**Họ tên: Hà Văn Lương**

**Lớp:08THMT**

**MSSV: 0850080031**

1. AnimatedWidget

AnimatedWidget không phải là một widget cụ thể trong Flutter. Tuy nhiên, có thể bạn đang nói đến AnimatedWidget trong ngữ cảnh của Flutter, một khái niệm liên quan đến các hiệu ứng hoạt hình trong giao diện người dùng.

Trong Flutter, AnimatedWidget là một lớp cơ sở (base class) được sử dụng để tạo các widget có thể thay đổi theo thời gian thông qua các hoạt hình. Khi bạn muốn tạo một widget có thể tự động cập nhật và hoạt hình mượt mà, bạn có thể kế thừa từ AnimatedWidget và triển khai phương thức build.

Cơ chế hoạt động của AnimatedWidget như sau:

* Nó nhận giá trị từ một hoặc nhiều đối tượng Animation, chịu trách nhiệm cung cấp các giá trị thay đổi theo thời gian.
* Mỗi khi giá trị của Animation thay đổi, AnimatedWidget sẽ được xây dựng lại thông qua phương thức build, cho phép giao diện người dùng cập nhật và hiển thị các thay đổi hoạt hình một cách tự động.
* AnimatedWidget giúp tách biệt logic hoạt hình ra khỏi widget giao diện, làm cho code dễ đọc và dễ bảo trì hơn. Nó thường được sử dụng cùng với các lớp AnimationController, Tween và các lớp khác trong Flutter để tạo các hiệu ứng hoạt hình phong phú trong ứng dụng của bạn.

Ví dụ:

Dưới đây là một ví dụ về cách sử dụng AnimatedWidget trong Flutter để tạo một hiệu ứng hoạt hình đơn giản. Chúng ta sẽ tạo một ứng dụng đơn giản với một hình vuông có thể thay đổi kích thước bằng cách sử dụng một nút.

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class SizeAnimation extends Animation<double> {

final double begin;

final double end;

SizeAnimation({this.begin, this.end, AnimationController controller})

: super(listenable: controller);

@override

double get value => lerpDouble(begin, end, controller.value);

}

class MyAnimation extends AnimatedWidget {

MyAnimation({Key key, Animation<double> animation})

: super(key: key, listenable: animation);

@override

Widget build(BuildContext context) {

final Animation<double> animation = listenable;

return Center(

child: Container(

width: animation.value,

height: animation.value,

color: Colors.blue,

),

);

}

}

class MyApp extends StatefulWidget {

@override

\_MyAppState createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> with SingleTickerProviderStateMixin {

AnimationController \_controller;

SizeAnimation \_animation;

@override

void initState() {

super.initState();

\_controller = AnimationController(

vsync: this,

duration: Duration(seconds: 1),

);

\_animation = SizeAnimation(

begin: 100.0,

end: 300.0,

controller: \_controller,

);

}

@override

void dispose() {

\_controller.dispose();

super.dispose();

}

@override

Widget build(BuildContext context) {

return MaterialApp(

home: Scaffold(

appBar: AppBar(title: Text('AnimatedWidget Example')),

body: MyAnimation(animation: \_animation),

floatingActionButton: FloatingActionButton(

onPressed: () {

if (\_controller.status == AnimationStatus.completed) {

\_controller.reverse();

} else {

\_controller.forward();

}

},

child: Icon(Icons.play\_arrow),

),

),

);

}

}

1. AnimatedBuilder

AnimatedBuilder cũng là một khái niệm quan trọng trong Flutter liên quan đến việc tạo hiệu ứng hoạt hình. Nó cũng giúp tách biệt logic hoạt hình ra khỏi widget giao diện, làm cho code dễ đọc và dễ bảo trì hơn.

Về cơ bản, AnimatedBuilder giúp bạn tạo một widget con sử dụng các giá trị hoạt hình (Animation) và xây dựng lại chỉ widget con này khi giá trị của Animation thay đổi. Điều này giúp tránh việc xây dựng lại toàn bộ widget cây khi chỉ một số thuộc tính thay đổi.

Ví dụ: cách sử dụng AnimatedBuilder trong Flutter để tạo hiệu ứng hoạt hình mượt mà khi thay đổi màu sắc của hình vuông

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class ColorAnimation extends Animation<Color> {

final Color beginColor;

final Color endColor;

ColorAnimation({this.beginColor, this.endColor, AnimationController controller})

: super(listenable: controller);

@override

Color get value => Color.lerp(beginColor, endColor, controller.value);

}

class MyApp extends StatefulWidget {

@override

\_MyAppState createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> with SingleTickerProviderStateMixin {

AnimationController \_controller;

ColorAnimation \_animation;

@override

void initState() {

super.initState();

\_controller = AnimationController(

vsync: this,

duration: Duration(seconds: 1),

);

\_animation = ColorAnimation(

beginColor: Colors.blue,

endColor: Colors.red,

controller: \_controller,

);

}

@override

void dispose() {

\_controller.dispose();

super.dispose();

}

@override

Widget build(BuildContext context) {

return MaterialApp(

home: Scaffold(

appBar: AppBar(title: Text('AnimatedBuilder Example')),

body: Center(

child: AnimatedBuilder(

animation: \_animation,

builder: (BuildContext context, Widget child) {

return Container(

width: 100,

height: 100,

color: \_animation.value,

);

},

),

),

floatingActionButton: FloatingActionButton(

onPressed: () {

if (\_controller.status == AnimationStatus.completed) {

\_controller.reverse();

} else {

\_controller.forward();

}

},

child: Icon(Icons.play\_arrow),

),

),

);

}

}

1. AnimationController

AnimationController là một lớp trong Flutter, nó là một đối tượng dùng để điều khiển các hoạt hình (animations). Nó giúp bạn quản lý thời gian và trạng thái của animation.

