

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ 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	Kiến trúc hệ thống (System Architecture)	2
2.1	Layered Architecture	2
2.2	Sơ đồ hiện thực của hệ thống	3
2.2.1	Presentation Layer	4
2.2.2	Business Layer	5
2.2.3	Persistence Layer	7
2.2.4	Data Layer	8
2.3	Deployment Diagram	9
2.4	Data Storage Approach	10
2.5	API management	12

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 Kiến trúc hệ thống (System Architecture)

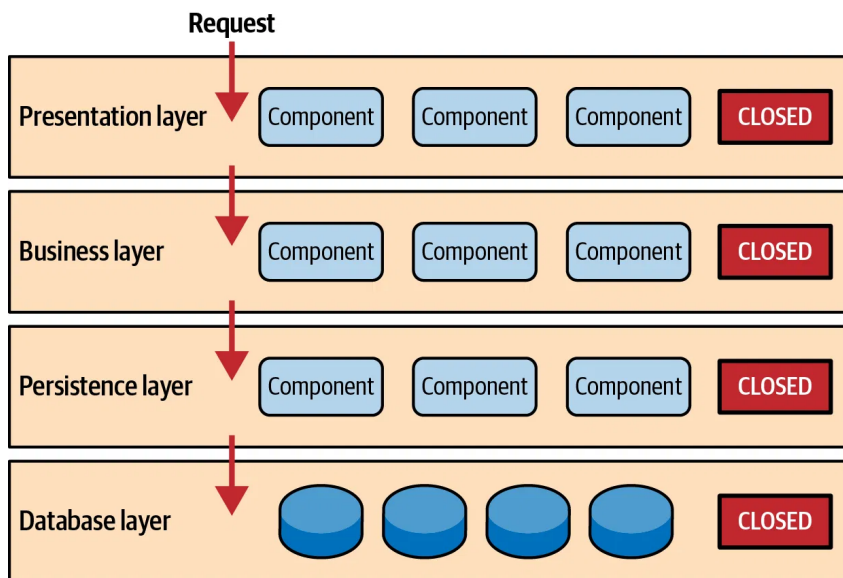
2.1 Layered Architecture

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.

