Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Desarrollo de Aplicaciones Móviles

Profesor: Reynold Osuna González

Alumno:

David Carrillo Castillo

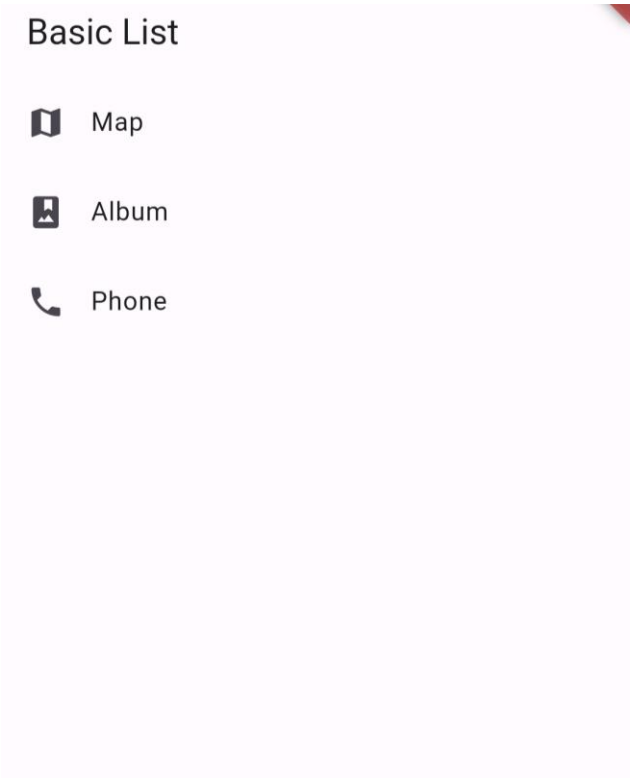
Listas y Grids

Primavera 2024

Leer y seguir las instrucciones del documento, entregar evidencia del código implementado con capturas de pantalla.

1) Flutter incluye el widget ListView para facilitar el uso de listas. El constructor estándar de ListView es útil para crear listas que contienen unos pocos elementos, y es posible utilizar el widget ListTile para darle a los elementos una estructura visual:

En el siguiente ejemplo, realizamos el código obteniendo el siguiente resultado:



Donde el código utilizado es:

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

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

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

  @override
  Widget build(BuildContext context) {
    const title = 'Basic List';

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ),
        body: ListView(
          children: const <Widget>[
            ListTile(
              leading: Icon(Icons.map),
              title: Text('Map'),
            ),
```

```
            ListTile(
              leading: Icon(Icons.photo_album),
              title: Text('Album'),
            ),
            ListTile(
              leading: Icon(Icons.phone),
              title: Text('Phone'),
            ),
          ],
        ),
      ),
    );
  }
}
```

2) Para crear un alista que se desplace horizontalmente en lugar de verticalmente, se utiliza el mismo constructor estándar de ListView pero pasando una dirección de desplazamiento horizontal, lo que anulará la dirección vertical que es la opción predeterminada:

El resultado de la actividad 2 es:

Donde el código es el siguiente:

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

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

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

  @override
  Widget build(BuildContext context) {
    const title = 'Horizontal List';

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ),
        body: Container(
          margin: const EdgeInsets.symmetric(vertical: 20),
          height: 200,
          child: ListView(
// This next line does the trick.

            scrollDirection: Axis.horizontal,

            children: <Widget>[
              Container(
                width: 160,
                color: Colors.red,
              ),
              Container(
                width: 160,
                color: Colors.blue,
              ),
              Container(
                width: 160,
                color: Colors.green,
              ),
              Container(
                width: 160,
                color: Colors.yellow,
              ),
              Container(
                width: 160,
                color: Colors.orange,
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

3) Si lo que se desea es mostrar una lista de elementos en una cuadrícula (grid) en lugar de una lista sencilla, entonces se utiliza el widget GridView. Haciendo uso del constructor GridView.count() se puede especificar cuántas filas o columnas se desea:

Nuestro ejercicio resuelto tendría la siguiente vista:

Y el siguiente código:

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

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

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

```
  @override
  Widget build(BuildContext context) {
    const title = 'Grid List';

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ),
        body: GridView.count(
// Create a grid with 2 columns. If you change the scrollDirection to

// horizontal, this produces 2 rows.

          crossAxisCount: 2,

// Generate 100 widgets that display their index in the List.

