HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN 1

KIẾN TRÚC VÀ THIẾT KẾ PHẦN MỀM

Assignment 1

Họ và tên	:	Tạ Trường Vũ
Mã sinh viên	:	B22DCCN918
Giảng viên	:	Trần Đình Quế
Khóa	:	2022-2027
Hệ	:	Đại học chính quy
Chuyên ngành	:	Công nghệ phần mềm

Hà Nội, 2026

MỤC LỤC

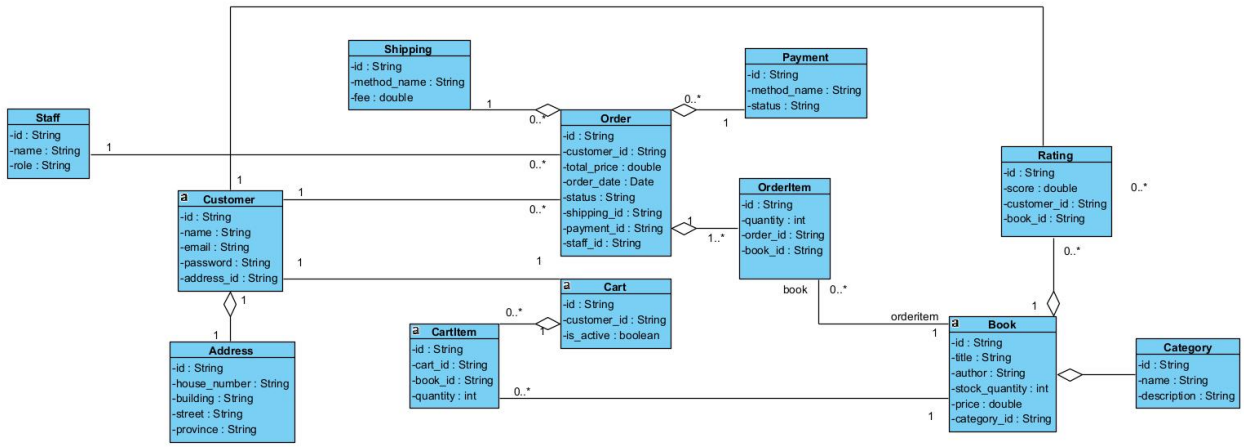
CHƯƠNG 1: THIẾT KẾ UML

1.1. Class diagram

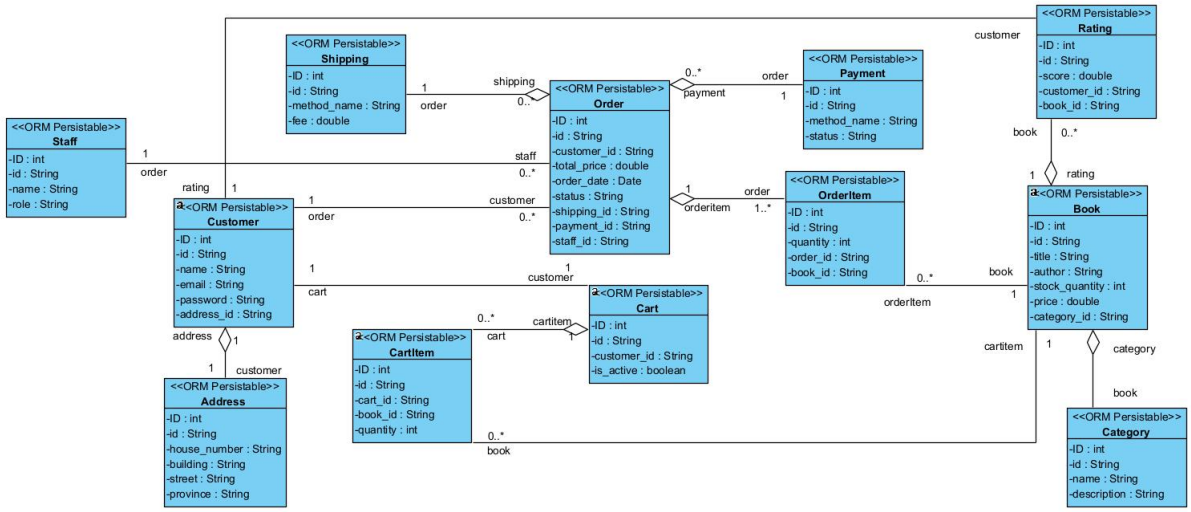
Mô tả các chức năng:

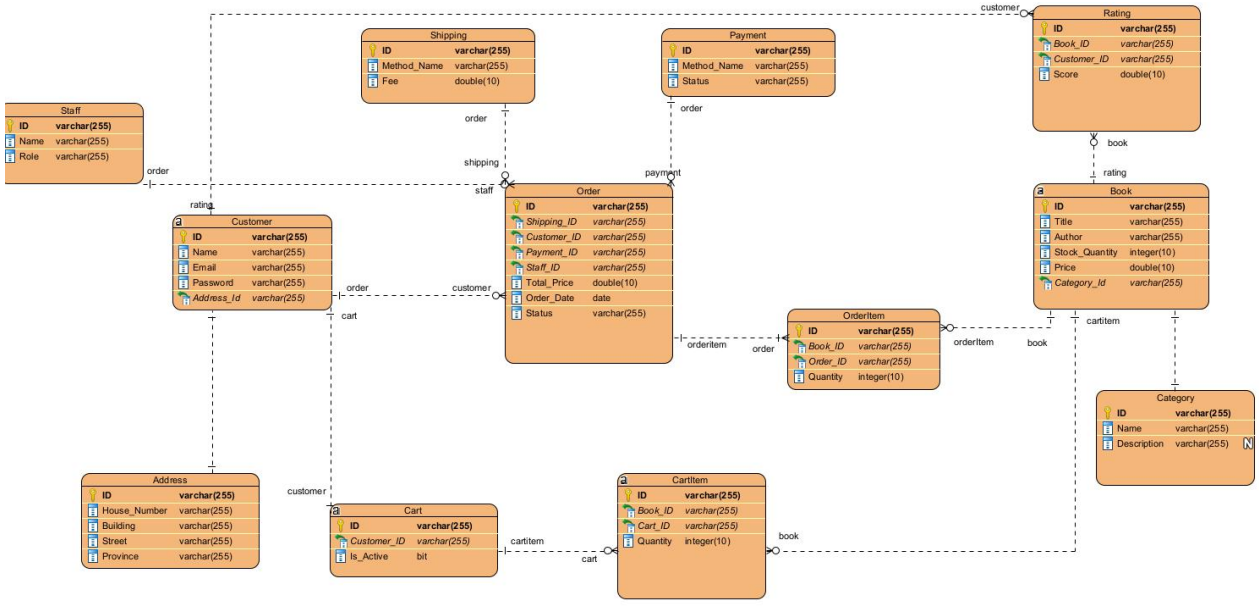
- Chức năng Đăng ký tài khoản mới: Khách truy cập chọn menu "Đăng ký" => Nhập các thông tin bắt buộc (Họ tên, Email, Mật khẩu, Số điện thoại) => Nhấn nút "Submit " => Hệ thống lưu user mới vào CSDL => Hiển thị thông báo "Đăng ký thành công" và chuyển hướng sang trang Đăng nhập.
- Chức năng Đăng nhập: Người dùng chọn menu "Đăng nhập" => Nhập Email và Mật khẩu => Nhấn nút "Đăng nhập" => Hệ thống kiểm tra thông tin đăng nhập => Nếu đúng thông tin hệ thống chuyển hướng về Trang chủ với quyền Customer.
- Chức năng xem danh mục sách: Người dùng truy cập vào trang chủ hoặc chọn menu "Danh mục sách" => Hệ thống lấy danh sách các sách (Hình ảnh, Tên sách, Tác giả, Giá bán) => Hệ thống hiển thị danh sách sách => Người dùng nhập từ khóa tìm kiếm hoặc lọc theo giá => Người dùng click vào ảnh hoặc tên của một cuốn sách cụ thể => Hệ thống hiển thị trang "Chi tiết sách" với đầy đủ thông tin (Mô tả, Tồn kho, nút Thêm vào giỏ).
- Chức năng Thêm sách vào giỏ hàng: Tại giao diện "Chi tiết sách" => Người dùng nhập số lượng sách muốn mua => Nhấn nút "Thêm vào giỏ hàng" => Hệ thống cập nhật giỏ hàng của khách hàng
- Chức năng Xem giỏ hàng: Người dùng click vào biểu tượng "Giỏ hàng" trên thanh menu => Hệ thống dựa lấy danh sách các sản phẩm trong giỏ hàng của người dùng hiển thị thông tin từng sản phẩm thông tin chi tiết (Tên, Ảnh, Giá hiện tại) => Hệ thống tính "Thành tiền" cho từng dòng (Giá x Số lượng) và tính "Tổng tiền" cho cả giỏ hàng => Hiển thị bảng chi tiết giỏ hàng ra màn hình (bao gồm nút Sửa số lượng hoặc Xóa sản phẩm) => Người dùng xem lại tổng tiền trước khi quyết định thanh toán.

