Atelier 2 Flutter
Les Widgets

I. Introduction

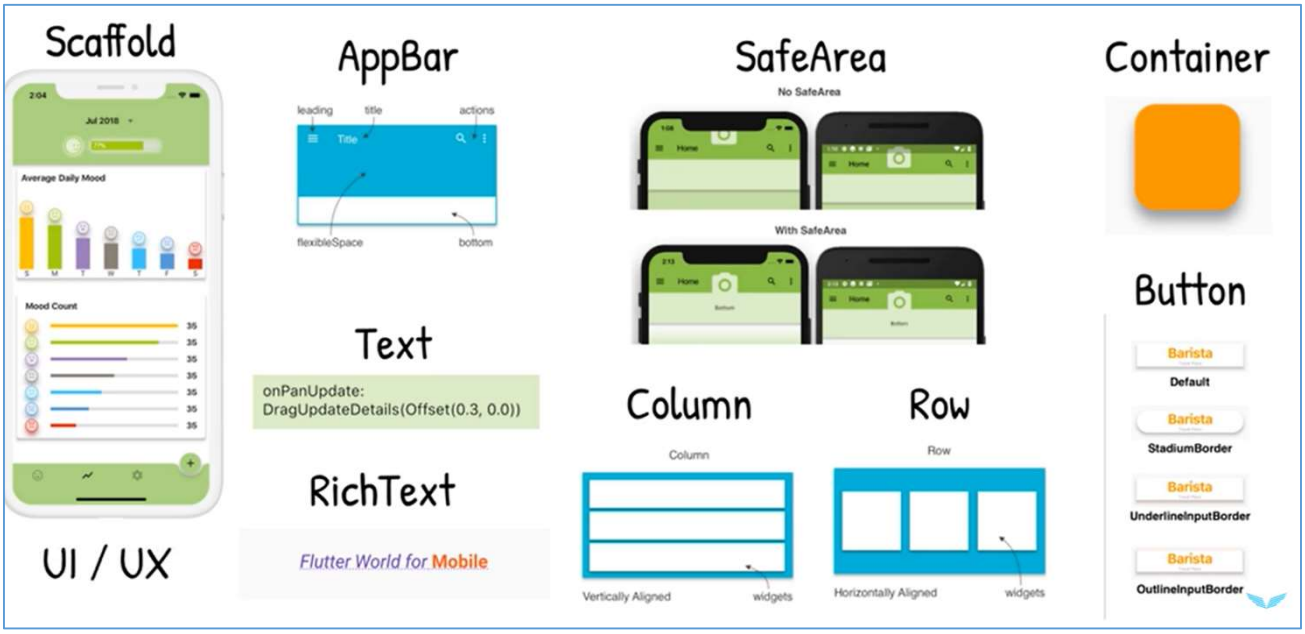
Un **widget** est un composant d'interface graphique.

Les widgets Flutter sont les principaux composants d'une application. Ils donnent l'apparence en fonction de leur configuration et de leur état.

Dans Flutter, tout ce qui apparaît dans l'interface utilisateur s'appelle widget et ils héritent tous de la classe Widget. Lorsque vous créez une interface utilisateur dans Flutter, vous le faites en attachant chaque widget à l'écran de l'application afin qu'il s'adapte exactement à l'endroit où vous le souhaitez.

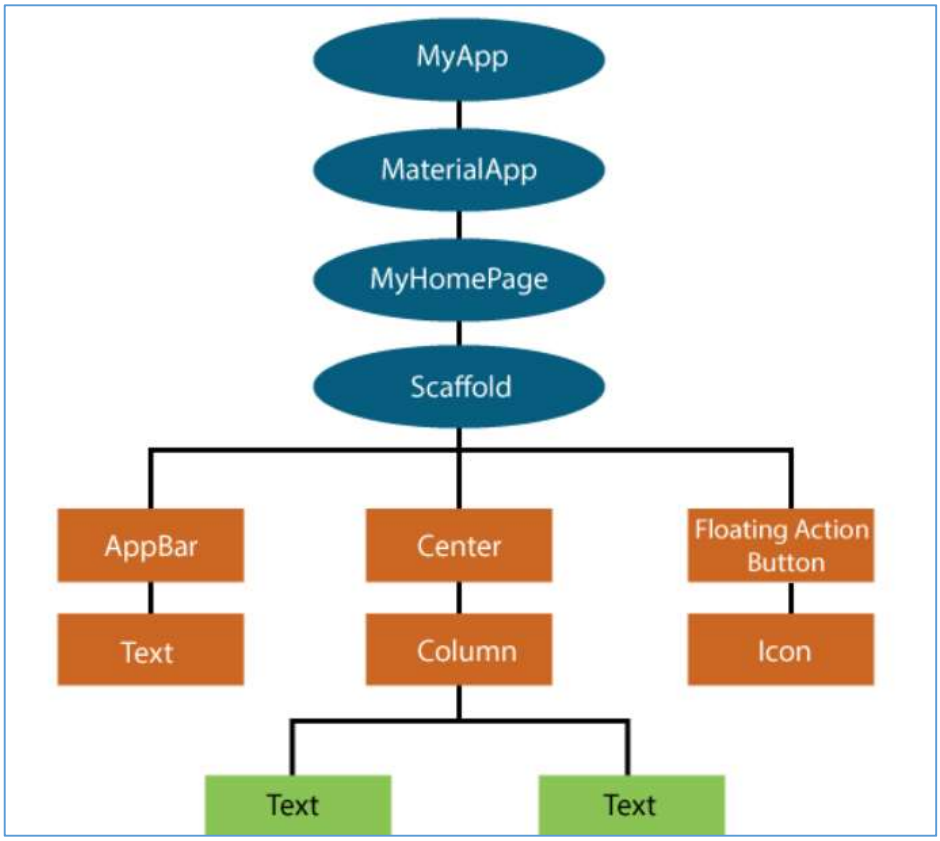
Pensez à un widget comme un composant visuel, ou un composant qui interagit avec l'aspect visuel d'une application. Lorsque vous avez besoin de construire quelque chose de visuel (un bouton par exemple) ou quelque chose qui ordonne un ensemble d'éléments visuels (ex : les colonnes, les lignes) ou encore quelque chose qui interagit avec un élément visuel (ex : un bouton cliquable qui produit un résultat souhaité), alors vous utilisez un widget.

Exemples de Widgets :



En les imbriquant ensemble, vous créerez une hiérarchie appelée **Arborescence des Widgets** (Widget tree).

Les widgets sont organisés sous forme d'arborescence (arbre). Un widget qui contient un ou plusieurs autres widgets est appelé parent widget, les widgets contenus dans un widget parent sont appelés children.



Un **Context** n'est rien d'autre qu'**une référence à l'emplacement d'un widget** dans l'arbre de tous les widgets déjà construits par le Framework. Il n'est lié qu'à un seul widget.

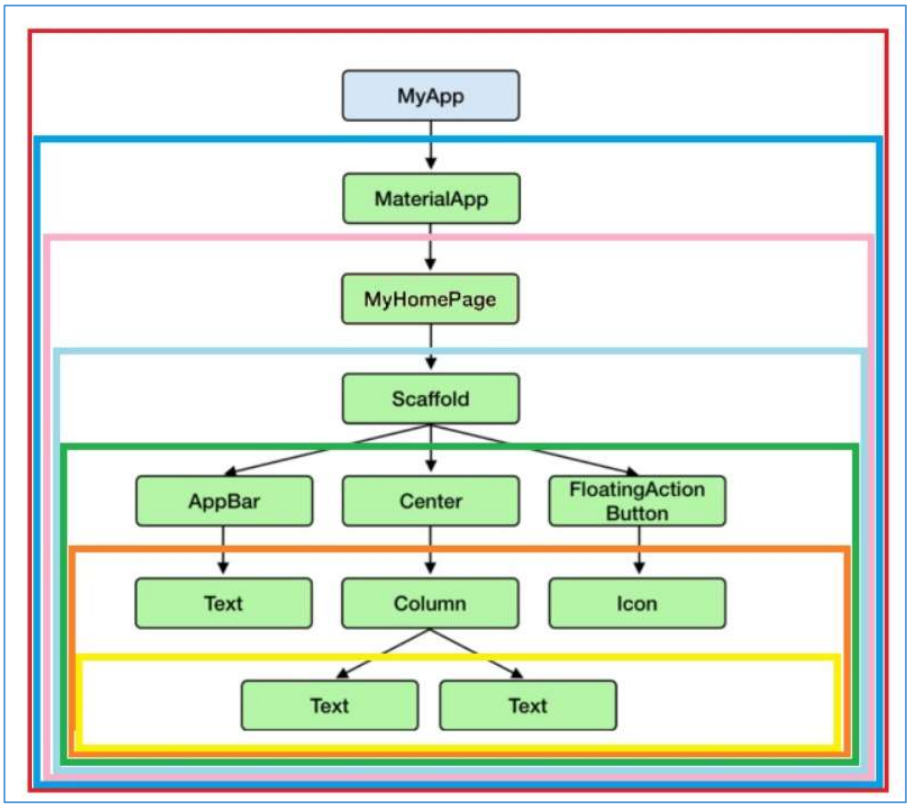
Le context du parent devient le parent context de tous les widgets enfants directs. Par conséquent les Context sont liés, et forment aussi un arbre de Context (relation parent-enfant).

