Laboratorio di Applicazioni Mobili

Bachelor in Computer Science &
Computer Science for Management

University of Bologna

Flutter

Nicolas Lazzari
nicolas.lazzari3@unibo.it

Slides based on https://www.udemy.com/course/learn-flutter-dart-to-build-ios-android-apps

Table of Contents

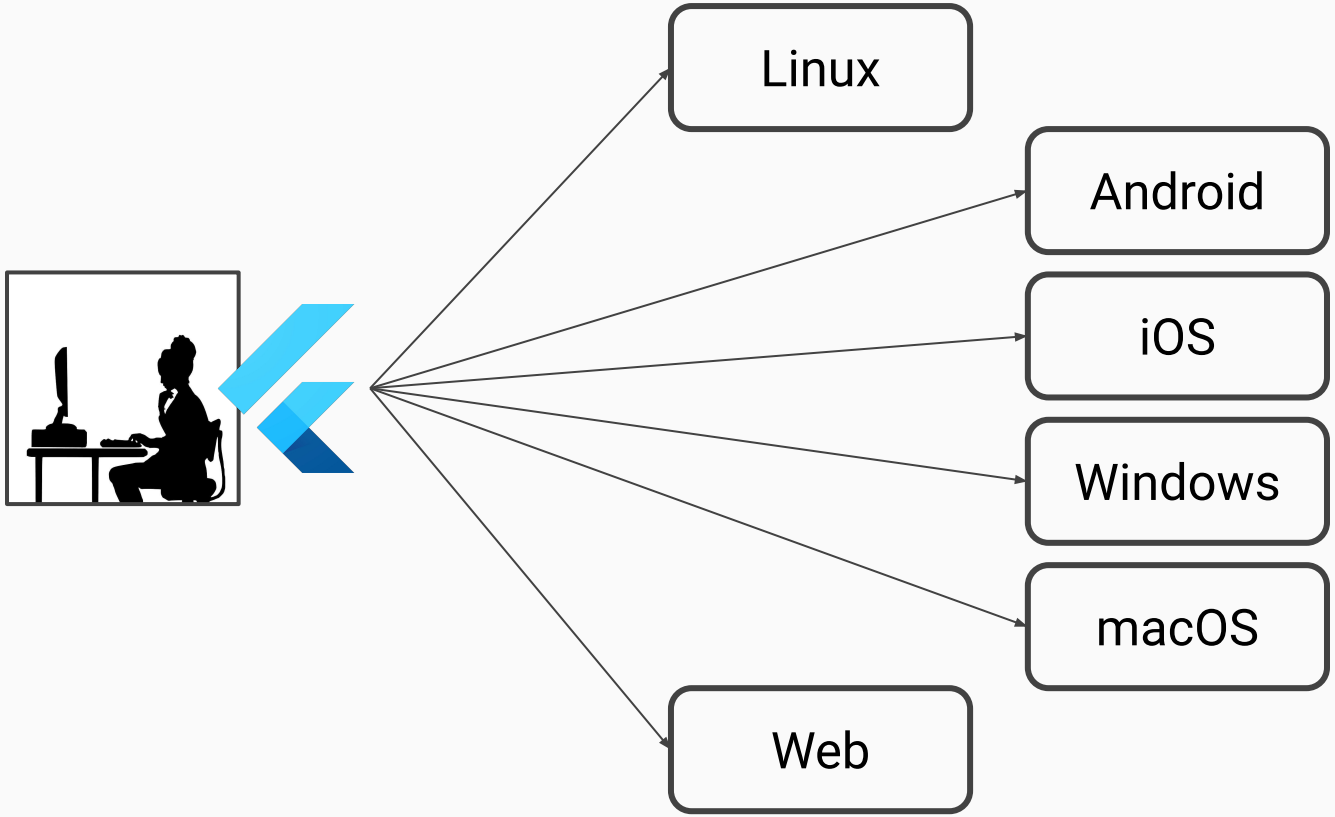
What is Flutter?