Khi tạo một animation, bạn sẽ cần tạo một AnimationController để điều khiển quá trình chạy của animation. AnimationController cung cấp các phương thức để bắt đầu, dừng, ngược chiều và kiểm tra trạng thái của animation.

Một số thuộc tính và phương thức quan trọng của AnimationController là:

* AnimationController({ double value, Duration duration, this.vsync, this.lowerBound, this.upperBound }): Constructor để tạo một đối tượng AnimationController. Đối số vsync là một SingleTickerProviderStateMixin hoặc TickerProviderStateMixin để đồng bộ với v-sync (vertical synchronization) của thiết bị. LowerBound và upperBound là giới hạn dưới và giới hạn trên của giá trị animation, mặc định là 0.0 và 1.0.
* double value: Thuộc tính để lấy hoặc thiết lập giá trị hiện tại của animation.
* Duration duration: Thuộc tính để lấy hoặc thiết lập thời gian thực hiện animation từ giá trị ban đầu đến giá trị kết thúc.
* forward(): Phương thức để bắt đầu chạy animation theo hướng tăng giá trị (từ giá trị ban đầu đến giá trị kết thúc).
* reverse(): Phương thức để bắt đầu chạy animation theo hướng giảm giá trị (từ giá trị kết thúc về giá trị ban đầu).
* stop(): Phương thức để dừng animation tại vị trí hiện tại.
* addListener(VoidCallback listener): Phương thức để thêm một hàm lắng nghe (listener) để lắng nghe sự thay đổi giá trị của animation.
* removeListener(VoidCallback listener): Phương thức để loại bỏ hàm lắng nghe đã được thêm trước đó.

Ví dụ: sử dụng AnimationController để thay đổi màu sắc của một hình vuông khi người dùng nhấp vào nó. Khi người dùng nhấp vào, animation sẽ chạy từ màu xanh đến màu đỏ và ngược lại.

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatefulWidget {

@override

\_MyAppState createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> with SingleTickerProviderStateMixin {

AnimationController \_controller;

@override

void initState() {

super.initState();

\_controller = AnimationController(

vsync: this,

duration: Duration(seconds: 2),

);

}

@override

void dispose() {

\_controller.dispose();

super.dispose();

}

@override

Widget build(BuildContext context) {

return MaterialApp(

home: Scaffold(

appBar: AppBar(title: Text('AnimationController Example')),

body: Center(

child: Container(

width: 100,

height: 100,

color: Colors.blue,

child: GestureDetector(

onTap: () {

if (\_controller.status == AnimationStatus.completed) {

\_controller.reverse();

} else {

\_controller.forward();

}

},

child: Center(

child: Text(

'Tap me!',

style: TextStyle(color: Colors.white),

),

),

),

),

),

),

);

}

}

1. CuvedAnimation

Trong Flutter, CurvedAnimation là một lớp dùng để tạo các hoạt hình (animations) có hiệu ứng giảm/giãn tốc, chậm/tăng tốc dựa trên một đường cong xác định.

Khi bạn tạo một hoạt hình với CurvedAnimation, bạn cần cung cấp một đối tượng AnimationController và một đường cong (curve) xác định cách giá trị animation thay đổi theo thời gian.

CurvedAnimation được sử dụng chủ yếu làm đối số đầu vào cho các Tween, để tạo ra hiệu ứng chậm/tăng tốc khi animation diễn ra.

Một số điểm quan trọng cần nhớ về CurvedAnimation:

* Curves: Flutter cung cấp sẵn một số đường cong xây sẵn (curves) trong lớp Curves. Bạn có thể sử dụng chúng để tạo hiệu ứng chậm/tăng tốc cho animation. Ví dụ: Curves.linear, Curves.easeIn, Curves.easeOut, Curves.easeInOut, vv.
* Thuộc tính curve: CurvedAnimation có một thuộc tính curve để lưu trữ đường cong dùng để điều khiển tốc độ của animation.

