ỦY BAN NHÂN DÂN TP . HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP GIỮA KỲ

Học phần: Kiểm thử phần mềm

Đề tài: **Bước 02 – Thiết kế kiến trúc hệ thống cho đề tài**

Giảng viên hướng dẫn: **TS. Đỗ Như Tài**

Nhóm sinh viên thực hiện:

STT	Họ và tên	MSSV
1	**Trang Gia Huy**	**3122411068**
2	**Nguyễn Lê Quỳnh Hương**	**3122411078**

TP. HỒ CHÍ MINH, THÁNG 4 NĂM 2025

MỤC LỤC

BẢNG PHÂN CÔNG CÔNG VIỆC

STT	Họ và tên	MSSV	Nội dung công việc
1	Trang Gia Huy	3122411068	- Soạn báo cáo - Thiết kế Decomposition View C4
2	Nguyễn Lê Quỳnh Hương	3122411078	- Thiết kế Communication View - Thiết kế Deployment View - Thiết kế Sơ đồ khối

1. Sơ đồ khối:

1.1. Cơ sở phương pháp luận:

Thiết kế dựa trên Clean Architecture / Hexagonal Architecture:

- **Chia layer (UI, Application, Domain, Infrastructure):** dễ mở rộng, thay đổi mà không ảnh hưởng toàn hệ thống.
- **Mỗi bounded context là một module/dịch vụ nghiệp vụ**: giảm coupling, tăng cohesion.
- **SPA ở frontend**: tối ưu UX, tận dụng API backend.
- **Load balancer:** mở rộng theo chiều ngang, hỗ trợ high availability.

Đây là sự kết hợp:

- **Microservices (hoặc Modular Monolith):** mỗi module là một service logic rõ ràng.
- **DDD (Domain-Driven Design):** domain layer làm trung tâm.
- **N-tier / Layered Architecture:** backend tách 4 tầng.

2.2. Lý do lựa chọn kiến trúc:

1. Khả năng mở rộng (Scalability)

- Load balancer và SPA hỗ trợ nhiều user đồng thời.
- Backend chia nhỏ thành module, có thể scale từng phần.

2. Dễ bảo trì và phát triển (Maintainability)

- Layered structure → rõ trách nhiệm.
- Dễ thay thế công nghệ ở tầng UI hoặc Infrastructure.

3. Tính linh hoạt (Flexibility)

- Có thể triển khai thành microservice hoặc giữ ở modular monolith.
- Domain độc lập → dễ tái sử dụng business logic

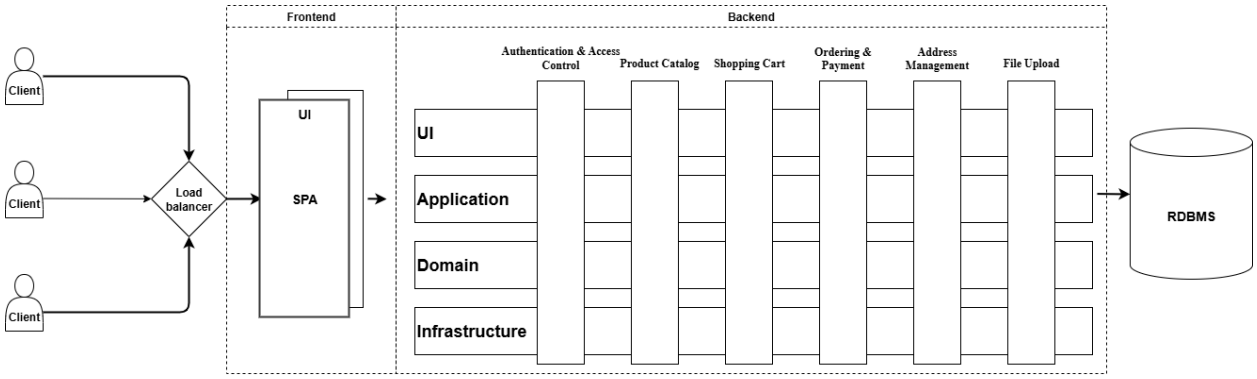
4. Tối ưu trải nghiệm người dùng

- SPA giảm số lần reload, frontend xử lý nhiều phần UI, backend tập trung vào nghiệp vụ.

5. An toàn & Quản lý truy cập

- Authentication & Access Control riêng biệt, dễ tích hợp OAuth2, JWT.

2.3. Thiết kế sơ đồ:



1. Frontend:

- **Client:** Người dùng (browser/app) truy cập hệ thống.
- **Load balancer:** Cân bằng tải để phân phối request từ nhiều client, tránh quá tải một server.
- **SPA (Single Page Application):** Ứng dụng giao diện một trang (React), giảm tải cho backend, tăng trải nghiệm mượt mà.
- **UI:** Giao diện hiển thị cho người dùng.

2. Backend:

Chia thành nhiều bounded context/dịch vụ nghiệp vụ, mỗi dịch vụ tuân theo 4 lớp (theo mô hình Layered/Clean Architecture):

- **UI layer:** Giao tiếp giữa frontend và backend (REST API).
- **Application layer:** Chứa luồng nghiệp vụ, orchestration, gọi đến domain.
- **Domain layer:** Business logic, rule nghiệp vụ cốt lõi.
- **Infrastructure layer:** Kết nối DB, external service, file storage, message broker…

Các dịch vụ cụ thể:

- **Authentication & Access Control:** Đăng nhập, phân quyền.
- **Product Catalog:** Quản lý sản phẩm, danh mục.
- **Shopping Cart:** Giỏ hàng.
- **Ordering & Payment:** Đặt hàng, thanh toán.
- **Address Management:** Quản lý địa chỉ giao hàng.
- **File Upload:** Quản lý upload ảnh, tài liệu.

3. Database:

- **RDBMS:** Hệ quản trị cơ sở dữ liệu quan hệ (PostgreSQL) dùng để lưu trữ dữ liệu chính.

