Chapitre 1 – Squelette d’une première application Flutter

Table des matières

Squelette de base (écran sans état : StatelessWidget)

Code d’une application générée avec un projet vide « Empty » :


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

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

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: Scaffold(
        body: Center(
          child: Text('Hello World!'),
        ),
      ),
    );
  }
}
```

Commentaires :

import 'package:flutter/material.dart'; // importe le package material.dart qui contient les bibliothèques et les widgets de « Material Design » utilisés pour construire des interfaces utilisateur dans Flutter.

La fonction main est le point d'entrée de toute application « Dart ». Ici, elle appelle la fonction « runApp », qui prend en argument une instance de « MainApp ».

MainApp est une classe qui hérite de « StatelessWidget », ce qui signifie qu'il s'agit d'un widget qui ne gère pas d'état (ses propriétés ne peuvent pas changer au cours du temps).
Le constructeur de MainApp accepte un argument key qui est passé à la classe parente grâce à super.key. Les clés aident Flutter à identifier de manière unique les widgets à travers les reconstructions.

La méthode build retourne un « MaterialApp ».
« **const** » devant MaterialApp, signifie que l'instance de « MaterialApp » ne change pas et peut être compilée comme une constante.

« **home** » est la propriété qui définit l'écran de départ de l'application. « Scaffold » est utilisé, qui fournit une structure visuelle de base pour les écrans « Material Design ».

À l'intérieur de « Scaffold », la propriété body définit le contenu principal de l'écran. « Center » est un « widget » qui centre son enfant à l'intérieur de lui-même. Vous verrez que des « widget » peuvent avoir un seul enfant « child » ou plusieurs enfants « children » (tableau de widgets).

Text('Hello World!') est l'enfant de « Center » et affiche le texte "Hello World!".

Écran avec gestion des états : StatefulWidget

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

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

class MainApp extends StatelessWidget {
  const MainApp({super.key});
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Flutter Demo',
      home: Home(),
    );
  }
}

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 const Scaffold();
  }
}
```

Commentaires :

Home est un « StatefulWidget », ce qui signifie qu'il peut maintenir et modifier son état en réponse aux interactions de l'utilisateur par exemple.

La méthode createState retourne une instance de _Home, la classe qui gèrera l'état de ce widget.

_Home est la classe d'état pour Home. State<Home> indique que cette classe d'état est spécifique au widget Home. La méthode build de cette classe construit l'interface utilisateur du widget Home.

L'utilisation des const dans Dart

L'utilisation de const dans « Dart » sert à indiquer que l'objet sera une constante à la compilation. Il **ne changera jamais** après sa création initiale.

L'utilisation de const permet plusieurs choses :

1) **Optimisation des performances :** Les objets constants sont créés une seule fois et stockés en mémoire. Flutter peut réutiliser ces objets constants à plusieurs endroits de l'application, ce qui réduit l'utilisation de la mémoire et peut améliorer la vitesse de l'application.

2) **Prévention des modifications :** Vous garantissez qu'il ne peut pas être modifié après sa création. Cela peut aider à éviter les bugs liés à des modifications inattendues.

3) **Comparaison immédiate :** Les constantes peuvent être comparées de manière plus efficace, car « Dart » sait qu'elles ne changeront pas et peut donc optimiser les comparaisons à la compilation.