= UI Framework + Tools

UI Framework → Components, utilities and functions for cross-platform development

Tools → CLI and software to build and test cross-platform apps

Why Flutter?



Why Flutter?
Apps built with Flutter

Google Pay

Google Earth

Google Ads

Google Classroom

西瓜视频
点亮对生活的好奇心

ebay motors

Abbey Road Studios

See https://flutter.dev/showcase for more

Not a programming language!

Flutter is **not** a programming language.

It is a **framework** which allows the definition of user interfaces using **Dart**. The user interfaces built with Flutter can be *compiled* to machine code compatible with different platforms.

Dart

- Designed by Google in 2011:
  - Object oriented
  - C-like interface (similar to Java and Kotlin)
  - compiles to machine code, Javascript and WebAssembly

Dart
Getting started

Similarly to Kotlin and Java, Dart is a general programming language, and can be used for:

- Game development (https://github.com/luanpotter/flame)
- REST APIs (https://www.theconduit.dev/)
- Mobile Apps (through Flutter)

Dart Variables

```
var x = 42;                 // Declaration of a variable with inferred type Int
x = “forty-two”;            // Error
const X = 42;               // Declaration of a constant
x = 420;                    // Error
final x = 42;               // Declaration of a constant
x = 420;                    // Error
```

const must be used when the value can be determined at compile-time, while **final** can be determined at runtime.

Dart Variables

```
const X = 20240506;                 // Ok, is fixed at compile time

final x = new Datetime.now();       // Computed only once at runtime
```

Dart Variables

Dart's most controversial feature is its *Static Type safety* and *Sound Type* checks.

```
dynamic x = 42;          // x is now inferred of being of type int

x = "forty-two";         // Dart does not complain: x is now of type str,
                         // being an int is forgotten.
                         // Using dynamic makes variable behave like in
                         // Python. Use cautiously!
```

Dart Operations

Operations in Dart are straightforward

- Arithmetic Operators: + - * / %
- Logical Operators: && || !
- Comparison Operators: < > == >= <= !=
- See https://dart.dev/language/operators

Dart
Print and strings

Like some other imperative languages, the access point is the **main** function.

```
main() {
    dynamic x = “Mario”;
    print(“Hello ${x} (${x.toLowerCase()})”) // Prints ‘Hello Mario (mario)’
    x = 42;
    print(“Hello ${x} (${x.toLowerCase()})”) // Error
}
```

Dart
If-then-else

The IFTE construct is straightforward too…

```
if ( x == 42 ) {
    y = 1;
} else {
    y = 0;
}
```

There is a contract syntax for assignments

```
var y = 1 ? (x == 42) : 0;
```

Dart Array

Arrays are **not** objects! They are equivalent to their C primitive (immutable in size, type invariant) and

```
var arr = const [1, 2, 3]
print(arr[0])
var arr = [1, 2, 3]
print(arr[0])
```

Dart List

Differently to arrays, lists are mutable **only** in functions and classes.

```
List<String> list = new List<String>();
list.add("SomeArray");
print(list[0]);
```

Dart Loops

The iteration constructs are straightforward too

```
// While loop
var i = 0
while (counter < list.length) {
      print(list[i]);
      counter++;
}
// For loop
for(x in list) {
      print(x)
}
```

Dart Functions

Ordinary functions

```
isEven(int number) {
        return number % 2 == 0
}

isEven(14);          // true
```

Dart Functions

Defaults and optionals

```
helloWorld(String name, { bool date = false }) {
        print(“Hello world, $name”);
        if (date) print(new DateTime.now());
}

helloWorld(“Mario”);
helloWorld(“Mario”, date: true);
```

Dart Class

Classes are pretty much like in Java, however they typically have a primary constructor:

```
class Animal {
      var name, legCount;
      Animal({this.name, this.legCount = 4});     // Constructor can be used to quickly initialize the
                                                  // object
}

var dog = new Animal(name: "dog");
var duck = new Animal(name: "duck", legCount: 2);
```

Dart allows much more such as inheritance, syntactic sugar, exceptions, explicit typing.