2. Communication View:

2.1. Cơ sở phương pháp luận:

Thiết kế này dựa trên các nguyên tắc:

1. Service-Oriented Communication

- Mỗi nhóm nghiệp vụ được gom vào 1 controller riêng → high cohesion, low coupling.
- API contract rõ ràng → dễ bảo trì và mở rộng.

2. RESTful Design Principle

- Sử dụng các HTTP verb chuẩn (GET, POST, PUT, DELETE).
- Tài nguyên được định danh qua URL (/api/products/{id}).
- CRUD mapping trực quan:
  - GET: đọc
  - POST: tạo
  - PUT: cập nhật
  - DELETE: xoá

3. API First & Documentation

- Áp dụng API-First Development: định nghĩa API contract trước → rồi mới implement backend/frontend.
- Swagger giúp tạo communication view:
  - Developer backend biết implement cái gì.
  - Developer frontend biết cách gọi.
  - Tester biết cách verify.
  - Stakeholder có thể xem và góp ý.

4. Alignment với kiến trúc tổng thể

- **Phù hợp với sơ đồ khối trước** (SPA frontend + backend nhiều module).

- **SPA chỉ cần call API:** không phụ thuộc vào ngôn ngữ backend.
- **Có thể mở rộng sang mobile app, partner integration** mà không thay đổi backend core.

2.2. Lý do lựa chọn kiến trúc:

1. Tính rõ ràng trong giao tiếp

- API là hợp đồng chính thức giữa các thành phần.
- Giúp giảm misunderstand giữa team frontend & backend.

2. Khả năng tích hợp và mở rộng

- Dễ tích hợp với mobile app, hệ thống đối tác (qua API Gateway).
- Có thể triển khai microservices sau này mà vẫn giữ contract API.

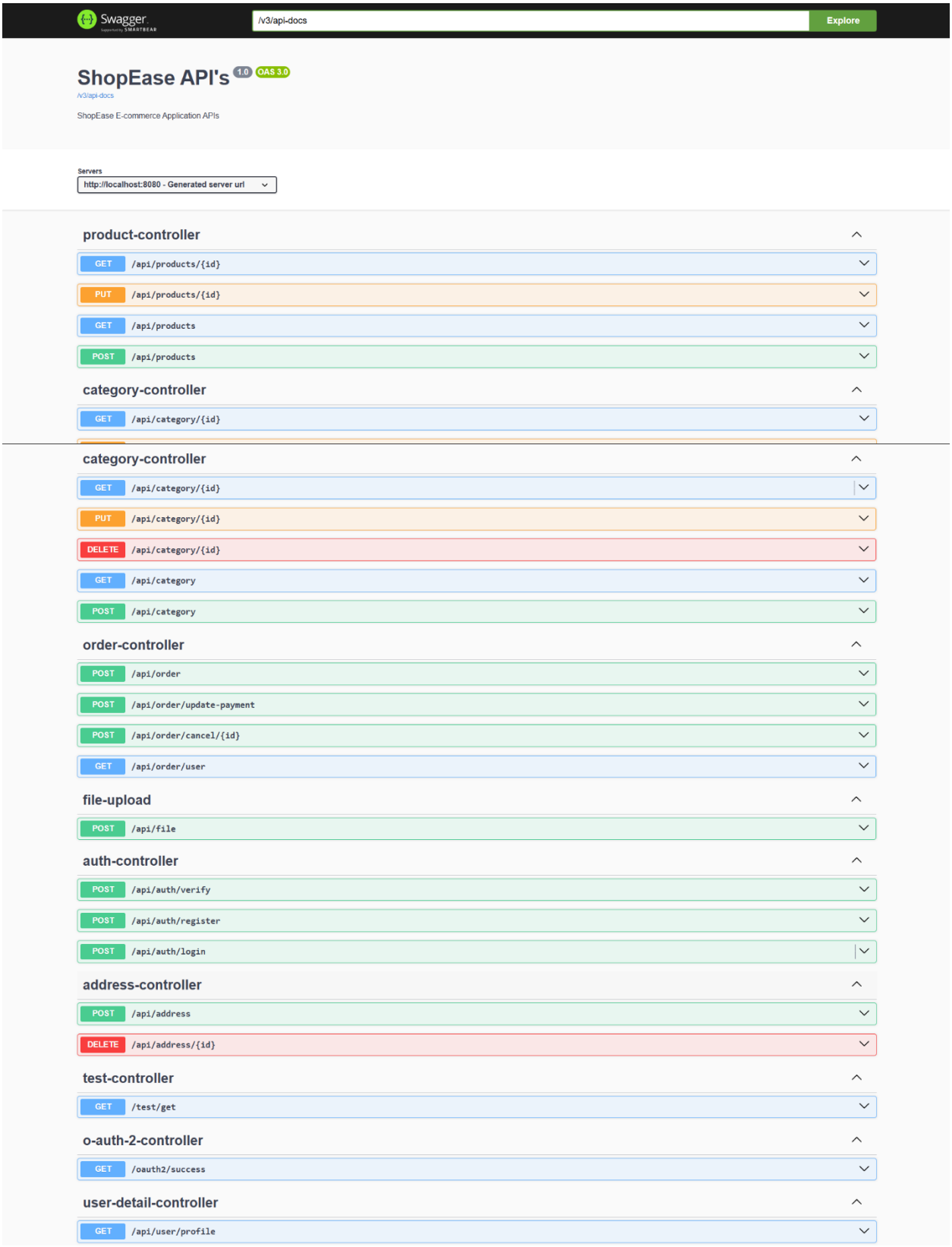
3. Khả năng kiểm thử và bảo trì

- Swagger UI cho phép test trực tiếp.
- Developer, tester, BA đều có thể xem và kiểm tra.

4. Chuẩn hóa & công cụ hỗ trợ

- OpenAPI là chuẩn công nghiệp, hỗ trợ sinh code stub, SDK, test script.
- Tự động sinh tài liệu: tiết kiệm effort viết doc thủ công.

