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

BÁO CÁO ĐỒ ÁN MÔN HỌC

MÔN HỌC: Đồ Án Tổng Hợp - Công nghệ Phần mềm

Đề tài: Grade Portal for students at HCMUT

HK241

Giảng viên hướng dẫn: Lê Đình Thuận
Lớp, nhóm hiện thực: L07 - Nhóm thầy Thuận
Sinh viên: Trần Đại Việt - 2213951
Phạm Văn Quốc Việt - 2213950
Nguyễn Nhật Khoa - 2211629
Phạm Việt Anh - 2210128
Nguyễn Gia Nguyên - 2212303
Lê Đăng Khoa - 2211599

HO CHI MINH CITY, SEPTEMBER 2024

Mục lục

1 Danh Sách Thành Viên Và Phân Công Nhiệm Vụ **1**

2 Giới Thiệu Chung Về Dự Án **2**
2.1 Bối cảnh chung và tính cấp thiết của đề tài . . . 2
2.2 Các bên liên quan (Stakeholders) và nhu cầu của họ . . . 3
2.3 Lợi ích nhận được khi dự án hoàn thành . . . 3

3 Phân Tích Các Yêu Cầu Của Dự Án **4**
3.1 Các yêu cầu chức năng (Functional Requirements) . . . 4
3.2 Các yêu cầu phi chức năng (Non-functional Requirements) . . . 4

4 Kiến trúc hệ thống (System Architecture) **6**
4.1 Layered Architecture . . . 6
4.2 Sơ đồ hiện thực của hệ thống . . . 7
4.2.1 Presentation Layer . . . 8
4.2.2 Business Layer . . . 9
4.2.3 Persistence Layer . . . 11
4.2.4 Data Layer . . . 12
4.3 Deployment Diagram . . . 13
4.4 Data Storage Approach . . . 14
4.5 API management . . . 16

5 Mô hình hóa hệ thống (System Modelling) **18**
5.1 Sơ Đồ Chức Năng Của Hệ Thống (Use-case Diagram) . . . 18
5.1.1 Xác thực người dùng . . . 19
5.2 Activity Diagram . . . 21
5.2.1 Activity Diagram chung cho các Usecase trong hệ thống . . . 21
5.2.2 Activity Diagram cho chức năng xác thực của hệ thống . . . 22
5.2.3 Activity Diagram cho chức năng tạo lớp học mới . . . 24
5.2.4 Activity Diagram cho chức năng cập nhật điểm số của giảng viên . . . 25
5.2.5 Activity Diagram cho chức năng xem điểm của sinh viên . . . 26
5.3 Sequence Diagram . . . 27
5.3.1 Sequence Diagram chung cho các Usecase trong hệ thống . . . 27
5.3.2 Sequence Diagram cho chức năng xác thực của hệ thống . . . 28
5.3.3 Sequence Diagram cho chức năng tạo lớp học mới . . . 30
5.3.4 Sequence Diagram cho chức năng cập nhật điểm số của giảng viên . . . 31
5.3.5 Sequence Diagram cho chức năng xem điểm của sinh viên . . . 32

6 Tổ chức và quản lý Code trong mã nguồn **33**
6.1 Mô tả chi tiết các thành phần chính . . . 34
6.2 Code style . . . 34

7 Quản lý phiên bản (Version Control) **35**
7.1 Lý Do Lựa Chọn GitHub . . . 35
7.2 Vấn đề về việc mã hóa file `.env` khi đưa lên repo dự án . . . 35
7.3 Gitflow . . . 36

8 Quản lý phân công việc hiện thực cho các thành viên trong nhóm **39**
8.1 Quy trình phát triển phần mềm - Mô hình Waterfall . . . 39
8.2 Biểu đồ Gantt hỗ trợ lên kế hoạch . . . 39

9 Tổng kết công việc hiện thực của nhóm **40**

10 Đường dẫn tới từng kết quả hiện thực của nhóm **41**

1 Danh Sách Thành Viên Và Phân Công Nhiệm Vụ

MSSV	Họ và tên	Nhiệm vụ	Phần trăm công việc
2213951	Trần Đại Việt	PO	100%
2213950	Phạm Văn Quốc Việt	DEV	100%
2211629	Nguyễn Nhật Khoa	DEV	100%
2210128	Phạm Việt Anh	DEV	100%
2212303	Nguyễn Gia Nguyên	DEV	100%
2211599	Lê Đăng Khoa	DEV	100%

Bảng 1: *Danh sách sinh viên và nhiệm vụ*

2 Giới Thiệu Chung Về Dự Án

2.1 Bối cảnh chung và tính cấp thiết của đề tài

Trong bối cảnh số lượng sinh viên tại Đại học Bách Khoa - Đại học Quốc gia TP.HCM không ngừng gia tăng qua từng năm, việc quản lý thông tin học tập và điểm số của sinh viên đang trở thành một thách thức ngày càng lớn. Cùng với đó, chương trình đào tạo tại trường liên tục được cập nhật để bắt kịp với sự phát triển nhanh chóng của công nghệ và kiến thức toàn cầu, đặt ra yêu cầu cấp bách về việc xây dựng các hệ thống hỗ trợ quản lý và tra cứu thông tin học tập một cách hiệu quả. Đặc biệt, việc cung cấp một hệ thống cho phép sinh viên có thể tra cứu điểm số trong quá trình học tập là một yếu tố quan trọng, không chỉ giúp sinh viên có thể theo dõi kết quả học tập của mình mà còn tạo điều kiện thuận lợi cho giảng viên trong việc quản lý lớp học và các dữ liệu liên quan đến điểm số.

Hiện nay, tại nhiều trường đại học trên thế giới, việc áp dụng các hệ thống quản lý điểm số thông minh đã trở thành xu hướng phổ biến, nhằm giúp cả sinh viên lẫn giảng viên dễ dàng truy cập, quản lý và cập nhật thông tin về kết quả học tập. Tại Đại học Bách Khoa, nhu cầu này cũng đang trở nên cấp thiết khi số lượng sinh viên ngày càng tăng, đặc biệt là trong những môn học có đông sinh viên tham gia. Việc áp dụng một hệ thống giúp sinh viên có thể tra cứu điểm số, theo dõi quá trình học tập và cập nhật kết quả của mình sẽ giúp giảm bớt áp lực cho cả giảng viên và sinh viên trong việc xử lý các thông tin về điểm số một cách thủ công, đồng thời giúp nâng cao tính minh bạch và chính xác trong việc công bố kết quả học tập.

Vì vậy, dự án Grade Portal được đề xuất với mục tiêu xây dựng một hệ thống quản lý điểm số trực tuyến nhằm hỗ trợ sinh viên và giảng viên tại Đại học Bách Khoa - Đại học Quốc gia TP.HCM trong việc theo dõi, tra cứu và quản lý điểm số một cách thuận tiện và chính xác. Với sự phát triển không ngừng của công nghệ thông tin và nhu cầu hiện đại hóa trong giáo dục, Grade Portal hứa hẹn sẽ là một giải pháp toàn diện giúp nâng cao trải nghiệm học tập và giảng dạy trong nhà trường.

Một trong những yếu tố làm cho hệ thống này trở nên cần thiết là tính minh bạch và khả năng tự động cập nhật của nó. Khi sinh viên có thể tra cứu điểm số một cách thường xuyên , họ sẽ có cái nhìn rõ ràng hơn về tiến trình học tập của mình, từ đó có thể tự điều chỉnh chiến lược học tập một cách phù hợp. Bên cạnh đó, giảng viên sẽ có khả năng đính kèm các bảng điểm dưới dạng tập tin CSV, Excel, giúp họ dễ dàng quản lý và theo dõi các môn học mà mình phụ trách. Ngoài ra, việc hệ thống cho phép admin phân quyền cho giảng viên cũng sẽ tạo điều kiện thuận lợi cho nhà trường trong việc quản lý hệ thống điểm số một cách linh hoạt và hiệu quả.

Một điểm nổi bật khác của hệ thống này là tính năng Hall of Fame, nơi ghi nhận và tôn vinh các sinh viên có thành tích học tập xuất sắc. Điều này không chỉ tạo động lực cho sinh viên phấn đấu học tập mà còn góp phần xây dựng môi trường học tập tích cực và khuyến khích sự nỗ lực từ phía sinh viên. Việc tôn vinh những cá nhân có thành tích cao sẽ khuyến khích tinh thần học tập và cạnh tranh lành mạnh trong toàn trường, từ đó nâng cao chất lượng giảng dạy và học tập.

Ngoài ra, việc xây dựng một hệ thống quản lý điểm số thông minh còn đáp ứng nhu cầu hiện đại hóa quá trình giảng dạy và quản lý học tập tại Đại học Bách Khoa, góp phần cải thiện trải nghiệm của sinh viên và giảng viên. Thay vì phải tra cứu thông tin điểm số thông qua các phương thức truyền thống như email, giấy tờ, hoặc trực tiếp gặp gỡ giảng viên, hệ thống mới sẽ giúp sinh viên truy cập thông tin một cách dễ dàng hơn, tiết kiệm thời gian và nâng cao hiệu quả trong việc quản lý thông tin học tập.

Nhìn chung, nếu thành công, Grade Portal không chỉ là công cụ hỗ trợ đắc lực cho sinh viên và giảng viên mà còn đóng góp quan trọng vào việc hiện đại hóa và nâng cao chất lượng giáo dục tại Đại học Bách Khoa. Hệ thống sẽ trở thành một bước tiến lớn trong quá trình chuyển đổi số trong giáo dục, mang lại hiệu quả cao và đáp ứng được nhu cầu ngày càng tăng của một ngôi trường đại học hiện đại.

2.2 Các bên liên quan (Stakeholders) và nhu cầu của họ

- **Sinh viên:** Sinh viên cần một hệ thống tra cứu điểm số nhanh chóng, minh bạch và dễ dàng truy cập để theo dõi kết quả học tập của mình. Họ mong muốn có thể kiểm tra điểm số một cách tự động và liên tục cập nhật khi có sự thay đổi. Ngoài ra, sinh viên cũng mong đợi hệ thống cung cấp thông tin rõ ràng và có tính bảo mật cao để đảm bảo quyền lợi học tập của mình.
- **Giảng viên và cán bộ công tác tại trường:** Giảng viên cần một hệ thống quản lý điểm số hiệu quả, có khả năng xử lý các tệp dữ liệu lớn (danh sách điểm sinh viên) và hỗ trợ cập nhật liên tục để đảm bảo độ chính xác của điểm số. Họ cần một hệ thống linh hoạt, cho phép dễ dàng đính kèm và cập nhật các bảng điểm từ file CSV, Excel và hệ thống phải tự động tải và cập nhật khi có phiên bản mới.
- **HCMUT Administrator:** Admin có nhiệm vụ quản lý người dùng và phân quyền cho giảng viên trong hệ thống. Họ cần một hệ thống dễ sử dụng để phân quyền, quản lý tài khoản của giảng viên và sinh viên, đồng thời phải đảm bảo bảo mật thông tin trong suốt quá trình vận hành hệ thống. Admin cũng cần đảm bảo tính ổn định của hệ thống trong quá trình cập nhật và xử lý dữ liệu.

2.3 Lợi ích nhận được khi dự án hoàn thành

- **Sinh viên:** Sinh viên sẽ có thể dễ dàng tra cứu điểm số một cách chính xác và cập nhật kịp thời trong quá trình học tập. Hệ thống giúp sinh viên theo dõi kết quả học tập của mình một cách minh bạch và rõ ràng, từ đó điều chỉnh chiến lược học tập phù hợp. Việc truy cập bảng điểm qua hệ thống trực tuyến sẽ giúp sinh viên tiết kiệm thời gian và tăng cường sự chủ động trong quá trình học tập. Đặc biệt, tính năng Hall of Fame sẽ tạo động lực cho sinh viên phấn đấu đạt thành tích cao.
- **Giảng viên và cán bộ công tác tại trường:** Giảng viên sẽ được hỗ trợ tối đa trong việc quản lý và cập nhật điểm số. Họ có thể dễ dàng tải lên và quản lý các bảng điểm từ file CSV, và hệ thống sẽ tự động cập nhật khi có thay đổi. Điều này giúp giảm thiểu khối lượng công việc thủ công, đồng thời đảm bảo tính chính xác và minh bạch trong quá trình chấm điểm và công bố kết quả.
- **HCMUT Administrator:** Hệ thống sẽ cung cấp cho Admin khả năng quản lý người dùng và phân quyền một cách dễ dàng, đồng thời đảm bảo tính bảo mật trong việc quản lý thông tin của sinh viên và giảng viên. Ngoài ra, Admin cũng có thể giám sát toàn bộ hệ thống để đảm bảo quá trình vận hành trơn tru, đồng thời xử lý các vấn đề phát sinh nhanh chóng và hiệu quả.

3 Phân Tích Các Yêu Cầu Của Dự Án

3.1 Các yêu cầu chức năng (Functional Requirements)

1. **Admin:**
   - Phải trải qua bước đăng nhập để xác thực quyền truy cập vào hệ thống thông qua Google Authentication.
   - Có thể đăng xuất khỏi hệ thống sau khi thực hiện xong các hành động của mình.
   - Có khả năng thêm mới và quản lý tài khoản giảng viên.
   - Có khả năng tạo lớp học mới, gán sinh viên và giảng viên phụ trách vào lớp học.
   - Có thể phân quyền cho giảng viên quản lý lớp học tương ứng.
   - Có thể thêm hàng loạt sinh viên vào hệ thống bằng danh sách Gmail do trường cung cấp.
   - Được cung cấp các hàm chức năng cho việc cập nhật và chỉnh sửa dữ liệu các đối tượng trong hệ thống.
2. **Giảng viên:**
   - Phải trải qua bước đăng nhập để xác thực quyền truy cập vào hệ thống thông qua Gmail trường cấp.
   - Có thể đăng xuất khỏi hệ thống sau khi thực hiện xong các hoạt động của mình.
   - Có khả năng quản lý các lớp học được phân quyền bởi admin.
   - Có thể thêm sinh viên vào lớp học mình quản lý.
   - Có thể tải lên file CSV hoặc Excel chứa bảng điểm cho môn học.
   - Hệ thống sẽ tự động giám sát và đồng bộ file CSV khi có phiên bản mới và cập nhật điểm vào cơ sở dữ liệu.
3. **Sinh viên:**
   - Phải trải qua bước đăng nhập để xác thực quyền truy cập vào hệ thống thông qua Gmail trường cấp.
   - Có thể tra cứu và xem điểm của mình cho tất cả các môn học đã đăng ký.
   - Có thể tìm kiếm điểm theo môn học, học kỳ, năm học hoặc MSSV.
   - Có thể gửi phản hồi trực tiếp đến giảng viên phụ trách thông qua hệ thống (tùy chọn).
   - Có thể xem thông tin chi tiết về tình trạng lớp học đang tham gia và giảng viên phụ trách lớp đó.
4. **Chức năng Bảng Vinh Danh (Hall of Fame):**
   - Hệ thống phải có một Bảng Vinh Danh, nơi liệt kê các sinh viên có thành tích xuất sắc dựa trên điểm trung bình tích lũy (GPA) hoặc kết quả môn học cụ thể.
   - Tiêu chí để được liệt kê vào Bảng Vinh Danh có thể do Admin hoặc giảng viên thiết lập.

3.2 Các yêu cầu phi chức năng (Non-functional Requirements)

1. **Bảo mật (Security Requirements)**
   - Tất cả người dùng (Admin, giảng viên, sinh viên) đều phải xác thực bằng tài khoản Gmail trường cấp trước khi sử dụng hệ thống.
   - Thông tin về điểm số, dữ liệu bảng điểm và các thông tin cá nhân liên quan đến sinh viên và giảng viên phải được bảo mật, chỉ những người có quyền truy cập mới có thể xem hoặc chỉnh sửa.
   - Hệ thống phải đảm bảo an toàn cho dữ liệu CSV được tải lên và đồng bộ hóa, chỉ có giảng viên phụ trách lớp học và admin mới có quyền chỉnh sửa các bảng điểm này.
2. **Hiệu suất hoạt động (Performance)**
   - Hệ thống phải hỗ trợ nhiều người dùng đồng thời mà không bị giảm hiệu suất, đặc biệt là trong các thời điểm cao điểm như cuối học kỳ khi sinh viên tra cứu điểm.
   - Hệ thống phải có khả năng phản hồi nhanh chóng, đặc biệt đối với các thao tác tải lên file CSV, Excel hoặc khi tra cứu thông tin điểm số.
   - Hệ thống phải cung cấp phản hồi ngay lập tức về việc tải lên thành công hay thất bại của bảng điểm, cũng như khi sinh viên thực hiện tra cứu điểm.

3. **Khả năng sử dụng (Usability)**
    - Giao diện người dùng (UI) của hệ thống phải rõ ràng, trực quan, dễ dàng sử dụng để giảng viên và sinh viên có thể tra cứu, tải lên, hoặc chỉnh sửa thông tin một cách nhanh chóng.
    - Hệ thống phải dễ sử dụng để người dùng (giảng viên, sinh viên) có thể nhanh chóng nắm bắt cách thao tác, ngay cả khi họ sử dụng lần đầu.
    - Đảm bảo số lần thao tác tối thiểu để truy cập vào tính năng tra cứu điểm hoặc cập nhật bảng điểm.
    - Hệ thống cần cung cấp các thông báo lỗi rõ ràng khi có vấn đề xảy ra, chẳng hạn như file không hợp lệ, lỗi trong quá trình đồng bộ dữ liệu, hoặc kết nối gián đoạn.
4. **Toàn vẹn dữ liệu (Data Integrity)**
    - Mỗi yêu cầu tải lên bảng điểm và tra cứu điểm số phải ghi lại chính xác thông tin như ID sinh viên, môn học, thời gian tải lên, và điểm số tương ứng.
    - Hệ thống phải đảm bảo rằng các dữ liệu bảng điểm không thể bị chỉnh sửa hoặc xóa bỏ bởi những người không có thẩm quyền.
    - Mọi lịch sử và báo cáo liên quan đến việc tải lên và chỉnh sửa bảng điểm phải được duy trì toàn vẹn và bảo mật, đảm bảo rằng không có sự thay đổi dữ liệu trái phép.
5. **Tính sẵn sàng (Availability)**
    - Hệ thống phải luôn sẵn sàng phục vụ giảng viên và sinh viên trong suốt thời gian học tập và giảng dạy (6h-22h mỗi ngày).
    - Hệ thống phải được bảo trì định kỳ mà không làm gián đoạn hoạt động của người dùng. Các thông báo bảo trì phải được gửi trước để người dùng có thể sắp xếp công việc.
    - Hệ thống phải có khả năng xử lý tình huống khẩn cấp, đảm bảo rằng dữ liệu điểm số không bị mất hoặc hư hỏng do lỗi hệ thống.
6. **Khả năng mở rộng (Scalability)**
    - Hệ thống phải được thiết kế để có thể dễ dàng mở rộng khi số lượng sinh viên và giảng viên sử dụng hệ thống tăng lên trong tương lai.
7. **Khả năng bảo trì (Maintainability)**
    - Tất cả các công nghệ và chi tiết kỹ thuật sử dụng trong dự án phải được ghi lại rõ ràng trong tài liệu kỹ thuật, đảm bảo dễ dàng tra cứu và bảo trì.
    - Hệ thống phải có kế hoạch kiểm tra và nâng cấp định kỳ nhằm đảm bảo luôn hoạt động ổn định.
    - Các lỗi liên quan đến việc đồng bộ bảng điểm hoặc hệ thống phải được thông báo và khắc phục kịp thời.

