Interface utilisateur (UI) Android

1. Objectif

- Apprendrez à créer une interface utilisateur (UI) à l'aide de composants Android.

2. Qu'est-ce que l'interface utilisateur (UI) Android ?

- L'interface utilisateur (UI) fait référence à la disposition d'une application : son apparence et son style général. L'interface utilisateur est principalement axée sur l'interaction directe de l'utilisateur avec un ordinateur, un site Web ou un produit.
- L'interface utilisateur (UI) est tout ce que les utilisateurs voient. Android fournit de nombreux composants d'interface utilisateur aux développeurs Android pour concevoir une interface utilisateur graphique (UI) dans l'application Android.
- L'interface utilisateur de votre application est tout ce que l'utilisateur peut voir et avec lequel interagir. Android fournit une variété de composants d'interface utilisateur prédéfinis, tels que des objets de mise en page structurés et des contrôles d'interface utilisateur, qui vous permettent de créer l'interface utilisateur graphique de votre application.
- Android fournit également d'autres modules d'interface utilisateur pour des interfaces spéciales telles que des boîtes de dialogue, des notifications et des menus.
- Dans Android, l'interface utilisateur (UI) peut être une vue texte, des boutons, une vue image, modifier du texte, des mises en page, une vue liste, une vue calendrier, un sélecteur de date, une boîte de dialogue, un menu déroulant, une barre d'action/barre d'application, des onglets, un tiroir de navigation, etc.
- Avec l'aide de tous ces composants, nous pouvons créer une application Android d'interface utilisateur.
- Construire une interface utilisateur (UI) dans Android n'est pas très difficile car Android fournit de nombreux composants pour créer une bonne application Android UI.

3. Eléments clés de l'interface utilisateur Android

- L'interface utilisateur (UI) Android est l'aspect visuel et interactif d'une application Android. Elle est conçue pour permettre aux utilisateurs d'interagir avec l'application de manière conviviale et intuitive.
- Voici quelques éléments clés de l'interface utilisateur Android :

- **Activités** : Une activité est une composante essentielle de l'interface utilisateur Android. Chaque écran visible de votre application est généralement implémenté en tant qu'activité. Par exemple, une application de messagerie peut avoir une activité pour la boîte de réception, une autre pour la rédaction de messages, etc.
- **Fenêtres et Mises en Page** : Les fenêtres sont des conteneurs pour les éléments de l'interface utilisateur. Une activité Android est généralement une fenêtre à part entière. Les mises en page (layouts) sont utilisées pour organiser les éléments de manière structurée à l'intérieur d'une fenêtre. Les mises en page courantes incluent LinearLayout, RelativeLayout, ConstraintLayout, etc.

- **Vues (Views)** : Les vues sont des éléments d'interface utilisateur tels que des boutons, des zones de texte, des images, des listes déroulantes, etc. Chaque élément interactif visible à l'écran est généralement une vue. Les vues sont généralement placées dans des mises en page.
- **Widgets** : Les widgets sont des composants graphiques préfabriqués qui peuvent être réutilisés dans votre application Android. Par exemple, un bouton est un widget courant. Vous pouvez personnaliser les widgets pour répondre aux besoins de votre application.
- **Barres d'Action et Barres d'Outils** : Les barres d'action (action bars) et les barres d'outils (toolbars) fournissent un moyen de naviguer dans l'application et d'effectuer des actions. La barre d'action est généralement située en haut de l'écran et contient des actions contextuelles, tandis que la barre d'outils peut être personnalisée et placée n'importe où dans l'interface.
- **Navigation** : La navigation dans une application Android est généralement gérée via des boutons de navigation, des onglets, des menus de navigation, des listes déroulantes, etc. Il est important de concevoir une navigation fluide pour permettre aux utilisateurs de se déplacer facilement dans l'application.
- **Couleurs et Thèmes** : Le choix des couleurs, des polices et des thèmes joue un rôle crucial dans la conception de l'interface utilisateur. Les thèmes permettent de définir l'apparence globale de l'application, tandis que les couleurs et les polices contribuent à son esthétique.
- **Réactivité et Convivialité** : Une interface utilisateur Android doit être réactive aux actions de l'utilisateur. Cela signifie que les éléments de l'interface utilisateur doivent réagir rapidement aux interactions, tels que les clics, les balayages et les zooms. La convivialité est essentielle pour offrir une expérience utilisateur de qualité.
- **Adaptabilité aux Différentes Tailles d'Écran** : Étant donné que les appareils Android sont disponibles dans de nombreuses tailles d'écran différentes, il est important de concevoir une interface utilisateur qui s'adapte à ces variations, en utilisant des mises en page flexibles et adaptatives.
- **Tests Utilisateur** : Avant de publier une application Android, il est recommandé de mener des tests utilisateur pour recueillir des commentaires et identifier les problèmes potentiels d'utilisabilité. Cela permet d'ajuster et d'améliorer l'interface utilisateur en fonction des retours d'expérience.

- L'interface utilisateur Android est un élément clé de la conception d'applications Android réussies.
- Elle joue un rôle majeur dans l'expérience utilisateur globale et nécessite une attention particulière pour assurer la convivialité, la réactivité et la cohérence visuelle de l'application.

4. Les composants d'interface utilisateur (UI) Android

- Les composants d'interface utilisateur (**UI**) Android sont les éléments qui forment l'apparence et le comportement d'une application Android. Voici une description détaillée de ces composants, étape par étape :

- **Conception d'Interface Utilisateur (UI) Android Layout**

- **Disposition Linéaire (Linear Layout)** : Une mise en page qui organise les éléments de manière linéaire, soit horizontalement, soit verticalement. C'est utile pour aligner les éléments les uns en dessous des autres ou côte à côte.
- **Disposition Relative (Relative Layout)** : Cette mise en page permet de définir les relations entre les éléments en fonction de leur position les uns par rapport aux autres

ou par rapport au parent. Elle offre une grande flexibilité pour la conception d'interfaces utilisateur complexes.