2.3. Thiết kế sơ đồ:


Swagger
/v3/api-docs
Explore
ShopEase API's
1.0
OAS 3.0
/v3/api-docs
ShopEase E-commerce Application APIs
Servers
http://localhost:8080 - Generated server url
product-controller
GET /api/products/{id}
PUT /api/products/{id}
GET /api/products
POST /api/products
category-controller
GET /api/category/{id}
category-controller
GET /api/category/{id}
PUT /api/category/{id}
DELETE /api/category/{id}
GET /api/category
POST /api/category
order-controller
POST /api/order
POST /api/order/update-payment
POST /api/order/cancel/{id}
GET /api/order/user
file-upload
POST /api/file
auth-controller
POST /api/auth/verify
POST /api/auth/register
POST /api/auth/login
address-controller
POST /api/address
DELETE /api/address/{id}
test-controller
GET /test/get
o-auth-2-controller
GET /oauth2/success
user-detail-controller
GET /api/user/profile

Swagger (OpenAPI 3.0) ở đây mô tả giao tiếp giữa frontend ↔ backend thông qua các REST API endpoint.

Nó chính là giao diện giao tiếp chính thức cho client, thay thế cho việc client gọi trực tiếp vào logic backend.

Các controller đại diện cho các nhóm nghiệp vụ chính:

- **product-controller:** CRUD sản phẩm (/api/products)
- **category-controller:** CRUD danh mục (/api/category)
- **order-controller:** đặt hàng, cập nhật thanh toán, huỷ đơn, xem đơn hàng (/api/order)
- **file-upload:** upload file (/api/file)
- **auth-controller:** xác thực, đăng ký, đăng nhập (/api/auth)
- **address-controller:** quản lý địa chỉ (/api/address)
- **user-detail-controller:** lấy thông tin hồ sơ người dùng (/api/user/profile)
- **o-auth-2-controller:** OAuth2 login callback (/oauth2/success)
- **test-controller:** endpoint test (/test/get)

3. Deployment View:

3.1. Cơ sở phương pháp luận:

1. DevOps & CI/CD pipeline

- Ứng dụng DevOps culture: code → build → deploy tự động hóa.
- GitHub Actions đảm bảo build/test liên tục khi có thay đổi.
- Giảm rủi ro manual deployment, tăng tốc release.

2. Containerization (Docker-based Deployment)

- Dùng Docker để đóng gói ứng dụng: portable, nhất quán giữa dev/test/prod.
- Docker Hub làm registry trung tâm cho image.
- Docker Compose hỗ trợ triển khai nhiều service dễ dàng, phù hợp với kiến trúc microservices hoặc modular monolith.

3. Infrastructure as Code (IaC)

- Dockerfile và docker-compose.yml là code mô tả hạ tầng, dễ version control và tái sử dụng.
- Phù hợp với nguyên lý 12-Factor App.

4. Agile/Continuous Delivery methodology

- Tích hợp với quy trình phát triển Agile.
- Mỗi commit/pull request đều có thể build: luôn sẵn sàng release.

3.2. Lý lo lựa chọn kiến trúc:

1. Tính tự động & đáng tin cậy

- Giảm sai sót do thao tác thủ công.
- CI/CD giúp phát hiện lỗi sớm.

2. Khả năng mở rộng & tính linh hoạt

- Dùng Docker → triển khai được trên nhiều môi trường (local, cloud).
- Compose → dễ mô phỏng production trên môi trường dev/test.

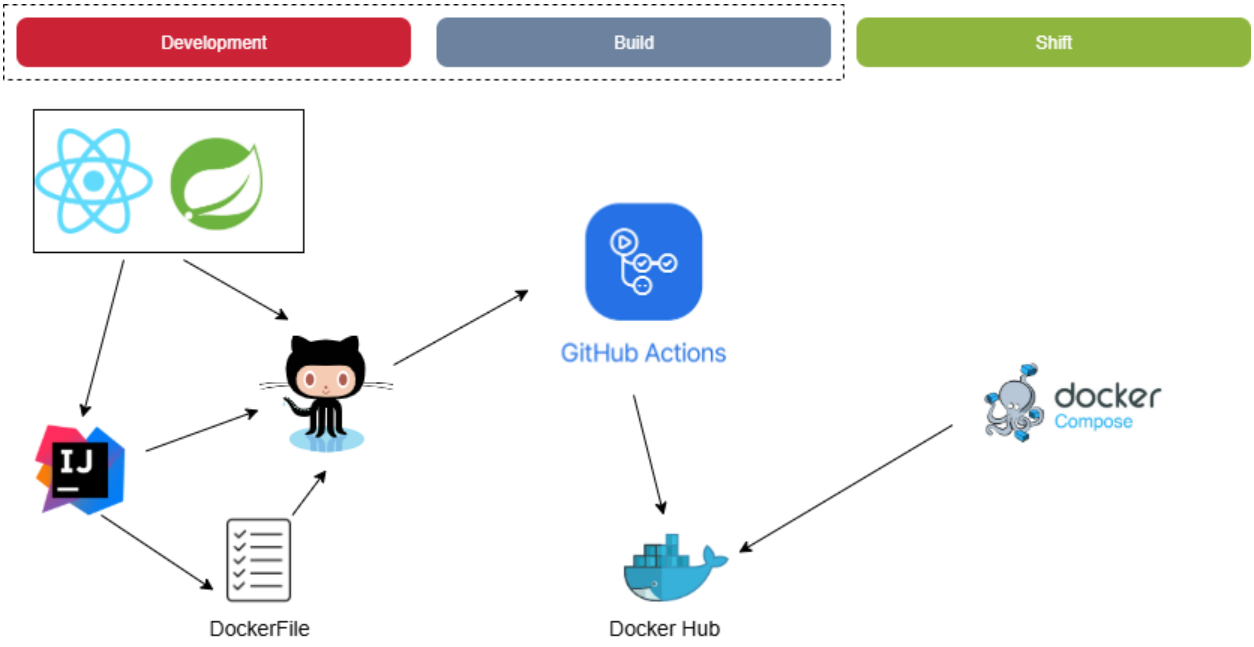
3. Tính tái sử dụng & chuẩn hóa

- Docker image chuẩn → ai cũng chạy được, bất kể OS/hệ điều hành.
- Docker Hub lưu trữ version image, dễ rollback khi lỗi.