4 Kiến trúc hệ thống (System Architecture)

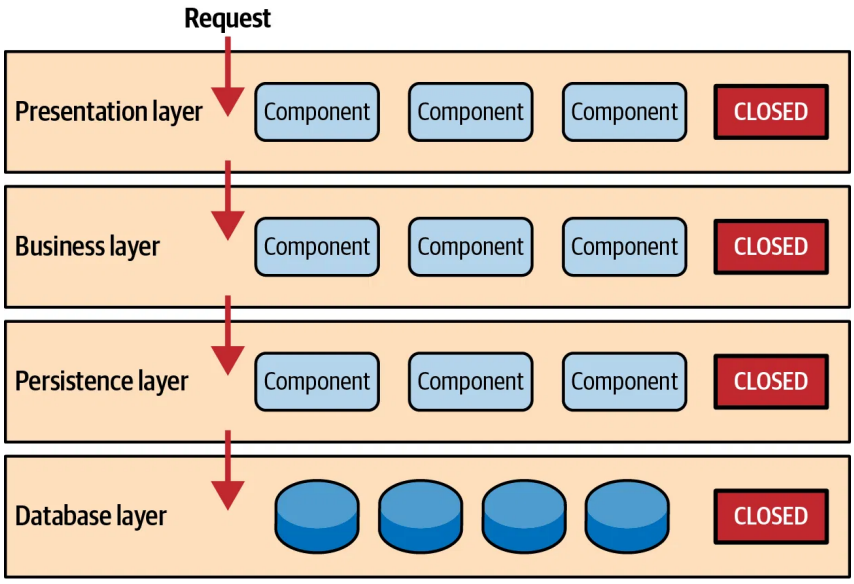
4.1 Layered Architecture

Khi phát triển một hệ thống như HCMUT Grade Portal, việc xác định kiến trúc hệ thống là yếu tố vô cùng quan trọng. Kiến trúc hệ thống đóng vai trò là "bộ khung" tổng thể, giúp định hình cách các thành phần trong hệ thống tương tác với nhau và đảm bảo hệ thống vận hành hiệu quả, an toàn và dễ bảo trì. Một kiến trúc được thiết kế tốt không chỉ hỗ trợ việc phát triển ban đầu mà còn đảm bảo hệ thống có khả năng mở rộng trong tương lai, đáp ứng được các nhu cầu mới mà không gây ảnh hưởng lớn đến các phần khác của hệ thống. Đặc biệt, với hệ thống yêu cầu độ chính xác cao khi tương tác với nhiều đối tượng thành phần như Grade Portal việc có một kiến trúc chặt chẽ và dễ bảo trì sẽ giúp việc vận hành và mở rộng hệ thống trở nên đơn giản hơn.

Mặc dù kiến trúc **Monolithic** có một số hạn chế, nhưng trong trường hợp của Grade Portal, đây vẫn là lựa chọn khả thi. Hạn chế chính của kiến trúc monolithic bao gồm việc khó mở rộng khi hệ thống lớn dần lên, khả năng bảo trì thấp khi phải thay đổi một phần nhỏ của hệ thống nhưng có thể ảnh hưởng đến toàn bộ ứng dụng, và độ phức tạp gia tăng khi tích hợp các chức năng mới. Việc kiểm tra và triển khai một thay đổi nhỏ trong hệ thống monolithic có thể dẫn đến việc phải triển khai lại toàn bộ hệ thống, gây ra thời gian ngừng hoạt động và tăng nguy cơ lỗi.

Tuy nhiên, vẫn có lý do để chọn kiến trúc **Monolithic** ở giai đoạn này. Đầu tiên, kiến trúc monolithic dễ triển khai và phát triển nhanh hơn, đặc biệt trong giai đoạn khởi đầu của dự án. Với một đội ngũ nhỏ và khi chưa có nhiều yêu cầu phức tạp về khả năng mở rộng, việc tập trung vào một ứng dụng duy nhất giúp tiết kiệm thời gian phát triển, đơn giản hóa việc quản lý code và dễ dàng kiểm thử hệ thống. Hơn nữa, với quy mô của Grade Portal ở giai đoạn ban đầu, hệ thống vẫn có thể duy trì được hiệu suất tốt khi hoạt động dưới dạng một ứng dụng monolithic. Khi hệ thống lớn dần và có nhiều tính năng mới, việc chuyển đổi sang các kiến trúc phức tạp hơn (như microservices) hoàn toàn có thể được cân nhắc sau này.

Ngoài ra để có thể nhanh chóng nắm rõ kiến trúc cần hiện thực, nhóm chúng em quyết định sẽ xây dựng hệ thống theo kiến trúc phân lớp (**Layered Architecture**), một kiến trúc thuộc loại thuộc loại kiến trúc monolithic, để đảm bảo tính tổ chức và dễ bảo trì trong quá trình phát triển. Layered Architecture sẽ chia hệ thống thành nhiều tầng (layers) với các chức năng cụ thể, đảm bảo mỗi tầng chỉ chịu trách nhiệm cho một nhiệm vụ cụ thể và độc lập với các tầng khác.

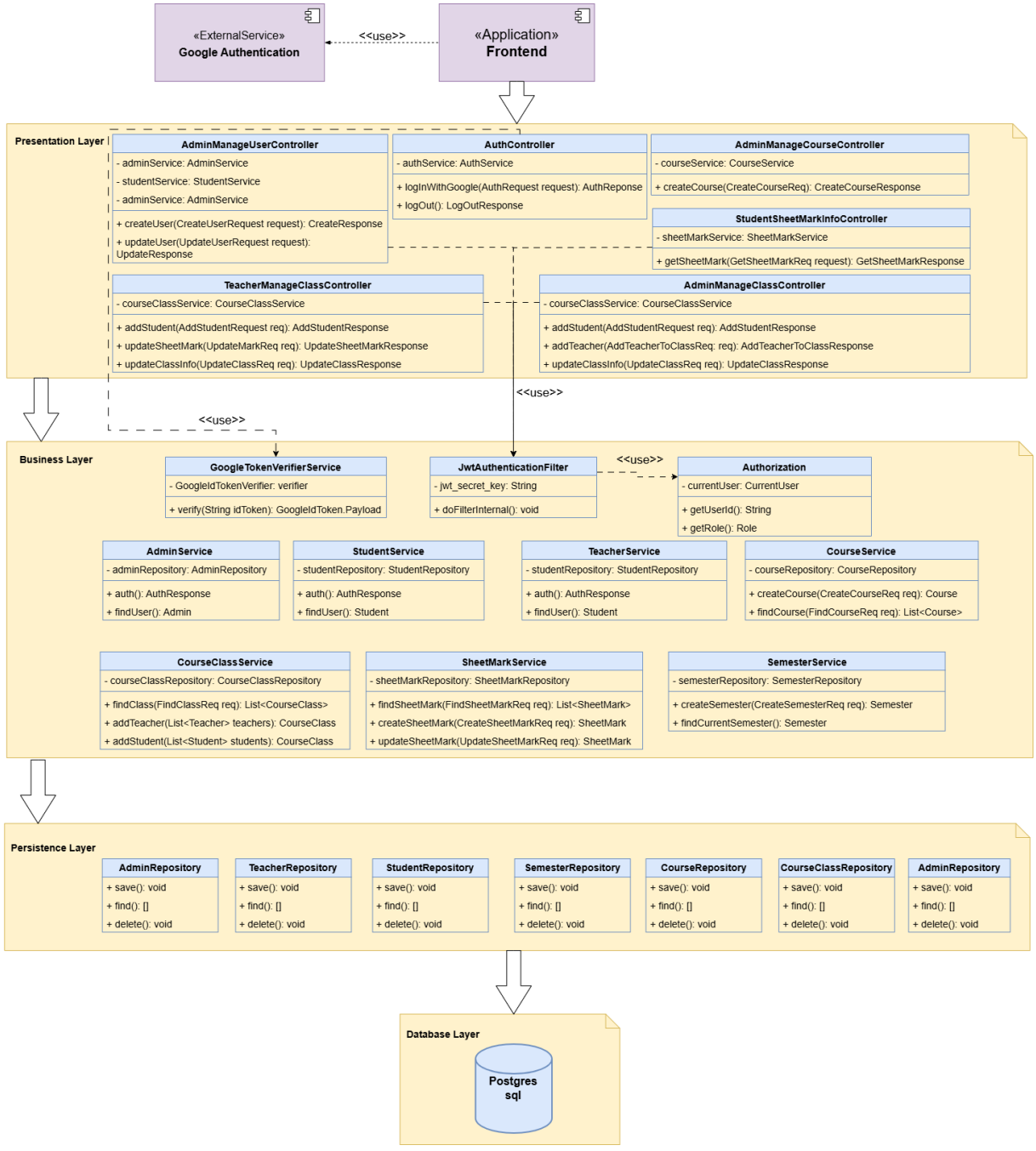


Hình 1: *Sơ đồ kiến trúc phân tầng*

Đây là một architecture pattern 3 lớp vật lý (3-tier) hay n-tier. Tất cả logic điều được move lên phía server, do đó giải quyết triệt để vấn đề về mở rộng, client bây giờ chỉ làm nhiệm vụ render mà không cần biết các thay đổi từ server ra sao.

4.2 Sơ đồ hiện thực của hệ thống

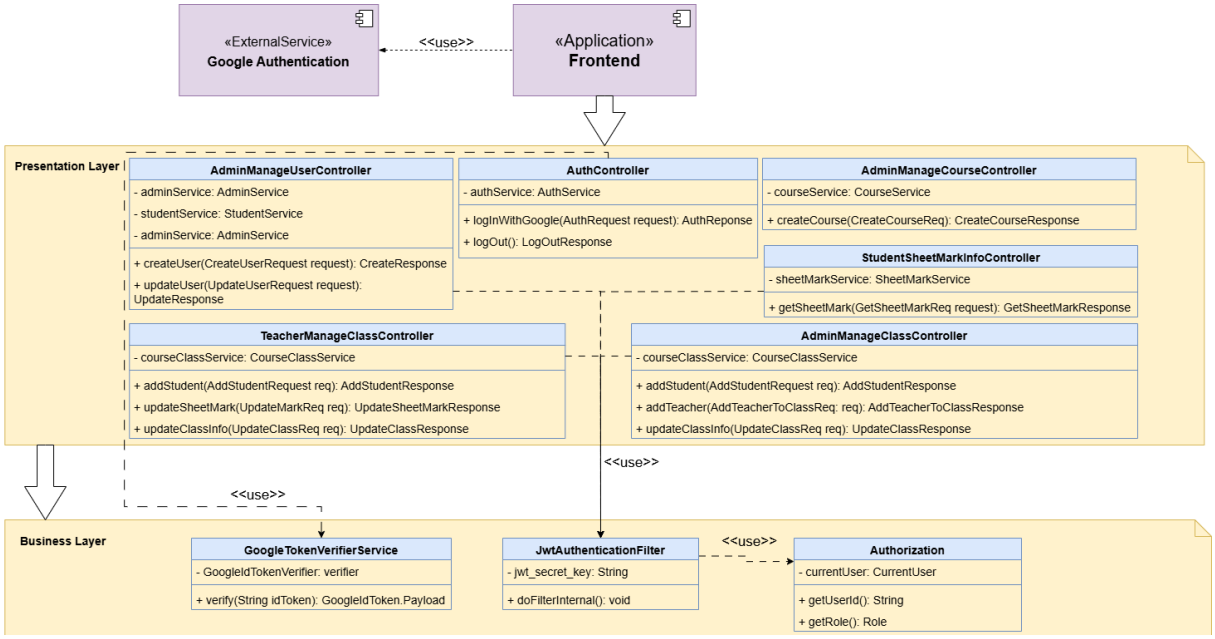
Trong phần này nhóm đã sử dụng **Draw.io** để hiện thực sơ đồ. Đường dẫn cụ thể tới sơ đồ được hiện thực nằm ở **đây**.



Hình 2: *Sơ đồ hiện thực chi tiết hệ thống*

Áp dụng kiến trúc **Layered Architecture** vào việc hiện thực Server của hệ thống. Luồng dữ liệu được thiết kế chạy theo một luồng đơn hướng, có nghĩa là Layer cao hơn sẽ có khả năng kéo được dữ liệu từ các Layer thấp hơn, nhưng ngược lại, các Layer thấp hơn sẽ không được kéo dữ liệu được từ các Layer trên mình. Điều đó tuy đảm bảo sự tách biệt rõ ràng giữa các Layer, tăng cường khả năng bảo trì và khả năng mở rộng của hệ thống nhưng nếu chúng ta không tuân theo, chúng ta có thể gặp phải vấn đề như phụ thuộc tuần hoàn (Dependency Injection).

4.2.1 Presentation Layer



Hình 3: *Presentation Layer Of Layered Architecture*

Presentation Layer là tầng đầu tiên trong kiến trúc hệ thống, đảm nhận vai trò trung gian giữa **Client** (có thể là Web App hoặc Mobile App) và hệ thống phía dưới. Tầng này chịu trách nhiệm tiếp nhận các yêu cầu từ phía người dùng thông qua API, sau đó chuyển tiếp xuống tầng **Business Layer** để thực hiện các logic xử lý và trả về kết quả phù hợp.

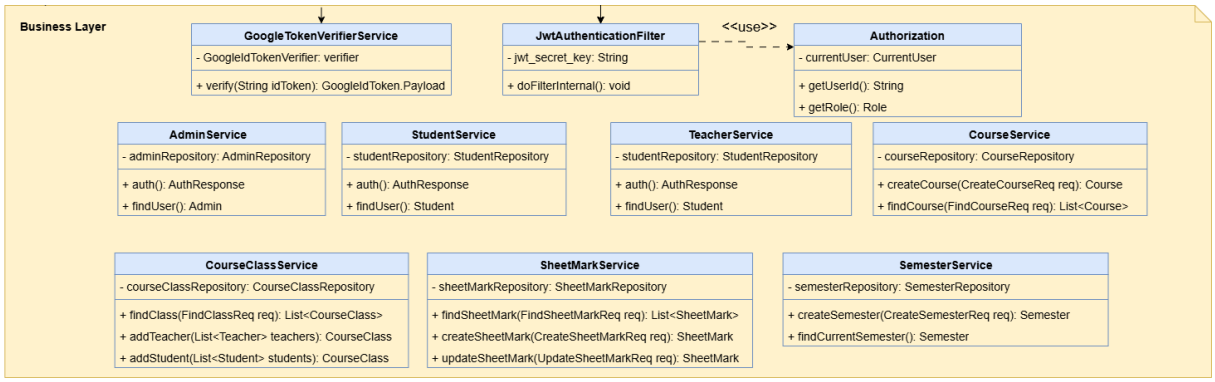
- **Vai trò chính**:
  - Tiếp nhận và xử lý dữ liệu từ Client (Web App hoặc Mobile App).
  - Kiểm tra tính đúng đắn và đầy đủ của dữ liệu đầu vào.
  - Chuyển dữ liệu hợp lệ xuống tầng **Business Layer** để thực thi logic nghiệp vụ.
  - Định dạng và đảm bảo dữ liệu phản hồi trả về Client theo đúng cấu trúc đã được mô tả trong tài liệu (Document).
  - Sử dụng các cơ chế bảo mật (security) để xác thực và phân quyền, đảm bảo rằng chỉ những người dùng có vai trò (**Role**) phù hợp mới được phép truy cập các API cụ thể.
- **Các thành phần chính**: Dựa vào sơ đồ, Presentation Layer bao gồm các **Controller** đóng vai trò chính trong việc xử lý yêu cầu từ Client:
  - **AdminManageUserController**:
    - Chịu trách nhiệm quản lý người dùng.
    - Ví dụ các chức năng: tạo người dùng mới (`createUser`) và cập nhật thông tin người dùng (`updateUser`).
    - Sử dụng xác thực và phân quyền để đảm bảo chỉ người dùng có quyền `ADMIN` mới có thể truy cập các API này.
  - **AuthController**:
    - Đảm nhiệm việc xác thực người dùng thông qua dịch vụ **Google Authentication**.
    - Ví dụ các chức năng: đăng nhập với Google (`logInWithGoogle`) và đăng xuất (`logOut`).
    - Dựa vào lớp `JwtAuthenticationFilter` để tạo và kiểm tra **JWT Token** nhằm xác thực người dùng.
  - **AdminManageCourseController**:
    - Quản lý các khóa học trong hệ thống.
    - Ví dụ chức năng: tạo mới khóa học (`createCourse`).

    - Kiểm tra quyền truy cập để đảm bảo chỉ `ADMIN` mới được phép thao tác các API này.
  - **TeacherManageClassController**:
    - Chịu trách nhiệm quản lý lớp học và cập nhật thông tin học sinh.
    - Ví dụ chức năng: thêm học sinh vào lớp (`addStudent`) và cập nhật điểm số (`updateSheetMark`).
    - Sử dụng phân quyền để đảm bảo rằng chỉ người dùng có vai trò `TEACHER` mới được truy cập các API này.
  - **StudentSheetMarkInfoController**:
    - Tiếp nhận yêu cầu liên quan đến điểm số của học sinh.
    - Ví dụ chức năng: lấy thông tin điểm số (`getSheetMark`).
    - Phân quyền để đảm bảo chỉ học sinh có quyền truy cập vào điểm số của chính mình.
  - **AdminManageClassController**:
    - Quản lý lớp học bao gồm học sinh và giáo viên.
    - Ví dụ chức năng: thêm học sinh vào lớp (`addStudent`), thêm giáo viên vào lớp (`addTeacher`) và cập nhật thông tin lớp học (`updateClassInfo`).
    - Phân quyền để đảm bảo chỉ người dùng có vai trò `ADMIN` hoặc `TEACHER` mới được phép thực hiện các chức năng cụ thể.

- **Cơ chế bảo mật và phân quyền**:
  - Tầng **Presentation Layer** kết hợp với các **Service** như `JwtAuthenticationFilter` và `Authorization` để thực hiện xác thực người dùng bằng **JWT Token**.
  - Dựa trên thông tin người dùng (ví dụ: `Role`) trong Token, hệ thống kiểm tra quyền truy cập và chỉ cho phép các vai trò phù hợp gọi đến API.
  - Điều này đảm bảo rằng chỉ những người dùng được phép mới có quyền thực thi các chức năng tương ứng.

Tóm lại, tầng Presentation Layer không chỉ tiếp nhận và xử lý dữ liệu từ Client mà còn thực thi các cơ chế bảo mật và phân quyền, đảm bảo hệ thống hoạt động an toàn và đúng với logic nghiệp vụ.

4.2.2 Business Layer



Hình 4: *Business Layer Of Layered Architecture*

Business Layer là tầng thứ hai trong kiến trúc hệ thống, đảm nhiệm vai trò hiện thực **logic nghiệp vụ** của hệ thống. Tầng này đảm bảo rằng mọi quy trình, yêu cầu từ tầng **Presentation Layer** được thực thi đúng với nghiệp vụ đã định nghĩa.

- **Vai trò chính**:
  - Thực thi các logic nghiệp vụ cụ thể của hệ thống dựa trên dữ liệu nhận được từ tầng **Presentation Layer**.
  - Gọi tới tầng **Persistence Layer** để truy vấn hoặc cập nhật dữ liệu trong Database khi cần thiết.
  - Tổ chức logic nghiệp vụ thành các **Service** (module) riêng biệt nhằm đảm bảo tính tái sử dụng và dễ bảo trì.

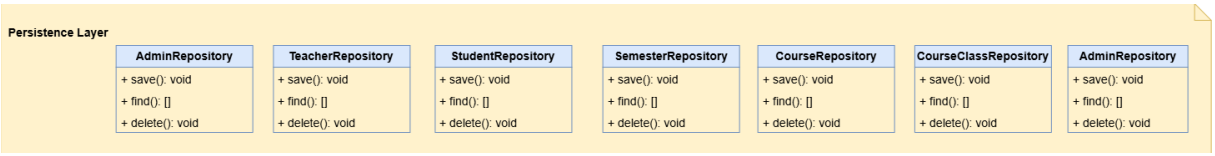
  - Giảm thiểu sự phụ thuộc của tầng **Presentation Layer** vào logic xử lý nội bộ, giúp các API không bị ảnh hưởng khi có thay đổi trong logic nghiệp vụ.

