Họ và Tên: **Nguyễn Phương Nga**

Lớp: **08-THMT**

MSSV: **0850080033**

**LÝ THUYẾT BUỔI 7**

1. **AnimatedWidget**

AnimatedWidget là một lớp cơ bản trong Flutter, được sử dụng để tạo các widget có hiệu ứng hoạt hình dễ dàng. Trong Flutter, các hiệu ứng hoạt hình thường dựa vào việc thay đổi giá trị của các thuộc tính, chẳng hạn như chiều cao, chiều rộng, màu sắc, hoặc vị trí của widget. AnimatedWidget giúp bạn tách biệt việc quản lý trạng thái và giao diện người dùng, giúp cho việc tạo các hiệu ứng hoạt hình trở nên dễ dàng và dễ quản lý hơn.

Khi sử dụng AnimatedWidget, bạn cần kế thừa từ lớp AnimatedWidget và triển khai phương thức build() để xây dựng giao diện của widget dựa trên giá trị của các thuộc tính hoạt hình. Điểm quan trọng là bạn không cần phải quản lý trực tiếp AnimationController, việc điều khiển hoạt hình sẽ được thực hiện tự động bởi AnimatedWidget.

Cách thức hoạt động của AnimatedWidget như sau:

* AnimatedWidget nhận các giá trị hoặc thuộc tính cần theo dõi (ví dụ: animation) thông qua constructor.
* Khi giá trị hoặc thuộc tính này thay đổi, AnimatedWidget sẽ thực hiện một rebuild và gọi hàm build().
* Hàm build() sẽ được gọi bởi framework và trả về widget mới, dựa trên giá trị hiện tại của thuộc tính. Sử dụng AnimatedWidget, bạn có thể tạo ra các hiệu ứng hoạt hình phức tạp, chẳng hạn như biến đổi kích thước, màu sắc, vị trí và các thuộc tính khác của các widget.
* import 'package:flutter/material.dart';
* void main() {
* runApp(MyApp());
* }
* class AnimatedSquare extends StatefulWidget {
* @override
* \_AnimatedSquareState createState() => \_AnimatedSquareState();
* }
* class \_AnimatedSquareState extends State<AnimatedSquare>
* with SingleTickerProviderStateMixin {
* late AnimationController \_animationController;
* late Animation<Offset> \_animation;
* @override
* void initState() {
* super.initState();
* *// Tạo AnimationController với thời gian hoạt hình là 1 giây*
* \_animationController = AnimationController(
* vsync: this,
* duration: Duration(seconds: 1),
* );
* *// Tạo tween để di chuyển từ điểm (0,0) đến điểm (1,1)*
* \_animation = Tween<Offset>(begin: Offset(0, 0), end: Offset(1, 1))
* .animate(\_animationController);
* *// Bắt đầu hoạt hình*
* \_animationController.forward();
* }
* @override
* void dispose() {
* \_animationController.dispose();
* super.dispose();
* }
* @override
* Widget build(BuildContext context) {
* return AnimatedBuilder(
* animation: \_animation,
* builder: (context, child) {
* *// Di chuyển hình vuông dựa vào giá trị của \_animation*
* return FractionalTranslation(
* translation: \_animation.value,
* child: Container(
* width: 100,
* height: 100,
* color: Colors.blue,
* ),
* );
* },
* );
* }
* }
* class MyApp extends StatelessWidget {
* @override
* Widget build(BuildContext context) {
* return MaterialApp(
* home: Scaffold(
* appBar: AppBar(
* title: Text('AnimatedWidget Example'),
* ),
* body: Center(
* child: AnimatedSquare(),
* ),
* ),
* );
* }
* }

1. **AnimatedBuilder**

AnimatedBuilder là một widget trong Flutter, được sử dụng để tạo các hiệu ứng hoạt hình. Tương tự như AnimatedWidget, AnimatedBuilder cũng giúp tạo các hiệu ứng hoạt hình dễ dàng, nhưng với một sự khác biệt quan trọng. Thay vì kế thừa từ AnimatedWidget, bạn sử dụng AnimatedBuilder như một hàm có tham số là một widget builder.

