Développement Mobile

2025-2026

Chapitre 4 : Découverte des widgets et Material Design



I. Introduction aux widgets

1. Qu'est ce qu'un widget ?

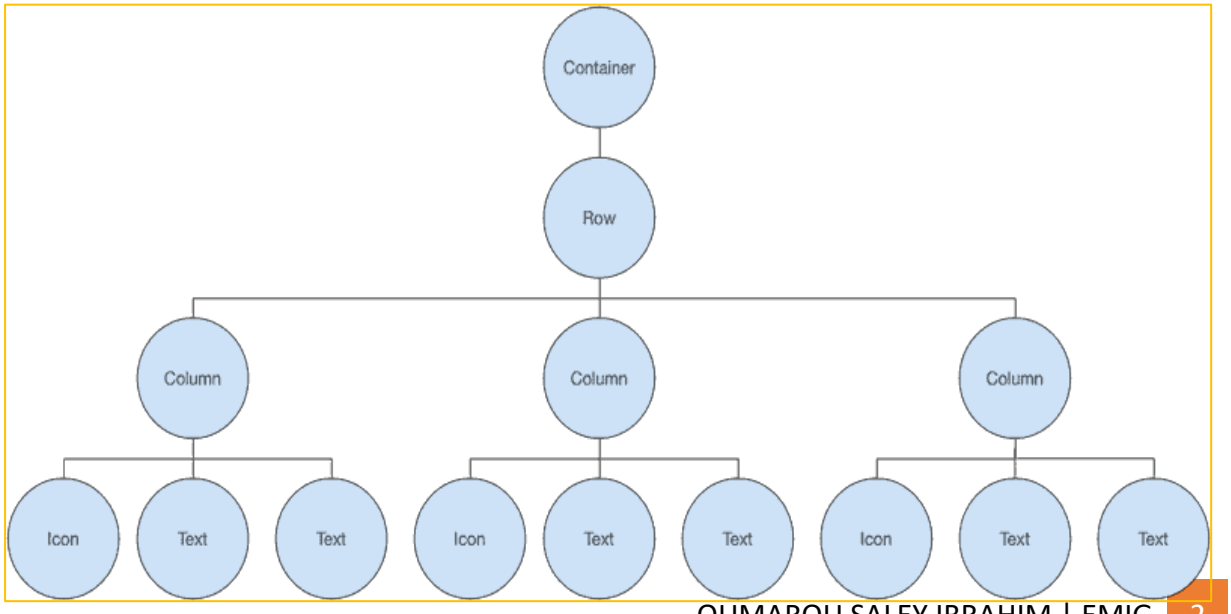
Un widget est tout ce qui compose votre application que ce soit un text, une image, un bouton, l'ensemble d'une page ou bien les opérations qui permettent de donner un résultat.

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

✓ Les widgets sont des classes

```
Text(
  'Bonjour, $_prenom ! Comment ça va ?',
  textAlign: TextAlign.center,
  overflow: TextOverflow.ellipsis,
  style: TextStyle(fontWeight: FontWeight.bold),
)
```

2. Arbre des widgets



3. La fonction runApp

La fonction runApp() prend en paramètre un Widget et en fait le widget racine de votre application. Il est au sommet de l'arbre des widgets.

Il ne faut donc utiliser cette fonction qu'une seul fois dans votre application, et lui passer le widget le plus haut dans l'arbre.

4. Le widget Center

Le widget Center permet de centrer le widget enfant passé en paramètre nommé child.

5. Le widget Text

Le widget Text permet d'afficher une chaîne de caractères en utilisant un seul style.

```
void main() {
 runApp(
  const Center(
   child: Text(
    'Hello, world !',
    textDirection: TextDirection.ltr,
    style: TextStyle(
       fontWeight: FontWeight.bold,
       fontStyle: FontStyle.italic,
       fontSize: 40),
   ),
  ),
 );
}
```

5.1 Le constructeur de la classe Text

```
Text(
  this.data, {
  Key key,
  this.style,
  this.strutStyle,
  this.textAlign,
  this.textDirection,
  this.locale,
  this.softWrap,
  this.overflow,
  this.textScaleFactor,
  this.maxLines,
  this.semanticsLabel,
 })
```

5.2 Le paramètre style

Le style peut recevoir un TextStyle qui a pour constructeur :

```
const TextStyle({
  this.inherit = true,
  this.color,
  this.backgroundColor,
  this.fontSize,
  this.fontWeight,
  this.fontStyle,
  this.letterSpacing,
  this.wordSpacing,
  this.textBaseline,
this.locale,
  this.foreground,
  this.background,
  this.shadows,
  this.decoration,
  this.decorationColor,
,
 })
```

Exemple:

```
void main() {
  runApp(
    const Center(
      child: Text(
        'Hello, world !',
        textDirection: TextDirection.ltr,
        style: TextStyle(
            fontWeight: FontWeight.bold,
            fontStyle: FontStyle.italic,
            fontSize: 40),
      ),
    ),
  );
}
```

6. Le widget Image

Le widget Image permet d'afficher une image avec Flutter

6.1 La source network

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

void main() {

  runApp(

    Center(

      child: Image.network("https://images.pexels.com/photos/414612/pexels-photo-414612.jpeg?auto=compress&cs=tinysrgb&dpr=3&h=750&w=1260")

    ),

  );

}
```

6.2 La source asset

Dans Flutter, les assets sont gérés dans le fichier pubspec.yaml.

Pour inclure des assets dans votre application, commencez par créer un dossier assets au même niveau que le fichier pubspec.yaml.

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

