

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY



CAPSTONE PROJECT REPORT

Báo cáo phân tích Conhension và Coupling

NHÓM 20

Information and communications technology industry

SUPERVISOR NAME: Ph. D. Nguyen Thi Thu Trang

Class: Software Design & Construction

Code: 154041

Department: Computer Science

School: School of Information and Communications Technology

HA NOI, 12/2024

Mục lục

1	Vấn đề chung	2
2	Lớp CartItem	2
3	Lớp OrderItem	2
4	Lớp CartServiceImpl	3
5	Lớp OrderServiceImpl	4
6	Lớp OrderController	5
7	Lớp InvoiceController	6
8	Các vấn đề liên quan đến SOLID	7

1 Vấn đề chung

Vấn đề: Trong code hiện tại đang sử dụng chính Entity để làm data luân chuyển trong code và data gửi lên cũng như trả về. Điều này dễ dẫn đến các vấn đề về Coupling

Giải pháp: Tạo và sử dụng DTOs (Data Transfer Objects) cho việc luân chuyển dữ liệu. Tạo các lớp Response, Request phục vụ cho việc lấy dữ liệu và trả dữ liệu cho client

2 Lớp CartItem

Vấn đề 1: Truy cập trực tiếp vào đối tượng **Product** làm tăng Stamp Coupling. Điều này gây khó khăn khi thay đổi cấu trúc dữ liệu của **Product**, dẫn đến việc phải cập nhật ở nhiều nơi.

Giải pháp: Chỉ lưu trữ **Product** bằng ID thay vì đối tượng đầy đủ. Điều này giúp giảm mức độ phụ thuộc giữa các lớp.

```
@Data
@AllArgsConstructor
@Document(collection = "cart_item")
public class CartItem {
    // Vấn đề 1: Truy cập trực tiếp product làm tăng Stamp Coupling.
    // Giải pháp: Có thể đổi cách lưu trữ (chỉ lưu bằng id)
    private Product product;
    private int quantity;
}
```

Hình 1: Hình ảnh minh họa.

3 Lớp OrderItem

Vấn đề 1: Truy cập trực tiếp vào đối tượng **Product** làm tăng Stamp Coupling. Khi sử dụng đối tượng đầy đủ của **Product**, các thay đổi trong lớp **Product** sẽ tác động đến lớp **OrderItem**.

Giải pháp: Tương tự như lớp **CartItem**, nên chỉ lưu trữ **Product** bằng ID.

```

@Data
@AllArgsConstructor
@NoArgsConstructor
public class OrderItem {
    // Vấn đề 1: Truy cập trực tiếp product làm tăng Stamp Coupling.
    // Giải pháp: Có thể đổi cách lưu trữ (chỉ lưu bằng id)
    private Product product;
    private int quantity;
    private int price;
}

```

Hình 2: Hình ảnh minh họa.

4 Lớp CartServiceImpl

Vấn đề 1: Thao tác trực tiếp trên `listCartItem` làm tăng Stamp Coupling. Lớp `CartServiceImpl` phụ thuộc vào chi tiết của `CartItem`, dẫn đến khó khăn khi thay đổi cấu trúc của giỏ hàng.

Giải pháp: Chuyển logic thao tác với `listCartItem` vào trong phương thức của lớp `Cart`, ví dụ như `cart.addItem()` hoặc `cart.removeItem()` để giảm mức độ phụ thuộc.

Vấn đề 2: Logic tính tổng giá của giỏ hàng bị lặp lại ở nhiều nơi, giảm Cohesion của lớp.