- **Disposition du Tableau (Table Layout)** : Organise les éléments dans un tableau à deux dimensions de lignes et de colonnes, similaire à une grille.
- **Disposition du Cadre (Frame Layout)** : Une mise en page simple qui empile les éléments les uns sur les autres. Souvent utilisée pour afficher un seul élément à la fois.
- **Disposition Absolue (Absolute Layout)** : Moins recommandée, cette mise en page positionne les éléments de manière absolue à des coordonnées spécifiques. Elle est moins flexible en termes de réactivité aux différentes tailles d'écran.

- **Conception d'Interface Utilisateur (UI) Android View**

- **Affichage de Défilement Vertical (Scroll View – Vertical)** : Permet aux éléments enfants de défiler verticalement lorsque l'espace disponible est insuffisant. Utile pour afficher du contenu qui ne tient pas à l'écran.
- **Vue de Défilement Horizontal (Horizontal Scroll View)** : Similaire au ScrollView vertical, mais pour le défilement horizontal.
- **Exemple de Vue de Liste Simple (Simple List View Example)** : Une vue qui affiche une liste déroulante simple d'éléments. Souvent utilisée pour afficher des listes de données.
- **Affichage de Liste dans Android (List View in Android)** : Une vue plus avancée pour afficher des listes de données avec des éléments personnalisables. Utile pour les listes longues et interactives.
- **Position des Vues dans Android (Views Position in Android)** : Cette section explique comment positionner les vues à l'intérieur des mises en page en utilisant les attributs de disposition.

- **Conception de l'interface utilisateur Android : commandes d'entrée**

- **Bouton Android** : Un élément interactif courant pour déclencher des actions.
- **Bouton d'Image** : Un bouton avec une image comme étiquette.
- **Éditer le Texte** : Une zone de texte où les utilisateurs peuvent entrer du texte.
- **Case à Cocher** : Permet aux utilisateurs de sélectionner ou de désélectionner une option.
- **Bouton Radio** : Les boutons radio permettent aux utilisateurs de choisir une option parmi plusieurs.
- **Bouton de l'Interrupteur** : Utilisé pour activer ou désactiver une fonction.
- **Bouton à Bascule** : Permet aux utilisateurs de basculer entre deux états.
- **Barre d'Évaluation** : Une barre pour noter ou évaluer quelque chose.
- **Spinner** : Une liste déroulante d'options à sélectionner.
- **Sélecteur de Date et Sélecteur de Temps** : Utilisés pour choisir une date ou une heure dans une interface utilisateur.

- **Conception de l'interface utilisateur Android : Toast, dialogue et snackbar**

- **Message de Pain Grillé (Toast)** : Une petite notification éphémère affichée à l'écran pour informer l'utilisateur.
- **Boîte de Dialogue d'Alerte** : Une fenêtre contextuelle qui demande à l'utilisateur de prendre une décision ou de confirmer une action.
- **Snackbar** : Une notification s'affichant en bas de l'écran, souvent utilisée pour fournir des retours d'action.

- **Conception de l'interface utilisateur Android : navigation, barre d'action et menus**

- **Tiroir de Navigation (Drawer Navigation)** : Un menu latéral rétractable pour la navigation et les options de menu.

- **Menu Coulissant avec WebView** : L'intégration d'une vue Web dans un menu coulissant.
- **Menu Déroulant (Dropdown Menu)** : Un menu déroulant qui s'affiche lorsqu'un élément est sélectionné.
- **Barre d'Action/Barre d'Application (Action Bar/App Bar)** : Une barre en haut de l'écran qui contient des actions contextuelles.
- **Barre d'Outils (Toolbar)** : Une barre personnalisable pour la navigation et les actions.
- **Onglets de la Barre d'Action (Action Bar Tabs)** : Des onglets pour basculer entre différentes vues ou fonctionnalités.

- **Conception d'interface utilisateur Android : bibliothèque d'aide à la conception**

- **Affichage du Tiroir de Navigation (Navigation Drawer)** : Comment afficher le tiroir de navigation dans une interface utilisateur.
- **Bouton d'Action Flottant (Floating Action Button – FAB)** : Un bouton d'action circulaire souvent utilisé pour les actions principales.
- **Casse-Croûte (Snackbar)** : Comment afficher une Snackbar dans une interface utilisateur.
- **Étiquette Flottante pour EditText** : Comment ajouter une étiquette flottante à une zone de texte pour améliorer l'expérience utilisateur.
- Ces composants et concepts sont essentiels pour la création d'interfaces utilisateur efficaces dans les applications Android. En les maîtrisant, vous pouvez concevoir des interfaces utilisateur conviviales et fonctionnelles pour vos applications Android.

Layouts dans Flutter ou Widgets de mise en page

1. Objectif

- Apprenez à utiliser des widgets de mise en page pour conserver la dimension et l'orientation de n'importe quel composant dans les applications Flutter.

2. Présentation

- Un widget dans Flutter peut être créé en combinant un ou plusieurs widgets. Flutter fournit une vaste gamme de widgets avec des fonctionnalités de mise en page qui peuvent être utilisées pour connecter de nombreux widgets en un seul widget. Le widget enfant, par exemple, peut être centré à l'aide du widget Center.
- Certains des widgets fournis par Flutter ont pour but de présenter l'interface utilisateur qui n'est généralement pas tout à fait visible. Ces widgets incluent **Row**, **Column**, **Stack**, **Expanded**, **Padding**, **Center**, **Container**, **SizedBox**, etc.
- Les widgets de mises en page dans Flutter vous aident à définir la structure de l'interface utilisateur de votre application. Certaines de ces mises en page sont des mises en page enfant uniques, tandis que d'autres sont des mises en page enfants multiples.