Biểu đồ lớp phân tích:

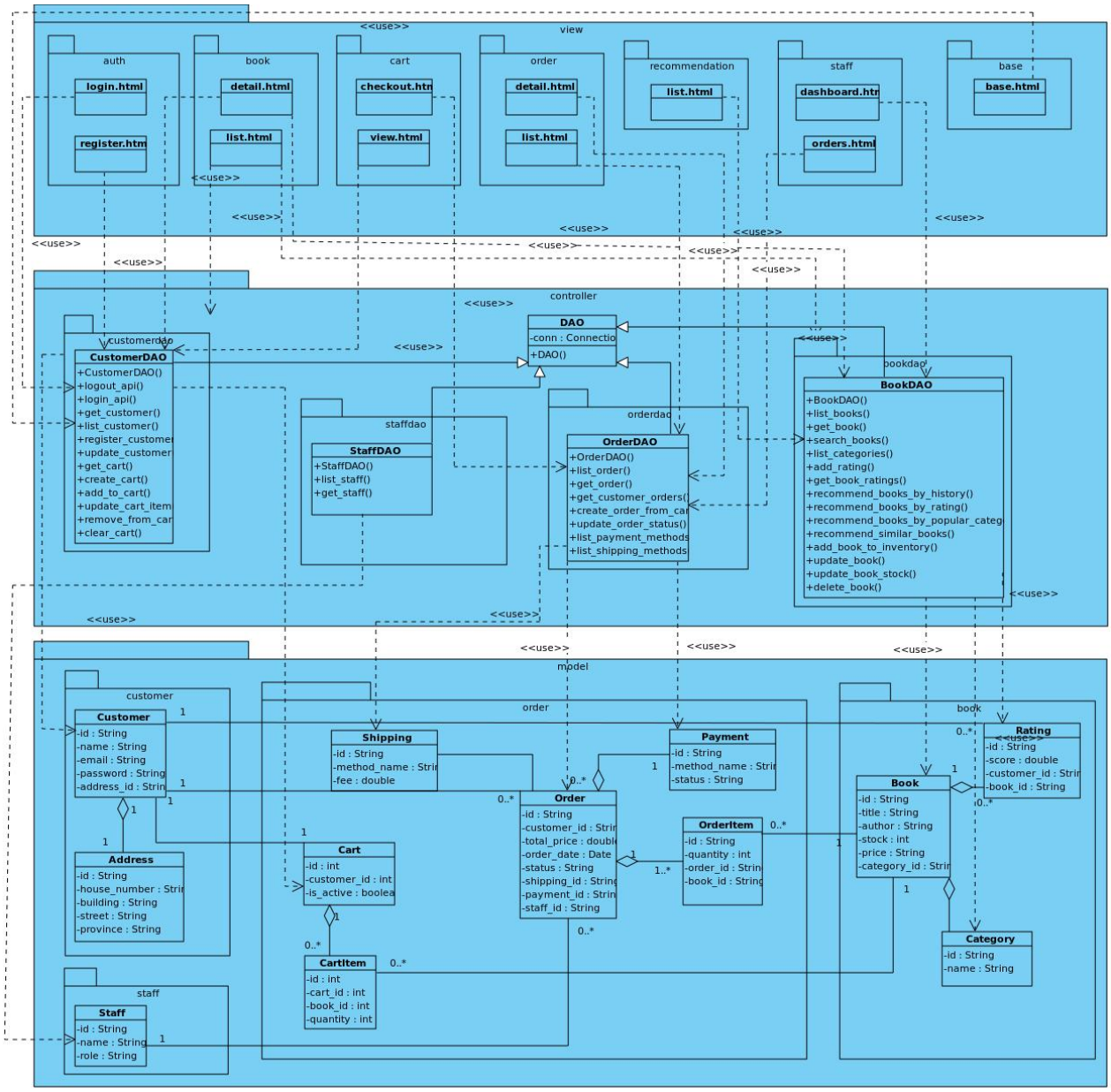


Datamodel:

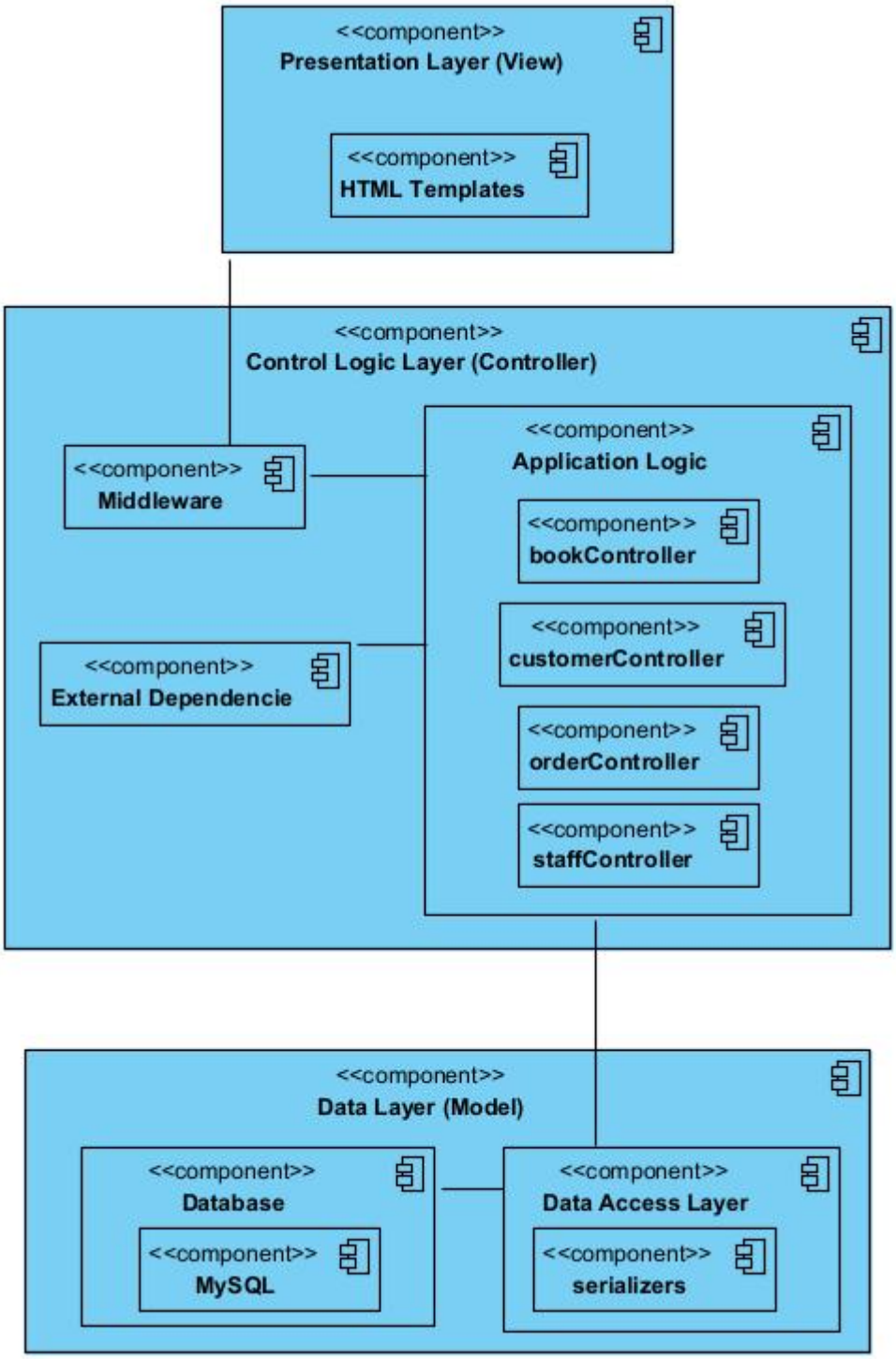




Biểu đồ lớp thiết kế:



1.2. MVC Layer Diagram



Presentation Layer (Tầng Giao diện - View) Đây là tầng tương tác trực tiếp với người dùng cuối.

HTML Templates: Đảm nhận vai trò hiển thị dữ liệu và giao diện người dùng. Thành phần này nhận dữ liệu đã được xử lý từ tầng Controller và render thành các trang web động để phản hồi lại trình duyệt của người dùng.

Control Logic Layer (Tầng Điều khiển logic - Controller) Đóng vai trò trung tâm trong việc tiếp nhận yêu cầu, xử lý nghiệp vụ và điều phối luồng dữ liệu của hệ thống.

Middleware: Hoạt động như lớp trung gian kiểm soát các yêu cầu đầu vào (Requests). Thành phần này thực hiện các tác vụ cắt ngang (cross-cutting concerns) như xác thực người dùng, bảo mật, và xử lý phiên trước khi yêu cầu được chuyển đến logic ứng dụng.

Application Logic: Chứa các quy tắc nghiệp vụ cốt lõi. Logic được phân chia thành các Controller chuyên biệt nhằm đảm bảo tính mô-đun hóa:

BookController: Quản lý các nghiệp vụ liên quan đến danh mục sách và tìm kiếm.

CustomerController: Xử lý đăng ký, đăng nhập và thông tin khách hàng.

OrderController: Xử lý quy trình giỏ hàng và tạo đơn hàng.

StaffController: Quản lý các tác vụ quản trị hệ thống.

External Dependencies: Quản lý việc tích hợp và giao tiếp với các thư viện hoặc dịch vụ bên thứ ba, tách biệt sự phụ thuộc khỏi logic nghiệp vụ chính.