Lorsque vous créez un nouveau widget et vous exécutez le code, le Framework se charge de l'insérer dans l'arborescence des widgets existants. Il cherchera le parent direct de votre nouveau widget afin d'utiliser son context qui servira à insérer le widget dans l'arbre.

Une analogie pour encore mieux comprendre : considérons une famille quelconque, si nous choisissons un individu de cette famille et que nous souhaitons connaitre son identité, la question naturelle qu'on pourrait lui poser est : « qui est ton parent ? » et non « qui est ton enfant ? », car sa position dans cette famille s'obtient par rapport à son parent direct.

De cette analogie nous pouvons donc dire que quelque chose n'est visible que dans son propre contexte ou dans celui de son parent. De ce fait, il est aisé de trouver un widget ancêtre à partir d'un context enfant (un peu comme une généalogie).

Si nous essayons maintenant d'illustrer la notion de Context dans le diagramme précédent, nous obtenons (toujours comme une vue très simplifiée) où chaque couleur représente un context (à l'exception de MyApp, qui est différent) :



II. MaterialApp

Le "Material Design" est un "langage de conception " développé par Google. Cette nouvelle méthodologie de conception a été créée en 2014 et est aujourd'hui l'une des plus grandes tendances du design.

Pensé pour être fluide, naturel, intuitif et simple à comprendre, Material Design a plusieurs particularités et fondements, et vise à synthétiser les concepts classiques d'un bon design avec l'innovation et les possibilités apportées par la technologie et la science.

Il offre une expérience transparente sur de nombreuses plates-formes différentes, qu'il s'agisse de smartphones, d'ordinateurs ou de montres intelligentes.

MaterialApp est l'un des widgets les plus puissants de Futter. Si vous créez une application Flutter de base, le premier widget que vous verrez est MaterialApp.

Voir : https://flutter.github.io/samples/web/material_3_demo/

Cependant, il est possible d'utiliser **Cupertino** à la place de MaterialApp. Les widgets Cupertino implémentent le langage de conception iOS actuel basé sur les directives d'interface humaine d'Apple.

Le langage de conception Material a été créé pour n'importe quelle plate-forme, pas seulement pour Android. Lorsque vous écrivez une application Material dans Flutter, elle a l'apparence Material sur tous les appareils, même iOS. Si vous voulez que votre application ressemble à une application standard de style iOS, vous utiliserez la bibliothèque Cupertino.

Vous pouvez techniquement exécuter une application Cupertino sur Android ou iOS, mais (en raison de problèmes de licence) Cupertino n'aura pas les bonnes polices sur Android. Pour cette raison, utilisez un appareil spécifique à iOS lors de l'écriture d'une application Cupertino.

Le Widget MaterialApp est une classe prédéfinie dans un flutter. C'est probablement le composant principal ou central du flutter qui permet l'accès à tous les autres composants et widgets fournis par Flutter SDK.

Comment on va le mettre en place ?

Dans le fichier **/lib/main.dart**

1. Vous devez d'abord importer le package Flutter.

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

2. main : Fonction principale qui lance l'application Flutter

```
void main() {
  // Lance l'application en exécutant MyApp
  runApp(const MyApp());
}
```

3. En enveloppant votre application dans MaterialApp, vous indiquez à votre application d'utiliser Material Design d'Android.

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

  // Ce widget est la racine de l'application
  @override
  Widget build(BuildContext context) {
    // Retourne un MaterialApp configuré
    return MaterialApp(
      // Titre de l'application
      title: 'Flutter Application',
      // Thème de l'application avec une palette de couleurs personnalisée
      theme: ThemeData(
        // Utilisation d'un ColorScheme basé sur une couleur de départ
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        // Activation de Material Design 3
        useMaterial3: true,
      ),

      home: const Material(color: Colors.amber
    ),
    );
  }
}
```

En fait, la tâche principale d'un widget consiste à fournir une méthode build() qui décrit comment afficher le widget en fonction d'autres widgets de niveau inférieur. Donc, la fonction build() renvoie ce qui doit être affiché à l'écran.

BuildContext est utilisé pour localiser un widget particulier dans une arborescence et chaque widget possède son propre BuidContext. En fait, chaque widget dans Flutter est créé par la méthode build(), qui prend BuildContext comme argument.

Ce package nous fournit tous les widgets que nous pouvons utiliser dans notre application. Par exemple : AppBar, Scaffold, BottomNavigationBar, Card, Chip, BottomSheet, etc.

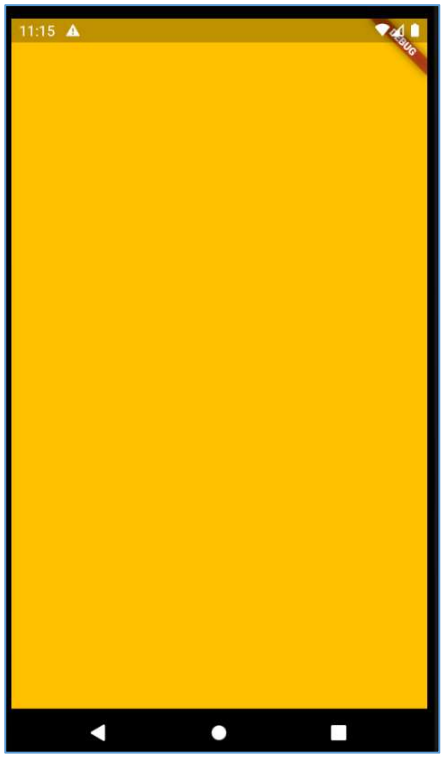
MaterialApp doit avoir au moins l'une des propriétés home, routes, onGenerateRoute ou builder non nulles. Sans cela, vous obtiendrez une erreur.

Le code complet :

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

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

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

  // Ce widget est la racine de l'application
  @override
  Widget build(BuildContext context) {
    // Retourne un MaterialApp configuré
    return MaterialApp(
      // Titre de l'application
      title: 'Flutter Application',
      // Thème de l'application avec une palette de couleurs personnalisée
      theme: ThemeData(
        // Utilisation d'un ColorScheme basé sur une couleur de départ
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        // Activation de Material Design 3
        useMaterial3: true,
      ),

      home: const Material(color: Colors.amber
    ),
    );
  }
}
```

Résultat



Quelques attributs ou propriétés de MaterialApp

title : le nom de l'application qui s'affiche dans la fenêtre d'administration de l'application.

theme : le thème, qui définit la couleur de l'interface utilisateur de l'application.

color : la couleur du thème de l'application, qui est également la couleur de la fenêtre d'administration de l'application.

home : widget de l'interface principale de l'application.

routes : la table de navigation supérieur de l'application, qui permet de naviguer entre les pages.

Comment supprimer la bannière DEBUG ?

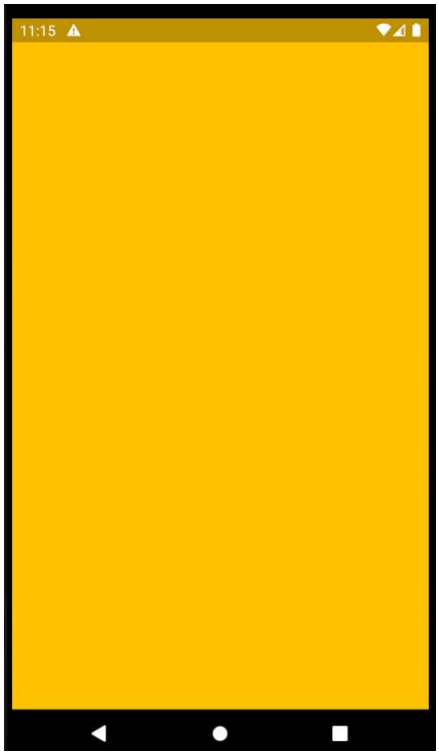
Si vous exécutez votre application mobile Flutter, une étiquette DEBUG s'affichera dans le coin supérieur droit.

En fait, cette bande sera automatiquement supprimée lorsque vous créerez votre application mobile Flutter dans un mode de publication. Cependant, nous pouvons également la supprimer d'une application de débogage.

Tout ce que vous avez à faire et de définir la propriété debugShowCheckedModeBanner sur false dans le widget MaterialApp de votre application.

