự bị động trong việc xây dựng lộ trình học riêng.

1.2 Mô tả ý tưởng

Để giải quyết những hạn chế của giáo dục truyền thống và tối đa hóa hiệu quả học tập, Hệ thống Gia sư Thông minh (ITS) đã được nghiên cứu và phát triển. ITS là một ứng dụng phần mềm mạnh mẽ sử dụng các kỹ thuật Trí tuệ Nhân tạo (AI) để mô phỏng và cung cấp hướng dẫn cá nhân hóa. ITS tự điều chỉnh linh hoạt theo tốc độ, khả năng, điểm mạnh, điểm yếu và sở thích của từng người học thông qua các tính năng tiên tiến, tập trung vào học tập chủ động:

- Tăng cường thực hành: ITS cung cấp bài tập thực hành thiết kế theo thời gian thực, kèm theo phản hồi thích ứng để đảm bảo học viên liên tục áp dụng kiến thức và phát triển kỹ năng giải quyết vấn đề.
- Hỗ trợ kỹ năng mềm: ITS tích hợp các module hỗ trợ thảo luận nhóm, giúp học viên luyện tập giao tiếp và hợp tác.
- Tự đánh giá và giải thích: Để đạt mức độ lưu giữ kiến thức cao nhất, ITS yêu cầu học viên giải thích các khái niệm thiết kế cho gia sư AI, buộc họ phải trở thành "người dạy", từ đó củng cố khả năng tự điều chỉnh (Self-Regulation) và hiểu sâu.
- Lộ trình học tập tùy chỉnh: AI của hệ thống phân tích chi tiết hoạt động học tập của học viên để tự động điều chỉnh lộ trình, gợi ý các tài liệu (như video, bài đọc) và chủ đề tiếp theo có độ khó phù hợp, thay thế cho chương trình giảng dạy chung cứng nhắc.
- Đánh giá liên tục: Thay vì chỉ dựa vào kiểm tra định kỳ, ITS cung cấp các bài kiểm tra ngắn sau mỗi nội dung của chương. Hệ thống AI có khả năng tự động tạo ra các câu hỏi đánh giá mới, đa dạng, phân tầng theo độ khó và tập trung vào các lỗ hổng kiến thức cụ thể của từng học viên.
- Phát hiện sớm lỗ hổng kiến thức: Mọi tương tác của học viên đều được ghi lại và phân tích chi tiết. Điều này giúp phát hiện sớm các lĩnh vực mà học viên đang gặp khó khăn, cho phép can thiệp và hỗ trợ kịp thời, đảm bảo không bỏ sót bất kỳ lỗ hổng kiến thức nào.

Nhờ những tính năng này, ITS cung cấp phản hồi, gợi ý và lộ trình học tập tùy chỉnh, mang lại hiệu quả tương đương với mô hình gia sư một kèm một, giúp người học phát huy tối đa tiềm năng của mình.

2 Cơ sở lý thuyết và công nghệ

Hệ thống được xây dựng trên nền tảng Java/Spring Boot kết hợp với kiến trúc phân tán Spring Cloud và giao diện người dùng React. Việc lựa chọn bộ công nghệ này nhằm tận dụng khả năng phát triển nhanh, tính ổn định cao và sự hỗ trợ mạnh mẽ cho kiến trúc Service-Based Architecture – SBA.

2.1 Công nghệ Lõi và Backend

Các dịch vụ nghiệp vụ và hỗ trợ (Domain & Infrastructure Services) đều sử dụng hệ sinh thái Spring.

- Spring Boot: Đây là framework phát triển lõi cho toàn bộ các dịch vụ backend, bao gồm Identity Access Management, Quiz, System, Learning, Course, AI, và Service Registry. Spring Boot cung cấp môi trường chạy độc lập và cấu hình tự động (auto-configuration), đảm bảo mỗi module là một đơn vị triển khai rời rạc và dễ dàng quản lý.
- Spring Cloud Gateway: Công nghệ này được sử dụng để triển khai module API Gateway. Nó đóng vai trò là điểm truy cập duy nhất cho client, chịu trách nhiệm chính trong việc định tuyến yêu cầu tới dịch vụ tương ứng và thực hiện lọc yêu cầu (filtering) trước khi đến các dịch vụ nghiệp vụ.
- Spring Cloud Discovery (Eureka Server): Đây là công nghệ nền tảng cho module Service Registry. Nó đóng vai trò là máy chủ đăng ký dịch vụ, lưu trữ metadata và danh sách các instance đang chạy của các dịch vụ.
- Spring Cloud Discovery (Eureka Client): Tất cả các Service nghiệp vụ (IAM, Quiz, System, Learning, Course, AI) và API Gateway đều được cấu hình là Eureka Client. Điều này cho phép chúng tự động đăng ký với Eureka Server và tận dụng cơ chế khám phá dịch vụ (Service Discovery) để xác định instance hợp lệ tại thời điểm chạy (runtime).
- Spring AI: Đây là thư viện được sử dụng trong AI Service. Nó hỗ trợ việc tích hợp các mô hình Trí tuệ Nhân tạo (AI Models) bên ngoài một cách dễ dàng. Spring AI giúp AI Service thực hiện các chức năng như phân tích hành vi và tiến trình học, cũng như đề xuất lộ trình học tập phù hợp dựa trên dữ liệu phân tích.

Hình 2: *Các công nghệ Back-end*

2.2 Công nghệ Frontend

React là một thư viện JavaScript mã nguồn mở do Facebook phát triển, được sử dụng rộng rãi để xây dựng giao diện người dùng (User Interface – UI), đặc biệt là các ứng dụng web có tính tương tác cao. React cho phép nhà phát triển xây dựng giao diện thông qua các thành phần (components) độc lập, có khả năng tái sử dụng, giúp tổ chức mã nguồn rõ ràng và dễ bảo trì hơn.

Một trong những ưu điểm nổi bật của React là sử dụng Virtual DOM – một cấu trúc DOM ảo giúp tối ưu hóa hiệu năng. Khi giao diện thay đổi, React chỉ cập nhật những phần tử cần thiết thay vì render lại toàn bộ trang, nhờ đó cải thiện tốc độ xử lý và trải nghiệm của người dùng.

React hỗ trợ mô hình lập trình declarative (khai báo), cho phép mô tả giao diện theo trạng thái của ứng dụng. Khi trạng thái thay đổi, React tự động cập nhật UI tương ứng, giảm thiểu lỗi và đơn giản hóa việc quản lý vòng đời giao diện.

Ngoài ra, React có một hệ sinh thái phát triển rộng lớn bao gồm:

- React Hooks: Cung cấp cách tiếp cận mới để quản lý state và vòng đời trong function components.
- React Router: Hỗ trợ điều hướng nhiều trang trong ứng dụng single-page.

Hình 3: *React*

Nhờ vào cấu trúc linh hoạt, hiệu năng cao và cộng đồng lớn mạnh, React trở thành một trong những công nghệ phổ biến nhất được lựa chọn trong phát triển các ứng dụng web hiện đại, đặc biệt trong các dự án cần giao diện tương tác, trải nghiệm mượt mà và khả năng mở rộng tốt

2.3 Nền tảng Cloud: Amazon Web Services (AWS)

Amazon Web Services (AWS) là nền tảng điện toán đám mây hàng đầu thế giới, cung cấp hơn 200 dịch vụ toàn diện bao phủ hầu hết mọi nhu cầu về hạ tầng và ứng dụng. Với các trung tâm dữ liệu được phân bố trên toàn cầu, AWS mang đến khả năng mở rộng linh hoạt, độ sẵn sàng cao, bảo mật nhiều lớp, cùng với mô hình chi phí tối ưu theo nhu cầu sử dụng.

Hình 4: *Amazon Web Services*

AWS cung cấp đầy đủ bốn nhóm dịch vụ cốt lõi trong kiến trúc cloud hiện đại:

- Compute: EC2, ECS, Lambda cho phép triển khai ứng dụng dưới nhiều mô hình từ máy ảo, container đến serverless.
- Storage: S3, EBS, EFS hỗ trợ lưu trữ đối tượng, khối và file system với độ bền dữ liệu cao.
- Networking: VPC, Elastic Load Balancing, Route 53 cho phép kiểm soát mạng nội bộ, định tuyến và phân phối tải.
- Database: RDS, DynamoDB, Aurora cung cấp hệ quản trị cơ sở dữ liệu quan hệ và NoSQL hiệu năng cao.

Một điểm mạnh nổi bật của AWS là khả năng tự động mở rộng (auto scaling) và cân bằng tải (load balancing) giúp hệ thống đáp ứng linh hoạt khi lưu lượng tăng đột biến mà không gián đoạn hoạt động. Đồng thời, AWS áp dụng mô hình bảo mật theo nhiều lớp (multi-layer security) gồm mã hoá dữ liệu, kiểm soát truy cập theo IAM, network isolation qua VPC và giám sát liên tục với CloudWatch.

Nhờ các đặc điểm này, việc triển khai hệ thống trên AWS giúp giảm tải phần vận hành hạ tầng, tăng độ tin cậy, đồng thời cho phép nhóm phát triển tập trung vào logic ứng dụng thay vì lo lắng về máy chủ hay cấu hình mạng. Đây là nền tảng phù hợp cho các hệ thống IoT, AI và xử lý dữ liệu theo thời gian thực, đặc biệt khi yêu cầu khả năng mở rộng cao và vận hành ổn định.

2.4 Docker và Amazon Elastic Container Service (ECS)

Để đảm bảo tính nhất quán trong quá trình phát triển và triển khai, toàn bộ các thành phần backend của hệ thống được đóng gói dưới dạng Docker container. Docker cho phép đóng gói mã nguồn kèm theo môi trường chạy (runtime, thư viện, cấu hình) vào một Docker image duy nhất. Nhờ vậy, ứng dụng có thể vận hành giống nhau ở mọi môi trường — từ máy lập trình viên, môi trường staging, cho đến môi trường production. Điều này giúp giảm thiểu xung đột phụ thuộc (dependency conflict), đơn giản hoá quá trình triển khai và tăng tốc độ phát hành phiên bản mới.

Để quản lý và vận hành các container này, hệ thống sử dụng Amazon Elastic Container Service (ECS), một dịch vụ điều phối container (container orchestration) mạnh mẽ của AWS. ECS cho phép triển khai các container lên một ECS Cluster, theo dõi trạng thái của chúng và tự động khởi động lại khi xảy ra lỗi. ECS hỗ trợ hai mô hình chạy: EC2 (tự quản lý EC2 instances) và Fargate (serverless, không cần quản lý máy chủ), giúp linh hoạt tối ưu chi phí và hiệu năng.

Hình 5: *Docker và Amazon Elastic Container Service*

Thông qua ECS, hệ thống được hưởng các cơ chế vận hành tự động như:

- Tự động mở rộng (Auto Scaling): tăng hoặc giảm số lượng container dựa trên tải.
- Health Check và tự phục hồi: container lỗi sẽ được thay thế ngay lập tức để duy trì tính sẵn sàng.
- Load Balancing: tích hợp với Elastic Load Balancer (ELB) để phân phối đều lưu lượng.
- Triển khai phiên bản mới không gián đoạn: thông qua cơ chế rolling update.

Việc áp dụng Docker và ECS giúp hệ thống đạt được khả năng mở rộng linh hoạt, ổn định trong vận hành, dễ dàng bảo trì và phù hợp với kiến trúc microservices. Đồng thời, nền tảng cloud-native của ECS giúp giảm tối đa công sức quản lý hạ tầng, cho phép nhóm phát triển tập trung hoàn toàn vào logic ứng dụng.

2.5 Spring Cloud Gateway

Spring Cloud Gateway đóng vai trò là tầng định tuyến trung tâm (API Gateway) trong kiến trúc của hệ thống. Đây là một giải pháp hiện đại được xây dựng trên nền tảng Spring WebFlux và mô hình lập trình phản ứng (reactive programming), cho phép xử lý lượng lớn yêu cầu với hiệu năng cao và độ trễ thấp. Spring Cloud Gateway chịu trách nhiệm tiếp nhận toàn bộ lưu lượng từ người dùng hoặc từ API Gateway bên ngoài, sau đó định tuyến chúng đến đúng dịch vụ phía backend trong hệ thống.

Về mặt chức năng, Spring Cloud Gateway cung cấp nhiều khả năng phù hợp cho hệ thống:

- **Định tuyến linh hoạt (Dynamic Routing):** cho phép điều hướng yêu cầu tới các service khác nhau dựa trên URL, header, tham số truy vấn hoặc quy tắc tùy chỉnh.
- **Cân bằng tải (Load Balancing):** tích hợp liền mạch với Spring Cloud LoadBalancer hoặc Service Discovery để phân phối yêu cầu đều giữa các bản sao container.
- **Lọc và xử lý trước/sau (Pre/Post Filters):** hỗ trợ chèn logic xử lý như xác thực, ghi log, giới hạn tốc độ (rate limiting), sửa đổi header hoặc payload trước khi chuyển đến service đích.
- **Hỗ trợ bảo mật:** dễ dàng tích hợp với Spring Security để triển khai cơ chế xác thực, phân quyền, JWT, OAuth2.
- **Khả năng mở rộng:** được thiết kế theo kiến trúc phi đồng bộ, phù hợp cho phân tán tải cao trong môi trường container.

Với vai trò cổng vào duy nhất của hệ thống, Spring Cloud Gateway giúp đơn giản hóa giao tiếp, giảm độ phức tạp giữa client và các microservice, đồng thời tăng tính bảo mật bằng cách che giấu cấu trúc nội bộ. Nhờ hoạt động trên nền phản ứng (reactive), gateway đạt hiệu năng tốt ngay cả khi số lượng kết nối và yêu cầu tăng cao.

2.6 Spring Cloud Discovery và Eureka

Trong hệ thống, việc các dịch vụ có thể tìm thấy và giao tiếp với nhau một cách tự động là một yêu cầu quan trọng. Spring Cloud Discovery cùng với Eureka cung cấp một giải pháp hoàn chỉnh cho cơ chế Service Registry và Service Discovery, đảm bảo sự linh hoạt và khả năng mở rộng của hệ thống phân tán.

2.6.1 Eureka Server – Service Registry

Eureka Server hoạt động như một trung tâm đăng ký dịch vụ (Service Registry). Nó duy trì một danh sách động các service instance đang hoạt động trong hệ thống cùng với metadata liên quan (như địa chỉ, port, trạng thái...). Các dịch vụ khi khởi động sẽ gửi yêu cầu đăng ký (registration) đến Eureka Server và định kỳ gửi heartbeat để thông báo rằng chúng vẫn còn hoạt động.

Nhờ đó, hệ thống có thể:

- **Xóa bỏ cấu hình tĩnh**: Không cần khai báo cứng địa chỉ IP hay endpoint của từng service.
- **Hỗ trợ mở rộng tự động**: Khi có service instance mới xuất hiện hoặc bị hỏng, registry sẽ phản ánh ngay lập tức.
- **Tăng tính linh hoạt**: Cho phép dịch chuyển container, scaling theo tải mà không ảnh hưởng tới cấu trúc hệ thống.

2.6.2 Eureka Client – Service Discovery

Các dịch vụ trong hệ thống đóng vai trò là Eureka Client, nghĩa là chúng sẽ:

- **Tự động đăng ký** với Eureka Server khi khởi động.
- **Lấy danh sách các service khác** từ registry để biết cách giao tiếp mà không cần cấu hình thêm.
- **Cập nhật danh sách instance theo thời gian thực**, bảo đảm rằng mỗi yêu cầu được định tuyến đến service đang khả dụng.

Cơ chế Service Discovery giúp các dịch vụ giao tiếp theo kiểu "tên dịch vụ" thay vì địa chỉ cố định, tạo ra:

- **Khả năng chịu lỗi cao**: Khi một instance ngừng hoạt động, client sẽ tự động chuyển sang instance khác.
- **Tự động cân bằng tải**: Kết hợp với các công cụ như Spring Cloud LoadBalancer hoặc Gateway.
- **Tối ưu hoá vận hành trong môi trường container**: Các địa chỉ container thay đổi liên tục nhưng không ảnh hưởng đến toàn hệ thống.

2.7 Giao tiếp giữa các Service: REST API trên mạng nội bộ ECS

REST API (Representational State Transfer Application Programming Interface) là phương thức giao tiếp phổ biến nhờ tính đơn giản, linh hoạt và khả năng mở rộng. REST tuân theo các nguyên tắc của kiến trúc hướng tài nguyên, trong đó mỗi tài nguyên được biểu diễn bằng một định danh duy nhất (URI) và được thao tác thông qua các phương thức chuẩn của giao thức HTTP như `GET`, `POST`, `PUT` và `DELETE`. Cách tiếp cận này giúp các dịch vụ duy trì tính độc lập, giảm sự phụ thuộc lẫn nhau và tạo điều kiện thuận lợi cho việc phát triển và triển khai riêng biệt từng thành phần.

Trong hệ thống, các dịch vụ giao tiếp với nhau thông qua REST API truyền qua mạng nội bộ của ECS Cluster. Giao tiếp nội bộ mang lại nhiều lợi ích: (1) tăng cường bảo mật do không cần công khai endpoint ra Internet, (2) giảm thiểu độ trễ vì dữ liệu truyền trong hạ tầng mạng tối ưu hoá của cụm container, và (3) đảm bảo tính nhất quán trong cách các dịch vụ trao đổi dữ liệu. Việc sử dụng REST trong môi trường tách biệt cũng giúp việc giám sát, kiểm thử và mở rộng dịch vụ trở nên dễ dàng hơn, đồng thời cho phép từng dịch vụ có thể phát triển bằng công nghệ hoặc ngôn ngữ lập trình khác nhau mà không ảnh hưởng đến tổng thể hệ thống.

2.8 AWS Amplify

AWS Amplify là một nền tảng phát triển ứng dụng web và di động được xây dựng nhằm đơn giản hóa quá trình triển khai, quản lý và vận hành frontend trong môi trường đám mây. Amplify cung cấp một tập hợp các công cụ và dịch vụ giúp tự động hóa toàn bộ vòng đời phát triển của ứng dụng, từ xây dựng (build), kiểm thử (test), đến triển khai (deploy) và phân phối (hosting).

Một trong những điểm mạnh của Amplify là khả năng hỗ trợ CI/CD tích hợp, cho phép ứng dụng tự động được build và triển khai mỗi khi có thay đổi mới trên các nhánh mã nguồn. Điều này giảm thiểu sai sót thủ công, đồng thời đảm bảo rằng bản phát hành luôn nhất quán với mã nguồn mới nhất.

Ngoài ra, Amplify cung cấp hạ tầng Hosting tĩnh với CDN phân tán toàn cầu, giúp tối ưu hóa tốc độ tải trang cho người dùng ở nhiều khu vực địa lý khác nhau. Bằng việc sử dụng mạng phân phối nội dung (Content Delivery Network), các tệp frontend có thể được truy xuất nhanh chóng, giảm độ trễ và cải thiện trải nghiệm người dùng.

Về mặt bảo mật và vận hành, Amplify hỗ trợ cấu hình dễ dàng các domain tùy chỉnh, chứng chỉ HTTPS, ghi log, theo dõi hiệu năng và khả năng rollback về phiên bản trước khi gặp lỗi. Nhờ sự tích hợp chặt chẽ với các dịch vụ AWS khác, Amplify mang đến môi trường triển khai mạnh mẽ, linh hoạt và phù hợp cho các ứng dụng web hiện đại.

Hình 6: *AWS Amplify*

3 Yêu cầu về hệ thống

3.1 Xác định người dùng

Do hệ thống sẽ được sử dụng trong môi trường đại học, các phân loại người dùng cũng sẽ dựa trên môi trường thực tế. Điều này đảm bảo hệ thống được vận hành trơn tru khi áp dụng vào một trường đại học bất kỳ, cũng như không phải thuê thêm người điều hành riêng. Phân loại người dùng bao gồm:

- Admin: Người phụ trách vận hành, giám sát hệ thống sau khi triển khai. Nhiệm vụ của người dùng này còn bao gồm cài đặt, điều chỉnh cách vận hành của một số tính năng. Admin cũng có thể là một giảng viên trong trường.
- Trưởng khoa: Người phụ trách quản lý một số môn học của một khoa. Vai trò chủ yếu của người dùng này là kiểm duyệt các yêu cầu về quyền hạn người dùng, nội dung của các khóa học.
- Giảng viên: Là người phụ trách đăng tải tài liệu, xây dựng bài học, quiz,... Giảng viên cũng là người sẽ trả lời các câu hỏi trên diễn đàn của sinh viên, theo dõi tiến độ học tập,...
- Trợ giảng (TA): Vai trò chủ yếu của người dùng này là trợ giúp các giảng viên về các vấn đề liên quan tới một khóa học, như đề xuất chỉnh sửa nội dung, thảo luận trên diễn đàn, theo dõi tiến độ sinh viên,...
- Sinh viên: Là người dùng trực tiếp sử dụng các tài nguyên được đăng tải trên hệ thống. Sinh viên có quyền được truy cập các khóa học, xem nội dung khóa học, làm bài tập,...

3.2 Yêu cầu chức năng

Biểu diễn các yêu cầu chức năng dưới dạng User story, ta có bảng sau:

Mã	Ai	Muốn gì	Để làm gì
A-US01	Admin	Quản lý người dùng (tạo, xóa, sửa)	Đảm bảo hệ thống vận hành ổn định và đúng đối tượng sử dụng
A-US02	Admin	Quản lý quyền (thêm, xóa)	Đảm bảo hệ thống vận hành ổn định và đúng đối tượng sử dụng
A-US03	Admin	Cấu hình các tham số hệ thống	Tinh chỉnh hoạt động hệ thống theo yêu cầu thực tế
A-US04	Admin	Theo dõi hoạt động hệ thống (logs, thống kê)	Giám sát, phát hiện lỗi và tối ưu hiệu năng
D-US01	Trưởng khoa	Xác thực yêu cầu cấp quyền cho giảng viên hoặc trợ giảng	Đảm bảo đúng người đúng vai trò trong khoa
D-US02	Trưởng khoa	Tạo, kiểm duyệt nội dung các khóa học của khoa	Đảm bảo chất lượng và tính phù hợp chuyên môn
L-US01	Giảng viên	Yêu cầu quyền truy cập khóa học	Giới hạn chỉnh sửa các khóa học được cấp phép
L-US02	Giảng viên	Tạo và quản lý nội dung khóa học (bài giảng, bài tập, quiz)	Xây dựng tài liệu giúp sinh viên học tập hiệu quả
L-US03	Giảng viên	Theo dõi tiến độ học tập của sinh viên	Đánh giá và hỗ trợ học tập kịp thời
L-US04	Giảng viên	Trả lời câu hỏi trên diễn đàn khóa học	Giải đáp thắc mắc cho sinh viên
T-US01	Trợ giảng	Yêu cầu quyền truy cập khóa học	Giới hạn chỉnh sửa các khóa học được cấp phép
T-US02	Trợ giảng	Hỗ trợ giảng viên chỉnh sửa hoặc đề xuất nội dung khóa học	Giảm tải công việc và nâng cao chất lượng khóa học
T-US03	Trợ giảng	Tham gia thảo luận trên diễn đàn	Hỗ trợ giải đáp câu hỏi của sinh viên
T-US04	Trợ giảng	Theo dõi tiến độ sinh viên	Hỗ trợ giảng viên trong đánh giá và nhắc nhở sinh viên
S-US01	Sinh viên	Truy cập và xem nội dung khóa học	Học tập theo chương trình giảng viên cung cấp
S-US02	Sinh viên	Làm bài tập, quiz và xem kết quả	Đánh giá mức độ hiểu bài và cải thiện kiến thức
S-US03	Sinh viên	Đặt câu hỏi và thảo luận trên diễn đàn	Trao đổi với giảng viên, trợ giảng và bạn học
S-US04	Sinh viên	Theo dõi tiến độ học tập của bản thân	Quản lý quá trình học tập và hoàn thành yêu cầu khóa học

Bảng 1: *Bảng User Stories*

3.3 Yêu cầu phi chức năng

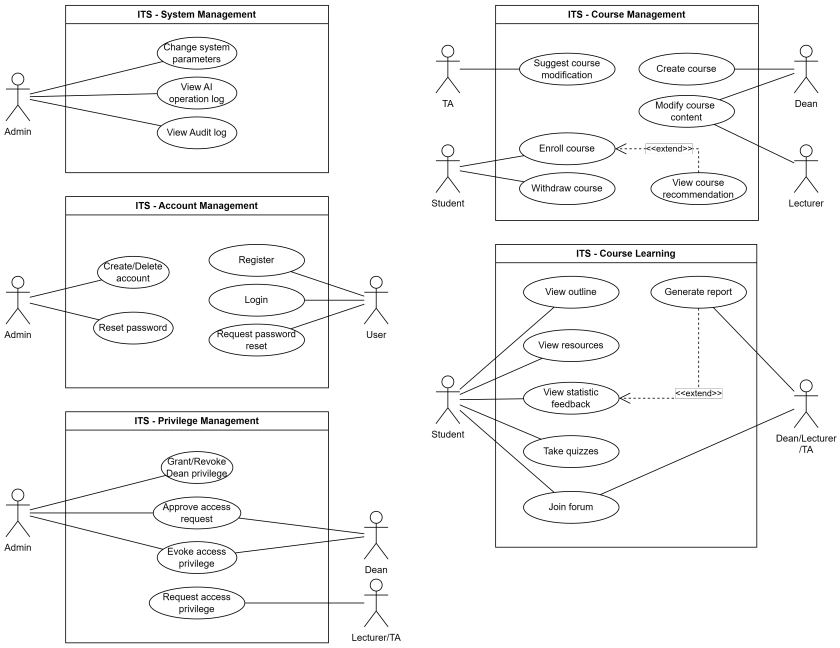
Từ kết quả thăm dò ý kiến người dùng, kết hợp brainstorm trong các buổi họp nhóm, các yêu cầu phi chức năng của hệ thống được miêu tả như sau:

Mã NFR	Mô tả yêu cầu phi chức năng
NRF01	Hệ thống phải cho phép thay đổi hành vi vận hành mà không cần chỉnh sửa mã nguồn, bao gồm bật/tắt tính năng và điều chỉnh tham số hoạt động.
NRF02	Hệ thống phải hỗ trợ thay đổi mô hình AI mà không gây gián đoạn dịch vụ (zero-downtime model update).
NRF03	Hệ thống phải cho phép cấu hình loại tài liệu được phép đăng tải trong khóa học mà không cần triển khai lại.
NRF04	Hệ thống phải cho phép chỉnh sửa layout hoặc cấu trúc giao diện khóa học thông qua cơ chế cấu hình.
NRF05	Module tạo báo cáo phải hỗ trợ cấu hình layout của báo cáo.
NRF06	Module tạo báo cáo phải cho phép cấu hình phạm vi dữ liệu được sử dụng trong báo cáo.
NRF07	Module tạo quiz bằng AI phải hỗ trợ cấu hình số lượng câu hỏi, độ khó và phân bố độ khó.
NRF08	Module gợi ý hint phải cho phép cấu hình tần suất và mức độ chi tiết của gợi ý.
NRF09	Module gợi ý chủ đề phải cho phép thêm hoặc loại bỏ chủ đề quan tâm thông qua cấu hình.
NRF10	Hệ thống phải đảm bảo mức độ sẵn sàng không dưới 99% trong thời gian có lưu lượng truy cập cao.
NRF11	Khi xảy ra sự cố downtime, hệ thống phải tự phục hồi trong thời gian không vượt quá 5 phút.
NRF12	Các thành phần quan trọng phải được triển khai theo mô hình nhiều vùng sẵn sàng (multi-AZ) hoặc tương đương.
NRF13	Dữ liệu bài giảng đăng tải phải được kiểm tra tính hợp lệ (định dạng, nội dung đầy đủ).
NRF14	Mọi dữ liệu tải lên phải được lưu trữ an toàn, bao gồm mã hóa khi lưu trữ và khi truyền tải.
NRF15	Mọi dữ liệu hiển thị phải đảm bảo nhất quán giữa các người dùng và các phiên xử lý.
NRF16	Các thao tác quan trọng phải đảm bảo tính toàn vẹn giao dịch (transactional integrity).
NRF17	Lịch sử chỉnh sửa nội dung khóa học phải được ghi lại để phục vụ truy vết.
NRF18	Các API liên quan đến AI phải có thời gian phản hồi trung bình dưới 2 giây.
NRF19	Thời gian tải nội dung khóa học không vượt quá 1 giây trên kết nối mạng phổ thông.
NRF20	Hệ thống phải sử dụng cơ chế caching để cải thiện tốc độ truy xuất dữ liệu.
NRF21	Hệ thống phải hỗ trợ ít nhất X sinh viên truy cập đồng thời (giá trị X được xác định sau khi ước tính tải).
NRF22	Hệ thống phải tránh xung đột dữ liệu (race condition) khi nhiều người dùng thao tác song song.
NRF23	Hệ thống phải tự động mở rộng tài nguyên khi tải tăng cao, đặc biệt trong mùa thi.
NRF24	Hệ thống phải tự động thu hẹp tài nguyên khi tải giảm để tối ưu chi phí vận hành.
NRF25	Các dịch vụ AI và quiz phải hỗ trợ autoscaling dựa trên CPU/GPU hoặc số lượng yêu cầu mỗi giây.