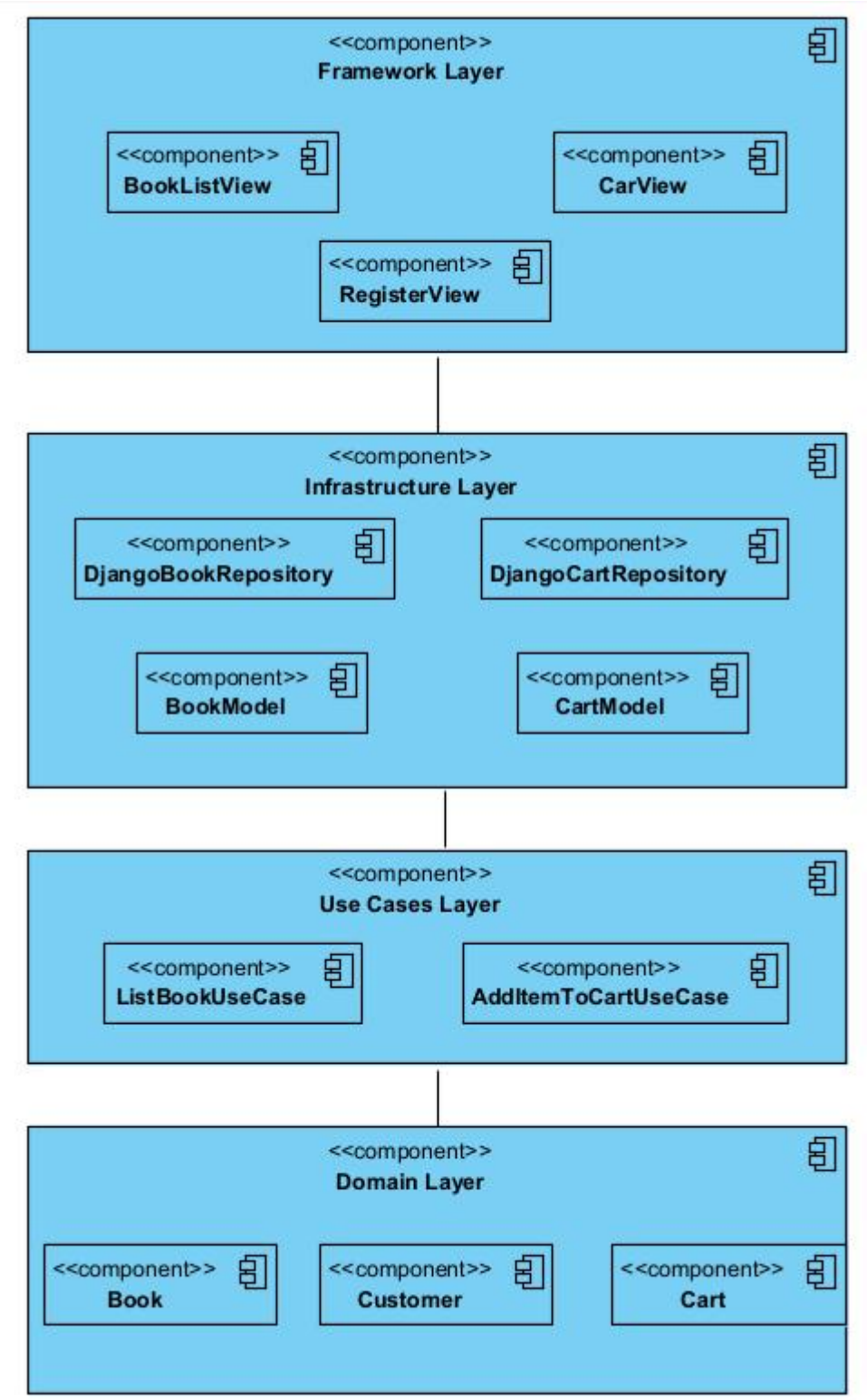
Data Layer (Tầng Dữ liệu - Model) Tầng thấp nhất chịu trách nhiệm về tính bền vững và toàn vẹn của dữ liệu.

Data Access Layer & Serializers: Cung cấp lớp trừu tượng để truy cập cơ sở dữ liệu. Thành phần Serializers đặc biệt quan trọng trong việc chuyển đổi dữ liệu phức tạp từ cơ sở dữ liệu sang các định dạng tiêu chuẩn (như JSON hoặc Python Objects) để tầng Controller có thể sử dụng dễ dàng.

Database (MySQL): Hệ quản trị cơ sở dữ liệu quan hệ (RDBMS) dùng để lưu trữ vật lý toàn bộ dữ liệu của hệ thống bao gồm thông tin sách, khách hàng và đơn hàng.

