Le widget ListView dans Flutter

1. Objectifs

- Apprendre à utiliser efficacement le widget ListView dans Flutter pour afficher des listes de données de manière fluide et optimisée.

2. Présentation

- Le widget `ListView` est utilisé pour afficher une liste d'éléments dans une vue défilante.
- La `ListView` est un outil très pratique pour afficher une grande quantité d'informations à l'écran.
- ListView est une liste déroulante de widgets disposés linéairement. De plus, il affiche ses enfants les uns après les autres dans le sens du défilement.
- Voici quelques points clés à retenir :
  - **Affichage de Contenu Défilant :**
    - ListView est idéal pour afficher des listes d'éléments, qu'il s'agisse de messages, d'articles, de contacts, etc.
    - Lorsque la liste d'éléments dépasse l'espace disponible à l'écran, ListView permet le défilement fluide pour afficher tous les éléments.
  - **Types de ListView :**
  - Il existe plusieurs types de ListView en fonction de vos besoins :
    - **ListView.builder** : Utilisé pour générer dynamiquement des éléments à partir d'une source de données (par exemple, une liste d'articles).
    - **ListView.separated** : Similaire à ListView.builder, mais avec des séparateurs personnalisés entre les éléments.
    - **ListView.custom**: Pour une personnalisation avancée de la façon dont les éléments sont affichés.
  - **Personnalisation :**
    - Vous pouvez personnaliser l'apparence des éléments dans ListView en utilisant des widgets tels que ListTile, Card, Container, etc.
    - N'oubliez pas d'ajouter des interactions (comme des gestes de balayage ou des clics) pour rendre l'expérience utilisateur plus riche.
  - **Performance :**
    - ListView.builder est recommandé pour les performances optimales, car il ne crée que les éléments visibles à l'écran.

apcpedagogie.com

3. Créer une ListView dans Flutter

- Pour créer une ListView dans Flutter, vous pouvez utiliser les éléments et propriétés suivants :
  - **`children`** :
    - Contient une liste de widgets comme valeur et les organise dans un format de liste. Cet attribut est le plus important car il contient les valeurs de la liste.
    - Cet attribut est le plus couramment utilisé pour définir les éléments de la liste. Vous pouvez lui fournir une liste de widgets à afficher dans la ListView.
    - **Exemple:**



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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          backgroundColor: Colors.blueAccent,
// Couleur de fond de l'appBar
          title: const Text('Exemple ListView'),
        ),
        body: ListView(
// Utilisation de la propriété children pour
définir les éléments de la liste
          children: [
            ListTile(
              tileColor: Colors.redAccent,
// Couleur du premier ListTile
              leading: const
Icon(Icons.account_circle),
              title: const Text('John Doe'),
              subtitle: const
Text('john.doe@example.com'),
              onTap: () {
                print('Tappé sur John Doe');
              },
            ),
```

