TRƯỜNG ĐẠI HỌC CẦN THƠ

**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

****

**NIÊN LUẬN CƠ SỞ**

**MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG DỮ LIỆU**

**ĐỀ TÀI**

**PHÁT TRIỂN ỨNG DỤNG WEB BÁN LAP TOP**

**Sinh viên thực hiện: Ngô Gia Hạo**

**MSSV: B2110964**

**Khóa: 47**

***Cần Thơ, 4/2025***

# **LỜI CẢM ƠN**

Cần Thơ, ngày ... tháng ... năm 20...

Sinh viên thực hiện

# 

# **NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

**..............................................................................................................................**

Cần Thơ, ngày ... tháng ... năm 20...

Giảng viên hướng dẫn

Mục lục

[LỜI CẢM ƠN 2](#__RefHeading___Toc289_853921268)

[NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN 3](#__RefHeading___Toc291_853921268)

[CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI 7](#__RefHeading___Toc42120_2616427129)

[!. ĐẶT VẤN ĐỀ 7](#__RefHeading___Toc42122_2616427129)

[2. MỤC TIÊU CỦA ĐỀ TÀI 8](#__RefHeading___Toc42124_2616427129)

[2.1 Mục tiêu chính 8](#__RefHeading___Toc753_853921268)

[2.2 Mục tiêu kỹ thuật 8](#__RefHeading___Toc42126_2616427129)

[2.3 Mục tiêu học tập 8](#__RefHeading___Toc42128_2616427129)

[3. PHẠM VI CỦA ĐỀ TÀI 9](#__RefHeading___Toc42130_2616427129)

[3.1 Những chức năng sẽ triển khai 9](#__RefHeading___Toc1226_853921268)

[3.1.1 Dành cho người dùng: 9](#__RefHeading___Toc42132_2616427129)

[3.1.2 Dành cho quản trị viên: 9](#__RefHeading___Toc42134_2616427129)

[4. CÁC CÔNG NGHÊ SỬ DỤNG 9](#__RefHeading___Toc959_2901295804)

[4.1 Công nghệ cho Backend. 10](#__RefHeading___Toc42136_2616427129)

[4.1.1 Framework và Runtime: 10](#__RefHeading___Toc42138_2616427129)

[4.1.2 Database và ORM: 10](#__RefHeading___Toc42140_2616427129)

[4.1.3 Bảo mật và xác thực: 10](#__RefHeading___Toc42142_2616427129)

[4.2 Công nghệ cho Frontend. 10](#__RefHeading___Toc42144_2616427129)

[4.2.1. Framework: 10](#__RefHeading___Toc42146_2616427129)

[4.2.2. Styling và UI Components: 10](#__RefHeading___Toc42148_2616427129)

[4.2.3. Quản lý State: 11](#__RefHeading___Toc42150_2616427129)

[4.2.4. Routing và Navigation: 11](#__RefHeading___Toc42152_2616427129)

[4.3. Workspace cho SharedType. 11](#__RefHeading___Toc42154_2616427129)

[4.4. Lý do lựa chọn công nghệ. 11](#__RefHeading___Toc42156_2616427129)

[4.4.1. NestJS đươc chọn vì: 11](#__RefHeading___Toc42158_2616427129)

[5 BỐ CỤC NIÊN LUẬN 12](#__RefHeading___Toc311_853921268)

[CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG 13](#__RefHeading___Toc315_853921268)

[1. PHÂN TÍCH YÊU CẦU 13](#__RefHeading___Toc961_2901295804)

[1.1. Về yêu cầu chức năng: 13](#__RefHeading___Toc42160_2616427129)

[1.1.1. Chức năng dành cho người dùng thông thường: 13](#__RefHeading___Toc42162_2616427129)

[1.1.2. Chức năng dánh cho quản trị viên (Admin): 14](#__RefHeading___Toc42164_2616427129)

[1.2. Về yêu cầu phi chức năng 14](#__RefHeading___Toc42166_2616427129)

[2. THIẾT KẾ KIẾN TRÚC HỆ THỐNG. 17](#__RefHeading___Toc963_2901295804)

[2.1. Kiến trúc tổng quan. 17](#__RefHeading___Toc965_2901295804)

[2.2. Kiến trúc Backend (NestJS) 18](#__RefHeading___Toc42168_2616427129)

[3. THIẾT KẾ CƠ SỞ DỮ LIỆU 22](#__RefHeading___Toc858_3857496495)

[3.1. Mô tả các thực thể chính. 22](#__RefHeading___Toc42170_2616427129)

[3.2. Mối quan hệ giữa các thực thể. 22](#__RefHeading___Toc42172_2616427129)

[3.3. Mô tả chi tiết các thực thể 24](#__RefHeading___Toc42174_2616427129)

[CHƯƠNG 3: TRIỂN KHAI BACKEND (NESTJS) 28](#__RefHeading___Toc42176_2616427129)

[1. TỔNG QUAN VỀ NESTJS 28](#__RefHeading___Toc42178_2616427129)

[1.1. Cấu trúc thư mục 29](#__RefHeading___Toc42180_2616427129)

[1.2. Sơ đồ mối quan hệ giữa các module 30](#__RefHeading___Toc42182_2616427129)

[2. THIẾT KẾ API REST 32](#__RefHeading___Toc42184_2616427129)

[2.1. Nguyên tắc thiết kế của REST API 32](#__RefHeading___Toc42186_2616427129)

[2.2. Cấu trúc API Endpoints của Backend 32](#__RefHeading___Toc42188_2616427129)

[2.3. Xác thực và phân quyền. 35](#__RefHeading___Toc42190_2616427129)

[2.4. Thiết kế quy tắc Request/Response. 37](#__RefHeading___Toc42192_2616427129)

[2.4.1. Validation Dtos 37](#__RefHeading___Toc42194_2616427129)

[2.4.2. Thiết kế chuẩn phân trang. 37](#__RefHeading___Toc42196_2616427129)

**TÓM TẮT**

Đề tài: Xây dựng một ứng dụng web thương mại điện tử chuyên bán laptop, cung cấp trải nghiệm mua sắm thuận tiện, tích hợp tình năng báo mật (JWT) và có tính mở rộng cao.

# 

# 

# **CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI**

# **!.** **ĐẶT VẤN ĐỀ**

Trong thời đại công nghệ 4.0, nhu cầu sử dụng ngày càng cao, đặc biệt sau giai đoạn đại dịch COVID-19 khi xu hướng làm việc từ xa và học trực tuyến ngày càng trở nên phổ biến**.** Tuy nhiên việc mua sắm, vẫn gặp nhiều khó khăn.

Dưới đây là phần trình bày các vấn đề:

**Hạn chế của các cửa hàng truyền thống**: Các của hàng bán lẻ laptop và thiết bị công nghệ truyền thống có giờ mở cửa cố định trong ngày, diện tích trưng bày không đủ để hiển thị đầy đủ sản phẩm và thông số kỹ thuật chi tiết. Do đó không thể phục vụ số lượng lớn khách hàng khi có nhu cầu mua sắm ở một thời điểm tăng cao, dẫn đến mất doanh thu và khách hàng sẽ chuyển qua các cữa hàng khác với khả năng phục vụ tốt hơn.

Vấn đề thứ hai là **thiếu thông tin so sánh,** khách hàng khó so sánh thông số kỹ thuật, giá cả giữa các dòng laptop và thiết bị công nghệ khác nhau một cách trực quan và hiệu quả.

Thứba là **quy trình mua hàng phức tạp,** việc tìm hiểu, tư vấn và mua sắm các sản phẩm công nghệ thường mất nhiều thời gian, đặc biệt khi cần tư vấn chuyên sâu về cấu hình phù hợp. Thứ tư là **nhu cầu mua sắm online tăng cao,** sau đại dịch COVID-19 xu hướng mua sắm các sản phẩm online tăng cao nhưng nhiều doanh nghiệp vừa và nhỏ chưa có nền tảng bán hàng online chuyên nghiệp. Việc xây dựng nhanh một ứng dụng bán hàng online với chi phí thấp và có tính mở rộng cao nếu tình hình kinh doanh thuận lợi và ứng dụng cần thêm, cải thiện dịch vụ là một bước đi sáng suốt.

## **2.** **MỤC TIÊU CỦA ĐỀ TÀI**

### 2.1 Mục tiêu chính

Xây dựng một sản phẩm thực tế web thương mại điện tử với đầy đủ các chức năng cơ bản của một ứng dụng web hiện đại, bao gồm:

* Hệ thống xác thực và phân quyền với JWT tokens.
* Quản lý sản phẩm và danh mục với khả năng CRUD đầy đủ.
* Hệ thống giỏ hàng cho phép thêm, sửa, xóa sản phẩm.
* Quy trình đặt hàng từ giỏ hàng đến thanh toán.
* Tích hợp thanh toán với cổng thanh toán SePay QR Code.
* Admin panel để quản lý toàn bộ hệ thống.
* Dashboard thống kê với các chỉ số quan trọng.

### **2.2 Mục tiêu kỹ thuật**

* + **Backend Development:**
  + Thiết kế và triển khai API với chuẩn RESTful sử dụng NestJS.
  + Áp dụng kiến trúc modular.
  + Implement authentication và authorization với JWT strategy
  + Thiết kế database với PostgreSQL và TypeORM.
  + Sử dụng Validation Pipe cho Data Transfer Object (DTO) và error handling.
  + **Frontend Development:**
  + Xây dựng Single Page Application (SPA) với React và TypeScript.
  + Implement responsive design với TailwindCSS và ShadcnUI components.
  + Quản lý state với Zustand theo pattern phân chia theo domain.
  + Tích hợp API với error handling và loading states.

### 2.3 Mục tiêu học tập

* + - Thực hành nguyên lý thiết kế phần mềm SOLID.
    - Hiểu về full-stask development workflow.
    - Học cách tích hợp các dịch vụ bên thứ ba (payment gateway, email service).
    - Học cách thiết kế kiến trúc hệ thống dễ dàng maintain và mở rộng.
    - Cũng cố kiến thức về NestJS Framework, ReactJS.
    - Phát triển kỹ năng debugging.

