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.

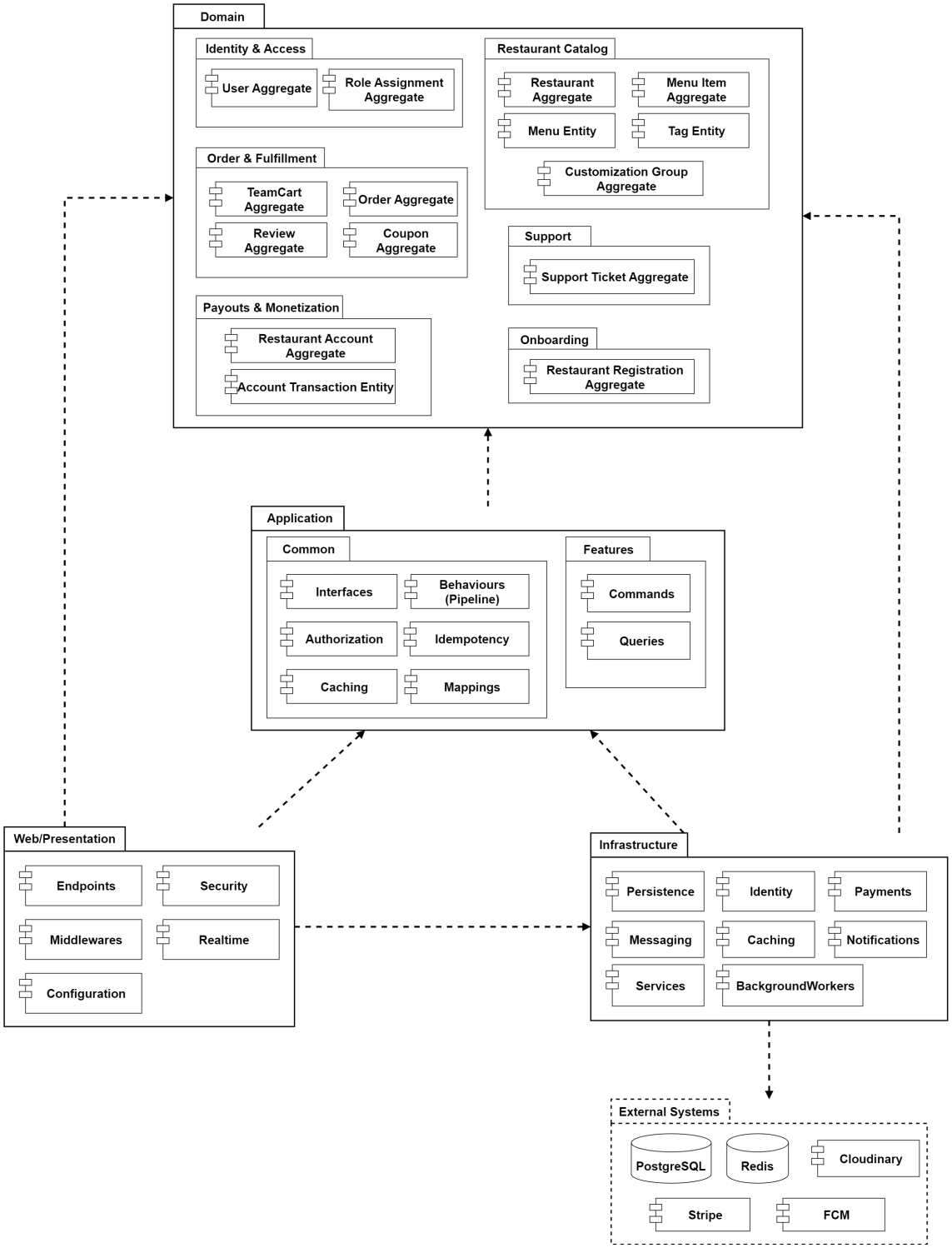
e, Kết luận

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

Biểu đồ gói tổng quan của hệ thống YummyZoom thể hiện sự phân tầng rõ ràng và các ràng buộc phụ thuộc tuân thủ quy tắc Clean Architecture.



Hình 0.1: Biểu đồ phụ thuộc gói tổng quan YummyZoom (Backend)

0.1.3 Thiết kế chi tiết gói

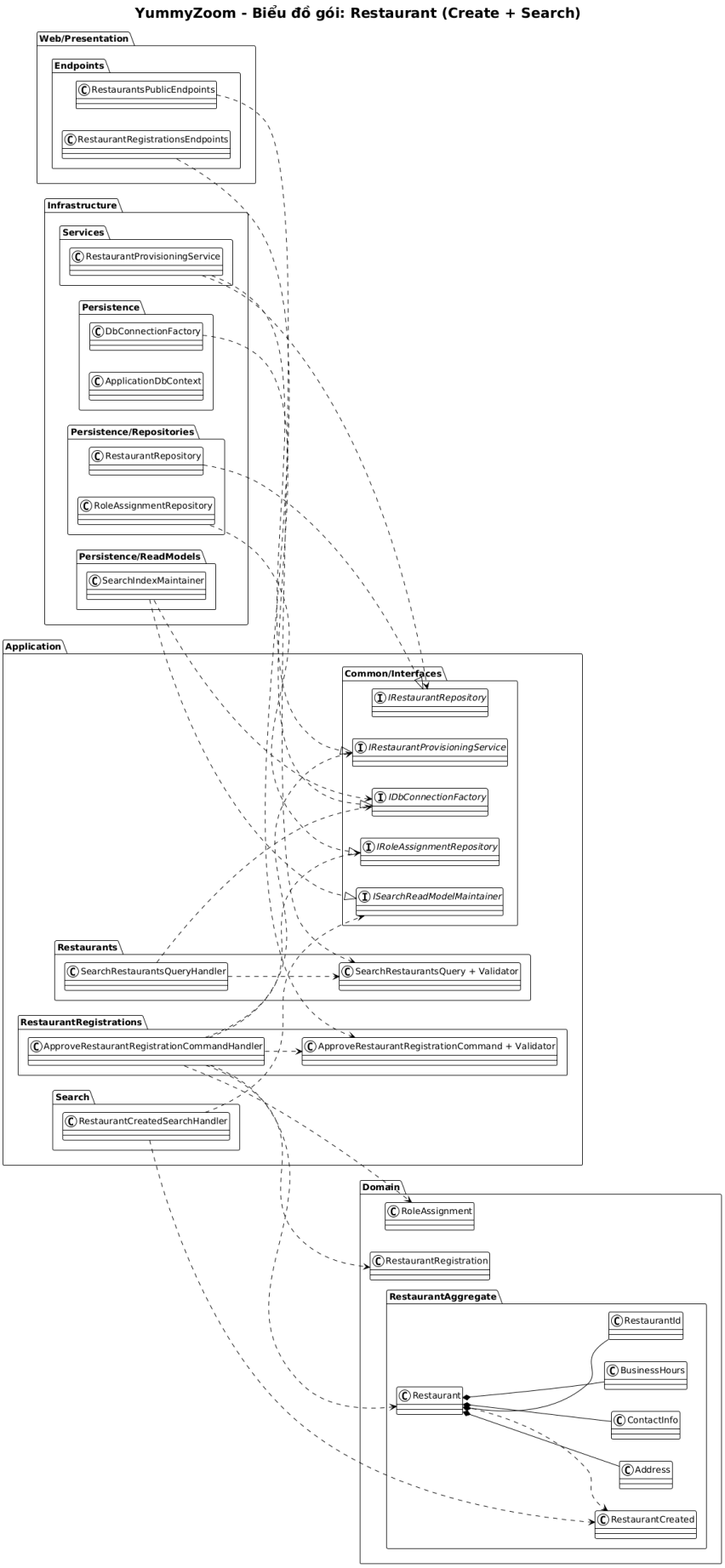
Mục tiêu của phần này là minh họa chi tiết cách tổ chức mã nguồn và hiện thực hóa kiến trúc Clean Architecture thông qua các biểu đồ gói (Package Diagrams). Tôi đã lựa chọn 2 luồng nghiệp vụ tiêu biểu đại diện cho hai phân hệ quan trọng nhất: Restaurant (Nhà hàng) và Order (Đơn hàng) để phân tích.

Các biểu đồ được xây dựng dựa trên các quy tắc sau nhằm đảm bảo tính trực quan và tránh sự quá tải thông tin do số lượng lớp lớn của hệ thống:

- **Phạm vi:** Mỗi biểu đồ tập trung vào một luồng nghiệp vụ cụ thể (bao gồm 1 Command và 1 Query), không bao gồm toàn bộ các lớp trong mã nguồn.
- **Cấu trúc:** Các thành phần được vẽ và sắp xếp dọc theo 4 tầng của Clean Architecture (Web, Application, Domain, Infrastructure).
- **Chi tiết:** Chỉ thể hiện tên lớp và các mối quan hệ chính, lược bỏ các phương thức và thuộc tính chi tiết.

a, Biểu đồ 1: Restaurant (Create Restaurant + Search Restaurants)

Biểu đồ này minh họa luồng xử lý cho hai chức năng: Phê duyệt đăng ký nhà hàng (Create) và Tìm kiếm nhà hàng (Search).



