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

BÁO CÁO NHIỆM VỤ LÀM VIỆC

MÔN HỌC: Công Nghệ Phần Mềm

Đề tài: A smart printing service for students at HCMUT

HK241

Giảng viên hướng dẫn: Trần Trương Tuấn Phát
Sinh viên: Trần Đại Việt - 2213951 - L02
Lương Thanh Tùng - 2213866 - L02
Trần Ngọc Châu Long - 2111682 - L04
Trần Trung Kiên - 2211738 - L02
Trần Quang Huy - 2211288 - L02
Lê Đăng Khoa - 2211599 - L02

HO CHI MINH CITY, SEPTEMBER 2024

Mục lục

1 Kiến trúc hệ thống (System Architecture)

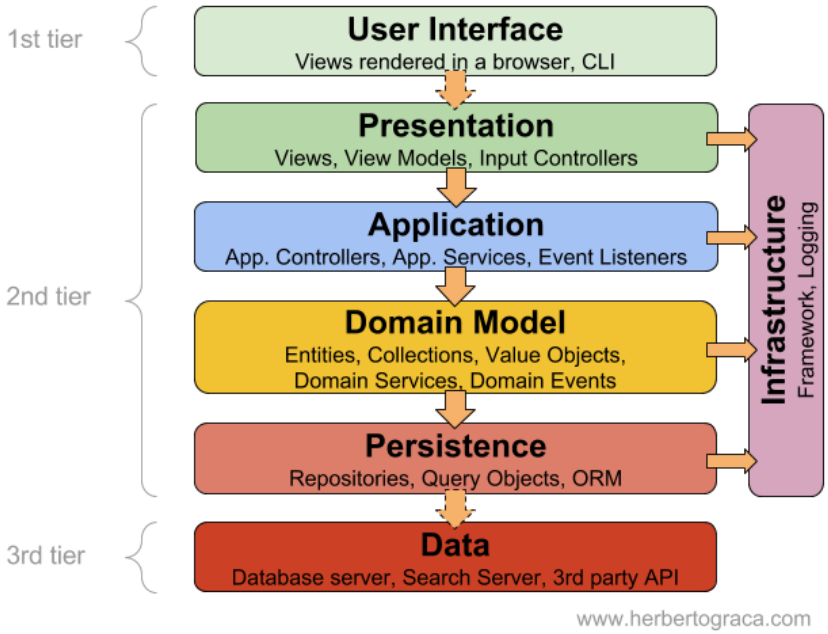
1.1 Layered Architecture

Khi phát triển một hệ thống như Student Smart Printing Service (HCMUT_SSPS), 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 HCMUT_SSPS, nơi có nhiều tính năng liên quan đến in ấn, quản lý số lượng trang in, theo dõi lịch sử in ấn, và giao dịch thanh toán, 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 HCMUT_SSPS, đây vẫn là lựa chọn khả thi cho giai đoạn phát triển ban đầu của hệ thống. 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 HCMUT_SSPS 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.

