1. Tổng quan tính năng

Tính năng **Update Product** cho phép sửa đổi thông tin sản phẩm, bao gồm cả:

- Thuộc tính chung (ví dụ: `product_name`, `product_price`) cập nhật ở collection `product`.
- Thuộc tính đặc thù (ví dụ: `manufacturer`, `model` cho Electronics) cập nhật ở collection con (ví dụ: `electronics`).

Cơ chế triển khai:

- **Controller** gọi đến **ProductFactory.updateProduct**.
- **ProductFactory** chọn class con phù hợp (Clothing, Electronics, Furniture,...) dựa trên trường `product_type`.
- **Class con** thực hiện cập nhật riêng cho phần thuộc tính đặc thù. Sau đó gọi hàm của lớp cha (`super.updateProduct`) để cập nhật các trường chung.

---

2. Luồng xử lý khi gọi update

Giả sử có request `PATCH /update/:id` (hoặc `PUT /update/:id` tuỳ thiết kế API).

1. **(🔑 Nhận request)**
   - Controller `ProductController.updateProduct` nhận `req.body` và `req.params.id`.
   - Trong `req.body` bắt buộc phải có `product_type` để xác định loại sản phẩm.
2. **(⚙️ Controller → ProductFactory)**
   - Gọi `ProductFactory.updateProduct(req.body.product_type, req.params.id, payload)`.
   - `payload` chứa dữ liệu cập nhật (VD: `product_name`, `product_price`, `product_attributes`,...).
3. **(🏭 ProductFactory.updateProduct)**
   - Lấy class con (Clothing/Electronics/Furniture) tương ứng với `type`.
   - Tạo instance: `new Electronics(payload)` (nếu `type = Electronics`).
   - Gọi `instance.updateProduct(productId)`.
4. **(🔧 Class con xử lý - Electronics)**
   - **Loại bỏ giá trị undefined** bằng hàm `removeUndefinedObject(this)`.
   - Kiểm tra xem trong payload có `product_attributes` không:

        - Nếu có, gọi `updateProductById` để cập nhật vào collection con (`electronics`) bằng **dot notation**.
    - Sau đó, gọi `super.updateProduct(productId, ...)` để cập nhật tiếp bảng cha (`product`).
5. **(✅ Hoàn tất)**
    - Dữ liệu cập nhật thành công, Controller trả về response xác nhận.

Ghi chú: Tuỳ từng class (Clothing, Electronics, Furniture), việc ghi vào collection con sẽ khác nhau.

---

3. Chi tiết các hàm liên quan

3.1. `updateProductById` (trong `product.repo.js`)

```
const updateProductById = async ({ product_id, bodyUpdate, model, isNew = true }) => {
  return await model.findByIdAndUpdate(product_id, bodyUpdate, {
     new: isNew })
}
```

- **Chức năng**: Nhận `product_id`, data cập nhật `bodyUpdate` và thực hiện `findByIdAndUpdate` trên model tương ứng (`product`, `electronics`, `clothing`,...).
- **Tham số**:
    - `product_id`: ID document trong MongoDB.
    - `bodyUpdate`: object chứa thông tin cập nhật (có thể đã được flatten).
    - `model`: model tương ứng (`product`, `electronics`,...).

3.2. `removeUndefinedObject(object)`

```
const removeUndefinedObject = object => {
  Object.keys(object).forEach(key => {
    if (object[key] == null) {
      delete object[key];
    }
  })
  return object;
```

```
}
```

- **Mục đích**: Xoá các key có giá trị `null` hoặc `undefined` trong object.

Ví dụ:

```
// Trước
{ name: "iPhone", price: undefined, model: null }
// Sau khi removeUndefinedObject
{ name: "iPhone" }
```

-

3.3. `updateNestedObjectParser(obj)`

```
const updateNestedObjectParser = obj => {
  const final = {}
  Object.keys(obj || {}).forEach(key => {
    if (!Array.isArray(obj[key]) && typeof obj[key] === 'object') {
      const response = updateNestedObjectParser(obj[key])
      Object.keys(response || {}).forEach(a => {
        final[`${key}.${a}`] = response[a]
      })
    } else {
      final[key] = obj[key]
    }
  })
  return final
}
```

- **Mục đích**: “Trải phẳng” (flatten) cấu trúc lồng nhau thành **dot notation** phù hợp với MongoDB.
- **Cơ chế**: Lần lượt duyệt các key, nếu value là object con thì gọi đệ quy, còn không thì gán trực tiếp.

Ví dụ:

```
const input = {
  product_name: "Laptop ASUS",
  product_attributes: {
    manufacturer: "ASUS",
    info: { series: "ZenBook", year: 2022 }
  }
}
updateNestedObjectParser(input.product_attributes)
// Kết quả:
// {
//   "manufacturer": "ASUS",
//   "info.series": "ZenBook",
//   "info.year": 2022
// }
```

---

4. Ví dụ request và cách sử dụng

4.1. Request ví dụ

```
PATCH /update/67a55c4255cddcd441628e2c
Content-Type: application/json
x-api-key: ...
x-client-id: ...
authorization: ...

{
  "product_attributes": {
    "manufacturer": "Apple",
    "model": "iPhone 13 Pro VN/A"
  },
  "product_type": "Electronics",
  "product_name": "iPhone 13 Pro",
  "product_price": 15000000
}
```

- **product_type**: `Electronics` → lớp `Electronics` sẽ được sử dụng.

- **product_attributes**: thông tin đặc thù (như `manufacturer`, `model`) sẽ ghi vào collection `electronics` sau khi được flatten.
- **product_name** & `product_price`: thông tin chung, cập nhật vào collection `product`.

4.2. Kết quả

- **Trên collection `electronics`**:
  - `manufacturer` và `model` được cập nhật (sử dụng dot notation).
- **Trên collection `product`**:
  - `product_name` = `"iPhone 13 Pro"`,
  - `product_price` = `15000000`.

4.3. Lưu ý

1. **Truyền đúng tham số** `product_id` khi gọi `updateProductById`, tránh nhầm thành `productId`.
2. **Hàm `updateNestedObjectParser` phải có `return final`** để trả về object đã flatten.
3. **Kiểm tra mã lỗi** (nếu có). Nếu không tìm thấy document tương ứng, server có thể trả về `404`.

🎉 Kết luận

Với thiết kế **Factory Pattern**, ta có thể quản lý tốt các loại sản phẩm khác nhau (Clothing, Electronics, Furniture,...) và logic cập nhật riêng của mỗi loại. Hệ thống đảm bảo:

- Tách biệt logic cập nhật của từng loại sản phẩm.
- Tận dụng chung flow “remove undefined → flatten object → update 2 nơi: collection con & bảng cha”.
- Dễ mở rộng hoặc sửa đổi trong tương lai.