Les bases de Flutter 1/2

Table des matières

I. Squelette d'une application

Contexte

Flutter repose sur des widgets qui sont des composants plus ou moins complexes qui affichent de l'information à l'écran. Lorsque vous développez en Flutter, « *tout est widget* ». Dans la documentation de Flutter, les widgets sont décrits comme étant construits à partir d'un framework moderne inspiré de React. L'idée centrale est de construire l'interface utilisateur à partir de widgets, plus celle-ci est élaborée, plus il y aura de widgets.

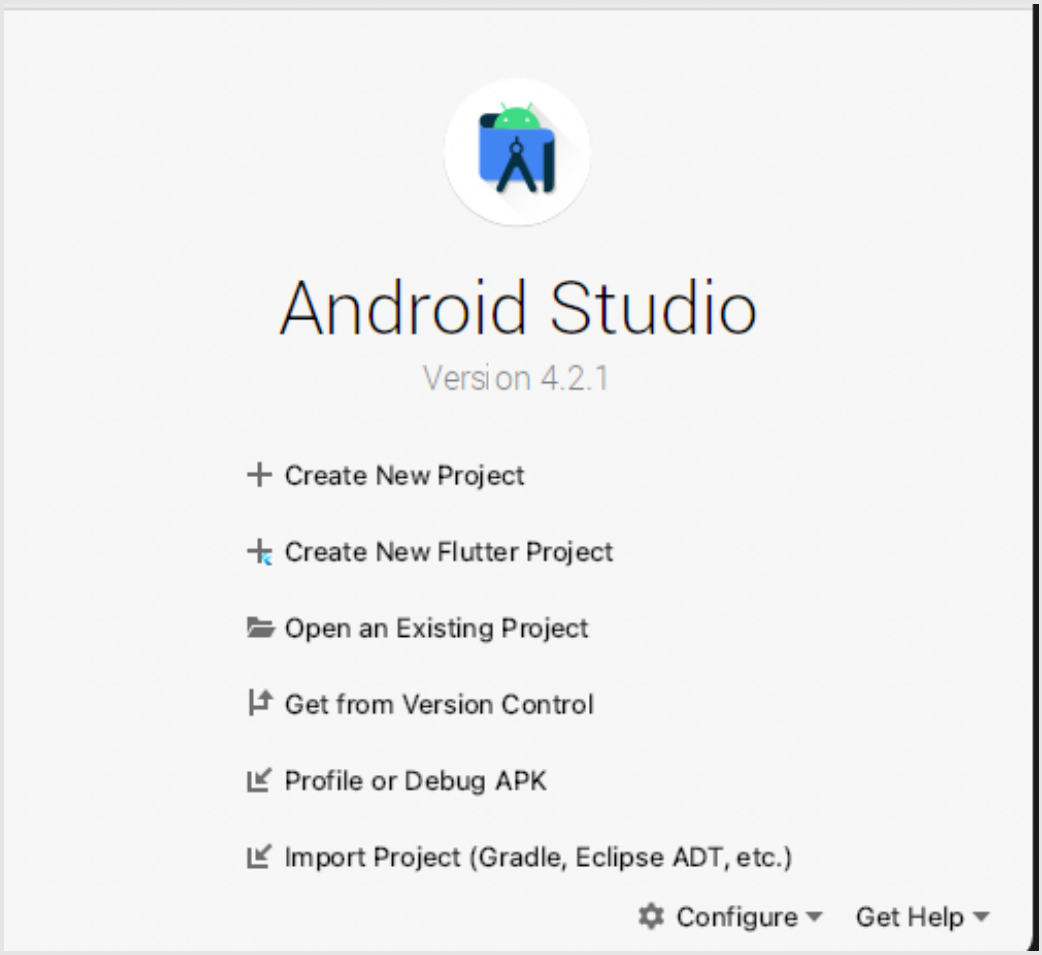
Le développeur va construire une interface avec des widgets. Ces derniers peuvent être un espacement, une vue ou une gesture. Ils ont des propriétés comme la hauteur, la largeur, la couleur de fond, etc. Ils ont un ou des enfants. Le widget se reconstruit lorsque son état change. Flutter fait une comparaison de l'état A (avant) et l'état B (après). Lors de la reconstruction du module, le nombre de modifications sera réduit au minimum. La vitesse et la qualité du rendu sont optimisées.

Comme les poupées russes, les widgets vont s'imbriquer les uns dans les autres. On va composer des vues à l'aide de widgets.

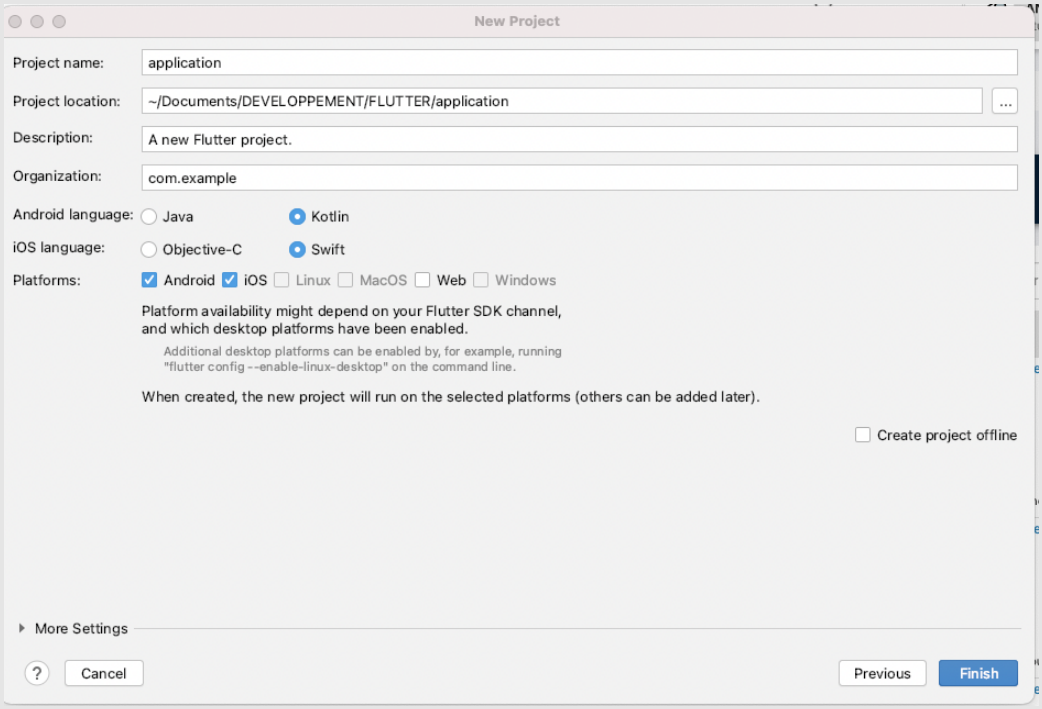
Dans ce cours, nous verrons une application de base, générée à la création d'un projet flutter. Nous verrons ensuite la construction d'un widget principal, la structure de l'application et les widgets de base.

Méthode **Création d'une application de base**

Étape 1 : après le lancement d'Android Studio, nous allons créer un nouveau projet Flutter.



On choisit : « *Create New Flutter Project* ». À l'écran suivant, il faut faire « *Next* ».

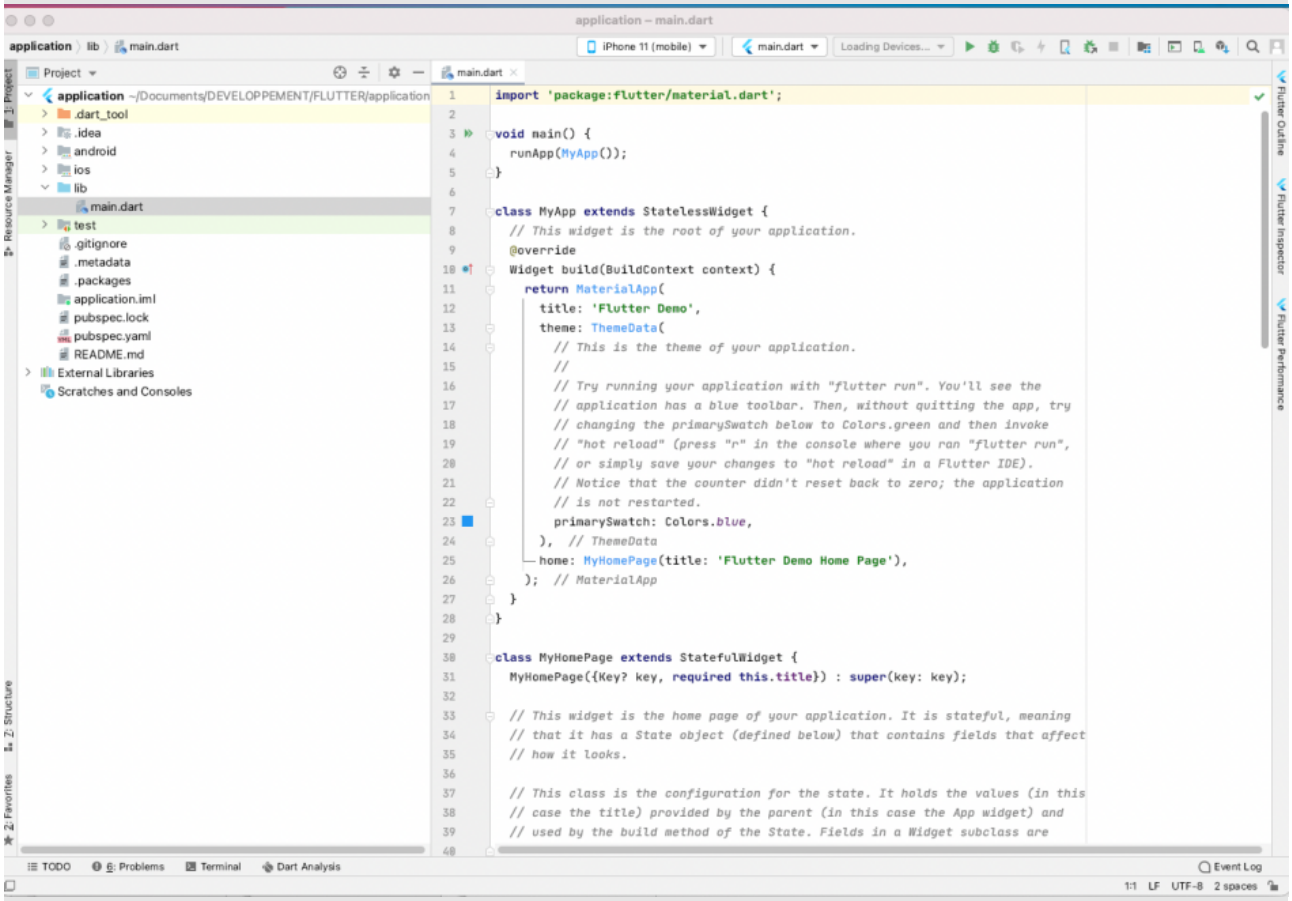


Étape 2 : ensuite on donne un nom au projet. Il faut écrire son nom en minuscule.

