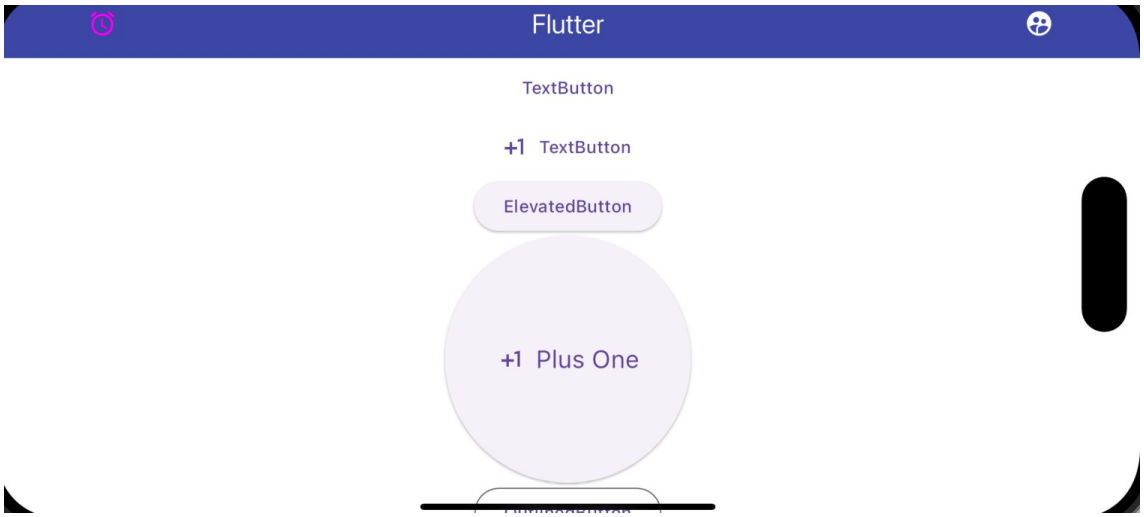
Chapitre 4 – Les boutons et les « states »

Table des matières

L'objectif de ce chapitre est de parcourir l'ensemble des boutons disponibles dans Flutter.

Différents boutons :




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

void main() {
  runApp(const MainApp());
}

class MainApp extends StatelessWidget {
  const MainApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      home: const Home(),
      theme: ThemeData(
        appBarTheme: const AppBarTheme(
          backgroundColor: Colors.indigo,
        ),
```

```
      ),
      debugShowCheckedModeBanner: false,
    );
  }
}

class Home extends StatefulWidget {
  const Home({super.key});
  @override
  State<StatefulWidget> createState() {
    return _Home();
  }
}

class _Home extends State<Home> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Flutter', style: TextStyle(color: Colors.white)),
        leading: const Icon(Icons.access_alarm, color: Color(0xFFFF00FF)),
        actions: <Widget>[
          IconButton(
              onPressed: btDown,
              icon: const Icon(
                Icons.supervised_user_circle,
                color: Colors.white,
              )),
        ],
        elevation: 10.0,
        centerTitle: true,
      ),
      backgroundColor: const Color.fromRGBO(255, 255, 255, 1),
      body: SingleChildScrollView(
        scrollDirection: Axis.vertical,
        child: Center(
          child: Column(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              children: <Widget>[
                TextButton(
                  onPressed: btDown,
                  child: const Text("TextButton"),
                ),
                TextButton.icon(
                  onPressed: btDown,
                  label: const Text("TextButton"),
                  icon: const Icon(Icons.plus_one),
                ),
```