- **Các thành phần chính**: Dựa vào sơ đồ, tầng này bao gồm các **Service** đảm nhận việc xử lý logic nghiệp vụ cho từng khía cạnh của hệ thống:
  - **GoogleTokenVerifierService**:
    * Xác thực Token ID Google để đảm bảo tính hợp lệ.
    * Chức năng: `verify` – xác thực token và trả về thông tin người dùng.
  - **JwtAuthenticationFilter**:
    * Đảm nhiệm việc xác thực **JWT Token** để bảo vệ các API.
    * Chức năng: `doFilterInternal` – lọc và xác thực token của mỗi request.
  - **Authorization**:
    * Chịu trách nhiệm phân quyền dựa trên thông tin người dùng.
    * Chức năng: lấy ID người dùng (`getUserId`) và vai trò (`getRole`).
  - **AdminService**:
    * Xử lý nghiệp vụ liên quan đến quản lý admin.
    * Chức năng: `auth` (xác thực) và `findUser` (tìm thông tin admin).
  - **StudentService**:
    * Xử lý nghiệp vụ cho học sinh.
    * Chức năng: `auth` (xác thực) và `findUser` (tìm thông tin học sinh).
  - **TeacherService**:
    * Xử lý nghiệp vụ liên quan đến giáo viên.
    * Chức năng: `auth` (xác thực) và `findUser` (tìm thông tin giáo viên).
  - **CourseService**:
    * Xử lý nghiệp vụ liên quan đến khóa học.
    * Chức năng: `createCourse` (tạo khóa học mới) và `findCourse` (tìm danh sách khóa học).
  - **CourseClassService**:
    * Xử lý nghiệp vụ liên quan đến lớp học.
    * Chức năng: tìm lớp học (`findClass`), thêm giáo viên vào lớp (`addTeacher`) và thêm học sinh vào lớp (`addStudent`).
  - **SheetMarkService**:
    * Xử lý nghiệp vụ liên quan đến điểm số học sinh.
    * Chức năng: tìm điểm số (`findSheetMark`), tạo mới điểm số (`createSheetMark`) và cập nhật điểm số (`updateSheetMark`).
  - **SemesterService**:
    * Xử lý nghiệp vụ liên quan đến học kỳ.
    * Chức năng: `createSemester` (tạo học kỳ mới) và `findCurrentSemester` (lấy học kỳ hiện tại).

- **Cơ chế bảo mật**:
  - Tầng **Business Layer** kết hợp với các lớp như `GoogleTokenVerifierService`, `JwtAuthenticationFilter` và `Authorization` để xác thực và phân quyền người dùng.
  - Các service sẽ gọi đến các cơ chế bảo mật để đảm bảo chỉ những người dùng có vai trò phù hợp (`ADMIN`, `TEACHER`, `STUDENT`) mới được phép thực hiện các thao tác tương ứng.

Tóm lại, tầng **Business Layer** đóng vai trò trung tâm trong việc hiện thực logic nghiệp vụ của hệ thống. Với việc tổ chức thành các module riêng biệt, tầng này vừa đảm bảo tính tái sử dụng của code, vừa giảm thiểu sự phụ thuộc giữa các tầng và đảm bảo tính bảo mật trong hệ thống.

4.2.3 Persistence Layer



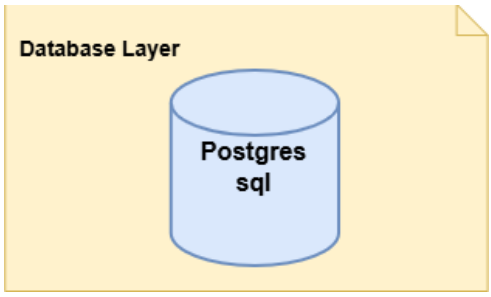
Hình 5: *Persistence Layer Of Layered Architecture*

Persistence Layer là tầng chịu trách nhiệm chính trong việc tương tác và thao tác với **Database** của hệ thống. Tầng này đóng vai trò trung gian giữa tầng **Business Layer** và nơi lưu trữ dữ liệu thực tế, giúp tách biệt logic nghiệp vụ khỏi các hoạt động truy vấn dữ liệu.

- **Vai trò chính**:
  - Gửi các yêu cầu truy vấn và cập nhật dữ liệu trực tiếp tới **Database**.
  - Cung cấp các hàm trừu tượng thông qua các **Repository Interface** giúp nhà phát triển dễ dàng định nghĩa và sử dụng.
  - Tự động tạo các câu lệnh truy vấn cần thiết mà không cần viết tay khi sử dụng các framework như **Spring Data JPA**.
  - Giảm sự phụ thuộc vào hệ quản trị cơ sở dữ liệu cụ thể (ví dụ: PostgreSQL, MySQL, MongoDB), giúp dễ dàng thay đổi nhà cung cấp dịch vụ lưu trữ khi cần.
- **Các thành phần chính**: Dựa vào sơ đồ, tầng này bao gồm các **Repository Interface**, đại diện cho từng thực thể (Entity) trong hệ thống:
  - **AdminRepository**:
    - Cung cấp các chức năng: `save()` để lưu dữ liệu, `find()` để truy vấn dữ liệu và `delete()` để xóa dữ liệu.
  - **TeacherRepository**:
    - Cung cấp các chức năng: `save()`, `find()` và `delete()`.
  - **StudentRepository**:
    - Đảm bảo các thao tác lưu trữ, tìm kiếm và xóa dữ liệu liên quan đến học sinh.
    - Chức năng: `save()`, `find()`, `delete()`.
  - **SemesterRepository**:
    - Đảm nhiệm lưu trữ và truy xuất dữ liệu liên quan đến học kỳ.
    - Chức năng: `save()`, `find()`, `delete()`.
  - **CourseRepository**:
    - Quản lý dữ liệu liên quan đến các khóa học.
    - Chức năng: `save()`, `find()`, `delete()`.
  - **CourseClassRepository**:
    - Chịu trách nhiệm lưu trữ và truy vấn dữ liệu liên quan đến lớp học.
    - Chức năng: `save()`, `find()`, `delete()`.
- **Tính linh hoạt và tiện lợi**:
  - Sử dụng các **Repository Interface** giúp tự động hóa việc tạo câu truy vấn dữ liệu thông qua các phương pháp mặc định như `save()`, `find()`, `delete()`.
  - Dễ dàng mở rộng và tùy chỉnh các phương thức truy vấn dựa trên nhu cầu cụ thể của hệ thống.
  - Giúp thay đổi hệ quản trị cơ sở dữ liệu đơn giản, đảm bảo tính linh hoạt trong bảo trì và phát triển.

Tóm lại, tầng **Persistence Layer** chịu trách nhiệm tương tác với nơi lưu trữ dữ liệu của hệ thống thông qua các **Repository Interface**. Việc sử dụng tầng này giúp hệ thống dễ bảo trì, giảm sự phụ thuộc vào logic lưu trữ và tăng khả năng mở rộng khi có nhu cầu thay đổi hệ quản trị dữ liệu.

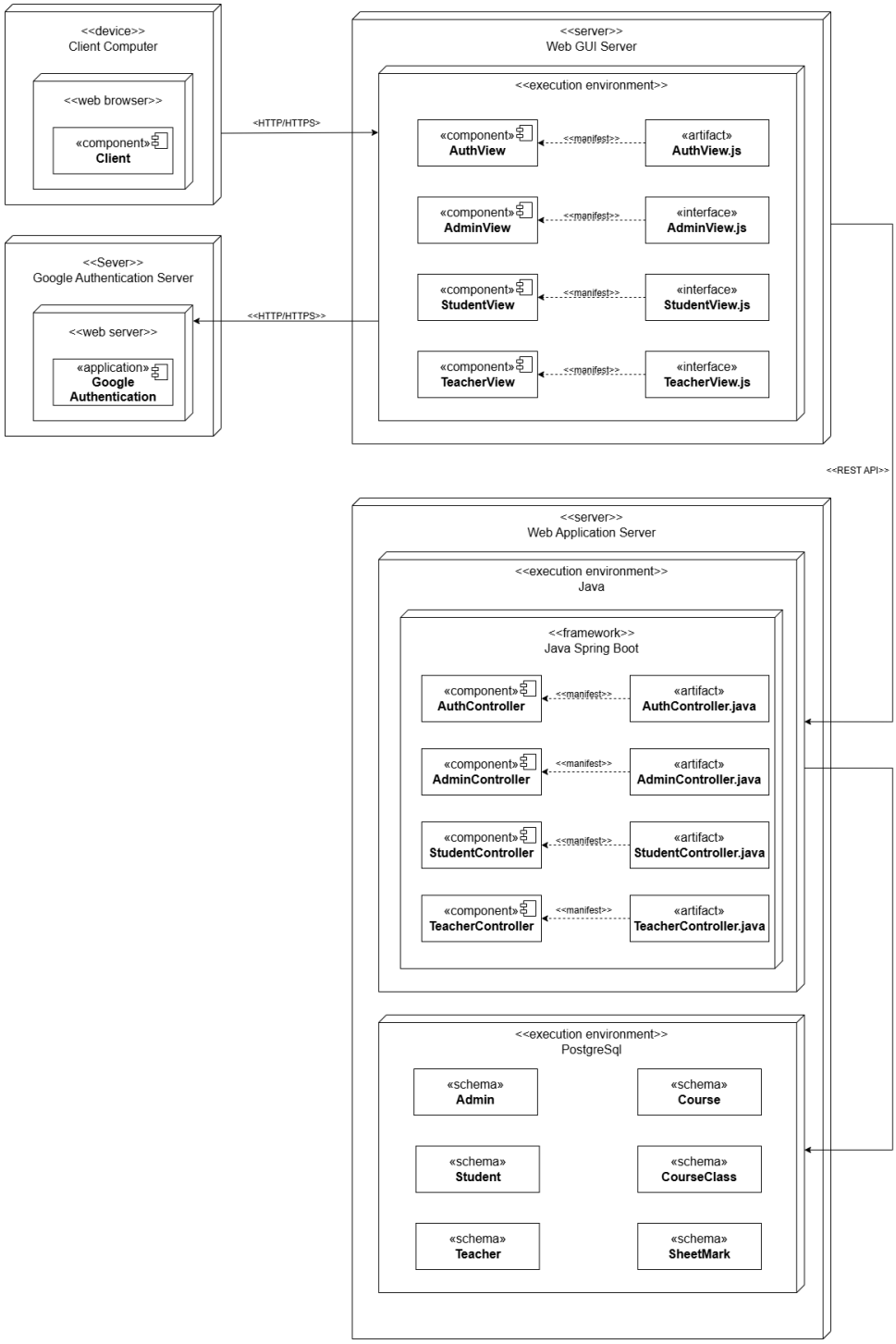
4.2.4 Data Layer



Hình 6: *Database Layer Of Layered Architecture*

Đây là tầng thấp nhất được hiện thực trong kiến trúc, cũng như đây là nơi lưu trữ dữ liệu cho toàn bộ Logic nghiệp vụ của hệ thống. Nhìn chung đây là cơ sở dữ liệu và các thành phần liên quan như hệ quản trị cơ sở dữ liệu, có nhiệm vụ quản lý cơ sở dữ liệu, thực hiện thao tác đọc và ghi dữ liệu và triển khai các truy vấn và lưu trữ dữ liệu theo cách được định nghĩa từ lớp Persistence.

4.3 Deployment Diagram



Hình 7: *Deployment Diagram*

Sau khi đọc yêu cầu của hệ thống đề ra, nhóm chúng em quyết định xây dựng bản vẽ Deployment Diagram cho hệ thống này. Tuy hệ thống cần hiện thực trong dự án này vẫn trong giai đoạn khởi đầu, nghiên cứu và tính chất kiến thức yêu cầu vẫn còn cơ bản, chúng em quyết định sẽ chia hệ thống thành 2 server riêng, một dành cho phía giao diện người dùng và phần còn lại dành cho các Logic nghiệp vụ được hiện thực trong hệ thống. Việc xây dựng mô hình này sẽ giúp cho hệ thống đảm bảo an ninh, sức chịu đựng khi có đông người sử dụng và khả năng mở rộng hệ thống.

4.4 Data Storage Approach

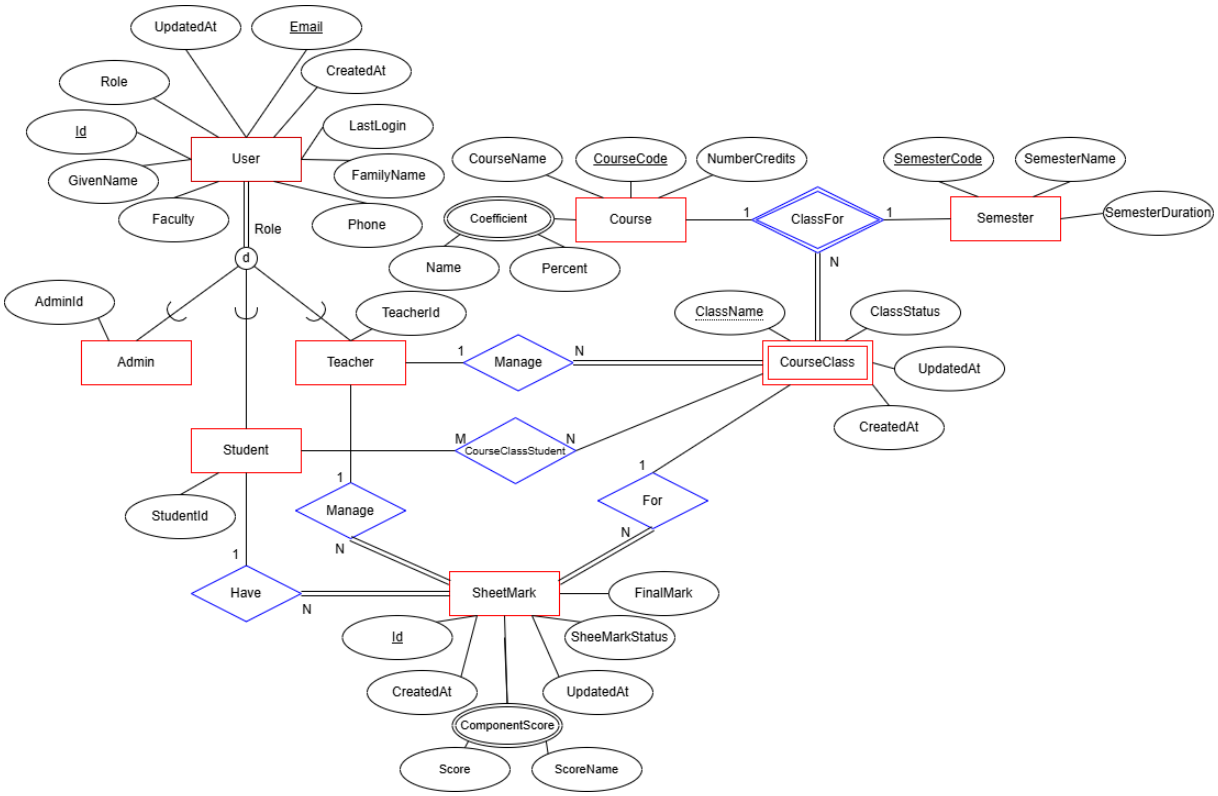
Đối với các yêu cầu của hệ thống Grade Portal, không khó để nhận ra, các hành động liên quan tới việc truy xuất, xử lý và cập nhập dữ liệu là rất quan trọng và cần sự chính xác cao. Và đó cũng chính là một trong những lí do kiến trúc phân tầng được chọn để hiện thực hệ thống, các thao tác liên quan tới việc xử lý dữ liệu hệ thống sẽ được tập trung ở tầng Persistence để tạo sự nhất quán và tránh sự phân tán cấu trúc không rõ ràng.

Ngoài ra, dữ liệu của hệ thống sẽ được lưu bằng cơ sở dữ liệu quan hệ cụ thể là sử dụng hệ quản trị cơ sở dữ liệu PostgreSql, có nghĩa là dữ liệu trong hệ thống sẽ được chia thành các thực thể (Entity) với cấu trúc các trường thuộc tính cùng các mối quan hệ được mô tả rõ ràng trong các Class được khai báo với Anotation @Table trong dự án.

Để đảm bảo hỗ trợ và cung cấp dữ liệu đầy đủ cho các Logic nghiệp vụ hệ thống, sau đây là các sơ đồ EERD, Relational Mapping hay class Diagram để chúng ta có cái nhìn chi tiết nhất về cách các thực thể được lưu trong trong Database và mối quan hệ giữa các thực thể đó.

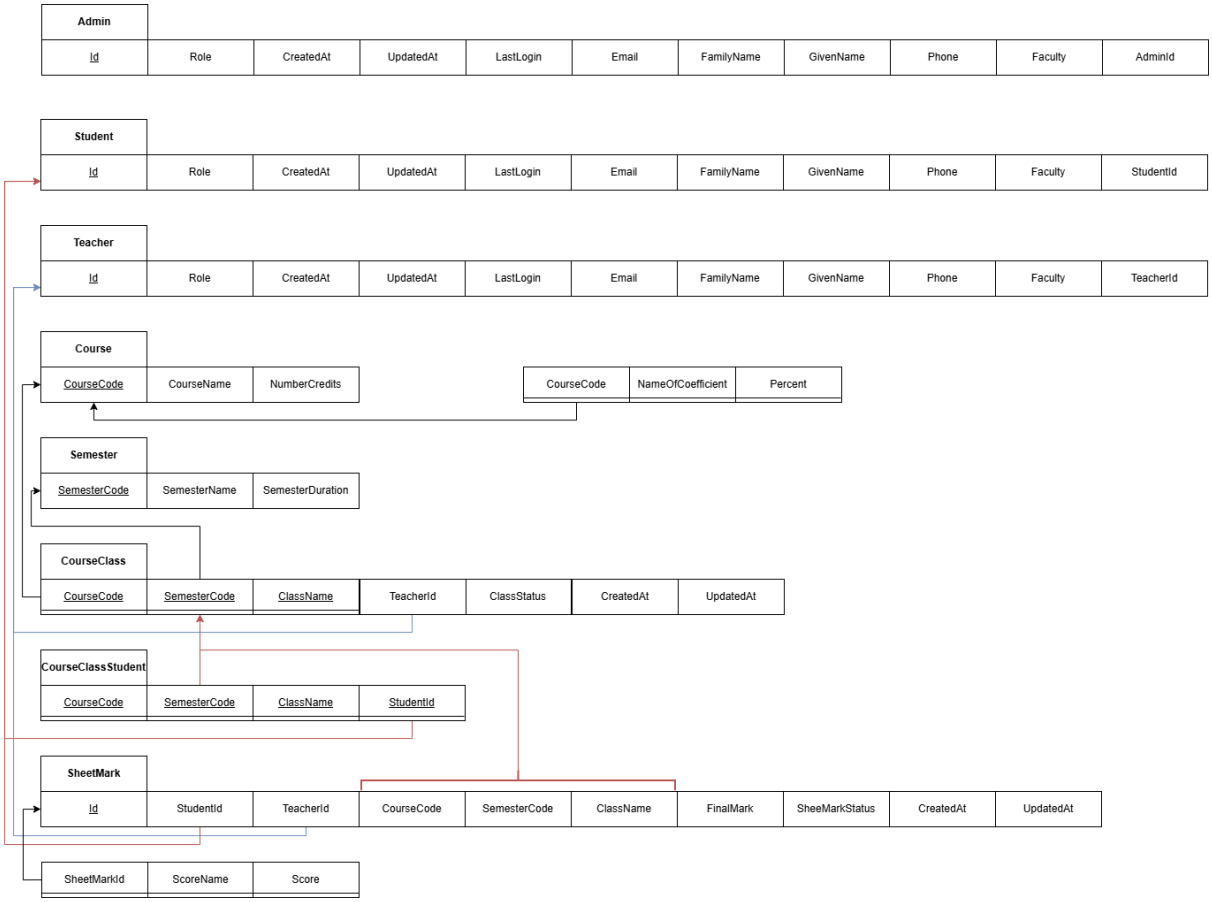
Trong phần này nhóm đã sử dụng **Draw.io** để hiện thực sơ đồ. Đường dẫn cụ thể tới sơ đồ được hiện thực nằm ở **đây**.

