**CÔNG NGHỆ LẬP TRÌNH ĐA NỀN TẢNG CHO ỨNG DỤNG DI ĐỘNG**

**Lớp:** 08\_ĐH\_CNPM

**Tên:** Đồng Văn Tuấn Đạt

**MSSV:** 0850080015

**Đề: Lý thuyết buổi 7**

**Bài làm**

Nội dung:

**AnimatedWidget**

Là một lớp trừu tượng được sử dụng để xây dựng các widget có hiệu ứng chuyển động (animation). Được sử dụng chủ yếu trong kịch bản chuyển đổi giữa hai trạng thái của widget, như khi thay đổi thuộc tính của widget như màu sắc, kích thước, vị trí, độ trong suốt, và nhiều thuộc tính khác để tạo hiệu ứng hoạt hình cho giao diện người dùng.

Cách hoạt động của AnimatedWidget:

1. AnimatedWidget sẽ tạo và điều khiển một Animation, đại diện cho giá trị thay đổi của thuộc tính cần hiệu ứng.
2. Animation sẽ sinh ra các giá trị trung gian theo khoảng thời gian cụ thể, chẳng hạn như thay đổi từ 0 đến 1 trong thời gian một giây.
3. Khi giá trị của Animation thay đổi, AnimatedWidget sẽ tái xây dựng và vẽ lại giao diện với giá trị mới của thuộc tính, tạo hiệu ứng chuyển động.

Lợi ích của việc sử dụng AnimatedWidget:

* Đơn giản hóa việc thực hiện các hiệu ứng hoạt hình trong Flutter.
* Tự động quản lý quá trình tái xây dựng và vẽ lại giao diện khi giá trị thay đổi, giúp giảm công sức viết mã.

Ví dụ:

class AnimatedLogo extends AnimatedWidget {

// Constructor nhận vào animation và các thuộc tính cần thay đổi

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

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

@override

Widget build(BuildContext context) {

// Lấy giá trị animation hiện tại

final Animation<double> animation = listenable;

return Center(

child: Container(

width: animation.value \* 100, // Thay đổi kích thước theo animation

height: animation.value \* 100,

child: FlutterLogo(),

),

);

}

}

// Sử dụng trong hàm StatefulWidget

class MyAnimatedLogo extends StatefulWidget {

@override

\_MyAnimatedLogoState createState() => \_MyAnimatedLogoState();

}

class \_MyAnimatedLogoState extends State<MyAnimatedLogo> with SingleTickerProviderStateMixin {

AnimationController \_controller;

Animation<double> \_animation;

@override

void initState() {

super.initState();

\_controller = AnimationController(

vsync: this,

duration: Duration(seconds: 1),

);

\_animation = Tween(begin: 0.0, end: 1.0).animate(\_controller);

\_controller.forward();

}

@override

Widget build(BuildContext context) {

return AnimatedLogo(animation: \_animation);

}

@override

void dispose() {

\_controller.dispose();

super.dispose();

}

}

**AnimatedBuilder**

Là một widget được sử dụng để tạo hiệu ứng hoạt hình cho các widget con. Nó giúp tách biệt quá trình xây dựng widget và quá trình thay đổi giá trị của thuộc tính, giúp tối ưu hóa hiệu suất và giảm thiểu việc tái xây dựng không cần thiết của widget.

Cách hoạt động của AnimatedBuilder:

1. Bạn cung cấp một Animation (hoặc Listenable, Tween, ...) cho AnimatedBuilder.
2. AnimatedBuilder sẽ theo dõi sự thay đổi của Animation.
3. Mỗi khi giá trị của Animation thay đổi, hàm xây dựng của AnimatedBuilder được gọi lại và các widget con được xây dựng lại với các giá trị mới của thuộc tính.

Điều này giúp giảm thiểu việc xây dựng lại các widget không liên quan khi chỉ một thuộc tính của widget thay đổi.

Ví dụ:

AnimatedBuilder(

animation: animation, // Animation mà widget này theo dõi

builder: (BuildContext context, Widget child) {

// Hàm xây dựng của AnimatedBuilder

// Trả về các widget con được xây dựng lại với các giá trị mới của thuộc tính

return YourWidget(

property: animation.value, // Giá trị mới của thuộc tính

);

},

)

**AnimationController**

Là một lớp được sử dụng để tạo và quản lý các animation. Nó là một loại Animation<double> đặc biệt, được sử dụng để điều khiển quá trình thay đổi giá trị của một thuộc tính từ một giá trị đầu đến giá trị cuối trong một khoảng thời gian nhất định.