## **3. PHẠM VI CỦA ĐỀ TÀI**

Nhằm đảm bảo có một sản phẩm có thể áp dụng vào thực tế, dưới đây là một số tính năng cốt lõi sẽ triển khai mà một ứng dụng bán hàng cần có

### **3.1 Những chức năng sẽ triển khai**

#### **3.1.1** Dành cho người dùng:

* Đăng ký đăng nhập với email và mật khẩu.
* Quản lý thông tin cá nhân (profile, địa chỉ, số diện thoại).
* Duyệt và tím kiếm sản phẩm với bộ lọc.
* Viết đánh giá sản phẩm sau khi mua hàng.
* Xem đánh giá sản phẩm.
* Xem chi tiết sản phẩm với hình ảnh và thông tin đây đủ.
* Thêm sản phẩm vào giỏ hàng và quản lý giỏ hàng.
* Đặt hàng với thông tin giao hàng.
* Thanh toán qua Sepay QR Code hoặc thanh toán khi nhận hàng (COD).
* Theo dõi trạng thái đơn hàng và lịch sử các đơn hàng.

#### **3.1.2** Dành cho quản trị viên:

* Dashboard thống kê tổng quan (tổng người dùng, đơn hàng, doanh thu, sản phẩm).
* Quản lý người dùng (view, tìm kiếm người dùng theo tên hoặc email, khóa/mở tài khoảng).
* Quản lý sản phẩm (CRUD, tìm kiếm sản phẩm).
* Quản lý danh mục sản phẩm (CRUD, tìm kiếm danh mục).
* Quản lý đơn hàng (view, tìm kiếm, cập nhật trạng thái đơn hàng).
* Xem chi tiết đơn hàng với thông tin giao hàng.

## **4.** CÁC CÔNG NGHÊ SỬ DỤNG

Phần này sẽ liệt kê các cộng nghệ được sử dụng kèm theo thông tin version tại thời điểm đang làm đồ án.

### 4.1 Công nghệ cho Backend.

#### **4.1.1 Framework và Runtime:**

* NestJS 11.0.7: Framework Backend của NodeJS phù hợp cho việc xây dựng API dễ bảo trì và mở rộng.
* NodeJS 22.15.0: Môi trường chạy JavaScript runtime.
* TypeScript 5.7.3: Ngôn ngữ mở rộng của JavaScript, hỗ trợ kiểm tra kiểu tĩnh, giúp phát hiện lỗi lúc biên dịch, cải thiện khả năng debug.

#### **4.1.2 Database và ORM:**

* PostgreSQL 14.18: Hệ thống quản lý cơ sở dữ liệu quan hệ**.**
* TypeORM 0.3.24: Thư viện TypeScript hỗ trợ tương tác với Database dễ dàng**.**

#### **4.1.3 Bảo mật và xác thực:**

* JWT (JSON Web Token)**.**
* Passport.js với JET Strategy: dùng để xây dưng middleware cho logic xác thực request gửi lên Backend**.**
* Thư viện bcrypt 6.0.0: dùng để hashing mật khẩu**.**
* Class-validator và Class-transformer: 2 thư viện dùng để validation các input đâu vào và chuyển đổi kiểu dữ liệu các trường**.**

**4.1.4. Dịch vụ bên thứ 3:**

* SePaSePay API: Dịch vụ API dành cho tính năng thanh toán.
* Mailtrap API: Dành cho tính năng đăng ký tài khoảng, sẽ gửi mail chào mừng khi đăng ký thành công. Gói miễn phí chỉ cho phép gửi nhận mail trong sandbox.
* Axios 1.9.0: Thư viện hỗ trợ gọi các API bên ngoài ứng dụng.

### **4.2 Công nghệ cho Frontend.**

#### **4.2.1.** Framework:

* React 19: Thư viện làm UI phổ biến nhất với kiến trúc component.
* TypeScript: Ngôn ngữ mở rộng của Javasript.
* Vite 6.3.5: Cộng cụ xậy dựng base ứng dụng React nhanh.

#### **4.2.2.** Styling và UI Components:

* TailwindCSS 4.1.7: CSS framework nhẹ, không cần viết style riêng như CSS thuần.
* ShadcnUi: Thư viện components được build sẵn.
* Lucide React 0.511.0: Thư viện icon.

#### **4.2.3.** Quản lý State:

* Zustand 5.0.5: Thư viện quản lý state cho ứng dụng React**.**
* React Hook Form: 7.56.4: Thư viện tối ưu hiệu suất cho form kèm validation**.**
* Zod 3.25.28: Thư viện tạo validation schema cho form**.**

#### **4.2.4. Routing và Navigation:**

* React Router Dom 7.56.4: Thư viện định nghĩa routing cho ứng dụng React.

### 4.3. Workspace cho SharedType.

* Monorepo: công cụ quản lý nhiều app trong một repository duy nhất, giúp chia sẽ tài nguyên giữa các app sử dụng các cộng nghệ khác nhau**.**
* PNPM >= 8.0.0: Trính quán lí gói nhanh hơn NPM mặc định.
* Shared Types Package: Package định nghĩa type chung cho Frontend và Backend, tránh boilerplate type.

### 4.4. Lý do lựa chọn công nghệ.

#### **4.4.1. NestJS đươc chọn vì:**

* Dễ học nếu đã biết NodeJS kèm Express.
* Cú pháp dễ đọc
* Sử dụng cùng ngôn ngữ lập trình với Frontend giúp giảm thiểu thời gian xây dựng ứng dụng fullstask vì không cần phải học 2 ngôn ngữ riêng biệt.
* Kiến trúc modular tự nhiên, dễ scale và maintain.
* Dependency injection, giảm thiểu sự phụ thuộc giữa các class, tối ưu hiệu suất do chỉ cần tạo một instance rồi sử dụng ở nhiều nơi và quản lý ở một nơi (class định nghĩa service), giúp tách biệt sự quan tâm.
* Hệ sinh thái hỗ trợ các middleware được xây dựng sẵn cho bảo mật như Guards, Interceptors và Pipes.

#### 4.4.2. React với TypeScript vì:

* Nhờ kiến trúc component-based giúp tái sử dụng code hiệu quả.
* Sử dụng Virture DOM và thuật toán đối chiếu giúp tối ưu hiệu suất hơn khi thao tác với DOM thật.
* Hệ sinh thái lớn, có nhiều công cụ hỗ trợ.
* TypeScript giúp debug dễ dàng hơn.

#### 4.4.3. Quản lý state với Zustand vì:

* Zustand nhẹ và dễ học hơn Redux
* Không cần setup context rườm rà, không cần phải wrapper app component, dùng càng nhiều context thì càng phải bọc nhiều, nhìn càng rối.
* Với Zustand chúng ta chỉ cần định nghĩa stores và có thể gọi state, funtion được viết trong store ở mọi nơi trong pham vi ứng dụng.

#### 4.4.4. PostgresSQL vì:

* Hỗ trợ kiểu dữ liệu quan hệ chặc chẽ.
* Phù hợp làm ứng dụng thương mại có tính năng giao dịch do tuân thủ ACID.
* Mãnh mẽ, dễ thao tác khi dùng với TypeORM.

## **5** BỐ CỤC NIÊN LUẬN

Bố cục niên luận được chia thành 6 chương chính, mỗi chương sẽ tập trung vào một phía cạnh cụ thể của quá trình phát triển.

**Chương 1 - Tổng quan đề tài:** Chương này sẽ giới thiệu bối cảnh, lý do chọn đề tài, mục tiêu và pham vi công nghệ sử dụng.

**Chương 2 - Phân tích và thiết kế hệ thống:** Chương này sẽ phân tích các yêu cầu chi tiết, thiết kế kiến trúc tổng thể, thiết kế database schema với sơ đồ ERD và lập kế hoạch cho việc thực hiện.

**Chương 3 - Triển khai Backend (NestJS):** Chương này sẽ đi vào phát triển Backend, giải thích cách triển khai từng module, luồng xác thực, thiết kế API, tích hợp database và logic nghiệp vụ.

**Chương 4 - Triển khai Frontend (React):** Chương này sẽ đi vào xây dựng user interface, triển khai quản lý state, kiến trúc component, tích hợp API, tối ưu hóa.

**Chương 5 - Các tính năng nâng cao:** Chương này sẽ trình bày về tích hợp các dịch vụ bên thứ 3.

**Chương 6 - Kết luận và hướng phát triển:** Tổng kết kết quả đạt được, bài học kình nghiệm, những thách thức và lộ trình cải tiến trong tương lai.

Bố cục này được thiết kế để người đọc có thể hiểu được toàn bộ quá trình phát triển từ ý tưởng ban đầu đến sản phẩm hoàn thiện.

# 

# **CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG**

## **1. PHÂN TÍCH YÊU CẦU**

Hệ thống Web được thiết kế để phục vụ cho 2 người người dùng chính, và một người dùng phụ. Người dùng cao hơn sẽ kế thừa các tính năng của người dùng thấp hơn.