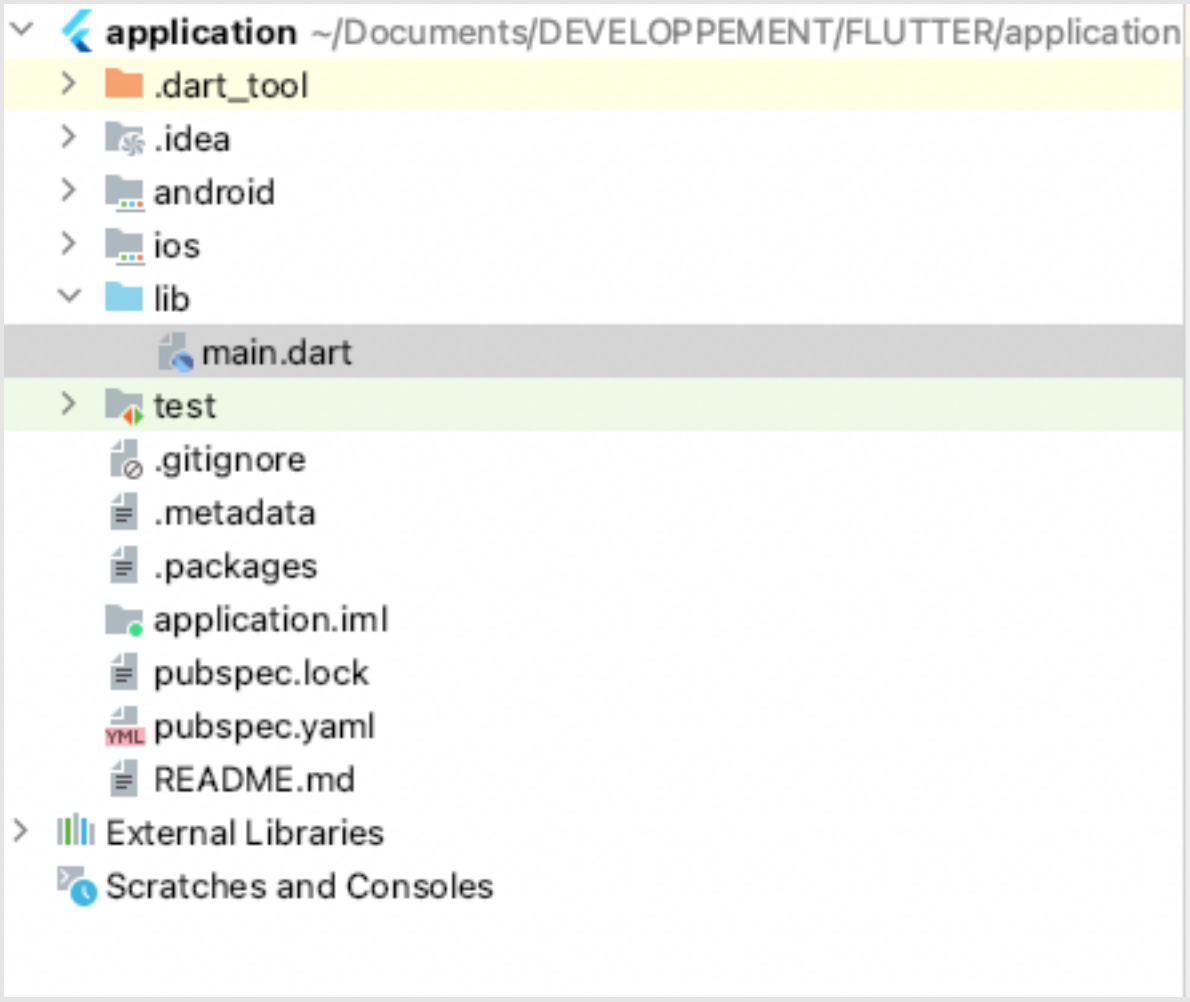
- Le champ « *Project location* » permet de définir l'emplacement où sera stocké le projet sur le disque dur,
- Le champ « *Description* » permet de faire une rapide description du projet,
- Les champs des langages de programmation : Kotlin pour Android et Swift pour iOS,
- Pour « *Plateforms* », on coche iOS et Android.

On valide. Android Studio va nous finaliser le projet.

On arrive alors sur l'écran suivant :



La fenêtre principale est divisée en deux : à gauche l'arborescence du projet et à droite le fichier sélectionné.



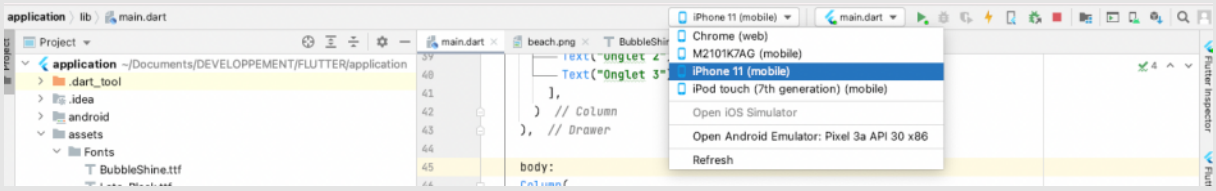
Les fichiers qui contiennent du code « *Dart* » se trouvent dans le répertoire lib. Ces fichiers ont l'extension .dart. Au départ, le projet comprend uniquement le fichier : `main.dart`. Toute l'application est dans ce fichier.

```dart
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),  // ThemeData
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );  // MaterialApp
  }
}
```

L'application retourne le widget « *MaterialApp* ». Ce widget contient les propriétés suivantes :

- **title :** pour le site titre de l'application.
- **thème :** pour choisir un thème en appelant un autre widget.
- **home :** permet de spécifier le widget de départ de notre application. Dans le cas présent, c'est `MyHomePage` qui définit le widget principal de l'application. Ensuite, nous allons le supprimer pour recréer notre propre widget.

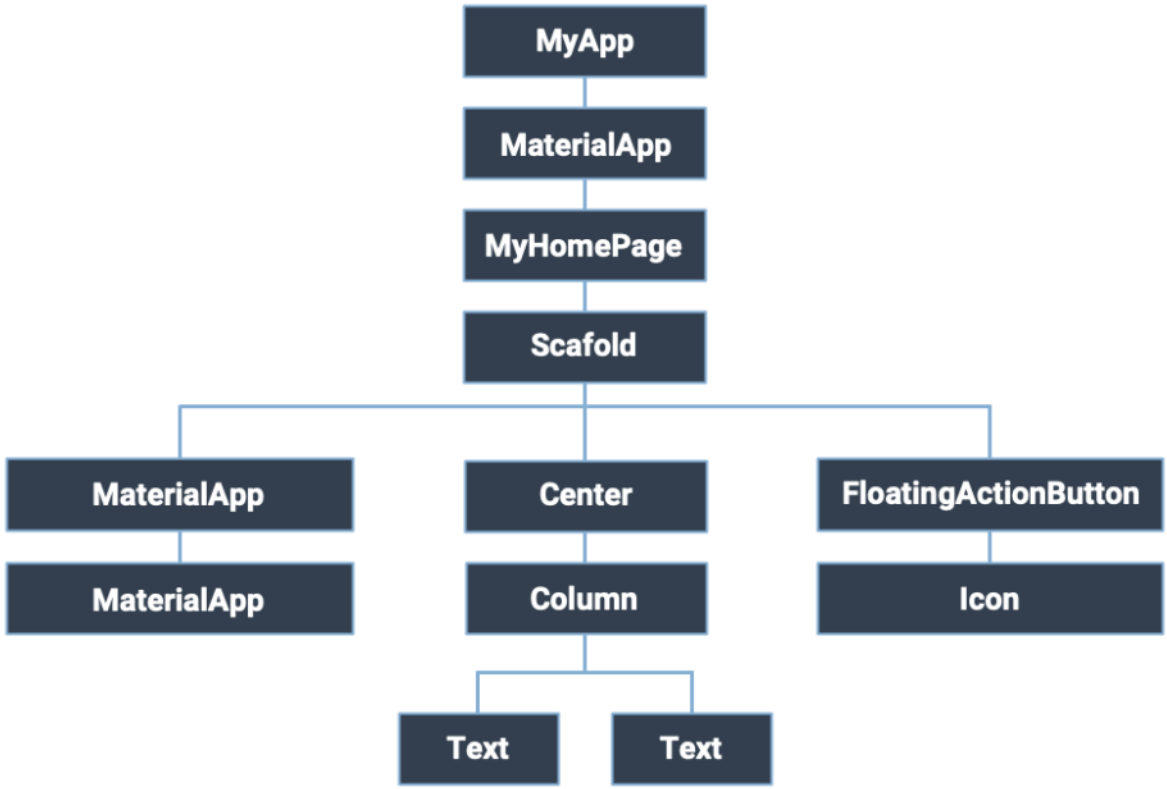
Étape 3 : pour lancer l'application, on sélectionne l'émulateur ou l'appareil, puis on clique sur le petit triangle vert à droite de `main.dart`.



L'arbre d'un widget

- Un widget est présenté sous la forme d'un arbre,
- Un widget peut contenir un ou plusieurs widgets enfants,
- Le widget enfant sera contenu par un autre widget.

On peut représenter l'application de base par le diagramme suivant :



Les widget « StateLessWidget »

Les « ***StatelessWidget*** » sont des widgets basiques sur lesquels on ne peut pas interagir comme un texte, une image, etc. Leur valeur ne va pas changer une fois construit.

Un widget se définit en appelant son constructeur à partir du mot clé `class`. Le widget container s'appelle avec son nom puis ouvrant une parenthèse et fermant la parenthèse : `Container()`.

Un widget est composé de plusieurs propriétés dont l'une s'appelle `child`. Avec cette dernière, on pourra définir le widget imbriqué à l'intérieur.

Exemple **Appel du widget Center**

```
Center(
 child: Text("Bonjour",
 textAlign: TextAlign.center,
 style: TextStyle(fontStyle: FontStyle.italic),),
),
```

On construit un widget Center qui a un fils : un widget Text. Ce dernier a deux propriétés : `textAlign` et `style`. Ce dernier appelle un widget `TextStyle`.

Méthode

On pourra aussi créer nos propres widgets. La création de widget va permettre de réutiliser notre code à différents endroits de notre application.

La définition d'un widget va commencer par le mot clé `class` en étant étendu avec un `stateLessWidget`. Le nom widget commencera par une majuscule : « *`BasicsPage`* ». Le widget est défini dans un bloc entre accolades. À l'intérieur de ce bloc, on a une fonction obligatoire : la fonction `build`. Pour la définir, il faut taper le mot clé `build`. Cette fonction va retourner le widget et donc un élément de l'interface utilisateur. Dans notre widget `MaterialApp`, on va mettre au niveau de la propriété `home`.

Exemple

```
class MyApp extends StatelessWidget {
 // This widget is the root of your application.
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Flutter Demo',
     theme: ThemeData(
       primarySwatch: Colors.blue,
     ),
     home: BasicsPage()  // appel du widget créé dessous
   );
 }
}

class BasicsPage extends StatelessWidget{
 @override
 Widget build(BuildContext context) {
  return
 }
}
```

La fonction `build` va construire le widget. Le constructeur prend en paramètres un argument de type « *`BuildContext`* ». Il décrit l'interface de ce widget, soit le contexte dans lequel ce widget est créé : sa hauteur, sa largeur, la plateforme, etc.

Grâce à ce contexte, on pourra faire appel à MediaQuery pour en définir les propriétés, comme la taille. On peut aussi récupérer la plateforme (iOS ou Android).

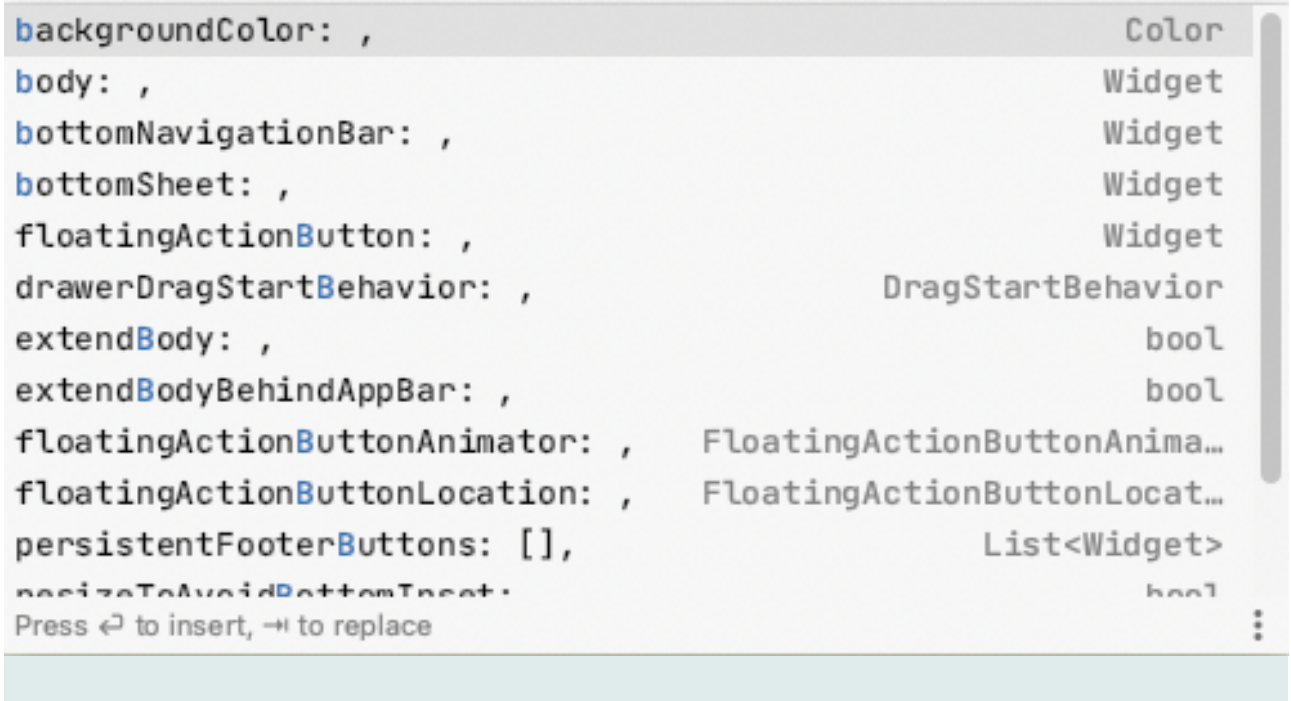
Dans notre constructeur, on retournera un widget `Container()`.

```
@override
Widget build(BuildContext context) {
 var size = MediaQuery.of(context).size;
 var platform = Theme.of(context).platform;
 print("size : $size");
 print("plateform : $platform");
return Container(
  height: 10,
    width: 10,
);
}
```

Définition **Scaffold**

Le `scaffold` est le squelette de l'application, il est présent dans toutes les applications. Il comprend les widgets suivants avec les propriétés :

- **appBar :** la barre du haut
- **body :** le corps de notre application
- **floatingButtonAction :** un bouton en bas
- **drawer :** un menu sur le côté



Définition **AppBar**

L'`appBar` est un widget qui vient se situer en haut de l'écran. En général, il contient le titre de l'application. Il peut aussi accueillir des boutons.