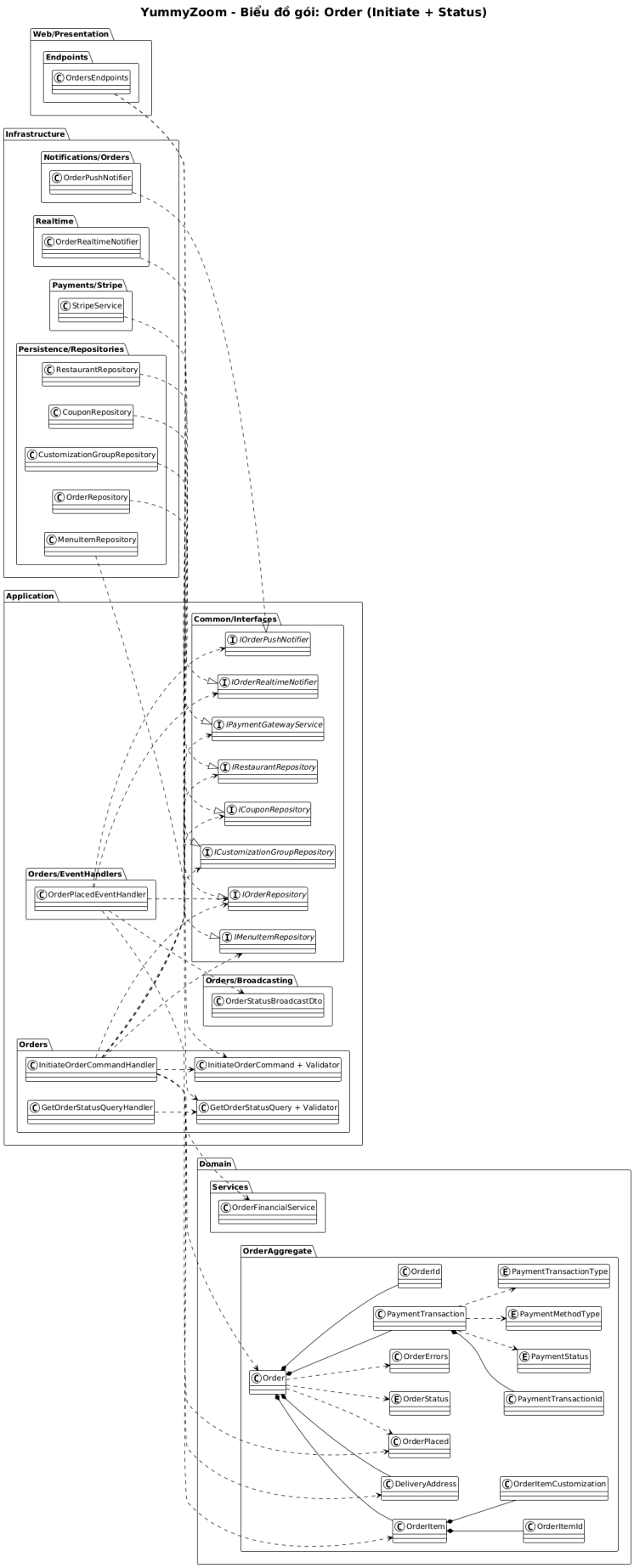
Hình 0.2: Biểu đồ gói chi tiết phân hệ Restaurant (Create & Search)

Mô tả luồng logic: Biểu đồ thể hiện cách các tầng phối hợp để xử lý yêu cầu:

- **Luồng Ghi (Command):** Bắt đầu từ yêu cầu POST đến Endpoints/RestaurantRegistrations để phê duyệt đăng ký nhà hàng, yêu cầu phê duyệt được chuyển đến ApproveRestaurantRegistrationCommandHandler (Application). Handler này tương tác với Domain (RestaurantAggregate, RestaurantRegistration) để đảm bảo quy tắc nghiệp vụ, rồi một đối tượng Restaurant được tạo và lưu trữ qua RestaurantRepository (Infrastructure) xuống database. Khi thành công, sự kiện RestaurantCreated được phát ra và được xử lý bởi RestaurantCreatedSearchHandler (Application) để cập nhật bảng SearchIndex trong database để phục phụ cho thao tác tìm kiếm sau này.
- **Luồng Đọc (Query):** Khi yêu cầu GET đến Restaurants để tìm kiếm nhà hàng theo các tiêu chí, SearchRestaurantsQueryHandler (Application) thực hiện truy vấn tối ưu bằng SQL vào bảng SearchIndex là một Read Model tối ưu hóa để phục vụ tìm kiếm. Dữ liệu phục vụ tìm kiếm được cập nhật bất đồng bộ bởi SearchIndexMaintainer (Infrastructure) ngay khi nhận được sự kiện RestaurantCreated, minh họa cho mô hình CQRS tách biệt Ghi và Đọc.

b, Biểu đồ 2: Order (Initiate Order + Get Order Status)

Biểu đồ này mô tả quy trình khởi tạo đơn hàng mới và truy vấn trạng thái đơn hàng, đóng vai trò cốt lõi trong giao dịch thương mại của hệ thống.



Hình 0.3: Biểu đồ gói chi tiết phân hệ Order (Initiate & Status)

Mô tả luồng logic: Biểu đồ minh họa sự phối hợp chặt chẽ giữa các thành phần để xử lý đơn hàng:

- **Luồng Ghi (Command):** Yêu cầu đặt hàng từ người dùng (Web Endpoints) được đóng gói thành InitiateOrderCommand. InitiateOrderCommandHandler đóng vai trò điều phối chính: nó sử dụng OrderFinancialService (một Domain Service) để tính toán chi phí, xác thực qua IPaymentGatewayService, và tương tác với các repositories (IOrderRepository, ICouponRepository) để lưu trữ Aggregate Order. Thành công kích hoạt sự kiện OrderPlaced, dẫn đến việc OrderPlacedEventHandler gọi các dịch vụ thông báo (IOrderRealtimeNotifier, IOrderPushNotifier).
- **Luồng Đọc (Query):** GetOrderStatusQueryHandler nhận truy vấn qua IDbConnectionFactory (Dapper) để lấy projection gọn của đơn hàng (OrderStatusDto). Handler kiểm tra quyền truy cập dựa trên IUser/claims (khách hàng hoặc nhân viên/owner nhà hàng), sau đó trả về trạng thái và mốc thời gian cập nhật, giúp client theo dõi tiến trình đơn hàng theo thời gian thực mà không tải dư dữ liệu.