4. Chi phí & hiệu quả

- Tận dụng công cụ open-source (GitHub Actions, Docker).
- Không cần hạ tầng phức tạp như Kubernetes ở giai đoạn đầu.

3.3. Thiết kế sơ đồ:



1. Dev code trên IntelliJ (React, Spring).

2. Commit code lên GitHub.

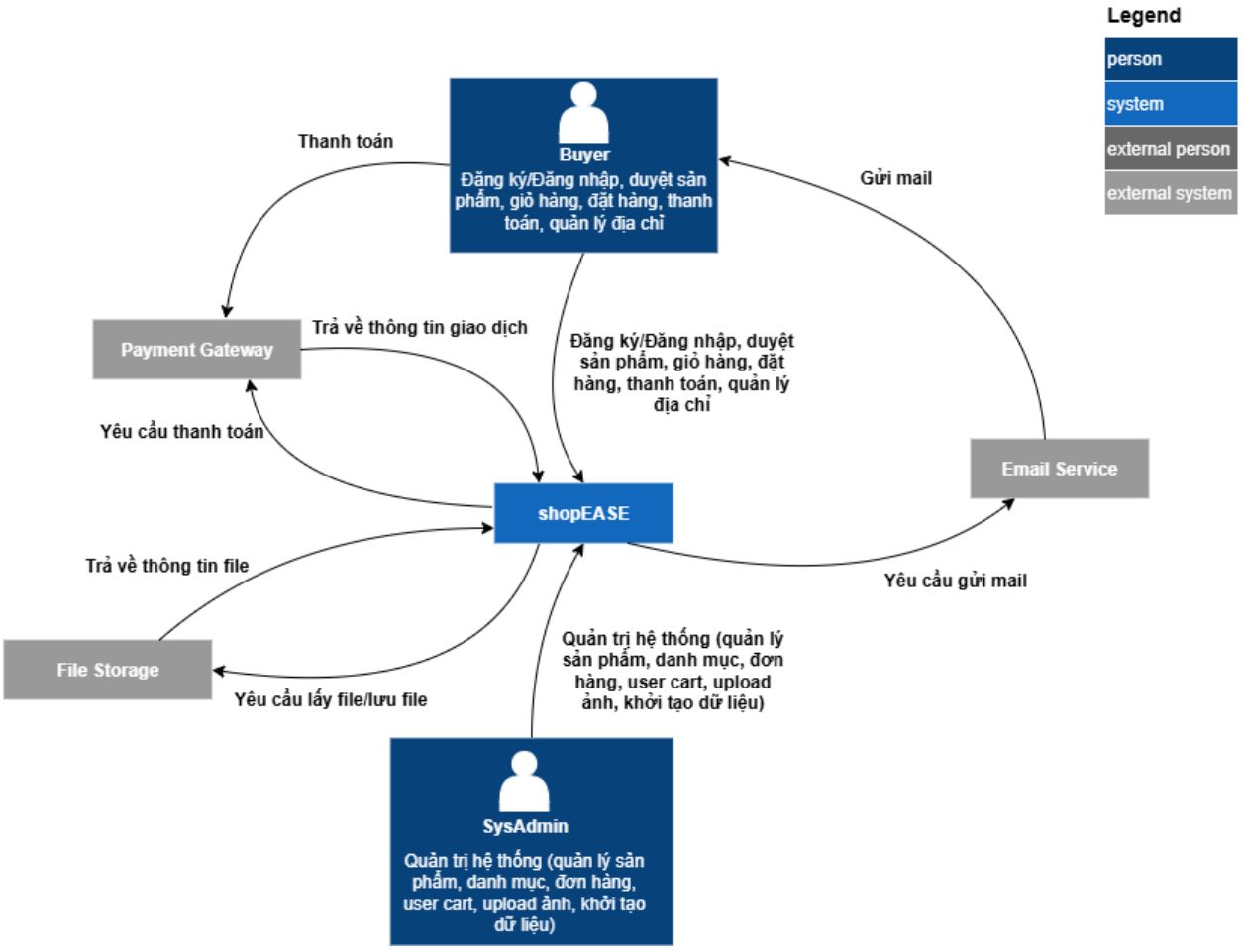
3. GitHub Actions trigger CI/CD pipeline:

- Build ứng dụng,
- Tạo Docker image dựa trên Dockerfile,
- Push image lên Docker Hub.

4. Docker Compose pull image từ Docker Hub → triển khai thành nhiều container (frontend, backend, db).

4. Decomposition View C4:

4.1. Sơ đồ C1 - System Context:



Tóm lược:

Hệ thống là một Modular Monolith phục vụ miền E-Commerce/Online Clothing Shop: người mua đăng ký tài khoản, duyệt sản phẩm, quản lý giỏ hàng, đặt đơn và thanh toán; quản trị viên quản lý sản phẩm, danh mục, đơn hàng, người dùng và dữ liệu mẫu.

Các actor/chức năng cấp cao được hiện thực qua các module sau:

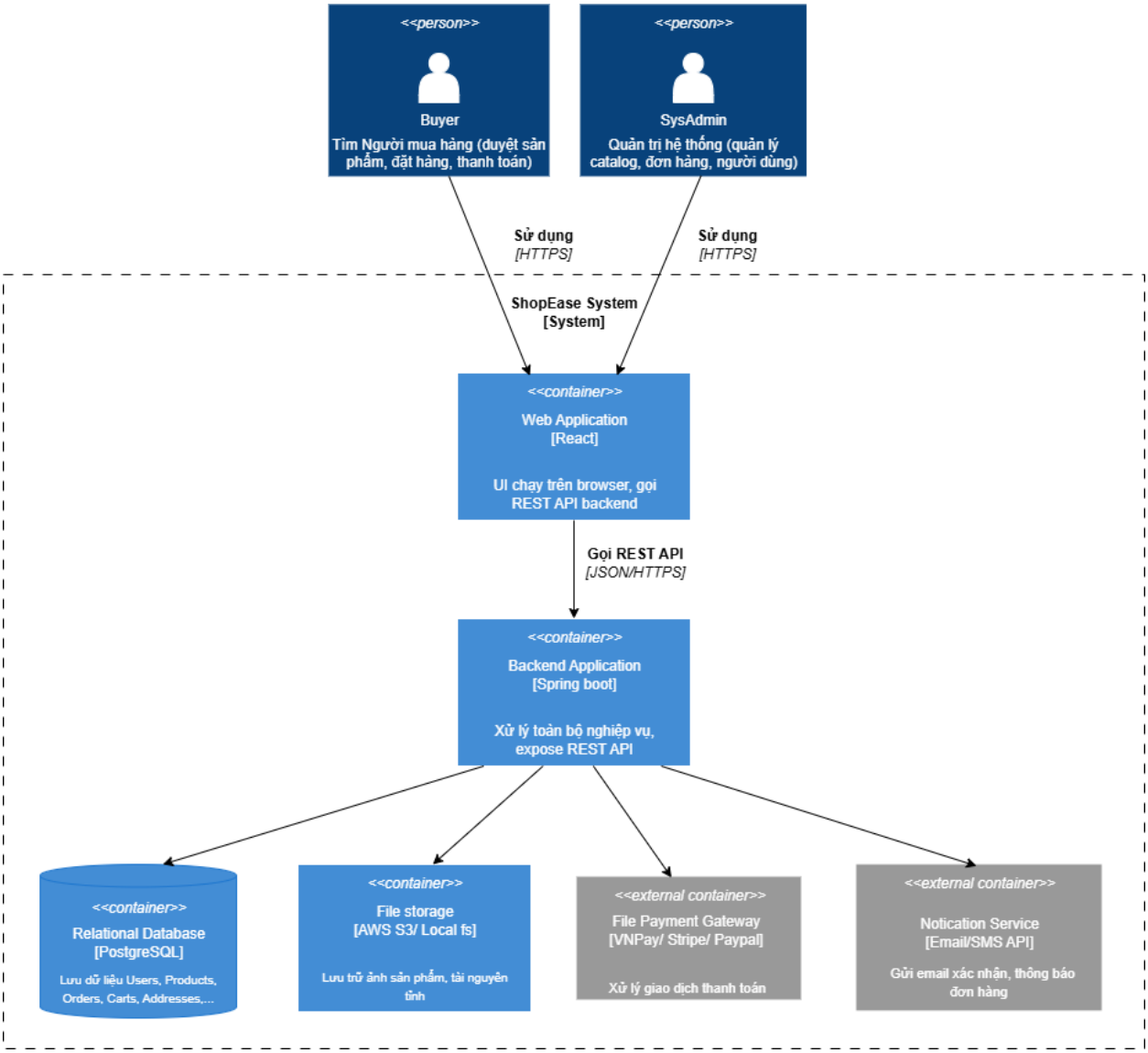
- **Authentication & Access Control**: đăng ký, đăng nhập, quản lý profile, phân quyền.
- **Product Catalog**: duyệt/bộ lọc sản phẩm, xem chi tiết, quản lý danh mục.
- **Shopping Cart**: thêm/xóa/cập nhật giỏ hàng, quản lý cart người dùng.

- **Ordering & Payment**: checkout, xác nhận, trạng thái đơn hàng, tích hợp Payment Gateway.
- **Address Management**: lưu, cập nhật, chọn địa chỉ giao hàng.
- **File Upload**: quản lý hình ảnh sản phẩm, tích hợp lưu trữ ngoài.
- **Administration**: quản lý dữ liệu mẫu, theo dõi hoạt động hệ thống.

Các external systems:

- **Payment Gateway** để xử lý giao dịch an toàn.
- **Notification Service (Email/SMS)** để gửi xác nhận và thông báo đơn hàng.
- **File Storage/CDN** để lưu trữ và phân phối hình ảnh sản phẩm.

4.2. Sơ đồ C2 – Container:



Tóm lược:

Web Client (SPA):

Vai trò: giao diện cho Buyer và SysAdmin; chỉ hiển thị UI và gửi request đến API backend. Không chứa business logic.

API Layer (Spring Boot REST Controllers)

Vai trò: nhận request → xác thực/ủy quyền (dựa trên User Access) → mapping DTO → gửi Command/Query tới module nghiệp vụ → trả response JSON.

Nguyên tắc: API không chứa business logic, chỉ làm "cửa ngõ" vào hệ thống.

Các Module nghiệp vụ (trong cùng tiến trình — modular monolith, đóng gói chặt chẽ và độc lập dữ liệu):

- **Authentication & User Access:** đăng ký, đăng nhập, quản lý session/token, phân quyền Buyer/Admin.
- **Product Catalog:** quản lý sản phẩm, biến thể, tồn kho, danh mục, hình ảnh, new arrivals.
- **Shopping Cart:** quản lý giỏ hàng, số lượng, tính tổng tiền, kiểm soát giỏ không hợp lệ.
- **Ordering:** quản lý đơn hàng, trạng thái (pending → confirmed → shipped → delivered).
- **Payment:** tích hợp cổng thanh toán ngoài (Stripe/VNPay/PayPal), xác nhận giao dịch.
- **Address Management:** CRUD địa chỉ giao hàng.
- **File Upload:** upload ảnh sản phẩm, quản lý resource chính/phụ.
- **Administration:** thao tác đặc thù SysAdmin (quản lý catalog, orders, carts).

Database Layer (PostgreSQL/MySQL):

Mỗi module sở hữu schema riêng trong cùng database. Không chia sẻ bảng. Có thể tách ra DB riêng khi scale.

File Storage (AWS S3 / Local FS)

Lưu trữ ảnh sản phẩm, tài nguyên tĩnh, tham chiếu bởi Catalog.

External Integrations:

- Payment Gateway: xử lý thanh toán online.

