Flutter Bootcamp

Animaciones

Emanuel López – emanuel.lopez@globant.com

Globant

Repaso

- Bases de datos NoSQL
- Firestore: listado, creación, actualización, suscripción, etc.
- Firestore: creación del proyecto en la consola de Firebase
- Plugins
    - MethodChannel
    - EventChannel

Animaciones

Las animaciones bien diseñadas contribuyen a una apariencia más elegante en una aplicación y mejoran la experiencia del usuario.

Flutter facilita la implementación de una amplia variedad de animaciones. Muchos widgets vienen con los efectos de movimiento estándar definidos en su especificación de diseño, pero también es posible personalizarlos.

Hay dos tipos principales de animaciones que podemos incluir en nuestra aplicación:

- **Las animaciones basadas en código** se centran en los widgets y se basan en primitivas de diseño y estilo estándar. Tienden a mejorar la apariencia o la transición de un widget existente.
- **Las animaciones basadas en dibujos** parecen dibujadas por alguien. A menudo son sprites independientes o implican transformaciones que serían difíciles de expresar únicamente con código.

Animaciones basadas en código

Las animaciones basadas en código de Flutter vienen en dos tipos:

- **Las animaciones implícitas**:
    - Se basan en establecer un nuevo valor para alguna propiedad del widget
    - Flutter se encarga de animarlo desde el valor actual hasta el nuevo valor.
    - Widgets fáciles de usar e increíblemente poderosos.
    - Son un buen punto de partida cuando buscas animar algo.
- **Las animaciones explícitas**:
    - Requieren un `AnimationController`.
    - Solo comienzan a animarse cuando se les pide explícitamente que lo hagan.
    - Debemos administrar manualmente el ciclo de vida del `AnimationController`.

Animaciones implícitas

Son las animaciones más fáciles de implementar y se encargan automáticamente de los cambios en las propiedades.

El proceso de animación a través de los valores entre el valor antiguo y el nuevo se denomina **interpolación**. El widget se encarga de interpolar sus propiedades entre los valores antiguos y los nuevos siempre que cambien.

Algunos ejemplos de widgets con animaciones implícitas son:

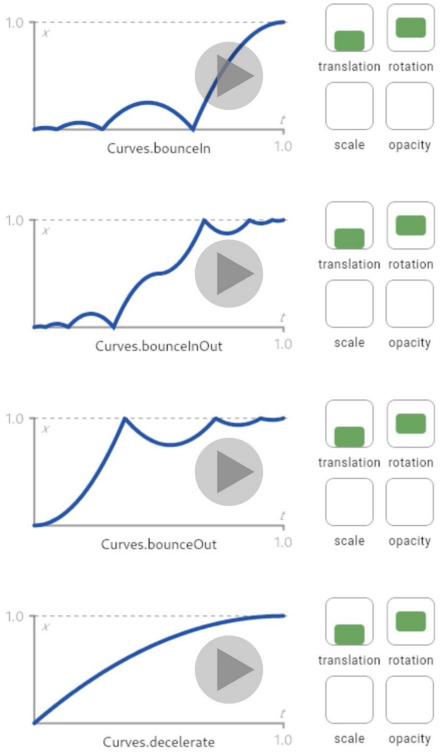
- `AnimatedContainer`
- `AnimatedOpacity`
- `AnimatedAlign`
- `AnimatedPositioned`
- `AnimatedCrossFade`
- `AnimatedDefaultTextStyle`
- `AnimatedSize`
- `AnimatedSwitcher`

Animaciones implícitas

Los widgets animados implícitamente tienen dos propiedades que podemos usar para controlar el comportamiento de la animación:

- `duration:` Podemos controlar cuánto tiempo lleva interpolar hasta el nuevo valor.
- `curve:` Podemos controlar la forma en que el widget interpola del valor anterior al nuevo mediante una curva . Las curvas controlan la velocidad de cambio a lo largo del tiempo y pueden ayudar a que tus animaciones parezcan más realistas.

Hay muchas curvas integradas diferentes disponibles para darle a nuestras animaciones un poco de personalidad. También podemos definir nuestras propias curvas personalizadas.



Globant

Animaciones implícitas con TweenAnimationBuilder

Si necesitamos crear una animación básica y ninguno de los widgets para animaciones implícitas nos sirve, aún podemos crear nuestra animación con `TweenAnimationBuilder`.

