초격차 패키지 Online.

안녕하세요,
Part 2 기초강의를
맡게 된,유병욱입니다.

Chap1 | Flutter Widget 101
Flutter에서 가장 기본이 되는 단위인 Widget

Chap2 | Flutter Layout
Flutter UI 레이아웃을 짜는 전략 / 노하우를 알아봅니다

Chap3 | Flutter State Basic
Flutter의 상태 관리의 기초를 알아봅니다.

Chap4 | Flutter Architecture
Flutter 프레임워크의 기본 동작 구조를 알아봅니다.

Chap5 | Flutter Gesture
Flutter의 입력 방식들에 대해 알아봅니다.

Chap6 | Flutter Setting
Flutter의 다양한 추가 옵션들을 설정하는 방법을 알아봅니다.

Chap7 | Flutter Navigator
Flutter의 페이지 전환에 대해 알아봅니다.

Chap8 | 예시 프로젝트
프로젝트를 활용하여 입문 / 기초에서 배운 내용을 되짚어봅니다.

Flutter Widget 101

Flutter는 거의 모두 Widget으로 이루어져 있다고 흔히들 이야기 합니다.

UI를 구성하는 기본 단위인 Widget이라는 게 어떤 것인지,
그리고 Flutter의 Widget을 구성하기 위한 디자인 가이드 라인인
Material Design과 Cupertino Design에 대해 알아봅니다.

또한 간단한 위젯들의 종류와, 대표적인 Widget들에 대해서도 알아봅니다.

Flutter Layout

Flutter에서 하나 이상의 Widget을 활용하여 화면을 구성해봅니다.

Widget을 어떻게 배치하는 지 부터,
다양한 디스플레이 사이즈에 대응하기 위해 어떤 식으로 작업하는 지,

그리고 레이아웃의 기틀을 잡기 위해 개발자와 디자이너가 중점적으로
논의 해야 할 사항 들에 대해서도 알아보고자 합니다.

Flutter State Basic

Flutter의 Widget은
크게 StatefulWidget과 StatelessWidget으로 구성 되어있습니다.

여기에서 State란 것은 어떤 것인지,
그리고 화면을 갱신하기 위한 작업이 어떻게 이루어 지며,

하나의 Widget이 UI로 나타나고 사라지는 과정, 라이프 사이클에 대해서도
기초적인 내용을 알아봅니다.

Flutter Architecture

Flutter의 기본적인 동작 구조를 조금 더 깊게 알아봅니다.

어떤 방식으로 Flutter가 UI를 처리하고 비즈니스 로직을 처리하는 지 부터,
좀더 효율적이고 안정적인 개발을 위해
어떤 점을 중점적으로 봐야 할 지와, 주의해야 할 점들을 알아봅니다.

Flutter Gesture

Flutter는 Cross Platform을 지원하는 것에 걸맞게,
다양한 입력을 제공합니다.

간단한 텍스트 입력부터, 다양한 터치 상황 등을
어떻게 입력을 받고 어떻게 처리하는 지 등에 대해서 알아봅니다.

Flutter Setting

Flutter에서 Asset(Image, Font) 등의 데이터를
어떻게 사전에 준비하고, 활용하는 지 부터

Open Source라는 특징에 걸맞게 공개된 라이브러리들을
어떻게 검색하고, Flutter 프로젝트로 가지고 와 활용할 수 있는 지

그리고 기본적인 Flutter 프로젝트의 설정 방법들에 대해 알아봅니다.

Flutter Navigator

Flutter에서는 앱 내에 화면 이동이 가능합니다.

화면을 이동할 때, 데이터를 넘겨주고 받아오는 것 부터,
이동 방식들을 어떻게 정하고 진행하는 지

그리고 기본으로 Flutter에서 제공하는 화면 전환 방식에 대해 알아본 뒤,
현업에서는 어떤 라이브러리를 화면 전환에 주로 활용하는 지 알아봅니다.

예시 프로젝트

지금까지 입문과 기초에서 알아본 Dart와 Flutter를 활용하여,
기본적인 앱 들을 개발하거나, 유명 앱의 UI를 클론 코딩을 진행하면서,

그동안 배웠던 내용에 대한 점검 / 복습을 진행합니다.

