**ĐẠI HỌC PHENIKAA**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN**

A logo for a university

AI-generated content may be incorrect.

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

**ĐỀ TÀI:**

**PHÁT TRIỂM WEBSITE ĐẶT BÁNH NGỌT ONLINE THEO KIẾN TRÚC MICROSERVICER**

|  |  |
| --- | --- |
| Giảng viên hướng dẫn | : ThS.Vũ Quang Dũng  TS.Trịnh Thanh Bình |
| Nhóm 5 | : Nguyễn Thanh Phong - 22010251 |
|  | Trần Mạnh Hoàng - 23010290 |
|  | Đỗ Minh Thanh - 23012921 |
|  | Hà Đức Thắng - 23010284 |
|  | Lê Anh Minh - 23010228 |
| 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 & Demo | 12–15 slide, demo 7’ |
| Tổng hợp hồ sơ nộp | Gói .zip đầy đủ, chạy được |

# 

|  |  |  |  |
| --- | --- | --- | --- |
| **TT** | **Họ và tên** | **Nhiệm vụ phân công** | **Đánh giá** |
| 1 | **Mạnh Hoàng** | Plan dự án , SRS (đồng), Kiến trúc (đồng), Gateway/Compose, BE support, Logging, Pack hồ sơ | 100% |
| 2 | **Minh Thanh** | SRS (đồng), UC/DFD, Kiến trúc (đồng), Review API/FE | 100% |
| 3 | **Thanh Phong** | TK kiến trúc chi tiết **,**TK chức năng , ERD , BE lead, Redis/RabbitMQ | 100% |
| 4 | **Thang Ha** | Báo cáo tuần, FE lead (KH), Test hộp đen, Report, Slide | 100% |
| 5 | **Lê Minh** | Báo cáo tuần, FE Admin, Test hộp đen, Seed data, Report | 100% |

**1) Mạnh Hoàng — MSSV: …**

|  |  |  |  |
| --- | --- | --- | --- |
| **TT** | **Nhiệm vụ phân công** | **Nội dung đã thực hiện** | **Đánh giá** |
| 1 | **Plan dự án (SOLO)** | Lập **Project Plan v1.0** (scope, mục tiêu, WBS theo T01–T21, timeline 5 tuần, mốc checkpoint, buffer 15%); **RACI** theo từng task; **Risk Register** (10 rủi ro + mitigation); **Gantt** (CSV+PNG). | 20% |
| 2 | SRS (đồng tác giả) | Viết **FR/NFR** khớp đề tài; chuẩn hóa **actor + glossary**; tạo **RTM** (Requirement Traceability Matrix) map FR→API→Test; review phần UC để tránh chồng chéo. | 10% |
| 3 | Thiết kế kiến trúc tổng thể (đồng tác giả) | Đồng dựng **Hình 3.1** (Gateway, 6 services, Redis, RabbitMQ, 2 Postgres, Logging); note trade-off Microservices vs Monolith; quy ước **naming & ports**. | 10% |
| 4 | API Gateway + Compose | Viết **nginx.conf** (route /api/{users,products,cart,orders,payment}; CORS http://localhost:5173; pass headers); tạo **docker-compose.yml** (18 services, networks, volumes); thêm **healthcheck** cho DB/Redis/Rabbit; verify docker-compose ps all **healthy**. | 20% |
| 5 | Backend support | Implement **Product Service core** (GET list/detail, filter/sort/paginate), middleware log; code **Postman collection** cho products; review 6 PR (code style + DI). | 20% |
| 6 | Logging + Tổng hợp hồ sơ nộp | Triển khai **Loki/Promtail/Grafana** (datasource, dashboard query theo container\_name); export ảnh dashboard (Errors/Latency); cuối kỳ **pack release .zip** (source, slides, report, seed). | 20% |
|  |  | **Tổng** | **100%** |

**2) Minh Thanh — MSSV: …**

|  |  |  |  |
| --- | --- | --- | --- |
| **TT** | **Nhiệm vụ phân công** | **Nội dung đã thực hiện** | **Đánh giá** |
| 1 | SRS (đồng tác giả) | Soạn **SRS v1.0**: mô tả bài toán, **Use Case List** (KH/Admin), **flow đặt hàng & xử lý đơn**; hoàn thiện **NFR** (perf <2s, bảo mật JWT, usability, consistency); chuẩn **template** & numbering. | 25% |
| 2 | Use Case & DFD | Vẽ **Hình 1.1–1.5** trên draw.io; mô tả UC (pre/post-condition, normal/alternate flow); thống nhất tên sự kiện với BE. | 25% |
| 3 | Kiến trúc tổng thể (đồng tác giả) | Đồng dựng **Architecture.md** (mô tả tiers, kết nối, security boundary, CORS); sketch **CI/CD stub** cho tương lai. | 25% |
| 4 | Review API/FE | Cross-check **OpenAPI spec** với SRS; review **routing FE** (public/private/admin); mở 7 issue UI/UX (label/empty state/loading). | 25% |
|  |  | **Tổng** | **100%** |

**3) Thanh Phong — MSSV: …**

|  |  |  |  |
| --- | --- | --- | --- |
| **TT** | **Nhiệm vụ phân công** | **Nội dung đã thực hiện** | **Đánh giá** |
| 1 | **Thiết kế kiến trúc chi tiết (SOLO)** | 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 (SOLO)** | Soạn **API spec** (User/Product/Cart/Order/Payment/Notification) + **Sequence Diagrams** cho **Checkout**, **IPN**; define **status model** (order/payment). | 15% |
| 3 | **Thiết kế dữ liệu/ERD (SOLO)** | Vẽ **ERD** cho postgres\_db\_user\_product & postgres\_db\_order\_cart; chọn kiểu dữ liệu, **index**; viết **migration** (TypeORM) & **seed skeleton**. | 15% |
| 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% |
| 5 | Redis & RabbitMQ | Code **cache-aside** sản phẩm (TTL 3600s); giỏ hàng Redis cart:{userId} (Hash); publish **order.created**, consume **payment.processed** (idempotent bằng dedupe-key). | 15% |
|  |  | **Tổng** | **100%** |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | |  |  | |  | |
| **TT** | **Nhiệm vụ phân công** | | **Nội dung đã thực hiện** | | | **Đánh giá** | |
| 1 | Báo cáo hàng tuần | | Làm **slide Thứ 2** hằng tuần (progress, risk, next); maintain **progress log** trên Trello/Sheet; host buổi stand-up. | | | 20% | |
| 2 | Frontend lead (KH) | | Build các trang **Home/Product List/Detail/Cart/Checkout/Order History/Account**; **Axios client** + interceptors (JWT/401 refresh flow); **Route guards**; **Toast**/loading; **responsive**. | | | 35% | |
| 3 | Kiểm thử hộp đen | | Soạn **TestCases.xlsx** (35 case: login, filter/sort, add-to-cart, checkout success/fail, IPN sai checksum, admin update status); chạy **Postman/Newman**; chụp **evidence**. | | | 20% | |
| 4 | Báo cáo cuối kỳ | | Biên tập **Report.pdf**: tổng quan, thiết kế, kết quả, ưu/nhược; lựa hình minh họa; rà soát chính tả & format; check chống trùng nội dung. | | | 15% | |
| 5 | Slide bảo vệ | | Làm **Slides.pptx** 12–15 trang (agenda–demo–Q&A); chèn video demo ngắn; timebox 15’. | | | 10% | |
|  |  | | **Tổng** | | | **100%** | |