Cơ chế hoạt động của AnimatedBuilder:

* builder callback: Trong AnimatedBuilder, bạn cung cấp một hàm callback builder, nhận vào hai tham số là BuildContext và Widget?. Hàm này sẽ được gọi lại mỗi khi giá trị của animation thay đổi.
* listenForAnimation: Trong constructor của AnimatedBuilder, bạn truyền vào một đối tượng Listenable (thường là một Animation). Đối tượng này sẽ lắng nghe và thông báo sự thay đổi giá trị. Khi giá trị hoạt hình thay đổi, AnimatedBuilder sẽ tự động gọi lại hàm builder, giúp cập nhật giao diện của widget.
* return của builder callback: Trong hàm builder, bạn xây dựng giao diện của widget dựa trên giá trị hoạt hình. Điều này cho phép bạn tái sử dụng các widget con không thay đổi giữa các khung hình, trong khi chỉ cập nhật phần hiệu ứng của widget.
* import 'package:flutter/material.dart';
* void main() {
* runApp(MyApp());
* }
* class AnimatedSquare extends StatefulWidget {
* @override
* \_AnimatedSquareState createState() => \_AnimatedSquareState();
* }
* class \_AnimatedSquareState extends State<AnimatedSquare>
* with SingleTickerProviderStateMixin {
* late AnimationController \_animationController;
* late Animation<Offset> \_animation;
* @override
* void initState() {
* super.initState();
* *// Tạo AnimationController với thời gian hoạt hình là 1 giây*
* \_animationController = AnimationController(
* vsync: this,
* duration: Duration(seconds: 1),
* );
* *// Tạo tween để di chuyển từ điểm (0,0) đến điểm (1,0)*
* \_animation = Tween<Offset>(begin: Offset(0, 0), end: Offset(1, 0))
* .animate(\_animationController);
* *// Thêm listener để theo dõi sự kết thúc của hoạt hình*
* \_animationController.addStatusListener((status) {
* if (status == AnimationStatus.completed) {
* *// Khi hoạt hình kết thúc, đặt lại giá trị hoạt hình về ban đầu*
* \_animationController.reverse();
* } else if (status == AnimationStatus.dismissed) {
* *// Khi hoạt hình đảo ngược kết thúc, chạy lại hoạt hình*
* \_animationController.forward();
* }
* });
* *// Bắt đầu hoạt hình*
* \_animationController.forward();
* }
* @override
* void dispose() {
* \_animationController.dispose();
* super.dispose();
* }
* @override
* Widget build(BuildContext context) {
* return AnimatedBuilder(
* animation: \_animationController,
* builder: (context, child) {
* *// Di chuyển hình vuông dựa vào giá trị của \_animation*
* return FractionalTranslation(
* translation: \_animation.value,
* child: Container(
* width: 100,
* height: 100,
* color: Colors.blue,
* ),
* );
* },
* );
* }
* }
* class MyApp extends StatelessWidget {
* @override
* Widget build(BuildContext context) {
* return MaterialApp(
* home: Scaffold(
* appBar: AppBar(
* title: Text('AnimatedBuilder Example'),
* ),
* body: Center(
* child: AnimatedSquare(),
* ),
* ),
* );
* }
* }

# **AnimationController**

Trong Dart, AnimationController là một lớp được sử dụng để quản lý các hoạt động hoạt hình. Nó giúp bạn tạo ra và điều khiển các giá trị hoạt động, như các giá trị độ rộng, độ cao, màu sắc, và vị trí của các thành phần trong một hoạt động. AnimationController có thể được sử dụng để tạo các hoạt động liên tục hoặc disktrét trong ứng dụng của bạn. Bạn có thể sử dụng nó để tạo ra các hiệu ứng như di chuyển, scale, rotate, thay đổi màu sắc và nhiều hơn nữa. Ví dụ, bạn có thể sử dụng AnimationController để điều khiển hoạt ảnh di chuyển của một widget từ điểm A đến điểm B trong một khoảng thời gian nhất định. Bằng cách cung cấp các giá trị bắt đầu và kết thúc, và sử dụng các phương thức của AnimationController như forward(), reverse() hoặc repeat() để bắt đầu, đảo chiều hoặc lặp lại hoạt ảnh. Để sử dụng AnimationController trong Dart, bạn cần import package "flutter/animation.dart".