Bảng 2: *Danh sách yêu cầu phi chức năng (NFR)*

4 Đặc tả chi tiết các use-case

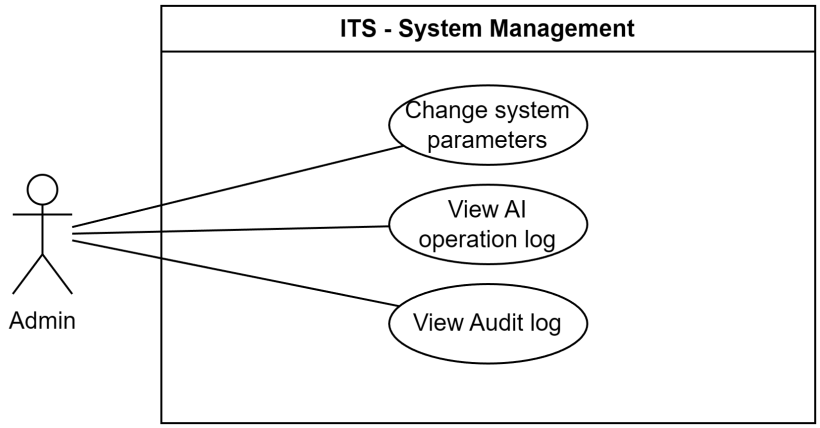
Để thiết kế kiến trúc của hệ thống, nhóm đã họp và triển khai các User story thành các use-case cụ thể.



Hình 7: *Tổng quan các Use-case*

4.1 Quản lý hệ thống

4.1.1 Tổng quan



Hình 8: *Các use-case về Quản lý hệ thống*

4.1.2 Đặc tả chi tiết

Bảng 3: *Usecase «Cấu hình hệ thống»*

Usecase	**Cấu hình hệ thống**
Nguồn	A-US03, NFR01
Use-case ID	UC-SM-01
Tổng quan	Admin chỉnh sửa các tham số hệ thống để tinh chỉnh hoạt động
Actor	Admin

Usecase	**Cấu hình hệ thống**
Tiền điều kiện	• Hệ thống đang hoạt động ổn định. • Admin đã đăng nhập và có quyền.
Điều kiện kích hoạt	• Admin truy cập vào trang cấu hình hệ thống.
Các bước	1. Hiển thị giao diện cấu hình tham số hệ thống. 2. Admin thay đổi các tham số cấu hình (lập lịch trích xuất, quản lý phạm vi trích xuất, mức hoạt động tối thiểu, hệ số đa dạng đề xuất). 3. Admin lưu thay đổi và xác nhận. 4. Hệ thống cập nhật và áp dụng cấu hình mới.
Hậu điều kiện	• Các tham số hệ thống được cập nhật thành công. • Dịch vụ liên quan áp dụng tham số mới.
Xử lý ngoại lệ	ex1. Không có thay đổi khi lưu, hệ thống hỏi xác nhận. ex2. Lỗi lưu trữ, hiển thị thông báo lỗi.

Bảng 4: *Usecase «Xem nhật ký hoạt động AI»*

Usecase	**Xem nhật ký hoạt động AI**
Nguồn	A-US04, NFR18
Use-case ID	UC-SM-02
Tổng quan	Admin xem nhật ký hoạt động (logs) của hệ thống AI để giám sát và phát hiện sự cố.
Actor	Admin
Tiền điều kiện	• Admin đã đăng nhập với quyền xem nhật ký.
Điều kiện kích hoạt	• Admin truy cập trang nhật ký hoạt động AI.
Các bước	1. Hệ thống hiển thị danh sách các bản ghi nhật ký AI, hỗ trợ lọc theo thời gian và mức độ nghiêm trọng. (Message, Warning, Error) 2. Admin lựa chọn bản ghi hoặc khoảng thời gian cần xem. 3. Hệ thống hiển thị chi tiết nhật ký theo yêu cầu.
Hậu điều kiện	• Admin có thể nắm bắt được thông tin hoạt động, cảnh báo hoặc lỗi của hệ thống AI.
Xử lý ngoại lệ	ex1. Nếu không có dữ liệu nhật ký trong khoảng thời gian yêu cầu, hệ thống hiển thị thông báo "Không có dữ liệu". ex2. Nếu xảy ra lỗi truy xuất dữ liệu nhật ký (ví dụ: mất kết nối), hệ thống thông báo lỗi và gợi ý tải lại trang.

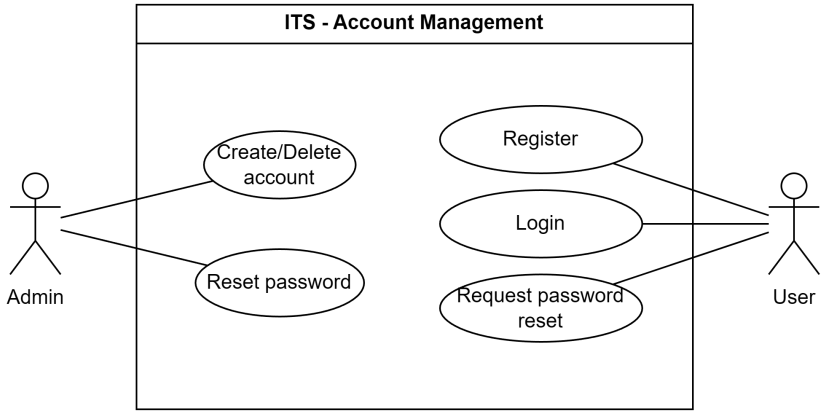
Bảng 5: *Usecase «Xem nhật ký kiểm toán (Audit log)»*

Usecase	**Xem nhật ký kiểm toán (Audit log)**
Nguồn	A-US04
Use-case ID	UC-SM-03
Tổng quan	Admin xem nhật ký kiểm toán để giám sát các hành động hệ thống, đảm bảo an toàn và phát hiện hành vi bất thường.

Usecase	Xem nhật ký kiểm toán (Audit log)
Actor	Admin
Tiền điều kiện	• Admin đã đăng nhập với quyền xem audit log.
Điều kiện kích hoạt	• Admin truy cập trang nhật ký kiểm toán.
Các bước	1. Hệ thống hiển thị danh sách các sự kiện audit log, hỗ trợ lọc theo người dùng, loại sự kiện, thời gian. 2. Admin chọn bộ lọc hoặc tìm kiếm cụ thể. 3. Hệ thống hiển thị chi tiết các sự kiện phù hợp.
Hậu điều kiện	• Admin có thể kiểm tra và truy vết các thao tác hệ thống quan trọng.
Xử lý ngoại lệ	ex1. Nếu không có sự kiện nào phù hợp bộ lọc, hệ thống hiển thị thông báo "Không tìm thấy sự kiện". ex2. Nếu lỗi truy xuất dữ liệu audit log, hệ thống thông báo lỗi và yêu cầu tải lại.

4.2 Quản lý tài khoản

4.2.1 Tổng quan



Hình 9: *Các use-case về Quản lý tài khoản*

4.2.2 Đặc tả chi tiết

Bảng 6: *Usecase «Tạo hoặc Xóa tài khoản»*

Usecase	Tạo hoặc Xóa tài khoản
Nguồn	A-US01
Use-case ID	UC-AM-01
Tổng quan	Admin quản lý tài khoản người dùng: tạo mới hoặc xóa tài khoản khi cần thiết.
Actor	Admin
Tiền điều kiện	• Admin đã đăng nhập với quyền quản lý tài khoản.
Điều kiện kích hoạt	• Admin truy cập vào trang quản lý tài khoản.

Usecase	**Tạo hoặc Xóa tài khoản**
Các bước	1. Admin chọn thao tác tạo tài khoản mới hoặc xóa tài khoản hiện có. 2. Nếu tạo, Admin nhập các thông tin cần thiết (tên, email, quyền hạn,...). 3. Hệ thống kiểm tra dữ liệu hợp lệ và tạo tài khoản. 4. Nếu xóa, Admin chọn tài khoản cần xóa và xác nhận. 5. Hệ thống xóa tài khoản và thông báo thành công.
Hậu điều kiện	• Tài khoản được tạo mới hoặc xóa khỏi hệ thống thành công.
Xử lý ngoại lệ	ex1. Nếu dữ liệu nhập không hợp lệ (ví dụ: email trùng, thiếu thông tin bắt buộc), hệ thống thông báo lỗi. ex2. Nếu tài khoản cần xóa đang liên kết với dữ liệu quan trọng (ví dụ: đang quản lý khóa học), hệ thống cảnh báo và yêu cầu xác nhận thêm hoặc từ chối xóa. ex3. Nếu lỗi hệ thống khi lưu hoặc xóa, thông báo lỗi và gợi ý thử lại.

Bảng 7: *Usecase «Đặt lại mật khẩu (Reset password)»*

Usecase	**Đặt lại mật khẩu (Reset password)**
Nguồn	A-US01
Use-case ID	UC-AM-02
Tổng quan	Admin có thể đặt lại mật khẩu cho tài khoản người dùng khi cần thiết.
Actor	Admin
Tiền điều kiện	• Admin đã đăng nhập . • Admin có quyền đặt lại mật khẩu.
Điều kiện kích hoạt	• Admin truy cập trang quản lý tài khoản và chọn đặt lại mật khẩu.
Các bước	1. Admin chọn tài khoản người dùng cần đặt lại mật khẩu. 2. Nhập mật khẩu mới hoặc hệ thống sinh mật khẩu ngẫu nhiên. 3. Xác nhận thao tác đặt lại mật khẩu. 4. Hệ thống cập nhật mật khẩu và thông báo thành công.
Hậu điều kiện	• Mật khẩu của tài khoản được thay đổi và có hiệu lực ngay lập tức.
Xử lý ngoại lệ	ex1. Nếu tài khoản không tồn tại hoặc không hợp lệ, hệ thống thông báo lỗi. ex2. Nếu lỗi hệ thống khi cập nhật, thông báo lỗi và yêu cầu thử lại.

Bảng 8: *Usecase «Đăng ký tài khoản»*

Usecase	**Đăng ký tài khoản**
Nguồn	
Use-case ID	UC-AM-03
Tổng quan	Người dùng mới có thể đăng ký tài khoản để sử dụng hệ thống.
Actor	User
Tiền điều kiện	• Người dùng chưa có tài khoản trong hệ thống.

Usecase	**Đăng ký tài khoản**
Điều kiện kích hoạt	• Người dùng truy cập trang đăng ký tài khoản.
Các bước	1. Người dùng nhập thông tin cá nhân và tài khoản (email, mật khẩu, tên,...). 2. Hệ thống kiểm tra tính hợp lệ của dữ liệu. 3. Hệ thống tạo tài khoản mới và gửi email xác nhận (nếu có). 4. Người dùng kích hoạt tài khoản qua email (nếu áp dụng).
Hậu điều kiện	• Người dùng có thể đăng nhập và sử dụng hệ thống.
Xử lý ngoại lệ	ex1. Nếu email đã được đăng ký, hệ thống báo lỗi. ex2. Nếu dữ liệu không hợp lệ, yêu cầu nhập lại. ex3. Nếu lỗi gửi email xác nhận, cảnh báo và hướng dẫn liên hệ quản trị.

Bảng 9: *Usecase «Đăng nhập»*

Usecase	**Đăng nhập**
Nguồn	A-US01
Use-case ID	UC-AM-04
Tổng quan	Người dùng đăng nhập vào hệ thống để sử dụng các chức năng.
Actor	User
Tiền điều kiện	• Người dùng đã có tài khoản hợp lệ.
Điều kiện kích hoạt	• Người dùng truy cập trang đăng nhập.
Các bước	1. Người dùng nhập thông tin đăng nhập (tên đăng nhập và mật khẩu). 2. Hệ thống xác thực thông tin. 3. Nếu thành công, hệ thống cho phép truy cập vào hệ thống. 4. Nếu thất bại, hiển thị thông báo lỗi và hướng dẫn thử lại.
Hậu điều kiện	• Người dùng có phiên làm việc hợp lệ.
Xử lý ngoại lệ	ex1. Nếu tài khoản bị khóa hoặc chưa kích hoạt, hệ thống thông báo trạng thái. ex2. Nếu lỗi hệ thống khi xác thực, thông báo lỗi và yêu cầu thử lại.

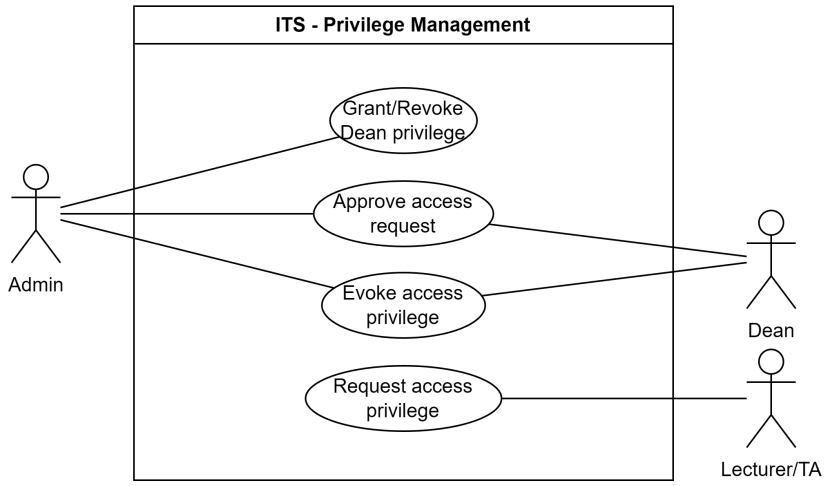
Bảng 10: *Usecase «Yêu cầu đặt lại mật khẩu»*

Usecase	**Yêu cầu đặt lại mật khẩu**
Nguồn	
Use-case ID	UC-AM-05
Tổng quan	Người dùng yêu cầu hệ thống gửi hướng dẫn đặt lại mật khẩu khi quên mật khẩu.
Actor	User
Tiền điều kiện	• Người dùng có tài khoản đã đăng ký.
Điều kiện kích hoạt	• Người dùng truy cập trang "Quên mật khẩu" và nhập email đăng ký.

Usecase	Yêu cầu đặt lại mật khẩu
Các bước	1. Người dùng nhập địa chỉ email đã đăng ký. 2. Hệ thống kiểm tra email có tồn tại trong hệ thống. 3. Nếu có, hệ thống gửi tạo yêu cầu đặt lại mật khẩu cho Admin 4. Hệ thống hiển thị thông báo gửi yêu cầu thành công.
Hậu điều kiện	• Yêu cầu đặt lại mật khẩu được tạo.
Xử lý ngoại lệ	ex1. Nếu email không tồn tại, hệ thống thông báo không tìm thấy tài khoản.

4.3 Quản lý quyền truy cập

4.3.1 Tổng quan



Hình 10: *Các use-case về Quản lý quyền chỉnh sửa khóa học*

4.3.2 Đặc tả chi tiết

Bảng 11: *Usecase «Cấp và thu hồi quyền Trưởng khoa»*

Usecase	Cấp và thu hồi quyền Trưởng khoa
Nguồn	A-US02
Use-case ID	UC-PM-01
Tổng quan	Admin thực hiện cấp hoặc thu hồi quyền Trưởng khoa cho người dùng phù hợp để đảm bảo phân quyền đúng đối tượng.
Actor	Admin
Tiền điều kiện	• Admin đã đăng nhập. • Admin có quyền quản lý quyền truy cập. • Người dùng cần cấp/thu hồi quyền tồn tại trong hệ thống.
Điều kiện kích hoạt	• Admin truy cập trang quản lý quyền và chọn chức năng cấp hoặc thu hồi quyền Trưởng khoa.

Usecase	**Cấp và thu hồi quyền Trưởng khoa**
Các bước	1. Admin tìm kiếm hoặc chọn người dùng cần thay đổi quyền. 2. Admin chọn cấp quyền Trưởng khoa hoặc thu hồi quyền. 3. Xác nhận thao tác cấp hoặc thu hồi. 4. Hệ thống cập nhật quyền truy cập cho người dùng. 5. Hệ thống hiển thị thông báo thực hiện thành công.
Hậu điều kiện	• Quyền Trưởng khoa được cập nhật chính xác theo yêu cầu.
Xử lý ngoại lệ	ex1. Nếu người dùng không tồn tại hoặc đã có quyền tương ứng, hệ thống thông báo lỗi và ngăn thao tác. ex2. Nếu lỗi hệ thống khi cập nhật quyền, hệ thống hiển thị lỗi và yêu cầu thử lại. ex3. Nếu Admin không có quyền thực hiện thao tác, hệ thống từ chối truy cập.

Bảng 12: *Usecase «Phê duyệt yêu cầu quyền truy cập»*

Usecase	**Phê duyệt yêu cầu quyền truy cập**
Nguồn	A-US02, D-US01
Use-case ID	UC-PM-02
Tổng quan	Admin hoặc Trưởng khoa xem và phê duyệt các yêu cầu cấp quyền truy cập của giảng viên hoặc trợ giảng.
Actor	Admin / Dean (Trưởng khoa)
Tiền điều kiện	• Admin hoặc Trưởng khoa đã đăng nhập với quyền phê duyệt. • Có các yêu cầu quyền truy cập đang chờ xử lý.
Điều kiện kích hoạt	• Người có quyền truy cập vào trang phê duyệt yêu cầu quyền.
Các bước	1. Xem danh sách các yêu cầu cấp quyền đang chờ. 2. Chọn từng yêu cầu để xem chi tiết. 3. Phê duyệt hoặc từ chối yêu cầu. 4. Hệ thống cập nhật trạng thái và thông báo kết quả cho người yêu cầu.
Hậu điều kiện	• Quyền truy cập được cấp hoặc yêu cầu bị từ chối chính xác. • Người yêu cầu được thông báo kết quả kịp thời.
Xử lý ngoại lệ	ex1. Nếu yêu cầu không tồn tại hoặc đã được xử lý, hệ thống thông báo và không thực hiện thao tác. ex2. Nếu lỗi cập nhật quyền truy cập, hệ thống báo lỗi và cho phép thử lại. ex3. Nếu người phê duyệt không có quyền hợp lệ, hệ thống từ chối thao tác.

Bảng 13: *Usecase «Thu hồi quyền truy cập»*

Usecase	**Thu hồi quyền truy cập**
Nguồn	A-US02, D-US01
Use-case ID	UC-PM-03
Tổng quan	Admin hoặc Trưởng khoa thu hồi quyền truy cập của giảng viên hoặc trợ giảng khi không còn phù hợp.
Actor	Admin / Dean (Trưởng khoa)

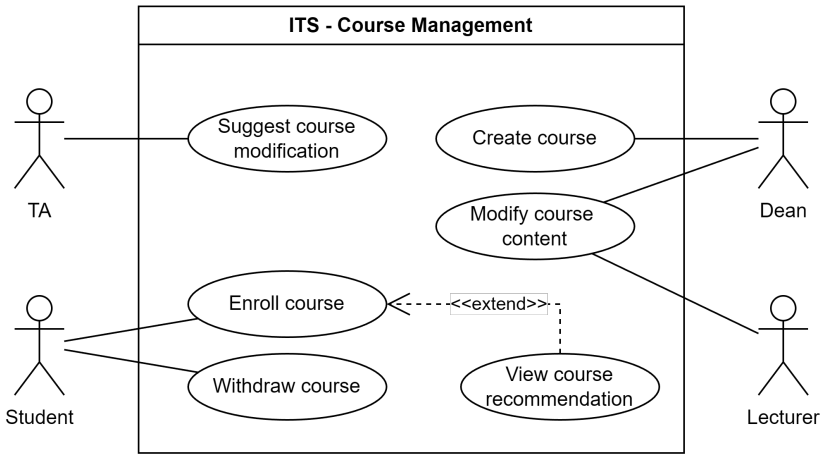
Usecase	**Thu hồi quyền truy cập**
Tiền điều kiện	• Người thực hiện đã đăng nhập với quyền thu hồi quyền. • Người dùng bị thu hồi quyền đang có quyền hợp lệ trong hệ thống.
Điều kiện kích hoạt	• Người thực hiện truy cập trang quản lý quyền và chọn thu hồi quyền.
Các bước	1. Tìm hoặc chọn người dùng cần thu hồi quyền. 2. Chọn quyền cần thu hồi. 3. Xác nhận thao tác thu hồi. 4. Hệ thống cập nhật và ghi nhận thay đổi. 5. Hiển thị thông báo thu hồi quyền thành công.
Hậu điều kiện	• Quyền truy cập bị thu hồi có hiệu lực ngay lập tức.
Xử lý ngoại lệ	ex1. Nếu người dùng không tồn tại hoặc không có quyền được thu hồi, thông báo lỗi. ex2. Nếu lỗi hệ thống khi cập nhật, hiển thị thông báo và đề nghị thử lại. ex3. Nếu người thực hiện không đủ quyền, hệ thống từ chối thao tác.

Bảng 14: *Usecase «Yêu cầu quyền truy cập»*

Usecase	**Yêu cầu quyền truy cập**
Nguồn	L-US01, T-US01
Use-case ID	UC-PM-04
Tổng quan	Giảng viên hoặc trợ giảng gửi yêu cầu xin cấp quyền truy cập để thực hiện các chức năng nâng cao.
Actor	Lecturer / TA (Giảng viên / Trợ giảng)
Tiền điều kiện	• Người dùng đã đăng nhập và chưa có quyền cần xin cấp.
Điều kiện kích hoạt	• Người dùng truy cập trang yêu cầu quyền truy cập.
Các bước	1. Người dùng chọn loại quyền cần xin cấp. 2. Nhập lý do hoặc mô tả nhu cầu sử dụng quyền. 3. Gửi yêu cầu lên hệ thống. 4. Hệ thống lưu yêu cầu và thông báo đã gửi thành công.
Hậu điều kiện	• Yêu cầu quyền được ghi nhận và chờ phê duyệt.
Xử lý ngoại lệ	ex1. Nếu người dùng đã có quyền tương ứng, hệ thống báo không cần gửi yêu cầu. ex2. Nếu dữ liệu gửi lên không hợp lệ (ví dụ: thiếu lý do), hệ thống thông báo lỗi. ex3. Nếu lỗi hệ thống khi lưu yêu cầu, hệ thống báo lỗi và đề nghị thử lại.

4.4 Quản lý khóa học

4.4.1 Tổng quan



Hình 11: *Các use-case về Quản lý khóa học*

4.4.2 Đặc tả chi tiết

Bảng 15: *Usecase «Đề xuất chỉnh sửa khóa học»*

Usecase	**Đề xuất chỉnh sửa khóa học**
Nguồn	T-US02
Use-case ID	UC-CM-01
Tổng quan	Trợ giảng đề xuất chỉnh sửa nội dung khóa học để nâng cao chất lượng.
Actor	TA (Trợ giảng)
Tiền điều kiện	• Trợ giảng đã đăng nhập và có quyền truy cập vào khóa học được phân công.
Điều kiện kích hoạt	• Trợ giảng truy cập trang khóa học và chọn chức năng đề xuất chỉnh sửa.
Các bước	1. Trợ giảng xem nội dung khóa học hiện tại. 2. Trợ giảng chỉnh sửa hoặc nhập đề xuất thay đổi. 3. Gửi đề xuất cho trưởng khoa hoặc giảng viên xem xét. 4. Hệ thống lưu và thông báo trạng thái đề xuất.
Hậu điều kiện	• Đề xuất được ghi nhận và chờ phê duyệt.
Xử lý ngoại lệ	ex1. Nếu dữ liệu đề xuất không hợp lệ, hệ thống thông báo lỗi. ex2. Nếu lỗi lưu trữ, thông báo lỗi và yêu cầu thử lại.

Bảng 16: *Usecase «Tạo khóa học»*

Usecase	**Tạo khóa học**
Nguồn	D-US02
Use-case ID	UC-CM-02
Tổng quan	Trưởng khoa tạo mới khóa học thuộc khoa mình quản lý.

Usecase	**Tạo khóa học**
Actor	Dean (Trưởng khoa)
Tiền điều kiện	• Trưởng khoa đã đăng nhập và có quyền tạo khóa học.
Điều kiện kích hoạt	• Trưởng khoa truy cập trang quản lý khóa học và chọn tạo mới.
Các bước	1. Nhập các thông tin khóa học (tên, mã, mô tả,...). 2. Xác nhận thông tin và lưu khóa học mới. 3. Hệ thống tạo khóa học và hiển thị thông báo thành công.
Hậu điều kiện	• Khóa học được thêm mới trong hệ thống và sẵn sàng cho giảng viên quản lý.
Xử lý ngoại lệ	ex1. Nếu thông tin không hợp lệ hoặc trùng mã khóa học, thông báo lỗi. ex2. Nếu lỗi hệ thống khi lưu, thông báo lỗi và yêu cầu thử lại.

Bảng 17: *Usecase «Sửa nội dung khóa học»*

Usecase	**Sửa nội dung khóa học**
Nguồn	L-US01
Use-case ID	UC-CM-03
Tổng quan	Giảng viên sửa đổi nội dung khóa học để cập nhật bài giảng, bài tập, quiz,...
Actor	Lecturer (Giảng viên)
Tiền điều kiện	• Giảng viên đã đăng nhập và được phân quyền chỉnh sửa khóa học.
Điều kiện kích hoạt	• Giảng viên truy cập trang quản lý khóa học và chọn chỉnh sửa.
Các bước	1. Hiển thị nội dung khóa học hiện tại (các chương, phần trong chương, chủ đề của phần, tài li) 2. Giảng viên thực hiện chỉnh sửa các phần nội dung. 3. Lưu thay đổi và hệ thống cập nhật nội dung. 4. Hệ thống ghi lại lịch sử chỉnh sửa để truy vết.
Hậu điều kiện	• Nội dung khóa học được cập nhật và hiển thị cho sinh viên.
Xử lý ngoại lệ	ex1. Nếu dữ liệu nhập không hợp lệ, cảnh báo và yêu cầu chỉnh sửa. ex2. Nếu lỗi lưu dữ liệu, hệ thống thông báo và cho phép thử lại.

