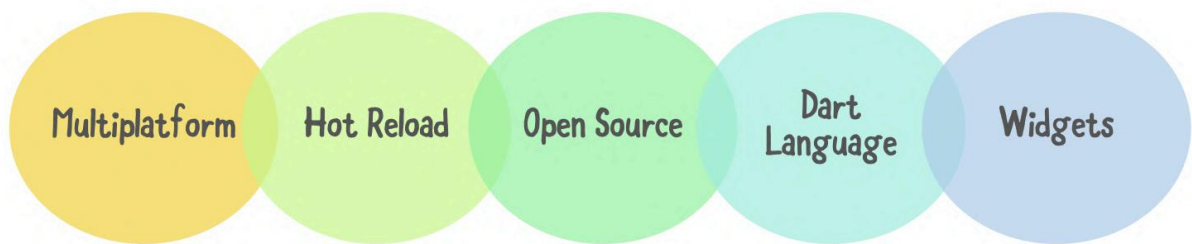


Section : IT Systems Development (DSI)
Module : Framework cross-platform workshop
Teaching unit : Mobile Development and Web
Level : 3rd Year (Applied license : LMD)

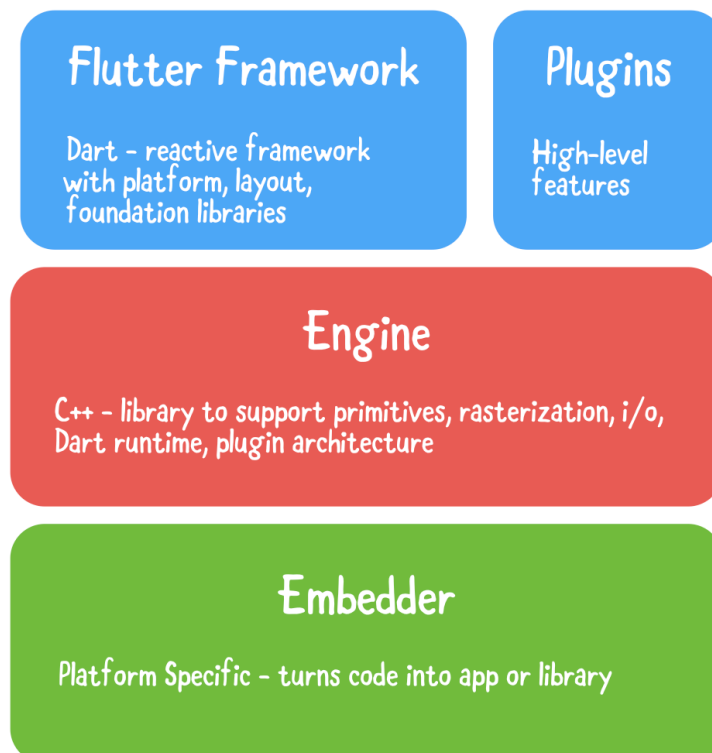
Workshop 1: Starting with Flutter

What is Flutter

In the simplest terms, Flutter is a software development toolkit from Google for building cross-platform apps. Flutter apps consist of a series of packages, plugins and widgets — but that's not all. Flutter is a process, a philosophy and a community as well.



Flutter has a modular, layered architecture. This allows you to write your application logic once and have consistent behavior across platforms, even though the underlying engine code differs depending on the platform



The Flutter architecture consists of three main layers:

1. The **Framework** layer is written in Dart and contains the high-level libraries that you'll use directly to build apps. This includes the UI theme, widgets, layout and animations, gestures and foundational building blocks. Alongside the main Flutter framework are **plugins**: high-level features like JSON serialization, geolocation, camera access, in-app payments and so on. This plugin-based architecture lets you include only the features your app needs.

2. The **Engine** layer contains the core C++ libraries that make up the primitives that support Flutter apps. The engine implements the low-level primitives of the Flutter API, such as I/O, graphics, text layout, accessibility, the plugin architecture and the Dart runtime. The engine is also responsible for rasterizing Flutter scenes for fast rendering onscreen.