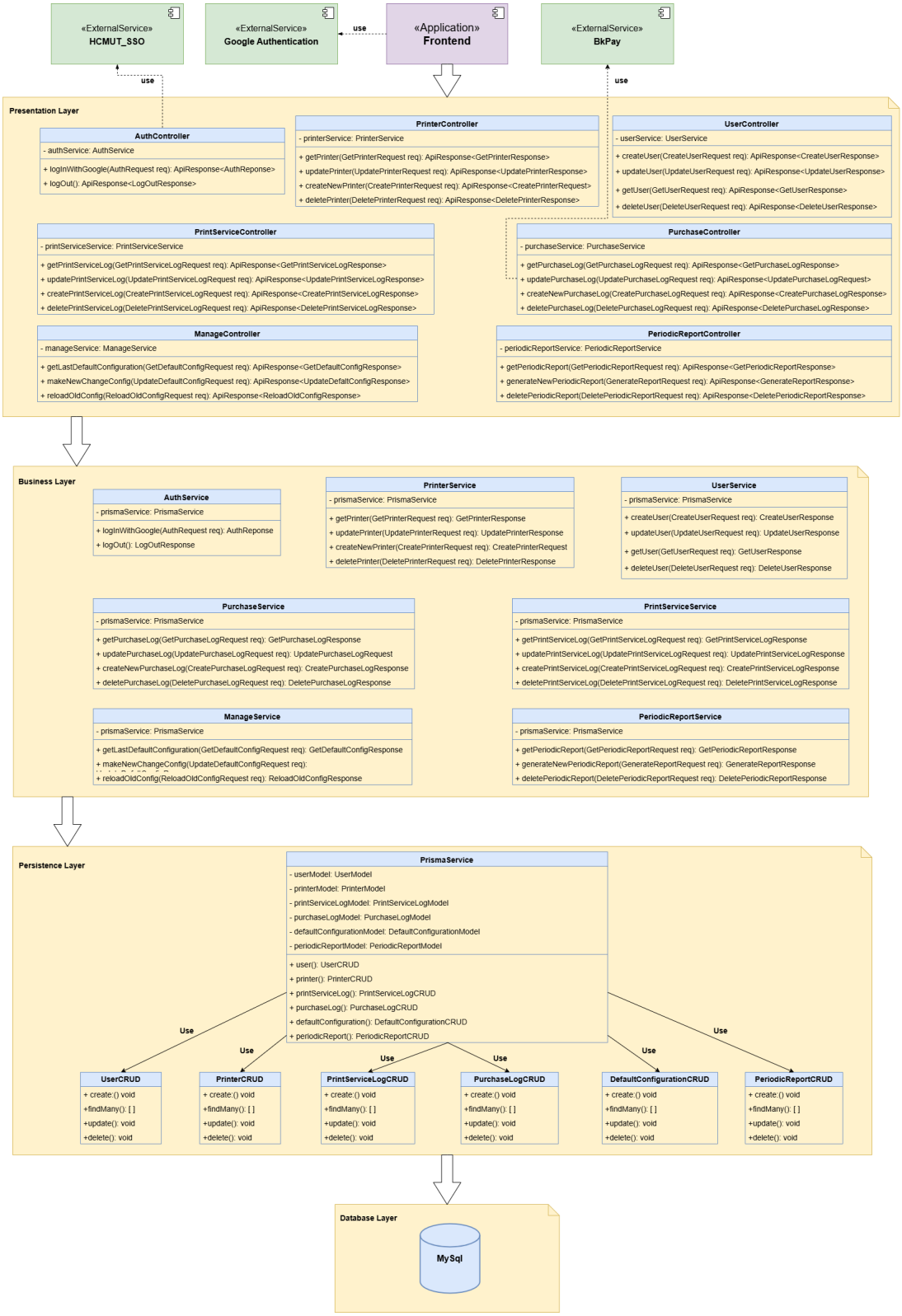


Hình 1: *Sơ đồ kiến trúc phân tầng*

Đây là một architecture pattern 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.

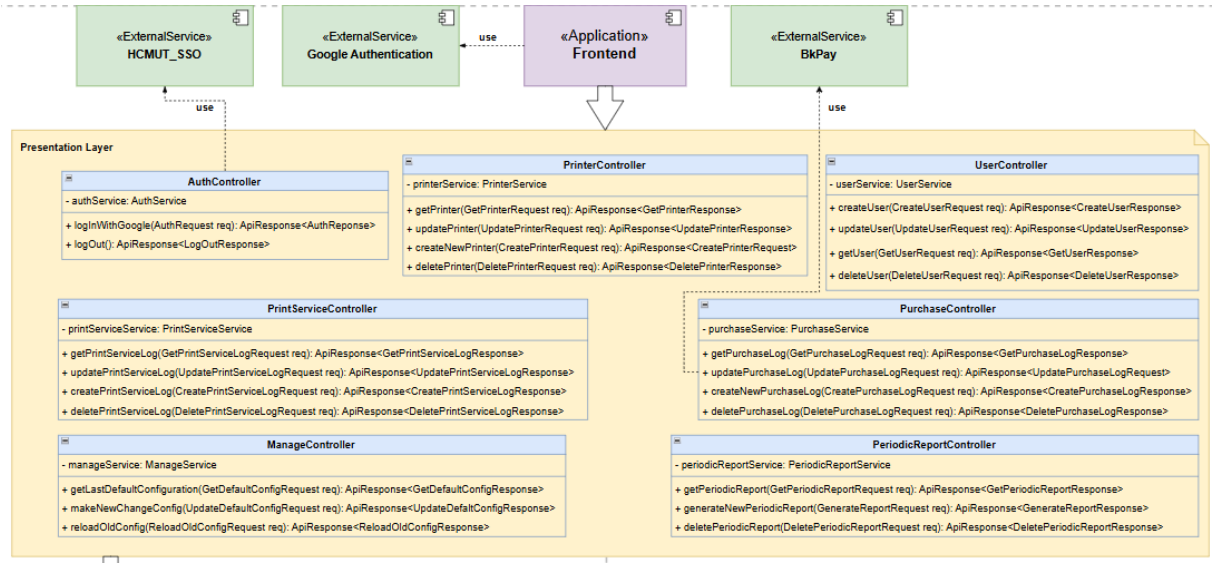
1.2 Triển khai kiến trúc vào hệ thống

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 áp dụng kiến trúc phân tầng vào hệ thống*