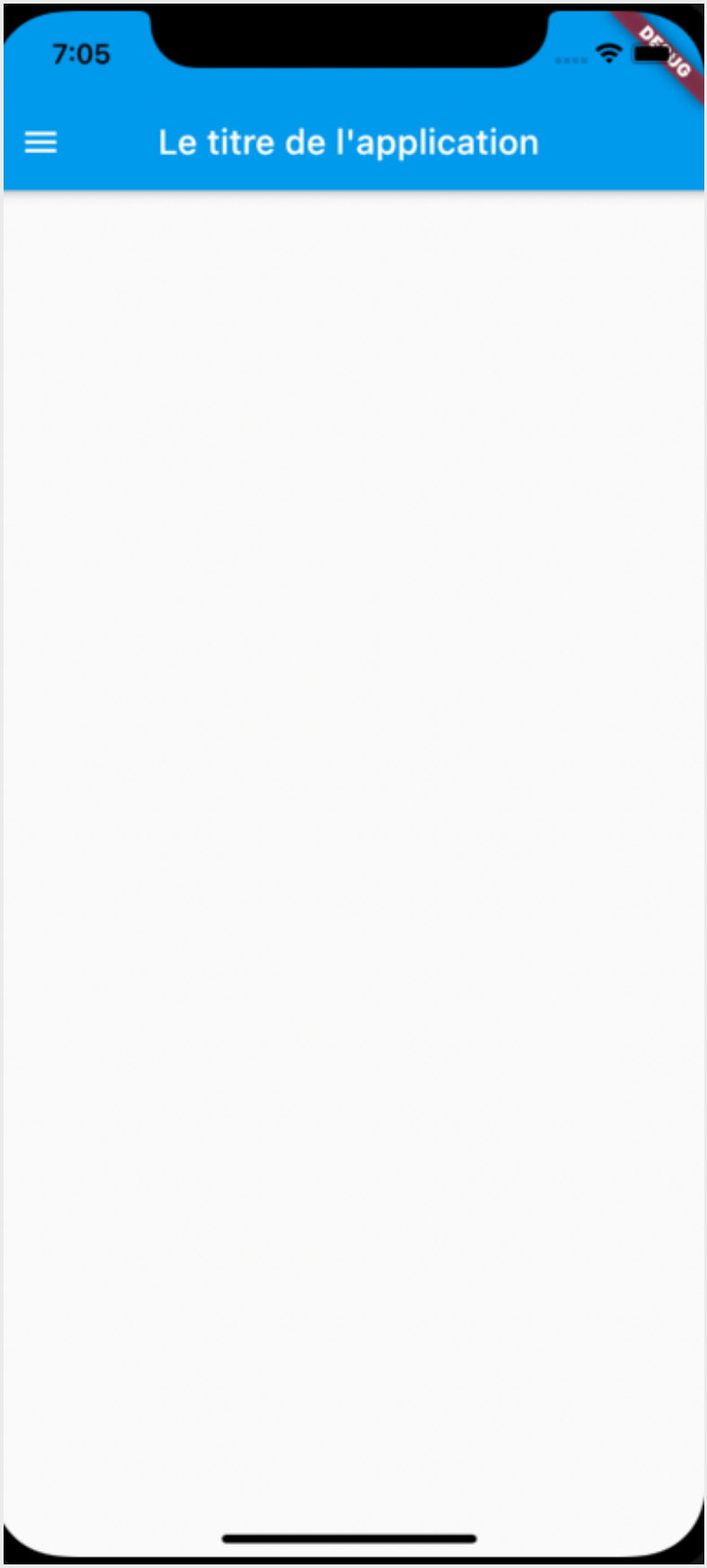
```
return  Scaffold(
  appBar: AppBar(
    title: Text("Le titre de l'application"),
    actions:<Widget> [
      IconButton(onPressed: pressedAction, icon:Icon(Icons.nat))
    ], ),
);
```

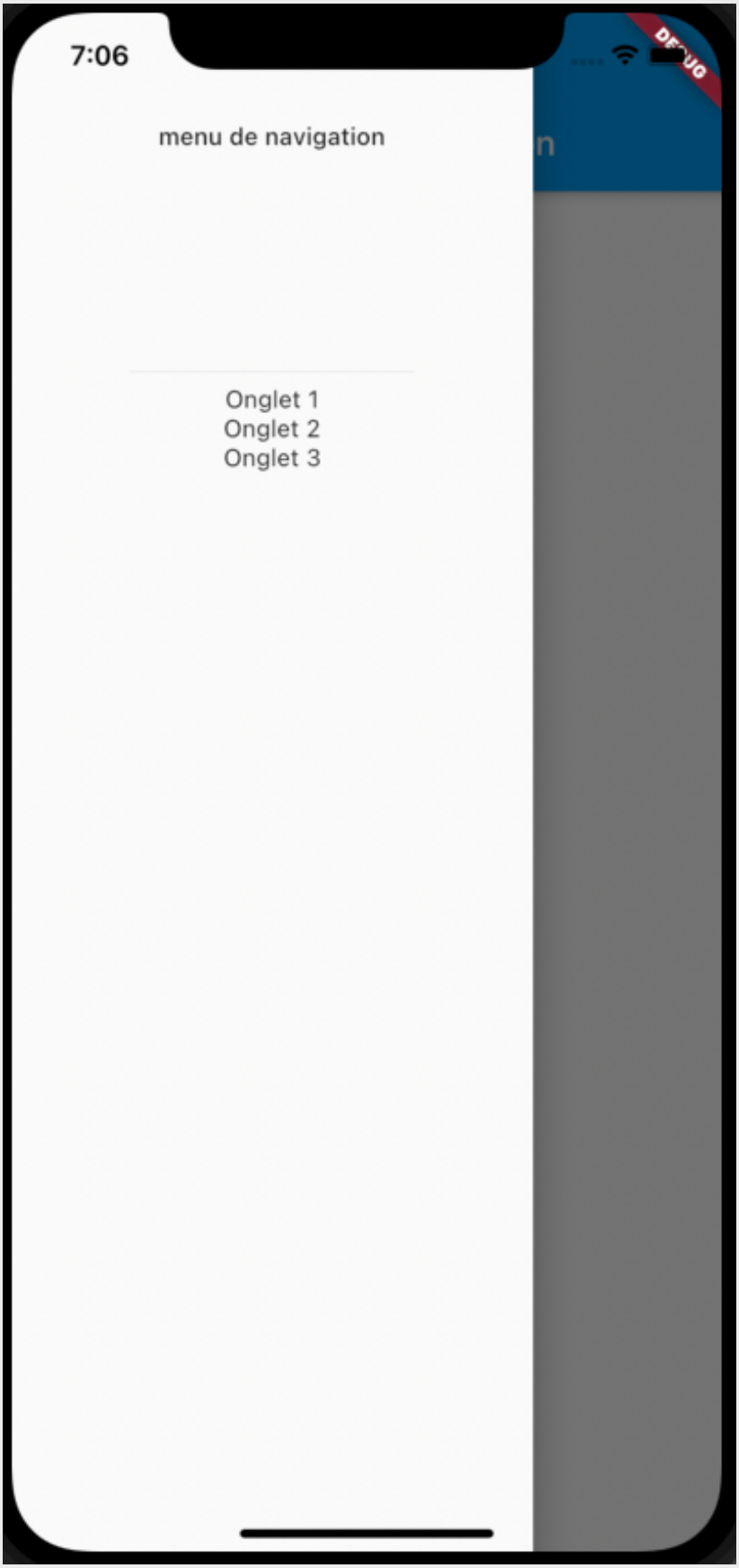
Définition **Le drawer**

Le `drawer` est un menu caché qui glisse horizontalement, il est déplié avec l'icône « *Hamburger* », 3 traits. Pour construire le `drawer`, le widget `Drawer` est appelé, il a comme enfant un widget `Column` que nous verrons plus tard dans le cours qui contiendra l'entête du menu et la liste des choix avec des widgets `Text`.

```
drawer: Drawer(
 child:Column(
   children: [
     DrawerHeader(child: Text('menu de navigation')
     ),
     Text("Onglet 1"),
     Text("Onglet 2"),
     Text("Onglet 3"),],
 )
),
```

Exemple



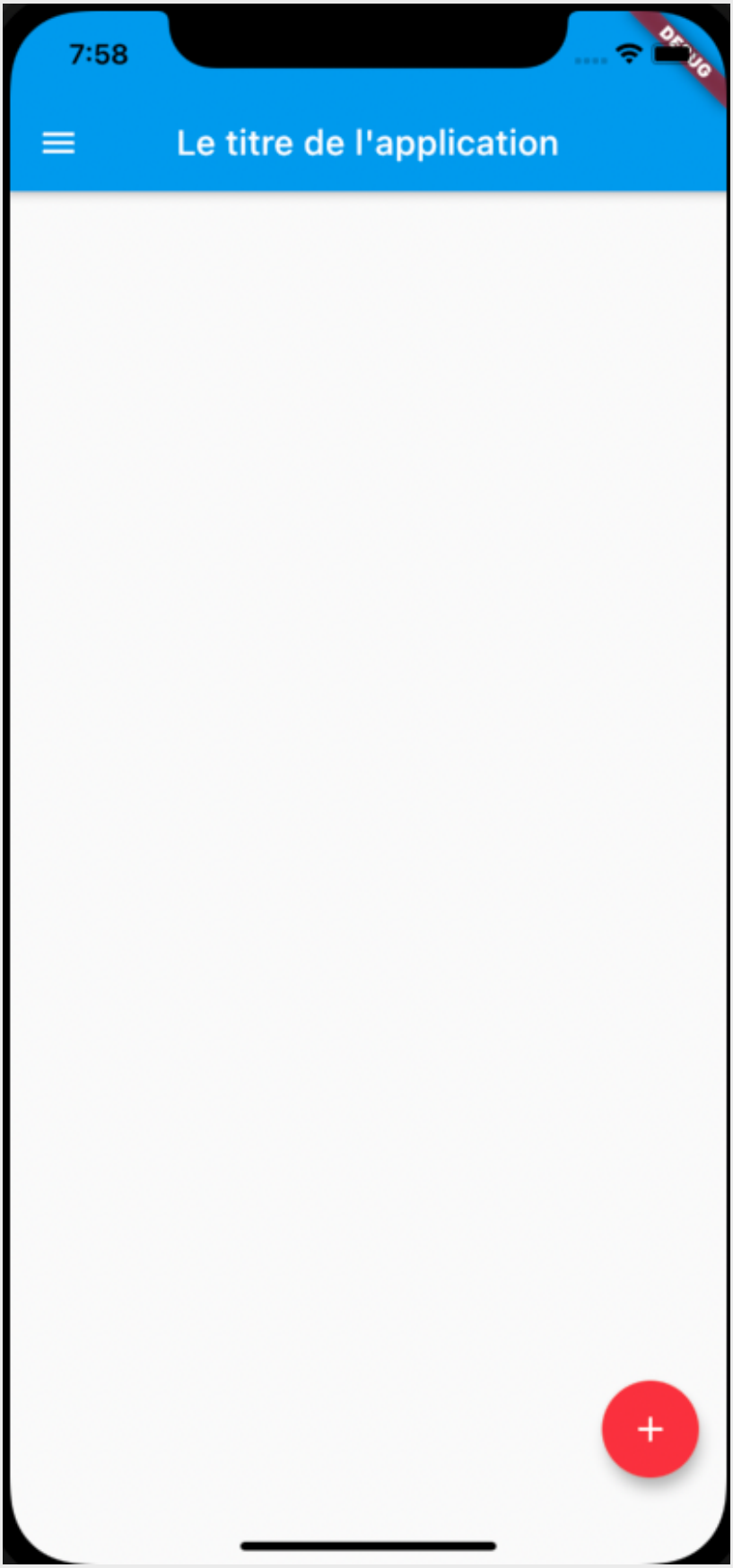


Définition **Le FloatingActionButton**

Le `floatingActionButton` est un widget qui est un bouton flottant en bas à gauche, on peut le retrouver dans certaines applications de messagerie. En appuyant dessus, on crée un nouveau message, c'est ce bouton qui va exécuter une méthode.

```
),
floatingActionButton: FloatingActionButton(
   onPressed: pressedAction,
 backgroundColor: Colors.red,
   child: Icon(Icons.add),
),
```

