Atelier 2 - Corrigé

Matière : ATELIER FRAMEWORK CROSS-PLATFORM	DSI3
Enseignants : M. Hadiji & S. Hadhri	

I. INTRODUCTION

Flutter est une boîte à outils d'interface utilisateur multiplateforme conçue pour permettre la réutilisation du code sur des systèmes d'exploitation tels qu'iOS et Android, tout en permettant aux applications de s'interfacer directement avec les services de plateforme sous-jacents.

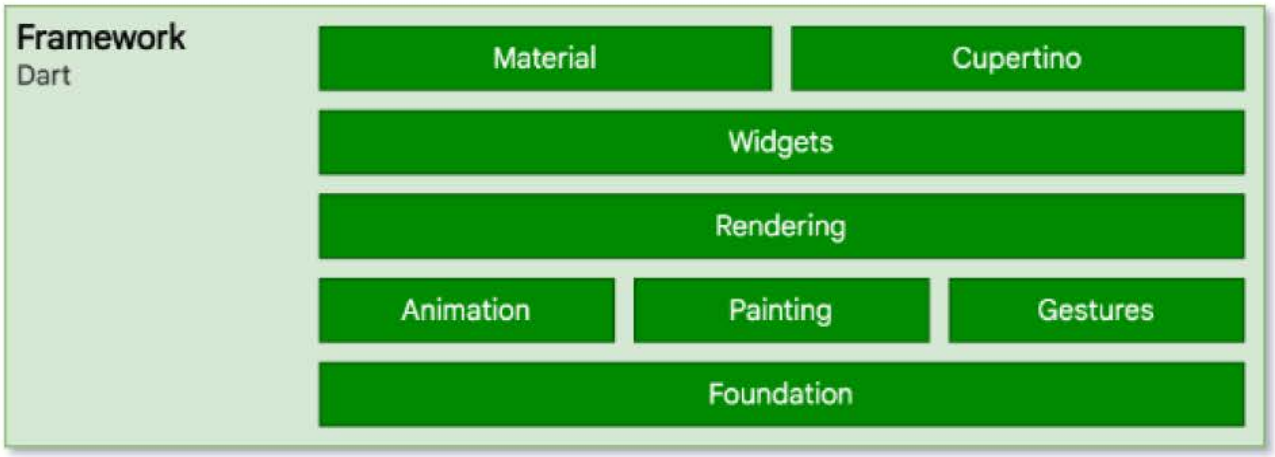
Pendant le développement, les applications Flutter s'exécutent dans une machine virtuelle qui offre un rechargement à chaud (Hot Reload) avec état des modifications sans avoir besoin d'une recompilation complète. Pour la publication, les applications Flutter sont compilées directement en code machine, qu'il s'agisse d'instructions Intel x64 ou ARM, ou en JavaScript si elles ciblent le Web. Le framework open source, dispose d'un écosystème florissant de packages tiers qui complètent les fonctionnalités de base de la bibliothèque.

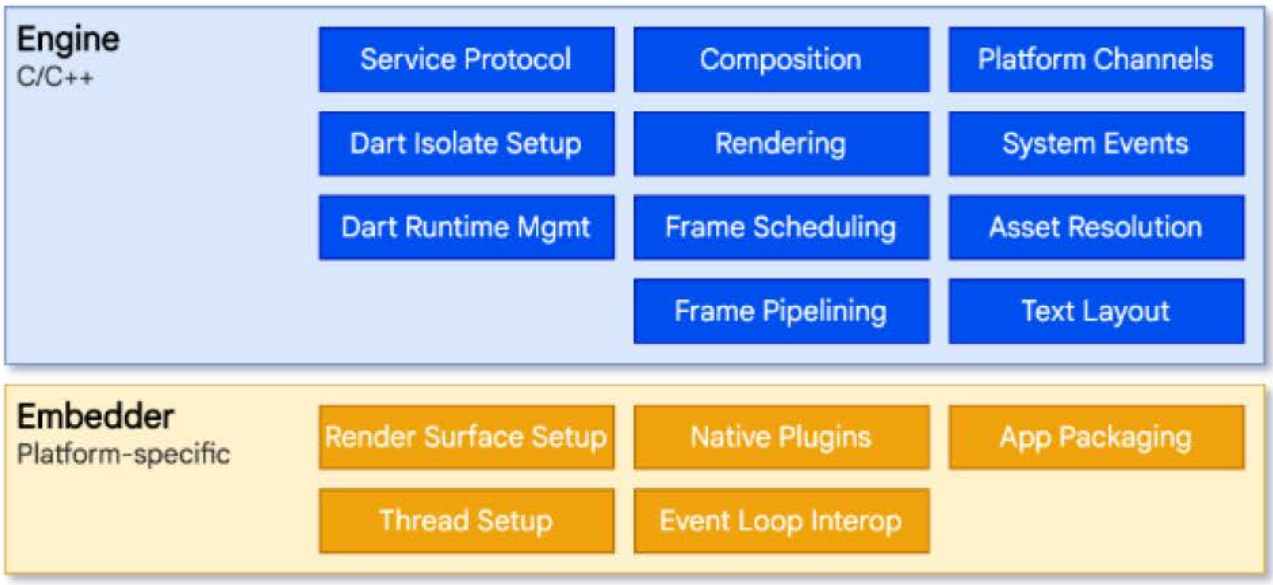
Cet atelier est divisé en plusieurs parties :