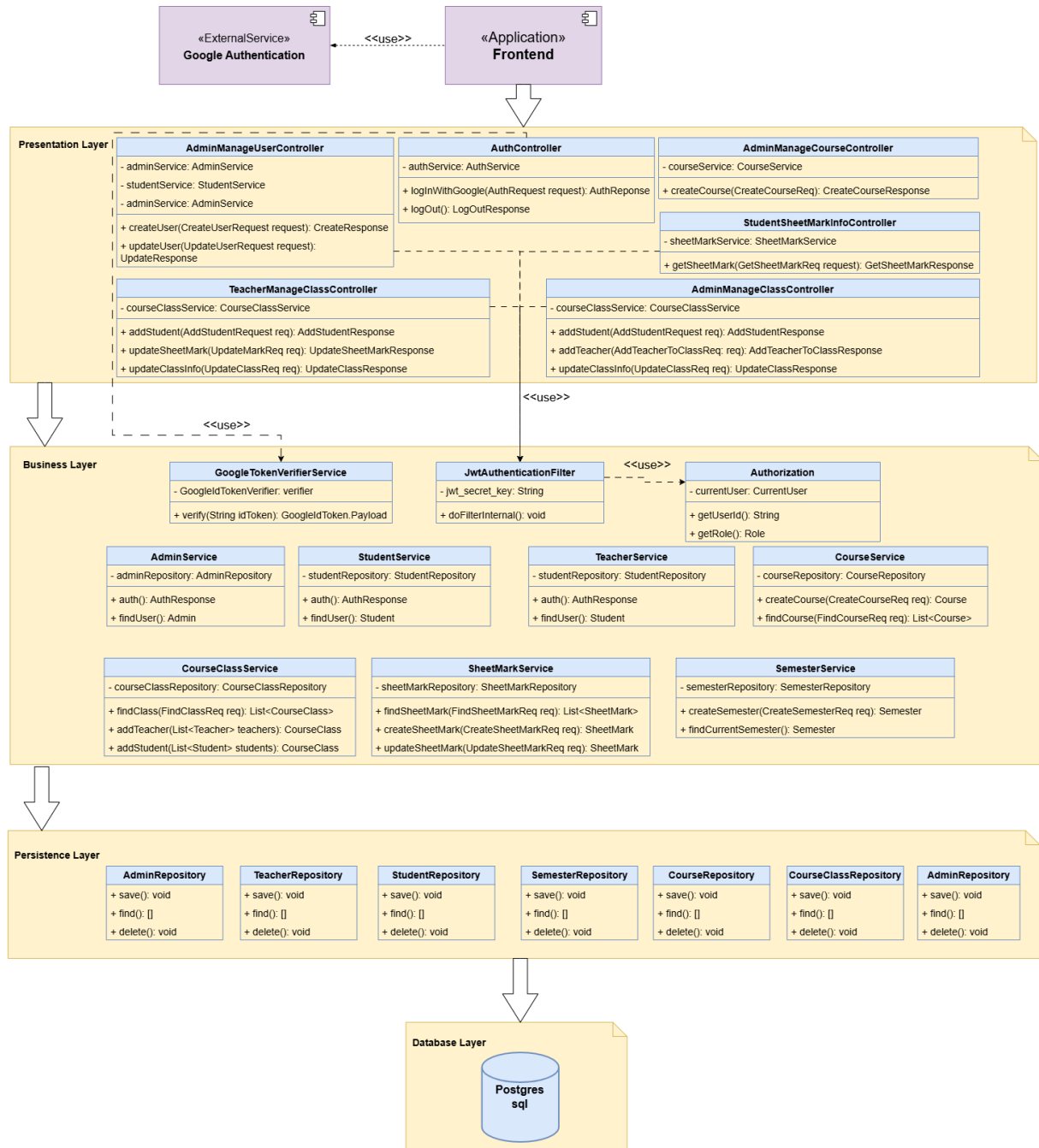


Hình 1: Sơ đồ kiến trúc phân tầng

Đây là một architecture pattern 3 lớp vật lý (3-tier) hay n-tier. Tất cả logic đều 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.

2.2 Sơ đồ hiện thực của hệ thống

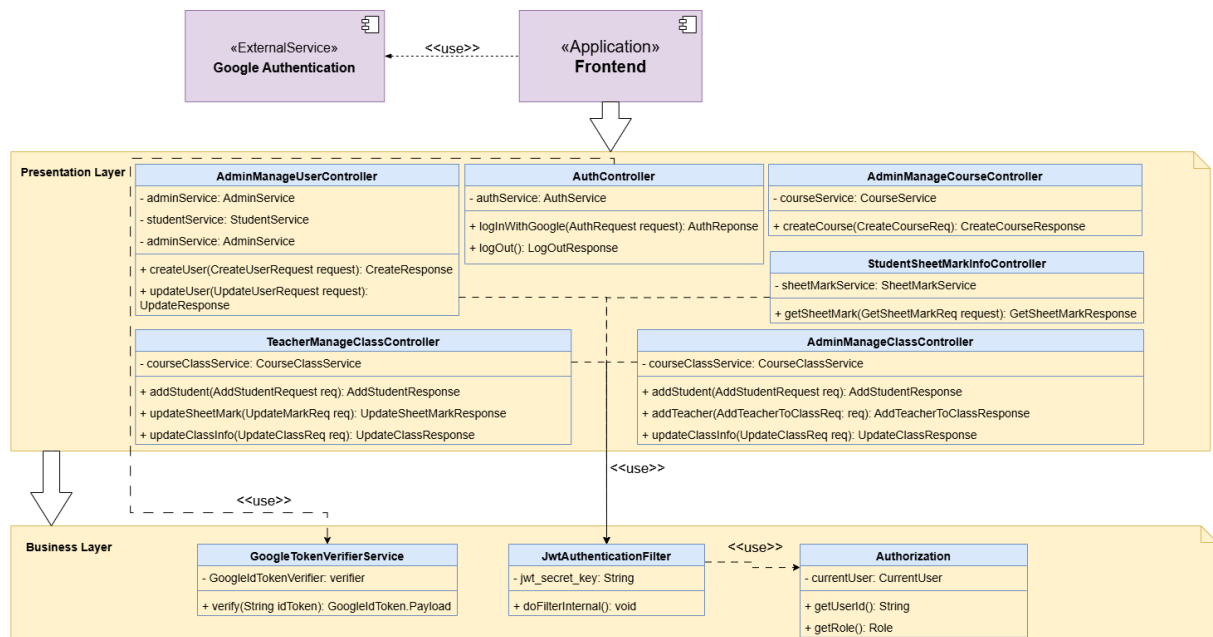
Trong phần này nhóm đã sử dụng **Draw.io** để hiện thực sơ đồ. Đường dẫn cụ thể tới sơ đồ được hiện thực nằm ở [đây](#).



