I. Vấn đề hiện tại

- Với mỗi sản phẩm mới được thêm vào, phương thức tạo mới sản phẩm sẽ thực hiện các tác vụ tuần tự như sau:
  - + Thêm mới hình ảnh sản phẩm - lưu hình ảnh vào mã nguồn
  - + Gán sản phẩm vào các chi nhánh (nếu có) - tạo bản ghi ProductStock giữa sản phẩm và chi nhánh
  - + Gán danh mục vào sản phẩm - tạo bản ghi ProductCategory giữa danh mục và sản phẩm
  - + Gán danh mục con vào sản phẩm - tạo bản ghi ProductCategoryChild giữa danh mục con và sản phẩm
  - + Tạo biến thể sản phẩm (nếu có) - tạo mới bản ghi Variant (lưu giá biến thể, số lượng tồn kho) và VariantValue (thuộc tính phân loại của biến thể đó)
  - + Cuối cùng là tạo sản phẩm
- Mỗi tác vụ mà phương thức tạo sản phẩm bên trên đang thực hiện tuần tự tức khi thêm mới sản phẩm thì trước tiên là thêm hình ảnh vào server -> xong mới thêm bản ghi chi nhánh -> xong mới thêm danh mục sản phẩm… Phương thức cập nhật sản phẩm tương tự.

```
2 references
public async Task<int> CreateAsync(ProductRequest dto)
{
    Product product = mapper.Map<Product>(dto);
    product.Description = dto.Description ?? string.Empty;
    product.Images = await ProcessUpload(dto.Images);

    // add list product-branch
    if (dto.BranchIds.Any())
    {
        await branchService.AssignProductToBranches(product.Id, dto.BranchIds);
    }

    // add list product-category
    if (dto.CategoryIds.Any())
    {
        await categoryService.AssignProductToCategories(product.Id, dto.CategoryIds);
    }

    // add list product-category-child
    if (dto.CategoryIds.Any())
    {
        await categoryService.AddOrUpdateCategoryChildByProduct(product.Id, dto.CategoryChildIds);
    }

    // thêm mới variants cho product
    if (dto.Variants.Any())
    {
        await productPropertyService.CreateProductVariants(product.Id, dto.Variants);
    }

    return await base.CreateAsync(product);
}
```

- Vấn đề xảy ra nếu 1 trong các tác vụ bị lỗi, ví dụ khi đã thêm tất cả các bản ghi của sản phẩm vào Db rồi nhưng bước cuối cùng thêm sản phẩm xảy ra lỗi thì sản phẩm không được thêm -> nhưng các bản ghi ProductStock, ProductCategory, ProductCategoryChild… vẫn được lưu vào Db trước đó, kể cả hình ảnh cũng được lưu vào server mặc dù sản phẩm chính bản chất không tạo thành công.

```
4 references
public async Task<int> AssignProductToBranches(string productId, List<string> branchIds)
{
    // Get the repository for ProductBranch
    var productBranchRepo = unitOfWork.GetRepository<ProductStock>();

    // Get existing product-branch relationships for this product
    var existingBranches = await productBranchRepo.AsQueryable()
        .Where(pb => pb.ProductId == productId)
        .ToListAsync();

    // Identify branches to remove (exists in DB but not in the new list)
    var branchesToRemove = existingBranches
        .Where(pb => !branchIds.Contains(pb.BranchId))
        .ToList();

    // Identify branches to add (exists in the new list but not in DB)
    var existingBranchIds = existingBranches.Select(pb => pb.BranchId).ToList();
    var branchesToAdd = branchIds
        .Where(branchId => !existingBranchIds.Contains(branchId))
        .Select(branchId => new ProductStock
        {
            ProductId = productId,
            BranchId = branchId
        })
        .ToList();

    // Remove branches that are no longer associated
    if (branchesToRemove.Any())
    {
        productBranchRepo.DeleteRange(branchesToRemove);
    }

    // Add new branch associations
    if (branchesToAdd.Any())
    {
        productBranchRepo.AddRange(branchesToAdd);
    }

    // Save changes and return the total number of affected records
    await unitOfWork.SaveChangesAsync();
    return branchesToRemove.Count + branchesToAdd.Count;
}
```

=> Tóm lại:

- Phương thức thực hiện nhiều thao tác liên quan đến cơ sở dữ liệu (lưu hình ảnh, gán chi nhánh, danh mục, danh mục con, và biến thể). Nếu một thao tác thất bại (ví dụ: lỗi khi lưu biến thể), các thao tác trước đó (như gán chi nhánh, danh mục hoặc lưu hình ảnh) có thể đã được thực hiện và lưu vào Db, dẫn đến trạng thái không nhất quán trong cơ sở dữ liệu, lãng phí tài nguyên khi lưu dữ liệu rác nếu có lỗi xảy ra.
- Về mặt hiệu suất, các tác vụ thực hiện tuần tự, lưu xong các bản ghi này rồi tiếp tục lưu các bản ghi khác vào Db => lãng phí thời gian xử lý I/O.

