

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG  
KHOA ĐÀO TẠO SAU ĐẠI HỌC



**BÁO CÁO ĐỀ TÀI**  
**MICROSERVICE**  
**MÔN HỌC: HỆ THỐNG PHÂN TÁN**

GIẢNG VIÊN

TS. Kim Ngọc Bách

THÀNH VIÊN NHÓM

Vũ Đức Uyên

B25CHHT068

Nguyễn Mạnh Kỳ

B25CHHT033

Nguyễn Tiến Đạt

B25CHHT010

*Hà Nội, 15 tháng 12 năm 2025*

<b>PHẦN I: PHÂN TÍCH YÊU CẦU.....</b>	<b>2</b>
1.1. Bài Toán và Bối CảnhBối Cảnh Công Nghệ: Thách Thức của Monolithic.....	2
1.2. Yêu Cầu Chức Năng (Functional Requirements)Actor (Tác Nhân).....	4
1.3. Yêu Cầu Phi Chức Năng (Non-Functional Requirements)Hiệu Năng (Performance).....	7
<b>PHẦN II: TỔNG HỢP HỆ THỐNG &amp; TÍNH NĂNG.....</b>	<b>8</b>
2.1. Mô Tả Chi Tiết Các Dịch Vụ (Services)Service 1: User Service.....	9
2.2. Các Bảng Tổng HợpBảng 1: Tổng Quan Hệ Thống.....	10
<b>PHẦN III: THIẾT KẾ PHẦN MỀM.....</b>	<b>12</b>
3.1. Thiết Kế Tổng Thể.....	12
3.1.1. Kiến Trúc Phân Tầng (Layers).....	12
3.1.2. Design Patterns Cốt Lõi.....	12
3.2. Thiết Kế Chi Tiết (Diagrams).....	13
3.2.1. ERD (Entity Relationship Diagrams)ERD cho orderservice_db (PostgreSQL).....	13
3.2.2. Sequence Diagram: Tạo Đơn Hàng (Hybrid - Sync + Async).....	13
3.3. Thiết Kế API (Chi Tiết)Nguyên Tắc RESTful.....	14
3.4. Thiết Kế Message Queue (RabbitMQ)Cấu Hình Queue.....	15
3.5. Thiết Kế Bảo Mật & DeploymentCơ Chế Xác Thực (Authentication).....	16

## PHẦN I: PHÂN TÍCH YÊU CẦU

### 1.1. Bài Toán và Bối CảnhBối Cảnh Công Nghệ: Thách Thức của Monolithic

Trong môi trường kinh doanh số hóa, các hệ thống **Monolithic** đã bộc lộ những điểm yếu chí mạng, không đáp ứng được nhu cầu của một nền tảng E-Commerce hiện đại:

- **Khó Bảo Trì (Maintainability):** Toàn bộ mã nguồn nằm trong một codebase. Việc thay đổi nhỏ cũng đòi hỏi phải hiểu và kiểm tra toàn bộ hệ thống, làm chậm chu kỳ phát hành và tăng rủi ro lỗi.
- **Vấn Đề Mở Rộng (Scaling Inflexibility):** Hệ thống chỉ có thể mở rộng bằng cách nhân bản toàn bộ ứng dụng. Ví dụ, module xử lý đơn hàng cần tài nguyên gấp 5 lần module quản lý người dùng, nhưng vẫn phải scale toàn bộ, dẫn đến lãng phí tài nguyên máy chủ.
- **Rủi Ro Thất Bại Cao (High Risk of Failure/Tight Coupling):** Một lỗi rò rỉ bộ nhớ hoặc lỗi logic nhỏ trong bất kỳ module nào cũng có thể khiến toàn bộ

ứng dụng bị sập, gây mất doanh thu và ảnh hưởng nghiêm trọng đến trải nghiệm người dùng.

- **Technology Lock-in:** Việc áp dụng công nghệ mới hoặc nâng cấp phiên bản framework cho một phần của hệ thống là bất khả thi vì nó ảnh hưởng đến tính tương thích của toàn bộ ứng dụng.