**4) Thang Ha — MSSV: …**

**5) Lê Minh — MSSV: …**

|  |  |  |  |
| --- | --- | --- | --- |
| **TT** | **Nhiệm vụ phân công** | **Nội dung đã thực hiện** | **Đánh giá** |
| 1 | Báo cáo hàng tuần | Đồng tác giả slide, tổng hợp blocker + đề xuất giải pháp; update KPI hoàn thành task theo tuần. | 20% |
| 2 | Frontend Admin | Xây **Admin**: **Login**, **Dashboard** (doanh thu/đơn mới/sản phẩm bán chạy), **Users**, **Products** (CRUD + upload ảnh), **Categories**, **Orders** (filter trạng thái + update). | 35% |
| 3 | Kiểm thử hộp đen | Chạy **Newman CLI** CI-like; log kết quả (JSON/HTML); tạo **bug list** (12 bug, close 11). | 15% |
| 4 | Dữ liệu mẫu & seed | Viết **seed.ts**: 50 products + 6 categories + 3 users (admin/customer); script **npm run seed**; ảnh chụp DB sau seed. | 10% |
| 5 | Báo cáo cuối kỳ (đồng tác giả) | Biên tập **Chương 4 – Kết quả thực nghiệm** (ảnh UI KH/Admin, docker-compose healthy); dàn trang mục lục & danh mục hình/bảng. | 20% |
|  |  | **Tổng** | **100%** |

**MỤC LỤC**

[MỤC LỤC 1](#_Toc212415801)

[LỜI CẢM ƠN 4](#_Toc212415802)

[DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ VIẾT TẮT 5](#_Toc212415803)

[DANH MỤC CÁC BẢNG 6](#_Toc212415804)

[DANH MỤC CÁC HÌNH ẢNH, ĐỒ THỊ 8](#_Toc212415805)

[PHÂN CÔNG NHIỆM VỤ 9](#_Toc212415806)

[MỞ ĐẦU……………………………………………………………………………..11](#_Toc212415807)

**CHƯƠNG 1: GIỚI THIỆU & LẬP KẾ HOẠCH …………………………………**

1.1. Giới thiệu đề tài, mục tiêu, phạm vi……………………………………………….  
 1.1.1. Giới thiệu đề tài……………………………………………………………….  
 1.1.2. Mục tiêu hệ thống…………………………………………………………….  
 1.1.3. Phạm vi chức năng MVP……………………………………………………..  
 1.1.4. Ràng buộc & giả định…………………………………………………………  
1.2. Mô hình quy trình & lý do chọn…………………………………………………..  
 1.2.1. Quy trình phát triển…………………………………………………………..  
 1.2.2. Lý do lựa chọn quy trình……………………………………………………..

**CHƯƠNG 2 : PHÂN TÍCH YÊU CẦU……………………………………………..**   
2.1. Yêu cầu dữ liệu (bảng tóm tắt)……………………………………………………  
2.2. Yêu cầu chức năng (FR)………………………………………………………….  
2.3. Yêu cầu phi chức năng (NFR)……………………………………………………  
2.4. Mô hình hóa………………………………………………………………………  
 2.4.1Use Case tổng quan (Hình 1.1)……………………………………………………  
 2.4.2 Use Case chi tiết: Khách hàng, Quản trị viên…………………………………….  
 2.4.3DFD / Luồng nghiệp vụ……………………………………………………………

**CHƯƠNG 3 : THIẾT KẾ HỆ THỐNG……………………………………………**  
3.1. Kiến trúc tổng thể…………………………………………………………………  
 - Cấu trúc microservices…………………………………………………………….  
 - Mô hình tầng hệ thống…………………………………………………………….  
 - Hình 3.1: Sơ đồ kiến trúc………………………………………………………….  
3.2. Thiết kế chi tiết (UML & DB)……………………………………………………..  
 - Class Diagram……………………………………………………………………….  
 - Sequence Diagram……………………………………………………………………  
  • Checkout & Payment (Return/IPN)……………………………………………….  
  • Admin cập nhật trạng thái đơn + gửi thông báo……………………………………  
 - ERD  
  • postgres\_db\_user\_product (Users, Products, Categories)…………………………  
  • postgres\_db\_order\_cart (Orders, OrderItems)……………………………………..  
 - Giao tiếp service……………………………………………………………………..  
  • REST qua Nginx…………………………………………………………………..  
  • RabbitMQ (orders\_exchange, payments\_exchange)………………………………  
3.3. Thiết kế API (REST)…………………………………………………………………  
 - Bảng endpoints chính theo service……………………………………………………  
 - Quy ước base path, mã lỗi, bảo vệ JWT………………………………………………  
3.4. Thiết kế cache & key………………………………………………………………….  
 - Cache-aside Product, cart:{user\_id}…………………………………………………..  
3.5. API Gateway (Nginx)………………………………………………………………..  
 - Routing, CORS, proxy headers………………………………………………………..  
 - Trích snippet nginx.conf……………………………………………………………

**CHƯƠNG 4 : TRIỂN KHAI & KIỂM THỬ………………………………………….**  
4.1. Công nghệ & môi trường……………………………………………………………..  
 - Backend, Frontend, Hạ tầng, Observability…………………………………………..  
4.2. Kịch bản build/run…………………………………………………………………….  
 - Cấu trúc repo, .env, docker-compose.yml……………………………………………..  
 - Hình 4.19: docker-compose ps………………………………………………………..  
4.3. Kết quả thực nghiệm minh họa………………………………………………………..  
 - FE (Khách hàng): Home, List, Detail, Cart, Checkout.................................................  
 - Admin: Login, Dashboard, Users, Products, Orders.....................................................  
 - Gateway và DB kiểm tra cấu trúc…………………………………………………….  
4.4. Kiểm thử…………………………………………………………………………….  
 - Black-box test cases………………………………………………………………….  
  • Đăng nhập (đúng/sai)……………………………………………………………..  
  • Thêm giỏ hàng / hết hang…………………………………………………………  
  • Checkout → Payment return/ipn…………………………………………………..  
  • Admin cập nhật trạng thái đơn…………………………………………………….  
 - Kết quả test: Pass/Fail, log ảnh Postman…………………………………………….  
 - Unit/Integration (tùy chọn)………………………………………………………….