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

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

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

  // Ce widget est la racine de l'application
  @override
  Widget build(BuildContext context) {
    // Retourne un MaterialApp configuré
    return MaterialApp(
      // Titre de l'application
      title: 'Flutter Application',
      // Thème de l'application avec une palette de couleurs personnalisée
      theme: ThemeData(
        // Utilisation d'un ColorScheme basé sur une couleur de départ
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        // Activation de Material Design 3
        useMaterial3: true,
      ),
```

```
      home: const Material(color: Colors.amber
    ),
     // Désactivation du bandeau "DEBUG"
      debugShowCheckedModeBanner: false,
    );
  }
}
```



III. Le StatelessWidget c'est quoi ?

Le StatelessWidget va servir à composer d'autres widgets en blocs facilement réutilisables. C'est le widget le plus basique qui est généralement utilisé en Flutter.

Le StatelessWidget est immuable. Alors, il est qualifié de widget sans état.

Un widget sans état ne peut pas changer pendant l'exécution d'une application Flutter. Cela signifie qu'il est impossible de le modifier pendant que l'application est en action. Pour cette raison, l'apparence et les propriétés restent inchangées pendant toute la durée de vie du widget.

De plus, il ne stocke pas les valeurs susceptibles de changer. Les widgets sans état peuvent être utiles lorsque la partie de l'interface utilisateur que nous décrivons ne dépend d'aucun autre widget. Par exemple, le texte, les icônes et les boutons d'icônes.

Étapes pour implémenter le widget sans état :

1- Créez une classe qui étend '`StatelessWidget`'.

2- Créez une méthode '`build()`' pour les widgets qui ne changent jamais leurs valeurs pendant l'exécution.

3- La méthode '`build()`' renvoie le widget.

Le code dans le constructeur of the MyApp contient {Key? key }

Il déclare un paramètre nommés facultatif (nommé facultatifs à cause de {}) où le nom est key avec le type Key.

Lorsqu'une classe est initialisée, l'accès en lecture à celle-ci est interdit jusqu'à ce que l'appel au super constructeur soit terminé.

La liste d'initialisation est souvent utilisée pour valider les valeurs de paramètre passées avec assert (key != null) ou pour initialiser les champs finaux avec des valeurs calculées (les champs finaux ne peuvent pas être initialisés ou mis à jour ultérieurement).

super(key: key) transmet au constructeur de la super classe et transmet le paramètre key passé à MyApp au paramètre key des super constructeurs.

Flutter utilise généralement des clés (key) lorsqu'il doit identifier de manière unique des widgets spécifiques au sein d'une collection.

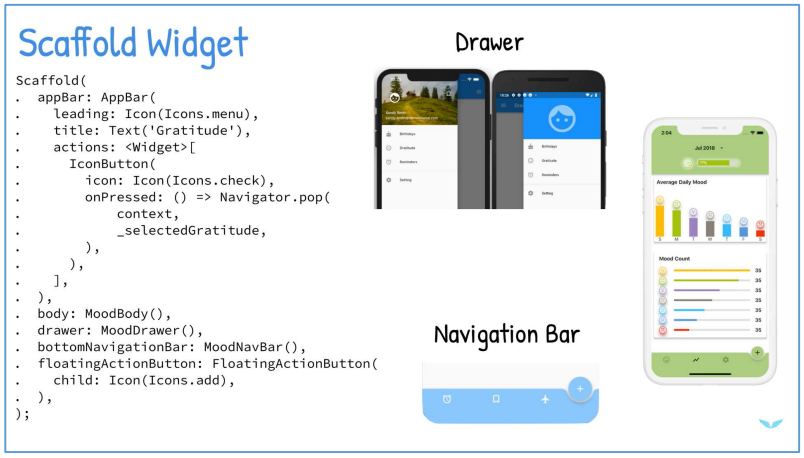
IV. Scaffold

Le widget Scaffold (qui veut dire échafaudage), issu de la bibliothèque Material, fournit une barre d'application par défaut, un titre et un corps qui contient l'arborescence de widgets de l'écran d'accueil. La sous-arborescence du widget peut être assez complexe. De plus, Scaffold est généralement utilisé comme sous-widget de MaterialApp, il remplit l'espace disponible et occupe la totalité de la fenêtre ou de l'écran de l'appareil.

Quelle est la différence entre Scaffold et MaterialApp dans Flutter ?

MaterialApp Widget est le point de départ de votre application, il indique à Flutter que vous allez utiliser les composants Material et suivre la conception de Material dans votre application.

Widget Scaffold est utilisé sous MaterialApp, il vous offre de nombreuses fonctionnalités de base, comme AppBar, BottomNavigationBar, Drawer, FloatingActionButton, etc.



Scaffold étendra ou occupera tout l'écran de l'appareil. Il occupera l'espace disponible. Scaffold fournira un cadre pour mettre en œuvre la disposition de base de la conception matérielle de l'application.

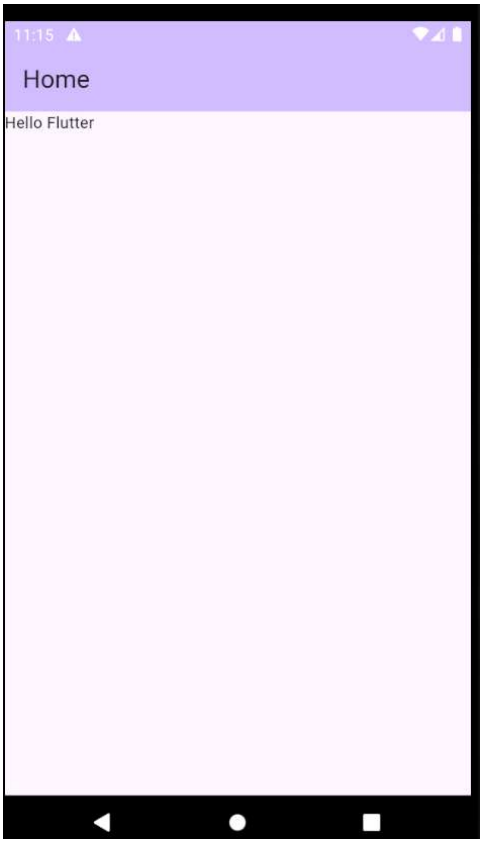
Exemple Scaffold

```dart
// Importation du package Flutter Material
import 'package:flutter/material.dart';

// Fonction principale qui lance l'application Flutter
void main() {
  // Lance l'application en exécutant MyApp
  runApp(const MyApp());
}

// Définition de la classe stateless MyApp
class MyApp extends StatelessWidget {
  // Constructeur constant avec une clé facultative
  const MyApp({super.key});

  // Ce widget est la racine de l'application
  @override
  Widget build(BuildContext context) {
    // Retourne un MaterialApp configuré
    return MaterialApp(
      // Titre de l'application
      title: 'Flutter Application',
      // Thème de l'application avec une palette de couleurs personnalisée
      theme: ThemeData(
        // Utilisation d'un ColorScheme basé sur une couleur de départ
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        // Activation de Material Design 3
        useMaterial3: true,
      ),
      // Désactivation du bandeau "DEBUG"
      debugShowCheckedModeBanner: false,
      // Définition de la page d'accueil de l'application
      home: Scaffold(
        // AppBar de l'application
```

```
        appBar: AppBar(
      // Couleur de fond de l'AppBar utilisant le schéma de couleurs inversé du thème
          backgroundColor: Theme.of(context).colorScheme.inversePrimary,
          // Titre de l'AppBar
          title: const Text('Home'),
        ),
        // Corps du Scaffold
        body: const Text('Hello Flutter'),
      ),
    );
  }
}
```

Le widget body est une propriété très importante de Scaffold, puisqu'il affiche les principaux composants de l'écran.

Résultat



Explication

Dans Flutter, AppBar (La barre d'applications) comprend une barre d'outil (Tool Bar) et autres Widget potentiels. Précisément, AppBar est divisée en cinq zones : leading, title, Tool Bar (actions), flexiableSpace, bottom.

Exemple leading et actions

Dans cet exemple, on va ajouter à l'AppBar :

- Un bouton leading qui affiche une icône de menu (utilisant Icons.menu) et qui ouvre le Drawer lorsqu'on appuie dessus.

- Deux boutons actions :

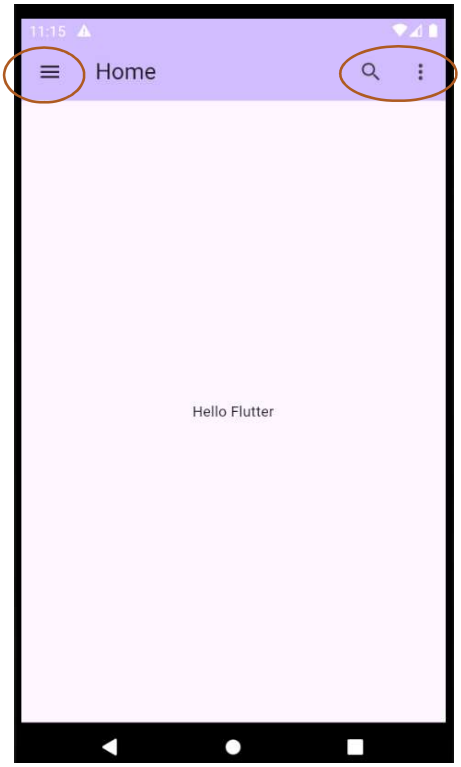
    Un pour la recherche (utilisant Icons.search).

    Un pour d'autres options (utilisant Icons.more_vert).

