ỦY BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH

**TRƯỜNG ĐẠI HỌC SÀI GÒN**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỀ TÀI : FAST FOOD DELIVERY DÙNG CI/CD**



Giảng viên hướng dẫn : TS. Nguyễn Quốc Huy

Sinh viên thực hiện: Trần Nguyễn Phúc Mạnh 312241121

Nguyễn Vũ Quang Minh 3122411126

Lớp : DCT122C4

**Thành phố Hồ Chí Minh, tháng 9 năm 2025**

[**PRD: Hệ Thống Đặt Hàng và Giao Hàng Thức Ăn Bằng Drone**](#_plsaugxyhdpo) **3**

[Problem Alignment](#_5399bnsqxnlc) 3

[High Level Approach](#_obnorehurr41) 4

[Narrative](#_4n4p6084g1mu) 4

[Goals](#_dpy7by41t0wx) 4

[Solution Alignment](#_lvexielnk2ms) 5

[**DANH SÁCH CHỨC NĂNG FOOD FAST DRONE DELIVERY**](#_se71ufrlq1iw) **6**

[CHỨC NĂNG KHÁCH HÀNG (Client App)](#_jq8rgl22slnf) 6

[CHỨC NĂNG NHÀ HÀNG (Restaurant App)](#_pa4gxvftafet) 6

[CHỨC NĂNG QUẢN TRỊ VIÊN (Admin App)](#_1eutv16tt2mc) 6

[Key Flows](#_ppwyiv4lddgb) 7

[Operational Flow](#_kg8w1bhvb4l8) 8

[**FOCUSED USE CASE AND ACTIVITY DIAGRAM**](#_vcetf69rslxg) **9**

[UC1: Đặt hàng](#_tudpj18hkyby) 9

[UC2: Quản lý danh mục của từng nhà hàng](#_em56l5nv0ozg) 13

[Sequence Diagram](#_kto9zlmpbsap) 17

[Đăng nhập](#_72nm5jx5h3si) 17

[Đăng ký](#_b4txhxnwyje) 18

[Đặt hàng và thanh toán](#_g6ejt7ezffw8) 19

[Quản lý món ăn nhà hàng](#_alwaqvdlkoq7) 20

[**Giao diện WEB FAST FOOD DELIVERY BẰNG DRONE**](#_9mss1yn1geei) **21**

[**Kịch bản test – FastFood Delivery bằng Drone**](#_dfxg118jp4vy) **26**

[**API ENDPOINT**](#_mlr954z4e94m) **27**

[**Công nghệ sử dụng**](#_m974ewndti7b) **30**

[Frontend](#_yanef15fii86) 30

[Backend](#_1bku31dl2921) 31

[CI/CD](#_ntt1yyra59kp) 31

[Testing & Dev Tooling](#_j5u4ny7mj9jk) 31

[Component diagram:](#_y8ss7hxrnoru) 32

[Deployment Diagram](#_94v5k2ao10l8) 33

[ERD](#_clixh4t5h5m1) 33

# **PRD: Hệ Thống Đặt Hàng và Giao Hàng Thức Ăn Bằng Drone**

## **Problem Alignment**

● Khách hàng phải chờ đợi thời gian giao hàng dài (30-60 phút) do tắc đường và khoảng cách xa nhà hàng.  
● Chi phí giao hàng cao do cần shipper và chi phí nhiên liệu, ảnh hưởng đến lợi nhuận nhà hàng.  
● Phạm vi giao hàng bị hạn chế bởi giao thông và khả năng di chuyển của shipper truyền thống.  
● Khách hàng thiếu tính minh bạch trong việc theo dõi đơn hàng real-time và không biết chính xác thời gian giao hàng.

## **High Level Approach**

● Xây dựng web application tích hợp với hệ thống quản lý drone tự động cho giao hàng thức ăn.  
● Tích hợp GPS và IoT để theo dõi real-time vị trí drone và tối ưu hóa route giao hàng.  
● Sử dụng AI để tự động phân bổ drone, dự đoán thời gian giao hàng và tối ưu hóa logistics.  
● Tích hợp multiple payment gateways và authentication system để đảm bảo giao dịch an toàn.

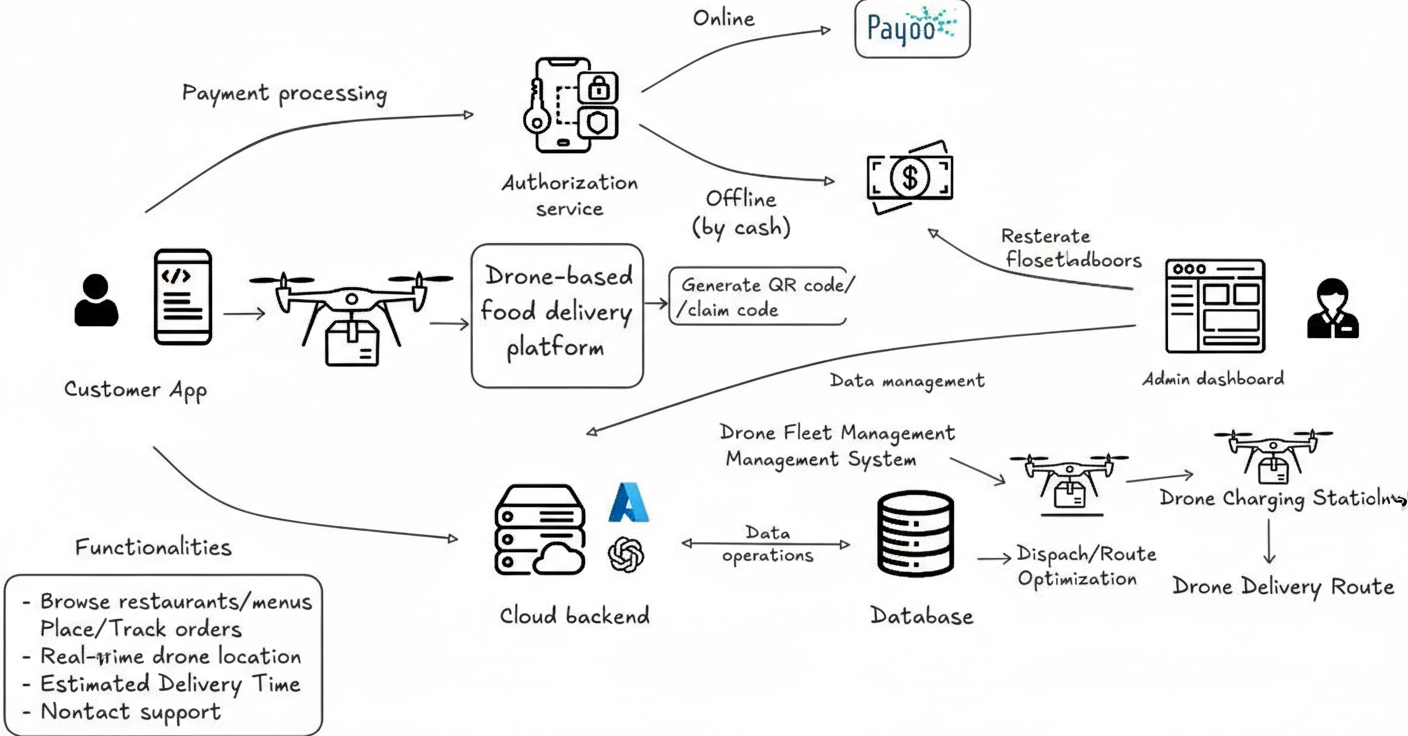
## **Narrative**

● Khách hàng truy cập web app, chọn nhà hàng và đặt món ăn yêu thích với thông tin địa chỉ giao hàng chi tiết.  
● Sau khi thanh toán online, hệ thống tự động assign drone gần nhất và gửi thông báo đến nhà hàng.  
● Nhà hàng chuẩn bị thức ăn, đóng gói phù hợp với drone payload requirements và đặt vào cargo box.  
● Drone tự động bay đến nhà hàng pickup, sau đó bay theo route tối ưu đến địa chỉ khách hàng.  
● Khách hàng nhận OTP, xác thực để drone thả hàng tại vị trí đã chỉ định và hoàn tất giao dịch.