```dart
ListTile(
  tileColor: Colors.greenAccent,

  // Couleur du deuxième ListTile
              leading: const
Icon(Icons.account_circle),
              title: const Text('Jane Smith'),
              subtitle: const
Text('jane.smith@example.com'),
              onTap: () {
                print('Tappé sur Jane Smith');
              },
            ),
            ListTile(
              tileColor: Colors.orangeAccent,
// Couleur du troisième ListTile
              leading: const
Icon(Icons.account_circle),
              title: const Text('Bob Johnson'),
              subtitle: const
Text('bob.johnson@example.com'),
              onTap: () {
                print('Tappé sur Bob Johnson');
              },
            ),
 // Ajoutez plus de ListTiles selon vos besoins
          ],
        ),
      ),
    );
  }
}
```

    - **`childrenDelegate`** : cette propriété prend **SliverChildDelegate** comme objet. Et il sert de délégué qui a fourni aux enfants le **ListView**.
    - **`clipBehaviour`**: cette propriété contient Clip enum (final) comme objet. Il contrôle si le contenu de ListVew sera tronqué ou non.
    - **`itemExtent`**: l'itemExtent prend une valeur double en tant qu'objet pour contrôler la zone de défilement dans le ListView .
    - **`padding`**:
      - Il contient **EdgeInsetsGeometryI** comme objet pour donner de l'espace entre la Listview et ses enfants.
      - prend la valeur de la classe EdgetInsets. Cette propriété organise l’ensemble de ListView et lui donne une marge interne.
    - **`scrollDirection`**:

apcpedagogie.com

        - Cette propriété prend Axis enum comme objet pour décider de la direction du défilement sur le ListView .
        - Permet aux éléments de défiler horizontalement ou verticalement.
    - **`shrinkWrap`**: cette propriété prend une valeur booléenne comme objet. Ainsi, il décidera si la taille de la zone de défilement sera déterminée par le contenu à l'intérieur de **ListView**.
    - **`primary`** : si cette propriété est définie sur true, seul ce widget ListView peut défiler. Si c'est false, le widget ne pourra pas défiler.

4. Les types de ListViews

- Il existe quatre types de Widget ListViews comme indiqué ci-dessous :

1. **ListView()**

    - Le Widget `ListViewest` est le constructeur par défaut d'une classe **ListView**. Un **ListView** prend simplement la liste des enfants et la rend déroulante.
    - Lors de l'utilisation de `ListView Widget`, nous n'avons qu'à passer une List contenant tous les enfants à afficher dans ListView .
    - Tous les widgets de ces enfants seront créés de manière statique, même si certains d'entre eux ne sont pas actuellement visibles dans la fenêtre.
    - L'utilisateur ne doit pas créer un grand nombre de widgets enfants lors de l'utilisation de `ListView Widget`.
    - Voici un exemple d'application qui utilise le widget `ListView`:

apcpedagogie.com



2. **ListView.builder()**

- Pour des quantités d'informations plus conséquentes, il vaut mieux utiliser le constructeur nommé `ListView.builder`. Il prend un paramètre de type `IndexedWidgetBuilder` qui va créer les enfants à la demande, au fur et à mesure qu'ils seront visibles,
- `ListView.builder` a deux propriétés principales, à savoir **itemCount** (nombre d'éléments de la liste) et **itemBuilder** (pour créer des vues à partir d'éléments de liste).
- Pour bien comprendre ce concept réaliser l'exemple suivant :

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

Run | Debug | Profile
void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
// Ce widget est la racine
// de votre application.
  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
        title: "ListView.builder",
        debugShowCheckedModeBanner: false,
        home: ListViewBuilder()); // MaterialApp
  }
}
```

```dart
class ListViewBuilder extends StatelessWidget {
  const ListViewBuilder({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("ListView.builder")),
      body: ListView.builder(
          itemCount: 8,
          itemBuilder: (BuildContext context, int index) {
            return ListTile(
              leading: const Icon(Icons.favorite),
              trailing: const Icon(
                Icons.keyboard_arrow_right,
                color: Color.fromARGB(255, 6, 30, 247),size: 30,),
              title: Text("Élément de liste $index"),
              ); // ListTile
            }), // ListView.builder
    ); // Scaffold
  }
}
```

- Dans le code ci-dessus, on a utilisé la classe `ListViewBuilder` qui est une classe sans état (**StatelessWidget**). Il renvoie un nouveau **Scaffold** composé de **appBar** et de **body**.

