State란 무엇인가

명령형 UI : 전통적인 UI구성

- 어떻게 해야하는지 그 방법을 지정
- 그리고자 하는 요소를 일일히 직접 그려야함
  이후 나온게 선언형 UI
- 무엇이 나타나야 할지를 지정
- 우리가 원하는 상태를 입력하면 이미 그 골격이 있어서
- **원하는 상태를 넣으면 그걸 View에다 그려줌!!**

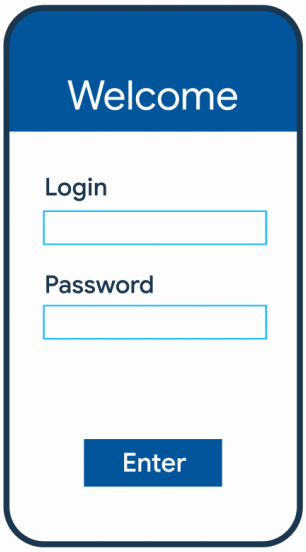
이 상태가 State이다

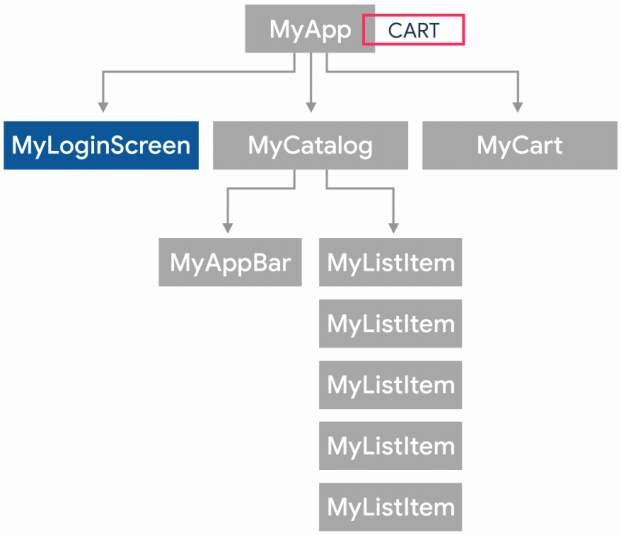
$$View = F(State)$$

우리가 원하는 상태가 State, 그 State 를 제공하면 Flutter가 알아서 이케이케 잘 처리해 View로 만들어 주는 것이다 ~~그림노예~~

추상적이다. 그래서 이게 잘 작동하는지 의심이 갈 순 있지만, 명확하게 지정만 되어있다면 잘 작동한다. 되려 이런 추상적인 이미지 덕분에 우리가 상태가 변함에 따라 변한 결과 화면을 예상호기 쉬워진다.

앱도 기억력이 필요하다 (당연한 소리 같지만...)





Ephemeral state (= UI state, local state, WidgetData)

> ...본래 질병이나 생명주기와 관련하여 '단 하루만 지속됨'; '무언가가 잠시 동안만 지속됨' 이란 확장된 의미는 1630년대부터 사용되었습니다.
> ephemeral 뜻 - 영어 어원·etymonline

하나의 위젯에서 사용하는 정보. 즉 하나의 페이지에서만 사용되며, 다른 페이지를 업데이트할 때 에는 필요없다.
예를 들어

- `PageView`의 현재 페이지
- 복잡한 애니메이션의 진행상황
- `BottomNavigationBar`에서 현재 선택된 탭
  등의 정보를 저장하는데에 사용할 수 있다.

간단하게는 `State`와 `setState()`를 사용해서 만들 수 있다.

App state (= Shared state, global state, AppData)

여러 다른 페이지, 여러 위젯 간 서로 공유해야 하는 정보가 있다면 그건 Application state(이하 App State)에 해당한다.
예를 들어

- 사용자 환경설정
- 로그인 정보
- SNS에서의 알림
- 전자상거래 앱에서 카드에 담은 물품
- 메일/뉴스글을 읽었는지 여부
  같은 정보를 App state로 저장할 수 있겠다.

그래서 언제 뭐 씀?

결론 : 명확하게 정해진 건 없다. 이론적으로 모든 정보를 App state로 넣어도
원래 Ephemeral state 로 저장했던 정보들도 나중에 앱이 커지면 App state로 바꾸게 될 수도 있다. 그때그때 적절해 보이는 방법이면 그만이라는 것이다.

