Thiết kế & Hướng dẫn Triển khai E‑Commerce (Modular Monolith)  
Backend Node.js (JavaScript) + MongoDB + Redis + Socket.IO  
Frontend React (JavaScript) + Material UI (Storefront) & Ant Design (Admin)  
KHÔNG dùng Next.js, KHÔNG dùng TypeScript

# 1) Mục tiêu & Phạm vi

- Kiến trúc: Modular Monolith một tiến trình backend Node.js (JavaScript thuần).

- CSDL: MongoDB (Replica Set 1 node dev để dùng transaction), cache Redis.

- UI: React (JavaScript), Storefront dùng Material UI (MUI), Admin dùng Ant Design.

- Tính năng: Catalog (lọc/sắp xếp/phân trang), chi tiết sản phẩm + biến thể + ảnh, giỏ hàng realtime, checkout + coupon 5 ký tự (≤10 lần), loyalty 10%, review (guest) & rating (login) realtime, lịch sử trạng thái đơn hàng, admin CRUD & dashboard.

- Triển khai: Docker Compose, chỉ cần `docker compose up -d` để chạy toàn hệ thống.

- Không sử dụng: Next.js, TypeScript.

# 2) Kiến trúc tổng quan (Modular Monolith)

- Frontend:

• /apps/web: React + Material UI (storefront).

• /apps/admin: React + Ant Design (backoffice).

- Backend:

• /apps/server: Express hoặc NestJS (viết bằng JavaScript), tổ chức modules.

• Modules chính: auth, user, catalog, product-variants, inventory, cart, order, coupon, loyalty, review, rating, upload, notification(email), dashboard, websocket.

- Nền tảng chung:

• MongoDB cho dữ liệu, Redis cho cache/rate-limit/Socket.IO adapter, Mailhog để test email.

• Socket.IO cho realtime (cart/review/rating).

• Nodemailer để gửi email xác nhận đơn hàng.

# 3) Cấu trúc thư mục đề xuất

/

apps/

web/ # React + Material UI (JS)

admin/ # React + Ant Design (JS)

server/ # Node.js (JS) - modular monolith

src/

modules/

auth/

user/

catalog/

variants/

inventory/

cart/

order/

coupon/

loyalty/

review/

rating/

upload/

dashboard/

websocket/

shared/ # middlewares, validators, utils, error handlers

infra/ # mongo, redis, mailer, s3/local storage, socket.io

package.json

.env.example

docker/

docker-compose.yml

README.md

# 4) Mô hình dữ liệu MongoDB (collections)

users: {

\_id, email, name, passwordHash, role: 'admin'|'customer',

loyaltyPoints: Number, addresses: [{\_id, label, recipient, phone, detail, isDefault}],

createdAt

}

brands: { \_id, name }

categories: { \_id, name, slug, parentId }

products: {

\_id, name, slug, brandId, categoryId,

shortDesc, longDesc, images:[string],

ratingAvg, ratingCount, createdAt

}

product\_variants: { \_id, productId, sku, attributes:{...}, price, compareAtPrice, stock }

carts: { \_id, userId|null, guestId, items:[{ variantId, qty, priceAtAdd }], updatedAt }

Ngắn gọn: **Có** – theo yêu cầu “Guest Checkout”, giỏ hàng phải hoạt động **khi chưa đăng nhập** và khi mở lại trang **vẫn hiện** (nếu còn định danh khách/guest). Cách làm chuẩn:

**Cách thiết kế để đạt yêu cầu**

**1) Lưu giỏ hàng cho khách chưa đăng nhập**

* Server có collection carts như bạn viết:  
  carts: { \_id, userId|null, guestId, items:[{ variantId, qty, priceAtAdd }], updatedAt }
* Khi người dùng chưa login:
  + Tạo (hoặc đọc) **guestId** (UUID) và **lưu vào cookie httpOnly** (ví dụ guest\_id, max-age 30–90 ngày).
  + Giỏ hàng sẽ được **lưu ở server** theo guestId. (Không phụ thuộc session của trình duyệt.)

Vì dùng cookie httpOnly, giỏ hàng vẫn tồn tại sau khi đóng/mở lại trình duyệt trong thời hạn cookie ⇒ quay lại website, **cart hiện ngay** sau khi client gọi /cart kèm cookie.

**2) Hiển thị lại khi mở trang**

* Lần load đầu: frontend gọi GET /cart.
  + Nếu có userId → trả giỏ theo user.
  + Nếu không, có guestId cookie → trả giỏ theo guest.
* (Tùy chọn) Dùng **localStorage** giữ “bản cache UI” để hiện tức thì, rồi **đồng bộ** với server khi API trả về.