**CHƯƠNG 5 : TỔNG KẾT & HƯỚNG PHÁT TRIỂN……………………………..**   
5.1. Đánh giá & rút kinh nghiệm…………………………………………………………  
 - So sánh mục tiêu vs kết quả………………………………………………………….  
 - Bài học về kiến trúc, dữ liệu, logging, DevOps…………………………………….  
5.2. Hướng phát triển…………………………………………………………………..  
 - Kỹ thuật: Saga, Prometheus, autoscale, CI/CD………………………………………  
 - Tính năng: khuyến mãi, loyalty, inventory, mobile app…………………………….  
 - Bảo mật nâng cao: WAF, IDS/IPS, audit log………………………………………

# **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** 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!

# **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**

* **Bối cảnh & lý do chọn đề tài (E-commerce, nhu cầu đặt bánh; hạn chế monolith; chọn microservices).**
* **Mục tiêu: xây dựng nền tảng đặt bánh; áp dụng Microservices + Nginx (API Gateway) + RabbitMQ + Redis + PostgreSQL + Docker + Logging (Loki/Promtail/Grafana).**
* **Đối tượng phục vụ: Khách hàng & Quản trị viên.**
* **Phạm vi: tính năng cốt lõi; thanh toán VNPay mô phỏng; logging tập trung; health check.**
* **Bố cục báo cáo (4 chương + Kết luận), khớp guide.**

**PHẦN 1 : GIỚI THIỆU & LẬP KẾ HOẠCH**

**1.1. Giới thiệu đề tài, mục tiêu, phạm vi**

**1.1.1. Giới thiệu đề tài**

**Tên đề tài:** Nền tảng Đặt Bánh Ngọt Online theo kiến trúc Microservices.  
Hệ thống hướng tới việc cung cấp trải nghiệm mua bánh trực tuyến nhanh, ổn định; đồng thời cung cấp trang quản trị cho cửa hàng để vận hành sản phẩm, đơn hàng và dữ liệu người dùng.

**1.1.2. Mục tiêu hệ thống**

* Xây dựng ứng dụng web hoàn chỉnh gồm **khách hàng** (Customer) và **quản trị** (Admin).
* Áp dụng kiến trúc **microservices** tách biệt theo domain (User, Product, Cart, Order, Payment, Notification); kết nối qua **API Gateway (Nginx)**, **Redis** (cache/giỏ hàng), **RabbitMQ** (bất đồng bộ), **PostgreSQL** (CSDL), đóng gói **Docker Compose**.
* Chuẩn hóa tài liệu: **Project Plan, SRS, Thiết kế kiến trúc/UML, Thiết kế chương trình (API/ERD/Sequence), Test plan**, báo cáo tuần và báo cáo cuối kỳ.
* Trình diễn **demo end-to-end**: khách đặt hàng → thanh toán mô phỏng VNPay → IPN → cập nhật đơn → gửi thông báo.

**1.1.3. Phạm vi chức năng MVP**

**Khách hàng**

* Đăng ký/đăng nhập/đăng xuất; quản lý tài khoản cơ bản.
* 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 (lưu **Redis**).
* Đặt hàng/checkout; **thanh toán mô phỏng VNPay** (return + IPN).
* Xem lịch sử & chi tiết đơn.
* Đánh giá sản phẩm.
* Thiết kế tùy chỉnh

**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

**VNPay**

* Đặt hàng.
* Thanh toán.

**Email Sever**

* Thanh toán.
* Thanh toán.

**Ngoài phạm vi MVP:** khuyến mãi/loyalty, tích hợp vận chuyển, thanh toán thật, mobile app, autoscaling.

**1.1.4. Ràng buộc & giả định**

* **Thời gian:** trong 01 học kỳ; kế hoạch thực thi **5 tuần** (mục 1.3).
* **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 (20%)**

**2.1. Yêu cầu dữ liệu (bảng tóm tắt)**

* **User, Product, Category, Cart (Redis), Order, OrderItem, Payment, Logs**
* **Chèn Bảng 1 – Thực thể dữ liệu chính (như tài liệu bạn đã liệt kê).**

**2.2. Yêu cầu chức năng (FR)**

* **Khách hàng: tài khoản, duyệt/tìm/lọc/sort sản phẩm, giỏ hàng, checkout, thanh toán (VNPay mô phỏng), lịch sử đơn.**
* **Quản trị viên: quản lý user, sản phẩm, danh mục, đơn hàng; xem stats cơ bản.**

**2.3. Yêu cầu phi chức năng (NFR)**

* **Hiệu năng: trang chính <2s; cache Redis; phân trang.**
* **Bảo mật: JWT, phân quyền, chống XSS/SQLi ở tầng service.**
* **Khả dụng & mở rộng: container hóa; scale theo service.**
* **Usability: UI/UX rõ, nhất quán.**
* **Bảo trì: code convention, module hóa NestJS.**
* **Nhất quán dữ liệu: strong (nội bộ service) & eventual (liên service qua MQ).**

**2.4. Mô hình hóa**

* **Use Case tổng quan (Hình 1.1).**
* **Use Case chi tiết: Khách hàng (Hình 1.2), Quản trị viên (Hình 1.3).**
* **DFD / Luồng nghiệp vụ: đặt hàng (Hình 1.4), xử lý đơn admin (Hình 1.5).**

