**NỘI DUNG ANIMATIONS:**

1. **AnimatedWidget** là một lớp con của Flutter được sử dụng để tạo widget có thể cập nhật giao diện người dùng một cách tự động khi giá trị của một animation thay đổi. Nó giúp tách biệt logic của animation và cập nhật giao diện, giúp mã nguồn trở nên dễ đọc và dễ quản lý hơn.

Ví dụ về cách sử dụng AnimatedWidget để tạo một widget đơn giản mà thay đổi kích thước của một hình tròn dựa trên giá trị của animation:

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

*void main() {*

*runApp(MyApp());*

*}*

*class AnimatedCircle extends AnimatedWidget {*

*AnimatedCircle({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,*

*decoration: BoxDecoration(*

*shape: BoxShape.circle,*

*color: Colors.blue,*

*),*

*),*

*);*

*}*

*}*

*class MyApp extends StatelessWidget {*

*@override*

*Widget build(BuildContext context) {*

*final AnimationController controller = AnimationController(*

*vsync: context.vsync,*

*duration: Duration(seconds: 2),*

*)..forward();*

*final Animation<double> animation = Tween<double>(*

*begin: 50.0,*

*end: 200.0,*

*).animate(controller);*

*return MaterialApp(*

*home: Scaffold(*

*appBar: AppBar(*

*title: Text('AnimatedWidget Example'),*

*),*

*body: AnimatedCircle(animation: animation),*

*),*

*);*

*}*

*}*

AnimatedCircle là một lớp kế thừa từ AnimatedWidget. Nó nhận một đối tượng Animation<double> và cập nhật kích thước của hình tròn dựa trên giá trị animation. Khi animation thay đổi, widget này sẽ tự động cập nhật giao diện người dùng để phản ánh sự thay đổi trong giá trị animation.

1. **AnimatedBuilder** là một widget trong Flutter được sử dụng để tái sử dụng phần giao diện người dùng và xây dựng lại nó khi giá trị của animation thay đổi. Điều này giúp tối ưu hóa việc cập nhật giao diện và tránh việc tái render toàn bộ widget khi chỉ có một phần nhỏ của giao diện thay đổi.

Ví dụ về cách sử dụng AnimatedBuilder để tạo một widget đơn giản có thể thay đổi kích thước của một hình tròn bằng cách sử dụng animation:

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

*void main() {*

*runApp(MyApp());*

*}*

*class AnimatedCircle extends StatefulWidget {*

*@override*

*\_AnimatedCircleState createState() => \_AnimatedCircleState();*

*}*

*class \_AnimatedCircleState extends State<AnimatedCircle>*

*with SingleTickerProviderStateMixin {*

*late AnimationController \_controller;*

*late Animation<double> \_animation;*

*@override*

*void initState() {*

*super.initState();*

*\_controller = AnimationController(*

*vsync: this,*

*duration: Duration(seconds: 2),*

*)..forward();*

*\_animation = Tween<double>(*

*begin: 50.0,*

*end: 200.0,*

*).animate(\_controller);*

*}*

*@override*

*Widget build(BuildContext context) {*

*return Center(*

*child: AnimatedBuilder(*

*animation: \_animation,*

*builder: (context, child) {*

*return Container(*

*width: \_animation.value,*

*height: \_animation.value,*

*decoration: BoxDecoration(*

*shape: BoxShape.circle,*

*color: Colors.blue,*

*),*

*);*

*},*

*),*

*);*

*}*

*@override*

*void dispose() {*

*\_controller.dispose();*

*super.dispose();*

*}*

*}*

*class MyApp extends StatelessWidget {*

*@override*

*Widget build(BuildContext context) {*

*return MaterialApp(*

*home: Scaffold(*

*appBar: AppBar(*

*title: Text('AnimatedBuilder Example'),*

*),*

*body: AnimatedCircle(),*

*),*

*);*

*}*

*}*

AnimatedCircle là một widget có trạng thái được tạo ra từ StatefulWidget. Nó sử dụng AnimationController để quản lý animation và tạo một Animation<double> để định rõ giá trị thay đổi của animation.

Trong phần build của AnimatedCircle, sử dụng AnimatedBuilder để xây dựng lại phần giao diện người dùng muốn thay đổi khi animation thay đổi. Trong trường hợp này, xây dựng lại hình tròn với kích thước phụ thuộc vào giá trị animation.

1. **AnimationController** là một lớp trong Flutter được sử dụng để quản lý và điều khiển các animation. Nó cung cấp các phương thức để bắt đầu, dừng, tạm dừng và thay đổi giá trị của animation.