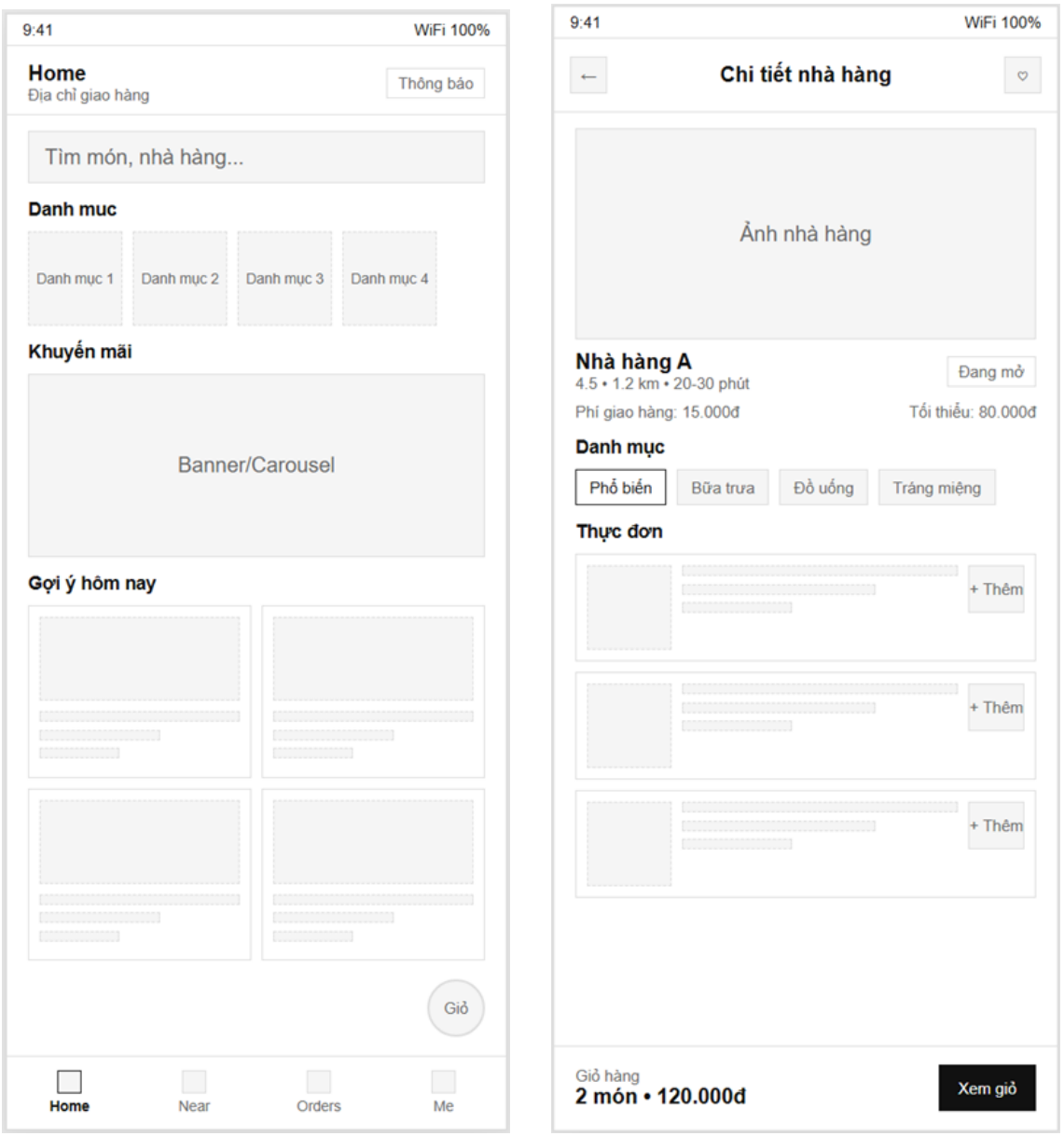
0.2 Thiết kế chi tiết

0.2.1 Thiết kế giao diện

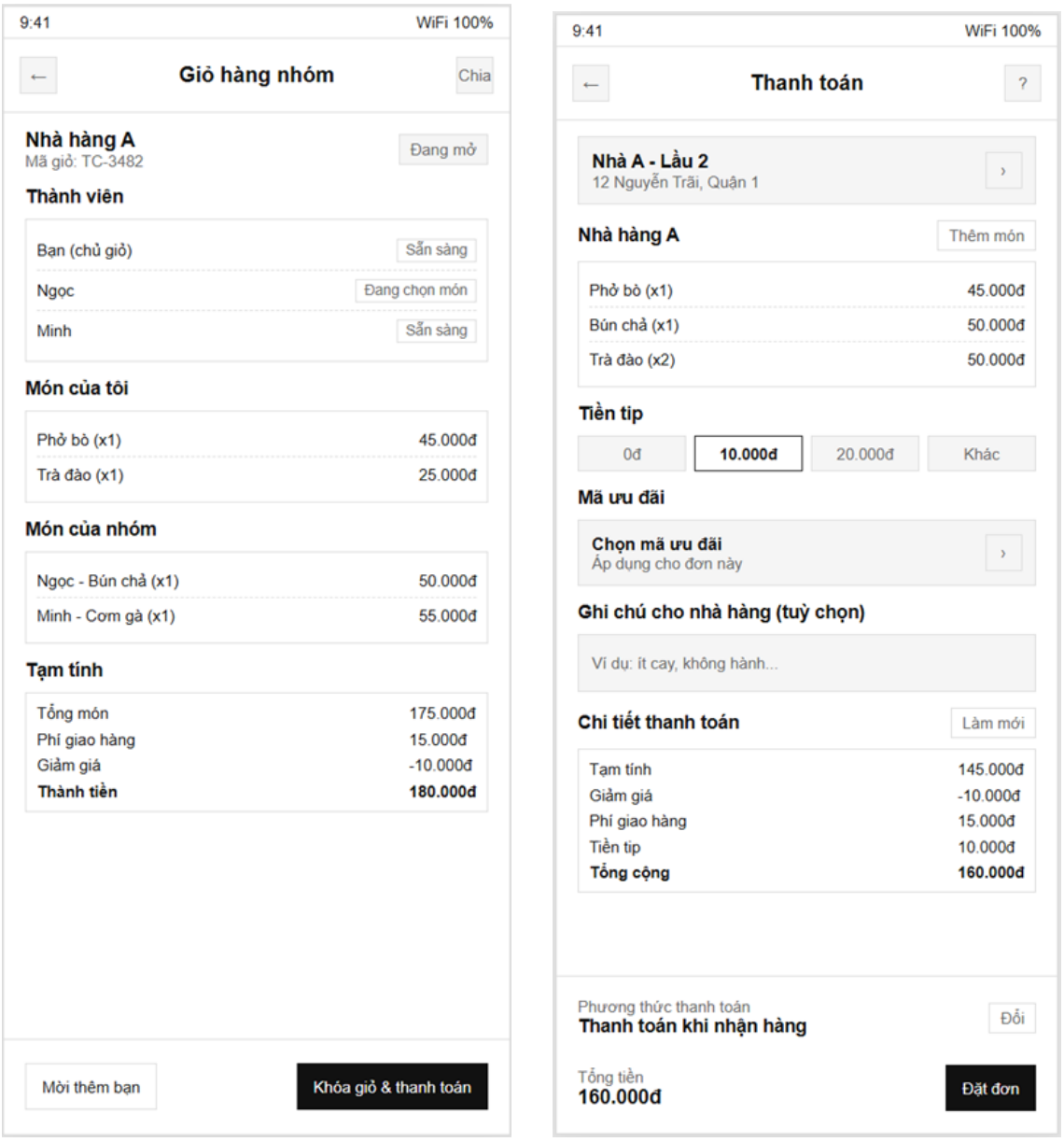
a, Thiết kế giao diện ứng dụng di động dành cho khách hàng

Ứng dụng di động được thiết kế hướng tới thiết bị 384x854 để đảm bảo tỷ lệ hiển thị thống nhất với cấu hình thiết kế, ưu tiên trải nghiệm đặt món nhanh và rõ ràng. Bố cục được tổ chức theo thứ bậc thông tin, tiêu đề và dữ liệu chính đặt ở vùng nhìn đầu tiên, các hành động quan trọng giữ vị trí nổi bật nhằm giảm thao tác thừa. Hệ thống kiểu chữ và kích thước chữ được quy đổi theo thang sp để bảo toàn độ đọc trên nhiều thiết bị, trong khi các thành phần nhập liệu và nút bấm thống nhất bán kính bo góc và khoảng đệm để tạo cảm giác thân thiện. Màu sắc thương hiệu (màu chính, màu phụ và màu nhấn) được chuẩn hóa trong ứng dụng, tuy nhiên bộ mockup được thể hiện theo tông đen trắng để tập trung vào cấu trúc và luồng thao tác, đồng thời phản hồi người dùng chủ yếu thông qua thông báo dạng thanh trượt ngắn gọn ở cuối màn hình.

Các màn hình tiêu biểu được lựa chọn nhằm phản ánh đầy đủ hành trình người dùng từ khám phá đến thanh toán, bao gồm trang Home/Menu và Restaurant Detail (Hình 0.4), cùng với trang TeamCart Lobby và Checkout/Payment (Hình 0.5).



Hình 0.4: Minh hoạ màn hình Home/Menu (trái) và Restaurant Detail (phải) trên ứng dụng di động

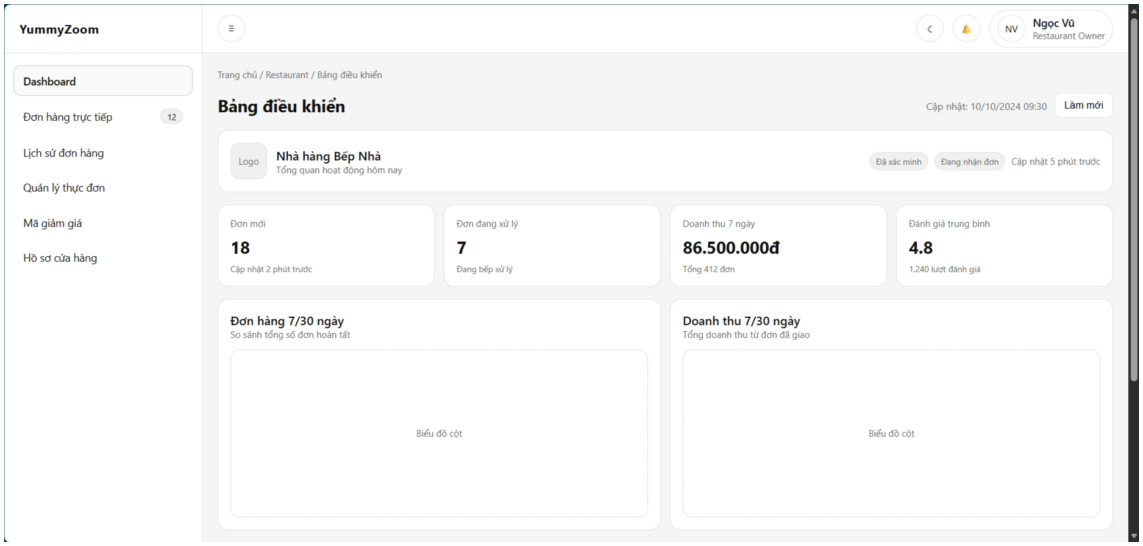


Hình 0.5: Minh hoạ màn hình TeamCart Lobby (trái) và Checkout/Payment (phải) trên ứng dụng di động

