**ĐẠI HỌC PHENIKAA**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN PHENIKKA**

A logo for a university

AI-generated content may be incorrect.

**NHÓM 05**

**HỌC PHẦN: PHÂN TÍCH VÀ THIẾT KẾ PHẦN MỀM**

**ĐỀ TÀI: PHÁT TRIỂN WEBSITE ĐẶT BÁNH NGỌT ONLINE THEO KIẾN TRÚC MICROSERVICES**

|  |  |
| --- | --- |
| Giảng viên hướng dẫn | : ThS.Vũ Quang Dũng  TS.Trịnh Thanh Bình |
| Sinh viên thực hiện | : Nguyễn Thanh Phong – 22010251 |
| Lớp tín chỉ | * : [CSE703048-1-1-25(N06)](https://canvas.phenikaa-uni.edu.vn/courses/19865) |

**Hà Nội, tháng 10 năm 2025**

**PHÂN CÔNG NHIỆM VỤ ĐỒ ÁN**

|  |  |
| --- | --- |
| **Danh sách các công việc/nhiệm vụ** | **Mô tả tóm tắt công việc** |
| Plan dự án | Kế hoạch tổng thể, timeline, phân công |
| Báo cáo hàng tuần | Cập nhật tiến độ, blocker, kết quả |
| SRS | Đặc tả FR/NFR, UC, luồng nghiệp vụ |
| Use Case & DFD | Sơ đồ UC tổng quan/chi tiết, DFD quy trình |
| Thiết kế kiến trúc tổng thể | Sơ đồ hệ, Nginx/Redis/RabbitMQ/Postgres/Logging |
| Thiết kế kiến trúc chi tiết | Module BE (NestJS), chuẩn code |
| Thiết kế chức năng chi tiết | API spec, Sequence Checkout/Payment |
| Thiết kế dữ liệu/ERD | ERD 2 DB, khóa/index/migration |
| Coding Backend/Frontend | BE 6 services; FE KH + Admin |
| Hạ tầng & Gateway | docker-compose, .env, nginx.conf, healthcheck |
| Caching & MQ | Redis cache/cart; RabbitMQ events |
| Kiểm thử hộp đen | Test cases & kết quả |
| Logging | Loki/Promtail/Grafana + dashboard |
| Báo cáo dự án | Report cuối kỳ (Word/PDF) |
| Slide | 12–15 slide |

**Nguyễn Thanh Phong — MSSV: 22010251**

|  |  |  |  |
| --- | --- | --- | --- |
| **TT** | **Nhiệm vụ phân công** | **Nội dung đã thực hiện** | **Đánh giá** |
| 1 | **Thiết kế kiến trúc chi tiết** | Viết **System Design (detail).md**: layout module NestJS, **exception filter**, **logging interceptor**, chính sách **DTO/Validation**, quy ước **error code**; guideline repository/service/controller. | 20% |
| 2 | **Thiết kế chức năng chi tiết** | Soạn **API spec** (User/Product/Cart/Order/Payment/Notification) + **Sequence Diagrams** cho **Checkout**, **IPN**; define **status model** (order/payment). | 25% |
| 3 | **Thiết kế dữ liệu/ERD** | Vẽ **ERD** cho postgres\_db\_user\_product & postgres\_db\_order\_cart; chọn kiểu dữ liệu, **index**; viết **migration** (TypeORM) & **seed skeleton**. | 20% |
| 4 | Backend lead coding | Implement 6 services (auth JWT, products CRUD, cart Redis, order create/history, payment mock VNPay, notification consumer); viết **unit tests** (Jest) ~ **62%** lines cho User/Product; thiết lập **Swagger**. | 35% |
|  |  | **Tổng** | **100%** |

# **I. LỜI CẢM ƠN**

Để hoàn thành đồ án tốt nghiệp với đề tài **“PHÁT TRIỂM WEBSITE ĐẶT BÁNH NGỌT ONLINE THEO KIẾN TRÚC MICROSERVICER”**, bên cạnh sự nỗ lực của bản thân, em đã nhận được rất nhiều sự quan tâm, giúp đỡ và động viên từ thầy cô, gia đình và bạn bè. Với lòng biết ơn sâu sắc, em xin được gửi lời cảm ơn chân thành nhất đến tất cả mọi người.

Lời đầu tiên, em xin trân trọng gửi lời cảm ơn sâu sắc nhất đến thầy ThS.Vũ Quang Dũn và TS.Trịnh Thanh Bình. , người đã tận tình chỉ bảo, định hướng và cung cấp cho em những kiến thức chuyên môn quý báu trong suốt quá trình thực hiện đồ án. Sự nhiệt tình, trách nhiệm và những góp ý xác đáng của thầy không chỉ giúp em giải quyết những khó khăn về mặt kỹ thuật mà còn là nguồn động lực lớn để em hoàn thiện đề tài một cách tốt nhất.

Em cũng xin chân thành cảm ơn các thầy cô trong Khoa **CNTT** và Trường **CNTT PHENIKAA** đã truyền đạt cho em những kiến thức nền tảng vững chắc trong suốt những năm tháng học tập tại trường. Những bài giảng tâm huyết của các thầy cô chính là hành trang quan trọng để em tự tin bước vào quá trình nghiên cứu và thực hiện đồ án này.

Sự thành công của đồ án này sẽ không thể trọn vẹn nếu thiếu đi sự ủng hộ về mặt tinh thần từ gia đình và bạn bè. Con xin gửi lời biết ơn đến bố mẹ và gia đình, những người đã luôn ở bên cạnh, động viên, và tạo mọi điều kiện tốt nhất để con yên tâm học tập và nghiên cứu.

Tôi cũng muốn gửi lời cảm ơn đến những người bạn thân thiết, những người đã cùng tôi chia sẻ những khó khăn, trao đổi kiến thức và đưa ra những lời khuyên hữu ích trong suốt quá trình làm đồ án.

Mặc dù đã rất cố gắng, nhưng do kiến thức và kinh nghiệm thực tế còn hạn chế, đồ án chắc chắn không thể tránh khỏi những thiếu sót. Em rất mong nhận được những ý kiến đóng góp quý báu từ quý thầy cô và các bạn để đề tài được hoàn thiện hơn nữa.

Một lần nữa, em xin chân thành cảm ơn!

# **II. MỤC LỤC**

[**I. LỜI CẢM ƠN** 3](#_Toc212447506)

[**II. MỤC LỤC** 4](#_Toc212447507)

[**III. DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT** 7](#_Toc212447508)

[**MỞ ĐẦU** 9](#_Toc212447509)

[**PHẦN 1: TỔNG QUAN DỰ ÁN** 10](#_Toc212447510)

[1.1. Giới thiệu đề tài, mục tiêu và phạm vi 10](#_Toc212447511)

[1.1.1. Giới thiệu đề tài 10](#_Toc212447512)

[1.1.2. Mục tiêu hệ thống 10](#_Toc212447513)

[1.1.3. Phạm vi chức năng MVP 11](#_Toc212447514)

[1.1.4. Ràng buộc & giả định 12](#_Toc212447515)

[1.2. Mô hình quy trình & lý do chọn 12](#_Toc212447516)

[1.2.1. Quy trình phát triển 12](#_Toc212447517)

[1.2.2. Lý do lựa chọn quy trình 14](#_Toc212447518)

[**PHẦN 2: PHÂN TÍCH YÊU CẦU** 15](#_Toc212447519)

[2.1. Yêu cầu dữ liệu 15](#_Toc212447520)

[2.2. Yêu cầu chức năng (FR) 16](#_Toc212447521)

[2.2.1. Khách hang 16](#_Toc212447522)

[2.2.2. Quản trị viên 16](#_Toc212447523)

[2.3. Yêu cầu phi chức năng (NFR) 17](#_Toc212447524)

[2.3.1. Hiệu năng 17](#_Toc212447525)

[2.3.2. Bảo mật 17](#_Toc212447526)

[2.3.3. Khả dụng và khả năng mở rộng 17](#_Toc212447527)

[2.3.4. Tính dễ sử dụng 17](#_Toc212447528)

[2.3.5. Bảo trì và vận hành 18](#_Toc212447529)

[2.3.6. Nhất quán dữ liệu 18](#_Toc212447530)

[2.3.7. Quy trình xử lý chính 18](#_Toc212447531)

[**PHẦN 3 : THIẾT KẾ HỆ THỐNG** 22](#_Toc212447532)

[3.1. Phân tích yêu cầu (Requirement Analysis) 22](#_Toc212447533)

[*3.1.1. Thu thập yêu cầu* 22](#_Toc212447534)

[*3.1.2. Phân tích chi tiết yêu cầu người dùng (User Stories / Use Cases)* 22](#_Toc212447535)

[*3.1.3. Xác định các Bounded Contexts và phân chia Microservices* 23](#_Toc212447536)

[3.2. Thiết kế hệ thống (System Design) 24](#_Toc212447537)

[*3.2.1. Thiết kế kiến trúc tổng thể* 24](#_Toc212447538)

[*3.2.2. Thiết kế API cho từng Microservice (API Design)* 26](#_Toc212447539)

[*3.2.3. Thiết kế cơ sở dữ liệu (Database Design)* 28](#_Toc212447540)

[*3.2.4. Thiết kế giao tiếp giữa các services* 30](#_Toc212447541)

[3.2.5. Thiết kế giao diện người dùng (UI/UX Design) 32](#_Toc212447542)

[3.3. Cài đặt (Implementation) 32](#_Toc212447543)

[*3.3.1. Thiết lập môi trường phát triển* 32](#_Toc212447544)

[*3.3.2. Phát triển Backend Microservices (Node.js/NestJS)* 33](#_Toc212447545)

[*3.3.4. Phát triển Frontend (React/TypeScript)* 36](#_Toc212447546)

[3.3.5. Triển khai Logging Stack (Loki, Promtail, Grafana) 39](#_Toc212447547)

[3.4. Hiệu chỉnh (Refinement & Debugging) 40](#_Toc212447548)

[**PHẦN 4: TRIỂN KHAI & KIỂM THỬ** 41](#_Toc212447549)

[4.1. Công nghệ & môi trường 41](#_Toc212447550)

[4.2. Kết quả các chức năng đạt được 44](#_Toc212447551)

[4.2.1. Giao diện người dùng (Khách hàng) 44](#_Toc212447552)

[4.2.2. Giao diện quản trị (Admin) 51](#_Toc212447553)

[4.3. Kết quả hoạt động của các thành phần Backend và Cơ sở hạ tầng 57](#_Toc212447554)

[*4.3.1. API Gateway (Nginx)* 57](#_Toc212447555)

[*4.3.2. PostgreSQL Databases* 58](#_Toc212447556)

[*4.3.3. Health Checks* 58](#_Toc212447557)

[4.4 Kiểm thử (Testing) 59](#_Toc212447558)

[*4.4.1. Kiểm thử tích hợp (Integration Testing)* 59](#_Toc212447559)

[*4.4.2. Kiểm thử chấp nhận người dùng (User Acceptance Testing - UAT)* 60](#_Toc212447560)

[**PHẦN 5: TỔNG KẾT & HƯỚNG PHÁT TRIỂN** 61](#_Toc212447561)

[5.1. Đánh giá và rút kinh nghiệm 61](#_Toc212447562)

[5.1.1. Mức độ hoàn thành mục tiêu 61](#_Toc212447563)

[5.1.2. Điểm mạnh 61](#_Toc212447564)

[5.1.3. Hạn chế 62](#_Toc212447565)

[5.1.4. Bài học kinh nghiệm 62](#_Toc212447566)

[5.2. Hướng phát triển 62](#_Toc212447567)

[5.2.1. Cải tiến kỹ thuật trọng tâm 62](#_Toc212447568)

[5.2.2. Hoàn thiện tính năng sản phẩm 63](#_Toc212447569)

[5.2.3. Nâng chuẩn chất lượng và quy trình 64](#_Toc212447570)

[5.3. Kết luận 64](#_Toc212447571)

[**TÀI LIỆU THAM KHẢO** 65](#_Toc212447572)

# **III. DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT**

|  |  |
| --- | --- |
| **Viết tắt** | **Tên đầy đủ** |
| API | Application Programming Interface |
| AMQP | Advanced Message Queuing Protocol |
| CSS | Cascading Style Sheets |
| DB | Database |
| DNS | Domain Name System |
| HTTP | HyperText Transfer Protocol |
| HTTPS | HyperText Transfer Protocol Secure |
| JWT | JSON Web Token |
| JSON | JavaScript Object Notation |
| UI | User Interface |
| UX | User Experience |
| CRUD | Create, Read, Update, Delete |
| SPA | Single Page Application |
| SQL | Structured Query Language |
| NoSQL | Not Only SQL |
| VM | Virtual Machine |
| OS | Operating System |
| ENV | Environment (variables) |
| FC | Functional Component (trong React) |
| TSX | TypeScript XML (sử dụng trong React) |
| VNPAY | Cổng thanh toán VNPAY |
| IPN | Instant Payment Notification |
| RabbitMQ | Hệ thống hàng đợi tin nhắn (Message Queue) |
| Redis | Hệ thống lưu trữ key-value trong bộ nhớ (Cache) |
| PostgreSQL | Hệ quản trị cơ sở dữ liệu quan hệ đối tượng |
| Nginx | Web server, reverse proxy, load balancer |
| Docker | Nền tảng container hóa ứng dụng |
| GF | Grafana (trong biến môi trường) |
| CMD | Command |
| OC | Order-Cart (trong tên service/DB) |
| UP | User-Product (trong tên service/DB) |

# **MỞ ĐẦU**

Trong bối cảnh công nghệ thông tin và thương mại điện tử (E-commerce) đang phát triển với tốc độ vũ bão, nhu cầu mua sắm trực tuyến của người tiêu dùng ngày càng gia tăng, đặc biệt đối với các mặt hàng thiết yếu và các sản phẩm mang tính tiện lợi cao như bánh ngọt. Việc xây dựng một nền tảng đặt bánh ngọt trực tuyến không chỉ nhằm đáp ứng xu thế chung của thị trường mà còn mang lại trải nghiệm mua sắm tiện ích, nhanh chóng cho người dùng, đồng thời mở ra những cơ hội kinh doanh hiệu quả và kênh tiếp cận khách hàng mới cho các cửa hàng. Tuy nhiên, để phát triển một hệ thống E-commerce đa chức năng, có khả năng đáp ứng được lượng truy cập lớn, dễ dàng trong việc bảo trì và có khả năng mở rộng linh hoạt trong tương lai, việc lựa chọn một kiến trúc hệ thống phù hợp đóng vai trò vô cùng quan trọng. Trước những hạn chế của kiến trúc nguyên khối truyền thống, kiến trúc microservices, khi được kết hợp với các giải pháp hệ thống phân tán như API Gateway, hàng đợi tin nhắn (message queues) và cơ chế caching, đã nổi lên và chứng minh được ưu thế vượt trội trong việc giải quyết những thách thức phức tạp này.

