**Bài 1**

**A diagram of a flowchart

AI-generated content may be incorrect.**

**Bài 6**

**ATM**

**A diagram of a process

AI-generated content may be incorrect.**

**Card controller**

**A diagram of a bank card

AI-generated content may be incorrect.**

**Bài ứng dụng**

**Câu 4**

**Use Case Name:** Mua sản phẩm từ trang chi tiết sản phẩm  
**Actor(s):** Buyer (Người mua), System  
**Maturity:** Focused  
**Summary:** Người mua chọn sản phẩm tại trang chi tiết và thực hiện mua sản phẩm, hệ thống sẽ kiểm tra tồn kho, thêm sản phẩm vào giỏ hàng và cập nhật thông tin.

**Basic Course of Events**

| **Actor Action** | **System Response** |
| --- | --- |
| 1. Buyer đã đăng nhập và chọn sản phẩm từ danh sách. | 2. Hệ thống hiển thị trang chi tiết sản phẩm (tên, mô tả, giá, tồn kho, đánh giá). |
| 3. Buyer nhấn nút **“Mua hàng”**. | 4. Hệ thống kiểm tra số lượng tồn kho. |
|  | 5. Nếu còn hàng, hệ thống thêm sản phẩm vào giỏ hàng. |
| 6. Buyer chọn **“Xem giỏ hàng”** hoặc tiếp tục mua sắm. | 7. Hệ thống cập nhật giỏ hàng và hiển thị tổng chi phí, khuyến mãi, phí vận chuyển, tổng thanh toán. |
| 8. Buyer có thể chọn **“Thanh toán”** để bắt đầu quy trình thanh toán. | 9. Hệ thống chuyển sang quy trình thanh toán. |
|  | Use case kết thúc. |

**Alternative Paths**

* **A1:** Nếu Buyer chưa đăng nhập → Hệ thống chuyển hướng sang trang đăng nhập.
* **A2:** Nếu sản phẩm **hết hàng** → Hệ thống thông báo lỗi “Sản phẩm tạm hết hàng” và quay lại trang chi tiết.
* **A3:** Buyer có thể hủy thao tác mua và quay lại trang chi tiết.

**Exception Paths**

* **E1:** Lỗi hệ thống khi cập nhật giỏ hàng → Thông báo lỗi và yêu cầu thử lại.
* **E2:** Lỗi kết nối CSDL tồn kho → Hệ thống hiển thị thông báo “Không thể kiểm tra tồn kho lúc này”.

**Extension Points**

* **Thanh toán (Checkout Process)** → Khi Buyer nhấn nút **Thanh toán**, hệ thống chuyển sang quy trình thanh toán.

**Triggers**

* Buyer nhấn nút **“Mua hàng”** từ trang chi tiết sản phẩm.

**Assumptions**

* Buyer chỉ mua được sản phẩm còn trong kho.

**Preconditions**

* Buyer đã đăng nhập vào hệ thống.
* Trang chi tiết sản phẩm đã được load thành công.

**Postconditions**

* Sản phẩm được thêm thành công vào giỏ hàng.
* Giỏ hàng của Buyer được cập nhật đúng trạng thái.

**References**

* Business Rules: BR1 (tồn kho), BR2 (giỏ hàng), BR3 (thanh toán).

**Activity Diagram**

**A diagram of a flowchart

AI-generated content may be incorrect.**

**Sequence Diagram**

**A diagram with text and images

AI-generated content may be incorrect.**

**Or**

**Tên Use Case:** Mua sản phẩm từ trang chi tiết sản phẩm

**Tác nhân chính:** Người mua (Buyer)

**Mục tiêu:** Người mua có thể chọn và mua sản phẩm trực tiếp từ trang chi tiết sản phẩm để đưa vào giỏ hàng, tiến hành thanh toán.

**Tiền điều kiện:**

* Người mua đã đăng nhập hệ thống.
* Hệ thống hiển thị trang chi tiết sản phẩm với thông tin đầy đủ.

**Luồng chính:**

1. Người mua nhấn nút **“Mua hàng”** tại trang chi tiết sản phẩm.
2. Hệ thống kiểm tra tình trạng tồn kho của sản phẩm.
3. Nếu sản phẩm còn hàng, hệ thống thêm sản phẩm vào giỏ hàng.
4. Hệ thống cập nhật và hiển thị giỏ hàng của người mua (bao gồm sản phẩm vừa thêm và tổng chi phí).
5. Người mua có thể chọn **“Thanh toán”** để tiếp tục quy trình mua hàng.

**Luồng thay thế:**

* 2a. Nếu sản phẩm **hết hàng** → Hệ thống thông báo lỗi và không cho thêm vào giỏ.
* 5a. Người mua không chọn thanh toán → Giỏ hàng vẫn được lưu lại để xử lý sau.

**Hậu điều kiện:**

* Sản phẩm được thêm vào giỏ hàng thành công (nếu còn hàng).
* Giỏ hàng phản ánh chính xác số lượng và giá trị sản phẩm.