Giải pháp: Chuyển logic tính tổng giá vào trong phương thức của lớp `Cart`, ví dụ `cart.updateTotalPrice()`.

```

@Override
public Cart addCartProduct(String cartId, String productId, int quantity) {
    Cart cart = getCart(cartId);
    Product product = productRepository.findById(productId)
        .orElseThrow(() -> new ProductNotAvailableException("Product not found"));

    if (cart.getListCartItem() == null) {
        cart.setListCartItem(new ArrayList<>());
    }

    // Vấn đề: Thao tác trực tiếp trên listCartItem làm tăng Stamp Coupling.
    // Giải pháp: Chuyển logic này vào phương thức của class Cart (cart.addItem()).
    Optional<CartItem> existingItem = cart.getListCartItem().stream()
        .filter(item -> item.getProduct().getId().equals(productId))
        .findFirst();

    if (existingItem.isPresent()) {
        existingItem.get().setQuantity(existingItem.get().getQuantity() + quantity);
    } else {
        cart.getListCartItem().add(new CartItem(product, quantity));
    }

    // Vấn đề: Logic tính tổng giá đang lặp lại ở nhiều nơi, làm giảm Cohesion.
    // Giải pháp: Di chuyển logic này vào class Cart (cart.updateTotalPrice())
    cart.setTotalPrice(cart.getListCartItem().stream() Stream<CartItem>
        .mapToInt(item -> item.getProduct().getSellPrice() * item.getQuantity()) IntStream
        .sum());
    return cartRepository.save(cart);
}

```

Hình 3: Hình ảnh minh họa.

```

@Override
public Cart removeCartProduct(String cartId, String productId) {
    Cart cart = getCart(cartId);
    // Vấn đề: Thao tác xóa sản phẩm trực tiếp từ listCartItem làm tăng Coupling.
    // Giải pháp: Tạo phương thức cart.removeItem(productId) trong class Cart.
    cart.getListCartItem().removeIf(item -> item.getProduct().getId().equals(productId));
    cart.setTotalPrice(cart.getListCartItem().stream() Stream<CartItem>
        .mapToInt(item -> item.getProduct().getSellPrice() * item.getQuantity()) IntStream
        .sum());
    return cartRepository.save(cart);
}

```

Hình 4: Hình ảnh minh họa.

5 Lớp OrderServiceImpl

Vấn đề 1: Việc chuyển đổi từ CartItem sang OrderItem, tính tổng giá, tính VAT và phí vận chuyển đang thực hiện trực tiếp trong phương thức createOrder. Điều này làm giảm Cohesion của lớp OrderServiceImpl.

Giải pháp: Chuyển các thao tác này vào trong lớp Order, ví dụ: order.calculateTotalAmount().

```

@Override
public Order createOrder(String cartId, DeliveryInfo deliveryInfo) {
    List<CartItem> cartItems = cartService.getAllCartItems(cartId);

    // Vấn đề: Việc chuyển đổi từ CartItem sang OrderItem, tính tổng giá, tính VAT và phí vận chuyển đang thực hiện trực tiếp trong phương thức.
    // Giải pháp: Chuyển các thao tác này vào trong class Order để tăng Cohesion, ví dụ: order.calculateTotalAmount().
    List<OrderItem> orderItems = cartItems.stream()
        .map(cartItem -> new OrderItem(cartItem.getProduct(), cartItem.getQuantity(), cartItem.getProduct().getSellPrice()))
        .toList();
    int totalAmount = orderItems.stream().mapToInt(item -> item.getQuantity() * item.getPrice()).sum();
    totalAmount += totalAmount * Constants.PERCENT_VAT/100 + deliveryInfo.getShippingFees();
    Order order = new Order();
    order.setCartId(cartId);
    order.setListOrderItem(orderItems);
    order.setDeliveryInfo(deliveryInfo);
    order.setTotalAmount(totalAmount);
    order.setStatus(Constants.ORDER_STATUS_PENDING);
    return orderRepository.save(order);
}

```

Hình 5: Hình ảnh minh họa.