Đề tài "Phát triển website đặt bánh ngọt online theo kiến trúc microservices" được thực hiện với mục tiêu cốt lõi là nghiên cứu, thiết kế và triển khai một hệ thống thương mại điện tử hoàn chỉnh, chuyên biệt cho việc đặt và mua bánh ngọt, trong đó ứng dụng sâu rộng kiến trúc microservices và các công nghệ phân tán tiên tiến. Dự án tập trung vào việc phân tách các chức năng nghiệp vụ thành các dịch vụ độc lập, bao gồm quản lý người dùng, quản lý sản phẩm, quản lý giỏ hàng, xử lý đơn hàng, tích hợp cổng thanh toán (mô phỏng) và hệ thống thông báo. Bên cạnh đó, các thành phần quan trọng của một hệ thống phân tán như Nginx đóng vai trò API Gateway, RabbitMQ đảm nhiệm giao tiếp bất đồng bộ giữa các dịch vụ, và Redis được sử dụng cho mục đích caching nhằm tối ưu hóa hiệu năng và khả năng mở rộng của toàn hệ thống. Thông qua việc hiện thực hóa một bài toán mang tính thực tiễn cao, đề tài không chỉ hướng đến việc tạo ra một sản phẩm phần mềm có giá trị ứng dụng thực tế mà còn là cơ hội quý báu để người thực hiện vận dụng, đào sâu và mở rộng kiến thức chuyên ngành về thiết kế hệ thống, phát triển phần mềm và quản lý các hệ thống phức tạp hoạt động trong môi trường phân tán.

# **PHẦN 1: TỔNG QUAN DỰ ÁN**

## 1.1. Giới thiệu đề tài, mục tiêu và phạm vi

### 1.1.1. Giới thiệu đề tài

**Tên dự án:** Phát triển website đặt bánh ngọt online theo kiến trúc microservices

**Mô tả tổng quan:**Dự án xây dựng một nền tảng thương mại điện tử hoàn chỉnh chuyên về lĩnh vực kinh doanh bánh ngọt, áp dụng kiến trúc **Microservices** hiện đại với các công nghệ tiên tiến. Hệ thống được thiết kế theo mô hình phân tán (distributed system), mỗi dịch vụ độc lập về cơ sở dữ liệu (Database-per-Service pattern), giao tiếp qua API Gateway và Message Queue, nhằm đảm bảo khả năng mở rộng (scalability), bảo trì (maintainability) và khả năng chịu lỗi (fault tolerance).

**Đặc điểm nổi bật:**

* Kiến trúc Microservices với 6 dịch vụ backend độc lập
* Tích hợp thanh toán thực tế qua VNPay Payment Gateway
* Hệ thống giỏ hàng hiệu năng cao với Redis Cache
* Xử lý bất đồng bộ với RabbitMQ Message Broker
* Tính năng độc đáo: Custom Cake Builder (thiết kế bánh tùy chỉnh)
* Triển khai hoàn toàn trên Docker Container

### 1.1.2. Mục tiêu hệ thống

**A. Mục tiêu nghiệp vụ**

|  |  |  |
| --- | --- | --- |
| **STT** | **Mục tiêu** | **Mô tả chi tiết** |
| 1 | **Trải nghiệm mua sắm toàn diện** | Cung cấp đầy đủ tính năng từ duyệt sản phẩm, tùy chỉnh bánh, thanh toán đến theo dõi đơn hàng |
| 2 | **Tính năng đặc trưng** | Custom Cake Builder cho phép khách hàng tự thiết kế bánh với 27 tùy chọn đa dạng |
| 3 | **Thanh toán an toàn** | Tích hợp VNPay Gateway với mã hóa giao dịch, bảo mật thông tin thanh toán |
| 4 | **Quản trị tập trung** | Dashboard quản lý toàn diện cho admin: sản phẩm, đơn hàng, người dùng, đánh giá |
| 5 | **Thông báo tự động** | Gửi email xác nhận đơn hàng, cập nhật trạng thái tự động qua RabbitMQ |

**B. Mục tiêu kỹ thuật**

|  |  |  |
| --- | --- | --- |
| **STT** | **Mục tiêu** | **Công nghệ/Phương pháp** |
| 1 | **Kiến trúc phân tán** | Microservices với API Gateway (Nginx), độc lập database |
| 2 | **Hiệu năng cao** | Redis Cache cho giỏ hàng, Database Indexing, Query Optimization |
| 3 | **Khả năng mở rộng** | Horizontal Scaling ready, Container-based deployment |
| 4 | **Bảo mật** | JWT Authentication, Bcrypt Password Hashing, Role-based Authorization |
| 5 | **Nhất quán dữ liệu** | Database Transaction, Eventual Consistency cho notification |
| 6 | **Monitoring & Logging** | Winston Logger, Health Check Endpoints, Docker Logs |

**C. Mục tiêu học tập**

* Áp dụng kiến thức Phân tích & Thiết kế Hệ thống vào thực tế
* Thực hành làm việc nhóm với quy trình Agile
* Nắm vững công nghệ hiện đại: NestJS, React, Docker, Microservices
* Kỹ năng tài liệu hóa (UML, ERD, API Documentation)
* Kinh nghiệm xử lý vấn đề trong hệ thống phân tán

### 1.1.3. Phạm vi chức năng MVP

**Khách hàng**

* Đăng ký/đăng nhập/đăng xuất.
* Quản lý tài khoản.
* Catalog sản phẩm: xem danh sách, tìm kiếm, lọc theo danh mục, sắp xếp; xem chi tiết.
* Giỏ hàng: thêm/xóa/cập nhật số lượng
* Đặt hàng/checkout; **thanh toán mô phỏng VNPay** (return + IPN).
* Xem lịch sử & chi tiết đơn.
* Đánh giá sản phẩm.
* Đặt bánh custome

**Quản trị**

* Đăng nhập admin.
* Quản lý **Sản phẩm/Danh mục** (CRUD, upload ảnh).
* Quản lý **Người dùng** (xem/dổi vai trò cơ bản).
* Quản lý **Đơn hàng** (lọc, xem chi tiết, cập nhật trạng thái).
* Dashboard thống kê cơ bản (đơn mới, doanh thu ước tính, sản phẩm bán chạy).
* Quản lý đánh giá

### 1.1.4. Ràng buộc & giả định

* **Thời gian:** trong 01 học kỳ; kế hoạch thực thi **5 tuần**
* **Tài nguyên:** chạy **local** trên máy sinh viên; RAM khuyến nghị ≥ 16GB khi bật toàn bộ stack.
* **Triển khai:** **demo cục bộ qua Docker Compose**, không yêu cầu public cloud.
* **Bảo mật:** dùng **JWT** mức cơ bản; không triển khai OAuth/SSO.
* **Dữ liệu:** dùng bộ **seed** phục vụ demo và kiểm thử.

## 1.2. Mô hình quy trình & lý do chọn

### 1.2.1. Quy trình phát triển

Nhóm áp dụng mô hình **Agile “lite”** – một phiên bản tinh gọn của Agile/Scrum – nhằm đảm bảo tính linh hoạt trong quá trình phát triển, đồng thời vẫn đáp ứng yêu cầu về tài liệu và quy trình kiểm thử theo chuẩn học phần.  
Quy trình được chia thành **4 Sprint ngắn**, mỗi Sprint kéo dài từ **1 đến 1.5 tuần**, với các hoạt động lặp lại gồm:

* **Đặt mục tiêu Sprint:** Xác định rõ phạm vi công việc và các tính năng sẽ hoàn thành trong Sprint.
* **Xây dựng Sprint Backlog:** Liệt kê cụ thể các task, ước lượng thời gian và phân công cho từng thành viên.
* **Daily Check-in (họp nhanh hằng ngày):** Mỗi thành viên báo cáo tiến độ, khó khăn gặp phải và kế hoạch trong ngày.
* **Sprint Review:** Tổng kết kết quả thực hiện, demo các tính năng đã hoàn thành.
* **Sprint Retrospective:** Đánh giá lại quy trình làm việc, rút kinh nghiệm cho Sprint kế tiếp.

Mặc dù áp dụng Agile, nhóm vẫn **tuân thủ cấu trúc tài liệu của V-Model** để đáp ứng yêu cầu học phần, đảm bảo các **artefact (tài liệu)** tương ứng cho từng giai đoạn:

* **SRS (Software Requirement Specification):** Đặc tả yêu cầu phần mềm.
* **Thiết kế hệ thống:** Mô hình kiến trúc, ERD, Use Case, DFD.
* **Cài đặt:** Source code, tài liệu triển khai.
* **Kiểm thử:** Test case, kết quả kiểm thử, báo cáo lỗi.

Các **artefact chính của từng Sprint** được xác định như sau:

|  |  |  |  |
| --- | --- | --- | --- |
| **Sprint** | **Thời gian** | **Mục tiêu chính** | **Artefact đầu ra** |
| **Sprint 1** | Tuần 1 | Xác định phạm vi, yêu cầu và kế hoạch dự án | Project Plan, Scope, Use Case/DFD khung, **SRS v1.0** |
| **Sprint 2** | Tuần 2 | Thiết kế hệ thống & hạ tầng | Kiến trúc tổng thể, **API Spec**, **ERD**, docker-compose skeleton |
| **Sprint 3** | Tuần 3 | Phát triển chức năng lõi và kiểm thử đợt 1 | Coding backend services + Frontend khách hàng, Redis/MQ, test hộp đen đợt 1 |
| **Sprint 4** | Tuần 4 | Hoàn thiện hệ thống & tài liệu bàn giao | Frontend Admin, hoàn thiện test, logging stack, **báo cáo & slide trình bày** |

Việc chia nhỏ công việc theo Sprint giúp nhóm dễ dàng quản lý tiến độ, theo dõi hiệu quả từng giai đoạn và linh hoạt điều chỉnh khi có thay đổi từ giảng viên hoặc yêu cầu người dùng.

### 1.2.2. Lý do lựa chọn quy trình

Nhóm lựa chọn mô hình **Agile “lite”** thay vì các mô hình truyền thống như **Waterfall** hoặc **V-Model** vì các lý do sau:

1. **Lặp nhanh – kiểm thử sớm:**
   * Mỗi Sprint đều có giai đoạn kiểm thử và phản hồi sớm, giúp nhóm phát hiện và xử lý lỗi tích hợp (đặc biệt là giữa các thành phần Redis, Message Queue, API Gateway) ngay trong quá trình phát triển.
   * Điều này giảm thiểu rủi ro “dồn lỗi cuối kỳ” và đảm bảo chất lượng phần mềm ổn định hơn.
2. **Linh hoạt với thay đổi yêu cầu:**
   * Trong giai đoạn đầu, yêu cầu của đề tài và phạm vi chức năng thường chưa hoàn toàn cố định. Agile cho phép điều chỉnh backlog hoặc mục tiêu Sprint mà không ảnh hưởng lớn đến tổng tiến độ.
3. **Phù hợp với quy mô nhóm sinh viên:**
   * Với nhóm 4–5 thành viên, mô hình Agile lite giúp phân công rõ ràng, quản lý công việc bằng các công cụ đơn giản (Trello, Notion, GitHub Project).
   * Không yêu cầu overhead lớn như Scrum chuẩn (không có Product Owner/ Scrum Master cố định).
4. **Dễ theo dõi tiến độ và đánh giá đóng góp cá nhân:**
   * Thông qua các buổi daily check-in và review cuối Sprint, giảng viên và nhóm trưởng có thể dễ dàng theo dõi hiệu quả làm việc của từng thành viên.
5. **Đảm bảo chuẩn học phần (kết hợp V-Model):**
   * Mặc dù Agile chú trọng tốc độ, nhóm vẫn xây dựng đầy đủ các tài liệu như SRS, thiết kế, và test case – đảm bảo đúng yêu cầu kiểm định của lớp và giúp báo cáo được chặt chẽ.

# **PHẦN 2: PHÂN TÍCH YÊU CẦU**

## 2.1. Yêu cầu dữ liệu

Hệ thống quản lý tám nhóm dữ liệu cốt lõi và lưu trữ theo mô hình “mỗi dịch vụ một cơ sở dữ liệu”. Cụ thể, phần người dùng, danh mục, sản phẩm, đánh giá và cấu hình bánh tùy chỉnh được đặt trong một cơ sở dữ liệu quan hệ dành riêng; phần đơn hàng và chi tiết đơn hàng sử dụng một cơ sở dữ liệu quan hệ khác; giỏ hàng được lưu trong bộ nhớ đệm Redis nhằm tối ưu tốc độ; ngoài ra hệ thống lưu nhật ký hoạt động để phục vụ giám sát và truy vết. Tất cả khóa chính sử dụng định danh dạng UUID phiên bản 4; mật khẩu người dùng được băm bằng thuật toán bcrypt với số vòng muối tối thiểu là mười; các trường dữ liệu có cấu trúc linh hoạt như “tùy chọn bánh” được lưu ở dạng JSON nhị phân để dễ mở rộng về sau; các cột thường xuyên truy vấn như mã người dùng và mã danh mục đều có chỉ mục.

