

Biomedical Wearable Technologies for Healthcare and Wellbeing

Hello, Flutter!

A.Y. 2022-2023

Giacomo Cappon



Outline

- **Flutter**

- Creating a new project
 - App dissection
 - Expanding our first app
-
- Homework & Resources
-
- Project overview

Flutter

➤ What is Flutter?

- Simply a declarative framework for Dart

➤ Why this choice?

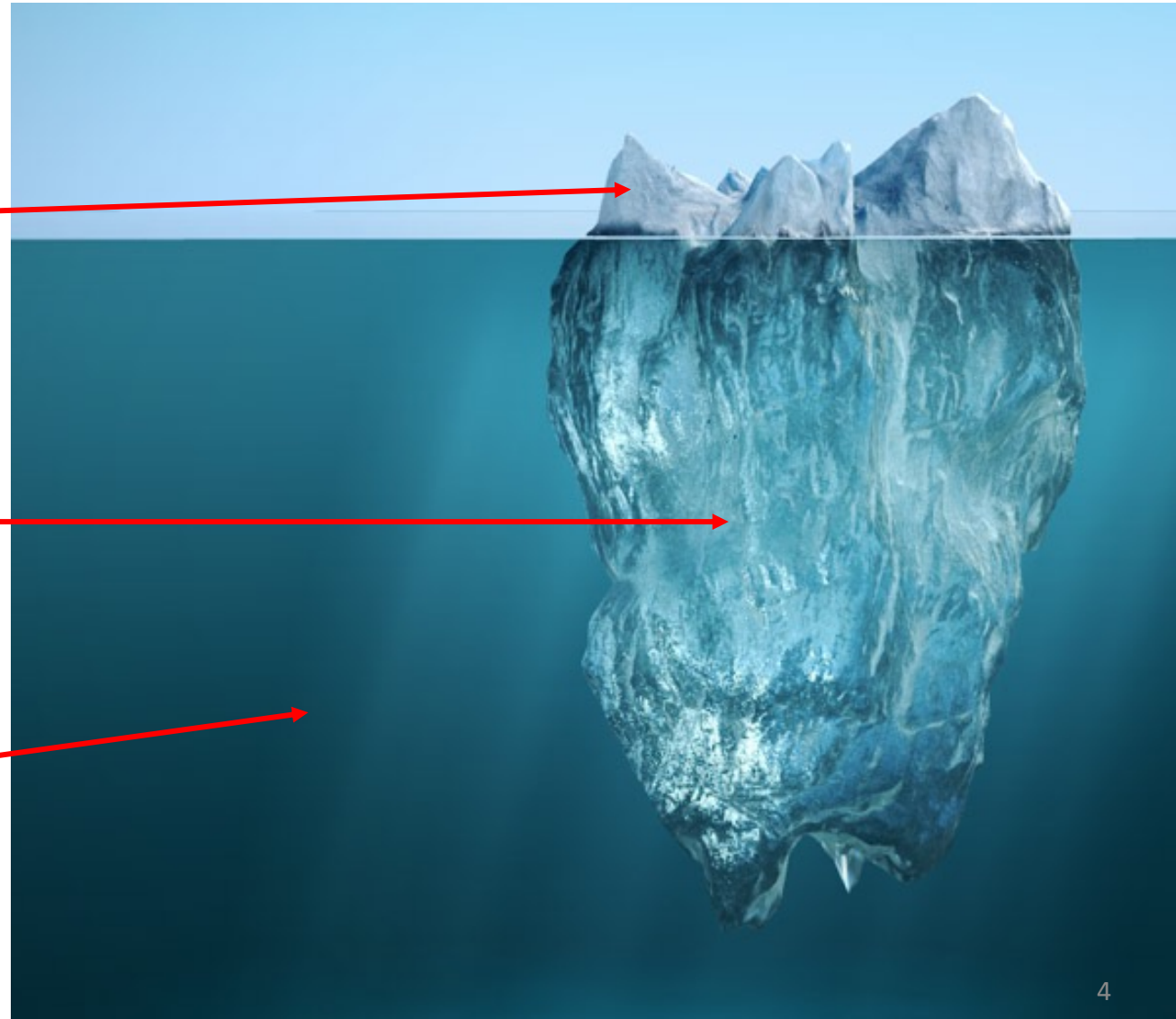
- State-of-the-art and Google-maintained
- Single codebase for iOS and Android (and Mac, Windows, Web)
- Relatively easy to learn
- Lots of examples
- Fastly growing job market

➤ Today we will create and study our first Flutter app



Before starting...

- What you'll see in these labs about Flutter capabilities
- Actual things that you will probably use
- Flutter possibilities



Outline

- Flutter
- **Creating a new project**
- App dissection
- Expanding our first app

- Homework & Resources

- Project overview

Hello, Flutter!

- In this lesson, we will run and analyse our first Flutter app
- First, setup VS Code to work with Flutter (**this should have already been done**)
 1. Start VS Code.
 2. Invoke **View > Command Palette....**
 3. Type “install”, and select **Extensions: Install Extensions**.
 4. Type “flutter” in the extensions search field, select **Flutter** in the list, and click **Install**. This also installs the required Dart plugin.
- Then, create the app
 1. Invoke **View > Command Palette**.
 2. Type “flutter”, and select the **Flutter: New Project**.
 3. Select **Application**
 4. Select the parent directory that will contain the app
 5. Enter a project name, such as “my_first_app”, and press **Enter**.
 6. Wait for project creation to complete and the main.dart file to appear.

Hello, Flutter!

- Replace all the code of main.dart with

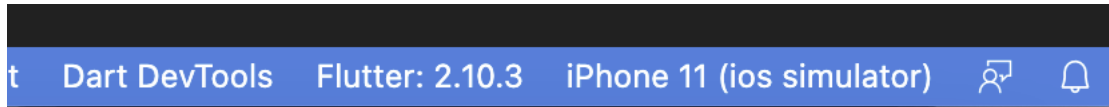
```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
} //main
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(title: const Text('Welcome to Flutter'),),
        body: const Center(child: Text('Hello World'),),),
    );
  } //build
} //MyApp
```

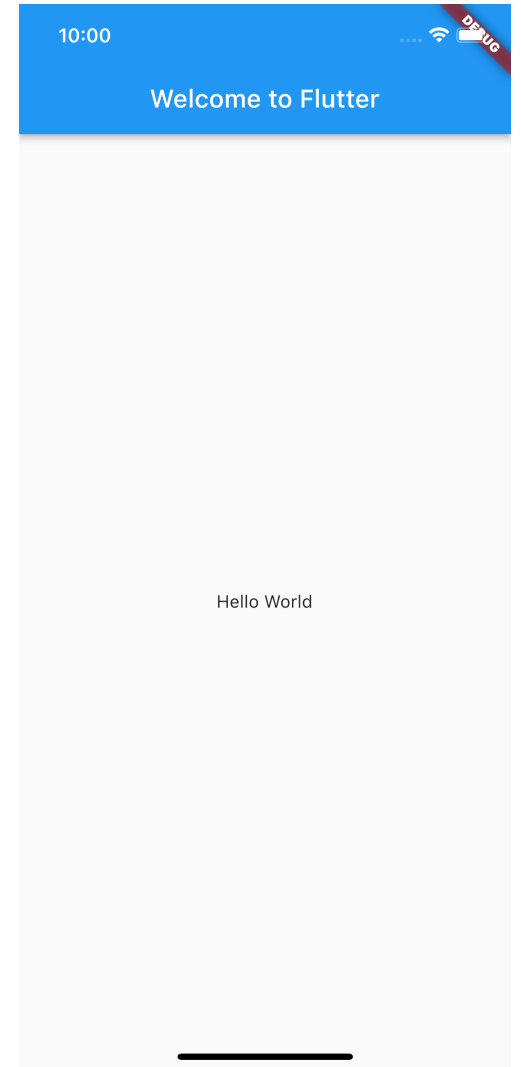
Hello, Flutter!

➤ Finally, run the app!