```
import 'package:flutter/material.dart';
void main() {
  runApp(
    Center(
      child: Image.asset('assets/images/lake.jpg'),
    ),
  );
}
```

7. Les widgets de layout

Avec Flutter, les éléments de Layout comme par exemple les rangées, colonnes et les grilles, sont des widgets écrits en Dart.

Il existe en effet plus de 30 widgets de layout et tous les voir d'un coup serait totalement contre-productif.

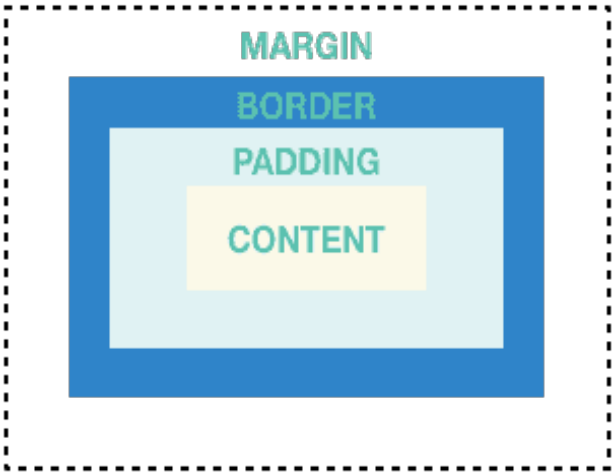
Nous allons voir les plus importants dans ce chapitre, à savoir les widgets container, padding, center, column, row, stack et expanded.

✓ Paramètre nommé commun

Tous les widgets de layout prennent en propriété nommé soit un child soit des children suivant qu'ils s'appliquent à un ou plusieurs autres widgets.

7.1 Le widget container

- Le widget container permet de gérer un widget enfant qu'il contient.
- Il permet d'ajouter du padding, de la margin et des borders à son widget enfant (child).
- Le fonctionnement est le suivant (avec border comme decoration, le content comme child)



✓ Exemple

```
void main() {
            runApp(
              Container(
                color: Colors.red,
                child: const Text("Hello world", textDirection: TextDirection.ltr),
              ),
            );
}
```

Le constructeur de Container

```
Container({
  Key key,
  this.alignment,
  this.padding,
  Color color,
  Decoration decoration,
  this.foregroundDecoration,
  double width,
  double height,
  BoxConstraints constraints,
  this.margin,
  this.transform,
  this.child,
 })
```

7.2 Le widget padding

Ce widget est extrêmement simple, il permet de créer un décalage d'espace vide autour de son child.

En fait, le widget Container utilise ce widget lorsque vous utilisez la propriété padding. Il construit ce widget pour vous.

```
void main() {
  runApp(Padding(
    padding: const EdgeInsets.all(50.0),
    child: Container(color: Colors.red),
  ));
}
```

7.3 Les widgets column et expanded

✓ Les widgets column

Le widget Column permet d'afficher des widgets enfants en colonne.
Ce widget utilise toujours children car l'objectif est d'aligner verticalement des éléments.

```
void main() {
  runApp(
    Column(
      children: [
        Container(
          color: Colors.red,
          height: 100,
          width: 100,
        ),
        Container(
          color: Colors.blue,
          height: 100,
          width: 100,
        ),
        Container(
          color: Colors.yellow,
          height: 100,
          width: 100,
        )
      ],
    ),
  );
}
```

✓ Le widget Expanded

Ce widget permet d'étendre un child d'une Column ou d'une Row pour qu'il remplisse tout l'espace disponible. Le child s'étendra sur l'axe principal donc vertical pour Column.
Il est également possible d'utilise Expanded sur plusieurs child.

```
void main() {
 runApp(
  Column(
   crossAxisAlignment: CrossAxisAlignment.center,
   textDirection: TextDirection.ltr,
   children: [
    Container(
     color: Colors.red,
     height: 100,
     width: 100,
    ),
    Expanded(
      child: Container(
      color: Colors.blue,
      height: 100,
      width: 100,
     )
    ),
    Container(
     color: Colors.green,
     height: 100,
     width: 100,
    ),
    Container(
     color: Colors.yellow,
     height: 100,
     width: 100,
    )
   ],
  )
 );
        }
```

7.4 Le widget row

Le widget Row est très similaire au widget Column, la différence étant que l'axe principal est l'axe horizontal et l'axe croisé est l'axe vertical.

```
void main() {
 runApp(
  Row(
   crossAxisAlignment: CrossAxisAlignment.center,
   textDirection: TextDirection.ltr,
   children: [
    Container(
     color: Colors.red,
     height: 80,
     width: 80,
    ),
    Container(
     color: Colors.blue,
     height: 80,
     width: 80,
    ),
    Container(
     color: Colors.green,
     height: 80,
     width: 80,
    ),
    Container(
     color: Colors.yellow,
     height: 80,
     width: 80,
    ),
   ],
  )
 );
```

7.5 Le widget stack

Ce widget, "pile" en français, permet de superposer plusieurs éléments child.Par exemple, il permet de placer un texte ou un bouton sur une image.

```
void main() {
 runApp(
            Stack(
              textDirection: TextDirection.ltr,
              children: <Widget>[
                Container(
                  width: 400,
                  height: 400,
                  color: Colors.red,
                ),
                Container(
                  width: 300,
                  height: 300,
                  color: Colors.green,
                ),
                Container(
                  width: 200,
                  height: 200,
                  color: Colors.blue,
                ),
              ],
            )
          );
}
```

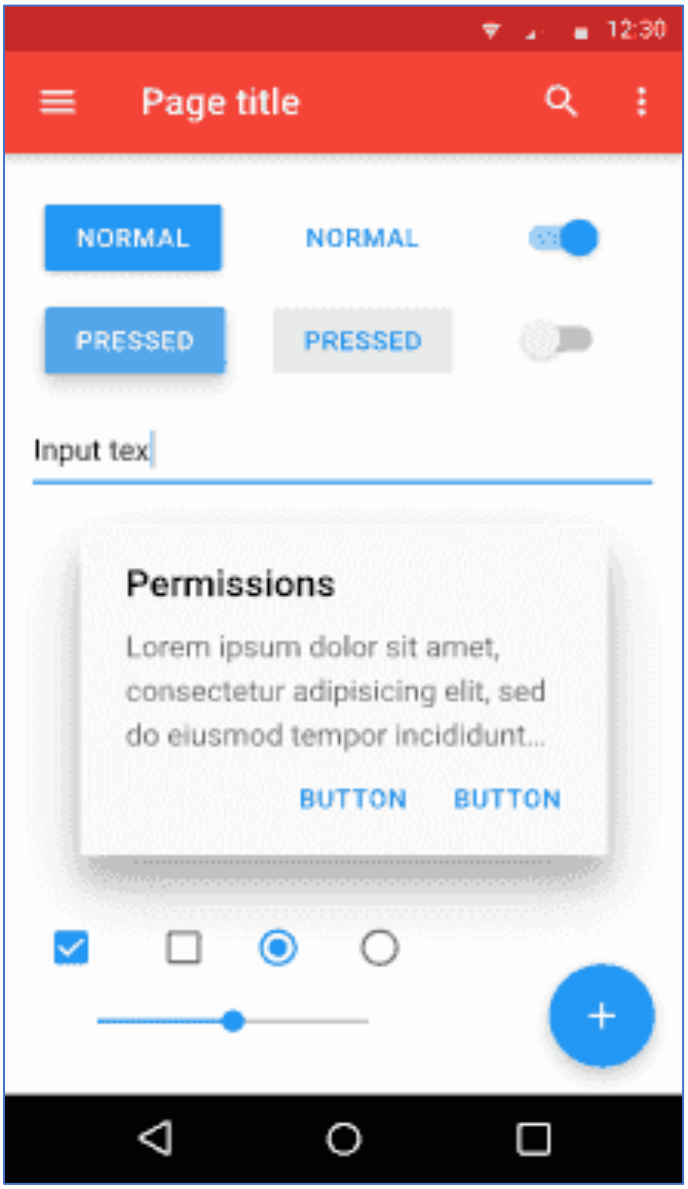
II. Introduction á Material

1. Le design Material

Le design Material est un ensemble de règles proposées par Google pour les interfaces utilisateurs.
Google le définit comme "un langage visuel qui synthétise les principes classiques d'un bon design avec l'innovation technologique et scientifique".

Ces principes concernent le layout, la navigation, les couleurs, la typographie, l'iconographie, les formes, le mouvement (animations), les interactions possibles etc.

Ces règles sont ensuite mises en oeuvre dans des systèmes d'exploitation (Android), par des librairies de composants pour des frameworks (librairie Material pour Angular), ou directement par des frameworks (Flutter).



2. Les applications Material

Pour utiliser tous les widgets Material de Flutter, il suffit d'importer le package correspondant :

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

✓ Notre première application Material

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

void main() {

 runApp(MaterialApp(

    title: 'Flutter Demo',

    theme: ThemeData(primarySwatch: Colors.blue),

    home: Container(

       alignment: Alignment.center, child: const Text('Hello world !'))));

}
```



