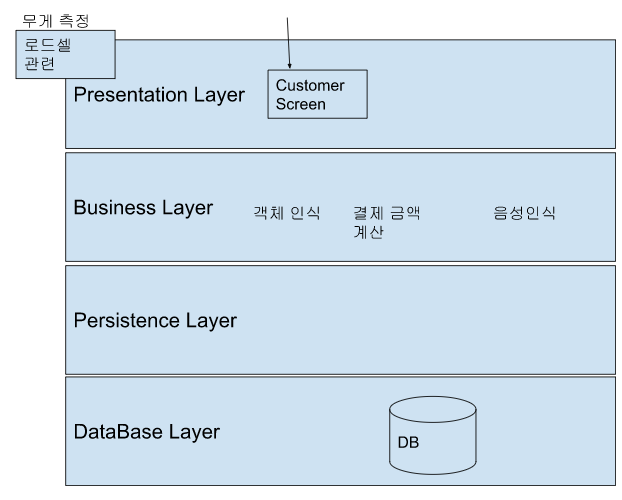
오후

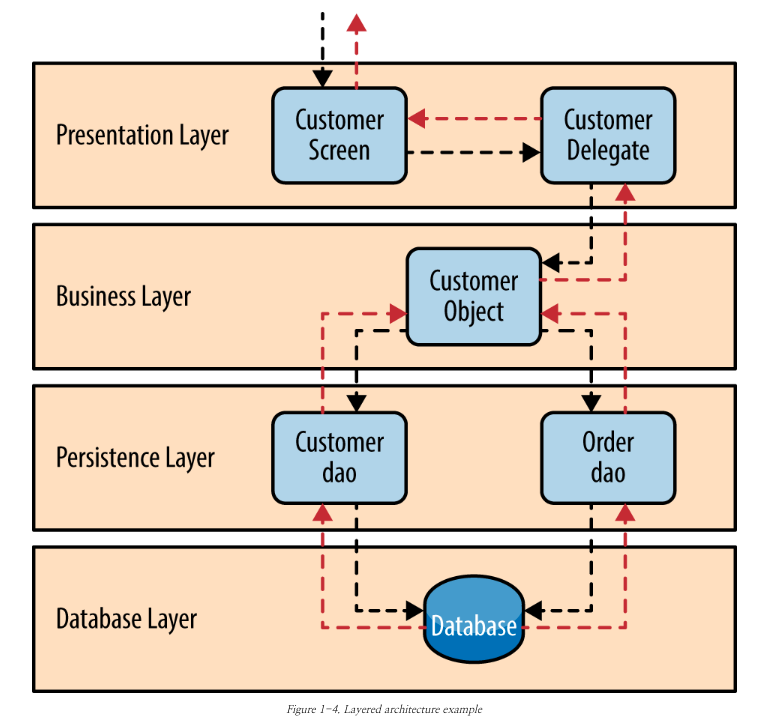


​

구조 패턴만 보고 만들기가 어려워서 실제 예시 보고 다시 만들어 보기.

​

참고 자료:



[1. Layered Architecture - Software Architecture Patterns [Book]](https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html)

Presentation Layer에는 사용자 인터페이스나 브라우저 통신 등을 처리한다. Business Layer에서는 실제적인 서비스를 구현하는 비즈니스 로직이 있고, Persistence Layer는 데이터베이스에 객체를 매핑시키는 역할을 하고, Database Layer에서는 이를 처리한다.

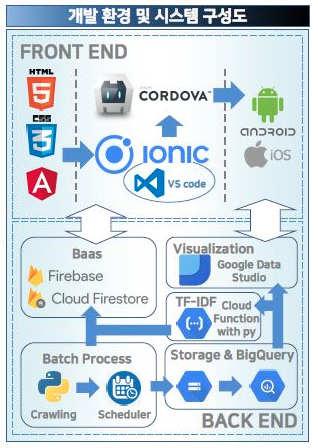
이 구조의 핵심은 수행하는 동작을 분리시키고, 하위 레이어에만 의존하도록 하는 것이다. 예를 들면 Presentation Layer는 하위 레이어인 Business Layer에만 의존하여 동작한다. 그 아래 계층인 Persistence Layer에서 무슨 처리를 하는지 알 수 없으며, 변화가 생기더라도 문제가 없다.

이 구조의 장점은 우선 각 계층이 확실하게 분리되어 있기 때문에 테스트 하기가 쉽다는 점이다. 각 계층을 유연하게 변경할 수 있다. 예를 들면 가장 상위 계층인 Presentation Layer를 변경하면 시스템 전체를 수정하지 않고도 웹에서 동작하던 것을 앱에서 동작하게 할 수도 있다.