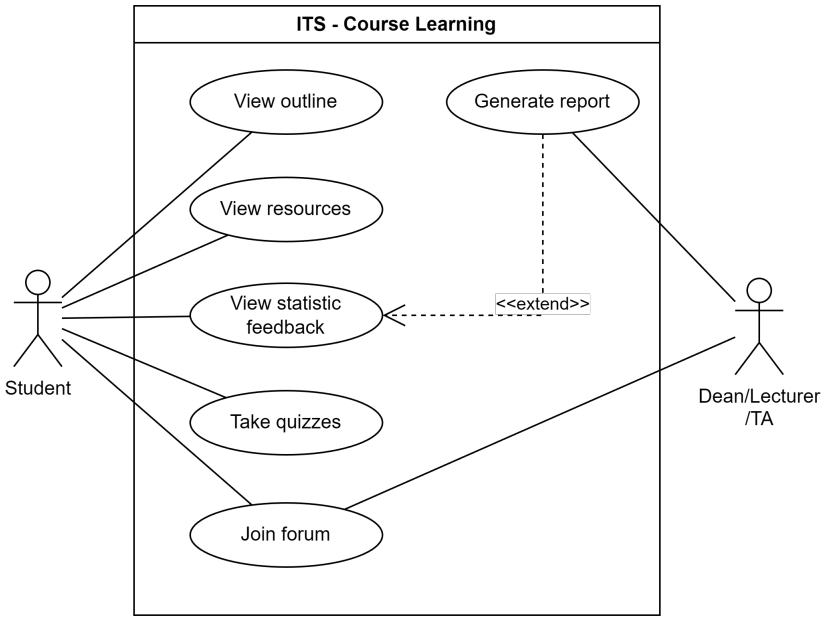
Bảng 18: *Usecase «Ghi danh khóa học»*

Usecase	**Ghi danh khóa học**
Nguồn	S-US01
Use-case ID	UC-CM-04
Tổng quan	Sinh viên ghi danh tham gia khóa học.
Actor	Student (Sinh viên)
Tiền điều kiện	• Sinh viên đã đăng nhập. • Khóa học đang mở ghi danh.

Usecase	Ghi danh khóa học
Điều kiện kích hoạt	• Sinh viên chọn khóa học và chọn ghi danh.
Các bước	1. Hiển thị thông tin khóa học. 2. Sinh viên nhấn ghi danh. 3. Hệ thống kiểm tra điều kiện và ghi nhận đăng ký. 4. Hiển thị xác nhận ghi danh thành công.
Hậu điều kiện	• Sinh viên được ghi danh và có thể truy cập nội dung khóa học.
Xử lý ngoại lệ	ex1. Nếu khóa học không cho phép ghi danh, thông báo lỗi. ex2. Nếu sinh viên đã ghi danh trước đó, thông báo không thể ghi danh lại. ex3. Nếu lỗi hệ thống khi ghi danh, cảnh báo và đề nghị thử

4.5 Học khóa học

4.5.1 Tổng quan



Hình 12: *Các use-case về Học khóa học*

4.5.2 Đặc tả chi tiết

Bảng 19: *Usecase «Xem đề cương khóa học»*

Usecase	Xem đề cương khóa học
Nguồn	S-US01, NFR19
Use-case ID	UC-CL-01
Tổng quan	Sinh viên xem đề cương chi tiết của khóa học để nắm được nội dung và kế hoạch học tập.
Actor	Student
Tiền điều kiện	• Sinh viên đã đăng nhập và đã ghi danh khóa học. • Đề cương khóa học đã được cập nhật trên hệ thống.

Usecase	**Xem đề cương khóa học**
Điều kiện kích hoạt	Sinh viên chọn khóa học và yêu cầu xem đề cương.
Các bước	1. Hệ thống hiển thị danh sách khóa học mà sinh viên đã đăng ký. 2. Sinh viên chọn khóa học muốn xem đề cương. 3. Hệ thống tải và hiển thị đề cương khóa học.
Hậu điều kiện	• Sinh viên nhận được đề cương đầy đủ, rõ ràng.
Xử lý ngoại lệ	ex1. Sinh viên chưa đăng ký khóa học, hệ thống thông báo không có quyền truy cập. ex2. Đề cương chưa được cập nhật hoặc lỗi tải, hệ thống thông báo lỗi và hướng dẫn liên hệ quản trị. ex3. Lỗi kết nối mạng hoặc hệ thống, hệ thống yêu cầu tải lại trang hoặc thử lại sau.

Bảng 20: *Usecase «Xem tài liệu học tập»*

Usecase	**Xem tài liệu học tập**
Nguồn	S-US01, NFR19, NFR20
Use-case ID	UC-CL-02
Tổng quan	Sinh viên truy cập và sử dụng tài liệu học tập được cung cấp trong khóa học.
Actor	Student
Tiền điều kiện	• Sinh viên đã đăng nhập và ghi danh khóa học. • Tài liệu học tập đã được giảng viên hoặc trợ giảng đăng tải hợp lệ.
Điều kiện kích hoạt	Sinh viên chọn mục tài liệu học tập trong khóa học.
Các bước	1. Hệ thống liệt kê tài liệu học tập hiện có trong khóa học. 2. Sinh viên chọn tài liệu cần xem hoặc tải về. 3. Hệ thống hiển thị hoặc cung cấp file tải cho sinh viên.
Hậu điều kiện	• Sinh viên tiếp cận tài liệu học tập đầy đủ, thuận tiện.
Xử lý ngoại lệ	ex1. Tài liệu bị thiếu hoặc lỗi tải, hệ thống thông báo và đề nghị liên hệ quản trị. ex2. Sinh viên không có quyền truy cập do chưa đăng ký khóa học, hệ thống thông báo rõ. ex3. Lỗi mạng hoặc lỗi hệ thống, hệ thống hỗ trợ tải lại hoặc báo lỗi tạm thời.

Bảng 21: *Usecase «Xem phản hồi thống kê»*

Usecase	**Xem phản hồi thống kê**
Nguồn	S-US04, NFR19
Use-case ID	UC-CL-03
Tổng quan	Sinh viên xem các thống kê về tiến độ học tập và kết quả để theo dõi sự tiến bộ của bản thân.
Actor	Student

Usecase	**Xem phản hồi thống kê**
Tiền điều kiện	• Sinh viên đã ghi danh và có dữ liệu học tập trên hệ thống. • Hệ thống đã tổng hợp dữ liệu thống kê tiến độ học tập.
Điều kiện kích hoạt	Sinh viên truy cập trang thống kê học tập.
Các bước	1. Hệ thống hiển thị các số liệu thống kê (điểm quiz, bài tập, tiến độ khóa học). 2. Sinh viên xem và phân tích kết quả.
Hậu điều kiện	• Sinh viên có thông tin đầy đủ để điều chỉnh kế hoạch học tập.
Xử lý ngoại lệ	ex1. Dữ liệu thống kê chưa đủ hoặc bị lỗi, hệ thống thông báo. ex2. Lỗi hệ thống khi tải dữ liệu, hiển thị thông báo lỗi.

Bảng 22: *Usecase «Làm bài kiểm tra (quiz)»*

Usecase	**Làm bài kiểm tra (quiz)**
Nguồn	S-US02, NFR07, NFR18
Use-case ID	UC-CL-04
Tổng quan	Sinh viên làm các bài kiểm tra để đánh giá kiến thức đã học.
Actor	Student
Tiền điều kiện	• Sinh viên đã đăng ký khóa học và được phép làm quiz. • Quiz đã được tạo và kích hoạt.
Điều kiện kích hoạt	Sinh viên chọn làm bài kiểm tra trong khóa học.
Các bước	1. Hệ thống hiển thị danh sách bài quiz có thể làm. 2. Sinh viên bắt đầu làm bài, trả lời các câu hỏi. 3. Hệ thống ghi nhận và đánh giá câu trả lời. 4. Kết thúc bài kiểm tra, hệ thống hiển thị kết quả và phản hồi.
Hậu điều kiện	• Điểm số và kết quả được lưu trữ để theo dõi và đánh giá.
Xử lý ngoại lệ	ex1. Quiz hết hạn hoặc không tồn tại, hệ thống thông báo không thể làm bài. ex2. Lỗi khi lưu kết quả, hệ thống báo lỗi và cho phép thử lại. ex3. Sự cố kết nối hoặc gián đoạn trong quá trình làm bài, hệ thống có cơ chế lưu tạm thời hoặc hướng dẫn xử lý.

Bảng 23: *Usecase «Tạo báo cáo thống kê»*

Usecase	**Tạo báo cáo thống kê**
Nguồn	L-US03, T-US04, NFR05, NFR06, NFR18
Use-case ID	UC-CL-05
Tổng quan	Trưởng khoa, giảng viên hoặc trợ giảng tạo báo cáo thống kê tiến độ và kết quả học tập sinh viên.
Actor	Dean / Lecturer / TA
Tiền điều kiện	• Người dùng có quyền truy cập chức năng tạo báo cáo. • Dữ liệu học tập đã được tổng hợp đầy đủ.

Usecase	**Tạo báo cáo thống kê**
Điều kiện kích hoạt	Người dùng chọn tạo báo cáo thống kê trong khóa học.
Các bước	1. Chọn phạm vi dữ liệu và khoảng thời gian cần báo cáo. 2. Chọn mẫu báo cáo (layout) nếu có. 3. Hệ thống sinh báo cáo thống kê theo cấu hình. 4. Người dùng xem, tải hoặc xuất báo cáo.
Hậu điều kiện	• Báo cáo được tạo đúng định dạng, dữ liệu chính xác.
Xử lý ngoại lệ	ex1. Không có dữ liệu phù hợp để tạo báo cáo, hệ thống thông báo. ex2. Lỗi khi sinh báo cáo, hệ thống thông báo lỗi. ex3. Người dùng không có quyền truy cập chức năng, hệ thống từ chối.

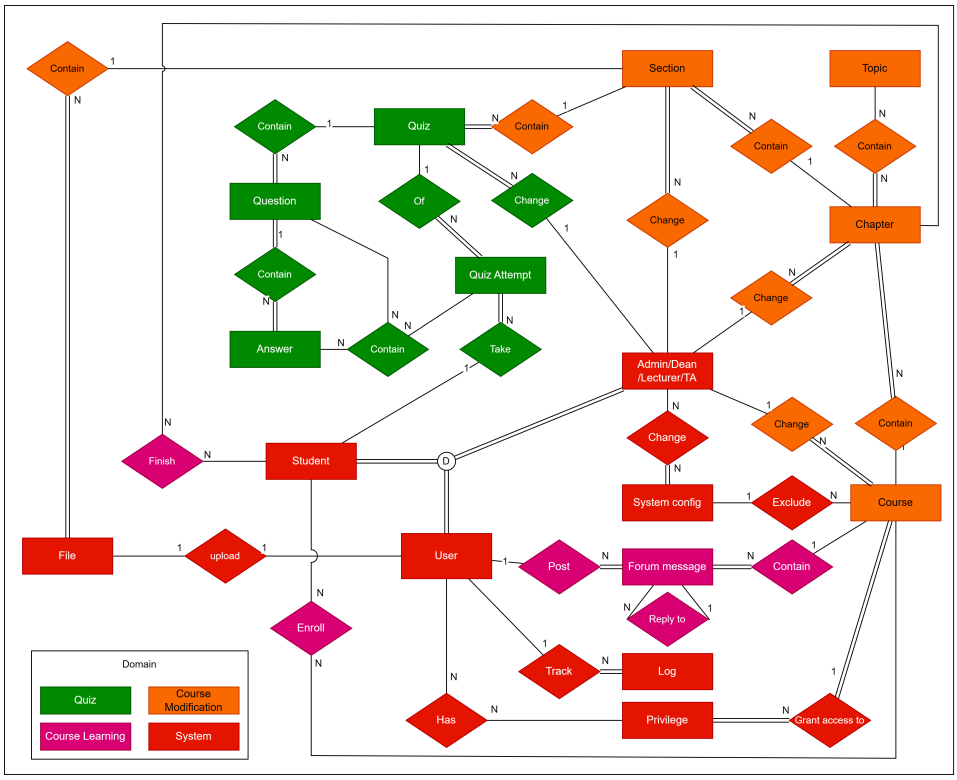
Bảng 24: *Usecase «Tham gia diễn đàn khóa học»*

Usecase	**Tham gia diễn đàn khóa học**
Nguồn	S-US03, T-US03, L-US04
Use-case ID	UC-CL-06
Tổng quan	Người dùng (sinh viên, giảng viên, trợ giảng) tham gia trao đổi thảo luận trên diễn đàn khóa học.
Actor	User
Tiền điều kiện	• Người dùng đã đăng nhập. • Có quyền truy cập diễn đàn của khóa học.
Điều kiện kích hoạt	Người dùng truy cập và tham gia diễn đàn khóa học.
Các bước	1. Xem các chủ đề và bài viết hiện có. 2. Đăng bài mới hoặc trả lời các bài viết. 3. Hệ thống lưu bài viết và cập nhật diễn đàn.
Hậu điều kiện	• Nội dung thảo luận được cập nhật, hỗ trợ trao đổi hiệu quả.
Xử lý ngoại lệ	ex1. Lỗi khi lưu bài viết, hệ thống thông báo và đề nghị thử lại. ex2. Người dùng không có quyền truy cập, hệ thống từ chối và thông báo. ex3. Nội dung vi phạm quy định, hệ thống cảnh báo hoặc khóa bài.

5 Phân tích & Thiết kế hệ thống

5.1 Entity Relationship Diagram (ERD)

5.1.1 Tổng quan



Hình 13: *Entity Relationship Diagram*

Tổng quan các Thực thể

Sơ đồ có tổng cộng 16 thực thể được chia thành 4 lĩnh vực (Domain): Quiz (Bài kiểm tra), Course Modification (Điều chỉnh Khóa học), Course Learning (Học tập Khóa học), và System (Hệ thống).

Course Modification, Quiz và Course Learning chia hệ thống thành 2 mảng tách rời nhưng tương trợ lẫn nhau: chỉnh sửa nội dung khóa học và sử dụng khóa học.

- **Domain: Quiz (Bài kiểm tra)**
  Bao gồm các thực thể phục vụ định nghĩa các bài quiz, và ghi lại tương tác của Student trên các bài quiz này.
  - **Quiz**: Đại diện cho một bài kiểm tra hoặc bài tập tổng thể.
  - **Question**: Đại diện cho một câu hỏi cụ thể trong Quiz.
  - **Answer**: Đại diện cho một lựa chọn trả lời khả dụng cho một Question.
  - **Quiz Attempt**: Đại diện cho một lần làm Quiz cụ thể của một Student.
- **Domain: Course Modification (Chỉnh sửa Khóa học)**
  Bao gồm các thực thể phục vụ định nghĩa các khóa học, phân chia một khóa học thành các đơn vị nhỏ, dễ truy xuất, tham chiếu.
  - **Course**: Đại diện cho một khóa học hoàn chỉnh.
  - **Chapter**: Đại diện cho một chương (bài học lớn) trong Course.
  - **Topic**: Đại diện cho một chủ đề nhỏ hơn nằm trong Chapter.
  - **Section**: Đại diện cho một phần nội dung, chứa các Quiz hoặc Topic.
- **Domain: Course Learning (Học Khóa học)**
  Bao gồm các thực thể phục vụ các tương tác lên khóa học và quiz.
  - **Forum message**: Đại diện cho một bài viết/bình luận trong diễn đàn.

- **Domain: System (Hệ thống)**
  Bao gồm các thực thể phục vụ việc định danh người dùng, phân quyền, điều khiển và vận hành cho Admin.
  - **User**: Đại diện cho người dùng chung của hệ thống (thực thể cha).
  - **Admin / Dean / Lecturer / TA**: Đại diện cho người dùng có vai trò quản trị/giảng dạy, là thực thể con của User.
  - **Student**: Đại diện cho người dùng có vai trò là học viên, là thực thể con của User.
  - **File**: Đại diện cho các tài liệu được Upload lên hệ thống.
  - **System config**: Đại diện cho các cài đặt cấu hình của toàn bộ hệ thống.
  - **Privilege**: Đại diện cho các quyền hạn cụ thể của User.
  - **Log**: Đại diện cho các bản ghi nhật ký hoạt động của User.

Các mối quan hệ giữa các thực thể

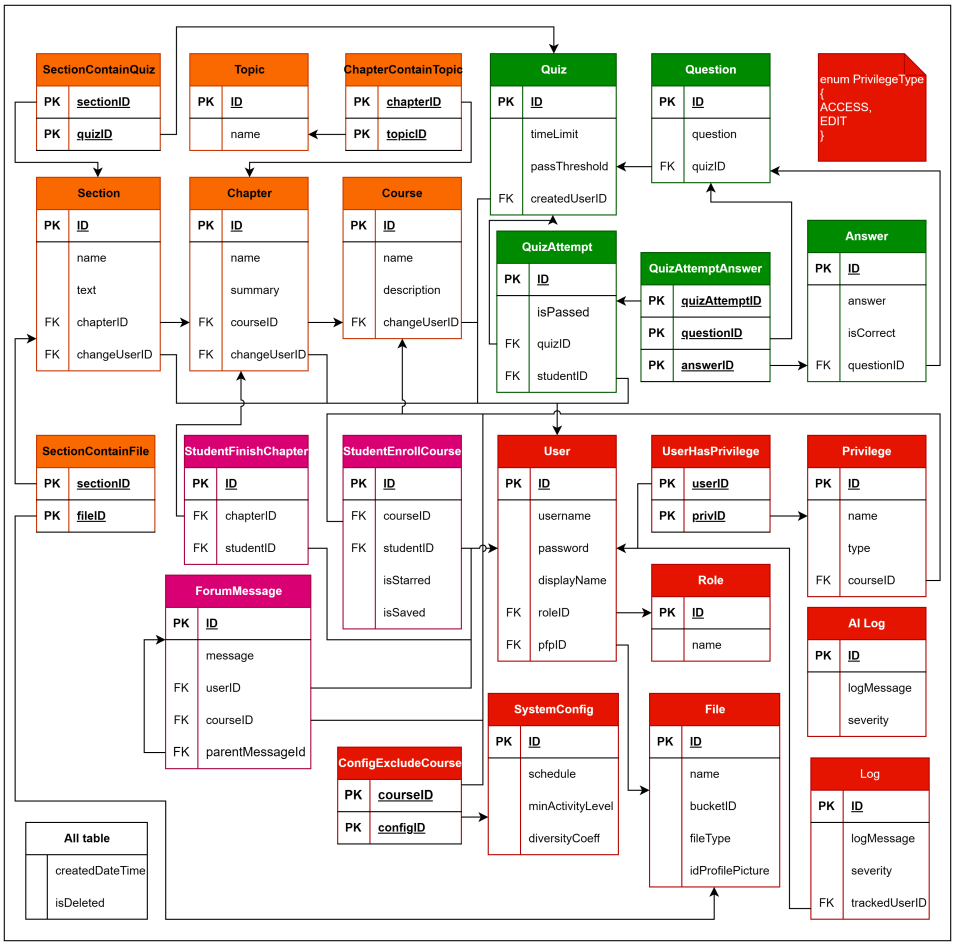
Domain	Thực thể 1	Tên Mối Quan hệ	Thực thể 2	Kiểu Mối Quan hệ (Cardinality)	Yêu cầu liên quan
Quiz	Quiz	Contain	Question	1:N - Một Quiz có nhiều Question	L-US02, T-US02, NFR07, NFR15
Quiz	Question	Contain	Answer	1:N - Một Question có nhiều Answer	
Quiz	Quiz Attempt	Of	Quiz	N:1 - Nhiều QuizAttempt thuộc về một Quiz	
Quiz	Student	Take	Quiz Attempt	1:N - Một Student thực hiện nhiều bài Quiz	
Quiz	Admin / Dean / Lecturer / TA	Change	Quiz	1:N - Một người dùng có quyền hạn có thể chỉnh sửa nhiều bài quiz	
Course Mod.	Course	Contain	Chapter	1:N - Một Course có nhiều Chapter	
Course Mod.	Chapter	Contain	Section	1:N - Một Chapter có nhiều Section	
Course Mod.	Chapter	Contain	Topic	1:N - Một Chapter có nhiều Topic	
Course Mod.	Section	Contain	File	1:N - Một Section có nhiều file (tài liệu, video,...)	
Course Mod.	Section	Contain	Quiz	1:N - Một Section có nhiều Quiz	
Course Mod.	Admin / Dean / Lecturer / TA	Change	Course	1:N - Một người dùng có quyền hạn chỉnh sửa nhiều Course	
Course Mod.	Admin / Dean / Lecturer / TA	Change	Chapter	1:N - Một người dùng có quyền hạn chỉnh sửa nhiều Chapter	
Course Mod.	Admin / Dean / Lecturer / TA	Change	Section	1:N - Một người dùng có quyền hạn chỉnh sửa nhiều Section	
Course Learn.	Student	Finish	Chapter	N:N - Nhiều Student hoàn thành nhiều Chapter	
Course Learn.	Student	Enroll	Course	N:N - Nhiều Student đăng ký nhiều Course	
Course Learn.	User	Post	Forum message	1:N - Một User đăng tải nhiều Forum message	
Course Learn.	User	Post	Forum message	1:N - Một User đăng tải nhiều Forum message	

Course Learn.	Course	Contain	Forum message	1:N - Một Course có nhiều Forum message	
System	User	Upload	File	1:1 - Một User đăng tải một File (ảnh đại diện)	
System	User	Has	Privilege	N:N - Nhiều User có nhiều Privilege	
System	Privilege	Grant access to	Course	N:1 - Nhiều Privilege cho phép truy cập một Course (2 dạng chỉ xem hoặc chỉnh sửa)	
System	Log	Track	User	N:1 - Nhiều Log theo dõi một User - để truy vấn trách nhiệm khi có sửa đổi	
System	Admin / Dean / Lecturer / TA	Change	System config	N:N - Nhiều người dùng có quyền hạn thay đổi nhiều System config	
System	System config	Exclude	Course	1:N - Một System config chứa thông tin về bỏ qua nhiều Course (hỗ trợ AI Service - use-case UC-SM-01)	

Mối Quan hệ Kế thừa (Inheritance/Specialization): Mối quan hệ này được biểu thị bằng ký hiệu vòng tròn **D** (Disjoint), nghĩa là người dùng Student sẽ không thể có được các quyền hạn ngang hàng với Admin, Lecturer, TA:

Mối Quan hệ bậc 3: Mối quan hệ Contain N-N-N giữa Quiz Attempt, Question và Answer có vai trò lưu giữ câu trả lời của Student cho mỗi câu hỏi tương ứng trong một lần làm quiz.

5.1.2 Mapping



Hình 14: *Relational Mapping Diagram*

Triển khai từ ERD ở phần trước, sơ đồ mapping bổ sung thêm các trường dữ liệu cho các bảng để hoàn thiện thiết kế cấu trúc dữ liệu.

Một số chi tiết đáng lưu ý:

- Bảng AI Log chỉ dùng để lưu trữ các thông tin hoạt động của AI service, không có quan hệ với bảng nào khác.
- Tất cả các bảng đều có các trường createDateTime - thời điểm record đó được tạo ra, và isDeleted - hỗ trợ tính năng soft-delete
- Privilege được chia làm 2 loại (type): ACCESS - chỉ truy cập và đọc nội dung, dành cho Student, EDIT - truy cập và chỉnh sửa nội dung, dành cho Lecturer, TA.

5.2 Lựa chọn kiến trúc hệ thống

5.2.1 Đặc tính kiến trúc (Architecture characteristics)

Ứng dụng phương pháp **Architecture Katas** lên các yêu cầu hệ thống ở bảng 1 và 2, nhóm đề xuất ra 3 đặc tính chính của hệ thống:

1. **Configurability** , vì:
    - **Quản lý hệ thống:** Cho phép thay đổi cách thức vận hành của hệ thống mà không cần can thiệp hoặc chỉnh sửa trực tiếp vào mã nguồn. Đặc tính này bao gồm việc *Đối với mô hình Trí tuệ nhân tạo (AI)* thì có khả năng thiết lập *chu kỳ tự động bảo trì*. Tương ứng với: NFR01, NFR02.
    - **Chỉnh sửa khóa học:** Cho phép cấu hình *kiểu tài liệu* người dùng được phép đăng tải, cũng như chỉnh sửa *layout (bố cục) giao diện khóa học* hiển thị cho người dùng. Tương ứng với: NFR03, NFR04.

- **Tạo báo cáo:** Người dùng có thể *thay đổi layout* của báo cáo được tạo ra và *thay đổi phạm vi dữ liệu* được sử dụng để lập báo cáo. Tương ứng với: NFR05, NFR06.
- **Tạo câu đố bằng AI:** Cho phép cấu hình các tham số tạo câu hỏi như *số lượng câu*, *độ khó* mong muốn, và *phân bổ độ khó* giữa các câu hỏi. Tương ứng với: NFR07.
- **Hỏi gợi ý:** Có thể thiết lập *tần suất* hiển thị gợi ý và *mức độ chi tiết* của gợi ý. Tương ứng với: NFR08.
- **Đề xuất chủ đề liên quan:** Cho phép người dùng *thêm* các chủ đề quan tâm hoặc *bỏ* các chủ đề không quan tâm ra khỏi danh sách đề xuất.Tương ứng với: NFR09.

2. **Availability** : Đặc tính này tập trung vào việc đảm bảo hệ thống luôn có thể được truy cập và sử dụng, đặc biệt là trong các tình huống *thời gian sinh viên truy cập không ổn định* hoặc có sự tập trung người dùng lớn. Việc mất khả năng truy cập hệ thống sẽ ảnh hưởng đến chất lượng học tập của sinh viên - lớp người dùng luôn phải cân đối thời gian cho mọi việc, cũng như ảnh hưởng tiến độ xây dựng bài học của các giảng viên, trợ giảng,... Tương ứng với: NFR10, NFR11, NFR12.
3. **Data consistency + integrity** , vì:
   - Yêu cầu hệ thống phải *đảm bảo tính đúng đắn* của các tài liệu bài giảng được đăng tải. Tương ứng với: NFR13.
   - Yêu cầu *dữ liệu đăng tải phải được giữ an toàn* khỏi các truy cập trái phép hoặc mất mát. Tương ứng với: NFR14.
   - Đảm bảo *mọi dữ liệu hiển thị phải đồng bộ* và nhất quán giữa tất cả các người dùng đang sử dụng hệ thống. Tương ứng với: NFR15.
   - Các thao tác quan trọng phải đảm bảo tính toàn vẹn giao dịch (transactional integrity) và lịch sử chỉnh sửa nội dung khóa học phải được ghi lại để phục vụ truy vết. Tương ứng với: NFR16, NFR17

Các đặc tính phụ liên quan

Các đặc tính sau đây mô tả các yêu cầu hiệu năng và khả năng mở rộng của hệ thống:

- **Performance (Hiệu năng):** Yêu cầu *tối ưu hóa hiệu năng* của các mô hình AI để đảm bảo phản hồi nhanh chóng. Tương ứng với: NFR18, NFR19, NFR20.
- **Concurrency (Đồng thời):** Đảm bảo hệ thống có khả năng xử lý và phục vụ cho *nhiều người dùng cùng lúc* mà không gặp lỗi hoặc giảm chất lượng dịch vụ. Tương ứng với: NFR21, NFR22.
- **Elasticity (Đàn hồi/Co giãn):** Đảm bảo hệ thống có khả năng mở rộng hoặc thu hẹp tài nguyên để đối phó với *thời gian sử dụng không đều*, đặc biệt là khi tải trọng hệ thống *tập trung cao vào mùa thi*. Tương ứng với: NFR23, NFR24, NFR25.