3. Le widget Scaffold

Scaffold est un widget Material et il ne peut s'utiliser que dans une MaterialApp.
Ce widget permet d'appliquer un design Material et de structurer votre application. Vous pouvez le voir comme une base de la structure de votre application qui va structurer les autres widgets.

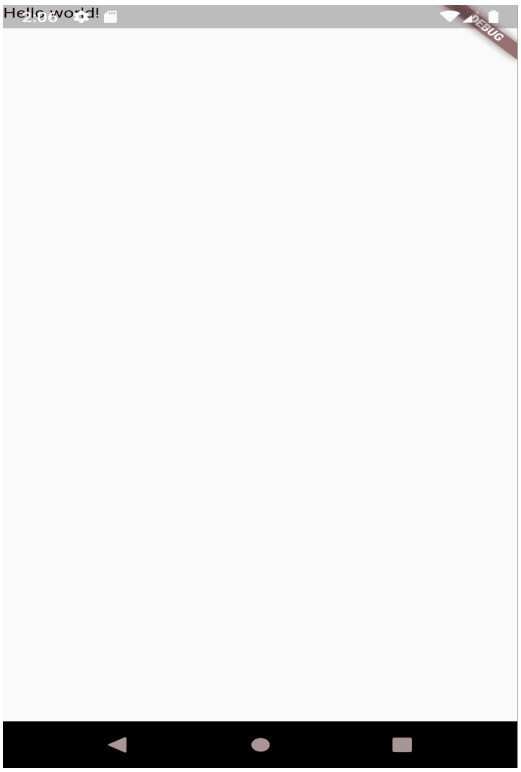
```
const Scaffold({
  Key key,
  this.appBar,
  this.body,
  this.floatingActionButton,
  this.floatingActionButtonLocation,
  this.floatingActionButtonAnimator,
  this.persistentFooterButtons,
  this.drawer,
  this.endDrawer,
  this.bottomNavigationBar,
  this.bottomSheet,
  this.backgroundColor,
  this.resizeToAvoidBottomPadding,
  this.resizeToAvoidBottomInset,
  this.primary = true,
  this.drawerDragStartBehavior = DragStartBehavior.start,
  this.extendBody = false,
}
```

3.1 La propriété body

La propriété body est le contenu principal du scaffold

```
import 'package:flutter/material.dart';
void main() {

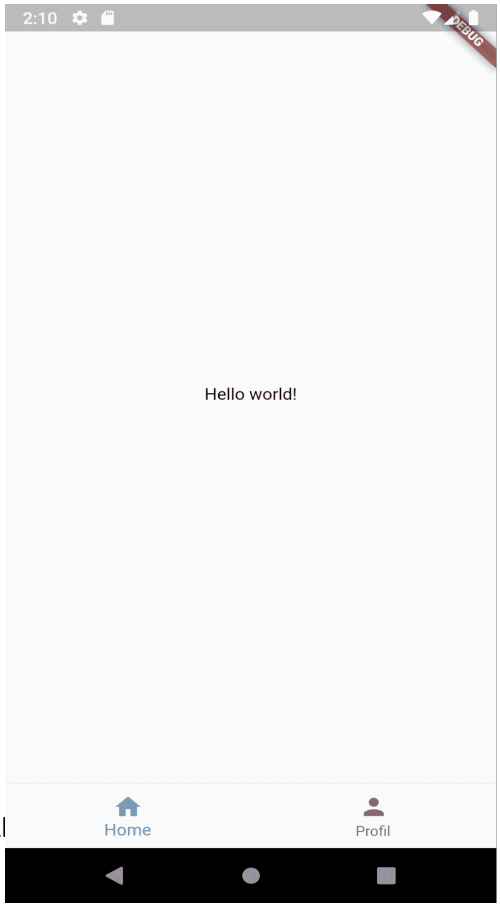
  runApp(const MaterialApp(
     home: Scaffold(body: Text('Hello world!'))));
}
```



3.2 La propriété bottomNavigationBar

Cette propriété permet d'afficher une barre de navigation en bas de l'écran.