3. The **Embedder** is different for each target platform and handles packaging the code as a stand-alone app or embedded module

Technical requirements

In order to start with Flutter, we need a few tools:

- A PC with a recent Windows version, or a Mac with a recent version of the macOS or Linux operating system. we can also use a Chrome OS machine, with a few tweaks.
- An Android/iOS setup.
- The Flutter SDK. It's free, light, and open source.
- Physical device/emulator/simulator
- Android Studio/IntelliJ IDEA or Visual Studio Code

Installation and configuration

- 1- Install Android Studio
- 2- Install Android SDK in (for example) D:\Android\Sdk
- 3- Install Flutter
- 4- Configure Flutter\bin in user environment variable
- 5- Execute Flutter doctor

```
C:\Users\user>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 2.5.1, on Microsoft Windows [version 10.0.19042.1237], locale fr-FR)
[!] Android toolchain - develop for Android devices (Android SDK version 30.0.2)
    ! Some Android licenses not accepted.  To resolve this, run: flutter doctor --android-licenses
[✓] Chrome - develop for the web
[✓] Android Studio (version 2020.3)
[✓] IntelliJ IDEA Ultimate Edition (version 2020.3)
[✓] Connected device (2 available)

! Doctor found issues in 1 category.

C:\Users\user>
```

You may accept all the Android license agreements. You can do this quickly from the terminal line with this command:

flutter doctor --android-licenses

```
C:\Users\user>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, 2.5.1, on Microsoft Windows [version 10.0.19042.1237], locale fr-FR)
[✓] Android toolchain - develop for Android devices (Android SDK version 30.0.2)
[✓] Chrome - develop for the web
[✓] Android Studio (version 2020.3)
[✓] IntelliJ IDEA Ultimate Edition (version 2020.3)
[✓] Connected device (2 available)
```

If the sdk is not configured, execute the next commande

flutter config --D:\Android\Sdk

6- If Android SDK Command-line Tools is not configured

Install then Android SDK Command-line Tools in Android Studio:

Preferences > Appearance & Behavior > System Settings > Android SDK > SDK Tools > Android SDK Command-line Tools (latest)

7- In some cases you need to add the following lines to your path variable

C:\Program Files\Git\bin C:\Program Files\Git\cmd C:\Windows\System32 C:\Windows\System32\WindowsPowerShell\v1.0