> "The rule of thumb is: Do whatever is less awkward."
> (가장 중요한 규칙은 : 뭐든지 가장 안 어색하게 만들면 된다.)
>
> - Dan Abramov, Redux의 저자

Stateless 와 Stateful

모든 위젯이 다 state가 필요한 건 아니다. 예를 들어 간단히 글자 좀 보여주는 경우에는 state가 필요없을 것이다. 따라서 state와 이로 인한 업데이트가 필요한지 아닌지를 기준으로 Stateful 위젯과 Stateless 위젯으로 구분한다.

실제 코드로 만나기

간단하게 `StatefulWidget`으로 구현할 수 있다.

State 를 관리하기

State 변화를 관리하는 방법에는 여러 방법이 있다. 상황에 따라, 계획에 따라 선택하면 될 거 같다.
공식 사이트에서는 다음과 같은 방향들을 소개하고 있다.

Provider

`provider` 패키지.
`ChangeNotifier`, `ChangeNotifierProvider`, `Consumer`등의 개념을 사용해서 구성한다.
기본적으로 Notifier가 State의 변경을 공지하면 그걸 Listener들이 받아 실제 변경이 일어나는 구조이다.

기본적으로 특정 조건을 만족하면 `ChangeNotifier`를 상속한 클래스의 method를 부른다. 그럼 이 클래스에서 논리적인 부분을 다 처리하고 나면 `notifyListeners()` 함수를 통해 변경이 필요한지를 알린다.
그럼 이를 `Consumer<ChangeNotifier를 상속받은 클래스>`가 받이 실제 UI가 어떻게 변할지를 지정한다.

- Google이 추천
- 큰 프로젝트에서는 비추
  Flutter 공식 사이트에서 State를 설명하며 보여주는 예제이다. Provider를 사용하는 예로는 적당한 거 같다.
  Simple app state management | Flutter

Riverpod

Provide와 비슷하게 작동한다. Provider와 유사하게 `ConsumerWidget`, `ProviderScope` 등 자체적인 클래스를 사용해서 구현한다.
컴파일과 testing에서 Flutter SDK 에 의존하지 않고도 안정성을 가지게 해 준다고 한다.

사용법(야매)

```
@riverpond
String control(ControlRef ref){
        return 'wow';
}
```

클래스 이름을 정하고, 이름 첫 문자를 대문자로 바꾸고 그 뒤에 'Ref'를 붙여 reference를 만든다. 그 뒤에 업데이트가 될 위젯에게 `StatelessWidget` 대신 `ConsumerWidget`을 상속받게 하고, 안에

```
String val = ref.watch(controlProvider);
```

클래스 이름 뒤에 'Provider'를 붙여 사용한다.
이렇게 만들고 명령어로 `dart run build_runner build`를 사용하면 애가 뚱땅뚱땅 거리면서 필요한 클래스들을 만들어 준다(사용한 피일명에 '.g'를 삽입해서 새로운 파일이 만들어진다).

setState

별도의 package 설치없이 사용 가능하다.
변경될 부분이 있는 위젯을 `StatefulWidget` 을 상속받아 만들고, 이를 `State<Stateful로 만든 위젯>`에서 `setState()` 함수를 사용하여 변화를 알린다.

Ephemeral state을 관리할 때만 사용할 수 있다. 물론 위젯 매개변수로 값들을 넘겨주는 방식으로 App state를 관리하는데에 사용할 수 도 있다고 한다.

Android Studio로 새 프로젝트를 만들면 예제로 적혀있는 Click counter가 `State`와 `setState`로 만들어져있다.

```
class MyHomePage extends StatefulWidget {
        const MyHomePage({super.key, required this.title});

// ...

class _MyHomePageState extends State<MyHomePage> {
        int _counter = 0;

        void _incrementCounter() {
                setState(() {
                        // ...
                        _counter++;
                });
        }
// ...
```

ValueNotifier & InheritedNotifier

오직 Flutter에서 제공하는 방식만을 사용하여 state와 UI의 변경을 관리하는 방법이다.

InheritedWidget & InheritedModel

setState처럼 low-level방식으로 부모 위젯과 자식 위젯간 통신을 위해 사용하던 방법이다.
`Provider`와 같이 다른 접근 방법들이 이 방식을 기저로 사용한다고 한다.

June

Flutter에 이미 내장되어 있는 state 관리 방법과 비슷한 구조로 관리할 수 있도록 하는 방식이라고 한다.

Redux

유명한 state 관리 방법이다
Redux 자체는 flutter에만 사용되는것이 아닌, javascript에서 state를 관리할 때 쓰던 방식인데, 이를 Flutter에서도 사용할 수 있게 `flutter_redux`가 나왔다.
사용방법 자체는 Javascript용의 원 Redux와 유사한 거 같다.