Flutter Widget 101

Flutter UI의 시작과 끝

Flutter에게 UI란?

Flutter를 소개 할 때 Cross Platform이라고도 설명하지만,
한 편으론 UI Toolkit이라는 소개 문구도 심심치 않게 등장하곤 합니다.

사실 Flutter에게 UI란 상당히 중요한 요소이기도 하고,
Flutter가 각광받는 이유 또한, 하나의 소스코드로,
수려한 UI를 여러 Platform에 출력 해 낼 수 있기 때문입니다.

조금의 과장을 더해서 Flutter에는 UI가
사용하는 이유의 70~80%의 비중을 차지한다고 해도
과언이 아닐 정도입니다.

이처럼 Flutter에서는 UI가 상당히 중요합니다.

Flutter UI에서 Widget이란?

Flutter의 UI의 가장 기본적인 단위는 Widget입니다.

Flutter는 UI를 구성하는 각각의 요소가 모두 Widget으로 이루어져 있기 때문인데요,
간단한 블록부터, 텍스트 입력 창, 심지어 화면 내 특정 요소 간의 간격 조차 Widget으로 관리 됩니다.

Flutter 개발자에게 UI를 효율적이고 디자이너의 시안을 구현하는 데에 요구되는 것 중 하나가
바로 이 Widget을 얼마나 적제적소에 효과적으로 사용할 수 있느냐가 관건입니다.

Flutter 기본 SDK의 Widget만 해도 상당히 많을 뿐 더러,
pub.dev를 통해 제공되는 오픈소스 Widget을 모두 더한다면
상당히 많은 수의 Widget들이 있는 것을 볼 수 있습니다.

Widget Tree

Flutter의 Widget은 마치 블록과도 같습니다.

Widget을 조합해서 사용하는 방식에 따라 전혀 다른 UI를 만들어 낼 수도 있고,
한 화면에서 만들어 뒀던 Widget을 다른 화면에서 그대로 가지고 와 사용하는 것도 가능합니다.

또한 Widget들 간에는 수평적인 관계도 존재하지만,
수직적인 관계도 만들어 낼 수 있습니다.

이런 Widget들 간의 관계를 나타낸 형태를 Widget Tree 라고 하며,
이 개념은 추후 상태를 관리하는 데에 있어서도 상당히 중요한 개념입니다.

Stateless Widget / Stateful Widget

Flutter 개발자들이 보통 직접 Widget을 만들 때,
위의 2가지 형태를 활용하게 됩니다.

이 두 개념에 대해서는 추후 Chap 3 Flutter State Basic에서 조금 더 자세히 다룰 예정이며,
간략하게 먼저 알려드리자면

화면을 갱신 할 필요가 없는 정적인 화면을 구성할 때에는 Stateless Widget을,
특정한 상황에 따라 화면을 갱신할 필요가 있다면 Stateful Widget을 사용한다. 정도로만
기억을 해 두시면 좋을 것 같습니다.

이번 실습에서 Chap 3에 들어가기 이전까지는 모두 Stateless Widget을 활용하여,
Widget을 구현 해 보도록 하겠습니다.

Hot restart? Hot reload?

Dart 언어를 소개 할 때 JIT 컴파일러와 AOT 컴파일러가 사용되는 점을 알려드린 적이 있습니다.

Flutter에서는 이 언어적 특징을 사용해서,
빠른 개발을 위해 Hot restart / Hot Reload 기능을 제공하고 있습니다.

이 기능을 사용하면 JIT 컴파일러를 활용하여,
앱을 시간이 오래 걸리는 컴파일 과정을 거치지 않고서도
코드의 변경 사항을 즉시 반영할 수 있습니다.

Hot restart

Hot restart는 말 그대로 앱을 재시작 하는 형태입니다.

이 기능을 활용할 시, 앱의 모든 상태가 초기화가 되고,
앱의 코드 변경 사항을 반영 할 수 있게 됩니다.