Be sure you have installed git framework

8- Create the first application my_first_app

flutter create my_first_app

Execute then

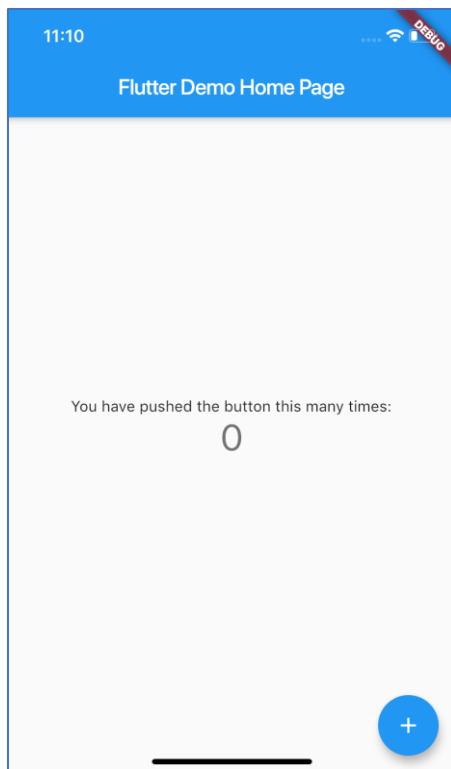
flutter devices

9- It is sometimes recommended to install an emulator in Android studio

flutter run

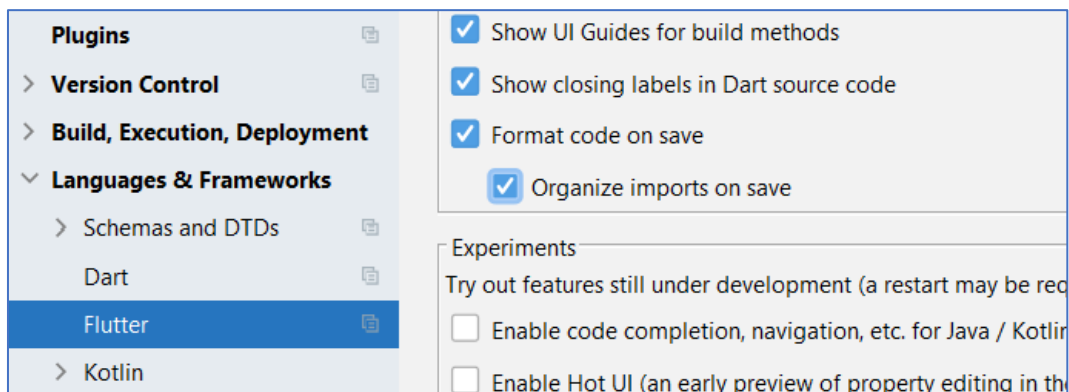
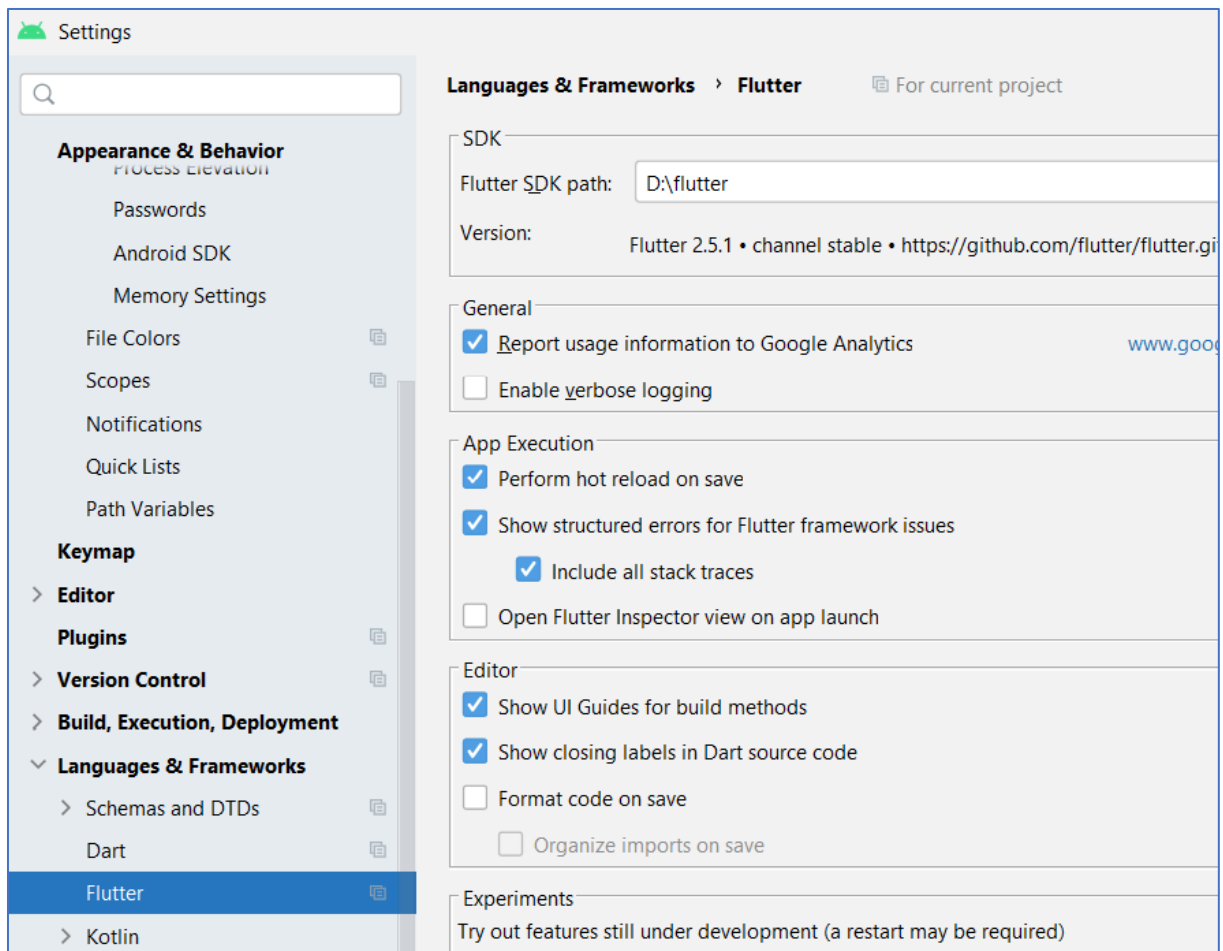
10-To run your app on one of the available devices, type the following command:

flutter run -d [your_device_name]



11- It will be necessary to install the flutter plugins and configure flutter in Android studio

File->Settings->Language & Framework->Flutter



12-To speed up the tests, it is wise to consider the default browser to run these tests
 In an Android Studio terminal select chrome as an emulator when executing the
 Flutter run command

```

Terminal: Local x +
Application finished.

E:\TP_JEE\Etudiant\flutter_apps\Flutter-Projects-master\ch_03>flutter devices
2 connected devices:

Chrome (web) • chrome • web-javascript • Google Chrome 93.0.4577.82
Edge (web) • edge • web-javascript • Microsoft Edge 93.0.961.52

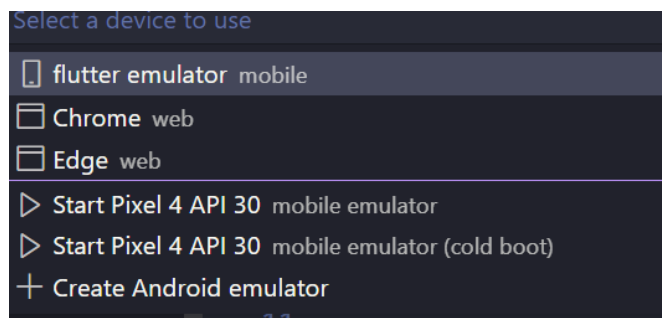
E:\TP_JEE\Etudiant\flutter_apps\Flutter-Projects-master\ch_03>

```

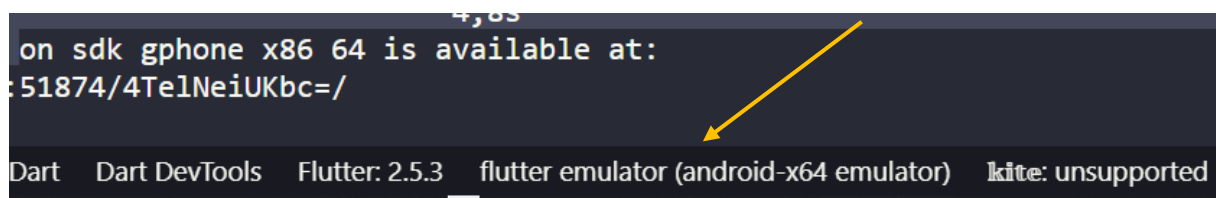
Install Android SDK in Visual Studio code

The simplest way to set up the prerequisites is to download "Android Studio for Windows", run it, and follow the "Setup Wizard" it will present to guide you through installing the Android SDK. After that, the Android SDK will be available in %APPDATA%/Android/SDK , where vscode will pick it up automatically.

In vscode to lunch an emulator Do Ctrl + Shift + P
Then type Flutter:launch emulator



You can see the bottom menu in VSCode, click on this button and you will able to see all the available devices.