5.2.2 Đánh giá sự đánh đổi

Performance và Security

Giữa hiệu năng và bảo mật luôn tồn tại sự đánh đổi rõ rệt. Các cơ chế bảo mật như mã hóa dữ liệu, xác thực token, phân quyền theo vai trò hay ghi nhận audit log đều đảm bảo an toàn cho thông tin học viên, nhưng đồng thời tạo thêm chi phí xử lý cho mỗi request. Điều này có thể ảnh hưởng trực tiếp đến yêu cầu phản hồi dưới 300ms theo NFR-01.

Scalability và Consistency

Khi hệ thống cần phục vụ lượng lớn học viên trong các khung giờ cao điểm, việc mở rộng theo chiều ngang là điều bắt buộc. Tuy nhiên, sự phân tán này khiến dữ liệu khó giữ được tính nhất quán tuyệt đối. Trong môi trường có nhiều instance dịch vụ và nhiều replica cơ sở dữ liệu, các thao tác cập nhật thường dẫn đến eventual consistency. Điều này không gây vấn đề lớn với các tác vụ như truy cập nội dung bài học, nhưng lại nguy hiểm đối với các thao tác quan trọng như nộp bài quiz, chấm điểm hoặc cập nhật tiến độ học tập, nơi yêu cầu strong consistency gần như bắt buộc.

Scalability và Performance

Khả năng mở rộng cũng tạo ra xung đột trực tiếp với hiệu năng. Khi hệ thống scale-out, mỗi request phải đi qua nhiều tầng hơn như load balancer, service mesh hoặc gateway; đồng thời dữ liệu được phân tán khiến việc truy cập trở nên tốn thời gian hơn. Điều này có thể làm tăng đáng kể độ trễ, đặc biệt với các thao tác được gọi thường xuyên như tải quiz hoặc lấy gợi ý từ AI engine.

Extensibility và Performance

Cuối cùng, khả năng mở rộng chức năng cũng tạo ra sự đánh đổi với hiệu năng. Khi hệ thống được chia nhỏ thành nhiều module hoặc microservices, các boundary tăng lên, kéo theo chi phí giao tiếp giữa các module — cả về network, serialization và kiểm soát phiên bản. Điều này khiến performance tổng thể có thể giảm so với kiến trúc monolithic.

5.2.3 Đề Xuất Các Phong Cách Kiến Trúc

Dựa trên việc phân tích các đặc tính kiến trúc cốt lõi (**Configurability**, **Availability**, **Data Consistency + Integrity**) và các đặc tính phụ (**Performance**, **Concurrency**, **Elasticity**), có thể đề xuất ba phong cách kiến trúc phù hợp để triển khai hệ thống ITS: **Modular Monolith**, **Microservices Architecture** và **Service-Based Architecture**. Mỗi phong cách đều có thế mạnh riêng trong việc đáp ứng các yêu cầu phi chức năng (NFRs), đồng thời cũng có những đánh đổi cần cân nhắc.

5.2.3.a Kiến trúc Modular Monolith

Kiến trúc Modular Monolith (*ModularMonolith*) phù hợp cho các hệ thống cần **Performance** cao và độ trễ thấp, vì toàn bộ logic vận hành trong cùng một tiến trình, không phải chịu chi phí truyền thông mạng giữa các module. Điều này đặc biệt hữu ích với các thao tác yêu cầu phản hồi dưới 300ms như tải nội dung bài học, nộp bài hoặc lấy gợi ý từ AI engine.

Kiến trúc này cũng hỗ trợ mức độ **Configurability** tốt nếu được tổ chức theo các domain rõ ràng (Domain-Driven Design). Mỗi module được cô lập theo chức năng như *LearningProgress*, *Content*, *Quiz*, *Recommendation*, giúp việc mở rộng hoặc cập nhật thuật toán mới trở nên dễ dàng mà không ảnh hưởng đến toàn bộ hệ thống.

Tuy nhiên, Modular Monolith bị hạn chế về **Elasticity** và **Concurrency**. Toàn bộ hệ thống chỉ có thể *scale* theo chiều ngang ở mức *deploy instance* toàn bộ ứng dụng, dẫn đến lãng phí tài nguyên khi một số module nóng (*hot modules*) như *Quiz* hoặc *AIrecommendation* bị quá tải nhưng các module khác vẫn nhàn rỗi. Ngoài ra, lỗi ở một module có thể ảnh hưởng đến toàn bộ ứng dụng, làm giảm mức độ **Availability** của hệ thống.

5.2.3.b Kiến trúc Microservices

Kiến trúc Microservices (*MicroservicesArchitecture*) giúp giải quyết triệt để các yêu cầu **Elasticity** và **Availability**. Mỗi dịch vụ có thể *scale* độc lập theo lượng truy cập thực tế. Trong các khung giờ cao điểm (7–10h tối), chỉ những *service* chịu tải nặng mới cần *scale-out*, tối ưu chi phí vận hành đồng thời đảm bảo hệ thống duy trì **Performance** ổn định.

Microservices cũng hỗ trợ **Availability** cao nhờ *isolation*. Khi một *service* gặp sự cố, các *service* khác vẫn có thể hoạt động bình thường, tránh tình trạng "crash toàn hệ thống" thường thấy ở *monolith*. Các cơ chế *retry*, *idempotent API* và *event-driven communication* cũng giúp đảm bảo **Data Consistency + Integrity** (không mất dữ liệu khi nộp bài hoặc cập nhật tiến độ).

Tuy nhiên, Microservices tạo ra nhiều đánh đổi liên quan đến **Performance**. Mỗi yêu cầu có thể phải đi qua API Gateway, *service mesh* hoặc nhiều *hop* mạng, làm tăng độ trễ. Đây là rủi ro đối với NFR yêu cầu phản hồi dưới 300ms. Đồng thời, độ phức tạp bảo mật (**Data Consistency + Integrity**: *zero-trust*, RBAC phân tán, *audit* nhiều *service*) cũng tăng lên đáng kể.

5.2.3.c Kiến trúc Service-Based

Kiến trúc Service-Based (*Service-Based Architecture*) là giải pháp trung gian giữa Modular Monolith và Microservices. Trong phong cách này, hệ thống được chia thành một số dịch vụ kích thước vừa (*coarse-grained services*) như *UserService*, *LearningService*, *QuizService* và *RecommendationService*. Mỗi *service* có thể triển khai độc lập nhưng kích thước lớn hơn *microservice*, giúp giảm bớt số lượng *service* và độ phức tạp tổng thể.

Phong cách này tạo sự cân bằng tốt giữa các đặc tính kiến trúc:

- Cải thiện **Elasticity** và **Concurrency** so với Modular Monolith vì có thể *scale* từng *service* độc lập.
- Giảm độ trễ (**Performance**) so với Microservices vì số lượng *hop* mạng ít hơn và hạn chế chồng chéo giao tiếp.
- Tăng **Availability** bằng cách cô lập lỗi ở cấp *service* nhưng không tạo quá nhiều *dependency* phân tán.
- Giữ mức **Configurability** tốt nhờ cấu trúc theo domain rõ ràng.

Service-Based Architecture cũng dễ bảo mật hơn (**Data Consistency + Integrity**) so với Microservices vì số lượng *entry points* và *communication paths* ít hơn, giúp triển khai các yêu cầu *Security* như RBAC, *audit logging* và *encryption* thuận lợi hơn.

Nhược điểm là khả năng *scale* mịn (*fine-grained scalability*) vẫn không bằng Microservices, và các *service* có thể trở nên "quá lớn" theo thời gian nếu không quản lý module hợp lý.

5.2.4 Lựa Chọn Phong Cách Kiến Trúc

Dựa trên việc phân tích các đặc tính kiến trúc cốt lõi (**Performance**, **Elasticity**, **Availability**) và các đặc tính quan trọng khác như **Data Consistency + Integrity** và **Configurability**, nhóm quyết định lựa chọn **Service-Based Architecture** làm phong cách kiến trúc chính cho hệ thống ITS.

Kiến trúc Service-Based được xem là lựa chọn phù hợp nhất vì nó mang lại sự cân bằng hợp lý giữa hiệu năng, khả năng mở rộng và độ phức tạp hệ thống. Với kiến trúc này, hệ thống được chia thành một số dịch vụ kích thước vừa, mỗi dịch vụ đại diện cho một domain quan trọng như *UserService*, *LearningService*, *QuizService*, *CourseService* và *AIService*. Các dịch vụ này chạy độc lập nhưng không bị chia nhỏ quá mức như *microservices*, giảm đáng kể chi phí giao tiếp mạng và phức tạp vận hành.

Trước yêu cầu **Performance** (phản hồi dưới 300ms), kiến trúc Service-Based có khả năng đảm bảo độ trễ thấp nhờ số lượng *service hop* hạn chế. Điều này giúp hệ thống dễ đạt được mục tiêu phản hồi dưới 300ms, đồng thời vẫn duy trì logic module hoá theo domain như ở Modular Monolith.

Về **Elasticity** và **Concurrency**, Service-Based đáp ứng tốt khi cho phép *scale-out* độc lập từng *service* theo nhu cầu thực tế. Các dịch vụ có lưu lượng cao trong giờ cao điểm — đặc biệt là *QuizService* và *RecommendationService* — có thể mở rộng theo chiều ngang mà không ảnh hưởng những phần ít sử dụng hơn. Kiến trúc này không tạo ra hàng chục dịch vụ nhỏ như *microservices*, giúp chi phí *scaling* hợp lý và dễ kiểm soát hơn.

Đối với **Availability**, Service-Based cung cấp mức độ cô lập lỗi tốt hơn *monolith*. Một lỗi xảy ra trong *QuizService* không làm toàn bộ hệ thống ngừng hoạt động, từ đó hỗ trợ mục tiêu SLA 99.9%. Kiến trúc cũng hỗ trợ các kỹ thuật như *retry*, *idempotent update* và *event-driven synchronization* để đảm bảo **Data Consistency + Integrity** (không mất dữ liệu khi nộp bài hoặc cập nhật tiến độ học tập).

Tuy **Data Consistency + Integrity** (Bảo mật) không phải đặc tính cốt lõi quyết định cấu trúc, nhưng Service-Based Architecture giúp triển khai các yêu cầu bảo mật dễ dàng hơn so với *microservices* nhờ số lượng điểm giao tiếp và đường truyền ít hơn.

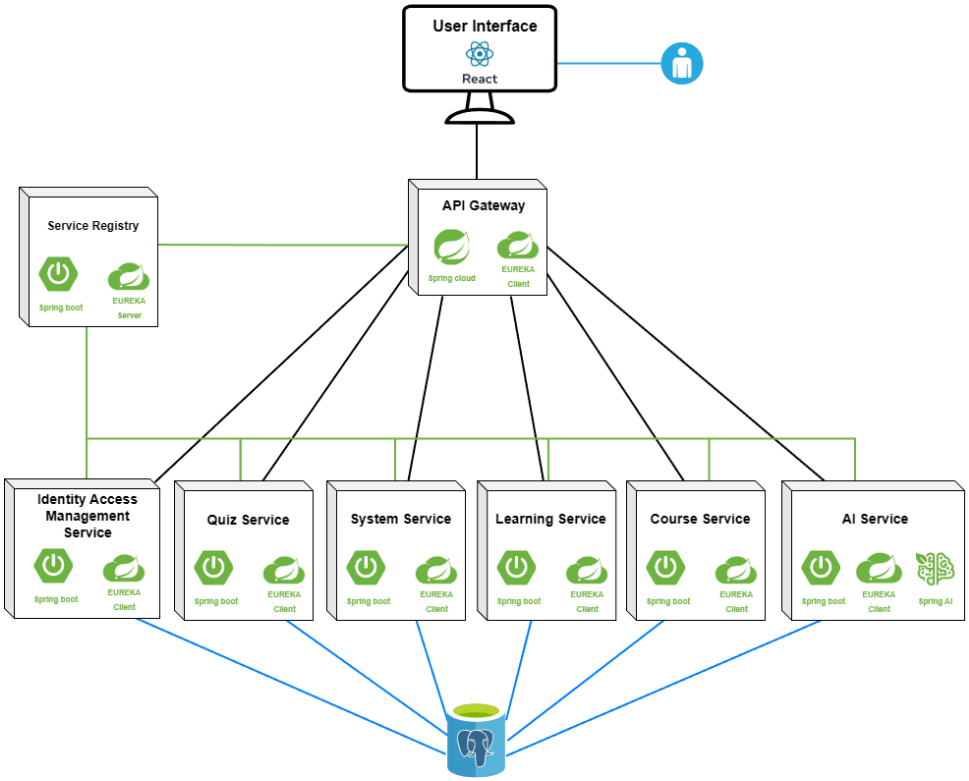
Cuối cùng, Service-Based cũng đáp ứng tốt yêu cầu **Configurability**. Khi cần bổ sung module nội dung, thêm thuật toán gợi ý mới hoặc tích hợp với hệ thống LMS/Payment Gateway bên ngoài, đội ngũ chỉ cần mở rộng hoặc thêm mới các *service* mà không ảnh hưởng đến cấu trúc tổng thể.

Tóm lại, Service-Based Architecture cung cấp sự cân bằng tối ưu giữa hiệu năng, khả năng mở rộng, độ sẵn sàng và khả năng tiến hóa hệ thống, đồng thời có độ phức tạp phù hợp với năng lực triển khai. Vì vậy, đây là phong cách kiến trúc phù hợp nhất với các NFR và đặc tính đã phân tích, và được nhóm lựa chọn làm nền tảng cho thiết kế kiến trúc của hệ thống ITS.

Chi tiết xem mục 8.2.

5.3 Thiết kế kiến trúc

5.3.1 Module view



Hình 15: *Module view - Nguyên tắc tổ chức*

1. **Nguyên tắc Tổ chức Module**

   (a) Hệ thống được thiết kế dựa trên **Service-Based Architecture (SBA)** nhằm tối ưu hóa tính module hóa và đảm bảo **phân tách mối quan tâm (separation of concerns)**.

   (b) Cấu trúc Module View chi tiết hóa cách thức mã nguồn hệ thống được tổ chức thành 9 module, tuân thủ **Nguyên tắc Trách nhiệm Đơn nhất (Single Responsibility Principle)**.

2. **Mô tả Tổng quát các Module**

   Các module được phân loại thành hai nhóm chính: Dịch vụ Nghiệp vụ Miền (Domain Services) và Dịch vụ Hỗ trợ Hạ tầng và Truy cập (Infrastructure & Access Services).

   (a) **Dịch vụ Nghiệp vụ Miền (Domain Services)**
   Gồm 6 module cốt lõi, mỗi module tập trung quản lý một miền nghiệp vụ chuyên biệt:

Module	Mục đích	Trách nhiệm chính
IAM Service	Quản lý Danh tính, Truy cập và Thông tin người dùng.	Thực hiện xác thực và phân quyền; quản lý hồ sơ, quyền hạn, vai trò và vòng đời người dùng.
Quiz Service	Quản lý mọi hoạt động liên quan tới bài kiểm tra.	Tạo, chỉnh sửa, lưu trữ cấu trúc câu hỏi và bài quiz; chấm điểm và lưu trữ phiên làm bài.
System Service	Cung cấp các chức năng hỗ trợ chung mang tính nền tảng.	Cung cấp cơ chế logging cho các thao tác bảo mật; quản lý các tài nguyên media (video, document, ảnh).
Learning Service	Quản lý hành trình học tập và tiến trình của người dùng.	Theo dõi tiến trình học (completion, progress); quản lý các forum/thảo luận của mỗi khóa học.

Tiếp theo trên trang sau

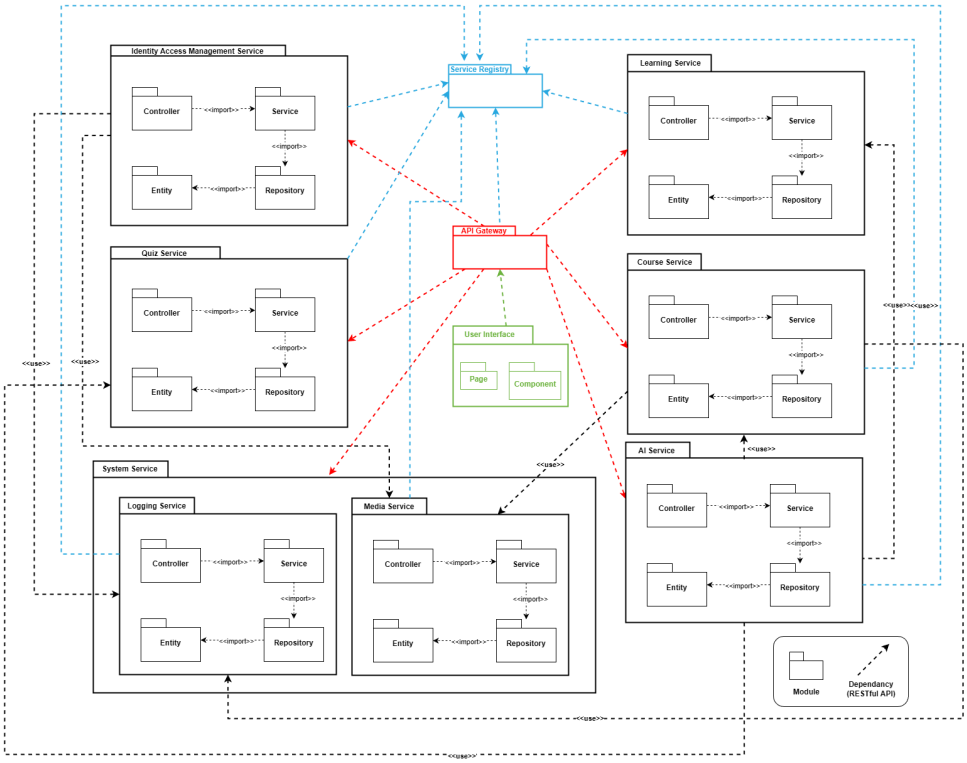
Bảng Domain Services tiếp theo

Module	Mục đích	Trách nhiệm chính
Course Service	Quản lý vòng đời của khóa học.	Quản lý metadata, trạng thái khóa học; xử lý quy trình đăng ký học viên và các quy tắc truy cập.
AI Service	Hỗ trợ học viên thông qua việc áp dụng Trí tuệ Nhân tạo.	Phân tích hành vi và tiến trình học; đề xuất nội dung, tài liệu, hoặc lộ trình học tập phù hợp.

(b) **Dịch vụ Hỗ trợ Hạ tầng và Truy cập (Infrastructure & Access Services)**
Các module thiết yếu trong việc quản lý kết nối, giao tiếp và cấu hình:

- **API Gateway:** Đóng vai trò là Điểm truy cập duy nhất (Single Entry Point) cho client. Trách nhiệm chính là Định tuyến yêu cầu tới dịch vụ tương ứng.
- **Service Registry:** Cung cấp cơ chế Tìm kiếm Dịch vụ (Service Discovery). Trách nhiệm chính là Lưu trữ metadata và danh sách các instance đang chạy của các dịch vụ.
- **User Interface (UI):** Là điểm tương tác cuối cùng với người dùng. Trách nhiệm chính là Hiển thị nội dung, thu thập input, gọi các API qua API Gateway và xử lý kết quả.

3. **Tổ chức, phân lớp, phụ thuộc giữa các Module**



Hình 16: *Module view - Package diagram*

(a) **Nguyên tắc Phân lớp Nội bộ:** Mỗi module dịch vụ đều tuân theo **Layered Architecture** nội bộ thống nhất (Controller → Service → Repository → Entity) để phân tách rõ ràng trách nhiệm.

(b) **Luồng Truy cập:** UI → API Gateway → Domain Services. Gateway đóng vai trò là **Facade**, bảo vệ các dịch vụ nghiệp vụ.

(c) **Phụ Thuộc giữa các Module (Inter-Module Dependencies):** Các module giao tiếp qua REST API theo hai kiểu:

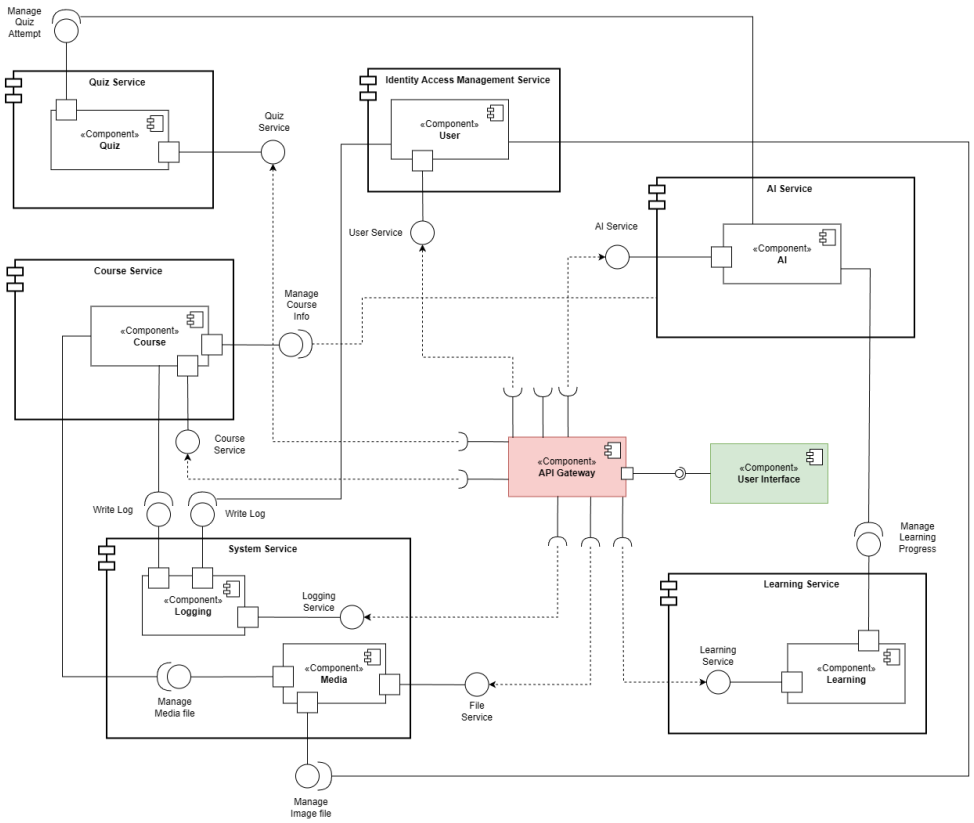
- **Phụ thuộc qua API Gateway («use»):** Tất cả Domain Services đều có mối quan hệ «use» với API Gateway (Gateway là thành phần duy nhất được phép gọi các dịch vụ này).
- **Phụ thuộc giữa các Domain Services (S2S):**
  - IAM Service và Course Service đều «use» **Media Service** và **Logging Service** (trong System Service).
  - AI Service «use» **Learning Service, Course Service, Quiz Service** để lấy dữ liệu tiến trình học và điểm số.

(d) **Phụ thuộc vào Service Registry:** Tất cả Domain Service và API Gateway phụ thuộc vào Service Registry theo hai chiều:

- **Đăng ký (Registration):**
  - Mỗi Domain Service (IAM, Learning, Quiz, Course, System, AI) khi khởi động sẽ tự động đăng ký các instance của mình bao gồm địa chỉ IP, cổng và các metadata cần thiết.
  - Service Registry duy trì danh sách các instance hợp lệ dựa trên cơ chế Heartbeat/Healthcheck, TTL (Time-To-Live), và thực hiện deregister khi một service không còn hoạt động. Việc đăng ký là bắt buộc, vì nếu một service không được đăng ký, các service khác sẽ không thể phát hiện và gọi đến nó.
- **Tra cứu (Discovery):**
  - Các thành phần cần gọi tới dịch vụ khác sẽ thực hiện tra cứu thông qua Service Registry để lấy thông tin địa chỉ runtime của các service. Ví dụ, API Gateway khi nhận request từ client sẽ query Service Registry để xác định instance phù hợp của service tương ứng dựa trên tên logic.
  - Tương tự, trong các kịch bản service-to-service, một service khi muốn gọi service khác (ví dụ AI tới Learning) cũng phải tham khảo Registry để lấy endpoint mới nhất.

Cơ chế này cho phép giao tiếp giữa các service dựa trên tên service thay vì địa chỉ IP hoặc cổng cứng, từ đó đảm bảo loose coupling giữa các thành phần trong hệ thống.

5.3.2 Component & Connector View



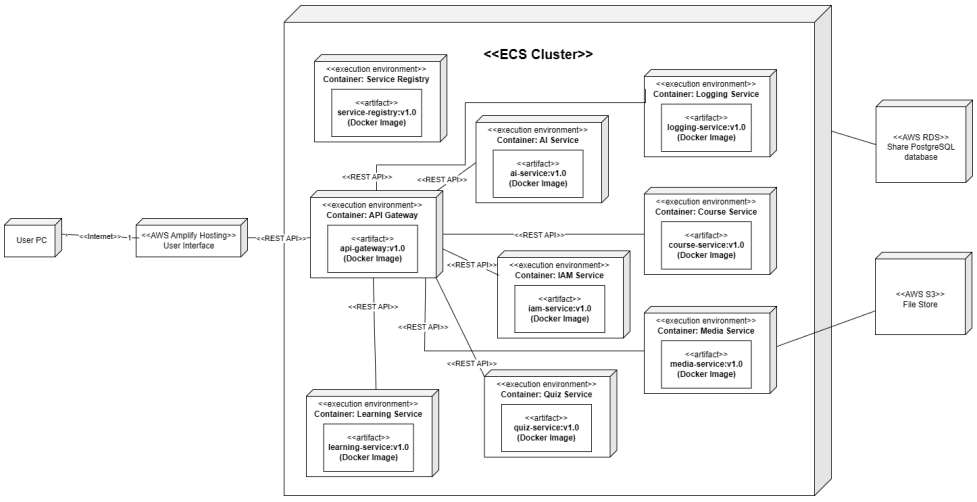
Hình 17: *Component & Connector view*

1. **Identity & Access Management (IAM) Service**
   - **User Component:** Xử lý toàn bộ tác vụ liên quan đến danh tính và truy cập người dùng: đăng ký, đăng nhập, xác thực (authentication), quản lý phiên, cấp token và quản lý hồ sơ người dùng.
   - **Interface:** `User Service` — cung cấp cho `API Gateway` để thực hiện xác thực và truy vấn thông tin người dùng.
2. **Quiz Service**

- **Quiz Component:** Quản lý toàn bộ nghiệp vụ về bài kiểm tra: tạo quiz, quản lý câu hỏi, thực hiện quiz, chấm điểm tự động và lưu kết quả. Tích hợp với Learning Service để cung cấp dữ liệu đánh giá.
- **Manage Quiz Attempt (Interface):** Cung cấp cho `AI Service` để lấy thông tin về kết quả các lần làm quiz của học viên trong khóa học, từ đó đưa ra gợi ý lộ trình học.
- **Interface:** `Quiz Service` — cho phép `API Gateway` gửi request về quiz và kết quả đánh giá.