### **1.1. Về yêu cầu chức năng:**

#### **1.1.1. Chức năng dành cho người dùng thông thường:**

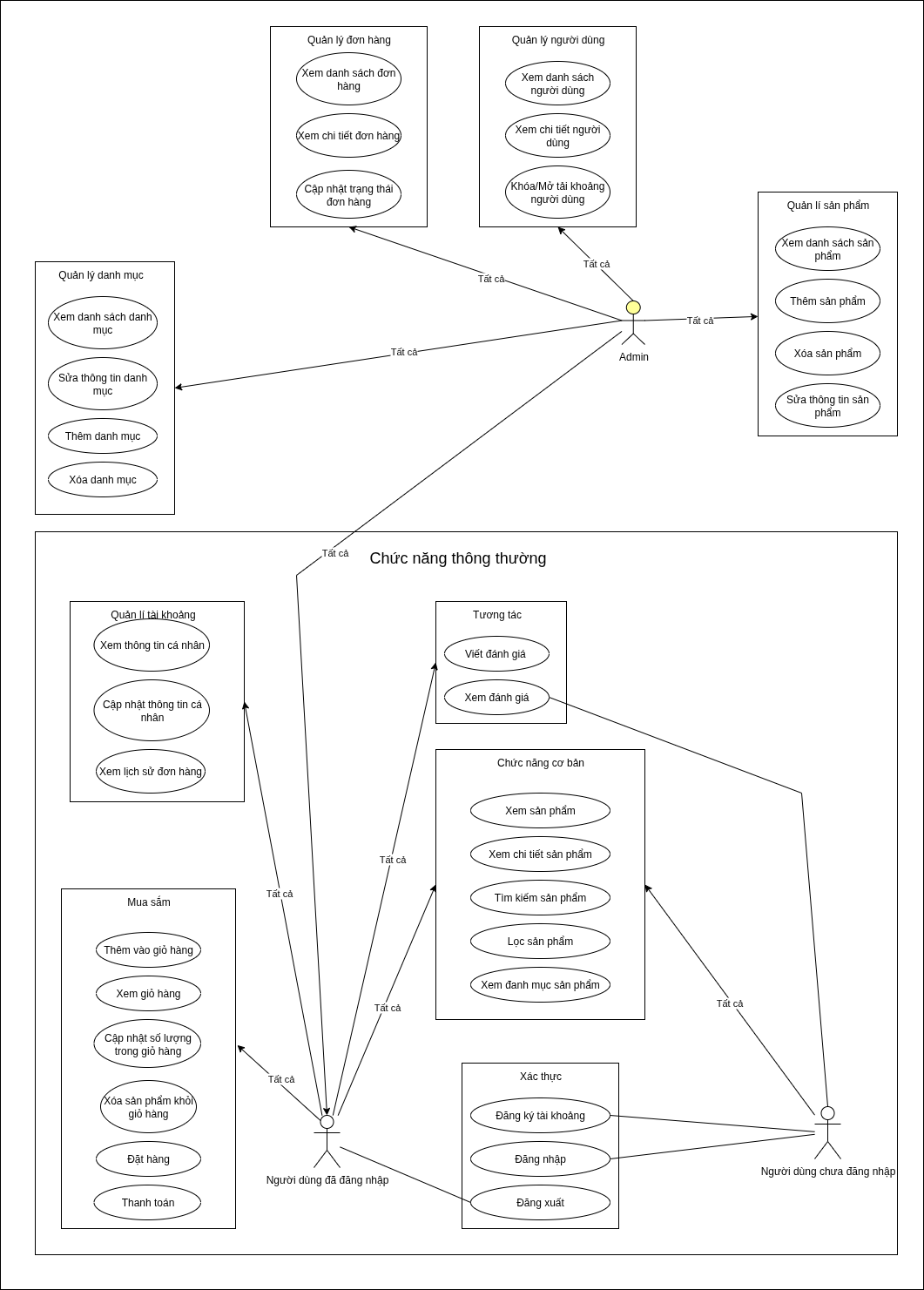
* **Quản lý tài khoảng cả nhân:**
  + Đăng ký tài khoảng mới với email và username duy nhất.
  + Đăng nhập/Đăng xuất sử dụng JWT cho xác thực.
  + Quản lý thông tin cá nhân: cập nhật địa chỉ, số điện thoại, username.
* **Duyệt và tìm kiếm sản phẩm:**
  + Xem danh sách sản phẩm với phân trang.
  + Lọc sản phẩm theo danh mục, khoảng giá tiền.
  + Tìm kiếm sản phẩm theo tên.
  + Xem chi tiết sản phẩm bao gồm mô tả, giá, bình luận, số lượng sản phẩm còn lại.
* **Quản lý giỏ hàng:**
  + Thêm sản phẩm vào giỏ hàng với số lượng tùy chọn (giới hạn bởi số lượng sản phẩm còn trong kho).
  + Cập nhật số lượng hoặc xóa sản phẩm khỏi giỏ hàng (Không thể xóa sản phẩm nếu đang có đơn hàng).
  + Xem tổng quan giỏ hàng kèm theo tổng tiền phải thanh toán.
* **Đặt hàng và thanh toán:**
  + Tạo đơn hàng từ giỏ hàng
  + Nhập thông tin giao hàng như địa chi, số điện thoại, ghi chú thêm.
  + Thanh toán qua SePay QR code hoặc thanh toán khi nhận hàng (COD).
  + Theo dõi trạng thái đơn hàng và thanh toán.
* **Lịch sử mua hàng:**
  + Xem danh sách đơn hàng đã đạt, kém trạng thái đơn hàng.

#### **1.1.2. Chức năng dánh cho quản trị viên (Admin):**

* **Dashboard tổng quan:** 
  + Xem thống kê tổng quan: tổng số người dùng, sản phẩm, đơn hàng.
  + Doanh thu theo thời gian.
  + Báo cáo số người dùng mới, đơn hàng mới, đơn hàng theo trạng thái.
* **Quản lí người dùng:** 
  + Xem danh sách người dùng với phân trang.
  + Tìm kiếm người dùng theo email, username.
  + Xem thông tìn người dùng.
  + Khóa người dùng.
* **Quản lý sản phẩm:** 
  + Thêm sản phẩm với đầy đủ thông tin.
  + Cập nhật thông tin sản phẩm (tên, giá, mô tả, hình ảnh, số lượng, danh mục).
  + Xóa mềm sản phẩm (soft delete).
  + Quản lí trạng thái hoạt động của sản phẩm.
* **Quản lý danh mục:**
  + CRUD danh mục sản phẩm.
  + Phân loại sản phẩm theo danh mục.
* **Quản lý đơn hàng:** 
  + Xem danh sách tất cả đơn hàng với khả năng lọc theo trạng thái.
  + Cập nhật trạng thái đơn hàng (Pending → Processing → Shipped → Delivered).
  + Xem chi tiết đơn hàng bao gồm thông tin giao hàng và sản phẩm.

### **1.2. Về yêu cầu phi chức năng**

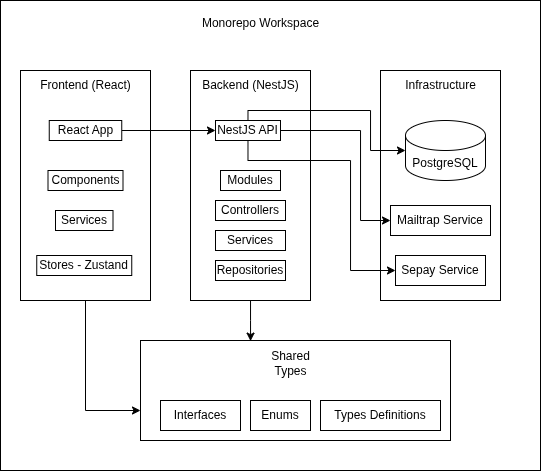
* **Bảo mật:**
  + Mã hóa mật khảu bằng bcrypt.
  + Xác thực dựa trên JWT dùng access token, refresh token.
  + Phân quyền rõ ràng giữa Admin và User.
  + Bảo vệ các API endpoint với NestJS Guards.
* **Khả năng mở rộng:**
  + Kiến trúc modular theo domain.
  + Separation of concerns rõ ràng.
  + Dễ thêm tính năng mới theo pattern có sẵn.
* **Về UX:**
  + Giao diện responsive với TailwindCSS + ShadcnUi.
  + Có loading states và error handling



***Hình 1 - Sơ đồ usecase tổng quan***

## **2. THIẾT KẾ KIẾN TRÚC HỆ THỐNG.**

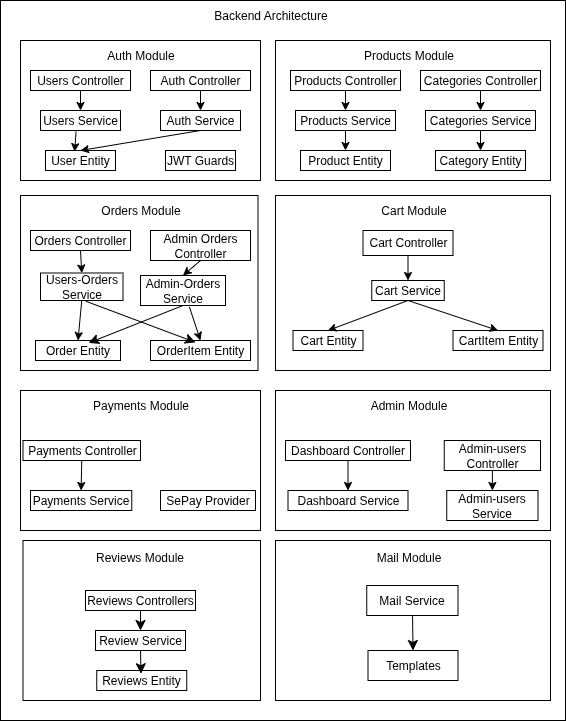
### 2.1. Kiến trúc tổng quan.

Hệ thống được thiết kế theo mô hình Client – Server sử dụng Monorepo cho phép chia sẽ kiểu dữ liệu giữa Frontend và Backend, giúp đồng bộ kiểu dữ liệu từ Backend đến Frontend hiệu quả, tránh boilerplate code, lỗi định nghĩa thiếu các trường bên trong Interface hoặc các Type Definition khác.

Hình 2 -Sơ đồ kiến trúc tổng quan

### 2.2. Kiến trúc Backend (NestJS)

Backend được tổ chức theo kiến trúc modular phân chia theo từng module riêng biệt giúp dẽ chúng ta có thể tập trung phát triển từng tính năng riêng biệt mà không làm ảnh hưởng đến toàn bộ ứng dụng.