Exemple



Fondamental **Récapitulatif du Scaffold**

```
@override
Widget build(BuildContext context) {
return  Scaffold(
  appBar: AppBar(
    title: Text("Le titre de l'application"),
    actions:<Widget> [
      IconButton(onPressed: pressedAction, icon:Icon(Icons.photo))
    ], ),

  drawer: Drawer(
    child:Column(
      children: [
        DrawerHeader(child: Text('menu de navigation')
        ),
        Text("Onglet 1"),
        Text("Onglet 2"),
        Text("Onglet 3"),
      ],
    )
  ),
  floatingActionButton: FloatingActionButton(
    onPressed: pressedAction,
    backgroundColor: Colors.red,
    child: Icon(Icons.add),
  ),
);
}
```

Définition **Le widget Center et container**

Le widget Center a pour rôle de centrer son enfant (`child`). En mettant le widget container en tant qu'enfant de center, il sera au centre de notre écran.

```
body:Center(
   child:Container(
     height: 80,
     width: 100,
     color: Colors.red,
   )
)
```

Ici, on centre le `container` dans le `body` du `Scaffold`. Les propriétés `height` et `width` permettent de définir la taille du `container`.

Exercice : Quiz

[solution n°1 p.43]

Question 1

Question 2

Question 3

Question 4

Question 5

III. Widgets de base

Widget Text

Le widget `Text` est un élément très important, il est présent dans toutes les applications. Son constructeur demande un paramètre obligatoire : une chaîne de caractères. On peut travailler sur ce texte en modifiant son style pour changer son aspect, son alignement, sa couleur.

Exemple

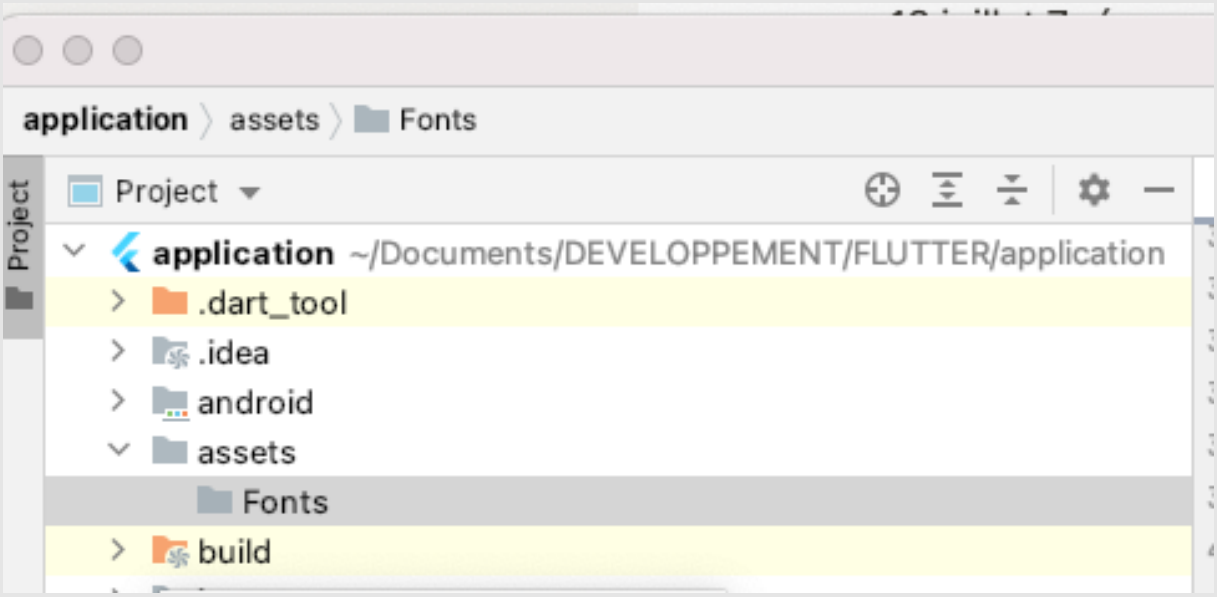
```
Text("Ceci est une première chaine",
 style: TextStyle(
   color: Colors.red,
   fontSize:20,
   fontWeight: FontWeight.w800,
   fontStyle:FontStyle.italic
 ),
 textAlign: TextAlign.center,
)
```

Ici, on définit un widget `Text` qui prend en argument la chaîne de caractères : « *Ceci est une chaîne de caractères* ». On lui spécifie un `Style` avec le widget `TextStyle`. À l'intérieur de ce style, on va définir la couleur du texte et l'aspect de la police de caractères. On peut aussi spécifier l'alignement de cette zone de texte. Ici, on centre le texte.

Si on veut pousser la personnalisation de l'application, on peut mettre en place une police de caractères spécifique dans le constructeur de `TextStyle` avec la propriété `fontFamily`.

Méthode

Étape 1 : dans l'arborescence de notre projet, il faut créer un répertoire « *assets* ». On fait un clic droit - new - Directory. À l'intérieur de ce dossier, on crée un dossier « *fonts* » pour stocker nos polices dans notre application. On va « *glisser-déposer* » la police de caractères dans le répertoire.



Complément

Sur Internet, on peut trouver des polices de caractères gratuites sur le site Google Fonts[1].

Méthode

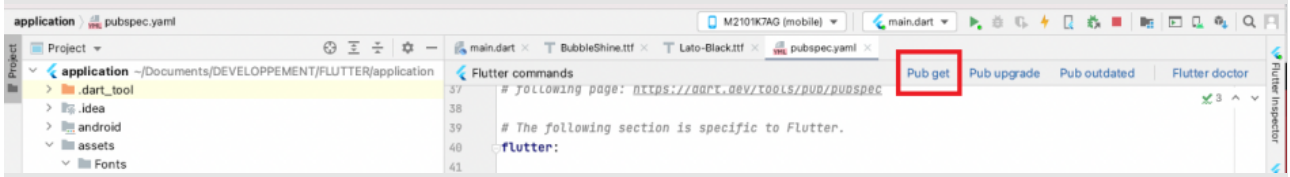
Étape 2 : maintenant, il convient de modifier le fichier pubspec.yaml et de l'enrichir d'une rubrique « *fonts* ».

```
flutter:

  uses-material-design: true

  fonts:
    - family: BubbleShine
      fonts:
        - asset: assets/Fonts/BubbleShine.ttf
```

Dans Android Studio, cliquez sur Pub get pour finir l'installation de la police de caractères.

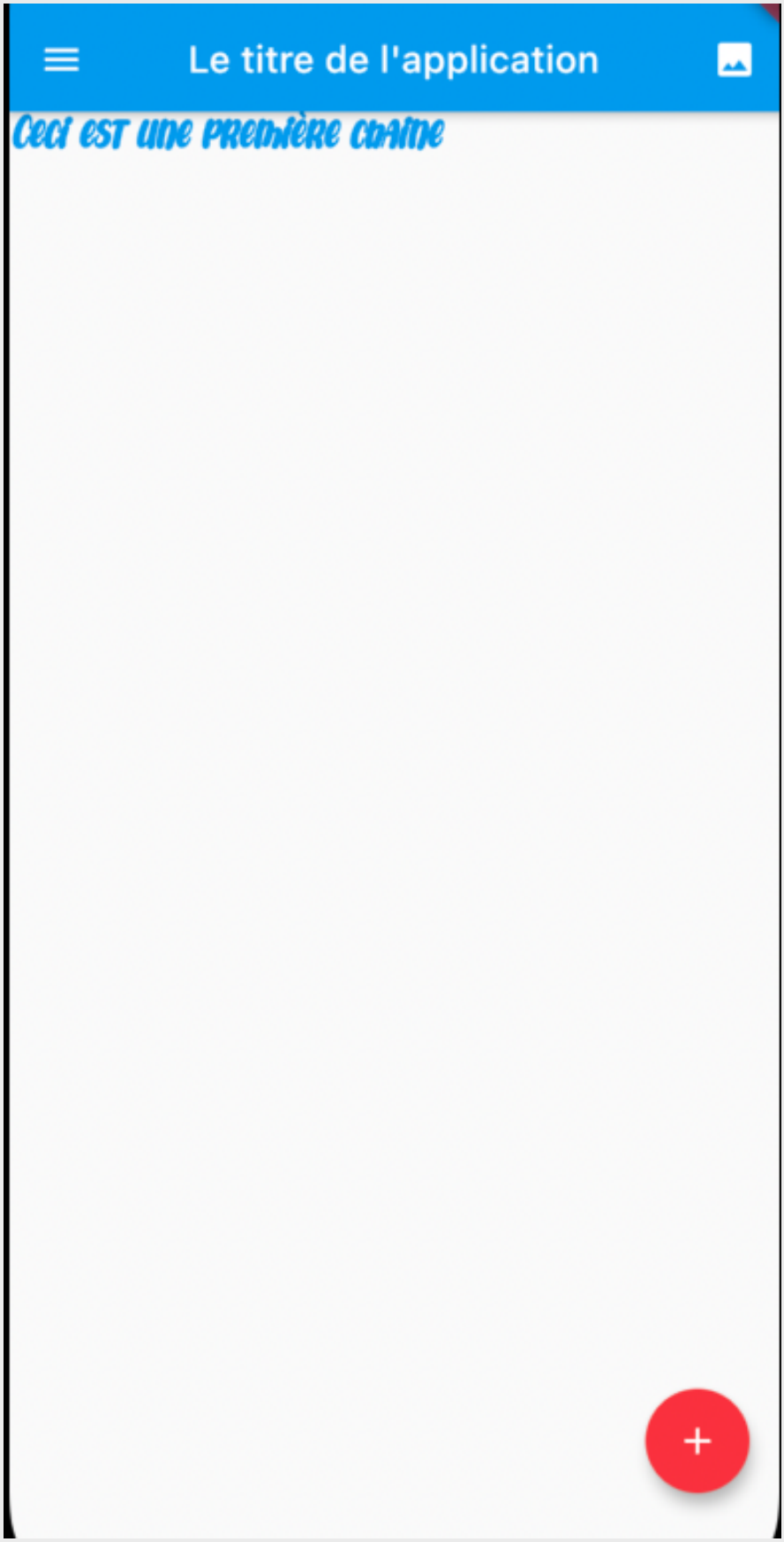


Étape 3 : une fois installée, on peut définir la font dans la propriété `font-family` dans le widget `TextStyle`.