**Ghi chú:**

* Có thể có tính năng gợi ý sản phẩm liên quan sau khi thêm vào giỏ.

Câu 6

**Class Diagram cho module Giỏ hàng (Shopping Cart)**.

A screenshot of a diagram

AI-generated content may be incorrect.

**Câu 8. Kiến trúc phần mềm theo mô hình 3 lớp (3-tier architecture)**

1. **Presentation Layer (Tầng giao diện):**
   * Giao diện Web (React, Angular hoặc HTML/CSS/JS)
   * Chức năng: hiển thị danh mục sản phẩm, giỏ hàng, chi tiết sản phẩm, đăng nhập, đánh giá, quản trị.
   * Giao tiếp với Application Layer thông qua API (REST/GraphQL).
2. **Application Layer (Tầng nghiệp vụ):**
   * Các dịch vụ chính:
     + ProductService (quản lý sản phẩm)
     + CartService (giỏ hàng)
     + PaymentService (thanh toán)
     + InventoryService (tồn kho)
     + RatingService (đánh giá)
     + AuthService (đăng nhập, phân quyền)
   * Chứa toàn bộ **logic nghiệp vụ** (business logic).
3. **Data Layer (Tầng dữ liệu):**
   * CSDL quan hệ (MySQL/PostgreSQL) hoặc NoSQL (MongoDB).
   * Repository/Data Access Object (DAO) để truy vấn dữ liệu sản phẩm, người dùng, giỏ hàng, đánh giá.

Ưu điểm của mô hình:

* Tách biệt giao diện, nghiệp vụ, dữ liệu → dễ bảo trì, mở rộng.
* Dễ dàng thay đổi giao diện hoặc database mà không ảnh hưởng tầng khác.
* Bảo mật hơn do tầng Application làm trung gian giữa Client và Data.

**Or**

**Kiến trúc 3-lớp (3-tier) cho hệ thống CoolStore — Chi tiết + Ưu / Nhược điểm**

Dưới đây là thiết kế chi tiết, kèm thành phần công nghệ đề xuất, các quyết định về dữ liệu/đa luồng/độ nhất quán, và cuối cùng là phân tích **ưu — nhược điểm**. Mình giả định hệ thống theo hướng *microservices trong tầng ứng dụng* (tức là vẫn giữ nguyên nguyên tắc 3 lớp: Presentation / Application / Data nhưng business logic được tách nhỏ thành dịch vụ theo miền nghiệp vụ).

**1. Tổng quan (mapping 3-tier)**

* **Presentation Layer (Client / UI):** Web app (React/Vue/Angular) + Mobile apps. Hiển thị danh mục, chi tiết sản phẩm, giỏ hàng, checkout, admin UI.
* **Application Layer (Business / Middle Tier):** API Gateway → nhiều microservice/domain services: ProductService, CartService, Checkout/PaymentService, InventoryService, RatingService, AuthService, AdminService. Chịu trách nhiệm logic nghiệp vụ, orchestration.
* **Data Layer (Persistence):** CSDL quan hệ / NoSQL theo nhu cầu mỗi service, cache (Redis), message broker (Kafka/RabbitMQ) cho giao tiếp bất đồng bộ.

**2. Chi tiết từng tầng**

**A. Presentation Layer**

**Thành phần**

* Single Page App (React) / Mobile (iOS/Android).
* CDN (CloudFront, Akamai) để phục vụ ảnh, JS/CSS.
* Edge cache & HTTP caching (Cache-Control).
* Web Server / Static hosting (S3 + CloudFront).

**Trách nhiệm**

* Hiển thị product list/detail, cart UI, checkout flow, login/logout.
* Gọi REST/GraphQL API qua HTTPS tới API Gateway.
* Lưu session client-side (localStorage) hoặc session cookie (HTTPOnly).

**Bảo mật**

* TLS everywhere.
* Không lưu dữ liệu nhạy cảm (thẻ) trên client.
* CSRF protection (cookie flows), CORS cấu hình đúng.

**B. Application Layer (Middle Tier)**

**Kiến trúc**

* **API Gateway** (nginx/kong/traefik) làm entry point: routing, rate-limit, auth (JWT validation), TLS termination, caching edge.
* **Auth Service / Identity Provider:** OAuth2/OpenID Connect (Keycloak/Auth0) — issue JWT + refresh tokens.
* **Domain Microservices:**
  + ProductService (catalog, search index integration)
  + CartService (persistent cart per user)
  + CheckoutService / PaymentService (orchestrator + payment gateway adapter)
  + InventoryService (stock counts, reservations)
  + RatingService (reviews)
  + AdminService (product management, warehouse assignment)
* **Supporting services:** Email worker (send order confirmations), NotificationService, Monitoring/Logging agents.

**Giao tiếp**

* Sync: REST/GraphQL or gRPC cho low-latency internal calls.
* Async: Message broker (Kafka/RabbitMQ) cho event-driven (order placed → inventory reserved → shipping) và để tách rời bounded contexts.