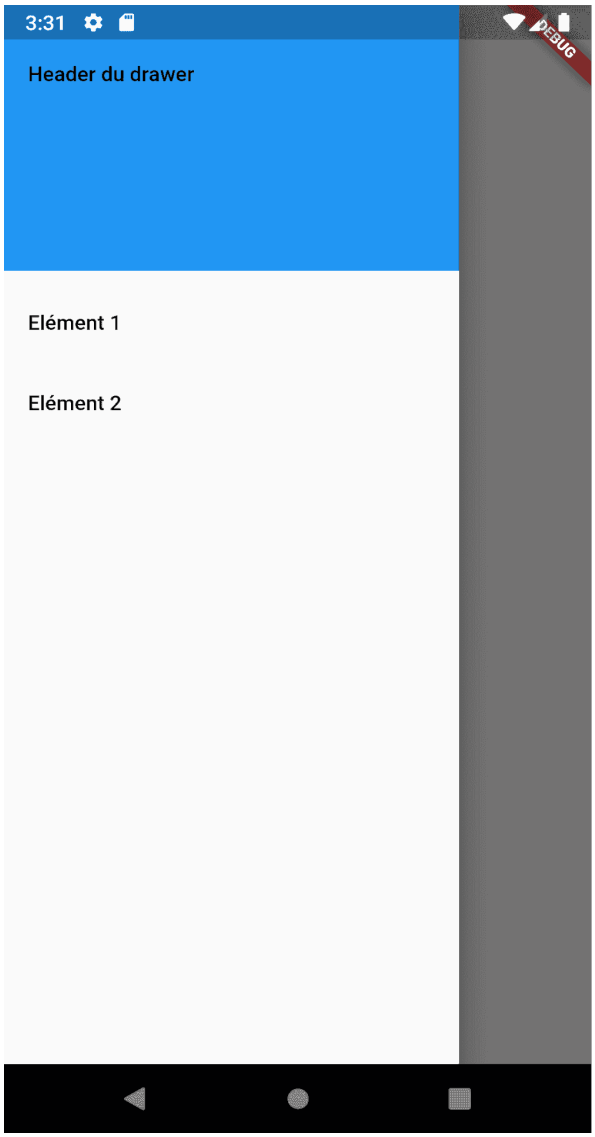
```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(
      home: Scaffold(
    body: const Center(
      child: Text('Hello world!'),
    ),
    bottomNavigationBar: BottomNavigationBar(
   items: const [
      BottomNavigationBarItem(label: 'Home', icon: Icon(Icons.home)),
      BottomNavigationBarItem(label: 'Profil', icon: Icon(Icons.person))
    ]),
  )));
}
```



3.3 La propriété drawer

Un drawer est un tiroir, c'est-à-dire une sorte de panneau dépliant sur le côté du body.

```
import 'package:flutter/material.dart';
void main() {
 runApp(MaterialApp(
   home: Scaffold(
  appBar: AppBar(title: const Text('Test')),
  body: const Center(
   child: Text('Hello world!'),
  ),
  drawer: Drawer(
   child: ListView(
    padding: EdgeInsets.zero,
    children: const <Widget>[
     DrawerHeader(
      decoration: BoxDecoration(
       color: Colors.blue,
      ),
      child: Text('Header du drawer'),
     ),
     ListTile(
      title: Text('Elément 1'),
     ),
     ListTile(
      title: Text('Elément 2'),
     ),
    ],
   ),
  ),
 )));
}
```