3. Types de mise en page

- En effet, il existe plusieurs types de mises en page (layouts) dans le développement d'applications Android, chacun ayant ses propres caractéristiques et avantages. Voici quelques-uns des types de mises en page les plus couramment utilisés :

- **LinearLayout** : Cette mise en page organise les éléments de manière linéaire, soit horizontalement, soit verticalement. Il est utile lorsque vous souhaitez empiler des éléments les uns en dessous des autres ou les aligner côte à côte.
- **RelativeLayout** : Avec cette mise en page, vous spécifiez les relations entre les éléments en fonction de leur position les uns par rapport aux autres ou par rapport au parent. Cela offre une grande flexibilité pour la conception d'interfaces utilisateur complexes.
- **FrameLayout** : Il s'agit d'une mise en page simple qui empile les éléments les uns sur les autres. Il est souvent utilisé pour afficher un seul élément à la fois, par exemple une seule image ou une seule vue.
- **ConstraintLayout** : C'est une mise en page puissante qui permet de définir des relations complexes entre les éléments, tout en garantissant que l'interface utilisateur s'adapte à différentes tailles d'écran. C'est particulièrement utile pour la conception d'applications multiplateformes.
- **TableLayout** : Cette mise en page organise les éléments dans un tableau à deux dimensions de lignes et de colonnes. Il est principalement utilisé lorsque vous avez besoin de disposer des éléments dans une structure de grille.
- **GridLayout** : Similaire à TableLayout, GridLayout organise les éléments dans une grille, mais il offre plus de contrôle sur la disposition des éléments. Vous pouvez spécifier le nombre de lignes et de colonnes ainsi que la taille des cellules.

- **ScrollView** : Cette mise en page permet aux éléments enfants de défiler lorsque l'espace disponible est insuffisant. C'est utile lorsque vous avez du contenu dépassant l'écran.
- **DrawerLayout** : Il est utilisé pour créer un tiroir de navigation (drawer) qui peut être glissé depuis le bord de l'écran pour révéler des options de navigation ou de menu.
- **ViewPager** : Cette mise en page est spécialement conçue pour créer des interfaces utilisateur basées sur un système de navigation par onglets ou de diapositives. Elle est couramment utilisée dans les applications de type "Swiping" pour basculer entre différentes vues.
- **FragmentLayout** : Les fragments sont des éléments d'interface utilisateur réutilisables dans une activité. Vous pouvez utiliser cette mise en page pour organiser des fragments dans une activité.
- Ces types de mises en page offrent une flexibilité considérable lors de la création de l'interface utilisateur d'une application Android.
- Le choix de la mise en page dépendra des besoins spécifiques de votre application et de la manière dont vous souhaitez organiser et afficher vos éléments à l'écran.

4. Type de widgets de mise en page

- Les widgets de mise en page peuvent être regroupés en deux catégories distinctes en fonction de leur enfant –
- Widget prenant en charge un seul **child**
- Widget prenant en charge plusieurs **child**s

Widget prenant en charge un seul child (Single Childwidgets)

- Dans cette catégorie, les widgets n'auront qu'un seul widget comme **child** et chaque widget aura une fonctionnalité de mise en page spéciale.
- Les widgets prenant en charge un seul **child** sont les widgets de base et utiles, comme les `Containers`, `GridView` pour n'en nommer que quelques-uns.
- Les `Containers` sont utilisés lorsque nous devons faire du rembourrage, de la marge ou de la bordure. Ce widget ne peut spécifiquement avoir qu'un seul widget enfant sous lui.
- Les widgets de mise en page à **child** unique sont ceux qui n'accepteront qu'un seul widget comme **child** Tels que : `Container()`, `Center()`, `Expanded()`, `Align()`, `SizedBox()`, `Padding()`, `FittedBox()`, `AspectRatio()` , `Baseline()` , `ConstrainedBox()` , `CustomSingleChildLayout()` , `FittedBox()` ,`FittedBox()` ,`FractionallySizedBox()` ,`IntrinsicHeight()` , `IntrinsicWidth()` ,`LimitedBox()` ,`Offstage()` , `OverflowBox()` etc.
- *Le widget `container` dans Flutter*
- Dans **Flutter**, `Container` est une boîte utilisée pour contenir un widget enfant. En même temps, il est possible de définir son style grâce à des propriétés telles que **padding**, **margin**, **alignment**, etc.
- *Le widget `center` dans Flutter*
- Le widget `Center` de Flutter est un widget pratique pour amener n'importe quel widget au centre.
- Le `Center` prend d'abord tout l'espace disponible (tant que son parent le permet) puis place le widget enfant au centre à la fois verticalement et horizontalement.

Widget prenant en charge plusieurs childs (Multi-Child Layout Widget)

- Les widgets de mise en page multi-enfants (Multi-Child) sont ceux qui accepteront plus d'un widget comme child. Tels que : `Column()`, `Row()`, `Stack()`, `GridView()`, `ListView()`, `Table()` etc.

5. Disposition des colonnes :

- Dans notre prochain exemple, nous allons créer un conteneur (Container) avec une colonne, qui est également ce **Layouts**
- La colonne (**Column**) aura plusieurs enfants de texte de type afin que nous comprenions que toutes les mises en page peuvent Il a un nombre illimité d'enfants.