Thách Thức Dự Án Hiện Tại và Giải Pháp Kiến Trúc

Dự án này được xây dựng để giải quyết các thách thức trên bằng kiến trúc Microservice, đảm bảo hiệu suất và độ tin cậy vượt trội:

Thách Thức Cốt Lõi	Mục Tiêu Cần Đạt	Giải Pháp Kiến Trúc (Microservice Pattern)
Xử lý Đồng Thời	Xử lý hiệu quả lượng lớn request từ đa người dùng cùng lúc (tối thiểu 100 concurrent).	<b>Scale Horizontal</b> với các dịch vụ <b>Stateless</b> (dễ dàng nhân bản).
Tính Nhất Quán Dữ Liệu	Đảm bảo dữ liệu đồng bộ và nhất quán giữa các dịch vụ độc lập.	<b>Database per Service</b> và <b>Eventual Consistency</b> qua RabbitMQ.
Xử lý Bất Đồng Bộ	Áp dụng cơ chế xử lý bất đồng bộ cho các tác vụ tốn thời gian (tạo đơn hàng, gửi email).	<b>Event-Driven Architecture</b> sử dụng <b>RabbitMQ</b> (Message Queue).
Logging và Monitoring	Cần có hệ thống theo dõi và ghi log tập trung, dễ dàng truy vấn và phân tích.	<b>Centralized Logging</b> sử dụng <b>MongoDB</b> (NoSQL, cấu trúc linh hoạt).
Deployment Độc Lập	Cho phép triển khai (deploy) từng service mà không ảnh hưởng đến các service khác.	<b>Containerization</b> (Docker) và <b>Docker Compose/Kubernetes</b> (Orchestration).

Lý Do Lựa Chọn Kiến Trúc Microservice

Kiến trúc Microservice được lựa chọn dựa trên các nguyên tắc cơ bản sau:

1. **Độc Lập và Cô Lập Lỗi (Fault Isolation):** Các dịch vụ nhỏ, độc lập (User, Product, Order) giúp cô lập lỗi. Nếu một service lỗi, các service khác vẫn hoạt động bình thường.
2. **Trung Gian Tin Cậy (API Gateway - Ocelot):** Đóng vai trò là điểm truy cập duy nhất (Single Entry Point), xử lý Routing, Load Balancing, và tăng cường bảo mật, ẩn cấu trúc nội bộ phức tạp.

3. **Xử lý Bất Đồng Bộ (RabbitMQ):** Giảm sự phụ thuộc lẫn nhau giữa các service (loose coupling) và cho phép xử lý các tác vụ phức tạp một cách bất đồng bộ, không làm tắc nghẽn luồng chính.
4. **Database per Service:** Mỗi service sở hữu database riêng (PostgreSQL), đảm bảo tính độc lập về dữ liệu và cho phép đội ngũ phát triển chọn database tối ưu cho từng service.
5. **Linh Hoạt Công Nghệ (Technology Diversity):** Cho phép các service sử dụng các ngôn ngữ, framework và công nghệ khác nhau nếu cần.

## 1.2. Yêu Cầu Chức Năng (Functional Requirements) Actor (Tác Nhân)

1. **Customer (Khách hàng):** Người dùng cuối, thực hiện các giao dịch (Đăng ký, Đăng nhập, Xem sản phẩm, Tạo đơn hàng).
2. **Admin (Quản trị viên):** Quản lý hệ thống, người dùng, sản phẩm và đơn hàng (CRUD operations).
3. **System (Hệ thống):** Các dịch vụ tự động xử lý (cập nhật tồn kho, ghi log, điều hướng request, xử lý event).