Para esto, necesitamos especificar la duración de la animación (`duration`) y el rango de valores entre los que quiero animarla con el parámetro `tween`. `Tween` es un objeto te permite especificar un rango de valores entre los que queremos animar. (`tween` viene de be`tween`)

Lo último que necesitamos especificar es el parámetro `builder`, que devuelve cómo se verá nuestro widget animado en un momento determinado. Esta función `builder` toma un parámetro que es del mismo tipo que los valores del `Tween`, que básicamente le dice a Flutter cuál es el valor de la animación actual en un momento determinado.

Animaciones explícitas. Transition Widgets

Los widgets de transición son un conjunto de widgets de Flutter cuyos nombres terminan en `Transition`. Se ven y se sienten muy similares a los `Animated` widgets.

`PositionedTransition`, por ejemplo, anima la transición de un widget entre diferentes posiciones. Esto es muy parecido a `AnimatedPositioned`.

Hay una diferencia importante: estos widgets de transición son extensiones de `AnimatedWidget`. Esto los convierte en animaciones explícitas .

Inheritance

Object › DiagnosticableTree › Widget › StatefulWidget › AnimatedWidget › PositionedTransition

Ejemplo: RotationTransition

`RotationTransition` es un widget que se encarga de todas las operaciones trigonométricas y de las transformaciones matemáticas necesarias para que todo gire. Su constructor acepta 3 parámetros:

- `child:` el widget que queremos rotar.
- `alignment:` punto alrededor del cual gira nuestra animación, los cálculos de rotación estarán alineados con ese punto.
- `turns:` Indicador sobre el porcentaje de una rotación que se ha completado y notifica cuando este valor cambia.

Podríamos lograr el mismo efecto de rotación con widgets de animación implícita, pero nuestra animación rotaría una vez y se detendría. Con las animaciones explícitas, tenemos el control del tiempo y podemos hacer que nuestro widget nunca deje de girar.

Ejemplo: RotationTransition

La propiedad `turns` es de tipo `Animation<double>`. Una de las formas más sencillas de obtener un `Animation<double>` es crear un `AnimationController`, que es un controlador para una animación. Este controlador nos brinda algunos controles útiles sobre lo que está haciendo la animación.

Debido a que `AnimationController` también tiene su propio estado para administrar, lo inicializamos en `initState` y lo eliminamos en `dispose`.

Hay dos parámetros que debemos darle al constructor de `AnimationController`:

- `duration:` cuánto dura nuestra animación.
- `vsync:` (vertical synchronization) se utiliza para evitar que el controlador de animación gaste recursos ejecutando fotogramas que no se mostrarán en pantalla.

¿Cómo funciona vsync?

Cuando una animación está activa, Flutter actualiza la pantalla tantas veces por segundo como la tasa de refresco del dispositivo lo permita (generalmente 60 fps o 120 fps).

El parámetro vsync permite que la animación esté sincronizada con ese refresco de pantalla, asegurando que no se generen más fotogramas de los que pueden ser dibujados, lo que ayuda a ahorrar recursos y a que las animaciones sean más suaves.

`vsync` generalmente se proporciona a través de una clase que implementa la interfaz `TickerProvider`. En la mayoría de los casos, se hace a través de la clase `State` del widget que contiene la animación incorporando alguno de los siguientes mixins:

- `SingleTickerProviderStateMixin`: Se usa cuando solo se necesita un `Ticker` o animación. Es el más común y eficiente para la mayoría de los casos.
- `TickerProviderStateMixin`: Se usa cuando necesitas múltiples `AnimationController` en la misma clase.

Animaciones Explícitas

Los widgets que nos permiten realizar animaciones explícitas son los siguientes:

- `FadeTransition:` Anima la opacidad de un widget.
- `ScaleTransition:` Anima el escalado de un widget.
- `RotationTransition:` Anima la rotación de un widget.
- `SlideTransition:` Anima la posición de un widget deslizándose de un lugar a otro.
- `PositionedTransition:` Anima la posición de un widget dentro de un Stack.
- `SizeTransition:` Anima el tamaño de un widget usando un eje específico.
- `DecoratedBoxTransition:` Anima las diferentes propiedades de un decoration

Animaciones con AnimatedWidget o AnimatedBuilder

Si ninguno de los widgets anteriores puede hacer lo que estamos buscando, es hora de crear nuestra animación propio usando `AnimatedWidget` o `AnimatedBuilder`.

Queremos crear una animación de "rayo hacia abajo", comenzando desde el centro de ese degradado, y queremos que se repita. Esto significa que necesitaremos crear una **animación explícita**.

Lamentablemente, no hay una animación explícita incorporada para animar degradados en forma de embudo para el rayo, entonces usaremos `AnimatedWidget` y `AnimatedBuilder` para animarlo.

Animaciones Lottie

Las animaciones `Lottie` son archivos de animación vectorial en formato `JSON` que se pueden reproducir en aplicaciones móviles o web de manera eficiente y escalable.

