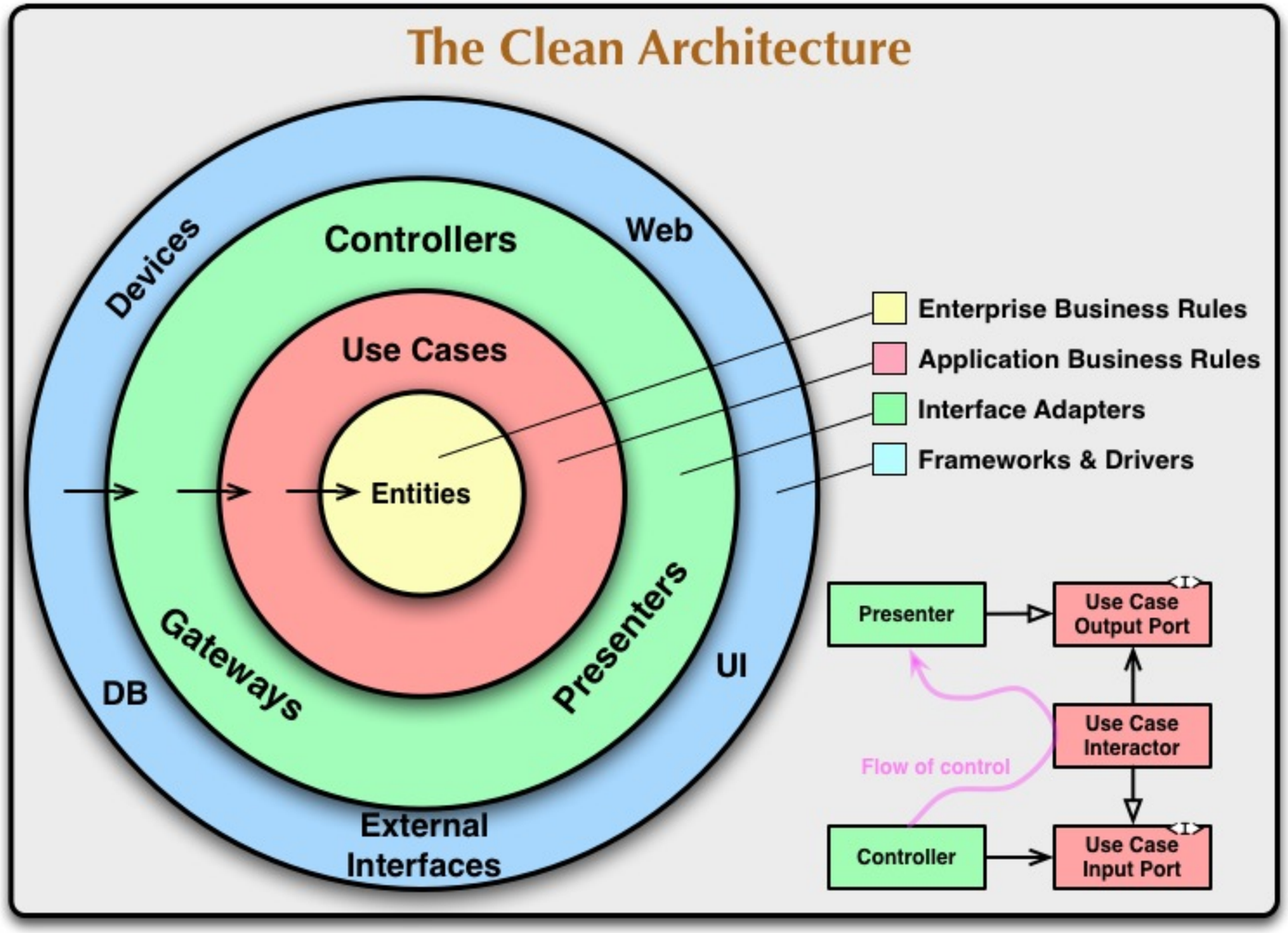
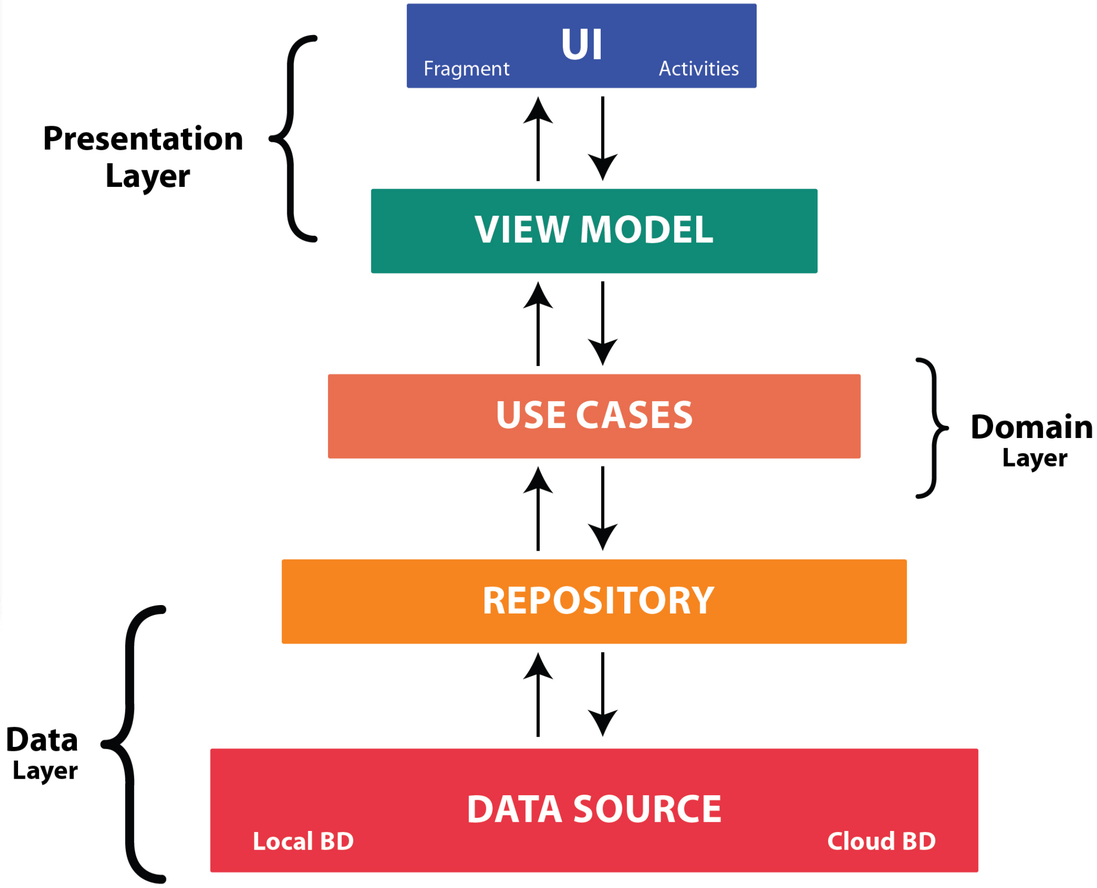
플러터 다트 아키텍처 패턴