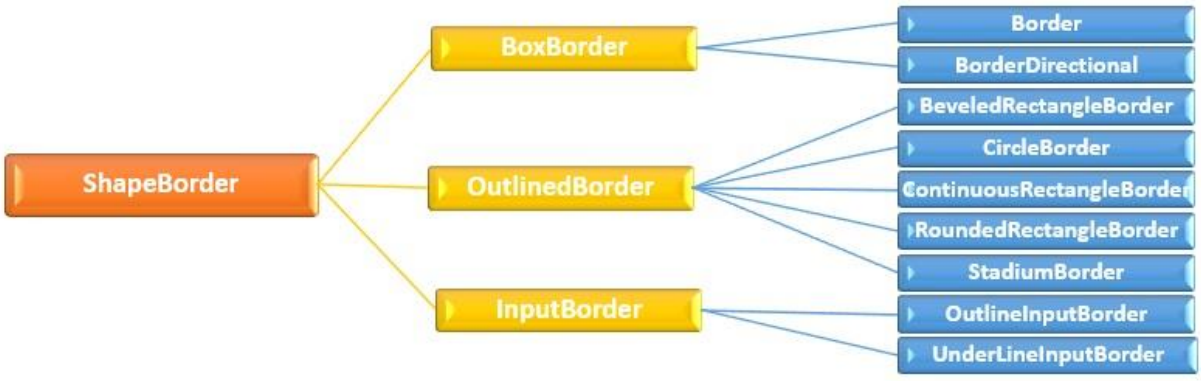
Le widget Container dans Flutter

1. Présentation

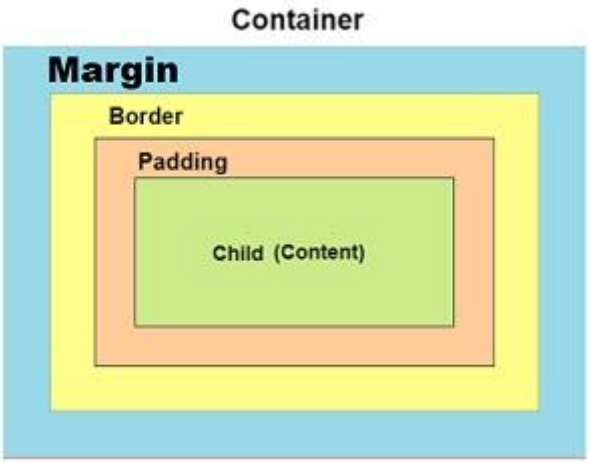
- Le Container est simplement un conteneur à taille défini dans lequel d'autres widgets peuvent venir se placer. Par défaut, sa dimension est celle de son enfant et comme il ne possède pas de couleur, il est invisible.
- En revanche, il est possible de lui attribuer une couleur grâce à la propriété **color**. Il devient alors visible.
- L'avantage du Container est qu'il est possible de lui donner un style bien précis grâce à une propriété : decoration. Elle peut accepter des widgets **BoxDecoration** ou **ShapeDecoration**. Grâce à ceux-ci, il est possible de définir des bordures par exemple.
- La dimension du Container est modulable avec plusieurs propriétés. Il est possible de jouer sur le padding qui correspond à la marge interne. De la même façon, il faudra utiliser `margin` pour gérer l'écart du container vis-à-vis des widgets qui l'entourent.
- Enfin, il dispose de deux propriétés essentielles : `width` et `height`. Elles permettent de définir une taille pour la largeur et la hauteur.

2. Propriétés du widget `Container` dans Flutter

- `child` : Cette propriété est utilisée pour stocker le widget child du conteneur.
- `color` : Cette propriété est utilisée pour définir la couleur de fond du texte . Cela change également la couleur d'arrière-plan de l'ensemble du conteneur.
- `height` et `width`: Cette propriété est utilisée pour définir la hauteur et la largeur du container
- `margin`: Cette propriété est utilisée pour entourer l' espace vide autour du conteneur .
- `padding`: cette propriété est utilisée pour définir la distance entre la bordure du conteneur (les quatre directions) et son widget **child**.
- `alignment`: cette propriété est utilisée pour définir la position de l'enfant dans le container.
- `Decoration`: Cette propriété permet d'ajouter une décoration sur le widget.
- `transform` : La propriété transform permet aux développeurs de faire pivoter le conteneur .
- `Constraints:` Cette propriété est utilisée pour ajouter des contraintes supplémentaires au **child**.
- `ClipBehaviour`: cette propriété prend Clip Enum comme objet. Cela décide si le contenu à l'intérieur du conteneur sera coupé ou non.
- `Foreground Decoration`: ce paramètre contient la classe Décoration en tant qu'objet. Il contrôle la décoration devant le widget Container.



3. Utilisation de container dans Flutter



- Dans **Flutter**, `Container` est une boîte utilisée pour contenir un widget enfant. En même temps, il est possible de définir son style grâce à des propriétés telles que **padding**, **margin**, **alignment**, etc.
- **Container** surligne le contenu ou le sépare des autres.
- Il y a beaucoup de paramètres impliqués dans la création de Container, tels que width, height, child, alignment, etc.
- Tout comme la balise ** ** en html, si le widget conteneur flutter ne contient aucun widget enfant, il remplira toute la zone de l'écran. D'un autre côté, s'il y a des widgets enfants, le widget conteneur les enveloppera en fonction de la hauteur et de la largeur spécifiées.
  Le Widget container ne doit pas s'afficher directement sans aucun widget parent. Vous pouvez utiliser le widget **Center**, le widget **Padding**, le widget **Column**, le widget **Row** ou le widget **Scaffold** en tant que parent.

- **Constructeurs**

```
Container({Key key,
  AlignementGeometry alignement,
  EdgeInsetsGeometry padding,
  Color color,
  double width,
  double height,
  Decoration decoration,
  Decoration foregroundDecoration,
  boxConstraints constraints,
  Widget child,
  Clip clipBehaviour: Clio.none
})
```

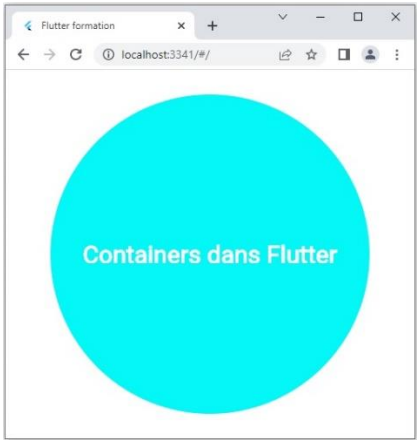
- **Exemple**

  Code de l'exemple :

```
import 'package:flutter/material.dart';
void main() => runApp(const MyApp());
class MyApp extends StatelessWidget {
```

```
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Flutter formation',
      debugShowCheckedModeBanner: false,
      home: Home(),
    );
  }
}
class Home extends StatelessWidget {
  const Home({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Container(
        width: 300,
        height: 300,
        alignment: Alignment.center,
        padding: const EdgeInsets.all(20),
        margin: const EdgeInsets.symmetric(horizontal: 50, vertical:
30),
        decoration: const BoxDecoration(
          color: Color.fromARGB(255, 4, 248, 248),
          shape: BoxShape.circle,
          //borderRadius: BorderRadius.circular(25)
        ),
        child: const Text(
          'Containers dans Flutter',
          style: TextStyle(
            fontSize: 30,
            fontWeight: FontWeight.bold,
            color: Colors.white,
          ),
        ));
  }
}
```