**Tip: đính kèm mô tả Use Case từng UC (tiêu chí hoàn thành).**

**PHẦN 3 :THIẾT KẾ HỆ THỐNG**

**3.1. Kiến trúc tổng thể**

### *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.)*

* **Hình 3.1: Sơ đồ kiến trúc.**

**3.2. Thiết kế chi tiết (UML & DB)**

**3.2.1Class Diagram (thực thể nghiệp vụ chính).**

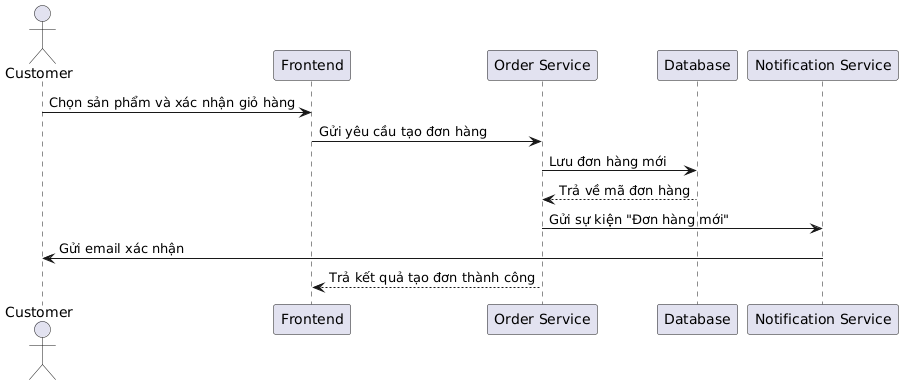
|  |  |  |
| --- | --- | --- |
| **Lớp** | **Thuộc tính tiêu biểu** | **Mối quan hệ** |
| **User** | userId, name, email, password, role | 1 – n với Order |
| **Product** | productId, name, price, quantity, category, image | n – n với Order thông qua OrderDetail |
| **Order** | orderId, orderDate, totalAmount, status, paymentMethod | 1 – n với OrderDetail, 1 – 1 với Payment |
| **OrderDetail** | orderId, productId, quantity, price | Kết nối giữa Order và Product |
| **Payment** | paymentId, orderId, amount, method, status, transactionDate | 1 – 1 với Order |
| **Notification** | notiId, userId, content, type, createdAt | n – 1 với User |

**3.2.2Sequence Diagram (ít nhất 2 flow phức tạp):**

Biểu đồ Sequence minh họa trình tự trao đổi thông điệp giữa các thành phần của hệ thống trong quá trình xử lý một nghiệp vụ cụ thể.  
Dưới đây là mô tả cho 3 luồng quan trọng nhất trong hệ thống:

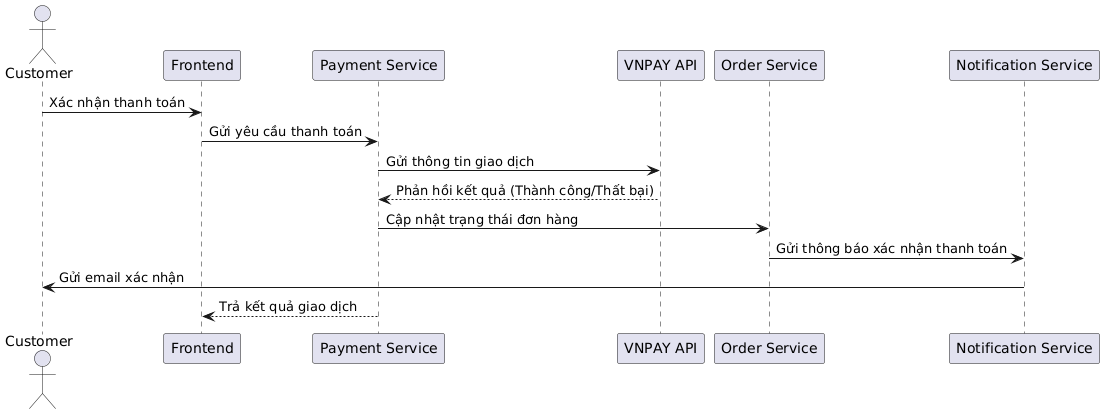
**a. Trình tự: Đặt hàng**

1. Người dùng chọn sản phẩm và xác nhận giỏ hàng.
2. Gửi yêu cầu “Tạo đơn hàng” đến Order Service.
3. Order Service lưu đơn hàng và phản hồi kết quả.
4. Notification Service gửi thông báo “Đơn hàng mới” đến email khách hàng.



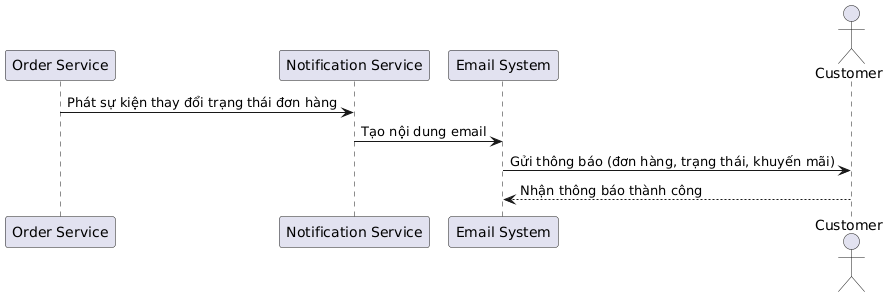
**b. Trình tự: Thanh toán trực tuyến**

1. Người dùng xác nhận thanh toán.
2. Hệ thống gửi yêu cầu đến VNPAY API.
3. VNPAY xác thực giao dịch và trả kết quả về Payment Service.
4. Payment Service cập nhật trạng thái đơn hàng và thông báo cho người dùng.



**c. Trình tự: Gửi thông báo tự động**

1. Khi trạng thái đơn thay đổi, Order Service phát sự kiện đến Notification Service.
2. Notification Service tạo nội dung thông báo.
3. Gửi email hoặc thông báo hệ thống đến khách hàng.



**3.2.3ERD:**

**3.2.3.1Cơ sở dữ liệu postgres\_db\_user\_product (cho User Service và Product Service):**

* 1. **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).
  2. **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).
  3. **Bảng Categories:** category\_id (PK, UUID), name (VARCHAR, UNIQUE), description (TEXT, NULLABLE), created\_at (TIMESTAMP), updated\_at (TIMESTAMP).
  4. **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).
  5. **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):**
  1. **Giỏ hàng:** Key: cart:{user\_id} (Hash). Fields trong hash: {product\_id}: {quantity}.
  2. **Cache sản phẩm:** Key: product:{product\_id} (String/JSON). Value: Chuỗi JSON chứa thông tin chi tiết sản phẩm. TTL được thiết lập.
  3. **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.)*

**1Cơ sở dữ liệu** *postgres\_db\_order\_cart*

A screenshot of a computer

AI-generated content may be incorrect.

*(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.)*

**3.2.4Giao tiếp service:**

**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

}

}

**Bất đồng bộ: RabbitMQ (orders\_exchange, payments\_exchange); Hình 2.1.**

**3.3. Thiết kế API (REST)**

### *3.3.1 Khái niệm và vai trò*

API Gateway là một máy chủ đóng vai trò là điểm vào (entry point) duy nhất cho tất cả các yêu cầu từ các ứng dụng client (ví dụ: web frontend, mobile app) đến các microservices ở backend. Thay vì client phải gọi trực tiếp đến nhiều microservices khác nhau (mỗi service có thể có địa chỉ và port riêng), client chỉ cần tương tác với API Gateway.