```dart
// Importation du package Flutter Material
import 'package:flutter/material.dart';

// Fonction principale qui lance l'application Flutter
void main() {
  // Lance l'application en exécutant MyApp
  runApp(const MyApp());
}

// Définition de la classe stateless MyApp
class MyApp extends StatelessWidget {
  // Constructeur constant avec une clé facultative
  const MyApp({super.key});

  // Ce widget est la racine de l'application
  @override
  Widget build(BuildContext context) {
    // Retourne un MaterialApp configuré
    return MaterialApp(
      // Titre de l'application
      title: 'Flutter Application',
      // Thème de l'application avec une palette de couleurs personnalisée
      theme: ThemeData(
        // Utilisation d'un ColorScheme basé sur une couleur de départ
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        // Activation de Material Design 3
        useMaterial3: true,
      ),
      // Désactivation du bandeau "DEBUG"
      debugShowCheckedModeBanner: false,
```

```
      // Définition de la page d'accueil de l'application
      home: Scaffold(
        // AppBar de l'application
        appBar: AppBar(
          // Couleur de fond de l'AppBar utilisant le schéma de couleurs inversé du
thème
          backgroundColor: Theme.of(context).colorScheme.inversePrimary,
          // Titre de l'AppBar
          title: const Text('Home'),
          // Ajout du leading à l'AppBar
          leading: IconButton(
            icon: const Icon(Icons.menu),
            onPressed: () {
              // Action à réaliser lorsqu'on appuie sur le leading
              // Par exemple, ouvrir un drawer ou afficher un menu
              Scaffold.of(context).openDrawer();
            },
          ),
          // Ajout des actions à l'AppBar
          actions: <Widget>[
            IconButton(
              icon: const Icon(Icons.search),
              onPressed: () {
                // Action à réaliser lorsqu'on appuie sur l'icône de recherche
                // Par exemple, ouvrir une barre de recherche
              },
            ),
            IconButton(
              icon: const Icon(Icons.more_vert),
              onPressed: () {
                // Action à réaliser lorsqu'on appuie sur l'icône de menu
                // Par exemple, afficher plus d'options
              },
            ),
          ],
        ),
        // Corps du Scaffold
        body: const Center(
          child: Text('Hello Flutter'),
        ),
      ),
    );
  }
}
```

Résultat



Explication

IconButton est un widget dans Flutter qui permet de créer un bouton avec une icône. C'est un bouton cliquable qui ne contient que l'icône sans texte. Vous pouvez définir l'icône à afficher et spécifier l'action à exécuter lorsqu'il est cliqué.

icon : Représente l'icône affichée dans le bouton. C'est un widget Icon qui prend un type d'icône pré-défini (comme Icons.menu, Icons.search, etc.).

onPressed : C'est une fonction de rappel (callback) qui est déclenchée lorsqu'on appuie sur le bouton. Si vous laissez onPressed null, le bouton sera désactivé.

tooltip (facultatif) : Un texte d'aide qui s'affiche lorsque l'utilisateur maintient le bouton enfoncé (ou passe la souris dessus sur le web).

Exemple FloatingActionButton

Le FloatingActionButton est un bouton flottant dans Flutter, généralement utilisé pour représenter une action principale dans une application. Il est souvent circulaire et positionné au-dessus du contenu de l'interface utilisateur, ce qui le rend facilement accessible et visuellement distinct.

```
// Importation du package Flutter Material
import 'package:flutter/material.dart';
```

```dart
// Fonction principale qui lance l'application Flutter
void main() {
  // Lance l'application en exécutant MyApp
  runApp(const MyApp());
}

// Définition de la classe stateless MyApp
class MyApp extends StatelessWidget {
  // Constructeur constant avec une clé facultative
  const MyApp({super.key});

  // Ce widget est la racine de l'application
  @override
  Widget build(BuildContext context) {
    // Retourne un MaterialApp configuré
    return MaterialApp(
      // Titre de l'application
      title: 'Flutter Application',
      // Thème de l'application avec une palette de couleurs personnalisée
      theme: ThemeData(
        // Utilisation d'un ColorScheme basé sur une couleur de départ
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        // Activation de Material Design 3
        useMaterial3: true,
      ),
      // Désactivation du bandeau "DEBUG"
      debugShowCheckedModeBanner: false,
      // Définition de la page d'accueil de l'application
      home: Scaffold(
        // AppBar de l'application
        appBar: AppBar(
          // Couleur de fond de l'AppBar utilisant le schéma de couleurs inversé du
thème
          backgroundColor: Theme.of(context).colorScheme.inversePrimary,
          // Titre de l'AppBar
          title: const Text('Home'),
          // Ajout du leading à l'AppBar
          leading: IconButton(
            icon: const Icon(Icons.menu),
            onPressed: () {
              // Action à réaliser lorsqu'on appuie sur le leading
              // Par exemple, ouvrir un drawer ou afficher un menu
              Scaffold.of(context).openDrawer();
            },
          ),
          // Ajout des actions à l'AppBar
          actions: <Widget>[
            IconButton(
              icon: const Icon(Icons.search),
              onPressed: () {
                // Action à réaliser lorsqu'on appuie sur l'icône de recherche
                // Par exemple, ouvrir une barre de recherche
```

```
              },
            ),
            IconButton(
              icon: const Icon(Icons.more_vert),
              onPressed: () {
                // Action à réaliser lorsqu'on appuie sur l'icône de menu
                // Par exemple, afficher plus d'options
              },
            ),
          ],
        ),
        // Corps du Scaffold
        body: const Center(
          child: Text('Hello Flutter'),
        ),
        floatingActionButton: FloatingActionButton(
          backgroundColor: Colors.purple[300],
          onPressed: () {
            print('action bouton Ajout');
          },
          child: const Icon(Icons.add),
        ),

      ),
    );
  }
}
```

Résultat

```
I/flutter ( 8569): action bouton Ajout
```

Exemple Image.network

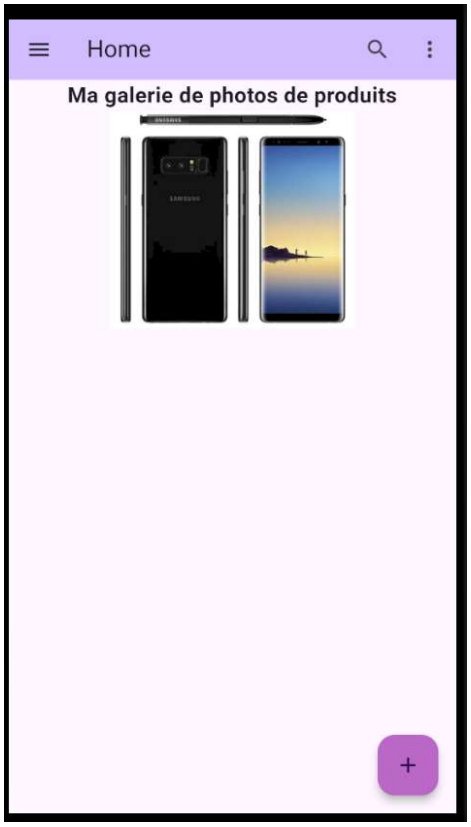
Image est un widget dans Flutter qui permet de charger et d'afficher une image. Lorsque l'image provient d'une URL on met **Image.network**. Il est utilisé pour afficher des images hébergées sur le web directement dans votre application Flutter.

```dart
// Importation du package Flutter Material
import 'package:flutter/material.dart';

// Fonction principale qui lance l'application Flutter
void main() {
  // Lance l'application en exécutant MyApp
  runApp(const MyApp());
}

// Définition de la classe stateless MyApp
class MyApp extends StatelessWidget {
  // Constructeur constant avec une clé facultative
  const MyApp({super.key});

  // Ce widget est la racine de l'application
  @override
  Widget build(BuildContext context) {
    // Retourne un MaterialApp configuré
    return MaterialApp(
      // Titre de l'application
      title: 'Flutter Application',
      // Thème de l'application avec une palette de couleurs personnalisée
      theme: ThemeData(
        // Utilisation d'un ColorScheme basé sur une couleur de départ
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        // Activation de Material Design 3
        useMaterial3: true,
      ),
      // Désactivation du bandeau "DEBUG"
      debugShowCheckedModeBanner: false,
      // Définition de la page d'accueil de l'application
      home: Scaffold(
        // AppBar de l'application
        appBar: AppBar(
          // Couleur de fond de l'AppBar utilisant le schéma de couleurs inversé du
thème
          backgroundColor: Theme.of(context).colorScheme.inversePrimary,
          // Titre de l'AppBar
          title: const Text('Home'),
          // Ajout du leading à l'AppBar
          leading: IconButton(
```

```
            icon: const Icon(Icons.menu),
            onPressed: () {
              // Action à réaliser lorsqu'on appuie sur le leading
              // Par exemple, ouvrir un drawer ou afficher un menu
              Scaffold.of(context).openDrawer();
            },
          ),
          // Ajout des actions à l'AppBar
          actions: <Widget>[
            IconButton(
              icon: const Icon(Icons.search),
              onPressed: () {
                // Action à réaliser lorsqu'on appuie sur l'icône de recherche
                // Par exemple, ouvrir une barre de recherche
              },
            ),
            IconButton(
              icon: const Icon(Icons.more_vert),
              onPressed: () {
                // Action à réaliser lorsqu'on appuie sur l'icône de menu
                // Par exemple, afficher plus d'options
              },
            ),
          ],
        ),
        // Corps du Scaffold
        body: Center(
        child: Column(
          children: [
            const Text(
              "Ma galerie de photos de produits",
              style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
            ),
            Image.network('https://cdn.lesnumeriques.com/optim/produits/36/35887/galaxy-note-8_de798d6a6e39be82__450_400.jpg',
            height: 200,
            width: 300,
            ),
          ],
        ),
      ),
       floatingActionButton: FloatingActionButton(
          backgroundColor: Colors.purple[300],
          onPressed: () {
            print('action bouton Ajout');
          },
          child: const Icon(Icons.add),
        ),

      ),
    );
  }
```

```
}
```

Résultat



Explication

Center est un widget qui centre son enfant dans l'espace disponible.

child: Column(: Center a un enfant qui est un Column.

Column est un widget de disposition qui place ses enfants (widgets) en une seule colonne verticale.

Dans ce cas, le Column va aligner ses enfants (le texte et l'image) verticalement, un au-dessus de l'autre.

children: [: Column prend une liste de widgets dans son paramètre children.

Chaque widget dans cette liste est un enfant du Column, et sera affiché verticalement.

const Text(: Le premier widget dans la liste des children est un Text.

Ce Text affiche la chaîne de caractères "Ma galerie de photos de produits".

Le mot-clé const est utilisé ici pour indiquer que ce widget ne changera jamais, ce qui peut améliorer les performances.

style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold) : Le texte est stylisé pour avoir une taille de police de 20 et être en gras (fontWeight: FontWeight.bold).

Le deuxième widget dans la liste des children est un Image.network.

Image.network est utilisé pour afficher une image à partir d'une URL.

Exemple Padding et SizedBox

Pour ajouter un espace entre le texte et le haut de la colonne ainsi qu'entre le texte et l'image, vous pouvez utiliser les widgets Padding ou SizedBox.