**3) Realtime không reload (điểm realtime)**

* Dùng **Socket.IO** channel cart:
  + Khi thêm/sửa/xóa item → server emit cart.updated → UI cập nhật tổng tiền, thuế, phí ship **không reload**.

**4) Merge giỏ khi đăng nhập**

* Khi user đăng nhập:
  1. Lấy cart\_guest theo guestId và cart\_user theo userId.
  2. **Gộp**: cộng dồn số lượng theo variantId.
  3. Lưu vào cart\_user, **xóa** cart\_guest (hoặc đánh dấu merged).
  4. Xoá/đặt lại guestId cookie để tránh lẫn lộn.

Luôn **recalculate giá** (theo priceAtAdd chỉ là tham chiếu hiển thị; giá thanh toán lấy **giá hiện tại** và kiểm tra tồn kho).

**5) API tối thiểu**

* GET /cart → trả giỏ theo user hoặc guest.
* POST /cart/items → { variantId, qty } (tự tạo cart nếu chưa có).
* PATCH /cart/items/:id → cập nhật qty.
* DELETE /cart/items/:id → xóa item.
* (Server) emit cart.updated sau mỗi thao tác.

**6) Checkout cho guest (đáp ứng yêu cầu)**

* Guest có thể vào luồng /checkout bình thường:
  + POST /checkout/apply-coupon → kiểm tra **mã 5 ký tự**, **không hết hạn**, **tối đa 10 lần dùng**, và **trả hiệu lực + tác động giá trước khi thanh toán**.
  + POST /checkout/confirm:
    - Kiểm tồn & trừ kho (transaction).
    - Tạo order (có thể gắn guestEmail).
    - (Tùy chọn) Nếu email này chưa có tài khoản → tạo tài khoản & gửi mail đặt mật khẩu.
* Sau khi thanh toán xong, giỏ hàng khách được **xóa** (hoặc làm rỗng) để tránh double-order.

**7) Các lưu ý quan trọng**

* **Cookie guestId**: httpOnly + Secure + SameSite=Lax; gia hạn mỗi lần hoạt động (sliding expiration).
* **LocalStorage** (nếu dùng): chỉ là cache UI; luôn đồng bộ từ server để tránh lệch.
* **Tính giá**: tổng tiền, thuế, phí ship nên tính ở server mỗi lần /cart và khi apply coupon.
* **Giảm giá**: validate regex ^[A-Z0-9]{5}$, usedCount < maxUse (≤10), hiển thị lợi ích **trước** confirm.
* **Idempotency** cho /checkout/confirm để tránh trùng đơn khi người dùng F5.
* **SEO/Analytics**: ghi event khi thêm giỏ để dùng dashboard.

**Tóm lại**

* **Có lưu giỏ khi chưa đăng nhập**: dùng guestId cookie + lưu server-side.
* **Mở lại vẫn thấy giỏ**: miễn cookie guestId còn hiệu lực, gọi GET /cart sẽ trả đúng giỏ.
* Khi đăng nhập, **merge** giỏ guest → user.
* Tất cả cập nhật cart phải **realtime** (Socket.IO) để đáp ứng tiêu chí “không reload”.

orders: {

\_id, code, userId|null, guestEmail,

items:[{ variantId, qty, unitPrice }],

subtotal, discount, loyaltyUsed, total,

status:'pending'|'confirmed'|'shipping'|'delivered',

history:[{status, at}], createdAt

}

coupons: { \_id, code(5), valueType:'percent'|'amount', value, usedCount, maxUse(<=10) }

product\_reviews: { \_id, productId, userId|null, guestName, content, createdAt }

product\_ratings: { \_id, productId, userId, stars(1..5), createdAt }

analytics\_events: { \_id, type, payload, createdAt }

# 5) Indexes & Query tối ưu

- products: text index {name, longDesc} (default\_language:'none'); {categoryId:1, brandId:1, createdAt:-1}

- product\_variants: {productId:1}, {sku:1}

- orders: {userId:1, createdAt:-1}, {code:1}

- coupons: {code:1} unique

- product\_ratings: unique compound {productId:1, userId:1}

# 6) Thiết kế API (REST, JSON)

Auth:

- POST /auth/register

- POST /auth/login

- GET /auth/profile

- POST /auth/password/recover

Catalog:

- GET /products?q=&brandId=&categoryId=&priceGte=&priceLte=&ratingGte=&sort=name\_asc|name\_desc|price\_asc|price\_desc&page=&limit=