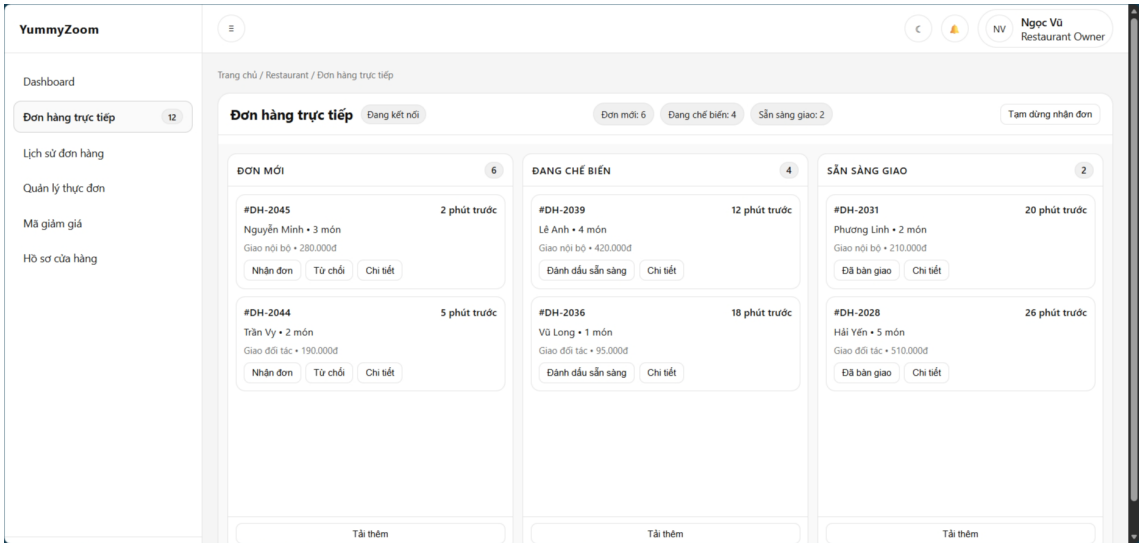
b, Thiết kế giao diện hệ thống quản trị trên web

Hệ thống quản trị hướng tới màn hình desktop 1440x900 với mật độ thông tin cao, đồng thời đảm bảo khả năng hiển thị trên tablet ngang. Bố cục ưu tiên khả năng quét nhanh dữ liệu với lưới khoảng cách theo bước nhỏ, các khối thông tin và bảng dữ liệu được sắp xếp gọn để phục vụ tác vụ điều hành. Kiểu chữ được phân cấp rõ ràng từ nhãn điều hướng đến tiêu đề trang, kết hợp các mức đậm khác nhau để nhấn mạnh số liệu quan trọng. Hệ màu thương hiệu được sử dụng cho hành động chính và trạng thái tương tác, trong khi phiên bản nền tối được hỗ trợ để cải thiện khả năng đọc ở môi trường ánh sáng yếu. Các thành phần nhập liệu, nút bấm và thông báo hệ thống được chuẩn hóa về viền, trạng thái tập trung và hiển thị lỗi nhằm giữ tính nhất quán trong các luồng nghiệp vụ.

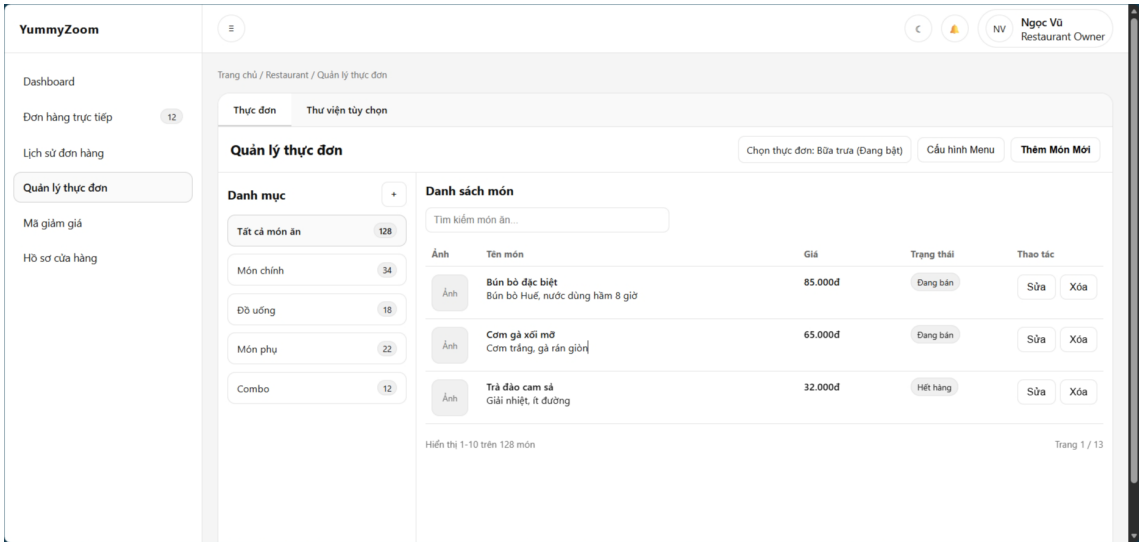
Các màn hình minh họa tập trung vào những chức năng cốt lõi của quản trị, gồm trang Restaurant Dashboard (Hình 0.6), trang Live Orders (Hình 0.7), trang Quản lý thực đơn (Hình 0.8) và trang Duyệt đăng ký nhà hàng (Hình 0.9).



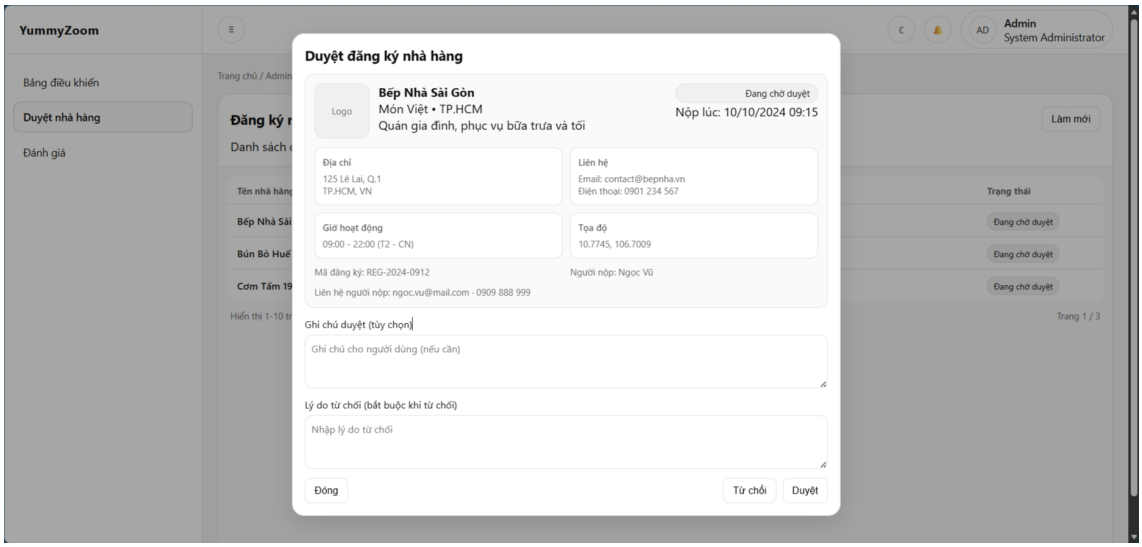
Hình 0.6: Minh hoạ màn hình Restaurant Dashboard trên hệ thống quản trị



Hình 0.7: Minh hoạ màn hình Live Orders trên hệ thống quản trị



Hình 0.8: Minh hoạ màn hình Quản lý thực đơn trên hệ thống quản trị



Hình 0.9: Minh hoạ màn hình Duyệt đăng ký nhà hàng trên hệ thống quản trị

0.2.2 Thiết kế lớp

Trong dự án YummyZoom, phần lớn quy tắc nghiệp vụ được đóng gói tại lớp miền (Domain layer) theo Clean Architecture, do đó phần thiết kế lớp tập trung vào các aggregate root tiêu biểu. Bốn lớp được lựa chọn gồm Order, TeamCart, Restaurant và MenuItem vì đây là các lớp chịu trách nhiệm quản lý vòng đời, trạng thái và các bất biến nghiệp vụ quan trọng của hệ thống. Đối với các lớp ở lớp ứng dụng (Application layer), nội dung chỉ trình bày ở mức vai trò điều phối luồng, bởi đa số các handler chỉ thực hiện một phương thức *Handle* với logic mỏng.