1.2.1 Presentation Layer

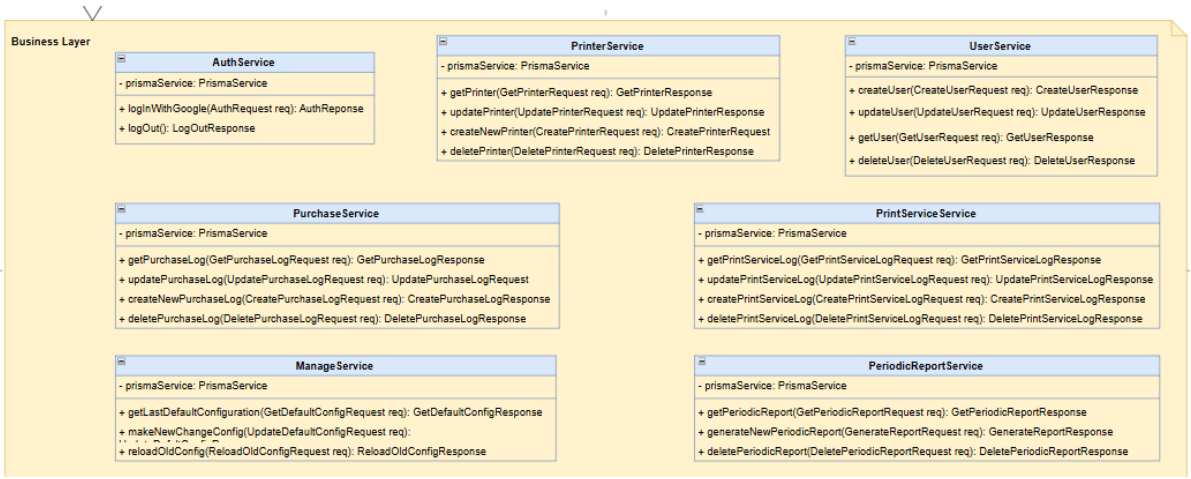


Hình 3: *Presentation Layer Of Layered Architecture*

Để có thể đem đến cho hệ thống khả năng cung cấp cho người dùng sự tương tác và cập nhập thông tin trực tiếp với giao diện, Presentation Layer là tầng đầu tiên, cũng như là tầng đảm nhiệm việc tiếp nhận các lần gọi Api từ phía Client có thể từ Mobile App hay Web App. Nhiệm vụ của tầng này là tiếp nhận và xử lý dữ liệu đầu vào được người dùng gửi từ giao diện, sau đó kiểm tra tính đầy đủ, đúng đắn để sau đó chuyển tiếp chúng xuống tầng Business Layer nơi hiện thực đầy đủ logic hệ thống để xử lý yêu cầu của người dùng. Ngoài ra đây cũng là tầng phải đảm bảo rằng dữ liệu trả về cho Client là đúng đắn, có nghĩa là phải đúng với cấu trúc được mô tả trong Document.

Tóm lại, trong kiến trúc phân tầng, tầng **Presentation** đóng một vai trò quan trọng, chịu trách nhiệm tiếp nhận lời gọi từ Client, cũng như đảm bảo dữ liệu đầu vào và đầu ra để cho hệ thống hoạt động một cách đúng đắn nhất.

1.2.2 Business Layer



Hình 4: *Business Layer Of Layered Architecture*

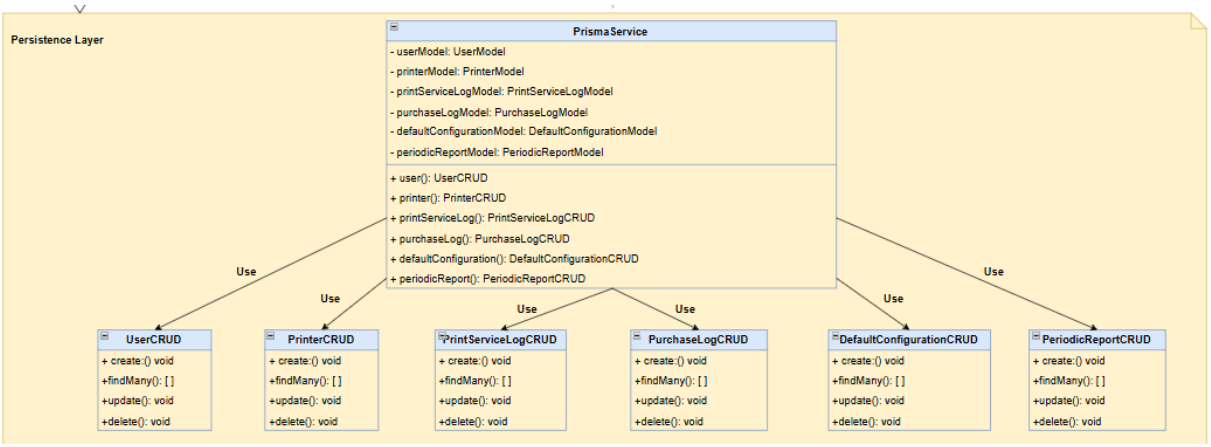
Tiếp ngay sau tầng Presentation là tầng Business, đây là tầng chịu trách nhiệm cho việc đảm bảo Logic nghiệp vụ của hệ thống. Nó sẽ nhận dữ liệu đầu vào được truyền xuống từ tầng Presentation sau đó hiện thực Logic nghiệp vụ ứng với lời gọi từ Client, ngoài ra nó còn có thể gọi dữ liệu hay các function được cung cấp từ tầng Persistence để truy vấn hoặc cập nhập dữ liệu trong Database nếu cần. Ngoài ra để phần nào giải quyết việc khó bảo trì hay

phát triển codebase do ảnh hưởng của kiến trúc Monolithic, các logic nghiệp vụ thường được tổ chức thành các Module có thể tái sử dụng ứng với từng nghiệp vụ khác nhau của hệ thống.