Module 1: Quản Lý Người Dùng (User Service - Port 5001)

Mã UC	Tên Use Case	Actor	Business Rules Chính
UC-000	Đăng Nhập	Customer ▾	Validate Username/Email và Password. Password Hash bằng BCrypt. Trả về <b>JWT Token</b> và <b>Refresh Token</b> . Log vào MongoDB.
UC-001	Đăng Ký Tài Khoản	Customer ▾	Username/Email <b>unique</b> . Password Hash bằng <b>BCrypt</b> (Cost Factor=12). Tự động set Role mặc định: "Customer".
UC-002	Xem Danh Sách Users	Admin, Syst... ▾	Hỗ trợ <b>pagination</b> (mặc định 10). Chỉ trả về users có <b>IsDeleted = false</b> . Có thể Search theo username, email.

UC-003	Xem Chi Tiết User	Customer, A... ▾	Customer chỉ xem được thông tin của chính mình. <b>Không trả về password hash.</b>
UC-004	Cập Nhật Thông Tin	Customer, A... ▾	<b>Không cho phép cập nhật Username, Email.</b> Tự động set <b>UpdatedAt</b> . Log action vào MongoDB.
UC-005	Xóa User (Soft Delete)	Admin ▾	<b>Soft Delete (IsDeleted = true).</b> Cập nhật <b>UpdatedAt</b> . Log vào MongoDB.
UC-006	Quản Lý Địa Chỉ	Customer, A... ▾	Customer chỉ quản lý địa chỉ của chính mình. Chỉ có một địa chỉ được set <b>IsDefault = true</b> tại một thời điểm.

## Module 2: Quản Lý Sản Phẩm (Product Service - Port 5002)

Mã UC	Tên Use Case	Actor	Business Rules Chính
UC-007	Xem DS Sản Phẩm	Customer, A... ▾	Chỉ hiển thị sản phẩm có <b>IsAvailable = true</b> và <b>IsDeleted = false</b> . Hỗ trợ filter/search.
UC-008	Xem Chi Tiết SP	Customer, A... ▾	Bao gồm thông tin <b>Reviews</b> và tính <b>Average Rating</b> từ reviews.
UC-009	Tạo Sản Phẩm Mới	Admin ▾	<b>Price &gt; 0, Stock &gt;= 0.</b> Tự động

			set <b>IsAvailable = true</b> . Log vào MongoDB.
<b>UC-010</b>	<b>Cập Nhật Sản Phẩm</b>	Admin ▾	Có thể update tất cả fields trừ Id. Nếu <b>Stock = 0</b> , tự động set <b>IsAvailable = false</b> . Log vào MongoDB.
<b>UC-011</b>	<b>Cập Nhật Tồn Kho</b>	System, Admin ▾	<b>Stock mới = Stock cũ + Quantity</b> . Trả về lỗi nếu <b>Stock &lt; 0</b> . Nếu Stock = 0, set IsAvailable = false.
<b>UC-012</b>	<b>Xóa Sản Phẩm</b>	Admin ▾	<b>Soft delete (IsDeleted = true)</b> . Không cho phép xóa nếu đang có <b>Orders sử dụng</b> (cần check Order Service).

### Module 3: Quản Lý Đơn Hàng (Order Service - Port 5003)

<b>Mã UC</b>	<b>Tên Use Case</b>	<b>Actor</b>	<b>Business Rules Chính</b>
<b>UC-013</b>	<b>Tạo Đơn Hàng Mới</b>	Customer ▾	<b>Validate User &amp; Stock</b> (HTTP Call). Tính <b>TotalAmount</b> . <b>PUBLISH EVENT ``order.created``</b> vào <b>RabbitMQ</b> (Async) sau khi DB Transaction (ACID) hoàn tất.
<b>UC-014</b>	<b>Xem Danh Sách Đơn Hàng</b>	Admin ▾	Hỗ trợ filter theo status, sắp xếp theo <b>CreatedAt</b>

			DESC.
UC-015	Xem Đơn Hàng Theo User	Customer, A... ▾	Customer chỉ xem được đơn hàng của chính mình.
UC-016	Cập Nhật Trạng Thái Đơn Hàng	Admin, Syst... ▾	Validate luồng chuyển trạng thái hợp lệ. <b>PUBLISH EVENT</b> <code>""order.status.updated""</code> vào <b>RabbitMQ</b> (Async). Lưu vào StatusHistory.
UC-017	Xóa Đơn Hàng	Admin ▾	<b>Soft delete.</b> Chỉ cho phép xóa nếu <b>status = "Cancelled" hoặc "Pending"</b> .