Lớp Order đại diện cho đơn hàng, quản lý dữ liệu tài chính, danh sách món đã đặt và các mốc trạng thái trong vòng đời xử lý. Thiết kế của lớp thể hiện rõ các thuộc tính tài chính (Subtotal, DiscountAmount, DeliveryFee, TipAmount,

TaxAmount, TotalAmount) cùng các phương thức chuyển trạng thái như Accept, Reject, MarkAsPreparing, MarkAsReadyForDelivery và MarkAsDelivered, đồng thời phát sinh các sự kiện miền tương ứng phục vụ đồng bộ trạng thái theo thời gian thực. Hình 0.10 mô tả cấu trúc lớp Order và các mối quan hệ chính.

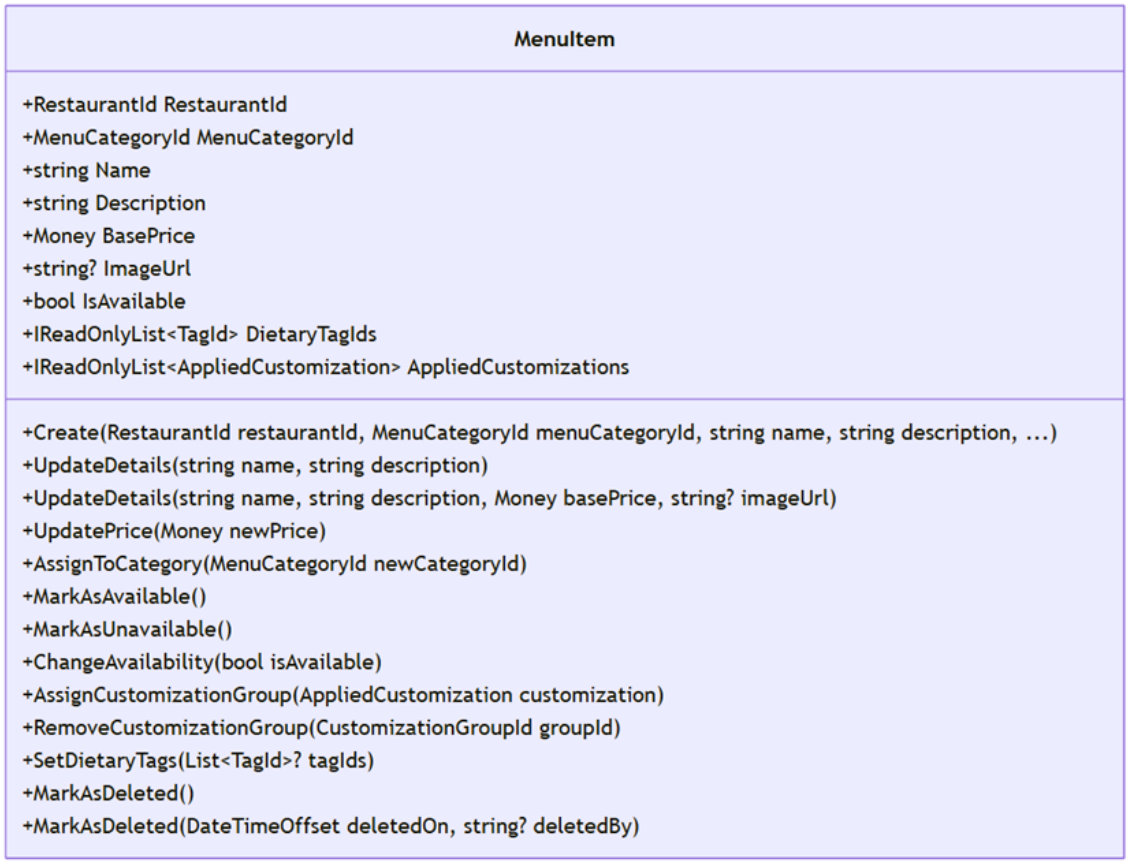
Hình 0.10: Biểu đồ lớp của Order (Aggregate root)

Lớp TeamCart mô hình hóa giỏ hàng nhóm, cho phép nhiều người cùng đặt món và phối hợp thanh toán. Các thuộc tính cốt lõi gồm thông tin thành viên, danh sách món, trạng thái giỏ và các cấu phần tài chính như TipAmount và QuoteVersion. Các phương thức như AddItem, LockForPayment, FinalizePricing và RecordSuccessfulOnlinePayment giúp đảm bảo tính nhất quán của luồng thanh toán nhóm. Hình 0.11 trình bày thiết kế lớp TeamCart.

Hình 0.11: Biểu đồ lớp của TeamCart (Aggregate root)

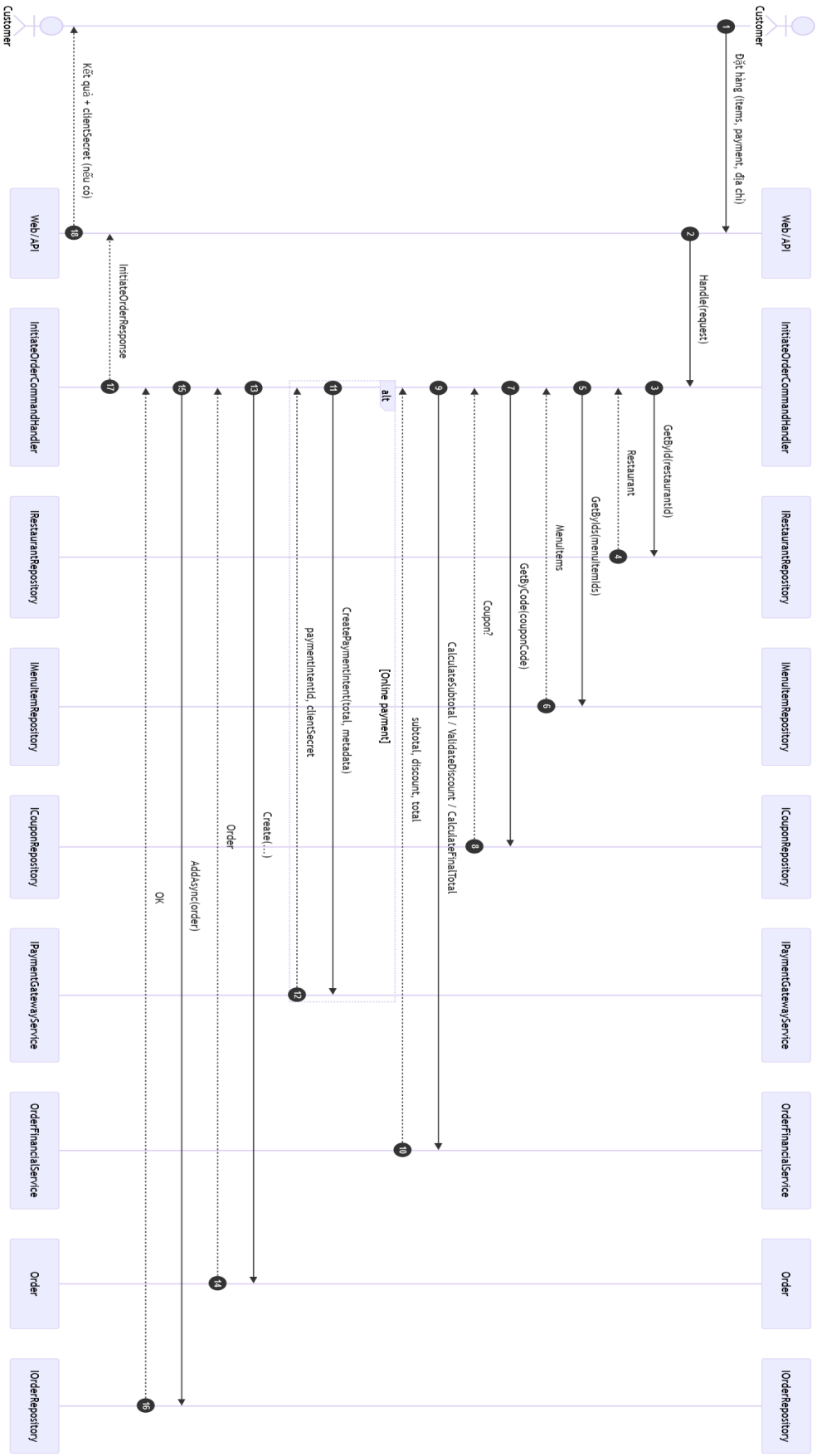
Lớp Restaurant quản lý hồ sơ nhà hàng, các thông tin định danh và trạng thái hoạt động như đã xác thực hay đang nhận đơn. Thiết kế này đảm bảo các ràng buộc nghiệp vụ khi thay đổi thông tin thương hiệu, địa điểm, khung giờ hoạt động và trạng thái xác thực. Lớp MenuItem quản lý món ăn của nhà hàng, hỗ trợ các thao tác cập nhật mô tả, giá, khả dụng và cấu hình nhóm tùy chọn. Hai lớp này lần lượt được minh họa tại Hình 0.12 và Hình 0.13.