1. Locate the VS Code status bar (the blue bar at the bottom of the window):



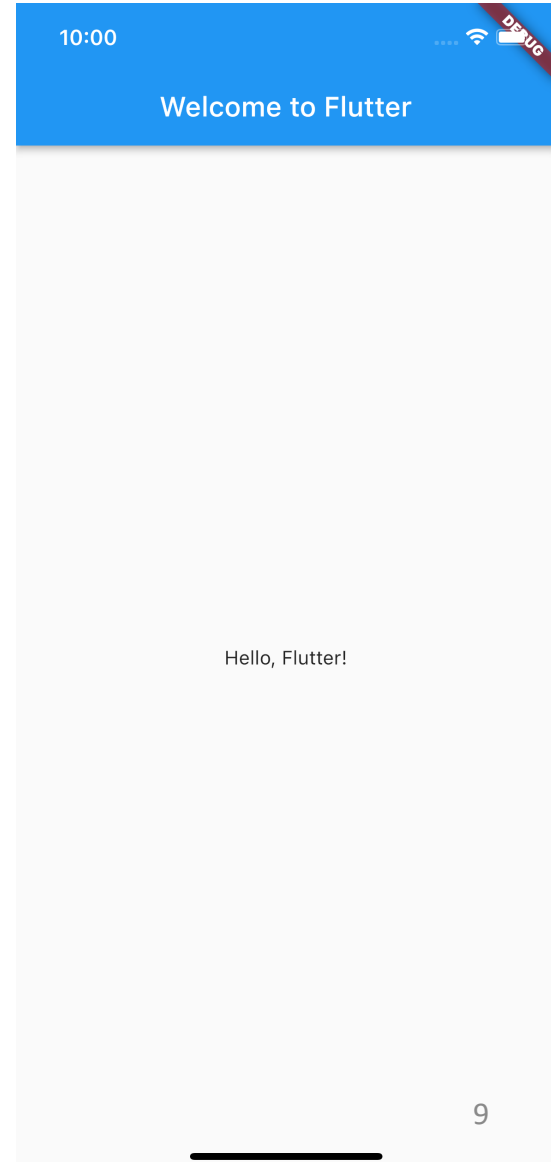
2. Select a mobile device from the **Device Selector** area
3. Invoke **Run > Start Debugging** or press **F5**
4. Wait for the app to launch — progress is printed in the **Debug Console** view.
5. After the app build completes, you'll see the starter app on your device.



A great feature: Hot reload

- Dart offers a fast development cycle with *Stateful Hot Reload*, the ability to reload the code of a live running app without restarting or losing app state. Make a change to app source, tell your IDE or command-line tool that you want to hot reload, and see the change in your simulator, emulator, or device.

- Try that!
 1. Open lib/main.dart.
 2. Change the string
 `'Hello World'`
 with
 `'Hello, Flutter!'`
 3. Save your changes: invoke **Save All**, or click **Hot Reload**
 4. You'll see the updated string in the running app almost immediately.



Outline

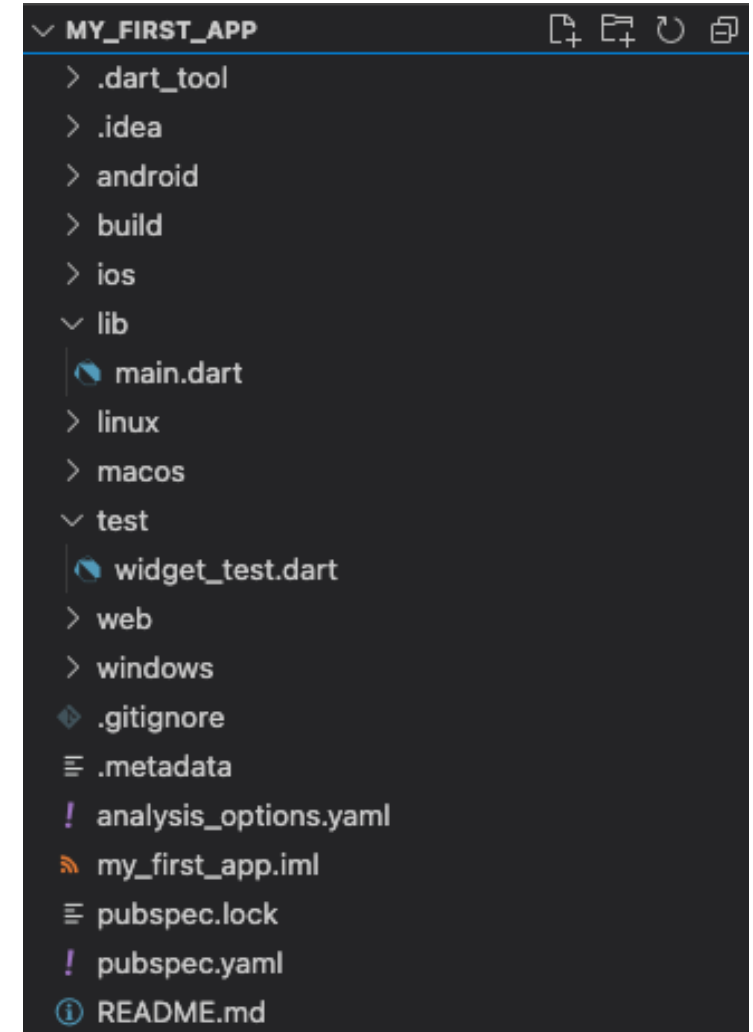
- Flutter
- Creating a new project
- **App dissection**
- Expanding our first app
- Homework & Resources
- Project overview

Let's dissect the app

- Let's understand what we have done.

Let's dissect the app – Project folder

- First, what's inside the project folder?
- Important things
 - **lib folder:** it contains the app source code
 - **main.dart file:** the entry point for the compiler
 - **pubspec.yaml file:** it specifies high level app features as well as listing which third party libraries our app needs and uses
 - **README.md file:** a markdown file describing the app
- (Less) Important things
 - **android/ios/linux/macOS/windows/web folders:** where native specific code can be defined if needed
 - **test folder:** where to put code for running automatic testers
- (Even less) Important things
 - All other folders and files are very use case specific and probably you will never use those in this course. If you are curious...



Let's dissect the app – main.dart

➤ Let's understand the main.dart file.

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
} //main
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(title: const Text('Welcome to Flutter')),
        body: const Center(child: Text('Hello World'))),
    );
  } //build
} //MyApp
```

To run an app using the Flutter framework we can use the **runApp** method which takes a **Widget** object as an input.

What's a **Widget**?

Everything is a Widget

- In Flutter, almost everything is (inherits from) a Widget!
- A Widget is a building block for your user interface (UI). Using widgets is like combining Legos.
- More technically, a Widget is a sort of blueprint for displaying your app state.
- Widgets can be thought as a function of UI. Given a state, the build() method (that every custom Widget must override and implement) constructs the widget UI:



$$\text{UI} = f(\text{state})$$

Screen build

Let's dissect the app – main.dart

➤ In **bold** the Widgets of our app

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
} //main
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(title: const Text('Welcome to Flutter'),),
        body: const Center(child: Text('Hello World'),),),
    );
  } //build
} //MyApp
```

Key method for building the Widget that must be implemented

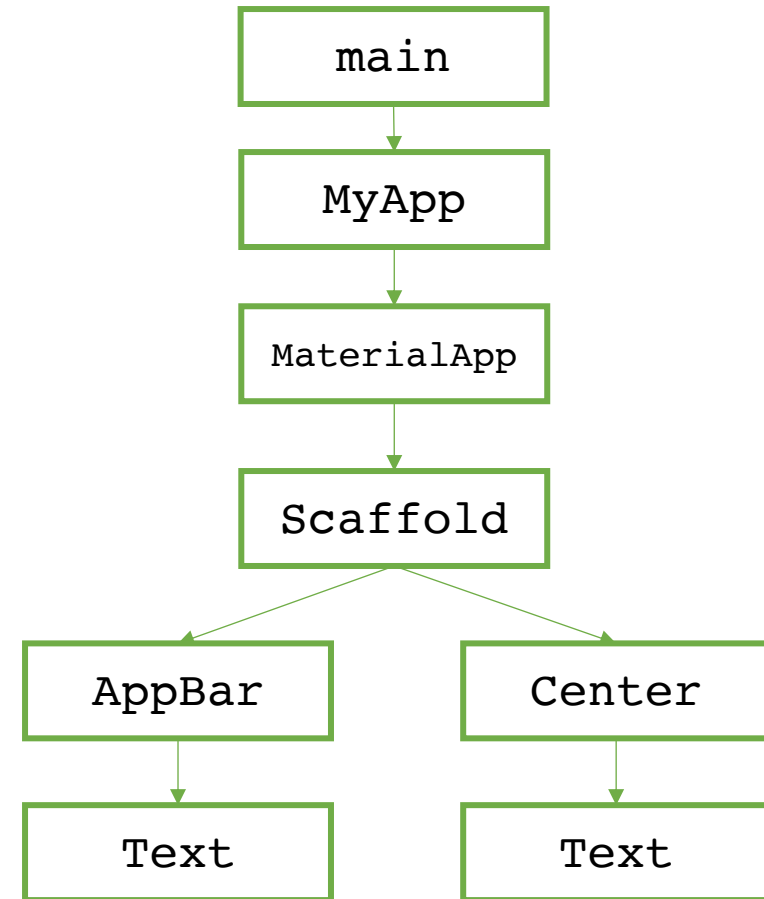
But how Widgets are combined together?