Module 4: API Gateway (Port 5000 - Ocelot)

Mã UC	Tên Use Case	Actor	Business Rules Chính
UC-018	Điều Hướng Requests	System	<b>Route</b> request đến Microservice tương ứng (User, Product, Order). Xử lý <b>CORS</b> và <b>Load Balancing</b> .

### 1.3. Yêu Cầu Phi Chức Năng (Non-Functional Requirements)Hiệu Năng (Performance)

Yêu Cầu	Mục Tiêu Cụ Thể	Tiêu Chuẩn Kỹ Thuật
Thời gian phản hồi	API Gateway (routing) < <b>50ms</b> . Read operations (User/Product) < <b>200ms</b> . Create Order (Hybrid transaction) < <b>500ms</b> .	Đảm bảo trải nghiệm người dùng tối ưu và không bị gián đoạn.

<b>Khả năng chịu tải</b>	Hỗ trợ tối thiểu <b>100 concurrent users</b> . Mỗi service xử lý ít nhất <b>1000 requests/phút</b> .	Đảm bảo hệ thống ổn định trong các đợt cao điểm.
<b>Throughput</b>	API Gateway tối thiểu <b>5000 requests/phút</b> . Microservice tối thiểu <b>2000 requests/phút</b> .	Chỉ số về khả năng xử lý khối lượng giao dịch lớn trong một đơn vị thời gian.

#### Khả Năng Mở Rộng (Scalability)

1. **Scale Horizontal (Web Tier):** Tất cả các services phải là **Stateless** (không lưu session state) để dễ dàng nhân bản (run multiple instances). API Gateway (Ocelot) sẽ được cấu hình để thực hiện **Load Balancing** giữa các instances.
2. **Tách Biệt Database (Data Tier):** Áp dụng mô hình **Database per Service**. Mỗi service sử dụng **PostgreSQL độc lập** để tránh xung đột và cho phép scale database riêng biệt dựa trên tải thực tế (ví dụ: Orders DB có thể cần cấu hình mạnh hơn Users DB).
3. **Message Queue: RabbitMQ** hỗ trợ **Multiple Consumers**, cho phép dễ dàng thêm các worker/service mới để xử lý các tác vụ bất đồng bộ (ví dụ: Inventory Service) khi hệ thống phát sinh nhu cầu.

#### Độ Tin Cậy (Reliability)

1. **Tính Toàn Vẹn Dữ Liệu (ACID):** Mọi giao dịch quan trọng trong mỗi database (như tạo Order, cập nhật Stock) phải tuân thủ thuộc tính ACID để đảm bảo tính toàn vẹn. Đặc biệt, giao dịch tạo Order trong **Order Service** phải là một transaction nguyên tử.
2. **Cơ Chế Retry:** Khi giao tiếp giữa các service (Service-to-Service Communication) thất bại do lỗi tạm thời (Transient Faults - như timeout, service busy), áp dụng cơ chế **Retry tối đa 3 lần** với **Exponential Backoff** (ví dụ: 1s, 2s, 4s delay).
3. **Không Mất Tin Nhắn: RabbitMQ** sử dụng **Durable Queues** và yêu cầu **Message Acknowledgment** từ Consumer để đảm bảo tin nhắn không bị mất nếu Consumer hoặc RabbitMQ Server bị crash.
4. **Fault Tolerance:** API Gateway sẽ triển khai cơ chế **Circuit Breaker** (mở rộng) và có khả năng handle lỗi để trả về lỗi **503** (Service Unavailable) nếu một service bị down, ngăn chặn sự cố lan truyền toàn hệ thống.