EERD Diagram



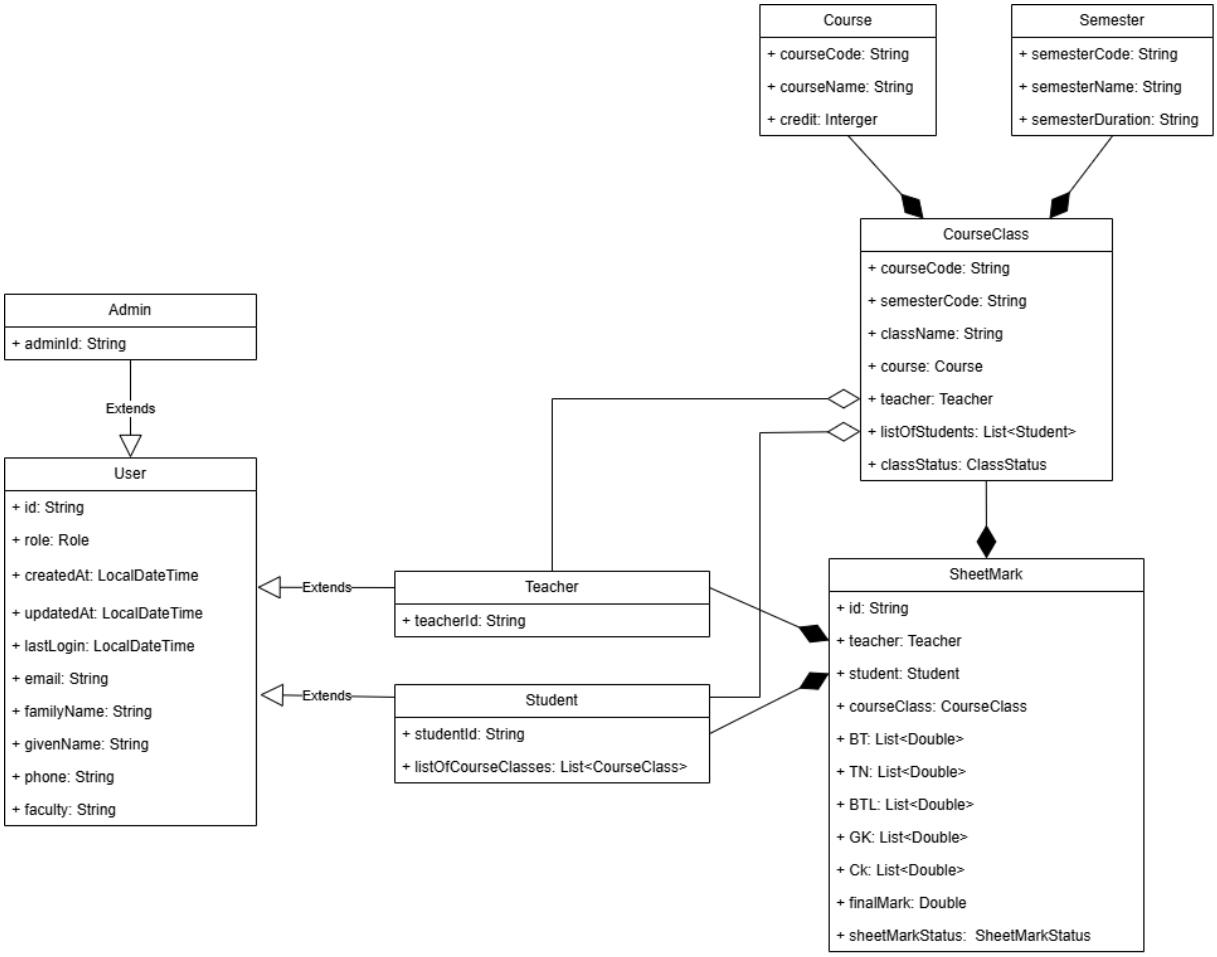
Hình 8: *EERD Diagram*

Relational Data Modal



Hình 9: *Relational Data Modal*

Class Diagram



Hình 10: *Class Diagram*

4.5 API management

Ngoài ra trong hệ thống, để đảm bảo việc giao tiếp giữa FE và BE, việc quản lý và lập chiến lược đối với việc phát triển và quản lý các API Application Programming Interface- Giao diên lập trình ứng dụng) là một trong những công việc quan trọng khi hiện thực hệ thống.

Các API do phía BE Server cung cấp phải đảm bảo có cấu trúc đường dẫn rõ ràng, các tham số đầu vào và Response trả về phải đúng theo cấu trúc được quy định trong Document. Từ các yêu cầu đó, chiến lược của nhóm chúng em trong việc phát triển và quản lý cái API từ Server như sau.

1. Phát triển và kiểm thử các Api bằng Postman:
    - Đầu tiên, các thành viên trong nhóm có nhiệm vụ phát triển BE Server sẽ tạo ra các Api Endponint, quy định cấu trúc các tham số Request của Api và Response trả về thông qua các đối tượng như ApiResponse, ApiRequest tương ứng với các thực thể khác nhau trong mã nguồn hệ thống.
    - Sau đó, nhóm sẽ kiểm thử đầu ra của API và hiệu chỉnh nếu cần, giúp đảm bảo tính chính xác của API khi được sử dụng trong ứng dụng. Từ đó làm nền tảng để viết Api Document cho các thành viên bên FE sử dụng bằng phần mềm **Postman**. Phần mềm Postman sẽ được dùng để kiểm thử và kiểm tra đầu ra của API, đồng thời hỗ trợ viết tài liệu API để các thành viên FE dễ dàng sử dụng. Quá trình này đảm bảo tính rõ ràng, tiện lợi và đồng bộ giữa các thành viên trong việc sử dụng API.
    - Những bước này sẽ giúp nhóm đảm bảo được sự rõ ràng và tiện lợi trong việc sử dụng các Api của thành viên nhóm FE.
2. Xác thực JWT (JSON Web Token):

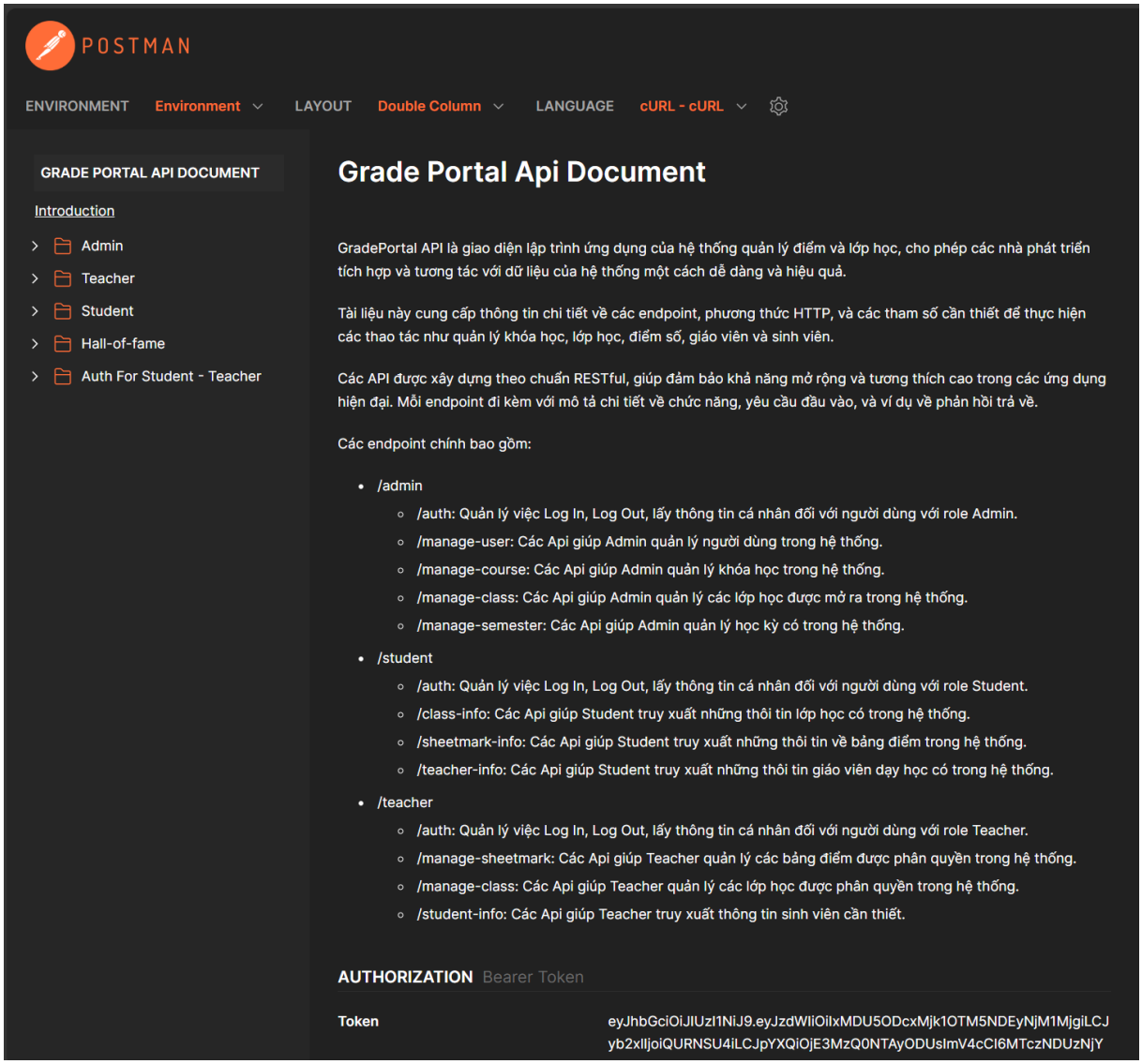
- Sử dụng JWT cho việc xác thực giữa các thành phần trong hệ thống, đảm bảo rằng chỉ những yêu cầu có token hợp lệ mới được xử lý. JWT là giải pháp nhẹ và phổ biến cho xác thực API, phù hợp cho các dự án còn mang tính chất đơn giản.

3. Tích hợp đơn giản với các API bên ngoài:

- Khi cần truy cập các dịch vụ bên ngoài (như dịch vụ xác thực Google), nhóm có thể tạo các Service trong backend để gửi yêu cầu HTTP/ HTTPS trực tiếp đến các API này bằng cách dùng Axios hoặc Fetch.

Các bước trên không chỉ giúp đảm bảo hiệu quả, mà còn phù hợp với yêu cầu bảo mật và tính tiện lợi trong phát triển API, giúp việc hiện thực hệ thống một cách mượt mà và dễ chỉnh sửa.

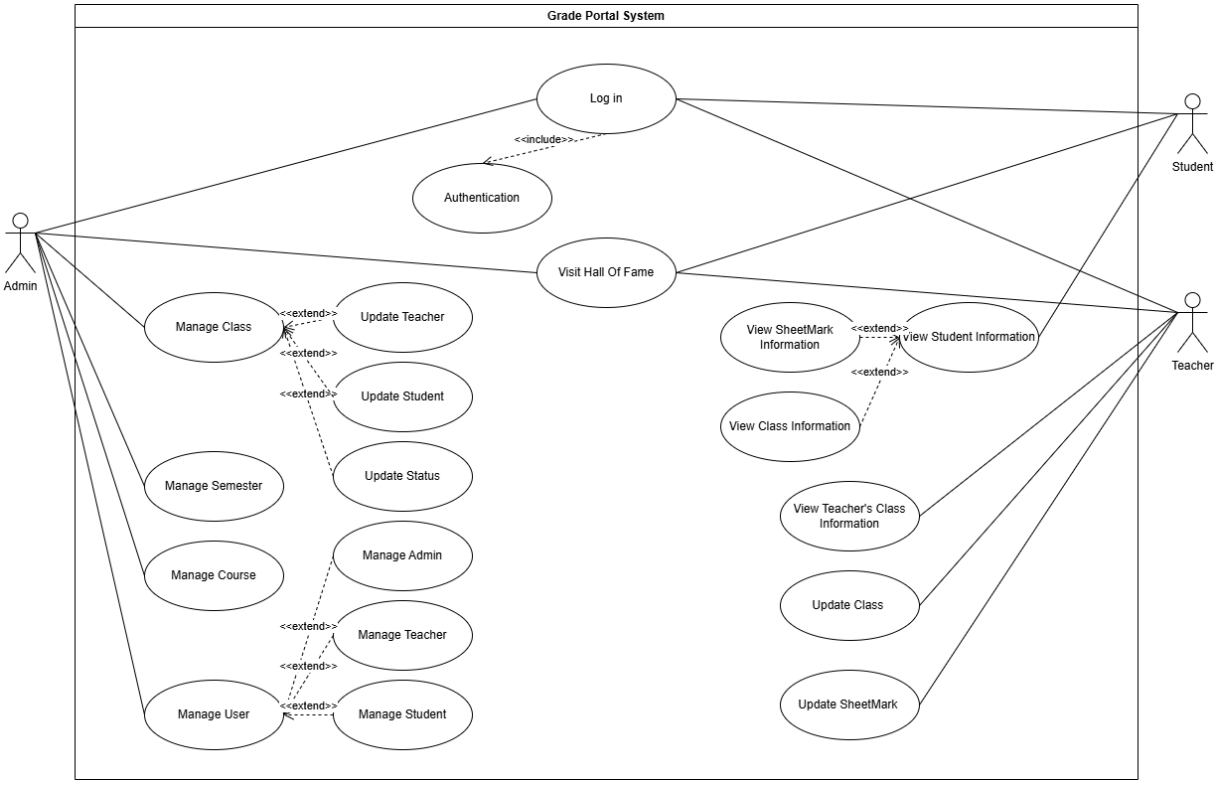
Chi tiết kết quả hiện thực nằm ở đây.



Hình 11: *Postman Grade Portal Api Document*

5 Mô hình hóa hệ thống (System Modelling)

5.1 Sơ Đồ Chức Năng Của Hệ Thống (Use-case Diagram)



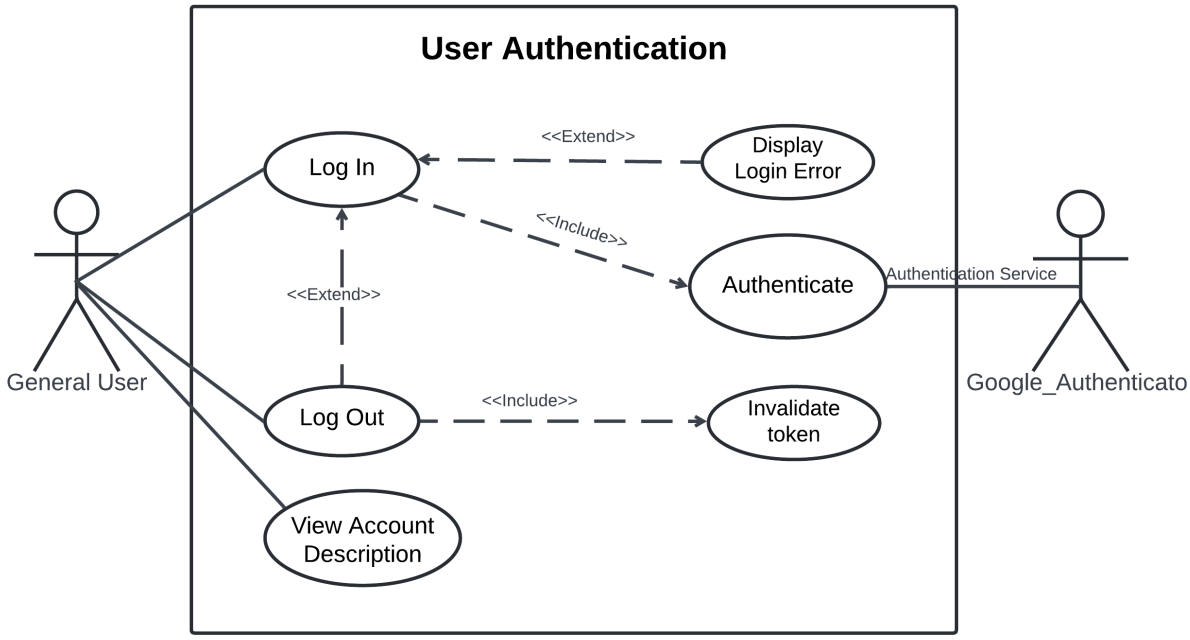
Hình 12: *Sơ đồ chức năng của hệ thống*

Sơ đồ chức năng trên thể hiện các chức năng chính của hệ thống Grade Portal. Các actor chính bao gồm Admin, Teacher, và Student, mỗi actor có những quyền hạn và chức năng riêng phù hợp với vai trò trong hệ thống:

- **Admin:** Quản lý các thành phần hệ thống như lớp học, sinh viên, giáo viên, khoá học và học kỳ. Admin cũng có quyền phân quyền và quản lý người dùng.
- **Teacher:** Quản lý thông tin lớp học, cập nhật điểm số (SheetMark) và xem thông tin học tập của sinh viên.
- **Student:** Tra cứu điểm số, thông tin lớp học và theo dõi tiến độ học tập của mình.

Quan hệ giữa các use-case được tổ chức theo cấu trúc rõ ràng, đảm bảo tính logic và dễ hiểu, hỗ trợ hiệu quả cho quá trình phát triển hệ thống.