Vai trò chính của API Gateway bao gồm:

* **Routing (Định tuyến):** Nhận diện yêu cầu từ client và định tuyến nó đến microservice phù hợp ở backend.
* **Request Aggregation/Composition:** Tổng hợp kết quả từ nhiều microservices để trả về một phản hồi duy nhất cho client, giúp giảm số lượng lời gọi mạng từ client.
* **Protocol Translation:** Chuyển đổi giao thức giữa client và các microservices nếu cần (ví dụ: client dùng HTTP, một số service nội bộ dùng AMQP).
* **Offloading Cross-Cutting Concerns (Giảm tải các mối quan tâm chung):** Xử lý các tác vụ chung như:
  + **Authentication và Authorization:** Xác thực token của người dùng, kiểm tra quyền truy cập.
  + **SSL Termination:** Giải mã các kết nối HTTPS từ client và chuyển tiếp yêu cầu HTTP thông thường đến các service nội bộ.
  + **Rate Limiting và Throttling:** Giới hạn số lượng yêu cầu từ một client trong một khoảng thời gian nhất định để bảo vệ backend.
  + **Logging và Monitoring:** Ghi log tất cả các yêu cầu và phản hồi, thu thập metrics.
  + **Response Caching:** Cache các phản hồi thường xuyên được yêu cầu để giảm tải cho backend.
  + **CORS (Cross-Origin Resource Sharing) Handling:** Xử lý các header CORS để cho phép client từ các domain khác gọi API.

### *3.3.2. Nginx làm API Gateway*

Nginx là một web server mã nguồn mở hiệu năng cao, cũng thường được sử dụng như một reverse proxy, load balancer, mail proxy và HTTP cache. Với sự linh hoạt và hiệu năng của mình, Nginx là một lựa chọn phổ biến để triển khai API Gateway.

Trong dự án "Nền tảng Đặt Bánh Ngọt Online", Nginx được sử dụng với các chức năng chính sau:

* **Reverse Proxy:** Đóng vai trò là máy chủ trung gian, nhận yêu cầu từ client và chuyển tiếp đến các microservices tương ứng đang chạy trên các port khác nhau trong mạng nội bộ Docker.
* **Định tuyến yêu cầu (Request Routing):** Dựa trên đường dẫn (path) của URL yêu cầu (ví dụ: /api/users, /api/products), Nginx sẽ định tuyến yêu cầu đến user\_service, product\_service, v.v.
* **Xử lý CORS:** Cấu hình các header Access-Control-Allow-Origin, Access-Control-Allow-Methods, Access-Control-Allow-Headers để cho phép frontend (chạy trên http://localhost:5173) có thể gọi đến các API backend (chạy qua Nginx trên port 80).

*(Một trích đoạn quan trọng từ file cấu hình api-gateway/nginx.conf sẽ được trình bày ở đây, ví dụ: phần định nghĩa upstream và một location block cho một service.)*

* # api-gateway/nginx.conf (trích đoạn)
* http {
* # ... các cấu hình khác ...
* upstream user\_service\_backend { server user\_service:3001; }
* upstream product\_service\_backend { server product\_service:3002; }
* # ... các upstream khác ...
* server {
* listen 80 default\_server;
* server\_name localhost;
* location ~ ^/api/(users|auth)(/.\*)?$ {
* set $backend\_path\_prefix /$1;
* set $backend\_sub\_path $2;
* if ($request\_method = 'OPTIONS') {
* add\_header 'Access-Control-Allow-Origin' 'http://localhost:5173' always;
* add\_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, PATCH, DELETE, OPTIONS' always;
* add\_header 'Access-Control-Allow-Headers' 'Authorization, Content-Type, Accept, Origin' always;
* add\_header 'Access-Control-Allow-Credentials' 'true' always;
* add\_header 'Access-Control-Max-Age' 1728000;
* add\_header 'Content-Length' 0;
* return 204;
* }
* add\_header 'Access-Control-Allow-Origin' 'http://localhost:5173' always;
* add\_header 'Access-Control-Allow-Credentials' 'true' always;
* proxy\_pass http://user\_service\_backend$backend\_path\_prefix$backend\_sub\_path$is\_args$args;
* proxy\_set\_header Host $host;
* proxy\_set\_header X-Real-IP $remote\_addr;
* proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;
* proxy\_set\_header X-Forwarded-Proto $scheme;
* }
* # ... các location block khác cho product, order, cart, payment ...
* }
* }

### *3.3.3. Lựa chọn Nginx làm API Gateway*

Nginx được lựa chọn làm API Gateway cho dự án này vì những lý do sau:

* **Hiệu năng cao và ổn định:** Nginx nổi tiếng với khả năng xử lý hàng ngàn kết nối đồng thời với lượng tài nguyên sử dụng thấp.
* **Cấu hình linh hoạt:** Hệ thống cấu hình dựa trên file text của Nginx rất mạnh mẽ và linh hoạt, cho phép định nghĩa các quy tắc routing phức tạp, rewrite URL, và thêm các header tùy chỉnh.
* **Cộng đồng lớn và tài liệu phong phú:** Có rất nhiều tài liệu hướng dẫn, ví dụ và sự hỗ trợ từ cộng đồng khi gặp vấn đề.
* **Phổ biến và dễ tích hợp:** Nginx là một công cụ rất phổ biến trong các kiến trúc web hiện đại, dễ dàng tích hợp với Docker và các công nghệ khác được sử dụng trong dự án.
* **Đáp ứng đủ nhu cầu của dự án:** Cho các yêu cầu cơ bản của một API Gateway như reverse proxy, routing và xử lý CORS trong phạm vi đồ án, Nginx là một lựa chọn hiệu quả và không quá phức tạp để triển khai so với các giải pháp API Gateway chuyên dụng khác như Kong hay Spring Cloud Gateway (nếu không dùng Spring).

### *3.3.4 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.

**3.4. Thiết kế cache & key**

* **Cache-aside cho Product (TTL), cart:{user\_id} (Redis Hash).**

**3.5. API Gateway (Nginx)**

* **Routing theo path, CORS, proxy headers.**
* **Trích snippet nginx.conf (định tuyến /api/users|auth|products…).**

**PHẦN 4 TRIỂN KHAI & KIỂM THỬ**

**4.1. Công nghệ & môi trường**

## 4.1.1Công nghệ phát triển Backend: Node.js và Framework NestJS

Untuk việc phát triển các microservices backend trong dự án, Node.js được lựa chọn làm nền tảng runtime, kết hợp với framework NestJS. Node.js là một môi trường thực thi JavaScript phía máy chủ, được xây dựng trên engine V8 của Google Chrome. Đặc điểm nổi bật của Node.js là mô hình I/O bất đồng bộ, không chặn (non-blocking I/O), và kiến trúc dựa trên sự kiện (event-driven), làm cho nó rất phù hợp cho việc xây dựng các ứng dụng mạng có khả năng mở rộng cao, xử lý nhiều kết nối đồng thời mà không tốn nhiều tài nguyên hệ thống, đặc biệt hiệu quả cho các ứng dụng I/O-bound như các API của hệ thống E-commerce.