3. **Course Service**
   - **Course Component:** Quản lý nội dung khóa học: tạo khóa học, chương, bài học; quản lý nội dung đa phương tiện; thiết lập điều kiện tiên quyết và quyền truy cập. Tương tác với Media Component để lưu trữ tài nguyên.
   - **Manage Course Info (Interface):** Cung cấp cho `AI Service` để lấy thông tin về khóa học của học viên.
   - **Interface:** `Course Service` — dùng bởi `API Gateway` để thao tác dữ liệu khóa học.
4. **Learning Service**
   - **Learning Component:** Theo dõi tiến độ học tập: ghi nhận trạng thái hoàn thành, thời gian học, điểm số, tính phần trăm tiến độ và lưu lịch sử học. Cung cấp dữ liệu cho AI Service để phân tích và gợi ý.
   - **Manage Learning Progress (Interface):** Cung cấp cho `AI Service` để lấy thông tin về quá trình học của học viên.
   - **Interface:** `Learning Service` — cho `API Gateway` truy vấn tiến độ và lịch sử học tập.
5. **AI Service**
   - **AI Component:** Phân tích dữ liệu từ Learning, Course và Quiz để xây dựng mô hình học tập cá nhân hóa; đề xuất lộ trình học, khóa học phù hợp và gợi ý nội dung thông minh.
   - **Interface:** `AI Service` — cung cấp cho `API Gateway` để lấy đề xuất cá nhân hóa.
6. **System Service**
   - **Logging Component:** Thu thập và lưu trữ log từ mọi service; phân loại log; hỗ trợ phân tích và giám sát hệ thống.
     - **Write Log (Interface):** Cung cấp cho `IAM Service` và `Course Service` để ghi log lại các thao tác về xác thực, đăng ký người dùng và tạo, chỉnh sửa khóa học.
     - **Interface:** `Logging Service` — cung cấp cho `API Gateway` để xem log.
   - **Media Component:** Xử lý tài nguyên đa phương tiện: upload, lưu trữ, tối ưu hóa và phân phối nội dung media. Hỗ trợ Course và IAM (ảnh đại diện).
     - **Manage media file, Manage image file (Interfaces):** Cung cấp cho `IAM Service` và `Course Service` để thao tác với các file media như ảnh đại diện, video bài giảng.
     - **Interface:** `File Service` — cho `API Gateway` thao tác với file media.
7. **API Gateway Component**
   - **Chức năng:** Thực hiện định tuyến request tới đúng service, xác thực/ủy quyền dựa trên IAM, cân bằng tải và áp dụng chính sách bảo mật tập trung.
8. **User Interface Component**
   - **Chức năng:** Tầng giao diện người dùng: hiển thị nội dung khóa học, làm quiz, xem tiến độ, nhận gợi ý từ AI và quản lý hồ sơ cá nhân.
   - **Giao tiếp:** Giao tiếp với backend thông qua `API Gateway`.

5.3.3 Allocation view



Hình 18: *Component & Connector view*

1. Tổng quan mô hình triển khai hệ thống

   Hệ thống được triển khai theo mô hình service-based architecture trên nền tảng container hóa, trong đó toàn bộ các service backend được triển khai tập trung trong một ECS Cluster thuộc Amazon Web Services. Người dùng truy cập hệ thống từ máy cá nhân thông qua Internet, giao diện người dùng được host độc lập trên AWS Amplify. Tất cả các yêu cầu từ phía frontend đều được chuyển tiếp về backend thông qua API Gateway, đóng vai trò là điểm vào duy nhất của hệ thống. Mô hình này giúp tách biệt rõ ràng giữa tầng giao diện và tầng xử lý nghiệp vụ, đồng thời tăng cường tính bảo mật và khả năng kiểm soát truy cập.

2. Kiến trúc service-based trong ECS Cluster

   Trong ECS Cluster, mỗi service được triển khai dưới dạng một container độc lập, tương ứng với một Docker image (artifact). Hệ thống không sử dụng kiến trúc microservices mà áp dụng mô hình service-based, trong đó các service vẫn tách biệt về mặt triển khai, chức năng và vòng đời, nhưng chia sẻ cùng một cơ sở dữ liệu trung tâm. Các service chính bao gồm: API Gateway, Service Registry, IAM Service, AI Service, Course Service, Learning Service, Quiz Service, Media Service và Logging Service. Mỗi service tồn tại như một môi trường thực thi riêng biệt (execution environment) và giao tiếp với các service khác thông qua REST API trên mạng nội bộ của ECS.

3. Cơ chế định tuyến và cân bằng tải với Spring Cloud Gateway và Eureka

   API Gateway trong mô hình này được hiện thực bằng Spring Cloud Gateway và tích hợp với Eureka từ Service Registry. Thông qua cơ chế service discovery, API Gateway có khả năng tự động phát hiện các instance của service, thực hiện cân bằng tải (load balancing), kiểm tra trạng thái (health check) và định tuyến request đến đúng service còn hoạt động. Điều này giúp hệ thống đạt được tính chịu lỗi cao (fault tolerance) và khả năng mở rộng động (dynamic scalability) khi số lượng instance của từng service thay đổi.

4. Service Registry và cơ chế phát hiện dịch vụ

   Service Registry được triển khai như một container riêng trong ECS Cluster và đóng vai trò trung tâm trong việc đăng ký và tra cứu service. Khi mỗi service khởi động, nó sẽ tự động đăng ký thông tin (tên service, địa chỉ, cổng, trạng thái) với Registry. API Gateway dựa trên thông tin này để định tuyến request một cách linh hoạt mà không cần cấu hình cứng địa chỉ từng service. Kiến trúc này giúp hệ thống tránh phụ thuộc tĩnh, đồng thời đặc biệt phù hợp với môi trường container có khả năng scale linh động theo tải.

5. Kiến trúc tầng dữ liệu và lưu trữ

   Về tầng dữ liệu, hệ thống sử dụng một PostgreSQL Database dùng chung được triển khai dưới dạng dịch vụ managed của Amazon RDS for PostgreSQL. Tất cả các service nghiệp vụ, bao gồm IAM Service, AI Service, Course Service, Learning Service, Quiz Service và Logging Service, đều kết nối trực tiếp tới cơ sở dữ liệu này để đọc và ghi dữ liệu. Riêng Media Service không lưu trữ dữ liệu file trong database mà chỉ sử dụng database để quản lý metadata, còn toàn bộ dữ liệu media vật lý được lưu trữ trên Amazon S3. Việc tách riêng dữ liệu quan hệ và dữ liệu file giúp tối ưu hiệu năng truy xuất, chi phí lưu trữ và khả năng mở rộng hệ thống.

6. Quy trình triển khai hệ thống và CI/CD với GitHub Actions và CloudFormation

   Quy trình triển khai của hệ thống được tự động hóa theo mô hình CI/CD. Mỗi service được build từ mã nguồn và đóng gói thành một Docker image với phiên bản tương ứng, chẳng hạn như api-gateway:v1.0, course-service:v1.0, media-service:v1.0. Khi push source code lên repository trên GitHub, GitHub Actions sẽ tự động kích hoạt pipeline CI để thực hiện các bước: kiểm tra mã nguồn, build Docker image và đẩy image lên Amazon Elastic Container Registry (ECR).

   Sau khi image được lưu trữ trên ECR, quá trình triển khai hạ tầng và cập nhật service được thực hiện thông qua AWS CloudFormation. CloudFormation chịu trách nhiệm khởi tạo và cập nhật các tài nguyên như ECS Cluster, Task Definition, Service, Load Balancer và các cấu hình mạng liên quan. ECS sẽ tự động pull các image mới nhất từ ECR để khởi tạo hoặc cập nhật container. Khi container khởi động, các service sẽ kết nối tới Service Registry để đăng ký, đồng thời thiết lập kết nối tới PostgreSQL Database, riêng Media Service thiết lập thêm kết nối tới S3. Sau khi toàn bộ service sẵn sàng, API Gateway bắt đầu tiếp nhận request từ frontend và điều phối xử lý trong nội bộ hệ thống. Quy trình này cho phép triển khai tự động, lặp lại, giảm sai sót thủ công và rút ngắn thời gian release.

7. Luồng xử lý tổng thể trong môi trường triển khai

   Luồng xử lý tổng thể của hệ thống diễn ra theo trình tự: người dùng thao tác trên giao diện web được host trên AWS Amplify, request được gửi qua Internet tới API Gateway trong ECS Cluster. API Gateway tiến hành xác thực thông qua IAM Service, sau đó dựa trên Service Registry để định tuyến request tới Course Service, Learning Service, Quiz Service, AI Service hoặc Media Service tùy theo nghiệp vụ. Các service này xử lý logic chuyên biệt và truy cập trực tiếp vào PostgreSQL Database để lấy hoặc cập nhật dữ liệu. Trong trường hợp xử lý file, Media Service sẽ thực hiện lưu trữ và truy xuất nội dung từ S3. Toàn bộ hoạt động của hệ thống được Logging Service ghi nhận và lưu trữ phục vụ công tác giám sát, theo dõi và phân tích hệ thống.

6 Hiện thực

Đường dẫn xem các diagram: https://drive.google.com/file/d/1uBsjoXb8VRKfoT2vgGVCYIc1MjsF7qjq/view?usp=sharing

Đường dẫn repo Github: https://github.com/SpyBurner/KTPM_HK251.git

6.1 API Gateway

Ba lớp CustomJwtDecoder, JwtAuthenticationEntryPoint và SecurityConfig hợp tác với nhau để xây dựng pipeline xác thực JWT trong API Gateway. Mỗi lớp đảm nhiệm một vai trò riêng biệt nhưng phối hợp chặt chẽ để tạo thành cơ chế bảo mật thống nhất.

1. Lớp CustomJwtDecoder

   Lớp CustomJwtDecoder hiện thực interface ReactiveJwtDecoder, đóng vai trò là bộ giải mã JWT tùy chỉnh. Lớp này chứa ba thuộc tính chính: khóa ký (signerKey), thuật toán mã hóa (algorithm) và đối tượng giải mã Nimbus (nimbusJwtDecoder). Các thuộc tính này cung cấp thông tin cấu hình cần thiết để phân tích và xác thực token.

   Phương thức trung tâm decode(String) thực hiện toàn bộ quá trình xác thực JWT. Khi nhận token đầu vào, phương thức này giải mã token, kiểm tra thời hạn hiệu lực và xác minh các claims bắt buộc. Kết quả trả về là đối tượng Jwt hợp lệ, được Spring Security sử dụng để tạo authentication context cho request.

   Để hỗ trợ phương thức decode, lớp cung cấp các phương thức bổ trợ: getDecoder() khởi tạo đối tượng NimbusJwtDecoder với cấu hình đúng thuật toán và khóa ký; buildDecoder() tạo decoder thực tế dựa trên thuật toán được chọn; validateTokenExpiration(Jwt) kiểm tra token còn hiệu lực hay đã hết hạn; và validateTokenClaims(Jwt) xác minh token chứa đầy đủ thông tin định danh yêu cầu.

2. Lớp JwtAuthenticationEntryPoint

   Lớp JwtAuthenticationEntryPoint hiện thực interface ServerAuthenticationEntryPoint, đảm nhận vai trò xử lý lỗi xác thực. Lớp này sử dụng ObjectMapper cùng các thuộc tính cấu hình khác để tạo ra phản hồi lỗi chuẩn hóa.

   Phương thức commence(ServerWebExchange, AuthenticationException) được Spring Security tự động gọi khi phát hiện request không hợp lệ hoặc thiếu token xác thực. Phương thức này xây dựng phản hồi JSON chứa mã lỗi, thông điệp mô tả và HTTP status code tương ứng. Cơ chế này đảm bảo mọi lỗi xác thực đều được trả về theo định dạng thống nhất, giúp client dễ dàng xử lý và hiển thị thông báo lỗi cho người dùng.

3. Lớp SecurityConfig

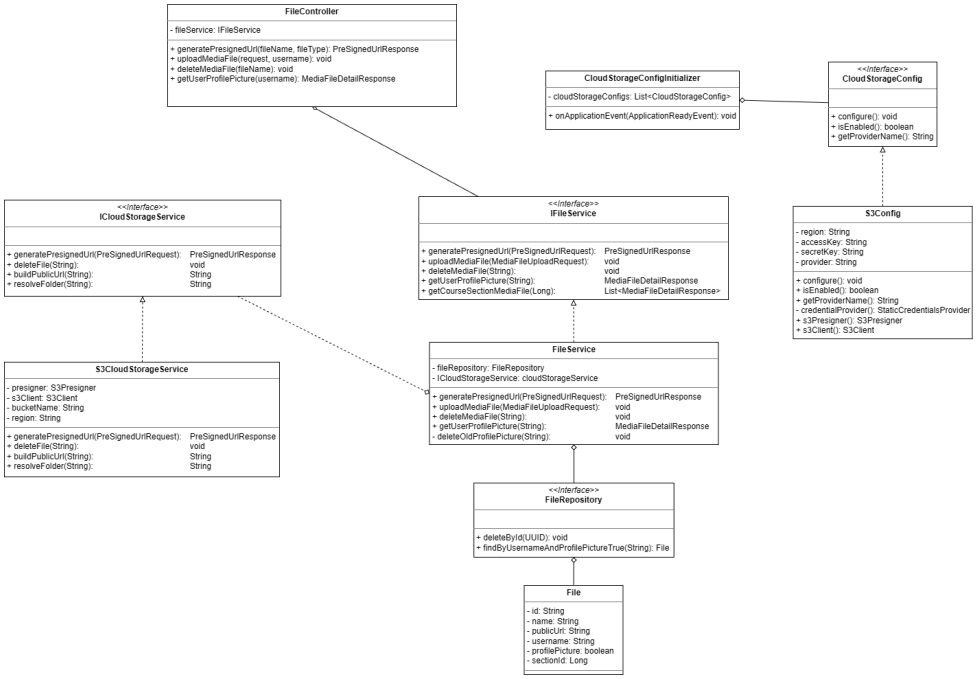
   Lớp SecurityConfig định nghĩa toàn bộ tác vụ bảo mật cho API Gateway. Lớp này khai báo các thuộc tính cấu hình quan trọng như danh sách endpoint công khai (PUBLIC_ENDPOINTS), danh sách nguồn gốc được phép truy cập CORS (allowedOrigins), và hai dependency là ReactiveJwtDecoder và ServerAuthenticationEntryPoint. Lớp này có mối quan hệ dependency với hai interface này, cho phép Spring Container inject các implementation tương ứng thông qua constructor injection.

   Phương thức springSecurityFilterChain(ServerHttpSecurity) là thành phần cốt lõi, thiết lập quy tắc phân quyền truy cập, tích hợp bộ giải mã JWT tùy chỉnh, và cấu hình authentication entry point xử lý lỗi. Phương thức corsConfigurationSource() tạo cấu hình CORS, cho phép API Gateway xử lý request từ các frontend domain được chỉ định một cách an toàn. Phương thức jwtAuthenticationConverter() định nghĩa cơ chế chuyển đổi authorities từ JWT claims thành các đối tượng GrantedAuthority mà Spring Security sử dụng để kiểm tra phân quyền.

4. Mối quan hệ giữa các lớp

   SecurityConfig có mối quan hệ aggregation với hai interface ReactiveJwtDecoder và ServerAuthenticationEntryPoint bởi vì nó sử dụng hai interface này như một phần của nó để thực hiện các tác vụ bảo mật của nó và vòng đời của hai interface này sẽ không phụ thuộc vào Security Config. CustomJwtDecoder hiện thực ReactiveJwtDecoder và sử dụng. JwtAuthenticationEntryPoint hiện thực.

6.2 Class diagram cho Media Service



Hình 19: *Class diagram cho Media Service*

Media Service được thiết kế theo mô hình nhiều lớp (layered architecture), trong đó các thành phần được phân chia rõ ràng theo trách nhiệm: lớp điều khiển (controller), lớp dịch vụ nghiệp vụ (service), lớp truy xuất dữ liệu (repository), và lớp tích hợp dịch vụ lưu trữ đám mây (cloud storage integration). Sơ đồ lớp (Class Diagram) được xây dựng nhằm mô tả mối quan hệ và vai trò của từng thành phần như sau:

1. FileController:

   FileController là lớp chịu trách nhiệm tiếp nhận mọi yêu cầu từ API Gateway. Mỗi endpoint được ánh xạ vào một phương thức tương ứng, sau đó chuyển tiếp tới tầng dịch vụ để xử lý nghiệp vụ. Lớp này có quan hệ aggregation với interface IFileService, cho phép sử dụng các chức năng nghiệp vụ mà không phụ thuộc vào lớp hiện thực cụ thể. FileController đảm nhiệm việc sinh Pre-Signed URL để frontend trực tiếp upload file lên dịch vụ lưu trữ, gửi yêu cầu lưu metadata sau khi file được tải lên, thực hiện xóa file media dựa trên ID hoặc key, truy xuất ảnh đại diện của người dùng thông qua username và lấy danh sách media thuộc về một Course Section.

2. IFileService – Interface xử lý nghiệp vụ:

   IFileService định nghĩa toàn bộ hành vi nghiệp vụ liên quan tới media file. Mục tiêu của interface là tạo ra lớp trừu tượng giúp Controller không phải phụ thuộc vào logic hiện thực. Các phương thức bao gồm:

   - Hoạt động sinh Pre-Signed URL thông qua hàm generatePresignedUrl(PreSignedUrlRequest) để frontend có thể upload trực tiếp lên S3;
   - Lưu metadata file upload với uploadMediaFile(MediaFileUploadRequest);
   - Xóa mediaFile thông qua deleteMediaFile(String fileIdOrKey);
   - Truy xuất ảnh đại diện của người dùng bằng getUserProfilePicture(String username);
   - Danh sách file thuộc một khóa học hoặc một section thông qua getCourseSectionMediaFile(Long sectionId).

3. FileService – Lớp hiện thực của IFileService:

   FileService hiện thực toàn bộ logic nghiệp vụ mô tả trong IFileService. Lớp này có quan hệ aggregation với FileRepository để làm việc với dữ liệu trong database và với ICloudStorageService để tương tác với dịch vụ lưu trữ đám mây. FileService thực hiện điều phối giữa hai tầng này, đảm bảo dữ liệu metadata và file lưu trữ trên file storage trên cloud được đồng bộ hóa. Ngoài ra, FileService còn chịu trách nhiệm thực hiện các kiểm tra nghiệp vụ như xác định folder lưu trữ dựa trên ngữ cảnh, thực hiện validate dữ liệu upload để đảm bảo đúng định dạng mong muốn, cũng như xử lý việc xóa file cũ khi một file mới được cập nhật.

4. FileRepository – Tầng truy xuất dữ liệu:

   FileRepository đóng vai trò là lớp giao tiếp với database theo chuẩn Spring Data. Interface này cung cấp các phương thức quan trọng phục vụ Media Service, chẳng hạn như deleteById(String id) để xóa metadata file dựa vào ID, và findByUsernameAndProfilePictureTrue(String username) để tìm file đóng vai trò là ảnh đại diện của một người dùng. Repository đảm bảo việc truy xuất và lưu trữ dữ liệu diễn ra nhất quán và tách biệt khỏi logic nghiệp vụ.

5. ICloudStorageService – Lớp trừu tượng hóa dịch vụ lưu trữ đám mây:

   ICloudStorageService là lớp trừu tượng giúp Media Service có thể làm việc với nhiều provider như AWS S3, Google Cloud Storage hoặc Azure Blob mà không cần thay đổi logic nghiệp vụ. Interface này định nghĩa các phương thức cần thiết cho mọi hệ thống lưu trữ, chẳng hạn như sinh Pre-Signed URL thông qua generatePresignedUrl(...), xóa file bằng deleteFile(String key), xây dựng đường dẫn public URL với buildPublicUrl(String key), và xác định folder lưu trữ thích hợp dựa trên ngữ cảnh nghiệp vụ thông qua resolveFolder(String context).

6. S3CloudStorageService – Hiện thực cho AWS S3:

   S3CloudStorageService là lớp hiện thực cụ thể của ICloudStorageService dành cho AWS S3. Toàn bộ logic làm việc với AWS SDK được đóng gói tại đây. Lớp này sử dụng S3Presigner để sinh Pre-Signed URL an toàn phục vụ upload; sử dụng S3Client để gọi các API thao tác trực tiếp với bucket; quản lý thông tin cấu hình như tên bucket và region. Bên cạnh việc sinh URL upload/download, lớp này còn hỗ trợ xóa file khỏi S3, xây dựng public URL dựa trên object key và xác định folder lưu trữ theo từng ngữ cảnh nghiệp vụ nhằm đảm bảo file được tổ chức theo cấu trúc nhất quán

7. CloudStorageConfigInitializer:

   CloudStorageConfigInitializer là lớp chịu trách nhiệm khởi tạo toàn bộ cấu hình của các cloud storage provider ngay khi ứng dụng bắt đầu chạy. Các thuộc tính và phương thức bao gồm:

   - cloudStorageConfigs: Danh sách chứa tất cả các cấu hình của từng provider (S3, GCP Storage, Azure Blob...). Mỗi phần tử trong danh sách là một cấu hình độc lập, tuân theo interface CloudStorageConfig.
   - onApplicationEvent(ApplicationReadyEvent): Phương thức này được gọi khi ứng dụng khởi động xong. Lúc này CloudStorageConfigInitializer sẽ duyệt qua danh sách cloudStorageConfigs và gọi configure() trên từng config. Việc khởi tạo tập trung giúp đảm bảo mọi cloud provider đều sẵn sàng trước khi bất kỳ service nào trong hệ thống sử dụng chúng.

8. CloudStorageConfig:

   CloudStorageConfig là một interface định nghĩa hợp đồng cho tất cả các loại cấu hình của các cloud storage provider.

   Interface này đảm bảo rằng mọi provider đều tuân theo cùng một bộ hành vi, bao gồm:

   - configure(): Thiết lập và chuẩn bị toàn bộ tài nguyên cần thiết để provider hoạt động (client, credential, presigner...).
   - isEnabled(): Cho biết config có đang được bật bởi hệ thống hay không (đọc từ application.yml).
   - Điều này cho phép bật/tắt provider linh hoạt theo môi trường.
   - getProviderName(): Trả về tên của provider (ví dụ: "AWS_S3" hoặc "GCP_STORAGE").
   - Hỗ trợ logging, debugging, hoặc chọn provider theo tên.

   Interface này giúp hệ thống dễ dàng mở rộng thêm provider mới bằng cách chỉ cần tạo class mới implement đúng hợp đồng này.

9. S3Config:

   S3Config là lớp triển khai cụ thể của CloudStorageConfig dành cho AWS S3. Lớp này chịu trách nhiệm chuẩn bị toàn bộ thông tin cấu hình, client và credential để kết nối tới dịch vụ S3. Các thuộc tính liên quan đến cấu hình AWS S3, chẳng hạn như accessKey, secretKey, region, bucketName, presignerExpiration và các tham số khác cần thiết để hoạt động.

   Phương thức:

   - configure(): Hàm khởi tạo toàn bộ tài nguyên cần thiết cho AWS S3 như credential provider, S3Client và S3Presigner. Sau khi hàm này chạy, các service trong hệ thống có thể tương tác với S3 một cách ổn định.

- isEnabled(): Kiểm tra xem S3 có được bật trong cấu hình hệ thống hay không (ví dụ cloud.aws.s3.enabled=true). Giúp linh hoạt bật/tắt từng provider.
- getProviderName(): Trả về tên provider: "AWS_S3".
- credentialProvider(): Tạo provider chứa access key và secret key, phục vụ cho việc xác thực với AWS. Trả về StaticCredentialsProvider đảm bảo credential luôn sẵn sàng cho các client.
- s3Presigner(): Tạo đối tượng presigner phục vụ việc tạo URL tải lên/tải xuống có thời hạn.
- Đây là thành phần quan trọng trong việc xử lý pre-signed URL.
- s3Client(): Tạo instance S3Client để thực thi các thao tác như upload, delete, copy hoặc kiểm tra object. Client này được cấu hình đúng region, credential và các tuỳ chọn liên quan.

Giải thích tuân theo nguyên tắc SOLID:

1. Single Responsibility Principle (SRP) – Nguyên tắc trách nhiệm đơn

   Thiết kế của Media Service tuân thủ SRP bằng cách phân tách rõ ràng trách nhiệm của từng lớp trong hệ thống. Mỗi lớp chỉ đảm nhiệm một vai trò duy nhất và có một lý do để thay đổi.

   - FileController chỉ chịu trách nhiệm tiếp nhận và xử lý HTTP request từ API Gateway. Tất cả nghiệp vụ liên quan đến media đều được chuyển xuống tầng service, giúp Controller sạch, dễ bảo trì và không bị trộn lẫn logic.
   - FileService giữ vai trò duy nhất là xử lý logic nghiệp vụ của media. Mọi quyết định như xác định thư mục lưu trữ, validate thông tin upload, đồng bộ metadata hay xử lý xoá file đều được tập trung tại đây.
   - FileRepository đảm nhiệm việc tương tác với cơ sở dữ liệu — truy vấn, lưu, xoá metadata file. Repository không can thiệp vào nghiệp vụ hay xử lý đối với cloud storage.
   - ICloudStorageService và S3CloudStorageService tập trung hoàn toàn vào nhiệm vụ tương tác với hệ thống lưu trữ đám mây: sinh Pre-Signed URL, xoá file trên cloud, hoặc xây dựng public URL.
   - CloudStorageConfigInitializer: Chỉ chịu trách nhiệm khởi tạo tất cả cloud provider khi ứng dụng bắt đầu chạy. Nó không xử lý nghiệp vụ, không thao tác file.
   - CloudStorageConfig: Chỉ mô tả hợp đồng cấu hình cho từng provider. Interface này chỉ tập trung vào cấu hình – không tham gia xử lý logic hay lưu trữ.

   Nhờ sự phân tách này, khi logic liên quan đến request thay đổi, chỉ Controller bị tác động; khi nghiệp vụ thay đổi chỉ Service bị ảnh hưởng; và khi thay đổi provider lưu trữ cloud thì chỉ cần thay đổi lớp implementation tương ứng. Điều này giúp hệ thống dễ bảo trì, dễ mở rộng và giảm rủi ro phát sinh lỗi chéo giữa các tầng.

2. Open/Closed Principle (OCP) – Mở để mở rộng, đóng để sửa đổi

   Media Service được thiết kế để hỗ trợ mở rộng tính năng mà không cần sửa đổi các thành phần đã ổn định. Điều này thể hiện rõ qua việc sử dụng interface để định nghĩa hành vi cốt lõi.

   - IFileService cho phép bổ sung các service mới mà không cần thay đổi FileController. Nếu sau này mở rộng hệ thống để hỗ trợ xử lý video, tạo thumbnail, nén file..., ta có thể cung cấp thêm implementation hoặc decorator mà không động đến logic hiện tại.
   - ICloudStorageService cho phép thay đổi hoặc mở rộng provider lưu trữ (AWS S3, GCP Storage, Azure Blob...) chỉ bằng cách tạo thêm lớp mới như GCPCloudStorageService hoặc AzureCloudStorageService. FileService sẽ hoạt động bình thường mà không cần chỉnh sửa, do phụ thuộc vào abstraction chứ không phải implementation cụ thể.
   - CloudStorageConfig (interface) cho phép bổ sung thêm config mới (GCPConfig, AzureConfig...) mà không chỉnh sửa logic khởi tạo.

   Nhờ thiết kế đóng để sửa đổi và mở để mở rộng, Media Service có khả năng phát triển lâu dài mà không ảnh hưởng đến sự ổn định của hệ thống. Mỗi lần mở rộng chỉ cần bổ sung module mới thay vì chỉnh sửa module đang hoạt động.

3. Liskov Substitution Principle (LSP) – Nguyên tắc thay thế Liskov

   Hệ thống đảm bảo rằng mọi lớp con hoặc lớp hiện thực của một interface đều có thể thay thế cho class cha mà không làm thay đổi hành vi của toàn bộ Media Service.