To optimize time execution, you can change the path to android\avd in my Android Studio

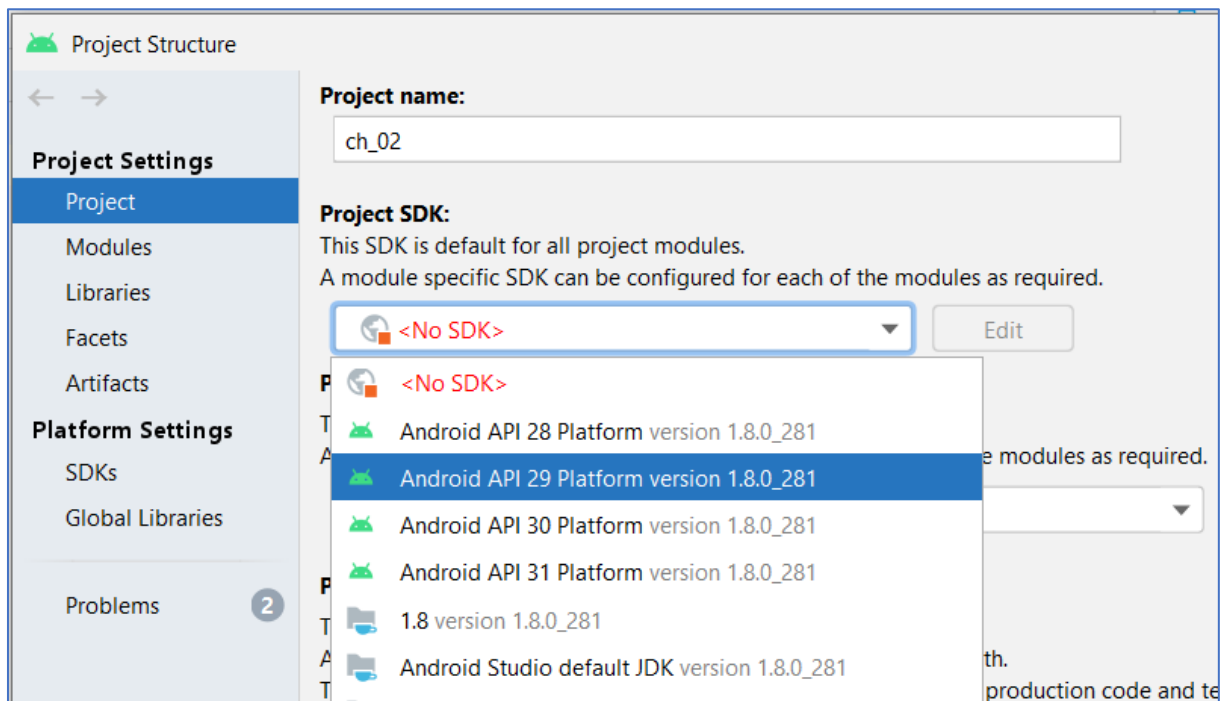
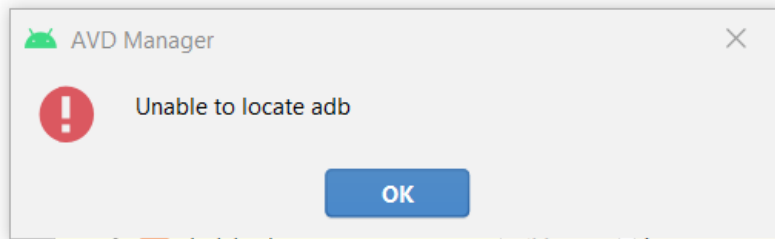
- 1- Open the directory: C:\Users\Username\.android\avd , you will find the directory named with your AVD and a .ini file.
- 2- move the file .avd to your desirable path, then change the path variable in the .ini file to the new location.

3- start the Android studio, you will find that the AVD can work properly.

