0.1 Thiết kế kiến trúc

0.1.1 Lựa chọn kiến trúc phần mềm

Dựa trên các yêu cầu nghiệp vụ phức tạp của hệ thống YummyZoom, đặc biệt là tính năng đặt hàng nhóm (TeamCart) với yêu cầu về tính nhất quán dữ liệu cao và khả năng xử lý đồng thời, đồ án lựa chọn áp dụng kiến trúc **Clean Architecture** (Kiến trúc Sạch) kết hợp với tư duy thiết kế **Domain-Driven Design (DDD)**. Về mặt triển khai, hệ thống được xây dựng theo mô hình **Monolithic Modular** (Đơn khối module hóa).

a, Mô hình kiến trúc tổng thể

Thay vì lựa chọn kiến trúc Microservices ngay từ đầu - vốn đòi hỏi chi phí vận hành và quản lý hạ tầng lớn, hay kiến trúc Layered (3 lớp) truyền thống dễ gây ra sự phụ thuộc chặt chẽ vào cơ sở dữ liệu, nhóm phát triển quyết định sử dụng mô hình Monolithic Modular.

Mô hình này giúp cân bằng giữa tốc độ phát triển và khả năng mở rộng:

- **Phát triển nhanh và nhất quán:** Toàn bộ mã nguồn nằm trong một Solution thống nhất, giúp việc gỡ lỗi (debug), kiểm thử và tái cấu trúc (refactor) trở nên dễ dàng hơn.
- **Ranh giới nghiệp vụ rõ ràng (Modular Boundaries):** Các module được tổ chức độc lập về mặt logic dựa trên các Bounded Contexts. Điều này tạo tiền đề vững chắc để tách thành các Microservices riêng biệt khi hệ thống cần mở rộng quy mô trong tương lai mà không làm phá vỡ cấu trúc hiện tại.
- **Tập trung vào nghiệp vụ cốt lõi:** Clean Architecture giúp cô lập logic nghiệp vụ (Domain) khỏi các yếu tố thay đổi thường xuyên như giao diện người dùng (UI) hay công nghệ lưu trữ (Database).

b, Tổ chức các tầng kiến trúc (Layered Architecture)

Hệ thống tuân thủ nghiêm ngặt quy tắc phụ thuộc (Dependency Rule) của Clean Architecture: "Mọi sự phụ thuộc của mã nguồn chỉ được hướng vào bên trong". Cấu trúc dự án được ánh xạ từ lý thuyết vào thực tế như sau:

1. **Tầng Miền (Domain Layer) -** ***Project: YummyZoom.Domain***
   Đây là lõi trung tâm của hệ thống, nơi chứa các quy tắc nghiệp vụ bất biến của doanh nghiệp. Tầng này hoàn toàn không phụ thuộc vào bất kỳ thư viện bên ngoài hay công nghệ cơ sở dữ liệu nào.
   - **Thành phần:** Entities (Thực thể), Value Objects (Đối tượng giá trị), Aggregates (Hợp nhất), Domain Events.

- **Ví dụ thực tế:** Aggregate `TeamCart` đóng gói logic nghiệp vụ của giỏ hàng nhóm, đảm bảo các quy tắc như: một giỏ hàng chỉ có một chủ phòng (Host), và chỉ Host mới có quyền chốt đơn hoặc khóa giỏ hàng.

2. **Tầng Ứng dụng (Application Layer) - *Project: YummyZoom.Application***
   Đóng vai trò lớp vỏ bao quanh Domain, điều phối các yêu cầu từ người dùng và chuyển đổi chúng thành các thao tác nghiệp vụ.
   - **Thành phần:** Use Cases (được triển khai dưới dạng Command/Query Handlers), Validators (Kiểm tra dữ liệu), Interfaces cho Infrastructure.
   - **Ví dụ thực tế:** `AddItemToTeamCartCommand` là một Use Case nhận yêu cầu thêm món, kích hoạt phương thức `AddItem` trong Domain Entity, và sau đó lưu thay đổi thông qua `ITeamCartRepository`.
3. **Tầng Hạ tầng (Infrastructure Layer) - *Project: YummyZoom.Infrastructure***
   Nơi chưa các thực thi của các interfaces kỹ thuật được định nghĩa ở tầng Application. Đây là nơi các công nghệ cụ thể được "cắm" vào hệ thống.
   - **Thành phần:** DbContext (Entity Framework Core), Repositories Implementation, Services giao tiếp bên ngoài (EmailService, CloudinaryService, StripeService, FCMService).
   - **Chức năng:** Truy xuất dữ liệu từ SQL Server/PostgreSQL, tương tác với hệ thống file, gửi email xác nhận, xử lý thanh toán qua Stripe, gửi thông báo đẩy với FCM.
4. **Tầng Giao diện/Trình bày (Web/Presentation Layer) - *Project: YummyZoom.Web***
   Lớp vỏ ngoài cùng, chịu trách nhiệm giao tiếp với Client (Mobile App, Web Admin).
   - **Thành phần:** Minimal APIs, SignalR Hubs, Middlewares.
   - **Nhiệm vụ:** Nhận HTTP Request, xác thực token, gọi xuống Application Layer để xử lý và trả về HTTP Response chuẩn (JSON).