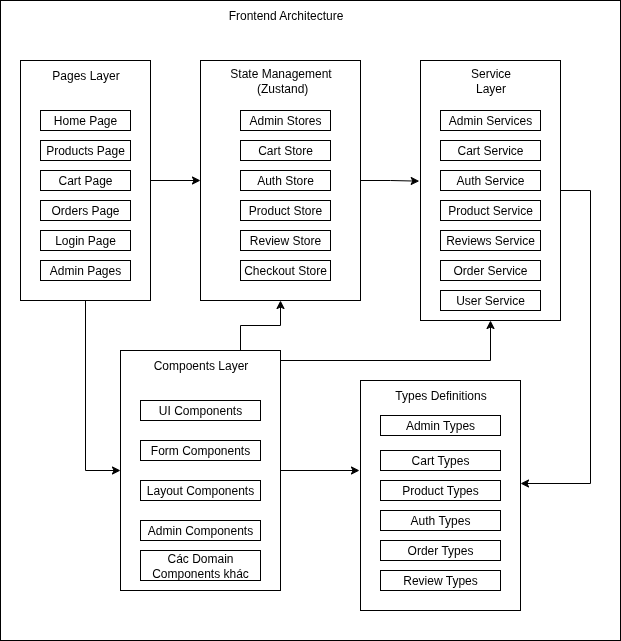
***Hình 3 - Sơ đồ kiến trúc tổng quan của Backend***

**Nguyên lý thiết kế được áp dung:**

1. **Single Responsibillity Principle (SRP)**: Mỗi module chỉ chịu trách nhiệm cho một domain cụ thể (Auth, Products, Orders, Cart, Payments, Admin).
2. **Open/Closed Principle (OCP):** Các module có thể mở rộng thêm chức năng mà không cần sửa đổi code hiện tại.
3. **Separation of Concerns:** Tách biệt rõ ràng giữa Controller (xử lý HTTP requests), Service (business logic), và Entity (data model).

**2.3. Kiến trúc Frontend (React):**

Frontend được tổ chức theo kiến trúc component-based và quản lý state tập trung sử dụng Zustand.



***Hình 4 - Sơ đồ tổng quan kiến trúc Frontend***

**Đặc điểm của kiến trúc Frontend:**

* 1. **Kiến trúc Layered:** Phân chia rõ ràng thành các lớp Pages, Components, Services và Stores.
  2. **Quản lý state:** Sử dụng Zustand với cách tổ chúc stores theo domain giúp dễ quản lý, tách biệt các mối quan tâm.
  3. **An toàn kiểu dữ liệu:** TypeScript được sử dụng toàn bộ với type definition được chia sẽ từ shared-types package.
  4. **Components tái sử dụng:** Các UI components được xây dựng với ShadcnUI có thể tái sử dụng lại ở nhiều nơi, có thể dùng Tailwind để customize lại khi cần giúp tăng tính linh hoạt khi sử dụng.

## **3. THIẾT KẾ CƠ SỞ DỮ LIỆU**

### 3.1. Mô tả các thực thể chính.

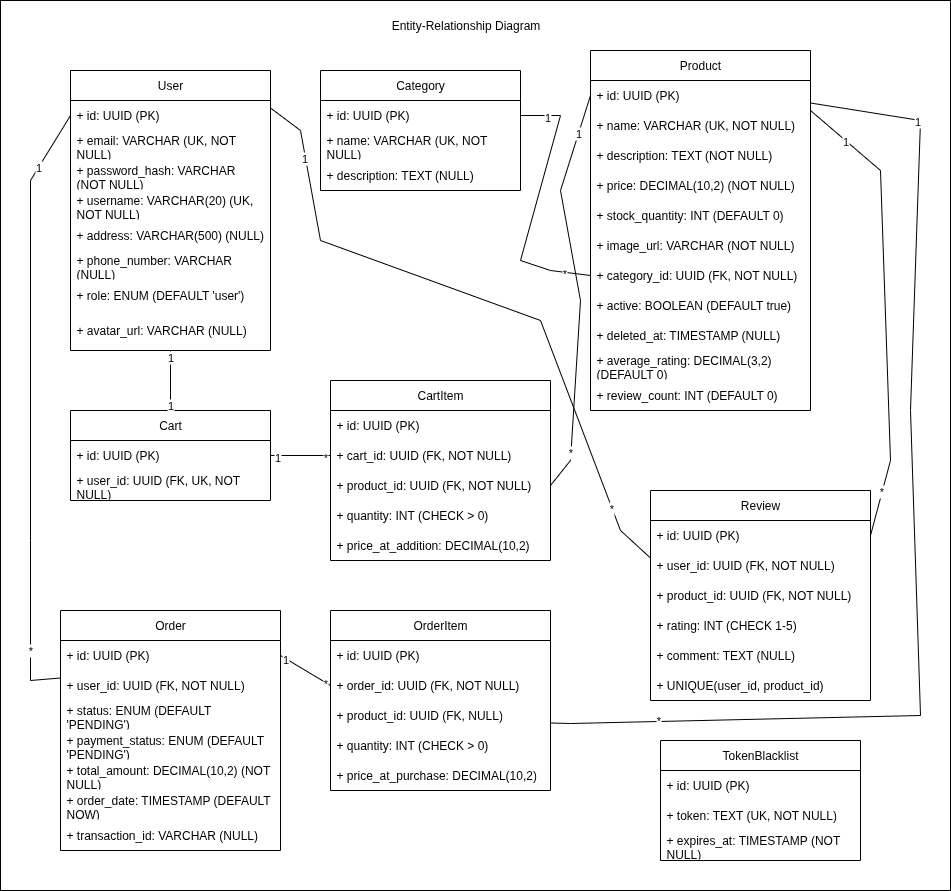
Các thực thể sẽ đại diện cho một table, khi có dữ liệu được sinh ra tại thực thể tương ứng no sẽ tạo thành một table tương ứng và được lưu trong database. NestJS mặc định hỗ trợ TypeORM để tạo và tương tác với các thực thể.

Cơ sở dữ liệu được thiết kế với các thực thể chính như sau:

* + - * + **User**: Lưu trữ thông tin người dùng và quản trị viên
        + **Product**: Lưu trữ thông tin sản phẩm được bay bán trên web.
        + **Category**: Dùng để phân loại sản phẩm. giúp người dùng dễ dàng tìm kiếm và duyệt sản phẩm..
        + **Cart**: Lưu trữ giỏ hàng của một người dùng, chứa các sản phẩm mà họ muốn thanh toán.
        + **CartItem**: Đại diện cho một mặt hàng cụ thể (kèm với số lượng) trong giỏ hàng của người dùng.
        + **Order**: Lưu trữ thông tin đơn hàng do người dùng tạo, kèm trạng thái, tổng tiền và địa chỉ giao hàng, giúp cho người quản trị có thể xử lí đơn hàng này**.**
        + **OrderItem**: Lưu thông tin chi tiết về từng sản phẩm trong một đơn hàng, bao gồm số lượng và giá tại thời điểm mua.
        + **Review:** Lưu trữ các bài đánh giá của người dùng cho các sản phẩm họ đã mua.
        + **TokenBlacklist:** Lưu trữ các access token và refresh token đã bị vô hiệu hóa sau khi người dùng đăng xuất. Bất kì request nào dùng các token này (dù chưa hết hạn) sẽ bị từ chối.

### 3.2. Mối quan hệ giữa các thực thể.

* **User (1) ↔ (1) Cart:** Mỗi người dùng có một giỏ hàng.
* **Cart (1) ↔ (\*) CartItem:** Mỗi giỏ hàng chứa nhiều Items.
* **User (1) ↔ (\*) Order:** Một người dùng có thể có nhiều đơn hàng.
* **Order (1) ↔ (\*) OrderItem:** Mỗi đơn hàng có thể có nhiều Items.
* **Category (1) ↔ (\*) Product:** Mỗi danh mục có thể có nhiều sản phẩm thuộc về.
* **Product (1) ↔ (\*) CartItem:** Có thể có nhiều hơn một sản phẩm trong giỏ hàng.
* **Product (1) ↔ (\*) OrderItem:** Số lượng sản phẩm có thể có nhiều trong một đơn hàng.
* **User (1) ↔ (\*) Review:** Một người dùng có thể đánh giá nhiều sản phẩm.
* **Product (1) ↔ (\*) Review:** Mỗi sản phẩm có thể có nhiều đánh giá.
* **UNIQUE(user\_id, product\_id):** Trong review, mỗi người dùng chỉ review được mỗi sản phẩm một lần (sau khi đã mua sản phẩm).