Luồng xử lý dữ liệu (Data Flow): Yêu cầu từ người dùng sẽ đi qua Presentation Layer, được kiểm tra tại Middleware, sau đó được định tuyến đến Controller phù hợp. Controller sẽ tương tác với Data Access Layer để truy xuất dữ liệu từ MySQL, sau đó kết quả được trả ngược lại quy trình để hiển thị lên HTML Templates.

1.3. Clean Architecture Diagram



1. Framework Layer (Tầng Framework & Drivers) Đây là tầng ngoài cùng, nơi chứa các chi tiết kỹ thuật cụ thể và cơ chế phân phối của ứng dụng.

Các thành phần: Bao gồm các View cụ thể như BookListView, CarView, và RegisterView.

Nhiệm vụ: Tầng này chỉ chịu trách nhiệm nhận yêu cầu HTTP từ người dùng, chuyển đổi dữ liệu và gọi đến các Use Case bên trong. Nó phụ thuộc hoàn toàn vào các công cụ như Django Web Framework để hiển thị giao diện người dùng.

2. Infrastructure Layer (Tầng Hạ tầng & Interface Adapters) Tầng này đóng vai trò cầu nối (Adapter), chuyển đổi dữ liệu giữa các định dạng thuận tiện cho Use Case và Entity sang định dạng thuận tiện cho Cơ sở dữ liệu hoặc Framework.

RepositoriesImplementation:Bao gồm DjangoBookRepository và DjangoCartRepository. Các thành phần này thực thi các giao diện được định nghĩa bởi các tầng bên trong, sử dụng ORM để truy xuất dữ liệu thực tế.

Data Models: Chứa BookModel và CartModel. Đây là các mô hình dữ liệu gắn liền với Database (kế thừa từ Django Model), khác biệt hoàn toàn với các Entity nghiệp vụ thuần túy ở tầng Domain.

3. Use Cases Layer (Tầng Nghiệp vụ Ứng dụng) Đây là tầng chứa các quy tắc nghiệp vụ đặc thù của ứng dụng (Application Business Rules), điều phối luồng dữ liệu đến và đi từ các thực thể.

Các thành phần:

ListBookUseCase: Đóng gói logic nghiệp vụ để lấy và lọc danh sách sản phẩm.

AddItemToCartUseCase: Xử lý logic nghiệp vụ thêm sản phẩm vào giỏ, bao gồm việc kiểm tra tồn kho và cập nhật số lượng.

Đặc điểm: Tầng này hoàn toàn không biết về sự tồn tại của Database hay Web Framework (Django View). Nó chỉ thao tác với các Entity và các Interface trừu tượng.

4. Domain Layer (Tầng Thực thể - Enterprise Business Rules) Nằm ở vị trí trung tâm (trong cùng), đây là tầng quan trọng nhất chứa các quy tắc nghiệp vụ cốt lõi, bất biến của hệ thống.

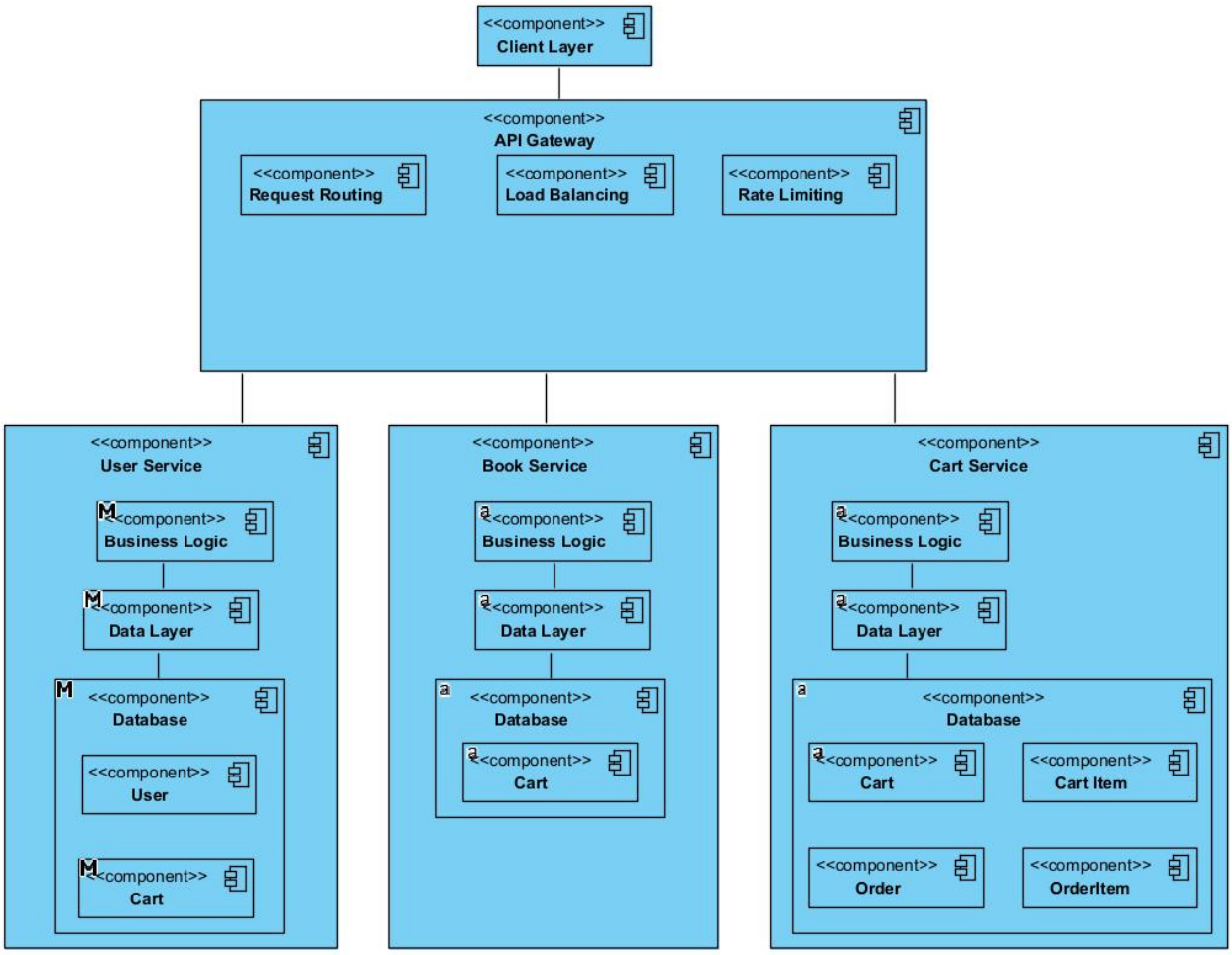
Các thực thể (Entities): Bao gồm Book, Customer, và Cart.