```
// Importation du package Flutter Material
import 'package:flutter/material.dart';

// Fonction principale qui lance l'application Flutter
void main() {
  // Lance l'application en exécutant MyApp
  runApp(const MyApp());
}

// Définition de la classe stateless MyApp
class MyApp extends StatelessWidget {
  // Constructeur constant avec une clé facultative
  const MyApp({super.key});

  // Ce widget est la racine de l'application
  @override
  Widget build(BuildContext context) {
    // Retourne un MaterialApp configuré
    return MaterialApp(
      // Titre de l'application
      title: 'Flutter Application',
      // Thème de l'application avec une palette de couleurs personnalisée
      theme: ThemeData(
        // Utilisation d'un ColorScheme basé sur une couleur de départ
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        // Activation de Material Design 3
        useMaterial3: true,
      ),
      // Désactivation du bandeau "DEBUG"
      debugShowCheckedModeBanner: false,
      // Définition de la page d'accueil de l'application
      home: Scaffold(
        // AppBar de l'application
        appBar: AppBar(
          // Couleur de fond de l'AppBar utilisant le schéma de couleurs inversé du
thème
          backgroundColor: Theme.of(context).colorScheme.inversePrimary,
          // Titre de l'AppBar
          title: const Text('Home'),
          // Ajout du leading à l'AppBar
          leading: IconButton(
            icon: const Icon(Icons.menu),
            onPressed: () {
              // Action à réaliser lorsqu'on appuie sur le leading
              // Par exemple, ouvrir un drawer ou afficher un menu
              Scaffold.of(context).openDrawer();
            },
```

```
          ),
          // Ajout des actions à l'AppBar
          actions: <Widget>[
            IconButton(
              icon: const Icon(Icons.search),
              onPressed: () {
                // Action à réaliser lorsqu'on appuie sur l'icône de recherche
                // Par exemple, ouvrir une barre de recherche
              },
            ),
            IconButton(
              icon: const Icon(Icons.more_vert),
              onPressed: () {
                // Action à réaliser lorsqu'on appuie sur l'icône de menu
                // Par exemple, afficher plus d'options
              },
            ),
          ],
        ),
        // Corps du Scaffold
   body: Center(
  child: Column(
    children: [
      const Padding(
        padding: EdgeInsets.only(top: 20.0), // Espace entre le haut de la colonne
et le texte
        child: Text(
          "Ma galerie de photos de produits",
          style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
        ),
      ),
      const SizedBox(height: 20), // Espace entre le texte et l'image
      Image.network(
        'https://cdn.lesnumeriques.com/optim/produits/36/35887/galaxy-note-
8_de798d6a6e39be82__450_400.jpg',
        height: 200,
        width: 300,
      ),
    ],
  ),
),
       floatingActionButton: FloatingActionButton(
          backgroundColor: Colors.purple[300],
          onPressed: () {
            print('action bouton Ajout');
          },
          child: const Icon(Icons.add),
        ),

      ),
    );
  }
```

```
}
```

Explication

Padding : Le widget Padding est utilisé pour ajouter un espace entre le haut de la colonne et le texte. Ici, EdgeInsets.only(top: 20.0) ajoute 20 pixels d'espace au-dessus du texte.

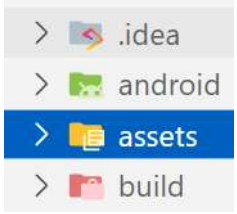
SizedBox : Le widget SizedBox est utilisé pour ajouter un espace fixe entre le texte et l'image. SizedBox(height: 20) ajoute 20 pixels d'espace vertical.

Résultat

Image.asset

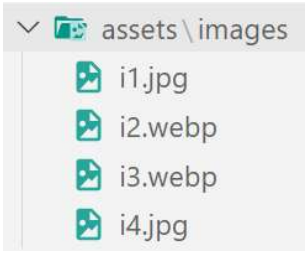
Les images sont affichées à partir d'Internet, si on veut les afficher à partir d'un dossier local :

1. Créer un nouveau dossier appelé /assets dans votre projet (nomDuProjet/assets/).



2. Dans assets créer un dossier appelé images

3. Mettre des images dans le dossier assets/images/



4. Définir les assets à utiliser dans pubspec.yaml.

Avant :

```
# assets:
#   - images/a_dot_burr.jpeg
#   - images/a_dot_ham.jpeg
```

Après :

```
assets:
  - assets/images/i1.jpg
  - assets/images/i2.webp
  - assets/images/i3.webp
  - assets/images/i4.jpg
```

Attention à la tabulation ! Assurez-vous que l'indentation est correcte dans le fichier pubspec.yaml. Flutter utilise YAML, qui est sensible à l'indentation. La section assets doit être correctement indentée sous la clé flutter.

C'est possible également de mettre juste le repertoire des images.

```
assets:
   - assets/images/
```

5. Charger l'image à partir des assets dans le code.

```
// Importation du package Flutter Material
```

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

// Fonction principale qui lance l'application Flutter
void main() {
  // Lance l'application en exécutant MyApp
  runApp(const MyApp());
}

// Définition de la classe stateless MyApp
class MyApp extends StatelessWidget {
  // Constructeur constant avec une clé facultative
  const MyApp({super.key});

  // Ce widget est la racine de l'application
  @override
  Widget build(BuildContext context) {
    // Retourne un MaterialApp configuré
    return MaterialApp(
      // Titre de l'application
      title: 'Flutter Application',
      // Thème de l'application avec une palette de couleurs personnalisée
      theme: ThemeData(
        // Utilisation d'un ColorScheme basé sur une couleur de départ
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        // Activation de Material Design 3
        useMaterial3: true,
      ),
      // Désactivation du bandeau "DEBUG"
      debugShowCheckedModeBanner: false,
      // Définition de la page d'accueil de l'application
      home: Scaffold(
        // AppBar de l'application
        appBar: AppBar(
          // Couleur de fond de l'AppBar utilisant le schéma de couleurs inversé du
thème
          backgroundColor: Theme.of(context).colorScheme.inversePrimary,
          // Titre de l'AppBar
          title: const Text('Home'),
          // Ajout du leading à l'AppBar
          leading: IconButton(
            icon: const Icon(Icons.menu),
            onPressed: () {
              // Action à réaliser lorsqu'on appuie sur le leading
              // Par exemple, ouvrir un drawer ou afficher un menu
              Scaffold.of(context).openDrawer();
            },
          ),
          // Ajout des actions à l'AppBar
          actions: <Widget>[
            IconButton(
              icon: const Icon(Icons.search),
              onPressed: () {
```

```
                // Action à réaliser lorsqu'on appuie sur l'icône de recherche
                // Par exemple, ouvrir une barre de recherche
              },
            ),
            IconButton(
              icon: const Icon(Icons.more_vert),
              onPressed: () {
                // Action à réaliser lorsqu'on appuie sur l'icône de menu
                // Par exemple, afficher plus d'options
              },
            ),
          ],
        ),
        // Corps du Scaffold
        body: Center(
          child: Column(
            children: [
              const Padding(
                padding: EdgeInsets.only(
                    top:
                        20.0), // Espace entre le haut de la colonne et le texte
                child: Text(
                  "Ma galerie de photos de produits",
                  style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
                ),
              ),
              const SizedBox(height: 20), // Espace entre le texte et l'image
              Image.asset(
                'assets/images/i1.jpg',
                height: 200,
                width: 300,
              ),
              const SizedBox(height: 20), // Espace entre les 2 images
              Image.asset(
                'assets/images/i2.webp',
                height: 200,
                width: 300,
              ),
            ],
          ),
        ),
        floatingActionButton: FloatingActionButton(
          backgroundColor: Colors.purple[300],
          onPressed: () {
            print('action bouton Ajout');
          },
          child: const Icon(Icons.add),
        ),
      ),
    );
  }
}
```

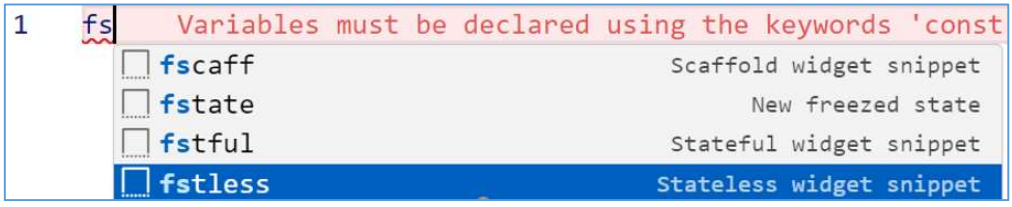
Résultat

Passage Direct de Widget

On va assigner directement un widget (const HomePage()) à la propriété du widget parent (MaterialApp). Cela signifie que le widget HomePage sera le widget de base affiché lorsque l'application démarre.

