Flutter

- declarative UI framework.
- launched in 2018
- Focus is more on modern UI Patterns
  - single source of Truth (hierarchy)
  - composable components, to be reused (widgets)
  - multi-platform
  - Dependency Rendering
    - widget ~~on~~ → variables
      - → properties
      - → States

      (to know they rendered)
  - Data Binding & Reactive Programming (basics)
  - UI is expressed in widgets

    (its a Dart class that we extend from widget super class.

- Flutter relies on SDK's and compilers from the official SDK's.
  - android SDK. - android apps
  - Xcode (IDE) - iOS apps

compared to other mobile UI toolkit frameworks, Flutter is a low-level.

FRONT-END MAP.

- make native clients
  - official SDK's
    - swift → iOS
    - kotlin → android
  - cross-platform SDK's
    - React Native (until 2020)
    - Microsoft Xamarin
    - Flutter (now 2022)
- web / PWA.

* Flutter does not use any UI framework from OS, it renders its own button in a low level canvas.

→ calling the button, looks like pixel perfect.

vs.

other attach a button from OS's.

FLUTTER compiles for

- Google Play Store.
  creating AAB file
  (Android App Bundle)
  to Android & Chromebook.
- Amazon AppStore
  creates an APK to
  fire OS & windows 11
- Huawei AppGallery.
  create APK to Harmony OS
- Apple AppStore
  create IPA to iOS & iPadOS.
  (iPhone Application)
- Microsoft Store.
  - UWP to window 10/11
  (Universal windows Application)

— does not use any SDK (OS's)

— its own "widgets" & design.

— two widget sets, ready to use.

  — material

  — cupertino (iOS like)

— they clone pixel-perfect the Android Material Design & Apple Human Interface guidelines

— high performance

— Same gestures & animations.

- Multi-User Interface

— does not happen automatically because

— widgets are not translatable.

— There are design patterns to reduce coding apps twice.

• Products from Flutter Project

① IPA ——— iOS App (compatible with iPadOS)

② APK & AAB – Android App

③ Web Build Folder – ready to deploy – PWA

④ EXE ——— windows (beta)

⑤ APPX ——— windows universal (alpha)

App ——— mac OS (beta).

* when you create a flutter application and you want to compile for Android, it creates Android Studio Project internally.

* flutter supports only snake_case names for the project & for our files.

* keep Emulator (phone, web window TARGET) running and everything, that you change really quick in file & hit save, it shows.

Hot Reload in Flutter

VS Code → Ctrl + Shift + P → Command Pallete

folders explanation

- .dart_tools — private folder for some tools, part of Dart Compiler or the Flutter SDK.

- android — android Studio project, that flutter uses to compile android app.

Sort of, container for ANDROID application

- iOS — contains an XCode project.

everytime you build an app on flutter, its not going to change the contents of these proj, these Android, iOS projects are static.

- web — similar to iOS / Android, wrapper of our web version

what is wrapper for web?

↓

an index.html

- test — we will insert test suites for unit testing
- windows — same as Android wrapper, to compile a windows executable.
- lib — library, our code resides here. flutter store our Dart files.

* src as folder name. → general convention

- pubspec.yaml — yaml file

configuration file, where
- we can change properties,
- we can ADD DEPENDENCIES.
- set somethings that are used by flutter SDK.

- every time you compile ↓ build app ↓ SDK takes info. from pubspec.yaml

(Similar to package.json in an NPM project)

- flutter doctor command on Terminal to debug errors

- flutter is heavily based on the concept of a widget.

↓

Something that decides, how to render content on the screen.

— it's a class, that extends from widget, but widget class is Abstract, we cannot actually extend from widget class.

dg—

simplest sub-class of widget

```
class HelloWorld extends StatelessWidget {
  // constructor, taking key for superclass, and passing it in
  const HelloWorld ({super.key});

  @override
  Widget build (BuildContext context){
    return const Text ("Hello World");
  }
}
```

↑ first widget ↑

↓

its a class with build method, returning another widget.

widget has only one required property, key property

keys being used by framework for the tree.