   - S3CloudStorageService hoàn toàn có thể thay thế interface ICloudStorageService mà FileService đang sử dụng, và nếu triển khai thêm provider khác, implementation mới cũng phải đáp ứng đúng hợp đồng mà interface đưa ra.

-
- Tương tự, FileService phải tuân thủ đầy đủ hành vi được mô tả trong IFileService để Controller có thể sử dụng mà không cần biết implementation phía sau.

Nhờ đảm bảo đúng LSP, Media Service đạt được tính ổn định cao. Việc mở rộng hay thay đổi implementation không làm phá vỡ các tầng phía trên. Hệ thống luôn vận hành đúng dù sử dụng S3, GCP, Azure hay một storage provider mới.

4. Interface Segregation Principle (ISP) – Nguyên tắc phân tách interface

   Media Service áp dụng ISP thông qua việc tách interface theo đúng domain của chúng, tránh tạo ra các interface cồng kềnh buộc class phải implement những phương thức không liên quan.

   - IFileService chỉ tập trung vào nghiệp vụ file (sinh Pre-Signed URL, lưu metadata, xoá file, lấy avatar, lấy media theo section).
   - ICloudStorageService chỉ mô tả chức năng liên quan đến thao tác cloud storage (tạo presigned URL, xây public URL, xoá file).
   - FileRepository chỉ mô tả các truy vấn cần thiết trên bảng file.

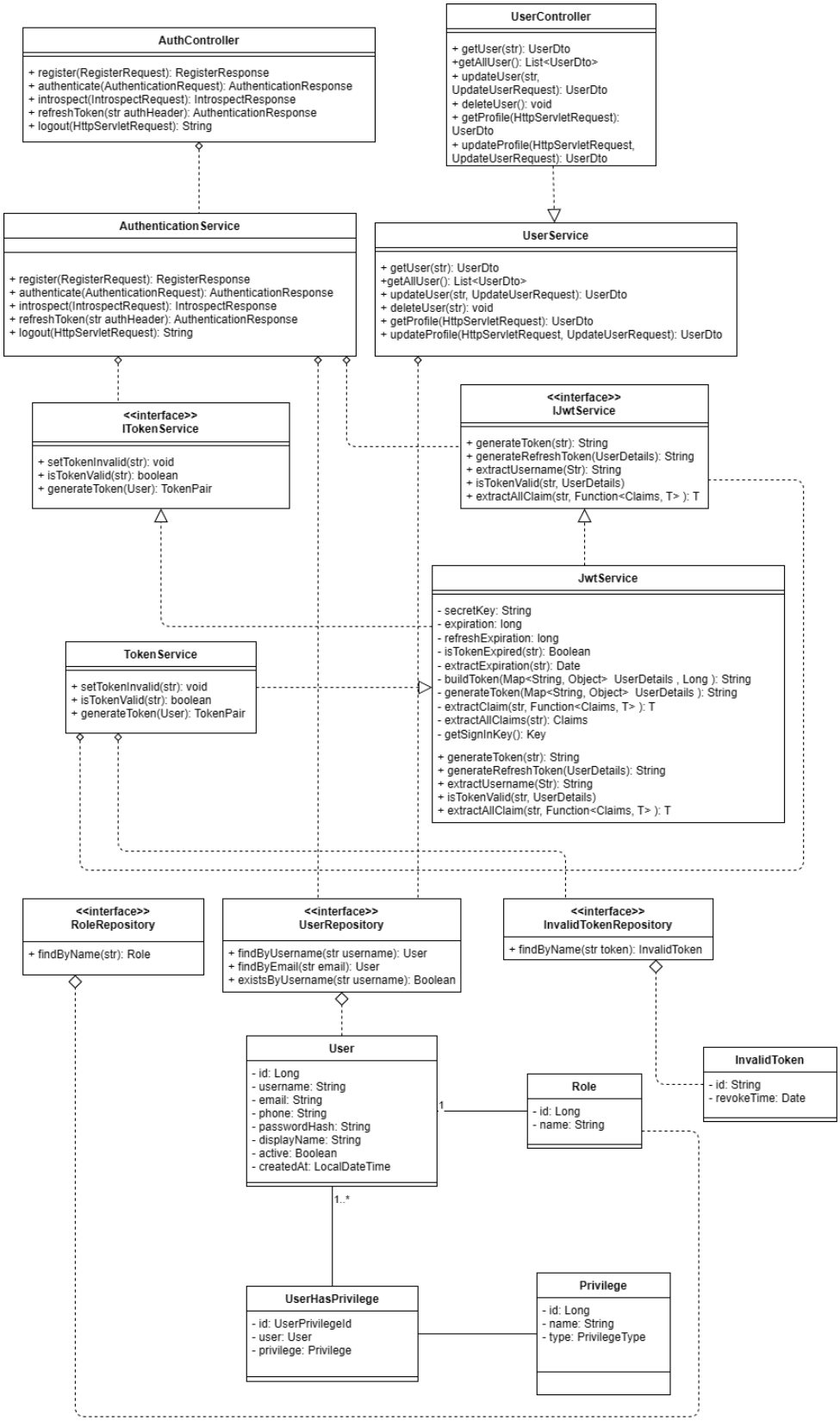
   Việc phân tách interface theo đúng chức năng giúp hệ thống rõ ràng hơn, module nào chỉ dùng tính năng gì thì chỉ phụ thuộc vào interface đó. Từ đó giảm sự phụ thuộc không cần thiết, tăng khả năng tái sử dụng code và đơn giản hoá việc viết unit test.

5. Dependency Inversion Principle (DIP) – Nguyên tắc đảo ngược phụ thuộc

   Các tầng cấp cao không phụ thuộc vào tầng cấp thấp, mà cả hai đều phụ thuộc vào abstraction.

   - FileController không phụ thuộc trực tiếp vào FileService, mà phụ thuộc vào interface IFileService. Điều này giúp Controller không bị ràng buộc vào một triển khai cụ thể.
   - FileService không phụ thuộc trực tiếp vào S3CloudStorageService, mà chỉ phụ thuộc vào abstraction ICloudStorageService. Nhờ đó, việc thay đổi provider cloud là hoàn toàn trong suốt đối với tầng service.
   - Repository cũng được sử dụng thông qua abstraction (do Spring Data tạo), giúp FileService hoàn toàn tách biệt khỏi công nghệ lưu trữ dữ liệu.
   - CloudStorageConfigInitializer Chỉ phụ thuộc vào danh sách CloudStorageConfig, không phụ thuộc vào S3Config cụ thể.

6.3 Class diagram cho Module quản lý tài khoản



Hình 20: *Class diagram cho Module quản lý tài khoản*

Identity Access Management (IAM) Service được thiết kế theo mô hình nhiều lớp (layered architecture), trong đó các thành phần được phân chia rõ ràng theo trách nhiệm: lớp điều khiển (controller), lớp dịch vụ nghiệp vụ (service), lớp giao diện xử lý token (token interface), và lớp truy xuất dữ liệu (repository). Sơ đồ lớp (Class Diagram) được xây dựng nhằm mô tả mối quan hệ và vai trò của từng thành phần trong việc xác thực và quản lý

người dùng như sau:

1. AuthController:

   AuthController là lớp chịu trách nhiệm tiếp nhận mọi yêu cầu liên quan đến xác thực và bảo mật từ API Gateway. Mỗi endpoint được ánh xạ vào một phương thức tương ứng như register, authenticate, introspect, refreshToken, và logout. Lớp này có quan hệ dependency với AuthenticationService, chuyển tiếp các yêu cầu tới tầng dịch vụ để xử lý logic. AuthController đóng vai trò là cổng giao tiếp chính, đảm bảo các yêu cầu HTTP được chuyển đổi thành dữ liệu nghiệp vụ để thực hiện đăng ký người dùng mới, xác minh danh tính, làm mới token khi hết hạn hoặc đăng xuất người dùng khỏi hệ thống.

2. AuthenticationService – Lớp dịch vụ xác thực trung tâm:

   AuthenticationService là trái tim của hệ thống IAM, chịu trách nhiệm điều phối toàn bộ luồng nghiệp vụ xác thực. Lớp này tương tác với UserRepository để kiểm tra thông tin định danh, RoleRepository để gán quyền hạn, và AuthenticationService cũng tích hợp với các dịch vụ xử lý token để cấp phát quyền truy cập. Các phương thức chính bao gồm authenticate để xác thực thông tin đăng nhập, register để tạo tài khoản mới và logout để thực hiện vô hiệu hóa phiên làm việc hiện tại.

3. ITokenService và IJwtService – Các Interface xử lý Token:

   Hai interface này định nghĩa toàn bộ hành vi liên quan đến việc quản lý vòng đời của token, tạo ra lớp trừu tượng giúp hệ thống không phụ thuộc chặt chẽ vào một thuật toán cụ thể.

   IJwtService tập trung vào các thao tác cấp thấp với JWT như generateToken, extractClaim, isTokenValid(phương thức này đối với Interface này chỉ kiểm tra token dựa trên thời gian hết hạn), và extractUsername.

   ITokenService mở rộng hoặc bao đóng các nghiệp vụ cao hơn như setTokenInvalid (đưa token vào danh sách đen) hay generateToken trả về đối tượng TokenPair (gồm Access và Refresh token), cũng như kiểm tra trạng thái của token đã bị thu hồi hay chưa thông qua phương thức isTokenInvalid. Khái niệm invalid ở interface này cho việc token bị thu hồi do hành động logout của người dùng trước khi token hết hạn.

4. TokenService và JwtService – Lớp hiện thực xử lý Token:

   TokenService và JwtService hiện thực hóa các logic được định nghĩa trong interface.

   JwtService chịu trách nhiệm về mặt kỹ thuật mật mã: ký token bằng secretKey, xác định thời gian hết hạn (expiration, refreshExpiration) và trích xuất thông tin từ chuỗi token.

   TokenService sử dụng các khả năng này để quản lý trạng thái token, bao gồm việc tương tác với InvalidTokenRepository để kiểm tra xem một token có nằm trong danh sách bị thu hồi (revoked) hay không thông qua phương thức isTokenInvalid và thu hồi token thông qua phương thức setTokenInvalid.

5. UserController và UserService – Quản lý thông tin người dùng:

   Cặp thành phần này quản lý các nghiệp vụ xoay quanh hồ sơ người dùng (User Profile).

   UserController cung cấp các API để lấy thông tin (getUser, getAllUser), cập nhật hồ sơ (updateProfile) hoặc xóa người dùng.

   UserService chứa logic nghiệp vụ để thực thi các yêu cầu này, thực hiện validate dữ liệu đầu vào và gọi xuống UserRepository để thao tác với cơ sở dữ liệu. Lớp này tách biệt hoàn toàn logic xử lý dữ liệu người dùng khỏi logic xác thực.

6. User, Role, Privilege – Các thực thể dữ liệu (Entities):

   Đây là các lớp đại diện cho cấu trúc dữ liệu trong hệ thống.

   User: Chứa thông tin định danh như username, email, passwordHash, và trạng thái hoạt động.

   Role: Định nghĩa vai trò của người dùng trong hệ thống.

   Privilege: Định nghĩa các quyền hạn cụ thể.

   Quan hệ giữa chúng được thể hiện qua UserHasPrivilege và liên kết trực tiếp giữa User và Role, tạo nên mô hình phân quyền linh hoạt.

7. Tầng Repository (UserRepository, RoleRepository, InvalidTokenRepository):

   Các interface này đóng vai trò là lớp giao tiếp với cơ sở dữ liệu (Data Access Layer).

   UserRepository: Cung cấp các phương thức tìm kiếm người dùng theo username hoặc email (findByUsername, findByEmail) và kiểm tra sự tồn tại.

   RoleRepository: Truy xuất thông tin vai trò dựa trên tên.

InvalidTokenRepository: Quản lý danh sách các token đã bị vô hiệu hóa (blacklist) để ngăn chặn việc tái sử dụng token sau khi đăng xuất (findById).

8. InvalidToken – Thực thể quản lý bảo mật:

   InvalidToken là thực thể lưu trữ các token đã bị thu hồi cùng với thời gian thu hồi (revokeTime). Thành phần này cực kỳ quan trọng trong cơ chế bảo mật Statefull hoặc Hybrid, giúp hệ thống từ chối các request mang token hợp lệ về mặt chữ ký nhưng đã bị người dùng chủ động hủy bỏ (Logout).

Giải thích tuân theo nguyên tắc SOLID:

1. Single Responsibility Principle (SRP) - Nguyên lý Đơn nhiệm

   Đánh giá: Thiết kế này tuân thủ SRP rất chặt chẽ thông qua việc phân chia các tầng (Layered Architecture) và tách nhỏ các Service:

   - Tách biệt Controller và Service:
     - AuthController và UserController chỉ chịu trách nhiệm tiếp nhận request (HTTP), validate cơ bản và trả về response.
     - Logic nghiệp vụ phức tạp được đẩy xuống AuthenticationService và UserService.
   - Tách biệt xử lý Token: Thay vì nhồi nhét logic xử lý chuỗi JWT vào AuthenticationService, thiết kế đã tách hẳn ra thành JwtService (lo các việc kỹ thuật như ký, giải mã, check hạn) và TokenService (lo các việc nghiệp vụ như tạo cặp token, blacklist token).
   - Tách biệt quản lý User: Các logic quản lý người dùng như việc xem thông tin và cập nhật thông tin người dùng được tách riêng biệt, không trộn lẫn với logic đăng nhập/đăng ký của AuthenticationService.

2. Open/Closed Principle (OCP) - Nguyên lý Đóng/Mở

   Đánh giá: Hệ thống có khả năng mở rộng mà không cần sửa đổi nhiều mã nguồn cũ nhờ vào việc sử dụng Interface:

   - Token Interface: Nếu hệ thống muốn thay đổi thuật toán ký JWT token (HS256 sang RS256) chỉ cần tạo một class mới implement ITokenService (hoặc implement cả IJwtService) mà không cần sửa đổi code trong AuthenticationService (nơi đang phụ thuộc vào các interface này).
   - Repository Pattern: Các UserRepository, RoleRepository là interface (chuẩn Spring Data JPA). Nếu muốn đổi database từ SQL sang MongoDB, chỉ cần đổi implement của Repository mà ít ảnh hưởng đến tầng Service.

3. Liskov Substitution Principle (LSP) - Nguyên lý thay thế Liskov

   Đánh giá: Trong sơ đồ này, quan hệ kế thừa chủ yếu nằm ở việc implements các Interface (TokenService implements ITokenService).

   - Mặc dù không thể nhìn thấy code chi tiết bên trong, nhưng về mặt thiết kế cấu trúc, TokenService cung cấp đầy đủ các phương thức mà ITokenService yêu cầu (generateToken, isTokenValid,...).
   - Miễn là TokenService không ném ra các ngoại lệ không mong muốn hoặc từ chối thực hiện hành vi đã định nghĩa trong Interface, nguyên lý này được đảm bảo.

4. Interface Segregation Principle (ISP) - Nguyên lý phân tách Interface

   Đánh giá: Thay vì tạo một Interface khổng lồ tên là ISecurityService chứa tất cả mọi thứ, thiết kế đã chia nhỏ:

   - IJwtService: Chỉ chứa các hàm kỹ thuật cấp thấp (extractClaim, buildToken, isTokenExpired). Dành cho các class cần thao tác trực tiếp với cấu trúc chuỗi token.
   - ITokenService: Chứa các hàm nghiệp vụ cao cấp hơn (setTokenInvalid, generateToken trả về TokenPair).
   - Lợi ích: AuthenticationService có thể chỉ cần inject ITokenService để lấy token trả về cho user mà không cần quan tâm đến các logic lấy quyền phức tạp của IJwtService. Client không bị phụ thuộc vào những hàm mà họ không dùng.

5. Dependency Inversion Principle (DIP) - Nguyên lý đảo ngược sự phụ thuộc

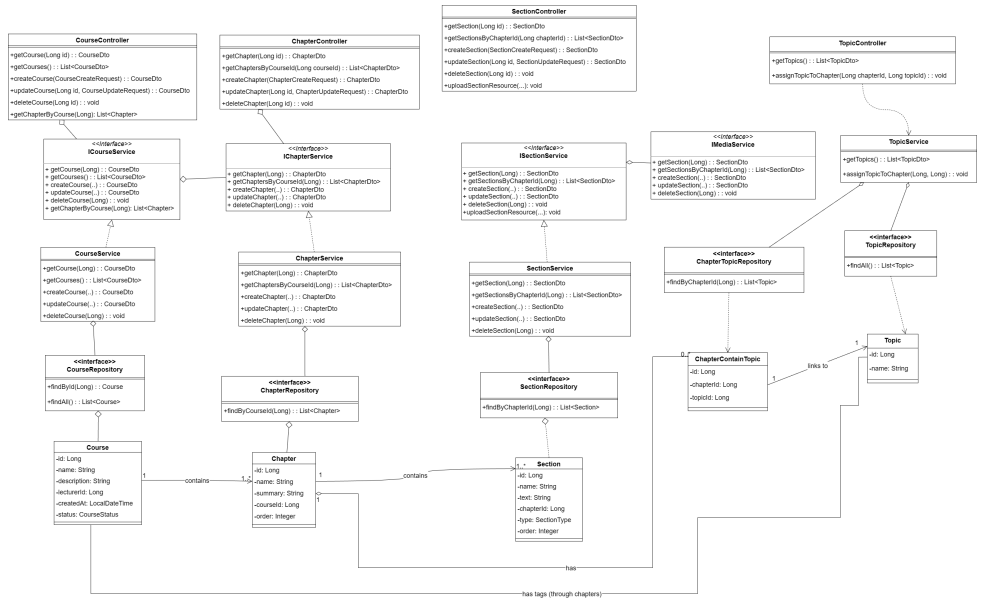
   Đánh giá: Thiết kế tuân thủ DIP thông qua việc các module cấp cao phụ thuộc vào abstraction (interface) thay vì module cấp thấp (concrete class).

   Service phụ thuộc Interface:

- AuthenticationService phụ thuộc vào ITokenService (Interface) chứ không phụ thuộc trực tiếp vào TokenService (Class cụ thể). Mũi tên dependency từ AuthenticationService trỏ vào ITokenService thể hiện điều này.
- Các Service phụ thuộc vào UserRepository, RoleRepository (đều là Interface).

Lưu ý: Giữa Controller và Service (AuthController -> AuthenticationService). Để đáp ứng tuyệt đối DIP, AuthenticationService và UserService cũng nên là Interface, và có các class AuthenticationServiceImpl, UserServiceImpl thực thi chúng. Tuy nhiên, trong thực tế với quy mô vừa phải, việc Controller gọi trực tiếp Service Class là chấp nhận được để giảm độ phức tạp không cần thiết (Pragmatic Architecture).

6.4 Class diagram cho Module quản lý khóa học



Hình 21: *Class diagram cho Module quản lý khóa học*

Course Service được thiết kế theo mô hình kiến trúc nhiều lớp (layered architecture), trong đó các thành phần được phân chia rõ ràng theo trách nhiệm: lớp điều khiển (controller), lớp dịch vụ nghiệp vụ (service), và lớp truy xuất dữ liệu (repository). Sơ đồ lớp (Class Diagram) được xây dựng nhằm mô tả mối quan hệ và vai trò của từng thành phần trong việc quản lý cấu trúc phân cấp của khóa học như sau:

1. Hệ thống Controller – Tầng giao tiếp (Course, Chapter, Section, Topic, Enrollment)

   Tầng Controller đóng vai trò là cửa ngõ tiếp nhận các yêu cầu HTTP từ Client. Thay vì dồn nén vào một Controller khổng lồ, hệ thống chia nhỏ thành các Controller chuyên biệt theo chức năng nghiệp vụ:

   - CourseController: Chỉ chịu trách nhiệm quản lý vòng đời cấp cao của thực thể Khóa học. Các endpoint ánh xạ vào các phương thức tạo mới, cập nhật, xóa và lấy thông tin chung của khóa học (getCourse, getCourses). Controller này đã được giải phóng khỏi nhiệm vụ truy xuất dữ liệu chi tiết bên trong (như danh sách chương).
   - ChapterController: Chuyên biệt quản lý thực thể Chương. Điểm cải tiến quan trọng trong thiết kế này là phương thức getChaptersByCourseId được đặt tại đây. Khi client cần danh sách chương của một khóa học, yêu cầu sẽ được gửi tới controller này, đảm bảo tính nhất quán trong quản lý tài nguyên chương.
   - SectionController: Quản lý đơn vị nhỏ nhất là Bài học. Tương tự, phương thức getSectionsByChapterId thuộc về controller này, chịu trách nhiệm truy xuất nội dung chi tiết (text, type) thuộc về một chương cụ thể.

   Các Controller này hoạt động như một lớp điều phối (Delegator), không chứa logic nghiệp vụ phức tạp mà chuyển tiếp yêu cầu xuống tầng Service tương ứng.

2. Tầng Service – Tầng xử lý nghiệp vụ

   Các lớp Service này là trái tim của hệ thống quản lý nội dung, chịu trách nhiệm điều phối toàn bộ luồng nghiệp vụ cốt lõi, đảm bảo tính toàn vẹn của cấu trúc dữ liệu phân cấp:

- CourseService: Đóng vai trò quản lý hồ sơ khóa học tổng quan. Nó tương tác với CourseRepository để thực hiện các tác vụ CRUD cơ bản.
- ChapterService: Chứa logic nghiệp vụ liên quan đến cấu trúc chương. Phương thức getChaptersByCourseId tại đây sẽ gọi xuống Repository để lấy dữ liệu, đảm bảo rằng việc truy xuất danh sách chương được xử lý độc lập với logic của khóa học.
- SectionService: Xử lý nghiệp vụ cho từng bài học chi tiết. Nó đảm bảo các bài học được tạo ra phải gắn liền với một chương hợp lệ (getSectionsByChapterId, createSection).

3. Tầng Repository – Tầng truy xuất dữ liệu

   Các interface này đóng vai trò là lớp giao tiếp với cơ sở dữ liệu (Data Access Layer). Thiết kế chia nhỏ Repository theo từng Entity giúp tối ưu hóa và chuyên biệt hóa truy vấn:

   - CourseRepository: Cung cấp các phương thức tìm kiếm cơ bản cho bảng Course.
   - ChapterRepository: Bổ sung phương thức truy vấn đặc thù findByCourseId, phục vụ trực tiếp cho nhu cầu lấy danh sách chương của ChapterService.
   - SectionRepository: Bổ sung phương thức findByChapterId, phục vụ cho nhu cầu lấy nội dung bài học của SectionService.

4. Các thực thể dữ liệu (Entities) & Mối quan hệ

   Mô hình dữ liệu phản ánh cấu trúc cây (Tree Structure) chặt chẽ của một khóa học trực tuyến:

   - Course (Gốc): Chứa thông tin tổng quan (tên, mô tả, giảng viên, trạng thái).
   - Chapter (Nhánh): Liên kết trực tiếp với Course qua courseId.
   - Section (Lá): Đơn vị nội dung cuối cùng, liên kết với Chapter qua chapterId.

Giải thích tuân theo nguyên tắc SOLID:

1. Nguyên lý Đơn nhiệm (Single Responsibility Principle – SRP)

   Mỗi Controller chỉ quản lý một loại tài nguyên duy nhất: CourseController xử lý các request liên quan đến khóa học, ChapterController chuyên quản lý chương, còn SectionController chịu trách nhiệm cho section. Không có controller nào "ôm" logic của domain khác. Tương tự, các lớp Service như CourseService, ChapterService hay SectionService chỉ chứa logic nghiệp vụ đúng với miền dữ liệu của mình. Những thao tác truy cập dữ liệu lại được tách xuống các Repository riêng: CourseRepository, ChapterRepository, SectionRepository.

2. Nguyên lý Đóng/Mở (Open–Closed Principle – OCP)

   ICourseService, IChapterService, ISectionService, IMediaService — cho phép hệ thống mở rộng mà không cần sửa code cũ. Các Controller chỉ biết đến interface, không biết đến class triển khai. Điều này tạo ra một ranh giới mở rộng rất an toàn để triển khai CourseServiceV2, thêm caching, refactor logic hoặc thay đổi thuật toán xử lý mà không cần chỉnh sửa CourseController.

   Mỗi lớp Controller vốn đã "đóng" với thay đổi nhưng "mở" cho mở rộng thông qua việc bind một implementation khác vào interface.

3. Nguyên lý Thay thế Liskov (LSP)

   Class diagram cho thấy các Service implementation đều tuân thủ hoàn toàn họ phương thức được định nghĩa trong interface. Điều này có nghĩa là bất kỳ implementation nào của ICourseService đều có thể thay thế cho một implementation khác trong Controller mà không làm thay đổi hành vi mong đợi.

   Ví dụ như nếu chia CourseService thành hai class khác nhau — CourseServiceNormal và CourseServiceWithAI — thì Controller vẫn hoạt động bình thường chỉ bằng cách thay đổi injection.

4. Nguyên lý Phân tách Interface (ISP)

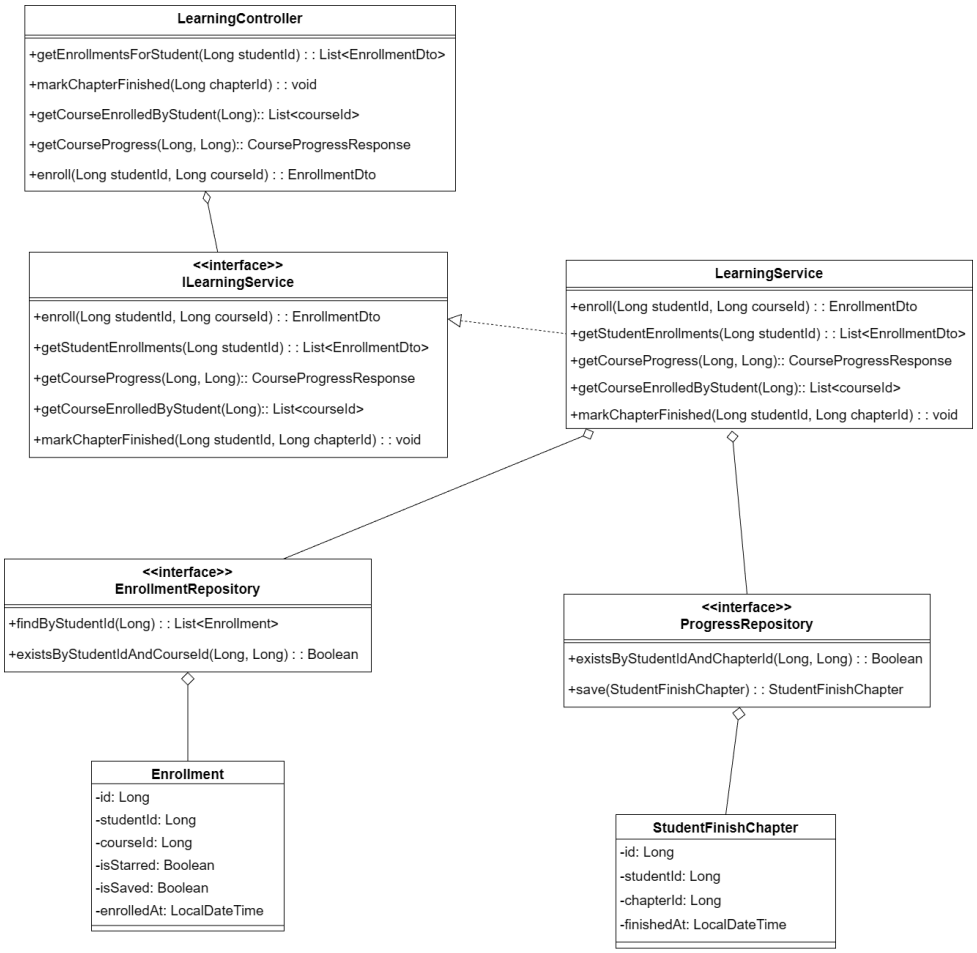
   Các interface trong thiết kế được chia nhỏ đúng theo từng miền nghiệp vụ: ICourseService, IChapterService, ISectionService, IMediaService. Mỗi interface chỉ chứa các chức năng phục vụ trực tiếp domain tương ứng, không có phương thức thừa hoặc ép buộc class phải implement những thứ không cần.

