Chương này trình bày chi tiết về các công nghệ, nền tảng và giải pháp kỹ thuật được lựa chọn để xây dựng hệ thống YummyZoom. Các quyết định công nghệ được đưa ra dựa trên việc phân tích kỹ lưỡng các yêu cầu chức năng và phi chức năng đã được khảo sát ở Chương 2, đặc biệt là các vấn đề về hiệu năng, tính toàn vẹn dữ liệu, khả năng bảo trì và trải nghiệm người dùng.

0.1 Kiến trúc hệ thống và Ứng dụng Backend

Hệ thống backend đóng vai trò là xương sống của toàn bộ ứng dụng YummyZoom, chịu trách nhiệm xử lý các nghiệp vụ cốt lõi, quản lý dữ liệu và điều phối các tương tác giữa các phân hệ khách hàng và nhà hàng. Để đáp ứng các yêu cầu khắt khe về **Khả năng bảo trì** (Maintainability) và **Độ tin cậy** (Reliability) được đặt ra trong Mục 2.4, việc lựa chọn kiến trúc phần mềm và nền tảng phát triển phù hợp là yếu tố tiên quyết.

0.1.1 Kiến trúc Clean Architecture và Domain-Driven Design (DDD)

Trong bối cảnh phát triển tính năng **TeamCart** (Giỏ hàng nhóm), hệ thống phải đối mặt với các nghiệp vụ phức tạp như đồng bộ trạng thái giữa nhiều người dùng, tính toán chia sẻ hóa đơn và xử lý các giao dịch đồng thời. Để giải quyết các thách thức này, đồ án áp dụng kiến trúc **Clean Architecture** (Kiến trúc sạch) do Robert C. Martin đề xuất, kết hợp với các nguyên lý của **Domain-Driven Design** (Thiết kế hướng tên miền) [1], [2].

Về mặt cấu trúc, hệ thống được tổ chức thành các lớp với quy tắc phụ thuộc hướng tâm (Dependency Rule), đảm bảo sự độc lập của logic nghiệp vụ:

1. **Lớp miền (Domain Layer):** Đây là lõi của hệ thống, chứa các thực thể (Entities), đối tượng giá trị (Value Objects) và các quy tắc nghiệp vụ (Business Rules) bất biến. Ví dụ, logic tính toán tổng tiền của một giỏ hàng nhóm hay quy tắc khóa đơn hàng khi thanh toán được cài đặt tại đây, hoàn toàn không phụ thuộc vào cơ sở dữ liệu hay giao diện người dùng.
2. **Lớp ứng dụng (Application Layer):** Đóng vai trò điều phối các ca sử dụng (Use Cases). Lớp này nhận yêu cầu từ bên ngoài, sử dụng các thực thể trong Domain để thực hiện tác vụ và trả về kết quả.
3. **Lớp hạ tầng (Infrastructure Layer):** Cài đặt chi tiết các giao tiếp kỹ thuật như truy xuất cơ sở dữ liệu, gửi email, tích hợp cổng thanh toán.
4. **Lớp giao diện (Presentation/Web Layer):** Nơi chứa các API Controller để tiếp nhận yêu cầu HTTP từ ứng dụng di động hoặc web quản trị.

Sự kết hợp này mang lại lợi ích vượt trội so với các kiến trúc khác:

- **So với Kiến trúc phân tầng truyền thống (Layered Architecture):** Kiến

trúc truyền thống thường dẫn đến sự phụ thuộc chặt chẽ giữa logic nghiệp vụ và cơ sở dữ liệu (ví dụ: thay đổi cấu trúc bảng làm hỏng logic code). Clean Architecture loại bỏ vấn đề này bằng cách đảo ngược sự phụ thuộc (Dependency Inversion), giúp logic nghiệp vụ ổn định và dễ dàng kiểm thử đơn vị (Unit Test) mà không cần kết nối database thực tế.