Trong trường hợp đó, việc tổ chức cũng như sửa chữa Logic nghiệp vụ của hệ thống sẽ không ảnh hưởng nhiều tới các Api gọi từ Client, bởi lẽ, các service đối với tầng Presentation như các hàm trừu tượng, đồng nghĩa với việc tầng này sẽ không quan tâm tới Logic được hiện thực như thế nào mà chỉ quan tâm kết quả trả về. Vì vậy việc sửa chữa hay bảo trì các logic nghiệp vụ sẽ không ảnh hưởng tới các Api mà hệ thống cung cấp.

Nhìn chung, việc chia tầng như thế này sẽ giúp nhóm có một cái nhìn trực quan hơn về luồng đi cũng như cấu trúc của dữ liệu đầu vào, đầu ra như thế nào để việc hiện thực báo cáo sẽ hiểu quả hơn.

1.2.3 Persistence Layer



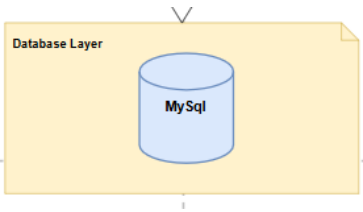
Hình 5: *Persistence Layer Of Layered Architecture*

Để xử lý các nhu cầu về việc truy vấn và cập nhật dữ liệu trong database, Persistence Layer sẽ là nơi chứa các đối tượng trung gian thực hiện điều đó. Các đối tượng này sẽ đảm nhiệm việc gửi các yêu cầu đến lớp Database để thực hiện các thao tác liên quan đến dữ liệu.

Trong hệ thống này, các đối tượng đó sẽ là các Prisma Service và Prisma Model, giúp cho người phát triển thuận tiện hơn trong việc tự động tạo ra các câu truy vấn trực tiếp tới MySQL. Prisma cung cấp các hàm trừu tượng giúp tối ưu hóa quá trình truy vấn và quản lý dữ liệu một cách dễ dàng. Ngoài ra, khi sử dụng tầng này, việc thay đổi nhà cung cấp dịch vụ lưu trữ dữ liệu cũng trở nên dễ dàng hơn, ví dụ như chuyển đổi giữa các hệ quản trị cơ sở dữ liệu như MySQL và PostgreSQL mà không cần chỉnh sửa nhiều trong mã nguồn.

Tóm lại, đây sẽ là tầng chịu trách nhiệm cho việc tương tác trực tiếp với nơi lưu dữ liệu của hệ thống, giúp hệ thống hoạt động linh hoạt và dễ bảo trì hơn.