Exemple

```
style: TextStyle(
 color: Colors.blue,
 fontSize:25,
 fontWeight: FontWeight.w800,
 fontStyle:FontStyle.italic,
 fontFamily:'BubbleShine'
```

1 https://fonts.google.com/



Méthode

Voici comment créer un widget `Text` configuré avec une police de caractères, un style et une couleur :

```
Text simpleText(String text){
 return Text(
   text,
   style: TextStyle(
       color: Colors.blue,
       fontSize:25,
       fontWeight: FontWeight.w800,
       fontStyle:FontStyle.italic,
       fontFamily:'BubbleShine'
   ),
   textAlign: TextAlign.center
 );
```

Complément

Pour créer un widget avec cette configuration, on aura juste à l'appeler :

`simpleText("On peut réutiliser notre texte")`

Cela évite que l'on ait à chaque fois à reconfigurer notre widget `Text` de base.

Définition **Widget Icon**

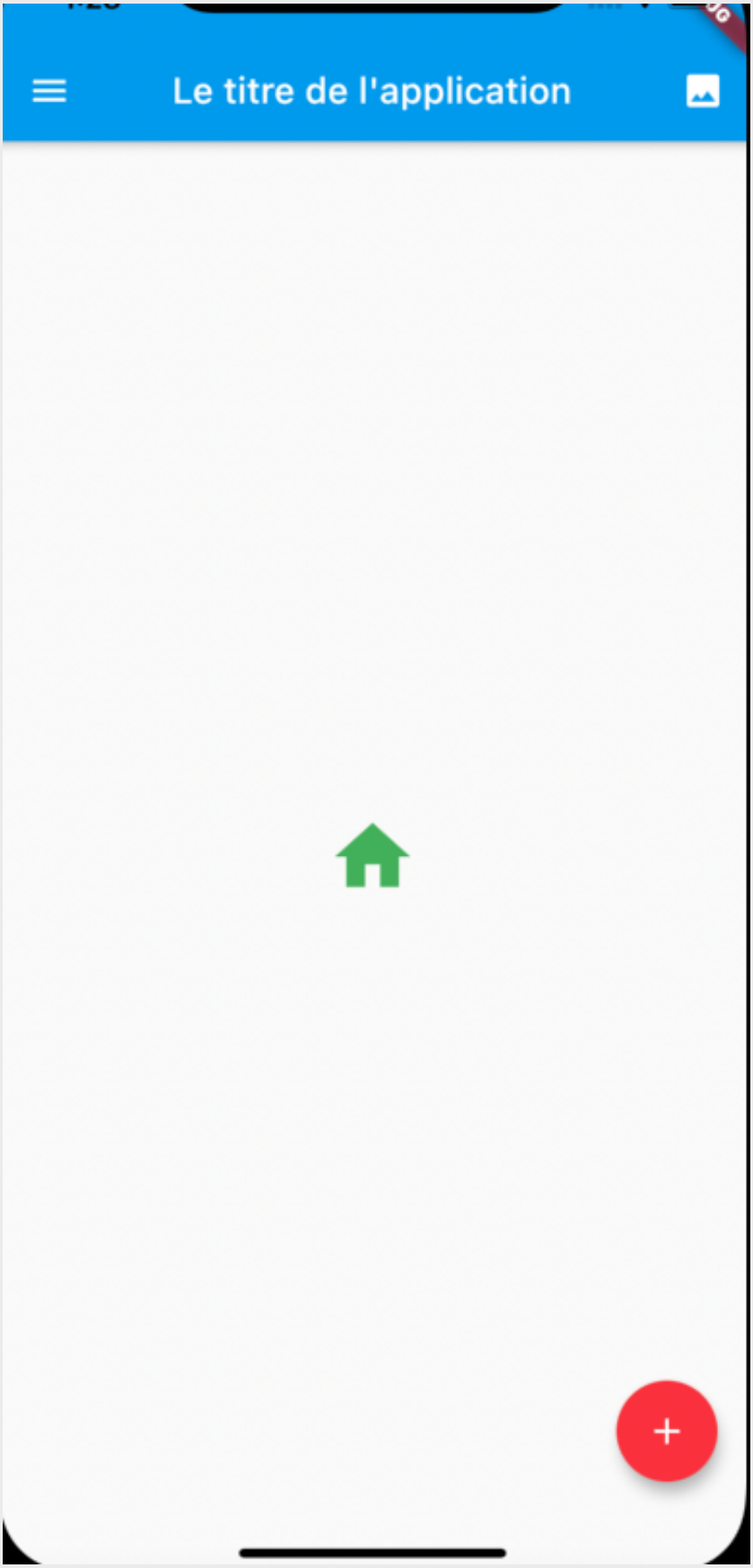
De nos jours, les icônes sont aussi des éléments essentiels de nos applications. Leur utilisation permet une compréhension rapide de l'interface utilisateur. Par exemple, tout le monde associe une icône « *pin* » à la cartographie.

L'utilisation du widget `Icon` permet l'insertion d'une icône. On pourra spécifier certaines de ses propriétés comme sa taille.

Exemple

```
body:Center(
child: Container(
  height: 150,
  width: 150,
  child: Icon(Icons.home,
  color:Colors.green,
  size: 50,)
),
)
```

Dans le container, on met un `Icon` Home en tant qu'enfant. On lui applique une taille de 50 avec `size` et on lui applique la couleur verte.



En important : `import 'package:flutter/cupertino.dart';`

On pourra mettre des styles d'icônes qui se rapprochent de ceux utilisés sur iOS.

```
Icon(CupertinoIcons.add,
color:Colors.green,
size: 50,)
```

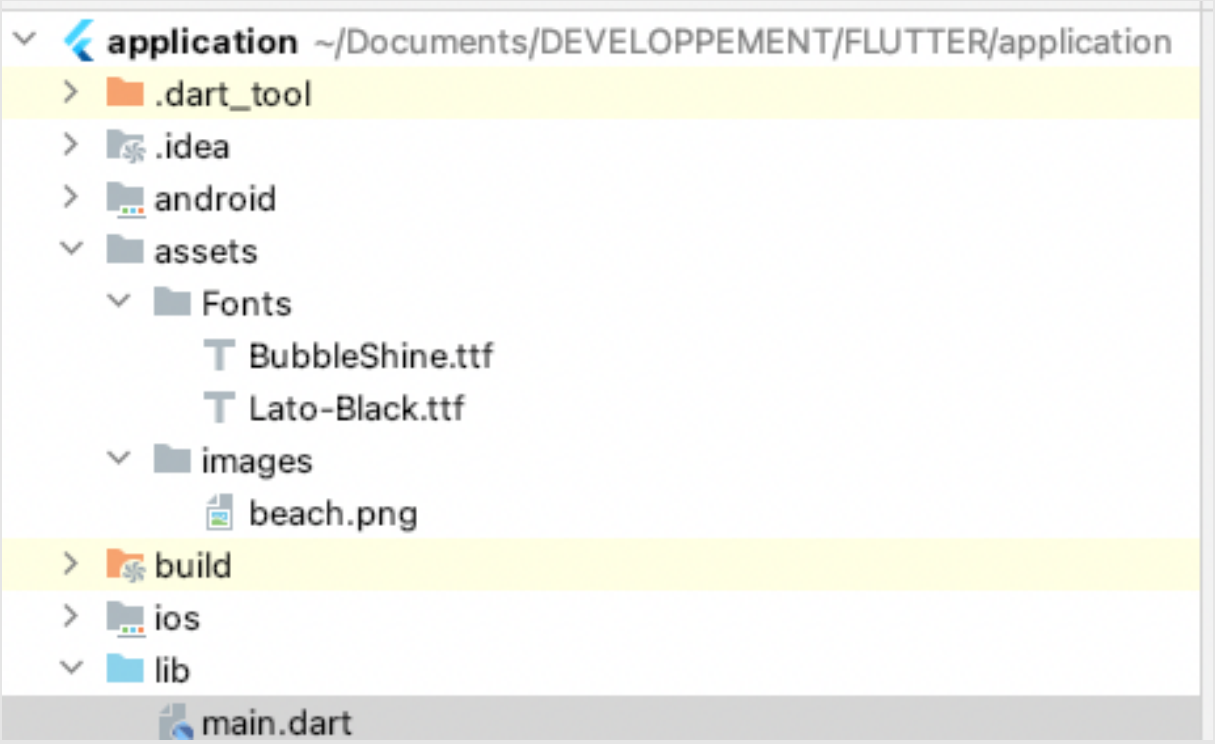
Les icônes pourront être ajoutées à l'endroit souhaité pour personnaliser le `floatingActionButton` ou la barre d'onglets.

Définition **Widget Image**

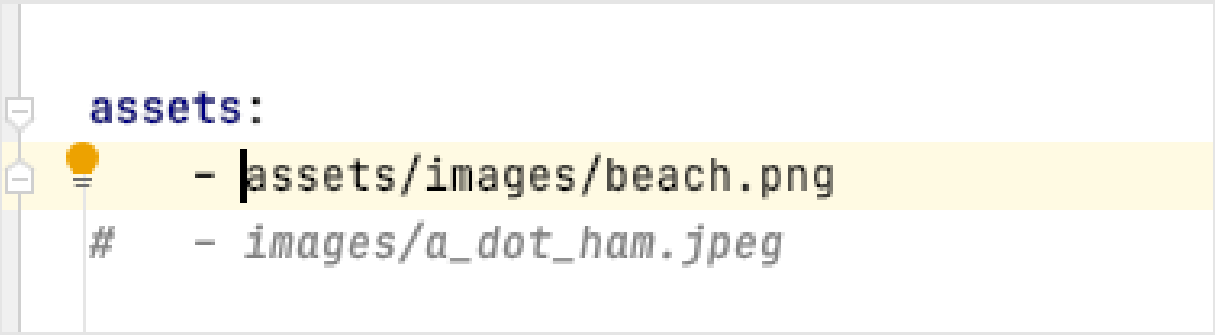
Afin d'enjoliver une application, il est fréquent d'avoir recours à des images. Flutter a un widget dédié `Image`.

Méthode

De la même manière que pour les polices de caractères, il convient de créer un dossier images dans le répertoire assets. Ensuite, on dépose les images dans ce dossier. On va aussi ajouter le chemin des images dans le fichier pubspec.yaml.



On modifie le fichier puspec.yaml en indiquant où se trouvent les fichiers images.



Attention

Il s'agit d'un fichier yaml. Il faut faire attention à l'indentation dont l'importance est cruciale. Par exemple `assets` doit être à deux espaces de la gauche.

Méthode

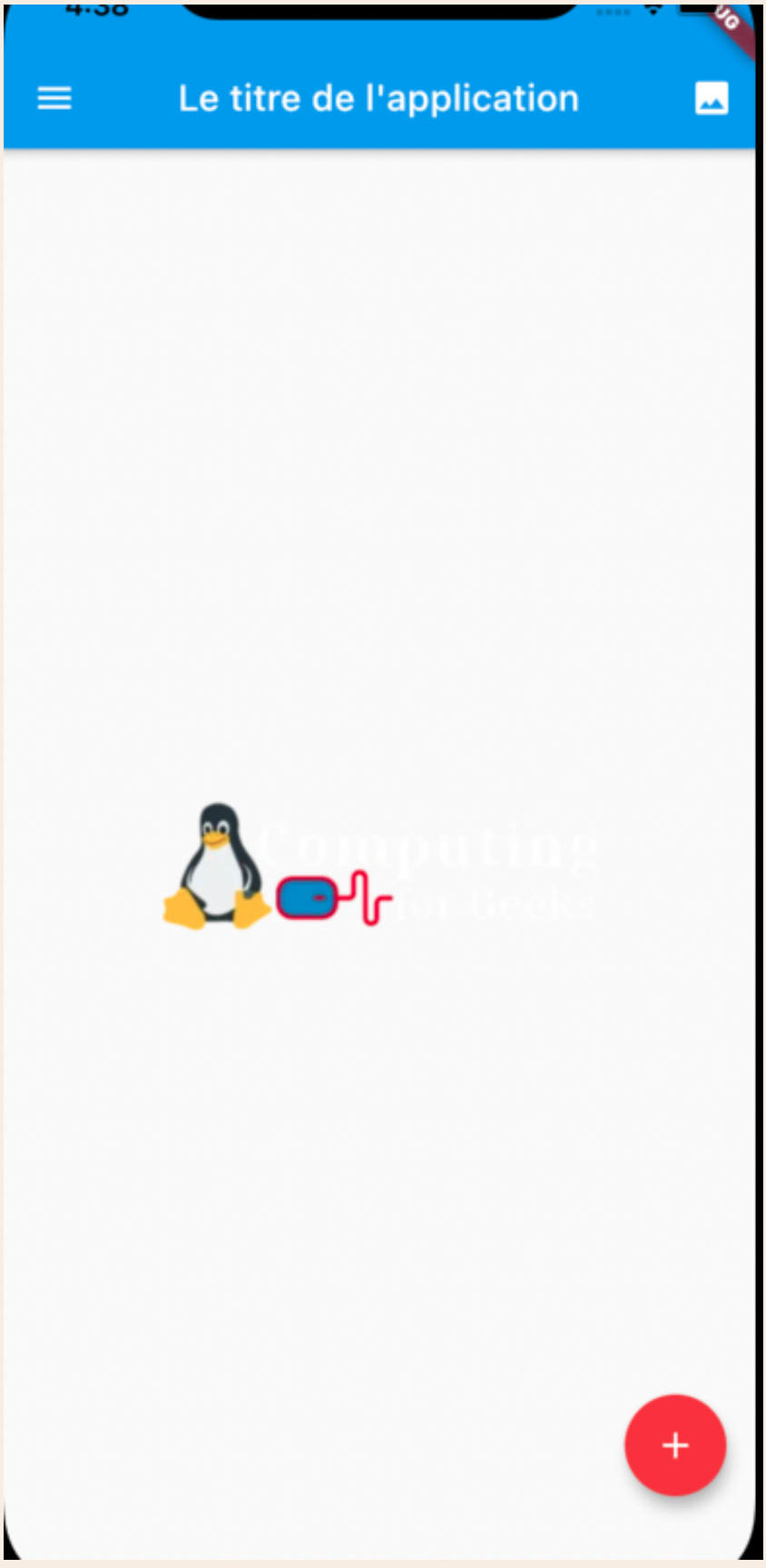
Une fois le fichier à jour, il faut cliquer sur le bouton `pub get`. Maintenant que l'image est déclarée, il ne reste plus qu'à l'incorporer dans le projet.