**Patterns**

* **Idempotency keys** cho checkout/payment requests.
* **Circuit breaker + retry** (Hystrix / Resilience4j).
* **Saga pattern** cho business transactions phân tán (reserve inventory, charge payment, confirm order) — dùng choreography (events) hoặc orchestration (CheckoutService điều phối).
* **Cache** (Redis) cho product read-heavy data và session/cart hot data.

**Scale & HA**

* Các service stateless (hợp lý) → horizontal scaling.
* Stateful components: DB, Redis cluster, Kafka clusters → scale theo tải.

**C. Data Layer**

**Đề xuất lưu trữ theo service**

* ProductService: PostgreSQL + Elasticsearch (search & filter full-text).
* CartService: Redis (fast ephemeral carts) + persistent store (Postgres / DynamoDB) for long-term carts.
* InventoryService: Relational DB (Postgres/MySQL) with strong consistency; hoặc use NewSQL if cần.
* RatingService: MongoDB / Cassandra (document storage) — nếu cần scale nhiều đọc/ghi.
* PaymentService: không lưu CC — dùng payment gateway tokenization (Stripe, Adyen).
* **Shared infra:** Redis cache, CDN, Message broker (Kafka/RabbitMQ), Blob storage (S3) cho hình ảnh.

**Backup / DR**

* Daily snapshot, point-in-time recovery (PITR), cross-region replica nếu cần.

**3. Operational concerns (đề xuất triển khai)**

* **Containerization:** Docker + Kubernetes (K8s) cho orchestrating, autoscaling, rolling update.
* **Ingress & Load Balancer:** nginx/ALB + API Gateway.
* **Config & Secret management:** Vault / Kubernetes Secrets.
* **CI/CD:** Pipeline (GitHub Actions / GitLab CI / Jenkins) với unit → integration → contract tests → canary deploy.
* **Monitoring & Observability:** Prometheus + Grafana, ELK/EFK (Elasticsearch/Fluentd/Kibana) logs, Jaeger/Zipkin tracing.
* **Health checks & Liveness/Readiness probes** cho K8s.
* **Rate limiting / WAF** cho public API.

**4. Bảo mật & tuân thủ**

* **Auth:** OAuth2 + JWT, RBAC cho admin.
* **Payment:** Tuân thủ PCI-DSS — *không lưu* thông tin thẻ; dùng tokenization của provider.
* **Encryption:** TLS in transit; AES-256 at rest for DB sensitive fields.
* **Audit logging** cho hành vi admin/checkout.
* **Input validation & sanitization** (XSS/SQLi prevention).

**5. Ưu điểm (của kiến trúc 3-tier áp dụng với microservices)**

1. **Tách bạch trách nhiệm rõ ràng**: dễ phát triển, bảo trì (UI / Business / Data tách biệt).
2. **Mở rộng linh hoạt:** Tầng middle có thể scale từng service độc lập (theo miền nghiệp vụ).
3. **Tái sử dụng & tích hợp dễ:** API gateway làm điểm entry chung; client không cần biết chi tiết backend.
4. **An toàn hơn:** tách các lớp giúp dễ triển khai chính sách bảo mật (Auth ở trung tâm).
5. **Hỗ trợ CI/CD & deployment độc lập** cho từng service → release nhanh, rollback dễ.
6. **Tối ưu hiệu năng:** caching tại nhiều tầng (CDN + Redis + DB Read replicas).

**6. Nhược điểm và tính hạn chế**

1. **Độ phức tạp vận hành tăng**: nhiều service → cần orchestration (K8s), monitoring, tracing.
2. **Chi phí hạ tầng** (nhiều thành phần: Kafka, Redis cluster, ES, replicas) có thể cao.
3. **Độ nhất quán dữ liệu phức tạp**: transactions phân tán cần Saga/compensation; dễ xảy ra inconsistency tạm thời (eventual consistency).
4. **Phát triển phức tạp hơn**: contract testing / API versioning cần nghiêm ngặt.
5. **Tối ưu và debug khó hơn**: trace một luồng nghiệp vụ qua nhiều microservice cần tracing.
6. **Quá gắt nếu hệ thống nhỏ**: nếu traffic thấp, mô hình này có thể “over-engineering”.

**Lời khuyên triển khai (tối ưu cho CoolStore)**

Bắt đầu: tách **CartService + ProductService + AuthService** trước. Sau đó tách Payment/Inventory khi lượng traffic tăng.

Dùng **Redis** cho cart để đảm bảo tốc độ UX (thêm/xóa sản phẩm phải phản hồi nhanh).

Sử dụng **SAGA choreography** cho checkout nếu muốn ít coupling; dùng orchestration nếu muốn flow dễ kiểm soát.

**Search**: tích hợp Elasticsearch cho filter/ tên/giá sắp xếp.

**Tránh lưu thẻ**: luôn dùng tokenization + PCI compliant gateway.

Áp dụng **feature flags** cho release (LaunchDarkly / Unleash).

Đặt mục observability từ đầu: logs + traces + metrics để dễ dò lỗi.