## **Goals**

● Giảm thời gian giao hàng xuống dưới 15 phút trong bán kính 5km từ nhà hàng.  
● Cắt giảm chi phí giao hàng 40% so với phương thức shipper truyền thống.  
● Mở rộng phạm vi phục vụ đến các khu vực khó tiếp cận hoặc xa trung tâm.  
● Tạo trải nghiệm khách hàng độc đáo với công nghệ giao hàng tự động hiện đại.  
● Đảm bảo độ chính xác giao hàng 95% và customer satisfaction rating trên 4.5/5.

## **Solution Alignment**



Customer App: Dành cho khách hàng (đặt hàng, theo dõi đơn, nhận thông báo)

Backend Server:

* API Gateway quản lý giao tiếp giữa các microservice.
* Các service chính:
  + User Service: Quản lý thông tin người dùng.
  + Product/Restaurant Service: Quản lý thực đơn và cửa hàng.
  + Order Service: Quản lý đơn hàng và trạng thái.
  + Payment Service: Quản lý thanh toán (qua ví điện tử hoặc thẻ).
  + Drone Control Service: Kết nối và điều phối hệ thống drone.
* CSDL: PostgreSQL (dữ liệu giao dịch), MongoDB (dữ liệu drone và vị trí).

Drone Delivery System:

* Drone Control Station: Điều phối, giám sát và giao tiếp với drone.
* Drone: Thiết bị bay thực hiện giao hàng.

# **DANH SÁCH CHỨC NĂNG FOOD FAST DRONE DELIVERY**

## **CHỨC NĂNG KHÁCH HÀNG (Client App)**

* Đăng ký/Đăng nhập: Tạo tài khoản, đăng nhập với 3 roles (user, owner, admin)
* Duyệt nhà hàng: Xem danh sách nhà hàng theo thành phố
* Chọn món ăn: Xem chi tiết món ăn, thêm vào giỏ hàng
* Giỏ hàng: Quản lý sản phẩm trong giỏ, tính tổng tiền
* Thanh toán: Hỗ trợ COD và MoMo, thanh toán nhiều đơn hàng
* Theo dõi đơn hàng: Real-time status updates, vị trí drone
* Xác nhận nhận hàng: OTP verification khi drone giao hàng
* Đánh giá: Rating sau khi nhận hàng
* Quản lý tài khoản: Cập nhật thông tin cá nhân, lịch sử đơn hàng

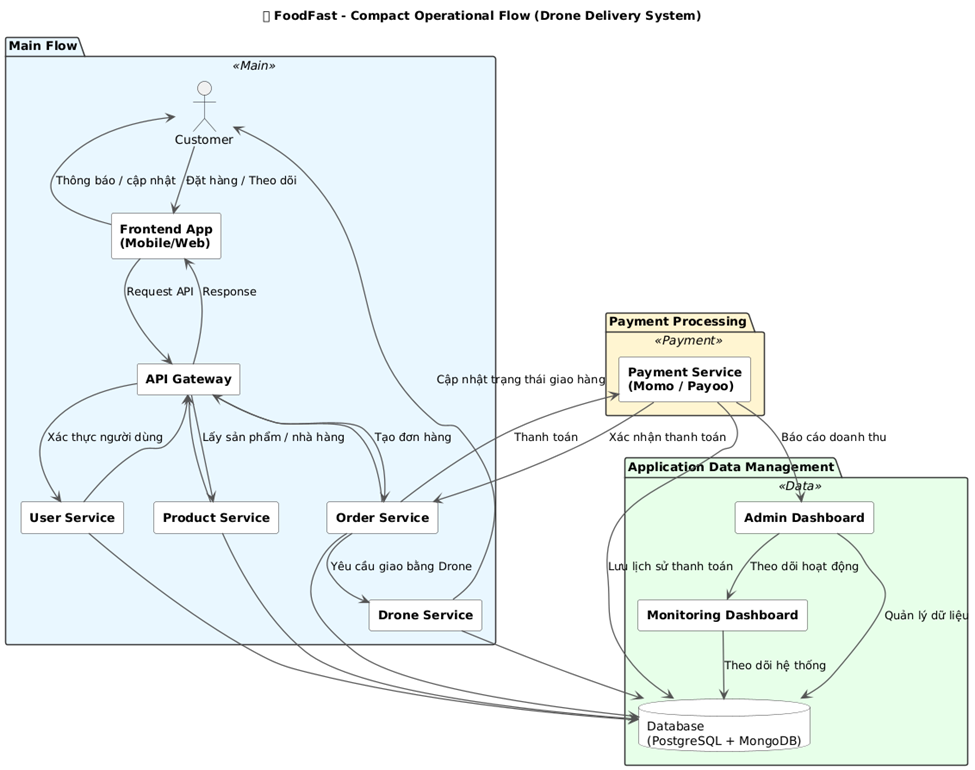
## **CHỨC NĂNG NHÀ HÀNG (Restaurant App)**

* Tạo/Sửa nhà hàng: Quản lý thông tin cửa hàng (tên, địa chỉ, hình ảnh)
* Quản lý sản phẩm: Thêm/sửa/xóa món ăn, cập nhật giá, danh mục
* Quản lý drone: Xem danh sách 5 drone cố định, cập nhật trạng thái
* Quản lý đơn hàng: Xem đơn hàng của cửa hàng, cập nhật trạng thái
* Thông báo: Nhận thông báo khi drone được assign, chuẩn bị hàng
* Phân công drone: Tự động hoặc thủ công assign drone available
* Báo cáo: Thống kê đơn hàng, doanh thu, hiệu suất