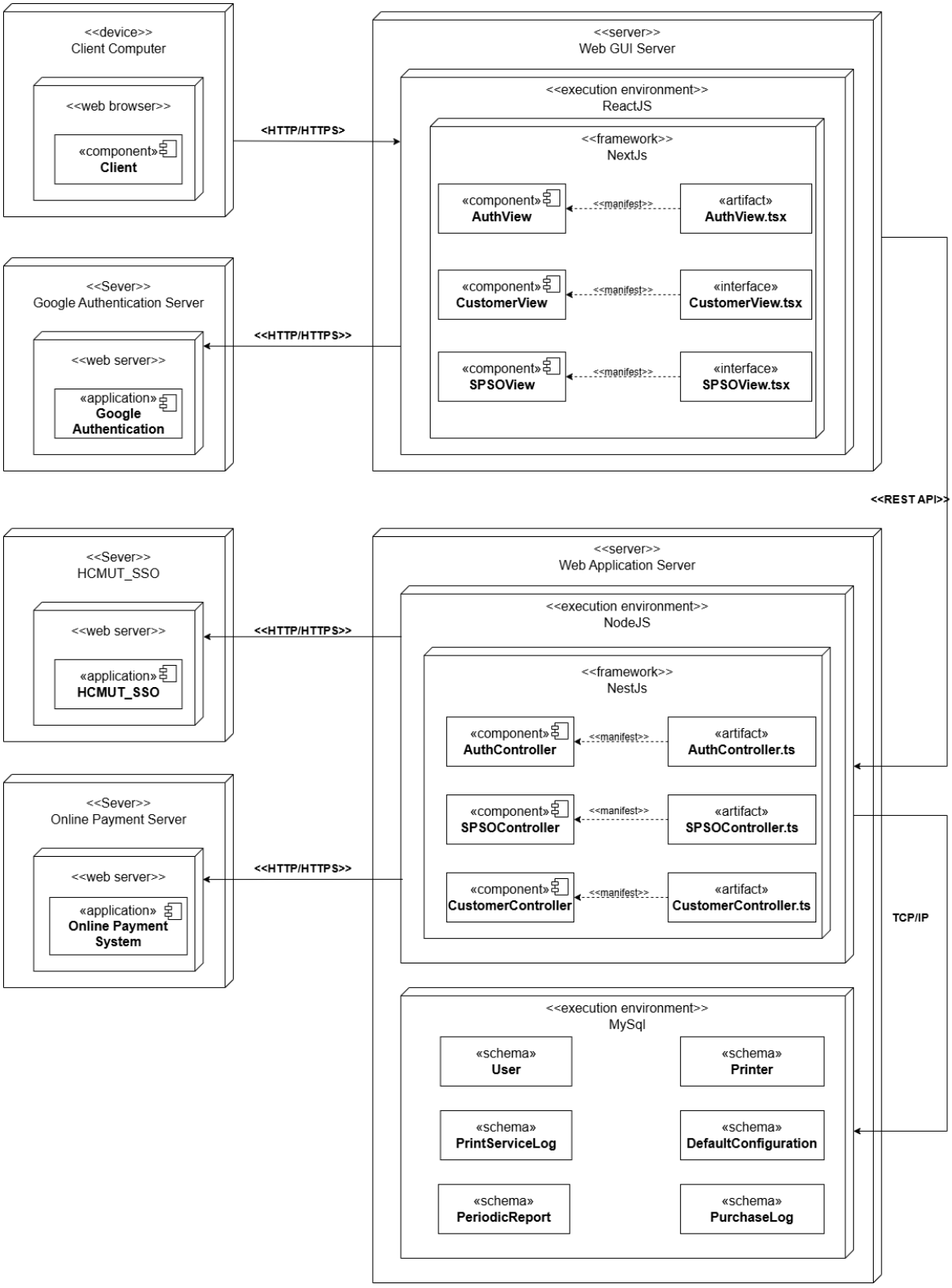
1.2.4 Data Layer



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.

1.3 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.

Trong sơ đồ này, Client Computer đại diện cho thiết bị của người dùng. Người dùng truy cập vào ứng dụng thông qua trình duyệt web, được biểu diễn bằng Web Browser. Từ trình duyệt, người dùng gửi các yêu cầu thông qua giao thức HTTP hoặc HTTPS tới Web GUI Server để thực hiện các thao tác với giao diện Web của hệ thống.

Google Authentication Server là một máy chủ được sử dụng để xác thực người dùng qua Google Authentication. Khi cần xác thực, ứng dụng trên Web sẽ Redirect người dùng sang phần giao diện xác thực tài khoản do Google cung cấp, sau khi xác thực thành công thì phía Client sẽ tự động gửi thông tin tài khoản xuống BE để kiểm tra một lần nữa trước khi chuyển người dùng tới phần giao diện tương ứng cho từng Role.

Web GUI Server là thành phần cung cấp giao diện người dùng cho hệ thống. Được triển khai với môi trường thực thi ReactJS trên nền tảng NextJS, server này sẽ cung cấp các giao diện như AuthView với việc đăng nhập hay xác thực Role người dùng, CustomerView là giao diện để cung cấp dịch vụ hệ thống cho khách hàng, và SPSOView là giao diện phục vụ cho việc quản lý của nhân viên trong hệ thống in ấn (SPSO). Web GUI Server sẽ gửi các yêu cầu qua REST API tới Web Application Server để thực hiện các tác vụ backend.

Web Application Server chịu trách nhiệm xử lý các Logic nghiệp vụ của hệ thống. Được triển khai bằng NodeJS trên framework NestJS, server này bao gồm các controller để cung cấp các EndPoint Api cho phía FE để nhận và xử lý dữ liệu được gửi từ Client. Ngoài ra Web Application Server giao tiếp với MySQL Database qua giao thức TCP/IP để xử lý các vấn đề liên quan tới việc truy xuất và cập nhập dữ liệu.

1.4 Presentation strategy

Với sự quan trọng và cấp thiết của dự án này, nếu hệ thống HCMUT SSPS được hiện thực thành công, nó sẽ đóng một vai trò quan trọng trong công việc hàng ngày của sinh viên, giáo viên và các cán bộ trong nhà trường đại học Bách khoa. Vì vậy, bên cạnh việc xây dựng một kiến trúc hệ thống phù hợp để hiện thực logic nghiệp vụ, việc xây dựng giao diện người dùng cho hệ thống cũng là một việc không kém tầm quan trọng.

Giao diện người dùng trang Web là tầng đầu tiên cũng như là thứ giúp người dùng nhận được dịch vụ mà hệ thống cung cấp, vì vậy tuy hệ thống được hiện thực với Logic nghiệp vụ rõ ràng, chính xác, nhưng lại đi kèm với UI quá sơ sài, thiếu tiện lợi và thân thiện với người dùng, sẽ phần nào khiến cho giá trị hệ thống xây dựng ra giảm xuống. Ngoài ra, như đã đề cập ở trên, hệ thống in ấn này sẽ đóng vai trò quan trọng đối với việc học tập trong nhà trường của các bên liên quan, vì vậy giao diện của hệ thống phải đảm bảo sự đơn giản, chính xác, cung cấp đúng, đủ những gì Client cần, liên tục cập nhập để thông tin trạng thái của từng giao dịch để Client chủ động trong việc sắp xếp công việc học tập hay dạy học của mình.