* import 'package:flutter/material.dart';
* void main() {
* runApp(MyApp());
* }
* class AnimatedSquare extends StatefulWidget {
* @override
* \_AnimatedSquareState createState() => \_AnimatedSquareState();
* }
* class \_AnimatedSquareState extends State<AnimatedSquare>
* with SingleTickerProviderStateMixin {
* late AnimationController \_animationController;
* late Animation<Offset> \_animation;
* @override
* void initState() {
* super.initState();
* *// Tạo AnimationController với thời gian hoạt hình là 1 giây*
* \_animationController = AnimationController(
* vsync: this,
* duration: Duration(seconds: 1),
* );
* *// Tạo tween để di chuyển từ điểm (0,0) đến điểm (0,1)*
* \_animation = Tween<Offset>(begin: Offset(0, 0), end: Offset(0, 1))
* .animate(\_animationController);
* *// Bắt đầu hoạt hình và thêm listener để khi hoạt hình kết thúc, đảo ngược lại*
* \_animationController.forward().then((\_) {
* \_animationController.reverse();
* });
* }
* @override
* void dispose() {
* \_animationController.dispose();
* super.dispose();
* }
* @override
* Widget build(BuildContext context) {
* return AnimatedBuilder(
* animation: \_animation,
* builder: (context, child) {
* *// Di chuyển hình vuông dựa vào giá trị của \_animation*
* return FractionalTranslation(
* translation: \_animation.value,
* child: Container(
  + width: 100,
  + height: 100,
  + color: Colors.blue,
* ),
* );
* },
* );
* }
* }
* class MyApp extends StatelessWidget {
* @override
* Widget build(BuildContext context) {
* return MaterialApp(
* home: Scaffold(
* appBar: AppBar(
* title: Text('AnimationController Example'),
* ),
* body: Center(
* child: AnimatedSquare(),
* ),
* ),
* );
* }
* }

# **CuvedAnimation**

CuvedAnimation là một lớp trong Flutter Framework của Dart, được sử dụng để tạo các hiệu ứng hoạt hình cong dẻo. Lớp này kế thừa từ lớp Animation<double>, cho phép chúng ta điều chỉnh các giá trị Animation thành các giá trị cong theo một hàm cong cụ thể. Với CuvedAnimation, bạn có thể tùy chỉnh các thông số như chiều dài, tốc độ, hình dáng của đường cong, và các thuộc tính khác. Nó thường được sử dụng để tạo các hiệu ứng phức tạp như chuyển động yên ả, mượt mà, hoặc các hiệu ứng cong dẻo trong các ứng dụng Flutter. Để sử dụng CuvedAnimation, bạn cần khởi tạo nó bằng cách cung cấp một đối tượng AnimationController làm tham số và chỉ định hàm cong mà bạn muốn áp dụng. Sau đó, bạn có thể sử dụng CuvedAnimation để lấy giá trị hiện tại của animation và áp dụng nó vào các thành phần giao diện người dùng khác, ví dụ như độ trong suốt, kích thước, hoặc vị trí của một widget.

1. **Hero**

Trong ngôn ngữ lập trình Dart, khái niệm "hero" thường được sử dụng để đề cập đến hiệu ứng "Hero animation". Hero animation là một hiệu ứng chuyển động mượt mà giữa các thành phần UI trong ứng dụng Dart, thường được sử dụng để tạo ra các hiệu ứng chuyển đổi trực quan hấp dẫn. Khi áp dụng hiệu ứng Hero, các thành phần giao diện người dùng sẽ được kết nối với nhau thông qua một "hero tag". Khi chuyển đổi giữa các màn hình hoặc chuyển đổi trạng thái, các thành phần có cùng hero tag sẽ có hiệu ứng chuyển động mượt mà như di chuyển, mở rộng hoặc thu gọn vị trí. Điều này giúp tạo ra trải nghiệm người dùng ấn tượng và giữ cho ứng dụng có giao diện thân thiện hơn. Để sử dụng hiệu ứng hero trong Dart, bạn cần định nghĩa các Hero widget và xác định hero tag cho từng thành phần bạn muốn áp dụng hiệu ứng. Khi chuyển đổi giữa các widget, Flutter sẽ tự động tạo ra hiệu ứng mượt mà dựa trên các hero tag tương ứng. Tổng quan về hero trong Dart là về hiệu ứng Hero animation, giúp tạo ra một trải nghiệm giao diện người dùng mượt mà và ấn tượng.