- Description de l'**architecture du framework Flutter**
- Explication du concept « **reactive user interfaces** » pour le développement d'interfaces utilisateur Flutter.
- Introduction aux **widgets** qui sont les blocs de construction fondamentaux des interfaces utilisateur Flutter.
- Présentation du **Widget state** : *Stateless*widgets et *StatefulWidgets*.

II. ARCHITECTURE EN COUCHE DU FRAMEWORK FLUTTER

Flutter est conçu comme un système en couches extensible. Il existe sous la forme d'une série de bibliothèques indépendantes qui dépendent chacune de la couche sous-jacente.





L'architecture de Flutter est en constituée de 3 couches :

- La couche supérieure est le « Framework », écrit dans le langage Dart, comprend un riche ensemble de bibliothèques de plate-forme, de mise en page et de base, composé d'une série de couches :
  - Classes fondamentales de base et services de blocs de construction tels que l'animation , la painting et les gestures.
  - La rendering layer fournit une abstraction pour traiter la mise en page. Avec cette couche, vous pouvez construire une arborescence d'objets rendus.
  - La couche widgets est une abstraction de composition. Chaque objet de rendu dans la rendering layer a une classe correspondante dans la couche de widgets.
  - Les bibliothèques Material et Cupertino offrent des ensembles complets de contrôles qui utilisent les primitives de composition de la couche de widget.
- Le Flutter « Engine », qui est au cœur du Framework, est écrit en C++, s'occupe de créer une image à chaque fois qu'un écran doit être « dessiné » lors d'un changement d'état par exemple. Il implémente les librairies principales de Flutter, y compris les animations et graphiques à travers le moteur de rendu Skia (librairie graphique 2D open source), la mise en page du texte, les entrées/sorties de fichiers et de réseau, …
- La couche du bas, appelé « Embedder », est spécifique à chaque plateforme et communique avec l'OS. Ainsi, il nous permet d'accéder aux fonctionnalités du système natif. Ce dernier est écrit en Java et C++ pour Android, Objective-C pour iOS et Mac et C++ pour Windows et Linux.

III. REACTIVE USER INTERFACES

Dans Flutter, les widgets sont représentés par des classes qui sont utilisées pour configurer une arborescence d'objets. Ces widgets sont utilisés pour gérer une arborescence distincte d'objets pour la mise en page, qui est ensuite utilisée pour gérer une arborescence distincte d'objets pour la composition.

Un widget déclare son interface utilisateur en remplaçant la méthode **build()**, qui est une fonction qui convertit le state en UI : UI = f(state)

La méthode **build()** est par conception rapide à exécuter et doit être exempte d'effets secondaires, ce qui lui permet d'être appelée par le framework chaque fois que nécessaire

IV. WIDGETS

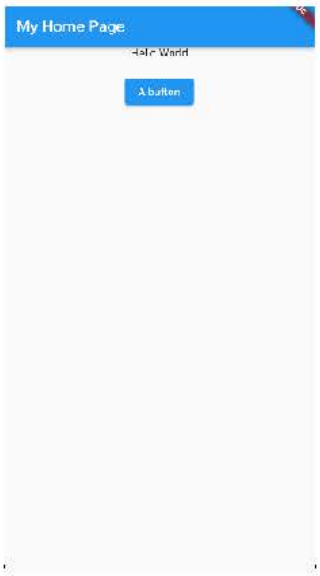
Comme mentionné, Flutter met l'accent sur les widgets en tant qu'unité de composition. Les widgets sont les éléments constitutifs de l'interface utilisateur d'une application Flutter et chaque widget est une déclaration d'une partie de l'interface utilisateur.