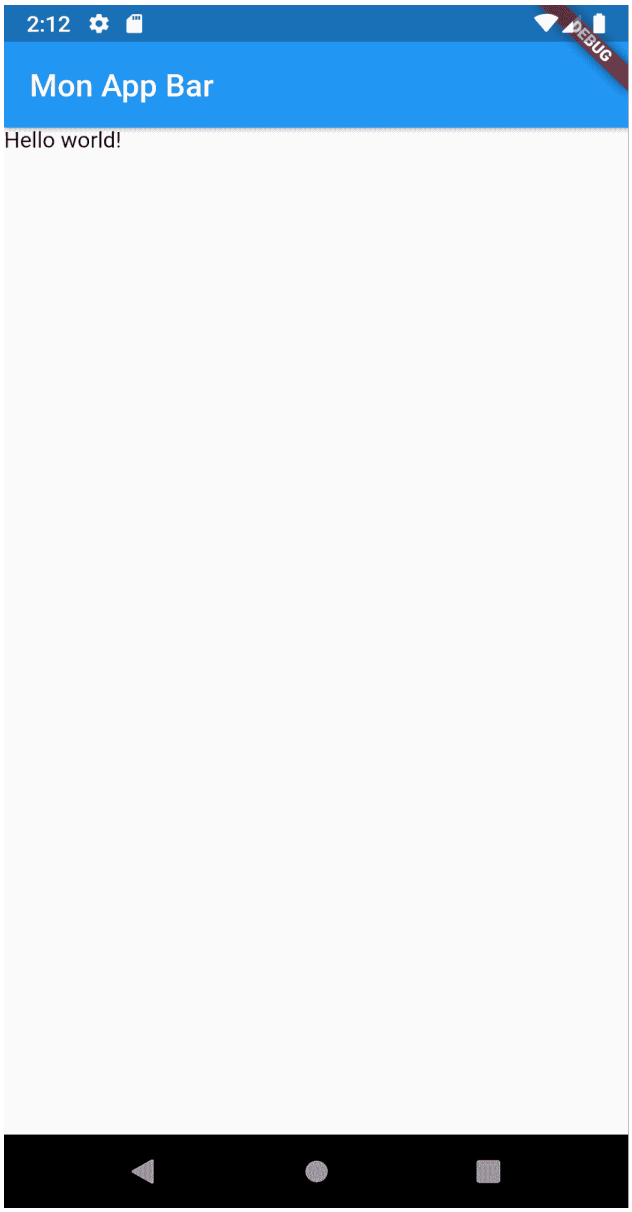


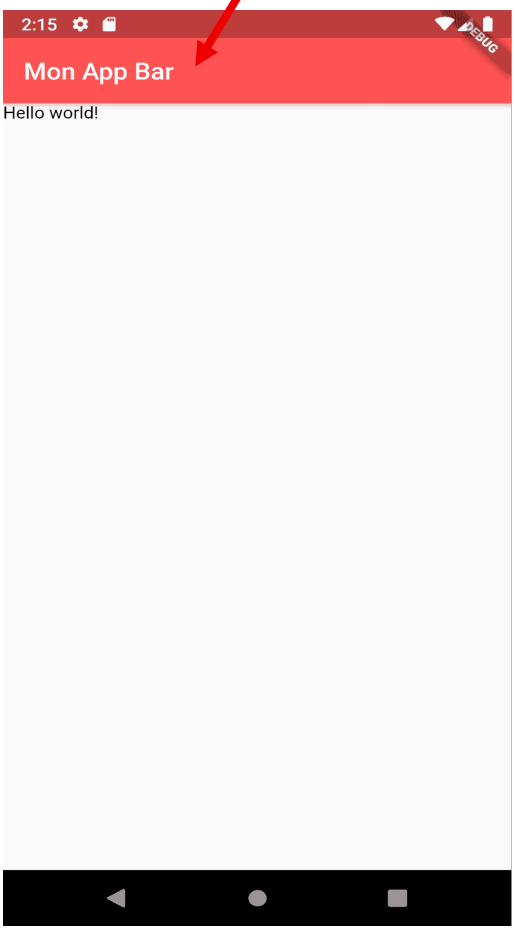
3.4 La propriété appBar

Cette propriété du widget Scaffold permet de créer une top bar dans votre application. Vous devez lui passer une instance de la classe AppBar. Vous pouvez ne passer aucun argument et vous aurez une top barre par défaut.

```
import 'package:flutter/material.dart';
void main() {
 runApp(MaterialApp(
    home: Scaffold(
       appBar: AppBar(
         title: const Text('Mon App Bar'),
         backgroundColor: Colors.redAccent,

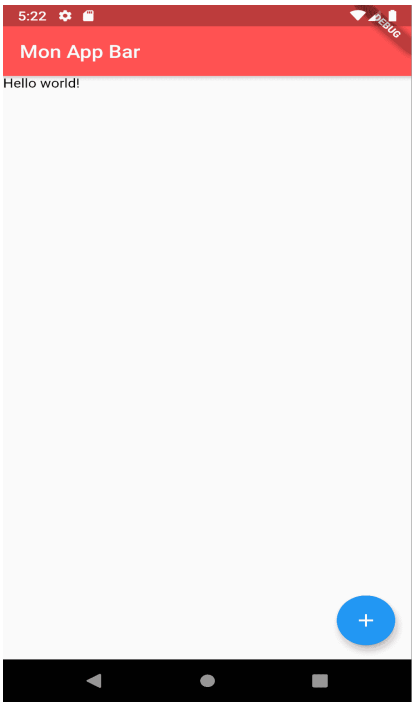
       ),
       body: const Text('Hello world!'))));
}
```

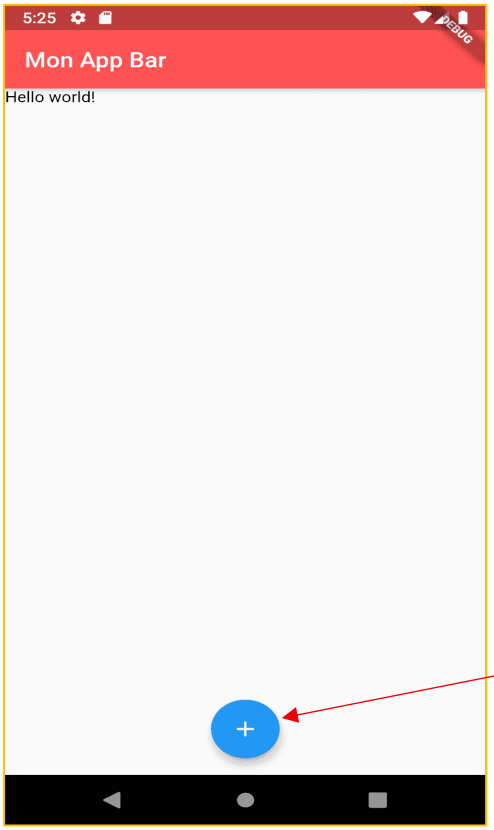




3.5 La propriété floatingActionButton

```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(
     home: Scaffold(
    appBar: AppBar(
     title: const Text('Mon App Bar'),
     backgroundColor: Colors.redAccent,
    ),
    body: const Text('Hello world!'),
    floatingActionButton: FloatingActionButton(
       child: const Icon(Icons.add), onPressed: () =>
print('Appuyé !')),
  )));
}
```





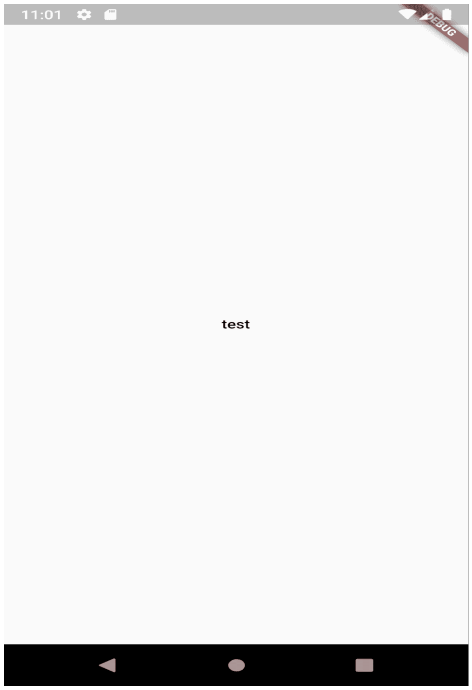

```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(
     home: Scaffold(
    appBar: AppBar(
     title: const Text('Mon App Bar'),
     backgroundColor: Colors.redAccent,
    ),
    body: const Text('Hello world!'),
    floatingActionButton: FloatingActionButton(
       child: const Icon(Icons.add), onPressed: () =>
print('Appuyé !')),
    floatingActionButtonLocation:
FloatingActionButtonLocation.centerFloat,
  )));
}
```

Les valeurs possibles sont : endFloat (valeur par défaut), centerFloat, endDocked, centerDocked, startTop, miniStartTop et endTop.

4. Le widget TextButton

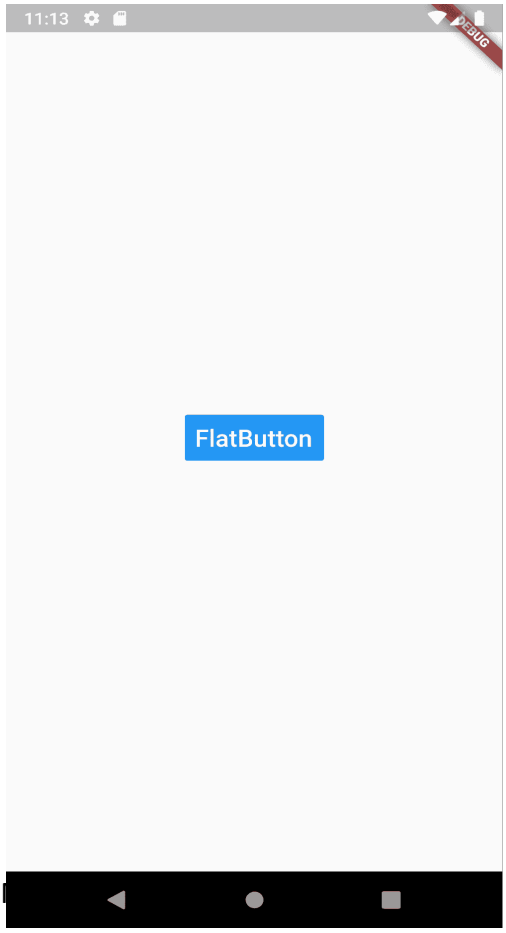
Un TextButton est un label textuel qui réagit au pressions en se remplissant avec une couleur.

```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(
     home: Scaffold(
    body: Center(
      child: TextButton(
        child: const Text('test'),
        onPressed: () {},
      ),
    ),
  )));
}
```