* import 'package:flutter/material.dart';
* void main() {
* runApp(MyApp());
* }
* class FirstScreen extends StatelessWidget {
* @override
* Widget build(BuildContext context) {
* return Scaffold(
* appBar: AppBar(
* title: Text('First Screen'),
* ),
* body: Center(
* child: GestureDetector(
* onTap: () {
  + Navigator.push(
  + context,
  + MaterialPageRoute(
    - builder: (context) => SecondScreen(),
  + ),
  + );
* },
* child: Hero(
  + tag: 'square\_tag',
  + child: Container(
  + width: 100,
  + height: 100,
  + color: Colors.blue,
  + ),
* ),
* ),
* ),
* );
* }
* }
* 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: 'square\_tag',
  + child: Container(
  + width: 200,
  + height: 200,
  + color: Colors.red,
  + ),
* ),
* ),
* ),
* );
* }
* }
* class MyApp extends StatelessWidget {
* @override
* Widget build(BuildContext context) {
* return MaterialApp(
* home: FirstScreen(),
* );
* }
* }

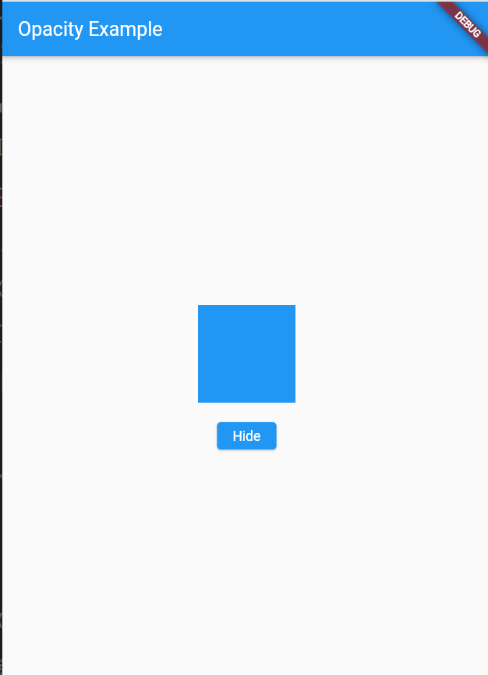
A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

1. **Opacity**

Trong ngôn ngữ lập trình Dart, "opacity" là một thuộc tính được sử dụng để xác định mức độ mờ của một đối tượng hoặc widget. Giá trị của thuộc tính này nằm trong khoảng từ 0.0 đến 1.0, với 0.0 biểu thị cho đối tượng hoàn toàn trong suốt và 1.0 biểu thị cho đối tượng hoàn toàn không trong suốt. Khi bạn đặt giá trị opacity cho một widget, nó sẽ ảnh hưởng đến cả widget hiện tại và tất cả các widget con bên trong. Nếu giá trị opacity là 1.0 (hoặc mặc định), widget sẽ không có hiệu ứng mờ và hiện diện một cách rõ ràng. Ngược lại, khi giá trị opacity gần đến 0.0, widget sẽ trở nên mờ mờ hơn và con số càng gần 0.0 thì widget càng không rõ ràng.

* import 'package:flutter/material.dart';
* void main() {
* runApp(MyApp());
* }
* class AnimatedSquare extends StatefulWidget {
* @override
* \_AnimatedSquareState createState() => \_AnimatedSquareState();
* }
* class \_AnimatedSquareState extends State<AnimatedSquare> {
* bool \_isVisible = true;
* @override
* Widget build(BuildContext context) {
* return Scaffold(
* appBar: AppBar(
* title: Text('Opacity Example'),
* ),
* body: Center(
* child: Column(
* mainAxisAlignment: MainAxisAlignment.center,
* children: [
  + AnimatedOpacity(
  + duration: Duration(seconds: 1),
  + opacity: \_isVisible ? 1.0 : 0.0,
  + child: Container(
    - width: 100,
    - height: 100,
    - color: Colors.blue,
  + ),
  + ),
  + SizedBox(height: 20),
  + ElevatedButton(
  + onPressed: () {
    - setState(() {
    - \_isVisible = !\_isVisible;
    - });
  + },
  + child: Text(\_isVisible ? 'Hide' : 'Show'),
  + ),
* ],
* ),
* ),
* );
* }
* }
* class MyApp extends StatelessWidget {
* @override
* Widget build(BuildContext context) {
* return MaterialApp(
* home: AnimatedSquare(),
* );
* }
* }A screen shot of a computer

  Description automatically generated