Check out the official documentation (https://dart.dev/) if you are interested or DartPad (https://dartpad.dev) for examples and a sandbox to test the code.

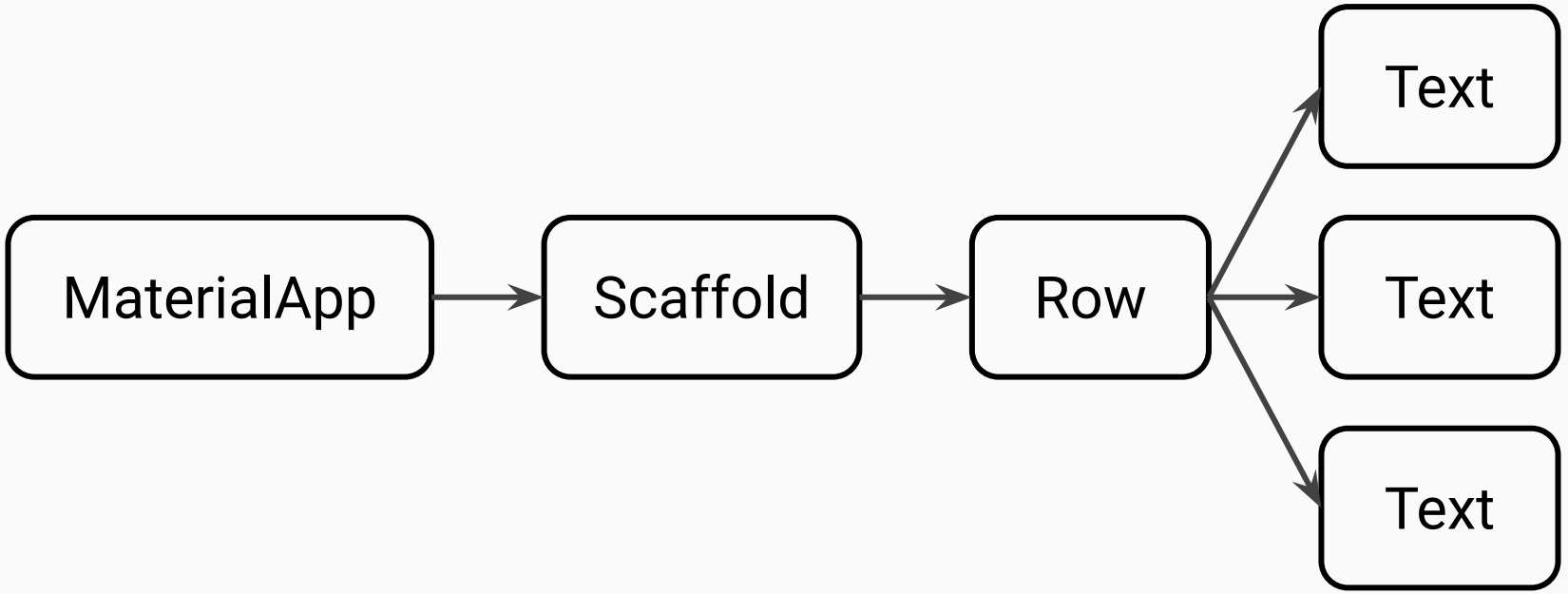
Flutter Philosophy

The philosophy of Flutter follows an approach similar to latest trends in web development: **reusable components**.

An application is built by nesting widgets.

```
Center(
      child: Text('Hello World'),
);
```

Flutter Widget Tree



Flutter Widgets

There are two types of widgets:

- **stateless:**
  - Do not manage any data
  - Only updates elements on screen
  - Should use as often as possible!
- **stateful**:
  - Maintain internal data
  - When data changes, the entire UI is re-rendered

Flutter
Main entry

The **main()** function is the entrypoint for a flutter app.

```
import ‘package:flutter/material.dart’

void main() {
      runApp( . . .);
}
```

The runApp function takes a **widget** as input and creates the **widget tree**

```dart
import 'package:flutter/material.dart';

void main() {
    runApp( MyApp());
}

class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: "Welcome to Flutter",
            home: Scaffold(
                appBar: AppBar(
                    title: Text("Welcome to Flutter"),
                ),
                body: Center(
                    child: Text("That's it!"),
                ),
            ),
        );
    }
}
```

Flutter

Example of a simple app

```
void main() {
      runApp( MyApp());
}
```

We define a custom widget to encapsulate our simple app.