***Hình 5 - Sơ đồ mối quan hệ giữa các thực thể (ERD)***

### 3.3. Mô tả chi tiết các thực thể

* **Thực thể User:**
  + **id: UUID - Primary Key, định danh duy nhất cho mỗi người dùng.**
  + **email**: string - Not Null, Unique, địa chỉ email của người dùng, dùng để đăng nhập và liên lạc.
  + **passwordHash**: string - Not Null, chuỗi băm mật khẩu của người dùng.
  + **username**: string - Not Null, Unique, tên tài khoản của người dùng.
  + **address**: string - Nullable, địa chỉ giao hàng của người dùng.
  + **phoneNumber**: string - Nullable, số điện thoại của người dùng.
  + **role**: enum - Not Null, Default: 'USER', vai trò của người dùng trong hệ thống (ví dụ: 'USER', 'ADMIN').
  + **avatarUrl**: string - Nullable, đường dẫn đến ảnh đại diện của người dùng.
  + **createdAt**: timestamp - Not Null, thời điểm tài khoản được tạo.
  + **updatedAt**: timestamp - Not Null, thời điểm thông tin tài khoản được cập nhật lần cuối.
* **Thực thể Product:**
  + **id:** UUID - Primary Key, định danh duy nhất cho mỗi sản phẩm.
  + **name:** string - Not Null, Unique, tên sản phẩm.
  + **description:** text - Not Null, mô tả chi tiết về sản phẩm.
  + **price:** decimal - Not Null, giá bán của sản phẩm.
  + **stockQuantity:** int - Not Null, Default: 0, số lượng sản phẩm còn trong kho.
  + **imageUrl:** string - Not Null, đường dẫn đến hình ảnh chính của sản phẩm.
  + **categoryId:** string - Not Null, khóa ngoại tham chiếu đến ID của Category.
  + **active:** boolean - Not Null, Default: true, trạng thái kích hoạt của sản phẩm (true là đang bán, false là đã ẩn).
  + **averageRating:** decimal - Not Null, Default: 0, điểm đánh giá trung bình của sản phẩm.
  + **reviewCount:** int - Not Null, Default: 0, tổng số lượt đánh giá của sản phẩm.
  + **createdAt:** timestamp - Not Null, thời điểm sản phẩm được tạo.
  + **updatedAt:** timestamp - Not Null, thời điểm thông tin sản phẩm được cập nhật lần cuối.
  + **deletedAt:** timestamp - Nullable, thời điểm sản phẩm bị xóa (soft delete).
* **Thực thể Category:**
  + **id**: UUID - Primary Key, định danh duy nhất cho mỗi danh mục.
  + **name**: string - Not Null, Unique, tên của danh mục.
  + **description**: text - Nullable, mô tả chi tiết về danh mục.
  + **createdAt**: timestamp - Not Null, thời điểm danh mục được tạo.
  + **updatedAt**: timestamp - Not Null, thời điểm thông tin danh mục được cập nhật lần cuối.
* **Thực thể Cart:**
  + **id**: UUID - Primary Key, định danh duy nhất cho mỗi giỏ hàng.
  + **userId**: UUID - Not Null, Unique, khóa ngoại tham chiếu đến ID của User, đảm bảo mỗi người dùng chỉ có một giỏ hàng.
  + **createdAt**: timestamp - Not Null, thời điểm giỏ hàng được tạo.
  + **updatedAt**: timestamp - Not Null, thời điểm giỏ hàng được cập nhật lần cuối.
* **Thực thể CartItem:**
  + **id**: UUID - Primary Key, định danh duy nhất cho mỗi mục trong giỏ hàng.
  + **cartId**: UUID - Khóa ngoại, tham chiếu đến ID của Cart.
  + **productId**: UUID - Khóa ngoại, tham chiếu đến ID của Product.
  + **quantity**: int - Not Null, số lượng của sản phẩm được thêm vào giỏ hàng. Ràng buộc CHECK đảm bảo quantity > 0.
  + **priceAtAddition**: decimal - Not Null, giá của sản phẩm tại thời điểm được thêm vào giỏ hàng để tránh ảnh hưởng bởi việc thay đổi giá sau này.
  + **addedAt**: timestamp - Not Null, thời điểm mục được thêm vào giỏ hàng.
* **Thực thể Order:**
  + **id**: UUID - Primary Key, định danh duy nhất cho mỗi đơn hàng.
  + **userId**: UUID - Not Null, khóa ngoại tham chiếu đến ID của User.
  + **orderDate**: timestamp - Not Null, Default: CURRENT\_TIMESTAMP, ngày và giờ đơn hàng được tạo.
  + **status**: enum - Not Null, Default: 'PENDING', trạng thái hiện tại của đơn hàng (ví dụ: 'PENDING', 'PROCESSING', 'SHIPPED', 'DELIVERED', 'CANCELLED').
  + **totalAmount**: decimal - Not Null, tổng số tiền của đơn hàng.
  + **shippingAddress**: text - Not Null, địa chỉ giao hàng.
  + **paymentMethod**: enum - Not Null, Default: 'SEPAY\_QR', phương thức thanh toán đã chọn.
  + **transactionId**: string - Nullable, ID giao dịch từ nhà cung cấp dịch vụ thanh toán.
  + **paymentStatus**: enum - Not Null, Default: 'PENDING', trạng thái thanh toán của đơn hàng (ví dụ: 'PENDING', 'COMPLETED', 'FAILED').
  + **note**: text - Nullable, ghi chú của khách hàng cho đơn hàng.
  + **createdAt**: timestamp - Not Null, thời điểm đơn hàng được ghi nhận vào hệ thống.
  + **updatedAt**: timestamp - Not Null, thời điểm thông tin đơn hàng được cập nhật lần cuối.
* **Thực thể OrderItem:**
  + **orderId**: UUID - Khóa chính phức hợp (Composite Primary Key), khóa ngoại tham chiếu đến ID của Order.
  + **productId**: UUID - Khóa chính phức hợp (Composite Primary Key), khóa ngoại tham chiếu đến ID của Product.
  + **quantity**: int - Not Null, số lượng sản phẩm trong mục đơn hàng này.
  + **priceAtPurchase**: decimal - Not Null, giá của sản phẩm tại thời điểm mua hàng.
* **Thực thể Review:**
  + **id**: UUID - Primary Key, định danh duy nhất cho mỗi đánh giá.
  + **rating**: int - Not Null, điểm đánh giá của người dùng cho sản phẩm (thường từ 1 đến 5).
  + **comment**: text - Nullable, nội dung bình luận, nhận xét của người dùng về sản phẩm.
  + **userId**: UUID - Not Null, (ẩn từ quan hệ user), khóa ngoại tham chiếu đến ID của User.
  + **productId**: UUID - Not Null, (ẩn từ quan hệ product), khóa ngoại tham chiếu đến ID của Product.
  + **createdAt**: timestamp - Not Null, thời điểm đánh giá được tạo.
  + **updatedAt**: timestamp - Not Null, thời điểm đánh giá được cập nhật lần cuối.
* **Thực thể TokenBlacklist:**
  + **id**: UUID - Primary Key, định danh duy nhất cho mỗi bản ghi.
  + **token**: string - Not Null, chuỗi token (JWT) đã bị vô hiệu hóa (ví dụ: khi người dùng đăng xuất).
  + **type**: string - Not Null, loại token, có thể là 'access' hoặc 'refresh'.
  + **expiresAt**: timestamp - Not Null, thời điểm token hết hạn, dùng để dọn dẹp các token cũ khỏi danh sách đen.
  + **createdAt**: timestamp - Not Null, thời điểm token được thêm vào danh sách đen.
  + **updatedAt**: timestamp - Not Null, thời điểm bản ghi được cập nhật lần cuối.

Thiết kế database này giúp đảm bảo tính toàn vẹn dữ liệu, hiệu suất truy vấn tốt, giao dịch an toàn và đảm bảo khả năng mở rộng trong tương lai.

# **CHƯƠNG 3: TRIỂN KHAI BACKEND (NESTJS)**

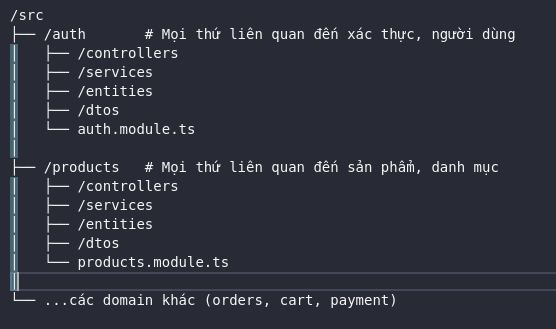
## **1. TỔNG QUAN VỀ NESTJS**

Backend của ứng dụng được xây dựng là NestJS, một framework mạnh mẽ trong hệ sinh thái NodeJS. NestJS nhẹ, linh hoạt, có cấu trúc rõ ràng của một framework chuyên nghiệp. Phù hợp để triển khai cho các Startup muốn có một ứng dụng với đầy đủ tính năng nghiệp vụ chuyên nghiệp cho các ứng dụng có quy mô từ nhỏ, vừa đến lớn, vì scale khá tốt, chi phí vận hành thấp **(đều mà các startup quan tâm nhất)**. Việc chọn NestJS mang lại nhiều lợi ích:

* **Kiến trúc module hóa:** Chúng ta có thể chia nhỏ ứng dụng thành các khối chức năng độc lập (như đã đề cập ở chương trước) giúp dễ quản lý, bảo trì và tái sử dụng.
* **Hỗ trợ TypeScript:** NestJS mặc định hỗ trợ TypeScript, giúp phát hiện lỗi sớm, dễ debug, tiết kiệm thời gian phát triển về lâu dài so với JavaScript.
* **Dependency Injection (DI):** Đây là một trong những điểm mạnh cốt lõi của NestJS, cơ chế DI giúp giảm sự phụ thuộc lẫn nhau giữa các class, dễ dàng thay thế, bảo trì ứng dụng. Tối ưu hiệu suất vì chỉ cần tạo một đối tượng (instance) duy nhất rồi sử dụng lại ở nhiều nơi (inject nó vào các class khác) mà không cần phải tạo một instance cụ thể rồi tự quản lý nó bên trong các class khác.
* **Hệ sinh thái phong phú:** Ngoài hệ sinh thái từ NodeJS nói chung, NestJS tích hợp sẵn và hỗ trợ tốt các thư viện như TypeORM (cho database), Passport, JWT (cho xác thực), Class-validator (cho validate in-output), Class-transformer(cho convert type in-output),...

### **1.1. Cấu trúc thư mục**

Ứng dụng áp dụng nguyên tắc Domain-Driven Design (DDD). Mỗi module là một domain riêng biệt, đại diện cho từng chức năng cụ thể. Ví dụ như module tên domain là “products”, bên trong nó sẽ có các thư mục như services/, controllers/, entities/, dtos/… Cấu trúc domain này áp dụng tương tự với các module khác. Các file bên trong sẽ được khai báo và quản lý bới file [tên domain].module.ts, nếu không khai báo lên file [tên domain].module.ts, khi chạy ứng dụng, NestJS sẽ báo lỗi. Vai trò của file .module.ts này như là một anh giám đốc quản lý, nhân viên là các file khác dưới cấp.



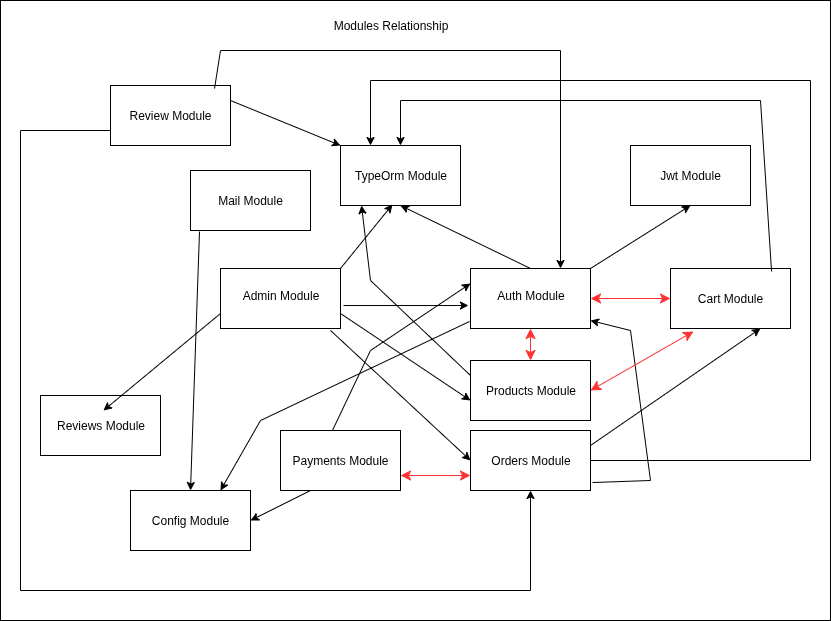
***Hình 6 - Minh họa cấu trúc thư mục theo domain***

**Tóm gọn lợi ích của cách tổ chức này:**

* **Tăng tính gắn kết:** Các file liên quan đến cùng một thư mục sẽ được đặt gần nhau. Ví dụ: các [tên-file].service.ts sẽ nhằm trong thư mục service/.
* **Dễ dàng điều hướng:** Khi cần sửa một tính năng cụ thể thì tất cả code đều nhằm trong cùng một thư mục duy nhất. Rất cần thiết nếu dự án lớn dần.
* **Dễ dàng mở rộng:** Thêm một tính năng mới chỉ đơn giản là tạo ra một thư mục module mới. Nó không ảnh hưởng gì nhiều đến các code ở module khác.