- **So với Kiến trúc Microservices:** Mặc dù Microservices mang lại khả năng mở rộng tốt, nhưng mô hình này đòi hỏi hạ tầng vận hành phức tạp (DevOps, Kubernetes) và chi phí giao tiếp giữa các dịch vụ lớn. Với quy mô nhân sự của một đồ án tốt nghiệp và mục tiêu phục vụ 50-100 người dùng đồng thời, việc áp dụng Clean Architecture dưới dạng Monolithic Modular (đơn khối module hóa) là giải pháp tối ưu, cân bằng giữa tốc độ phát triển và chất lượng mã nguồn, đồng thời dễ dàng tách module thành microservice khi cần mở rộng sau này.

0.1.2 Nền tảng phát triển: .NET 9 và ASP.NET Core

Để hiện thực hóa kiến trúc trên, hệ thống sử dụng nền tảng **.NET 9** cùng framework **ASP.NET Core Web API** [3]. Đây là một trong những nền tảng phát triển web hiệu năng cao và ổn định nhất hiện nay.

Lựa chọn .NET 9 giải quyết trực tiếp hai bài toán lớn được đặt ra ở phần khảo sát:

- **Tối ưu hóa hiệu năng và khả năng chịu tải:** Web server Kestrel của ASP.NET Core được tối ưu hóa cực tốt cho việc xử lý hàng nghìn yêu cầu mỗi giây. Kết hợp với mô hình lập trình bất đồng bộ (Async/Await), hệ thống có thể xử lý hiệu quả các tác vụ I/O (như đọc nghi dữ liệu, gọi API thanh toán) mà không làm tắc nghẽn luồng xử lý chính (Thread Blocking). Điều này đảm bảo thời gian phản hồi API luôn duy trì ở mức thấp (dưới 500ms) ngay cả khi có 50-100 người dùng thao tác cùng lúc trên TeamCart.
- **An toàn và dễ bảo trì:** Ngôn ngữ C# là ngôn ngữ định kiểu tĩnh mạnh (Strongly Typed), giúp phát hiện phần lớn các lỗi logic ngay trong quá trình biên dịch (Compile time), giảm thiểu đáng kể lỗi runtime so với các ngôn ngữ định kiểu động như JavaScript/Python.

Khi so sánh với các nền tảng phổ biến khác:

- **So với Node.js (Express/NestJS):** Node.js sử dụng mô hình đơn luồng (Single-thread event loop), rất tốt cho I/O nhưng có thể bị suy giảm hiệu năng nếu gặp các tác vụ tính toán phức tạp (như tổng hợp báo cáo doanh thu). .NET 9 với khả năng đa luồng (Multi-threading) quản lý tốt cả hai loại tác vụ này.

- **So với Java (Spring Boot):** Mặc dù Spring Boot rất mạnh mẽ, nhưng hệ sinh thái Java thường tiêu tốn nhiều tài nguyên bộ nhớ (RAM) hơn và có thời gian khởi động (Startup time) lâu hơn so với .NET Core hiện đại. Với yêu cầu triển khai linh hoạt và tiết kiệm tài nguyên trên môi trường Cloud hoặc VPS sinh viên, .NET 9 là lựa chọn kinh tế và hiệu quả hơn.

0.1.3 Điều phối hệ thống và Khả năng quan sát: .NET Aspire

Để giải quyết bài toán phức tạp trong việc quản lý, kết nối và vận hành các thành phần phân tán (Web API, Worker Service, Database, Redis) ngay từ môi trường phát triển (Local Development), dự án áp dụng stack công nghệ **.NET Aspire** [4].

Vai trò của .NET Aspire trong hệ thống YummyZoom:

- **Điều phối tài nguyên (Orchestration):** Thay vì phải cấu hình thủ công từng container Docker hoặc chạy nhiều cửa sổ dòng lệnh rời rạc, .NET Aspire cho phép định nghĩa toàn bộ hạ tầng (Infrastructure as Code) bằng C#. Chỉ với một nút nhấn "Start", toàn bộ hệ sinh thái bao gồm Backend, Database PostgreSQL container, và Redis container sẽ được khởi tạo và kết nối tự động thông qua cơ chế Service Discovery.
- **Khả năng quan sát (Observability):** Tích hợp sẵn Dashboard tập trung hiển thị Logs (Nhật ký), Metrics (Thông số hiệu năng) và Traces (Truy vết) theo chuẩn OpenTelemetry. Điều này giúp đội ngũ phát triển dễ dàng nhìn thấy luồng đi của một request từ khi vào API đến khi truy xuất Database, giúp việc gỡ lỗi (Debugging) và tối ưu hiệu năng trở nên trực quan hơn bao giờ hết.