Flutter

Example of a simple app

The custom widget is a **stateless widget**, since we do not have any data requirement

```
class MyApp extends StatelessWidget {
    ...
}
```

Flutter

Example of a simple app

The method **build** is called by Flutter whenever the interface is updated for some reason.
It should **never** have side-effects besides building the widget.

```
@override
Widget build(BuildContext context) { … }
```

The given **BuildContext** parameter contains information about the location of the widget in the tree.

Flutter

Example of a simple app

A **widget** that wraps a number of widgets that are commonly required for Material Design applications.

```
return MaterialApp( ... );
```

For example, it automatically creates a bar on top of the app with a title.

Flutter

Example of a simple app

A **widget** that wraps a number of widgets that are commonly required for Material Design applications.

```
return MaterialApp( ... );
```

Flutter

Example of a simple app

MaterialApp takes a number of parameters that defines the widgets look. Some include:

- color - sets the color of the app
- title - sets the title of the app
- home - configures the main content of the app
- See more at https://api.flutter.dev/flutter/material/MaterialApp-class.html

Flutter

Example of a simple app

```
return MaterialApp(
        title: “Welcome to Flutter”,
        home: Scaffold(
                appBar: AppBar(
                        title: Text(“Welcome to Flutter”),
                ),
                body: Center(
                        child: Text(“That’s it!”),
                ),
        ),
);
```

Set the title of the app

Flutter

Example of a simple app

```
return MaterialApp(
        title: “Welcome to Flutter”,
        home: Scaffold(
                appBar: AppBar(
                        title: Text(“Welcome to Flutter”),
                ),
                body: Center(
                        child: Text(“That’s it!”),
                ),
        ),
);
```

Implements the basic Material Design visual layout structure, with a top bar, a bottom bar and a body.

Flutter

Example of a simple app

```
return MaterialApp(
        title: “Welcome to Flutter”,
        home: Scaffold(
                appBar: AppBar(
                        title: Text(“Welcome to Flutter”),
                ),
                body: Center(
                        child: Text(“That’s it!”),
                ),
        ),
);
```

Builds the top bar (**AppBar**) by adding a text field inside.

Flutter

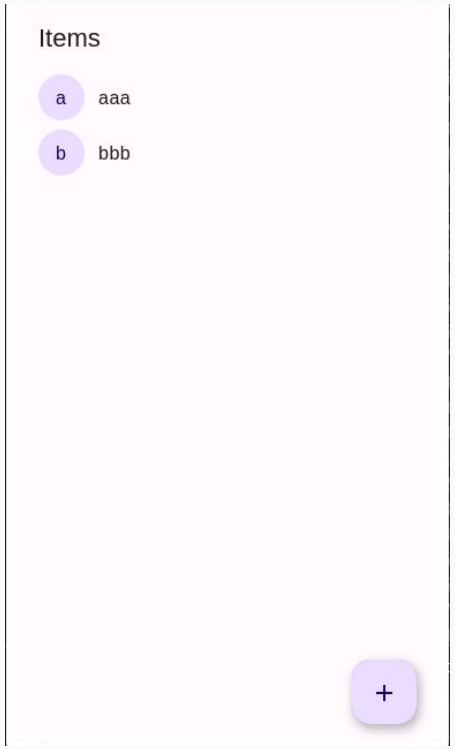
Example of a simple app

```
return MaterialApp(
        title: “Welcome to Flutter”,
        home: Scaffold(
                appBar: AppBar(
                        title: Text(“Welcome to Flutter”),
                ),
                body: Center(
                        child: Text(“That’s it!”),
                ),
        ),
);
```

Defines the body of the app as centered content (**Center**) where the content is a text label.

Flutter

Example of a stateful widget



A simple app to add items to a list.

The state represents the items in the list.

Flutter

Example of a stateful widget

```dart
import 'package:flutter/material.dart';

class Item extends StatelessWidget {
 Item({
       Key? key,
       required this.name,
 }) : super(key: key);

 final String name;

 @override
 Widget build(BuildContext context) {...}
```

Define an item as a stateless widget

Flutter

Example of a stateful widget

```dart
import 'package:flutter/material.dart';

class Item extends StatelessWidget {
 Item({
      Key? key,
      required this.name,
 }) : super(key: key);

 final String name;

 @override
 Widget build(BuildContext context) {...}
```

the StatelessWidget class takes a key parameter.

Key? is Dart's way of expressing optional parameters

Flutter

Example of a stateful widget

```dart
import 'package:flutter/material.dart';

class Item extends StatelessWidget {
 Item({
       Key? key,
       required this.name,
 }) : super(key: key);

 final String name;

 @override
 Widget build(BuildContext context) {...}
```

with **super** we call the constructor of the extended StatelessWidget

Flutter

Example of a stateful widget

```dart
import 'package:flutter/material.dart';

class Item extends StatelessWidget {
 Item({
      Key? key,
      required this.name,
 }) : super(key: key);

 final String name;

 @override
 Widget build(BuildContext context) {...}
```

name is the content
of the item in the list

Flutter

Example of a stateful widget

```
@override
Widget build(BuildContext context) {
      return ListTile(
            leading: CircleAvatar(
                  child: Text(name[0]),
            ),
            title: Text(name),
      );
 }
```

ListTile builds the item of a list

Flutter

Example of a stateful widget

```
@override
Widget build(BuildContext context) {
      return ListTile(
            leading: CircleAvatar(
                  child: Text(name[0]),
            ),
            title: Text(name),
      );
 }
```

CircleAvatar will display an icon with a custom text.

In this case, the first letter of the item.

Flutter

Example of a stateful widget

```
class ItemList extends StatefulWidget {
        @override
        State<ItemList> createState() {
                return new _ItemListState();
        }
}
```

A stateful widget must define a **createState** method, which instantiate the object **_ItemListState**

Flutter

Example of a stateful widget

```
class _ItemListState extends State<ItemList> {
        final TextEditingController _textController = TextEditingController();
        final List<String> _strings = <String>[];

        @override
        Widget build(BuildContext context) {
                ...
        }

        ...
}
```

The state of a stateful widget (conventionally written with a leading _) extends the class State

Flutter

Example of a stateful widget

```
class _ItemListState extends State<ItemList> {
        final TextEditingController _textController = TextEditingController();
        final List<String> _strings = <String>[];

        @override
        Widget build(BuildContext context) {
                …
        }

        …
}
```

The actual state is just a list of strings

Flutter

Example of a stateful widget

```
class _ItemListState extends State<ItemList> {
        final TextEditingController _textController = TextEditingController();
        final List<String> _strings = <String>[];

        @override
        Widget build(BuildContext context) {
                ...
        }

        ...
}
```

TextEditingController allows reading the content of a text field easily. More on this later…

Flutter

Example of a stateful widget

```
class _ItemListState extends State<ItemList> {
        final TextEditingController _textController = TextEditingController();
        final List<String> _strings = <String>[];

        @override
        Widget build(BuildContext context) {
                ...
        }

        ...
}
```

For the rest, a stateful widget works the same as a stateless one

Flutter

Example of a stateful widget

```
@override
Widget build(BuildContext context) {
        return Scaffold(
                appBar: AppBar(title: Text('Items')),
                        body: ListView(
                                children: _strings.map((String i) {
                                return Item(name: i);
                        }).toList(),
                ),
                floatingActionButton: FloatingActionButton(
                        onPressed: () => _displayDialog(),
                        tooltip: 'Add Item',
                        child: Icon(Icons.add)
                ),
        );
 }
```

The body of the app is a list view, which displays list items.

Each item contains a string

Flutter

Example of a stateful widget

```
@override
Widget build(BuildContext context) {
        return Scaffold(
                appBar: AppBar(title: Text('Items')),
                        body: ListView(
                                children: _strings.map((String i) {
                                return Item(name: i);
                        }).toList(),
                ),
                floatingActionButton: FloatingActionButton(
                        onPressed: () => _displayDialog(),
                        tooltip: 'Add Item',
                        child: Icon(Icons.add)
                ),
        );
 }
```

A floating button is added, with a ‘+’ icon.

When clicked, a dialog to add text is shown.

Flutter

Example of a stateful widget

```
@override
Widget build(BuildContext context) {
        return Scaffold(
                appBar: AppBar(title: Text('Items')),
                        body: ListView(
                                children: _strings.map((String i) {
                                return Item(name: i);
                        }).toList(),
                ),
                floatingActionButton: FloatingActionButton(
                        onPressed: () => _displayDialog(),
                        tooltip: 'Add Item',
                        child: Icon(Icons.add)
                ),
        );
 }
```

Show the dialog using a private method defined in the widget

Flutter

Example of a stateful widget

```
Future<void> _displayDialog() async {
  return showDialog<void>(
    context: context,
     barrierDismissible: false,
      builder: (BuildContext context) {
       return AlertDialog(
         title: const Text('Add an item'),
          content: TextField(controller: _textFieldController),
           actions: <Widget>[
            TextButton(
             child: const Text('Add'),
              onPressed: () {
               Navigator.of(context).pop();
                setState(() {
                   _strings.add(_textFieldController.text);
                });
},),], ); }, ); }
```

Future instructs Flutter that the returned type might be an error, since it is an asynchronous function

Flutter

Example of a stateful widget

```
Future<void> _displayDialog() async {
 return showDialog<void>(
  context: context,
   barrierDismissible: false,
    builder: (BuildContext context) {
     return AlertDialog(
       title: const Text('Add an item'),
        content: TextField(controller: _textFieldController),
         actions: <Widget>[
          TextButton(
           child: const Text('Add'),
            onPressed: () {
             Navigator.of(context).pop();
              setState(() {
                 _strings.add(_textFieldController.text);
                });
},), ], ); }, ); }
```

showDialog shows a Widget, which is a dialog window

Flutter

Example of a stateful widget

```
Future<void> _displayDialog() async {
  return showDialog<void>(
    context: context,
     barrierDismissible: false,
      builder: (BuildContext context) {
       return AlertDialog(
         title: const Text('Add an item'),
          content: TextField(controller: _textFieldController),
           actions: <Widget>[
            TextButton(
             child: const Text('Add'),
              onPressed: () {
               Navigator.of(context).pop();
                setState(() {
                   _strings.add(_textFieldController.text);
                });
},), ], ); }, ); }
```

provide to the widget the context of the main app

Flutter

Example of a stateful widget

```
Future<void> _displayDialog() async {
 return showDialog<void>(
  context: context,
   barrierDismissible: false,
   builder: (BuildContext context) {
    return AlertDialog(
      title: const Text('Add an item'),
       content: TextField(controller: _textFieldController),
        actions: <Widget>[
          TextButton(
           child: const Text('Add'),
            onPressed: () {
             Navigator.of(context).pop();
              setState(() {
               _strings.add(_textFieldController.text);
              });
},), ], ); }, ); }
```

since showDialog shows a widget, a builder method must be defined to configure its UI elements

Flutter

Example of a stateful widget

```
Future<void> _displayDialog() async {
 return showDialog<void>(
  context: context,
   barrierDismissible: false,
   builder: (BuildContext context) {
    return AlertDialog(
      title: const Text('Add an item'),
       content: TextField(controller: _textFieldController),
        actions: <Widget>[
          TextButton(
           child: const Text('Add'),
            onPressed: () {
             Navigator.of(context).pop();
              setState(() {
               _strings.add(_textFieldController.text);
              });
},),], ); }, ); }
```

AlertDialog is the type of widget displayed, but Flutter defines other types of dialogs

Flutter

Example of a stateful widget

```
Future<void> _displayDialog() async {
 return showDialog<void>(
  context: context,
   barrierDismissible: false,
   builder: (BuildContext context) {
    return AlertDialog(
      title: const Text('Add an item'),
       content: TextField(controller: _textFieldController),
       actions: <Widget>[
        TextButton(
         child: const Text('Add'),
         onPressed: () {
          Navigator.of(context).pop();
          setState(() {
           _strings.add(_textFieldController.text);
         });
},), ], ); }, ); }
```

AlertDialog is the type of widget displayed, but Flutter defines other types of dialogs

Flutter

Example of a stateful widget

```
Future<void> _displayDialog() async {
 return showDialog<void>(
  context: context,
   barrierDismissible: false,
   builder: (BuildContext context) {
    return AlertDialog(
      title: const Text('Add an item'),
       content: TextField(controller: _textFieldController),
       actions: <Widget>[
        TextButton(
         child: const Text('Add'),
         onPressed: () {
          Navigator.of(context).pop();
          setState(() {
           _strings.add(_textFieldController.text);
         });
},), ], ); }, ); }
```

The content of the widget is a textual field, with the controller that we specified.

This later allows us to read the text content.

Flutter

Example of a stateful widget

```
Future<void> _displayDialog() async {
 return showDialog<void>(
  context: context,
   barrierDismissible: false,
   builder: (BuildContext context) {
    return AlertDialog(
      title: const Text('Add an item'),
       content: TextField(controller: _textFieldController),
       actions: <Widget>[
        TextButton(
         child: const Text('Add'),
         onPressed: () {
          Navigator.of(context).pop();
          setState(() {
           _strings.add(_textFieldController.text);
          });
},), ], ); }, ); }
```

We define the elements at the bottom of the dialog using a button containing text

Flutter

Example of a stateful widget

```
Future<void> _displayDialog() async {
 return showDialog<void>(
  context: context,
   barrierDismissible: false,
   builder: (BuildContext context) {
    return AlertDialog(
      title: const Text('Add an item'),
       content: TextField(controller: _textFieldController),
       actions: <Widget>[
        TextButton(
         child: const Text('Add'),
         onPressed: () {
          Navigator.of(context).pop();
          setState(() {
           _strings.add(_textFieldController.text);
          });
},), ], ); }, ); }
```

When clicked, the dialog is hidden

Flutter

Example of a stateful widget

```
Future<void> _displayDialog() async {
 return showDialog<void>(
  context: context,
   barrierDismissible: false,
   builder: (BuildContext context) {
    return AlertDialog(
      title: const Text('Add an item'),
       content: TextField(controller: _textFieldController),
       actions: <Widget>[
        TextButton(
         child: const Text('Add'),
         onPressed: () {
          Navigator.of(context).pop();
          setState(() {
            _strings.add(_textFieldController.text);
          });
},),], ); }, ); }
```

And finally the state is updated.

Calling setState triggers the Flutter framework and the UI is redrawn.

Flutter

Example of a stateful widget

```
class MyApp extends StatelessWidget {
      @override
      Widget build(BuildContext context) {
            return new MaterialApp(
                  title: 'Item List',
                  home: new ItemList(),
            );
      }
}

void main() => runApp(new MyApp());
```

The entrypoint of the app is the same as a stateless app.

Flutter Takeaways

- Flutter follows an appealing approach to App development:
  - Everything revolves around the UI
  - It is similar to popular approaches (React, etc.) for web development
  - It is robust and Google's support will likely make it much better as time goes on
- However, the abstractions that allow this flexibility comes at a cost
  - It is less straightforward to rely on the devices API (accelerometer, gyroscope, custom drivers, etc.)
  - Defining a proper backend is hard. The main idea is that the app will be a dashboard over an external API.

Questions?

nicolas.lazzari3@unibo.it