For example in C:\Users\user\.android\avd we retain the file Pixel_3_API_24.ini witch contain :

```
avd.ini.encoding=UTF-8  
path=D:\Android\avd\Pixel_3_API_24.avd  
path.rel=avd\Pixel_3_API_24.avd  
target=android-24
```

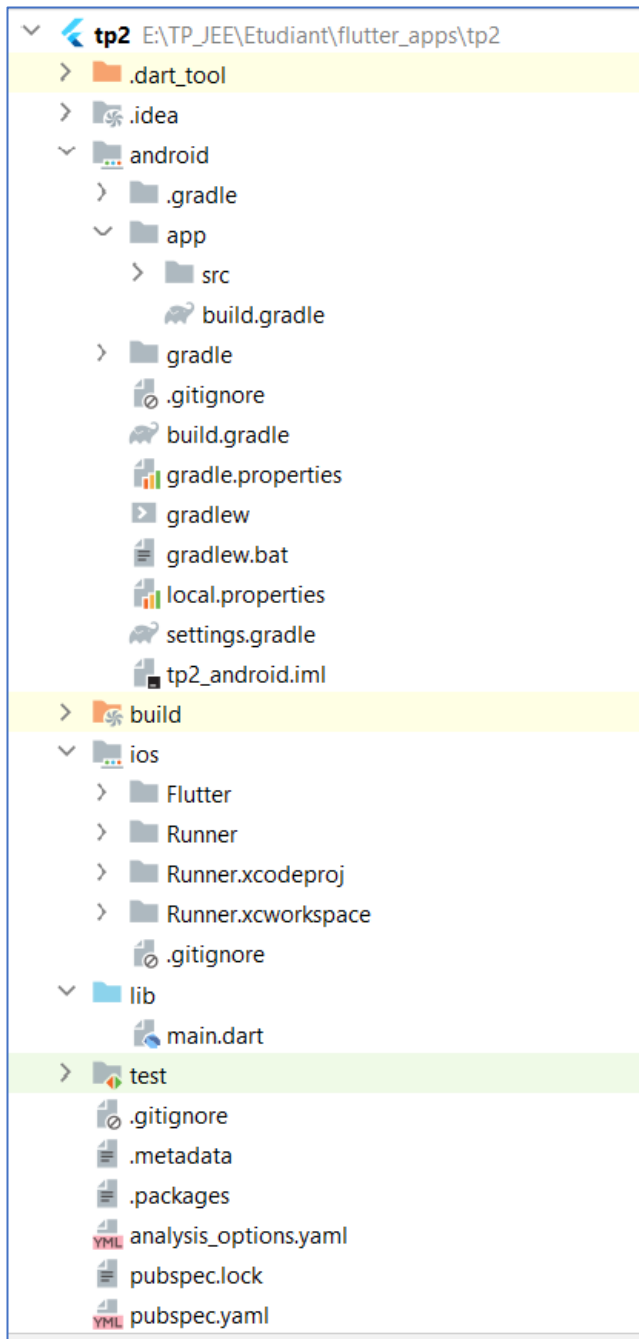
To avoid the next message during launch of an AVD, go to File -> Project Structure -> Project Settings -> Project, and select the Project SDK, which is set to [No SDK] by default.



The files that Flutter generates when you build a project should look like this:

The main folders in your projects are listed here:

```
android  
build  
ios  
lib  
test
```



- The android and ios folders contain the platform shell projects that host our Flutter code. You can open the Runner.xcworkspace file in Xcode or the android folder in Android Studio, and they should run just like normal native apps. Any platform-specific code or configurations should be placed in these folders.

- The build folder calls all the artifacts that are generated when you compile your app.

- The lib folder is the heart and soul (âme) of your Flutter app. This is where you will put all your Dart code. When a project is created for the first time, there is only one file in this directory: main.dart.

- The next file, pubspec.yaml, holds the configuration for your app. This configuration file uses a markup language called YAML Ain't Markup Language (YAML), which you can read more about at <https://yaml.org>. In the pubspec.yaml file, you'll declare your app's name, version number, dependencies, and assets.

pubspec.lock is a file that gets generated based on the results of your pubspec.yaml file. It can be added to your Git repository, but it shouldn't be edited.