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).

2.2.1 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:**

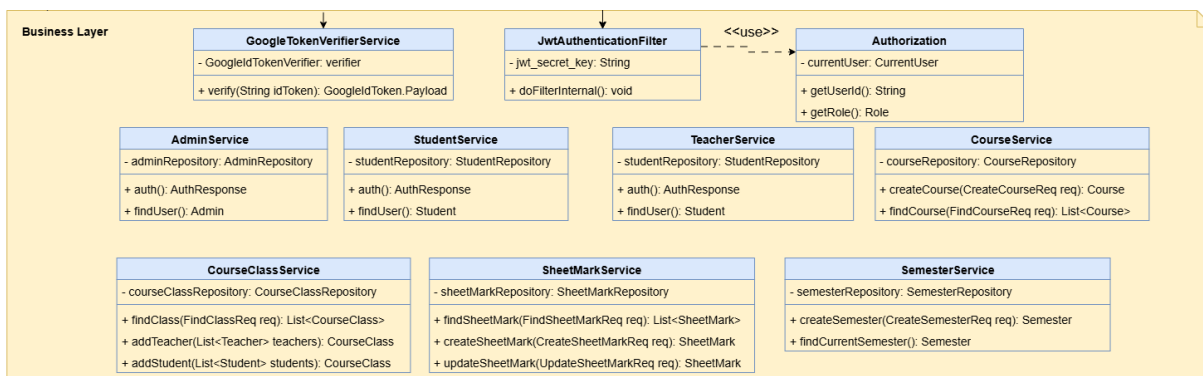
- * Quản lý lớp học bao gồm học sinh và giáo viên.
- * Ví dụ chức năng: thêm học sinh vào lớp (addStudent), thêm giáo viên vào lớp (addTeacher) và cập nhật thông tin lớp học (updateClassInfo).
- * Phân quyền để đảm bảo chỉ người dùng có vai trò ADMIN hoặc TEACHER mới được phép thực hiện các chức năng 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.

2.2.2 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.

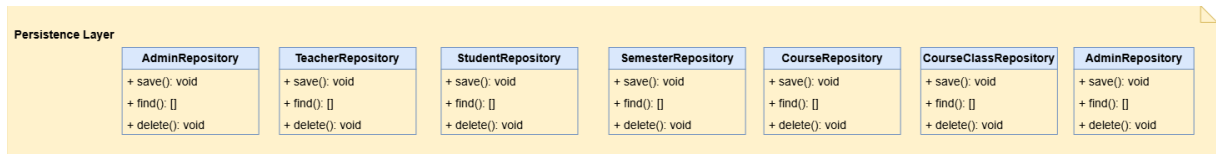
• **Vai trò chính:**

- 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.