Exemple

```
1 Image.asset('assets/images/beach.png',
2  width: 150,
3 )
```

Complément

Si on peut insérer des images en dur dans le projet, on peut aussi les afficher en les chargeant depuis internet. La syntaxe est `Image.network('http://www.....')`.



```
Image.network('https://computingforgeeks.com/wp-content/uploads/2019/05/cf_white-1.png?ezimgfmt=rs:272x90/rscb23/ng:webp/ngcb23')
```

Définition **Widget Card**

Le widget `Card` est utilisé pour créer une boîte rectangulaire avec un `container` comme enfant. On a l'impression que la `Card` est surélevée (une ombre portée). Cet effet peut être accentué avec la propriété `elevation`.

Exemple

```
body: Center(
 child: Card(
   child: Container(
     color: Colors.grey,
     height: 150,
     width: 300,
     child: simpleText("Ma premiere Card"),
   ),
   elevation: 10,
 ),
),
```

Ici, on a défini un widget `Card` à l'intérieur du widget `Center`. À l'intérieur, il contient un widget `Text`.

Définition **Widget Padding**

Le widget `Padding` permet de positionner son enfant à l'intérieur d'un `container` ou d'une `card`.

Exemple

```
child: Card(
 child:
 Container(
   color: Colors.grey,
   height: 150,
   width: 300,
   child: Padding(
     padding: EdgeInsets.fromLTRB(10, 10,0,0),
     child: simpleText("Ma premiere Card"),
   )
 ),
 elevation: 7.5,
),
```

Dans cet exemple, `Padding` permet positionner le widget `simpleText` à 10 par rapport au Haut et 10 par rapport à gauche.

Widget Column

Le widget `Column` permet d'aligner verticalement des widgets. On peut le voir comme une pile d'éléments. Dans d'autres langages, on parlera de `stackview`. Ce widget possède des enfants : `children`. Ils s'espacent dans la `column` selon la propriété `mainAxisAlignement`.

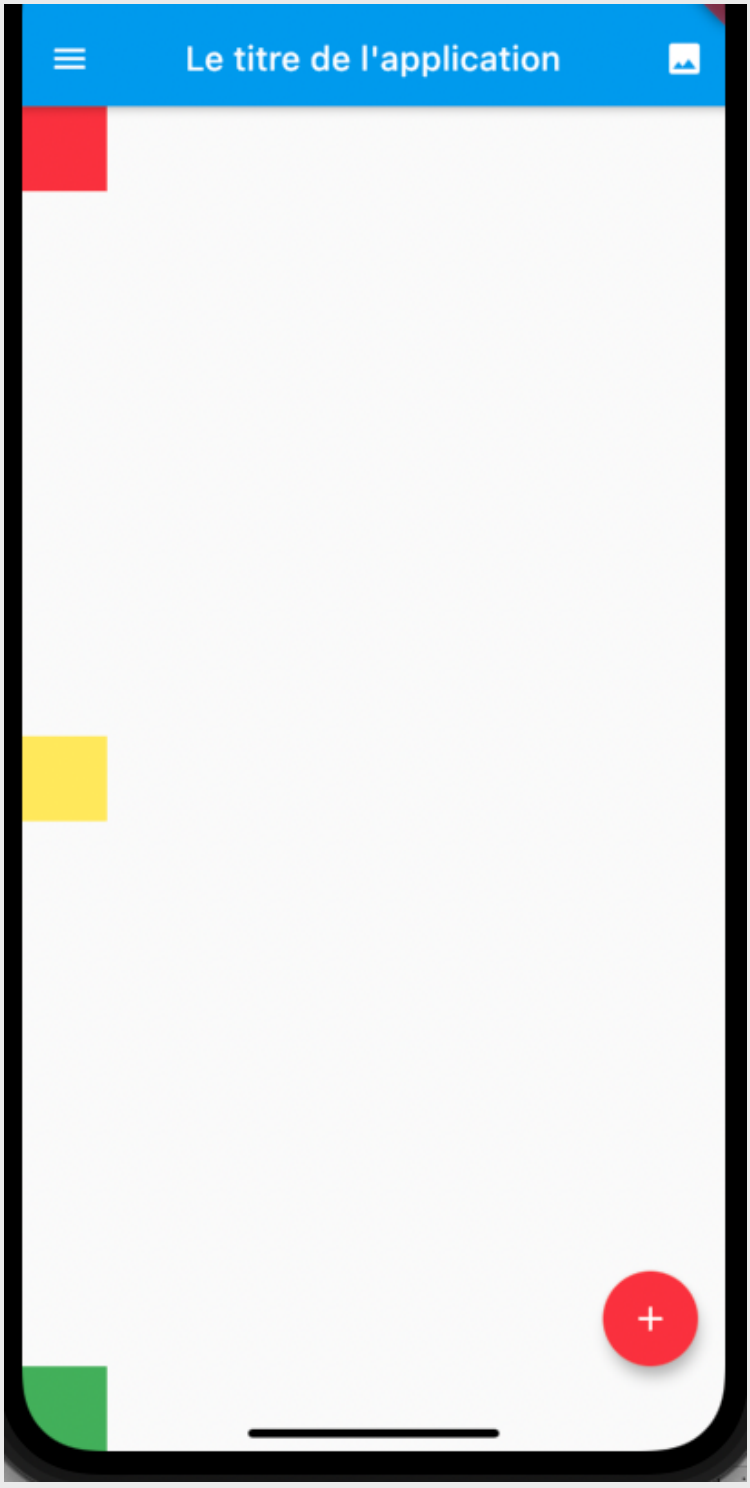
Cette propriété peut prendre différentes valeurs :

- **`Start`** : les éléments sont groupés en haut de colonne.
- **`End`** : les éléments sont groupés en bas de la colonne.
- **`Center`** : les éléments sont groupés au centre de la colonne.
- **`SpaceEvenly`** : les éléments sont séparés les uns des autres avec le même espacement.
- **`SpaceBetween`** : les éléments sont séparés les uns des autres avec le même espacement en étant collés aux extrémités.
- **`SpaceAround`** : les éléments sont séparés les uns des autres avec le même espacement avec des marges réduites aux extrémités.

Exemple **La propriété spaceBetween**

```
Column(
mainAxisAlignment: MainAxisAlignment.spaceBetween,
children: [
  Container(
    height: 50,
    color: Colors.red,
    width: 50,
  ),
  Container(
    height: 50,
```

```
    color: Colors.yellow,
    width: 50,
  ),
  Container(
    height: 50,
    width: 50,
    color: Colors.green
  )
],)
```

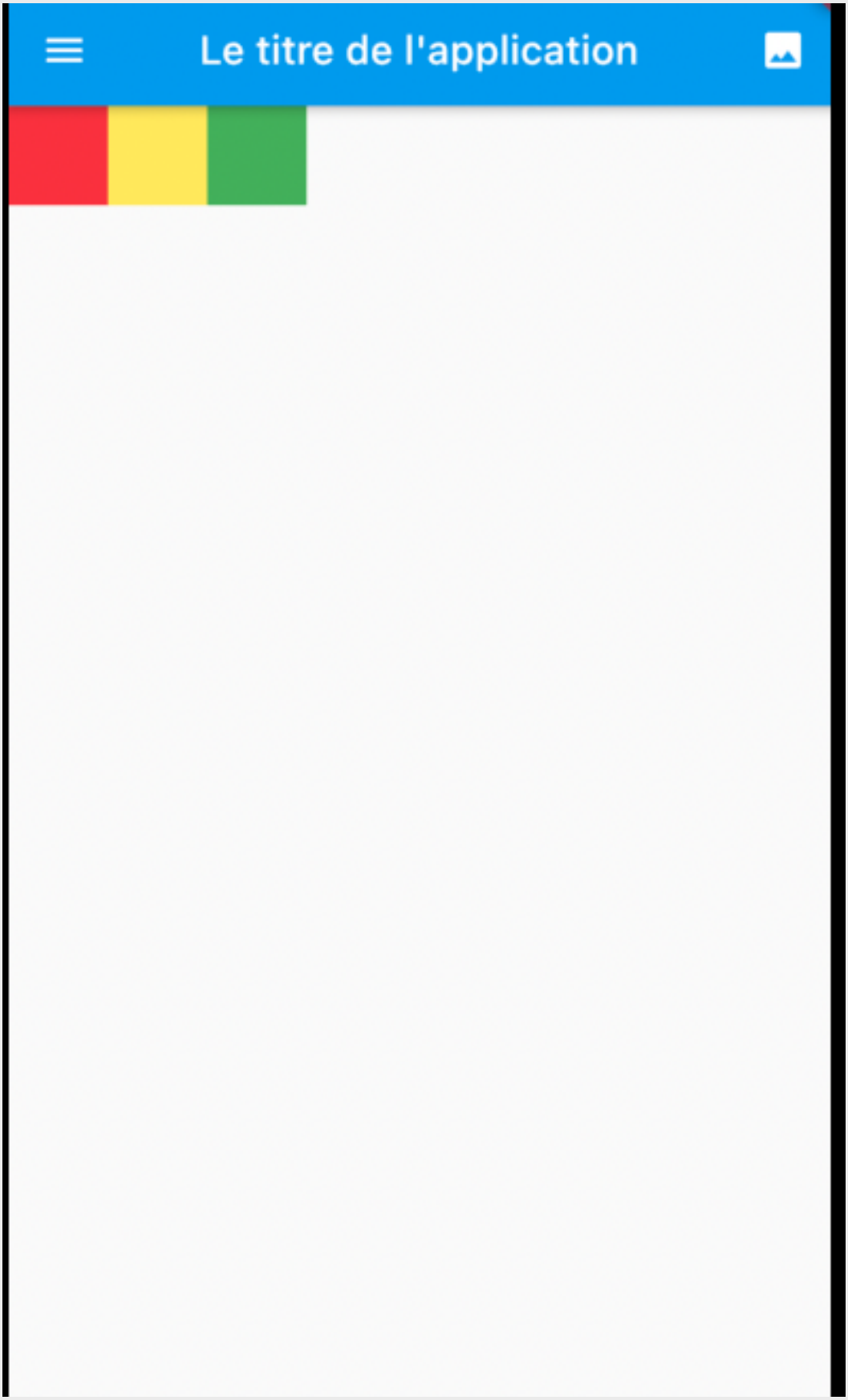


Définition **Widget Row**

Le widget `Row` a le même fonctionnement que `Column`. `Row` est un empilement de widgets enfants (`children`).

