**1. AnimatedWidget:**

Là một lớp trừu tượng trong Flutter, được sử dụng để xây dựng các widget động (animated). Điểm mạnh của AnimatedWidget là nó giúp tự động quản lý các thay đổi giá trị trong việc thay đổi trạng thái của widget, giúp bạn tập trung vào việc xây dựng giao diện người dùng và không cần lo lắng về việc quản lý hoạt hình.

Ví dụ:

class AnimatedLogo extends AnimatedWidget {

AnimatedLogo({Key? key, required Animation<double> animation})

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

@override

Widget build(BuildContext context) {

final Animation<double> animation = listenable as Animation<double>;

return Center(

child: Container(

width: animation.value,

height: animation.value,

child: FlutterLogo(),

),

);

}

}

2. AnimatedBuilder:

AnimatedBuilder cũng giống như AnimatedWidget, tuy nhiên, nó không phải là một widget trực tiếp mà nó cho phép bạn xây dựng một widget thông qua callback builder khi giá trị animation thay đổi. Điều này cho phép bạn tùy chỉnh hơn cách widget được xây dựng khi thay đổi trạng thái.

Ví dụ:

class AnimatedLogo extends StatefulWidget {

@override

\_AnimatedLogoState createState() => \_AnimatedLogoState();

}

class \_AnimatedLogoState extends State<AnimatedLogo>

with SingleTickerProviderStateMixin {

late AnimationController \_controller;

late Animation<double> \_animation;

@override

void initState() {

super.initState();

\_controller = AnimationController(

vsync: this,

duration: Duration(seconds: 2),

);

\_animation = CurvedAnimation(

parent: \_controller,

curve: Curves.easeInOut,

);

\_controller.forward();

}

@override

Widget build(BuildContext context) {

return AnimatedBuilder(

animation: \_animation,

builder: (context, child) {

return Center(

child: Container(

width: \_animation.value \* 200,

height: \_animation.value \* 200,

child: FlutterLogo(),

),

);

},

);

}

@override

void dispose() {

\_controller.dispose();

super.dispose();

}

}

**3. AnimationController:**

Là một lớp quản lý cho phép điều khiển các hoạt hình. Nó cung cấp các phương thức để bắt đầu, dừng, chuyển tiếp và ngược lại các hoạt hình, cũng như cơ chế để định lượng thời gian hoạt hình.

Ví dụ:

class AnimatedLogo extends StatefulWidget {

@override

\_AnimatedLogoState createState() => \_AnimatedLogoState();

}

class \_AnimatedLogoState extends State<AnimatedLogo>

with SingleTickerProviderStateMixin {

late AnimationController \_controller;

@override

void initState() {

super.initState();

\_controller = AnimationController(

vsync: this,

duration: Duration(seconds: 2),

);

\_controller.forward();

}

@override

Widget build(BuildContext context) {

return Center(

child: AnimatedBuilder(

animation: \_controller,

builder: (context, child) {

return Transform.scale(

scale: \_controller.value \* 2.0,

child: FlutterLogo(),

);

},

),

);

}

@override

void dispose() {

\_controller.dispose();

super.dispose();

}

}

**4. CurvedAnimation:**

CurvedAnimation là một lớp con của Animation<double>, cho phép bạn tạo ra các hiệu ứng hoạt hình cong hay chậm dần. Bạn có thể sử dụng các lớp con của Curves (như Curves.easeInOut trong ví dụ trên) để xác định các đường cong hiệu ứng.

**5.Hero:**

Là một widget cho phép bạn thực hiện chuyển đổi mượt mà giữa hai widget giống nhau giữa các màn hình. Nó hỗ trợ việc chuyển đổi các thuộc tính như kích thước, vị trí và màu sắc giữa hai widget trong các màn hình khác nhau.

Ví dụ:

class FirstScreen extends StatelessWidget {

@override

Widget build(BuildContext context) {

return GestureDetector(

onTap: () {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => SecondScreen(),

),

);

},

child: Hero(

tag: 'logoTag',

child: FlutterLogo(size: 100),

),

);

}

}

class SecondScreen extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

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

body: Center(

child: GestureDetector(

onTap: () {

Navigator.pop(context);

},

child: Hero(

tag: 'logoTag',

child: FlutterLogo(size: 200),

),

),

),

);

}

}

**6. Opacity:**

Là một widget dùng để điều chỉnh độ trong suốt của widget con bên trong nó. Giá trị của Opacity nằm trong khoảng từ 0.0 (hoàn toàn trong suốt) đến 1.0 (hoàn toàn không trong suốt).

Ví dụ:

class MyWidget extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Opacity(

opacity: 0.5,

child: Container(

width: 100,

height: 100,

color: Colors.blue,

),

);

}

}