#### Bảo Mật & Vận Hành

1. **Xác Thực:**
  - **Password Hashing:** Sử dụng thuật toán mạnh mẽ **BCrypt** với **Cost Factor = 12** là tiêu chuẩn bắt buộc để bảo vệ mật khẩu.
  - **Authorization (Future):** Mở rộng sang **JWT** (JSON Web Token) cho

xác thực và **RBAC** (Role-Based Access Control) cho phân quyền chi tiết.

- Log Tập Trung:** Tất cả services phải log vào **MongoDB** với format chuẩn (Timestamp, Service Name, Level, Message, Request Context) để dễ dàng truy vấn và phân tích lỗi.
- Deployment:** Sử dụng **Docker** và **Docker Compose** để đóng gói và quản lý Multi-container, đơn giản hóa quá trình phát triển (DevOps) và triển khai (Deployment).
- Monitoring:** Triển khai **Health Check Endpoints** (`/health`) cho mỗi service và theo dõi tài nguyên (CPU, Memory, Network) để kịp thời phát hiện vấn đề.

## PHẦN II: TỔNG HỢP HỆ THỐNG & TÍNH NĂNG

### 2.1. Mô Tả Chi Tiết Các Dịch Vụ (Services)Service 1: User Service

Thuộc tính	Chi tiết
Mục đích	Quản lý thông tin, xác thực, phân quyền người dùng (Customer, Admin).
Port	5001
Database	`userservice_db` (PostgreSQL)
MongoDB Collection	`microservice_users.user_logs`
API Endpoints	`GET /api/users`, `GET /api/users/{id}`, `POST /api/users`, `PUT /api/users/{id}`, `DELETE /api/users/{id}`, `POST /api/auth/login`
Business Logic	Xử lý đăng ký, đăng nhập (hashing password), phân trang, soft delete, quản lý địa chỉ.

#### Service 2: Product Service

Thuộc tính	Chi tiết
Mục đích	Quản lý sản phẩm, tồn kho (Stock), categories, reviews và rating.
Port	5002
Database	`productservice_db` (PostgreSQL)

<b>MongoDB Collection</b>	`microservice_products.product_logs`
<b>API Endpoints</b>	`GET /api/products`, `GET /api/products/{id}`, `POST /api/products`, `PUT /api/products/{id}`, `PATCH /api/products/{id}/stock`, `DELETE /api/products/{id}`
<b>Business Logic</b>	Tính toán Average Rating, quản lý tồn kho (cập nhật Stock), tự động cập nhật `IsAvailable` khi Stock = 0.

### Service 3: Order Service

Thuộc tính	Chi tiết
<b>Mục đích</b>	Quản lý đơn hàng, xử lý orders, tích hợp với các services khác, <b>Publish Events</b> .
<b>Port</b>	5003
<b>Database</b>	`orderservice_db` (PostgreSQL)
<b>MongoDB Collection</b>	`microservice_orders.order_events`
<b>API Endpoints</b>	`GET /api/orders`, `GET /api/orders/{id}`, `POST /api/orders`, `PUT /api/orders/{id}/status`, `DELETE /api/orders/{id}`
<b>Business Logic</b>	Xử lý giao dịch tạo Order (ACID Transaction), gọi HTTP đồng bộ để validate User/Stock, <b>Publish Event</b> ``order.created`` và ``order.status.updated`` vào <b>RabbitMQ</b> .

### Service 4: API Gateway (Ocelot)

Thuộc tính	Chi tiết
Mục đích	Single Entry Point, Routing, Load Balancing, Centralized CORS/Authentication.
Port	5000
Database	Không có (Stateless)
Routing Rules	`/api/auth/*` \$rightarrow\$ User Service, `/api/users/*` \$rightarrow\$ User Service, `/api/products/*` \$rightarrow\$ Product Service, `/api/orders/*` \$rightarrow\$ Order Service.