2.2.3 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:**

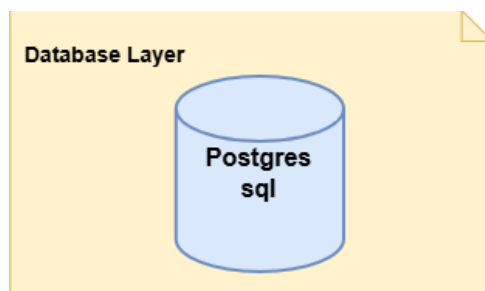
- * Chịu trách nhiệm lưu trữ và truy vấn dữ liệu liên quan đến lớp học.
- * Chức năng: `save()`, `find()`, `delete()`.

- **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.

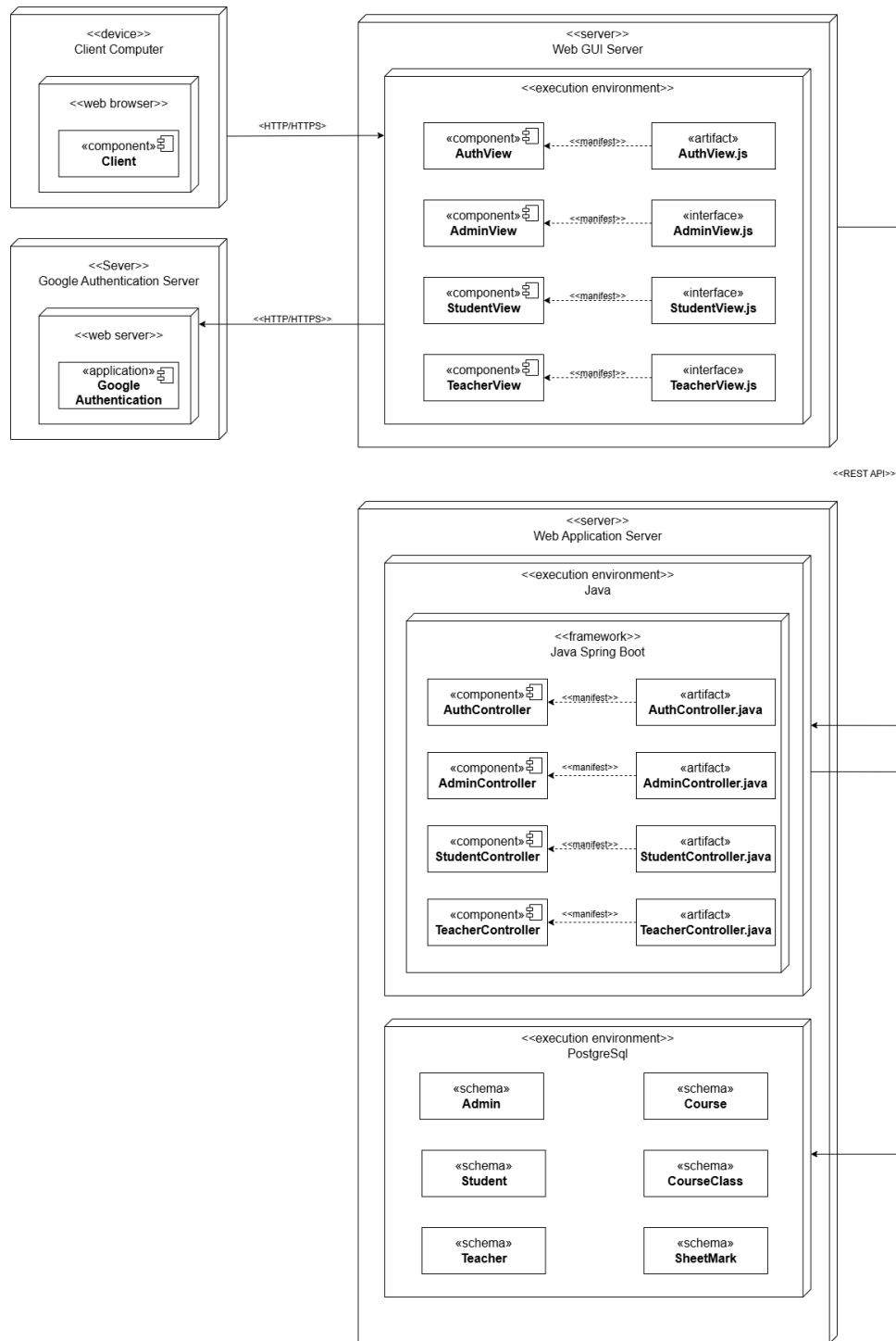
2.2.4 Data Layer



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.

2.3 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.

2.4 Data Storage Approach