```
            ElevatedButton(
               onPressed: () {}, child: const Text('ElevatedButton')),
            ConstrainedBox(
              constraints:
                 const BoxConstraints.tightFor(width: 200, height: 200),
              child: ElevatedButton.icon(
                onPressed: () {},
                label: const Text('Plus One'),
                icon: const Icon(Icons.plus_one),
                style: ElevatedButton.styleFrom(
                  padding: const EdgeInsets.symmetric(
                     horizontal: 30, vertical: 10),
                  textStyle: const TextStyle(fontSize: 20),
                  shape: const CircleBorder(),
                ),
              ),
            ),
            OutlinedButton(
              onPressed: btDown,
              child: const Text("OutlinedButton"),
            ),
            OutlinedButton.icon(
              icon: const Icon(Icons.plus_one),
              onPressed: btDown,
              label: const Text("OutlinedButton"),
            ),
          ]),
      ),
    ),
  );
 }

 void btDown() {
   setState(() {
     print('Bouton appuyé');
   });
 }
}
```

Commentaires :

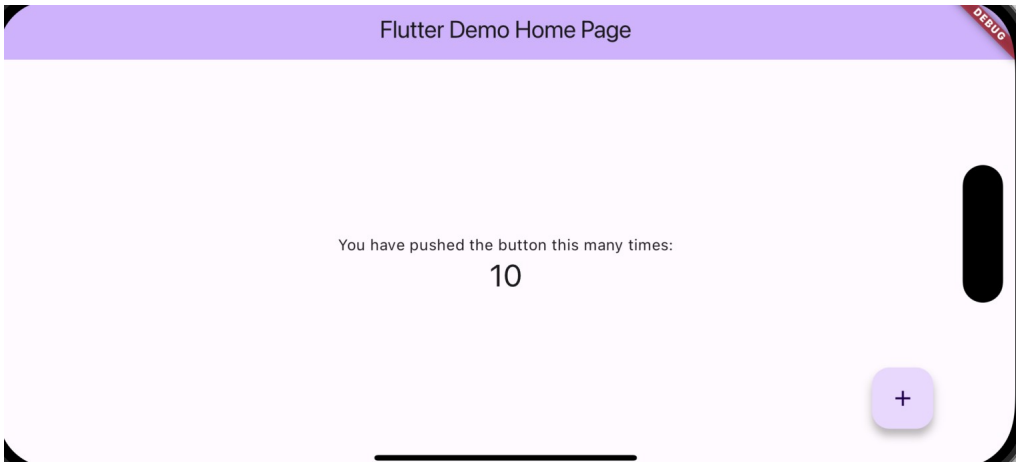
Ce code montre différents types de boutons qui appellent tous la même fonction « btDown ».
IconButton : pour être utilisé dans les barres d'outils et autres interfaces utilisateur où l'espace est limité.
TextButton et TextButton.icon : Pour les actions textuelles simples et les actions avec des icônes.
ElevatedButton : utilisé pour des actions primaires.
OutlinedButton : Pour des actions secondaires, avec un style visuel moins imposant.

ConstrainedBox : pour appliquer des contraintes spécifiques à un élément, dans ce cas pour fixer les dimensions d'un bouton.

Dans la console de debogage lorsque vous cliquez sur un des boutons :

```
Launching lib/main.dart on iPhone 15 Pro Max in debug mode...
Xcode build done.                                           13,2s
[ERROR:flutter/shell/platform/darwin/graphics/FlutterDarwinContextMetalImpe
er.mm(42)] Using the Impeller rendering backend.
Connecting to VM Service at ws://127.0.0.1:49611/Ww6uvbpTamc=/ws
9 flutter: Bouton appuyé
```

Gestion des états avec l'exemple généré par défaut




```
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(
      title: 'Flutter Demo',
      theme: ThemeData(

        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
```

```
      useMaterial3: true,
    ),
    home: const MyHomePage(title: 'Flutter Demo Home Page'),
   );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

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

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {

    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headlineMedium,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
```

```
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

Commentaires :

