

1장. 계층형 아키텍처의 문제는 무엇일까?

계층형 아키텍처

3계층 아키텍처

장점

단점

데이터베이스 주도 설계를 유도하는 계층형 아키텍처

🌳 왜 **도메인 로직** 이 아닌 **데이터베이스** 를 토대로 아키텍처를 설계하는가?

비즈니스 관점

계층형 아키텍처 관점

🌳 지름길이 만든 비대한 영속성 계층

계층형 아키텍처의 규칙

😓 상위 컴포넌트에 접근하고 싶다면?

영속성 계층

지름길 끄기

🌳 어려워진 테스트

도메인 로직을 웹 계층에 구현하게 된다.

웹 계층 테스트에서 영속성 계층도 mocking해야 한다.

🌳 숨겨지는 유스케이스

아키텍처는 코드를 빠르게 탐색하는 데 도움이 돼야 한다.

🌳 동시 작업 불가

결론

계층형 아키텍처

3계층 아키텍처

- 웹 : 요청을 받는다.
- 도메인 : 비즈니스 로직을 수행한다.
- 영속성 : 도메인 엔티티의 현재 상태를 조회하거나 변경한다.

장점

- 계층을 잘 이해하고 구성한다면 계층 간 독립적으로 로직을 작성할 수 있다.


- 잘 만들어진 계층형 아키텍처는 선택의 폭을 넓히고 변화하는 요구사항과 외부 요인에 빠르게 적응할 수 있게 해준다.

단점

- 코드에 나쁜 습관들이 스며들기 쉽게 만든다.
- 시간이 지날수록 소프트웨어를 점점 더 변경하기 어렵게 만든다.

데이터베이스 주도 설계를 유도하는 계층형 아키텍처

| 모든 것이 영속성 계층을 토대로 만들어진다는 ...

 왜 **도메인 로직** 이 아닌 **데이터베이스** 를 토대로 아키텍처를 설계하는가?

비즈니스 관점

- 도메인 로직이 중요
- 상태가 아니라 행동을 중심으로 모델링
- 상태도 중요하지만 **행동이 상태를 바꾸는 주체**다.

계층형 아키텍처 관점

- 웹은 도메인에, 도메인은 영속성에 의존
 - 영속성 계층을 기반으로 아키텍처 설계 → 데이터베이스 주도 설계 !?
- ORM 프레임워크를 계층형 아키텍처와 결합하면 비즈니스 규칙을 영속성 관점과 섞고 싶게 된다.
- 도메인 계층과 영속성 계층에 강한 결합이 생긴다.

지름길이 만든 비대한 영속성 계층

계층형 아키텍처의 규칙

- 같은 계층에 있는 컴포넌트나 아래에 있는 계층에만 접근할 수 있다.

🤔 상위 컴포넌트에 접근하고 싶다면?

- **지름길** : 컴포넌트를 계층 아래로 내려버린다. 💔

영속성 계층

- 컴포넌트를 아래 계층으로 내릴수록 비대해진다.
- 어떤 계층에도 속하지 않는 것처럼 보이는 컴포넌트(헬퍼, 유틸리티 등)이 큰 후보다.

지름길 끄기

- 아키텍처 규칙을 **강제**하지 않는 한 불가
 - 규칙이 깨졌을 때 빌드가 실패하는 것

🌳 어려워진 테스트

👤 : 필드 하나만 바꾸면 되는데 웹에서 영속성 계층에 바로 접근하는 게 좋지 않을까?

도메인 로직을 웹 계층에 구현하게 된다.

- 유스케이스가 확장될수록 더 많은 도메인 로직을 웹 계층에 추가할 확률이 높다.
- 애플리케이션 전반에 걸쳐 책임이 섞인다.

웹 계층 테스트에서 영속성 계층도 mocking해야 한다.

- 단위 테스트의 복잡도가 올라간다.
- 테스트 설정이 복잡해진다.

🌳숨겨지는 유스케이스

아키텍처는 코드를 빠르게 탐색하는 데 도움이 돼야 한다.

- 기능을 추가하거나 변경할 위치를 찾는 일이 빈번하다.
- 레거시 뿐만 아니라 신규 프로젝트에서도 마찬가지다.
- 계층형 아키텍처는 도메인 서비스의 너비에 대한 규칙이 없다.
 - 아주 방대해질지도 ?
 - 아주 많은 의존성을 가질지도 ?

- 웹 계층의 컴포넌트들이 죄다 의존할지도 ?

동시 작업 불가

- 특정 기능은 한 명의 개발자만 작업할 수 있다.
 - 영속성 먼저 개발해야 하니까 !
- 넓은 서비스가 있다면 서로 다른 기능을 작업하기도 어렵다.

결론

계층형 아키텍처의 함정을 염두에 두면 유지보수하기 쉬운 코드를 작성할 수 있다.

크게 쉽나 ...