The Widget Tree

- Widgets are combined together using a **tree structure**

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
} //main
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(title: const Text('Welcome to Flutter'),),
        body: const Center(child: Text('Hello World'),),),
    );
  } //build
} //MyApp
```



State and widgets

➤ In bold the Widgets of our app

```
import 'package:flutter/material.dart';
void main() {
  runApp(const MyApp());
} //main
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(title: const Text('Welcome to Flutter'),),
        body: const Center(child: Text('Hello World'),),),
    );
  } //build
} //MyApp
```

MyApp is not just a Widget,
it is a StatelessWidget

Stateless vs. Stateful widgets

- **StatelessWidgets** are Widgets that always build the same way given a particular configuration and ambient state. So, they never re-build while they are displayed to the user (their lifetime).
- **StatefulWidget** for widgets that can build differently several times over their lifetime.
- You can think about StatelessWidget as a sort of constant and StatefulWidget as a variable.

Let's dissect the app – pubspec.yaml

- pubspec.yaml contains high-level instructions for the development environment and information on the app

```
name: my_first_app
description: A new Flutter project.
publish_to: 'none'
version: 1.0.0+1
```

my_first_app information (name, description, version, ...)

```
environment:
  sdk: ">=2.19.4 <3.0.0"
```

Flutter sdk version to be used

```
dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.2
```

App dependencies: what the app needs in order to work: other packages? Other libraries? Put them here.

```
dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.0
```

App dependencies while developing the app

```
flutter:
  uses-material-design: true
```

Information for the Flutter environment such as where to find assets.

Outline

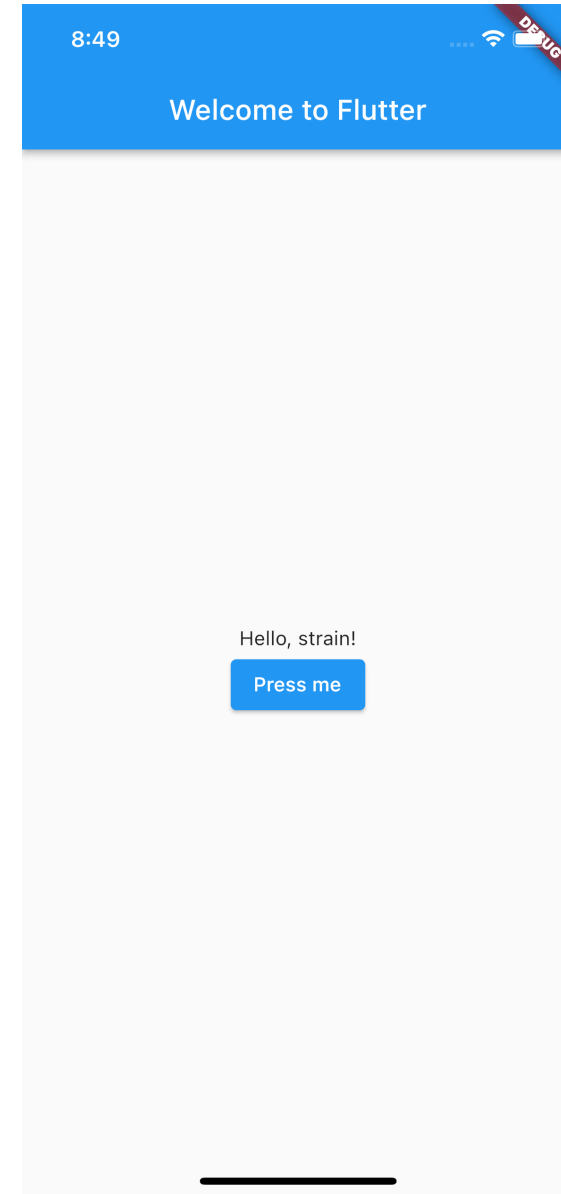
- Flutter
- Creating a new project
- App dissection
- **Expanding our first app**
- Homework & Resources
- Project overview

My first app with steroids

- Let's play with `my_first_app` and let's expand it
- We will learn how to:
 - Install an external package and add it as a dependency
 - Use the external package inside our app
 - StatefulWidget 101
 - How to modify the UI

My first app with steroids

- **Aim:** The result will be a very simple app that, each time a button is tapped, a new random "Hello" message is shown to the user.



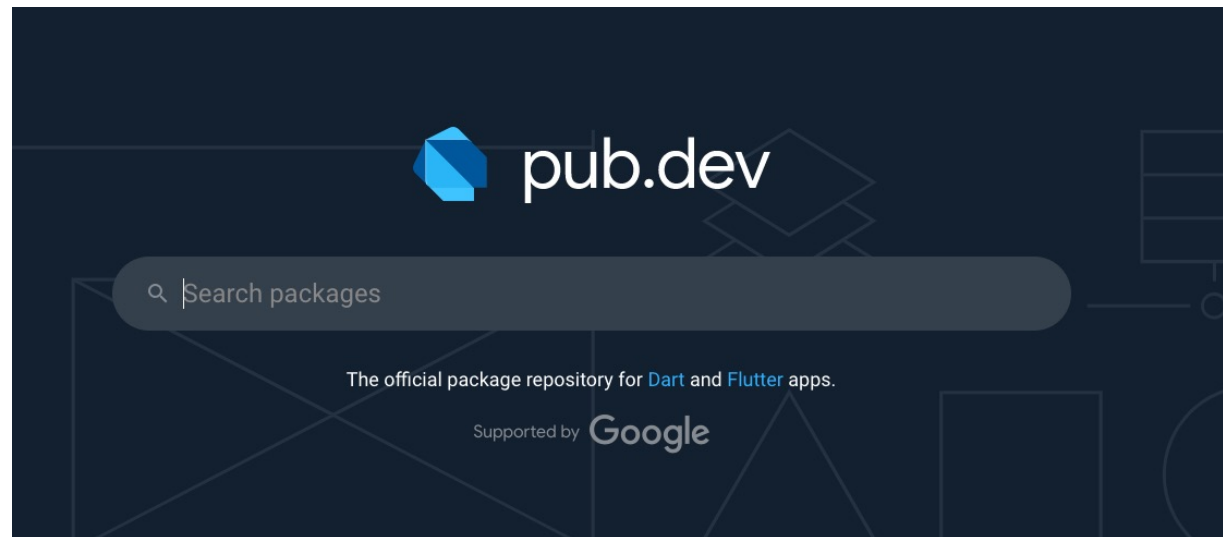
My first app with steroids

➤ Roadmap

1. Understand what to use to generate a random word
2. Generate a random word and check that everything is working
3. Display the word in the “Hello” message
4. Modify the UI to generate a new message each time a button is tapped

Solving point 1

- We do not want to code a random English word generator!
- On the Internet we can find a lot of already made code and ready-to-use packages that can fit your needs
- A place that we will visit often during this course is pub.dev:



This is the package I was looking for

- After some research, it seems like the **english_words** package can solve our needs
- It can generate words and words pairs!

How to use it? Docs!

Code is available too!

english_words 4.0.0

Published 9 months ago • [filiph.net](#) [Null safety](#)

[DART](#) [NATIVE](#) [JS](#) [FLUTTER](#) [ANDROID](#) [IOS](#) [LINUX](#) [MACOS](#) [WEB](#) [WINDOWS](#)

311

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)

english_words

build passing

A package containing the most ~5000 used English words and some utility functions.

Usage

Printing the top 50 most used nouns in the English language:

```
import 'package:english_words/english_words.dart';

main() {
  nouns.take(50).forEach(print);
}
```

Computing number of syllables in a word:

```
syllables('beautiful'); // 3
syllables('abatement'); // 3
syllables('zoology'); // 4
```

Generating 5 interesting 2-syllable word combinations:

```
generateWordPairs().take(5).forEach(print);
```

311 130 98%
LIKES PUB POINTS POPULARITY

Publisher

[filiph.net](#)

Metadata

Utilities for working with English words. Counts syllables, generates well-sounding word combinations, and provides access to the top 5000 English words by usage.

[Repository \(GitHub\)](#)
[View/report issues](#)

Documentation
[API reference](#)

License
MIT ([LICENSE](#))

Dependencies
[string_scanner](#)

More 25
[Packages that depend on english_words](#)

Including english_words in the app

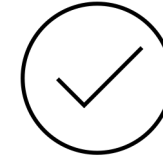
- Installing the english_words package in our app is very easy.
- By definition, it is a dependency right?
- So, let's add it under the dependency list of our app into pubspec.yaml
- After adding it, save pubspec.yaml and you will see VSCode running **flutter pub get** for you.
- Done!