Nhóm **Người dùng** gồm các thuộc tính: định danh, thư điện tử duy nhất, mật khẩu đã băm, họ và tên, số điện thoại, địa chỉ, ảnh đại diện, ngày sinh, giới tính, vai trò (người dùng thông thường hoặc quản trị), thời điểm tạo và cập nhật. Nhóm **Danh mục** có định danh, tên duy nhất, mô tả ngắn, đường dẫn ảnh đại diện, thời điểm tạo và cập nhật, đồng thời liên hệ một–nhiều với sản phẩm. Nhóm **Sản phẩm** chứa tên, mô tả, giá tiền theo dạng số thập phân có độ chính xác cố định, số lượng tồn, ảnh, khóa ngoại đến danh mục, trạng thái hiển thị và dấu thời gian. Nhóm **Đánh giá** ghi nhận sản phẩm được đánh giá, người viết, mã đơn hàng tham chiếu, số sao từ một đến năm, nội dung bình luận, danh sách ảnh minh họa, cờ “đã duyệt” và “đang hiển thị”, cùng dấu thời gian. Nhóm **Tùy chọn bánh** mô tả các lựa chọn cấu hình (kích cỡ, cốt bánh, lớp phủ, hương vị, trang trí) kèm phụ phí, thứ tự hiển thị và trạng thái kích hoạt.

Nhóm **Giỏ hàng** được lưu trên Redis theo khóa cart:{mã-người-dùng}, giá trị là một đối tượng JSON gồm danh sách mặt hàng (mã sản phẩm, tên, ảnh, số lượng, đơn giá chụp tại thời điểm thêm, thông tin tùy biến nếu là bánh thiết kế), thời hạn tồn tại mặc định là bảy ngày. Nhóm **Đơn hàng** lưu người mua, tổng tiền, địa chỉ giao, khung giờ giao, trạng thái xử lý (chờ, đang xử lý, hoàn tất, hủy hoặc thất bại) và dấu thời gian. Nhóm **Chi tiết đơn hàng** chụp lại ảnh, tên và giá của sản phẩm tại thời điểm mua để đảm bảo tính lịch sử, kèm số lượng, giá dòng và thông tin tùy biến. Cuối cùng, **nhật ký hệ thống** tập trung hóa các bản ghi từ cổng lập trình ứng dụng và các dịch vụ để phục vụ theo dõi, cảnh báo và khắc phục sự cố.

## 2.2. Yêu cầu chức năng (FR)

### 2.2.1. Khách hang

Người dùng cuối có thể tạo tài khoản, đăng nhập và cập nhật hồ sơ cá nhân. Sau khi đăng nhập, họ được duyệt danh mục và danh sách sản phẩm với khả năng phân trang, lọc theo danh mục, khoảng giá và trạng thái còn hàng, đồng thời sắp xếp theo giá, tên hoặc thời gian tạo. Màn hình chi tiết sản phẩm hiển thị mô tả, ảnh, tồn kho, điểm đánh giá trung bình và danh sách đánh giá đã được duyệt.

Hệ thống cung cấp trình tạo bánh tùy chỉnh cho phép người dùng lựa chọn năm nhóm tùy chọn bắt buộc; giá tiền được tính theo thời gian thực bằng tổng phụ phí của các lựa chọn; phía máy chủ kiểm tra hợp lệ của toàn bộ lựa chọn trước khi chấp nhận đưa vào giỏ. Giỏ hàng hỗ trợ thêm, bớt, thay đổi số lượng và tính tổng ngay lập tức nhờ lưu trữ trên bộ nhớ đệm. Quy trình thanh toán tạo đơn, chuyển hướng sang cổng thanh toán VNPay mô phỏng và xử lý cả luồng người dùng trở về lẫn thông báo máy–máy (IPN) để đảm bảo kết quả chính xác và chống lặp giao dịch. Sau khi mua, người dùng xem lịch sử đơn hàng, mở chi tiết từng đơn, và khi đơn đã hoàn tất có thể gửi đánh giá kèm ảnh minh họa.

### 2.2.2. Quản trị viên

Quản trị viên có các trang làm việc để xem danh sách và chi tiết người dùng, khóa hoặc mở lại tài khoản khi cần. Họ thực hiện đầy đủ thao tác tạo, đọc, sửa, xóa đối với danh mục và sản phẩm; hệ thống tự động vô hiệu hóa các bản ghi đệm liên quan để dữ liệu hiển thị luôn cập nhật. Chức năng quản lý tồn kho cho phép điều chỉnh số lượng thủ công và cảnh báo khi sắp hết hàng; số lượng tồn cũng được trừ tự động khi đơn hoàn tất.

Với đơn hàng, quản trị viên theo dõi toàn bộ danh sách, tra cứu theo trạng thái, ngày tháng hoặc người mua; vào chi tiết để xem thông tin giao hàng, lịch giao và cấu hình bánh tùy chỉnh; cập nhật trạng thái theo quy trình tiêu chuẩn. Phần đánh giá hỗ trợ duyệt hàng loạt, ẩn hoặc xóa bình luận vi phạm. Trang tổng quan cung cấp thẻ số liệu tổng hợp, biểu đồ doanh thu theo thời gian và phân bố trạng thái đơn giúp điều hành nhanh.

## 2.3. Yêu cầu phi chức năng (NFR)

### 2.3.1. Hiệu năng

Mục tiêu là thời gian tải trang chủ ở bách phân vị 95 nhỏ hơn hai giây và thời gian phản hồi lập trình ứng dụng ở bách phân vị 95 nhỏ hơn năm trăm mili giây. Để đạt được điều đó, giao diện được chia nhỏ gói mã, tối ưu ảnh và tải lười; phía máy chủ áp dụng phân trang bắt buộc, chỉ mục cơ sở dữ liệu hợp lý và truy vấn chọn trường cần thiết. Bộ nhớ đệm Redis được dùng cho giỏ hàng và các kết quả đọc tần suất cao nhằm nâng tỷ lệ trúng đệm, giảm tải cho cơ sở dữ liệu.

### 2.3.2. Bảo mật

Xác thực sử dụng thẻ truy cập theo chuẩn “Mã thông báo Web dạng JSON”, thời hạn phù hợp và lưu ở phía khách. Phân quyền dựa trên vai trò với hai mức chính là người dùng và quản trị; các tuyến lập trình được bảo vệ bằng cơ chế gác cổng tương ứng. Mật khẩu luôn băm trước khi lưu; đầu vào được kiểm tra và ràng buộc kiểu dữ liệu để phòng chống chèn câu lệnh và chèn mã kịch bản phía trình duyệt. Cổng ngược Nginx cấu hình chính sách chia sẻ tài nguyên khác nguồn theo danh sách miền được phép; có thể áp dụng giới hạn tốc độ nhẹ cho các điểm cuối nhạy cảm như xác thực. Luồng thanh toán kiểm tra chữ ký băm HMAC của VNPay và xác minh bí mật webhook.

### 2.3.3. Khả dụng và khả năng mở rộng

Toàn bộ hệ thống được đóng gói bằng Docker và điều phối bằng tệp cấu hình hợp nhất, mỗi dịch vụ có điểm kiểm tra sức khỏe và tự khởi động lại khi gặp sự cố. Kiến trúc chia theo dịch vụ giúp mở rộng theo chiều ngang đúng thành phần đang chịu tải; cổng ngược có cấu hình nhóm máy chủ để sẵn sàng cân bằng tải. Cơ sở dữ liệu tách biệt theo ngữ cảnh nghiệp vụ giúp triển khai sao lưu, phục hồi và nhân bản phù hợp; hệ thống nhật ký tập trung (Loki, Promtail, Grafana) cung cấp khả năng tra cứu và giám sát một cách nhất quán.

### 2.3.4. Tính dễ sử dụng

Giao diện người dùng tuân thủ một hệ thiết kế thống nhất: màu sắc, khoảng cách, kiểu chữ và thành phần lặp lại giữa các màn hình. Ứng dụng ưu tiên thiết kế dành cho thiết bị di động, hỗ trợ tốt trên nhiều kích thước màn hình. Trạng thái tải, trống dữ liệu và lỗi đều có hiển thị thân thiện; thông báo tương tác xuất hiện rõ ràng, dễ hiểu. Các biểu mẫu có hướng dẫn và thông báo kiểm tra dữ liệu ngay khi người dùng nhập.

### 2.3.5. Bảo trì và vận hành

Mã nguồn tuân thủ quy tắc định dạng thống nhất, đặt tên rõ nghĩa và tách lớp theo kiến trúc của NestJS: bộ điều khiển cho điểm cuối lập trình, dịch vụ cho nghiệp vụ, đối tượng truyền dữ liệu cho kiểm tra hợp lệ và mô hình cơ sở dữ liệu cho ánh xạ. Tài liệu lập trình ứng dụng được tạo tự động và phát hành dưới dạng bộ sưu tập thử nghiệm; mỗi thay đổi được rà soát ngang hàng để giảm rủi ro lỗi. Hệ thống ghi nhật ký theo cấp độ (lỗi nghiêm trọng, cảnh báo, thông tin nghiệp vụ, theo dõi khi phát triển) để dễ truy vết.

### 2.3.6. Nhất quán dữ liệu

Trong phạm vi một dịch vụ, hệ thống bảo đảm tính nhất quán mạnh nhờ giao dịch cơ sở dữ liệu theo chuẩn ACID: tạo đơn hàng, tạo chi tiết đơn hàng và điều chỉnh tồn kho phải cùng thành công hoặc cùng hủy bỏ. Giữa các dịch vụ, hệ thống hướng tới nhất quán cuối cùng thông qua hàng đợi thông điệp: sau khi hoàn tất thanh toán, dịch vụ thanh toán phát sự kiện “đã xử lý thanh toán”; dịch vụ đơn hàng cập nhật trạng thái khi tiêu thụ sự kiện; dịch vụ thông báo gửi thư xác nhận khi nhận được sự kiện “đơn hàng đã xác nhận”. Mọi thông điệp đều bền vững, có cơ chế thử lại, và đưa vào hàng đợi đặc biệt nếu thất bại nhiều lần, nhờ đó dữ liệu được đồng bộ theo thời gian thực một cách an toàn.

### 2.3.7. Quy trình xử lý chính

Mô tả một số quy trình nghiệp vụ quan trọng trong hệ thống:

* **Quy trình đặt hàng của khách hàng:**
  1. **Khách hàng lựa chọn sản phẩm:** Người dùng duyệt xem các sản phẩm trên website, xem thông tin chi tiết và quyết định thêm các sản phẩm mong muốn vào giỏ hàng. Số lượng sản phẩm có thể được điều chỉnh trong giỏ.
  2. **Tiến hành Checkout:** Sau khi đã chọn đủ sản phẩm, khách hàng nhấp vào nút "Thanh toán" hoặc "Checkout". Hệ thống yêu cầu khách hàng đăng nhập (nếu chưa) hoặc đăng ký tài khoản mới.
  3. **Cung cấp thông tin giao hàng:** Khách hàng điền hoặc xác nhận thông tin địa chỉ giao hàng, số điện thoại liên hệ.
  4. **Lựa chọn phương thức thanh toán và Xác nhận đơn hàng:** Khách hàng chọn phương thức thanh toán (trong dự án này là mô phỏng thanh toán qua VNPay). Sau khi xem lại toàn bộ thông tin đơn hàng (sản phẩm, số lượng, tổng tiền, địa chỉ), khách hàng xác nhận đặt hàng.
  5. **Xử lý thanh toán:** Hệ thống chuyển hướng người dùng đến giao diện mô phỏng của cổng thanh toán VNPay. Khách hàng thực hiện các bước thanh toán theo hướng dẫn.
  6. **Nhận kết quả thanh toán:** Sau khi hoàn tất, VNPay (mô phỏng) sẽ gửi thông tin kết quả giao dịch về cho hệ thống thông qua Return URL (trả về trình duyệt của khách hàng) và IPN URL (gọi ngầm tới backend của Payment Service).
  7. **Cập nhật đơn hàng và Thông báo:**
     + Payment Service xác nhận kết quả thanh toán và thông báo cho Order Service (có thể qua RabbitMQ).
     + Order Service dựa trên kết quả thanh toán để cập nhật trạng thái đơn hàng (ví dụ: 'chờ xử lý' nếu thanh toán thành công) và chính thức tạo bản ghi đơn hàng trong cơ sở dữ liệu.
     + Order Service publish một sự kiện (ví dụ: order\_created) lên RabbitMQ.
     + Notification Service lắng nghe sự kiện này và tự động gửi email xác nhận đơn hàng (bao gồm mã đơn hàng, chi tiết sản phẩm, tổng tiền) đến địa chỉ email của khách hàng.
     + (Tùy chọn) Nếu hệ thống có quản lý tồn kho, Product Service có thể lắng nghe sự kiện này để cập nhật số lượng tồn kho của sản phẩm.
* **Quy trình xử lý đơn hàng của Quản trị viên:**
  1. **Đăng nhập và Truy cập quản lý đơn hàng:** Quản trị viên đăng nhập vào trang quản trị và truy cập vào mục quản lý đơn hàng.
  2. **Xem danh sách đơn hàng:** Hệ thống hiển thị danh sách các đơn hàng mới hoặc các đơn hàng đang ở các trạng thái khác nhau (ví dụ: 'chờ xác nhận', 'đang xử lý').
  3. **Xem chi tiết và Xác nhận đơn hàng:** Quản trị viên chọn một đơn hàng cụ thể để xem chi tiết (thông tin khách hàng, sản phẩm, địa chỉ giao hàng). Sau khi kiểm tra, quản trị viên có thể xác nhận đơn hàng, chuyển trạng thái từ 'chờ xác nhận' sang 'đang xử lý'.
  4. **Cập nhật trạng thái đơn hàng:** Trong quá trình chuẩn bị và giao hàng, quản trị viên tiếp tục cập nhật trạng thái của đơn hàng (ví dụ: 'đang chuẩn bị hàng', 'đang giao hàng', 'đã giao thành công', 'giao hàng thất bại', 'đã hủy').
  5. **Thông báo cho khách hàng (Tùy chọn tự động):** Mỗi khi trạng thái đơn hàng được cập nhật bởi quản trị viên, hệ thống (thông qua Order Service và Notification Service, sử dụng RabbitMQ) có thể tự động gửi email thông báo cho khách hàng về sự thay đổi này.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4: Sơ đồ luồng dữ liệu chi tiết cho quy trình đặt hàng của khách hàng