Ví dụ: sử dụng CurvedAnimation với đường cong Curves.easeInOut để tạo hiệu ứng chậm/tăng tốc khi animation diễn ra. Khi bạn nhấp vào hình vuông, nó sẽ chuyển màu từ màu xanh đến màu đỏ và ngược lại với hiệu ứng chậm/tăng tốc

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatefulWidget {

@override

\_MyAppState createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> with SingleTickerProviderStateMixin {

AnimationController \_controller;

CurvedAnimation \_curvedAnimation;

@override

void initState() {

super.initState();

\_controller = AnimationController(

vsync: this,

duration: Duration(seconds: 2),

);

\_curvedAnimation = CurvedAnimation(

parent: \_controller,

curve: Curves.easeInOut, // Sử dụng đường cong easeInOut cho hoạt hình

);

}

@override

void dispose() {

\_controller.dispose();

super.dispose();

}

@override

Widget build(BuildContext context) {

return MaterialApp(

home: Scaffold(

appBar: AppBar(title: Text('CurvedAnimation Example')),

body: Center(

child: Container(

width: 100,

height: 100,

color: Colors.blue,

child: GestureDetector(

onTap: () {

if (\_controller.status == AnimationStatus.completed) {

\_controller.reverse();

} else {

\_controller.forward();

}

},

child: Center(

child: Text(

'Tap me!',

style: TextStyle(color: Colors.white),

),

),

),

),

),

),

);

}

}

1. Hero

Trong Flutter, Hero là một widget giúp thực hiện các hoạt hình chuyển đổi mượt mà giữa hai màn hình (screens) khi chuyển đổi giữa chúng. Điểm mạnh của Hero là nó tạo ra các hiệu ứng hoạt hình hấp dẫn khi chuyển đổi giữa hai màn hình, giúp người dùng có trải nghiệm tốt hơn trong ứng dụng.

Widget Hero thường được sử dụng trong cặp, với một Hero ở màn hình đầu tiên (màn hình gốc) và một Hero khác ở màn hình tiếp theo (màn hình đích). Cả hai Hero này có cùng thuộc tính heroTag, nó dùng để xác định cặp Hero và kết nối chúng lại với nhau.

Khi chuyển từ màn hình gốc đến màn hình đích, Hero sẽ tự động tạo một hoạt hình di chuyển cho đối tượng được chia sẻ giữa hai màn hình. Điều này tạo ra hiệu ứng mượt mà giống như một "an toàn bay" cho đối tượng đó.

Ví dụ: sử dụng widget Hero để tạo hiệu ứng chuyển đổi giữa hai màn hình. Khi người dùng nhấp vào hình ảnh trong màn hình đầu tiên, nó sẽ chuyển sang màn hình thứ hai và hiệu ứng di chuyển (Hero Animation) sẽ xảy ra khi hình ảnh tăng kích thước từ màn hình gốc đến màn hình đích.

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: HomePage(),

routes: {

'/second': (context) => SecondPage(),

},

);

}

}

class HomePage extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Hero Animation')),

body: Center(

child: GestureDetector(

onTap: () {

Navigator.pushNamed(context, '/second');

},

child: Hero(

tag: 'imageHero', // Đảm bảo heroTag này giống nhau trong cả hai màn hình

child: Image.network(

'https://via.placeholder.com/200',

width: 100,

height: 100,

),

),

),

),

);

}

}

class SecondPage extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Second Page')),

body: Center(

child: GestureDetector(

onTap: () {

Navigator.pop(context);

},

child: Hero(

tag: 'imageHero', // Đảm bảo heroTag này giống nhau trong cả hai màn hình

child: Image.network(

'https://via.placeholder.com/300', // Ảnh lớn hơn để tạo hiệu ứng di chuyển

width: 300,

height: 300,

),

),

),

),

);

}

}

1. Opacity

Trong Flutter, Opacity là một widget được sử dụng để điều chỉnh độ mờ của một widget con. Nó cho phép bạn thay đổi độ trong suốt của widget từ 0.0 (hoàn toàn trong suốt) đến 1.0 (hoàn toàn không trong suốt). Việc điều chỉnh độ trong suốt này có thể giúp bạn tạo các hiệu ứng như xuất hiện dần, biến mất dần hoặc làm mờ một widget.Để sử dụng Opacity trong Flutter, bạn chỉ cần bao bọc widget cần điều chỉnh độ trong suốt bên trong widget Opacity và đặt giá trị opacity tương ứng. Giá trị opacity là một số thực từ 0.0 đến 1.0.

Ví dụ: sử dụng widget Opacity để thay đổi độ trong suốt của một hình vuông màu xanh. Khi nhấn nút "Hide" hoặc "Show", hình vuông sẽ biến mất hoặc xuất hiện dần dựa vào giá trị opacity được thay đổi.

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: HomePage(),

);

}

}

class HomePage extends StatefulWidget {

@override

\_HomePageState createState() => \_HomePageState();

}

class \_HomePageState extends State<HomePage> {

bool \_visible = true;

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Opacity Example')),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Opacity(

opacity: \_visible ? 1.0 : 0.0, // Điều chỉnh độ trong suốt của widget

duration: Duration(seconds: 1), // Thời gian hoạt hình khi thay đổi opacity

child: Container(

width: 100,

height: 100,

color: Colors.blue,

),

),

SizedBox(height: 20),

ElevatedButton(

onPressed: () {

setState(() {

\_visible = !\_visible; // Thay đổi giá trị opacity khi nhấn nút

});

},

child: Text(\_visible ? 'Hide' : 'Show'),

),

],

),

),

);

}

}