DESARROLLO DE APP MOVILES I

GONZALEZ DIAZ ANSELMO ALEXIS

Aarón Hernández García

Animaciones

25/02/2025

Introducción

Las animaciones bien diseñadas hacen que la interfaz de usuario sea más intuitiva, contribuyen a la apariencia elegante de una aplicación pulida y mejoran la experiencia del usuario. La compatibilidad con animaciones de Flutter facilita la implementación de una variedad de tipos de animaciones. Muchos widgets, especialmente los widgets Material, vienen con los efectos de movimiento estándar definidos en su especificación de diseño, pero también es posible personalizar estos efectos.

Flutter ofrece dos enfoques principales para incorporar animaciones en una aplicación: animaciones **implícitas** y animaciones **explícitas**. A continuación, se describe cómo implementar cada método, junto con sus ventajas y desventajas.

1. Animaciones Implícitas

Las animaciones implícitas en Flutter se logran utilizando widgets que animan automáticamente los cambios en sus propiedades cuando se reconstruyen con nuevos valores. Estos widgets, como AnimatedContainer, AnimatedOpacity y AnimatedPositioned, simplifican la creación de animaciones al manejar internamente la transición entre valores.

Procedimiento para incorporar animaciones implícitas:

- **Paso 1:** Identificar el widget cuya propiedad desea animar.
- **Paso 2:** Reemplazar ese widget por su versión animada correspondiente (por ejemplo, Container por AnimatedContainer).
- **Paso 3:** Definir las propiedades iniciales y finales que desea animar.
- **Paso 4:** Establecer la duración de la animación mediante el parámetro duration.
- **Paso 5:** Utilizar setState para actualizar las propiedades y desencadenar la animación cuando sea necesario.

Ejemplo de uso de **AnimatedContainer**:

```dart
class MiWidgetAnimado extends StatefulWidget {
  @override
  _MiWidgetAnimadoState createState() => _MiWidgetAnimadoState();
}

class _MiWidgetAnimadoState extends State<MiWidgetAnimado> {
  bool _agrandar = false;

  @override
  Widget build(BuildContext context) {
    return Column(
      children: <Widget>[
        AnimatedContainer
          width: _agrandar ? 200.0 : 100.0,
          height: _agrandar ? 200.0 : 100.0,
          color: _agrandar ? Colors.blue : Colors.red,
          duration: Duration(seconds: 1),
          curve: Curves.easeInOut,
        ),
        ElevatedButton(
          onPressed: () {
            setState(() {
              _agrandar = !_agrandar;
            });
          },
          child: Text('Animaciones Apps Móviles'),
        ),
      ],
    );
  }
}
```

Ventajas*:*

- **Simplicidad:** Fáciles de implementar, ideales para animaciones básicas y transiciones simples.
- **Legibilidad:** El código es más limpio y fácil de mantener debido a su estructura declarativa.

Desventajas*:*

- **Flexibilidad Limitada:** Menos control sobre el proceso de animación, lo que puede ser restrictivo para animaciones más complejas.
- **Rendimiento:** Para animaciones avanzadas, pueden no ser tan eficientes como las animaciones explícitas.

2. Animaciones Explícitas

Las animaciones explícitas proporcionan un control más detallado sobre el proceso de animación. Involucran el uso de clases como AnimationController y Animation, permitiendo definir el comportamiento, la duración y la sincronización de la animación de manera precisa.

Procedimiento para incorporar animaciones explícitas:

- **Paso 1:** Crear una instancia de AnimationController, especificando la duración y el TickerProvider.
- **Paso 2:** Definir una animación (Animation) que describa cómo varían los valores durante la animación, utilizando clases como Tween.
- **Paso 3:** Asociar el controlador de animación con un Listener o un widget como AnimatedBuilder para reconstruir la interfaz en cada cambio de valor.
- **Paso 4:** Iniciar, detener o revertir la animación según sea necesario utilizando métodos del controlador como forward(), reverse() o stop().

Ejemplo de uso de **AnimationController** y **Tween**:

```dart
class MiAnimacionExplicita extends StatefulWidget {
  @override
  _MiAnimacionExplicitaState createState() => _MiAnimacionExplicitaState();
}

class _MiAnimacionExplicitaState extends State<MiAnimacionExplicita> with SingleTickerProviderStateMixin {
  AnimationController _controlador;
  Animation<double> _animacion;

  @override
  void initState() {
    super.initState();
    _controlador = AnimationController(
      duration: const Duration(seconds: 2),
      vsync: this,
    );
    _animacion = Tween<double>(begin: 0, end: 300).animate(_controlador)
      ..addListener(() {
        setState(() {});
      });
    _controlador.forward();
  }

  @override
  void dispose() {
    _controlador.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Container(
      width: _animacion.value,
      height: _animacion.value,
      color: Colors.green,
    );
  }
}
```

Ventajas*:*

- **Control Preciso:** Permite una gestión detallada de cada aspecto de la animación, incluyendo curvas de interpolación y sincronización.
- **Flexibilidad:** Adecuado para animaciones complejas y personalizadas que requieren un control exhaustivo.

Desventajas*:*

- **Complejidad:** Requiere una comprensión más profunda del sistema de animaciones de Flutter, lo que puede aumentar la complejidad del código.
- **Verbosidad:** El código puede ser más extenso y detallado en comparación con las animaciones implícitas.

Referencias

Conceptos básicos sobre animación en Flutter con animaciones implícitas. (s/f). Google Developers. Recuperado el 25 de febrero de 2025, de https://developers-latam.googleblog.com/2019/12/conceptos-basicos-sobre-animacion-en.html

Introduction to animations. (s/f). Flutter.dev. Recuperado el 25 de febrero de 2025, de https://docs.flutter.dev/ui/animations