## 2.2. Các Bảng Tổng Hợp

### Bảng 1: Tổng Quan Hệ Thống

Thành Phần	Số Lượng	Mô Tả Chi Tiết
Microservices	4 ▾	User Service, Product Service, Order Service, API Gateway
Databases	3 ▾	userservice_db, productservice_db, orderservice_db (PostgreSQL)
Message Queues	1 ▾	RabbitMQ (External Server)
Logging DB	1 ▾	MongoDB Atlas (Lưu trữ log và event history)

Bảng 2: Technology Stack (Chi Tiết)

Category	Technology	Version	Mục Đích
Backend Framework	.NET	8.0 ▾	Khung phát triển chính, hiệu suất cao, đa nền tảng.
Database (SQL)	PostgreSQL	Latest ▾	Cơ sở dữ liệu quan hệ cho từng service, hỗ trợ

			ACID.
NoSQL Database	MongoDB	Atlas ▾	Logging tập trung, lưu trữ dữ liệu phi cấu trúc (events, logs).
Message Queue	RabbitMQ	3.x ▾	Giao tiếp bất đồng bộ (Event-Driven), đảm bảo tin nhắn.
API Gateway	Ocelot	Latest ▾	Routing, Load balancing, Centralized CORS.
Containerization	Docker/Docker Compose	Latest ▾	Đóng gói và quản lý môi trường phát triển/triển khai.
API Doc	Swagger	Latest ▾	Tự động sinh tài liệu API (OpenAPI).
Security	BCrypt/JWT	Latest ▾	Password Hashing và Token-based Authentication.

## PHẦN III: THIẾT KẾ PHẦN MỀM

### 3.1. Thiết Kế Tổng Thể

#### 3.1.1. Kiến Trúc Phân Tầng (Layers)

Kiến trúc phân tầng được áp dụng để đảm bảo **tính tách biệt** (Separation of Concerns) và **khả năng mở rộng ngang** (Horizontal Scaling) cho từng thành phần:

- **PRESENTATION LAYER:** Frontend (Angular) và Swagger UI - Tiếp nhận và hiển thị dữ liệu cho người dùng.
- **GATEWAY LAYER:** API Gateway (Ocelot) - Điểm tiếp nhận request đầu tiên, điều hướng đến service đích và xử lý các vấn đề xuyên suốt (cross-cutting concerns).
- **SERVICE LAYER:** Gồm các Microservices (User, Product, Order) - Chứa toàn bộ Business Logic và Controllers.
- **DATA LAYER:** PostgreSQL (Database per Service) - Nơi lưu trữ dữ liệu nghiệp vụ quan trọng, đảm bảo ACID.

- **MESSAGE QUEUE LAYER:** RabbitMQ - Đảm nhiệm vai trò giao tiếp bất đồng bộ, truyền tải Events.
- **LOGGING LAYER:** MongoDB - Hệ thống ghi log tập trung từ tất cả các services.

### 3.1.2. Design Patterns Cốt Lõi

1. **Microservices Architecture:**
  - **Mô tả:** Tách biệt hệ thống thành các dịch vụ nhỏ, mỗi dịch vụ độc lập về code, database và deployment.
  - **Lợi ích:** Tăng tốc độ phát triển (velocity), giảm rủi ro, và cho phép sử dụng công nghệ linh hoạt.
2. **Database per Service:**
  - **Mô tả:** Mỗi Microservice sở hữu một Database riêng biệt, không chia sẻ với các service khác.
  - **Lợi ích:** Đảm bảo **Loose Coupling** (ràng buộc lỏng lẻo) giữa các service, cho phép **Independent Scaling** của Data Tier.
3. **API Gateway (Ocelot):**
  - **Mô tả:** Đóng vai trò là **Single Entry Point** cho tất cả các Clients.
  - **Lợi ích:** Ẩn đi cấu trúc nội bộ, xử lý Authentication, Load Balancing, và cung cấp một API thống nhất.
4. **Event-Driven Architecture (RabbitMQ):**
  - **Mô tả:** Các services giao tiếp thông qua việc Publish và Subscribe các Events.
  - **Lợi ích:** Hỗ trợ **Asynchronous Processing** (giảm thời gian phản hồi cho luồng chính) và tăng khả năng chịu lỗi (Resilience).
5. **Repository Pattern:**
  - **Mô tả:** Tách biệt logic truy cập dữ liệu (EF Core DbContext) khỏi logic nghiệp vụ.
  - **Lợi ích:** Tăng **Testability** (dễ dàng Mock DB), dễ dàng thay đổi ORM hoặc Data Source.