- GET /products/:slug # trả product, variants, images, mô tả (>=5 dòng)

Cart (realtime cập nhật UI):

- GET /cart - theo user hoặc guestId cookie

- POST /cart/items

- PATCH /cart/items/:id

- DELETE /cart/items/:id

Checkout/Order:

- POST /checkout/apply-coupon # trả hiệu lực + giá sau giảm (trước khi confirm)

- POST /checkout/confirm # tạo đơn, trừ kho, email, điểm thưởng

- GET /orders/mine

- GET /orders/:code # kèm history

Review & Rating (realtime):

- POST /reviews # cho phép guest

- POST /ratings # yêu cầu login

Admin:

- Products: CRUD + biến thể + ảnh + tồn kho

- Users: danh sách, ban/unban

- Orders: list 20/trang (mới nhất trước), cập nhật trạng thái

- Coupons: list + tạo mới (5 ký tự, tối đa 10 lần dùng)

- Dashboard: KPIs + biểu đồ theo khoảng thời gian

# 7) Luồng Checkout (MongoDB Transaction)

1) Bật MongoDB Replica Set (kể cả 1 node dev).

2) Bắt đầu session + transaction.

3) Validate coupon: regex ^[A-Z0-9]{5}$, usedCount < maxUse (<=10); nếu hợp lệ -> $inc usedCount.

4) Trừ tồn từng biến thể: findOneAndUpdate({\_id: vId, stock: {$gte: qty}}, {$inc: {stock: -qty}}, {session}). Nếu fail -> abort.

5) Tính tiền: subtotal -> áp coupon -> trừ loyaltyUsed (nếu dùng) -> total.

6) Tạo orders + push history {status:'pending', at:now}.

7) Nếu user: cộng loyaltyEarned = round(total \* 0.1) vào users.loyaltyPoints.

8) Commit; đẩy sự kiện realtime (Socket.IO) + enqueue email xác nhận.

9) Idempotency: lưu Idempotency-Key để tránh tạo đơn trùng khi refresh.

# 8) Quy tắc nghiệp vụ (để đạt điểm tối đa)

- Guest checkout: không cần đăng nhập; sau khi mua tạo account bằng email nếu chưa tồn tại.

- Coupon: 5 ký tự chữ-số, không hết hạn, maxUse <= 10; hiển thị mức giảm \*\*trước\*\* confirm.

- Loyalty: tích 10% giá trị đơn (sau giảm), dùng ngay đơn kế tiếp.

- Rating: bắt buộc đăng nhập; Review: cho phép guest.

- Order history: lưu và hiển thị timeline mới nhất trước.

- Pagination: backend trả totalPages; frontend \*\*luôn render\*\* phân trang kể cả totalPages=1.

- Sắp xếp: name A→Z, Z→A; price ↑, ↓ (tối thiểu 4).

# 9) Frontend Storefront (React + Material UI, JavaScript)

- Dựng bằng Vite: `npm create vite@latest web -- --template react` (JS).

- Thư viện: @mui/material, @mui/icons-material, @tanstack/react-query, react-router-dom, react-hook-form, axios, socket.io-client.

- Trang chính: Landing (New/Best Sellers).

- Catalog: Grid + Card; Filters (Checkbox brand, Slider price, Rating); Sort (Select); Pagination (Pagination component) — luôn hiển thị.

- Product Detail: ImageList (>=3 ảnh), ToggleButtonGroup chọn biến thể, mô tả >=5 dòng, Review & Rating realtime.

- Cart: Drawer + Badge; đồng bộ tổng giá tức thì.

- Checkout: Stepper (địa chỉ → mã giảm → xác nhận), hiển thị “Bạn tiết kiệm …đ”.

- State server: React Query (query + mutation, cache, optimistic update).

# 10) Frontend Admin (React + Ant Design, JavaScript)

- Dựng bằng Vite: `npm create vite@latest admin -- --template react` (JS).

- Thư viện: antd, @tanstack/react-query, react-router-dom, recharts, axios.

- Dashboard: DateRange + Card KPIs + Table top sản phẩm + biểu đồ.

- Orders: Table 20/trang, sort “mới nhất trước”, filter theo preset thời gian; trang chi tiết cập nhật trạng thái -> ghi history.

- Products: Form + Upload ảnh, Table biến thể (editable) + tồn kho.

- Coupons: Form tạo mã 5 ký tự, Table xem used/max + đơn đã dùng.

- Users: Table + ban/unban.