- **Résultat**



- Ajouter des bordures au widget `container` dans Flutter
- **Exemple: 01**

```
class Home extends StatelessWidget {
  const Home({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Container(
      height: 10,
      width: 10,
      decoration: BoxDecoration(
          shape: BoxShape.circle,
          border: Border.all(width: 20, color: Colors.blueAccent)), // BoxDecoration
    ); // Container
  }
}
```

- **Exemple: 02**

```
    return Container(
        width: 150,
        height: 300,
        decoration: const ShapeDecoration(
            color: Colors.white,
            shape:
                CircleBorder(side: BorderSide(width: 20,
                color: Colors.blue), // BorderSide
                )), // CircleBorder // ShapeDecoration
        child: const Center(child: Text("07",
        style: TextStyle(fontSize: 80)))); // Text // Center // Container
```

- Utilisez l'opérateur d'addition (+) pour ajouter plusieurs **ShapeBorder** (s) afin de créer une bordure associative.

```
return Container(
    width: 100,
    height: 200,
    child: Center(child: Text("ok", style: TextStyle(fontSize: 50))),
    decoration: ShapeDecoration(
        color: Colors.white,
        shape: const CircleBorder(
                side: BorderSide(width: 20, color: Colors.blue)) + //
            const CircleBorder(
                side: BorderSide(
                    width: 20, color: Color.fromRGBO(167, 211, 97, 1)))
            const CircleBorder(
                side: BorderSide(width: 20, color: Color(0xFF7F4CAF))) +
            const CircleBorder(
                side: BorderSide(
                    width: 20, color: Color.fromARGB(255, 224, 205, 26)) )}
```

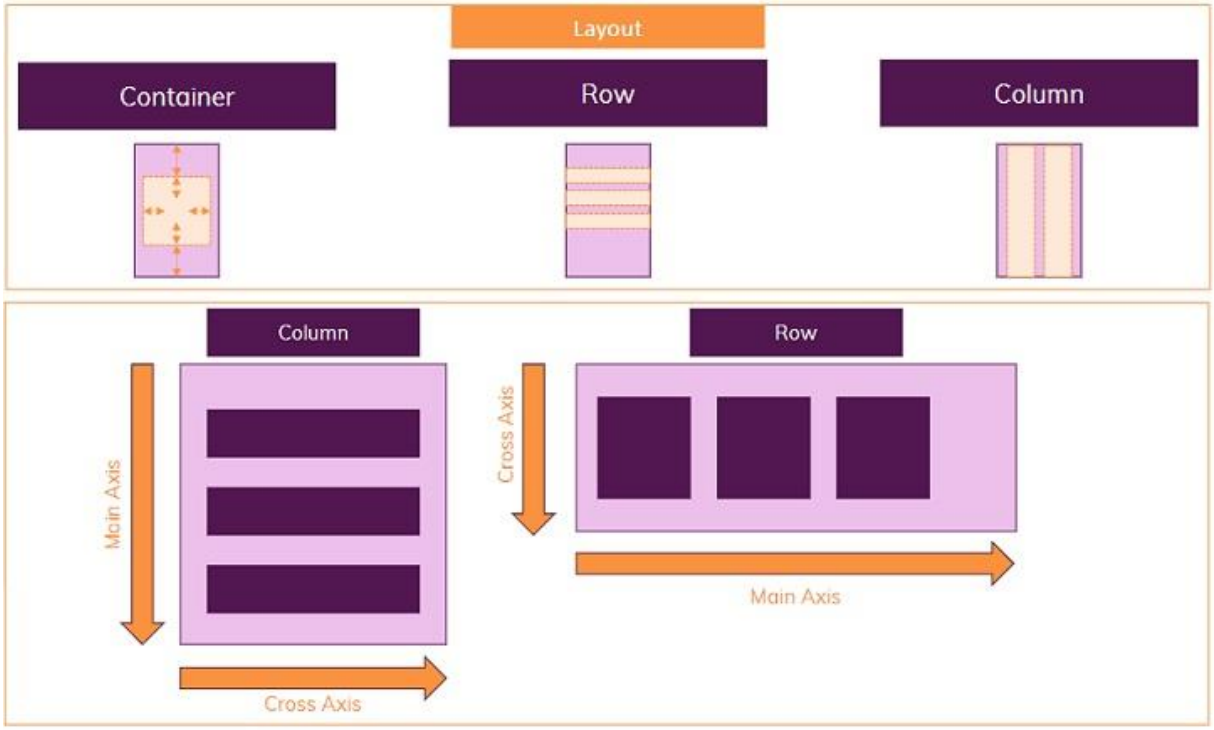
Row et Column dans Flutter

1. Objectif

- Apprendre à organiser les widgets en lignes et en colonnes sur l'écran en Flutter

2. Présentation

- **Row** et **Column** sont des widgets dans Flutter et la plupart du temps, chaque mise en page peut être décomposée en ligne et en colonne.
- La **ligne** et la **colonne** sont les deux widgets essentiels de Flutter qui permet aux développeurs d'aligner les **child** horizontalement et verticalement selon nos besoins.
- Ces widgets sont très nécessaires lorsque nous concevons l'interface utilisateur de l'application dans Flutter.



3. Le widget Row en Flutter

- **Utilisation**

- Une ligne est un widget de mise en page multi-enfant qui prend une liste de widgets comme ses enfants.
- Une ligne affiche les widgets dans la vue visible, c'est-à-dire qu'elle ne défile pas.
- Nous pouvons contrôler la façon dont un **widget de ligne** aligne ses enfants en fonction de notre choix en utilisant la propriété `crossAxisAlignment` et `mainAxisAlignment` .
- L'axe transversal de la ligne s'exécutera verticalement , et l'axe principal s'exécutera horizontalement .
- Voici comment vous pouvez utiliser le widget Row :

```
Row(
  children: <Widget>[
    // Listez ici les widgets que vous souhaitez
    // disposer horizontalement
    WidgetEnfant1(),
    WidgetEnfant2(),
    WidgetEnfant3(),
    // ...
  ],
)
```