*(Hình 2.1: Sơ đồ luồng dữ liệu chi tiết cho quy trình đặt hàng của khách hàng)*

*(Hình ảnh này sẽ thể hiện rõ các bước từ khi khách hàng chọn sản phẩm, qua giỏ hàng, checkout, thanh toán, và cách các Microservices (Cart, Order, Payment, Notification, Product) tương tác với nhau và với các hệ thống bên ngoài như VNPay (mô phỏng) và Email Server.)*

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5: Sơ đồ luồng dữ liệu cho quy trình xử lý đơn hàng của Quản trị viên

*(Hình 2.2: Sơ đồ luồng dữ liệu cho quy trình xử lý đơn hàng của Quản trị viên)*

*(Hình ảnh này sẽ mô tả cách Admin tương tác với Order Service để xem và cập nhật trạng thái đơn hàng, và luồng thông báo đến khách hàng nếu có.)*

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6: Sơ đồ minh họa luồng thông điệp qua RabbitMQ cho tác vụ xác nhận đơn hàng

*(Hình 2.3: Sơ đồ minh họa luồng thông điệp qua RabbitMQ cho tác vụ xác nhận đơn hàng)*

*(Hình ảnh này sẽ thể hiện: Order Service (Producer) -> gửi message (chứa Order Info) -> đến một Exchange (ví dụ: 'orders\_exchange' kiểu direct) với một routing key (ví dụ: 'order.created') -> Exchange định tuyến message đến một Queue (ví dụ: 'notifications\_queue') -> Notification Service (Consumer) đọc message từ 'notifications\_queue' và thực hiện gửi email.)*

# **PHẦN 3 : THIẾT KẾ HỆ THỐNG**

## 3.1. Phân tích yêu cầu (Requirement Analysis)

Giai đoạn phân tích yêu cầu là bước nền tảng, quyết định sự thành công của dự án. Mục tiêu của giai đoạn này là hiểu rõ và định nghĩa một cách chính xác những gì hệ thống cần phải làm, đối tượng người dùng là ai, và các ràng buộc kỹ thuật, nghiệp vụ liên quan.

### *3.1.1. Thu thập yêu cầu*

Các yêu cầu của hệ thống được thu thập từ nhiều nguồn:

* **Mô tả bài toán ban đầu:** Dựa trên đề cương đề tài và các ý tưởng cốt lõi về một nền tảng E-commerce cho bánh ngọt.
* **Nghiên cứu thị trường và các hệ thống tương tự:** Phân tích các website bán bánh ngọt trực tuyến hiện có, các nền tảng E-commerce lớn để học hỏi về các tính năng phổ biến, quy trình người dùng hiệu quả và các điểm cần cải thiện.
* **Xác định đối tượng người dùng và nhu cầu của họ:** Đặt mình vào vị trí của khách hàng mua bánh và người quản trị cửa hàng để hình dung các nhu cầu và mong đợi của họ từ hệ thống.
* **Thảo luận và định hướng với người hướng dẫn (nếu có):** Nhận các góp ý và định hướng về phạm vi, mục tiêu và các yêu cầu kỹ thuật.

### *3.1.2. Phân tích chi tiết yêu cầu người dùng (User Stories / Use Cases)*

Sau khi thu thập, các yêu cầu được phân tích và cụ thể hóa dưới dạng User Stories hoặc Use Cases chi tiết để đảm bảo sự rõ ràng và dễ hiểu. Dưới đây là một số ví dụ:

* **User Stories cho Khách hàng:**
  + "Là một khách hàng mới, tôi muốn có thể đăng ký tài khoản bằng email và mật khẩu để có thể lưu trữ thông tin và lịch sử mua hàng."
  + "Là một khách hàng, tôi muốn có thể tìm kiếm sản phẩm theo tên hoặc từ khóa để nhanh chóng tìm thấy loại bánh tôi cần."
  + "Là một khách hàng, tôi muốn có thể xem chi tiết sản phẩm, bao gồm hình ảnh, mô tả, và giá, trước khi quyết định mua."
  + "Là một khách hàng, tôi muốn có thể thêm nhiều sản phẩm vào giỏ hàng và cập nhật số lượng của chúng."
  + "Là một khách hàng, tôi muốn có một quy trình thanh toán đơn giản và an toàn (mô phỏng) để hoàn tất đơn hàng."
  + "Là một khách hàng, tôi muốn nhận được email xác nhận ngay sau khi đặt hàng thành công."
* **User Stories cho Quản trị viên:**
  + "Là một quản trị viên, tôi muốn có thể đăng nhập vào một khu vực quản trị riêng biệt để quản lý hệ thống."
  + "Là một quản trị viên, tôi muốn có thể thêm sản phẩm mới, bao gồm tên, mô tả, giá, hình ảnh và chọn danh mục cho sản phẩm đó."
  + "Là một quản trị viên, tôi muốn có thể xem danh sách tất cả các đơn hàng và cập nhật trạng thái của chúng (ví dụ: từ 'chờ xác nhận' sang 'đang xử lý')."
  + "Là một quản trị viên, tôi muốn có thể xem các thống kê cơ bản về doanh thu và số lượng đơn hàng để theo dõi hiệu quả kinh doanh."

### *3.1.3. Xác định các Bounded Contexts và phân chia Microservices*

Dựa trên các yêu cầu nghiệp vụ và các User Stories đã được phân tích, bước tiếp theo là xác định các "Bounded Contexts" theo nguyên lý Domain-Driven Design (DDD). Mỗi Bounded Context đại diện cho một phần cụ thể của miền nghiệp vụ với ngôn ngữ và mô hình dữ liệu riêng. Từ đó, các microservices tương ứng được phân chia:

* **User Context:** Quản lý tất cả các thông tin và nghiệp vụ liên quan đến người dùng, bao gồm đăng ký, đăng nhập, xác thực, quản lý hồ sơ. => **User Service**.
* **Product Catalog Context:** Quản lý thông tin về sản phẩm bánh ngọt, danh mục, giá cả, hình ảnh, mô tả. => **Product Service**.
* **Shopping Cart Context:** Quản lý trạng thái giỏ hàng hiện tại của người dùng, bao gồm các sản phẩm đã chọn và số lượng. => **Cart Service** (sử dụng Redis làm nơi lưu trữ chính).
* **Order Management Context:** Xử lý toàn bộ quy trình đặt hàng, từ khi tạo đơn hàng, quản lý trạng thái, đến lưu trữ lịch sử đơn hàng. => **Order Service**.
* **Payment Context:** Xử lý các tương tác với cổng thanh toán (mô phỏng VNPay), xác nhận kết quả thanh toán. => **Payment Service**.
* **Notification Context:** Chịu trách nhiệm gửi các thông báo (chủ yếu là email) đến người dùng, ví dụ như xác nhận đơn hàng, cập nhật trạng thái. => **Notification Service**.

Table 3: Ánh xạ yêu cầu nghiệp vụ vào các Microservices

|  |  |
| --- | --- |
| **Yêu cầu Nghiệp vụ Chính** | **Microservice Chịu Trách Nhiệm Chính** |
| Đăng ký/Đăng nhập người dùng, Quản lý hồ sơ | User Service |
| Hiển thị sản phẩm, Tìm kiếm sản phẩm | Product Service |
| Quản lý giỏ hàng của người dùng | Cart Service |
| Tạo và quản lý đơn hàng | Order Service |
| Xử lý thanh toán đơn hàng (mô phỏng) | Payment Service |
| Gửi email xác nhận, thông báo | Notification Service |
| Quản lý người dùng (Admin) | User Service |
| Quản lý sản phẩm, danh mục (Admin) | Product Service |
| Quản lý đơn hàng (Admin) | Order Service |

*Bảng 3: Ánh xạ yêu cầu nghiệp vụ vào các Microservices*

## 3.2. Thiết kế hệ thống (System Design)

Sau khi đã hiểu rõ các yêu cầu, giai đoạn thiết kế tập trung vào việc định hình kiến trúc tổng thể của hệ thống, thiết kế chi tiết cho từng microservice, cơ sở dữ liệu, cách các thành phần tương tác với nhau, và giao diện người dùng.

### *3.2.1. Thiết kế kiến trúc tổng thể*

Kiến trúc tổng thể của hệ thống "Nền tảng Đặt Bánh Ngọt Online" được xây dựng dựa trên các nguyên tắc của microservices, bao gồm các thành phần chính sau:

1. **Client Tier (Frontend):** Ứng dụng React SPA chạy trên trình duyệt của người dùng, cung cấp giao diện cho khách hàng và trang quản trị cho admin.
2. **API Gateway Tier:** Sử dụng Nginx, đóng vai trò là điểm vào duy nhất cho tất cả các yêu cầu từ client, thực hiện định tuyến, xử lý CORS, và có thể đảm nhiệm các chức năng khác như SSL termination.
3. **Application Tier (Backend Microservices):** Bao gồm các microservices độc lập được phát triển bằng Node.js/NestJS: User Service, Product Service, Cart Service, Order Service, Payment Service, Notification Service. Mỗi service có thể có cơ sở dữ liệu riêng hoặc logic nghiệp vụ riêng.
4. **Data Tier:**
   * **PostgreSQL Databases:** Hai instance PostgreSQL (postgres\_db\_user\_product và postgres\_db\_order\_cart) được sử dụng để lưu trữ dữ liệu persistent cho các service.
   * **Redis Cache:** Một instance Redis (redis\_cache) được sử dụng để caching dữ liệu sản phẩm và lưu trữ thông tin giỏ hàng.
5. **Messaging Tier:** Sử dụng RabbitMQ (rabbitmq\_queue) để cho phép giao tiếp bất đồng bộ giữa các microservices, ví dụ như giữa Order Service và Notification Service.
6. **Logging Tier:** Sử dụng Loki, Promtail, và Grafana để thu thập, lưu trữ và trực quan hóa log từ tất cả các thành phần của hệ thống.

A diagram of a company

AI-generated content may be incorrect.

Figure 7: Sơ đồ kiến trúc tổng thể hệ thống Nền tảng Đặt Bánh Ngọt Online

*Hình 3.1: Sơ đồ kiến trúc tổng thể hệ thống Nền tảng Đặt Bánh Ngọt Online*

*(Sơ đồ này sẽ thể hiện rõ các tầng: Client (Browser) -> API Gateway (Nginx) -> các Microservices (User, Product, Cart, Order, Payment, Notification). Mỗi microservice sẽ được vẽ liên kết đến các tài nguyên mà nó sử dụng (ví dụ: Product Service nối với postgres\_db\_user\_product và redis\_cache; Order Service nối với postgres\_db\_order\_cart và rabbitmq\_queue; Notification Service nối với rabbitmq\_queue và Mail Server). RabbitMQ sẽ có các mũi tên từ các service publisher đến nó và từ nó đến các service consumer. Loki, Promtail, Grafana cũng sẽ được thể hiện trong sơ đồ, với Promtail thu thập log từ các services và gửi đến Loki, Grafana đọc từ Loki.)*

### *3.2.2. Thiết kế API cho từng Microservice (API Design)*

Các API của từng microservice được thiết kế theo nguyên tắc RESTful, sử dụng các phương thức HTTP chuẩn (GET, POST, PUT, DELETE, PATCH) và định dạng dữ liệu JSON cho request và response. Việc định nghĩa API rõ ràng là rất quan trọng để đảm bảo sự tương tác hiệu quả giữa frontend và backend, cũng như giữa các microservices (nếu có giao tiếp đồng bộ).

* **User Service API:**
  + POST /api/auth/register: Đăng ký người dùng mới.
  + POST /api/auth/login: Đăng nhập người dùng, trả về JWT.
  + GET /api/users/profile: Lấy thông tin hồ sơ người dùng hiện tại (yêu cầu JWT).
  + PUT /api/users/profile: Cập nhật thông tin hồ sơ người dùng (yêu cầu JWT).
  + GET /api/users: Lấy danh sách người dùng (Admin, yêu cầu JWT).
  + GET /api/users/admin/stats/overview: Lấy thống kê về người dùng cho admin.
* **Product Service API:**
  + GET /api/products: Lấy danh sách sản phẩm (hỗ trợ phân trang, lọc, sắp xếp).
  + GET /api/products/{id}: Lấy chi tiết một sản phẩm.
  + POST /api/products: Tạo sản phẩm mới (Admin, yêu cầu JWT).
  + PUT /api/products/{id}: Cập nhật sản phẩm (Admin, yêu cầu JWT).
  + DELETE /api/products/{id}: Xóa sản phẩm (Admin, yêu cầu JWT).
  + GET /api/categories: Lấy danh sách danh mục.
  + POST /api/categories: Tạo danh mục mới (Admin, yêu cầu JWT).
  + PUT /api/categories/{id}: Cập nhật danh mục (Admin, yêu cầu JWT).
  + DELETE /api/categories/{id}: Xóa danh mục (Admin, yêu cầu JWT).
  + GET /api/products/admin/stats/overview: Lấy thống kê về sản phẩm cho admin.
* **Cart Service API:**
  + GET /api/cart: Lấy thông tin giỏ hàng của người dùng hiện tại (yêu cầu JWT).
  + POST /api/cart/items: Thêm một sản phẩm vào giỏ hàng (yêu cầu JWT).
  + PUT /api/cart/items/{itemId}: Cập nhật số lượng sản phẩm trong giỏ hàng (yêu cầu JWT).
  + DELETE /api/cart/items/{itemId}: Xóa một sản phẩm khỏi giỏ hàng (yêu cầu JWT).
  + DELETE /api/cart: Xóa toàn bộ giỏ hàng (yêu cầu JWT).
* **Order Service API:**
  + POST /api/orders: Tạo đơn hàng mới từ giỏ hàng (yêu cầu JWT).
  + GET /api/orders: Lấy lịch sử đơn hàng của người dùng hiện tại (yêu cầu JWT).
  + GET /api/orders/{orderId}: Lấy chi tiết một đơn hàng (yêu cầu JWT).
  + GET /api/orders/admin: Lấy danh sách tất cả đơn hàng (Admin, yêu cầu JWT).
  + PUT /api/orders/admin/{orderId}/status: Cập nhật trạng thái đơn hàng (Admin, yêu cầu JWT).
  + GET /api/orders/admin/stats/overview: Lấy thống kê về đơn hàng cho admin.
* **Payment Service API:**
  + POST /api/payment/vnpay/create-payment-url: Tạo URL thanh toán VNPay (mô phỏng).
  + GET /api/payment/vnpay/return: Xử lý khi VNPay trả về (Return URL).
  + GET /api/payment/vnpay/ipn: Xử lý thông báo IPN từ VNPay.
  + GET /api/payment/health: Health check endpoint.