Fish-Redux

Redux에 기반해서 만든 Flutter 앱 개발용 프레임워크이다. 중형~대형 앱을 개발할 때 적합하다고 한다.

BLoC/Rx

stream/observable을 사용한 패턴들을 모아놓은 것이다.

- Google이 추천
- 대형 어플리케이션 개발에 적합하다고 한다.

GetIt

BuildContext가 필요없는 방법이라고 한다. service locator에 기반하고 있다

MobX

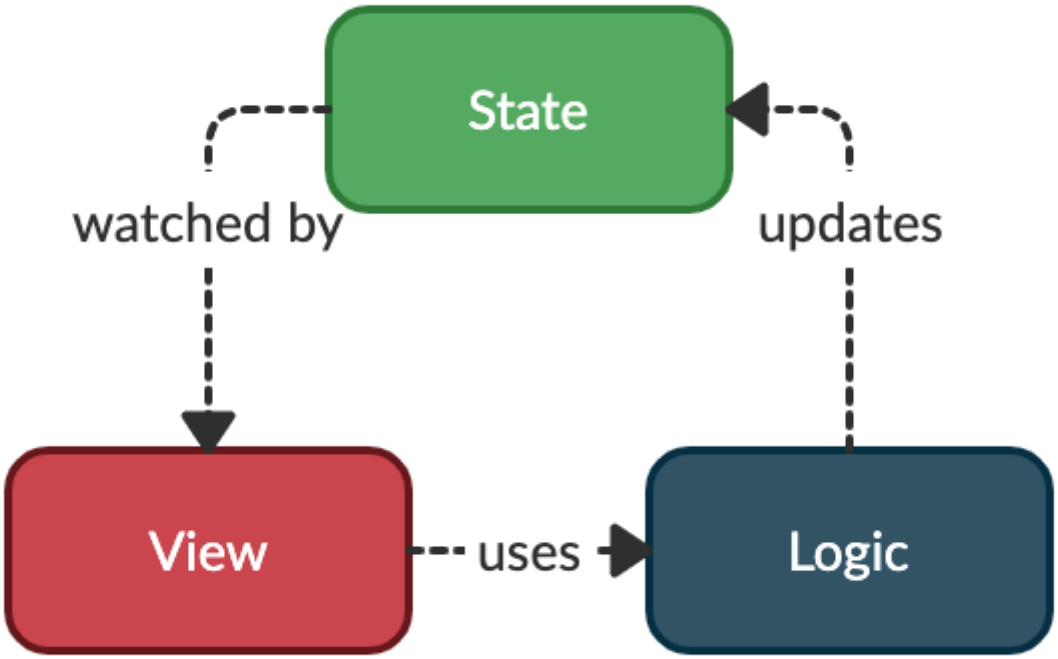
BLoC/Rx와 비슷하게 observable과 reactions에 기반한 라이브러리이다.

Flutter Commands

`ValueNotifiers`와 `Commands`의 디자인 패턴을 기반으로 만들어졌다.

Binder

`InheritedWidget` 을 기반으로 만든 state관리 패키지이다. 가볍고 강력하다고 한다.



GetX

> GetX is an extra-light and powerful solution for Flutter. It combines high-performance state management, intelligent dependency injection, and route management quickly and practically.
> get | Flutter package (pub.dev)

굉장히 가볍고 강력한 state 관리 도구라고 한다. ~~자신감이 대단하다~~
GetX는 state 관리 말고도 Flutter를 편하게 사용할 수 있는 기능들을 제공한다.

- 더 편한 페이지 변경
- Toast, bottomsheets 등을 더 간단하게 올림
- key&value 저장 쉬워짐
- Theme 변경 쉬워짐

간단하게 GetX로 state 다뤄보기

변수를 생성할 때 `observable` 한지를 끝에 .obs 를 붙여

```
var 변수=값.obs;
```

flutter에서는 다음 코드를 보여준다

```
class Controller extends GetxController{
        var count = 0.obs;
        increment() => count++;
}
```

이후에 Get.put(Controller())로 가져와서 편하게 사용할 수 있다. `Get.find()` 로 다른 페이지에서 쓰인 컨트롤러도 다져와서 사용할 수 있다.

states_rebuilder

State management + dependency injection + integrated router

참고자료

- [Flutter] Stateless & Stateful 알아보기 (tistory.com)
- Flutter 1부터 배우기 : State가 뭔데? [stateful, stateless 차이] (velog.io)
- State management | Flutter

히히 어렵당ㅠㅠ
나 자신 화이팅..... :)