Hình 0.12: Biểu đồ lớp của Restaurant (Aggregate root)

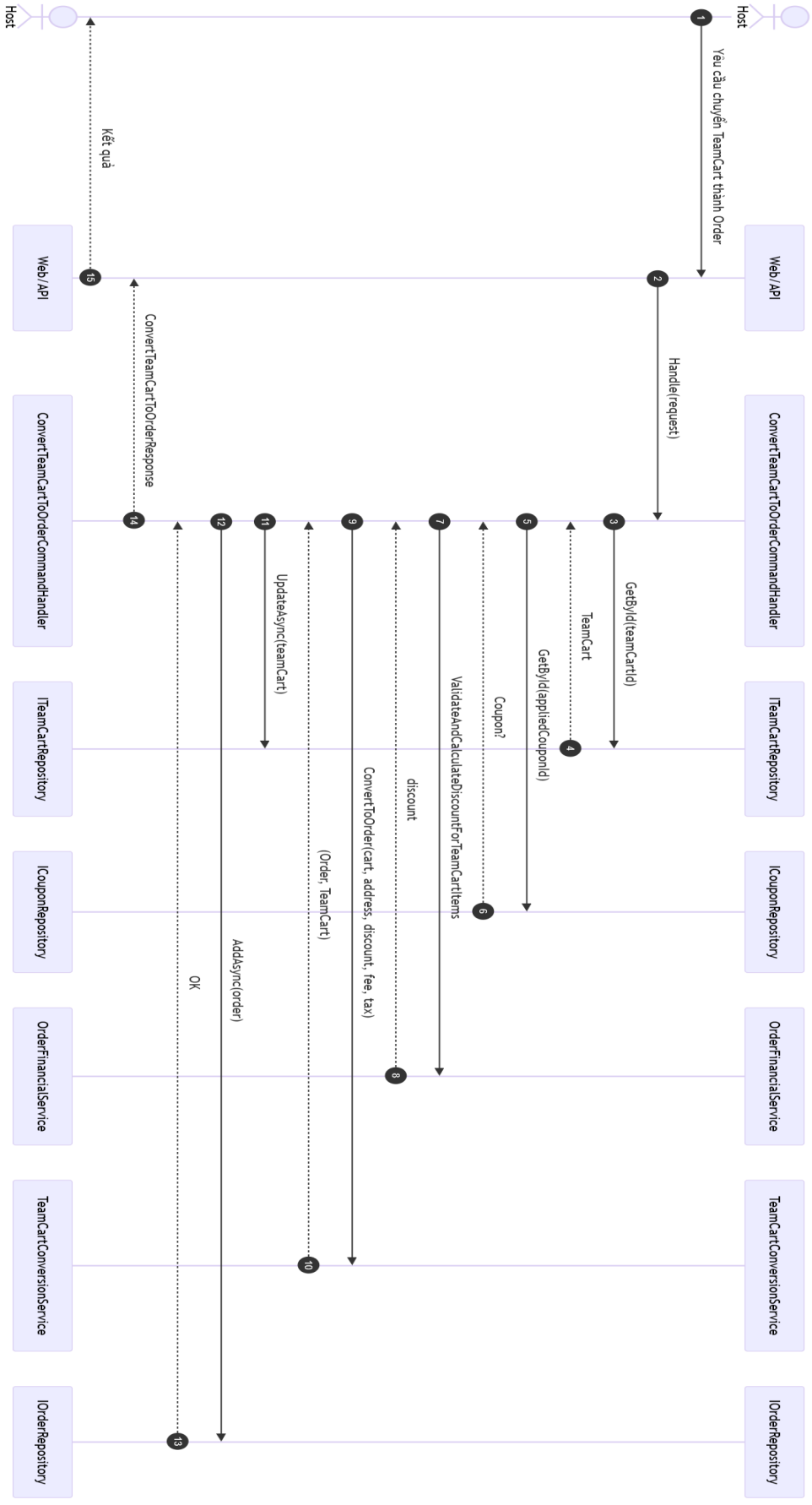


Hình 0.13: Biểu đồ lớp của MenuItem (Aggregate root)

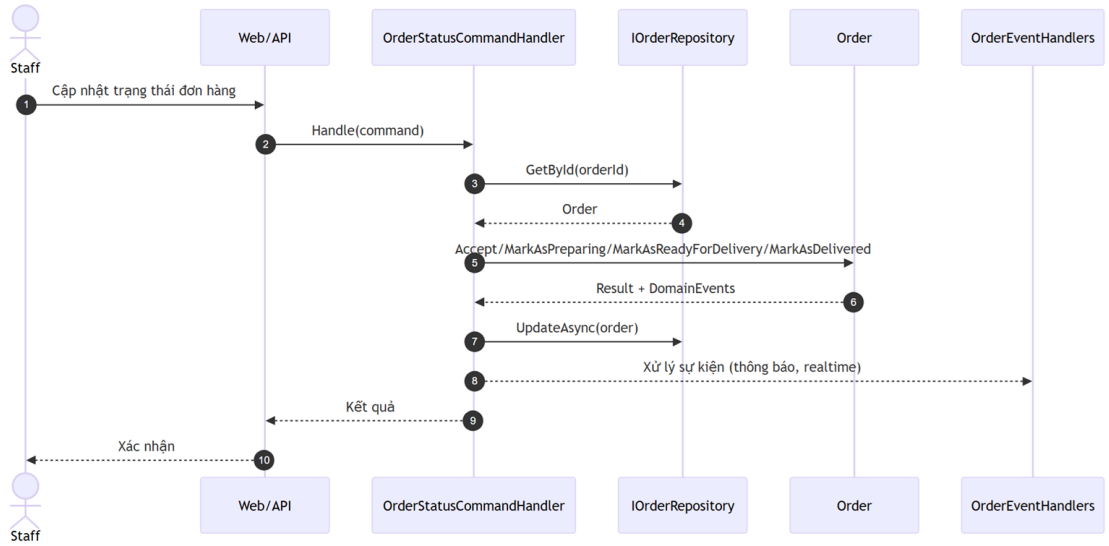
Để minh họa cách các lớp phối hợp trong các ca sử dụng quan trọng, ba luồng chính được lựa chọn gồm khởi tạo đơn hàng, chuyển TeamCart thành Order và cập nhật trạng thái đơn hàng. Luồng khởi tạo đơn hàng thể hiện kiểm tra nhà hàng, món ăn, tính phí và tạo thanh toán, sau đó sinh Order ở lớp miền. Luồng chuyển TeamCart thành Order nhấn mạnh vai trò của Domain Service trong việc ánh xạ dữ liệu và tính toán tài chính trước khi tạo Order và cập nhật TeamCart. Luồng cập nhật trạng thái đơn hàng thể hiện các chuyển trạng thái hợp lệ trong Order và phát sinh sự kiện miền để phục vụ cập nhật thời gian thực. Các luồng này được trình bày lần lượt tại Hình 0.14, Hình 0.15 và Hình 0.16.