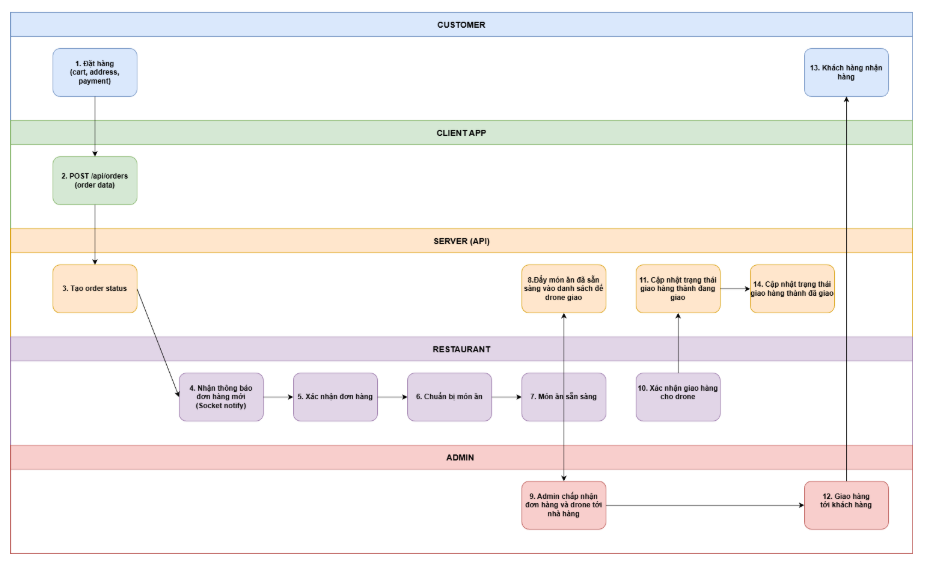
## **CHỨC NĂNG QUẢN TRỊ VIÊN (Admin App)**

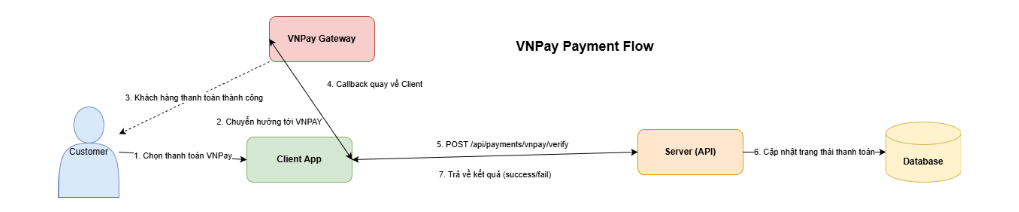
* Quản lý người dùng: Xem tất cả users, cập nhật trạng thái (active/inactive)
* Quản lý nhà hàng: Xem tất cả shops, phê duyệt hoặc vô hiệu hóa
* Quản lý đơn hàng: Xem tất cả orders, can thiệp khi cần
* Quản lý drone: Giám sát toàn bộ fleet, cập nhật maintenance
* Thanh toán: Giám sát payments, xử lý disputes
* Báo cáo tổng quan: Dashboard với metrics (doanh thu, số đơn, user activity)
* Hệ thống: Monitoring health, logs, performance analytics
* Cấu hình: Quản lý settings, payment gateways, drone parameters

# Key Flows

****

# Operational Flow

****

****

# FOCUSED USE CASE AND ACTIVITY DIAGRAM

## UC1: Đăng nhâp

## 

|  | |
| --- | --- |
| Số thứ tự Use Case | 1 |
| Tên Use Case | Đăng nhập |
| Actor(s) | Quản lý |
| Maturity | Focused |
| Tóm tắt | Cho phép người dùng đăng nhập vào trang web để thực hiện chức năng của mình. |
| Điều kiện kích hoạt | Chọn nút "Sign in" từ trang Odoo. |
| Tiền điều kiện | 1. Tài khoản đã được đăng ký/tạo trước đó.  2. Phải chạy server ở PyCharm. |
| Hậu điều kiện | Người dùng đăng nhập vào ứng dụng thành công. |
| Giả định | Tài khoản đăng ký Odoo là tài khoản Admin. |
| Luồng cơ bản (Basic Flow) | Actor (Quản lý)  1. Use case bắt đầu khi đăng nhập vào Trang chủ Odoo.  3. Người dùng nhập tên tài khoản và mật khẩu. (Tham chiếu **A1, A2**)  Hệ thống  2. Hệ thống hiển thị màn hình đăng nhập.  4. Hệ thống ghi nhận hoạt động đăng nhập thành công vào Odoo. (Tham chiếu **E1**) |
| Luồng thay thế (Alternative Flow) | A1: Người dùng chưa có tài khoản  1. (Actor) Người dùng chọn "Tạo account Odoo".  2. (Hệ thống) Hệ thống sẽ chuyển hướng sang trang Web có form "Tạo account".  A2: Người dùng quên mật khẩu  1. (Actor) Người dùng chọn "Đặt lại mật khẩu".  2. (Hệ thống) Hệ thống sẽ chuyển hướng sang trang Web có [form] "Đặt lại mật khẩu". |
| Luồng ngoại lệ (Exception Flow) | E1: Đăng nhập không thành công  Tại bước 3 của Luồng cơ bản {Nhập vào tài khoản, mật khẩu}, nếu hệ thống xác thực thông tin đăng nhập không thành công, hệ thống sẽ hiển thị thông báo [lỗi]. |
| Business Rules | Không |
| Điểm mở rộng | Không |

## 

## UC2: Đặt hàng

|  | | |
| --- | --- | --- |
| **Use case number** | UC-001 | |
| **Use case name** | Đặt hàng | |
| **Actor** | Khách hàng,Cổng thanh toán (VNPAY),Nhà hàng , hệ thống giao hàng (DRONE) | |
| **Maturity** | Focused | |
| **Summary** | Khách hàng tạo và hoàn tất một đơn hàng gồm các món từ một nhà hàng, thanh toán và nhận xác nhận đơn (sẵn sàng để xử lý/tạo lộ trình giao hàng). | |
| **Basic Create Order of Event** | **Actor Action** | **System Response** |
| [1.](http://1.nh)Thực hiện tính năng đăng nhập |  |
| 2. Chọn món và nhập số lượng muốn mua |  |
| 3.Bấm thanh toán |  |
|  | 4. Hiển thị thông tin tổng tiền, phí giao hàng, thời gian ước tính và form thanh toán |
| 5[.Nh](http://4.nh)ập thông tin cá nhân ,địa chỉ |  |
|  | 6.Hệ thống kiểm tra thông tin vừa nhập **(E1)** |
| [7.Ch](http://7.ch)ọn COD **(A1,A2)** |  |
|  | 8.Hệ thống tính toán giá tiền subtotal +tiền ship |
|  |  | 9. Thông báo cho người dùng giá tiền , vị trí drone (A3) |
|  |  | [10.Th](http://10.th)ông báo đặt hàng thành công và đợi xử lý từ nhà hàng và admin |
| **Alternative Paths** | **A1. Thanh toán qua MOMO** | |
| **A1.1.**Tại bước 7 người dùng chọn MoMo |  |
|  | A1.2.Hệ thống hiển thị form nhập thông tin thanh toán MoMo. |
| [A1.3.Ng](http://a1.3.ng)ười dùng nhập thông tin **(E2)** |  |
|  | A1.4.Hệ thống gửi yêu cầu đến MOMO để xử lý. |
|  | A1.5. Nếu giao dịch thành công, hệ thống hiển thị tổng tiền, thông báo đặt hàng thành công (chuyển về bước 10). |
| **A2. Thanh toán qua MOMO** | |
| A2.1. Tại bước 7, người dùng chọn **MOMO**. |  |
|  | A2.2. Hệ thống hiển thị giao diện nhập thông tin thanh toán MOMO. |
| A2.3. Người dùng nhập thông tin, xác nhận thanh toán **(E3)** |  |
|  | A2.4. Hệ thống xử lý giao dịch MOMO. |
|  | A2.5. Nếu thanh toán thành công, hệ thống quay lại bước 10. |
| **A3: Sử dụng Voucher** | |
| A3.1. Tại bước 6, người dùng nhập mã voucher. |  |
|  | A3.2. Hệ thống kiểm tra tính hợp lệ của mã.  (E4) |
|  | A3.3. Nếu hợp lệ, hệ thống áp dụng giảm giá và hiển thị lại tổng tiền. |
|  | A3.4. Quay lại bước 9. |
| **Exception Paths** | [E1.Ki](http://e1.ki)ểm tra thông tin vừa nhập nếu đúng thì tiếp tục . Nếu sai thì quay về bước 5  [E2.Ki](http://e2.ki)ểm tra thông tin vừa nhập của VNPAY. Nếu thỏa điều kiện thì tiếp tục quy trình . Còn không thì quay về bước A1.3  [E3.Ki](http://e3.ki)ểm tra thông tin nhập của MOMO . Nếu thỏa điều kiện thì tiếp tục quy trình . Còn không thì quay về bước A2.3 | |