```
...  
dependencies:  
  flutter:  
    sdk: flutter  
  
  cupertino_icons: ^1.0.2  
  
  english_words: ^4.0.0  
...
```

My first app with steroids

➤ Roadmap

1. Understand what to use to generate a random word
2. Generate a random word and check that everything is working
3. Display the word in the “Hello” message
4. Modify the UI to generate a new message each time a button is tapped



Generating a random word

- Let's add some line of code to main.dart to generate a word using the english_words package
- Modify the build method by adding

```
final word = WordPair.random().first;
```

before the return statement and run the app.

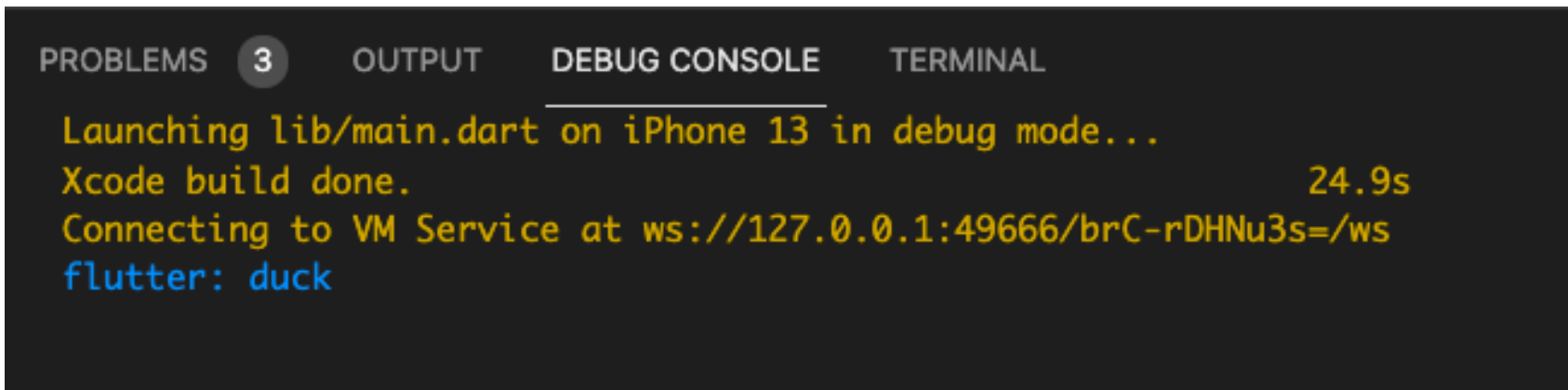
- Nothing it's happening. How to see if we are generating a random word?
- We can use the logger and the debug console!

Logging things

- Simply try to print the word value as a normal Dart program:

```
final word = WordPair.random().first;  
print(word);
```

- If you run the application now you will see something like this in the **Debug Console** of VS Code:



The screenshot shows the VS Code interface with the 'DEBUG CONSOLE' tab selected. The output text is as follows:

```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL  
Launching lib/main.dart on iPhone 13 in debug mode...  
Xcode build done. 24.9s  
Connecting to VM Service at ws://127.0.0.1:49666/brC-rDHNu3s=/ws  
flutter: duck
```

Logging things

- Every time you reload/restart the app



Restart button



Reload button

- ...you will see a different word

```
PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL
Restarted application in 416ms.
flutter: left
```

My first app with steroids

➤ Roadmap

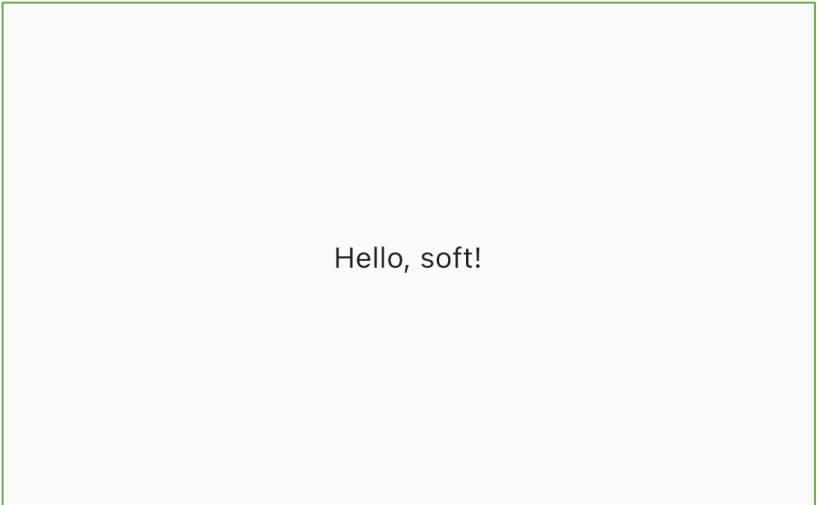
1. Understand what to use to generate a random word 
2. Generate a random word and check that everything is working 
3. Display the word in the “Hello” message
4. Modify the UI to generate a new message each time a button is tapped

Change the Hello message

- You should be able to solve this point by yourself now
- Simply, using string interpolation, change

`'Hello, Flutter!'` to `'Hello, $word!'`




- and save to reload the app and see the changes.



Hello, soft!

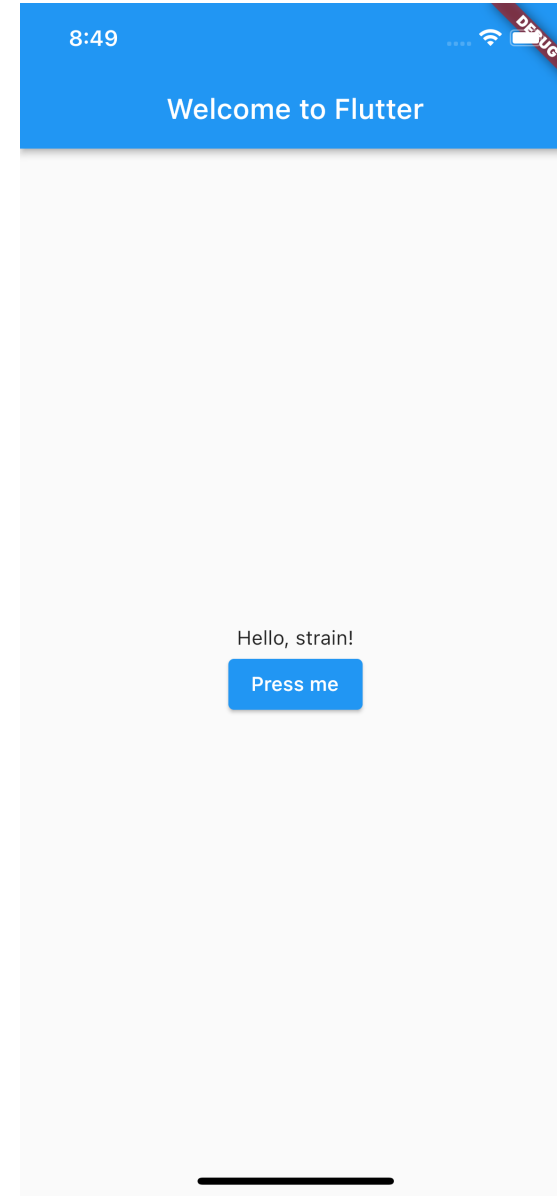
My first app with steroids

➤ Roadmap

1. Understand what to use to generate a random word 
2. Generate a random word and check that everything is working 
3. Display the word in the “Hello” message 
4. Modify the UI to generate a new message each time a button is tapped

Changing the UI

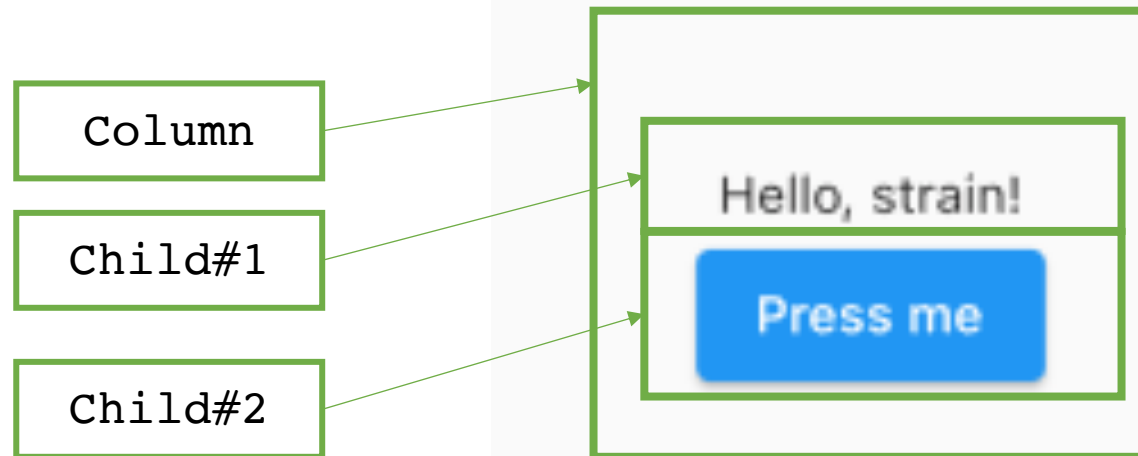
- Let's start by simply changing the UI
- We need to obtain something like
- Problems:
 1. How to add a button
 2. How to put it there



The Column Widget

- We can use the Column widget.
- It has a list of children (not like Text or Center or Scaffold)
- Children are lined up to a column from top to the bottom