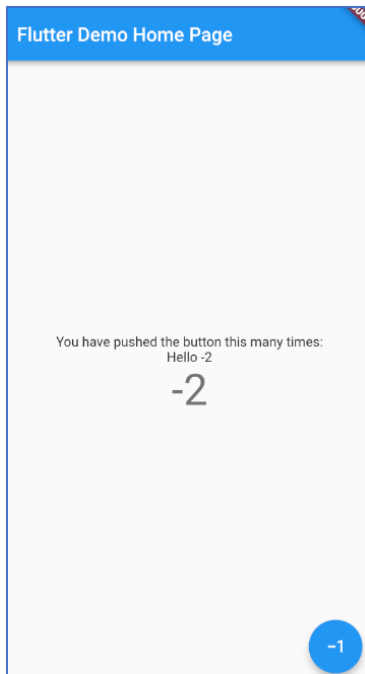
- Finally, the last folder is test. Here, you can put your unit and widget tests,

which are also just Dart code. As your app expands, automated testing will become an increasingly important technique to ensure the stability of your project.

13- Test : Update the primary swatch to green, as shown in the following code snippet, and hit Save:

```
primarySwatch: Colors.green
```

14- Now consider the icon button to decrement the value to display
In addition, display the message "Hello" with this value



You may insert the following code

```
class _MyHomePageState extends State<MyHomePage> with
  WidgetsBindingObserver {
  int _counter = 0;

  String message = "Hello";

  @override
  void didChangeAppLifecycleState(AppLifecycleState state) {
    print('state = $state');
  }

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addObserver(this);
  }

  void _incrementCounter() {
    setState(() {
      _counter--;
      message = "Hello ${_counter}";
    });
  }
}
```

choose the type of icon `exposure_minus_1` for the button at the bottom

```
child: Icon(Icons.exposure_minus_1),
```

2nd application

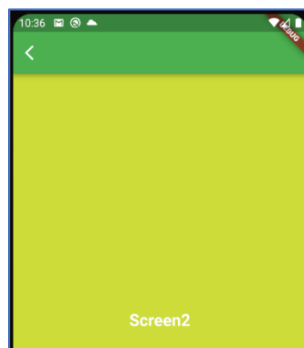
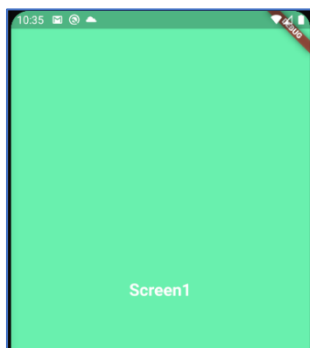
- 1- Create a new flutter application named “navigation”. The aim is to navigate between two screens via a `RoutePages` class

```
import 'package:flutter/material.dart';
import 'package:sec5_navigation/Screen1.dart';
import 'package:sec5_navigation/routes.dart';

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

class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      routes: PageRoutes().routeMaker(),
      initialRoute: PageRoutes.screen1_page,
    );
  }
}
```



2- Create routes.dart to configure routes

```
import 'package:flutter/material.dart';
import 'package:navigation/Screen1.dart';
import 'package:navigation/Screen2.dart';

class PageRoutes {
  static String screen1_page = "screen1";
  static String screen2_page = "screen2";

  Map<String, WidgetBuilder> routeMaker() {
    return {
      screen1_page: (context) => Screen1(),
      screen2_page: (context) => Screen2(),
    };
  }
}
```

3- Screen1 and screen2 classes are represented respectively in screen1.dart and screen2.dart

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

class Screen1 extends StatefulWidget {
  @override
```

```

    _ScreenState createState() => _ScreenState();
}

class _ScreenState extends State<Screen1> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.greenAccent,
      body: GestureDetector(
        behavior: HitTestBehavior.opaque,
        onTap: () {
          // Navigator.of(context)
          //       .pushReplacement(MaterialPageRoute(builder: (context) {
          //         return Screen2();
          //       }));
          Navigator.of(context).pushNamed(PageRoutes.screen2_page);
        },
      ),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.center,
        children: [
          Center(
            child: Text(
              "Screen1",
              style: TextStyle(
                color: Colors.white,
                fontSize: 22.0,
                fontWeight: FontWeight.bold),
            ),
          ),
        ],
      ),
    );
  }
}

```

screen2.dart