## **UC3: Quản lý danh mục của từng nhà hàng**

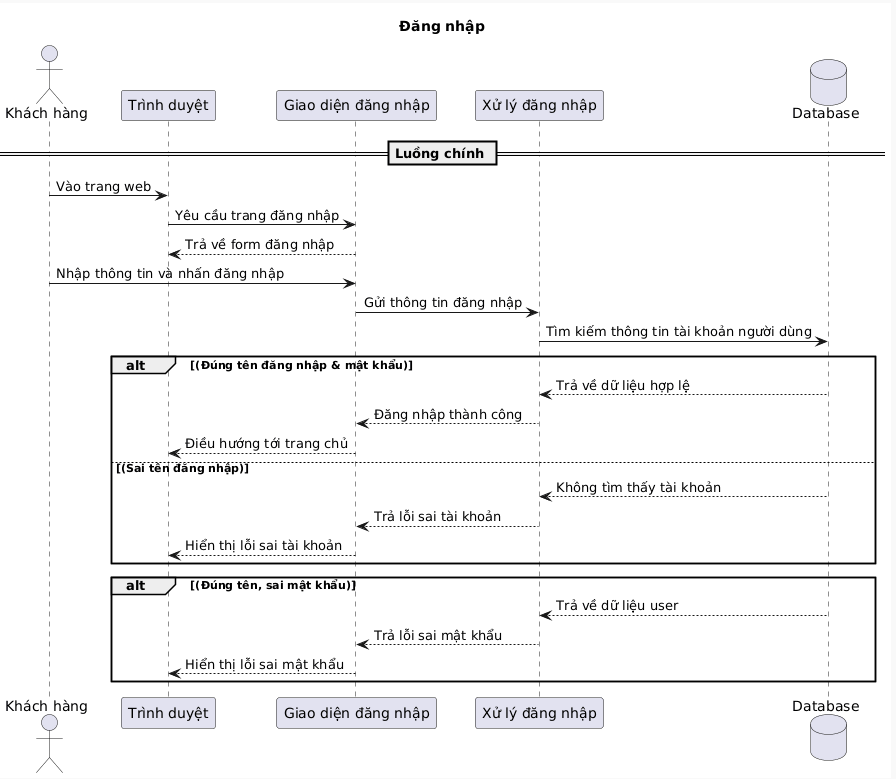
|  | | |
| --- | --- | --- |
| **Use case number** | UC-004 | |
| **Use case name** | Quản lý danh mục của từng nhà hàng | |
| **Actor** | Quản trị viên nhà hàng (Restaurant Admin) | |
| **Maturity** | Focused | |
| **Summary** |  | |
| **Basic Create Order of Event** | **Actor Action** | **System Response** |
| [1.Th](http://1.th)ực hiện chức năng đăng nhập |  |
| [2.Ch](http://2.ch)ọn quản lý thực đơn |  |
| [3.Ch](http://3.ch)ọn thêm **(A1,A2,A3,A4)** |  |
|  | 4.Hệ thống hiển thị giao diện để nhập |
| [**5**.Nh](http://5.nh)ập thông tin **(E1)** |  |
|  | 6.Hệ thống thông bao thành công và cập nhật vào data |
| **Alternative Paths** | **A1. Sửa** | |
| A1.1. Tại bước 3, Quản trị viên chọn **Sửa**. |  |
|  | A1.2. Hệ thống hiển thị danh sách các món hiện có. |
| A1.3. Quản trị viên tìm kiếm món ăn cần chỉnh sửa và nhập thông tin mới. |  |
|  | A1.4. Hệ thống kiểm tra dữ liệu và cập nhật món ăn. **(E2)** |
|  | A1.5. Hệ thống hiển thị thông báo “Cập nhật thành công” và làm mới danh sách thực đơn. |
| **A2.Xóa món trong thực đơn** | |
| A2.1. Tại bước 3, Quản trị viên chọn Xóa. |  |
|  | A2.2. Hệ thống hiển thị danh sách món ăn hiện có. |
| A2.3. Quản trị viên chọn món ăn muốn xóa. |  |
|  | A2.4. Hệ thống hiển thị yêu cầu xác nhận “Bạn có chắc muốn xóa món này không?”. **(A5)** |
| A2.5 Chọn có , hệ thống xóa món và hiển thị thông báo “Xóa thành công” |  |
| **A3.Xem thực đơn** | |
| A3.1. Tại bước 3, Quản trị viên chọn **Xem**. |  |
|  | A3.2. Hệ thống hiển thị form tìm kiếm món ăn. |
| A3.3. Quản trị viên nhập tên món cần xem. |  |
|  | A3.4. Hệ thống kiểm tra kết quả tìm kiếm ( E4) |
|  | A3.5. Hệ thống hiển thị kết quả tìm kiếm , thông tin chi tiết món ăn |
| **Exception Paths** | [E1.Ki](http://e1.ki)ểm tra thông tin vừa thêm có hợp với cấu trúc data không . Nếu thỏa thì đi tới bước 4. Còn không thỏa thì quay lại bước 3 để nhập lại và thông báo cho người dùng  E2.Hệ thống thực hiện tìm kiếm món ăn đã được nhập . Nếu có thì tới bước A1.5 .Nếu không thì tới bước A1.3 để nhập lại và hiển thị thông báo | |