0.2 Cơ sở dữ liệu và Lưu trữ

Việc lưu trữ và quản lý dữ liệu đóng vai trò then chốt trong việc đảm bảo tính chính xác của các giao dịch thương mại điện tử cũng như tốc độ phản hồi của ứng dụng. Để giải quyết các yêu cầu về **Tính toàn vẹn dữ liệu** (Data Integrity) và **Hiệu năng truy xuất** (Performance) đã được đề ra tại Mục 2.4, hệ thống sử dụng chiến lược lưu trữ kết hợp giữa Cơ sở dữ liệu quan hệ (RDBMS) và Bộ nhớ đệm (Caching).

0.2.1 Hệ quản trị cơ sở dữ liệu quan hệ: PostgreSQL

Dự án lựa chọn **PostgreSQL 16** làm hệ quản trị cơ sở dữ liệu chính, kết hợp với **Entity Framework Core** đóng vai trò là lớp ORM (Object-Relational Mapping) để tương tác dữ liệu từ mã nguồn .NET [5]. Đây là giải pháp lưu trữ bền vững, được thiết kế để giải quyết triệt để vấn đề toàn vẹn dữ liệu cho tính năng nhạy cảm như thanh toán và quản lý đơn hàng.

Các lý do chính cho quyết định này bao gồm:

- **Đảm bảo tính ACID tuyệt đối:** Đối với ứng dụng giao đồ ăn, đặc biệt là tính năng TeamCart nơi nhiều người cùng thao tác trên một đơn hàng, việc đảm bảo tính Nguyên tố (Atomicity), Nhất quán (Consistency), Cô lập (Isolation) và Bền vững (Durability) là bắt buộc. PostgreSQL nổi tiếng với cơ chế quản lý giao dịch (Transaction) chặt chẽ, giúp ngăn chặn triệt để tình trạng sai lệch dữ liệu tài chính (ví dụ: trừ tiền nhưng không tạo đơn hàng).
- **Bộ tính năng nâng cao tích hợp sẵn:** Điểm vượt trội của PostgreSQL nằm ở hệ sinh thái các tiện ích mở rộng giúp giải quyết trọn vẹn các bài toán nghiệp vụ đặc thù mà không cần phụ thuộc vào dịch vụ bên thứ ba:
  - *Tìm kiếm toàn văn (Full Text Search):* Hệ thống tận dụng engine tìm kiếm tích hợp sẵn để thực hiện chức năng tìm kiếm món ăn và nhà hàng. Với khả năng đánh chỉ mục (indexing) văn bản và hỗ trợ tách từ, giải pháp này đáp ứng tốt nhu cầu tìm kiếm tiếng Việt, giúp tiết kiệm tài nguyên so với việc triển khai các Search Engine rời như ElasticSearch.
  - *Xử lý dữ liệu địa lý (PostGIS):* Đây là tính năng quan trọng để hỗ trợ tìm kiếm theo vị trí. Hệ thống sử dụng PostGIS để thực hiện các truy vấn không gian như "tìm nhà hàng trong bán kính 3km" hay tính toán chính xác khoảng cách đường đi để xác định phí giao hàng, được thực hiện trực tiếp tại tầng cơ sở dữ liệu với tốc độ xử lý rất cao.
  - *Kiểu dữ liệu JSONB:* Cấu trúc Menu với nhiều nhóm Tùy chọn/Topping phức tạp được lưu trữ linh hoạt dưới dạng JSONB. Điều này cho phép hệ thống truy vấn nhanh trên các thuộc tính động mà không bị ràng buộc bởi schema cứng nhắc, kết hợp ưu điểm của NoSQL ngay trong lòng CSDL quan hệ.