하지만 한 기능을 수행하기 위해서 여러 계층을 거쳐서 동작하기 때문에 성능이 낮고, 가장 하위 계층인 데이터 베이스에 의존적이라는 단점이 있다.

<https://velog.io/@pond1029/software-architecture>

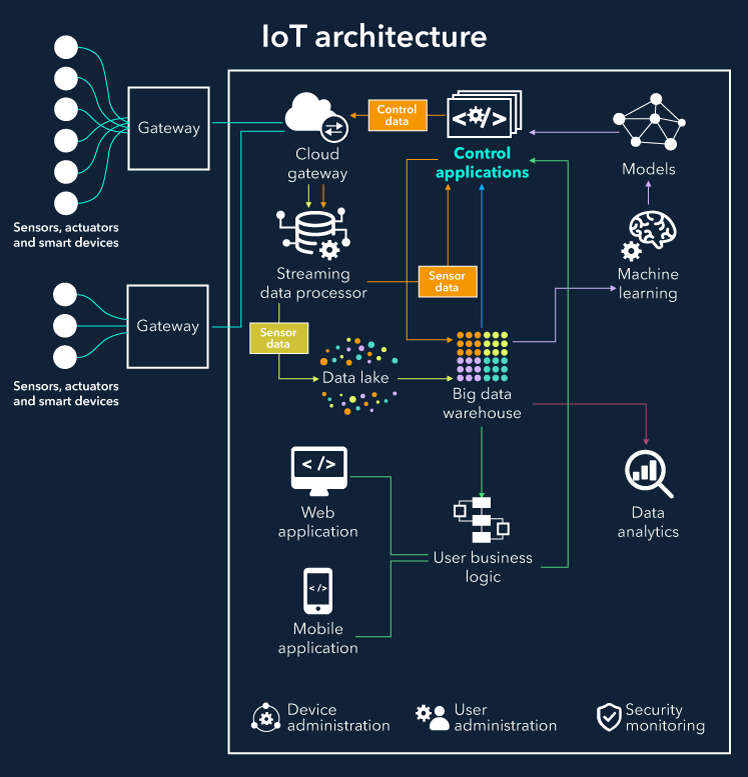
​



<https://camo.githubusercontent.com/928cce4c82a243050c36655587d851f85341723733ba90ecfc12200fad43992b/68747470733a2f2f696d6775722e636f6d2f4d4c34666867532e706e67>

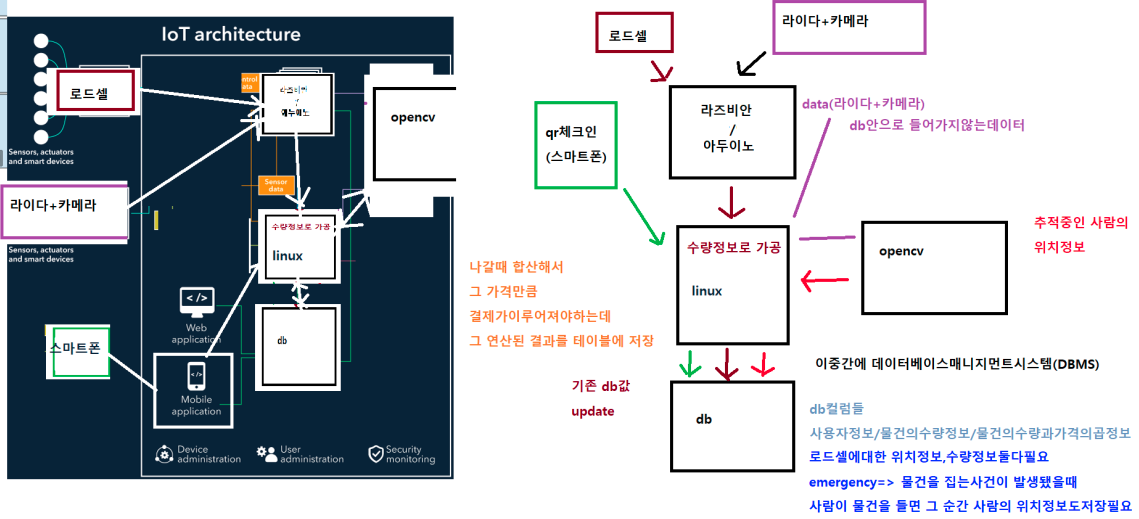
​

​



<https://www.scnsoft.com/blog/iot-architecture-in-a-nutshell-and-how-it-works>

​



​