   Nhờ vậy, SectionService không bị buộc phải quan tâm đến logic của Course, và CourseService không phải chứa phương thức để uploadMedia hay xử lý chapter. Cách chia nhỏ interface này giúp code dễ hiểu hơn, dễ mock trong unit test, và đặc biệt phù hợp với mô hình microservice hoặc tách module về sau.

5. Nguyên lý Đảo ngược Phụ thuộc (Dependency Inversion Principle – DIP)

   Tất cả các phụ thuộc trong class diagram đều trỏ từ lớp cấp cao sang abstraction thay vì implementation. Controller phụ thuộc vào interface của Service; Service lại phụ thuộc vào interface của Repository; còn Repository được Spring Data tạo implementation tự động.

6.5 Class diagram cho Module học tập



Hình 22: *Class diagram cho Module học tập*

1. LearningController – Tầng giao tiếp xử lý hành vi học tập

   LearningController giữ vai trò như điểm vào duy nhất cho toàn bộ hoạt động học tập của sinh viên. Khác với các Controller quản lý nội dung (như Course, Chapter hoặc Section), Controller này chỉ tập trung vào việc tiếp nhận và xử lý các hành vi động của người dùng trong quá trình học. Khi sinh viên đăng ký một khóa học mới, Controller tiếp nhận yêu cầu và chuyển cho học viên một EnrollmentDto phản ánh trạng thái ghi danh hiện tại. Trong các phiên học tiếp theo, Controller cũng chịu trách nhiệm trả về danh sách khóa học đã ghi danh và tiến trình học tập của từng khóa thông qua các phương thức tương ứng. Giao diện học tập cần hiển thị theo thời gian thực mức độ hoàn thành của từng chương, và LearningController đáp ứng điều này bằng cách gọi xuống tầng service để xây dựng CourseProgressResponse một cách nhất quán.

   Ngoài ra, khi người học kết thúc một chương, Controller tiếp nhận tín hiệu hoàn thành và gửi yêu cầu xuống service để ghi nhận vào hệ thống tiến trình học tập. Nhờ đó, dữ liệu về tiến trình được cập nhật liên tục, phục vụ các tính năng khác như dashboard cá nhân hoặc gợi ý lộ trình học. Tất cả các nghiệp vụ này đều được triển khai theo nguyên tắc “thin controller”, tức là Controller chỉ làm nhiệm vụ điều phối và chuyển tiếp, nhằm giữ cho tầng giao tiếp luôn nhẹ và dễ duy trì.

2. LearningService – Tầng xử lý nghiệp vụ trọng tâm

   LearningService đóng vai trò là tầng xử lý nghiệp vụ cốt lõi của toàn bộ module học tập, nơi tập hợp và điều phối mọi hoạt động liên quan đến hành trình học của sinh viên. Khi người dùng đăng ký một khóa học, service này kiểm tra trước trong EnrollmentRepository để đảm bảo rằng khóa học chưa được ghi danh trước đó. Việc ngăn chặn trùng lặp này đặc biệt quan trọng vì dữ liệu Enrollment là một trong những nền tảng quan trọng nhất của hệ thống, ảnh hưởng đến tiến trình học, hiển thị nội dung và các thuật toán gợi ý.

   Trong quá trình học tập, LearningService cũng chịu trách nhiệm quản lý toàn bộ thông tin về tiến độ học tập. Khi giao diện yêu cầu tiến trình của một khóa học, service sẽ tổng hợp dữ liệu từ nhiều nguồn: số chương trong khóa, số chương đã hoàn thành, cùng với trạng thái chi tiết của từng chương. Nhờ đó, service này hoạt

động như một bộ "orchestrator", kết hợp và chuẩn hóa dữ liệu để tạo ra một phản hồi chính xác và dễ sử dụng cho UI.

Một nghiệp vụ khác rất quan trọng là việc đánh dấu chương học đã hoàn thành. LearningService không chỉ xác minh xem bản ghi về việc hoàn thành chương đã tồn tại hay chưa mà còn chịu trách nhiệm tạo bản ghi mới một cách an toàn. Điều này đảm bảo rằng tiến trình học tập được ghi nhận chính xác và không bị trùng lặp, đồng thời hỗ trợ tốt cho các tính năng nâng cao như đánh giá năng lực, phân tích hành vi học hoặc gợi ý nội dung học phù hợp.

Tổng thể, LearningService đóng vai trò như trung tâm điều phối, đảm bảo mọi logic nghiệp vụ được xử lý đúng domain, không phụ thuộc vào tầng lưu trữ dữ liệu và không bị chi phối bởi tầng hiển thị.

3. Repository Layer – Tầng truy cập dữ liệu cho Enrollment và Progress

   Tầng Repository trong module Learning được thiết kế tách biệt hoàn toàn theo từng thực thể, đảm bảo tính chuyên biệt và tối ưu hóa truy vấn. EnrollmentRepository chịu trách nhiệm xử lý trực tiếp dữ liệu liên quan đến việc ghi danh khóa học, trong đó phương thức tìm kiếm theo studentId cho phép hệ thống nhanh chóng trả về danh sách khóa học mà sinh viên đang học. Bên cạnh đó, phương thức kiểm tra sự tồn tại của bản ghi theo studentId và courseId là công cụ chính để LearningService bảo vệ dữ liệu khỏi trùng lặp, đảm bảo tính nhất quán suốt vòng đời của Enrollment.

   Song song với đó, ProgressRepository được thiết kế để phục vụ việc truy xuất và lưu trữ tiến trình chương học. Đây là tầng chuyên dụng cho thực thể StudentFinishChapter, giúp xác minh xem chương đã được hoàn thành trước đó hay chưa. Khi có bản ghi mới, ProgressRepository đảm nhận nhiệm vụ lưu trữ và cập nhật. Nhờ kiến trúc phân tách rõ ràng này, mỗi repository chỉ phục vụ đúng một nhu cầu nghiệp vụ duy nhất, phù hợp với nguyên tắc SRP và đảm bảo hệ thống dễ bảo trì trong tương lai.

4. Entity Model – Cấu trúc dữ liệu phản ánh hành trình học tập

   Phần mô hình dữ liệu của module Learning được xây dựng dựa trên thực tế hoạt động học tập của sinh viên trong hệ thống. Thực thể Enrollment đóng vai trò là "chứng nhận" việc sinh viên đã đăng ký một khóa học cụ thể. Ngoài các thông tin căn bản như studentId, courseId và thời điểm đăng ký, nó còn chứa thêm các trường như isStarred hoặc isSaved, phục vụ cho các chức năng nâng cao của hệ thống như ghim khóa học, cá nhân hóa dashboard hoặc lưu khóa học để học sau.

   Thực thể StudentFinishChapter biểu diễn chi tiết từng chương mà sinh viên đã hoàn thành. Mỗi bản ghi trong bảng này tương ứng với một mốc quan trọng trong tiến trình học, cho phép hệ thống theo dõi sự tiến bộ theo thời gian. Thông tin này không chỉ phục vụ cho việc tính toán tiến độ học mà còn có thể dùng để phân tích hành vi học tập hoặc hỗ trợ các thuật toán recommendation.

   Nhờ mối quan hệ chặt chẽ giữa Enrollment và StudentFinishChapter, module Learning mô phỏng chính xác vòng đời của hành trình học tập của người dùng trong môi trường e-learning, đảm bảo tính toàn vẹn, minh bạch và khả năng mở rộng trong tương lai.

Giải thích tuân theo nguyên tắc SOLID:

1. Nguyên lý Đơn nhiệm (Single Responsibility Principle – SRP)

   Thiết kế trong sơ đồ Learning thể hiện rõ cách phân tách trách nhiệm theo từng cấp và từng loại dữ liệu. LearningController chỉ chịu trách nhiệm tiếp nhận và chuyển tiếp yêu cầu từ client, chẳng hạn lấy danh sách khóa học mà sinh viên đã đăng ký, đánh dấu hoàn thành chapter, hoặc xem tiến trình học. Tất cả logic nghiệp vụ phức tạp như kiểm tra sinh viên đã enroll hay chưa, tính toán tiến trình khóa học, xác định chapter đã hoàn thành… đều được đóng gói trong LearningService, giúp controller không bị "phình to" với logic. Ở tầng dữ liệu, các repository phụ trách các nhiệm vụ rất cụ thể: EnrollmentRepository chỉ xử lý dữ liệu enrollment (đăng ký khóa học), còn ProgressRepository chịu trách nhiệm lưu và truy vấn thông tin hoàn thành từng chapter. Tách nhiệm vụ theo đối tượng dữ liệu như vậy đảm bảo rằng mỗi lớp chỉ thay đổi khi nghiệp vụ liên quan đến đối tượng đó thay đổi. Điều này giúp mô-đun Learning dễ bảo trì, dễ mở rộng và dễ viết unit test từng phần độc lập.

2. Nguyên lý Đóng/Mở (Open–Closed Principle – OCP)

   Sự hiện diện của interface ILearningService đóng vai trò như một lớp trừu tượng để Controller giao tiếp. Nhờ đó, LearningController hoàn toàn không phụ thuộc vào implementation cụ thể của LearningService. Nếu trong tương lai muốn mở rộng nghiệp vụ (ví dụ: thêm tracking chi tiết từng phút học, hoặc áp dụng công nghệ event sourcing để ghi log tiến trình học), thì chỉ cần tạo một LearningService mới mà không phải chỉnh sửa bất kỳ dòng code nào của controller.

3. Nguyên lý Thay thế Liskov (LSP)

   Interface ILearningService đóng vai trò như một hợp đồng rõ ràng cho các lớp triển khai. Controller chỉ cần biết đến interface này, đảm bảo rằng bất kỳ implementation nào cũng có thể thay thế cho implementation hiện tại mà không thay đổi hành vi hệ thống. Giả sử muốn bổ sung một phiên bản LearningService sử dụng AI để gợi ý lộ trình học thông minh thì chỉ cần viết LearningServiceV2 và đảm bảo tuân thủ interface. Vì mọi hành vi đều được mô tả trong hợp đồng, Controller có thể hoạt động với cả hai implementation mà không cần thay đổi. Điều này thể hiện đúng tinh thần LSP: thay thế implementation mà không phá vỡ logic luồng xử lý đã có.

4. Nguyên lý Phân tách Interface (Interface Segregation Principle – ISP)

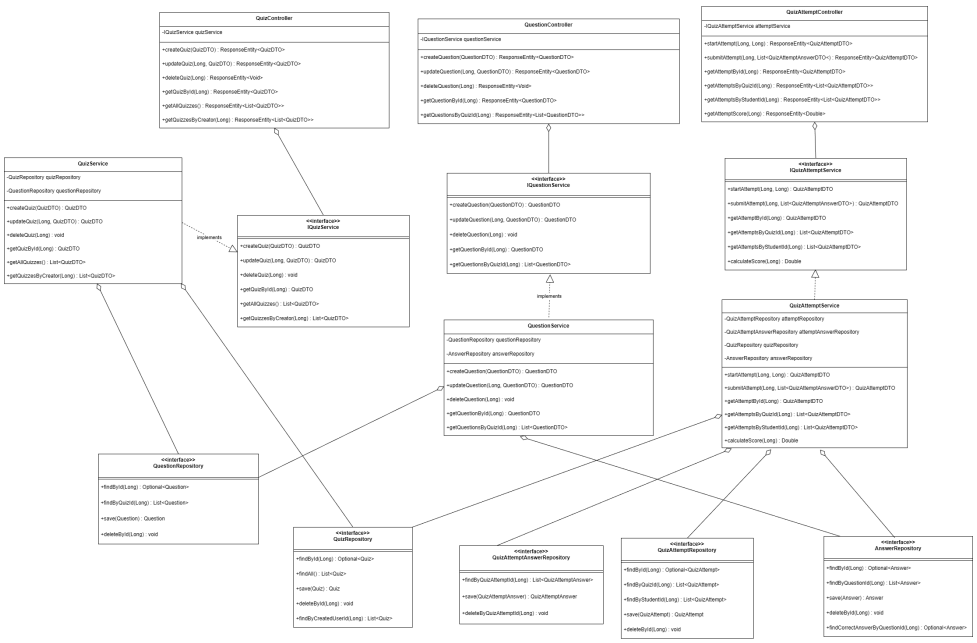
   Thiết kế chia interface phân tách theo đúng domain: ILearningService, EnrollmentRepository, ProgressRepository. Mỗi interface chỉ phục vụ đúng nhiệm vụ liên quan: LearningService giải quyết nghiệp vụ huấn luyện; EnrollmentRepository cung cấp khả năng truy vấn enrollment; ProgressRepository xử lý trạng thái hoàn thành chapter.

   Không có class nào phải implement hoặc phụ thuộc vào các phương thức thừa, không liên quan. Ví dụ, EnrollmentRepository không hề chứa phương thức để lưu tiến trình học, và ProgressRepository cũng không chứa logic liên quan đến việc kiểm tra khóa học đã được enroll chưa. Sự phân tách gọn gàng này giúp hệ thống dễ mở rộng, giảm độ coupling và tránh việc một service phải biết "quá nhiều".

5. Nguyên lý Đảo ngược Phụ thuộc (Dependency Inversion Principle – DIP)

   Controller trong sơ đồ luôn phụ thuộc vào abstraction là ILearningService, chứ không phụ thuộc trực tiếp vào LearningService. Điều này tạo nên một thiết kế ổn định, không bị phụ thuộc cứng vào bất kỳ implementation cụ thể nào. LearningService lại phụ thuộc vào abstraction của repository (EnrollmentRepository và ProgressRepository), giúp toàn bộ tầng nghiệp vụ tách biệt khỏi tầng dữ liệu.

6.6 Class diagram cho Module quiz



Hình 23: *Class diagram cho Module quiz*

1. QuizController – Tầng giao tiếp xử lý hoạt động quiz

   QuizController đóng vai trò là cổng giao tiếp cho tất cả thao tác liên quan đến quiz trong hệ thống. Khi giảng viên tạo, cập nhật hoặc xóa quiz, Controller tiếp nhận yêu cầu và chuyển xuống tầng service để xử lý nghiệp vụ. Với các thao tác truy vấn như lấy danh sách quiz theo khóa học hoặc theo người tạo, Controller chỉ giữ nhiệm vụ điều phối dữ liệu, đảm bảo luồng vào/ra đơn giản, rõ ràng. Việc tách riêng QuestionController giúp hệ thống xử lý chuyên biệt các thao tác CRUD của câu hỏi mà không làm QuizController trở nên cồng kềnh. Cách tổ chức này duy trì nguyên tắc "thin controller", nơi controller không chứa bất kỳ logic nghiệp vụ phức tạp nào, giúp mã nguồn dễ đọc và dễ bảo trì.

2. QuizAttemptController – Tầng giao tiếp xử lý hoạt động làm bài

   QuizAttemptController phụ trách toàn bộ vòng đời thực hiện quiz của sinh viên. Khi sinh viên bắt đầu làm bài, controller tạo một phiên làm bài mới thông qua service và trả về thông tin phiên đó. Khi sinh viên hoàn thành và nộp bài, controller chuyển toàn bộ thông tin câu trả lời xuống service để thực hiện chấm điểm tự động. Controller cũng cung cấp các API để xem lịch sử làm bài theo sinh viên hoặc theo quiz. Nhờ đảm nhận vai trò điều phối nhẹ nhàng và giao toàn bộ logic chấm điểm cho service, controller đảm bảo giao diện API REST tuân thủ chuẩn và dễ dàng tích hợp với frontend.

3. IQuizService – Tầng nghiệp vụ xử lý quiz

   IQuizService định nghĩa toàn bộ nghiệp vụ cốt lõi của quiz như tạo mới, cập nhật, xóa và truy vấn quiz. Implementation của service chịu trách nhiệm xử lý các logic phức tạp như kiểm tra tính hợp lệ của quiz, quản lý các câu hỏi và đáp án đi kèm, cũng như đảm bảo dữ liệu nhất quán khi cập nhật hoặc xóa quiz. Nhờ abstraction từ interface, controller không cần biết implementation cụ thể và hệ thống dễ dàng mở rộng — ví dụ khi cần thêm dạng quiz mới hoặc cần bổ sung cơ chế cấu hình bài kiểm tra nâng cao.

4. IQuestionService – Tầng nghiệp vụ xử lý câu hỏi

   IQuestionService quản lý độc lập vòng đời của từng câu hỏi mà không phụ thuộc vào toàn bộ quiz. Service xử lý việc thêm mới, chỉnh sửa hoặc xóa câu hỏi, đồng thời đảm bảo rằng mỗi câu hỏi luôn có đáp án phù hợp và dữ liệu luôn ổn định. Khi cập nhật câu hỏi, service đồng bộ lại các đáp án liên quan và đảm bảo không phá vỡ dữ liệu được tham chiếu bởi các phiên làm bài đã tồn tại. Nhờ tách riêng service này ra khỏi IQuizService, hệ thống dễ dàng mở rộng và thao tác trên câu hỏi trở nên an toàn, linh hoạt hơn.

5. IQuizAttemptService – Tầng nghiệp vụ làm bài và chấm điểm

   IQuizAttemptService đóng vai trò trọng tâm trong hoạt động làm bài. Service xử lý việc khởi tạo phiên làm bài, lưu trữ câu trả lời, tính toán điểm số và xác định kết quả đạt hay không đạt. Khi nộp bài, service so sánh các đáp án sinh viên chọn với đáp án đúng được lưu trong hệ thống để tính điểm chính xác. Service cũng xử lý các trường hợp đặc biệt như hết thời gian hoặc nhiều lần làm bài tùy theo cấu hình của quiz. Toàn bộ logic chấm điểm và xác thực đều nằm ở tầng này, giúp controller và repository không bị lẫn lộn nghiệp vụ.

6. Repository Layer – Tầng truy cập dữ liệu

   Tầng repository được phân tách theo từng domain: QuizRepository xử lý dữ liệu quiz, QuestionRepository xử lý câu hỏi, AnswerRepository xử lý đáp án, QuizAttemptRepository quản lý phiên làm bài, và QuizAttemptAnswerRepository lưu chi tiết câu trả lời. Mỗi repository chỉ chịu trách nhiệm với đúng một entity duy nhất, cung cấp các phương thức truy xuất dữ liệu theo nhu cầu của từng service. Cách tách này giúp dữ liệu được quản lý rõ ràng, truy vấn tối ưu và bám sát nguyên tắc SRP.

7. Entity Model – Cấu trúc dữ liệu của hệ thống đánh giá

   Các entity Quiz, Question, Answer, QuizAttempt và QuizAttemptAnswer xây dựng nên mô hình dữ liệu mô phỏng hoạt động kiểm tra trực tuyến. Quiz lưu cấu hình tổng thể của bài kiểm tra; Question lưu nội dung câu hỏi; Answer lưu các lựa chọn và đánh dấu đáp án đúng; QuizAttempt lưu thông tin mỗi lần làm bài của sinh viên; và QuizAttemptAnswer lưu chi tiết từng đáp án mà sinh viên đã chọn. Cấu trúc quan hệ chặt chẽ này giúp hệ thống truy vết chính xác từng hành vi làm bài và hỗ trợ chấm điểm tự động hiệu quả.

Giải thích tuân theo nguyên tắc SOLID:

1. Nguyên lý Đơn nhiệm (Single Responsibility Principle – SRP)

   QuizController chỉ chịu trách nhiệm tiếp nhận và chuyển tiếp yêu cầu từ client, chẳng hạn tạo quiz mới, cập nhật quiz, hoặc lấy danh sách quiz theo người tạo. QuestionController tách biệt để xử lý riêng các thao tác với câu hỏi, và QuizAttemptController chuyên xử lý việc sinh viên làm bài.

   Ở tầng dữ liệu, các repository phụ trách các nhiệm vụ rất cụ thể: QuizRepository chỉ xử lý dữ liệu quiz, QuestionRepository quản lý câu hỏi, AnswerRepository quản lý đáp án, QuizAttemptRepository xử lý phiên làm bài, và QuizAttemptAnswerRepository lưu trữ chi tiết câu trả lời.

2. Nguyên lý Đóng/Mở (Open–Closed Principle – OCP)

   Sự hiện diện của các interface IQuizService, IQuestionService và IQuizAttemptService đóng vai trò như các lớp trừu tượng để Controller giao tiếp. Nhờ đó, các Controller hoàn toàn không phụ thuộc vào implementation cụ thể của Service. Nếu trong tương lai muốn mở rộng nghiệp vụ (ví dụ: thêm chức năng quiz thích ứng - adaptive quiz dựa trên AI, hoặc áp dụng thuật toán chấm điểm phức tạp hơn với trọng số từng câu hỏi, hoặc tích hợp với hệ thống anti-cheating), thì chỉ cần tạo implementation mới như QuizServiceV2 hoặc AIBasedQuizAttemptService mà không phải chỉnh sửa bất kỳ dòng code nào của controller.

3. Nguyên lý Thay thế Liskov (Liskov Substitution Principle – LSP)

   Các interface IQuizService, IQuestionService và IQuizAttemptService đóng vai trò như các hợp đồng rõ ràng cho các lớp triển khai. Controller chỉ cần biết đến các interface này, đảm bảo rằng bất kỳ implementation nào cũng có thể thay thế cho implementation hiện tại mà không thay đổi hành vi hệ thống.

   Giả sử muốn bổ sung một phiên bản QuizAttemptService sử dụng machine learning để phát hiện gian lận dựa trên pattern làm bài (thời gian giữa các câu trả lời, độ tương đồng với bài làm của người khác), thì chỉ cần viết AnticheatQuizAttemptService và đảm bảo tuân thủ interface. Tương tự, nếu cần tạo một AdvancedQuizService hỗ trợ question pool (ngân hàng câu hỏi ngẫu nhiên) hoặc question branching (câu hỏi tiếp theo phụ thuộc vào câu trả lời trước đó), miễn là nó triển khai đúng IQuizService, toàn bộ hệ thống vẫn hoạt động ổn định.

4. Nguyên lý Phân tách Interface (Interface Segregation Principle – ISP)

   Thiết kế chia interface phân tách theo đúng domain: IQuizService quản lý quiz, IQuestionService quản lý câu hỏi, IQuizAttemptService quản lý việc làm bài. Mỗi interface chỉ phục vụ đúng nhiệm vụ liên quan và không bắt buộc class triển khai phải implement các phương thức không cần thiết.

   Ví dụ, IQuizAttemptService không chứa các phương thức createQuiz() hay updateQuestion() - những phương thức này không liên quan đến nghiệp vụ làm bài. Ngược lại, IQuizService không chứa phương thức calculateScore() vì việc chấm điểm là trách nhiệm của attempt service. Sự phân tách gọn gàng này giúp hệ thống dễ mở rộng, giảm độ coupling và tránh việc một service phải biết "quá nhiều" về các domain khác.

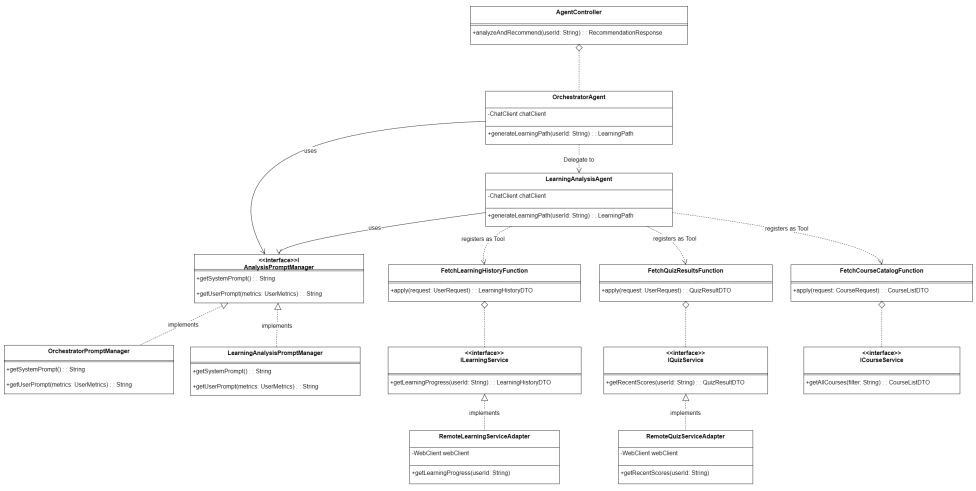
5. Nguyên lý Đảo ngược Phụ thuộc (Dependency Inversion Principle – DIP)

   Controller trong sơ đồ luôn phụ thuộc vào abstraction (IQuizService, IQuestionService, IQuizAttemptService), chứ không phụ thuộc trực tiếp vào các implementation cụ thể (QuizServiceImpl, QuestionServiceImpl, QuizAttemptServiceImpl).

   Service layer lại phụ thuộc vào abstraction của repository (QuizRepository, QuestionRepository, AnswerRepository, QuizAttemptRepository, QuizAttemptAnswerRepository), giúp toàn bộ tầng nghiệp vụ tách biệt khỏi tầng dữ liệu.

   Việc áp dụng DIP này đặc biệt quan trọng trong môi trường microservices hoặc khi cần viết integration test. Các abstraction cho phép dễ dàng mock hoặc stub các dependency, giúp test từng layer độc lập mà không cần khởi động toàn bộ hệ thống.

6.7 Class diagram cho Module AI



Hình 24: *Class diagram cho Module AI*

1. AgentController – Tầng giao tiếp xử lý yêu cầu AI

   AgentController đóng vai trò là cổng giao tiếp duy nhất (Entry Point) cho các yêu cầu phân tích và tư vấn từ phía người dùng (Client). Khi người dùng yêu cầu phân tích lộ trình học hoặc đề xuất khóa học, Controller tiếp nhận request (thường chứa userId), xác thực cơ bản, và chuyển tiếp yêu cầu đó xuống tầng Agent để xử lý. Lớp này tuân thủ nguyên tắc "thin controller", không chứa logic suy luận AI hay logic gọi API bên thứ 3, đảm bảo đầu mối giao tiếp RESTful API luôn gọn nhẹ và dễ bảo trì.

2. OrchestratorAgent – Tầng điều phối thông minh (Supervisor)

   OrchestratorAgent đóng vai trò là "nhạc trưởng" hay bộ não trung tâm của AI Service. Thay vì trực tiếp thực hiện mọi tác vụ, nó chịu trách nhiệm phân tích ý định của người dùng và điều phối công việc cho các Agent chuyên biệt (Worker) hoặc quyết định luồng xử lý chính. Ví dụ, khi nhận yêu cầu phức tạp, Orchestrator sẽ xác định xem cần gọi LearningAnalysisAgent để phân tích dữ liệu hay gọi một Agent khác để tư vấn tâm lý. Việc tách tầng điều phối này giúp hệ thống dễ dàng mở rộng sang mô hình Multi-Agent trong tương lai mà không làm rối luồng xử lý chính.