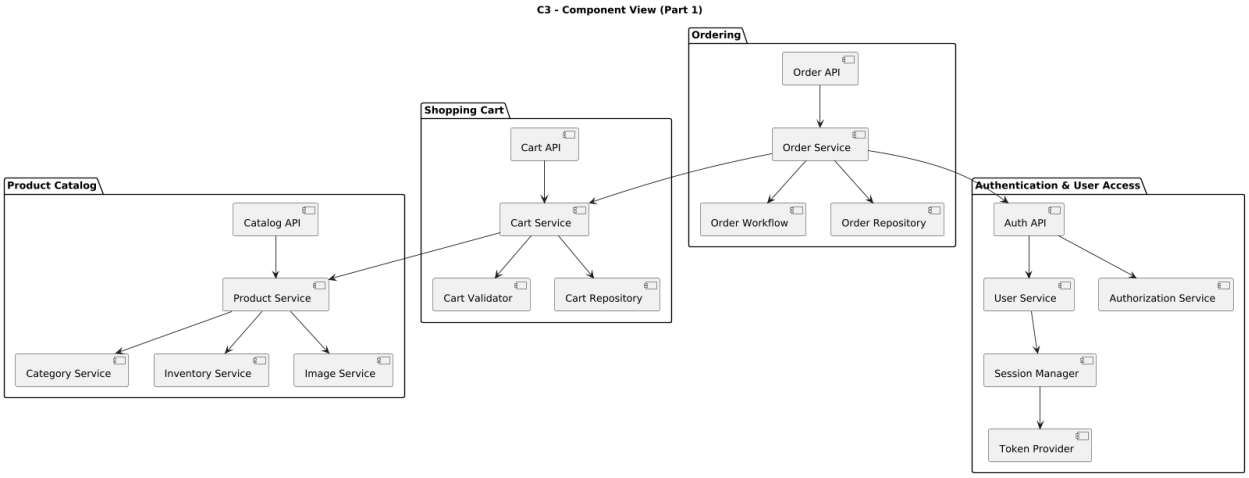
- Notification Service (Email/SMS): gửi email xác nhận đơn hàng, OTP, thông báo giao hàng.

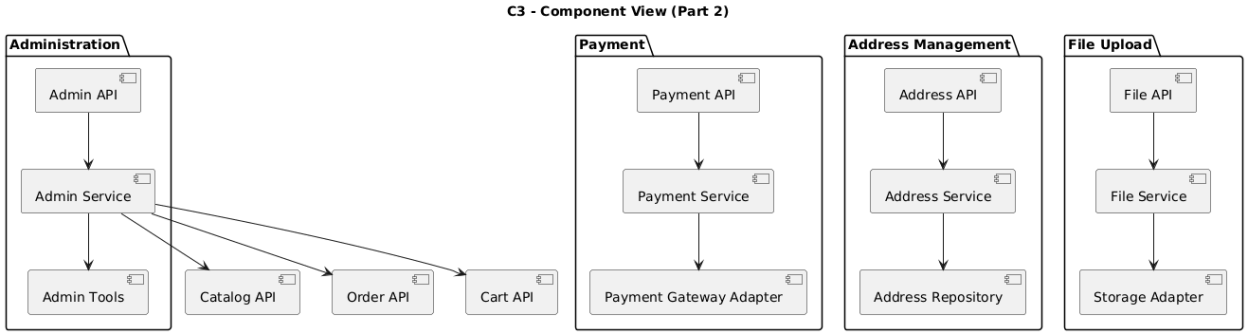
Integration Rules:

- API Layer không có logic nghiệp vụ; chỉ nhận/gửi.
- API ↔ Module thông qua interface nhỏ (Command/Query).
- Module-to-Module tích hợp bất đồng bộ qua In-Memory Event Bus (ví dụ: OrderCreatedEvent được Payment/Notification subscribe).
- Mỗi module quản lý dữ liệu riêng (schema riêng).
- Chỉ phụ thuộc vào Integration Events khi cần chia sẻ thông tin.
- Có Composition Root/IoC riêng, được API khởi tạo.
- Đóng gói cao (chỉ public những gì cần expose).

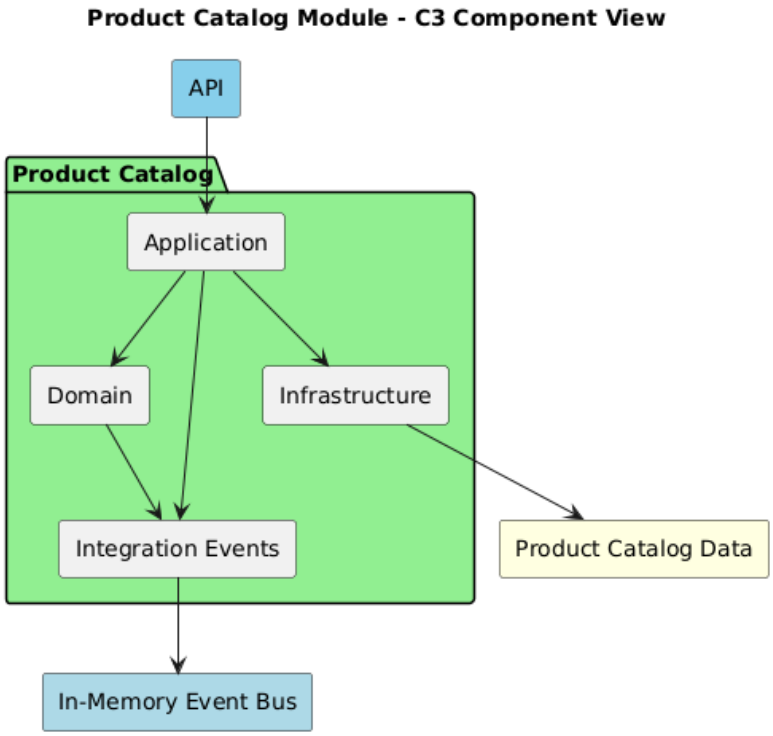
4.3. Sơ đồ C3 – Component:

High-Level:

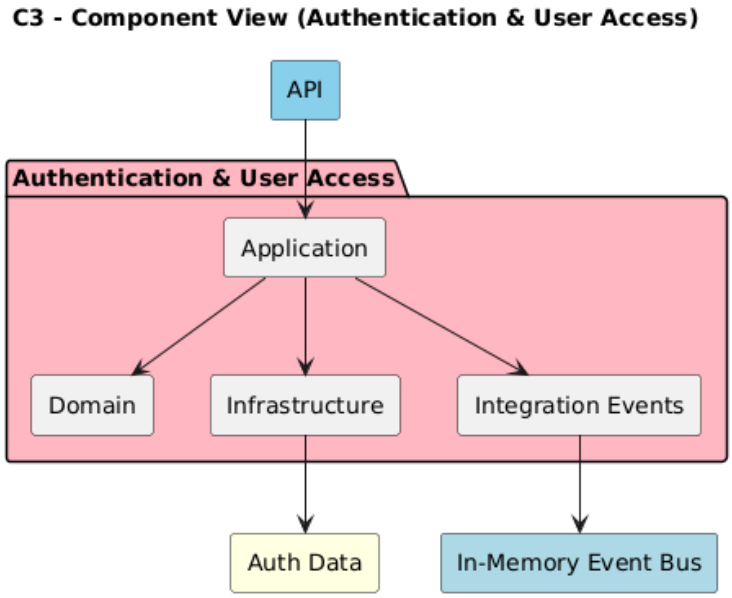




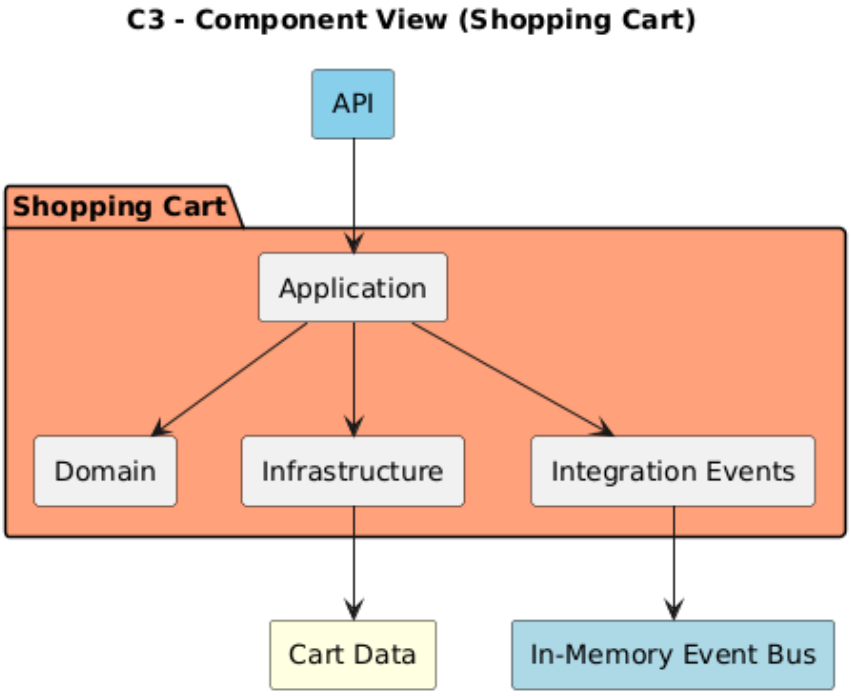
Module-Level: Product Catalog



Module-Level: Authentication & User Access



Module-Level: Shopping Cart



Mỗi module trong hệ thống E-Commerce (Authentication, Product Catalog, Shopping Cart, Ordering & Payment, Address, File Upload, Administration) tuân thủ Clean Architecture, được chia thành 4 submodules/assemblies chính:

Application

- Xử lý Use Case dưới dạng Command/Query (CQRS).
- Điều phối luồng nghiệp vụ, phát Domain Events nội bộ.
- Khi cần tích hợp liên-module, publish Integration Events lên Event Bus.

Domain

- Chứa mô hình DDD: Entities (Buyer, Order, Cart, Product…), Value Objects (Address, Price, StockQuantity…).
- Domain giàu hành vi, persistence-ignorant (không phụ thuộc hạ tầng).
- Đảm bảo encapsulation cao, chỉ expose ngôn ngữ nghiệp vụ.

Infrastructure

- Truy cập dữ liệu qua Repository (RDBMS).

- Giao tiếp với hệ thống ngoài (Payment Gateway, File Storage).
- Khởi tạo module, tích hợp logging, config, event bus.

IntegrationEvents

- Là hợp đồng sự kiện (contract) công bố ra Event Bus (ví dụ: OrderPlaced, ProductStockDecreased).
- Đây là điểm duy nhất để module khác phụ thuộc (Pub/Sub bất đồng bộ).

Dòng chảy xử lý (Request & CQRS)

1. API nhận HTTP request từ Buyer hoặc SysAdmin → xác thực/ủy quyền thông qua Authentication module.

2. API forward request thành Command/Query vào Application layer của module đích (ví dụ: AddProductCommand, PlaceOrderCommand).

3. Application layer gọi Domain để thực thi logic nghiệp vụ, phát sinh Domain Event nội bộ nếu có.

4. Nếu cần tích hợp liên-module, Application sẽ publish Integration Event lên Event Bus để module khác subscribe và xử lý bất đồng bộ (ví dụ: khi Order đặt thành công → CartService clear giỏ hàng).

Dữ liệu & biên giới module

- Data ownership: mỗi module quản lý schema dữ liệu riêng (User, Product, Cart, Order, Address, File). Không chia sẻ bảng giữa modules.
- Integration: các module giao tiếp qua Integration Events trên Event Bus, tuyệt đối không gọi method trực tiếp hoặc đọc chéo DB.

Bảo mật & khởi tạo

- Authentication & Access Control đảm trách AuthN/AuthZ (JWT). API sẽ kiểm tra quyền trước khi gọi use case.

- Khởi tạo module: tại Startup, API gọi Initialize(...) cho từng module, truyền cấu hình (connection string, logger, mail server, external API key…), đồng thời khởi động Event Bus.