c, Thiết kế chiến lược (Strategic Design)

Áp dụng chiến lược của DDD, hệ thống được chia nhỏ thành các **Bounded Contexts** (Ngữ cảnh giới hạn), mỗi context giải quyết một vấn đề nghiệp vụ cụ thể và có Ubiquitous Language (Ngôn ngữ chung) riêng:

- **Identity Context:** Quản lý người dùng, phân quyền và xác thực (Authentication/Authorization).
- **Catalog Context:** Quản lý thực đơn, danh mục, thông tin nhà hàng, món ăn

và các tùy chọn (Toppings).

- **TeamCart Context:** Ngữ cảnh quan trọng và phức tạp nhất, xử lý logic chia sẻ giỏ hàng, đồng bộ trạng thái thời gian thực và phân chia hóa đơn.
- **Ordering Context:** Xử lý quy trình đặt hàng, thanh toán và vòng đời của đơn hàng sau khi được chốt.

d, Các mẫu kỹ thuật chủ đạo (Tactical Patterns)

Để tối ưu hóa hiệu năng và khả năng bảo trì, hệ thống áp dụng các mẫu thiết kế kỹ thuật nâng cao:

CQRS (Command Query Responsibility Segregation): Hệ thống tách biệt rõ ràng luồng Đọc (Query) và Ghi (Command) dữ liệu:

- **Phần Ghi (Write Side):** Sử dụng **Entity Framework Core** kết hợp với Domain Model để đảm bảo tính toàn vẹn và nhất quán dữ liệu (Data Consistency) khi thực hiện các thay đổi nghiệp vụ.
- **Phần Đọc (Read Side):** Sử dụng **Dapper** với SQL để truy vấn dữ liệu trực tiếp và trả về các DTO phẳng. Đồng thời, hệ thống kết hợp sử dụng các **Read Models** được cập nhật song song với các bảng ghi, giúp tránh việc thực hiện các phép JOIN bảng quá mức. Cách tiếp cận này đảm bảo khả năng phục vụ hiệu quả các yêu cầu dữ liệu phức tạp từ ứng dụng người dùng với tốc độ phản hồi tối ưu.

Domain Events & Outbox Pattern: Giải quyết bài toán về tính nhất quán cuối cùng (Eventual Consistency) và xử lý bất đồng bộ:

- **Cơ chế:** Khi một nghiệp vụ quan trọng hoàn tất (ví dụ: `OrderCreated`), hệ thống không gọi trực tiếp các dịch vụ bên thứ ba (như gửi email) mà sinh ra một `Domain Event`. Event này được lưu vào bảng `OutboxMessages` trong cùng một transaction database với dữ liệu chính.
- **Tác dụng:** Một tiến trình nền (Background Worker) sẽ đọc bảng Outbox và thực hiện các tác vụ phụ sau đó. Điều này đảm bảo rằng giao dịch chính luôn nhanh và an toàn; nếu việc gửi email thất bại, nó sẽ được thử lại (Retry) mà không làm mất đơn hàng.

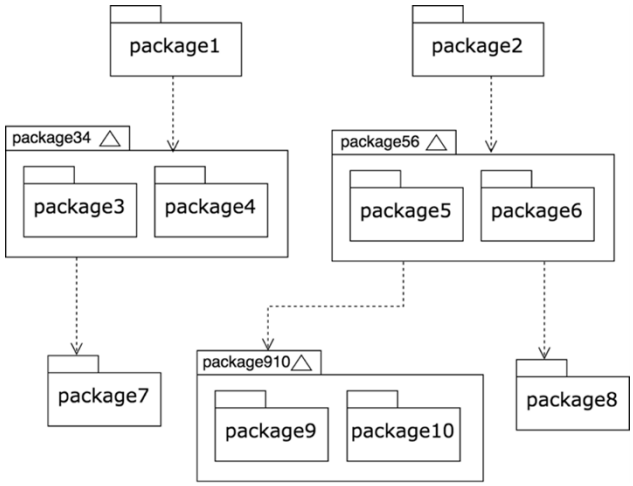
e, Kết luận

Việc lựa chọn kiến trúc Clean Architecture kết hợp DDD và CQRS mang lại nền tảng vững chắc cho hệ thống YummyZoom. Kiến trúc này không chỉ giải quyết tốt các bài toán nghiệp vụ phức tạp hiện tại về đặt hàng nhóm mà còn đảm bảo