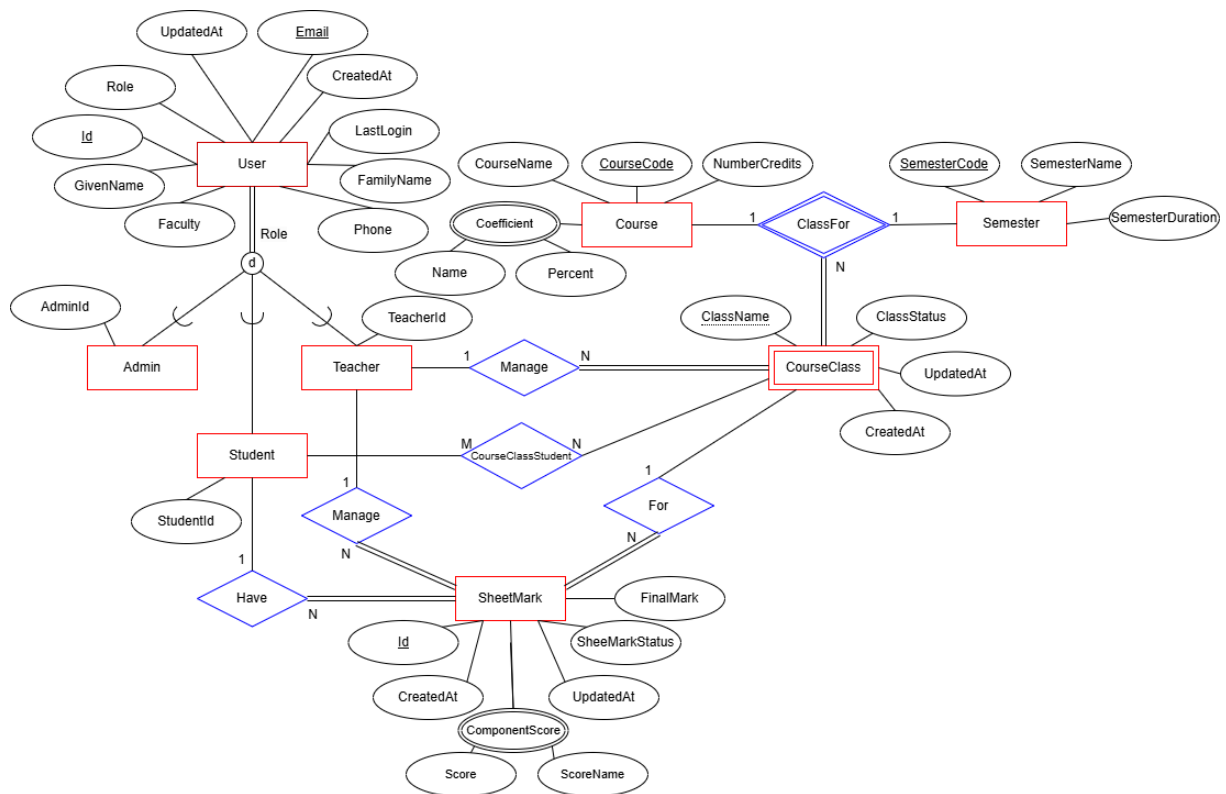
Đối với các yêu cầu của hệ thống Grade Portal, không khó để nhận ra, các hành động liên quan tới việc truy xuất, xử lý và cập nhập dữ liệu là rất quan trọng và cần sự chính xác cao. Và đó cũng chính là một trong những lí do kiến trúc phân tầng được chọn để hiện thực hệ thống, các thao tác liên quan tới 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 Annotation @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 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