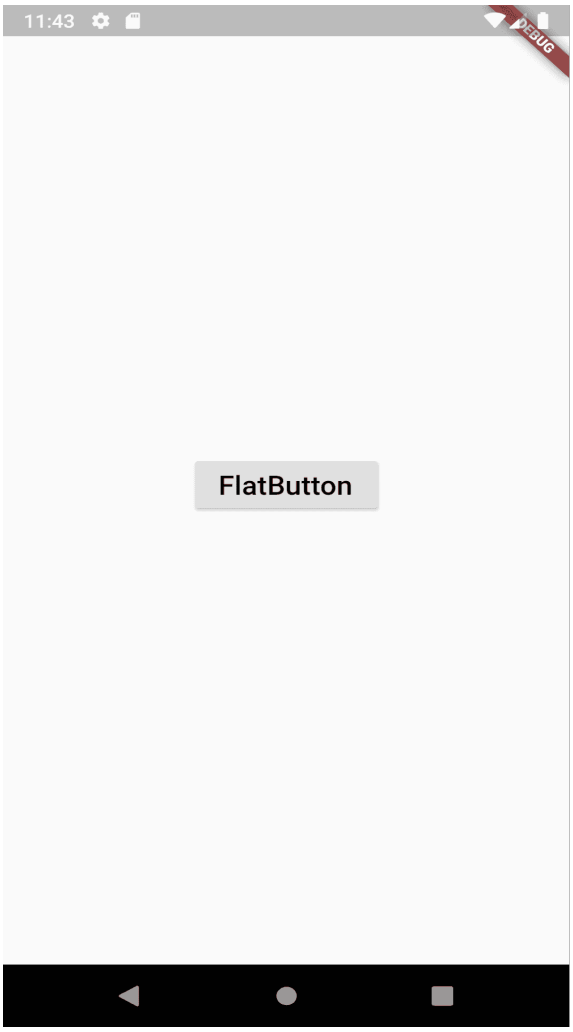

```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(
     home: Scaffold(
    body: Center(
      child: TextButton(
        style: TextButton.styleFrom(
          foregroundColor: Colors.white,
          backgroundColor: Colors.blue,
          disabledForegroundColor: Colors.grey,
          padding: const EdgeInsets.all(15.0),
          textStyle: const TextStyle(
            fontSize: 20.0,
          )),
        onPressed: () {},
        child: const Text("TextButton"),
      ),
    ),
  )));
}
```

OU



5. Le widget ElevatedButton

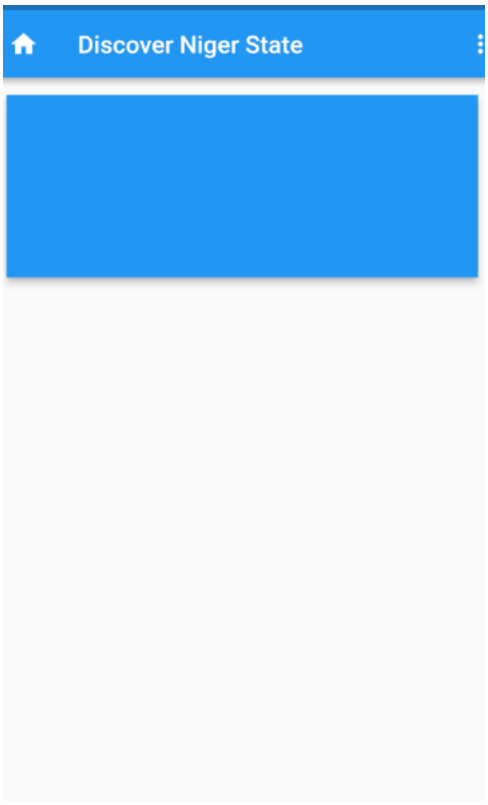
```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(
      home: Scaffold(
    body: Center(
      child: ElevatedButton(
        onPressed: () {},
        child: const Text(
          "ElevatedButton",
          style: TextStyle(fontSize: 20.0),
        ),
      ),
    ),
  )));
}
```



6. Le widget Card

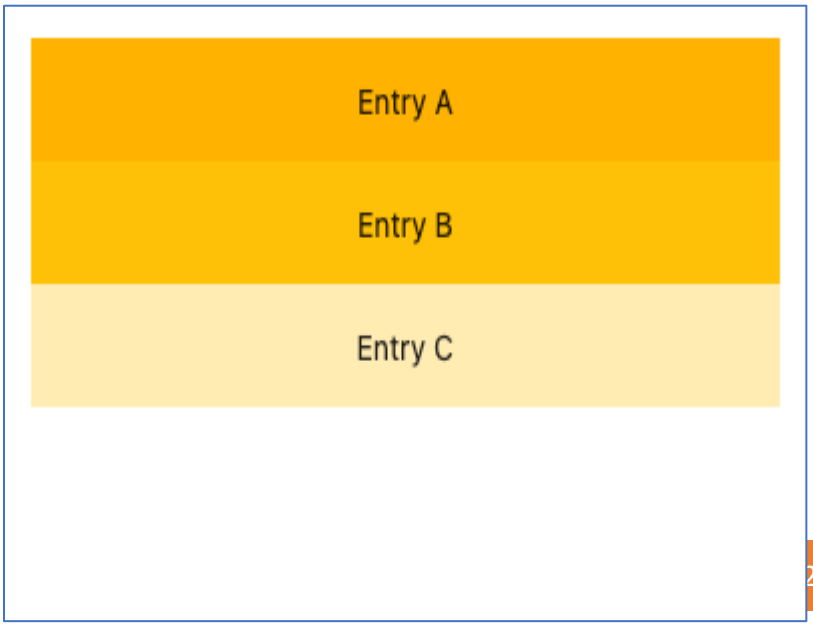
Une Card est une sorte de container qui a la forme d'une carte : rectangulaire, bord arrondi légérement et une ombre donnant un petit effet 3D.

```
      body: Container(
        padding: const EdgeInsets.all(10),
        child: Column(
          crossAxisAlignment:
CrossAxisAlignment.stretch,
          children: <Widget>[
            Card(
              elevation: 5,
              child: Container(
                height: 150,
                color: Colors.blue,
              ),
            )
          ],
        ),
      ),
```



7. ListView

Dans Flutter, ListView est une liste déroulante de widgets disposés linéairement.
Il affiche ses enfants les uns après les autres dans le sens du défilement, c'est-à-dire vertical ou horizontal.



