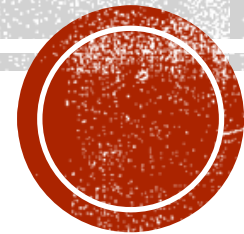



FLUTTER

Formation flutter Avancée

Union Of Education

Formateur :Kamel.bahmed.info@gmail.com





```

void main() {
  SystemChrome.setSystemUIOverlayStyle(
    const SystemUiOverlayStyle(statusBarColor: Colors.transparent));
  runApp(MyApp());
}

```

- **MyApp Class :**
 - MyApp étend StatelessWidget, ce qui signifie que c'est un widget dont l'interface utilisateur ne change pas en cours d'exécution.
 - Le constructeur de MyApp utilise super.key pour appeler le constructeur de la classe mère (StatelessWidget).
- **build Method :**
 - La méthode build est une méthode obligatoire pour tous les widgets Flutter. Elle est appelée à chaque fois que le widget doit être rendu à l'écran.
 - Le BuildContext est un objet qui détient des informations sur l'emplacement actuel dans le widget tree. Il est utilisé pour rechercher des informations sur la configuration du thème, la localisation (traductions), et pour naviguer vers un autre widget.
 - À l'intérieur de la méthode build, on retourne un widget MaterialApp. MaterialApp est un widget qui met en place le cadre de base pour une application basée sur le design "Material" de Google.
- **debugShowCheckedModeBanner :**
 - debugShowCheckedModeBanner est utilisé pour contrôler l'affichage ou non de la bannière "DEBUG" dans l'angle supérieur droit de l'application lorsque l'application est en mode debug. Ici, il est désactivé (false), ce qui signifie que la bannière DEBUG ne sera pas affichée.
- **theme :**
 - theme est utilisé pour définir le thème global de l'application. Dans cet exemple, seule la couleur de fond du scaffold (structure de base de l'interface utilisateur) est définie sur Colors.white.
- **routes :**
 - routes est un moyen de définir les itinéraires de navigation dans l'application. Dans cet exemple, une seule route est définie, où la route "/" (racine) mène au widget Home.

- **SystemChrome.setSystemUIOverlayStyle :**
- Cette méthode est utilisée pour définir le style de l'interface utilisateur du système. Dans ce cas, elle est utilisée pour spécifier que la barre d'état (status bar) doit être transparente. Cela affecte l'apparence de la barre d'état en haut de l'écran, généralement où se trouvent l'heure, la batterie, etc.
- SystemUiOverlayStyle est une classe qui prend plusieurs paramètres pour personnaliser l'apparence de l'interface utilisateur du système. Ici, statusBarColor est défini sur Colors.transparent pour rendre la barre d'état transparente.



```

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        scaffoldBackgroundColor: Colors.white,
      ),
      routes: {
        "/": (context) => Home(),
      },
    );
  }
}

```



```

return Container(
  padding: const EdgeInsets.all(25),
  child: Row(
    children: [
      const Icon(
        Icons.sort,
        size: 30,
        color: Color(0xFF4C53A5),
      ),
      const Padding(
        padding: EdgeInsets.only(
          left: 20,
        ),
        child: Text(
          "IdooMarket",
          style: TextStyle(
            fontSize: 23,
            fontWeight: FontWeight.bold,
            color: Color(0xFF4C53A5),
          ),
        ),
      ),
      const Spacer(),
      Badge(
        label: const Text('2'),
        child: InkWell(
          onTap: () {},
          child: const Icon(
            Icons.shopping_bag_outlined,
            size: 30,
            color: Color(0xFF4C53A5),
          ),
        ),
      ),
    ],
  ),
);

```

- Container :
 - Le widget Container est utilisé pour définir une boîte rectangulaire dans laquelle d'autres widgets peuvent être placés. Ici, il est utilisé pour envelopper toute la rangée d'éléments.
 - padding spécifie la marge intérieure du conteneur. Dans cet exemple, il y a une marge de 25 unités sur tous les côtés du conteneur.
- L'icône utilisée est celle du tri (Icons.sort).
- Padding :
 - Le widget Padding est utilisé pour ajouter de l'espace autour de son enfant.
- Spacer :
 - Le widget Spacer est utilisé pour remplir l'espace disponible dans une Row.