Đặc điểm: Các thực thể này là các đối tượng Python thuần, chứa logic nghiệp vụ chung nhất (ví dụ: quy tắc tính giá, quy tắc validate dữ liệu cơ bản). Tầng này độc lập hoàn toàn, không phụ thuộc vào bất kỳ tầng nào khác bên ngoài, đảm bảo tính ổn định cao cho hệ thống khi có sự thay đổi về công nghệ.

Luồng điều khiển và Quy tắc phụ thuộc: Trong thiết kế này, các mũi tên phụ thuộc (nếu được thể hiện logic) sẽ luôn hướng từ các tầng trên (Framework/Infrastructure) xuống các

tầng dưới (Use Cases/Domain). Điều này đảm bảo rằng việc thay đổi giao diện người dùng (Framework Layer) hoặc thay đổi cơ sở dữ liệu (Infrastructure Layer) sẽ không ảnh hưởng đến logic nghiệp vụ cốt lõi nằm tại Domain Layer.

1.4. Microservices Diagram



1. Client Layer (Tầng Khách hàng) Đây là điểm khởi đầu của mọi tương tác. Các ứng dụng khách (Web App, Mobile App) không giao tiếp trực tiếp với từng dịch vụ con mà gửi yêu cầu đến một điểm truy cập duy nhất là API Gateway.

2. API Gateway (Cổng giao tiếp API) Đóng vai trò là "người gác cổng" và điều phối trung tâm của hệ thống. Dựa trên sơ đồ thiết kế, API Gateway đảm nhận 3 nhiệm vụ cốt lõi:

- Request Routing (Định tuyến yêu cầu): Tiếp nhận yêu cầu từ Client Layer và định tuyến chính xác đến Service phù hợp (ví dụ: yêu cầu tìm sách chuyển về Book Service, yêu cầu đăng nhập chuyển về User Service).
- Load Balancing (Cân bằng tải): Phân phối lưu lượng truy cập đều giữa các bản sao (instances) của các dịch vụ, giúp hệ thống không bị quá tải cục bộ.

- Rate Limiting (Giới hạn tốc độ): Kiểm soát số lượng yêu cầu đến hệ thống trong một khoảng thời gian nhất định để ngăn chặn các cuộc tấn công DDoS hoặc lạm dụng API.