## UC4: Quản lý Drone

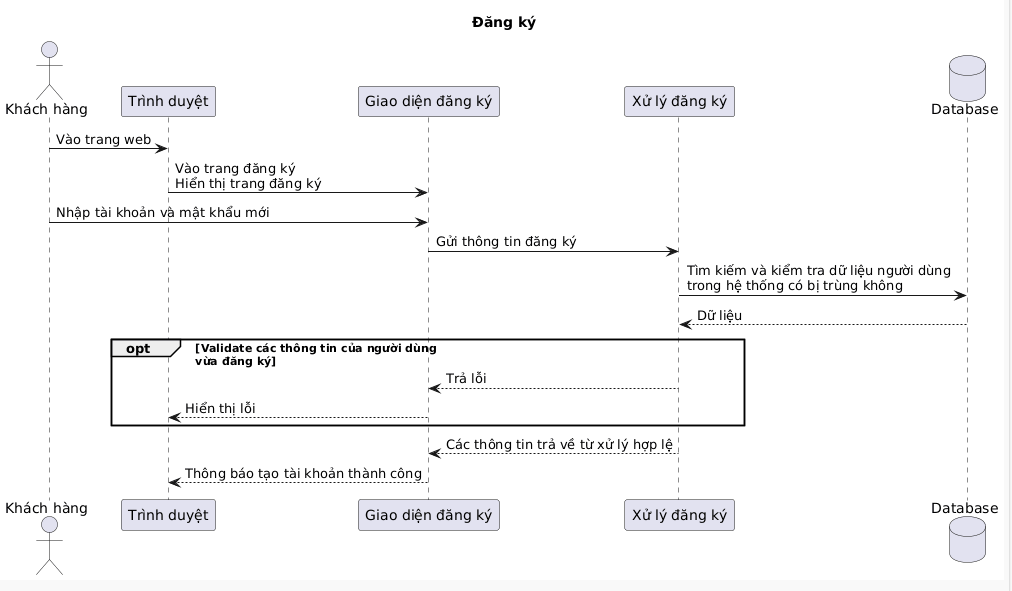
|  | |
| --- | --- |
| **Use Case Number** | UC-04 |
| **Use Case Name** | Quản lý Drone |
| **Actor** | Restaurant Manager |
| **Maturity** | Mature |
| **Basic Flow of Event**  (Luồng sự kiện cơ bản) | 1. Nhà hàng đăng nhập vào hệ thống.  2. Chọn mục "Quản lý Drone".  3. Thực hiện các thao tác: Thêm, sửa, xóa, xem danh sách drone.  4. Hệ thống lưu thông tin vào cơ sở dữ liệu.  5. Hiển thị kết quả cho nhà hàng. |
| **Alternative Paths**  (Luồng thay thế) | 1. Nhà hàng tìm kiếm drone theo trạng thái (đang hoạt động, đang bảo trì).  2. Nhà hàng lọc drone theo khu vực hoạt động. |
| **Exception Paths**  (Luồng ngoại lệ) | 1. Kết nối cơ sở dữ liệu thất bại.  2. Drone không tồn tại trong hệ thống.  3. Lỗi xác thực tài khoản. |