Hình 0.14: Biểu đồ trình tự khởi tạo đơn hàng



Hình 0.15: Biểu đồ trình tự chuyển TeamCart thành Order

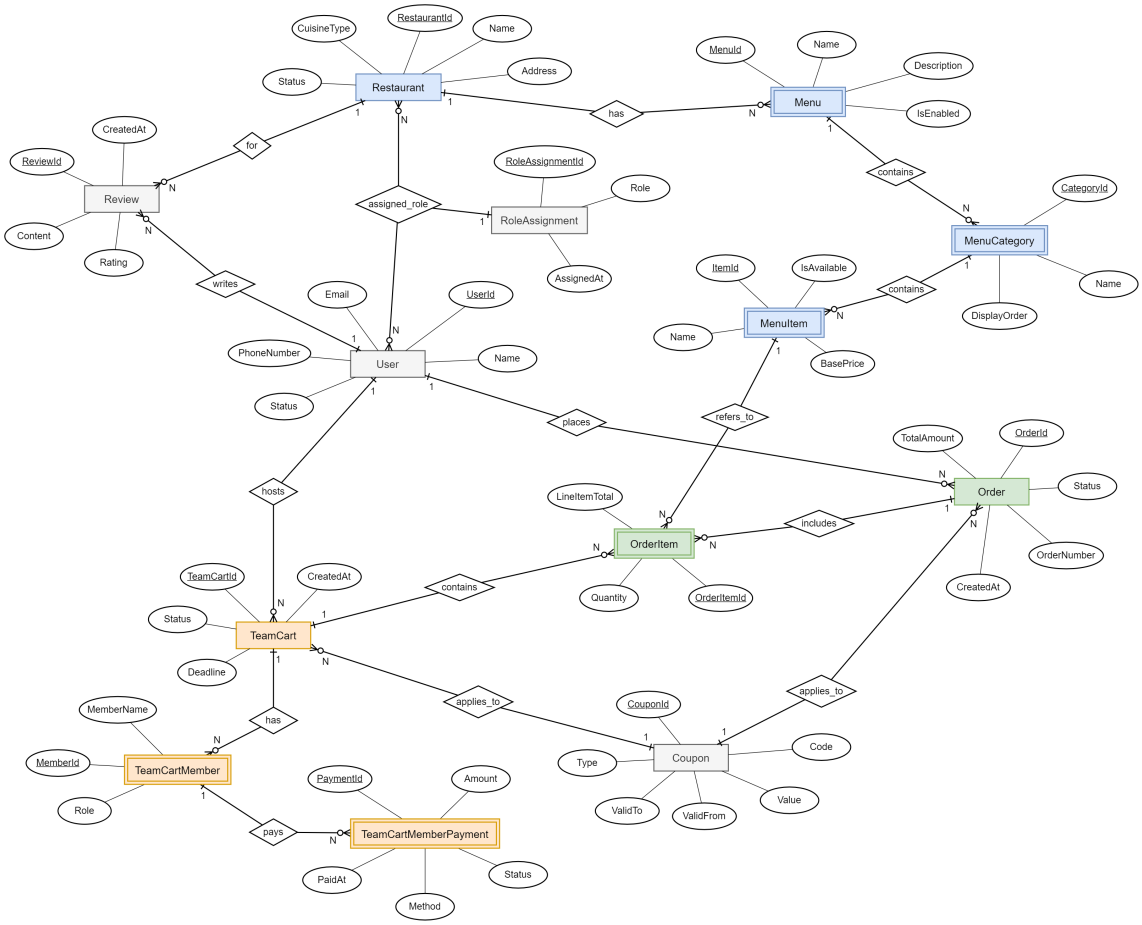


Hình 0.16: Biểu đồ trình tự cập nhật trạng thái đơn hàng

0.2.3 Thiết kế cơ sở dữ liệu

Trong dự án YummyZoom, cơ sở dữ liệu được triển khai trên PostgreSQL. Phần thiết kế dữ liệu được trình bày theo hai tầng: mô hình khái niệm bằng ERD kiểu Chen để mô tả thế giới thực và các thực thể nghiệp vụ, và mô hình quan hệ chi tiết phản ánh cấu trúc bảng sau khi ánh xạ vào cơ sở dữ liệu. Nguồn lược đồ tổng hợp được đối chiếu từ tài liệu kiến trúc, nhằm đảm bảo tính nhất quán giữa mô hình miền và dữ liệu lưu trữ.

Biểu đồ ERD khái niệm theo Chen tập trung vào các thực thể cốt lõi như Người dùng, Nhà hàng, Thực đơn, Món ăn, Đơn hàng và Giỏ nhóm, cùng các quan hệ nghiệp vụ giữa chúng. Mỗi thực thể chỉ giữ lại một số thuộc tính chính như định danh, trạng thái, thông tin mô tả và giá trị tiền tệ, giúp làm rõ bức tranh nghiệp vụ mà chưa bị ràng buộc bởi chi tiết triển khai. Hình 0.17 mô tả toàn bộ ERD khái niệm, trong đó các quan hệ chính như Nhà hàng–Thực đơn, Người dùng–Đơn hàng, Đơn hàng–Mục đơn hàng và Giỏ nhóm–Thành viên được biểu diễn rõ ràng theo bội số.