Ví dụ về cách sử dụng AnimationController để tạo một animation đơn giản, làm thay đổi kích thước của một hình tròn:

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

*void main() {*

*runApp(MyApp());*

*}*

*class MyApp extends StatelessWidget {*

*@override*

*Widget build(BuildContext context) {*

*return MaterialApp(*

*home: Scaffold(*

*appBar: AppBar(*

*title: Text('AnimationController Example'),*

*),*

*body: AnimationExample(),*

*),*

*);*

*}*

*}*

*class AnimationExample extends StatefulWidget {*

*@override*

*\_AnimationExampleState createState() => \_AnimationExampleState();*

*}*

*class \_AnimationExampleState extends State<AnimationExample>*

*with SingleTickerProviderStateMixin {*

*late AnimationController \_controller;*

*late Animation<double> \_animation;*

*@override*

*void initState() {*

*super.initState();*

*\_controller = AnimationController(*

*vsync: this,*

*duration: Duration(seconds: 2),*

*);*

*\_animation = Tween<double>(*

*begin: 50.0,*

*end: 200.0,*

*).animate(\_controller);*

*\_controller.forward();*

*}*

*@override*

*Widget build(BuildContext context) {*

*return Center(*

*child: AnimatedBuilder(*

*animation: \_animation,*

*builder: (context, child) {*

*return Container(*

*width: \_animation.value,*

*height: \_animation.value,*

*decoration: BoxDecoration(*

*shape: BoxShape.circle,*

*color: Colors.blue,*

*),*

*);*

*},*

*),*

*);*

*}*

*@override*

*void dispose() {*

*\_controller.dispose();*

*super.dispose();*

*}*

*}*

Trong ví dụ này, sử dụng AnimationController để tạo và điều khiển animation. Trong phần initState(), chúng ta tạo một AnimationController với vsync là this để đảm bảo nó tương thích với vòng đời của widget và đặt thời gian duration của animation là 2 giây.

Chúng ta cũng tạo một Animation<double> thông qua Tween, định nghĩa giá trị bắt đầu và giá trị kết thúc của animation.

Sau khi tạo xong, chúng ta sử dụng \_controller.forward() để bắt đầu animation.

Trong phần build, chúng ta sử dụng AnimatedBuilder để xây dựng lại phần giao diện người dùng mà chúng ta muốn thay đổi khi animation thay đổi.

1. **CurvedAnimation** là một lớp trong Flutter dùng để tạo ra hiệu ứng chuyển động theo đường cong. Nó thường được sử dụng để làm cho animation trông tự nhiên hơn bằng cách áp dụng các hàm đường cong vào animation. Đường cong có thể được điều chỉnh để thay đổi tốc độ của animation theo thời gian.

Ví dụ về cách sử dụng CurvedAnimation để tạo một animation di chuyển vật thể theo đường cong khi được kích hoạt:

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

*void main() {*

*runApp(MyApp());*

*}*

*class MyApp extends StatelessWidget {*

*@override*

*Widget build(BuildContext context) {*

*return MaterialApp(*

*home: Scaffold(*

*appBar: AppBar(*

*title: Text('CurvedAnimation Example'),*

*),*

*body: CurvedAnimationExample(),*

*),*

*);*

*}*

*}*

*class CurvedAnimationExample extends StatefulWidget {*

*@override*

*\_CurvedAnimationExampleState createState() => \_CurvedAnimationExampleState();*

*}*

*class \_CurvedAnimationExampleState extends State<CurvedAnimationExample>*

*with SingleTickerProviderStateMixin {*

*late AnimationController \_controller;*

*late Animation<double> \_animation;*

*late Animation<double> \_curvedAnimation;*

*@override*

*void initState() {*

*super.initState();*

*\_controller = AnimationController(*

*vsync: this,*

*duration: Duration(seconds: 2),*

*);*

*\_animation = Tween<double>(*

*begin: 0.0,*

*end: 1.0,*

*).animate(\_controller);*

*\_curvedAnimation = CurvedAnimation(*

*parent: \_animation,*

*curve: Curves.easeInOut, // Hàm đường cong có thể thay đổi*

*);*

*\_controller.forward();*

*}*

*@override*

*Widget build(BuildContext context) {*

*return Center(*

*child: SlideTransition(*

*position: Tween<Offset>(*

*begin: Offset(0, 1), // Vị trí ban đầu*

*end: Offset(0, 0), // Vị trí kết thúc*

*).animate(\_curvedAnimation),*

*child: Container(*

*width: 100,*

*height: 100,*

*color: Colors.blue,*

*),*

*),*

*);*

*}*

*@override*

*void dispose() {*

*\_controller.dispose();*

*super.dispose();*

*}*

*}*

Ví dụ này, sử dụng CurvedAnimation để tạo animation theo đường cong khi chuyển động .Tạo một AnimationController và một Animation<double> thông qua Tween, sau đó tạo một CurvedAnimation sử dụng Curves.easeInOut để làm cho animation bắt đầu và kết thúc một cách mượt mà.