## Sequence Diagram

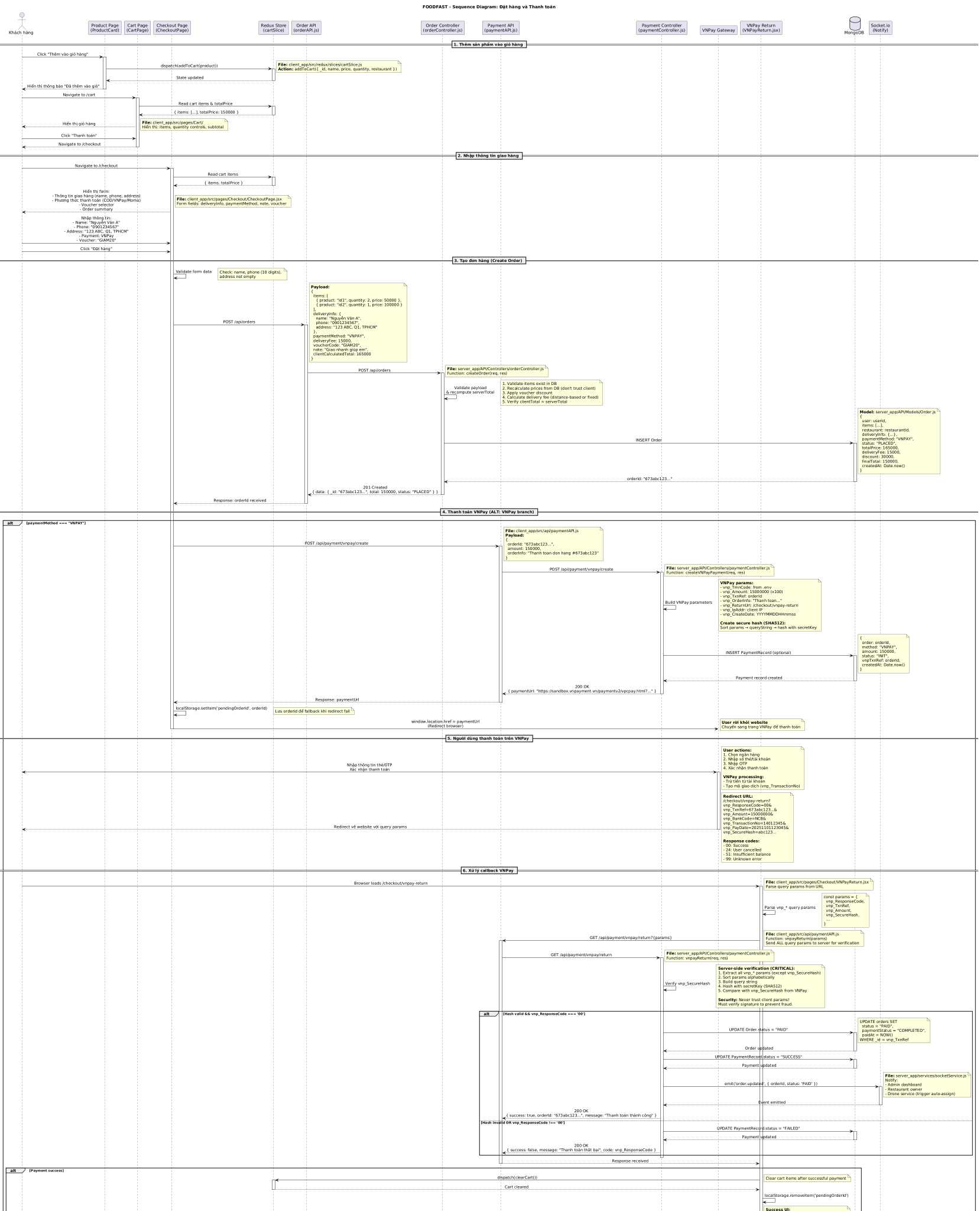
## **Đăng nhập**



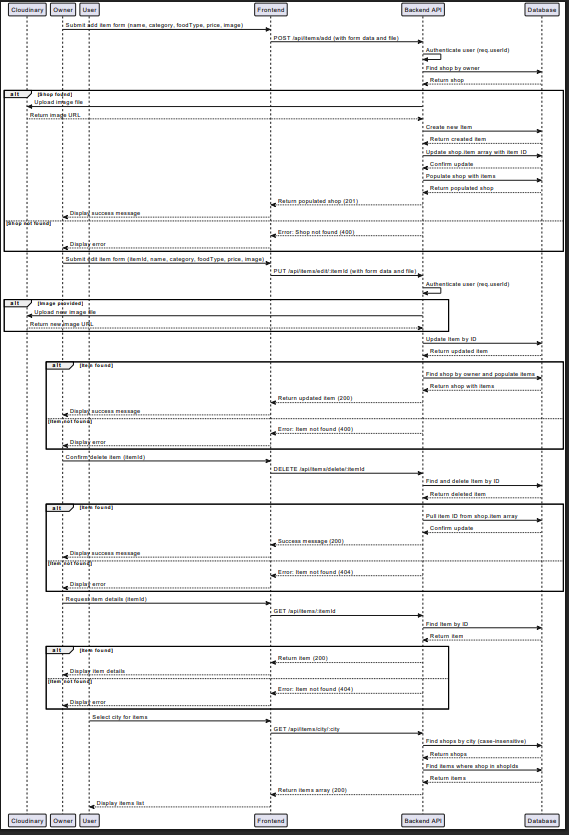
## **Đăng ký**

****

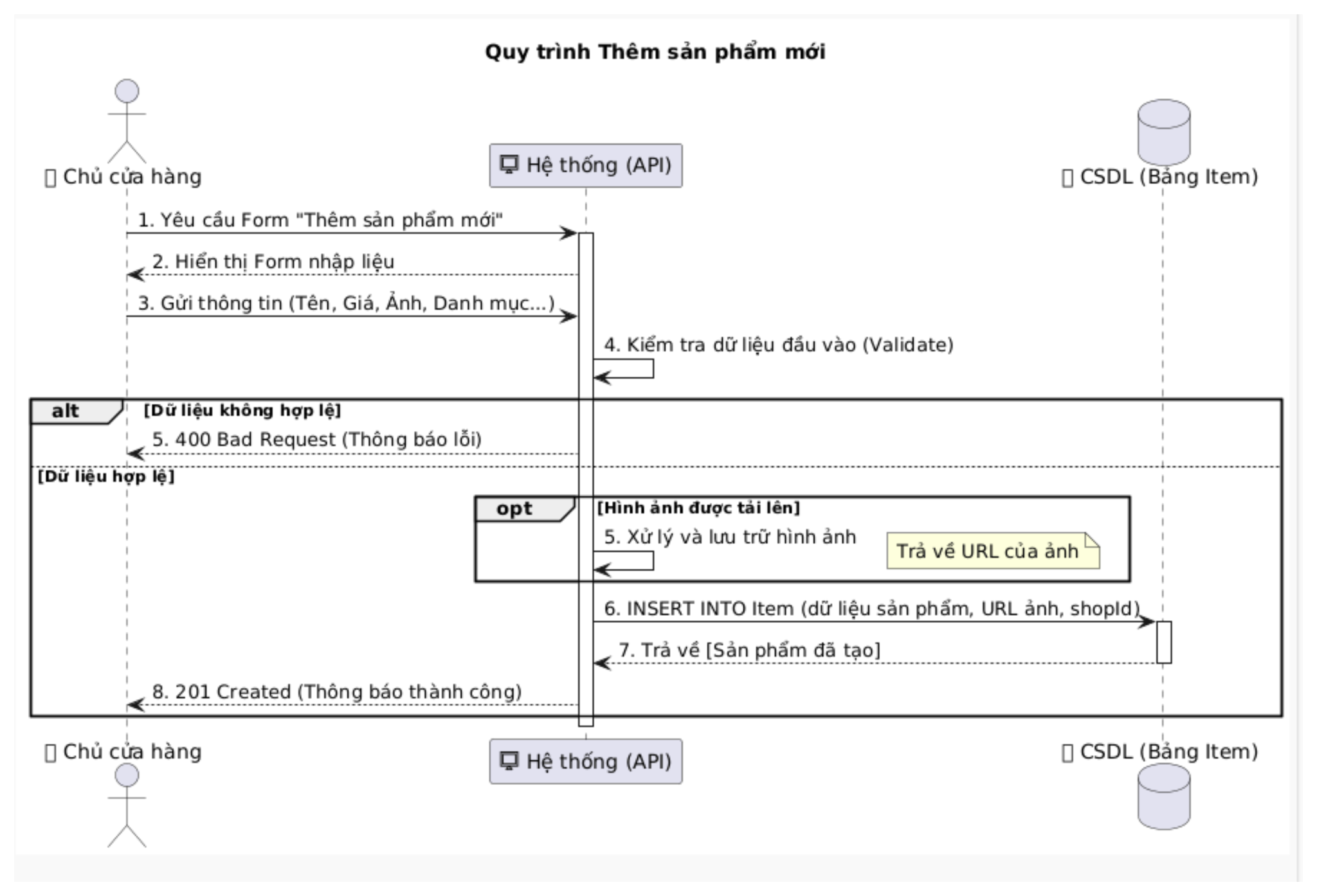
## **Đặt hàng và thanh toán**

****

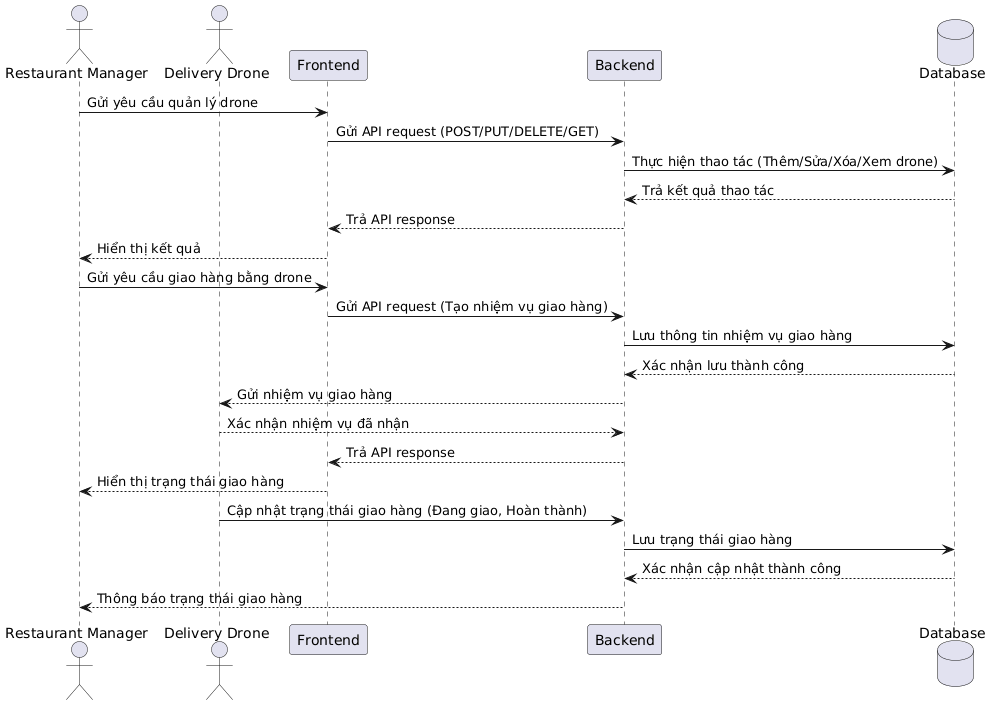
## **Quản lý món ăn nhà hàng**



## Quản lý thêm món ăn



## Quản Lý Drone



# Giao diện WEB FAST FOOD DELIVERY BẰNG DRONE

| Chức năng | UI |
| --- | --- |
| Đăng nhập |  |
| Đăng kí |  |
| Danh sách nhà Hàng |  |
| Xem Menu Nhà hàng |  |
| Quản lý giỏ hàng |  |
| Thanh Toán |  |
| Theo dõi đơn Hàng |  |
| Đăng kí |  |
| Đăng nhập nhà hàng |  |
| Thêm món ăn |  |
| Quản lý đơn hàng |  |
| Phân Công drone |  |

# Kịch bản test – FastFood Delivery bằng Drone

Kịch bản Test:

**Scenario 1 – Đặt món và thanh toán**

TC-S1-01: (Thành công) Đặt hàng và thanh toán (MoMo/VNPay)

Hành động: Người dùng chọn món, nhấn "Đặt hàng" (chọn MoMo), và hoàn tất thanh toán thành công trên cổng thanh toán (VNPay).