D'abord, on va créer le fichier /lib/**screens**/**myproducts.dart**

Remarque : on peut utiliser le raccourci : `fstless` si l'extension **Flutter widget snippets** est installée



Puis ajouter

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

Puis on va y mettre le code qui existe dans la partie **body** de **main.dart**, après **return** (à la place de Container).

Et n'oubliez pas le point virgule à la fin.

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

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

  @override
  Widget build(BuildContext context){
    return Center(
          child: Column(
            children: [
              const Padding(
                padding: EdgeInsets.only(
                    top:
                        20.0), // Espace entre le haut de la colonne et le texte
                child: Text(
                  "Ma galerie de photos de produits",
                  style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
                ),
              ),
              const SizedBox(height: 20), // Espace entre le texte et l'image
              Image.asset(
                'assets/images/i1.jpg',
                height: 200,
                width: 300,
              ),
              const SizedBox(height: 20), // Espace entre les 2 images
              Image.asset(
                'assets/images/i2.webp',
                height: 200,
                width: 300,
              ),
            ],
          ),
        );
  }
}
```

Puis faire l'appel de cette page dans main


```dart
// Importation du package Flutter Material
import 'package:flutter/material.dart';
```

```dart
import 'package:myflutterapplication/screens/myproducts.dart';

// Fonction principale qui lance l'application Flutter
void main() {
  // Lance l'application en exécutant MyApp
  runApp(const MyApp());
}

// Définition de la classe stateless MyApp
class MyApp extends StatelessWidget {
  // Constructeur constant avec une clé facultative
  const MyApp({super.key});

  // Ce widget est la racine de l'application
  @override
  Widget build(BuildContext context) {
    // Retourne un MaterialApp configuré
    return MaterialApp(
      // Titre de l'application
      title: 'Flutter Application',
      // Thème de l'application avec une palette de couleurs personnalisée
      theme: ThemeData(
        // Utilisation d'un ColorScheme basé sur une couleur de départ
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        // Activation de Material Design 3
        useMaterial3: true,
      ),
      // Désactivation du bandeau "DEBUG"
      debugShowCheckedModeBanner: false,
      // Définition de la page d'accueil de l'application
      home: Scaffold(
        // AppBar de l'application
        appBar: AppBar(
          // Couleur de fond de l'AppBar utilisant le schéma de couleurs inversé du
thème
          backgroundColor: Theme.of(context).colorScheme.inversePrimary,
          // Titre de l'AppBar
          title: const Text('Home'),
          // Ajout du leading à l'AppBar
          leading: IconButton(
            icon: const Icon(Icons.menu),
            onPressed: () {
              // Action à réaliser lorsqu'on appuie sur le leading
              // Par exemple, ouvrir un drawer ou afficher un menu
              Scaffold.of(context).openDrawer();
            },
          ),
          // Ajout des actions à l'AppBar
          actions: <Widget>[
            IconButton(
              icon: const Icon(Icons.search),
              onPressed: () {
```

```
                // Action à réaliser lorsqu'on appuie sur l'icône de recherche
                // Par exemple, ouvrir une barre de recherche
              },
            ),
            IconButton(
              icon: const Icon(Icons.more_vert),
              onPressed: () {
                // Action à réaliser lorsqu'on appuie sur l'icône de menu
                // Par exemple, afficher plus d'options
              },
            ),
          ],
        ),
        // Corps du Scaffold
        body: const Myproducts(),
        floatingActionButton: FloatingActionButton(
          backgroundColor: Colors.purple[300],
          onPressed: () {
            print('action bouton Ajout');
          },
          child: const Icon(Icons.add),
        ),
      ),
    );
  }
}
```

On va procéder à mettre le contenu de AppBar dans un autre fichierqu'on va appeler **myappbar.dart** puis de l'appeler dans **main.dart**

Créer /lib/**widgets**/**myappbar.dart**


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

class Myappbar extends StatelessWidget implements PreferredSizeWidget {
  const Myappbar({super.key})
      : preferredSize = const Size.fromHeight(56.0);

  @override
  final Size preferredSize;

  @override
  Widget build(BuildContext context){
    return AppBar(
      backgroundColor: Theme.of(context).colorScheme.inversePrimary,
      title: const Text('Home'),
      leading: IconButton(
```

```
        icon: const Icon(Icons.menu),
        onPressed: () {
          Scaffold.of(context).openDrawer();
        },
      ),
      actions: <Widget>[
        IconButton(
          icon: const Icon(Icons.search),
          onPressed: () {
            // Action pour l'icône de recherche
          },
        ),
        IconButton(
          icon: const Icon(Icons.more_vert),
          onPressed: () {
            // Action pour l'icône de menu
          },
        ),
      ],
    );
  }
}
```

Explication

L'ajout de @override final Size preferredSize; dans le code permet à Myappbar d'implémenter correctement l'interface PreferredSizeWidget. Cela assure que l'AppBar personnalisé est correctement intégré dans l'interface utilisateur avec la taille souhaitée.

La classe Myappbar implémente PreferredSizeWidget, ce qui signifie qu'elle doit fournir la propriété preferredSize.

@override final Size preferredSize; : Ici, on déclare preferredSize comme une propriété finale, ce qui signifie que sa valeur est fixée lors de l'initialisation de l'objet et ne peut pas être modifiée après.

La valeur de preferredSize est fixée à une hauteur constante de 56.0 pixels, indiquant que l'AppBar doit avoir cette hauteur.

Le constructeur Myappbar initialise preferredSize avec une taille constante pour garantir que l'AppBar respecte les dimensions spécifiées.

Par la suite on va faire l'appel de cette page dans main


```
// Importation du package Flutter Material
```

```dart
import 'package:flutter/material.dart';
import 'package:myflutterapplication/screens/myproducts.dart';
import 'package:myflutterapplication/widgets/myappbar.dart';

// Fonction principale qui lance l'application Flutter
void main() {
  // Lance l'application en exécutant MyApp
  runApp(const MyApp());
}

// Définition de la classe stateless MyApp
class MyApp extends StatelessWidget {
  // Constructeur constant avec une clé facultative
  const MyApp({super.key});

  // Ce widget est la racine de l'application
  @override
  Widget build(BuildContext context) {
    // Retourne un MaterialApp configuré
    return MaterialApp(
      // Titre de l'application
      title: 'Flutter Application',
      // Thème de l'application avec une palette de couleurs personnalisée
      theme: ThemeData(
        // Utilisation d'un ColorScheme basé sur une couleur de départ
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        // Activation de Material Design 3
        useMaterial3: true,
      ),
      // Désactivation du bandeau "DEBUG"
      debugShowCheckedModeBanner: false,
      // Définition de la page d'accueil de l'application
      home: Scaffold(
        // AppBar de l'application
        appBar:  const Myappbar(),
        // Corps du Scaffold
        body: const Myproducts(),
        floatingActionButton: FloatingActionButton(
          backgroundColor: Colors.purple[300],
          onPressed: () {
            print('action bouton Ajout');
          },
          child: const Icon(Icons.add),
        ),
      ),
    );
  }
}
```

V. Le StatefulWidget

Contrairement au StatelessWidget, le StatefulWidget est un widget qui possède une partie mutable. Il est qualifié de widget avec état.

StateFulWidget permet de créer un widget qui possède un état modifiable. Un widget avec état est dynamique.

StateFulWidget utilise l'objet State où les valeurs qui peuvent être modifiées sont stockées. En utilisant la méthode setState(), les valeurs stockées dans l'objet State peuvent être modifiées et cela permet de redessiner le widget.

En fait, un « État » n'est rien d'autre que les données qui ont été présentées ou calculées pendant la durée de vie d'un widget. Lorsque nous fermons l'application ou effectuons une opération qui rafraîchit l'écran, les données du widget peuvent être ignorées ou conservées pour une utilisation future. Ce comportement différencie un widget avec état d'un widget sans état.

Un widget avec état dépend soit de l'action de l'utilisateur, soit de la modification des données.

La classe ou l'objet State gère en fait l'état interne du widget avec état. Donc, un widget avec état dépend également de la classe State, qui n'est pas un widget.

Le StatefulWidget possède un objet appelé State. C'est lui qui est chargé de stocker toutes les variables mutables.

