앱 아키텍처

🏛️ 앱 아키텍처 **by @kiboom**

애플리케이션 역할

▼ `데이터` 👉 `가공` 👉 `화면`

- `데이터`를 `가공`해서 `화면`에 보여주는 것

아키텍처는 왜 생겼을까?

> 🌀 **제행무상:** 세상은 항상 돌고 변하며 잠시도 한 모양으로 머무르지 않는다.

▼ **세상에는 변화가 너무 많다.** 😂

- `사용자 요구`도 바뀌고, `스펙`도 바뀌고, `스펙아웃` 도 되고
- `로직`도 바뀌고, `멤버`도 바뀌고

▼ **변화에 '잘' 대응하기 위해서는 좋은 아키텍처가 필요!**

1. 빠르게 대처할 수 있어야 함. **(유지보수성, 개발효율성)**
2. 안전하게 대처할 수 있어야 함. **(테스트 용이성)**

3. 이해하기 쉬워야 함. **(학습 용이성, 가독성)**

아키텍처의 기본 원리

“레이어는 개인주의야. 혼자 밖에 생각하지 않아.” (~~이기주의~~ x)

▼ **변화에 ‘잘' 대응하려면 ‘`레이어`'를 나누자!**

1. 역할 별로 **레이어를 나눈다**.
2. 각 레이어는 **각자의 역할에만 집중**한다.
    - 역할 밖의 일은 **철저히 신경 끈다.**
    - **받아야 할 데이터**와 **줘야 할 데이터**에만 집중한다.
3. 레이어끼리는 **정해진 약속**(프로토콜)에 의해서만 데이터를 주고 받는다.
    - 정해진 약속만 지켜진다면,
      전달 받은 데이터가 어떻게 만들어져서 왔는지,
      전달 할 데이터를 누가 받게 되는지는 철저히 신경 끈다.

▼ **레이어가 이렇게 개인주의적이면,**

- 앱을 변경할 때 필요한 부분만 바꾸면 된다.
- 필요한 부분만 바꾸더라도 다른 곳에는 영향을 주지 않는다.

▼ **예시) 정육점**

- 소비자는 정육점을 통해서 고기를 구매한다.
- 정육점이 어느 축산업체로부터 고기를 가져와 가공해서 소비자에게 판매한다.
  축산업체가 바뀌어도 고기 등급만 괜찮으면 거래는 이뤄진다.
- 소비자는 축산업체를 고르지 않는다. 축산업체를 선정하는건 정육점이 ‘**대신**’ 해준다.

즉, 소비자는 축산업체 선정을 '**주입**' 받았다. (`Dependency Injection`)

- 덕분에, 소비자는 축산업체가 바뀌거나 망해도 영향을 받지 않는다.

▼ **예시) API 환경 변경**

- **또스** 주식 화면은 `종목 목록 데이터`를 필요로 한다.
- 해당 데이터를 가져오기 위해서는 `개발용 서버 API` `리얼용 서버 API` 가 있지만, 현황 화면은 어디서 데이터가 왔는지는 중요하지 않다.
- 어떤 서버 API 를 사용하는지는 네트워크 클래스가 **flavor**를 통해 '**대신**' 정해준다. 즉, 주식 화면은 네트워크 클래스를 통해 API 를 '**주입**' 받았다. (`Dependency Injection`)
- 덕분에, 서버 환경이 바뀌더라도 현황 화면은 잘만 동작한다.

아키텍처의 종류

▼ **MVC, MVP, MVVM, VIPER ...**
엄청나게 많지만.. 결국에는 원리는 똑같다!

- 역할 별로 레이어를 나누고,
- 각 레이어는 각자의 역할에만 집중함으로써,
- 각 레이어를 **수정•테스트**하기 쉽게 만든다.

▼ **Clean Architecture**

- 여러 플랫폼에 범용적으로 적용할 수 있는 아키텍처!
- 레이어를 역할에 따라 철저하게 구분하였으며,
- 프로젝트의 성격에 맞게 레이어 조절 가능.