AnimationController cung cấp các phương thức và thuộc tính để điều khiển và theo dõi quá trình animation, bao gồm:

1. forward(): Bắt đầu hoặc tiếp tục animation theo hướng tăng dần (từ giá trị đầu đến giá trị cuối).
2. reverse(): Bắt đầu hoặc tiếp tục animation theo hướng giảm dần (từ giá trị cuối đến giá trị đầu).
3. stop(): Dừng animation ở vị trí hiện tại.
4. reset(): Đặt giá trị animation về lại giá trị đầu.
5. duration: Độ dài thời gian của animation.
6. value: Giá trị hiện tại của animation.

Khi xây dựng một animation, bạn cần tạo một AnimationController và cấu hình các thuộc tính như duration và vsync (ví dụ: SingleTickerProviderStateMixin trong StatefulWidget). Sau đó, bạn có thể sử dụng AnimationController để tạo các animation với các giá trị thay đổi từ giá trị đầu đến giá trị cuối trong khoảng thời gian được xác định.

Ví dụ:

class MyAnimatedWidget extends StatefulWidget {

@override

\_MyAnimatedWidgetState createState() => \_MyAnimatedWidgetState();

}

class \_MyAnimatedWidgetState extends State<MyAnimatedWidget> with SingleTickerProviderStateMixin {

AnimationController \_controller;

Animation<double> \_animation;

@override

void initState() {

super.initState();

\_controller = AnimationController(

vsync: this,

duration: Duration(seconds: 1),

);

\_animation = Tween(begin: 0.0, end: 1.0).animate(\_controller);

\_controller.forward();

}

@override

Widget build(BuildContext context) {

return Opacity(

opacity: \_animation.value, // Sử dụng giá trị animation trong widget

child: YourWidget(),

);

}

@override

void dispose() {

\_controller.dispose();

super.dispose();

}

}

**Hero**

Là một widget đặc biệt dùng để tạo hiệu ứng chuyển đổi mượt mà giữa hai màn hình khi chuyển đổi giữa chúng. Hiệu ứng chuyển đổi thông qua Hero thường được gọi là "Hero animation".

Hero cho phép bạn định nghĩa một widget trong màn hình gốc và đặt nó vào một Hero widget. Sau đó, bạn có thể đặt một widget giống hệt như trên màn hình tiếp theo và đặt nó trong một Hero widget có cùng tag. Khi chuyển đổi giữa hai màn hình, Flutter sẽ tự động tạo hiệu ứng chuyển đổi mượt mà cho các widget có cùng tag. Hiệu ứng này thường là hiệu ứng phóng to/kéo nhỏ và di chuyển widget từ màn hình gốc đến màn hình tiếp theo.

Để sử dụng Hero, bạn cần chỉ định một tag duy nhất cho cả hai Hero widget và đảm bảo rằng chúng có cùng kích thước và vị trí trong cả hai màn hình. Khi chuyển đổi giữa hai màn hình, Flutter sẽ tự động tạo hiệu ứng chuyển đổi cho các widget có cùng tag.

Ví dụ:

class ScreenA extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Screen A'),

),

body: Center(

child: Hero(

tag: 'avatar', // Đặt tag cho Hero widget

child: CircleAvatar(

backgroundImage: AssetImage('assets/avatar.png'),

radius: 50,

),

),

),

);

}

}

class ScreenB extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Screen B'),

),

body: Center(

child: Hero(

tag: 'avatar', // Đặt tag giống với ScreenA

child: CircleAvatar(

backgroundImage: AssetImage('assets/avatar.png'),

radius: 150,

),

),

),

);

}

}

**Opacity**

Trong Flutter, Opacity là một widget dùng để điều chỉnh độ trong suốt của các widget con bên trong nó. Điều này cho phép bạn điều chỉnh mức độ hiển thị của các widget con từ 0.0 (hoàn toàn trong suốt) đến 1.0 (hoàn toàn không trong suốt).

Opacity là một trong những cách đơn giản nhất để thay đổi mức độ hiển thị của một widget hoặc phần của giao diện trong Flutter. Nó thường được sử dụng để tạo hiệu ứng làm mờ, hiển thị nội dung khi người dùng vuốt qua màn hình hoặc để thay đổi tính năng hoặc trạng thái của ứng dụng.

Ví dụ:

Opacity(

opacity: 0.5, // Điều chỉnh mức độ trong suốt từ 0.0 đến 1.0

child: // Widget con được áp dụng độ trong suốt

)