Từ những yêu cầu trên, việc cần xác định ngay bây giờ là một chiến lược cụ thể để tạo nên một giao diện hoàn thiện cho hệ thống. Sau đây là các công việc mà nhóm chúng em đề ra để đạt được những mục tiêu cụ thể quan trọng trong việc xây dựng một giao diện đơn giản nhưng đáp ứng đầy đủ yêu cầu nghiệp vụ của hệ thống, song hành với đó là sự tối ưu trong trải nghiệm người dùng.

1. Frontend Library và Framework:
    - Đối với hệ thống HCMUT-SSPS, nhóm chúng em sẽ sử dụng ReactJS kết hợp với NextJS để render phía server. Cách tiếp cận này giúp giao diện người dùng nhanh, mượt mà và phản hồi tốt, cho phép các trang tải động mà không cần tải lại toàn bộ, điều này rất quan trọng cho các ứng dụng có nhiều tương tác như SSPS. Với NextJS, hệ thống sẽ được cải thiện về SEO và tốc độ tải trang ban đầu nhờ render phía server, trong khi ReactJS hỗ trợ tăng tính tương tác và quản lý trạng thái.
    - Giao diện sẽ được chia thành các chế độ xem (views) khác nhau cho từng vai trò người dùng (ví dụ: Sinh viên, Nhân viên SPSO, Admin (nếu có)) để tạo ra trải nghiệm riêng biệt, thân thiện và dễ sử dụng cho từng đối tượng.
2. Responsive Design đảm bảo sự ổn định trên nhiều thiết bị:
    - Thiết kế responsive là một yếu tố quan trọng để đảm bảo giao diện của hệ thống HCMUT-SSPS hiển thị tốt và hoạt động hiệu quả trên nhiều loại thiết bị và kích thước màn hình, từ máy tính để bàn, laptop đến máy tính bảng và điện thoại di động. Với việc ngày càng nhiều người dùng truy cập hệ thống qua thiết bị di động, responsive giúp hệ thống trở nên linh hoạt và dễ sử dụng ở mọi hoàn cảnh.

- Thiết kế responsive cũng đảm bảo nội dung hiển thị rõ ràng và dễ đọc. Các bảng thông tin hoặc form nhập liệu sẽ được sắp xếp lại trên màn hình nhỏ để tránh cuộn ngang, giúp người dùng không phải kéo thả nhiều lần để xem hết nội dung. Với NextJS kết hợp Tailwind CSS, hệ thống sẽ tự động xác định kích thước màn hình và tải các bố cục, kiểu dáng thích hợp, đảm bảo trải nghiệm nhất quán và mượt mà trên mọi thiết bị.

3. Tối ưu Trải nghiệm Người dùng (UX)

   - Để tối ưu trải nghiệm người dùng (UX) cho hệ thống HCMUT-SSPS, nhóm chúng em sẽ tập trung vào việc cung cấp một giao diện thân thiện, dễ sử dụng và trực quan. Đầu tiên, nhóm sẽ thiết kế bố cục giao diện theo nguyên tắc người dùng làm trung tâm, với các thao tác chính được đặt ở những vị trí dễ truy cập để người dùng có thể hoàn thành tác vụ chỉ với vài cú nhấp chuột. Các biểu tượng và nút sẽ được sử dụng nhất quán để giúp người dùng nhận diện nhanh chóng các chức năng.
   - Ngoài ra, nhóm chúng em sẽ sử dụng tải dữ liệu không đồng bộ và các chỉ báo tải (loading indicators) để giảm thời gian chờ đợi và tránh cảm giác gián đoạn. Ví dụ, khi người dùng thực hiện yêu cầu in ấn hoặc kiểm tra lịch sử in, dữ liệu sẽ được tải động mà không cần tải lại trang, giúp duy trì sự liền mạch. Việc sử dụng NextJS cũng hỗ trợ render trước nội dung cần thiết từ server, giúp cải thiện tốc độ tải trang ban đầu và tăng tính mượt mà cho ứng dụng.

4. Cuối cùng, nhóm sẽ thực hiện kiểm tra người dùng và thu thập phản hồi từ sinh viên và nhân viên SPSO để liên tục cải thiện trải nghiệm. Các tính năng được sử dụng thường xuyên sẽ được tối ưu hóa để giúp người dùng hoàn thành nhiệm vụ một cách dễ dàng và nhanh chóng.