Características principales de Lottie:

- **Escalabilidad y resolución:** Al ser animaciones vectoriales, no pierden calidad al cambiar el tamaño de la pantalla o la resolución.
- **Tamaño reducido:** Los archivos Lottie son mucho más ligeros en comparación con archivos de video o imágenes animadas (como `GIF`).
- **Interactividad:** Las animaciones pueden controlarse programáticamente, lo que permite crear animaciones reactivas a eventos o interacciones del usuario.
- **Multiplataforma:** Las animaciones Lottie pueden usarse tanto en iOS, Android como en aplicaciones web con las mismas animaciones, lo que garantiza coherencia visual.

Animaciones Rive

Rive es una plataforma que permite diseñar, crear y programar animaciones de alta calidad, y luego exportarlas para su uso en aplicaciones móviles, web y juegos.

Características de las animaciones con Rive:

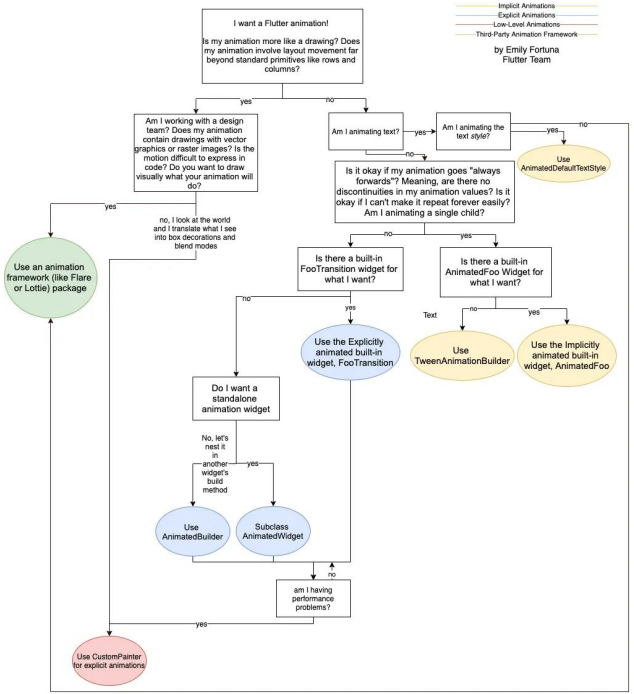
- **Interactivas:** Las animaciones pueden responder a entradas del usuario, como toques, deslizamientos o gestos. También se pueden conectar a datos en tiempo real.
- **Animaciones vectoriales:** Son animaciones basadas en gráficos vectoriales, lo que las hace ligeras, escalables y adecuadas para diferentes tamaños de pantalla sin perder calidad.
- **Control mediante código:** Puedes manipular la animación desde el código, lo que te permite cambiar el estado de la animación o reproducir diferentes partes de la animación según eventos en la aplicación.

Animaciones Rive

Características de las animaciones con Rive (continuación):

- **Estados animados:** Rive permite configurar múltiples estados para una animación (por ejemplo, "inactivo", "en movimiento", "completo"), lo que permite transiciones fluidas entre diferentes estados visuales.
- **Optimización de rendimiento:** Al ser vectoriales y estar optimizadas para rendimiento en tiempo real, las animaciones Rive son ideales para aplicaciones móviles y juegos donde el rendimiento es crítico.