- Dans le **body**, on a un `ListView.builder` avec **itemcount 8** et **itemBuilder** qui créera un nouveau widget encore et encore jusqu'à 8 fois car nous avons `itemCount=8`.
- Si nous n'utilisons pas `itemCount` dans `ListView.builder`, nous obtiendrons des éléments de liste infinis, il est donc recommandé d'utiliser `itemCount` pour éviter de telles erreurs.
- L'`itemBuilder` renvoie **ListTile**. Ce ListTile se construira encore et encore jusqu'à 8 fois.
- **Exemple d'application**
  - **Code**

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

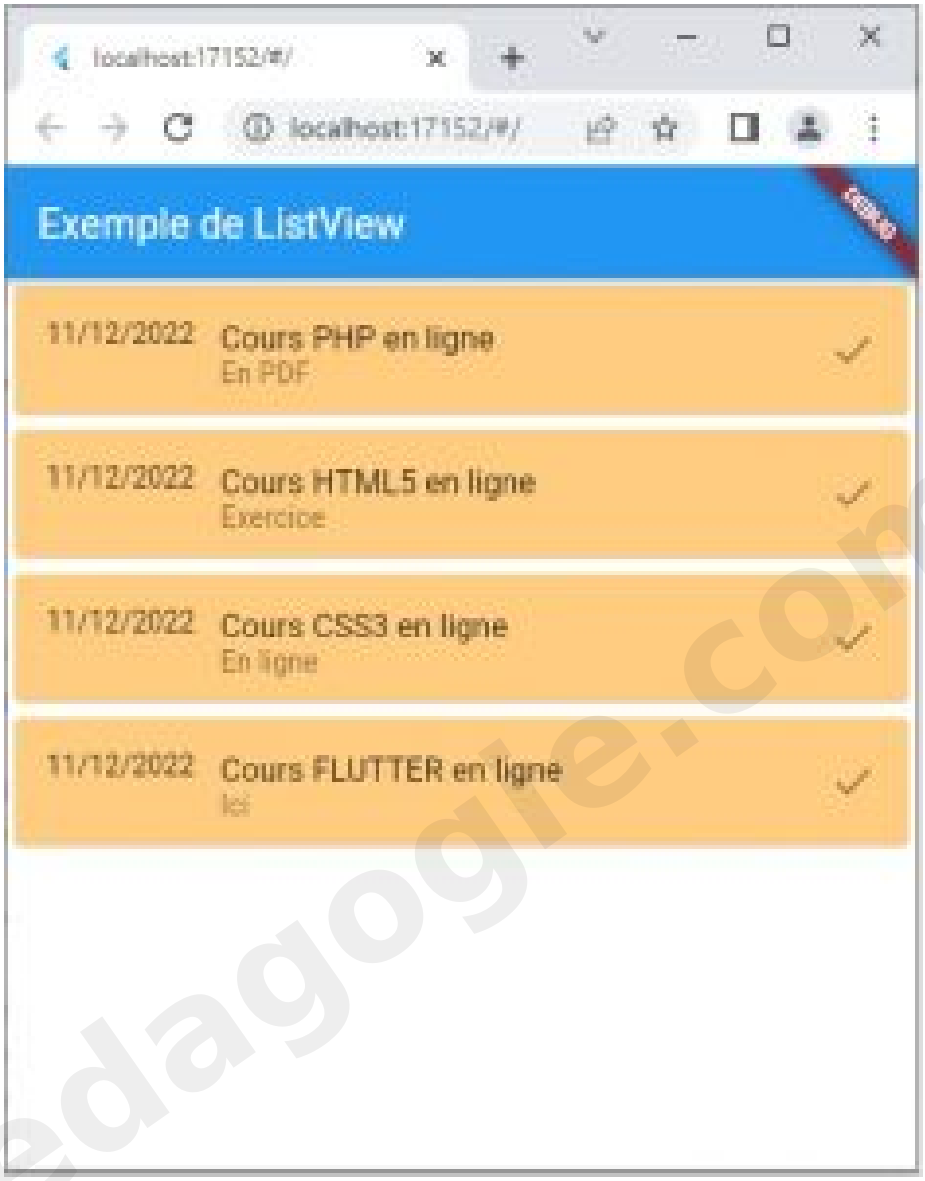
void main() => runApp(const MyWidget());