các tiêu chí phi chức năng quan trọng: dễ dàng kiểm thử (Testability), dễ bảo trì (Maintainability) và sẵn sàng mở rộng (Scalability) trong tương lai.

0.1.2 Thiết kế tổng quan

Sinh viên vẽ biểu đồ gói UML (UML package diagram), nêu rõ sự phụ thuộc giữa các gói (package). SV cần vẽ các gói sao cho chúng được phân theo các tầng rõ ràng, không được sắp đặt package lộn xộn trong hình vẽ. Sinh viên chú ý các quy tắc thiết kế (Các gói không phụ thuộc lẫn nhau, gói tầng dưới không phụ thuộc gói tầng trên, không phụ thuộc bỏ qua tầng, v.v.) và cần giải thích sơ lược về mục đích/nhiệm vụ của từng package. SV tham khảo ví dụ minh họa trong Hình 0.1



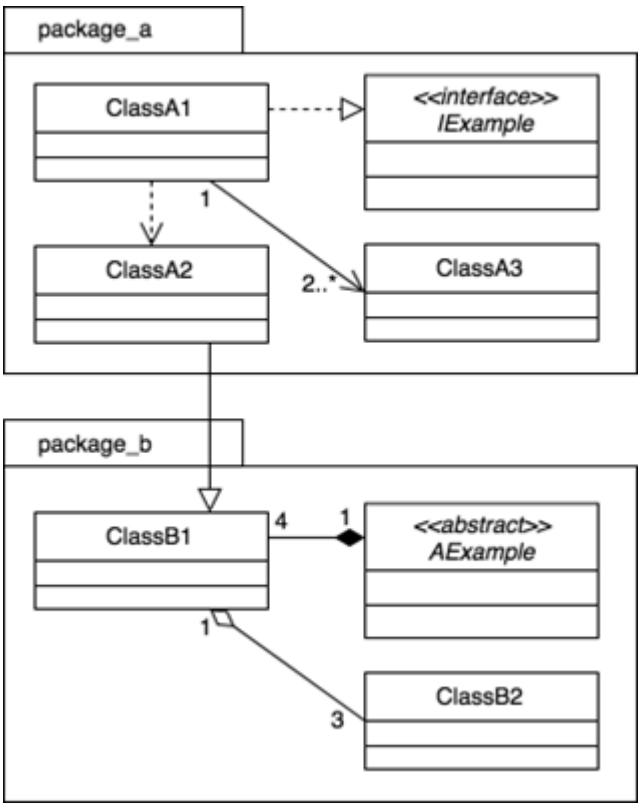
Hình 0.1: Ví dụ biểu đồ phụ thuộc gói

0.1.3 Thiết kế chi tiết gói

Sinh viên thiết kế và lần lượt vẽ biểu đồ thiết kế cho từng package, hoặc một nhóm các package liên quan để giải quyết một vấn đề gì đó. Khi vẽ thiết kế gói, sinh viên chỉ cần đưa tên lớp, không cần chỉ ra các thành viên phương thức và thuộc tính. SV tham khảo ví dụ minh họa trong Hình 0.2.

Sinh viên cần vẽ rõ ràng quan hệ giữa các lớp trong biểu đồ. Các quan hệ bao gồm: phụ thuộc (dependency), kết hợp (association), kết tập (aggregation), hợp thành (composition), kế thừa (inheritance), và thực thi (implementation). Các quan hệ này đều đã được minh họa trong 0.2.

Sau khi vẽ hình minh họa, sinh viên cần giải thích ngắn gọn về thiết kế của mình.