Sử dụng SlideTransition để áp dụng animation di chuyển theo đường cong cho một hình vuông.

1. **Hero** là một widget trong Flutter được sử dụng để tạo hiệu ứng chuyển đổi mượt mà giữa các màn hình bằng cách chia sẻ animation giữa hai widget có cùng tag trên hai màn hình khác nhau.

Ví dụ về cách sử dụng Hero để tạo một hiệu ứng chuyển đổi mượt mà giữa hai màn hình:

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

*void main() {*

*runApp(MyApp());*

*}*

*class MyApp extends StatelessWidget {*

*@override*

*Widget build(BuildContext context) {*

*return MaterialApp(*

*home: Screen1(),*

*);*

*}*

*}*

*class Screen1 extends StatelessWidget {*

*@override*

*Widget build(BuildContext context) {*

*return Scaffold(*

*appBar: AppBar(*

*title: Text('Hero Example - Screen 1'),*

*),*

*body: GestureDetector(*

*onTap: () {*

*Navigator.push(*

*context,*

*MaterialPageRoute(builder: (context) => Screen2()),*

*);*

*},*

*child: Hero(*

*tag: 'hero-tag',*

*child: Container(*

*width: 100,*

*height: 100,*

*color: Colors.blue,*

*),*

*),*

*),*

*);*

*}*

*}*

*class Screen2 extends StatelessWidget {*

*@override*

*Widget build(BuildContext context) {*

*return Scaffold(*

*appBar: AppBar(*

*title: Text('Hero Example - Screen 2'),*

*),*

*body: Center(*

*child: Hero(*

*tag: 'hero-tag',*

*child: Container(*

*width: 200,*

*height: 200,*

*color: Colors.blue,*

*),*

*),*

*),*

*);*

*}*

*}*

Ví dụ này, sử dụng Hero để tạo hiệu ứng chuyển đổi mượt mà giữa hai màn hình Screen1 và Screen2. Widget hình vuông trên Screen1 và Screen2 có cùng tag là 'hero-tag'. Khi chuyển từ Screen1 sang Screen2, Hero sẽ tạo hiệu ứng animation mở rộng hình vuông từ kích thước ban đầu trên Screen1 đến kích thước trên Screen2.

Chú ý rằng: để sử dụng Hero, các widget có cùng tag phải có thuộc tính key duy nhất. Trong ví dụ này, chúng ta không cần phải tự tạo key vì Flutter sẽ tự động tạo và quản lý chúng.

1. **Opacity** là một widget trong Flutter được sử dụng để thay đổi độ mờ của các widget con bên trong dựa trên giá trị từ 0.0 (hoàn toàn trong suốt) đến 1.0 (hoàn toàn rõ nét). Điều này cho phép bạn tạo các hiệu ứng làm mờ hoặc hiển thị dần dần cho các phần của giao diện người dùng.

Ví dụ về cách sử dụng Opacity để thay đổi độ mờ của một hình vuông:

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

*void main() {*

*runApp(MyApp());*

*}*

*class MyApp extends StatelessWidget {*

*@override*

*Widget build(BuildContext context) {*

*return MaterialApp(*

*home: OpacityExample(),*

*);*

*}*

*}*

*class OpacityExample extends StatefulWidget {*

*@override*

*\_OpacityExampleState createState() => \_OpacityExampleState();*

*}*

*class \_OpacityExampleState extends State<OpacityExample> {*

*double \_opacityLevel = 1.0;*

*void \_toggleOpacity() {*

*setState(() {*

*\_opacityLevel = \_opacityLevel == 1.0 ? 0.2 : 1.0;*

*});*

*}*

*@override*

*Widget build(BuildContext context) {*

*return Scaffold(*

*appBar: AppBar(*

*title: Text('Opacity Example'),*

*),*

*body: Center(*

*child: Column(*

*mainAxisAlignment: MainAxisAlignment.center,*

*children: <Widget>[*

*AnimatedOpacity(*

*opacity: \_opacityLevel,*

*duration: Duration(seconds: 1),*

*child: Container(*

*width: 200,*

*height: 200,*

*color: Colors.blue,*

*),*

*),*

*SizedBox(height: 20),*

*ElevatedButton(*

*onPressed: \_toggleOpacity,*

*child: Text('Toggle Opacity'),*

*),*

*],*

*),*

*),*

*);*

*}*

*}*

Ví dụ này, sử dụng AnimatedOpacity để tạo hiệu ứng thay đổi độ mờ của một hình vuông. AnimatedOpacity nhận giá trị opacity để xác định độ mờ của widget bên trong. Khi thay đổi giá trị của \_opacityLevel (0.2 hoặc 1.0), widget sẽ được hiển thị hoặc ẩn đi một cách mượt mà trong khoảng thời gian được xác định bởi duration.