class MyWidget extends StatelessWidget {
  const MyWidget({super.key});
  Widget item(String title, subtitle) {
    return Card(
      color: Colors.orange[200],
      child: ListTile(
        leading: Text(
            "${DateTime.now().day}/${DateTime.now().month}/${DateTime.now().year}"),
        subtitle: Text(subtitle),
        title: Text(title),
        trailing: const Icon(Icons.check),
      ),
    );
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
          appBar: AppBar(
            title: const Text("Exemple de ListView"),
          ),
          body: ListView(
            children: [
              item("Cours PHP en ligne", "En PDF"),
              item("Cours HTML5 en ligne", "Exercice"),
              item("Cours CSS3 en ligne", "En ligne"),
              item("Cours FLUTTER en ligne", "Ici"),
            ],
          )),
    );
  }
}
```

  - **Résultat**

apcpedagogie.com



3. **ListView.separated()**

- La `ListView.separated` intègre un paramètre obligatoire supplémentaire, le **separatorBuilder**, qui attend une instance de **IndexedWidgetBuilder**. Son implémentation sera réalisée comme dans l'itemBuilder, c'est-à-dire en incorporant un **BuildContext** et un index.
- Le constructeur `ListView.separated()` est utilisé pour générer une liste de widgets, mais en plus, un widget séparateur peut également être généré pour séparer les widgets.
- En bref, ce sont deux listes de widgets entrelacées : la liste principale et la liste de séparation. Contrairement au constructeur builder(), le paramètre itemCount est obligatoire ici.

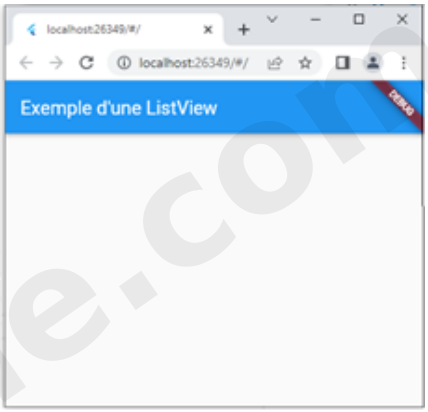
apcpedagogie.com

4. **ListView.custom()**

5. Exemples d'utilisation d'un `ListView` dans Flutter

- **Exemple 01**
  - **1-** Commencez par le code typique :

```dart
import 'package:flutter/material.dart';
Run | Debug | Profile
void main() => runApp(const ListViewApp());
class ListViewApp extends StatelessWidget {
  const ListViewApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text("Exemple d\'une ListView"),
        ), // AppBar
      ), // Scaffold
    ); // MaterialApp
  }
}
```



  - **2-**Ajouter le `ListView`:
    - **ListView le plus simple possible**

```
body: ListView(
          children: [
           Text('HTML5'),
           Text('CSS3'),
           Text('PHP8'),
           Text('FLUTTER'),
          ],
        ),
```

    - **On l'habille un peu**

```
body: ListView(
          children: [
           ListTile(
              title: Text('HTML5'),
            ), // ListTile
```

```
            ListTile(
              title: Text('CSS3'),
            ), // ListTile
            ListTile(
              title: Text('PHP8'),
            ), // ListTile
            ListTile(
              title: Text('FLUTTER'),
            ), // ListTile
          ],
        ),
```

- **Et il ne doit pas s'agir uniquement de ListTiles :**