↓

Since it creates a tree of widgets in memory, and each one has its own key, a unique ID

↓

and its internal, so you can't set value of these keys.

↓

* internally framework sets the key, but we provide placeholder (Space) to it ?

* you will need a constructor in your widget class, that recieves a key and passes that key to super class.

↓ why

because framework makes a tree of widgets.

* build function or method of my Class with name build, that recieves BuildContext

* @override — annotation saying, this is not just any function or any method, its overriding the superclass method.

↓

it needs to return a widget

↓

So build method typically returns a widget.

* simplest widget we can use is Text("__");
which recieves a text & renders that text on
that screen.

* VS code has shortcut to create this
StatelessWidget class.

eg.

VS Code shortcut. stless → output

```
void main(){
  runApp(const HelloWorld());
}
```

↑ asking system to run HelloWorld widget

```
class HiThere extends StatelessWidget {
  const HiThere({Super.key});
  @override
  Widget build(BuildContext context){
    return Container();
  }
}
```

Container() → includes another widget by default, called container.

↓ empty widget that contains other widgets.

↓ Similar to in HTML.

example

Text widget is trying to define its position and its position and its size based on a container.

↓

that one didn't declared

↓

So error showed (red)

* So, Basic Widget Tree —

- MyApp
  - MaterialApp
    - MyHomePage
      - HelloWorld
        - Text

* There is no way to return an array or multiple elements, we always need to return 1 widget. But, we have widgets that acts as Containers.

* Based on widgets layout, we pick a Container.

Column — stack children widgets vertically in a column.

have lots of properties in container, all of them, are optional

its an array of widgets

TextField (); — widget

Text has changed? — OnChanged:

argument

recieves a function

- Stateless Widget will never change.
  - it doesn't contain a State.
  - its not going to re-render itself

So, we have 2 kinds of widgets in flutter:-

Stateless

if you receive property from outside, its a Stateless widget

Stateful

↓

can change its own content, based on things that happen within that widget. like event, or external factor.

* clicked on class :  appears, ctrl+. (convert to a Stateful widget)

* Stateful widgets has 2 steps:-

① - contains key property

② - has state object that's of another class.

(11) It has two classes inside
(its just an execution order)
— the widget itself
—

eg. — Stateful Widget happening in two ~~classes~~ widgets

```
class HiThere extends StatefulWidget {
  const HiThere ({ super.key });
  @override
  State<HiThere> createState() => _HiThereState();
}

class _HiThereState extends State<HiThere> {
  @override
  Widget build (BuildContext context) {
    return const Text ("Hi There");
  }
}
```

current re-render happens in this State

* every time you make a change in the state, that will make it re-render.

  └ everytime Build function is being executed

* State Variable is one or more variables that are actually defining changes in your user Enterface in the widget.

- setState(() {

  });

  setState method, that receives a function as an argument, where you put all the changes that you trigger, every render

- How to make changes in your own user interface?

↓

create variables, as properties of $tate class, & then you change the values of those variables within a setState call.

(Similar to class components of React)

variable changes

↓

the system re-rendering the widget.

↓

so, going through build method again n again.

VS code shortcut - Stful. → output

↳ it create both the widget class & the state class.

* name styles be UpperCamelCases.

NOTE.

- Flutter code is large.

↓

lots of lines of source code

↓

but, we are heavily based on the tool to make that code

↓

to wrap, to add, to remove widgets & work with widgets.

- on flutter, not everything, but most of the things are widgets,

↓

to add padding, we have padding widget (its a widget Constructor)

* So, we are using OOP here in flutter, its everywhere.

SIMILARITIES

State management in flutter mirror react in lot of ways

↓

flutter copying the react model pre-hooks.

State-based UI component.