NestJS là một framework Node.js tiến bộ để xây dựng các ứng dụng phía máy chủ hiệu quả, có khả năng mở rộng và dễ bảo trì. NestJS được xây dựng bằng TypeScript (một superset của JavaScript bổ sung kiểu tĩnh) và lấy cảm hứng mạnh mẽ từ Angular về mặt kiến trúc module, dependency injection, decorators, và các khái niệm khác. Việc sử dụng NestJS mang lại nhiều lợi ích:

* **Cấu trúc dự án rõ ràng và có tổ chức:** NestJS cung cấp một cấu trúc module hóa (Modules, Controllers, Services) giúp tổ chức code một cách logic, dễ hiểu và dễ quản lý, đặc biệt quan trọng khi làm việc với nhiều microservices.
* **Hỗ trợ TypeScript mạnh mẽ:** Tận dụng các lợi ích của TypeScript như kiểm tra kiểu tĩnh giúp phát hiện lỗi sớm, cải thiện khả năng đọc hiểu code và tăng cường tính bảo trì.
* **Dependency Injection (DI):** Giúp quản lý các dependencies giữa các class một cách hiệu quả, tăng tính module hóa và khả năng kiểm thử.
* **Dễ dàng tích hợp với các thư viện Node.js phổ biến:** NestJS có thể dễ dàng tích hợp với các thư viện ORM/ODM (như TypeORM cho PostgreSQL), các thư viện xác thực (như Passport.js cho JWT), các client cho RabbitMQ (amqplib), Redis (ioredis), v.v.
* **Hỗ trợ xây dựng Microservices:** NestJS cung cấp các module và công cụ tích hợp sẵn để xây dựng các microservices giao tiếp qua nhiều transports khác nhau (bao gồm TCP, Redis, gRPC, RabbitMQ).
* **Cộng đồng phát triển tích cực và tài liệu tốt.**

Lựa chọn Node.js kết hợp với NestJS cho backend của dự án "Nền tảng Đặt Bánh Ngọt Online" dựa trên các yếu tố như hiệu năng tốt cho các tác vụ I/O, sự quen thuộc của ngôn ngữ JavaScript/TypeScript (nếu đội ngũ cũng phát triển frontend bằng React), khả năng phát triển nhanh, và cấu trúc mạnh mẽ mà NestJS mang lại giúp xây dựng các API và microservices một cách có tổ chức và dễ bảo trì.

## 4.1.2Công nghệ phát triển Frontend: React và TypeScript

Đối với việc xây dựng giao diện người dùng (cả cho khách hàng và trang quản trị), React được lựa chọn làm thư viện JavaScript chính, kết hợp với TypeScript để tăng cường chất lượng mã nguồn. React là một thư viện JavaScript phổ biến và mạnh mẽ được phát triển bởi Facebook, tập trung vào việc xây dựng các giao diện người dùng tương tác dựa trên kiến trúc component (component-based architecture). Mỗi phần của giao diện người dùng được chia thành các component độc lập, có thể tái sử dụng, giúp việc quản lý và phát triển UI trở nên dễ dàng hơn. React sử dụng Virtual DOM để tối ưu hóa việc cập nhật DOM thật, mang lại hiệu năng tốt cho các ứng dụng web động.

TypeScript là một superset của JavaScript, được Microsoft phát triển, bổ sung thêm hệ thống kiểu tĩnh tùy chọn và các tính năng hướng đối tượng mạnh mẽ hơn. Việc sử dụng TypeScript trong dự án React mang lại nhiều lợi ích:

* **Phát hiện lỗi sớm:** Hệ thống kiểu tĩnh giúp phát hiện nhiều lỗi logic và lỗi kiểu ngay trong quá trình phát triển (compile-time) thay vì phải đợi đến lúc chạy (runtime).
* **Cải thiện khả năng đọc hiểu và bảo trì code:** Kiểu dữ liệu rõ ràng giúp code dễ hiểu hơn, đặc biệt trong các dự án lớn hoặc khi làm việc nhóm.
* **Tái cấu trúc code an toàn hơn:** IDE có thể hỗ trợ tốt hơn cho việc tái cấu trúc code khi có thông tin về kiểu.
* **Tích hợp tốt với các thư viện hiện đại:** Nhiều thư viện React phổ biến đã có sẵn định nghĩa kiểu TypeScript hoặc được viết bằng TypeScript.

Bên cạnh React và TypeScript, các công nghệ và thư viện hỗ trợ khác được sử dụng trong phần frontend của dự án bao gồm:

* **React Router:** Để quản lý việc định tuyến (routing) giữa các trang khác nhau trong ứng dụng Single Page Application (SPA).
* **Context API (hoặc Redux/Zustand tùy lựa chọn cuối cùng):** Để quản lý trạng thái toàn cục của ứng dụng, ví dụ như thông tin xác thực của người dùng (AuthContext).
* **Axios:** Một thư viện HTTP client dựa trên Promise để thực hiện các lời gọi API đến backend.
* **Tailwind CSS:** Một framework CSS utility-first để xây dựng giao diện người dùng một cách nhanh chóng và tùy biến cao mà không cần viết nhiều CSS tùy chỉnh.
* **Framer Motion:** Một thư viện animation cho React, giúp tạo ra các hiệu ứng chuyển động mượt mà và tương tác phong phú.
* **Heroicons / React Icons:** Các bộ thư viện icon SVG để sử dụng trong giao diện.

Lựa chọn React và TypeScript cho frontend của dự án "Nền tảng Đặt Bánh Ngọt Online" dựa trên sự phổ biến rộng rãi, cộng đồng hỗ trợ lớn, hệ sinh thái phong phú của React, cùng với những lợi ích về chất lượng mã nguồn và khả năng bảo trì mà TypeScript mang lại. Kiến trúc component của React cũng rất phù hợp để xây dựng các giao diện phức tạp và có tính tương tác cao của một ứng dụng E-commerce

* **Hạ tầng: Docker & Docker Compose; PostgreSQL; Redis; RabbitMQ; Nginx.**
* **Observability: Loki, Promtail, Grafana; healthcheck.**

**4.2. Kịch bản build/run**

* **Cấu trúc repo, .env, docker-compose.yml, network/volumes.**
* **Hình 4.19: docker-compose ps (tất cả healthy).**

**4.3. Kết quả thực nghiệm minh họa**

## 4.1. 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. 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 : 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.)*