6 Lớp OrderController

Vấn đề 1: client gửi cả DeliveryInfo lên là không cần thiết (Stamp Coupling)

Giải pháp: Chuyển các thao tác này vào trong lớp Order, ví dụ: order.calculateTotalAmount().

```

@Override
public Order createOrder(String cartId, DeliveryInfo deliveryInfo) {
    List<CartItem> cartItems = cartService.getAllCartItems(cartId);

    // Vấn đề: Việc chuyển đổi từ CartItem sang OrderItem, tính tổng giá, tính VAT và phí vận chuyển đang thực hiện trực tiếp trong phương thức.
    // Giải pháp: Chuyển các thao tác này vào trong class Order để tăng Cohesion, ví dụ: order.calculateTotalAmount().
    List<OrderItem> orderItems = cartItems.stream()
        .map(cartItem -> new OrderItem(cartItem.getProduct(), cartItem.getQuantity(), cartItem.getProduct().getSellPrice()))
        .toList();
    int totalAmount = orderItems.stream().mapToInt(item -> item.getQuantity() * item.getPrice()).sum();
    totalAmount += totalAmount * Constants.PERCENT_VAT/100 + deliveryInfo.getShippingFees();
    Order order = new Order();
    order.setCartId(cartId);
    order.setListOrderItem(orderItems);
    order.setDeliveryInfo(deliveryInfo);
    order.setTotalAmount(totalAmount);
    order.setStatus(Constants.ORDER_STATUS_PENDING);
    return orderRepository.save(order);
}

```

Hình 6: Hình ảnh minh họa.

Vấn đề 2: khi cập nhật các trạng thái của order cho khách hàng như approve, cancel, reject, việc sử dụng cùng 3 phương thức có cùng 1 cấu trúc khác endpoint gây ra Logical Cohesion.

```

@PostMapping("/update-status/approve/{orderId}")
public ResponseEntity<AIMSResponse<Object>> approveOrder(@PathVariable String orderId) {
    Order order = orderService.updateStatusOrder(orderId, Constants.ORDER_STATUS_PROCESSING);
    return ResponseUtil.success200Response(message:"Approve order successfully", order);
}

@PostMapping("/update-status/cancel/{orderId}")
public ResponseEntity<AIMSResponse<Object>> cancelOrder(@PathVariable String orderId) {
    Order order = orderService.updateStatusOrder(orderId, Constants.ORDER_STATUS_CANCELLED);
    return ResponseUtil.success200Response(message:"Cancel order successfully", order);
}

```

Tan Nguyen, 3 weeks ago • update code and design

```

@PostMapping("/update-status/reject/{orderId}")
public ResponseEntity<AIMSResponse<Object>> rejectOrder(@PathVariable String orderId) {
    Order order = orderService.updateStatusOrder(orderId, Constants.ORDER_STATUS_REJECTED);
    return ResponseUtil.success200Response(message:"Reject order successfully", order);
}

```

Hình 7: Hình ảnh minh họa

Giải pháp : Gộp thành 1 phương thức duy nhất nhận status làm tham số.

```

@PostMapping("/update-status/{orderId}")
public ResponseEntity<AIMSResponse<Object>> updateOrderStatus(
    @PathVariable String orderId,
    @RequestParam OrderStatus status
) {
    Order order = orderService.updateStatusOrder(orderId, status);
    return ResponseUtil.success200Response(
        String.format("Update order status to %s successfully", status),
        order
    );
}

```

Hình 8: Hình ảnh minh họa

7 Lớp InvoiceController

Vấn đề: Truy cập trực tiếp Order và trong request gây ra stamp coupling.

```

@PostMapping("/create")
public ResponseEntity<AIMSResponse<Object>> create(@RequestBody Invoice invoice) {
    return invoiceService.create(invoice);
}

```

Hình 9: Hình ảnh minh họa

```

@NotNull
public class Invoice {
    @Id
    private String invoiceId;
    private Order order;
    private PaymentTransaction paymentTransaction;
}

```

Hình 10: Hình ảnh minh họa

Giải pháp: Chỉ cần sử dụng orderId và paymentTransactionId gửi lên request.

8 Các vấn đề liên quan đến SOLID

Project chủ yếu vi phạm đến tính Open-Close Principle trong đó, khi thêm các sản phẩm mới, việc tạo sản phẩm, thay đổi cách thức vận chuyển, thêm phương thức thanh toán với thể nội địa mới sẽ cần phải sửa lại mã nguồn mà không thể mở rộng, đồng thời vi phạm các vấn đề coupling như control coupling.

Chi tiết các vấn đề và giải pháp liên quan đến solid và additional requirement nằm trong file báo cáo DesignPattern của nhóm.