Guía de Laboratorio 2

Layout y Composición de Widgets

a) Objetivo del laboratorio

Comprender y aplicar los principios de layout y composición de widgets en Flutter para construir interfaces estructuradas, adaptables y visualmente coherentes.

b) Codificación

Crea un nuevo proyecto con el nombre layout_app

1. Estructura base

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Laboratorio Layout',
      theme: ThemeData(useMaterial3: true, colorSchemeSeed: Colors.indigo),
      home: const LayoutHome(),
    );
  }
}
```

2. Crear la pantalla principal

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const Color(0xFFF3F4F9),
      body: SafeArea(
        child: Padding(
          padding: const EdgeInsets.all(16),
          child: Column(
            children: const [
              HeaderSection(),
              SizedBox(height: 20),
              BalanceCard(),
```

```
              SizedBox(height: 20),
              Expanded(child: TransactionsList()),
            ],
          ),
        ),
      ),
    );
  }
}
```

3. Composición modular de widgets

3.1. Encabezado

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

  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        const CircleAvatar(
          radius: 24,
          backgroundImage: NetworkImage(
            'https://images.unsplash.com/photo-1544723795-3fb6469f5b39',
          ),
        ),
        const SizedBox(width: 12),
        const Expanded(
          child: Text(
            'Hola, Flutterista!',
            style: TextStyle(fontSize: 18, fontWeight: FontWeight.w600),
          ),
        ),
        IconButton(icon: const Icon(Icons.more_vert), onPressed: () {}),
      ],
    );
  }
}
```

3.2. Tarjeta de balance

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

  @override
  Widget build(BuildContext context) {
    return Container(
```

```
      height: 120,
      padding: const EdgeInsets.all(20),
      decoration: BoxDecoration(
        color: Colors.indigo,
        borderRadius: BorderRadius.circular(20),
      ),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: const [
          Text('Balance Actual', style: TextStyle(color: Colors.white70)),
          Text(
            '\$ 1,260.50',
            style: TextStyle(
              color: Colors.white,
              fontWeight: FontWeight.bold,
              fontSize: 22,
            ),
          ),
        ],
      ),
    );
  }
}
```

3.3. Lista de transacciones

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

  @override
  Widget build(BuildContext context) {
    final items = [
      {'title': 'Depósito recibido', 'amount': '+\$300'},
      {'title': 'Pago Netflix', 'amount': '-\$45'},
      {'title': 'Compra supermercado', 'amount': '-\$120'},
    ];

    return ListView.builder(
      itemCount: items.length,
      itemBuilder: (context, i) => ListTile(
        leading: const Icon(Icons.payment),
        title: Text(items[i]['title']!),
        trailing: Text(items[i]['amount']!),
      ),
    );
  }
}
```

c) Manual de Widgets para Layout en Flutter

Introducción

En Flutter, todo es un widget: desde un texto o un ícono hasta la estructura completa de la pantalla. Los widgets de layout son aquellos que definen cómo se organiza y distribuye el contenido en la interfaz. Dominar estos widgets es esencial para construir aplicaciones bien estructuradas, legibles y adaptables.

1. Widgets básicos de estructura

1.1. Container

Descripción: Widget contenedor versátil que permite definir tamaño, color, márgenes, bordes, fondo e incluso transformaciones.

Uso principal: Crear bloques, tarjetas o fondos de secciones.

Ejemplo:

```
Container(
  width: 200,
  height: 100,
  padding: EdgeInsets.all(16),
  decoration: BoxDecoration(
    color: Colors.blue.shade100,
    borderRadius: BorderRadius.circular(12),
  ),
  child: Text('Contenido dentro de un Container'),
)
```

Tip: Combina Container con BoxDecoration para bordes, sombras o gradientes.

1.2. Padding

Descripción: Añade espacio interno alrededor de un widget.

Ejemplo:

```
Padding(
  padding: EdgeInsets.symmetric(horizontal: 20, vertical: 10),
  child: Text('Texto con padding'),
)
```

Tip: Úsalo para separar widgets sin modificar su estructura.

1.3. Center

Descripción: Centra su hijo tanto horizontal como verticalmente.

Ejemplo:

```
Center(
  child: Text('Centrado en pantalla'),
)
```

1.4. Align

Descripción: Alinea su hijo según un punto específico del contenedor.

Ejemplo:

```
Align(
  alignment: Alignment.bottomRight,
  child: Icon(Icons.star, color: Colors.amber),
)
```

2. Organización en filas y columnas

2.1. Row

Descripción: Dispone los widgets horizontalmente.

Propiedades importantes:

- **mainAxisAlignment:** Alinea los elementos en el eje horizontal.
- **crossAxisAlignment:** Alinea los elementos en el eje vertical.

Ejemplo:

```
Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: const [
    Icon(Icons.home),
    Icon(Icons.search),
    Icon(Icons.person),
  ],
)
```

2.2. Column

Descripción: Dispone los widgets verticalmente.

Ejemplo:

```
Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: const [
    Text('Nombre'),
    Text('Correo electrónico'),
  ],
)
```

2.3. Expanded

Descripción: Hace que un widget hijo ocupe el espacio disponible dentro de un Row o Column.

Ejemplo:

```
Row(
  children: [
    Expanded(child: Container(color: Colors.red, height: 50)),
    Expanded(child: Container(color: Colors.green, height: 50)),
  ],
)
```

Tip: Usa flex: para controlar la proporción:

Expanded(flex: 2, child: Container(color: Colors.blue))

2.4. Flexible

Descripción: Similar a Expanded, pero permite que el contenido mantenga su tamaño natural si no necesita ocupar todo el espacio.

Ejemplo:

```
Row(
  children: [
    Flexible(child: Text('Texto que puede ocupar varias líneas')),
  ],
)
```

3. Disposición avanzada

3.1. Stack

Descripción: Superpone widgets unos sobre otros (como capas).

Ejemplo:

```
Stack(
  children: [
    Container(color: Colors.blue, width: 200, height: 200),
    Positioned(
      bottom: 10,
      right: 10,
      child: Icon(Icons.star, color: Colors.yellow),
    ),
  ],
)
```

Tip: Ideal para fondos decorativos o etiquetas sobre imágenes.

3.2. Positioned

Descripción: Se usa dentro de un Stack para ubicar un widget en una posición específica.

Ejemplo:

```
Positioned(
  top: 10,
  left: 10,
  child: Text('Etiqueta'),
)
```

3.3. SizedBox

Descripción: Define un espacio fijo o separa widgets con un tamaño específico.

Ejemplo:

```
SizedBox(height: 20)
```

Tip: Es más eficiente que un Container vacío para crear espacios.

3.4. Wrap

Descripción: Distribuye widgets horizontal o verticalmente y salta a la siguiente línea si no hay espacio disponible.

Ejemplo:

```
Wrap(
  spacing: 8,
  runSpacing: 8,
  children: List.generate(
    6,
    (i) => Chip(label: Text('Item $i')),
  ),
)
```

4. Widgets de desplazamiento (Scroll)

4.1. SingleChildScrollView

Descripción: Permite desplazar el contenido cuando no cabe en la pantalla.

Ejemplo:

```
SingleChildScrollView(
  child: Column(
    children: List.generate(20, (i) => Text('Elemento $i')),
  ),
```

```
)
```

4.2. ListView

Descripción: Lista desplazable de elementos (widgets repetitivos o dinámicos).

Ejemplo:

```
ListView.builder(
  itemCount: 10,
  itemBuilder: (context, i) => ListTile(
    leading: Icon(Icons.person),
    title: Text('Usuario $i'),
  ),
)
```

4.3. GridView

Descripción: Muestra los elementos en formato de cuadrícula.

Ejemplo:

```
GridView.count(
  crossAxisCount: 2,
  children: List.generate(
    6,
    (i) => Card(
      color: Colors.indigo.shade100,
      child: Center(child: Text('Item $i')),
    ),
  ),
)
```

5. Layout adaptable (Responsive)

5.1. LayoutBuilder

Descripción: Permite construir layouts basados en el tamaño disponible.

Ejemplo:

```
LayoutBuilder(
  builder: (context, constraints) {
    if (constraints.maxWidth > 600) {
      return Row(children: [Text('Vista de escritorio')]);
    } else {
      return Column(children: [Text('Vista móvil')]);
    }
  },
```

```
)
```

5.2. MediaQuery

Descripción: Permite obtener información del dispositivo (ancho, alto, orientación).

Ejemplo:

```
double width = MediaQuery.of(context).size.width;
Text('Ancho de pantalla: $width');
```