```
ElevatedButton(
              child: Text('Press Me'),
              onPressed: () {},
            ),
```

- **Exemple 02**



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

Run | Debug | Profile
void main() => runApp(const MyApp());

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

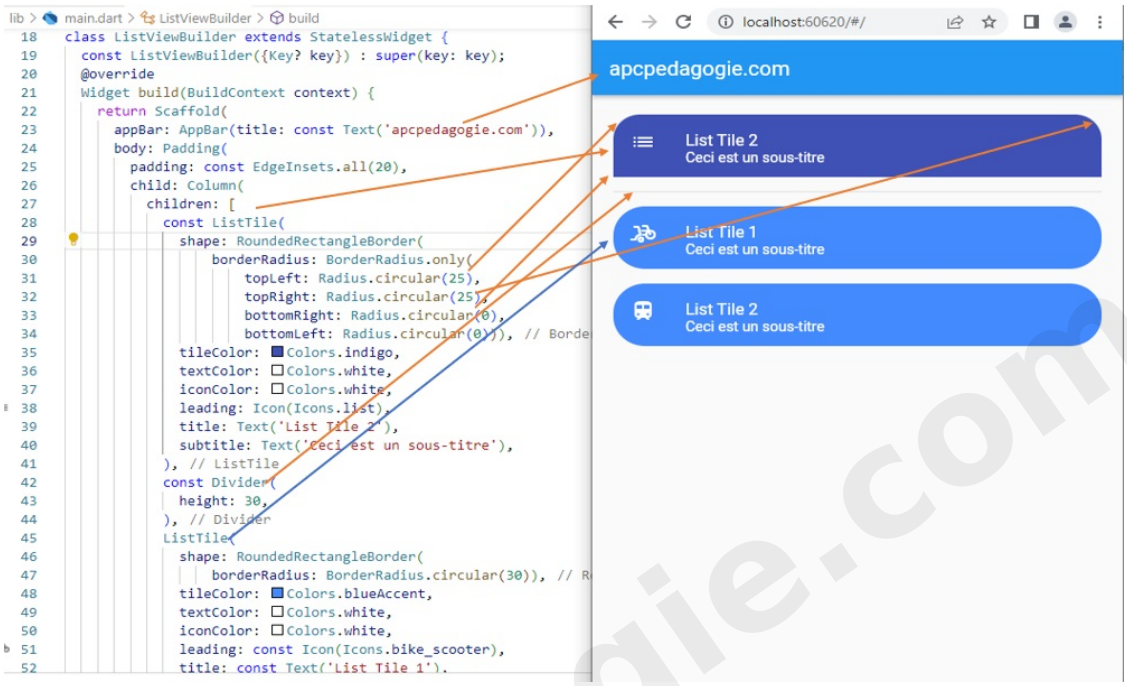
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'My App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: Scaffold(
        appBar: AppBar(title: const Text('ListTile guide')),
        body: BodyWidget(),
      ), // Scaffold
    ); // MaterialApp
  }
}
String horseUrl = 'https://i.stack.imgur.com/Dw6f7.png';
String cowUrl = 'https://i.stack.imgur.com/XPOr3.png';
String camelUrl = 'https://i.stack.imgur.com/VN0m7.png';
String sheepUrl = 'https://i.stack.imgur.com/wKzo8.png';
String goatUrl = 'https://i.stack.imgur.com/Qt4JP.png';

class BodyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ListView(
      children: <Widget>[
        ListTile(
          leading: CircleAvatar(
            backgroundImage: NetworkImage(horseUrl),
            //backgroundColor: Colors.transparent,
          ), // CircleAvatar
          title: const Text('Cheval'),
          subtitle: Text('Un animal fort'),
          trailing: Icon(Icons.keyboard_arrow_right),
          onTap: () {
            print('Cheval');
          },
          selected: true,
        ), // ListTile
        ListTile(
          leading: CircleAvatar(
            backgroundImage: NetworkImage(cowUrl),
          ), // CircleAvatar
          // title: Text('Vache'),
          subtitle: const Text('Fournit de lait'),
          trailing: Icon(Icons.keyboard_arrow_right),
          onTap: () {
            print('Vache');
          },
        ), // ListTile
        ListTile(
          leading: CircleAvatar(
            backgroundImage: NetworkImage(camelUrl),
          ), // CircleAvatar
          title: Text('chameau'),
          subtitle: Text('Propriétaire des bosses'),
          trailing: Icon(Icons.keyboard_arrow_right),
          onTap: () {
            print('chameau');
          },
          enabled: false,
        ), // ListTile
        ListTile(
          leading: CircleAvatar(
            backgroundImage: NetworkImage(sheepUrl),
          ), // CircleAvatar
          title: Text('Brebis'),
          subtitle: Text('Fournit de la laine'),
          trailing: Icon(Icons.keyboard_arrow_right),
          onTap: () {
            print('Brebis');
          },
        ), // ListTile
        ListTile(
          leading: CircleAvatar(
            backgroundImage: NetworkImage(goatUrl),
          ), // CircleAvatar
          title: Text('Chèvre'),
          subtitle: Text('Certains ont des cornes'),
          trailing: Icon(Icons.keyboard_arrow_right),
          onTap: () {
            print('Chèvre');
          },
        ), // ListTile
      ], // <Widget>[]
    ); // ListView
  }
```