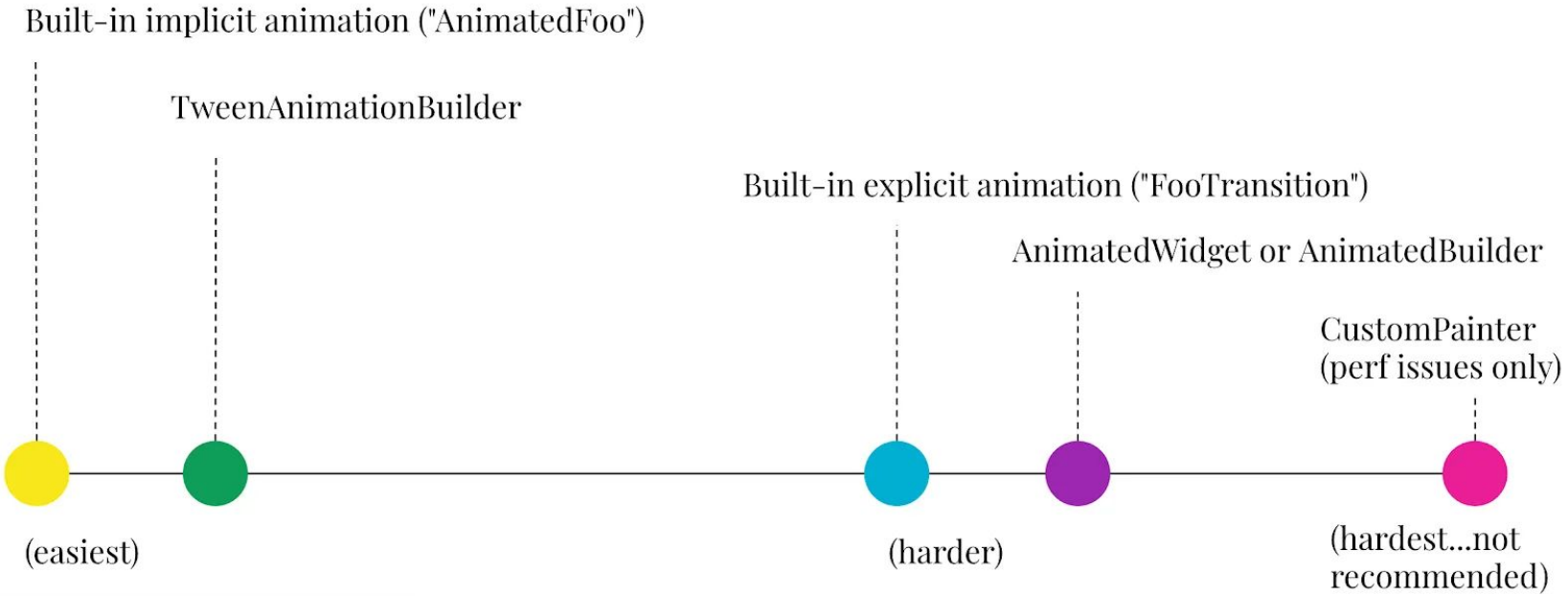
Cómo saber qué tipo de animación necesito?



Fuente: [How to Choose Which Flutter Animation Widget is Right for You?]

Globant

Animaciones según su dificultad

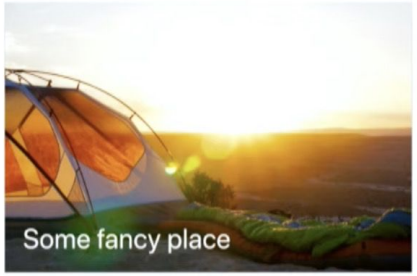


Fuente: [How to Choose Which Flutter Animation Widget is Right for You?]

Globant

Implementando UI complejas

```
Stack(
  children: <Widget>[
    Image.asset('assets/img1.jpeg'),
    Positioned(
      bottom: 16,
      left: 16,
      child: Text('Some fancy place'),
    ),
  ],
)
```



Stack

Fuente: Implementing complex UI in Flutter: youtube.com/watch?v=FCyoHclCqc8

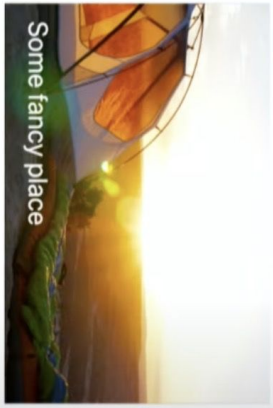
Implementando UI complejas

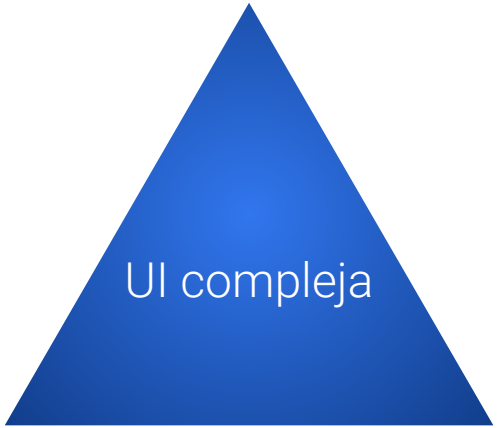
```
Transform.translate(
  offset: Offset(64, 32),
  child: MyImage(),
);
```

```
Transform.scale(
  scale: 0.5,
  child: MyImage(),
);
```

```
Transform.rotate(
  angle: math.pi / 2,
  child: MyImage(),
);
```

```
Transform(
  transform: Matrix4.identity()
    ..translate(128.0)
    ..rotateZ(math.pi / 2),
  child: MyImage(),
);
```