So sánh với các giải pháp thay thế:

- **So với MySQL:** Mặc dù MySQL có tốc độ đọc nhanh ở các tác vụ đơn giản, PostgreSQL vượt trội hơn ở khả năng xử lý các truy vấn phức tạp (Complex Queries) cần thiết cho các báo cáo doanh thu. Quan trọng hơn, engine của PostgreSQL hoạt động ổn định hơn trong các kịch bản ghi dữ liệu cường độ cao (High concurrency write) vào giờ cao điểm.
- **So với Microsoft SQL Server:** SQL Server tương thích tốt với .NET nhưng chi phí bản quyền là rào cản lớn khi triển khai thực tế. PostgreSQL là mã nguồn mở (Open Source), giúp tiết kiệm chi phí vận hành mà vẫn cung cấp đầy đủ các tính năng doanh nghiệp, đồng thời chạy tối ưu trên môi trường

Linux giúp giảm chi phí hạ tầng server.

0.2.2 Bộ nhớ đệm (Caching): Redis

Bên cạnh cơ sở dữ liệu chính, hệ thống triển khai **Redis** đóng vai trò là bộ nhớ đệm phân tán (Distributed Cache). Mục tiêu không chỉ là giảm tải cho PostgreSQL mà còn đảm bảo tốc độ xử lý thời gian thực cho các tính năng tương tác cao.

Cụ thể, Redis được ứng dụng trong hai kịch bản trọng yếu:

1. **Lưu trữ dữ liệu tĩnh (Read-heavy):** Các thông tin ít thay đổi như Danh sách nhà hàng và Thực đơn (Menu) được lưu trong cache. Điều này giúp các truy vấn điều hướng cơ bản đạt độ trễ cực thấp (sub-millisecond) mà không cần truy cập Database.
2. **Quản lý trạng thái TeamCart (Write-heavy):** Đây là ứng dụng quan trọng nhất của Redis trong hệ thống. Đối với các Giỏ hàng nhóm đang hoạt động (Active TeamCarts), toàn bộ dữ liệu hiển thị (View Model) được lưu trữ trực tiếp trên RAM của Redis. Khi các thành viên trong nhóm thao tác thêm/bớt món dồn dập, hệ thống đọc/ghi trực tiếp vào Redis thay vì Database. Cơ chế này loại bỏ hoàn toàn độ trễ I/O đĩa cứng, đảm bảo trải nghiệm người dùng mượt mà ngay cả khi cart có lưu lượng hoạt động cao. Dữ liệu chỉ được đồng bộ xuống PostgreSQL để lưu trữ bền vững (Persistence) khi đơn hàng được chốt hoặc kết thúc phiên.

Việc lựa chọn Redis thay vì sử dụng bộ nhớ cục bộ (In-Memory Cache) của server cho phép hệ thống dễ dàng mở rộng nhiều server (Scale-out) mà vẫn đảm bảo tính nhất quán dữ liệu giữa các phiên làm việc.

0.2.3 Quản lý và Phân phối đa phương tiện: Cloudinary

Đặc thù của ứng dụng giao đồ ăn là yêu cầu lưu trữ và hiển thị lượng lớn hình ảnh chất lượng cao (Avatar người dùng, Ảnh món ăn, Banner nhà hàng). Việc lưu trữ trực tiếp các file này trên Server hoặc trong Database sẽ gây áp lực lớn lên băng thông và dung lượng lưu trữ của hệ thống máy chủ.

Giải pháp được lựa chọn là nền tảng quản lý media dựa trên đám mây **Cloudinary** (SaaS) [6], với các ưu điểm kỹ thuật:

- **Tối ưu hóa tự động:** Cloudinary tự động nén và chuyển đổi định dạng ảnh phù hợp với thiết bị của người dùng (ví dụ: tự động chuyển sang định dạng WebP hoặc AVIF giúp giảm 30-50% dung lượng tải so với JPEG/PNG thông thường mà không giảm chất lượng hiển thị).
- **Mạng phân phối nội dung (CDN):** Hình ảnh được phân phối từ hệ thống Server toàn cầu của Cloudinary thay vì tải trực tiếp từ Backend của ứng dụng.

Điều này giúp giảm đáng kể độ trễ hiển thị ảnh trên Mobile App, mang lại trải nghiệm lướt thực đơn mượt mà (Mục 2.1.3).

- **Quản lý tập trung:** Cung cấp API mạnh mẽ để upload và xóa ảnh an toàn từ Backend, tách biệt hoàn toàn tầng lưu trữ file (File Storage Layer) khỏi tầng xử lý nghiệp vụ.