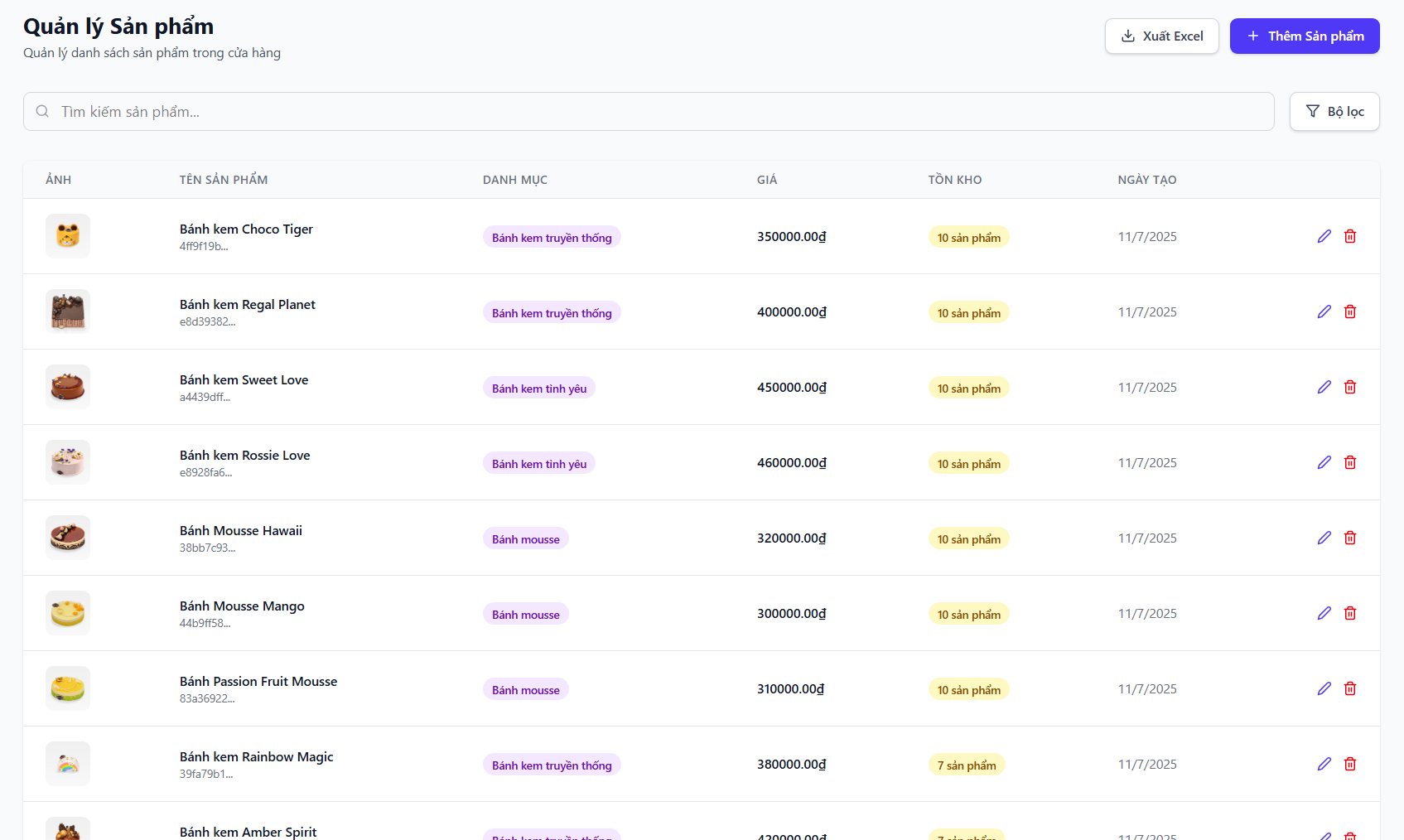
**

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**

* **Black-box test cases (bắt buộc theo guide) — bảng tối thiểu cho:**
  + **Đăng nhập (đúng/sai)**
  + **Thêm giỏ, giữ chỗ, trùng ghế/sản phẩm hết hàng (nếu có)**
  + **Checkout → Payment return/ipn thành công/thất bại**
  + **Admin cập nhật trạng thái đơn**
* **Kết quả test: Pass/Fail, log ảnh Postman/Grafana Explore.**
* **(Tùy chọn) Unit/Integration notes (Jest, RTL).**

**PHẦN 5 — TỔNG KẾT & HƯỚNG PHÁT TRIỂN (10%)**

**5.1. Đánh giá & rút kinh nghiệm**

### *5.1.1 So sánh với mục tiêu đề ra*

Đối chiếu với các mục tiêu đã được đặt ra trong phần Mở đầu và Chương 1, hệ thống "Nền tảng Đặt Bánh Ngọt Online" đã đạt được những kết quả chính sau:

* **Thiết kế và triển khai thành công hệ thống E-commerce hoàn chỉnh cho đặt bánh ngọt:** Hệ thống đã cung cấp đầy đủ các chức năng cốt lõi cho cả khách hàng (đăng ký, đăng nhập, xem sản phẩm, giỏ hàng, đặt hàng, thanh toán mô phỏng, xem lịch sử đơn hàng) và quản trị viên (quản lý sản phẩm, danh mục, đơn hàng, người dùng, xem thống kê cơ bản).
* **Áp dụng hiệu quả kiến trúc microservices:** Hệ thống đã được phân tách thành các microservices độc lập (User, Product, Cart, Order, Payment, Notification), mỗi service chịu trách nhiệm cho một mảng nghiệp vụ cụ thể. Việc này đã được hiện thực hóa bằng cách sử dụng Node.js/NestJS cho từng service và đóng gói chúng bằng Docker.
* **Tích hợp và vận dụng các kỹ thuật hệ thống phân tán:**
  + **API Gateway (Nginx):** Đã được triển khai làm điểm vào duy nhất, xử lý định tuyến và CORS.
  + **Message Queue (RabbitMQ):** Đã được sử dụng hiệu quả cho giao tiếp bất đồng bộ, ví dụ như trong quy trình gửi email xác nhận đơn hàng và thông báo cập nhật trạng thái đơn hàng, giúp tăng khả năng phục hồi và giảm tải cho các service chính.
  + **Caching (Redis):** Đã được áp dụng để cache dữ liệu sản phẩm và quản lý thông tin giỏ hàng, góp phần cải thiện hiệu năng truy xuất dữ liệu.
  + **Database (PostgreSQL) với mô hình Database per Service:** Các service chính có cơ sở dữ liệu riêng (hoặc schema riêng biệt logic), đảm bảo tính độc lập.
* **Xây dựng giao diện người dùng (Frontend) và giao diện quản trị (Admin) trực quan:** Giao diện được phát triển bằng React/TypeScript, đáp ứng các yêu cầu về chức năng và có tính thẩm mỹ cơ bản, dễ sử dụng.
* **Triển khai cơ chế logging tập trung và health check:** Hệ thống logging với Loki, Promtail, Grafana đã hoạt động, cho phép theo dõi log tập trung. Health checks đã được cấu hình cho các service quan trọng.