5. Clean Architecture

클린 아키텍처란?

- 데이터 중심의 전통적인 계층형 아키텍처에서 벗어나 의존성의 방향을 저수준에서 고수준으로 단방향화시켜 변화, 확장, 테스트 등에 강점을 갖는 아키텍처
- 클린아키텍처 또한 계층형 아키텍처의 일종. 다만 의존성 방향이 다름
- 크게 프레젠테이션 레이어, 데이터 레이어, 도메인 레이어 총 3가지 레이어로 구성됨
- 이전까지 배웠던 아키텍처 패턴이 프레젠테이션 레이어(UI)에 관한 것이었다면 클린 아키텍처는 서비스 전체를 아우르는 아키텍처이기 때문에 규모가 훨씬 큰 개념


The Clean Architecture
Devices
Controllers
Web
Use Cases
Entities
Gateways
Presenters
DB
UI
External
Interfaces
Enterprise Business Rules
Application Business Rules
Interface Adapters
Frameworks & Drivers
Presenter
Use Case
Output Port
Use Case
Interactor
Flow of control
Controller
Use Case
Input Port


UI
Fragment
Activities
Presentation
Layer
VIEW MODEL
USE CASES
Domain
Layer
REPOSITORY
Data
Layer
DATA SOURCE
Local BD
Cloud BD

클린 아키텍처의 장단점

- 현대 개발 패러다임에서 가장 중요시하는 거의 모든 것들이 다 고려된 아키텍처로 객체 및 레이어 간 의존성을 낮추어 변화, 확장, 테스트 등에 유연함

- 단순히 보일러플레이트 코드가 많다고 할 정도가 아니라 모든 것을 다 쪼개서 구성하는 경우가 많고 특히 도메인 간 통신의 경우 엄격하게 지킬 때는 코드의 수가 기하급수적으로 높아짐
- 러닝커브가 높은 것은 물론이고 팀 내에서 제대로 이해하지 못한 사람이 있을 경우 제대로 지켜지기가 사실상 불가능
- 처음 소개될 때부터 추상적인 이론이었으며 실무에서 베스트 프랙티스가 존재하지 않아 사용하는 사람이 정의하기 나름 - 정답이 존재하지 않음

플러터와 클린 아키텍처

- 클린 아키텍처는 언어와 프레임워크에 관계 없이 적용할 수 있는 이론
- 요즘 플러터 생태계가 성숙해지면서 플러터+클린 아키텍처를 적용한 프로젝트들이 많아지고 있다고 함(예: 중고나라)
- 다만 클린 아키텍처를 적용 중인 팀은 보통 이미 클린 아키텍처를 알고 경험해본 팀으로서 초보자가 많은 플러터의 특성 상 제대로 도입하기가 쉽지는 않을 것이라고 생각됨

코드 공유 시간

의존성 주입 패키지 (get it 등)을 사용하지 않고 단순화하다보니 객체를 외부에서 주입 받는 형태가 아니라 내부에서 직접 생성하는 형태로 구현했음에 주의

Bloc 패키지를 이용했으나 저번 주와 같은 이유로 지속적인 화면 갱신이 안 됨

결론

- 현대 패러다임에서 중요시하는 것들이 집대성된 이론으로서 적용하지 않는다고 해도 알아둘 필요는 있어 보임
- 굉장히 많은 디자인패턴과 객체지향 기법들이 광범위하게 사용되므로 개발 실력을 가늠할 수 있는 척도가 될 뿐만 아니라 도달해야하는 마일스톤으로 보아도 좋다고 생각
- 다만 기획이 바뀌면 도메인 부분 또한 다 바뀌어버리는 프론트엔드보다는 비교적 변화가 적은 백엔드에 더 잘 어울린다고 생각하며 스타트업이나 빠듯한 일정의 팀에게는 사용하기 어려울 것으로 예상됨