Les widgets forment une hiérarchie basée sur la composition. Chaque widget s'emboîte dans son parent et peut recevoir le contexte du parent. Cette structure remonte jusqu'au widget racine (le conteneur qui héberge l'application Flutter, généralement MaterialApp ou CupertinoApp), comme le montre cet exemple :

Fichier main.dart

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

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('My Home Page'),
        ),
        body: Center(
            child: Column(
          children: [
            Text('Hello World'),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () {print('Click!');},
              child: Text('A button'),
            ),
          ],
        )),
      ),
    );
  }
}
```

La fonction main va exécuter la méthode runApp de Dart, qui prend en paramètre un Widget et l'affiche. Ce Widget comprendra tout l'arbre qui décrira l'intégralité de l'UI de l'application.



V. WIDGET STATE

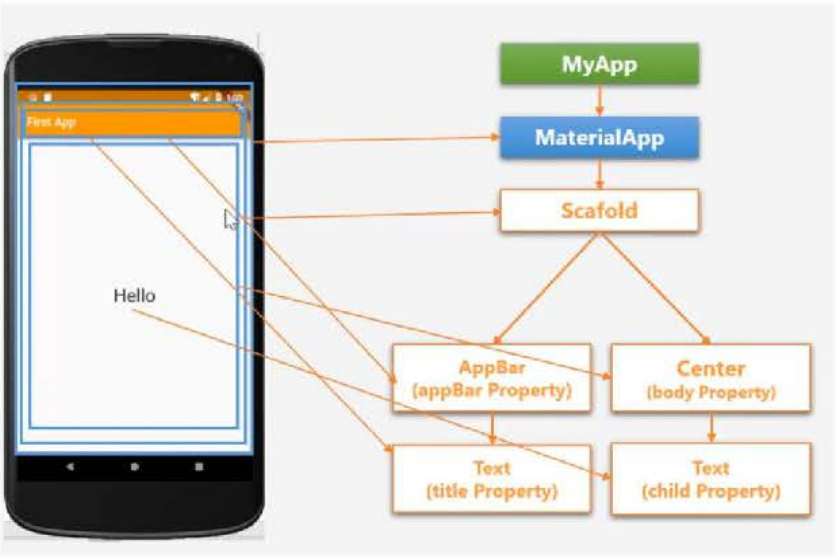
Le framework Flutter introduit 2 grandes classes de widgets : *stateless et stateful* widgets.

1. Stateless widget

Ce sont des widgets qui n'ont aucune propriété qui change avec le temps (une icône ou une étiquette,…). Par exemple afficher le texte « Hello » dans une interface

Exemple d'un stateless widget

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

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('First App'),
          backgroundColor: Colors.orange,
        ),
        body: Center(
            child: Text(
          'Hello',
          style: TextStyle(fontSize: 30),
          textAlign: TextAlign.center,
        )),
      ),
    );
  }
}
```



2. Stateful widget

StatefulWidget est un widget dont ses caractéristiques changent en fonction de l'interaction de l'utilisateur ou d'autres facteurs. Par exemple, si au clic sur un bouton la valeur d'un texte change, celle-ci correspond à l'état de ce widget. Lorsque cette valeur change, le widget doit être reconstruit pour mettre à jour sa partie de l'interface utilisateur.

Les StatefulWidgets stockent l'état mutable dans une classe distincte qui est une sous-classe de **State**. Ils n'ont pas de méthode de construction et leur interface utilisateur est construite à travers leur objet **State**.

Chaque fois que nous modifions un objet **State** (par exemple, en incrémentant le compteur), nous devons appeler la méthode **setState()** pour signaler au framework de mettre à jour l'interface utilisateur en appelant à nouveau la méthode **build()** de l'objet State .

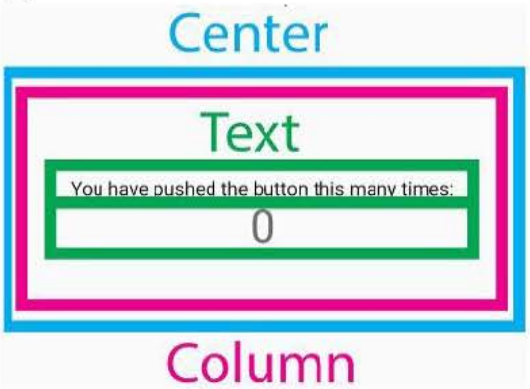
VI. CRÉER UNE APPLICATION FLUTTER COMPTEUR

L'objectif de cette partie est d'illustrer les étapes permettant de développer une application Flutter formée d'un Statefulwidget affichant un nombre et un bouton. Chaque fois que l'utilisateur appuie sur ce bouton, le nombre s'incrémente et s'affiche avec la nouvelle valeur.