Hot reload

Hot reload는 hot restart와는 달리 앱을 재실행하지는 않지만,
코드의 변경 사항을 반영하는 기능입니다.

이 기능을 활용할 시, 앱의 상태를 유지한 상태로,
코드의 수정사항을 확인하여 더욱 빠른 개발을 할 수 있게 해줍니다.

Design Guide

Flutter 내에서는 대표적인 2개의 디자인 가이드를 활용하여 개발할 수 있습니다.
물론 이 가이드에서 벗어난 커스텀 디자인 가이드도 적용 할 수 있습니다.

다만 이 2가지 디자인 가이드는 IT의 오랜 시간 앱을 만들어온 Google과 Apple의 노하우가 담긴
디자인 가이드이기 때문에 가급적 해당 가이드를 활용해서 만들어도, 충분히 앱을 개인화 하여 만들 수 있습니다.

Google의 철학이 녹아져 있는 디자인 가이드는 Material Design이며,
Apple의 철학이 녹아져 있는 디자인 가이드는 Cupertino Design 이라고
Flutter 내에서는 구분 짓고 있습니다.

해당 디자인 가이드들을 디바이스에 따라 분리해서 구현 할 수도 있으며,
혹은 Android기기 에서 Cupertino의 디자인 가이드를 따르거나,
iPhone에서 Material 디자인 가이드를 따르게도 만들 수 있습니다.

Material Design

앞서 이야기 했듯, Material Design은 Google에서 제시하는 디자인 가이드라인이며,
비교적 최근 Material 3 을 발표 하면서 개인화 된 앱을 구현하는데에 중점이 맞춰져 있습니다.

Material Design에 맞게 여러 Widget들이 이미 구현이 대부분 되어있으며,
Flutter에서는 Material 2와 3을 개발자의 취향에 맞게 선택하여 활용 할 수도 있습니다.

또한 Flutter의 경우 Google에서 주도적으로 개발하고 있는 프레임워크이다보니,
비교적 Cupertino Widget보다 Material Widget이 더 많이 존재하는 부분도 없잖아 있습니다.

디자인 가이드를 따르는 것이 좋을까?

디자인 가이드를 따르는 것이, 독창적인 앱을 만드는 데 방해가 될 수 있다고 생각할 수도 있지만,
사실 디자인 가이드 자체보다는
디자인 가이드에 내부에 들어가는 콘텐츠가 더 독창적인 요소를 불러일으킬 수 있습니다.

앱 전반의 Font 스타일링을 담당하는 Typography 부터, Color 등의 Theme 요소를
더욱 잘 활용하는 것이 독창적인 앱을 만드는데 효과적이며,

특히 별도의 디자이너 직군이 없는 소규모 앱 개발 인원을 보유한 상황이라면,
개발자들 만으로도 수려하고 사용성 높은 앱을 구현 하는데에 있어,
디자인 가이드를 지키는 것이 앱의 완성도를 높이는데에 큰 도움이 될 수 있습니다.