En résumé, ce morceau de code crée une barre de navigation ou une barre d'en-tête simple avec une icône de tri à gauche, le texte "IdooMarket" au centre, un badge avec une icône de sac de shopping à droite, et un espace élastique entre le texte et le badge.





```
Expanded(  
  child: ListView(  
    children: [  
      Column(  
        children: [  
          Container(  
            padding: EdgeInsets.symmetric(vertical: 20),  
            decoration: BoxDecoration(  
              color: Color(0xFFEDEC2),  
              borderRadius: BorderRadius.only(  
                topLeft: Radius.circular(35),  
                topRight: Radius.circular(35),  
              ),  
            ),  
          ),  
        ],  
      ),  
    ],  
  ),  
),
```


- Le widget Expanded est un conteneur utilisé dans Flutter pour prendre tout l'espace disponible dans une Row, Column ou Flex, selon l'axe spécifié (horizontal pour Row et vertical pour Column). Son objectif principal est de permettre à un widget de remplir l'espace disponible dans le sens spécifié, ce qui est particulièrement utile lorsque tu veux que certains éléments prennent plus d'espace que d'autres.
- Le widget ListView est utilisé pour afficher une liste d'éléments défilables. Il est particulièrement utile lorsqu'on a besoin d'afficher un grand nombre d'éléments ou une liste dynamique d'éléments.



```
Container(  
  margin: EdgeInsets.symmetric(horizontal: 15),  
  padding: EdgeInsets.symmetric(horizontal: 15),  
  height: 50,  
  decoration: BoxDecoration(  
    color: Colors.white,  
    borderRadius: BorderRadius.circular(30),  
  ),  
),
```

- Dans Flutter, la propriété decoration du widget Container permet de définir la décoration visuelle du conteneur. La classe utilisée pour spécifier la décoration est BoxDecoration. L'objet BoxDecoration offre plusieurs options pour personnaliser l'apparence du conteneur





```
CurvedNavigationBar navigationBar() {  
  return CurvedNavigationBar(  
    onTap: (index) {},  
    backgroundColor: Colors.transparent,  
    color: const Color(0xFF4C53A5),  
    height: 70,  
    items: const [  
      Icon(  
        Icons.home,  
        size: 30,  
        color: Colors.white,  
      ),  
      Icon(  
        CupertinoIcons.cart_fill,  
        size: 30,  
        color: Colors.white,  
      ),  
      Icon(  
        Icons.list,  
        size: 30,  
        color: Colors.white,  
      ),  
    ],  
  );  
}
```

- CurvedNavigationBar :
 - CurvedNavigationBar est un widget personnalisé souvent utilisé pour créer des barres de navigation avec un effet de courbure en bas. Il est généralement utilisé comme une alternative stylisée à la barre de navigation standard de Flutter.
 - Pour l'ajouter: on ajoute
`curved_navigation_bar: ^1.0.3`
Dans pubspec.yaml



```

SingleChildScrollView(
  scrollDirection: Axis.horizontal,
  child: Row(
    children: [
      for (int i = 0; i < categories.length; i++)
        Container(
          margin: const EdgeInsets.symmetric(horizontal: 10),
          width: 140,
          padding:
            const EdgeInsets.symmetric(horizontal: 10, vertical: 5),
          decoration: BoxDecoration(
            color: Colors.white,
            borderRadius: BorderRadius.circular(20),
          ),
          child: Row(
            crossAxisAlignment: CrossAxisAlignment.center,
            mainAxisAlignment: MainAxisAlignment.spaceAround,
            children: [
              Image.asset(
                "assets/images/dsl-x1852e.jpg",
                width: 30,
                height: 30,
              ),
              Text(
                categories[i],
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                  fontSize: 17,
                  color: Color(0xFF4C53A5),
                ),
              ),
            ],
          ),
        ),
    ],
  ),
),

```

- **SingleChildScrollView :**
 - Le widget SingleChildScrollView permet de créer une vue défilable lorsqu'il y a un dépassement de l'espace disponible. Dans ce cas, il est utilisé pour permettre le défilement horizontal des catégories.
- **scrollDirection: Axis.horizontal** spécifie que le défilement doit être horizontal.
- **Boucle for** pour créer les catégories :
 - Une boucle for est utilisée pour créer dynamiquement les widgets de catégories en fonction de la liste categories.
- **Image.asset :**
 - Le widget Image.asset est utilisé pour afficher une image provenant du répertoire des ressources de l'application.