5.1.1 Xác thực người dùng



Hình 13: *Sơ đồ chức năng xác thực người dùng*

Use Case ID	UA_01
Use Case	Log In
Primary Actor	General User
Secondary Actor	Google Authenticator
Description	Mô tả quá trình người dùng đăng nhập vào hệ thống
Precondition	1. Hệ thống hoạt động bình thường 2. Người dùng truy cập vào hệ thống qua thiết bị cá nhân
Postcondition	Người dùng đã xác thực và đăng nhập thành công hoặc nhận được thông báo lỗi đăng nhập
Trigger	Người dùng chọn vào tùy chọn “Đăng nhập” từ UI của hệ thống
Normal Flow	1. Người dùng chọn tùy chọn "Đăng nhập" 2. Hệ thống nhắc người dùng nhập thông tin đăng nhập 3. Người dùng nhập thông tin đăng nhập và gửi 4. Hệ thống gọi và sử dụng trường "Xác thực" kiểm tra thông tin đăng nhập với Dịch vụ xác thực 5. Nếu thông tin đăng nhập hợp lệ, hệ thống sẽ tạo mã thông báo và chuyển hướng người dùng đến giao diện tùy theo Role người dùng. 6. Người dùng đã đăng nhập thành công và có thể truy cập tài khoản của họ
Alternative Flow	Không có
Exception Flow	Thông tin đăng nhập không hợp lệ 5.1 Nếu thông tin đăng nhập không hợp lệ, hệ thống kích hoạt trường “Hiển thị lỗi đăng nhập” 5.2 Người dùng nhận thông báo lỗi về thông tin đăng nhập không hợp lệ 5.3 Người dùng có thể thử đăng nhập lại

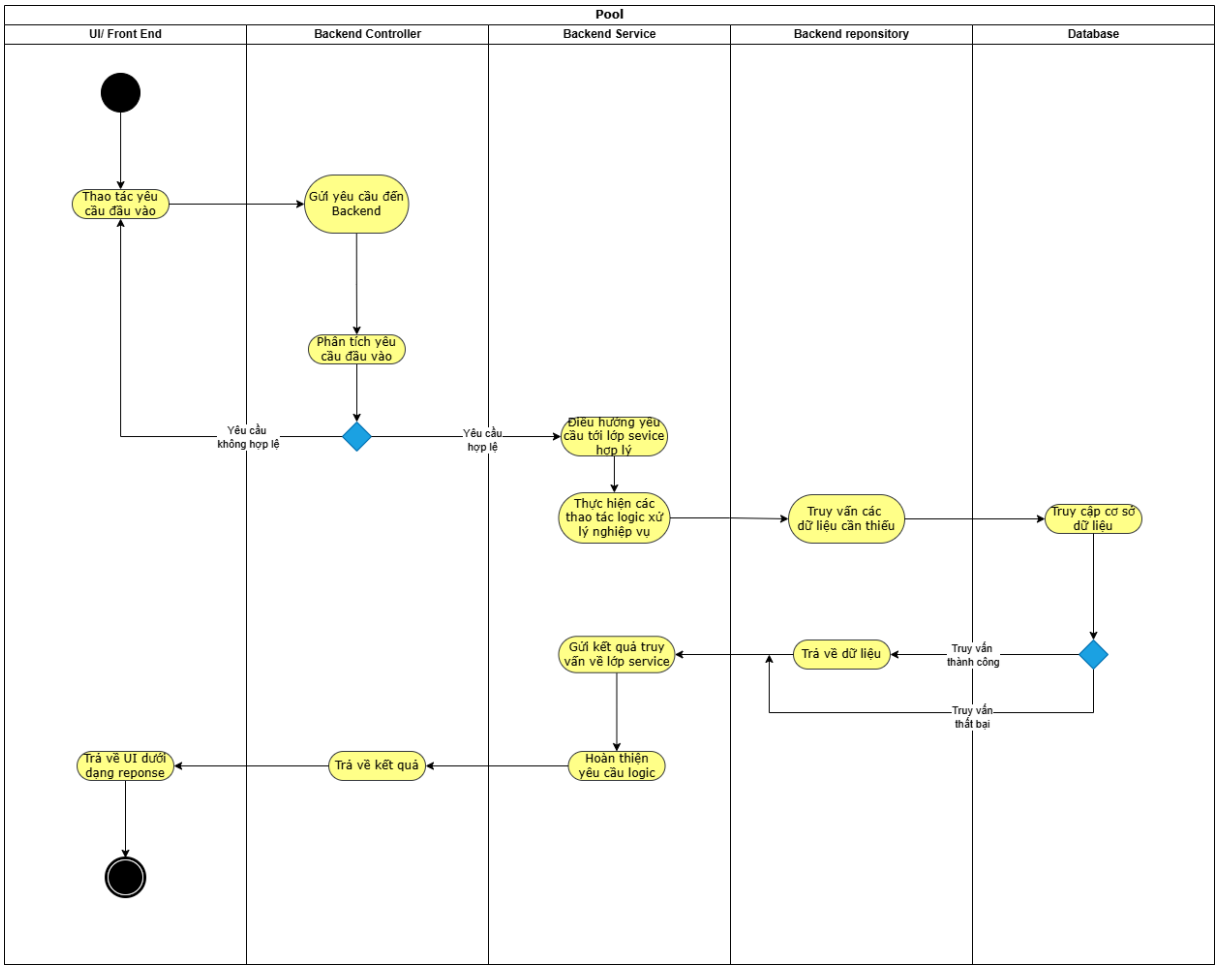
Use Case ID	UA_02
Use Case	Log Out
Primary Actor	General User
Secondary Actor	Không có
Description	Mô tả quá trình người dùng đăng xuất hệ thống khỏi hệ thống
Precondition	Người dùng đã đăng nhập vào hệ thống
Postcondition	Người dùng đã đăng xuất và chuyển hướng đến trang login
Trigger	Người dùng chọn vào tùy chọn "Đăng xuất"
Normal Flow	1. Người dùng chọn tùy chọn "Đăng xuất" 2. Hệ thống vô hiệu hóa token của người dùng 3. Người dùng được chuyển hướng đến trang chủ
Alternative Flow	Không có
Exception Flow	Không có

Use Case ID	UA_03
Use Case	View Account Description
Primary Actor	General User
Secondary Actor	Không có
Description	Mô tả quá trình người dùng chung xem mô tả tài khoản của mình.
Precondition	Người dùng đã đăng nhập vào hệ thống.
Postcondition	Người dùng xem được mô tả tài khoản của mình.
Trigger	Người dùng chọn tùy chọn "Xem mô tả tài khoản".
Normal Flow	1. Người dùng chọn tùy chọn "Xem mô tả tài khoản". 2. Hệ thống lấy thông tin tài khoản. 3. Hệ thống hiển thị thông tin tài khoản cho người dùng.
Alternative Flow	Không có
Exception Flow	Không có

5.2 Activity Diagram

Các diagram được hiện thực chi tiết ở đây.

5.2.1 Activity Diagram chung cho các Usecase trong hệ thống



Hình 14: *Activity Diagram chung cho các Usecase trong hệ thống*

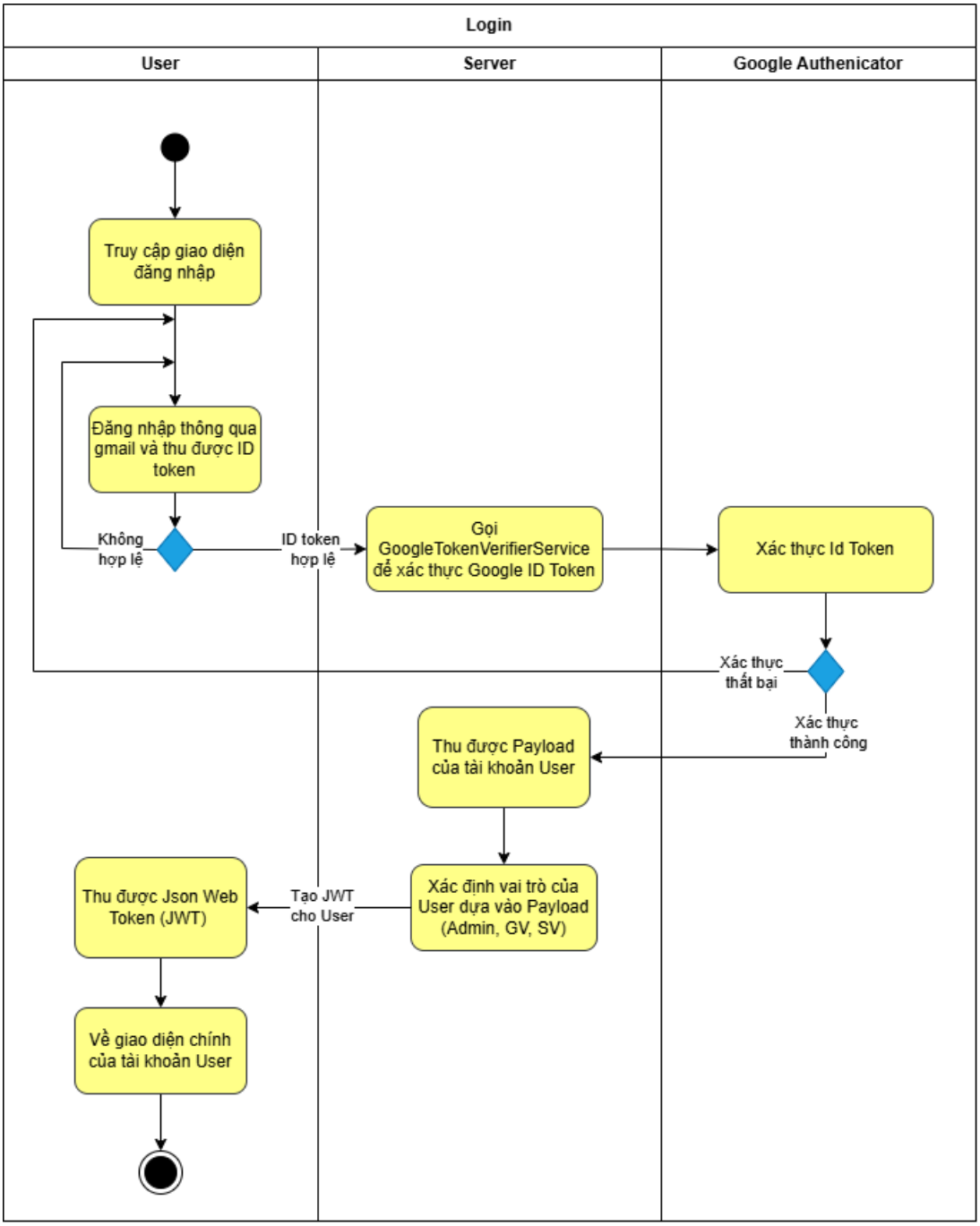
Dưới đây là mô tả luồng xử lý yêu cầu từ người dùng qua các tầng trong hệ thống:

- **UI/Front End:**
  - Người dùng thực hiện thao tác yêu cầu đầu vào.
  - Yêu cầu được gửi đến Backend Controller để xử lý.
- **Backend Controller:**
  - Phân tích yêu cầu đầu vào để kiểm tra tính hợp lệ:
    - Nếu yêu cầu không hợp lệ, trả về phản hồi lỗi cho UI.
    - Nếu yêu cầu hợp lệ, chuyển tiếp đến lớp Backend Service.
- **Backend Service:**
  - Thực hiện các thao tác xử lý nghiệp vụ dựa trên yêu cầu.
  - Gửi yêu cầu truy vấn dữ liệu đến lớp Repository.
- **Backend Repository:**
  - Truy vấn cơ sở dữ liệu (Database) để lấy dữ liệu cần thiết:
    - Nếu truy vấn thất bại, trả về phản hồi lỗi cho Service.

* Nếu truy vấn thành công, trả về dữ liệu cho lớp Service.

- **Hoàn thành xử lý:**
  - Backend Service nhận dữ liệu từ Repository, hoàn tất logic xử lý.
  - Trả kết quả cho Backend Controller, sau đó gửi phản hồi về UI.
- **Kết quả hiển thị:**
  - UI nhận phản hồi dưới dạng kết quả hoặc thông báo lỗi và hiển thị cho người dùng.

5.2.2 Activity Diagram cho chức năng xác thực của hệ thống



Hình 15: *Activity Diagram cho chức năng xác thực của hệ thống*

Dưới đây là mô tả chi tiết quy trình đăng nhập thông qua Google Authentication:

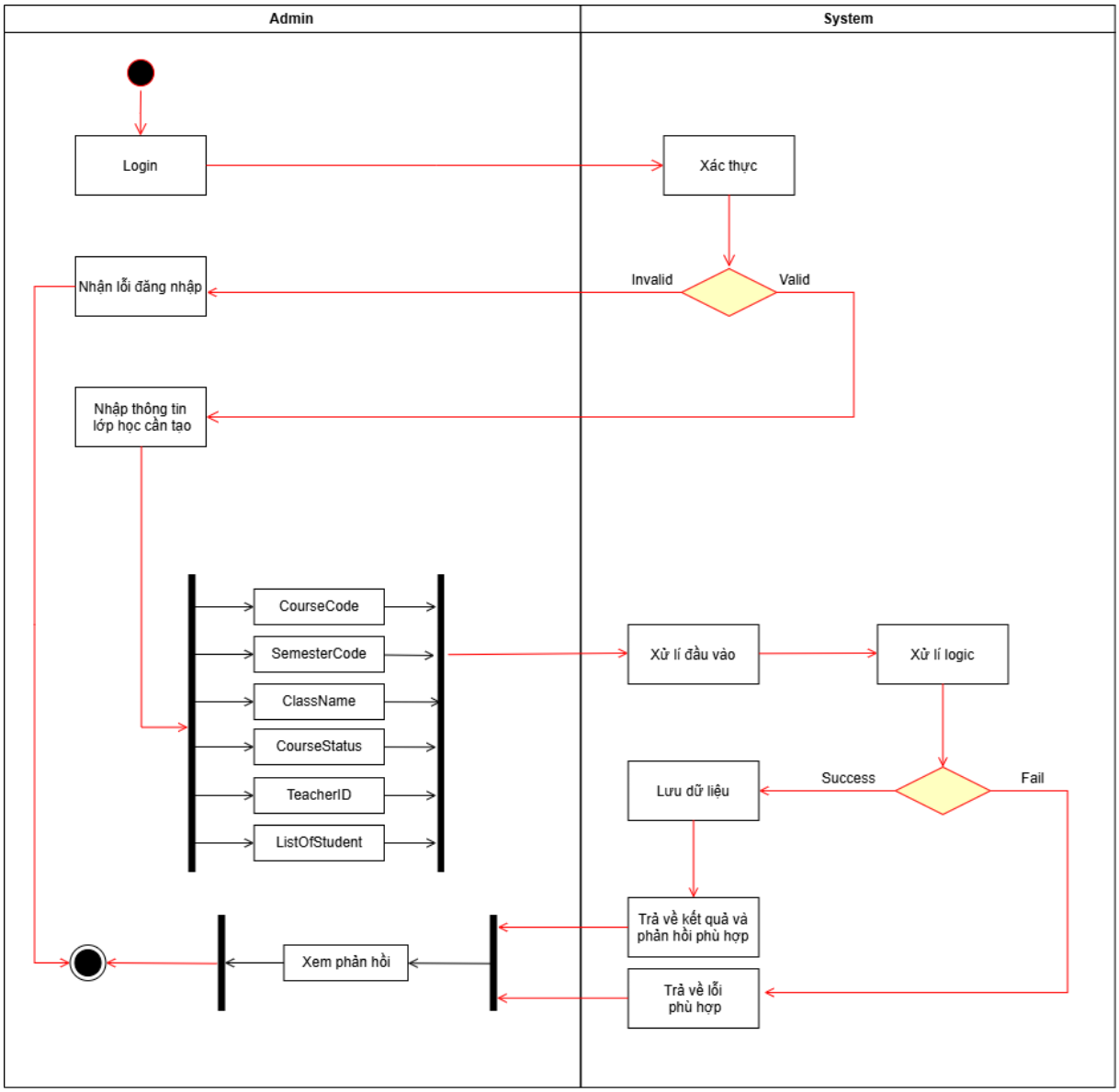
- **Người dùng bắt đầu đăng nhập:**
  - Người dùng truy cập giao diện đăng nhập và chọn đăng nhập qua Gmail.
  - Nếu ID Token không hợp lệ, quy trình đăng nhập kết thúc.
- **Xử lý tại Server:**
  - Server nhận ID Token từ người dùng và gửi đến GoogleTokenVerifierService để xác minh.
- **Xác thực qua Google Authenticator:**
  - Google tiến hành xác thực ID Token.
  - Nếu xác thực thất bại, quy trình đăng nhập dừng lại.
- **Lấy Payload:**
  - Nếu xác thực thành công, server nhận được payload của người dùng, bao gồm các thông tin như email và vai trò.
- **Xác định vai trò:**
  - Server xác định vai trò của người dùng dựa trên payload (ví dụ: Admin, Giáo viên, Sinh viên).
- **Tạo JWT:**
  - Hệ thống tạo một JWT (JSON Web Token) để đảm bảo bảo mật cho phiên làm việc.
- **Hoàn tất đăng nhập:**
  - Người dùng được chuyển đến giao diện chính của tài khoản sau khi đăng nhập thành công.

Điểm nổi bật:

- Sử dụng Google Authentication để tăng tính bảo mật.
- Xác minh danh tính người dùng qua nhiều giai đoạn.
- Sử dụng JWT để đảm bảo an toàn cho phiên làm việc.
- Phân loại người dùng dựa trên vai trò từ payload.

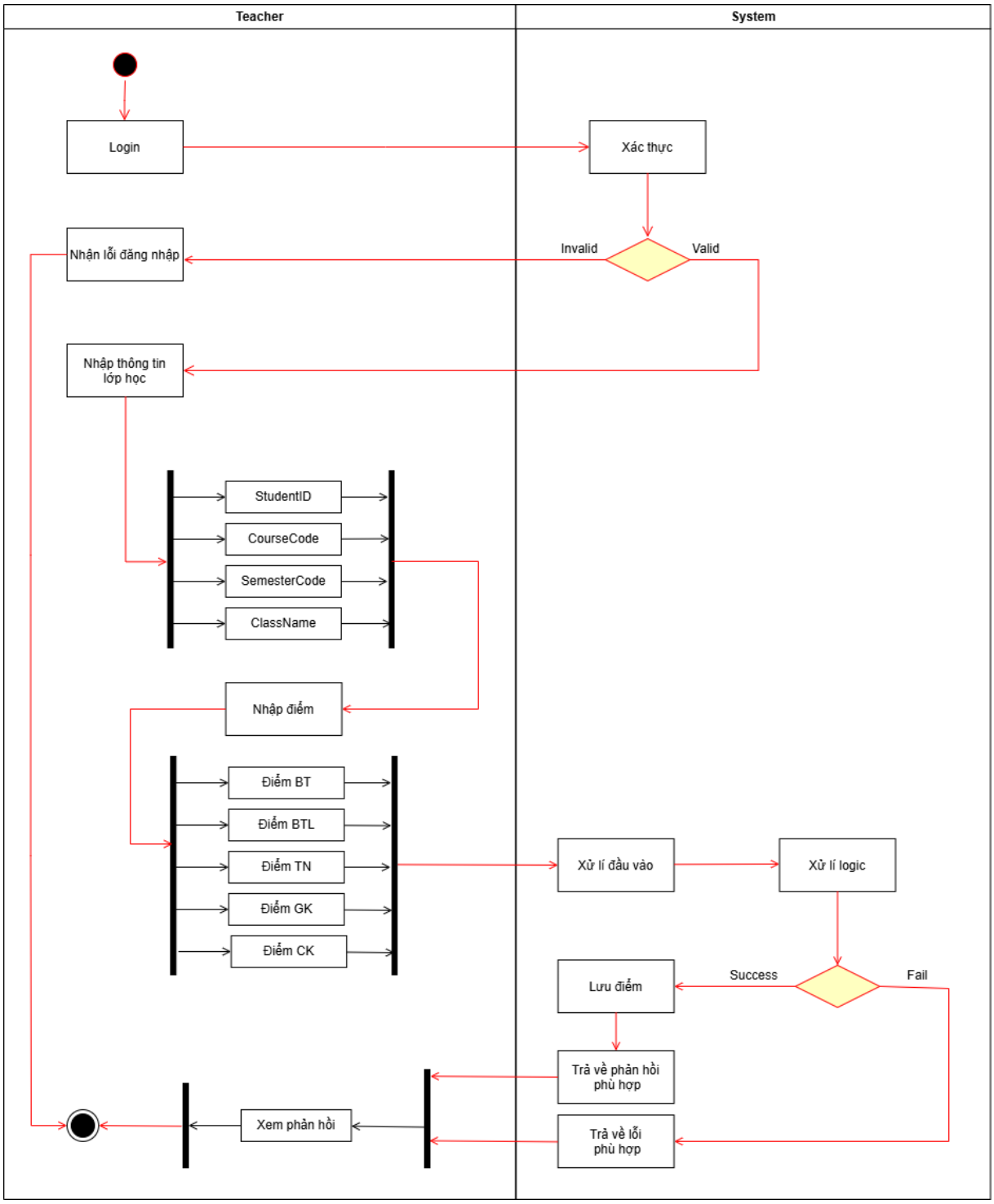
Quy trình này đảm bảo một hệ thống đăng nhập an toàn và hiệu quả cho người dùng.