KQ Mong đợi (Backend): POST /api/orders tạo đơn (status: 'pending'), trả URL thanh toán. restaurant\_app nhận event new-order. Nhận IPN, cập nhật order.paymentStatus = 'paid'.

KQ Mong đợi (Frontend): Hiển thị "Thanh toán thành công".

TC-S1-02: (Ngoại lệ) Hủy thanh toán tại cổng (MoMo)

Hành động: Người dùng được chuyển đến cổng MoMo rồi nhấn "Hủy" hoặc đóng tab.

KQ Mong đợi (Backend): paymentStatus vẫn là 'pending'.

KQ Mong đợi (Frontend): Hiển thị "Thanh toán thất bại", cho phép thử lại.

KQ Mong đợi (Tích hợp): restaurant\_app không nhận được đơn hàng.

TC-S1-03: (Ngoại lệ) Đặt hàng khi giá vừa thay đổi

Hành động: Giá sản phẩm A đổi từ 100k -> 120k. Người dùng nhấn "Đặt hàng" với giá cũ 100k.

KQ Mong đợi (Backend): createOrder phát hiện giá không khớp. API trả về lỗi 409 Conflict. Đơn hàng không được tạo.

**Kịch bản Test: Scenario 2 – Điều phối drone và theo dõi**

TC-S2-01: (Thành công) Admin gán drone cho đơn "sẵn sàng"

Hành động: Đơn hàng 123 'ready'. Admin thấy Drone D99 'available', nhấn "Gán drone". Nhân viên nhà hàng nhấn "Xác nhận đã giao".

KQ Mong đợi (Backend): order.drone = D99, drone.status = 'busy'. Sau đó, order.status = 'delivering'.

KQ Mong đợi (Tích hợp): restaurant\_app nhận event order:drone-assigned. client\_app nhận event order:delivering.

TC-S2-02: (Ngoại lệ) Admin gán drone đã 'busy'

Hành động: Drone D99 'available' (do cache). Admin khác gán D99 (DB status: 'busy'). Admin 1 nhấn "Gán drone" D99.

KQ Mong đợi (Backend): assignDrone phát hiện 'busy'. API trả về lỗi 409 Conflict. Đơn hàng không được gán.

Kịch bản Test: Scenario 3 – Giao hàng và xác thực

TC-S3-01: (Thành công) Khách hàng xác nhận nhận hàng

Hành động: Drone hạ cánh. Người dùng nhấn "Đã nhận được hàng" trên client\_app.

KQ Mong đợi (Backend): order.status = 'delivered'. drone.status = 'available'.

KQ Mong đợi (Frontend): client\_app hiển thị "Giao hàng thành công". admin\_app thấy drone D99 'available' trở lại.

TC-S3-02: (Ngoại lệ) Khách hàng báo cáo sự cố (Dự kiến)

Hành động: Drone báo đã đến. Người dùng không nhận được hàng, nhấn "Báo cáo sự cố".

KQ Mong đợi: admin\_app nhận cảnh báo. order.status chuyển thành 'issue'.

**Kịch bản Test: Scenario 4 – CI/CD triển khai hệ thống**

TC-S5-01: (CI) Push lên nhánh feature

Hành động: git push origin kiet.

Kích hoạt: ci-test.yml chạy (test, scan...).

KQ Mong đợi: ci-test.yml thành công -> docker-build-push.yml chạy. Images được build và push .Deploy-production.yml không chạy.

TC-S5-02: (CI) Push lên feature nhưng test thất bại

Hành động: Push code lỗi unit test lên kiet.

Kích hoạt: ci-test.yml chạy npm test và báo lỗi (exit code 1).

KQ Mong đợi: Workflow ci-test.yml báo "Failed". docker-build-push.yml không chạy.

TC-S5-03: (CD) Merge vào main (Triển khai thành công)

Hành động: Merge PR vào main.

Kích hoạt (CI): ci-test.yml -> docker-build-push.yml chạy. Images được push với tag :latest.

Kích hoạt (CD): deploy-production.yml chạy: SSH vào EC2, kéo image :latest, chạy docker run, chạy "Healthcheck".

KQ Mong đợi: Healthcheck trả về 200. Workflow "Success". Production đã được cập nhật.

TC-S5-04: (CD) Triển khai thất bại (Healthcheck fail)

Hành động: Merge code lỗi (làm server crash) vào main.

Kích hoạt (CI): Thành công.

Kích hoạt (CD): deploy-production.yml chạy, kéo image mới, docker run (nhưng container bị crash).

KQ Mong đợi: "Healthcheck" bị timeout/connection refused. Workflow deploy-production.yml báo "Failed". Team phải rollback thủ công.

# **API ENDPOINT**

| **Endpoint** | **Router File** | **Ghi chú** |
| --- | --- | --- |
| / | API/Routers/rootRouter.js | Thông tin chung và Health Check của API. |
| /health | API/Routers/rootRouter.js | (Như trên) |
| /api/auth | API/Routers/authRouter.js | Đăng ký, đăng nhập, đăng xuất (signup, signin, signout, v.v.). |
| /api/user | API/Routers/userRouter.js | Lấy thông tin người dùng hiện tại (yêu cầu xác thực). |
| /api/shop | API/Routers/shopRouter.js | Quản lý cửa hàng (tạo/sửa, lấy cửa hàng của tôi, tìm theo thành phố). |
| /api/item | API/Routers/itemRouter.js | Quản lý sản phẩm (thêm, sửa, xóa, lấy theo ID, lấy theo thành phố). |
| /api/order | API/Routers/orderRouter.js | Quản lý đơn hàng (đặt hàng, lấy đơn hàng, phân trang, thống kê, cập nhật status). |
| /api/cart | API/Routers/cartRouter.js | Quản lý giỏ hàng (lấy giỏ hàng, thêm, cập nhật, xóa, xóa hết). |
| /api/drone | API/Routers/droneRouter.js | Quản lý drone (lấy drone của shop, cập nhật status, gán cho đơn hàng). |
| /api/payment | API/Routers/paymentRouter.js | Xử lý thanh toán (tạo thanh toán MoMo, nhận callback, kiểm tra status). |
| /api/admin | API/Routers/adminRouter.js | **Bảo vệ, yêu cầu role = admin** (Quản lý Users, Shops). |



# **Công nghệ sử dụng**

## **Frontend**

* **React 18** — Xây dựng giao diện người dùng hiện đại, theo mô hình component-based.
* **Vite** — Công cụ build và dev server nhanh, tối ưu cho ứng dụng React.
* **Redux Toolkit** — Quản lý trạng thái toàn cục (người dùng, giỏ hàng, đơn hàng, v.v.).
* **React Router v6** — Điều hướng giữa các trang (Home, Orders, Tracking...).
* **Axios** — Gửi và nhận HTTP request giữa frontend và backend.
* **Leaflet / geoapifyMap**— Hiển thị bản đồ, vị trí drone và trạng thái đơn hàng trực quan.  
  .