0.3 Xác thực và Phân quyền (Authentication & Authorization)

Bảo mật thông tin người dùng và kiểm soát quyền truy cập là yêu cầu phi chức năng quan trọng hàng đầu (Mục 2.4). Hệ thống YummyZoom áp dụng các tiêu chuẩn an ninh hiện đại để bảo vệ dữ liệu, đồng thời đảm bảo tính tiện lợi cho người dùng cuối.

0.3.1 Cơ chế xác thực: JSON Web Token (JWT)

Hệ thống sử dụng tiêu chuẩn **JSON Web Token (JWT)** (RFC 7519) làm phương thức xác thực chính cho toàn bộ các giao tiếp giữa Client (Mobile App, Web Portal) và Server [7].

Cơ chế hoạt động và lợi ích:

- **Xác thực không trạng thái (Stateless Authentication):** Đây là yếu tố then chốt cho một ứng dụng di động có lượng người dùng lớn. Khác với xác thực dựa trên Session truyền thống (lưu trạng thái đăng nhập trên Server), JWT đóng gói toàn bộ thông tin định danh vào một chuỗi token duy nhất và gửi kèm theo mỗi request HTTP. Server chỉ cần xác thực chữ ký (Signature) của token mà không cần truy vấn lại cơ sở dữ liệu hay bộ nhớ đệm session.
- **Khả năng mở rộng (Scalability):** Nhờ tính chất stateless, hệ thống có thể dễ dàng mở rộng theo chiều ngang (thêm nhiều server xử lý) mà không gặp phải vấn đề đồng bộ session giữa các server (Sticky Session).
- **Hiệu năng:** Giảm tải đáng kể cho hệ thống lưu trữ vì không cần tra cứu session ID cho mỗi yêu cầu người dùng, giúp tối ưu hóa thời gian phản hồi API.

0.3.2 Quản lý định danh và Phân quyền: ASP.NET Core Identity

Để quản lý vòng đời tài khoản và phân quyền, hệ thống tích hợp framework **ASP.NET Core Identity**. Đây là giải pháp bảo mật toàn diện được Microsoft thiết kế sẵn, giúp đội ngũ phát triển tập trung vào logic nghiệp vụ thay vì phải tự xây dựng lại các module an ninh nhạy cảm.

Các chức năng chính được sử dụng:

- **Quản lý người dùng và mật khẩu:** Identity cung cấp sẵn các API an toàn để

đăng ký, đăng nhập. Mọi mật khẩu người dùng đều được băm (hashing) bằng các thuật toán hiện đại (như PBKDF2) kết hợp với Salt ngẫu nhiên trước khi lưu vào PostgreSQL, đảm bảo an toàn ngay cả khi dữ liệu bị rò rỉ.

- **Kiểm soát truy cập dựa trên vai trò (RBAC):** Hệ thống định nghĩa các vai trò (Roles) cụ thể như:
  - *Customer:* Người dùng đặt món, chỉ có quyền truy cập dữ liệu cá nhân.
  - *Restaurant Owner/Staff:* Đối tác nhà hàng, có quyền quản lý thực đơn và xem báo cáo doanh thu của quán mình sở hữu.
  - *Admin:* Quản trị viên hệ thống, có toàn quyền quản lý danh mục và tài khoản hệ thống.

  Identity bảo vệ các API Endpoint bằng cách kiểm tra Role (Claims) có trong JWT, từ chối ngay lập tức các truy cập trái phép.

0.4 Ứng dụng dành cho khách hàng (Mobile App)

Ứng dụng di động là điểm tiếp xúc chính giữa hệ thống và người dùng cuối (Sinh viên, nhân viên văn phòng). Do đó, việc xây dựng một trải nghiệm người dùng (UX) mượt mà, phản hồi nhanh và đồng nhất trên các thiết bị khác nhau là yêu cầu bắt buộc để đáp ứng tiêu chí đã đề ra tại Mục 2.1.3.