II. Giải pháp

- Tận dụng lớp UnitOfWork, hiện tại có các phương thức hữu ích chưa được sử dụng.

```
// Bắt đầu transaction
2 references
public async Task BeginTransactionAsync()
{
    if (_transaction == null)
    {
        _transaction = await context.Database.BeginTransactionAsync();
    }
}

// Xác nhận transaction
2 references
public async Task<int> CommitAsync(CancellationToken cancellationToken = default)
{
    if (_transaction != null)
    {
        var result = await context.SaveChangesAsync(cancellationToken);
        await _transaction.CommitAsync();
        await _transaction.DisposeAsync();
        _transaction = null;
        return result;
    }
    return 0; // Không có transaction để commit
}

// Hủy transaction nếu có lỗi
2 references
public async Task RollbackAsync()
{
    if (_transaction != null)
    {
        await _transaction.RollbackAsync();
        await _transaction.DisposeAsync();
        _transaction = null;
    }
}
```

- Trong đó:
  + BeginTransactionAsync: Khởi tạo một transaction thủ công thông qua EF Core, đảm bảo mọi thao tác sau đó đều nằm trong cùng một giao dịch – chỉ khi gọi CommitAsync() thì dữ liệu mới thực sự được lưu.
  + CommitAsync: Ghi toàn bộ thay đổi được theo xuống database (SaveChangesAsync), sau đó Commit transaction và Dispose để giải phóng tài nguyên.
  + RollbackAsync: Hủy bỏ toàn bộ thay đổi đang chờ commit trong transaction hiện tại khi có lỗi giữa chừng.
- Áp dụng vào tạo sản phẩm:

```
0 references
public async Task<int> CreateAsyncV2(ProductRequest dto)
{
    await unitOfWork.BeginTransactionAsync();

    Product product = mapper.Map<Product>(dto);
    product.Description = dto.Description ?? string.Empty;

    try
    {
        product.Images = await ProcessUpload(dto.Images);

        // add list product-branch
        if (dto.BranchIds.Any())
        {
            await branchService.AssignProductToBranches(product.Id, dto.BranchIds);
        }

        // add list product-category
        if (dto.CategoryIds.Any())
        {
            await categoryService.AssignProductToCategories(product.Id, dto.CategoryIds);
        }

        // add list product-category-child
        if (dto.CategoryIds.Any())
        {
            await categoryService.AddOrUpdateCategoryChildByProduct(product.Id, dto.CategoryChildIds);
        }

        // thêm mới variants cho product
        if (dto.Variants.Any())
        {
            await productPropertyService.CreateProductVariants(product.Id, dto.Variants);
        }

        //return await base.CreateAsync(product);

        _repository.Add(product);

        return await unitOfWork.CommitAsync();
    }
    catch(Exception err)
    {
        await unitOfWork.RollbackAsync();

        // Xóa hình ảnh đã lưu
        if (!string.IsNullOrEmpty(product.Images)) RemoveOldImage(product.Images);

        throw;
    }
}
```

- Ta sẽ đưa các thao tác vào một transaction, đảm bảo không có lỗi xảy ra thì mới commit transaction đó xuống SQLServer.

III. Ý nghĩa

- Duy trì toàn vẹn dữ liệu, tránh ghi từng phần nếu có lỗi giữa chừng.
- Hỗ trợ rollback an toàn, giúp code ổn định, dễ bảo trì, dễ debug hơn khi cần debug Create, Update hay Delete.
- Tăng khả năng mở rộng cho phương thức, sau này nếu thêm 5-10 bước xử lý cũng không lo bị mất kiểm soát.
- Ngoài ra cũng có thể xem là rule của BE cần được tuân thủ.

Hiện tại, không chỉ có thêm, xóa, sửa sản phẩm là cần áp dụng giải pháp trên, tin tức, khảo sát… cũng đang xử lý từng tác vụ nhỏ. Về mặc tích cực phương pháp trên có thể giảm tải cho hệ thống khi phải xử lý nhiều request, mỗi request thực hiện nhiều tác vụ như vậy.