1. Créer un nouveau projet Flutter nommé « atelier 2 »

   L'application de base générée Flutter n'est pas l'habituel hello world mais une application contenant un texte "You have pushed the button this many times: N" et un bouton. Le clic du bouton incrémente le nombre affiché dans le texte.

   Le widget du body est ici composé d'un widget Center, lui-même composé d'un widget Column, lui-même composé deux widgets Text.



2. Effacer le contenu du fichier « *main.dart* »

3. Ecrire la méthode « *main()* » et démarrer une application qui contient une classe « *MyApp* »

Fichier main.dart

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

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

NB :

Nous pouvons simplifier l'écriture en utilisant les arrows functions comme dans le cas de javaScript et en supprimant l'instruction new puisqu'elle n'est pas obligatoire. La méthode main() devient alors : void main()=>runApp(MyApp());

4. Créer la class « *MyApp* » et redéfinir dans cette classe la méthode *build()*

Fichier main.dart

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

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

class MyApp extends StatelessWidget{
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    throw UnimplementedError();
  }
}
```

NB :

Grâce au plugin dart, au lieu d'écrire tous ce code, il suffit d'écrire « *st* » puis sélectionner le type de widget

Fichier main.dart

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

5. Créer dans la classe MyApp un objet de type MaterialApp pour afficher une page CounterPage().

Fichier main.dart

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

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

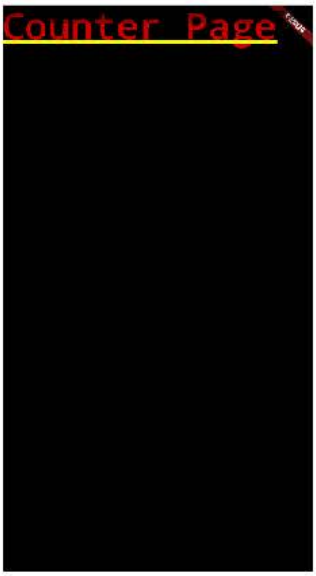
```dart
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: CounterPage(),
    );
  }
}
```

6. Ajouter la classe CounterPage de type Stateless Widget dans la même page main.dart et qui retourne un texte affichant « Counter Page ».

Fichier main.dart

```dart
...
class CounterPage extends StatelessWidget {
  const CounterPage({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Text('Counter Page');
  }
}
```

7. Exécuter le projet



8. Utiliser un objet Scaffold pour afficher :
    a. Une barre qui contient le texte « Counter »
    b. Le texte « Counter value => Contenu du compteur » avec une taille de police égale à 22 et de couleur deepOrange

Fichier main.dart

```dart
…
class CounterPage extends StatelessWidget {
  const CounterPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Counter'),
      ),
      body: Center(
```

```
        child: Text(
          'Counter value => Contenu du compteur ',
          style: TextStyle(color: Colors.deepOrange, fontSize: 22),
        ),
      ),
    );
  }
}
```

9. Modifier le thème de l'application

Fichier main.dart

```
…
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(primarySwatch: Colors.deepOrange),
      home: CounterPage(),
    );
  }
}
```

10. Créer dans le dossier lib un fichier counter.page.dart, déplacer la classe CounterPage dans ce fichier (importer la librairie Material) et appeler ce fichier dans main.dart.

Fichier counter.page.dart

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

class CounterPage extends StatelessWidget {
  const CounterPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Counter'),
      ),
      body: Center(
        child: Text(
          'Counter value => Contenu du compteur ',
```

```
          style: TextStyle(color: Colors.deepOrange, fontSize: 22),
        ),
      ),
    );
  }
}
```

Fichier main.dart

```
import 'package:flutter/material.dart';
import 'package:atelier2/counter.page.dart';

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(primarySwatch: Colors.deepOrange),
      home: CounterPage(),
    );
  }
}
```

11. Déclarer dans la classe CounterPage une variable Counter de type entier et l'initialiser à 0 et afficher le contenu de cette variable dans la page CounterPage