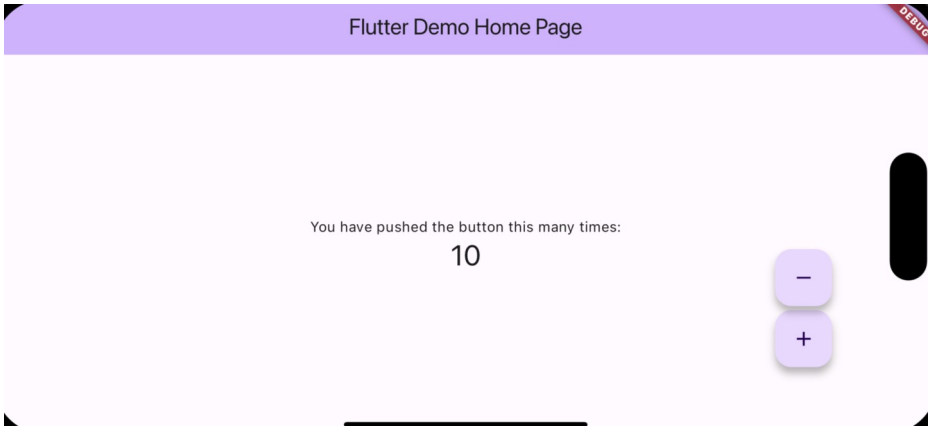
La variable « _counter » est définie comme un entier (int) initialisé à 0. Elle sert à stocker la valeur actuelle du compteur.
La fonction « _incrementCounter » : est une méthode de la classe de widget d'état. Son rôle est d'incrémenter la valeur de _counter.
La fonction _incrementCounter utilise setState() pour mettre à jour l'état du widget. Lorsque setState() est appelé, il indique à Flutter que l'état de ce widget a changé et que l'interface utilisateur doit être reconstruite. À l'intérieur de setState(), _counter est incrémenté de 1 (_counter++).

Text('$_counter') : affiche le texte qui est la valeur de la variable _counter.

Theme.of(context).colorScheme.inversePrimary pour backgroundColor, le widget utilise une couleur de fond qui est déterminée par le thème global de l'application, utilisant une couleur qui contraste fortement avec la couleur primary.

headlineMedium : styles prédéfinis dans textTheme. Il est généralement utilisé pour les titres ou les sous-titres de taille moyenne. Les styles comme headlineMedium sont configurés pour maintenir une cohérence visuelle à travers l'application.

Placer plusieurs FloatingButton :




```
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(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

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

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Theme.of(context).colorScheme.inversePrimary,
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
```

```
          const Text(
            'You have pushed the button this many times:',
          ),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.headlineMedium,
          ),
        ],
      ),
    ),
    floatingActionButton: Stack(
      fit: StackFit.expand,
      children: [
        Positioned(
          right: 30,
          bottom: 30,
          child: FloatingActionButton(
            onPressed: () {},
            child: Icon(Icons.add),
            heroTag: null,
          ),
        ),
        Positioned(
          right: 30,
          bottom: 90,
          child: FloatingActionButton(
            onPressed: () {},
            child: Icon(Icons.remove),
            heroTag: null,
          ),
        ),
      ],
    ),
  );
 }
}
```

Commentaire :

Stack: permet de superposer des widgets les uns sur les autres. Les enfants de Stack sont positionnés en fonction de leur coin supérieur gauche, sauf si des widgets « Positioned » sont utilisés pour spécifier leur emplacement exact.

StackFit.expand: ajuste la taille du Stack pour qu'il remplisse tout l'espace disponible. Le Stack va s'étendre pour occuper tout l'espace que son parent lui alloue. Cela signifie que les enfants de ce Stack qui ne sont pas positionnés (par exemple, avec un widget Positioned) seront étendus pour remplir tout l'espace disponible, à moins qu'ils n'aient leur propre contrainte de taille.

Les deux boutons utilisent **heroTag: null**, ce qui est nécessaire pour avoir plusieurs boutons flottants d'action sur la même route dans Flutter sans conflits d'animation Hero.

Exercice :

Écrire le code du bouton permettant de décrémenter le compteur.