8. GridView

Flutter GridView est un widget similaire à un tableau 2D dans n'importe quel langage de programmation.
Comme son nom l'indique, un widget GridView est utilisé lorsque nous devons afficher quelque chose sur une grille.
Nous pouvons afficher des images, du texte, des icônes, etc.



9. Divider

Le widget Divider est utilisé lorsque vous avez plusieurs enfants et que vous souhaitez séparer par la ligne ou le séparateur.
Un exemple d'image est donné ci-dessous pour avoir une idée de ce que nous allons faire dans cet article.

```
const Divider(
        height: 20,
        thickness: 5,
        indent: 20,
        endIndent: 0,
        color: Colors.black,
      ),
```



10. SizedBox

Il s'agit d'une simple boîte avec une taille spécifiée.

Il peut être utilisé pour définir des contraintes de taille sur le widget enfant, mettre un SizedBox vide entre les deux widgets pour obtenir de l'espace entre les deux, ou autre chose.

Il est quelque peu similaire à un widget Conteneur avec moins de propriétés.

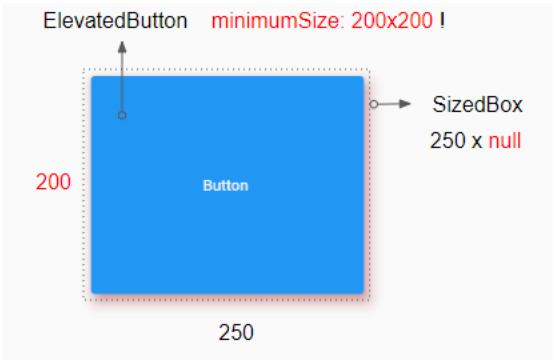
```
Const SizedBox(

width: 200.0,

height: 300.0,

child: Card(child: Text('Hello World!')),

 )
```



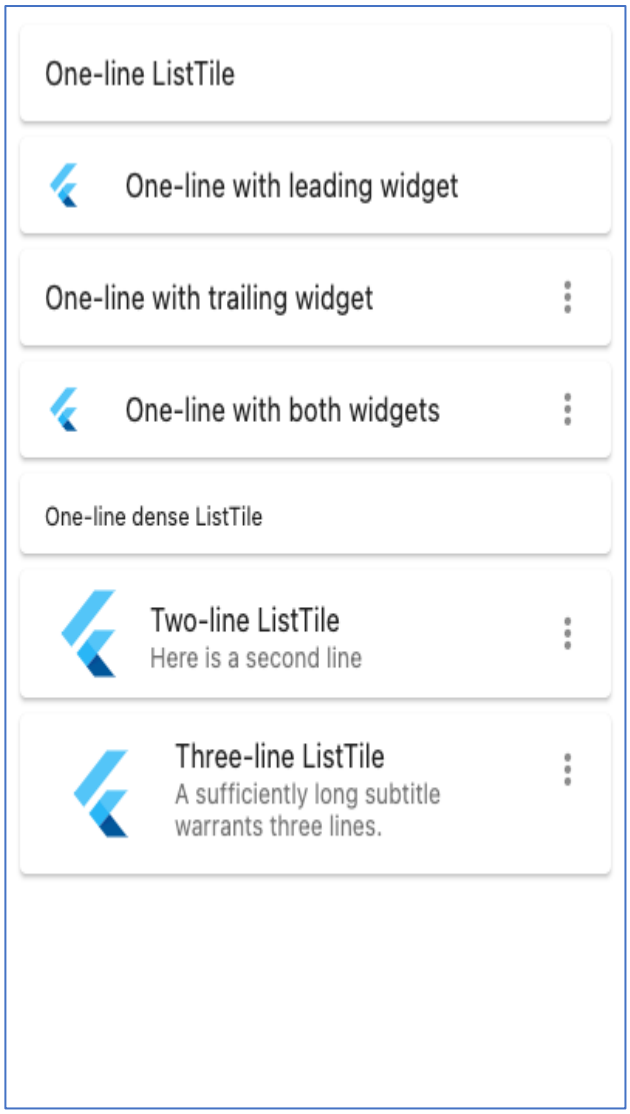
11. ListTile

Le widget ListTile est utilisé pour remplir un ListView dans Flutter.

Il contient le titre ainsi que les icônes de début ou de fin. Comprenons cela à l'aide d'un exemple.

✓ **Le Constructeur**

```
const ListTile({
  super.key,
  this.leading,
  this.title,
  this.subtitle,
  this.trailing,
  this.isThreeLine = false,
  this.dense,
  this.visualDensity,
  this.shape,
  this.style,
  this.selectedColor,
  this.iconColor,
  this.textColor,
  this.contentPadding,
  this.enabled = true,
  this.onTap,
  this.splashColor,
    this.enableFeedback,
  this.horizontalTitleGap,
  this.minVerticalPadding,
  this.minLeadingWidth,
})
```



12. CircleAvatar

Il s'agit simplement d'un cercle dans lequel nous pouvons ajouter une couleur d'arrière-plan, une image d'arrière-plan ou simplement du texte.

```
CircleAvatar(
backgroundImage:
NetworkImage(userAvatarUrl), )
```



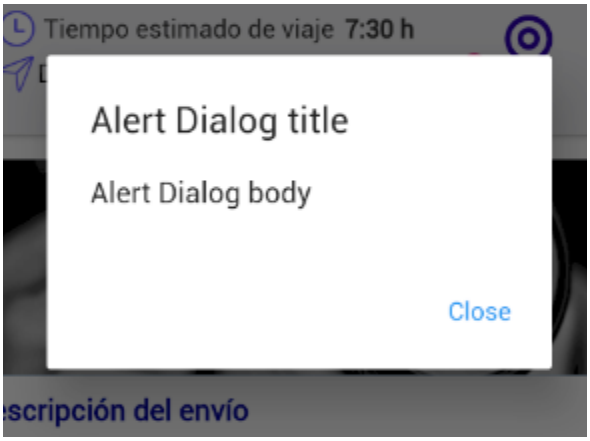
13. Dialog et le SnackBar

Les Dialog et SnackBar permettent d'interagir avec l'utilisateur en affichant des messages contextuels. Ils sont essentiels pour confirmer des actions, afficher des erreurs ou notifier le succès d'une opération.