Exemple **MainAxisAlignement.Start**

```
Row(
 mainAxisAlignment: MainAxisAlignment.start,
 children: [
   Container(
     height: 50,
     color: Colors.red,
     width: 50,
   ),
   Container(
     height: 50,
     color: Colors.yellow,
     width: 50,
   ),
   Container(
     height: 50,
     width: 50,
     color: Colors.green
   )
 ],
```



La propriété `mainAxisAlignement` est aussi présente. Elle se comporte similairement, mais horizontalement.

Définition **Widget Expanded**

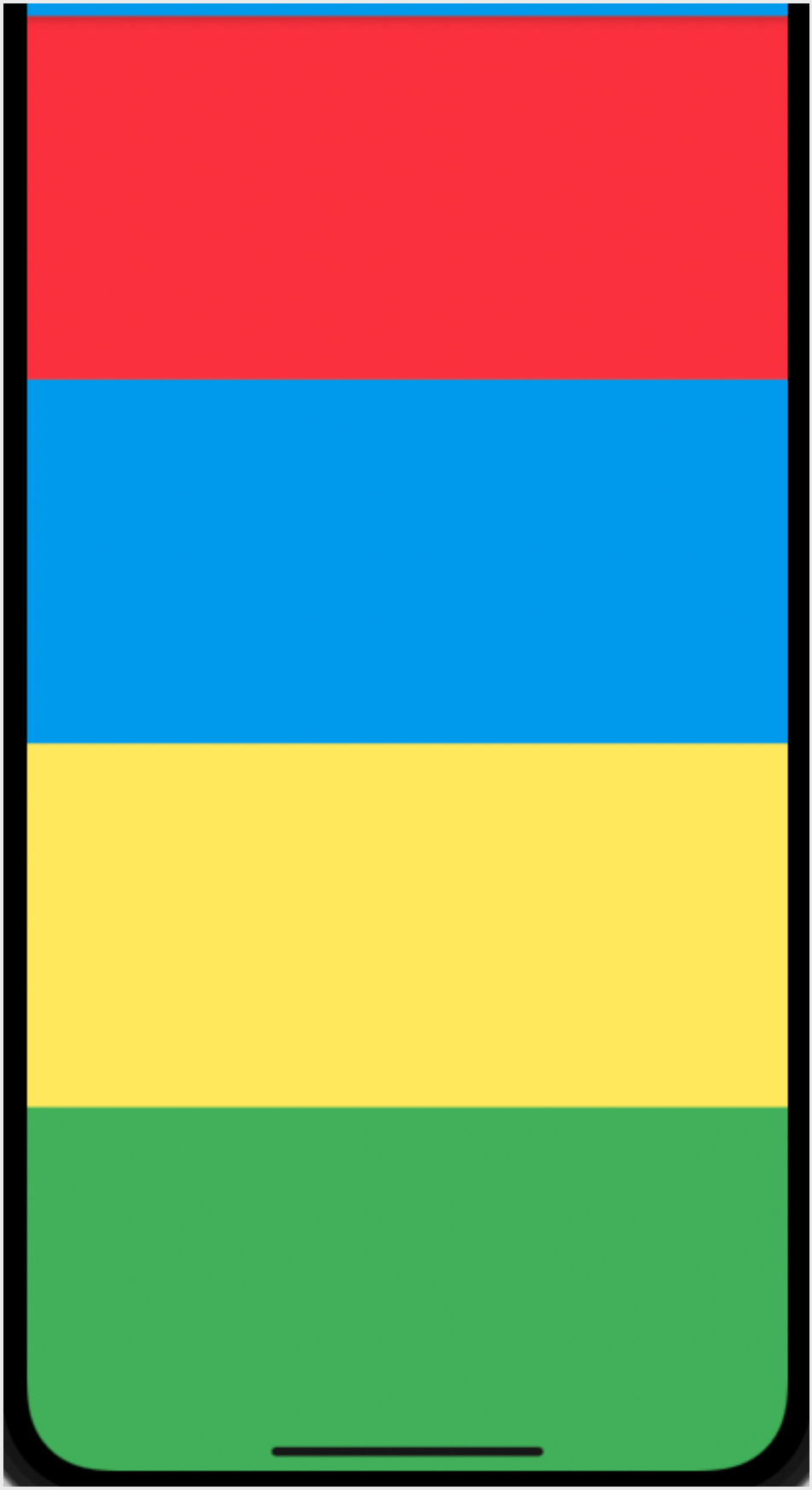
En utilisant, le widget `Expanded`, dans une `Column` ou `Row`, son enfant occupera tout l'espace libre.

La propriété `flex` à l'intérieur du widget permet d'établir des proportions.

Exemple

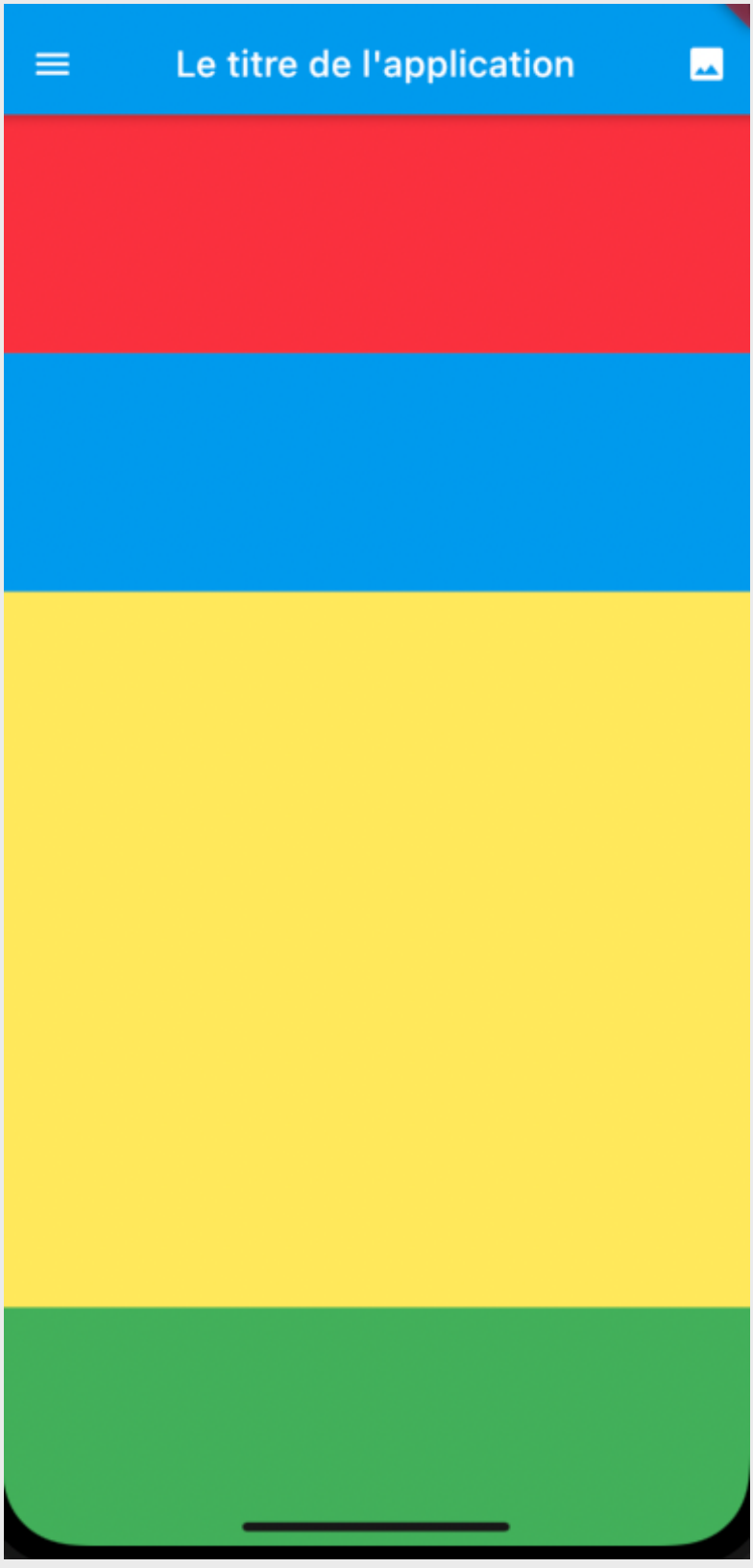
Commençons par définir une `column` à l'intérieur du `body`. À l'intérieur de `column`, nous allons empiler 4 widgets `expanded` qui ont pour enfant un widget `container`. On définit une couleur pour chaque `container`. Par défaut, la propriété `flex` est égale à 1.

```
Body(
Column(
 children: [
   Expanded(
     child: Container(
       color: Colors.red,
     ),
   ),
   Expanded(
     child: Container(
       color: Colors.blue,
     ),
   ),
   Expanded(
     child: Container(
       color: Colors.yellow,
     ),
   ),
   Expanded(
     child: Container(
       color: Colors.green,
    ),
   ),
 ],
)
```



On met une propriété `flex` à 3 au widget `expanded` contenant le `container` jaune.

```dart
Expanded(
 flex: 3,
 child: Container(
   color: Colors.yellow,
 ),
),
```



Si on fait la somme des propriétés sachant que par défaut elle est à 1 :

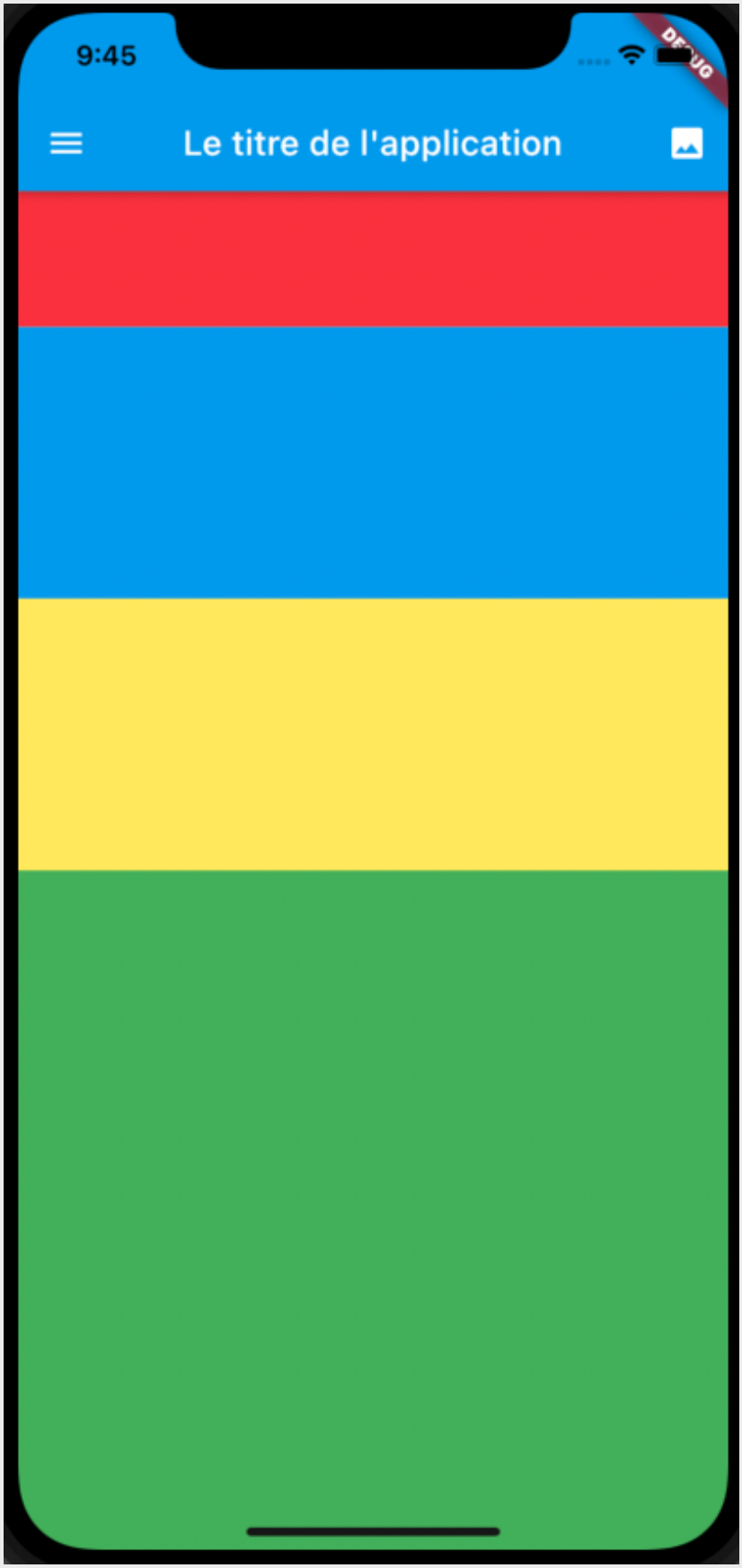
1 (`container` rouge) + 1 (`container` bleu) + 3 (`container` jaune) + 1 (`container` vert) = 6

La proportion du `container` jaune sera de 3/6 de l'écran. On voit que cela correspond bien à la moitié de l'écran.