### *3.2.3. Thiết kế cơ sở dữ liệu (Database Design)*

* **Cơ sở dữ liệu postgres\_db\_user\_product (cho User Service và Product Service):**
  + **Bảng Users:** user\_id (PK, UUID), full\_name (VARCHAR), email (VARCHAR, UNIQUE), password\_hash (VARCHAR), phone\_number (VARCHAR, NULLABLE), address (TEXT, NULLABLE), role (VARCHAR, DEFAULT 'customer'), created\_at (TIMESTAMP), updated\_at (TIMESTAMP).
  + **Bảng Products:** product\_id (PK, UUID), name (VARCHAR), description (TEXT), price (DECIMAL), image\_urls (TEXT[] hoặc JSONB), category\_id (FK đến Categories), stock\_quantity (INTEGER, DEFAULT 0), status (VARCHAR, DEFAULT 'active'), created\_at (TIMESTAMP), updated\_at (TIMESTAMP).
  + **Bảng Categories:** category\_id (PK, UUID), name (VARCHAR, UNIQUE), description (TEXT, NULLABLE), created\_at (TIMESTAMP), updated\_at (TIMESTAMP).
* **Cơ sở dữ liệu postgres\_db\_order\_cart (cho Order Service):**
  + **Bảng Orders:** order\_id (PK, UUID), user\_id (UUID, FK đến Users ở DB kia – hoặc lưu thông tin user\_email trực tiếp), order\_date (TIMESTAMP, DEFAULT NOW()), total\_amount (DECIMAL), status (VARCHAR, DEFAULT 'pending\_payment'), shipping\_address (TEXT), customer\_name (VARCHAR), customer\_phone (VARCHAR), payment\_method (VARCHAR, NULLABLE), transaction\_id (VARCHAR, NULLABLE), created\_at (TIMESTAMP), updated\_at (TIMESTAMP).
  + **Bảng OrderItems:** order\_item\_id (PK, UUID), order\_id (FK đến Orders), product\_id (UUID – hoặc lưu product\_name, product\_price trực tiếp để tránh phụ thuộc), product\_name (VARCHAR), quantity (INTEGER), unit\_price (DECIMAL), subtotal (DECIMAL), created\_at (TIMESTAMP).
* **Cấu trúc dữ liệu trong Redis (cho Cart Service và Product Service Cache):**
  + **Giỏ hàng:** Key: cart:{user\_id} (Hash). Fields trong hash: {product\_id}: {quantity}.
  + **Cache sản phẩm:** Key: product:{product\_id} (String/JSON). Value: Chuỗi JSON chứa thông tin chi tiết sản phẩm. TTL được thiết lập.
  + **Cache danh sách sản phẩm:** Key: products:category:{category\_id}:page:{page\_number} (String/JSON). Value: Chuỗi JSON chứa danh sách sản phẩm. TTL được thiết lập.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8: Sơ đồ ERD cho cơ sở dữ liệu postgres\_db\_user\_product

*(Hình 3.2: Sơ đồ ERD cho cơ sở dữ liệu postgres\_db\_user\_product)*

*(Sơ đồ này sẽ thể hiện các bảng Users, Products, Categories và mối quan hệ giữa chúng, ví dụ: một Category có nhiều Products.)*

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9: Sơ đồ ERD cho cơ sở dữ liệu postgres\_db\_order\_cart

*(Hình 3.3: Sơ đồ ERD cho cơ sở dữ liệu postgres\_db\_order\_cart)*

*(Sơ đồ này sẽ thể hiện các bảng Orders, OrderItems và mối quan hệ giữa chúng, ví dụ: một Order có nhiều OrderItems.)*

### *3.2.4. Thiết kế giao tiếp giữa các services*

**Giao tiếp đồng bộ (Synchronous Communication):**

* + Chủ yếu là từ Frontend -> API Gateway (Nginx) -> Microservice tương ứng (HTTP/REST).
  + Trong một số trường hợp, một microservice có thể cần gọi đồng bộ đến một microservice khác thông qua API Gateway hoặc trực tiếp (nếu trong cùng mạng nội bộ và được thiết kế cẩn thận). Tuy nhiên, cần hạn chế giao tiếp đồng bộ trực tiếp giữa các service để tránh coupling chặt chẽ. Ví dụ, Order Service khi tạo đơn hàng có thể cần lấy thông tin giá sản phẩm mới nhất từ Product Service.

**Giao tiếp bất đồng bộ (Asynchronous Communication):**

* + Sử dụng RabbitMQ.
  + **Luồng tạo đơn hàng và thông báo:** Order Service (Producer) publish message order.created (chứa orderId, userEmail) lên exchange orders\_exchange. Notification Service (Consumer) lắng nghe từ queue notification\_email\_queue (đã binding với orders\_exchange qua routing key order.created) để gửi email.
  + **Luồng xử lý thanh toán:** Payment Service (Producer) publish message payment.processed (chứa orderId, status) lên exchange payments\_exchange. Order Service (Consumer) lắng nghe từ queue order\_payment\_update\_queue để cập nhật trạng thái đơn hàng.
  + **Thiết kế Message Format:** Các message gửi qua RabbitMQ thường ở định dạng JSON, chứa các thông tin cần thiết để service consumer có thể xử lý. Ví dụ:

// Message cho order.created

{

"eventType": "order.created",

"timestamp": "2023-10-27T10:00:00Z",

"data": {

"orderId": "uuid-order-123",

"userEmail": "customer@example.com",

"totalAmount": 150000

}

}

### 3.2.5. Thiết kế giao diện người dùng (UI/UX Design)

Giai đoạn này tập trung vào việc phác thảo cấu trúc và giao diện trực quan cho cả trang người dùng và trang quản trị. Mục tiêu là tạo ra một trải nghiệm người dùng thân thiện, dễ sử dụng và thẩm mỹ.

**Wireframes/Mockups:**

* + **Trang chủ:** Bố cục chung, banner, danh sách sản phẩm nổi bật/mới, lời kêu gọi hành động.
  + **Trang danh sách sản phẩm:** Cách hiển thị sản phẩm (grid/list), bộ lọc (theo danh mục, giá), sắp xếp, phân trang, thanh tìm kiếm.
  + **Trang chi tiết sản phẩm:** Hiển thị hình ảnh sản phẩm (có thể có gallery), tên, giá, mô tả chi tiết, nút "Thêm vào giỏ hàng".
  + **Trang giỏ hàng:** Danh sách sản phẩm trong giỏ, số lượng, giá, tổng tiền, nút "Tiến hành thanh toán".
  + **Trang thanh toán (Checkout):** Form điền thông tin giao hàng, lựa chọn phương thức thanh toán, tóm tắt đơn hàng.
  + **Trang quản trị Dashboard:** Hiển thị các số liệu thống kê tổng quan.
  + **Trang quản trị sản phẩm:** Bảng danh sách sản phẩm, các nút thao tác (Thêm, Sửa, Xóa), form thêm/sửa sản phẩm.

**User Flow (Luồng người dùng):**

* + Vẽ sơ đồ các bước mà người dùng sẽ thực hiện cho các tác vụ quan trọng như: Đặt hàng, Đăng ký tài khoản, Quản trị viên thêm sản phẩm mới.

## 3.3. Cài đặt (Implementation)

Đây là giai đoạn hiện thực hóa các thiết kế đã được định hình ở các bước trước, bao gồm việc viết code cho backend microservices, frontend, cấu hình các thành phần cơ sở hạ tầng.

### *3.3.1. Thiết lập môi trường phát triển*

* **Cài đặt các công cụ cần thiết:** Node.js (với phiên bản LTS), npm/yarn, Docker Desktop (bao gồm Docker Engine và Docker Compose), một IDE (như Visual Studio Code), Git.
* **Cấu hình file .env:** Tạo file .env ở thư mục gốc của dự án (hoặc trong từng service nếu cần) để lưu trữ các biến môi trường nhạy cảm như credentials cho cơ sở dữ liệu, RabbitMQ, JWT secret, thông tin cổng thanh toán VNPay (mô phỏng), thông tin mail server, v.v. Các biến này sẽ được đọc bởi các service khi khởi chạy thông qua docker-compose.yml. Ví dụ:

# .env

DB\_USER=admin

DB\_PASSWORD=password

DB\_NAME=ecommerce\_db

DB\_PORT=5432

DB\_UP\_USER=user\_product\_admin

# ... các biến khác ...

JWT\_SECRET=YourSuperSecretKeyForJWT

RABBITMQ\_USER=rabbit\_user

RABBITMQ\_PASS=rabbit\_pass

* **Khởi tạo dự án và cấu trúc thư mục:**
  + Tạo thư mục gốc cho toàn bộ dự án (ví dụ: ecommerce-distributed-local).
  + Bên trong, tạo các thư mục con cho từng thành phần: services (chứa code của các microservices backend), api-gateway (chứa file nginx.conf), ecommerce-frontend (chứa code React).
  + Trong thư mục services, mỗi microservice (ví dụ: user-service, product-service) sẽ là một dự án Node.js/NestJS riêng, có package.json, Dockerfile riêng.

### *3.3.2. Phát triển Backend Microservices (Node.js/NestJS)*

Quá trình phát triển từng microservice bao gồm các bước chính:

* **Khởi tạo dự án NestJS:** Sử dụng Nest CLI (nest new <service-name>).
* **Định nghĩa Modules, Controllers, Services:**
  + **Modules:** Nhóm các controllers, services, providers liên quan (ví dụ: AuthModule, UsersModule trong User Service).
  + **Controllers:** Xử lý các HTTP request đến, gọi các services tương ứng để thực hiện logic nghiệp vụ, và trả về HTTP response. Sử dụng decorators (@Controller, @Get, @Post, @Body, @Param, @Query, @UseGuards) để định nghĩa routes và xử lý request.
  + **Services:** Chứa logic nghiệp vụ chính của ứng dụng, tương tác với cơ sở dữ liệu (thông qua ORM như TypeORM nếu dùng), cache, hoặc các service khác.
* **Tích hợp Cơ sở dữ liệu (TypeORM với PostgreSQL):**
  + Cài đặt typeorm, @nestjs/typeorm, pg.
  + Định nghĩa các Entities (tương ứng với các bảng trong DB).
  + Cấu hình kết nối DB trong AppModule (đọc từ biến môi trường).
  + Sử dụng Repositories để thực hiện các thao tác CRUD với DB.
* **Triển khai Xác thực JWT (User Service):**
  + Sử dụng Passport.js với passport-jwt strategy.
  + Tạo JwtStrategy để validate token.
  + Tạo JwtAuthGuard để bảo vệ các routes yêu cầu xác thực.
  + Trong AuthService, tạo hàm login để tạo JWT khi đăng nhập thành công, và hàm register để tạo người dùng mới.
* **Tích hợp Redis Cache (Product Service, Cart Service):**
  + Sử dụng thư viện ioredis.
  + Trong ProductService, triển khai logic cache-aside cho việc lấy thông tin sản phẩm.
  + Trong CartService, sử dụng Redis Hashes để lưu trữ và quản lý giỏ hàng.
* **Tích hợp RabbitMQ (Order Service, Payment Service, Notification Service):**
  + Sử dụng thư viện amqplib.
  + Tạo các service hoặc helper để publish message lên các exchange/queue cụ thể.
  + Trong NotificationService (hoặc các consumer khác), tạo các hàm để consume message từ các queue và xử lý logic tương ứng.
* **Xử lý Validation và Error Handling:**
  + Sử dụng class-validator và class-transformer để validate request DTOs.
  + Sử dụng các Exception Filters của NestJS để xử lý lỗi một cách nhất quán và trả về response lỗi phù hợp.
* **Viết Dockerfile cho từng service:** Để build Docker image cho service đó.

*(Một trích đoạn code minh họa từ một service, ví dụ, cách ProductService sử dụng Redis cache, sẽ được trình bày ở đây.)*

// Ví dụ trích đoạn từ ProductService (NestJS)

import { Injectable, Inject } from '@nestjs/common';

import { InjectRepository } from '@nestjs/typeorm';

import { Repository } from 'typeorm';

import { Product } from './entities/product.entity';

import { Redis } from 'ioredis'; // Giả sử đã inject Redis client

@Injectable()

export class ProductService {

constructor(

@InjectRepository(Product)

private productRepository: Repository<Product>,

@Inject('REDIS\_CLIENT') private readonly redisClient: Redis,

) {}

async findOne(id: string): Promise<Product | null> {

const cacheKey = `product:${id}`;

const cachedProduct = await this.redisClient.get(cacheKey);

if (cachedProduct) {

console.log(`Cache HIT for product: ${id}`);

return JSON.parse(cachedProduct) as Product;

}

console.log(`Cache MISS for product: ${id}`);

const product = await this.productRepository.findOneBy({ product\_id: id });

if (product) {

// Cache product for 1 hour (3600 seconds)

await this.redisClient.set(cacheKey, JSON.stringify(product), 'EX', 3600);

}

return product;

}

// ... other methods

}

*3.3.3. Phát triển API Gateway (Nginx)*

* Tạo file nginx.conf trong thư mục api-gateway.
* Định nghĩa các upstream servers, trỏ đến tên service và port của các microservices backend như đã định nghĩa trong docker-compose.yml.
* Cấu hình các location blocks để định tuyến các request dựa trên URL path đến các upstream tương ứng.
* Thêm các header cần thiết cho CORS (Access-Control-Allow-Origin, Access-Control-Allow-Methods, etc.) trong các location blocks hoặc ở mức server để cho phép frontend giao tiếp.
* Tạo Dockerfile cho api-gateway để copy file nginx.conf vào image Nginx chuẩn.

### *3.3.4. Phát triển Frontend (React/TypeScript)*

* **Khởi tạo dự án React:** Sử dụng Create React App với template TypeScript (npx create-react-app ecommerce-frontend --template typescript).
* **Cấu trúc thư mục:** Tổ chức code thành các thư mục components (UI components tái sử dụng), pages (các trang chính của ứng dụng), services (các hàm gọi API), contexts (quản lý state với Context API), layouts (UserLayout, AdminLayout), router (cấu hình routes), assets, hooks, utils.
* **Xây dựng UI Components:** Tạo các component như Navbar, Footer, ProductCard, CartItem, AdminSidebar, AdminHeader sử dụng JSX/TSX và Tailwind CSS (hoặc thư viện UI khác).
* **Xây dựng Pages:** Kết hợp các components để tạo ra các trang hoàn chỉnh như HomePage, ProductListPage, ProductDetailPage, LoginPage, RegisterPage, CartPage, CheckoutPage, AdminDashboardPage, AdminProductListPage, v.v.
* **Quản lý Routing:** Sử dụng react-router-dom để định nghĩa các routes cho public, private (yêu cầu đăng nhập), và admin (yêu cầu quyền admin). Tạo các Route Guard components (PrivateRoute, AdminRoute) để bảo vệ các routes này.
* **Quản lý State:** Sử dụng AuthContext để quản lý trạng thái đăng nhập của người dùng, thông tin user, và token JWT. Đối với các state cục bộ hoặc state phức tạp hơn trong một phần của ứng dụng, có thể sử dụng useState, useReducer, hoặc các thư viện quản lý state khác nếu cần.
* **Tích hợp API:** Viết các hàm trong thư mục services (sử dụng Axios) để gọi đến các API backend thông qua API Gateway. Xử lý việc gửi token JWT trong header Authorization cho các request yêu cầu xác thực. Xử lý response và lỗi từ API.
* **Styling:** Sử dụng Tailwind CSS để áp dụng style trực tiếp trong JSX. Viết thêm CSS tùy chỉnh nếu cần.
* **Triển khai Responsive Design:** Đảm bảo giao diện hiển thị tốt trên các kích thước màn hình khác nhau (desktop, tablet, mobile).

*(Một trích đoạn code React minh họa cách gọi API hoặc sử dụng AuthContext sẽ được trình bày ở đây.)*

// Ví dụ trích đoạn từ một component React gọi API sản phẩm

import React, { useEffect, useState } from 'react';

import axiosInstance from '../services/apiClient'; // Giả sử apiClient đã được cấu hình với baseURL và interceptors

interface Product {

product\_id: string;

name: string;

price: number;

// ... other fields

}

const ProductList: React.FC = () => {

const [products, setProducts] = useState<Product[]>([]);

const [loading, setLoading] = useState(true);

const [error, setError] = useState<string | null>(null);

useEffect(() => {

const fetchProducts = async () => {

try {

setLoading(true);

const response = await axiosInstance.get('/products'); // Gọi qua API Gateway

setProducts(response.data.data); // Giả sử API trả về { data: [...] }

setError(null);

} catch (err: any) {

setError(err.message || 'Failed to fetch products');

} finally {

setLoading(false);

}

};

fetchProducts();

}, []);

if (loading) return <p>Loading products...</p>;

if (error) return <p>Error: {error}</p>;

return (

<div>

<h1>Product List</h1>

<ul>

{products.map((product) => (

<li key={product.product\_id}>

{product.name} - ${product.price}

</li>

))}

</ul>

</div>

);

};

export default ProductList;

### 3.3.5. Triển khai Logging Stack (Loki, Promtail, Grafana)

* **Cấu hình Loki:** Tạo file loki-config.yaml (hoặc sử dụng cấu hình mặc định nếu phù hợp) để định nghĩa cách Loki lưu trữ log.
* **Cấu hình Promtail:** Tạo file promtail-config.yml. Trong file này, cấu hình scrape\_configs để Promtail:
  + Phát hiện các Docker container đang chạy (thông qua docker\_sd\_configs).
  + Gắn các labels cần thiết cho log streams (ví dụ: container\_name, job).
  + Chỉ định địa chỉ của Loki server để Promtail gửi log đến.
* **Cấu hình docker-compose.yml:**
  + Thêm các services loki, promtail, grafana.
  + Mount các file config đã tạo vào các container tương ứng.
  + Mount Docker socket (/var/run/docker.sock) vào container Promtail để nó có thể đọc log từ các container khác.
  + Cấu hình Grafana để tự động thêm Loki làm datasource (qua biến môi trường GF\_DATASOURCES\_\*).
* **Cấu hình Grafana:** Sau khi Grafana khởi chạy, truy cập giao diện web (thường là http://localhost:3000), kiểm tra datasource Loki đã được thêm. Sử dụng tab "Explore" để bắt đầu truy vấn log bằng LogQL.

## 3.4. Hiệu chỉnh (Refinement & Debugging)

Đây là một quá trình lặp đi lặp lại trong suốt vòng đời phát triển:

* **Gỡ lỗi (Debugging):** Dựa trên kết quả từ các giai đoạn kiểm thử (unit, integration, UAT) hoặc các lỗi phát sinh trong quá trình phát triển, tiến hành xác định nguyên nhân và sửa lỗi. Sử dụng các công cụ debugger của IDE, log từ hệ thống (đã có Loki/Grafana), và các kỹ thuật gỡ lỗi khác.
* **Tối ưu hóa hiệu năng (Performance Optimization):**
  + Nếu phát hiện các điểm nghẽn về hiệu năng (ví dụ: API phản hồi chậm, truy vấn DB tốn thời gian), tiến hành phân tích và tối ưu.
  + Ví dụ: Tối ưu các câu lệnh SQL, điều chỉnh chiến lược caching (thời gian TTL, các key được cache), xem xét việc thêm index cho DB.
  + Trong phạm vi đồ án, việc tối ưu hóa sâu có thể không phải là trọng tâm chính, nhưng cần ý thức về các yếu tố ảnh hưởng đến hiệu năng.
* **Cải thiện UI/UX:** Dựa trên phản hồi từ UAT hoặc tự đánh giá, có thể cần điều chỉnh lại giao diện, luồng người dùng để tăng tính thân thiện và dễ sử dụng.

Quá trình này đòi hỏi sự kiên nhẫn, kỹ năng phân tích vấn đề và khả năng làm việc lặp đi lặp lại để dần hoàn thiện sản phẩm.

# **PHẦN 4: TRIỂN KHAI & KIỂM THỬ**

## 4.1. Công nghệ & môi trường

***Backend Framework & Runtime***

Hệ thống backend được xây dựng trên NestJS 11.x - một framework Node.js hiện đại áp dụng kiến trúc modular và dependency injection, kết hợp với Node.js 20 LTS (Alpine Linux variant) để đảm bảo hiệu năng và bảo mật. Toàn bộ backend được viết bằng TypeScript 5.7, mang lại type safety và developer experience tốt hơn. Các microservices sử dụng Express.js (thông qua NestJS platform-express) làm HTTP server với performance cao.

***Database & Storage***

Database chính là PostgreSQL 14-alpine, triển khai theo mô hình Database-per-Service với 3 instances độc lập: (1) ecommerce\_up\_db cho User, Product, Category, Review và CakeOption; (2) ecommerce\_oc\_db cho Order và OrderItem; (3) instance dự trữ cho mở rộng tương lai. ORM framework sử dụng TypeORM 0.3.22 để mapping entities với database, hỗ trợ migrations và query builder. Giỏ hàng được lưu trữ trên Redis 6-alpine (in-memory cache) với cấu trúc key-value, TTL 7 ngày, đảm bảo hiệu năng đọc/ghi cao và khả năng scale horizontal.

***Authentication & Security***

Xác thực người dùng sử dụng JWT (JSON Web Token) thông qua thư viện @nestjs/jwt 11.x và Passport.js với strategies jwt và local. Mật khẩu được hash bằng bcrypt 5.1 với salt rounds là 10. Authorization áp dụng mô hình RBAC (Role-Based Access Control) với 2 roles: USER và ADMIN, được enforce thông qua NestJS Guards (JwtAuthGuard, AdminGuard). Input validation sử dụng class-validator và class-transformer để ngăn chặn SQL Injection, XSS và các lỗ hổng bảo mật phổ biến.

***Message Queue & Asynchronous Processing***

Giao tiếp bất đồng bộ giữa các services sử dụng RabbitMQ 3.13-management-alpine làm message broker. Payment Service đóng vai trò publisher gửi events "payment\_processed", Order Service và Notification Service đóng vai trò consumers xử lý các events này. Message queue đảm bảo eventual consistency, retry mechanism (3 attempts), và có management UI trên port 15672 để monitoring.

***Frontend Technology Stack***

Frontend là một Single Page Application (SPA) được xây dựng bằng React 19.x với TypeScript, sử dụng Vite 6.3 làm build tool và dev server (HMR support). UI framework chính là Tailwind CSS 4.1 cho utility-first styling, kết hợp Framer Motion 12.x cho animations mượt mà. State management sử dụng Context API (AuthContext, CartContext) và Redux Toolkit 2.7 cho global state phức tạp. Routing được xử lý bởi React Router DOM 7.5, API calls sử dụng Axios 1.9, và charts/analytics dùng Recharts 2.15. Các icon components từ React Icons 5.5 và Heroicons 1.0 cung cấp UI elements nhất quán.

***API Gateway & Reverse Proxy***

Nginx (Alpine variant) đóng vai trò API Gateway, cung cấp reverse proxy cho 6 backend services với upstream configuration. Nginx xử lý CORS headers (Access-Control-Allow-Origin: http://localhost:5173), load balancing (round-robin ready), request logging, và có thể cấu hình rate limiting. Routing rules: /api/users → user\_service:3001, /api/products|categories|reviews|cake-options → product\_service:3002, /api/cart → cart\_service:3003, /api/orders → order\_service:3004, /api/payment → payment\_service:3005.

***Payment Gateway Integration***

Tích hợp VNPay Payment Gateway (sandbox mode) cho thanh toán trực tuyến. Sử dụng HMAC SHA256 để tạo và verify signature, hỗ trợ các phương thức thanh toán: ATM nội địa, Visa/MasterCard. Webhook callback xử lý payment result với secure validation. Ngrok tunneling được sử dụng để expose local webhook endpoint cho VNPay callback trong môi trường development.

***Email Service***

Notification Service sử dụng NodeMailer để gửi email xác nhận đơn hàng thông qua Gmail SMTP (ssl: port 465 hoặc tls: port 587). Email templates hỗ trợ cả HTML (cho rich formatting) và plain text (fallback). Rate limit: 500 emails/day (Gmail free tier), đủ cho demo và testing.

***Containerization & Orchestration***

Toàn bộ hệ thống được container hóa với Docker, sử dụng Docker Compose (Compose file format v3.8+) để orchestrate 12 containers: 6 microservices + 3 PostgreSQL instances + Redis + RabbitMQ + Nginx. Mỗi service có Dockerfile riêng với multi-stage builds (builder stage + production stage) để tối ưu image size. Images base: node:20-alpine (services), postgres:14-alpine (databases), redis:6-alpine (cache), rabbitmq:3.13-management-alpine (message queue), nginx:alpine (gateway). Health checks được cấu hình cho tất cả services (interval: 10-15s, retries: 3-5) để đảm bảo high availability.

***Development Tools & Quality Assurance***

Code quality được đảm bảo bởi ESLint 9.x (linting), Prettier 3.4 (formatting), và TypeScript compiler (type checking). Testing framework: Jest 29.7 cho unit tests, Supertest 7.0 cho integration tests của backend, mục tiêu coverage 70%+. API documentation và testing sử dụng Postman, với collection export cho automation testing qua Newman CLI. Version control: Git với conventional commits, branching strategy: main, develop, feature/.

Logging & Monitoring

Logging sử dụng NestJS built-in Logger với Winston transport (optional). Log levels: ERROR, WARN, INFO, DEBUG. Request logging thông qua Morgan middleware (format: combined). Docker logs được aggregate và có thể view bằng docker logs [container\_name] hoặc centralized logging với Promtail + Loki (config có sẵn trong promtail-config.yml). Health check endpoints (/health, /users/health, /products/health, etc.) cho monitoring service availability.

Deployment Environment

Môi trường triển khai: Local development với Docker Compose. Yêu cầu hệ thống: Docker Desktop 4.x+, RAM tối thiểu 8GB (recommended 16GB), CPU 4 cores, Disk space 20GB+. Operating systems hỗ trợ: Windows 10/11 (WSL2), macOS (Apple Silicon/Intel), Linux (Ubuntu 20.04+). Port mapping: Services 3001-3005, Databases 5432/5434/5435, Redis 6379, RabbitMQ 5672/15672, Nginx 80, Frontend 5173 (Vite dev server). Network: Docker bridge network cake\_shop\_network cho internal service communication.

***Utility Libraries***

Backend utilities: date-fns 4.1 (date manipulation), class-validator và class-transformer (DTO validation), pg 8.15 (PostgreSQL driver), amqplib 0.10 (RabbitMQ client), passport-jwt 4.0 và passport-local 1.0 (authentication strategies). Frontend utilities: lodash 4.17 (utility functions), date-fns 4.1 (date formatting), aos 2.3 (animate on scroll), react-countup 6.5 (number animations), react-slick 0.30 (carousel).

## 4.2. Kết quả các chức năng đạt được

### 4.2.1. Giao diện người dùng (Khách hàng)

Phần này giới thiệu các màn hình giao diện chính mà khách hàng tương tác khi sử dụng nền tảng đặt bánh ngọt trực tuyến.