```

import 'package:flutter/material.dart';

class Screen2 extends StatefulWidget {
  @override
  _ScreenState createState() => _ScreenState();
}

class _ScreenState extends State<Screen2> {
  Future<bool> onBack() async {
    print("Back button clicked");
    return true;
  }

  @override
  Widget build(BuildContext context) {
    return WillPopScope(
      onWillPop: onBack,
      child: Scaffold(
        backgroundColor: Colors.lime,
        appBar: AppBar(
          backgroundColor: Colors.green,

```

```

        leading: IconButton(
          onPressed: () {
            Navigator.of(context).pop();
          },
          icon: Icon(
            Icons.arrow_back_ios,
          ),
        ),
      ),
    ),
    body: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      crossAxisAlignment: CrossAxisAlignment.center,
      children: [
        Center(
          child: Text(
            "Screen2",
            style: TextStyle(
              color: Colors.white,
              fontSize: 22.0,
              fontWeight: FontWeight.bold),
          ),
        ),
      ],
    ),
  ),
);
}

```

3rd application

- 1- Create new Flutter application named “tp1”
- 2- Open main.dart and delete everything! Then, type the following code into the editor:

```

void main() => runApp(StaticApp());

class StaticApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ImmutableWidget(),
    );
  }
}

```

- 3- Import the material.dart
- 4- type the following code to create a new stateless widget in a new dart file named immutable_widget.dart

```

import 'package:flutter/material.dart';

class ImmutableWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      color: Colors.green,
      child: Padding(
        padding: EdgeInsets.all(40),
        child: Container(

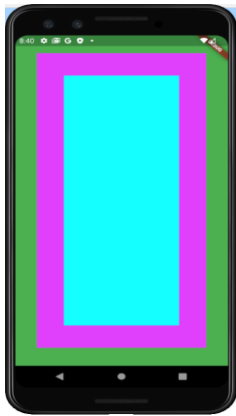
```

```

        color: Colors.purpleAccent,
        child: Padding(
          padding: const EdgeInsets.all(50.0),
          child: Container(
            color: Colors.cyanAccent,
          ),
        ),
      ),
    ),
  );
}

```

5- Run the application in either the iOS simulator or Android emulator



6- Using a Scaffold :

Scaffold provides a basic structure of a screen.

We will be using the Scaffold widget to add an AppBar to the top of the screen and a slide-out drawer that we can pull from the left.

- a. In `basic_screen.dart`, type stless to create a new stateless widget and name that widget BasicScreen. Don't forget to import the material library as well

```

import 'package:flutter/material.dart';

import './immutable_widget.dart';

class BasicScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.indigo,
        title: Text('Welcome to Flutter'),
        actions: <Widget>[
          Padding(
            padding: const EdgeInsets.all(10.0),
            child: Icon(Icons.edit),
          ),
        ],
      ),
      body: Center(
        child: AspectRatio(
          aspectRatio: 1.0,
          child: ImmutableWidget(),
        ),
      ),
    );
  }
}

```

```

    ),
  ),
);
}
}

```

- b. Now, in `main.dart`, replace `ImmutableWidget` with `BasicScreen`. Hit the save button to hot reload and your simulator screen should be completely white:

```

import 'package:flutter/material.dart';

import './basic_screen.dart';

void main() => runApp(StaticApp());

class StaticApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: BasicScreen(),
    );
  }
}

```

- c. Finally, let's add a drawer to the app. Add this code to `Scaffold`, just after `body`:

```

drawer: Drawer(
  child: Container(
    color: Colors.lightBlue,
    child: Center(
      child: Text("I'm a Drawer!"),
    ),
  ),
),

```

The final app should now have a hamburger icon in `AppBar`. If you press it, the drawer will be shown:

