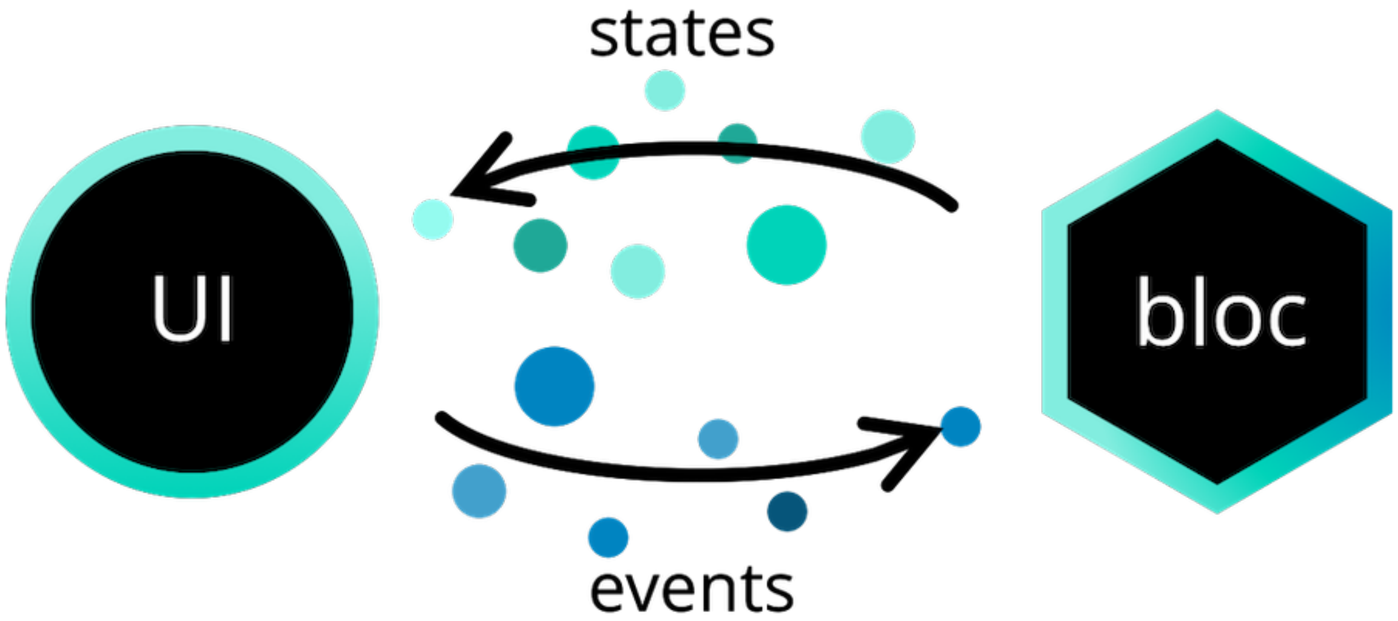
플러터 다트 아키텍처 패턴

4. BLoC 패턴

BLoC 패턴이란?

- 플러터 코드 분리를 위해 나온 패턴
- Business Logic Component의 약자로 비즈니스 로직과 UI를 구분하기 위한 패턴
- UI는 MV* 패턴의 View와 비슷하며
- bloc은 컨트롤러, 프리젠터, 뷰 모델과 비슷한 역할을 함
- 모델에 대해서 이야기하지 않지만 사실 MV* 패턴에서도 언제나 모델의 역할은 동일함



이전까지 배워온 MV*와는 비슷하면서 다른 형태를 띄는데 본질적으로는 비슷함
위 그림을 보면 마치 서로 UI와 bloc이 상호작용하는 것처럼 보이나 실제로는
MVVM처럼 bloc은 UI를 전혀 모르며, UI 또한 bloc을 참조하긴 하나 bloc의 로직을 알지 못 하고
단순히 bloc의 데이터를 구독만 하는 형태

MV* 패턴과의 비교

- 모델에 대해 직접적으로 언급하지 않지만 실제로 모델은 어떤 패턴에서나 동일한 역할을 가정하므로 본질적으로 비슷
- 다만 BLoC 패턴의 경우 모델과의 관계보다는 UI와 비즈니스 로직(혹은 뷰 로직)에 대한 관계에 집중하는 경향이 있음(레이어드 아키텍처의 프레젠테이션 영역에만 집중)
- 이벤트를 커맨드로 블록에 넘기고 상세 로직은 블록에서 처리한 후 결과물인 상태로 반환
- 즉, UI와 bloc 간의 통신에 있어서 엄격한 프로토콜이 존재하므로 대충 짜도 BLoC의 룰만 지킨다면 어느 정도의 아키텍처가 보장됨

플러터와 BLoC 패턴

- 처음부터 플러터를 위해 나온 패턴으로 플러터와 궁합이 좋음
- 특히 플러터 출시 전에 나온 MVC, MVP, MVVM 등이 플러터와 맞지 않는 점으로 인해 원칙을 조금씩 타협하며 적용해야했지만 BLoC은 그런 점이 없음
- 다만 지나친 보일러플레이트 코드와 높은 러닝커브로 인해 호불호가 존재
- 하나의 패턴으로 시작되었으나 flutter_bloc이란 상태관리 패키지로 관리되고 있으며 위와 같은 문제를 해결하기 위해 cubit이라는 간소화 버전도 함께 사용할 수 있음

코드 공유 시간

1. 첫 번째 버전은 패턴으로서의 블록을 강조하여 패키지 없이 스트림을 통해 구현
2. 두 번째 버전은 flutter_bloc 패키지를 사용..했으나 현재 UI가 갱신되지 않는 문제 있음

결론

- 플러터 개발자로 일하기 위해선 필수가 되었다고 생각
- 모든 상태관리와 비슷한 점이 있으면서도 장단점 또한 뚜렷하기에 다른 상태관리와 비교하기 좋음
- 설계 전문가나 아키텍처 관련 역량이 있는 사람이 없지만 MVP가 아닌 실제 프로덕트를 생산해야하는 팀이라면 추천할 만함
- 아키텍처 관련 역량이 뛰어난 팀이라면 꼭 선택할 필요는 없음
- MVP를 통한 PoC를 거쳐야하는 팀이라면 선택을 권장하지 않음