* **Hình 4.1: Màn hình Trang chủ**
  + *(Mô tả: Đây là ảnh chụp màn hình trang chủ của ứng dụng. Trang chủ được thiết kế với bố cục trực quan, bao gồm thanh điều hướng chính (logo, menu, tìm kiếm, giỏ hàng, nút đăng nhập/đăng ký hoặc thông tin người dùng đã đăng nhập), một banner quảng cáo hoặc hình ảnh nổi bật về các loại bánh. Bên dưới có thể là các sección hiển thị sản phẩm mới, sản phẩm bán chạy, hoặc các danh mục sản phẩm chính. Footer chứa các thông tin liên hệ, liên kết hữu ích.)*

*A screenshot of a website

AI-generated content may be incorrect.*

Figure 10: Giao diện Trang chủ của khách hàng *Hình 4.1: Giao diện Trang chủ của khách hàng*

* **Hình 4.2: Màn hình Danh sách sản phẩm**
  + *(Mô tả: Ảnh chụp màn hình trang hiển thị danh sách các sản phẩm bánh ngọt. Giao diện này cho phép người dùng xem nhiều sản phẩm cùng lúc, thường dưới dạng lưới (grid) hoặc danh sách (list). Có các tùy chọn để lọc sản phẩm theo danh mục (ví dụ: Bánh Kem, Bánh Mì Ngọt), sắp xếp theo giá (tăng/giảm dần), tên sản phẩm. Thanh tìm kiếm cũng có thể xuất hiện ở đây. Mỗi sản phẩm hiển thị hình ảnh thu nhỏ, tên, giá và nút "Thêm vào giỏ hàng" hoặc "Xem chi tiết".)*

A screenshot of a website

AI-generated content may be incorrect.

Figure 11: Giao diện Trang danh sách sản phẩm với các tùy chọn lọc và sắp xếp  
*Hình 4.2: Giao diện Trang danh sách sản phẩm với các tùy chọn lọc và sắp xếp*

* **Hình 4.3: Màn hình Chi tiết sản phẩm**
  + *(Mô tả: Ảnh chụp màn hình trang hiển thị thông tin chi tiết của một sản phẩm cụ thể. Trang này bao gồm hình ảnh lớn của sản phẩm (có thể có gallery nhiều ảnh), tên sản phẩm, giá bán, mô tả chi tiết về thành phần, hương vị, kích thước (nếu có). Người dùng có thể chọn số lượng và nhấp vào nút "Thêm vào giỏ hàng".)*

A screenshot of a website

AI-generated content may be incorrect.

Figure 12: Giao diện Trang chi tiết sản phẩm

*Hình 4.3: Giao diện Trang chi tiết sản phẩm*

* **Hình 4.4: Màn hình Giỏ hàng**
  + *(Mô tả: Ảnh chụp màn hình trang giỏ hàng. Tại đây, người dùng có thể xem lại tất cả các sản phẩm đã chọn, bao gồm hình ảnh, tên, đơn giá, số lượng (có thể điều chỉnh số lượng hoặc xóa sản phẩm khỏi giỏ). Tổng số tiền tạm tính của giỏ hàng được hiển thị rõ ràng. Có nút "Tiếp tục mua sắm" và nút "Tiến hành thanh toán".)*

A screenshot of a computer

AI-generated content may be incorrect.

Figure 13: Giao diện Trang giỏ hàng  
*Hình 4.4: Giao diện Trang giỏ hàng*

* **Hình 4.5: Màn hình Thanh toán (Checkout)**
  + *(Mô tả: Ảnh chụp màn hình trang thanh toán. Trang này yêu cầu người dùng cung cấp hoặc xác nhận thông tin giao hàng (tên, địa chỉ, số điện thoại). Hiển thị tóm tắt đơn hàng (danh sách sản phẩm, tổng tiền). Cho phép người dùng lựa chọn phương thức thanh toán (ví dụ: Thanh toán khi nhận hàng - COD nếu có, hoặc mô phỏng thanh toán qua VNPay). Có nút "Xác nhận đặt hàng".)*

A screenshot of a computer

AI-generated content may be incorrect.

Figure 14: Giao diện Trang thanh toán (Checkout)  
*Hình 4.5: Giao diện Trang thanh toán (Checkout)*

* **Hình 4.6: Màn hình Kết quả thanh toán VNPay (mô phỏng) và Xác nhận đơn hàng**
  + *(Mô tả: Ảnh chụp màn hình sau khi người dùng hoàn tất (mô phỏng) quy trình thanh toán trên cổng VNPay và được chuyển hướng trở lại website. Màn hình này sẽ hiển thị thông báo về kết quả giao dịch (thành công hoặc thất bại) và thông tin về đơn hàng vừa được tạo (mã đơn hàng). Có thể có nút "Tiếp tục mua sắm" hoặc "Xem chi tiết đơn hàng".)*

A screenshot of a chat

AI-generated content may be incorrect.

Figure 15: Giao diện thông báo kết quả thanh toán và xác nhận đơn hàng  
*Hình 4.6: Giao diện thông báo kết quả thanh toán và xác nhận đơn hàng*

* **Hình 4.7: Màn hình Lịch sử đơn hàng**
  + *(Mô tả: Ảnh chụp màn hình trang lịch sử đơn hàng của người dùng đã đăng nhập. Hiển thị danh sách các đơn hàng đã đặt, bao gồm mã đơn hàng, ngày đặt, tổng tiền, và trạng thái hiện tại của đơn hàng (ví dụ: Chờ xác nhận, Đang xử lý, Đã giao). Người dùng có thể nhấp vào để xem chi tiết từng đơn hàng.)*

*A screenshot of a computer

AI-generated content may be incorrect.*

Figure 16: Giao diện Trang lịch sử đơn hàng của người dùng

*Hình 4.7: Giao diện Trang lịch sử đơn hàng của người dùng*

* **Hình 4.8: Màn hình Thông tin tài khoản người dùng**
  + *(Mô tả: Ảnh chụp màn hình trang quản lý thông tin cá nhân của người dùng đã đăng nhập. Cho phép người dùng xem và cập nhật các thông tin như họ tên, email, số điện thoại, địa chỉ giao hàng mặc định, và thay đổi mật khẩu.)*

*A screenshot of a computer

AI-generated content may be incorrect.*

Figure 17: Giao diện Trang thông tin tài khoản người dùng  
*Hình 4.8: Giao diện Trang thông tin tài khoản người dùng*

### 4.2.2. Giao diện quản trị (Admin)

Phần này giới thiệu các màn hình giao diện chính của khu vực quản trị, nơi quản trị viên có thể quản lý các khía cạnh khác nhau của hệ thống.

* **Hình 4.9: Màn hình Đăng nhập Admin**
  + *(Mô tả: Ảnh chụp màn hình trang đăng nhập dành riêng cho quản trị viên. Yêu cầu nhập email và mật khẩu của tài khoản admin.)*

*A screenshot of a login form

AI-generated content may be incorrect.*

Figure 18: Giao diện Trang đăng nhập Admin  
*Hình 4.9: Giao diện Trang đăng nhập Admin*

* **Hình 4.10: Màn hình Dashboard Admin**
  + *(Mô tả: Ảnh chụp màn hình trang tổng quan (Dashboard) sau khi quản trị viên đăng nhập thành công. Trang này thường hiển thị các số liệu thống kê nhanh về hoạt động của website, ví dụ: tổng số đơn hàng mới trong ngày, tổng doanh thu, số lượng người dùng mới, các sản phẩm bán chạy. Các số liệu này được lấy từ các API /api/admin/stats/\*.)*

*A screenshot of a computer

AI-generated content may be incorrect.*

Figure 19: Giao diện Dashboard của trang quản trị

*Hình 4.10: Giao diện Dashboard của trang quản trị*

* **Hình 4.11: Màn hình Quản lý người dùng**
  + *(Mô tả: Ảnh chụp màn hình trang quản lý danh sách người dùng. Hiển thị thông tin cơ bản của các tài khoản đã đăng ký như ID, tên, email, vai trò, trạng thái. Có thể có chức năng tìm kiếm, lọc người dùng và các hành động như xem chi tiết, khóa/mở khóa tài khoản.)*

*A white rectangular object with text

AI-generated content may be incorrect.*

Figure 20: Giao diện Trang quản lý người dùng  
*Hình 4.11: Giao diện Trang quản lý người dùng*

* **Hình 4.12: Màn hình Quản lý sản phẩm - Danh sách**
  + *(Mô tả: Ảnh chụp màn hình trang hiển thị danh sách tất cả các sản phẩm bánh ngọt trong hệ thống. Thông tin bao gồm ID, hình ảnh thu nhỏ, tên sản phẩm, danh mục, giá, số lượng tồn (nếu có), trạng thái. Có các nút chức năng như "Thêm sản phẩm mới", "Sửa", "Xóa" cho từng sản phẩm. Hỗ trợ tìm kiếm và lọc sản phẩm.)*

*A screenshot of a computer

AI-generated content may be incorrect.*

Figure 21:Giao diện Trang quản lý sản phẩm - Danh sách

*Hình 4.12: Giao diện Trang quản lý sản phẩm - Danh sách*

* **Hình 4.13: Màn hình Quản lý sản phẩm - Form thêm/sửa**
  + *(Mô tả: Ảnh chụp màn hình form dùng để thêm mới hoặc chỉnh sửa thông tin một sản phẩm. Form bao gồm các trường nhập liệu cho tên sản phẩm, mô tả, giá, chọn danh mục, tải lên hình ảnh, nhập số lượng tồn, và chọn trạng thái.)*

*A screenshot of a computer

AI-generated content may be incorrect.*

Figure 22: Giao diện Form thêm mới/chỉnh sửa sản phẩm

*Hình 4.13: Giao diện Form thêm mới/chỉnh sửa sản phẩm*

* **Hình 4.14: Màn hình Quản lý danh mục**
  + *(Mô tả: Ảnh chụp màn hình trang quản lý các danh mục sản phẩm. Hiển thị danh sách các danh mục hiện có, cho phép quản trị viên thêm mới, sửa tên hoặc mô tả, và xóa danh mục.)*

*A screenshot of a computer

AI-generated content may be incorrect.*

Figure 23: Giao diện Trang quản lý danh mục sản phẩm  
*Hình 4.14: Giao diện Trang quản lý danh mục sản phẩm*

* **Hình 4.15: Màn hình Quản lý đơn hàng - Danh sách**
  + *(Mô tả: Ảnh chụp màn hình trang hiển thị danh sách tất cả các đơn hàng đã được đặt trên hệ thống. Thông tin bao gồm mã đơn hàng, tên khách hàng, ngày đặt, tổng tiền, và trạng thái hiện tại của đơn hàng. Có chức năng tìm kiếm, lọc đơn hàng theo trạng thái, ngày đặt, v.v.)*

*A screenshot of a computer

AI-generated content may be incorrect.*

Figure 24: Giao diện Trang quản lý đơn hàng - Danh sách  
*Hình 4.15: Giao diện Trang quản lý đơn hàng - Danh sách*

* **Hình 4.16: Màn hình Quản lý đơn hàng - Chi tiết**
  + *(Mô tả: Ảnh chụp màn hình trang hiển thị thông tin chi tiết của một đơn hàng cụ thể. Bao gồm thông tin khách hàng, địa chỉ giao hàng, danh sách các sản phẩm đã đặt (tên, số lượng, đơn giá, thành tiền), tổng tiền, lịch sử thay đổi trạng thái. Quản trị viên có thể cập nhật trạng thái đơn hàng tại đây (ví dụ: từ "Chờ xác nhận" sang "Đang xử lý").)*

*A screenshot of a chat

AI-generated content may be incorrect.*

Figure 25: Giao diện Trang chi tiết đơn hàng và cập nhật trạng thái  
*Hình 4.16: Giao diện Trang chi tiết đơn hàng và cập nhật trạng thái*

## 4.3. Kết quả hoạt động của các thành phần Backend và Cơ sở hạ tầng

Phần này trình bày bằng chứng về hoạt động của các thành phần backend và cơ sở hạ tầng đã được triển khai, cho thấy chúng hoạt động đúng như thiết kế.

### *4.3.1. API Gateway (Nginx)*

* + Việc kiểm tra bằng công cụ Postman cho thấy các yêu cầu gửi đến API Gateway (ví dụ: http://localhost/api/products) được định tuyến chính xác đến product\_service và trả về kết quả mong đợi. Các header CORS (Access-Control-Allow-Origin: http://localhost:5173) cũng được Nginx thêm vào response, cho phép frontend giao tiếp thành công. Log truy cập của Nginx (truy cập qua docker logs api\_gateway\_container) cũng ghi nhận các request này.

*A screenshot of a computer

AI-generated content may be incorrect.*

Figure 26: Kết quả kiểm tra API qua Nginx và header CORS  
*Hình 4.17: Kết quả kiểm tra API qua Nginx và header CORS*

### *4.3.2. PostgreSQL Databases*

* + Sử dụng các công cụ quản lý cơ sở dữ liệu như pgAdmin hoặc DBeaver để kết nối đến các instance PostgreSQL (postgres\_db\_up\_container và postgres\_db\_oc\_container) cho thấy các bảng (Users, Products, Categories, Orders, OrderItems) đã được tạo ra đúng theo thiết kế ERD. Dữ liệu được ghi vào các bảng này khi người dùng đăng ký, quản trị viên thêm sản phẩm, hoặc khách hàng đặt hàng.

*A screenshot of a computer

AI-generated content may be incorrect.*

Figure 27: Cấu trúc các bảng dữ liệu trong PostgreSQL  
*Hình 4.18: Cấu trúc các bảng dữ liệu trong PostgreSQL*

### *4.3.3. Health Checks*

* + Lệnh docker-compose ps cho thấy trạng thái của tất cả các container. Các container được cấu hình health check (ví dụ: postgres\_db\_user\_product, redis\_cache, rabbitmq\_queue, và các microservices backend) hiển thị trạng thái (healthy) sau khi đã khởi động và sẵn sàng. Điều này cho thấy cơ chế health check hoạt động, và depends\_on với condition: service\_healthy trong docker-compose.yml đảm bảo thứ tự khởi động đúng.

*A computer screen shot of a program

AI-generated content may be incorrect.*

Figure 28: Kết quả docker-compose ps cho thấy trạng thái healthy của các services  
*Hình 4.19: Kết quả docker-compose ps cho thấy trạng thái healthy của các services*

## 4.4 Kiểm thử (Testing)

Kiểm thử là một giai đoạn quan trọng để đảm bảo chất lượng, tính đúng đắn và độ tin cậy của hệ thống.