5.2.3 Activity Diagram cho chức năng tạo lớp học mới



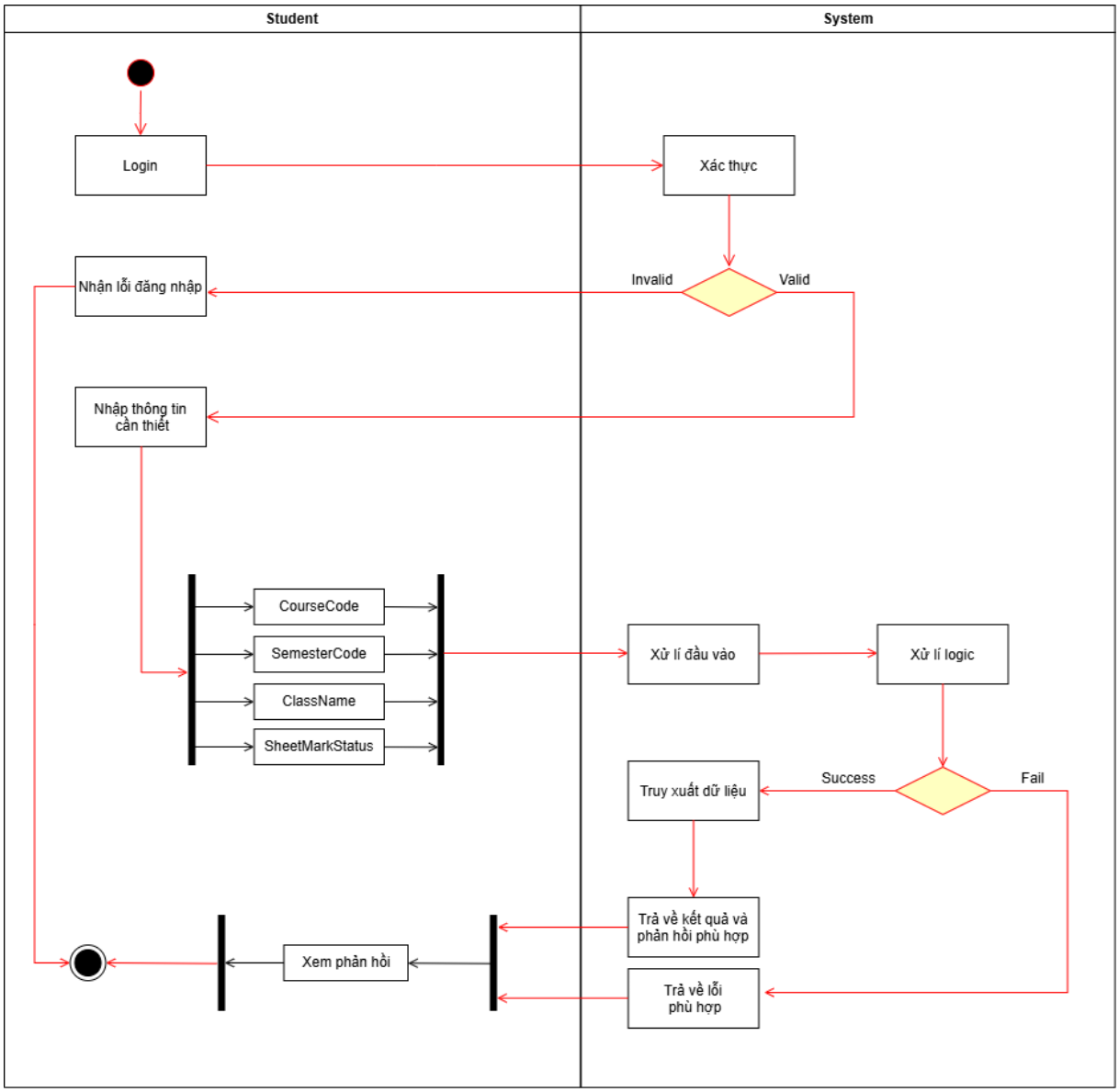
Hình 16: *Activity Diagram cho chức năng tạo lớp học mới*

5.2.4 Activity Diagram cho chức năng cập nhật điểm số của giảng viên



Hình 17: *Activity Diagram cho chức năng cập nhật điểm số của giảng viên*

5.2.5 Activity Diagram cho chức năng xem điểm của sinh viên



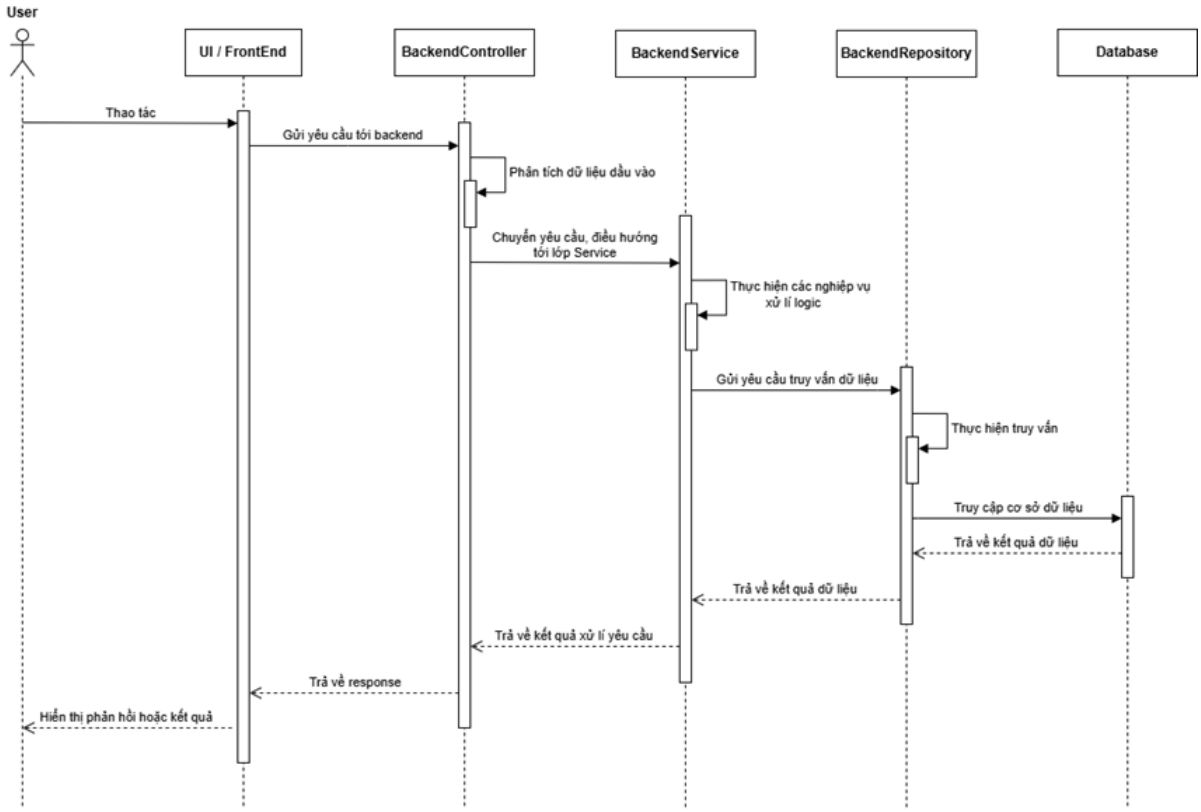
Hình 18: *Activity Diagram cho chức năng xem điểm của sinh viên*

5.3 Sequence Diagram

Các diagram được hiện thực chi tiết ở đây.

5.3.1 Sequence Diagram chung cho các Usecase trong hệ thống

Dưới đây là một Sequence Diagram minh họa luồng xử lý chung cho các Usecase trong hệ thống. Sequence Diagram này được xây dựng để mô tả cách các thành phần trong hệ thống tương tác với nhau, từ khi nhận yêu cầu từ người dùng cho đến khi trả về kết quả.



Hình 19: *Sequence Diagram chung cho các Usecase trong hệ thống*

Trong hệ thống được thiết kế theo kiến trúc nhiều tầng (Layered Architecture), các thành phần đảm nhận vai trò riêng biệt để đảm bảo tính rõ ràng, dễ bảo trì và phát triển. Các bước xử lý được diễn tả như sau:

- **User**:
  - Người dùng gửi yêu cầu (thường là HTTP Request) từ giao diện ứng dụng (web/mobile).
  - Yêu cầu này sẽ bao gồm các thông tin cần thiết để thực hiện chức năng trong hệ thống.
- **Controller**:
  - Nhận yêu cầu từ người dùng và kiểm tra thông tin cơ bản.
  - Gọi lớp Service để xử lý logic nghiệp vụ liên quan.
  - Sau khi nhận kết quả từ Service, định dạng phản hồi (thường là JSON hoặc XML) và gửi lại cho người dùng.
- **Service**:
  - Thực hiện xử lý logic nghiệp vụ của Usecase.
  - Kiểm tra tính hợp lệ của dữ liệu đầu vào từ Controller.
  - Gọi lớp Repository để thực hiện thao tác với cơ sở dữ liệu.
  - Trả kết quả đã xử lý cho Controller.

- **Repository**:
  - Chịu trách nhiệm thực hiện các thao tác truy vấn, lưu trữ hoặc truy xuất dữ liệu từ Database.
  - Đảm bảo rằng các truy vấn được tối ưu hóa và phù hợp với cấu trúc của cơ sở dữ liệu.
- **Database**:
  - Lưu trữ dữ liệu bền vững và xử lý các truy vấn từ Repository.
  - Đảm bảo tính toàn vẹn dữ liệu và hiệu suất truy cập cao.

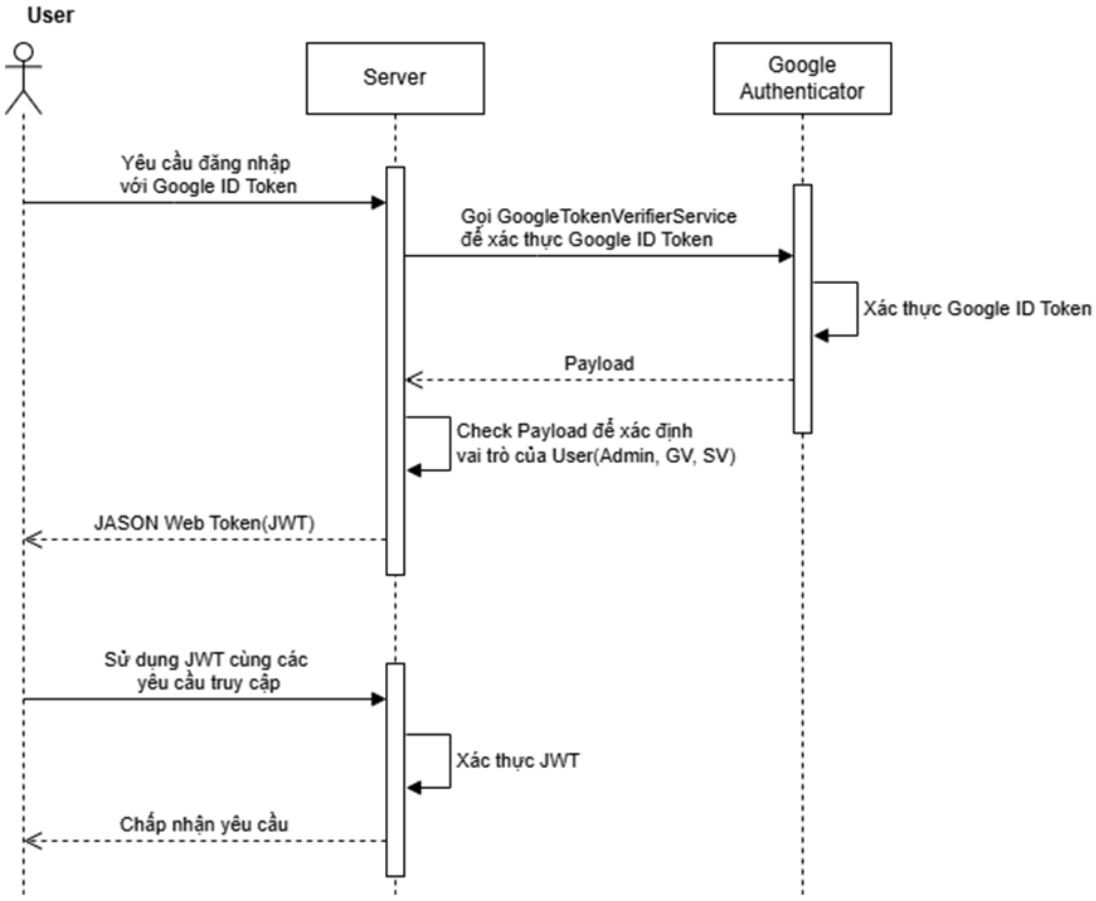
Sequence Diagram ở trên minh họa một trường hợp điển hình: Xác thực người dùng thông qua Google ID Token. Quy trình bao gồm các bước:

- Người dùng gửi Google ID Token đến Server.
- Server sử dụng dịch vụ Google Token Verifier để xác thực Token.
- Dựa vào thông tin trong Payload của Token, hệ thống xác định vai trò của người dùng (Admin, Giáo viên, Sinh viên, ...).
- Tạo JWT (JSON Web Token) để xác nhận danh tính người dùng trong các yêu cầu tiếp theo.
- Sử dụng JWT để xác thực và xử lý yêu cầu của người dùng.

Kịch bản trên chỉ là một ví dụ tiêu biểu. Các Usecase khác trong hệ thống sẽ tuân thủ luồng xử lý tương tự, với sự thay đổi cụ thể ở logic nghiệp vụ trong lớp Service và Repository.

5.3.2 Sequence Diagram cho chức năng xác thực của hệ thống

Chức năng xác thực trong hệ thống được thiết kế để xử lý quá trình đăng nhập và xác định vai trò người dùng thông qua Google ID Token. Sequence Diagram bên dưới minh họa các bước xử lý chi tiết của chức năng này.

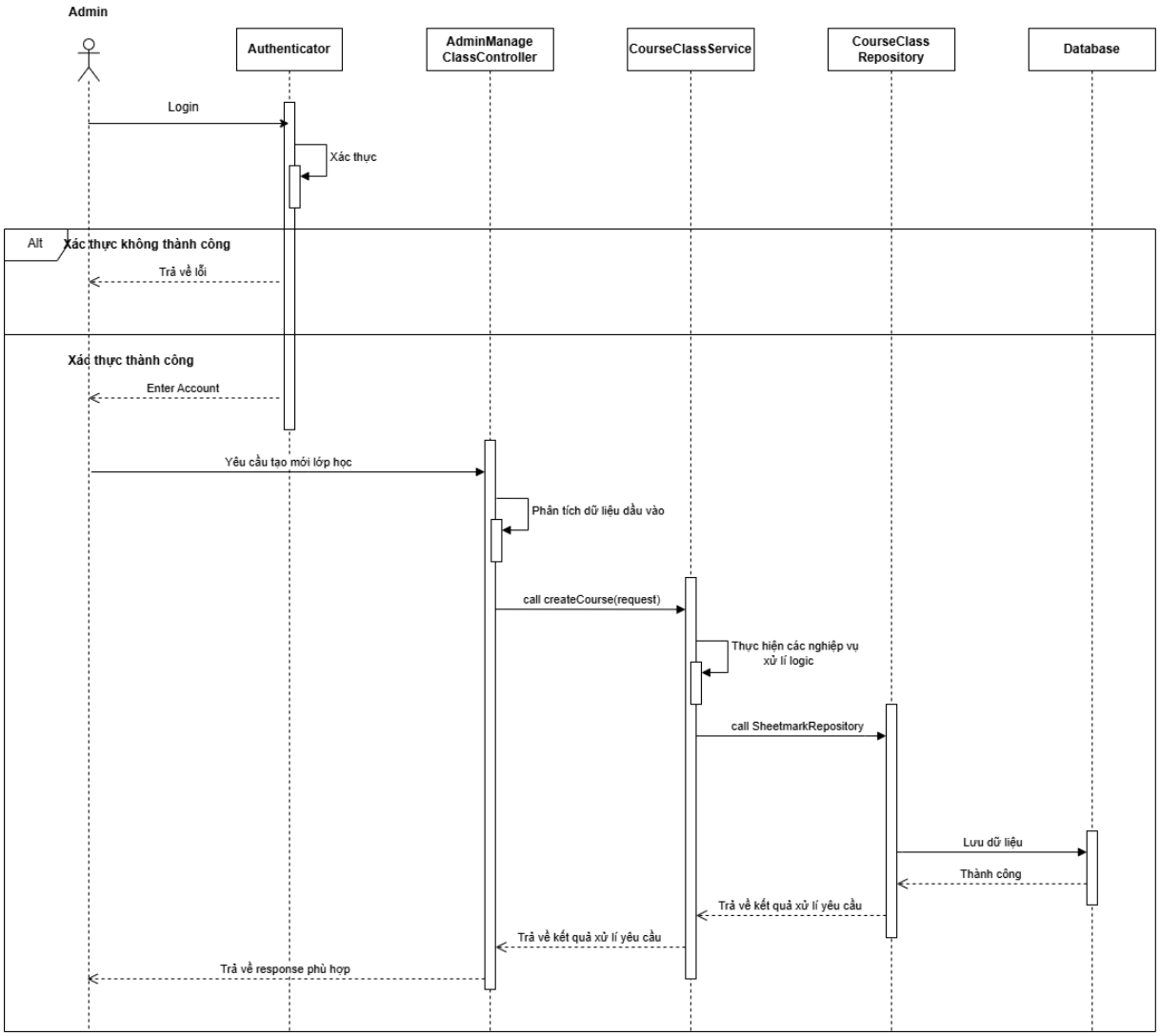


Hình 20: *Sequence Diagram cho chức năng xác thực của hệ thống*

Mô tả chi tiết quá trình xác thực:

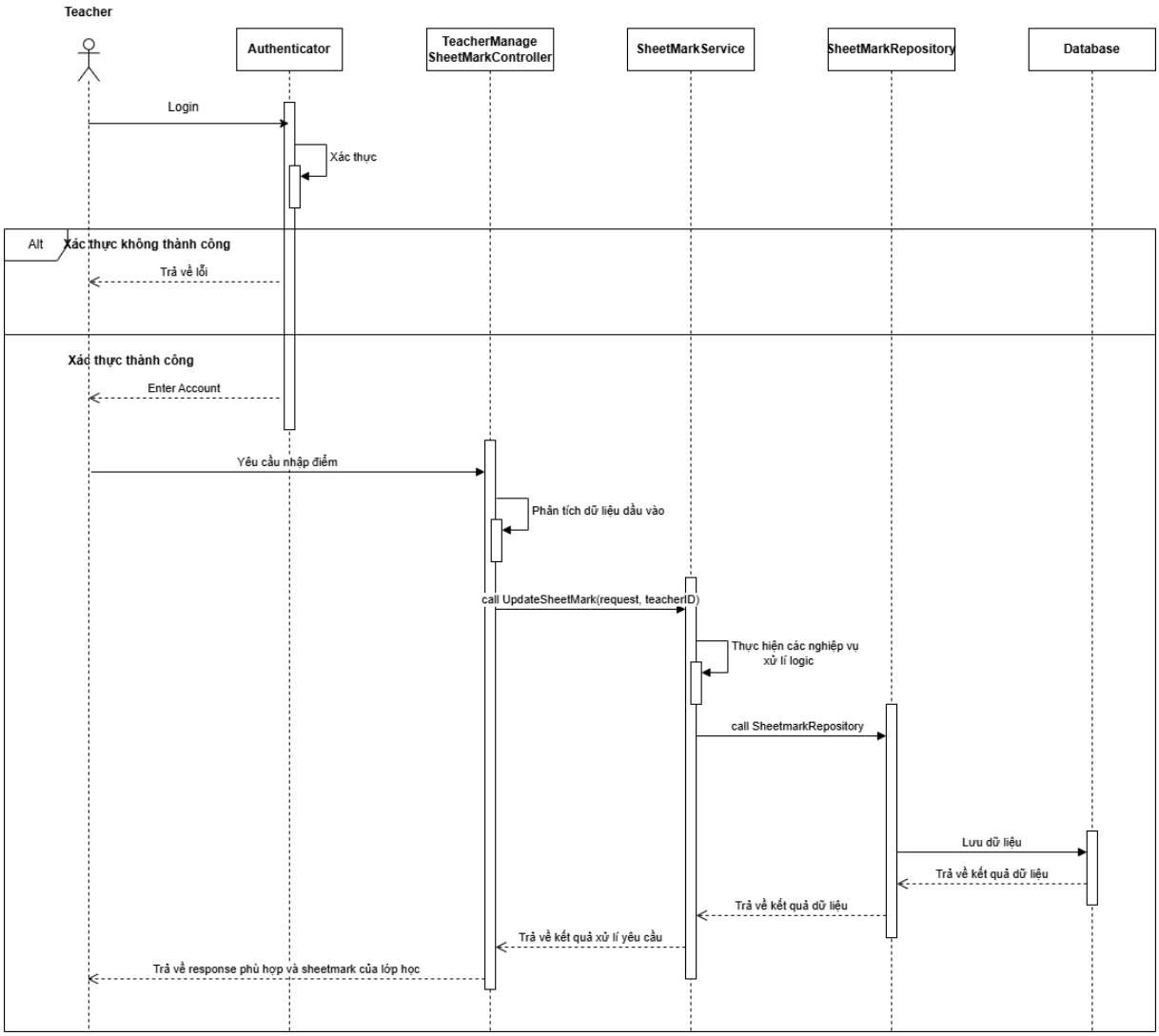
- **Client**:
  - Gửi yêu cầu đăng nhập với Google ID Token đến máy chủ.
- **Server**:
  - Nhận yêu cầu và gọi GoogleTokenVerifierService để xác minh Google ID Token.
  - Kiểm tra dữ liệu trả về từ GoogleTokenVerifierService (payload) để xác định vai trò của người dùng (Admin, Student, Teacher).
- **GoogleTokenVerifierService**:
  - Xác minh Google ID Token với máy chủ của Google.
  - Trả về thông tin xác thực (payload), bao gồm email và các thông tin liên quan.
- **Phân loại vai trò người dùng**:
  - **Đối với Admin**:
    - AdminService gọi adminRepository để tìm kiếm quản trị viên qua email.
    - Nếu tìm thấy quản trị viên, gọi JwtService để tạo JWT.
    - Trả về JWT và thông tin chi tiết của quản trị viên cho máy khách.
  - **Đối với Student**:
    - StudentService gọi studentRepository để tìm kiếm học sinh qua email.
    - Nếu tìm thấy học sinh, gọi JwtService để tạo JWT.
    - Trả về JWT và thông tin chi tiết của học sinh cho máy khách.
  - **Đối với Teacher**:
    - TeacherService gọi teacherRepository để tìm kiếm giáo viên qua email.
    - Nếu tìm thấy giáo viên, gọi JwtService để tạo JWT.
    - Trả về JWT và thông tin chi tiết của giáo viên cho máy khách.
- **Client**:
  - Nhận JWT và sử dụng trong các yêu cầu tiếp theo để truy cập vào các tài nguyên được bảo vệ.
- **Server**:
  - Xác thực JWT trong mỗi yêu cầu bằng JwtAuthenticationFilter.