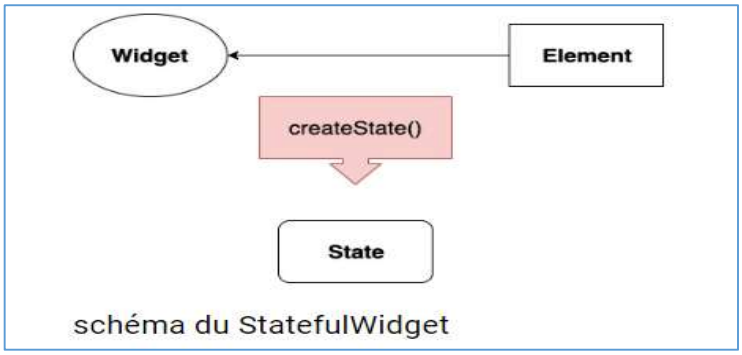


schéma du StatefulWidget

Étapes pour implémenter Stateful Widget

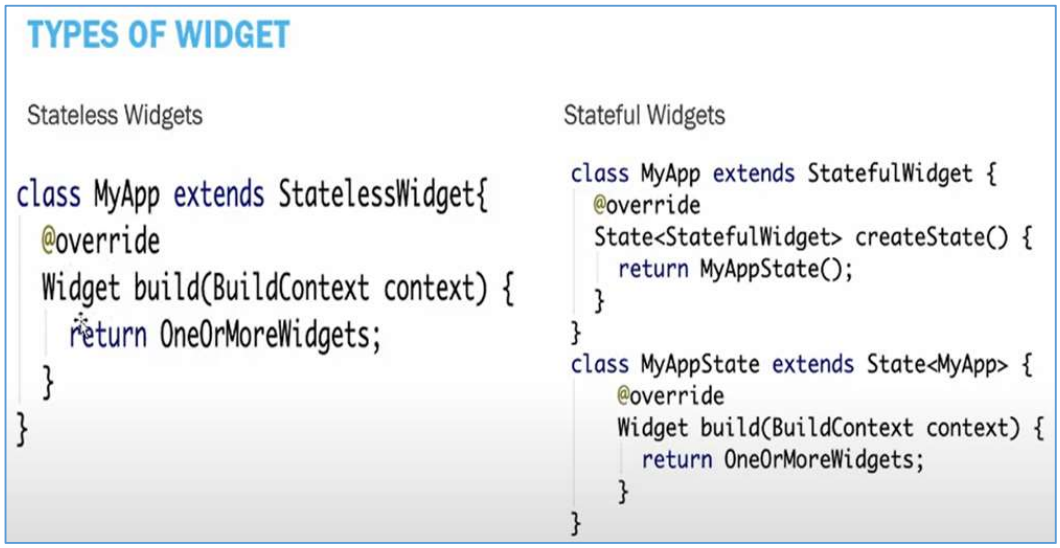
1- Créez une classe qui étend 'StatefulWidget', qui renvoie l'état dans 'createState()'

2- Créez une classe 'State' pour les widgets qui peuvent changer leurs valeurs pendant l'exécution.

3- Dans la classe 'State', implémentez la méthode 'build()'.

4- Appelez la fonction 'setState()'. La fonction 'setState()' redessine en fait les widgets.

Différence entre stateless et stateful widgets dans la syntaxe :



Créer le fichier /lib/**widgets**/**mydiapos.dart**


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

class Mydiapos extends StatefulWidget {
  const Mydiapos({super.key});

  @override
  MydiaposState createState() => MydiaposState();
}

class MydiaposState extends State<Mydiapos> {
  int counter = 0; // Compteur pour l'image actuelle
  List<String> images = [
    'assets/images/i1.jpg',
    'assets/images/i2.webp',
    'assets/images/i3.webp',
    'assets/images/i4.jpg'
  ];

  // Méthode pour passer à l'image suivante
  void _nextImage() {
    setState(() {
      if (counter < images.length - 1) {
        counter++;
      } else {
        counter = 0; // Retour au début pour un diaporama infini
      }
    });
  }
```

```
  // Méthode pour revenir à l'image précédente
  void _previousImage() {
    setState(() {
      if (counter > 0) {
        counter--;
      } else {
        counter = images.length - 1; // Aller à la dernière image pour un diaporama
infini
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center, // Centrer le contenu
      crossAxisAlignment: CrossAxisAlignment.center, // Centrer le contenu
      children: [
        Text(
          "Image Numéro : ${counter + 1}",
          style: const TextStyle(fontSize: 30, fontWeight: FontWeight.bold),
        ),
        const SizedBox(height: 20), // Espacement entre le texte et l'image
        Image.asset(
          images[counter],
          width: 300, // Largeur fixe
          height: 200, // Hauteur fixe
          fit: BoxFit.cover, // Ajuster le contenu de l'image à la taille du
conteneur
        ),
        const SizedBox(height: 20), // Espacement entre l'image et les boutons
        Row(
          mainAxisAlignment: MainAxisAlignment.center, // Centrer les boutons
          children: [
            ElevatedButton(
              onPressed: _previousImage,
              child: const Text('Précédent'),
            ),
            const SizedBox(width: 20), // Espacement entre les boutons
            ElevatedButton(
              onPressed: _nextImage,
              child: const Text('Suivant'),
            ),
          ],
        ),
      ],
    );
  }
}
```

Explication

L'utilisation de clés (key) aide également Flutter à préserver l'état des StatefulWidgets lorsqu'ils sont remplacés par d'autres widgets ou simplement déplacés dans l'arborescence des widgets. Presque tous les widgets Flutter acceptent les clés comme paramètres facultatifs dans leurs constructeurs.

Plusieurs widgets du même type et au même niveau dans une arborescence de widgets peuvent ne pas se mettre à jour comme prévu à moins qu'ils n'aient des clés uniques, étant donné que ces widgets possèdent un certain état.

Définir explicitement une clé sur un widget aide Flutter à comprendre quel widget il doit mettre à jour lorsque l'état change.

De plus, nous avons une classe Mydiapos qui hérite de StatefulWidget et elle implémente la méthode createState(). Cette méthode renvoie l'instance de la classe MydiaposState et à l'intérieur de cette classe, nous redéfinissons la méthode build.

Donc, l'avantage de surcharger la méthode build dans MydiaposState est qu'elle sera appelée chaque fois qu'il y aura un changement dans les variables associées aux Widgets et l'ensemble du widget sera à nouveau redessiné. Mais pour appeler la méthode build, vous devez ajouter une autre méthode appelée setState() qui appellera la méthode de construction chaque fois qu'il y a un changement d'état.

Méthodes _nextImage et _previousImage :

_nextImage : Augmente le compteur de 1, et le remet à 0 si le compteur atteint la fin de la liste (images.length - 1) en changeant le state de counter.

_previousImage : Diminue le compteur de 1, et le remet à la dernière image si le compteur est à 0.

Les boutons **ElevatedButton** appellent les méthodes _previousImage et _nextImage lorsqu'ils sont pressés.

Utilisation de SizedBox pour ajouter de l'espace entre les éléments pour une meilleure mise en page.

Utilisation de mainAxisAlignment et crossAxisAlignment pour centrer le contenu dans la Column et la Row.

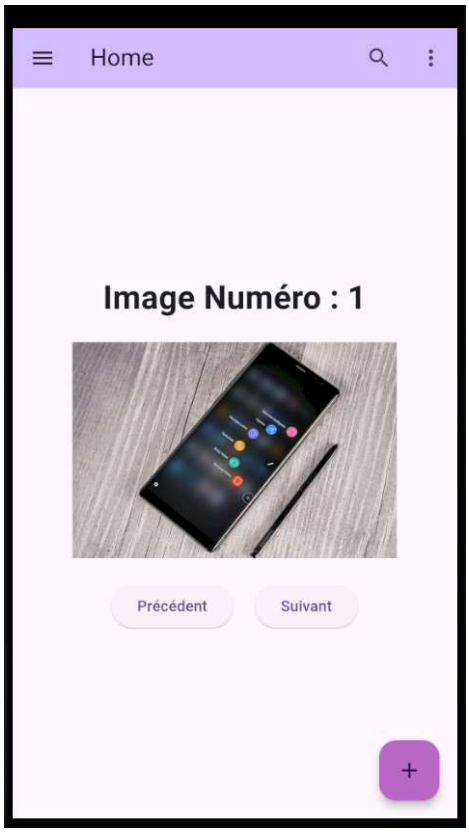
Ainsi on met en place un diaporama simple avec des boutons pour naviguer entre les images et gère le compteur pour afficher l'image actuelle et la position du diaporama.

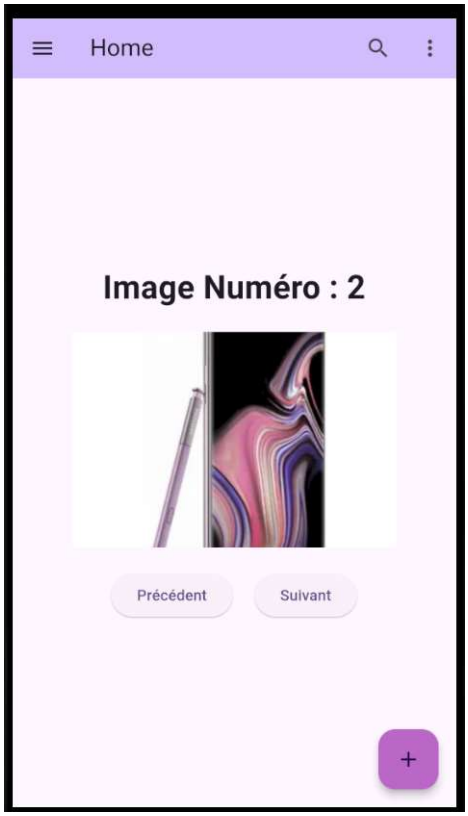
Par la suite on va faire appel à Mydiapos dans myproducts.dart