0.4.1 Nền tảng phát triển: Flutter & Dart

Dự án lựa chọn **Flutter SDK** sử dụng ngôn ngữ **Dart** để phát triển ứng dụng di động cho khách hàng [8]. Đây là bộ công cụ UI của Google cho phép xây dựng ứng dụng biên dịch natively (native-compiled) cho di động, web và máy tính để bàn từ một mã nguồn duy nhất.

Các lý do chính cho quyết định công nghệ này:

- **Hiệu năng Native:** Khác với các framework lai (hybrid) dựa trên WebView, Flutter sử dụng engine đồ họa riêng (Skia/Impeller) để vẽ trực tiếp lên màn hình, đảm bảo tốc độ khung hình ổn định ở mức 60fps (thậm chí 120fps). Điều này cực kỳ quan trọng đối với các thao tác cuộn danh sách món ăn dài hay các hiệu ứng chuyển động phức tạp trong giao diện đặt món.
- **Kiểm soát giao diện tuyệt đối (Pixel Perfect):** Tính năng *TeamCart* yêu cầu một giao diện chia sẻ trạng thái phức tạp và tùy biến cao (ví dụ: avatar người dùng bay vào giỏ hàng, cập nhật số lượng realtime). Flutter cho phép kiểm soát từng pixel trên màn hình, giúp hiện thực hóa chính xác các thiết kế giao diện độc đáo mà không bị phụ thuộc vào các widget mặc định của hệ điều hành.

- **Tăng tốc độ phát triển:** Tính năng *Hot Reload* của Flutter giúp đội ngũ phát triển xem ngay kết quả thay đổi mã nguồn mà không cần biên dịch lại từ đầu, rút ngắn đáng kể thời gian tinh chỉnh giao diện (UI Polish).

So sánh với các lựa chọn khác:

- **So với React Native:** React Native phụ thuộc vào cầu nối (Bridge) JavaScript để giao tiếp với các module Native, có thể gây nghẽn cổ chai về hiệu năng khi xử lý nhiều tác vụ phức tạp đồng thời. Flutter loại bỏ cầu nối này nhờ biên dịch trước (AOT Compilation), mang lại hiệu năng ổn định hơn.
- **So với Phát triển Native thuần túy (Kotlin/Swift):** Việc phát triển riêng biệt hai ứng dụng Android và iOS đòi hỏi gấp đôi nguồn lực nhân sự và thời gian bảo trì. Flutter giúp tiết kiệm chi phí này nhờ khả năng chia sẻ hơn 90% mã nguồn giữa các nền tảng.

0.4.2 Quản lý trạng thái và Tương tác API

Để xử lý luồng dữ liệu phức tạp, đặc biệt là sự đồng bộ hóa liên tục của Giỏ hàng nhóm, ứng dụng áp dụng mô hình quản lý trạng thái (State Management) chặt chẽ:

- **Quản lý trạng thái với Provider:** Ứng dụng lựa chọn sử dụng thư viện **Provider** để quản lý trạng thái. Giải pháp này được đánh giá là phù hợp nhất với quy mô hiện tại của dự án, chưa quá lớn nhưng vẫn đòi hỏi sự tổ chức code bài bản. Provider mang lại hiệu quả vượt trội so với việc sử dụng biến trạng thái thông thường (`setState`) nhờ khả năng tách biệt logic nghiệp vụ khỏi giao diện (UI) và giải quyết triệt để vấn đề truyền dữ liệu qua nhiều cấp (prop drilling), đồng thời giữ cho cấu trúc dự án gọn nhẹ, tránh sự phức tạp không cần thiết của các mô hình như BLoC.
- **Giao tiếp mạng với Dio:** Thư viện Dio được sử dụng để quản lý các kịch bản gọi API nâng cao như tự động hủy request (Cancellation), xử lý timeout và chặn bắt tập trung (Interceptors), đảm bảo ứng dụng luôn phản hồi chính xác tình trạng kết nối mạng cho người dùng.

0.4.3 Tích hợp bản đồ và Thanh toán

- **Bản đồ số:** Hệ thống tích hợp **Mapbox SDK** để thực hiện các chức năng định vị và bản đồ. Mapbox được ưu tiên lựa chọn nhờ sự hỗ trợ mạnh mẽ cho nền tảng Flutter giúp việc tích hợp mượt mà, đồng thời cung cấp hạn mức sử dụng miễn phí lớn cho nhà phát triển, giúp tối ưu chi phí xây dựng dự án.
- **Cổng thanh toán:** Ứng dụng tích hợp cổng thanh toán **Stripe** thông qua SDK chính thức. Stripe giúp đơn giản hóa quá trình kết nối với Backend, đồng thời

cung cấp môi trường Sandbox và Test Mode (Chế độ kiểm thử) toàn diện. Điều này cho phép đội ngũ phát triển mô phỏng các kịch bản thanh toán thành công, thất bại, hoặc hoàn tiền một cách dễ dàng và an toàn trước khi triển khai thực tế.

0.5 Ứng dụng Web quản trị (Admin & Restaurant Portal)

Đáp ứng nhu cầu quản lý của **Đối tác nhà hàng** đã được xác định trong Mục 2.1.3, hệ thống cung cấp một cổng thông tin quản trị tập trung. Đây là nơi các chủ nhà hàng thực hiện các thao tác quản lý thực đơn, theo dõi đơn hàng và xem báo cáo doanh thu.

0.5.1 Nền tảng Frontend: Angular

Dự án lựa chọn **Angular** (phiên bản 21 mới nhất) kết hợp với ngôn ngữ **TypeScript** để xây dựng ứng dụng quản trị với công nghệ Single Page Application (SPA) [9].

Các lý do chính cho quyết định này:

- **Cấu trúc Module chặt chẽ:** Khác với các thư viện tự do như ReactJS, Angular cung cấp một khung làm việc (Framework) hoàn chỉnh với kiến trúc hướng module (NgModules) và quản lý phụ thuộc (Dependency Injection) rõ ràng. Điều này rất phù hợp với tư duy lập trình hướng đối tượng (OOP) được sử dụng ở Backend, giúp đội ngũ phát triển dễ dàng đồng bộ kiến thức giữa hai nền tảng.
- **Tích hợp toàn diện:** Angular đi kèm sẵn với các công cụ mạnh mẽ như HttpClient (giao tiếp API), Reactive Forms (xử lý biểu mẫu phức tạp) và Router (điều hướng). Điều này giúp giảm thiểu việc phụ thuộc vào các thư viện bên thứ ba rời rạc, đảm bảo tính ổn định và bảo mật cho ứng dụng quản trị doanh nghiệp.

So sánh với các lựa chọn khác:

- **So với ReactJS:** React chỉ là một thư viện UI, đòi hỏi lập trình viên phải tự lựa chọn và cấu hình thêm nhiều thư viện hỗ trợ (State management, Routing, Form...). Điều này tạo ra sự linh hoạt nhưng cũng làm tăng độ phức tạp khi cần chuẩn hóa cấu trúc dự án.
- **So với VueJS:** Mặc dù VueJS dễ học, nhưng cộng đồng doanh nghiệp (Enterprise) thường ưu tiên Angular hơn cho các hệ thống quản trị lớn nhờ tính chặt chẽ và khả năng bảo trì mã nguồn tốt hơn về lâu dài.

0.5.2 Thư viện giao diện: PrimeNG và Tailwind CSS

Để tối ưu hóa thời gian phát triển giao diện nhưng vẫn đảm bảo tính thẩm mỹ và chuyên nghiệp, dự án sử dụng thư viện **PrimeNG** kết hợp với **Tailwind CSS** [10].

- **PrimeNG:** Cung cấp bộ sưu tập khổng lồ các thành phần UI cao cấp (UI Components) dành riêng cho các ứng dụng quản trị dữ liệu, đặc biệt là các bảng dữ liệu (Data Grid) với khả năng phân trang, lọc, sắp xếp (Sorting/Filtering) và các biểu đồ thống kê (Charts) trực quan.
- **Tailwind CSS:** Framework CSS dạng tiện ích (Utility-first) giúp tùy biến nhanh giao diện, bố cục (Layout) và Responsive mà không cần viết quá nhiều mã CSS thủ công.