5.3.3 Sequence Diagram cho chức năng tạo lớp học mới



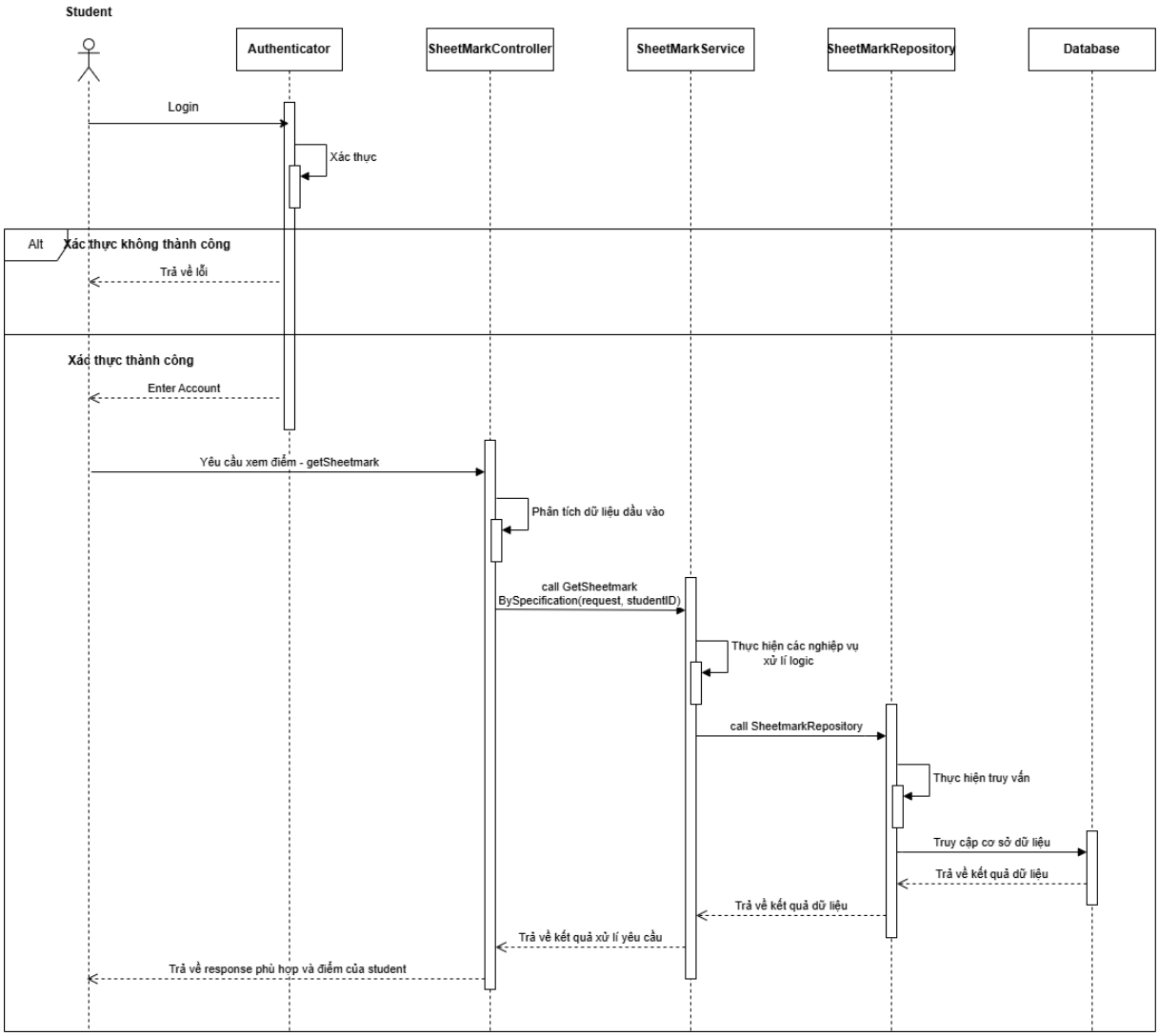
Hình 21: *Sequence Diagram cho chức năng tạo lớp học mới*

5.3.4 Sequence Diagram cho chức năng cập nhật điểm số của giảng viên



Hình 22: *Sequence Diagram cho chức năng cập nhật điểm số của giảng viên*

5.3.5 Sequence Diagram cho chức năng xem điểm của sinh viên



Hình 23: *Sequence Diagram cho chức năng xem điểm của sinh viên*

6 Tổ chức và quản lý Code trong mã nguồn

Cấu trúc mã nguồn của dự án được tổ chức theo dạng cây thư mục với các thư mục chính và chức năng cụ thể như sau:

```
.
|--- .mvn
|    |___ wrapper            % Cấu hình Maven Wrapper
|--- .vscode                 % Cấu hình cho VSCode
|--- docs                    % Thư mục chứa tài liệu dự án
|    |--- api-document       % Tài liệu API
|    |--- database-design    % Thiết kế CSDL
|    |--- deploy-guide       % Hướng dẫn triển khai
|    |--- git-flow           % Quy trình làm việc với Git
|    |___ system-architecture% Kiến trúc hệ thống
|--- reports                 % Báo cáo dự án
|    |--- mainReport         % Báo cáo chính
|    |___ weeklyReport       % Báo cáo hàng tuần
|         |--- meeting_minute_report % Biên bản họp
|         |___ weekly_result  % Kết quả hàng tuần
|              |___ img_10_27_24 % Hình ảnh minh họa
|--- src                     % Mã nguồn chính
|    |--- main
|    |    |--- java
|    |    |    |___ com
|    |    |         |___ hcmut
|    |    |              |___ gradeportal
|    |    |                   |--- config         % Cấu hình hệ thống
|    |    |                   |--- controller     % Các controller xử lý API
|    |    |                   |    |--- admin
|    |    |                   |    |--- client_auth
|    |    |                   |    |--- hall_of_fame
|    |    |                   |    |--- health_check
|    |    |                   |    |--- init_data
|    |    |                   |    |--- student
|    |    |                   |    |___ teacher
|    |    |                   |--- database       % Thao tác cơ sở dữ liệu
|    |    |                   |--- dtos           % Data Transfer Objects
|    |    |                   |    |--- admin
|    |    |                   |    |    |--- request
|    |    |                   |    |    |___ response
|    |    |                   |    |--- auth
|    |    |                   |    |--- course
|    |    |                   |    |--- sheetMark
|    |    |                   |    |___ teacher
|    |    |                   |--- entities       % Định nghĩa các entity
|    |    |                   |    |--- enums
|    |    |                   |    |___ idsOfEntities
|    |    |                   |--- repositories   % Repository làm việc với CSDL
|    |    |                   |--- security       % Bảo mật JWT
|    |    |                   |--- service        % Logic nghiệp vụ
|    |    |                   |--- specification  % Query Specification
|    |    |                   |___ validation     % Validation dữ liệu
|    |    |___ resources
|    |         |--- baseData      % Dữ liệu nền
|    |         |--- static        % File tĩnh
|    |         |___ templates     % Template giao diện
|    |___ test                  % Mã nguồn test
|--- target                  % Thư mục build của Maven
|___ pom.xml                 % File cấu hình Maven
```

6.1 Mô tả chi tiết các thành phần chính

- **docs**: Chứa các tài liệu liên quan đến dự án, bao gồm:
  - **api-document**: Tài liệu mô tả chi tiết các API trong hệ thống.
  - **database-design**: Thiết kế cơ sở dữ liệu, bao gồm lược đồ và quan hệ giữa các bảng.
  - **deploy-guide**: Hướng dẫn triển khai hệ thống.
  - **git-flow**: Quy trình làm việc với Git.
  - **system-architecture**: Mô tả kiến trúc hệ thống.
- **src/main/java**: Thư mục chính chứa mã nguồn của hệ thống:
  - **config**: Chứa các lớp cấu hình hệ thống, như cấu hình bảo mật, cấu hình CSDL hoặc các bean cần thiết.
  - **controller**: Chứa các lớp xử lý yêu cầu API từ người dùng:
    - Chia thành nhiều module như `admin`, `client_auth`, `student`, và `teacher` để dễ quản lý.
    - Mỗi lớp controller sẽ đảm nhận việc điều hướng và trả kết quả phù hợp về tầng Presentation.
  - **dtos (Data Transfer Objects)**: Chứa các lớp trung gian dùng để chuyển dữ liệu giữa các tầng.
    - Được chia thành các gói nhỏ như: **admin**, **auth**, **course**, **sheetMark**, **teacher**, v.v.
    - Mỗi gói chứa các lớp `request` và `response` để định nghĩa dữ liệu đầu vào và đầu ra.
  - **entities**: Định nghĩa các đối tượng ánh xạ với bảng trong cơ sở dữ liệu.
    - **enums**: Chứa các kiểu dữ liệu liệt kê được sử dụng trong hệ thống.
    - **idsOfEntities**: Định nghĩa các khóa chính tổng hợp hoặc khóa đặc biệt của các entity.
  - **repositories**: Cung cấp các phương thức làm việc với cơ sở dữ liệu (CRUD). Đây là các interface giúp giảm thiểu việc viết câu lệnh SQL thủ công.
  - **security**: Chứa các lớp hiện thực cơ chế bảo mật cho hệ thống.
    - **jwt**: Quản lý và xử lý token JWT cho việc xác thực và phân quyền.
    - **utils**: Các tiện ích hỗ trợ việc mã hóa và kiểm tra bảo mật.
  - **service**: Chứa các lớp xử lý logic nghiệp vụ của hệ thống. Mỗi module (như `admin`, `student`, `course`) sẽ có một lớp service tương ứng.
  - **specification**: Cung cấp các điều kiện để tạo truy vấn linh hoạt khi làm việc với cơ sở dữ liệu.
  - **validation**: Chứa các lớp xác thực dữ liệu đầu vào để đảm bảo tính đúng đắn và toàn vẹn.
- **src/test/java**: Chứa các lớp kiểm thử cho các thành phần của hệ thống. Các lớp test được tổ chức tương tự như cấu trúc mã nguồn chính.
- **resources**: Chứa các tệp cấu hình và dữ liệu cho hệ thống:
  - **baseData**: Dữ liệu nền để khởi tạo hệ thống.
  - **templates**: Các file template phục vụ cho giao diện hoặc email.
  - **static**: Chứa các tài nguyên tĩnh như hình ảnh, CSS hoặc JavaScript.
- **target**: Thư mục được tạo ra sau khi hệ thống được build bằng Maven. Bao gồm các file `.class`, báo cáo test và các tài nguyên được biên dịch.

6.2 Code style

Đoạn mã nguồn tuân thủ các quy tắc code style hiện đại nhằm đảm bảo tính nhất quán, dễ đọc và dễ bảo trì. Các quy tắc chính được áp dụng như sau:

- **Tổ chức import:** Các import được sắp xếp theo nhóm (Java chuẩn, thư viện bên thứ ba, module nội bộ) và loại bỏ các import dư thừa.
- **Đặt tên:** Sử dụng *camelCase* cho tên biến và phương thức (`getAllTeachers`, `updateTeacher`). Tên biến rõ ràng, mô tả đúng chức năng. *Class* sử dụng PascalCase (ví dụ: UserManager). *Hằng số* sử dụng chữ in hoa và gạch dưới (ví dụ: MAX_CONNECTIONS).
- **Tổ chức logic:** Các thao tác CRUD (`createTeacher`, `updateTeacher`, `deleteTeacher`) và các chức năng khác được viết tách biệt theo trách nhiệm.

- **Xử lý null:** Sử dụng `Optional` để tránh lỗi *NullPointerException*, đảm bảo mã nguồn an toàn.
- **Xử lý lỗi:** Sử dụng ngoại lệ như `IllegalArgumentException` với thông báo rõ ràng (*"Teacher not found"*).
- **Format mã:** Thụt lề 4 spaces, có khoảng cách giữa các toán tử và từ khóa (`if (condition) { ... }`), sử dụng dòng trắng để phân tách các khối logic.
- **Comment:** Comment block mô tả chức năng chính của phương thức; comment ngắn gọn giải thích từng bước xử lý.
- **Tuân thủ SOLID:** Phân chia rõ trách nhiệm giữa repository và service; mỗi phương thức đảm nhận một chức năng cụ thể.

Những quy tắc trên giúp mã nguồn đảm bảo tính dễ đọc, dễ bảo trì và sẵn sàng mở rộng trong tương lai.

7 Quản lý phiên bản (Version Control)

7.1 Lý Do Lựa Chọn GitHub

GitHub là một trong những nền tảng quản lý mã nguồn phổ biến nhất hiện nay, với nhiều tính năng mạnh mẽ, dễ sử dụng và tích hợp sẵn với các công cụ phát triển phần mềm. Các lý do chính khiến nhóm lựa chọn GitHub bao gồm:

- **Dễ dàng sử dụng**: GitHub cung cấp giao diện người dùng thân thiện, giúp cho việc quản lý và chia sẻ mã nguồn trở nên dễ dàng.
- **Quản lý phiên bản mạnh mẽ**: GitHub hỗ trợ Git, một công cụ quản lý phiên bản phân tán rất mạnh mẽ, giúp theo dõi và quản lý thay đổi mã nguồn theo thời gian.
- **Chia sẻ và cộng tác**: GitHub cho phép các thành viên trong nhóm cộng tác dễ dàng, với các tính năng như Pull Requests và Issues giúp theo dõi tiến độ và xử lý vấn đề.
- **Tính bảo mật và tính công khai**: GitHub cho phép tạo repository công khai hoặc riêng tư, đảm bảo tính bảo mật khi cần thiết và dễ dàng chia sẻ mã nguồn với cộng đồng khi sử dụng repository công khai.

Việc sử dụng GitHub giúp nhóm quản lý mã nguồn, tài liệu và các thay đổi một cách hiệu quả và dễ dàng. Các tính năng của GitHub không chỉ hỗ trợ trong việc cộng tác mà còn đảm bảo tính minh bạch và theo dõi tiến độ phát triển dự án một cách rõ ràng. Chúng em sẽ tiếp tục duy trì và cập nhật các repository này để đảm bảo dự án được phát triển liên tục và dễ dàng quản lý.

Dưới đây là đường dẫn đến repository của dự án:

- Repository dẫn tới Server của dự án (bao gồm mã nguồn và các tài liệu liên quan tới Backend dự án): `https://github.com/dath-241/grade-portal-be-java`

7.2 Vấn đề về việc mã hóa file `.env` khi đưa lên repo dự án

Trong các dự án phần mềm, file `.env` chứa các biến môi trường quan trọng như thông tin cấu hình hệ thống, khóa bảo mật, hoặc đường dẫn kết nối đến cơ sở dữ liệu. Việc để lộ file `.env` lên repository có thể dẫn đến rủi ro bảo mật nghiêm trọng. Để giải quyết vấn đề này, các công cụ mã hóa như **SOPS** kết hợp với **Age** thường được sử dụng để mã hóa các file chứa thông tin nhạy cảm.

Trường hợp bạn được cung cấp file khóa `key.txt`

Nếu bạn được cung cấp khóa Age trong file `key.txt`, cấu trúc file sẽ có dạng như sau:

```
# created: 2024-10-19T18:05:45+07:00
# public key: age1l4zc9ppdyvz6hvwj67uca0pfgyydm9efs7kgmmks3h9pz7xa9v3scrl3sn
AGE-SECRET-KEY-1T4Y4TH7S8GMAUW9J7SFMP6YXRG2FF0UFX08MSPWS540CA20NL40QVJ60VS
```

Thực hiện các bước sau để giải mã file chứa biến môi trường bị mã hóa (`dev.enc` hoặc `prod.enc`) và tạo file `.env`:

1. Đặt file `key.txt` vào thư mục `gradeportal`.

2. Chạy lệnh để giải mã file `dev.enc`:

```
sops --age key.txt --decrypt dev.enc > .env
```

3. Trong trường hợp máy của bạn gặp lỗi khi sử dụng lệnh trên, bạn cần thiết lập biến môi trường Age Key trực tiếp như sau:

```
set SOPS_AGE_KEY=AGE-SECRET-KEY-1T4Y4TH7S8GMAUW9J7SFMP6YXRG2FF0UFX08MSPWS540CA20NL40QVJ60
```

4. Sau đó, giải mã file `dev.enc` hoặc `prod.enc` bằng lệnh:

```
sops --decrypt --input-type dotenv --output-type dotenv dev.enc > .env
```

Trường hợp bạn không được cung cấp file khóa `key.txt`

Nếu bạn không có khóa trong file `key.txt`, bạn có thể tự tạo file `.env` như sau:

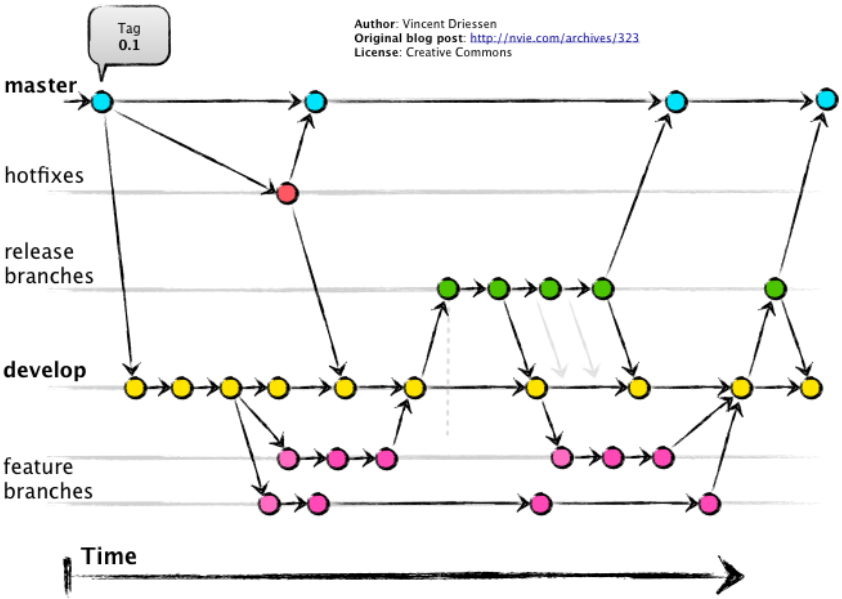
- Tạo file `.env` trong thư mục `gradeportal`.
- Điền các đường dẫn tài nguyên và thông tin cần thiết dựa trên file `example.env` được cung cấp sẵn.

