Flutter Basics

What? Why? How?

Important
This document is just a short summary of the basics taught in this module! Please also scroll to the **end of this document**. There, you find a link to further resources!

What is Flutter?

Flutter is actually a combination of two things: A SDK (software development kit) and a framework built upon Dart. Learn more about these two pieces below!

Flutter aims to make cross-platform development (of mobile apps) a breeze. The goal is to allow you to have one codebase that generates apps for multiple platforms (like iOS and Android).

To achieve this goal, Flutter is compiled to native machine code (to achieve good performance) and Flutter also offers a lot of utility tools and features to speed up your development work.

In particular, Flutter is all about so-called widgets - re-usable buildings blocks you can use to compose beautiful user interfaces. Unlike alternatives like React Native, Flutter does not use the platform primitives (i.e. the default UI building blocks Android/ iOS give you) but controls every pixel on the screen itself. This gives Flutter (and ultimately you) more control over the user interface.

What is Dart?

Dart is the programming language used by Flutter - you can learn more about it in a dedicated document which is attached to the last lecture of section 2 in this course!

Flutter uses Dart because it's a fast and efficient language which is compiled to native machine code.

Flutter itself is a framework (and a SDK - see below) that uses Dart.

Flutter SDK vs Flutter Framework

SDK stands for "Software Development Kit" and in Flutter's case, that means that you get a bunch of tools that help you develop, test, optimise and build your cross-platform/ mobile app.

You get tools for efficient development (e.g. the Flutter extension for Visual Studio Code) and testing. And you also get all the tools you need to then generate the Android APK or iOS IPA file which you can then deploy to the respective app store.

The framework part of Flutter is the part that you utilise when writing code. It's built upon Dart (hence that's the programming language you use when writing Flutter apps) and offers a broad variety of utility functions and built-in widgets. Because everything in Flutter is about widgets basically (see below).

You use these framework tools (i.e. the widgets) to compose beautiful user interfaces for your app.

Widgets, Widgets, Widgets

As mentioned, Flutter is all about widgets!

Widgets are building blocks with which you compose the user interface for your app. A button would be a widget, so would a text output be.

You also have "invisible" widgets which you don't see directly on the screen but which you use to structure and control other (visible) widgets. For example, there is a "Column" widget in Flutter, with which you can order other widgets (e.g. buttons or text widgets) in a column, sitting above each other.

You build your user interface in code only - there is no visual drag and drop editor or anything like that. Instead, you build a so-called "widget tree", which means, that you have one top-most widget (often the "MaterialApp" widget) which then takes other widget(s) as arguments - and so on.

You also build your own widgets in Flutter - actually, your entire app is wrapped in one core widget that you build. Your widgets than are composed from other built-in or custom widgets.

A basic Flutter app could look like this:

```
Widget build(BuildContext context) { // each widget has a build() method
  return MaterialApp(
    home: Scaffold(
      body: Column(
        children: [
          Text('This gets drawn on the screen!'),
          RaisedButton(child: Text('Tap me!'), onPressed: () {}),
        ],
      ),
    ),
  );
}
```

This would in the end draw some text and a button onto the screen. Typical for widget trees is the trailing comma at the end of each line. It's technical not required but can be used in conjunction with your IDEs auto-formatting functionality to structure your code in a highly readable way.

In this example, you can also see the widget tree: We have widgets nested into each other - `MaterialApp`, `Scaffold`, `Column`, `Text` and `RaisedButton` are all widgets built-into Flutter. And we pass them all into each other as named arguments in their constructors. For example, the `Column()` widget has a `children` argument which takes a list of other widgets (`Text` and `RaisedButton` in this example).

How a Flutter App Starts

A Flutter app starts by first executing the `main()` function in your `main.dart` file - this is done automatically (you don't have to do anything for that to happen).

Inside of the `main()` function, you need to call the `runApp()` function which is built-into Flutter (that's why you also need to import the Flutter package into the file where you run `runApp()`: `import 'package:flutter/material.dart'` enables this function).

`runApp()` expects to get a (custom) widget which in turn is a Dart class that extends either `StatelessWidget` or `StatefulWidget` - classes which you can extend since they're made available by the Flutter `material.dart` file/ package.

You instantiate your custom widget and pass the instance/ object as an argument to `runApp()`.

Your custom widget then needs to have a `build()` method which returns a `Widget` in turn. All custom widgets need to have this `build()` method - `StatelessWidget` / `StatefulWidget` (which you have to extend) force you to have one.

The `Widget` returned by your `build()` method is then what Flutter draws onto the screen of your app. It calls the `build()` method for you, you don't do that manually.

Visible (Input/ Output) and Invisible (Layout/ Control) Widgets

In Flutter, you work with different kinds of widgets. Typically, you have a lot of widgets which can be seen by the user (e.g. buttons, text output, ...). But you also need widgets to structure your widgets (e.g. in a `Row()` or `Column()`).

It's the combination of such widgets that allows you to create beautiful applications.

Stateless vs Stateful Widgets

Above, I mentioned two types of widgets you can build / extend: `StatelessWidget` and `StatefulWidget`.

`StatelessWidget` is the more common widget you will extend - it allows you to build widgets that in simply output data.

`StatefulWidget` is required to have a widget which can not just output data but also manage some internal data - that means it can change the data and update the UI after such a change. `StatelessWidget` can't do that, you can't update the UI from inside there.

Further Resources

Obviously, we'll dive into many, many more widgets and Flutter features throughout the next lectures and modules.

If you want to get a glimpse at all available widgets, the official docs and the widget catalog are the place to go though: https://flutter.dev/docs/development/ui/widgets

The official docs are a great place to learn all about a given widget, find additional instructions and information and learn about all available arguments you may use to configure/ use the widget!