1.5 Data storage approach

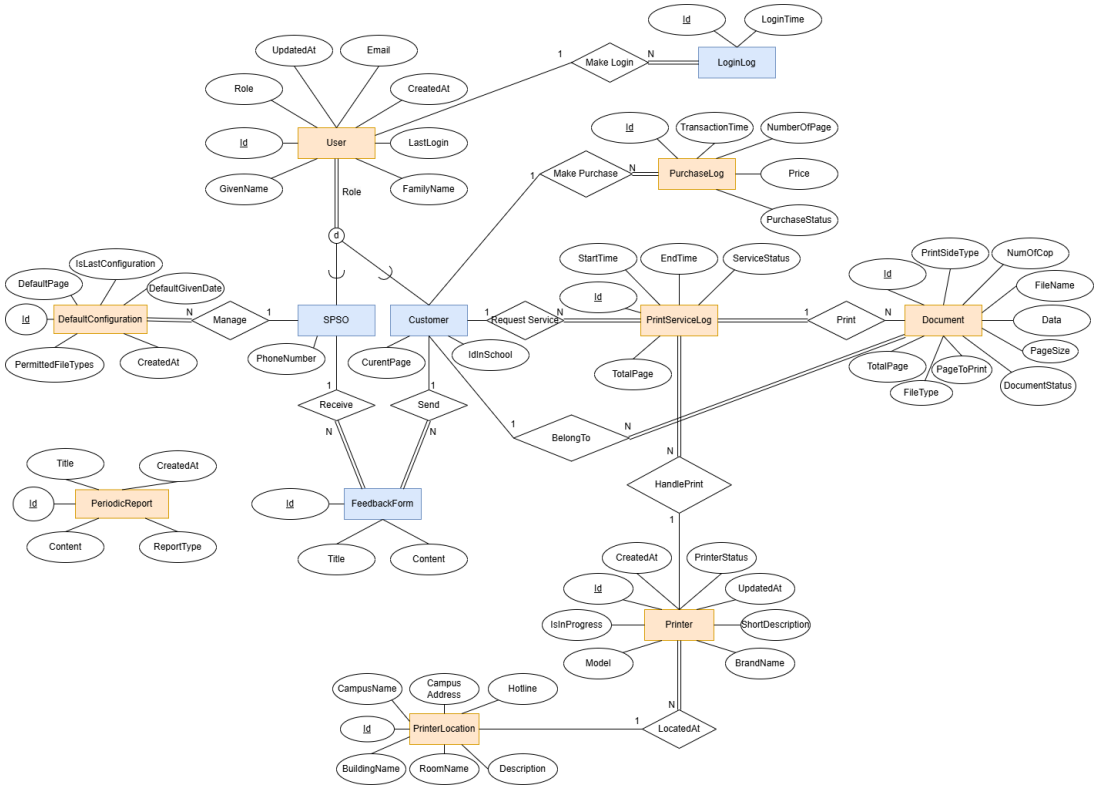
Đối với các yêu cầu của hệ thống HCMUT_SSPS, không khó để nhận ra, các hành động liên quan tới việc truy xuất, xử lý và cập nhập dữ liệu là rất quan trọng và cần sự chính xác cao. Và đó cũng chính là một trong những lí do kiến trúc phân tầng được chọn để hiện thực hệ thống, các thao tác liên quan tới 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. Nhóm chúng em sẽ sử dụng cơ sở dữ liệu quan hệ MySQL để lưu trữ dữ liệu, vì hệ thống cần quản lý các dữ liệu có cấu trúc rõ ràng như thông tin người dùng, lịch sử in ấn, cấu hình máy in, và các báo cáo định kỳ. MySQL là một lựa chọn phù hợp vì tính ổn định, hiệu suất cao, và khả năng mở rộng khi hệ thống phát triển. Ngoài ra, MySQL có hỗ trợ mạnh mẽ trong việc xử lý các truy vấn phức tạp, đảm bảo hiệu quả khi truy xuất và xử lý dữ liệu.

Cơ sở dữ liệu sẽ được thiết kế với các bảng (schema) cụ thể cho từng loại dữ liệu, như User để lưu thông tin người dùng, PrintServiceLog để ghi nhận lịch sử in, PeriodicReport cho các báo cáo định kỳ, và DefaultConfiguration để lưu các cấu hình mặc định của hệ thống,... Mỗi bảng sẽ có các mối quan hệ rõ ràng, chẳng hạn như mối quan hệ giữa người dùng và lịch sử in để dễ dàng truy vấn các bản ghi liên quan.

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.