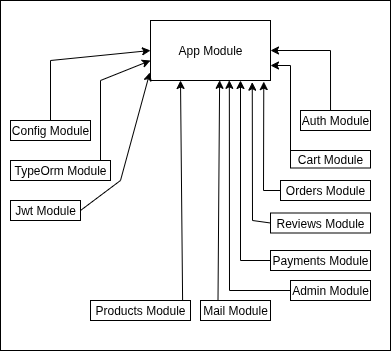
### **1.2. Sơ đồ mối quan hệ giữa các module**

Do sơ đồ kiến trúc đã được trình bày ở chương 2 nên phần này sẽ chỉ trình bày sơ đồ mối quan hệ giữa chúng (module này truy cập đến các service của module kia). Các [tên-domain].module.ts sẽ được quản lý bới file app.module.ts (file module cấp cao nhất). File này tương tự như chủ tịch của một công ty, các file .module.ts khác là các anh giám đốc.

 ***Hình 8 - Biểu đồ mối quan hệ giữa các module***

**Chú thích: - Mũi tên 1 chiều**: module A import các service của module B.

- **Mũi tên 2 chiều**: 2 module import lẫn nhau, dùng các service của nhau.



***Hình 7- Minh họa khai báo các modules cho app module***

## **2. THIẾT KẾ API REST**

### 2.1. Nguyên tắc thiết kế của REST API

Hệ thống backend của đồ án sẽ tuân thủ nguyên tắc thiết kế api theophong cách REST, đảm bảo các api dễ đọc và dễ sử dụng. Dưới đây là các nguyên tắc:

* **Stateless Design:**  Mỗi request gửi lên server phải mang đầy đủ thông tin cần thiết, server không lưu trữ trạng thái của session. Tính năng xác thực phải được thực hiện bằng token, đồ án này sẽ dùng JWT tokens.
* **Uniform Interface:** RestAPI quy định sử dụng các phương thức HTTP theo chuẩn dưới đây:
  + **GET:** Phương thức dùng cho resquet lấy data.
  + **POST:** Dùng cho resquest tạo mới resource.
  + **PUT:** Dùng cho request muốn cập nhật toàn bộ resource.
  + **PATCH:** Dùng cho request muốn cập nhật một phần resource.
  + **DELETE:** Dùng cho request muốn xóa resource.
* **Resource-Based URLs:** Tất cả các endpoint đều thiết kế với cách đặt tên rõ ràng:
  + /api/{resource}.
  + /api/{resource}/{id}.
  + /api/{resource}/{id}/{sub-resource}.

### 2.2. Cấu trúc API Endpoints của Backend

* **Auth Module ( /api/auth ):** Các endpoint của module này sẽ quản lý vòng đời của một session người dùng:
  + POST /api/auth/register : Đăng ký tài khoản mới.
  + POST /api/auth/login : Đăng nhập và nhận tokens.
  + POST /api/auth/logout : Đăng xuất (Đưa tokens vào blacklist).
  + POST /api/auth/refresh-token : Làm mới access token.

Access Token có thời hạn 15 phút, Refresh Token là 7 ngày.Kiễu dữ liệu truyền vào sẽ được validation bởi thư viện class-validator. Response trả về sẽ không bao gồm các thông tin nhạy cảm như password.

* **Products Module:** Enpoints của module này sẽ chịu trách nhiệm quản lý sản phẩm và danh mục sản phẩm:
  + **Các endpoint không cần xác thực bao gồm (Public Endpoints):**
    - GET /api/products : Lấy danh sách sản phẩm (có phân trang).
    - GET /api/products/{id} : Lấy thông tin chi tiết của một sản phẩm.
    - GET /api/categories : Lấy danh sách danh mục.
    - GET /api/categories/{id} : Lấy thông tin của một danh mục.
  + **Các endpoint cần xác thực (cần quyền admin) bao gồm:**
    - POST /api/admin/products : Tạo sản phẩm mới.
    - PUT /api/admin/products/{id} : Cập nhật thông tin sản phẩm.
    - DELETE /api/admin/products/{id} : Xóa sản phẩm.
    - POST /api/admin/categories : Tạo category mới.
    - PUT /api/admin/categories/{id} : Cập nhật category.
    - DELETE /api/admin/categories/{id} : Xóa category.
* **Cart Module ( /api/cart ):** Endpoints quản lý giỏ hàng, cần tối thiểu là quyền người dùng bình thường đã login.
  + GET /api/cart : Lấy giỏ hàng hiện tại.
  + POST /api/cart/items : Thêm sản phẩm vào giỏ hàng
  + PUT /api/cart/items/{productId} : Cập nhật số lượng sản phẩm.
  + DELETE /api/cart/items/{productId} : Xóa sản phẩm khỏi giỏ.
  + DELETE /api/cart : Xóa toàn bộ giỏ hàng.

Mỗi người dùng chỉ sẽ có duy nhất một giỏ hàng.

* **Orders Module:** Endpoints này chịu trách nhiệm quản lý các đơn hàng.
  + **APIs cho người dùng (đã login):**
    - POST /api/orders : Tạo đơn hàng từ giỏ hàng.
    - GET /api/orders : Lấy danh sách đơn hàng của người dùng.
    - GET /api/orders/{orderId} : Lấy thông tin chi tiết đơn hàng.
    - GET /api/orders/{orderId}/check-payment-status : Kiểm tra trạng thái thanh toán.
    - DELETE /api/orders/{orderId}/cancel : Hủy đơn hàng.
  + **APIs cho admin:**
    - GET /api/admin/orders : Lấy tất cả đơn hàng.
    - GET /api/admin/orders/{orderId} : Lấy thông tin chi tiết đơn hàng.
    - PATCH /api/admin/orders/{orderId}/status : Cập nhật trạng thái đơn hàng.
* **Payments Module ( /api/payment ):** Endpoints này dùng để quản lý api thanh toán (dùng Sepay).
  + POST /api/payment/create : Tạo QR code cho thanh toán.
  + POST /api/payment/webhook/sepay : Nhận webhook từ SePay.
  + GET /api/payment/methods : Lấy danh sách phương thức thanh toán.
  + POST /api/payment/switch/{orderId} : Chuyển đổi phương thức thanh toán.
* **Reviews Module ( /api/reviews ):** 
  + POST /api/reviews/{productId} : Tạo đánh giá cho một sản phẩm.
  + PUT /api/reviews/{reviewId} : Sửa lại đánh giá của một sản phẩm.
  + DELETE /api/reviews/{productId} : Xóa đánh giá một người dùng tại một sản phẩm.
  + GET /api/reviews/{productId} : Lấy danh sách đánh giá của một sản phẩm.
  + GET /api/reviews/{productId}/user-review : Lấy đánh già của người dùng (dã login) tại một sản phẩm (dùng để kiểm tra xem user đó có review hay chưa).
* **Admin Module :** Endpoints cho quản trị viên xem thống kê các chỉ số kinh doanh.
  + **Endpoints hiển thị thông tin quản lý tổng quan ( api/admin/dashboard ):**
    - GET /api/admin/summary : Lấy tổng quan dashboard với các chỉ số thống kê cơ bản.
    - GET /api/admin/detailed-stats : Lấy thống kê chi tiết một vài chi số như sô lương đơn hàng phân theo trạng thái, doanh thu 6 tháng gần nhất.
  + **Endpoints quản lý tài khoảng của người dùng ( api/admin/manager-users ):**
    - GET /api/admin/manager-users : Lấy tất cả danh sách tài khoảng người dùng.
    - GET /api/admin/manager-users/{userId} : Lấy thông tin của một người dùng.
    - PUT /api/admin/manager-users/{userId} : Cập nhật quyền và khóa/mở tài khoảng của một người dùng.

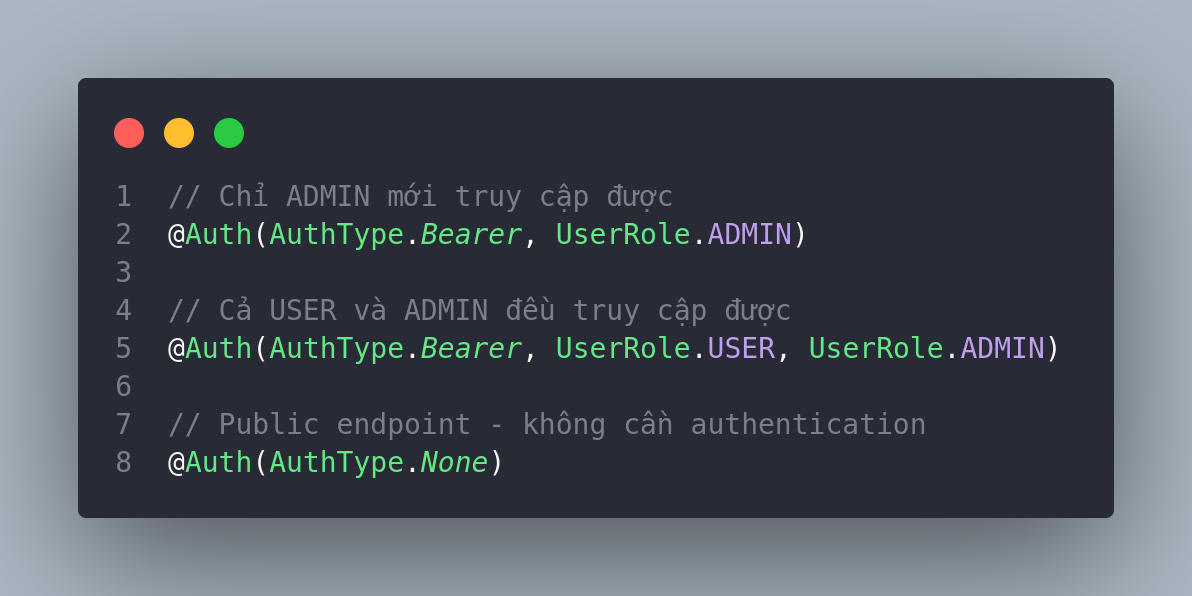
### 2.3. Xác thực và phân quyền.