Hình 0.2: Ví dụ thiết kế gói

0.2 Thiết kế chi tiết

0.2.1 Thiết kế giao diện

Phần này có độ dài từ hai đến ba trang. Sinh viên đặc tả thông tin về màn hình mà ứng dụng của mình hướng tới, bao gồm độ phân giải màn hình, kích thước màn hình, số lượng màu sắc hỗ trợ, v.v. Tiếp đến, sinh viên đưa ra các thống nhất/chuẩn hóa của mình khi thiết kế giao diện như thiết kế nút, điều khiển, vị trí hiển thị thông điệp phản hồi, phối màu, v.v. Sau cùng sinh viên đưa ra một số hình ảnh minh họa thiết kế giao diện cho các chức năng quan trọng nhất. Lưu ý, sinh viên không nhầm lẫn giao diện thiết kế với giao diện của sản phẩm sau cùng.

0.2.2 Thiết kế lớp

Phần này có độ dài từ ba đến bốn trang. Sinh viên trình bày thiết kế chi tiết các thuộc tính và phương thức cho một số lớp chủ đạo/quan trọng nhất của ứng dụng (từ 2-4 lớp). Thiết kế chi tiết cho các lớp khác, nếu muốn trình bày, sinh viên đưa vào phần phụ lục.

Để minh họa thiết kế lớp, sinh viên thiết kế luồng truyền thông điệp giữa các đối tượng tham gia cho 2 đến 3 use case quan trọng nào đó bằng biểu đồ trình tự (hoặc biểu đồ giao tiếp).

0.2.3 Thiết kế cơ sở dữ liệu

Phần này có độ dài từ hai đến bốn trang. Sinh viên thiết kế, vẽ và giải thích biểu đồ thực thể liên kết (E-R diagram). Từ đó, sinh viên thiết kế cơ sở dữ liệu tùy theo hệ quản trị cơ sở dữ liệu mà mình sử dụng (SQL, NoSQL, Firebase, v.v.)

0.3 Xây dựng ứng dụng

0.3.1 Thư viện và công cụ sử dụng

Sinh viên liệt kê các công cụ, ngôn ngữ lập trình, API, thư viện, IDE, công cụ kiểm thử, v.v. mà mình sử dụng để phát triển ứng dụng. Mỗi công cụ phải được chỉ rõ phiên bản sử dụng. SV nên kẻ bảng mô tả tương tự như Bảng 0.1. Nếu có nhiều nội dung trình bày, sinh viên cần xoay ngang bảng.

Mục đích	**Công cụ**	**Địa chỉ URL**
IDE lập trình	Eclipse Oxygen a64 bit	http://www.eclipse.org/
v.v.	v.v.	v.v.

Bảng 0.1: Danh sách thư viện và công cụ sử dụng

0.3.2 Kết quả đạt được

Sinh viên trước tiên mô tả kết quả đạt được của mình là gì, ví dụ như các sản phẩm được đóng gói là gì, bao gồm những thành phần nào, ý nghĩa, vai trò?

Sinh viên cần thống kê các thông tin về ứng dụng của mình như: số dòng code, số lớp, số gói, dung lượng toàn bộ mã nguồn, dung lượng của từng sản phẩm đóng gói, v.v. Tương tự như phần liệt kê về công cụ sử dụng, sinh viên cũng nên dùng bảng để mô tả phần thông tin thống kê này.

0.3.3 Minh họa các chức năng chính

Sinh viên lựa chọn và đưa ra màn hình cho các chức năng chính, quan trọng, và thú vị nhất. Mỗi giao diện cần phải có lời giải thích ngắn gọn. Khi giải thích, sinh viên có thể kết hợp với các chú thích ở trong hình ảnh giao diện.

0.4 Kiểm thử

Phần này có độ dài từ hai đến ba trang. Sinh viên thiết kế các trường hợp kiểm thử cho hai đến ba chức năng quan trọng nhất. Sinh viên cần chỉ rõ các kỹ thuật kiểm thử đã sử dụng. Chi tiết các trường hợp kiểm thử khác, nếu muốn trình bày, sinh viên đưa vào phần phụ lục. Sinh viên sau cùng tổng kết về số lượng các trường hợp kiểm thử và kết quả kiểm thử. Sinh viên cần phân tích lý do nếu kết quả kiểm thử không đạt.

0.5 Triển khai

Sinh viên trình bày mô hình và/hoặc cách thức triển khai thử nghiệm/thực tế. Ứng dụng của sinh viên được triển khai trên server/thiết bị gì, cấu hình như thế nào. Kết quả triển khai thử nghiệm nếu có (số lượng người dùng, số lượng truy cập, thời gian phản hồi, phản hồi người dùng, khả năng chịu tải, các thống kê, v.v.)