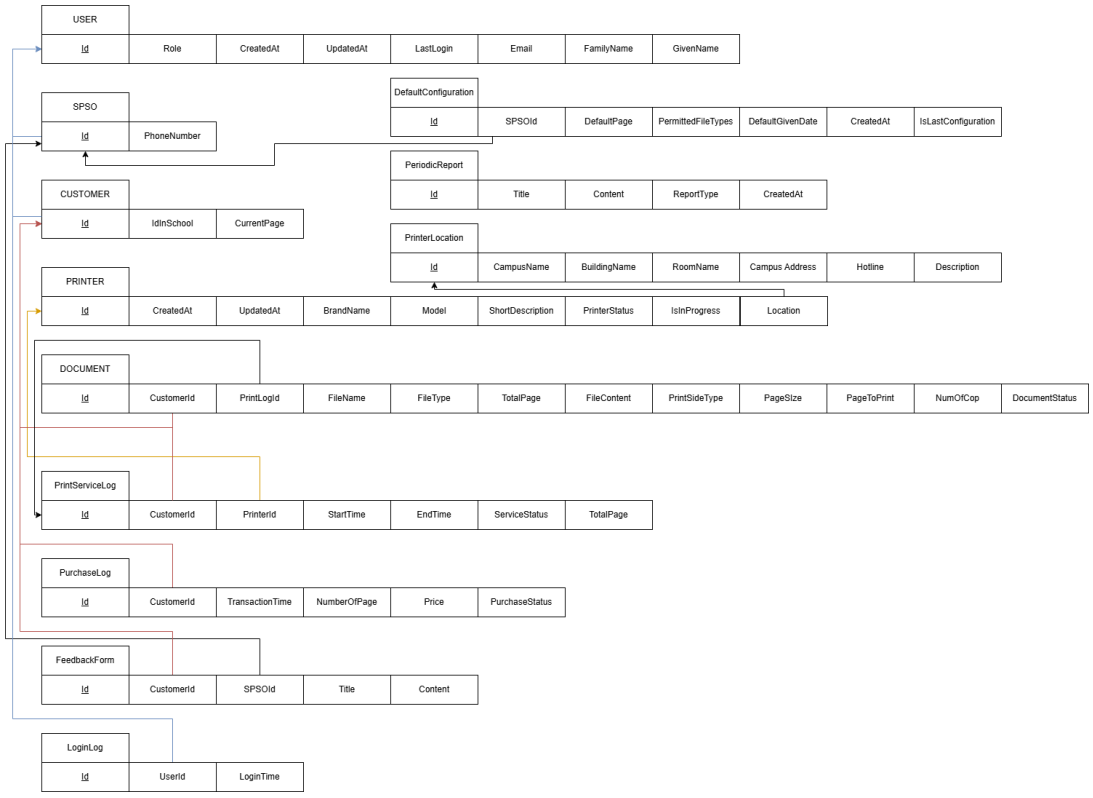
Trong phần này nhóm đã sử dụng **Draw.io** để hiện thực sơ đồ. Đường dẫn cụ thể tới sơ đồ được hiện thực nằm ở **đây**.

EERD Diagram



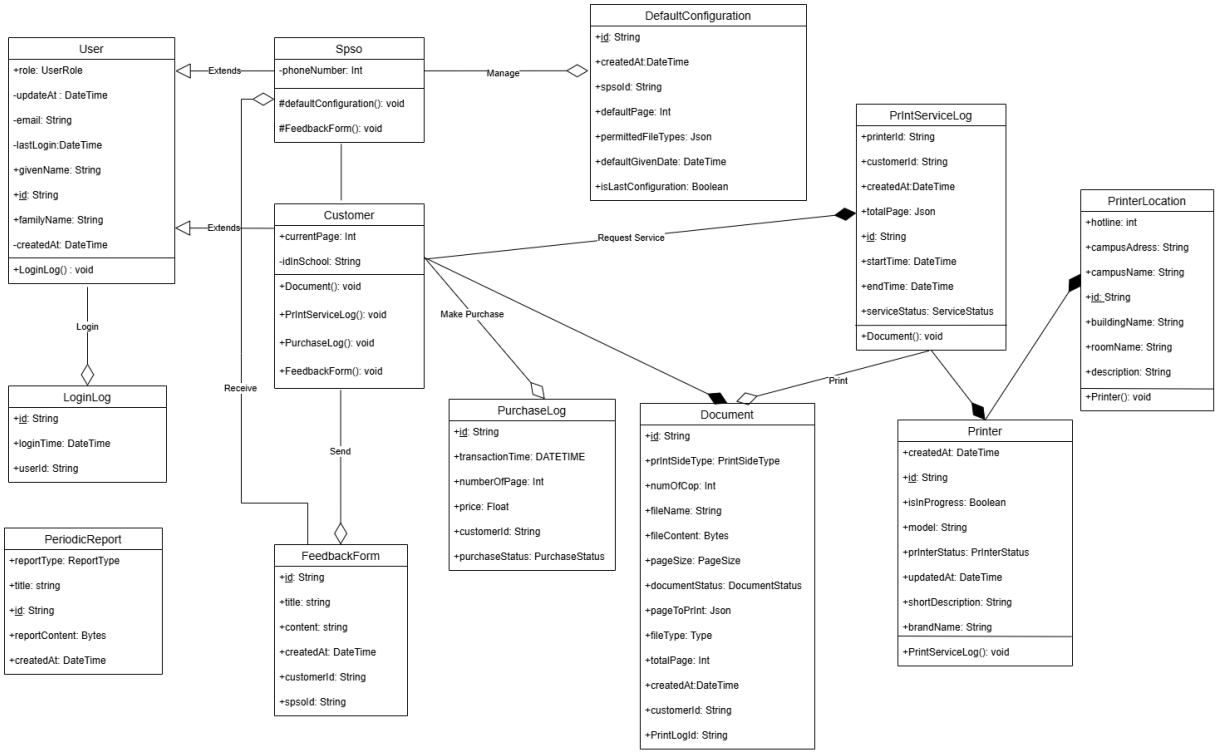
Hình 8: *EERD Diagram*

Relational Data Modal



Hình 9: *Relational Data Modal*

Class Diagram



Hình 10: *Class Diagram*

1.6 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 hoặc thanh toán), 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.