Lifecycle

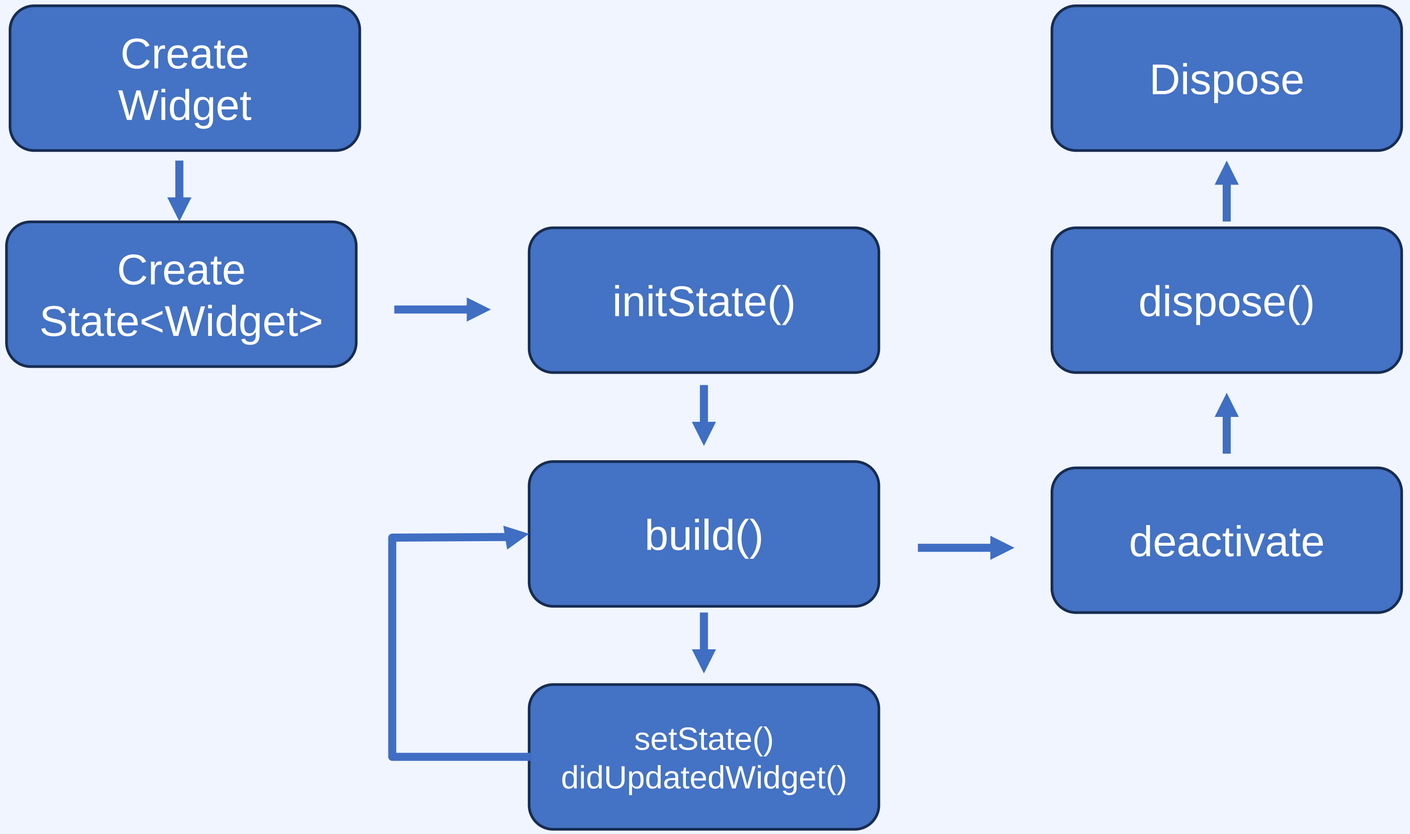
직역하면 생애주기로, 일반적인 프로그래밍에선,
한 객체가 생성이 되고 동작이 이루어진 뒤, 프로그램 내에서 정리되는 과정까지를 말합니다.

객체 지향 프로그래밍을 비롯해 여러 프로그래밍 방식에서 중요한 개념 중 하나이며,
최적화나 안정성을 고려하는 프로그램인 경우, 중요도가 더더욱 올라가는 개념입니다.

Flutter에서도 각각의 Widget을 비롯한 모든 객체가 각자의 Lifecycle을 가지고 있으며,
각 객체별 Lifecycle을 잘 컨트롤 하여 앱의 동작이나 레이아웃, UI등이 원활하게 동작할 수 있도록
컨트롤하고 코드를 작성하는 것은 매우 중요한 일입니다.

Stateless Widget Lifecycle

Stateful Widget Lifecycle



쉽지않은 상태관리

각각의 Widget 내부 뿐만 아니라,

다른 Widget간의 상호작용,
전역적인 변수와 함수에 대한 관리
시기 적절한 dispose / initState…

이러한 문제점들을 해결하기 위해 Flutter에는 여러 상태관리 모델이 오픈소스로 개발되어 공유되고 있습니다.

Riverpod / BLoC / GetX / Provider / Redux…

상태관리 모델에 대해서는 Part 2 이후에…