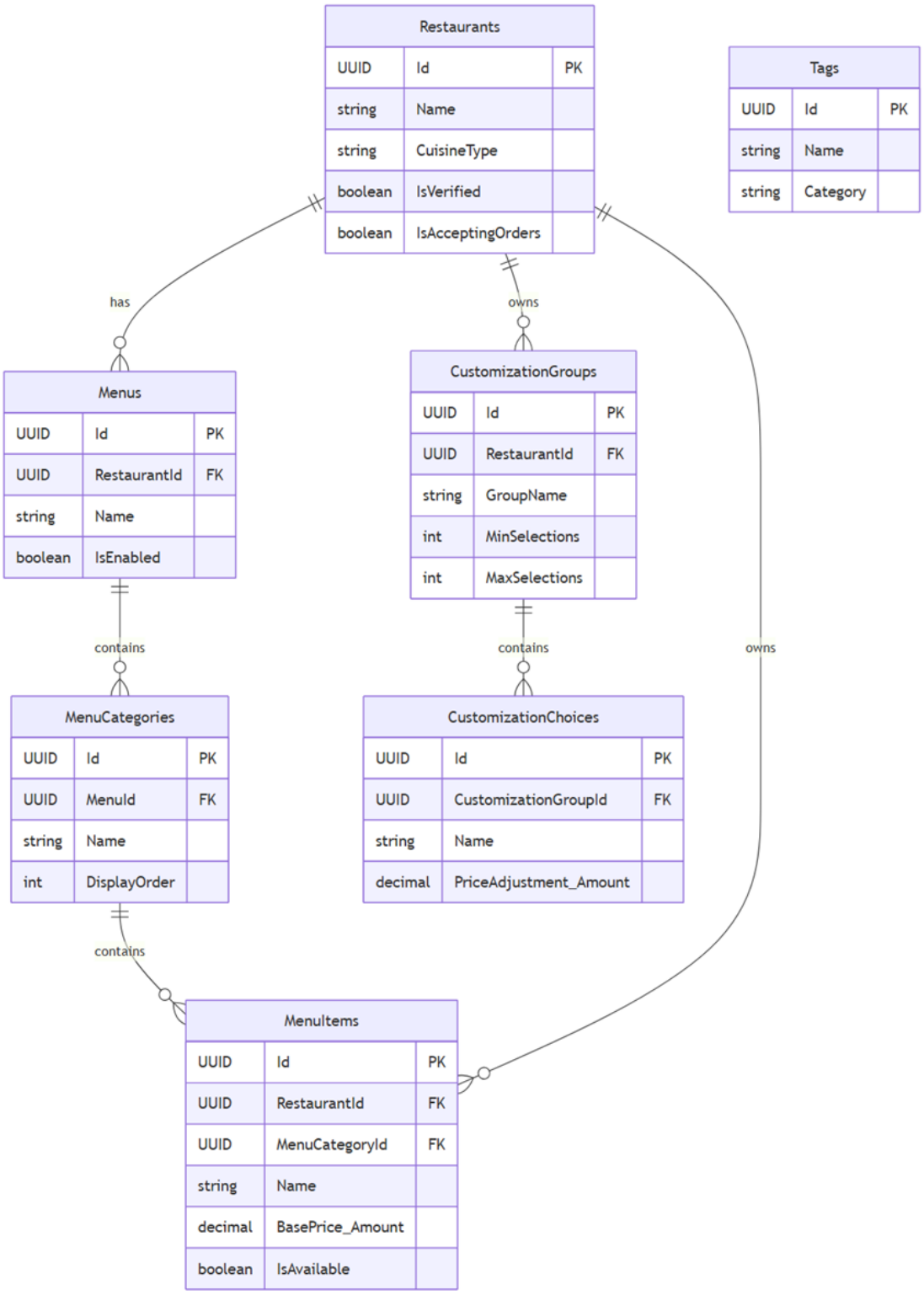


Hình 0.17: ERD khái niệm theo mô hình Chen cho YummyZoom

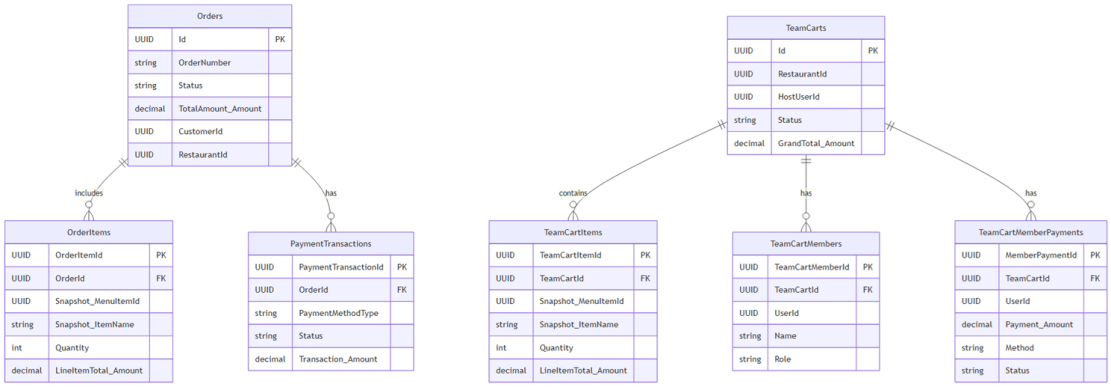
Từ mô hình khái niệm, hệ thống áp dụng quy tắc ánh xạ (mapping) từ mô hình miền (domain model) sang cơ sở dữ liệu quan hệ. Mỗi aggregate root được ánh xạ thành một bảng chính, ví dụ Order tương ứng bảng Orders, Restaurant tương ứng bảng Restaurants, TeamCart tương ứng bảng TeamCarts. Các thực thể có danh tính trong một aggregate được tách thành bảng con và liên kết bằng khóa ngoại về bảng gốc, chẳng hạn OrderItems và PaymentTransactions phụ thuộc Orders, TeamCartItems và TeamCartMembers phụ thuộc TeamCarts. Các giá trị bất biến dạng đối tượng giá trị (value object) được ánh xạ thành các cột trong bảng chính với tiền tố tên thuộc tính, ví dụ DeliveryAddress_Street hoặc TipAmount_Amount; với các giá trị phức tạp hoặc danh sách linh hoạt thì lưu dưới dạng JSONB để giảm độ phức tạp lược đồ. Quy tắc này được hiện thực nhất quán thông qua cấu hình EF Core (Fluent API) trong các lớp Configuration thuộc tầng Infrastructure, đồng thời các trạng thái dạng enum được lưu dưới dạng chuỗi để dễ đọc và theo dõi.

Ở mức triển khai vật lý, số lượng bảng lớn được chia nhỏ theo từng nhóm chức năng để trình bày rõ ràng. Hình 0.18 mô tả nhóm Restaurant Catalog, gồm Restaurants, Menus, MenuCategories, MenuItems và các bảng tùy chọn món; nhóm này phản ánh cấu trúc dữ liệu phục vụ duyệt thực đơn và quản trị nội dung. Hình 0.19

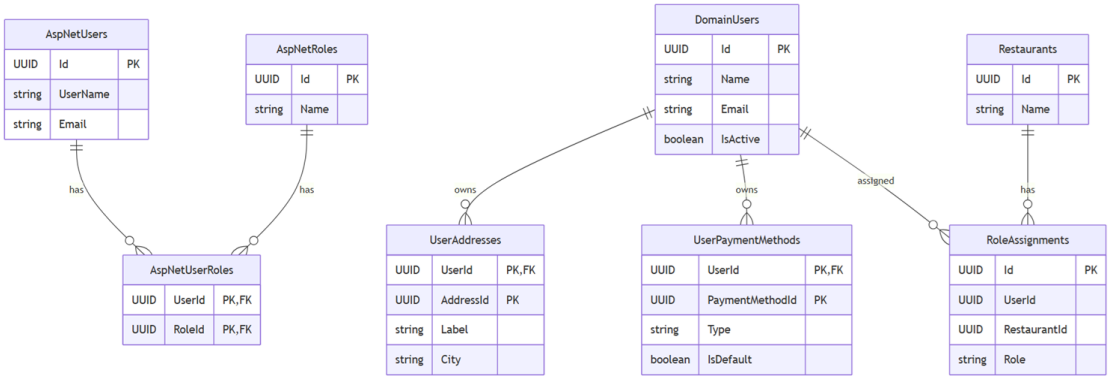
trình bày nhóm Order & TeamCart, làm rõ cấu trúc lưu trữ đơn hàng, mục đơn hàng, thanh toán, giỏ nhóm và thành viên. Hình 0.20 thể hiện nhóm Identity & Access, bao gồm các bảng nhận thực người dùng, bảng người dùng miền và bảng phân quyền nhà hàng.



Hình 0.18: ERD hiện đại cho nhóm Restaurant Catalog



Hình 0.19: ERD hiện đại cho nhóm Order và TeamCart



Hình 0.20: ERD hiện đại cho nhóm Identity và RoleAssignment

0.3 Xây dựng ứng dụng

0.3.1 Thư viện và công cụ sử dụng

Trong quá trình phát triển hệ thống YummyZoom, nhóm phát triển đã lựa chọn và sử dụng một loạt các thư viện và công cụ nhằm tối ưu hóa quy trình làm việc, nâng cao chất lượng mã nguồn và đảm bảo hiệu suất của ứng dụng. Bảng dưới đây liệt kê các công cụ chính cùng với mục đích sử dụng và địa chỉ URL tham khảo.

Mục đích	**Công cụ**	**Địa chỉ URL**
IDE lập trình	Visual Studio Code	`https://code.visualstudio.com/`
IDE lập trình cho .NET	JetBrains Rider	`https://www.jetbrains.com/rider/`
IDE lập trình cho Android	Android Studio	`https://developer.android.com/studio`
Quản lý mã nguồn	Git	`https://git-scm.com/`

(Tiếp tục trang sau)

(Tiếp theo từ trang trước)

Quản lý mã nguồn và CI/CD	GitHub	https://github.com/
AI hỗ trợ lập trình	GitHub Copilot	https://github.com/features/copilot
AI hỗ trợ lập trình	Cursor	https://www.cursor.com/
Backend nền tảng	.NET 9.0	https://dotnet.microsoft.com/
Backend framework	ASP.NET Core	https://learn.microsoft.com/aspnet/core/
ORM/DAL	Entity Framework Core	https://learn.microsoft.com/ef/core/
Backend kiểm thử	NUnit	https://nunit.org/
Backend kiểm thử	Testcontainers	https://testcontainers.com/
Công cụ quản lý runtime	.NET Aspire	https://learn.microsoft.com/dotnet/aspire/
Cơ sở dữ liệu	PostgreSQL	https://www.postgresql.org/
Bộ nhớ đệm	Redis	https://redis.io/
Mobile framework	Flutter	https://flutter.dev/
Mobile ngôn ngữ	Dart	https://dart.dev/
Mobile UI/UX	Flutter Material	https://docs.flutter.dev/ui/widgets/material
Mobile quản lý trạng thái	Provider	https://pub.dev/packages/provider
Mobile lưu trữ	Hive	https://pub.dev/packages/hive
Mobile bản đồ	Mapbox	https://www.mapbox.com/
Mobile thông báo đẩy	Firebase Messaging	https://firebase.google.com/docs/cloud-messaging
Mobile payment	flutter_stripe	https://pub.dev/packages/flutter_stripe
Web admin framework	Angular	https://angular.dev/

(Tiếp tục trang sau)

(Tiếp theo từ trang trước)

Web admin ngôn ngữ	TypeScript	`https://www.typescriptlang.org/`
Web admin UI	PrimeNG + PrimeIcons	`https://primeng.org/`
Web admin UI	Tailwind CSS	`https://tailwindcss.com/`
Web admin cộng tác thời gian thực	SignalR JS Client	`https://learn.microsoft.com/aspnet/core/signalr/javascript-client`
Dịch vụ bản đồ	Mapbox API	`https://docs.mapbox.com/`
Dịch vụ thông báo	Firebase Messaging	`https://firebase.google.com/`
Dịch vụ thanh toán	Stripe	`https://stripe.com/`

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

0.3.2 Kết quả đạt được

a, Backend

Sản phẩm backend được đóng gói dưới dạng dịch vụ ASP.NET Core Web API. Các số liệu thống kê chính được tổng hợp trong Bảng 0.2.

Chỉ tiêu	**Giá trị**	**Ghi chú**
Tổng số dòng code	185,965	Bao gồm comment và dòng trống
Số file `.cs`	1,348	Toàn bộ backend
Số lớp C#	1,301	Thống kê theo source code
Số bảng CSDL	49	Đếm theo `schema.sql`
Dung lượng mã nguồn `src/`	103 MB	Sau khi `dotnet clean`
Dung lượng mã nguồn `tests/`	179 MB	Sau khi `dotnet clean`
Dung lượng gói publish Release	34 MB	`dotnet publish`

Bảng 0.2: Thống kê kết quả backend

b, Mobile (Flutter/Dart)

Ứng dụng di động dành cho khách hàng được phát triển bằng Flutter/Dart. Các số liệu thống kê chính được tổng hợp trong Bảng 0.3.

Chỉ tiêu	Giá trị	Ghi chú
Tổng số dòng code	45,791	Bao gồm comment và dòng trống
Số file `.dart`	325	Toàn bộ ứng dụng mobile
Dung lượng mã nguồn	2.0 MB	Tổng thư mục mã nguồn `lib/`
Dung lượng APK Release	117.5 MB	Bản build APK

Bảng 0.3: Thống kê kết quả mobile

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.)