### 3.2. Thiết Kế Chi Tiết (Diagrams)

#### 3.2.1. ERD (Entity Relationship Diagrams) ERD cho orderservice\_db (PostgreSQL)

Bảng	Mục Đích	Các Trường Chính	Quan hệ
Orders	Lưu thông tin đơn hàng	Id, UserId, TotalAmount, Status, ShippingAddress, PaymentMethod,	1 \$rightarrow... ▾

		CreatedAt, IsDeleted	
<b>OrderItems</b>	Chi tiết items trong order	Id, OrderId (FK), ProductId, ProductName, Quantity, UnitPrice, SubTotal	* \$\\rightarrow...
<b>OrderStatusHistory</b>	Lịch sử thay đổi status	Id, OrderId (FK), Status, Notes, ChangedBy, CreatedAt	* \$\\rightarrow...

Tóm Tắt ERD của các Databases khác

- **userservice\_db**: Gồm các bảng **Users** (lưu thông tin đăng nhập, hash password) và **UserAddresses** (địa chỉ của User).
- **productservice\_db**: Gồm các bảng **Products** (thông tin sản phẩm, tồn kho), **ProductReviews** (đánh giá, rating) và **ProductTags**.

### 3.2.2. Sequence Diagram: Tạo Đơn Hàng (Hybrid - Sync + Async)

Luồng giao dịch phức tạp nhất, minh họa sự kết hợp giữa giao tiếp Đồng bộ (HTTP) và Bất đồng bộ (RabbitMQ):

Bước	Diễn Giải Chi Tiết	Kênh Giao Tiếp	Mục Đích
1-3	<b>Xác Thực Đồng Bộ (Validation)</b>	HTTP POST/GET	Order Service gọi User Service và Product Service để kiểm tra tính hợp lệ của User và Stock. Nếu lỗi, trả về 400 Bad Request ngay lập tức.
4	<b>Giao Dịch ACID (DB Transaction)</b>	PostgreSQL	Order Service bắt đầu Transaction, INSERT Order và OrderItems. <b>COMMIT</b> khi thành công.

5	Event Publishing (Async)	RabbitMQ	Order Service <b>PUBLISH</b> ``order.created`` vào Message Queue. Đây là điểm cắt Async.
6-7	Phản hồi Tức Thì	HTTP Response (201)	Order Service trả về <b>201 Created</b> cho Client, không chờ RabbitMQ xử lý.
8	Xử lý Hậu kỳ (Consumer)	RabbitMQ \$rightarrow\$ Inventory Service	<b>Inventory Service (Consumer)</b> lắng nghe event ``order.created`` và thực hiện <b>giảm tồn kho</b> một cách độc lập.

### 3.3. Thiết Kế API (Chi Tiết)Nguyên Tắc RESTful

- **Tài nguyên:** Sử dụng danh từ số nhiều: `/api/users`, `/api/products`.
- **Động từ:** Sử dụng HTTP Methods tương ứng:
  - **GET** (Retrieve), **POST** (Create), **PUT** (Update/Replace), **PATCH** (Partial Update), **DELETE** (Soft Delete).
- **Trạng thái:** Status Codes 200/201 (Thành công), 400 (Lỗi nghiệp vụ/Validation), 401/403 (Authentication/Authorization), 404 (Không tìm thấy).