13.1 Dialog

Un **Dialog** est une fenêtre modale qui s'affiche au-dessus du contenu principal pour demander une action ou afficher une information importante.

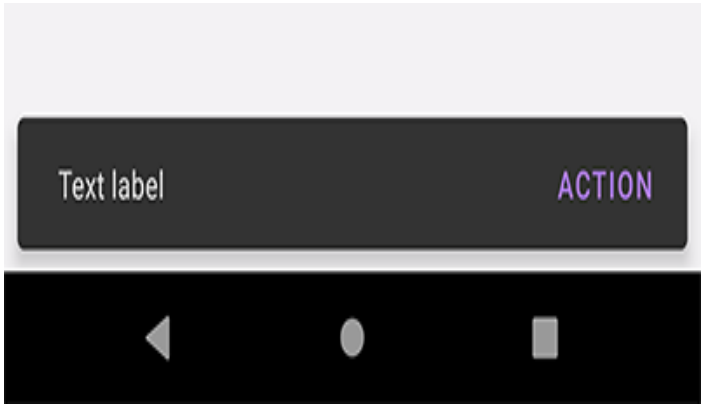
```
showDialog(
  context: context,
  builder: (BuildContext context) {
    return AlertDialog(
      title: const Text("Confirmation"),
      content: const Text("Souhaitez-vous
vraiment supprimer cet élément ?"),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context),
          child: const Text("Annuler"),
        ),
        TextButton(
          onPressed: () {
            // Action de suppression
            Navigator.pop(context);
          },
          child: const Text("Oui"),
        ),
```



13.2 SnackBar

Un SnackBar est un petit message affiché en bas de l'écran pendant quelques secondes. Il est souvent utilisé pour indiquer le succès ou l'échec d'une action sans interrompre la navigation.

```
ScaffoldMessenger.of(context).showSnackBar(
 const SnackBar(
  content: Text("Enregistrement effectué avec succès !"),
  duration: Duration(seconds: 3),
 ),
```

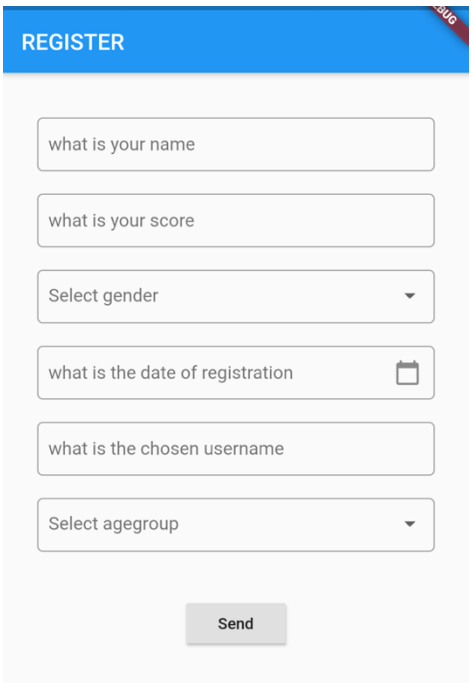


14. Gestion des formulaires

Les formulaires sont utilisés pour collecter des informations auprès de l'utilisateur (nom, e-mail, mot de passe, etc.).
Flutter propose le widget **Form** associé à une clé globale (**GlobalKey<FormState>**) pour la validation.

14.1 Création d'un formulaire de base

```
Form(
      key: _formKey,
      child: Column(
        children: const [
          TextField(decoration: InputDecoration(labelText: "Nom")),
          TextField(decoration: InputDecoration(labelText: "Email")),
        ],
      ),
```

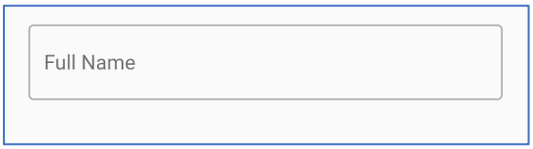


14.1.1 Le widget TextField()

Le widget TextField() est le composant de base pour permettre à l'utilisateur de saisir du texte dans une application Flutter.

```
TextField(decoration:
InputDecoration(labelText: "Nom")),

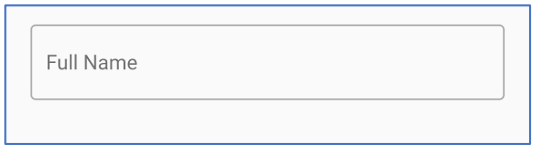
TextField(decoration:
InputDecoration(labelText: "Email")),
```



14.1.2 Le widget TextFormField()

Le widget TextFormField() est une version améliorée de TextField(), conçue pour être utilisé à l'intérieur d'un formulaire (Form).

```
TextFormField(
        decoration: const InputDecoration(
          labelText: "Nom complet",
          hintText: "Entrez votre nom",
          border: OutlineInputBorder(),
        ),
```



14.2 Création d'un formulaire complet

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

  @override
  State<ExempleTextFormField> createState() => _ExempleTextFormFieldState();
}
class _ExempleTextFormFieldState extends State<ExempleTextFormField> {
  final _formKey = GlobalKey<FormState>(); // clé pour le formulaire
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("TextFormField - Exemple")),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Form(
          key: _formKey,
          child: Column(
            children: [
              TextFormField(
                decoration: const InputDecoration(
                  labelText: "Nom complet",
                  hintText: "Entrez votre nom",
                  border: OutlineInputBorder(),
                ),
                validator: (value) {
                  if (value == null || value.isEmpty) {
                    return "Ce champ est obligatoire";
                  }
                  return null;
                },
                onSaved: (value) => nom = value,
              ),
              const SizedBox(height: 20),
              ElevatedButton(
                onPressed: () {
                },
                child: const Text("Valider"),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

15. Le BuildContext

- Un BuildContext est une référence à l'emplacement du widget dans l'arbre des widgets de votre application.
- Un BuildContext est propre à un seul widget.
- Chaque widget a son propre BuildContext et l'ensemble des contextes forment également un arbre, de la même manière que l'arbre des widgets.
- Graĉe au contexte, il est possible d'accéder dans un widget aux widgets parents. Ce qui est très utile comme nous le verrons.