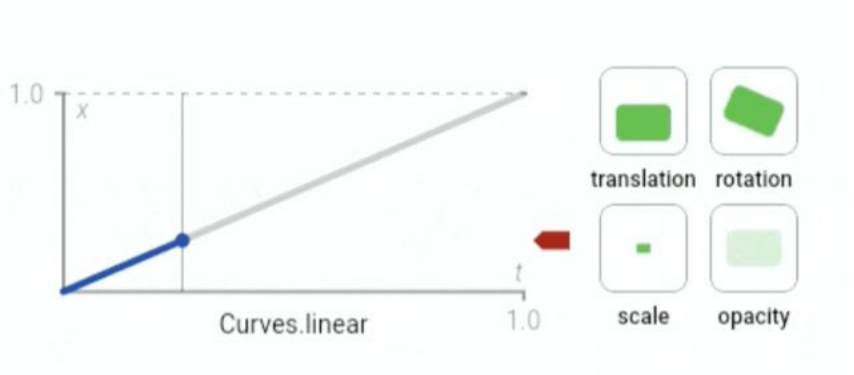




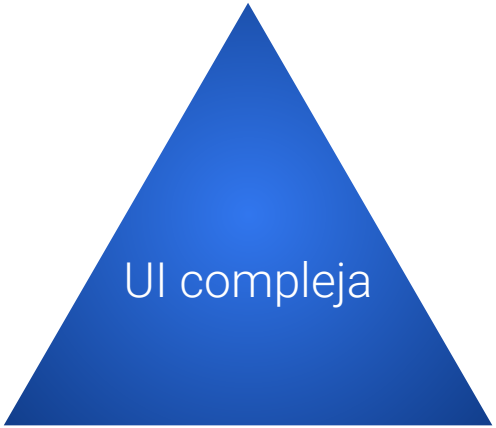
Fuente: Implementing complex UI in Flutter: youtube.com/watch?v=FCyoHclCqc8

Globant

Implementando UI complejas



AnimationController



Transform

Stack

Fuente: Implementing complex UI in Flutter: youtube.com/watch?v=FCyoHclCqc8

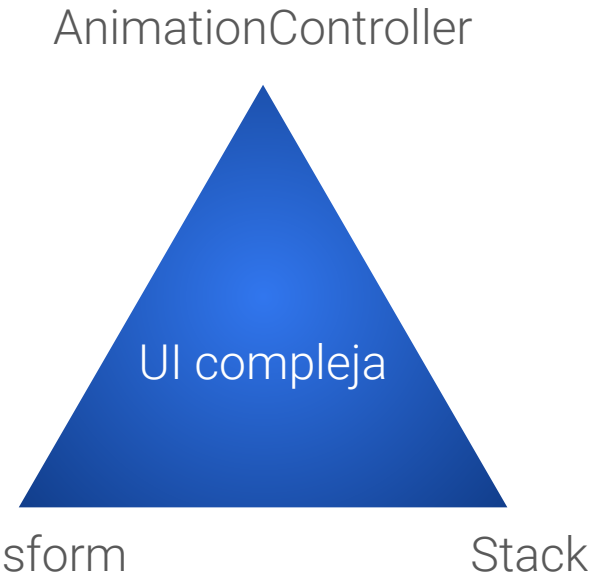
Globant

Implementando UI complejas

```dart
class _MyHomePageState extends State<MyHomePage>
  with SingleTickerProviderStateMixin {
  AnimationController _animationController;

  @override
  void initState() {
    super.initState();
    _animationController = AnimationController(
      vsync: this,
      duration: Duration(seconds: 1),
    );
    _animationController.forward();
  }

  @override
  Widget build(BuildContext context) {
    return AnimatedBuilder(
      animation: _animationController,
      child: MyCard(number: 1),
      builder: (context, child) => Transform.rotate(
        angle: _animationController.value * math.pi,
        child: child,
      ),
    );
  }
}
```



Fuente: Implementing complex UI in Flutter: youtube.com/watch?v=FCyoHclCqc8

Globant

Implementando UI complejas

Fuente: fidev.io/complex-ui

Globant

Links adicionales

- Animations & Transitions: docs.flutter.dev/ui/animations
- Flutter animation samples: flutter.github.io/samples/animations.html
- Implementing complex UI in Flutter (Flutter Europe 2020): youtube.com/watch?v=FCyoHclCqc8
- Animating a production app with ten thousands of users (FlutterCon 2023): droidcon.com/2023/08/06/animating-a-production-app-with-tens-of-thousands-of-users
- Lottie files: lottiefiles.com/featured-free-animations
- Rive: rive.app/community/files
- The history of everything: medium.com/rive/the-history-of-everything-981d989e1b45 (Play Store) (App Store) (github)

Preguntas?

Globant

Gracias!

Globant