- **Définir l'alignement de l'axe transversal**

- `crossAxisAlignmentargument`: L'alignement des 'child' dans l'axe transversal peut être défini en passant un `crossAxisAlignmentargument`. Si vous ne transmettez pas l'argument, l'alignement par défaut de l'axe transversal est centré. Les valeurs possibles sont :
- `start`: Placez les enfants aussi près que possible du début de l'axe transversal.
- `end`: Placez les enfants le plus près possible de l'extrémité de l'axe transversal.
- `center`: Placez les enfants de manière à ce que leurs centres soient alignés avec le milieu de l'axe transversal.
- `stretch`: Demander aux enfants de remplir l'axe transversal.
- `baseline`: placez les enfants le long de l'axe transversal de sorte que leurs lignes de base correspondent.
- **`mainAxisAlignment`**
- Le `mainAxisAlignment` contrôle les 'child' à afficher dans la direction **horizontale** de la ligne '**Row**', mais contrôle la direction verticale du Colonne '**column**'

- **Propriétés**

- Voici quelques propriétés couramment utilisées avec le widget Row :
- **`children`**: Une liste de widgets que vous souhaitez disposer horizontalement. Chaque élément de cette liste représente un widget enfant à afficher dans la rangée.
- **`mainAxisAlignment`**: Cette propriété vous permet de définir l'alignement principal des enfants dans la rangée. Par défaut, la valeur est MainAxisAlignment.start, ce qui signifie que les enfants seront alignés à gauche. Vous pouvez également utiliser MainAxisAlignment.center pour les aligner au centre ou MainAxisAlignment.end pour les aligner à droite, entre autres.
- **`crossAxisAlignment`**: Cette propriété contrôle l'alignement transversal des enfants dans la rangée. Par défaut, la valeur est CrossAxisAlignment.center, ce qui signifie que les enfants seront centrés verticalement. Vous pouvez également utiliser CrossAxisAlignment.start pour les aligner en haut ou CrossAxisAlignment.end pour les aligner en bas, par exemple.
- **`mainAxisSize`**: Cette propriété détermine la taille principale de la rangée en fonction de la somme des tailles principales de ses enfants. Par défaut, la valeur est

MainAxisSize.max, ce qui signifie que la rangée s'étendra pour occuper tout l'espace disponible dans la direction principale. Vous pouvez également utiliser MainAxisSize.min pour que la rangée ait la taille minimale possible en fonction de son contenu

- **Exemple**

```
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  crossAxisAlignment: CrossAxisAlignment.center,
  children: <Widget>[
    Text('Widget Enfant 1'),
    Text('Widget Enfant 2'),
    RaisedButton(
      onPressed: () {
        // Action à effectuer lorsque le bouton est pressé
      },
      child: Text('Bouton Enfant'),
    ),
  ],
)
```

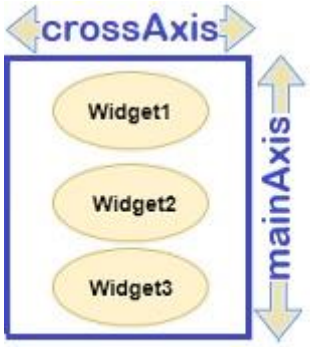
- Dans cet exemple, trois widgets enfants (deux Text et un RaisedButton) sont empilés verticalement dans la colonne, centrés à la fois horizontalement et verticalement en raison des propriétés mainAxisAlignment et crossAxisAlignment définies.

4. Le widget Column en Flutter

- **Utilisation**

- Le widget Column en Flutter est utilisé pour créer une disposition verticale d'enfants, ce qui signifie que les widgets enfants sont empilés les uns en dessous des autres.
- C'est l'une des manières de créer des mises en page verticales dans une application Flutter.
- Le widget de colonne comme la ligne est un widget de mise en page multi-enfant qui prend une liste de widgets comme ses enfants.
- Voici comment vous pouvez utiliser le widget Column :

```
Column(
  children: <Widget>[
    // Listez ici les widgets que vous souhaitez empiler
    //verticalement
    WidgetEnfant1(),
    WidgetEnfant2(),
    WidgetEnfant3(),
    // ...
  ],
)
```