6. Applications

1. **App01**

- Vous allez, dans ce TP, construire une application Flutter
- L'application présente tout d'abord une fenêtre avec trois items dans une liste.
- Utiliser l'image suivant pour créer une application Flutter similaire à cette dernière



2. **App02**

- Créez une application qui affiche une liste de tâches. Chaque tâche doit avoir un titre et une description.
  - **1-** Commencez par créer un modèle de données pour représenter une tâche. Il devrait contenir des champs pour le titre et la description.
  - **2-** Créez une liste de quelques tâches fictives.
  - **3-** Créez une nouvelle page pour afficher la liste des tâches.
  - **4-** Utilisez une ListView.builder pour afficher la liste des tâches.
  - **5-** Pour chaque tâche dans la liste, affichez le titre et la description dans une Card.
  - **6-** Ajoutez un bouton "Ajouter" en bas de la page pour permettre à l'utilisateur d'ajouter une nouvelle tâche.
  - **7-** Lorsque l'utilisateur appuie sur le bouton "Ajouter", ouvrez une nouvelle page avec un formulaire permettant de saisir le titre et la description de la nouvelle tâche.
  - **8-** Lorsque l'utilisateur soumet le formulaire, ajoutez la nouvelle tâche à la liste et revenez à la page principale pour afficher la liste mise à jour.

apcpedagogie.com

- **9-** Ajoutez également la possibilité de supprimer une tâche de la liste en faisant glisser la Card correspondante vers la gauche ou la droite.
- **10-** Ajoutez enfin la possibilité de modifier une tâche existante en appuyant dessus pour l'éditer.

Solution

1-

- Tout d'abord, créez une nouvelle classe Dart pour votre modèle de tâche. Vous pouvez le faire dans un fichier séparé, ou dans le même fichier que votre code d'interface utilisateur.
- Dans la classe, définissez deux champs pour le titre et la description de la tâche. Vous pouvez utiliser le type String pour les deux champs.

```
class Task {
  String title;
  String description;

  Task({this.title, this.description});
}
```

- Dans cet exemple, la classe Task contient deux champs, title et description, qui sont tous les deux des chaînes de caractères (String). La classe a également un constructeur qui prend deux paramètres optionnels : title et description.
- Bien sûr, vous pouvez ajouter d'autres champs à votre classe Task si nécessaire, en fonction de ce que vous voulez stocker pour chaque tâche.

2-

```
List tasks = [
  Task(title: 'Faire les courses', description: 'Acheter du lait, du pain, des oeufs'),
  Task(title: 'Aller à la salle de sport', description: 'Faire une séance de
musculation'),
  Task(title: 'Réviser pour l'examen', description: 'Relire les chapitres 1 à 3'),
  Task(title: 'Rendre le rapport', description: 'Finaliser le rapport et l'envoyer par e-
mail')
];
```

Cours Flutter

Source : https://apcpedagogie.com/le-widget-listview-dans-flutter · téléchargé le 06/11/2026 · © APC Pédagogie · apcpedagogie.com

apcpedagogie.com