          children: List.generate(100, (index) {
            return Center(
              child: Text(
                'Item $index',
                style: Theme.of(context).textTheme.headlineSmall,
              ),
            );
          }),
        ),
      ),
    );
  }
}
```

4) Puedes necesitar crear listas que muestren diferentes tipos de contenido. Por ejemplo, podrías estar trabajando en una lista que muestra un encabezado seguido de algunos elementos relacionados con el encabezado, seguido de otro encabezado, y así sucesivamente.

Así es como puedes crear una estructura de ese tipo con Flutter:

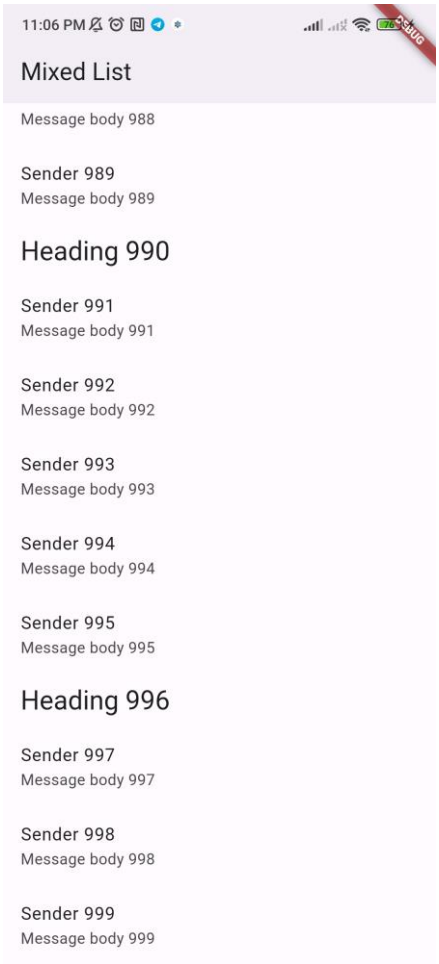
1. Crea una fuente de datos con diferentes tipos de elementos.

2. Convierte la fuente de datos en una lista de widgets.

Para representar diferentes tipos de elementos en una lista, define una clase para cada tipo de elemento.

En este ejemplo, crea una aplicación que muestra un encabezado seguido de cinco mensajes. Por lo tanto, crea tres clases: `ListItem` (elemento de lista), `HeadingItem` (elemento de encabezado) y `MessageItem` (elemento de mensaje).

El ejercicio terminado sería:



Donde el código es:

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

void main() {
  runApp(
    MyApp(
      items: List<ListItem>.generate(
        1000,
        (i) => i % 6 == 0
            ? HeadingItem('Heading $i')
            : MessageItem('Sender $i', 'Message body $i'),
      ),
    ),
  );
}

class MyApp extends StatelessWidget {
  final List<ListItem> items;

  const MyApp({super.key, required this.items});

  @override
  Widget build(BuildContext context) {
    const title = 'Mixed List';
```

```
    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ),
        body: ListView.builder(
// Let the ListView know how many items it needs to build.

          itemCount: items.length,

// Provide a builder function. This is where the magic happens.

// Convert each item into a widget based on the type of item it is.

          itemBuilder: (context, index) {
            final item = items[index];

            return ListTile(
              title: item.buildTitle(context),
              subtitle: item.buildSubtitle(context),
            );
          },
        ),
      ),
    );
  }
}

/// The base class for the different types of items the list can contain.

abstract class ListItem {
  /// The title line to show in a list item.

  Widget buildTitle(BuildContext context);

  /// The subtitle line, if any, to show in a list item.

  Widget buildSubtitle(BuildContext context);
}

/// A ListItem that contains data to display a heading.

class HeadingItem implements ListItem {
  final String heading;

  HeadingItem(this.heading);

  @override
  Widget buildTitle(BuildContext context) {
    return Text(
      heading,
```

```
      style: Theme.of(context).textTheme.headlineSmall,
    );
  }

  @override
  Widget buildSubtitle(BuildContext context) => const SizedBox.shrink();
}

/// A ListItem that contains data to display a message.

class MessageItem implements ListItem {
  final String sender;

  final String body;

  MessageItem(this.sender, this.body);

  @override
  Widget buildTitle(BuildContext context) => Text(sender);

  @override
  Widget buildSubtitle(BuildContext context) => Text(body);
}
```

5) Para crear una lista donde todos los elementos estén separados uniformemente de forma que éstos ocupen todo el espacio visible, o para permitir que el usuario se desplace a lo largo de la lista cuando no sea posible mostrar todos los elementos en pantalla, puedes recurrir a LayoutBuilder y SingleChildScrollView.