- **Propriétés**

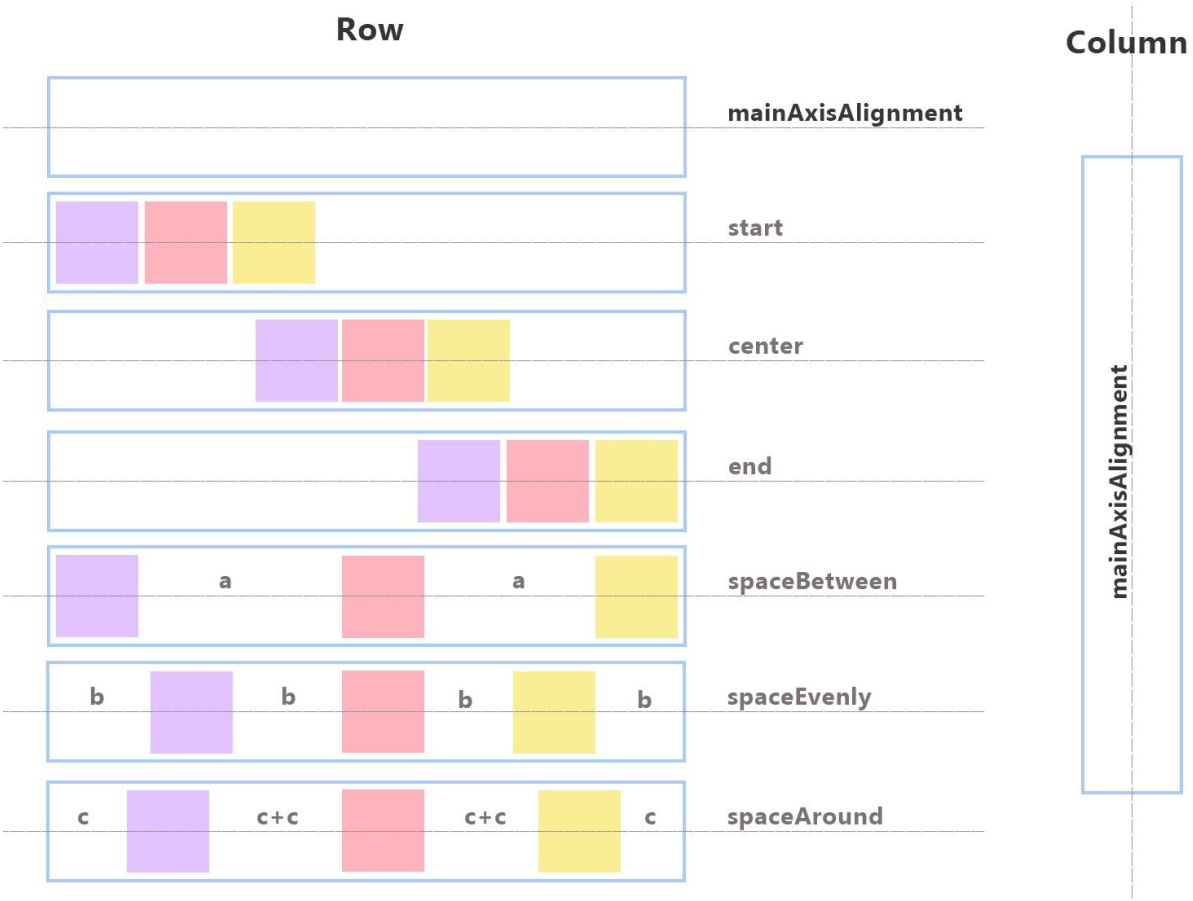
- Voici quelques propriétés couramment utilisées avec le widget Column :
- `children`: Une liste de widgets que vous souhaitez empiler verticalement. Chaque élément de cette liste représente un widget enfant à afficher dans la colonne.
- `mainAxisAlignment`: Cette propriété vous permet de définir l'alignement principal des enfants dans la colonne. Par défaut, la valeur est MainAxisAlignment.start, ce qui signifie que les enfants seront alignés à gauche. Vous pouvez également utiliser MainAxisAlignment.center pour les aligner au centre ou MainAxisAlignment.end pour les aligner à droite, entre autres.
- `crossAxisAlignment`: Cette propriété contrôle l'alignement transversal des enfants dans la colonne. Par défaut, la valeur est CrossAxisAlignment.center, ce qui signifie que les enfants seront centrés horizontalement. Vous pouvez également utiliser CrossAxisAlignment.start pour les aligner à gauche ou CrossAxisAlignment.end pour les aligner à droite, par exemple.
- `mainAxisSize`: Cette propriété détermine la taille principale de la colonne en fonction de la somme des tailles principales de ses enfants. Par défaut, la valeur est MainAxisSize.max, ce qui signifie que la colonne s'étendra pour occuper tout l'espace disponible dans la direction principale. Vous pouvez également utiliser MainAxisSize.min pour que la colonne ait la taille minimale possible en fonction de son contenu.

- **Exemple**

```
Column(
  mainAxisAlignment: MainAxisAlignment.center,
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
    Text('Widget Enfant 1'),
    Text('Widget Enfant 2'),
    RaisedButton(
      onPressed: () {
        // Action à effectuer lorsque le bouton est pressé
      },
      child: Text('Bouton Enfant'),
    ),
  ],
)
```

- Dans cet exemple, trois widgets enfants (deux Text et un RaisedButton) sont empilés verticalement dans la colonne, centrés à la fois horizontalement et verticalement en raison des propriétés mainAxisAlignment et crossAxisAlignment définies.

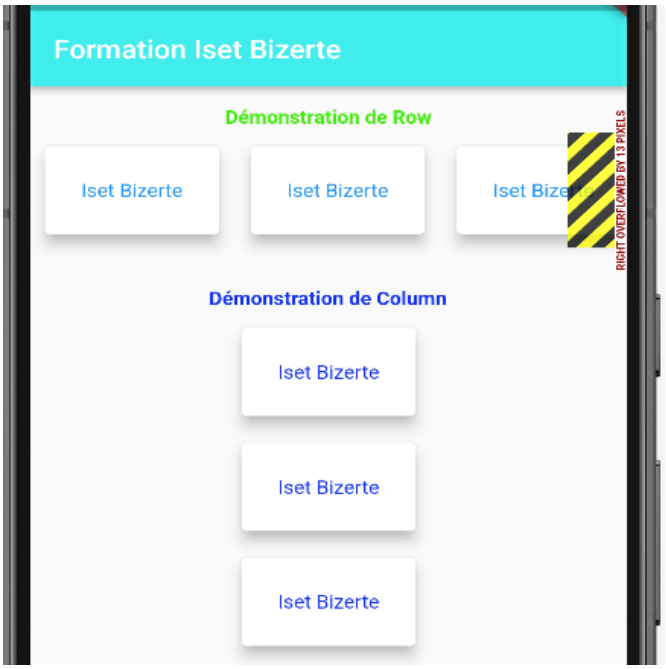
5. Comparaison entre Row et Column



Container	Colonne/Ligne
Prend exactement un widget enfant	Prend plusieurs widgets enfants (illimités)
Options d'alignement et de style riches disponibles	Options d'alignement disponibles, mais il n'y a pas d'options de style
Largeur flexible (ex. largeur enfant, largeur disponible, …)	Prend toujours la pleine hauteur (colonne) / largeur (ligne) disponible
Parfait pour un style et un alignement personnalisés	Indispensable si les widgets sont placés les uns à côté des autres

6. Applications

- **Exercice 0: Création d'une mise en page simple avec `Row` et `Column`**
  - Utiliser les widgets **row** et **column** pour réaliser la figure suivante:



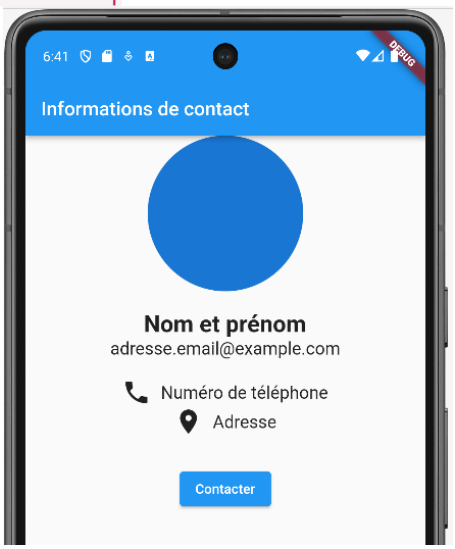

```dart
import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: Home(),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Formation Flutter'),
        backgroundColor: const Color.fromARGB(190, 24, 255, 255),
      ),
      body: Column(
        //Tous les éléments sont emballés
        children: [
          //Dans cette colonne
          const SizedBox(
            height: 15,
          ),
          const SizedBox(
            height: 20,
            child: Text('Démonstration de Row',
```

```
            style: TextStyle(
                color: Color.fromRGBO(69, 235, 18, 1),
                fontWeight: FontWeight.bold)),
      ),
      Row(
        children: [
          ////////////Premier élément
          Card(
            margin: const EdgeInsets.all(10),
            elevation: 8,
            child: Container(
              padding: const EdgeInsets.all(25),
              child: const Text(
                'Iset Bizerte',
                style: TextStyle(color: Colors.blue),
              ),
            ),
          ),
          const SizedBox(
            width: 2,
          ),
          ////////////////Deuxsième élément
          Card(
            margin: const EdgeInsets.all(10),
            elevation: 8,
            child: Container(
              padding: const EdgeInsets.all(25),
              child: const Text(
                'Iset Bizerte',
                style: TextStyle(color: Colors.blue),
              ),
            ),
          ),
          const SizedBox(
            width: 2,
          ),
          ////////////////Troisième élément
          Card(
            margin: const EdgeInsets.all(10),
            elevation: 8,
            child: Container(
              padding: const EdgeInsets.all(25),
              child: const Text(
                'Iset Bizerte',
                style: TextStyle(color: Colors.blue),
              ),
            ),
          ),
        ],
      ),
      const SizedBox(
        height: 30,
      ),
      const SizedBox(
        height: 20,
        child: Text(
          'Démonstration de Column',
          style: TextStyle(
              color: Color.fromARGB(255, 18, 50, 235),
```

```
                  fontWeight: FontWeight.bold),
            ),
          ),
          Column(
            children: [
              //////////////Premier élément
              Card(
                margin: const EdgeInsets.all(10),
                elevation: 8,
                child: Container(
                  padding: const EdgeInsets.all(25),
                  child: const Text(
                    'Iset Bizerte',
                    style: TextStyle(color: Color.fromARGB(255, 18, 50, 235)),
                  ),
                ),
              ),
              const SizedBox(
                width: 4,
              ),
              //////////////////Deuxsième élément
              Card(
                margin: const EdgeInsets.all(10),
                elevation: 8,
                child: Container(
                  padding: const EdgeInsets.all(25),
                  child: const Text(
                    'apcpedagogie',
                    style: TextStyle(color: Color.fromARGB(255, 18, 50, 235)),
                  ),
                ),
              ),
              //////////////////Troisième élément
              Card(
                margin: const EdgeInsets.all(10),
                elevation: 8,
                child: Container(
                  padding: const EdgeInsets.all(25),
                  child: const Text(
                    'apcpedagogie',
                    style: TextStyle(color: Color.fromARGB(255, 18, 50, 235)),
                  ),
                ),
              ),
            ],
          )
        ],
      ),
    );
  }
}
```

- **Exercice 02: Création d'une mise en page complexe avec Row et Column**
  - **Objectif :** Créer une mise en page complexe qui comprend plusieurs widgets Row et Column pour afficher des informations de contact. La mise en page doit ressembler à ceci :

```
|-------------------------------|
| Photo de profil |
|-------------------------------|
| Nom et prénom |
| Adresse email |
|-------------------------------|
| Numéro de téléphone |
| Adresse |
|-------------------------------|
| Bouton de contact |
|-------------------------------|
```



- Créez une nouvelle application Flutter.
- Utilisez les widgets Row et Column pour organiser les éléments de l'interface utilisateur conformément au schéma ci-dessus.
- Personnalisez l'apparence des éléments pour qu'ils aient l'apparence souhaitée.
- Ajoutez un bouton de contact qui affiche une boîte de dialogue lorsque vous appuyez dessus avec un message comme "Contacter cette personne".
- Assurez-vous que la mise en page est réactive et fonctionne bien sur différents appareils.

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: MyContactPage(),
    );
  }
}

```

```dart
class MyContactPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Informations de contact'),
      ),
      body: Center(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            CircleAvatar(
              radius: 80,
              backgroundImage:
NetworkImage('URL_de_votre_photo_de_profil'),
            ),
            SizedBox(height: 20),
            Text(
              'Nom et prénom',
              style: TextStyle(fontSize: 24, fontWeight:
FontWeight.bold),
            ),
            Text(
              'adresse.email@example.com',
              style: TextStyle(fontSize: 18),
            ),
            SizedBox(height: 20),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Icon(Icons.phone, size: 30),
                SizedBox(width: 10),
                Text(
                  'Numéro de téléphone',
                  style: TextStyle(fontSize: 18),
                ),
              ],
            ),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Icon(Icons.location_on, size: 30),
                SizedBox(width: 10),
                Text(
                  'Adresse',
                  style: TextStyle(fontSize: 18),
                ),
              ],
            ),
            SizedBox(height: 30),
            ElevatedButton(
              onPressed: () {
```

```
72.                   showDialog(
73.                     context: context,
74.                     builder: (context) {
75.                       return AlertDialog(
76.                         title: Text('Contact'),
77.                         content: Text('Contacter cette personne'),
78.                         actions: [
79.                           TextButton(
80.                             onPressed: () {
81.                               Navigator.of(context).pop();
82.                             },
83.                             child: Text('Fermer'),
84.                           ),
85.                         ],
86.                       );
87.                     },
88.                   );
89.                 },
90.                 child: Text('Contacter'),
91.               ),
92.             ],
93.           ),
94.         ),
95.       );
96.     }
97.   }
```