[아키텍쳐]패키지 구조: 계층형 VS 도메인형?

> 프로젝트 주제 + 전반적인 레이아웃을 확정짓고,
>
> 웹 애플리케이션을 구현하기 위해 백앤드 팀은 **프로젝트 패키지 구조**에 대해서 고민하기로 했다.
>
> 팀 회의 전, 미리 두가지 구조에 대한 장단점을 비교한 후 회의에 참여해보려 한다.

1. 계층형 구조

애플리케이션에서 사용하는 계층별로 패키지를 구성

대부분이 Layered Architecture를 사용, 컴포넌트들 및 관련 요소들이 패키지가 되는 경우가 많다.

- Presentation Layer (UI/컨트롤러)
- Business Layer (서비스)
- Persistence Layer (리포지토리)
- Database Layer (데이터베이스)

계층형 구조 예시

```
controller
     ├ ProductController
     ├ MemberController
     ├ CartController

service
     ├ ProductService
     ├ MemberService
     ├ CartService

dao
     ├ ProductDao
     ├ MemberDao
     ├ CartDao

domain
     ├ Product
     ├ Member
     ├ Cart
```

1-1. 계층형 구조의 장단점

- **장점**
  - 1. 프로젝트 전반적인 이해도가 낮아도, 패키지 구조만 보고 전체적인 구조를 파악할 수 있다.
    - 애플리케이션의 API를 보고 흐름을 파악하고 싶다면, Controller 패키지 하나만 보고 파악할 수 있다.
    - 애플리케이션의 비즈니스 로직을 보고 싶다면, Service 패키지 하나만 보고 파악할 수 있다.
  - 2. 계층별 응집도가 높아진다.
    - 계층별 수정이 일어날 때, 하나의 패키지만 보면 된다.

- **단점**
  - 1. 도메인별 응집도가 낮다. (패키지로 애플리케이션의 기능을 구분짓지 못한다)
    - 1-1. 도메인의 흐름을 파악하기 힘들다.
      - Product 도메인의 흐름을 보고 싶을 때, 모든 계층 패키지를 봐야한다.
      - 하나의 패키지안에 여러 도메인(상품, 장바구니, 사용자)들이 섞여 있다.
    - 1-2. 도메인과 관련된 스펙 & 기능이 변경되었을 때, 변경 범위가 크다.
      - Product에 대한 변경점이 있을 때, 여러 패키지에서 변경이 일어난다. (단순 기능에도 모든 패키지를 거쳐야 한다.)
  - 2. 유스케이스(사용자의 행위) 표현이 어렵다.
    - 규모가 커지면, 유스케이스별로 클래스를 분리할 때가 있다.
    - ex : 상품 등록 유스 케이스 -> RegisterProductService
    - 하지만, 계층형에서는 계층으로 패키지가 묶이기 때문에 위와 같이 네이밍해서 분리하기 어렵다.
  - 3. 규모가 커지면 하나의 패키지 안에 여러 클래스들이 모여서 구분이 어려워진다.

2. 도메인형 구조 (Domain-Driven Design)

💡 도메인을 기준으로 패키지를 나눈 구조

도메인형 구조 예시

```
product
     ├ controller
     ├ service
     ├ dao
     ├ dto

member
     ├ controller
     ├ service
     ├ dao
     ├ dto

cart
     ├ controller
     ├ service
     ├ dao
     ├ dto
```

2-1. 도메인형 구조의 장단점

- **장점**
  - 1. 도메인별 응집도가 높아진다.
    - 1-1. 도메인의 흐름을 파악하기 쉽다.
      - Product 도메인의 흐름을 보고 싶을 때, Product 패키지 하나만 보면 된다.
    - 1-2. 도메인과 관련된 스펙 & 기능이 변경되었을 때, 변경 범위가 적다.
      - Product에 대한 변경점이 있을 때, Product 패키지만 변경이 일어난다. (비즈니스 프로세스 변경에 빠르게 대응 가능)

    - 2. 유스케이스별로 세분화해서 표현이 가능하다.
        - ex : 상품 등록 유스 케이스 -> RegisterProductService
        - ex : 상품 검색 유스 케이스 -> SearchProductService
        - 도메인별로 패키지가 나뉘기 때문에 product 패키지에서 위와 같은 네이밍으로 분리할 수 있다.
- **단점**
    - 애플리케이션의 전반적인 흐름을 한눈에 파악하기가 어렵다.
        - 흐름을 파악하기 위해 여러 패키지를 왔다갔다 해야할 가능성이 높다.
    - 개발자의 관점에 따라 어느 패키지에 둘지 애매한 클래스들이 존재한다.
        - Welcome 페이지를 맵핑하는 컨트롤러일 때, 어느 도메인 패키지에 위치할지 개발자에 따라 다를 수 있다.
        - 자신이 예상하는 패키지와 다를 때, 해당 클래스를 찾기가 어렵다.

3. 결론

팀의 프로젝트 규모가 그렇게 크지 않고

단순 CRUD 중심의 애플리케이션에서는 **계층형 구조를 선택하여**

구조의 가독성을 높이며 직관적으로 파악할 수 있도록 설계하는 것이 좋다고 생각을 한다!