Tính năng xác thực và phân quyền người dùng sẽ dùng Guard và thư viện jwt cho việc kiểm tra access token từ request gửi lên. Phải cần tạo một custom decorator cho việc truyền tham số cấu hình này. Tên của custom decorator này sẽ là “Auth”.

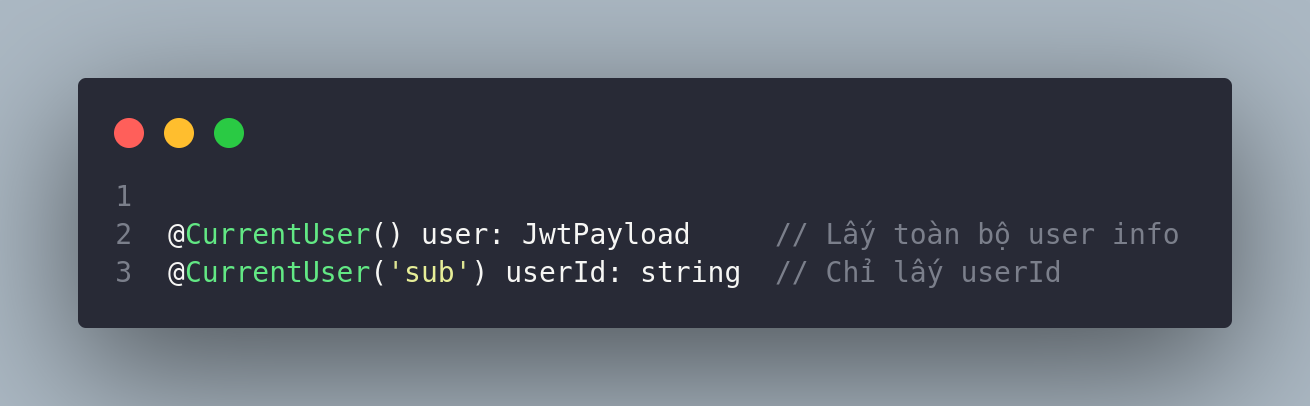


***Hình 9 - Guard dùng cho xác thực***

* Kiểm tra access token truyền lên có hợp lệ hay không.
* Giải mã token để lấy thông tin người dùng (nếu token không hợp lệ sẽ trả về lỗi ngay bước này).
* Inject thông tin người dùng vào biến request (có thể dùng cho các guard khác để kiểm tra quyền hạn của người dùng).

***Hình 10 - Guard dùng cho phân quyền***

* Sau khi kiểm tra xem token có hợp lệ chưa thì gọi đến Role Guard.
* Trích thông tin người dùng từ biến request đã được inject trước đó và thông tin quyền cần phải check từ Metadata được thêm vào thông qua @Auth decorator
* Nếu so sánh bằng nhau thì cho qua.

 ***Hình 11 - Decorator lấy thông tin người dùng từ request***

Decorator này sẽ trực lấy thông tin từ người dùng từ việc giải mã token rồi inject vào biến request ở các bước trên.

### 2.4. Thiết kế quy tắc Request/Response.

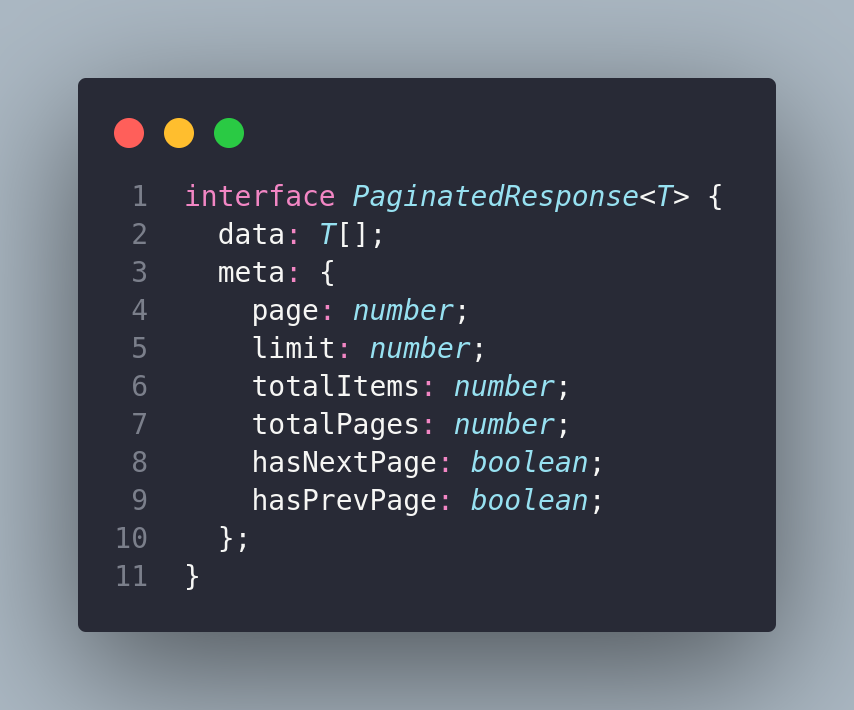
#### 2.4.1. Validation Dtos

Tất cả các trường dữ liệu bên trong Dtos đều phải thực hiên xác thực dữ liệu nhằm đảm bảo dữ liệu ttruyền vào là đúng.

***Hình 12 - Ví dụ với order Dto***

#### 2.4.2. Thiết kế phân trang.

Các endpoints có dữ liệu là trả về danh sách các phần tử đèu phải thực hiện phân trang nhất quán.



***Hình 13 - Interface sẽ được dùng cho chuẩn phân trang***

***Hình 14 - Webhook cho thanh toán***

Thiết kế api này đảm bảo mỗi endpoint chỉ thực hiện một nhiệm vụ cụ thể và rõ ràng. Kiến trúc này cũng dễ dàng mở rộng để bổ sung thêm các tính năng mới trong tương lai mà không ảnh hưởng đến các chức năng hiện có.

## **3. KIẾN TRÚC MODULE VÀ DEPENDENCY INJECTION**

### **3.1. Kiến trúc module trong NestJS.**

Hệ thống backend của sản phẩm sẽ được kiến kế theo kiến trúc modular, đây là kiến trúc mặc định của NestJS, mỗi module sẽ chịu trách nhiệm cho một domain cụ thể. Ngoài ra NestJS còn hỗ trợ các loại kiến trúc khác như MicroService, Event-driven,…

#### **3.1.1. Đặc điểm của kiến trúc modular.**

* Mỗi module tương ứng với một business domain. Ví dụ như Auth, Products, Orders, Payments,…
* Các module đều đôc lập ít phụ thuộc lẫn nhau và dễ dàng thay thế nhau hoặc chinh sửa.
* Nếu một module muốn truy cập service của module khác thì phải khai báo module đó tại file [tên domain].module.ts, với điều kiện là các service của module đó phải được export ra bên ngoài.

#### 3.1.2. **Root Module (AppModule).**

AppModule là điểm khởi đầu của ứng dụng, nó chưa các cấu hình database, file môi trường, là nơi khai báo các module khác bên trong hệ thống, nếu không khai báo, NestJS sẽ không biết rằng module đó có tồn tại và sẽ báo lỗi khi chạy ứng dụng.

Giải thích các thuộc tính của một module:

* Thuộc tính **import :** Đây là nơi import các module và cấu hình chúng nếu cần.
* Thuộc tính **controllers** : Đây là nơi khai báo các class thuộc file [tên].controller.ts, đây là các file chiu trách nhiệm định nghĩa các endpoint và điều hướng request từ client đến các service.
* Thuộc tính **provider :** Đây là khai báo các class business logic như service, guard, decorator và các file chưa logic bổ trơ cho chúng**.** Hoặc đôi khi là nơi cấu hình các global service cho việc sử dụng toàn cục mà không cần phải import. Trong trường hợp này là nơi khai báo AuthenticationGuard guard này sẽ tự động apply cho tất cả các endpoint trong ứng dụng. Sau đó sẽ truyền dữ liệu vào 2 Guard AcessTokenGuard và RoleGuard của AuthModule để xử lý logic xác thực và phân quyền.

***Hình 15 - AppModule của ứng dụng***

### **3.2. Thiết kế các module chức năng.**

#### **3.2.1. Auth Module**

Đây là module chịu trách nhiệm cho các logic bảo mật và xác thực của hệ thống, là nơi định nghĩa logic xác thực và phân quyền. Module nào cần bảo vệ sẽ import sử dụng các service của module này.

Đặc diểm của Auth Module:

* Export UsersService cho các module nào cần truy cập, sửa đổi thông tin người dùng.
* Export AccessTokenGuard phục vụ cho việc xác thục truy cấp đến các endpoint cần bảo vệ.
* Export RoleGuard để kiểm tra quyền của người dùng xem chó truy cập được endpoint hay không.
* Sử dụng logic mã hóa mật khảu người dùng trước khi lưu vào database.
* AuthService chứa các logic phục vụ cho việc tạo tài khoảng, đăng nhập, tạo token, kiểm tra token hợp lệ.