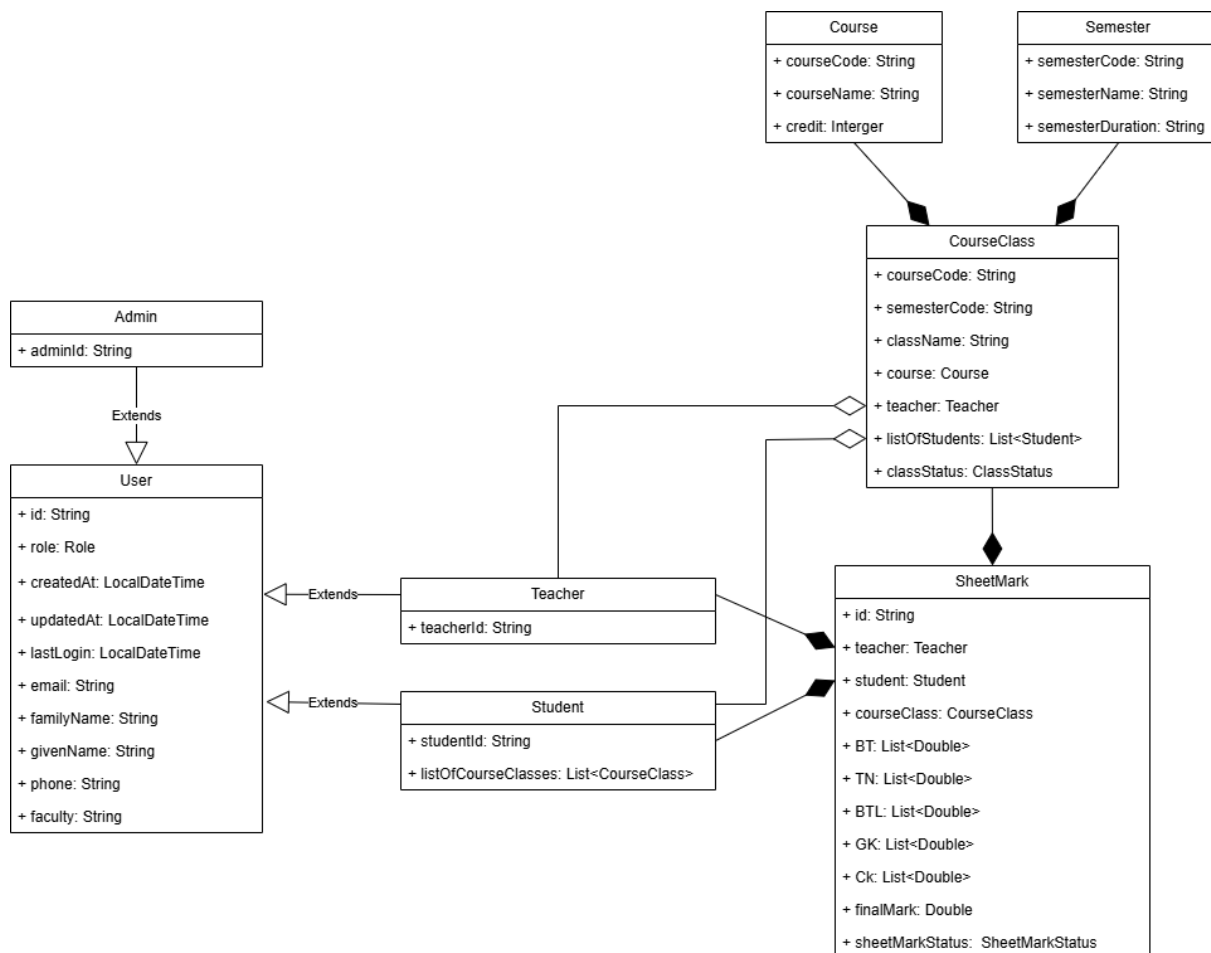
Hình 8: EERD Diagram

Relational Data Modal



Hình 9: *Relational Data Modal*

Class Diagram



Hình 10: Class Diagram

2.5 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 diện 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):