## **Backend**

* **Node.js & Express** — Xây dựng REST API và xử lý logic nghiệp vụ.
* **MongoDB & Mongoose** — Lưu trữ và quản lý dữ liệu (người dùng, đơn hàng, drone, v.v.).
* **JSON Web Tokens (JWT)** — Xác thực và phân quyền người dùng.
* **Bcrypt / bcryptjs** — Mã hóa và xác thực mật khẩu người dùng.
* **Multer** — Xử lý multipart/form-data để upload hình ảnh (món ăn, hóa đơn...).
* **Cloudinary** — Lưu trữ và phục vụ hình ảnh lên cloud
* **Email**: Nodemailer.
* **Code Quality: ESLint + Prettier**

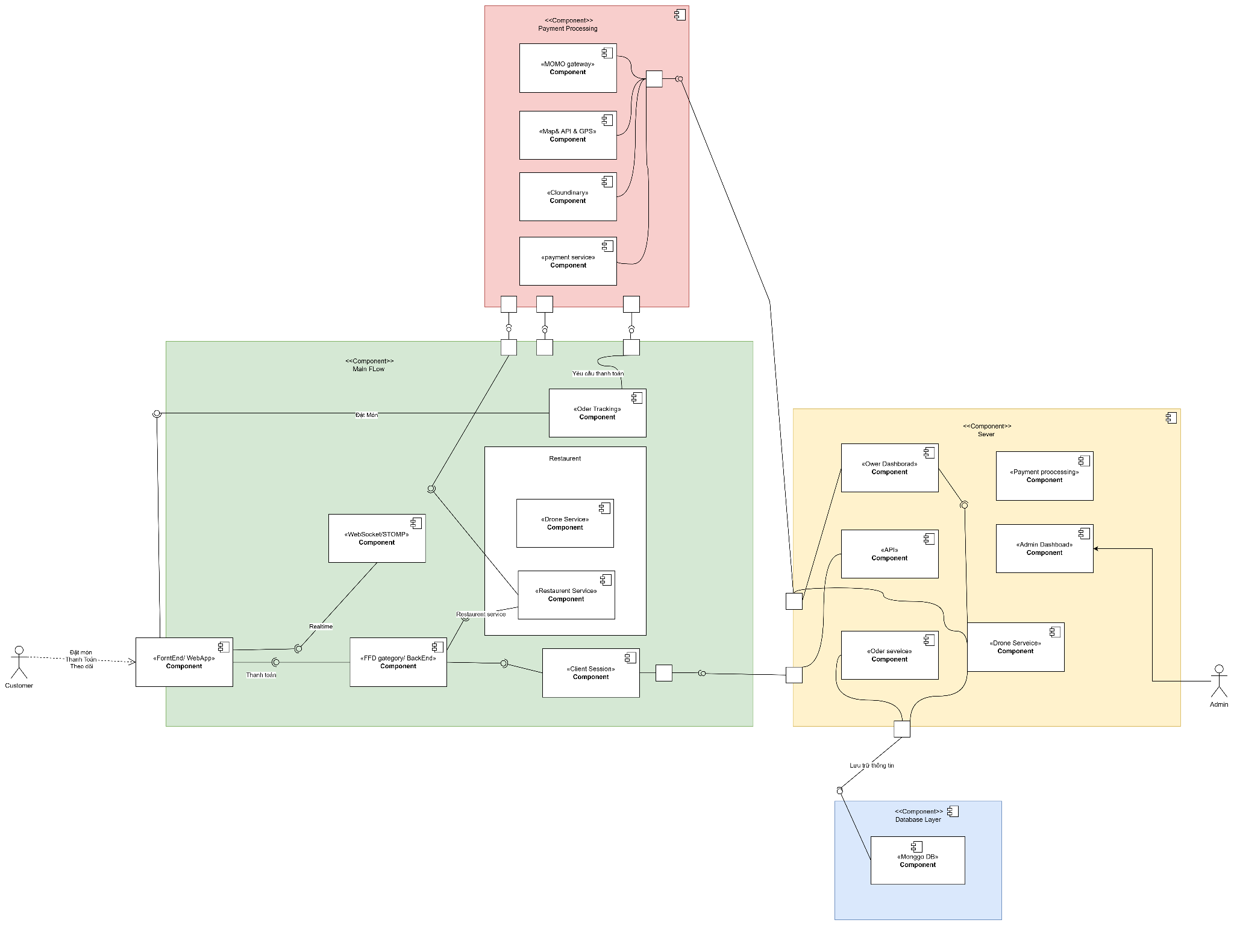
## **CI/CD**

* **GitHub Actions —** Tự động hóa quy trình kiểm thử, build Docker image và triển khai ứng dụng lên môi trường production.
* **Docker —** Dùng để *container hóa* (đóng gói) toàn bộ ứng dụng (frontend, backend) nhằm đảm bảo môi trường chạy đồng nhất và ổn định.
* **AWS (EC2) —** Môi trường server production, nơi các ứng dụng Docker được triển khai và vận hành thực tế.
* **Vercel —** Có thể được sử dụng để triển khai (frontend) nhằm đạt hiệu năng cao khi hosting.
* **Railway —** Sử dụng Dockerfile để build Image rồi tạo container tương tự docker có thể deploy rất tốt backend, fontend ( hỗ trợ nhiều phiên bản node cũ )
* **GHCR (GitHub Container Registry) —** Kho lưu trữ các Docker image đã được build, sau đó được pull về server để triển khai.

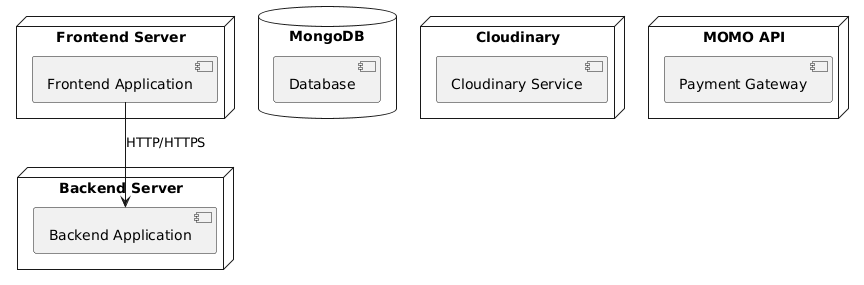
## **Testing & Dev Tooling**

* **Jest, Supertest —** Unit test và integration test cho backend.
* **mongodb-memory-server —** Cơ sở dữ liệu MongoDB chạy in-memory dùng cho môi trường test.
* **Cypress —** End-to-end testing (thư mục cypress/).
* **Nodemon —** Tự động reload server khi phát triển.
* **Concurrently —** Chạy nhiều dev servers song song (được khai báo trong script dev tại root).
* **Docker & Docker Compose —** Container hóa và orchestration cho môi trường local/dev.

## **Component diagram:**



## **Deployment Diagram**



## **ERD**