```
import 'package:flutter/material.dart';
import 'package:myflutterapplication/widgets/mydiapos.dart';
```

```dart
class Myproducts extends StatelessWidget {
const Myproducts ({ super.key });

  @override
  Widget build(BuildContext context){
    return const Center(
          child: Mydiapos()
        );
  }
}
```

Résultat





VI. GridView

Une GridView est un élément de contrôle graphique utilisé pour afficher les éléments sous forme de tableau. Dans cette section, nous allons apprendre à restituer des éléments dans une vue grille dans l'application Flutter.

GridView est un widget dans Flutter qui affiche les éléments dans un tableau 2D (lignes et colonnes bidimensionnelles). Comme son nom l'indique, il sera utilisé lorsque l'on souhaite afficher des éléments dans une grille. Nous pouvons sélectionner l'élément souhaité dans la liste de la grille en appuyant dessus. Ce widget peut contenir du texte, des images, des icônes, etc. à afficher sous forme de grille en fonction des besoins de l'utilisateur. Il est également appelé tableau de widgets 2D défilant. Puisqu'il peut défiler, nous pouvons spécifier uniquement la direction dans laquelle il défile.

La GridView peut être implémentée de différentes manières, indiquées ci-dessous :

1. count()
2. builder()
3. custom()
4. extent()

GridView.count()

Il s'agit de la disposition de grille la plus fréquemment utilisée dans Flutter car ici, nous connaissons déjà la taille de la grille. Il permet aux développeurs de spécifier le nombre fixe de lignes et de colonnes. GriedView.count() contient les propriétés suivantes :

- crossAxisCount : Il est utilisé pour spécifier le nombre de colonnes dans une vue grille.

- crossAxisSpacing : Il est utilisé pour spécifier le nombre de pixels entre chaque widget enfant répertorié dans l'axe transversal.

- mainAxisSpacing : Il est utilisé pour spécifier le nombre de pixels entre chaque widget enfant répertorié dans l'axe principal.

- padding(EdgeInsetsGeometry) : Il est utilisé pour spécifier l'espace autour de toute la liste des widgets.

- scrollDirection : Il est utilisé pour spécifier la direction dans laquelle les éléments de GridView défilent. Par défaut, il défile dans le sens vertical.

- reverse : si c'est vrai, cela inversera la liste dans la direction opposée le long de l'axe principal.

- physics : Il permet de déterminer le comportement de la liste lorsque l'utilisateur atteint la fin ou le début du widget lors du défilement.

- ShrinkWrap : S'il est faux, la liste déroulante prend plus d'espace pour défiler dans le sens du défilement. Ce n'est pas bon car cela gaspille de la mémoire et réduit les performances de l'application.

Créer le fichier /lib/**screens**/**menu.dart** pour afficher une interface utilisateur avec des options clairement définies et accessibles via des cartes, chaque carte ayant une icône et une couleur de fond personnalisée.

Le widget Menu crée une grille de cartes.

Chaque carte est représentée par un widget SelectCard contenant une icône et un titre.

Les données pour chaque carte proviennent de la liste choices.

Card est un widget intégré dans Flutter qui tire sa conception de la Material Design Library de Google.

Le widget de card en flutter est une feuille de material utilisée pour représenter toutes les informations similaires dans un seul bloc.

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

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

  @override
  Widget build(BuildContext context) {
    return GridView.count(
                crossAxisCount: 2,
                crossAxisSpacing: 10.0,
                mainAxisSpacing: 30.0,
                children: List.generate(choices.length, (index) {
                  return Center(
                    child: SelectCard(choice: choices[index]),
                  );
                }
               ),
     );
  }
}

class Choice {
  const Choice({required this.title, required this.icon, required this.colorB });
  final String title;
  final IconData icon;
  final MaterialAccentColor colorB;
}

const List<Choice> choices = <Choice>[
  Choice(title: 'Home', icon: Icons.home, colorB : Colors.purpleAccent ),
  Choice(title: 'Categories', icon: Icons.map, colorB : Colors.amberAccent),
  Choice(title: 'Documents', icon: Icons.document_scanner, colorB :
Colors.greenAccent),
  Choice(title: 'Products', icon: Icons.photo_album, colorB : Colors.pinkAccent ),
  Choice(title: 'Synchronization', icon: Icons.wifi, colorB : Colors.limeAccent ),
  Choice(title: 'Settings', icon: Icons.settings, colorB :
Colors.lightBlueAccent),

];

class SelectCard extends StatelessWidget {
  const SelectCard({super.key,  required this.choice}) ;
  final Choice choice;

  @override
  Widget build(BuildContext context) {

    return Card(
        color: Colors.white70,
        child: Center(child: Column(
            crossAxisAlignment: CrossAxisAlignment.center,
            children: <Widget>[
```

```
              Expanded(child: Icon(choice.icon, size:50.0, color: choice.colorB)),
              Text(choice.title,  style:
DefaultTextStyle.of(context).style.apply(fontSizeFactor: 1.6),),
            ]
        ),
        )
    );
  }
}
```

Explication

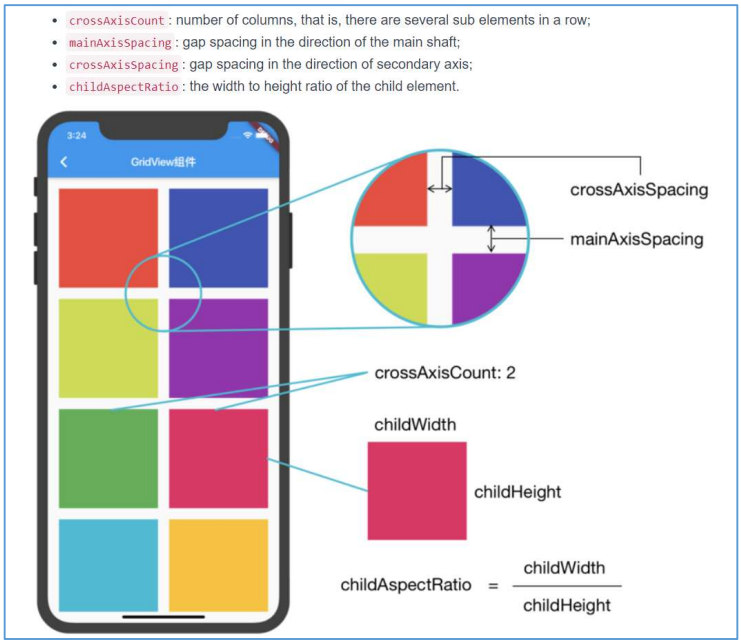
La classe Menu est un widget stateless qui utilise un GridView pour afficher une grille de cartes.

GridView.count : Crée une grille avec un nombre fixe de colonnes (ici, 2 colonnes).

crossAxisCount: 2 : Définit le nombre de colonnes.

crossAxisSpacing : Définit l'espacement entre les colonnes.

mainAxisSpacing : Définit l'espacement entre les lignes.



List.generate : Génère une liste de widgets SelectCard en fonction du nombre d'éléments dans choices. Chaque SelectCard est alimentée par un objet Choice de la liste choices.

La classe Choice représente les données d'un choix ou d'une option à afficher dans les cartes.

title : Le titre affiché sur la carte.

icon : L'icône associée à ce choix.

colorB : La couleur de l’icône.

La liste choices est une collection d'objets Choice. Chaque objet Choice contient un titre, une icône et une couleur pour une carte spécifique.

La classe SelectCard est un widget stateless qui représente une carte individuelle dans la grille.

Card : Un widget de carte avec une couleur de fond définie par choice.colorB.

Center : Centre son enfant à la fois horizontalement et verticalement.

Column : Dispose les enfants verticalement.

Expanded : Permet à l'icône de s'étendre et de prendre toute la place disponible dans la colonne.

Icon : Affiche l'icône associée au choix.

Text : Affiche le titre du choix avec une taille de police augmentée.

Par la suite faire appel à ce widget dans main.dart


```dart
// Importation du package Flutter Material
import 'package:flutter/material.dart';
import 'package:myflutterapplication/screens/menu.dart';
import 'package:myflutterapplication/widgets/myappbar.dart';

// Fonction principale qui lance l'application Flutter
void main() {
  // Lance l'application en exécutant MyApp
  runApp(const MyApp());
}

// Définition de la classe stateless MyApp
class MyApp extends StatelessWidget {
  // Constructeur constant avec une clé facultative
  const MyApp({super.key});

  // Ce widget est la racine de l'application
  @override
  Widget build(BuildContext context) {
    // Retourne un MaterialApp configuré
    return MaterialApp(
      // Titre de l'application
      title: 'Flutter Application',
      // Thème de l'application avec une palette de couleurs personnalisée
      theme: ThemeData(
        // Utilisation d'un ColorScheme basé sur une couleur de départ
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        // Activation de Material Design 3
        useMaterial3: true,
```

```
      ),
      // Désactivation du bandeau "DEBUG"
      debugShowCheckedModeBanner: false,
      // Définition de la page d'accueil de l'application
      home: const Scaffold(
        // AppBar de l'application
        appBar:  Myappbar(),
        // Corps du Scaffold
        body: Menu()
      ),
    );
  }
}
```

Résultat