Pour se simplifier dans les calculs, on prendra 10 pour la somme des `flex`. Ensuite, on répartit la somme des propriétés `flex` sur chaque widget `Expanded`.

Adoptons la répartition suivante :

- Rouge : `flex` = 1 soit 1/10
- Bleu : `flex` = 2 soit 2/10
- Jaune : `flex` = 2 soit 2/10
- Vert : `flex` = 5 soit 5/10



Exercice : Quiz

[solution n°2 p.43]

Question 1

Question 2

Question 3

Question 4

Question 5

V. Essentiel

Au fil de ce cours, nous avons vu la première partie des bases de Flutter. La philosophie de Flutter consiste à combiner des widgets : tout est widget. Un widget peut contenir un ou plusieurs widgets. On a abordé les widgets `StateLess`. Un widget « *`stateLess`* » ne change pas d'état au cours de sa vie.

Dans un premier temps, retenez le squelette d'une application Flutter :

- **Appbar :** la barre de titre de l'application avec la possibilité d'ajout de boutons.
- **Hamburger-menu :** drawer permettant la navigation à l'intérieur de l'application.
- **FloatingButtonAction :** un bouton en bas à gauche.
- **Body :** la racine de l'application qui peut contenir un widget `Container`.

Dans un second temps, quelques widgets de base de Flutter :

- Widget `Text` : composant qui permet d'afficher du texte dans nos applications avec la possibilité d'ajout de polices de caractères personnalisées.
- Widget `Icon` : composant affichant une icône.
- Widget `Row` : composant qui permet un alignement horizontal des widgets.
- Widget `Column` : composant qui permet un alignement vertical des widgets.
- Widget `Expanded` : qui permet d'étendre un widget à l'espace libre dans une `Column` ou `Row`.

VI. Auto-évaluation

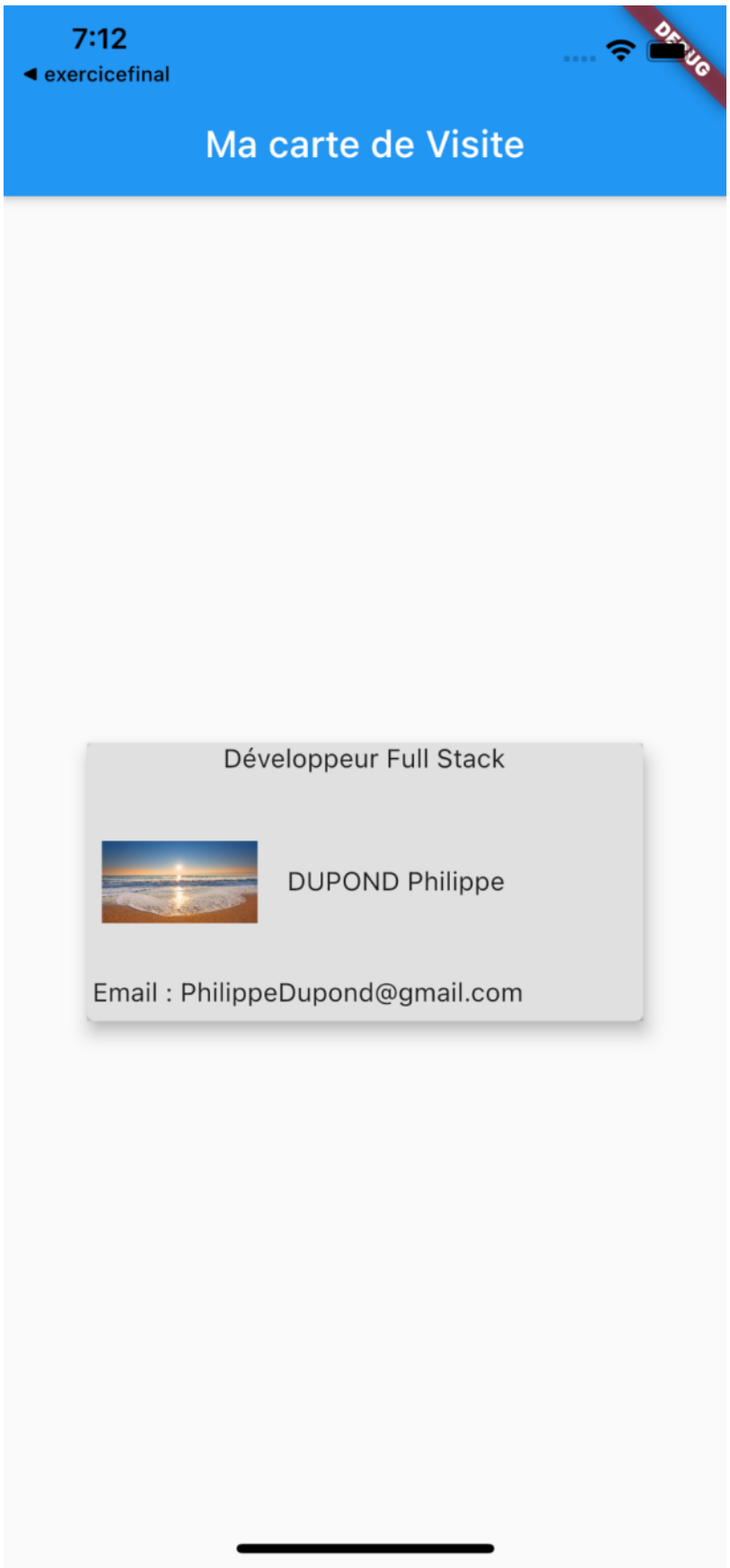
A. Exercice

Dans cet exercice, nous allons mettre en pratique les différents widgets étudiés dans ce cours. Le but est de construire une carte de visite minimaliste avec pour forme un rectangle. La carte sera composée d'une image et de textes. Les textes seront :

- L'intitulé de la formation
- Le nom de la personne
- L'adresse mail de la personne

Il conviendra d'utiliser les widgets `Row` et `Column` pour positionner les éléments.

Il faudra intégrer une image au projet.



Question 1 [solution n°3 p.43]

Créez un projet Flutter, Flutter App. Faites le ménage dans le `main.dart` de l'application. Créez un widget `CarteDeVisite` qui viendra au niveau du home dans le widget `MaterialApp`.

Question 2 [solution n°4 p.44]

Mettez en place une image dans le projet.

Question 3 [solution n°5 p.44]

Insérez le squelette de l'application dans le widget `CarteDeVisite`. Donnez un titre à l'application. En `body`, l'application aura un widget `Center` vide.

Question 4 [solution n°6 p.44]

Créez une `card` au milieu de la page d'une hauteur de 150 et largeur de 300 de couleur grey.

Question 5 [solution n°7 p.44]

Créez trois zones. La première contiendra l'intitulé de la spécialité. Elle occupera ⅕ de la `card` en hauteur. La deuxième zone contiendra la photo / avatar avec le nom de la personne. Elle fera une hauteur de ⅗ de la `card`. Une troisième zone de la même proportion que la première contiendra l'e-mail de la personne.

Question 6 [solution n°8 p.45]

Insérez l'intitulé de la section dans la première zone.

Question 7 [solution n°9 p.45]

Dans la zone du milieu, insérez une image à gauche et le nom à droite.

Question 8 [solution n°10 p.45]

Dans la zone du bas, insérez le texte d'une adresse mail.

Question 9 [solution n°11 p.46]

Vérifiez les widgets à l'intérieur de la construction du widget `MaCarteDeVisite`.

B. Test

Exercice 1 : Quiz [solution n°12 p.47]

Question 1

Question 2

Question 3

Question 4

Question 5

Solutions des exercices

Exercice p. 15 Solution n°1

Question 1

Question 2

Question 3

Question 4

Question 5

Exercice p. 36 Solution n°2

Question 1

Question 2

Question 3

Question 4

Question 5

p. 40 Solution n°3

```dart
import 'package:flutter/material.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Flutter Demo',
     theme: ThemeData(
       primarySwatch: Colors.blue,
     ),
     home: CarteDeVisite()
   );
 }
}

class CarteDeVisite extends StatelessWidget {

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

p. 40 Solution n°4

Créez un répertoire dans le projet qui contiendra les assets. Puis à l'intérieur de ce dernier, on crée un répertoire image. On glisse l'image dans le répertoire image. Ensuite, il faut modifier le fichier pubspec.yaml pour configurer / indiquer le chemin où se trouve l'image à la rubrique assets.

```
# To add assets to your application, add an assets section, like this:
assets:
- assets/images/beach.png
#   - images/a_dot_ham.jpeg
```

p. 40 Solution n°5

```
class CarteDeViste extends StatelessWidget {

 @override
 Widget build(BuildContext context) {
   return(
     Scaffold(
       appBar: AppBar(
         title: Text("Carte de visite"),
       ),
       body: Center()
       ,
     )
   );
 }
}
```

p. 40 Solution n°6

```
body: Center(
 child: Card(
   child: Container(
     height: 150,
     width: 300,
     color:Colors.grey
   ),
 ),
)
```

p. 40 Solution n°7

```
Card(
 child:
 Container(
   color: Colors.grey,
   height: 150,
   width: 300,
   child: Column(
     children: [
       Expanded(
           flex:2,
           child: Container(
```

```
            color: Colors.yellow,
        )),
        Expanded(
            flex:6,
            child: Container(
              color:Colors.red
            )),
        Expanded(
            flex:2,
            child: Container(
                color:Colors.blue
            ))
      ],
    )
  ),
  elevation: 7.5,
),
```

p. 40 **Solution n°8**

```
Expanded(
   flex:2,
   child: Text("Développeur Full Stack"

)),
```

p. 40 **Solution n°9**

```
Expanded(
   flex:6,
   child: Row(
     children: [
       Expanded(
           flex:1,
           child: Padding(
             padding: const EdgeInsets.all(8.0),
             child: Container(
             child: Image.asset('assets/images/beach.png',),
       ),
           )),
       Expanded(flex:2
           ,child: Padding(
             padding: const EdgeInsets.all(8.0),
             child: Container(
               child: Text("DUPOND Philippe"),

       ),
           ))
     ],
   )
),
```

p. 40 **Solution n°10**

```
Expanded(
   flex:2,
   child: Row(
     children: [
       Padding(
         padding: const EdgeInsets.all(3.0),
         child: Text("Email : PhilippeDupond@gmail.com"),
       )
     ],
   )
)
```

p. 40 **Solution n°11**

```
class CarteDeVisite extends StatelessWidget  {

 @override
 Widget build(BuildContext context) {
   // TODO: implement build
   return Scaffold(
     appBar: AppBar(
       title:Text('Ma carte de Visite')
     ),
     body: Center(
       child: Card(
         child:
         Container(
           color: Colors.black12,
           height: 150,
           width: 300,
           child: Column(
             children: [
               Expanded(
                   flex:2,
                   child: Text("Développeur Full Stack"
               )),
               Expanded(
                   flex:6,
                   child: Row(
                     children: [
                       Expanded(
                           flex:1,
                           child: Padding(
                             padding: const EdgeInsets.all(8.0),
                             child: Container(
                             child: Image.asset('assets/images/beach.png',),
                       ),
                           )),
                       Expanded(flex:2
                           ,child: Padding(
                             padding: const EdgeInsets.all(8.0),
                             child: Container(
                               child: Text("DUPOND Philippe"),

                       ),
                           ))
                     ],
```

```
                    )
                ),
                Expanded(
                    flex:2,
                    child: Row(
                      children: [
                        Padding(
                          padding: const EdgeInsets.all(3.0),
                          child: Text("Email : PhilippeDupond@gmail.com"),
                        )
                      ],
                    )
                )
              ],
            )
          ),
          elevation: 7.5,
        ),
      ),
    );
}
```

Exercice p. 40 **Solution n°12**

Question 1

Question 2

Question 3

Question 4

Question 5