# 11) Backend (Express hoặc NestJS, JavaScript)

- Lựa chọn 1 (Express): express + mongoose + jsonwebtoken + joi/celebrate (validate) + socket.io + nodemailer + ioredis.

- Lựa chọn 2 (NestJS JS): nest new --language js; @nestjs/mongoose, @nestjs/jwt, class-validator, socket.io, @nestjs/axios.

- Auth: JWT ngắn hạn + refresh; lưu refresh token (whitelist) theo user.

- Bảo mật: rate limit, CORS, helmet, sanitize.

- Logging: morgan hoặc pino; error handler chuẩn JSON.

- Realtime: Namespace `reviews`, `ratings`, `cart`; events: review.created, rating.created, cart.updated.

- Upload: lưu local (dev) hoặc S3-compatible (prod).

# 12) Docker Compose (dev & demo)

version: "3.9"

services:

mongo:

image: bitnami/mongodb:7.0

environment:

- MONGODB\_REPLICA\_SET\_MODE=primary

- MONGODB\_REPLICA\_SET\_KEY=rskey

- MONGODB\_ROOT\_USER=root

- MONGODB\_ROOT\_PASSWORD=pass

- MONGODB\_REPLICA\_SET\_NAME=rs0

ports: ["27017:27017"]

volumes: ["mongo:/bitnami/mongodb"]

redis:

image: redis:7

ports: ["6379:6379"]

mailhog:

image: mailhog/mailhog

ports: ["8025:8025"]

server:

build: ./apps/server

env\_file: ./apps/server/.env

depends\_on: [mongo, redis, mailhog]

ports: ["4000:4000"]

web:

build: ./apps/web

env\_file: ./apps/web/.env

depends\_on: [server]

ports: ["3000:3000"]

admin:

build: ./apps/admin

env\_file: ./apps/admin/.env

depends\_on: [server]

ports: ["3001:3001"]

volumes:

mongo:

# 13) Biến môi trường (.env mẫu)

# apps/server/.env

PORT=4000

MONGO\_URI=mongodb://root:pass@mongo:27017/app?authSource=admin&replicaSet=rs0

JWT\_SECRET=supersecret

JWT\_EXPIRES=900

REFRESH\_EXPIRES=604800

REDIS\_URL=redis://redis:6379

MAIL\_HOST=mailhog

MAIL\_PORT=1025

CLIENT\_URL=http://localhost:3000

ADMIN\_URL=http://localhost:3001

# apps/web/.env

VITE\_API\_BASE=http://localhost:4000

# apps/admin/.env

VITE\_API\_BASE=http://localhost:4000

# 14) Kiểm thử & Chất lượng

- Unit/Integration: Jest + mongodb-memory-server hoặc Testcontainers.

- E2E: Playwright/Cypress (luồng guest checkout, coupon 5 ký tự, loyalty 10%, rating yêu cầu login).

- Lint/Format: ESLint + Prettier.

- Hiệu năng: cache danh sách sản phẩm (Redis, TTL ngắn), chỉ select field cần thiết (projection).

# 15) Scale ngang & Realtime

- Backend stateless (JWT), giỏ hàng theo user/guest trong DB/Redis.

- Chạy nhiều replica `server` sau Nginx/Traefik (round-robin).

- Socket.IO dùng redis-adapter để broadcast trên nhiều node.

- Chứng minh bằng demo log mỗi container nhận request, và nhiều browser vẫn nhận được sự kiện realtime.

# 16) Hướng dẫn chạy nhanh

1) Cài Docker & Node 20+.

2) Viết file Dockerfile cho /apps/web, /apps/admin, /apps/server (có `npm ci` và `npm run build`/`npm start`).

3) `docker compose up -d` trong thư mục /docker (hoặc gốc nếu đặt compose ở đó).

4) Truy cập: Storefront http://localhost:3000, Admin http://localhost:3001, API http://localhost:4000, Mailhog http://localhost:8025.

5) Đăng nhập admin (seed script tạo tài khoản) để vào backoffice.

# 17) Checklist bàn giao & nộp bài

- Source code đầy đủ.

- Video demo: toàn bộ tính năng (lọc/sort/pagination; chi tiết & biến thể; cart realtime; checkout + coupon + loyalty; review/rating realtime; admin CRUD + dashboard).

- Ảnh GitHub Insights ≥ 1 tháng, mỗi thành viên ≥ 2 commit/tuần.

- README: hướng dẫn chạy, tài khoản demo, ghi rõ Bonus (nếu có).

- Docker Compose: chạy được end-to-end.