***Hình 16 - Auth Module***

#### **3.2.2. Products Module**

 Module này phục vụ cho việc tạo sản phẩm, sửa thông tin sản phẩm, xóa sản phẩm, lấy danh sách sản phẩm. Ngoài ra còn có logic tương tư như trên cho danh mục sản phẩm.

***Hình 17 - Product Module***

#### **3.2.3. Orders Module**

Đây là module phục vụ cho việc xử lý đơn hàng như tạo đơn hàng, cập nhật thông tin trạng thái đơn hàng, lấy danh sách đơn hàng.

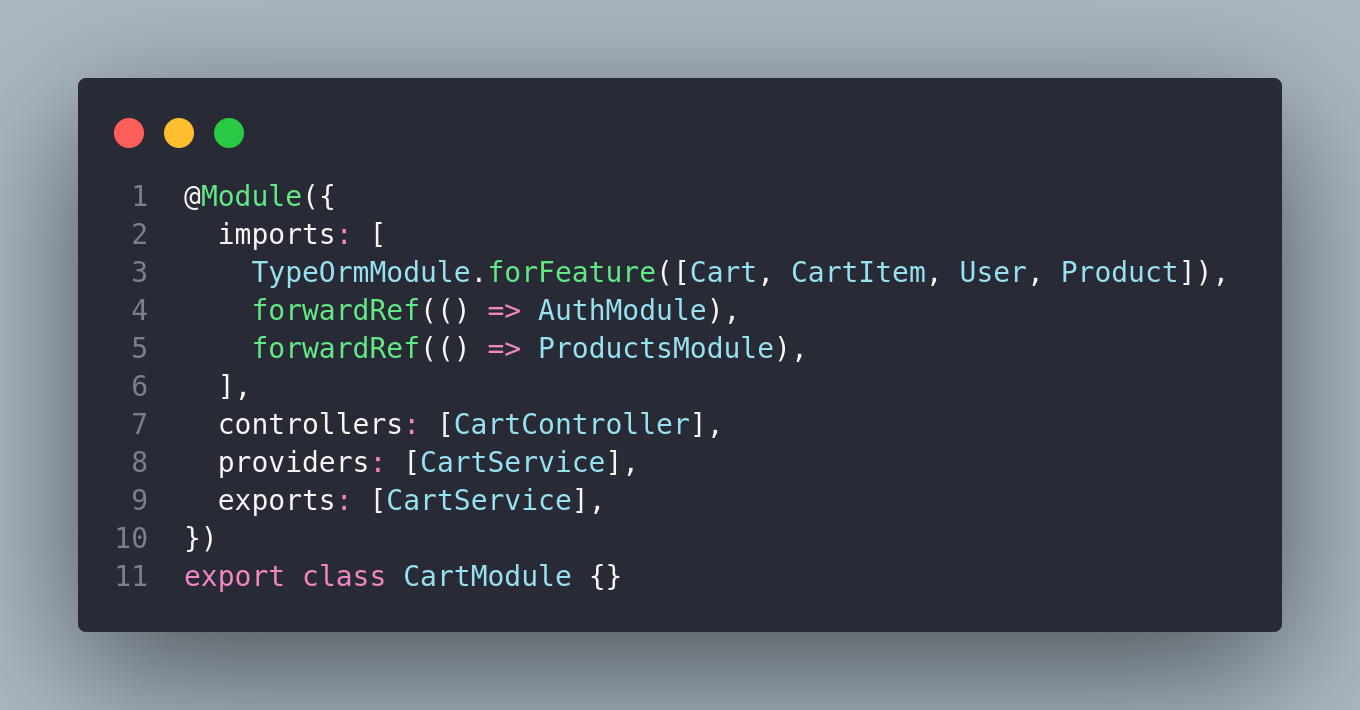
***Hình 18 - Order Module***

#### 3.2.4. **Payments Module**

Nhiệm vụ chính của module này là nhận WebHook từ api thanh toán (ở đây là Sepay). Ngoài ra module này còn có cách tính năng như, tạo Qrcode, lay danh sách các phương thức thanh toán, đổi phương thức thanh toán. Hệ thống hiện tại chỉ có 2 phương thúc thành toán đó là dùng Sepay và COD (thanh toán khi nhận hàng không dùng api bên thứ 3). Các phương thức thanh toán này sẽ được quan lý bởi một Factory, nơi quan lý các provider cho thanh toán, giúp thống nhất giao diện thanh toán khi dùng nhiều hơn một dịch vụ thanh toán. Dễ dàng mỡ rộng thêm các phương thức thanh toán mới như Momo, ZaloPay cho sau này.

***Hình 19 - Payments Module***

**3.2.5. Cart Module**

Module này chịu trách nhiệm quản lý giỏ hàng của người dùng như lấy giỏ hàng hiện tại, thêm sản phẩm vào giỏ hàng, cập nhật số lượng sản phẩm trong giỏ hàng, xóa giỏ hàng.

Hình 20 - Cart Module

#### **3.2.6. Mail Module**

Đây là module sẽ được cấu hình global. Module này chịu trách nhiệm kết nối với api bên ngoài (dùng mailtrap api) để phục vụ cho việc gửi mail khi tạo tài khoảng thành công. Nhưng trong phạm vi đồ án này, mail sẽ được gửi nhận bên trong sandbox của dịch vụ mailtrap do vấn đề chi phí.

***Hình 21 - Mail Module***

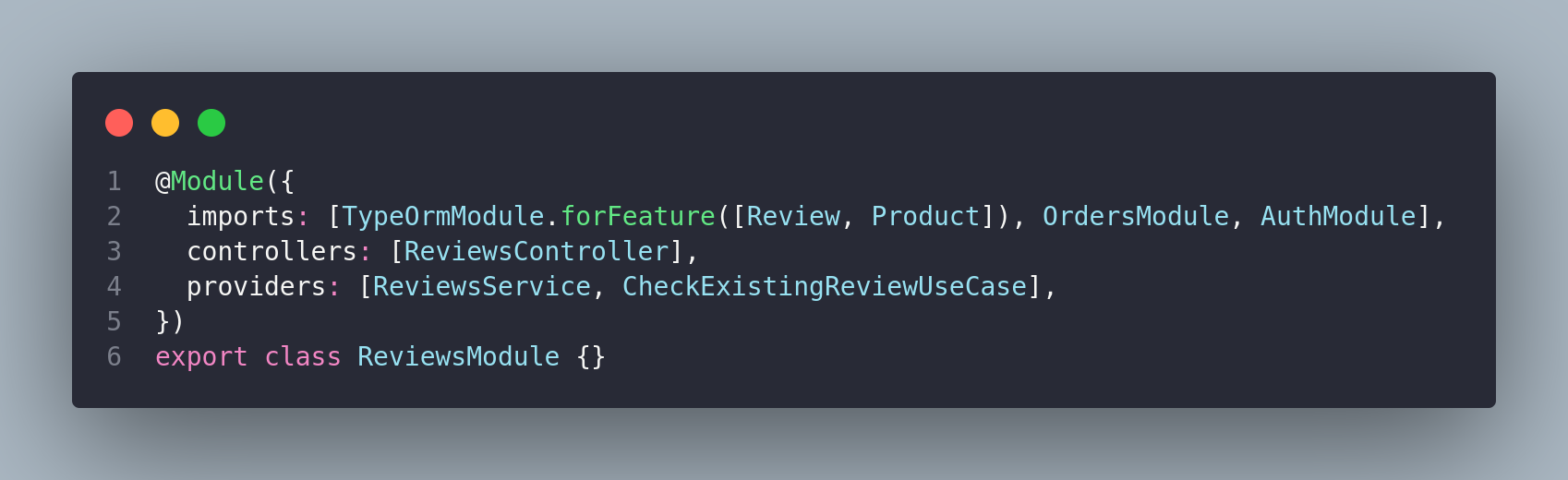
#### **3.2.7. Admin Module**

Hầu hết các module chức năng khác đều có một vài tính năng cho admin, còn module này sẽ định nghĩa các endpoint, business logic đặc biệt hơn dành cho admin như dashboard để xem thông tin tổng quan của hệ thống, manager-user cho quản lý người dùng như nâng cấp quyền, khóa tài khoảng người dùng.

  
***Hình 22 - Admin Module***

#### **3.2.8. Reviews Module**

Module này dùng để quản lý các đánh giá trên một sản phẩm với tính năng kiểm tra xem người dùng nào đã mua sản phẩm thi mới được đánh giá. Mội người dùng chỉ được đánh giá một sản phẩm đã mua một lần duy nhất, nhưng có thể tự sửa, xóa đánh giá của mình.

***Hình 23 - Reviews Module***

### **3.3. Nguyên lý IOC (Inversion of Control)**

IOC là một nguyên lý thiết kế phần mềm, trong đó quyền kiểm soát việc tạo và quản lý các phụ thuộc (dependencies) được chuyển từ lập trình viên sang framework. Thay vì lập trình viên phải tự tạo instance bằng từ khóa “new” và cấu hình phụ thuộc trong mỗi class, framework sẽ tự động đảm nhận việc này. Developer chỉ cần khai báo rõ ràng các phụ thuộc trong constructor, phần còn lại như khởi tạo, tiêm vào class và quản lý vòng đời sẽ do framework xử lý.

#### 3.3.**1. Dependency Injection (DI)**

#### Dependency Injection là cơ chế giúp hiện thực hóa nguyên lý IOC. NestJS sử dụng một hệ thống gọi là Dependency Injection Container (DI Container) để tự động:

* Quản lý vòng đời của các phụ thuộc
* Tạo instance đúng lúc cần
* Tiêm (inject) các phụ thuộc vào constructor của class sử dụng

Nhờ cơ chế này, các class không cần biết cách khởi tạo hay cấu hình các đối tượng phụ thuộc, chỉ cần khai báo trong constructor là có thể sử dụng được.

Mặc định, DI Container của NestJS hoạt động theo cơ chế singleton: mỗi provider chỉ được tạo một lần duy nhất trong toàn bộ ứng dụng, và cùng một instance sẽ được tái sử dụng tại mọi nơi cần thiết. Cơ chế này giúp tối ưu hiệu năng, giảm việc tạo thừa các instance không cần thiết, đặc biệt quan trọng khi ứng dụng có số lượng class lớn.