0.6 Cộng tác thời gian thực (Realtime Collaboration)

Đây là trái tim của tính năng **TeamCart**, đáp ứng yêu cầu kỹ thuật khắt khe về độ trễ thấp (dưới 500ms) được đặt ra tại Mục 2.4 để đảm bảo trải nghiệm đồng bộ mượt mà giữa các thành viên.

0.6.1 Giao thức và Framework: SignalR

Hệ thống sử dụng thư viện **ASP.NET Core SignalR**, tận dụng giao thức **WebSockets** làm phương thức vận chuyển chính (Transport) [11].

Vai trò của SignalR trong hệ thống:

- **Đồng bộ TeamCart tức thời:** Khi một thành viên trong nhóm thực hiện hành động (ví dụ: Thêm món "Trà sữa" vào giỏ), SignalR Server sẽ ngay lập tức đẩy (push) thông tin cập nhật xuống tất cả các thiết bị của thành viên khác trong cùng nhóm. Quá trình này diễn ra gần như tức thời, tạo cảm giác mọi người đang cùng thao tác trên một màn hình chung.
- **Thông báo đơn hàng mới:** Giúp ứng dụng Web của nhà hàng nhận thông báo đơn hàng mới (Real-time Notification) mà không cần tải lại trang (F5), giúp quy trình tiếp nhận đơn hàng diễn ra nhanh chóng.

Lý do lựa chọn SignalR so với các giải pháp khác:

- **So với Polling (Hỏi định kỳ):** Kỹ thuật Polling yêu cầu Client liên tục gửi request lên Server (ví dụ mỗi 5 giây) để kiểm tra dữ liệu mới. Điều này gây lãng phí băng thông mạng và tài nguyên máy chủ khủng khiếp khi số lượng người dùng tăng cao. SignalR sử dụng kết nối bền vững (Persistent Connection), chỉ gửi dữ liệu khi thực sự có thay đổi.
- **So với Firebase Realtime Database:** Mặc dù Firebase rất mạnh, việc phụ thuộc hoàn toàn vào dịch vụ bên thứ ba (BaaS) có thể dẫn đến chi phí vận

hành tăng vọt khi quy mô dữ liệu lớn. SignalR được tích hợp sẵn miễn phí trong ASP.NET Core, cho phép nhóm tự chủ hoàn toàn về hạ tầng và dữ liệu.

- **Cơ chế tự động điều phối (Auto-negotiation):** SignalR thông minh tự động lựa chọn phương thức kết nối tốt nhất mà Client và Server hỗ trợ (ưu tiên WebSockets, nếu không hỗ trợ sẽ tự chuyển sang Server-Sent Events hoặc Long Polling), đảm bảo ứng dụng hoạt động ổn định trên mọi môi trường mạng.

0.6.2 Thông báo đẩy tới thiết bị di động: Firebase Cloud Messaging (FCM)

Bên cạnh kết nối thời gian thực bằng SignalR, hệ thống tích hợp dịch vụ **Firebase Cloud Messaging (FCM)** [12] để giải quyết bài toán gửi thông báo đến người dùng ngay cả khi ứng dụng đang chạy nền (Background) hoặc đã tắt hoàn toàn.

Vai trò của FCM trong hệ thống:

- **Cập nhật trạng thái đơn hàng:** Khách hàng sẽ nhận được thông báo ngay lập tức khi đơn hàng chuyển đổi trạng thái (ví dụ: Nhà hàng đã xác nhận, Tài xế đang giao). Việc này giúp người dùng an tâm mà không cần mở ứng dụng liên tục để kiểm tra.
- **Tương tác TeamCart:** Gửi lời mời tham gia nhóm hoặc thông báo khi "Chủ phòng" chốt đơn đến các thành viên khác.

Lý do kết hợp FCM cùng SignalR: Mặc dù SignalR rất mạnh trong việc đồng bộ dữ liệu trực tuyến (Online), nhưng kết nối WebSockets sẽ bị ngắt khi người dùng tắt màn hình điện thoại hoặc chuyển sang ứng dụng khác để tiết kiệm pin. FCM là giải pháp bổ trợ hoàn hảo nhờ sử dụng cơ chế của hệ điều hành (APNs trên iOS và GCM trên Android) để đánh thức thiết bị và hiển thị thông báo, đảm bảo thông tin quan trọng luôn đến được tay người dùng.