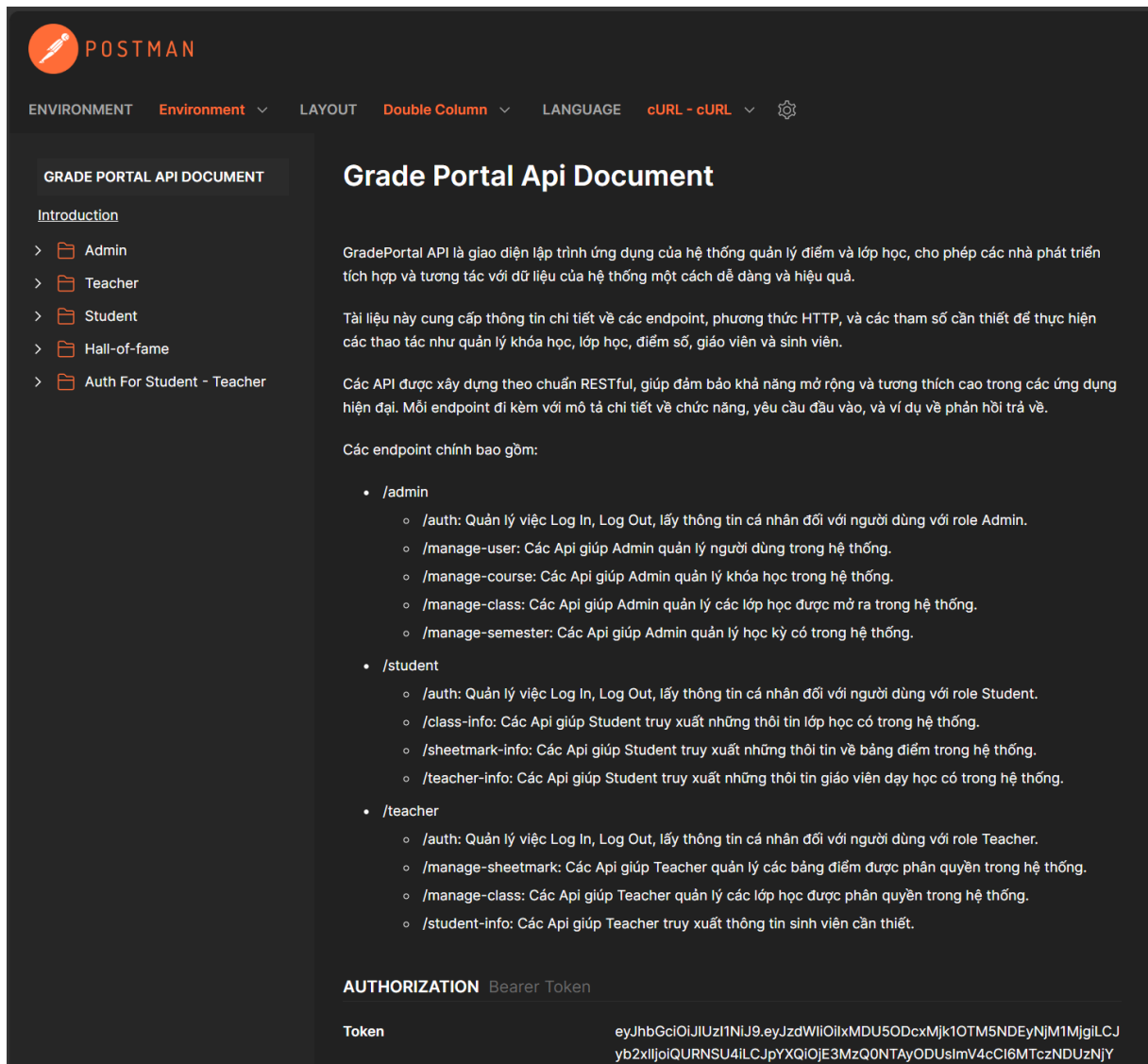
- Sử dụng JWT cho việc xác thực giữa các thành phần trong hệ thống, đảm bảo rằng chỉ những yêu cầu có token hợp lệ mới được xử lý. JWT là giải pháp nhẹ và phổ biến cho xác thực API, phù hợp cho các dự án còn mang tính chất đơn giản.