Lợi ích của việc mã hóa file `.env`

- Bảo vệ thông tin nhạy cảm như khóa API, mật khẩu cơ sở dữ liệu, và các thông tin quan trọng khác.
- Giảm thiểu nguy cơ rò rỉ dữ liệu khi đẩy mã nguồn lên repository công khai hoặc trong quá trình làm việc nhóm.
- Cho phép người phát triển dễ dàng quản lý và chia sẻ biến môi trường một cách an toàn thông qua file mã hóa.

7.3 Gitflow

Trong dự án, **Gitflow** được sử dụng như một mô hình quản lý mã nguồn nhằm tránh xung đột giữa các thành viên trong nhóm và đảm bảo quy trình phát triển diễn ra một cách mạch lạc, rõ ràng.



Hình 24: *Gitflow hiện thực dự án*

1. Cấu trúc Branch trong dự án

- **Nhánh chính (Main Branches):**
  - **main**: Chứa mã nguồn ổn định đã được phát hành. Đây là nhánh chính phục vụ cho người dùng cuối.
  - **staging**: Chứa mã nguồn đang được kiểm thử trước khi phát hành. Các tính năng mới và hotfix sẽ được tích hợp vào nhánh này.
- **Nhánh phụ (Supporting Branches):**
  - **feature**: Được tạo từ nhánh `staging` để phát triển tính năng mới. Tên nhánh có dạng `feature/<tên-tính-năng>`.
  - **release**: Được tạo từ nhánh `staging` khi chuẩn bị phát hành phiên bản mới. Tên nhánh có dạng `release/<phiên-bản>`.
  - **hotfix**: Được tạo từ nhánh `main` để sửa lỗi khẩn cấp. Tên nhánh có dạng `hotfix/<tên-lỗi>`.

2. Quy trình Gitflow

a. Làm việc với nhánh Feature:

1. Tạo nhánh `feature` từ nhánh `staging`:

```
git checkout -b feature/<tên-tính-năng> staging
```

2. Phát triển tính năng và commit các thay đổi:

```
git add .
git commit -m "Phát triển tính năng <tên-tính-năng>"
```

3. Push nhánh `feature` lên repository từ xa:

```
git push origin feature/<tên-tính-năng>
```

4. Tạo Pull Request để merge nhánh `feature` vào `staging`.
5. Sau khi merge, xóa nhánh `feature`:

```
git push origin --delete feature/<tên-tính-năng>
git branch -d feature/<tên-tính-năng>
```

b. Làm việc với nhánh Hotfix:

1. Tạo nhánh `hotfix` từ nhánh `main`:

```
git flow hotfix start <tên-lỗi>
```

2. Sửa lỗi, sau đó merge nhánh vào `main` và `staging`:

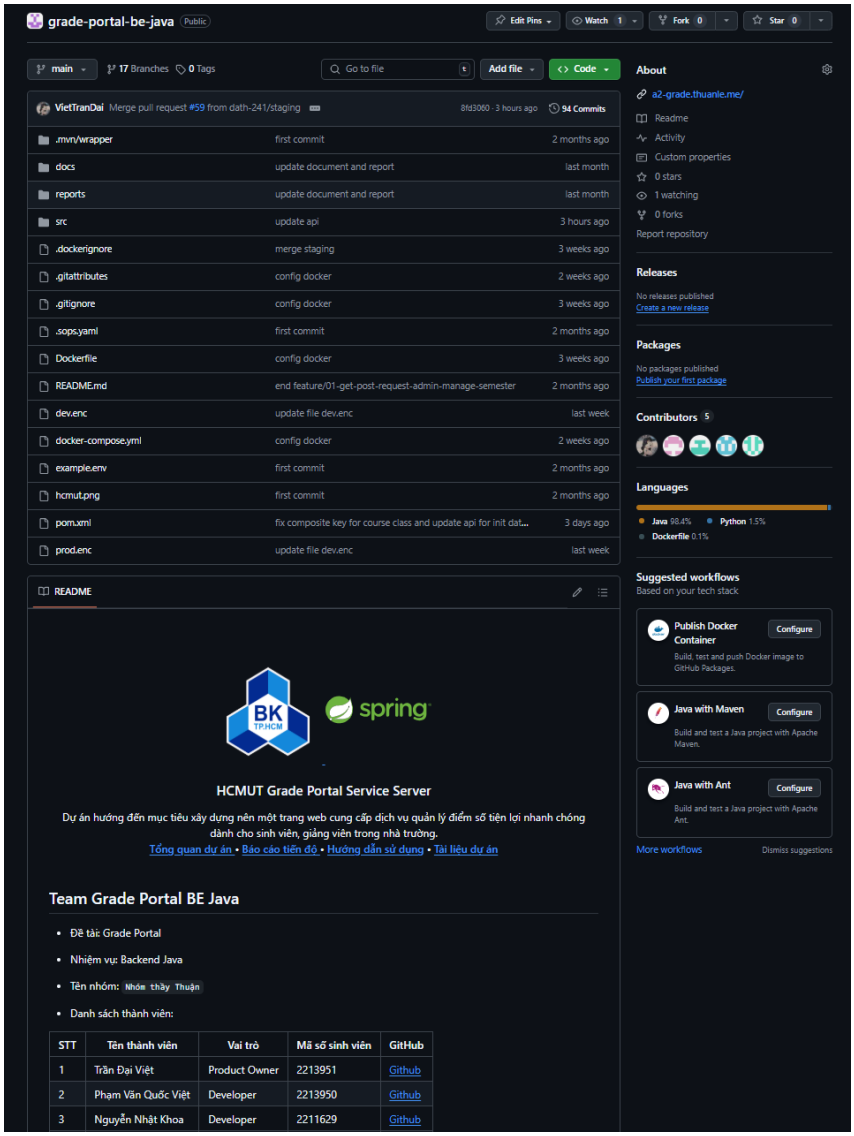
```
git flow hotfix finish <tên-lỗi>
```

3. Chú ý để tránh xung đột (Conflict) giữa các thành viên

- Luôn cập nhật nhánh `staging` hoặc `main` trước khi bắt đầu làm việc:

```
git checkout staging
git pull origin staging
```

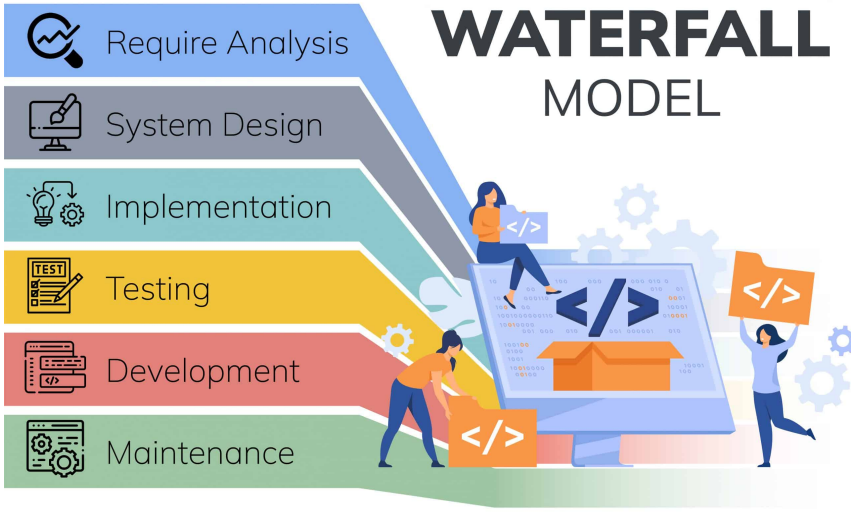
- Sử dụng nhánh riêng biệt cho từng tính năng hoặc sửa lỗi.
- Đặt tên nhánh theo quy ước để dễ quản lý, ví dụ:
  - `feature/<tên-tính-năng>`
  - `hotfix/<tên-lỗi>`
  - `release/<phiên-bản>`
- Đẩy nhánh lên repository từ xa thường xuyên để tránh xung đột.
- Thực hiện `Code Review` trên các Pull Request để đảm bảo chất lượng mã nguồn và phát hiện xung đột sớm.
- Tránh commit trực tiếp vào các nhánh chính như `main` hoặc `staging`.



Hình 25: *Repo Github cho Backend của dự án*

8 Quản lý phân công việc hiện thực cho các thành viên trong nhóm

8.1 Quy trình phát triển phần mềm - Mô hình Waterfall



Hình 26: *Quy trình Waterfall trong việc hiện thực dự án*

Trong quá trình thực hiện dự án, nhóm đã quyết định áp dụng quy trình phát triển phần mềm theo mô hình Waterfall. Đây là một quy trình truyền thống, thực hiện tuần tự qua các giai đoạn chính:

- Yêu cầu (Requirements)
- Thiết kế (Design)
- Triển khai (Implementation)
- Kiểm thử (Testing)
- Bàn giao (Delivery)

Mỗi giai đoạn cần được hoàn thành hoàn toàn trước khi chuyển sang giai đoạn tiếp theo. Điều này đảm bảo rằng mọi vấn đề hoặc khuyết điểm trong giai đoạn trước sẽ không ảnh hưởng đến các giai đoạn tiếp theo.

Lý do chọn Waterfall

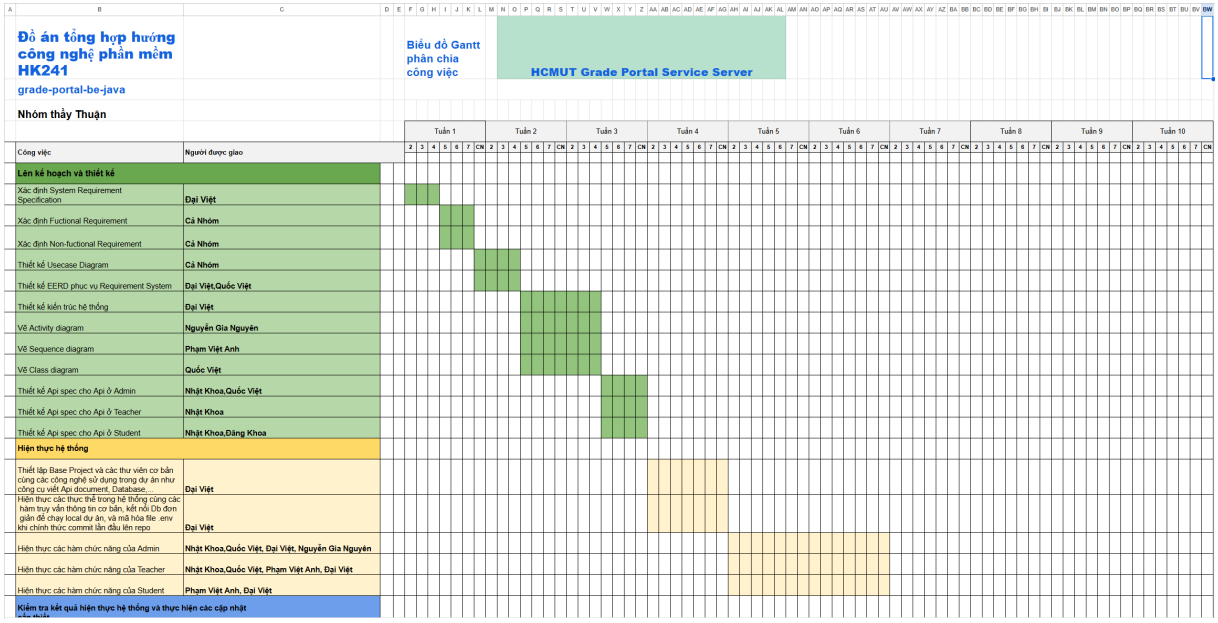
- **Phù hợp với các dự án có mục tiêu và yêu cầu rõ ràng từ đầu:** Mô hình Waterfall thích hợp cho bài tập lớn khi không cần thay đổi nhiều trong suốt quá trình thực hiện.
- **Yêu cầu ít thay đổi:** Với bài tập lớn, mục tiêu và yêu cầu thường ít thay đổi sau khi được xác định.
- **Thời gian và tài nguyên có giới hạn:** Mô hình Waterfall giúp nhóm tập trung hoàn thành từng giai đoạn một cách kỹ lưỡng, đảm bảo đúng tiến độ trong khung thời gian cố định.
- **Tài liệu hóa đầy đủ:** Mô hình này khuyến khích tạo tài liệu chi tiết cho từng giai đoạn, hỗ trợ tốt cho yêu cầu nộp báo cáo cuối kỳ.

Ví dụ cụ thể trong bài tập lớn này, mục tiêu bài tập đã được xác định rõ ràng ngay từ đầu (xây dựng hệ thống quản lý điểm số thông minh). Nhóm không cần sự linh hoạt hoặc thay đổi liên tục như các dự án thực tế, do đó mô hình Waterfall là lựa chọn phù hợp.

8.2 Biểu đồ Gantt hỗ trợ lên kế hoạch

Để hỗ trợ quá trình lên kế hoạch và quản lý phân công công việc trong nhóm, nhóm đã xây dựng biểu đồ Gantt (như hình minh họa bên dưới). Biểu đồ Gantt thể hiện rõ ràng các công việc, người thực hiện, và thời gian hoàn thành dự kiến của từng nhiệm vụ.

Biểu đồ Gantt chi tiết này được hiện thực và quản lý thông qua Google Sheets tại link: **Tài liệu Gantt**. Tại đây, nhóm có thể theo dõi tiến độ và điều chỉnh kế hoạch khi cần thiết.



Hình 27: *Biểu đồ Gantt phân chia công việc*

9 Tổng kết công việc hiện thực của nhóm

Sau quá trình phân tích, thiết kế và hiện thực hệ thống backend cho hệ thống quản lý điểm số trực tuyến tại Đại học Bách Khoa - Đại học Quốc gia TP.HCM, nhóm đã đạt được các kết quả quan trọng như sau:

- Hoàn thành phân tích yêu cầu nghiệp vụ, xác định rõ các chức năng chính của hệ thống backend, đảm bảo đáp ứng nhu cầu tra cứu, quản lý điểm số và phân quyền cho các vai trò như sinh viên, giảng viên, và admin.
- Thiết kế và hiện thực kiến trúc backend theo mô hình Layered Architecture, bao gồm các tầng Controller, Service, Repository và Database:
  - Tầng Controller: Đảm nhận việc tiếp nhận và điều hướng yêu cầu từ người dùng.
  - Tầng Service: Xử lý logic nghiệp vụ, bao gồm việc xác định vai trò người dùng và quản lý dữ liệu liên quan.
  - Tầng Repository: Đảm nhận việc giao tiếp với cơ sở dữ liệu.
  - Tầng Database: Lưu trữ thông tin bền vững và đảm bảo tính toàn vẹn dữ liệu.
- Hiện thực các API để hỗ trợ các tính năng chính như:
  - Đăng nhập và xác thực người dùng thông qua Google Authentication.
  - Tra cứu, quản lý điểm số của sinh viên.
  - Phân quyền và quản lý tài khoản cho giảng viên và admin.
- Xây dựng cơ chế bảo mật cho hệ thống, bao gồm:
  - Sử dụng JSON Web Token (JWT) để xác thực và ủy quyền.
  - Đảm bảo dữ liệu được truy cập chỉ bởi những người dùng được phân quyền.

Khó khăn trong quá trình thực hiện: Trong quá trình thực hiện dự án, nhóm đã gặp phải một số khó khăn như sau:

- Kiến thức về các công cụ và thư viện backend (như Spring Boot,...) yêu cầu nhiều thời gian để tìm hiểu và làm quen, đặc biệt là với các thành viên chưa có nhiều kinh nghiệm lập trình backend.
- Việc tích hợp Google Authentication và xây dựng cơ chế xác thực JWT đòi hỏi phải nắm rõ các nguyên tắc về bảo mật và ủy quyền, điều này gây khó khăn trong giai đoạn đầu.
- Quá trình kiểm tra và debug hệ thống backend, đặc biệt là với các yêu cầu liên quan đến bảo mật dữ liệu.
- Phối hợp và phân chia công việc giữa các thành viên để đảm bảo tiến độ và chất lượng của hệ thống.

Kết quả đạt được: Dự án đã hoàn thành đúng tiến độ, đáp ứng các mục tiêu đặt ra ban đầu. Hệ thống backend được thiết kế và hiện thực đảm bảo hiệu suất, bảo mật, và khả năng mở rộng, sẵn sàng để tích hợp với các hệ thống frontend trong tương lai.

Nhóm xin gửi lời cảm ơn chân thành đến thầy **Lê Đình THuận**, người đã tận tình hướng dẫn và hỗ trợ nhóm trong suốt quá trình thực hiện dự án. Những góp ý và hướng dẫn từ thầy đã giúp nhóm nắm vững hơn các kiến thức về backend và áp dụng chúng vào thực tế, đồng thời củng cố kỹ năng làm việc nhóm và giải quyết các vấn đề phức tạp trong phát triển phần mềm.

Kết luận: Hệ thống backend của Grade Portal là một bước tiến lớn trong việc hiện đại hóa công tác quản lý điểm số tại Đại học Bách Khoa. Với nền tảng đã xây dựng, hệ thống không chỉ hỗ trợ các chức năng cơ bản một cách hiệu quả mà còn mở ra khả năng phát triển và mở rộng trong tương lai, góp phần nâng cao chất lượng giáo dục và trải nghiệm học tập tại nhà trường.

10 Đường dẫn tới từng kết quả hiện thực của nhóm

Phía dưới đây là toàn bộ kết quả hiện thực của bài tập lớn của nhóm chúng em bao gồm mã nguồn, các diagram được hiện thực, slide và video thuyết trình demo dự án,...

1. Các Diagram được hiện thực:
   - **Usecase Diagram:**
     `https://drive.google.com/file/d/1SVd3zjJVpOsC4I7qyyPzUe3PvnhHKlQ_/view?usp=drive_link`
   - **Activity Diagram, Sequence Diagram và Component Diagram:**
     `https://drive.google.com/file/d/1bbg4_3zcnG-_MxGdfKAUyBxPP1IoeODp/view?usp=drive_link`
   - **Architecture Diagram, Database design và Class Diagram:**
     `https://drive.google.com/file/d/1SVd3zjJVpOsC4I7qyyPzUe3PvnhHKlQ_/view?usp=drive_link`
2. **Biểu đồ Gantt phân chia công việc**
   `https://docs.google.com/spreadsheets/d/1gHI25VZbI2IiyjJD7OkQ2sZHf8J-NaBwq7-RNx2U5-0/edit?usp=sharing`
3. Repo Github chứa mã nguồn của dự án:
   - **HCMUT Grade Portal Service Server:**
     `https://github.com/dath-241/grade-portal-be-java`
4. Slide và Video Presentation báo cáo kết quả hiện thực dự án:
   - Slide báo cáo của nhóm:
     `https://drive.google.com/file/d/1ZxBFmCUtsHKjyRBRmFbfhrAyHm78kOkW/view?usp=sharing`
   - Video báo cáo dự án của nhóm:
     `https://drive.google.com/file/d/1ZxBFmCUtsHKjyRBRmFbfhrAyHm78kOkW/view?usp=sharing`