📌 Giải thích chi tiết về kiến trúc, luồng xử lý và lý do sử dụng các thành phần

---

1️⃣ Tổng quan kiến trúc

Kiến trúc của code trên được thiết kế theo mô hình **MVC (Model - View - Controller)** kết hợp với **Factory Pattern** để dễ mở rộng. Cụ thể:

- **Model (`product.model.js`)**: Định nghĩa cấu trúc dữ liệu của các loại sản phẩm trong MongoDB.
- **Controller (`product.controller.js`)**: Xử lý yêu cầu từ client, gọi đến service để thực hiện logic.
- **Service (`product.service.js`)**: Chứa logic xử lý chính của ứng dụng, sử dụng Factory Pattern để tạo các loại sản phẩm khác nhau.
- **Router (`product.router.js`)**: Định nghĩa API endpoint và sử dụng middleware `authentication` để xác thực người dùng.
- **Middleware (`authUtils.js`)**: Xác thực người dùng bằng JWT trước khi cho phép thực hiện các hành động.

2️⃣ Phân tích chi tiết từng thành phần

🔹 1. Model (`product.model.js`)

Chịu trách nhiệm định nghĩa schema của MongoDB, gồm có:

- **`Product` (Sản phẩm tổng quát)**: Chứa thông tin chung của tất cả sản phẩm.
- **`Clothing` (Quần áo)**: Có thuộc tính riêng như `brand`, `size`, `material`.
- **`Electronics` (Đồ điện tử)**: Có `manufacturer`, `model`, `color`.
- **`Furniture` (Nội thất)**: Có `brand`, `material`, `color`.

➡️ **Lý do tách riêng các schema**: **Tối ưu hóa dữ liệu**, giúp tìm kiếm nhanh hơn, có thể mở rộng dễ dàng mà không làm ảnh hưởng đến cấu trúc chung.

🔹 2. Controller (`product.controller.js`)

Xử lý request từ người dùng và gọi đến service:

```
createProduct = async(req, res, next) => {
    console.log('👉 req.user', req.user)
    new SuccessResponse({
        message: '☑️🎉 Product created successfully',
        data: await
ProductFactory.createProduct(req.body.product_type, {
            ...req.body,
            product_shop: req.user.userId
        })
    }).send(res)
}
```

- **Lấy `req.user.userId` từ middleware `authentication`** để biết sản phẩm do shop nào tạo.
- **Gọi `ProductFactory.createProduct()`** để tạo sản phẩm tương ứng với `product_type` (`Clothings`, `Electronics`, `Furniture`).
- **Gửi phản hồi thành công bằng `SuccessResponse`**.

Tại sao không viết logic trực tiếp trong Controller?
➡️ **Tách biệt trách nhiệm (Separation of Concerns - SoC)**, giúp Controller chỉ lo xử lý request và response, còn logic phức tạp được đẩy về Service.

🔹 3. Service (`product.service.js`)

Chứa logic tạo sản phẩm, sử dụng **Factory Pattern** để tạo nhiều loại sản phẩm mà không cần viết nhiều `if-else`.

📌 ProductFactory (Factory Pattern)

```
class ProductFactory {
    static productRegistry = {}

    static registerProduct(type, product) {
        ProductFactory.productRegistry[type] = product
    }

    static async createProduct(type, payload) {
        const productClass = ProductFactory.productRegistry[type]
        if (!productClass)
           throw new BadRequestError('⛔🆘 Invalid product type')

        return new productClass(payload).createProduct()
    }
}
```

- **Lưu trữ danh sách các loại sản phẩm trong `productRegistry`.**
- **Dựa vào `product_type`, gọi class phù hợp để tạo sản phẩm.**
- **Dễ mở rộng**: Nếu có thêm loại sản phẩm mới, chỉ cần **đăng ký** thêm class vào `productRegistry`.