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](#).



POSTMAN

ENVIRONMENT **Environment** LAYOUT **Double Column** LANGUAGE **cURL - cURL**

GRADE PORTAL API DOCUMENT

Grade Portal Api Document

Introduction

GradePortal API là giao diện lập trình ứng dụng của hệ thống quản lý điểm và lớp học, cho phép các nhà phát triển tích hợp và tương tác với dữ liệu của hệ thống một cách dễ dàng và hiệu quả.

Tài liệu này cung cấp thông tin chi tiết về các endpoint, phương thức HTTP, và các tham số cần thiết để thực hiện các thao tác như quản lý khóa học, lớp học, điểm số, giáo viên và sinh viên.

Các API được xây dựng theo chuẩn RESTful, giúp đảm bảo khả năng mở rộng và tương thích cao trong các ứng dụng hiện đại. Mỗi endpoint đi kèm với mô tả chi tiết về chức năng, yêu cầu đầu vào, và ví dụ về phản hồi trả về.

Các endpoint chính bao gồm:

- **/admin**
 - **/auth**: Quản lý việc Log In, Log Out, lấy thông tin cá nhân đối với người dùng với role Admin.
 - **/manage-user**: Các Api giúp Admin quản lý người dùng trong hệ thống.
 - **/manage-course**: Các Api giúp Admin quản lý khóa học trong hệ thống.
 - **/manage-class**: Các Api giúp Admin quản lý các lớp học được mở ra trong hệ thống.
 - **/manage-semester**: Các Api giúp Admin quản lý học kỳ có trong hệ thống.
- **/student**
 - **/auth**: Quản lý việc Log In, Log Out, lấy thông tin cá nhân đối với người dùng với role Student.
 - **/class-info**: Các Api giúp Student truy xuất những thông tin lớp học có trong hệ thống.
 - **/sheetmark-info**: Các Api giúp Student truy xuất những thông tin về bảng điểm trong hệ thống.
 - **/teacher-info**: Các Api giúp Student truy xuất những thông tin giáo viên dạy học có trong hệ thống.
- **/teacher**
 - **/auth**: Quản lý việc Log In, Log Out, lấy thông tin cá nhân đối với người dùng với role Teacher.
 - **/manage-sheetmark**: Các Api giúp Teacher quản lý các bảng điểm được phân quyền trong hệ thống.
 - **/manage-class**: Các Api giúp Teacher quản lý các lớp học được phân quyền trong hệ thống.
 - **/student-info**: Các Api giúp Teacher truy xuất thông tin sinh viên cần thiết.

AUTHORIZATION Bearer Token

Token eyJhbGciOiJIUzI1NiJ9.eyJzdWwiOiIxMDU5ODcxMjk1OTM5NDYyMjM1MjgiLCJyb2xlljoicURNSU4iLCJpYXQiOiE3MzQ0NTAyODUsImV4cCI6MTczNDUzNjY

Hình 11: Postman Grade Portal Api Document