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

  @override
  State<HomePage> createState() => _HomePageState();
}
```

```
Widget build(BuildContext context) {
  return GridView.builder(
    gridDelegate: const SliverGridDelegateWithMaxCrossAxisExtent(
      maxCrossAxisExtent: 200,
      childAspectRatio: 0.65,
      crossAxisSpacing: 10,
      mainAxisSpacing: 10,
    ),
    physics: const NeverScrollableScrollPhysics(),
    itemCount: widget.products.length,
    shrinkWrap: true,
    itemBuilder: (BuildContext ctx, index) {
      return item(context, index);
    },
  );
}
```

shrinkWrap :

shrinkWrap est défini sur true, ce qui signifie que la hauteur de la grille sera ajustée en fonction du nombre d'éléments, et la grille ne prendra que l'espace nécessaire.

itemBuilder :

itemBuilder est une fonction de rappel qui est appelée pour construire chaque élément de la grille.

- La classe HomePage est déclarée en tant que widget StatefulWidget en étendant la classe StatefulWidget
- createState Method :
 - La méthode createState est une méthode de la classe StatefulWidget qui est responsable de créer l'objet State associé à ce widget.
- GridView.builder :
 - GridView.builder est utilisé pour créer une grille d'enfants basée sur une liste d'éléments. Il est plus efficace que GridView lorsqu'il s'agit de grandes listes ou de listes générées dynamiquement.
- SliverGridDelegateWithMaxCrossAxisExtent :
 - SliverGridDelegateWithMaxCrossAxisExtent est utilisé pour définir la disposition de la grille.
 - maxCrossAxisExtent définit la largeur maximale d'un élément dans la grille.
 - childAspectRatio définit le rapport hauteur/largeur d'un élément dans la grille.
 - crossAxisSpacing définit l'espace entre les éléments dans la direction transversale (horizontale dans ce cas).
 - mainAxisSpacing définit l'espace entre les éléments dans la direction principale (verticale dans ce cas).
 - physics désactive le défilement dans la grille en utilisant NeverScrollableScrollPhysics(). Cela signifie que la grille ne peut pas être défilée par l'itemCount :
 - itemCount définit le nombre total d'éléments dans la grille, qui est dérivé de la longueur de la liste de produits (widget.products).length.

```

if (state is SuccessProductsList) {
  List<Product> hamouda = state.productsFromState;

  if (hamouda.isEmpty) {

    WidgetsBinding.instance.addPostFrameCallback((_) {
      Navigator.pushNamed(context, "login");
    });
  }

  return GridView.builder(
    gridDelegate: const SliverGridDelegateWithMaxCrossAxisExtent(
      maxCrossAxisExtent: 200,
      childAspectRatio: 0.6,
      crossAxisSpacing: 10,
      mainAxisSpacing: 10,
    ),
    physics: const NeverScrollableScrollPhysics(),
    itemCount: hamouda.length,
    shrinkWrap: true,
    itemBuilder: (BuildContext ctx, index) {
      return item(hamouda, index);
    },
  );
}

```

- `WidgetsBinding.instance.addPostFrameCallback((_) { ... });` :
- `addPostFrameCallback` est un callback qui est exécuté après que la frame actuelle (cycle de rendu) est terminée. Cela garantit que le code à l'intérieur du callback est exécuté une fois que le rendu de l'interface utilisateur est complet.
- Dans ce cas, lorsque la liste de produits est vide, le callback est utilisé pour déclencher la navigation vers la page de connexion (`Navigator.pushNamed(context, "login")`).



UTILISATION DE HIVE

- Qu'est-ce que Hive?
 - Hive est une bibliothèque de gestion de base de données NoSQL rapide, efficace et facile à utiliser, spécialement conçue pour le développement d'applications Flutter et Dart. Contrairement aux bases de données SQL traditionnelles, Hive est orienté objet, ce qui le rend idéal pour stocker des données natives de Dart, telles que des objets et des collections.
- 2. Pourquoi Choisir Hive?
 - Performance Élevée : Hive est optimisé pour les performances, offrant des opérations de lecture et d'écriture rapides grâce à sa conception de base de données de type boîte (box-based).
 - Simplicité d'Utilisation : Hive est conçu pour être simple et facile à utiliser, avec une syntaxe claire et concise qui permet aux développeurs de se concentrer sur le développement plutôt que sur la gestion de la base de données.
 - Compatibilité avec Flutter : Hive est spécifiquement conçu pour fonctionner en tandem avec Flutter et Dart, offrant une intégration transparente dans le cadre de développement Flutter.



PRINCIPES FONDAMENTAUX DE HIVE

- 1. Boîtes (Boxes)
 - Définition : Dans Hive, une boîte (box) est l'équivalent d'une table dans une base de données SQL. C'est un conteneur logique qui stocke des objets de manière organisée.
 - Utilisation : Les boîtes sont utilisées pour stocker des objets spécifiques à un domaine particulier de l'application, créant ainsi une isolation des données.
- 2. Types de Données Supportés
 - Données Primitives : Hive prend en charge une variété de types de données natifs, tels que les chaînes de caractères, les entiers, les doubles, les listes et les cartes.
 - Objets Personnalisés : Hive permet également de stocker des objets personnalisés en les sérialisant et en les désérialisant automatiquement.
- 3. Gestion des Adapters
 - Adapters : Pour stocker des objets personnalisés, des adaptateurs peuvent être créés pour spécifier comment les objets doivent être sérialisés et désérialisés.
 - Personnalisation : Cette approche offre une grande flexibilité en permettant aux développeurs de personnaliser la gestion des objets complexes.