Fichier counter.page.dart

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

class CounterPage extends StatelessWidget {
  int counter = 0;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Counter'),
      ),
      body: Center(
        child: Text(
          'Counter value => $counter',
          style: TextStyle(color: Colors.deepOrange, fontSize: 22),
        ),
      ),
    );
  }
}
```

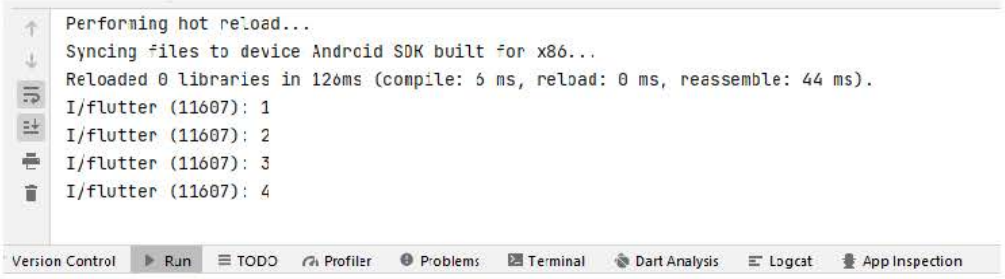
12. Ajouter dans la page Counterpage un bouton flottant pour incrémenter le compteur

Fichier counter.page.dart

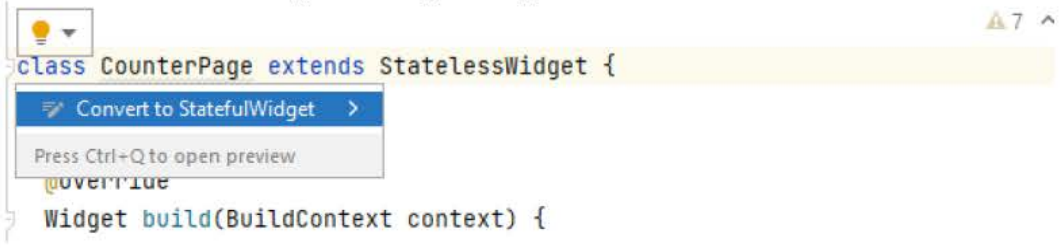
```
import 'package:flutter/material.dart';

class CounterPage extends StatelessWidget {
  int counter = 0;

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

```
    return Scaffold(
      appBar: AppBar(
        title: Text('Counter'),
      ),
      body: Center(
        child: Text(
          'Counter value => $counter',
          style: TextStyle(color: Colors.deepOrange, fontSize: 22),
        ),
      ),
      floatingActionButton: FloatingActionButton(
        child: Icon(Icons.add),
        onPressed: () {
          counter++;
          print(counter);
        },
      ),
    );
  }
}
```

Cliquer sur ce bouton et remarquer que la valeur du compteur s'incrémente dans la console et ne change pas au niveau de la page.



13. Convertir la classe *CounterPage* en *StatefulWidget*



14. Ajouter la méthode *setState()* pour changer le state de la page CounterPage

Fichier counter.page.dart

```
…

class _CounterPageState extends State<CounterPage> {
  int counter = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Counter'),
      ),
      body: Center(
        child: Text(
          'Counter value => $counter',
          style: TextStyle(color: Colors.deepOrange, fontSize: 22),
        ),
      ),
      floatingActionButton: FloatingActionButton(
        child: Icon(Icons.add),
        onPressed: () {
          setState(() {
            counter++;
            print(counter);
          });
        },
      ),
    );
  }
}
```

15. Ajouter un autre bouton flottant dans la page CounterPage permettant de décrémenter le compteur

Fichier counter.page.dart

```
…

class _CounterPageState extends State<CounterPage> {
  int counter = 0;
```

```dart
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Counter'),
      ),
      body: Center(
        child: Text(
          'Counter value => $counter',
          style: TextStyle(color: Colors.deepOrange, fontSize: 22),
        ),
      ),
      floatingActionButton: Row(
        children: [
          SizedBox(width: 20),
          FloatingActionButton(
            child: Icon(Icons.add),
            onPressed: () {
              setState(() {
                counter++;
                print(counter);
              });
            },
          ),
          FloatingActionButton(
            child: Icon(Icons.remove),
            onPressed: () {
              setState(() {
                counter--;
                print(counter);
              });
            },
          ),
        ],
      ),
    );
  }
}
```