```
Column(  
  children: [  
    Child#1,  
    Child#2,  
  ],  
);
```



Implement the new UI

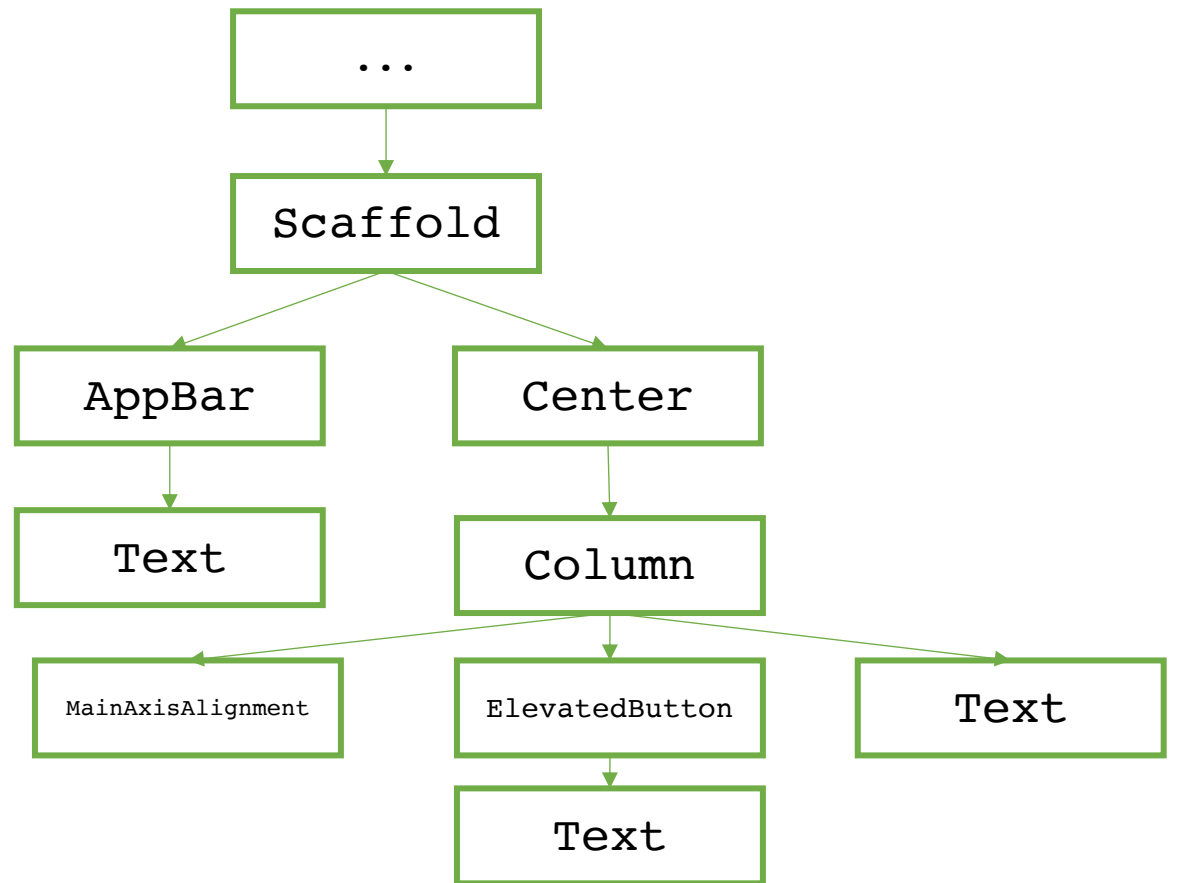
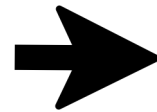
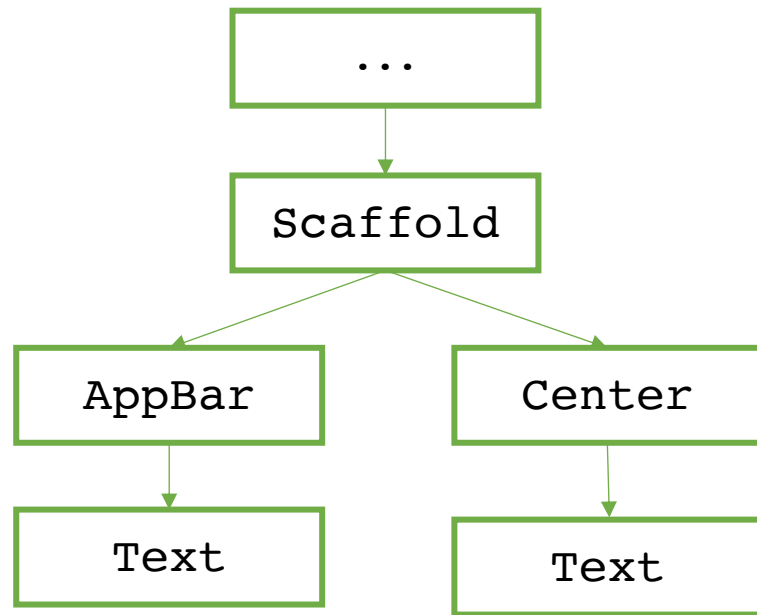
- Change the build method of MyApp to

```
Widget build(BuildContext context) {  
  final word = WordPair.random().first;  
  print(word);  
  return MaterialApp(  
    title: 'Welcome to Flutter',  
    home: Scaffold(  
      appBar: AppBar(title: const Text('Welcome to Flutter')),  
      body: Center(  
        child: Column(  
          mainAxisAlignment: MainAxisAlignment.center,  
          children: [  
            Text('Hello, $word!'),  
            ElevatedButton(onPressed: (){}, child: const Text('Press me')),  
          ],  
        ),  
      ),  
    ),  
  );  
}
```

//build

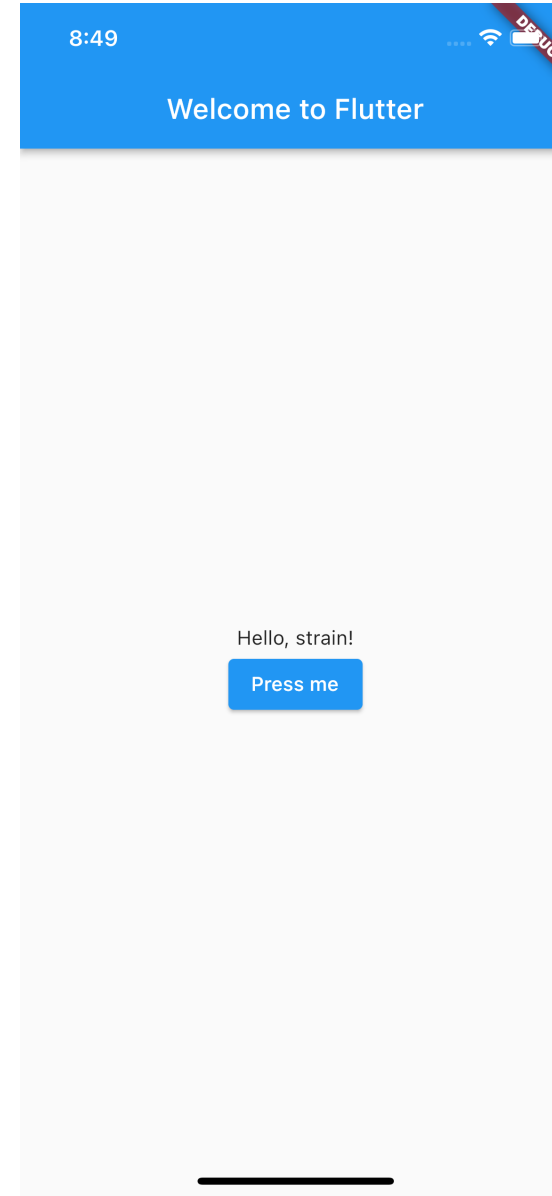
Different UI, different tree

- How the widget tree changed?



Changing the UI

- (New) Problem: How to change the message when we press the button?
- In other words: how to change the **app state** without reloading or restarting everything
- We need a **StatefulWidget**



StatefulWidget

- As we mentioned before, stateful widgets maintain state that might change during the lifetime of the widget.
- Implementing a stateful widget requires at least two classes:
 1. A **StatefulWidget** class that creates an instance of the `Widget` itself
 2. A **State** class: a class that manages the state of the **StatefulWidget**

The boilerplate code of a StatefulWidget

```
class RandomHello extends StatefulWidget{
  const RandomHello({Key? key}) : super(key: key);

  @override
  _RandomHelloState createState() => _RandomHelloState();
} //RandomHello

class _RandomHelloState extends State<RandomHello>{

  @override
  Widget build(BuildContext buildContext){
    //return some widget
  } //build

} // _RandomHelloState
```


Refactoring the UI - RandomHello

- Let's copy some code into the build method new Widget

```
...
class _RandomHelloState extends State<RandomHello>{

  @override
  Widget build(BuildContext buildContext){
    final word = WordPair.random().first;
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Text('Hello, $word!'),
        ElevatedButton(onPressed: (){}, child: const
Text('Press me')),
      ],
    );
  } //build

} // _RandomHelloState
```

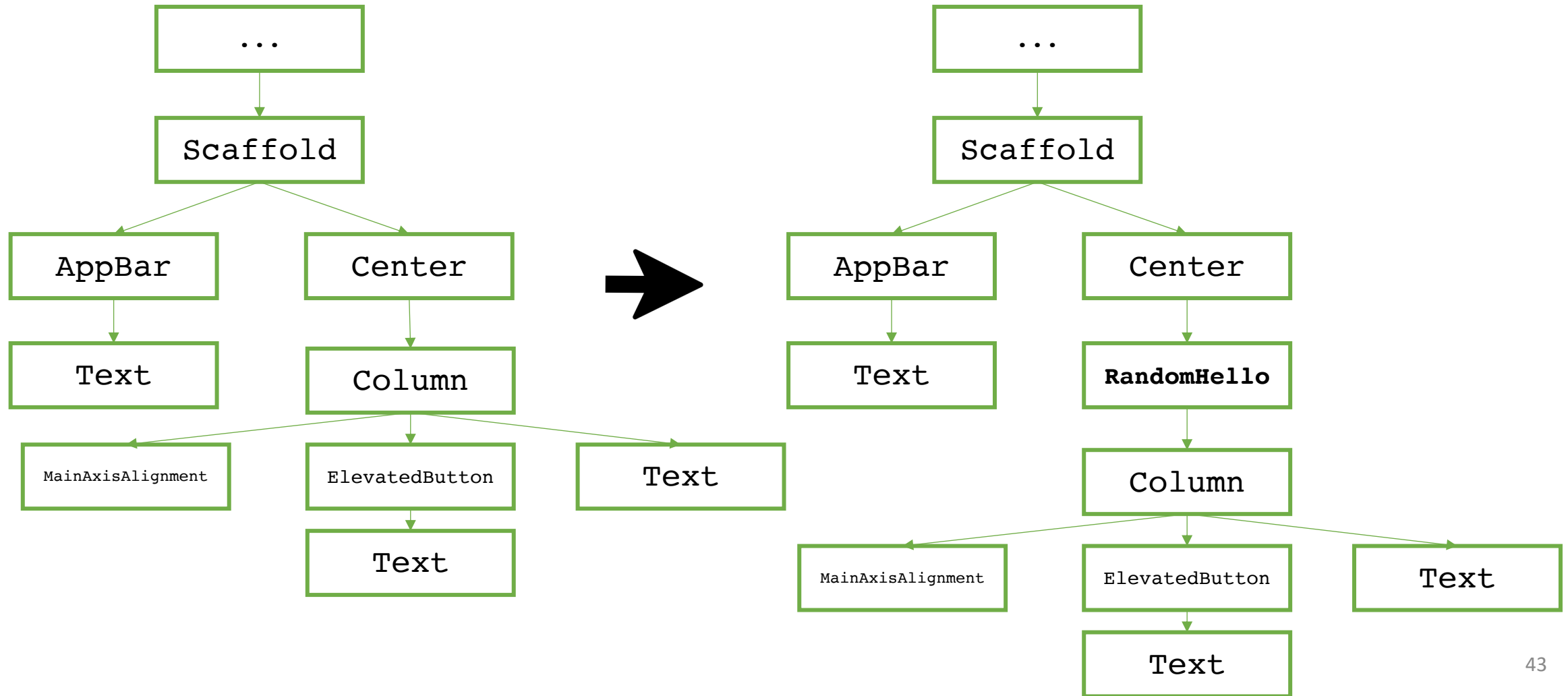
Refactoring the UI - MyApp

➤ Now let's refactor the MyApp code

```
class MyApp extends StatelessWidget {  
  const MyApp({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'Welcome to Flutter',  
      home: Scaffold(  
        appBar: AppBar(title: const Text('Welcome to Flutter')),  
        body: const Center(child: RandomHello()),  
      ),  
    );  
  } //build  
  
} //MyApp
```

Same UI, different tree

- The UI should look like the same as before, but we have a new widget tree



void initState(){} ---

- Let's do some changes to RandomHello to make it more **stateful**

```
class _RandomHelloState extends State<RandomHello>{
```

```
    String? _word; ←
```

_word will represent the state of the Widget

```
    @override
```

```
    void initState() { ←
```

initState is a special method that is called the first time the Widget is created. It is used (as its name suggest) to initialize the state of the Widget itself.

```
        _word = WordPair.random().first;
```

```
        super.initState();
```

```
    } //initState
```

```
    ...
```

setState((){})

- We are ready to implement the function to provide to onPressed

```
...
@override
Widget build(BuildContext buildContext){
  return Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Text('Hello, $_word!'),
      ElevatedButton(onPressed: _changeRandomWord, child: const
Text('Press me')),
    ],);
} //build

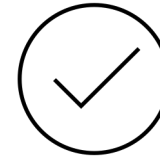
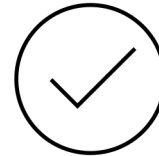
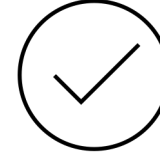
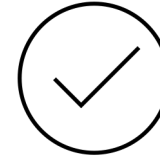
void _changeRandomWord(){
  setState(() { ←
    _word = WordPair.random().first;
  });
} // _changeRandomWord
...
```

setState is a special method that requires a callback function as input. setState notifies the Flutter framework that the state might be changed causing to delete and rebuild the widget itself.

My first app with steroids

➤ Roadmap

1. Understand what to use to generate a random word
2. Generate a random word and check that everything is working
3. Display the word in the “Hello” message
4. Modify the UI to generate a new message each time a button is tapped



Outline

- Flutter
- Creating a new project
- App dissection
- Expanding our first app
- **Homework & Resources**
- Project overview

Homework

- Play with our first example
- Get familiar with the structure of a Flutter project and how to install new packages using pubspec.yaml
- Get familiar with the concept of Widget
- To know what to do to create a StatelessWidget and a StatefulWidget
- Understanding the Flutter flow

Resources

➤ Introduction to Widgets

- <https://docs.flutter.dev/development/ui/widgets-intro>

➤ Write your first Flutter app, part 1 codelab

- <https://docs.flutter.dev/get-started/codelab>

➤ DevTools

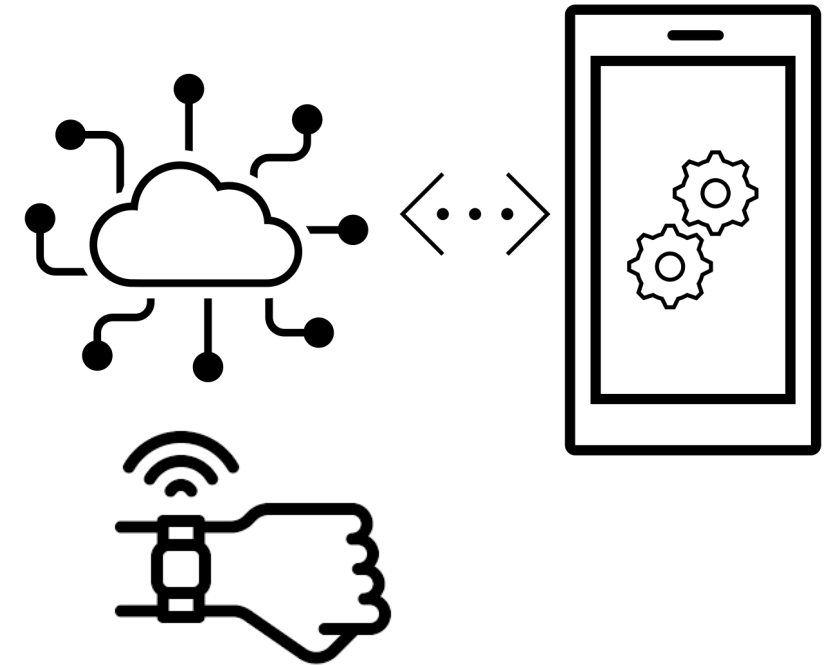
- <https://docs.flutter.dev/development/tools/devtools/overview>

Outline

- Flutter
- Creating a new project
- App dissection
- Expanding our first app
- Homework & Resources
- **Project overview**

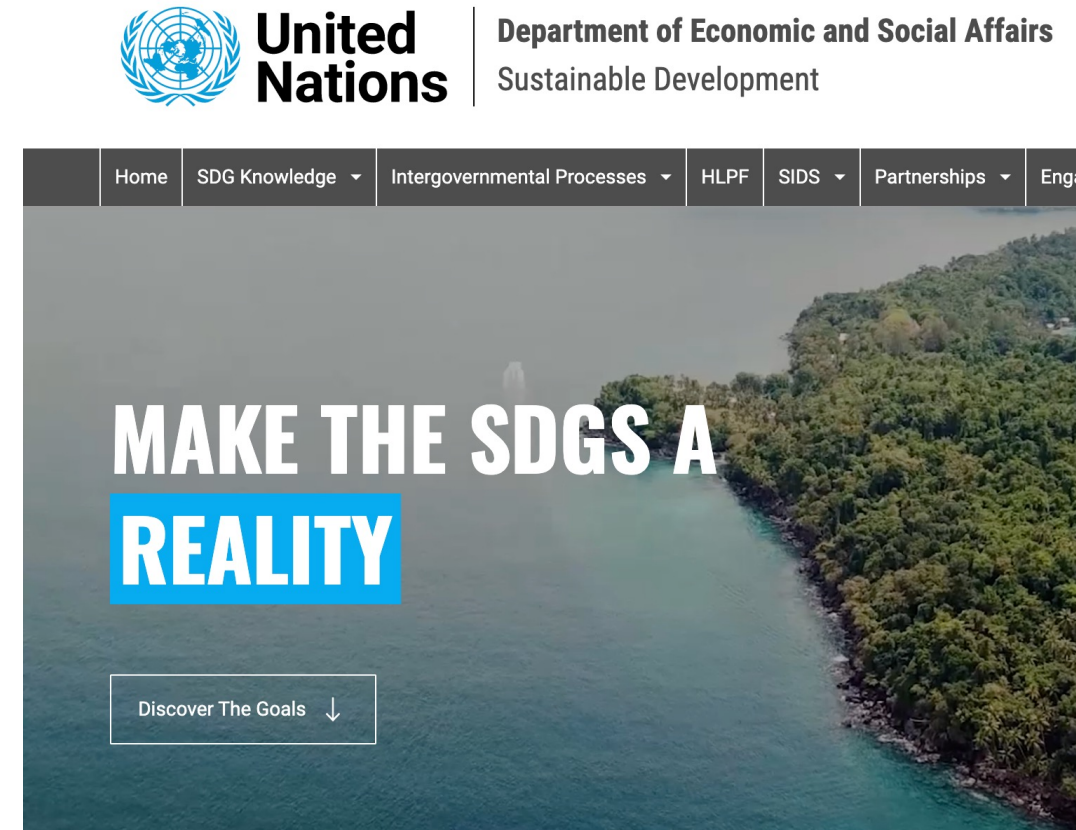
Project structure

- The project consists of building an app for iOS or Android that collects user data from a wearable device through Web APIs, stores them, visualizes them, and does some tricks with them.
- Core functionalities:
 - User authentication and management
 - Data collection
 - Data persistence
 - Data visualization and presentation
- Additional functionalities → It's up to you!
Some basic examples:
 - Run some analysis on data and provide suggestions to the user
 - Implement some literature algorithm
 - ...



A target for you app

- Your app must represent a solution for the current SDGs (Sustainable Development Goals) targets
- SDG are 17 topics representing the current big challenges humanity must face and solve



A target for you app

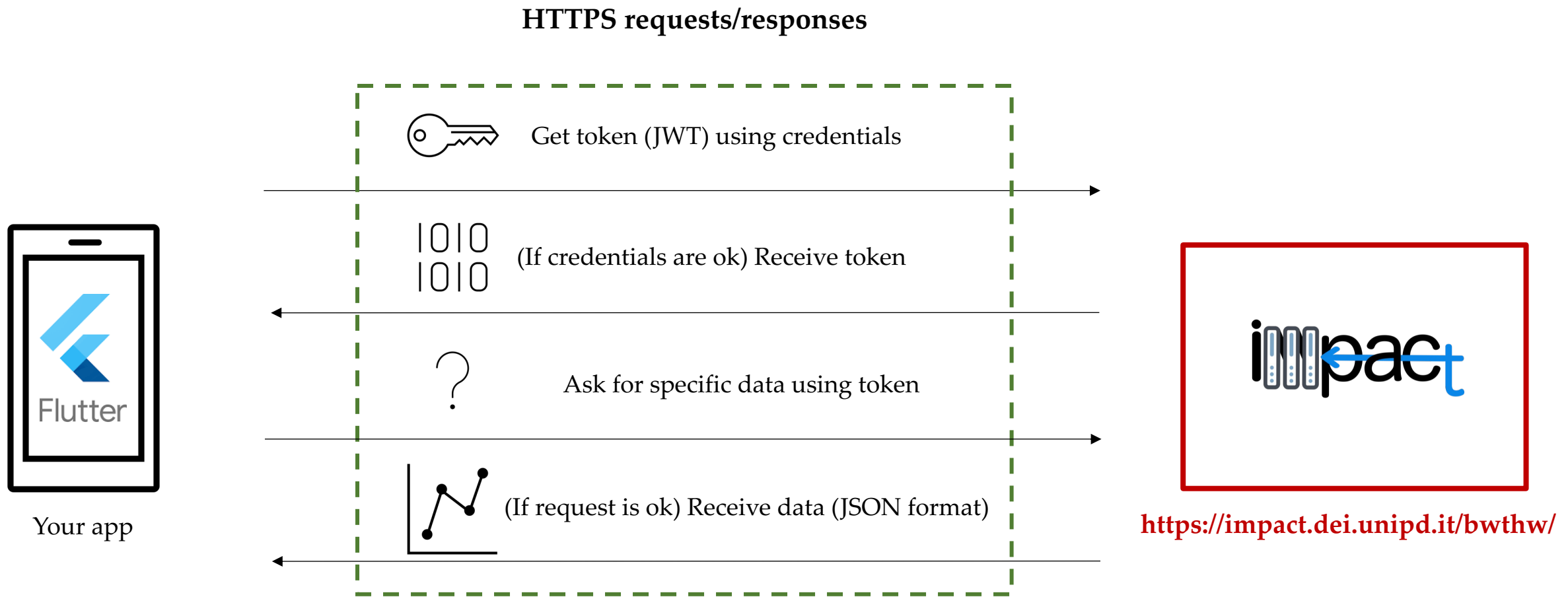
- These are the target you can choose from (see the moodle page for more details):
- **SDG 2: Target 2.2** (By 2030, end all forms of malnutrition, including achieving, by 2025, the internationally agreed targets on stunting and wasting in children under 5 years of age, and address the nutritional needs of adolescent girls, pregnant and lactating women and older persons)
 - **SDG 2: Target 2.3** (By 2030, double the agricultural productivity and incomes of small-scale food producers, in particular women, indigenous peoples, family farmers, pastoralists and fishers, including through secure and equal access to land, other productive resources and inputs, knowledge, financial services, markets and opportunities for value addition and non-farm employment).
 - **SDG 3: Target 3.4** (By 2030, reduce by one third premature mortality from non-communicable diseases through prevention and treatment and promote mental health and well-being).
 - **SDG 3: Target 3.5** (Strengthen the prevention and treatment of substance abuse, including narcotic drug abuse and harmful use of alcohol).
 - **SDG 4: Target 4.7** (By 2030, ensure that all learners acquire the knowledge and skills needed to promote sustainable development, including, among others, through education for sustainable development and sustainable lifestyles, human rights, gender equality, promotion of a culture of peace and non-violence, global citizenship and appreciation of cultural diversity and of culture's contribution to sustainable development).
 - **SDG 8: Target 8.9** (By 2030, devise and implement policies to promote sustainable tourism that creates jobs and promotes local culture and products).

Data source

- Data that you will use are collected by me using a Fitbit Versa 2
- Data are available starting from 09/02 and will be collected up to 30/11 and include the following data “types”:
 - Calories
 - Distance
 - Exercise sessions
 - Heart rate
 - Resting heart rate (Estimated)
 - Sleep
 - Steps
- Your project must use at least 1 type of data



How to get data - The IMPACT backend



- You will be provided with credentials (different for each group) and learn how to get data in the next lessons (lesson 9 and 10)

Focus on data – Calories

➤ Calories: a list with 1 data point exactly every minute

➤ Fields meaning:

- time: timestamp (hh:mm:ss format) of the calory entry
- value: the calories spent during the last 1 minute (rest + active)

```
[
  {
    "time": "00:00:00",
    "value": "1.29"
  },
  {
    "time": "00:01:00",
    "value": "1.29"
  },
  {
    "time": "00:02:00",
    "value": "1.29"
  },
  {
    "time": "00:03:00",
    "value": "1.55"
  },
  {
    "time": "00:04:00",
    "value": "1.29"
  },
  {
    "time": "00:05:00",
    "value": "1.29"
  },
  {
    "time": "00:06:00",
    "value": "1.29"
  },
  ...
]
```


Focus on data – Distance

- Distance: a list with 1 data point every minute or more depending on user's movement
- Fields meaning:
 - time: timestamp (hh:mm:ss format) of the distance entry
 - value: the distance travelled between the last timestamp and now (in cm)