3. LearningAnalysisAgent – Tầng nghiệp vụ phân tích chuyên sâu (Worker)

   LearningAnalysisAgent là thành phần cốt lõi thực hiện các nghiệp vụ cụ thể như phân tích hành vi học tập và gợi ý lộ trình. Agent này được cấu hình với các System Prompt chuyên biệt để đóng vai trò như một chuyên gia giáo dục. Điểm đặc biệt là Agent này không hoạt động độc lập; nó được trang bị các "Tools" (công cụ) để có thể tương tác với thế giới bên ngoài. Khi cần dữ liệu để suy luận, Agent sẽ tự động quyết định việc gọi các Function (Tools) để lấy lịch sử học tập hoặc điểm thi, sau đó tổng hợp thông tin để đưa ra câu trả lời cuối cùng dưới dạng JSON hoặc văn bản tự nhiên.

4. AnalysisPromptManager – Tầng quản lý ngữ cảnh và chỉ thị (Prompt Engineering)

   AnalysisPromptManager chịu trách nhiệm quản lý toàn bộ các câu lệnh chỉ dẫn (System Prompt) và mẫu câu hỏi (User Prompt) cho AI. Việc tách biệt quản lý Prompt ra khỏi logic code Java giúp lập trình viên dễ dàng tinh chỉnh "tính cách" và "luật chơi" của AI mà không cần biên dịch lại mã nguồn. Interface IAnalysisPromptManager đảm bảo rằng các Agent khác nhau có thể sử dụng các chiến lược Prompt khác nhau nhưng vẫn tuân thủ cùng một chuẩn giao tiếp.

5. Tools & Functions Layer – Tầng công cụ mở rộng (Function Calling)

   Đây là các lớp chức năng (FetchLearningHistoryFunction, FetchQuizResultsFunction, FetchCourseCatalogFunction) đóng vai trò là "cánh tay nối dài" của AI. Mỗi Function được đăng ký như một java.util.function.Function trong Spring AI Context. Chúng đóng gói logic nghiệp vụ để chuyển đổi yêu cầu từ ngôn ngữ tự nhiên của AI thành các lệnh gọi hàm Java cụ thể. Tầng này giúp AI có khả năng hành động thực tế (lấy dữ liệu, tra cứu) thay vì chỉ là một chatbot trả lời văn bản tĩnh.

6. Infrastructure & Gateway Layer – Tầng kết nối (Anti-Corruption Layer)

   Tầng này bao gồm các Interface Gateway (ILearningService, IQuizService, ICourseService) và các Implementation (Remote...Adapter). Nhiệm vụ của chúng là thực hiện các giao tiếp mạng (thông qua RestClient hoặc WebClient) sang các khác như Learning Service, Quiz Service hay Course Service. Việc sử dụng Adapter Pattern ở đây giúp cách ly AI Service khỏi sự thay đổi của các service bên ngoài. Nếu API của Quiz Service thay đổi, ta chỉ cần sửa RemoteQuizServiceAdapter, logic của AI Agent hoàn toàn không bị ảnh hưởng.

Giải thích tuân theo nguyên tắc SOLID:

1. Nguyên lý Đơn nhiệm (Single Responsibility Principle – SRP)

   - AgentController: Chỉ lo việc nhận/trả HTTP request.
   - OrchestratorAgent: Chỉ lo điều phối luồng hội thoại.
   - LearningAnalysisAgent: Chỉ tập trung vào nghiệp vụ phân tích giáo dục.
   - Fetch...Function: Chỉ làm nhiệm vụ chuyển đổi dữ liệu để AI hiểu.
   - Remote...Adapter: Chỉ lo việc gọi API sang service khác. Sự phân chia này giúp code rõ ràng, mỗi class chỉ thay đổi vì một lý do duy nhất.

2. Nguyên lý Đóng/Mở (Open–Closed Principle – OCP) Hệ thống được thiết kế để mở rộng khả năng của AI mà không cần sửa đổi code lõi.

   - Nếu muốn AI có thêm khả năng mới (ví dụ: xem thời khóa biểu), ta chỉ cần tạo thêm FetchScheduleFunction và đăng ký nó vào Agent. Không cần sửa logic suy luận bên trong LearningAnalysisAgent.
   - Nếu muốn thay đổi model từ OpenAI sang Gemini, chỉ cần thay đổi cấu hình trong ChatModel, code logic trong Agent giữ nguyên.
   - Việc có một lớp Orchestrator giữa lớp xử lý yêu cầu người dùng và lớp chính của nó. Điều này cho phép mở rộng ra hệ thống multi-agent khi chỉ cần hiện thực thêm agent và đăng ký nó đến Orchestrator

3. Nguyên lý Thay thế Liskov (Liskov Substitution Principle – LSP)

   Các Gateway Interface (ILearningService, IQuizService) đóng vai trò hợp đồng.

   - RemoteLearningServiceAdapter có thể được thay thế bằng MockLearningServiceAdapter (dữ liệu giả lập) trong môi trường kiểm thử (Unit Test) mà không làm hỏng logic của FetchLearningHistoryFunction. Hệ thống hoạt động đúng đắn bất kể implementation nào được inject vào.

4. Nguyên lý Phân tách Interface (Interface Segregation Principle – ISP)

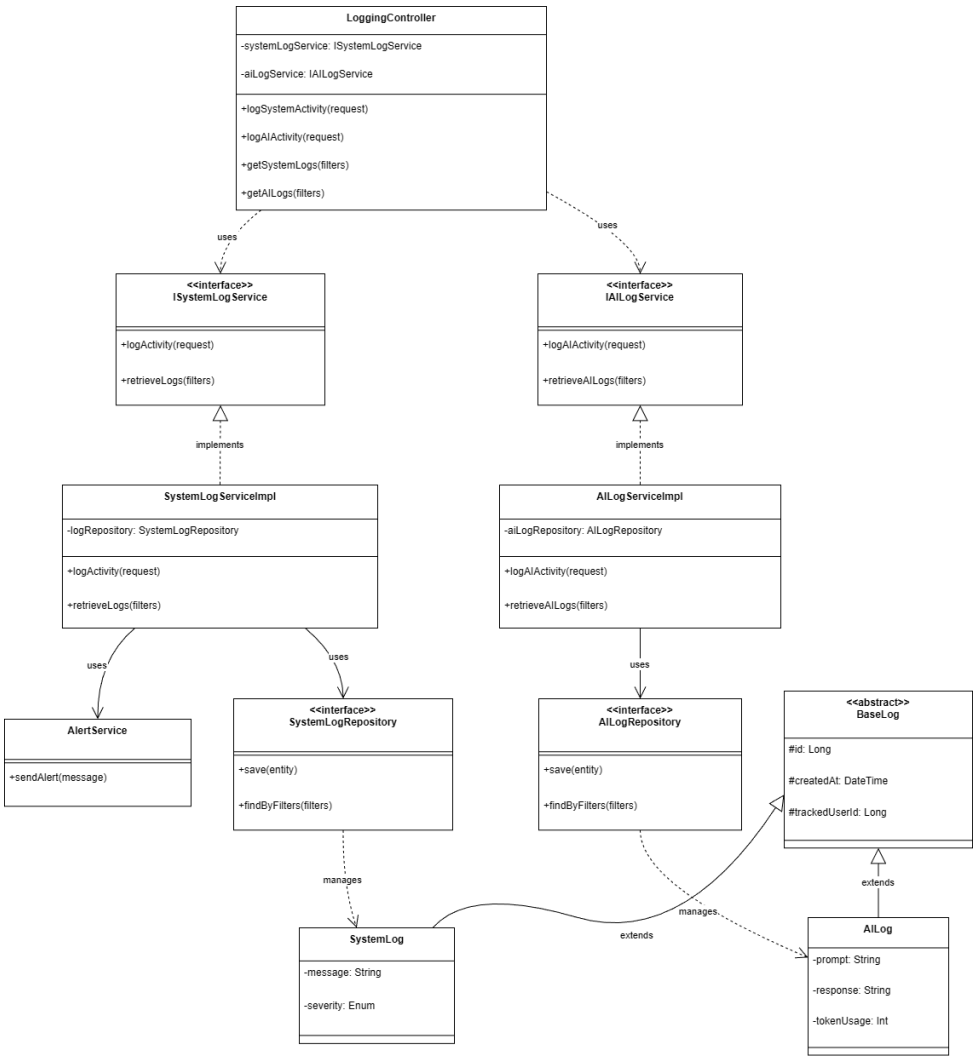
   Thay vì tạo một IGlobalServiceGateway khổng lồ chứa tất cả các hàm gọi API, hệ thống chia nhỏ thành ILearningService, IQuizService, ICourseService.

   - FetchQuizResultsFunction chỉ cần biết đến IQuizService, nó không cần phụ thuộc vào các phương thức liên quan đến Course hay Learning. Điều này giảm sự phụ thuộc chéo không cần thiết giữa các module.

5. Nguyên lý Đảo ngược Phụ thuộc (Dependency Inversion Principle – DIP)

   - Agent (High-level Module) không phụ thuộc trực tiếp vào RemoteServiceAdapter (Low-level Module).
   - Cả hai đều phụ thuộc vào Abstraction (Interface).
     - LearningAnalysisAgent phụ thuộc vào Function (Tool).
     - FetchLearningHistoryFunction phụ thuộc vào ILearningService.
     - Việc khởi tạo cụ thể (Dependency Injection) do Spring Container đảm nhận. Điều này giúp kiến trúc lỏng lẻo (loose coupling), dễ dàng bảo trì và kiểm thử độc lập.

6.8 Class diagram cho Module Logging



Hình 25: *Class diagram cho Module Logging*

1. LoggingController – Tầng giao tiếp tiếp nhận yêu cầu ghi Log

   LoggingController đóng vai trò là cổng giao tiếp duy nhất (Entry Point) cho các yêu cầu liên quan đến nhật ký hệ thống (System Logs) và nhật ký hoạt động AI (AI Logs). Khi các service khác hoặc client gửi yêu cầu ghi log hoặc truy xuất lịch sử, Controller tiếp nhận request, thực hiện các validate cơ bản và điều hướng đến service tương ứng (SystemLogService hoặc AILogService). Lớp này tuân thủ nguyên tắc "thin controller", chỉ làm nhiệm vụ điều phối luồng dữ liệu thông qua các Interface, đảm bảo tính tách biệt giữa giao diện API và logic xử lý nghiệp vụ bên dưới.

2. Service Layer (ISystemLogService & IAILogService) – Tầng trừu tượng hóa nghiệp vụ

   Đây là các Interface định nghĩa "hợp đồng" (Contract) cho các nghiệp vụ xử lý log. Việc tách biệt thành hai interface riêng biệt (ISystemLogService và IAILogService) giúp hệ thống phân định rõ ràng trách nhiệm: một bên chuyên xử lý các log vận hành hệ thống (lỗi, cảnh báo), một bên chuyên xử lý log tương tác với AI (prompt, token usage). Điều này giúp Controller không cần biết chi tiết cách log được lưu hay xử lý như thế nào, mà chỉ cần biết các phương thức được cung cấp.

3. Implementation Layer (SystemLogServiceImpl & AILogServiceImpl) – Tầng thực thi nghiệp vụ (Worker)

   Đây là nơi chứa logic nghiệp vụ cốt lõi:

   - SystemLogServiceImpl: Chịu trách nhiệm xử lý các log hệ thống. Điểm đặc biệt là nó có sự tương tác với AlertService. Khi một log có mức độ nghiêm trọng (Severity) cao được ghi nhận, Service này có thể tự động kích hoạt gửi cảnh báo. Nó cũng gọi xuống Repository để lưu trữ dữ liệu.
   - AILogServiceImpl: Chuyên xử lý việc lưu trữ và truy xuất các tương tác AI (câu hỏi, câu trả lời, số token). Logic ở đây tập trung vào việc đảm bảo dữ liệu tương tác AI được ghi nhận chính xác để phục vụ việc audit hoặc tính phí (token usage).

4. AlertService – Tầng tích hợp thông báo (Integration)

   AlertService đóng vai trò là một service bổ trợ, được sử dụng bởi SystemLogServiceImpl. Chức năng duy nhất của nó là sendAlert(message), giúp gửi cảnh báo ra bên ngoài (có thể là Email, Slack, hoặc SMS) khi hệ thống gặp sự cố nghiêm trọng. Việc tách Alert thành một service riêng giúp logic ghi log không bị trộn lẫn với logic gửi thông báo.

5. Repository Layer – Tầng truy xuất dữ liệu (Data Access)

   SystemLogRepository và AILogRepository đóng vai trò là lớp trừu tượng để tương tác với cơ sở dữ liệu. Các Interface này cung cấp các phương thức chuẩn như save(entity) và findByFilters(filters). Tầng này đảm bảo rằng các Service ở tầng trên không phụ thuộc vào công nghệ lưu trữ cụ thể (SQL hay NoSQL), giúp việc thay đổi database sau này trở nên dễ dàng mà không ảnh hưởng đến logic nghiệp vụ.

6. Entity Layer (BaseLog, SystemLog, AILog) – Tầng mô hình dữ liệu (Data Model)

   Hệ thống sử dụng mô hình thừa kế để quản lý dữ liệu log một cách hiệu quả:

   BaseLog (Abstract): Chứa các trường thông tin chung nhất mà mọi loại log đều có (id, createdAt, trackedUserId), giúp tránh lặp lại code (DRY - Don’t Repeat Yourself).

   SystemLog: Mở rộng từ BaseLog, bổ sung các trường đặc thù cho vận hành như message và severity (Mức độ nghiêm trọng).

   AILog: Mở rộng từ BaseLog, bổ sung các trường đặc thù cho AI như prompt (câu lệnh), response (phản hồi), và tokenUsage.

Giải thích tuân theo nguyên tắc SOLID:

1. Nguyên lý Đơn nhiệm (Single Responsibility Principle – SRP)

   - LoggingController: Chỉ chịu trách nhiệm nhận và trả kết quả HTTP, không chứa logic xử lý log.
   - AlertService: Chỉ có duy nhất một nhiệm vụ là gửi cảnh báo, không quan tâm log được lưu như thế nào.
   - SystemLogServiceImpl: Tập trung vào logic nghiệp vụ của log hệ thống.
   - AILog: Chỉ chứa dữ liệu liên quan đến AI, không chứa dữ liệu về mức độ nghiêm trọng của hệ thống.
   - Mỗi class trong sơ đồ đều có một trách nhiệm duy nhất và rõ ràng, giúp code dễ đọc và dễ bảo trì.

2. Nguyên lý Đóng/Mở (Open–Closed Principle – OCP)

   Hệ thống được thiết kế để dễ dàng mở rộng:

- Mở rộng loại Log: Nếu cần thêm loại log mới (ví dụ: AuditLog cho giao dịch tài chính), ta chỉ cần tạo class AuditLog kế thừa từ BaseLog và tạo thêm Service/Repository tương ứng mà không cần sửa đổi class BaseLog hay các Service hiện có.
- Mở rộng logic Alert: Nếu muốn thay đổi cách gửi thông báo (từ Email sang Slack), ta chỉ cần sửa hoặc mở rộng implementation của AlertService mà không ảnh hưởng đến SystemLogServiceImpl.

3. Nguyên lý Thay thế Liskov (Liskov Substitution Principle – LSP)

   - Entity: SystemLog và AILog đều kế thừa từ BaseLog. Ở bất kỳ đâu trong hệ thống cần xử lý thông tin cơ bản (như lấy createdAt), ta có thể truyền vào instance của SystemLog hoặc AILog mà chương trình vẫn hoạt động đúng đắn.
   - Service: SystemLogServiceImpl thực thi ISystemLogService. Trong controller, ta tham chiếu đến Interface. Nếu ta tạo một MockSystemLogService để chạy Unit Test, controller vẫn hoạt động bình thường mà không cần biết đó là đồ giả hay đồ thật.

4. Nguyên lý Phân tách Interface (Interface Segregation Principle – ISP)

   Thay vì tạo một IGeneralLogService khổng lồ chứa cả hàm logSystemActivity và logAIActivity, hệ thống đã tách thành hai interface nhỏ:

   - ISystemLogService: Chỉ chứa các hàm liên quan log hệ thống.
   - IAILogService: Chỉ chứa các hàm liên quan log AI.
   - Điều này đảm bảo rằng nếu một client chỉ quan tâm đến AI Logs, họ không bị buộc phải phụ thuộc vào các phương thức của System Logs mà họ không dùng đến.

5. Nguyên lý Đảo ngược Phụ thuộc (Dependency Inversion Principle – DIP)

   - High-level Module (LoggingController): Không phụ thuộc trực tiếp vào Low-level Module (SystemLogServiceImpl, AILogServiceImpl). Thay vào đó, nó phụ thuộc vào Abstraction (ISystemLogService, IAILogService).
   - Service Layer: Các Service implementation không phụ thuộc trực tiếp vào Database connection cụ thể, mà phụ thuộc vào Repository Interface (SystemLogRepository, AILogRepository).
   - Nhờ DIP, việc thay đổi implementation bên dưới (ví dụ đổi logic lưu log từ MySQL sang MongoDB) sẽ không làm ảnh hưởng đến code của Controller hay Service.

7 Tổng kết

7.1 Nhận xét về Thiết kế và Hiện thực

Thiết kế kiến trúc cho Hệ thống Gia sư Thông minh (ITS) đã được xây dựng trên nền tảng **Service-Based Architecture (SBA)** , với sự hỗ trợ mạnh mẽ từ bộ công nghệ **Spring Cloud** và nền tảng **Amazon Web Services (AWS)**.

7.1.1 Ưu điểm của Thiết kế

- **Cân bằng Tối ưu giữa các Đặc tính (Trade-offs):** SBA đã giải quyết được mâu thuẫn giữa **Performance** (yêu cầu $< 300ms$) và **Elasticity** (khả năng co giãn khi tải cao). Kiến trúc này đảm bảo *độ trễ thấp* hơn Microservices và *khả năng mở rộng độc lập* cho các dịch vụ trọng yếu như *QuizService* và *AIService*.
- **Đảm bảo Tính nhất quán Dữ liệu (Data Consistency + Integrity):** Bằng việc sử dụng CSDL tập trung (*managed PostgreSQL Database* trên AWS RDS) thay vì giao dịch phân tán (Saga) của MSA, hệ thống dễ dàng đảm bảo **tính toàn vẹn giao dịch** (*transactional integrity*, NFR16) và **tính nhất quán** (NFR15).
- **Tuân thủ SOLID và Tách biệt Mối quan tâm:** Các *Class Diagram* (cho *Media Service*, *IAM Service*, *Course Service*) đều thể hiện sự tuân thủ nghiêm ngặt **Nguyên tắc SOLID**. Các lớp được tách biệt rõ ràng trách nhiệm (SRP) và sử dụng *interface* để đảo ngược sự phụ thuộc (DIP), tạo ra một mã nguồn dễ bảo trì và mở rộng.

7.1.2 Hạn chế và Rủi ro

- **CSDL dùng chung (Shared Database):** Việc sử dụng một CSDL dùng chung cho tất cả các *service* (IAM, Quiz, Course, Learning, AI, Logging) mặc dù đơn giản hóa *Data Consistency* nhưng có thể gây tắc nghẽn ở tầng dữ liệu khi hệ thống *scale* lên quy mô lớn hơn nữa.
- **Tính mô đun không tối đa:** So với kiến trúc Microservices, SBA vẫn có nguy cơ các *service* trở nên "quá lớn" nếu không quản lý module hợp lý , và khả năng *scale* mịn (*fine-grained scalability*) không bằng.

7.2 Hướng phát triển và Chiến lược Tối ưu hóa Kiến trúc (SBA)

Hướng phát triển chiến lược của hệ thống ITS sẽ tập trung vào việc củng cố cấu trúc **Service-Based Architecture (SBA)** hiện tại, cải thiện các đặc tính kiến trúc (Architecture Characteristics) và mở rộng phạm vi nghiệp vụ, duy trì sự ổn định và dễ quản lý của mô hình CSDL dùng chung.

- **Tối ưu hóa Hiệu năng và Khả năng Co giãn cho Dịch vụ AI (Performance & Elasticity):**
  - **Cải thiện Thời gian Phản hồi API:** Tiếp tục tinh chỉnh các API liên quan đến AI để đảm bảo thời gian phản hồi trung bình không vượt quá 2 giây (NFR18). Điều này đặc biệt quan trọng cho các tính năng gợi ý theo thời gian thực.
  - **Tăng cường Cơ chế Caching:** Triển khai và tối ưu hóa việc sử dụng cơ chế caching (NFR20) cho các kết quả tính toán của AI Service, đặc biệt là các kết quả đề xuất đã ổn định, nhằm cải thiện tốc độ truy xuất dữ liệu.
  - **Tận dụng Autoscaling chuyên biệt:** Thiết lập và tinh chỉnh cấu hình *autoscaling* (NFR25) cho các dịch vụ *AIService* và *QuizService*, đảm bảo chúng có khả năng tự động mở rộng dựa trên tải trọng CPU/GPU hoặc số lượng yêu cầu mỗi giây, tối ưu hóa chi phí vận hành (NFR24).
- **Củng cố Tính nhất quán và An toàn Dữ liệu (Data Consistency + Integrity):**
  - **Tăng cường Kiểm soát Giao dịch:** Triển khai các lớp dịch vụ và *transactional boundaries* rõ ràng hơn trong các *service* để đảm bảo **tính toàn vẹn giao dịch** (*transactional integrity*, NFR16) cho các thao tác quan trọng (ví dụ: nộp bài quiz, cập nhật tiến độ học tập).
  - **Mô hình hóa Dữ liệu không quan hệ (NoSQL):** Đối với các dữ liệu có lưu lượng ghi cao và ít yêu cầu về *transactional integrity* như *AILog* và *LoggingService*, nên cân nhắc sử dụng các cơ sở dữ liệu phi quan hệ (ví dụ: DynamoDB hoặc MongoDB) thay vì PostgreSQL để giảm áp lực lên CSDL chính, tối ưu hóa **Elasticity**.
  - **Tăng cường Bảo mật Dữ liệu:** Áp dụng các biện pháp mã hóa dữ liệu nghiêm ngặt hơn khi lưu trữ và truyền tải (NFR14) để bảo vệ dữ liệu PII và nội dung bài giảng.

- **Mở rộng Tính năng Nghiệp vụ Cốt lõi (Configurability & Course Learning):**
  - **Tăng cường Học tập Chủ động:** Tập trung phát triển các tính năng hỗ trợ thảo luận nhóm và *diễn đàn khóa học* (S-US03, T-US03, L-US04) để khuyến khích sinh viên tham gia vào các hoạt động học tập chủ động, buộc họ phải trở thành "người dạy" để củng cố khả năng tự điều chỉnh, từ đó đạt được tỷ lệ lưu giữ kiến thức cao hơn theo *Tam giác Học tập*.
  - **Mở rộng Tính Cấu hình (Configurability):** Mở rộng module cấu hình hệ thống (UC-SM-01) để cho phép Admin dễ dàng bật/tắt tính năng, điều chỉnh các tham số hoạt động (NFR01) và cấu hình các tham số tạo câu đố bằng AI (NFR07). Đảm bảo việc cấu hình loại tài liệu được phép đăng tải (NFR03) và layout khóa học (NFR04) được thực hiện linh hoạt.
  - **Tích hợp Thanh toán và LMS:** Chuẩn bị các *interface* và *adapter* trong *CourseService* để tích hợp liền mạch với các hệ thống bên ngoài như Payment Gateway và Hệ thống Quản lý Học tập (LMS) trong tương lai.

8 Phụ lục

8.1 ADR 00: Lựa chọn kiến trúc ban đầu: Microservices Architecture (MSA)

1. **Tiêu đề:** Lựa chọn kiến trúc ban đầu: Microservices Architecture (MSA).
2. **Status (Trạng thái):** Superseded (Đã bị thay thế).
3. **Context (Bối cảnh):**
   (a) Hệ thống dự kiến sẽ phục vụ lượng lớn sinh viên, đặc biệt là vào các đợt cao điểm như mùa thi, đòi hỏi khả năng co giãn tối đa (Maximum Scalability).
   (b) Cần tối ưu hóa hiệu năng cho các Module AI (NFR18, NFR25) bằng cách sử dụng các công nghệ khác nhau (Polyglot) cho từng thành phần.
   (c) Các yêu cầu về **Elasticity** (NFR23, NFR24) và **Concurrency** (NFR21) là rất cao và cần được hỗ trợ bằng việc cô lập tài nguyên cho từng tính năng.
4. **Decision (Quyết định):**
   (a) Áp dụng **Microservices Architecture (MSA)** để phân tách các Module chức năng thành các dịch vụ độc lập (ví dụ: Quiz Generation Service, User Service).
   (b) Mỗi dịch vụ sẽ có khả năng **triển khai độc lập** và **mở rộng độc lập** (Autoscaling) dựa trên tải trọng CPU/GPU hoặc số lượng yêu cầu.
   (c) Sử dụng **Eventual Consistency** cho các giao tiếp liên Service, chấp nhận độ trễ nhỏ để đạt được tính độc lập cao.
5. **Consequences (Hệ quả):**
   (a) **Khả năng mở rộng tối đa (Maximum Elasticity):** Đáp ứng tốt yêu cầu NFR25 bằng cách mở rộng các dịch vụ AI một cách chi tiết.
   (b) **Tăng độ phức tạp về vận hành (DevOps):** Yêu cầu đội ngũ vận hành mạnh và các công cụ giám sát phức tạp.
   (c) **Rủi ro về Data Consistency:** Phải quản lý giao dịch phân tán (Saga) giữa các CSDL riêng biệt, làm tăng độ phức tạp trong việc đảm bảo NFR16.

8.2 ADR 01: Chuyển đổi kiến trúc từ Microservices (MSA) sang Service-Based Architecture (SBA)

1. **Tiêu đề:** Chuyển đổi kiến trúc từ Microservices (MSA) sang Service-Based Architecture (SBA).
2. **Status (Trạng thái):** Accepted (Đã chấp nhận).
3. **Context (Bối cảnh):**
   (a) Hệ thống có yêu cầu cao về **Tính nhất quán giao dịch dữ liệu** (NFR16) và các mối quan hệ dữ liệu phức tạp giữa các Domain (Quiz, Course, Student).
   (b) Độ phức tạp quản lý giao dịch phân tán (Saga) của MSA sẽ làm tăng chi phí và thời gian phát triển đáng kể.
   (c) Mặc dù yêu cầu về tải có đỉnh điểm (mùa thi), quy mô hiện tại phù hợp hơn với kiến trúc ít phức tạp hơn.
4. **Decision (Quyết định):**
   (a) Áp dụng **Service-Based Architecture (SBA)**, phân chia hệ thống thành các dịch vụ lớn (User/Access, Course Modification, Quiz/Assessment, Learning/Interaction, Reporting/Config).
   (b) Sử dụng **API Gateway** để định tuyến, chứng thực và cung cấp lớp caching thống nhất (NFR20).
   (c) Sử dụng **Service Registry** để quản lý vị trí động và trạng thái của các dịch vụ, hỗ trợ **Elasticity** (NFR23, NFR24) và **Availability** (NFR10, NFR11).
5. **Consequences (Hệ quả):**
   (a) **Cải thiện Data Consistency (NFR15, NFR16):** Dễ dàng đạt được thông qua giao dịch cục bộ hoặc quản lý CSDL đơn giản hơn.
   (b) **Giảm chi phí vận hành** và tăng tốc độ phát triển.
   (c) Vẫn giữ được tính linh hoạt mở rộng nhờ các thành phần hỗ trợ (API Gateway, Service Registry).