|  |  |  |
| --- | --- | --- |
| **Mục tiêu** | **Mức độ hoàn thành** | **Ghi chú** |
| Xây dựng hệ thống E-commerce hoàn chỉnh | Đạt | Các chức năng cốt lõi cho người dùng và admin đã được triển khai. |
| Áp dụng kiến trúc microservices | Đạt | Hệ thống được chia thành 6 microservices chính, mỗi service có trách nhiệm rõ ràng. |
| Tích hợp API Gateway (Nginx) | Đạt | Nginx hoạt động tốt trong vai trò định tuyến và xử lý CORS. |
| Tích hợp Message Queue (RabbitMQ) | Đạt | RabbitMQ được sử dụng cho các tác vụ thông báo bất đồng bộ. |
| Tích hợp Caching (Redis) | Đạt | Redis được dùng để cache sản phẩm và quản lý giỏ hàng. |
| Xây dựng Frontend và Admin UI | Đạt | Giao diện đáp ứng các chức năng cơ bản, dễ sử dụng. |
| Triển khai logging tập trung và health check | Đạt | Loki, Promtail, Grafana hoạt động. Health checks được cấu hình. |
| Đảm bảo tính nhất quán dữ liệu (ở mức độ phù hợp) | Đạt một phần | Đảm bảo trong phạm vi từng service. Tính nhất quán cuối cùng qua message queue. Các kịch bản phức tạp hơn chưa được xử lý sâu. |
| Mô phỏng và đánh giá khả năng chịu lỗi, sao lưu dữ liệu cơ bản | Chưa thực hiện sâu | Mới dừng ở mức đảm bảo service có thể khởi động lại. Chưa có kịch bản đánh giá chịu lỗi hay chiến lược sao lưu dữ liệu chi tiết. |

*Bảng 4: Đánh giá mức độ hoàn thành các mục tiêu của đề tài*

### *5.1.2. Ưu điểm của hệ thống*

Qua quá trình triển khai và kết quả thực nghiệm, hệ thống đã thể hiện được những ưu điểm sau:

* **Tính module hóa cao:** Việc chia nhỏ hệ thống thành các microservices giúp mã nguồn của từng phần trở nên nhỏ gọn, dễ hiểu, dễ quản lý và bảo trì hơn.
* **Khả năng phát triển và triển khai độc lập:** Mỗi microservice có thể được phát triển, kiểm thử và triển khai mà không ảnh hưởng lớn đến các service khác, tạo điều kiện cho việc nâng cấp và cập nhật hệ thống một cách linh hoạt.
* **Khả năng mở rộng tiềm năng:** Mặc dù chưa được kiểm chứng ở quy mô lớn, kiến trúc microservices cho phép mở rộng từng dịch vụ cụ thể khi có nhu cầu, thay vì phải mở rộng toàn bộ hệ thống như kiến trúc nguyên khối.
* **Sử dụng hiệu quả các công nghệ phân tán:** Việc tích hợp Redis cho caching và RabbitMQ cho message queuing đã góp phần cải thiện hiệu năng và tăng khả năng phục hồi cho một số tác vụ nhất định.
* **Đơn giản hóa quản lý môi trường với Docker:** Docker và Docker Compose đã giúp việc thiết lập, quản lý môi trường phát triển và (tiềm năng) triển khai trở nên nhất quán và dễ dàng hơn rất nhiều.
* **Giao diện người dùng đáp ứng nhu cầu cơ bản:** Cả giao diện cho khách hàng và trang quản trị đều cung cấp các chức năng cần thiết, với thiết kế tương đối trực quan.
* **Khả năng giám sát tốt hơn với logging tập trung:** Việc có thể xem log từ tất cả các service tại một nơi (Grafana) là một lợi thế lớn trong việc theo dõi và gỡ lỗi hệ thống phân tán.

### *5.1.3. Hạn chế của hệ thống*

Bên cạnh những ưu điểm, hệ thống vẫn còn một số hạn chế cần được nhìn nhận:

* **Độ phức tạp trong vận hành và giám sát tổng thể:** Mặc dù có logging tập trung, việc quản lý, giám sát và gỡ lỗi một hệ thống gồm nhiều microservices vẫn phức tạp hơn so với một ứng dụng nguyên khối. Cần thêm các công cụ giám sát metrics, alerting chuyên sâu.
* **Thách thức về tính nhất quán dữ liệu phân tán:** Trong các nghiệp vụ phức tạp hơn liên quan đến nhiều service (ví dụ: một giao dịch cập nhật thông tin ở nhiều nơi), việc đảm bảo tính nhất quán dữ liệu hoàn toàn (strong consistency) là rất khó. Hệ thống hiện tại chủ yếu dựa vào eventual consistency qua message queue, điều này có thể không phù hợp với tất cả các kịch bản. Các giải pháp như Saga pattern chưa được triển khai.
* **Chưa có các tính năng E-commerce nâng cao:** Hệ thống mới chỉ tập trung vào các chức năng cốt lõi. Các tính năng nâng cao như hệ thống gợi ý sản phẩm, quản lý khuyến mãi phức tạp, chương trình khách hàng thân thiết, quản lý kho hàng chi tiết, v.v., chưa được phát triển.
* **Phạm vi kiểm thử còn hạn chế:** Mặc dù có unit test và một số kịch bản integration/UAT, phạm vi kiểm thử (test coverage) có thể chưa đầy đủ, đặc biệt là các kịch bản lỗi và kiểm thử hiệu năng dưới tải cao.
* **Bảo mật ở mức cơ bản:** Hệ thống đã triển khai JWT cho xác thực và ủy quyền, cùng với xử lý CORS. Tuy nhiên, các biện pháp bảo mật chuyên sâu hơn như Web Application Firewall (WAF), Intrusion Detection/Prevention System (IDS/IPS), audit log chi tiết chưa được áp dụng.
* **Thiếu tự động hóa CI/CD:** Quá trình build, test và deploy hiện tại vẫn mang tính thủ công. Việc thiếu một pipeline CI/CD hoàn chỉnh làm giảm tốc độ và độ tin cậy khi cập nhật hệ thống.
* **Hiệu năng chưa được đánh giá toàn diện:** Chưa có các bài kiểm thử tải (load testing) hoặc kiểm thử sức chịu đựng (stress testing) để đánh giá hiệu năng của hệ thống dưới các điều kiện tải khác nhau.

**5.2. Hướng phát triển**

* **Kỹ thuật: Saga pattern, Prometheus + Alerting, autoscale, CI/CD.**
* **Tính năng: khuyến mãi/mã giảm giá, loyalty, gợi ý sản phẩm, inventory chuẩn, đánh giá/bình luận, mobile app.**
* **Bảo mật nâng cao: WAF, IDS/IPS, audit log.**