```
[
  {
    "time": "00:03:00",
    "value": "0"
  },
  {
    "time": "00:50:00",
    "value": "0"
  },
  {
    "time": "01:28:00",
    "value": "0"
  },
  {
    "time": "01:32:00",
    "value": "1440"
  },
  {
    "time": "01:33:00",
    "value": "3110"
  },
  {
    "time": "01:34:00",
    "value": "0"
  },
  {
    "time": "01:53:00",
    "value": "0"
  },
  {
    "time": "03:09:00",
    "value": "0"
  },
  ...
]
```

Focus on data – Steps

- Steps: a list with 1 data point every minute or more depending on user's movement
- Fields meaning:
 - time: timestamp (hh:mm:ss format) of the step entry
 - value: the number of steps done between the last timestamp and now
- **Note:** there can be days without data

```
[
  {
    "time": "00:07:00",
    "value": "0"
  },
  {
    "time": "00:10:00",
    "value": "0"
  },
  {
    "time": "01:57:00",
    "value": "0"
  },
  {
    "time": "02:29:00",
    "value": "0"
  },
  {
    "time": "02:40:00",
    "value": "0"
  },
  {
    "time": "02:57:00",
    "value": "11"
  },
  {
    "time": "02:58:00",
    "value": "18"
  },
  {
    "time": "02:59:00",
    "value": "0"
  },
  ...
]
```

Focus on data – Heart rate

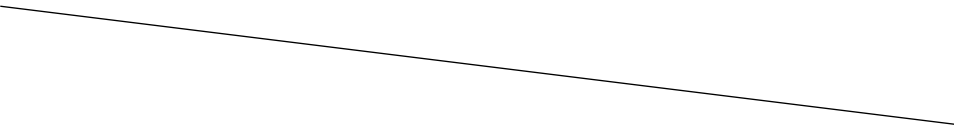
➤ Heart rate: a list with 1 data point every 5 seconds.

➤ Fields meaning:

- time: timestamp (hh:mm:ss format) of the heart rate entry
- value: the heart rate at the current time (in bpm)
- confidence: a number from 0 to 3 defining how much the reading is reliable (this probably depends on user movement, etc. You can ignore it)

➤ **Note:** there can be missing values

```
[
  {
    "time": "03:10:19",
    "value": 58,
    "confidence": 1
  },
  {
    "time": "03:10:24",
    "value": 54,
    "confidence": 3
  },
  {
    "time": "03:10:29",
    "value": 52,
    "confidence": 3
  },
  {
    "time": "03:10:44",
    "value": 52,
    "confidence": 3
  },
  {
    "time": "03:10:54",
    "value": 53,
    "confidence": 3
  },
  {
    "time": "03:11:09",
    "value": 54,
    "confidence": 3
  },
  {
    "time": "03:11:19",
    "value": 53,
    "confidence": 3
  },
  ...
]
```



Focus on data – Resting heart rate

- Resting heart rate: an entry per day.

```
{  
  "time": "00:00:00",  
  "value": 51.35,  
  "error": 17.86  
}
```

- Fields meaning:

- time: timestamp (hh:mm:ss format) of the resting heart rate entry (always 00:00:00)
- value: the estimated resting heart rate at the current time (in bpm)
- error: the estimate error (in %) (Can be ignored)

- **Note:** there can be days without data

Focus on data – Exercise

- Exercise: a list with 1 entry for each recorded exercise session
- Fields meaning:
 - `activityName`: the type of entry (“Corsa”: Run, “Bici”: bike, “Camminata”: Walk)
 - `averageHeartRate`: the average heart rate during the session
 - `calories`: the active calories spent during the session
 - `distance`: the distance covered during the session
 - `distanceUnit`: the unit of the “distance” field
 - `duration`: the total duration of the session (in ms)
 - `activeDuration`: the total active duration of the session (in ms)
 - `steps`: the total number of steps done during the session (if “Corsa” or “Camminata”)
 - `logType`: how this record was logged (“tracker” or “auto_detected”)
 - `heartRateZones`: a list that summarizes the number of minutes and calories spent in specific heart rate ranges (from min to max)
 - `speed`: the average speed (in km/h)
 - `vo2max`: the estimated running VO2Max (only for “Corsa” activities)
 - `elevationGain`: the elevation gained during the session
 - `time`: the starting time of the session (format hh:mm:ss)
- **Note**: different activity types can have (slightly) different fields
- **Note**: there can be days without data

```
[ {
  {
    "activityName": "Corsa",
    "averageHeartRate": 143,
    "calories": 727,
    "distance": 8.474117,
    "distanceUnit": "Kilometer",
    "duration": 3.548E+6,
    "activeDuration": 3.214E+6,
    "steps": 8604,
    "logType": "tracker",
    "heartRateZones": [
      {
        "name": "Fuori zona",
        "min": 30,
        "max": 110,
        "minutes": 3,
        "caloriesOut": 30.146219999999971
      },
      {
        "name": "Grassi bruciati",
        "min": 110,
        "max": 137,
        "minutes": 8,
        "caloriesOut": 97.13782
      },
      {
        "name": "Attivita aerobica",
        "min": 137,
        "max": 170,
        "minutes": 48,
        "caloriesOut": 659.73843
      },
      {
        "name": "Picco",
        "min": 170,
        "max": 220,
        "minutes": 0,
        "caloriesOut": 0
      }
    ],
    "speed": 9.4981754822650917,
    "vo2Max": {
      "vo2Max": 48.77853
    },
    "elevationGain": 28.042,
    "time": "11:22:43"
  },
  {
    "activityName": "Bici",
    "averageHeartRate": 150,
    "calories": 3893,
    "distance": 140.143695,
    "distanceUnit": "Kilometer",
    "duration": 2.0901E+7,
    "activeDuration": 1.756E+7,
    "logType": "tracker",
    "heartRateZones": [
      {
        "name": "Fuori zona",
        "min": 30,
        "max": 110,
        "minutes": 28,
        "caloriesOut": 214.835279999999842
      },
      {
        "name": "Grassi bruciati",
        "min": 110,
        "max": 136,
        "minutes": 64,
        "caloriesOut": 664.55027999999993
      },
      {
        "name": "Attivita aerobica",
        "min": 136,
        "max": 170,
        "minutes": 232,
        "caloriesOut": 3167.1500100000012
      },
      {
        "name": "Picco",
        "min": 170,
        "max": 220,
        "minutes": 24,
        "caloriesOut": 377.76059999999995
      }
    ],
    "speed": 28.734978587699313,
    "elevationGain": 663.55,
    "time": "11:03:54"
  }, ...
]
```

Focus on data – Sleep

- Sleep: a list with 1 entry for each sleep session
- Fields meaning:
 - dateOfSleep: day associated to the sleep entry (MM-DD format)
 - startTime: the starting timestamp of the sleep (MM-DD hh:mm:ss format)
 - endTime: the ending timestamp of the sleep (MM-DD hh:mm:ss format)
 - duration: the duration of the sleep session (in ms)
 - minutesToFallAsleep: the number of minutes spent to fall asleep
 - minutesAsleep: the number of minutes asleep during the sleep entry
 - minutesAwake: the number of minutes awake during the sleep entry
 - minutesAfterWakeup: the number of minutes the user spent in bed after waking up
 - efficiency: the estimated sleep efficiency (from 0 to 100)
 - logType: how the entry was logged (always “auto_detected”)
 - mainSleep: a boolean indicating if the entry corresponds to the main sleep session
 - levels: a summary of the sleeping stages (deep/wake/light/rem/restless) during the sleep session
 - data: the stage profile of the sleep session. Each entry contains the starting timestamp, the level, and how much it lasted.
- **Note:** there can be days without data

```
{
  "dateOfSleep": "02-14",
  "startTime": "02-13 22:44:00",
  "endTime": "02-14 06:36:30",
  "duration": 2.832E+7,
  "minutesToFallAsleep": 0,
  "minutesAsleep": 429,
  "minutesAwake": 43,
  "minutesAfterWakeup": 3,
  "timeInBed": 472,
  "efficiency": 96,
  "logType": "auto_detected",
  "mainSleep": true,
  "levels": {
    "summary": {
      "deep": {
        "count": 5,
        "minutes": 106,
        "thirtyDayAvgMinutes": 86
      },
      "wake": {
        "count": 34,
        "minutes": 43,
        "thirtyDayAvgMinutes": 46
      },
      "light": {
        "count": 28,
        "minutes": 235,
        "thirtyDayAvgMinutes": 281
      },
      "rem": {
        "count": 11,
        "minutes": 88,
        "thirtyDayAvgMinutes": 101
      }
    },
    "data": [
      {
        "dateTime": "02-13 22:44:00",
        "level": "wake",
        "seconds": 30
      },
      {
        "dateTime": "02-13 22:44:30",
        "level": "light",
        "seconds": 150
      },
      {
        "dateTime": "02-13 22:47:00",
        "level": "deep",
        "seconds": 1860
      },
      ...
    ]
  },
  ...
}
```