📌 Các lớp con (`Clothing`, `Electronics`, `Furniture`)

Mỗi class kế thừa từ `Product`, chỉ thay đổi cách lưu dữ liệu vào collection tương ứng.

```
class Clothing extends Product {
    async createProduct() {
        const newClothing = await clothing.create({
            ...this.product_attributes,
            product: this.product_shop
        })
        if (!newClothing)
           throw new BadRequestError('⛔🆘 Clothing not created')

        const newProduct = await super.createProduct(newClothing._id)
        if (!newProduct)
           throw new BadRequestError('⛔🆘 Product not created')
```

```
        return newProduct
    }
}
```

- **Tạo bản ghi trong collection `clothings`** để lưu thông tin riêng của quần áo.
- **Tạo bản ghi trong collection `products`**, liên kết với `clothings`.

Tại sao không lưu tất cả vào `products`?
➡️ **Tránh dư thừa dữ liệu**. Nếu mỗi sản phẩm có quá nhiều thuộc tính khác nhau, lưu hết vào một bảng sẽ **tốn bộ nhớ** và **gây khó khăn khi truy vấn**.

🔹 4. Middleware xác thực (`authUtils.js`)

Middleware này kiểm tra token trước khi cho phép user tạo sản phẩm.

```
const authentication = asyncHandler(async (req, res, next) => {
    const userId = req.headers[HEADER.CLIENT_ID];
    if(!userId) throw new UnauthorizedError('🥷 Unauthorized request
(Exc: 001) !!!');

    const keyStore = await findByUserId(userId);
    if(!keyStore) throw new NotFoundError('🥷 Key not found !!!');

    const accessToken = req.headers[HEADER.AUTHORIZATION];
    if(!accessToken) throw new UnauthorizedError('🥷 Unauthorized
request (Exc: 002) !!!');

    try {
        const decodedUser = jwt.verify(accessToken,
keyStore.publicKey);
        if(decodedUser.userId !== userId)
           throw new UnauthorizedError('🥷 Invalid userID !!!');

        req.keyStore = keyStore;
```

```
        req.user = decodedUser;
        return next();
    } catch (error) {
        throw new UnauthorizedError('🥷 Invalid accessToken !!!');
    }
});
```

- **Kiểm tra `client-id` trong request header** để xác định user.
- **Lấy khóa công khai của user từ database** để kiểm tra token.
- **Xác minh token bằng `jwt.verify()`**.
- **Lưu thông tin user vào `req.user`** để Controller có thể sử dụng.

Tại sao cần middleware này?
➡️ **Bảo vệ API**, chỉ cho phép người dùng đã đăng nhập thực hiện hành động.

3️⃣ Luồng xử lý của hệ thống

☑️ Khi một shop muốn tạo sản phẩm mới

1️⃣ Client gửi request:

```
POST /products/create
{
    "product_type": "Clothings",
    "product_name": "T-Shirt", "product_thumb": "url",
    "product_description": "Cool T-Shirt",
    "product_price": 100, "product_quantity": 10,
    "product_attributes": {
        "brand": "Nike", "size": "L",
        "material": "Cotton"
    }
}
```

2️⃣ **Middleware `authentication` kiểm tra token**, lấy `req.user.userId`.

3️⃣ **Controller gọi `ProductFactory.createProduct()`**, truyền loại sản phẩm và dữ liệu.

4️⃣ **`ProductFactory` gọi class tương ứng (`Clothing`) để tạo sản phẩm**.

5️⃣ **Dữ liệu được lưu vào database**:

- Lưu thông tin chung vào collection `products`.
- Lưu thông tin riêng vào collection `clothings`.

6️⃣ **Gửi phản hồi về client**.

4️⃣ Tóm tắt

- **Sử dụng Factory Pattern** giúp dễ mở rộng loại sản phẩm mới.
- **Tách biệt Model, Controller, Service** để code dễ bảo trì.
- **Sử dụng middleware `authentication`** để bảo vệ API.
- **Lưu trữ dữ liệu tách biệt** giữa `products` và các loại sản phẩm con để tối ưu database.