3. Services Layer (Tầng Dịch vụ) Hệ thống được chia thành 3 dịch vụ vi mô độc lập (Microservices), mỗi dịch vụ là một đơn vị triển khai riêng biệt bao gồm đầy đủ các tầng từ logic đến dữ liệu:

- User Service:
    - Chịu trách nhiệm quản lý định danh, xác thực và hồ sơ người dùng.
    - Cấu trúc bao gồm Business Logic xử lý nghiệp vụ người dùng, Data Layer truy xuất dữ liệu, và sở hữu một Database riêng biệt chứa bảng User.
- Book Service:
    - Chịu trách nhiệm quản lý danh mục sản phẩm (Catalog).
    - Tương tự như User Service, dịch vụ này có Business Logic và Data Layer riêng. Quan trọng nhất, nó sở hữu cơ sở dữ liệu riêng (Database per Service) để đảm bảo tính độc lập về dữ liệu.
- Cart Service:
    - Chịu trách nhiệm về quy trình mua sắm, bao gồm quản lý giỏ hàng (Cart, Cart Item) và đơn hàng (Order, OrderItem).
    - Đây là dịch vụ phức tạp nhất với cấu trúc dữ liệu quan hệ chặt chẽ giữa các bảng Cart và Order, được gói gọn hoàn toàn bên trong Database của dịch vụ này.

Đặc điểm Kiến trúc (Architectural Characteristics):

- Database per Service (Mỗi dịch vụ một Cơ sở dữ liệu): Như thể hiện trong sơ đồ, mỗi Service (User, Book, Cart) đều có một khối Database nằm bên trong ranh giới của nó. Điều này đảm bảo tính "Decoupling" (Tách rời) - dịch vụ này không thể truy cập trực tiếp vào dữ liệu của dịch vụ kia mà phải thông qua API, giúp loại bỏ các điểm nghẽn (bottlenecks) thường thấy trong kiến trúc Monolithic.
- Vertical Isolation (Cô lập theo chiều dọc): Mỗi dịch vụ hoạt động như một ứng dụng thu nhỏ với đầy đủ các tầng Business Logic -> Data Layer -> Database, cho phép các nhóm phát triển có thể làm việc, triển khai và nâng cấp công nghệ cho từng dịch vụ mà không ảnh hưởng đến toàn bộ hệ thống.

CHƯƠNG 2: TRIỂN KHAI CODE

2.1. Monolithic

https://github.com/ttvKieran/KienTrucHeThong

2.2. Clean Architecture

https://github.com/ttvKieran/KienTrucHeThong

2.3. Microservices

https://github.com/ttvKieran/KienTrucHeThong

KẾT LUẬN

Tổng kết về các phương án kiến trúc đã thiết kế:

1. Version A: Monolithic Architecture (MVC Pattern) Đây là giải pháp tiếp cận truyền thống và hiệu quả nhất cho giai đoạn khởi tạo dự án (MVP). Việc gom nhóm toàn bộ Presentation, Business Logic và Data Access vào một Single Deployable Unit giúp đơn giản hóa quy trình phát triển, testing và deployment. Tuy nhiên, thiết kế này cũng bộc lộ hạn chế về khả năng mở rộng và rủi ro tạo ra các thành phần phụ thuộc chặt chẽ khi hệ thống phát triển lớn.
2. Version B: Clean Architecture Việc áp dụng Clean Architecture đã giải quyết triệt để vấn đề phụ thuộc của phiên bản A. Bằng cách tuân thủ Dependency Rule và phân tách rõ ràng giữa Domain Entities, Use Cases và Infrastructure, hệ thống đạt được tính ổn định cao và khả năng kiểm thử vượt trội. Mặc dù độ phức tạp của mã nguồn tăng lên do số lượng file và interface nhiều hơn, nhưng đây là sự đánh đổi xứng đáng để đảm bảo tính Maintainability về lâu dài.
3. Version C: Microservices Architecture Bản thiết kế này minh họa một hệ thống phân tán hiện đại, nơi các dịch vụ User, Book, và Cart hoạt động độc lập với cơ sở dữ liệu riêng biệt (Database per Service). Kiến trúc này tối ưu hóa khả năng Scale Cube (đặc biệt là X-axis và Y-axis scaling) và cho phép các team phát triển làm việc song song với các công nghệ khác nhau. Tuy nhiên, nó cũng đặt ra thách thức lớn về quản lý tính nhất quán dữ liệu và độ phức tạp trong vận hành hạ tầng.

Đề xuất và Định hướng phát triển:

Dựa trên phân tích yêu cầu hiện tại của Book Store Web System, em nhận định rằng Clean Architecture (Version B) là sự cân bằng tốt nhất giữa chi phí phát triển và khả năng bảo trì. Nó đủ linh hoạt để phát triển tính năng mới mà không bị nợ kỹ thuật như Monolithic, đồng thời tránh được sự phức tạp quá mức của Microservices trong giai đoạn đầu.

Về lộ trình tương lai, khi lượng traffic và dữ liệu tăng trưởng, hệ thống được thiết kế theo Clean Architecture có thể dễ dàng tách các module (như Cart hoặc Order) thành các Microservices độc lập theo chiến lược Strangler Fig Pattern, nhờ vào sự tách biệt rạch ròi của các Use Case và Interface đã được thiết kế.