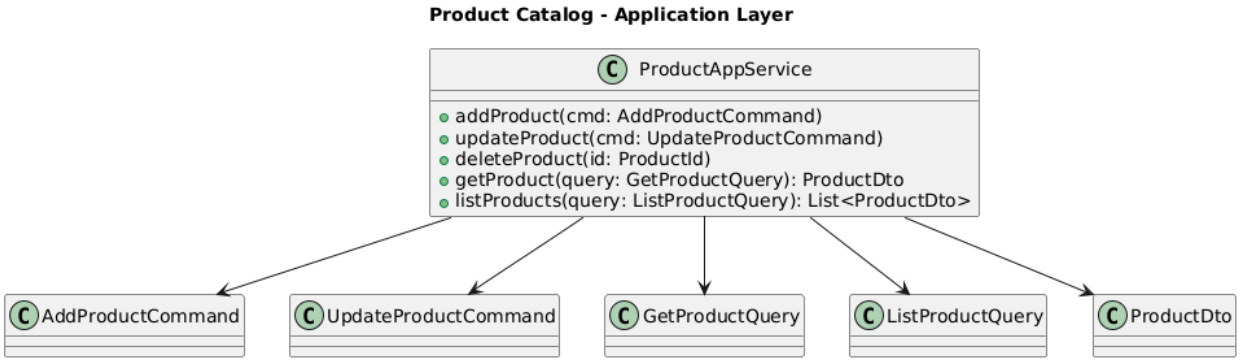
Kiểm thử & chất lượng

- Unit Test: kiểm thử logic Domain (ví dụ: tính tổng tiền order, cập nhật stock).
- Integration Test: test Application layer + Repository.
- System Integration Testing: test end-to-end giữa các module qua Event Bus.
- CI/CD pipeline: chạy static analysis, mutation testing, đảm bảo kiến trúc đúng chuẩn, production-ready.

4.4. Sơ đồ C4 - Code/Implementation Level:

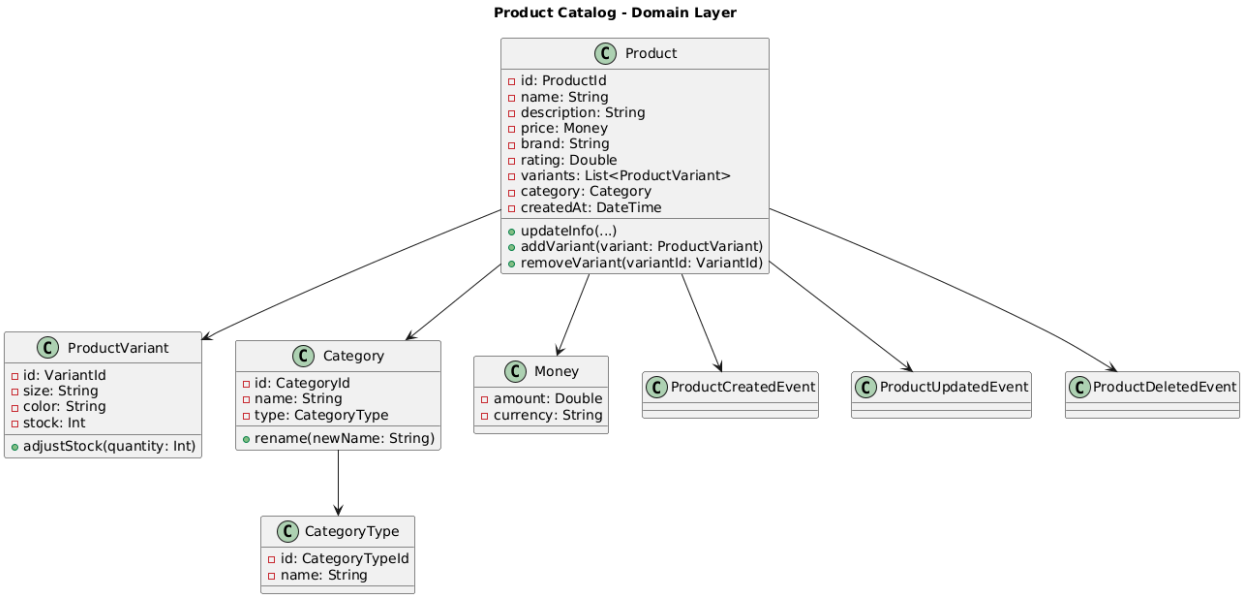
4.4.1. Class Diagram cho module Product Catalog:

Application Layer: Chứa ProductAppService xử lý các Command/Query, không chứa logic nghiệp vụ, chỉ gọi Domain.

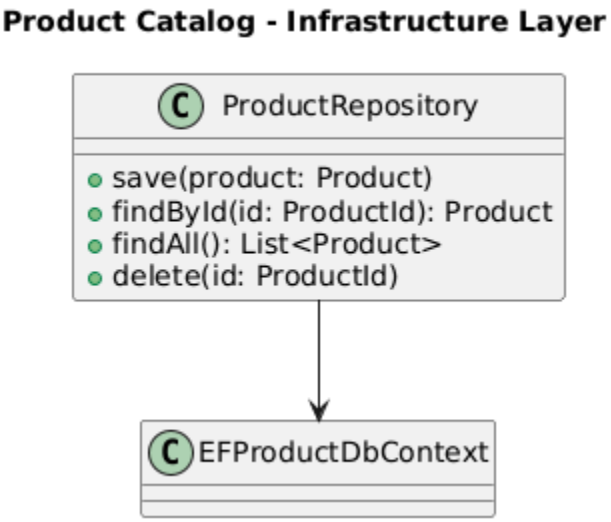


Domain Layer:

- Product là Aggregate Root, sở hữu ProductVariant, Category.
- Money là Value Object để tránh primitive obsession.
- Sinh ra Domain Events khi có thay đổi (Created/Updated/Deleted).



Infrastructure Layer: ProductRepository + EFProductDbContext để lưu trữ



Integration Events: contract để module khác subscribe khi product thay đổi.