Chi Tiết API Endpoints (Product Service - Ví dụ)

Method	Endpoint (Route)	Mô Tả	Logic Chính	Response Status
GET	<code>/api/pr...</code> ▾	Danh sách Sản Phẩm	Pagination, Filter theo `category`, Search theo `name`. Chỉ trả về `IsAvailable=true`.	200 OK ▾
POST	<code>/api/pr...</code> ▾	Tạo Sản Phẩm Mới	Validate Price>0, Stock>=0. INSERT vào	201 Cre... ▾

			DB, log vào MongoDB.	
<b>PATCH</b>	/api/pr... ▾	Cập nhật Tồn Kho	Tính toán Stock mới = Stock cũ + Quantity. Nếu Stock=0, set `IsAvailable=false`.	200 OK ▾
<b>DELETE</b>	/api/pr... ▾	Xóa Sản Phẩm	Soft Delete (`IsDeleted=true`). Check rằng buộc Orders trước khi xóa.	200 OK ▾

### 3.4. Thiết Kế Message Queue (RabbitMQ) Cấu Hình Queue

Queue Name	Mục Đích	Publisher	Consumer (Tương lai)	Queue Properties
``order.created``	Sự kiện Order được tạo	Order Service	Inventory Service, Email Service	Durable: True
``order.status.updated``	Sự kiện thay đổi trạng thái	Order Service	Notification Service, Shipping Service	Durable: True

#### Cơ Chế Xử Lý Lỗi Bất Đồng Bộ

1. **Retry Mechanism (Publisher):** Order Service sẽ retry tối đa 3 lần với Exponential Backoff nếu việc Publish Event vào RabbitMQ thất bại.
2. **Dead Letter Queue (DLQ):** Messages không thể xử lý sau nhiều lần retry của Consumer sẽ được tự động chuyển vào **DLQ** để được Admin xem xét và xử lý thủ công, đảm bảo không mất dữ liệu.
3. **Message Acknowledgment:** Consumer phải gửi tín hiệu **ACK** (Acknowledge) sau khi xử lý Event thành công. Nếu không ACK (Consumer crash), message sẽ được redeliver.

### 3.5. Thiết Kế Bảo Mật & Deployment Cơ Chế Xác Thực (Authentication)

- **Password Hashing:** BCrypt với Cost Factor = 12 là tiêu chuẩn bắt buộc

cho mọi mật khẩu.

- **JWT Flow (Sẽ triển khai):**

1. Client gửi **Credentials**  $\rightarrow$  API Gateway  $\rightarrow$  User Service.
2. User Service **Validate** và **Generate JWT Token** (chứa claims: UserId, Role).
3. Client gửi **JWT Token** trong header **Authorization: Bearer <token>** cho mọi request tiếp theo.
4. API Gateway/Microservice **Validate JWT** và cấp quyền truy cập.

### Cách Đóng Gói và Triển Khai (Docker & Orchestration)

1. **Containerization (Docker):** Mỗi Microservice có một **Dockerfile** riêng. Áp dụng **Multi-stage build** để:
  - Giảm kích thước image cuối cùng (chỉ chứa runtime, không chứa SDK).
  - Tăng tốc độ Build và tăng cường bảo mật.
2. **Orchestration (Docker Compose):** Sử dụng **docker-compose.yml** để định nghĩa và chạy toàn bộ hệ thống:
  - **Services:** Định nghĩa 4 Microservices (User, Product, Order, Gateway) và các dependency (PostgreSQL, RabbitMQ, MongoDB).
  - **Networks:** Tạo mạng nội bộ (**microservice-network**) để các services giao tiếp với nhau mà không cần expose ra ngoài.
  - **Port Mapping:** Chỉ expose API Gateway (5000) và Frontend (4200) ra bên ngoài.
3. **Quy Trình Triển Khai Production:**
  - Build images (**docker-compose build**) và đẩy lên Docker Registry.
  - Triển khai trên Server: **docker-compose up -d**.
  - Thiết lập **Nginx Reverse Proxy** phía trước API Gateway (Port 5000) để xử lý **SSL Certificates (HTTPS)** và caching.