BÁO CÁO PHÂN TÍCH THIẾT KẾ MODULE PAYMENT-TRANSACTION

1. TỔNG QUAN THIẾT KẾ HIỆN TẠI

Module payment-transaction được thiết kế theo mô hình **3-layer architecture** của NestJS:

- **Controller Layer**: PayOrderController - xử lý HTTP requests
- **Service Layer**: PaymentTransactionService - chứa business logic
- **Repository Layer**: TypeORM repositories - tương tác với database

1.2 Các thành phần chính

- PaymentTransactionService: Logic nghiệp vụ thanh toán
- VNPayService: Implementation cụ thể cho VNPAY
- PayOrderController: Controller xử lý API endpoints

2. PHÂN TÍCH DESIGN PATTERNS ĐƯỢC ÁP DỤNG

2.1 Factory Pattern

Vị trí áp dụng: PaymentGatewayFactory:

```
5   @Injectable()
6   export class PaymentGatewayFactory {
7     constructor(
8       private readonly vnpayService: VNPayService,
9     ) {}
10
11    createGateway(method: string): PaymentGateway {
12      switch(method) {
13        case 'VNPAY': return this.vnpayService;
14        default: throw new Error(`Unsupported: ${method}`);
15      }
16    }
17  }
```

Mục đích thay đổi yêu cầu:

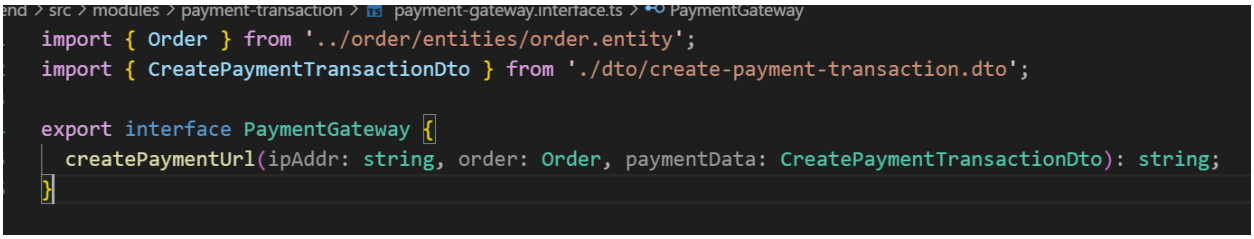
- **Yêu cầu ban đầu**: Chỉ tích hợp VNPAY
- **Yêu cầu mở rộng**: Cần thêm nhiều cổng thanh toán (Momo, Paypal, etc.)

Lý do sử dụng Factory Pattern:

- **Tập trung logic tạo object**: Logic chọn gateway được tập trung trong Factory
- **Dễ mở rộng**: Thêm gateway mới chỉ cần implement interface và đăng ký với Factory
- **Single Responsibility**: Factory chỉ có trách nhiệm tạo gateway instances

2.2 Strategy Pattern

Vị trí áp dụng: PaymentGateway interface + VNPayService



Mục đích thay đổi yêu cầu:

- **Yêu cầu ban đầu**: Chỉ hỗ trợ VNPAY
- **Yêu cầu mở rộng**: Hỗ trợ nhiều phương thức thanh toán khác nhau

Lý do sử dụng Strategy Pattern:

- **Thay thế algorithm**: Có thể thay đổi phương thức thanh toán mà không ảnh hưởng code client
- **Mở rộng dễ dàng**: Thêm gateway mới chỉ cần implement interface
- **Test dễ dàng**: Có thể mock interface để test
- **Loose coupling**: Client code không phụ thuộc vào implementation cụ thể