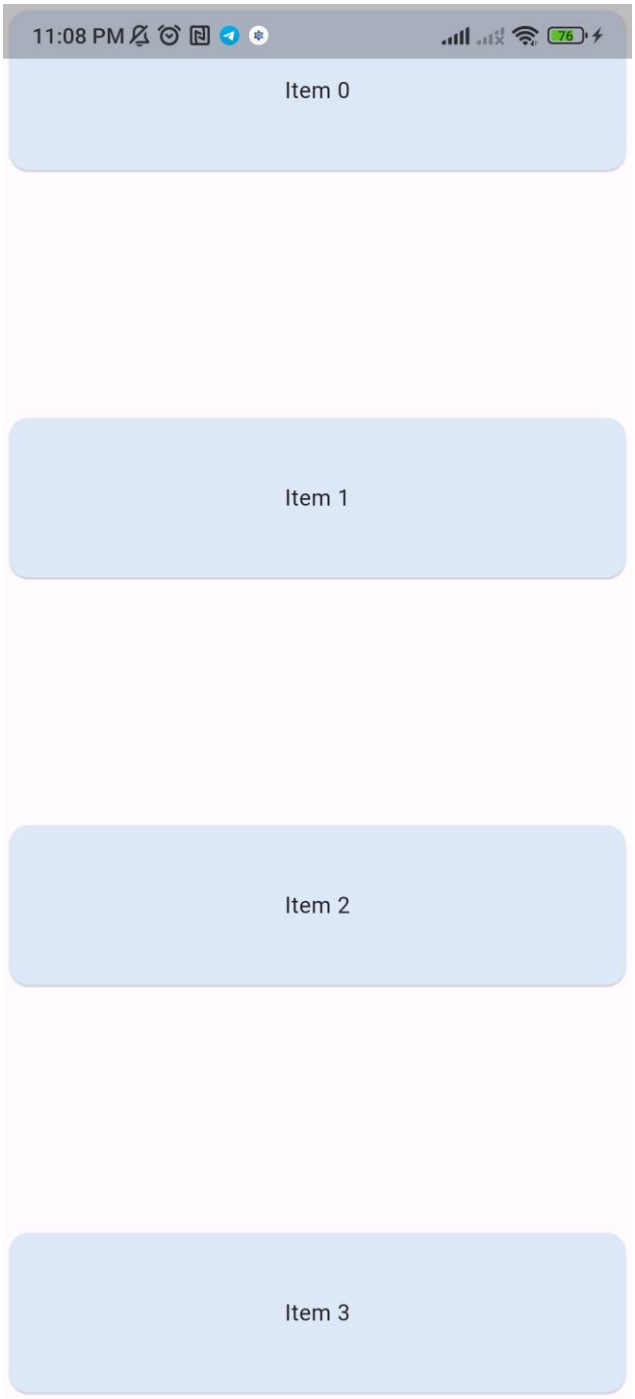
Primero crea un LayoutBuilder: debes proporcionar una función de devolución de llamada de constructor con dos parámetros:

1. El BuildContext proporcionado por el LayoutBuilder.

2. Las BoxConstraints del widget padre.

Dentro de la función de constructor, devuelve un SingleChildScrollView. Este widget asegura que el widget hijo se pueda desplazar, incluso cuando el contenedor principal es demasiado pequeño.

La solución es:



Y el código es:

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

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

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

  @override
  Widget build(BuildContext context) {
    const items = 4;

    return MaterialApp(
      title: 'Flutter Demo',
      debugShowCheckedModeBanner: false,
```

```
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        cardTheme: CardTheme(color: Colors.blue.shade50),
        useMaterial3: true,
      ),
      home: Scaffold(
        body: LayoutBuilder(builder: (context, constraints) {
          return SingleChildScrollView(
            child: ConstrainedBox(
              constraints: BoxConstraints(minHeight: constraints.maxHeight),
              child: Column(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                crossAxisAlignment: CrossAxisAlignment.stretch,
                children: List.generate(
                    items, (index) => ItemWidget(text: 'Item $index')),
              ),
            ),
          );
        }),
      ),
    );
  }
}

class ItemWidget extends StatelessWidget {
  const ItemWidget({
    super.key,
    required this.text,
  });

  final String text;

  @override
  Widget build(BuildContext context) {
    return Card(
      child: SizedBox(
        height: 100,
        child: Center(child: Text(text)),
      ),
    );
  }
}
```

6) Mientras que el constructor estándar de ListView es útil para listas con pequeños elementos, cuando se quiere trabajar con un número largo de elementos es mejor utilizar el constructor ListView.builder, que no requiere crear todos los elementos de la lista al mismo tiempo, si no que puede crear elementos conforme se van “deslizando” (scrolling) en la pantalla.

En este ejercicio se partirá de una lista creada de 10,000 cadenas, aunque en aplicaciones reales estos elementos usualmente provienen de distintas fuentes, como mensajes, resultados de búsquedas o productos en una tienda:

La solución es:

El código es:

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

void main() {
  runApp(
    MyApp(
      items: List<String>.generate(10000, (i) => 'Item $i'),
    ),
  );
}
```

```dart
class MyApp extends StatelessWidget {
  final List<String> items;

  const MyApp({super.key, required this.items});

  @override
  Widget build(BuildContext context) {
    const title = 'Long List';

    return MaterialApp(
      title: title,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title),
        ),
        body: ListView.builder(
          itemCount: items.length,
          prototypeItem: ListTile(
            title: Text(items.first),
          ),
          itemBuilder: (context, index) {
            return ListTile(
              title: Text(items[index]),
            );
          },
        ),
      ),
    );
  }
}
```