### *4.4.1. Kiểm thử tích hợp (Integration Testing)*

* **Mục tiêu:** Kiểm tra sự tương tác giữa các thành phần khác nhau của hệ thống, ví dụ:
  + Frontend gọi API đến Backend qua API Gateway.
  + Một microservice gọi đến một microservice khác.
  + Sự tương tác giữa một microservice và cơ sở dữ liệu của nó.
  + Luồng message qua RabbitMQ (ví dụ: Order Service publish, Notification Service consume và thực hiện hành động).
* **Công cụ và Phương pháp:**
  + **Kiểm thử API:** Sử dụng Postman hoặc Newman (CLI của Postman) để gửi request đến các API endpoint và kiểm tra response (status code, payload).
  + **Kiểm thử luồng bất đồng bộ:** Gửi một request kích hoạt việc publish message (ví dụ: tạo đơn hàng), sau đó kiểm tra xem consumer service (ví dụ: Notification Service) có xử lý message đúng không (ví dụ: kiểm tra email đã được gửi, hoặc log từ consumer service).
  + Đối với các microservices trong NestJS, có thể viết các bài test e2e (end-to-end) sử dụng supertest để kiểm thử toàn bộ luồng request-response của một service, bao gồm cả tương tác với DB (sử dụng test database).

### *4.4.2. Kiểm thử chấp nhận người dùng (User Acceptance Testing - UAT)*

* **Mục tiêu:** Đảm bảo hệ thống đáp ứng được các yêu cầu và mong đợi của người dùng cuối (cả khách hàng và quản trị viên).
* **Phương pháp:** Thực hiện các kịch bản sử dụng (test scenarios) mô phỏng các hành vi thực tế của người dùng.

# **PHẦN 5: TỔNG KẾT & HƯỚNG PHÁT TRIỂN**

## 5.1. Đánh giá và rút kinh nghiệm

### 5.1.1. Mức độ hoàn thành mục tiêu

Đề tài đã hiện thực hóa thành công một nền tảng đặt bánh ngọt trực tuyến theo kiến trúc microservices, bám sát các yêu cầu đã nêu ở phần tổng quan và phương pháp. Hệ thống được phân tách thành các dịch vụ độc lập theo miền nghiệp vụ gồm quản lý người dùng, sản phẩm, giỏ hàng, đơn hàng, thanh toán mô phỏng và thông báo. Toàn bộ các dịch vụ được đóng gói bằng Docker, điều phối bằng Docker Compose, có health check và cấu hình môi trường rõ ràng. Nginx đóng vai trò cổng API duy nhất, định tuyến và xử lý CORS chuẩn mực. Redis được sử dụng cho giỏ hàng và bộ nhớ đệm, còn RabbitMQ phục vụ giao tiếp bất đồng bộ giữa các dịch vụ. Dữ liệu bền vững lưu trên PostgreSQL theo hướng mỗi dịch vụ sở hữu dữ liệu của mình.

Trên giao diện, ứng dụng đơn trang phát triển bằng React và TypeScript đã cung cấp đầy đủ các luồng sử dụng chính cho cả khách hàng và quản trị viên, bao gồm duyệt – tìm kiếm – lọc – đặt hàng – thanh toán mô phỏng – theo dõi đơn, cùng các tác vụ quản trị sản phẩm, danh mục, người dùng và đơn hàng. Về vận hành, bộ ba Loki, Promtail và Grafana đã được triển khai để thu thập và trực quan hóa nhật ký hệ thống, giúp theo dõi và gỡ lỗi thuận tiện trong môi trường nhiều dịch vụ.

Nhìn chung, các mục tiêu cốt lõi của đồ án đã đạt. Các yêu cầu chức năng quan trọng đều có mặt và hoạt động ổn định; các yêu cầu phi chức năng trọng yếu như khả năng mở rộng theo chiều ngang ở cấp dịch vụ, hiệu năng truy xuất nhờ bộ nhớ đệm, cùng khả năng quan sát ở mức nhật ký cũng đã được hiện thực hóa ở mức phù hợp với phạm vi học thuật.

### 5.1.2. Điểm mạnh

Thứ nhất, cách phân tách theo “ngữ cảnh giới hạn” giúp biên giới trách nhiệm từng dịch vụ rõ ràng, giảm liên kết chặt và tạo tiền đề mở rộng độc lập. Thứ hai, sử dụng Redis cho giỏ hàng và bộ nhớ đệm sản phẩm đem lại thời gian phản hồi tốt, giảm tải cơ sở dữ liệu, nâng trải nghiệm người dùng. Thứ ba, đưa RabbitMQ vào các tình huống không yêu cầu phản hồi tức thì (như gửi thư xác nhận) đã làm mượt luồng xử lý và tăng tính chịu lỗi. Thứ tư, cổng Nginx giúp đơn giản hóa phía khách, quy tụ chính sách chung như CORS, và chuẩn bị cho các năng lực nâng cao như chặn tần suất và chấm dứt SSL khi triển khai sản xuất. Cuối cùng, việc container hóa toàn bộ cho phép “tính đồng nhất môi trường”, rút ngắn thời gian khởi tạo và chuyển giao.

### 5.1.3. Hạn chế

Hệ thống chưa có lớp giám sát chỉ số (metrics) chuyên sâu và cảnh báo tự động; quan sát hiện dừng ở nhật ký tập trung, thiếu truy vết phân tán và chỉ số độ trễ phần trăm vị (p95, p99). Tính nhất quán dữ liệu liên dịch vụ đang dựa chủ yếu vào nhất quán cuối cùng, chưa áp dụng các mẫu điều phối giao dịch phân tán như Saga ở mức hệ thống. Quy trình xây dựng – kiểm thử – triển khai còn thủ công, thiếu đường ống hội nhập – triển khai liên tục. Ngoài ra, bảo mật hiện ở mức nền tảng (xác thực bằng mã thông báo và CORS); các biện pháp phòng vệ nâng cao như tường lửa ứng dụng web, phát hiện – ngăn chặn xâm nhập, nhật ký kiểm toán chi tiết chưa được triển khai. Cuối cùng, một số tính năng thương mại điện tử nâng cao như khuyến mãi, khách hàng thân thiết, gợi ý sản phẩm và quản trị tồn kho chuyên sâu chưa có trong phiên bản hiện tại.

### 5.1.4. Bài học kinh nghiệm

Kiến trúc microservices đem lại lợi ích rõ rệt về tính mô-đun và khả năng mở rộng, nhưng cái giá phải đánh đổi là độ phức tạp vận hành và kiểm thử. Việc đầu tư sớm cho quan sát hệ thống (nhật ký, chỉ số, truy vết), kiểm thử tích hợp và kiểm thử hợp đồng giữa các dịch vụ sẽ tiết kiệm đáng kể chi phí khắc phục về sau. Ngoài ra, API Gateway giúp thống nhất chính sách mặt cắt ngang, nhưng hợp đồng API cần được quản lý có phiên bản và giao tiếp “API-first” để tránh gây gián đoạn giữa nhà cung cấp và bên tiêu thụ dịch vụ. Ở khía cạnh dữ liệu, việc chấp nhận ảnh chụp dữ liệu theo thời điểm trong đơn hàng là thực tế, đổi lại sự chính xác lịch sử và giảm phụ thuộc liên dịch vụ. Cuối cùng, container hóa và hạ tầng dưới dạng mã là nền tảng tốt, song để đi vào thực tế sản xuất cần sớm hóa tự động hóa quy trình và chuẩn hóa quản lý bí mật.

## 5.2. Hướng phát triển

### 5.2.1. Cải tiến kỹ thuật trọng tâm

*Giao dịch phân tán theo mẫu Saga.* Chuẩn hóa chuỗi tác vụ khi đặt hàng và xác định giao dịch bù trừ cho từng bước, giúp hệ thống tự phục hồi khi có lỗi giữa chừng và nâng tính nhất quán toàn cục. Có thể bắt đầu bằng mô hình điều phối (orchestration) để dễ kiểm soát nhật ký thực thi.

*Quan sát và cảnh báo toàn diện.* Bổ sung thu thập chỉ số với Prometheus, trực quan hóa trên Grafana, thiết lập cảnh báo theo ngưỡng cho trạng thái dịch vụ, tỉ lệ lỗi, độ trễ p95/p99, tài nguyên hệ thống. Kết hợp truy vết phân tán (OpenTelemetry, Jaeger) và Mã định danh tương quan để lần vết một yêu cầu xuyên suốt các dịch vụ.

*Tự động hóa xây dựng và triển khai.* Thiết kế đường ống hội nhập – triển khai liên tục với kiểm tra chất lượng mã nguồn, kiểm thử đơn vị và tích hợp, đóng gói ảnh, đẩy lên kho lưu trữ, triển khai thử nghiệm, duyệt phát hành và chiến lược triển khai xanh – lam hoặc thăm dò.

*Nâng cấp hạ tầng điều phối.* Từng bước di trú từ Docker Compose sang Kubernetes để có tự động co giãn theo chỉ số, cập nhật cuốn chiếu, tự phục hồi và cân bằng tải nội cụm. Áp dụng Ingress Controller thay cho cổng Nginx đơn lẻ và chuẩn hóa cấu hình, bí mật thông qua manifest.

*Gia cố an toàn thông tin.* Thiết lập tường lửa ứng dụng web, giới hạn tần suất truy cập, nhật ký kiểm toán giàu ngữ cảnh (ai, làm gì, ở đâu, khi nào, kết quả), phát hiện – ngăn chặn xâm nhập ở cấp máy chủ và mạng, đồng thời quản lý bí mật bằng kho bí mật chuyên dụng và mã hóa đầu–cuối khi triển khai sản xuất.

### 5.2.2. Hoàn thiện tính năng sản phẩm

*Khuyến mãi và mã giảm giá.* Quản lý vòng đời mã, điều kiện áp dụng và giới hạn sử dụng; tính toán giảm trừ tại thanh toán và lưu vết sử dụng theo người dùng – đơn hàng.

*Khách hàng thân thiết***.** Tích lũy – quy đổi điểm, phân hạng quyền lợi, ưu đãi sinh nhật; bảng sao kê giao dịch điểm minh bạch để tăng gắn kết.

*Gợi ý sản phẩm.* Khởi đầu bằng quy tắc dựa trên lịch sử mua và danh mục, sau đó mở rộng sang mô hình cộng tác khi dữ liệu đủ lớn, lưu đệm kết quả theo chu kỳ.

*Quản trị tồn kho nâng cao.* Theo dõi biến động nhập – xuất – trả – hủy, cảnh báo cạn kiệt, đa kho, lô hạn dùng và kiểm kê định kỳ để đối soát thực tế.

*Ứng dụng di động***.** Xây dựng ứng dụng gốc đa nền tảng để mở rộng kênh tiếp cận và tăng tương tác qua thông báo đẩy trạng thái đơn, khuyến mãi và nội dung cá nhân hóa.

### 5.2.3. Nâng chuẩn chất lượng và quy trình

*Kiểm thử theo tầng***.** Mở rộng độ phủ kiểm thử đơn vị cho các nhánh logic quan trọng; bổ sung kiểm thử hợp đồng giữa các dịch vụ; tự động hóa kiểm thử tích hợp và đầu–cuối cho các luồng chính (đặt hàng, thanh toán, thông báo).

*Đánh giá hiệu năng.* Thực hiện kiểm thử tải và chịu đựng, xác định ngưỡng dung năng cho từng dịch vụ, tối ưu truy vấn dữ liệu và chiến lược đệm; dùng số liệu quan sát để dẫn dắt cải tiến.

*Quản trị cấu hình và phiên bản***.** Áp dụng chuẩn đặt tên phiên bản ảnh, tách cấu hình khỏi mã nguồn, theo dõi thay đổi bằng kiểm soát phiên bản và phê duyệt thay đổi.

## 5.3. Kết luận

Hệ thống đã đạt mục tiêu của một nền tảng thương mại điện tử theo kiến trúc microservices ở mức đồ án, hội tụ đủ các thành phần then chốt của một hệ thống phân tán hiện đại: cổng API tập trung, cơ chế hàng đợi thông điệp, bộ nhớ đệm tốc độ cao, cơ sở dữ liệu tin cậy và quan sát nhật ký tập trung. Từ nền tảng này, lộ trình phát triển ưu tiên tập trung vào nhất quán liên dịch vụ bằng mẫu Saga, quan sát – cảnh báo theo chỉ số, tự động hóa quy trình xây dựng – triển khai, gia cố an toàn thông tin, và mở rộng các tính năng tạo giá trị như khuyến mãi, khách hàng thân thiết, gợi ý và quản trị tồn kho. Khi các hạng mục trên được hoàn tất, hệ thống sẽ sẵn sàng tiến từ bản triển khai học thuật sang một nền tảng vận hành thực tế, có thể mở rộng, có thể giám sát và bảo mật tốt, đáp ứng nhu cầu phục vụ người dùng ở quy mô lớn hơn.

# **TÀI LIỆU THAM KHẢO**

1. Fowler, M. (2014), *Microservices: a definition of this new architectural term*, https://martinfowler.com/articles/microservices.html
2. Newman, S. (2021), *Building Microservices: Designing Fine-Grained Systems (2nd Edition)*, O'Reilly Media, Inc., Sebastopol, CA.
3. PostgreSQL Global Development Group (2023), *PostgreSQL 14 Documentation*, https://www.postgresql.org/docs/14/,
4. RabbitMQ Team (2023), *RabbitMQ Tutorials*, https://www.rabbitmq.com/tutorials,
5. React Team (2023), *React Documentation*, https://react.dev/learn
6. Redis Ltd. (2023), *Redis Documentation*, https://redis.io/docs/latest
7. Richardson, C. (2018), *Microservices Patterns: With examples in Java*, Manning Publications, Shelter Island, NY.
8. The Docker Maintainers (2023), *Docker Documentation*, https://docs.docker.com
9. The NestJS Team (2023), *NestJS Documentation*, https://docs.nestjs.com
10. TypeScript Team (2023), *TypeScript Handbook*, https://www.typescriptlang.org/docs/handbook/intro.html,