**Lý thuyết buổi 7**

**1. AnimatedWidget**

AnimatedWidget là một widget đặc biệt được sử dụng để giúp tạo các hiệu ứng animation đơn giản và hiệu quả. Các chức năng chính của AnimatedWidget bao gồm:

Tự động cập nhật giao diện người dùng: Khi một thuộc tính trong AnimatedWidget thay đổi, AnimatedWidget sẽ tự động cập nhật lại giao diện người dùng mà không cần phải gọi phương thức setState.

Tối ưu hiệu suất: AnimatedWidget sử dụng các thư viện animation được cung cấp bởi Flutter để tối ưu hiệu suất của các hiệu ứng animation.

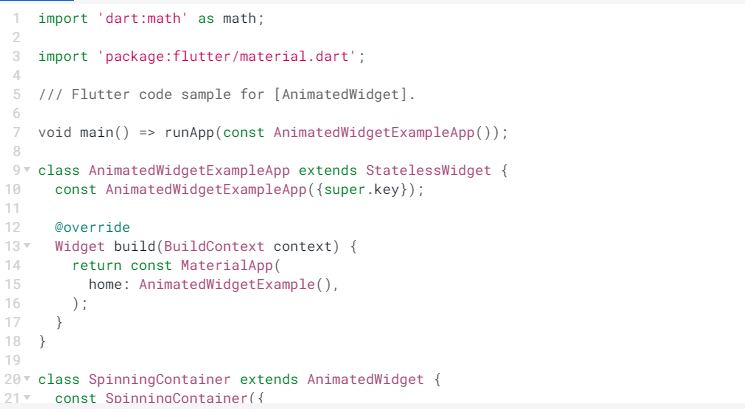
Dễ dàng sử dụng: AnimatedWidget rất dễ sử dụng, không cần phải viết mã phức tạp để tạo các hiệu ứng animation.

Tự động xử lý việc gọi phương thức dispose: Khi AnimatedWidget được loại bỏ khỏi cây widget, nó sẽ tự động gọi phương thức dispose để giải phóng tài nguyên và ngăn chặn thất thoát bộ nhớ.

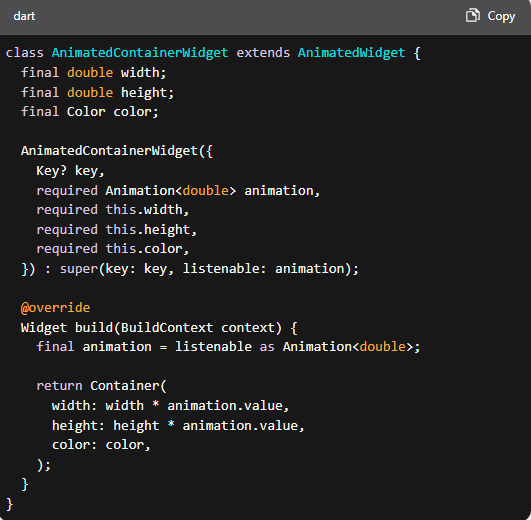
AnimatedWidget là một lớp trừu tượng, nó cần được mở rộng và triển khai phương thức build để tạo ra giao diện người dùng của nó. Bên trong phương thức build, bạn có thể sử dụng các giá trị animation để tạo các hiệu ứng animation cho widget của mình.

Vi du





Ví dụ, ta có thể tạo một AnimatedContainer bằng cách sử dụng AnimatedWidget như sau:



**2. AnimatedBuilde**

AnimatedBuilder là một widget đặc biệt được sử dụng để tạo các hiệu ứng animation tùy chỉnh và linh hoạt hơn so với AnimatedWidget. Các chức năng chính của AnimatedBuilder bao gồm:

Tự động cập nhật giao diện người dùng: Khi một thuộc tính trong AnimatedBuilder thay đổi, AnimatedBuilder sẽ tự động cập nhật lại giao diện người dùng mà không cần phải gọi phương thức setState.

Tùy chỉnh và linh hoạt: AnimatedBuilder cho phép bạn tùy chỉnh các thuộc tính animation và sử dụng chúng theo cách tùy ý để tạo các hiệu ứng animation phong phú và linh hoạt.

Dễ dàng sử dụng: AnimatedBuilder rất dễ sử dụng, không cần phải viết mã phức tạp để tạo các hiệu ứng animation.

Tự động xử lý việc gọi phương thức dispose: Khi AnimatedBuilder được loại bỏ khỏi cây widget, nó sẽ tự động gọi phương thức dispose để giải phóng tài nguyên và ngăn chặn thất thoát bộ nhớ.

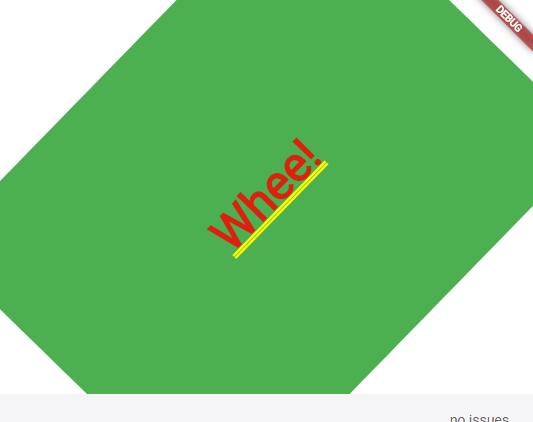
Để sử dụng AnimatedBuilder, bạn cần làm theo các bước sau:

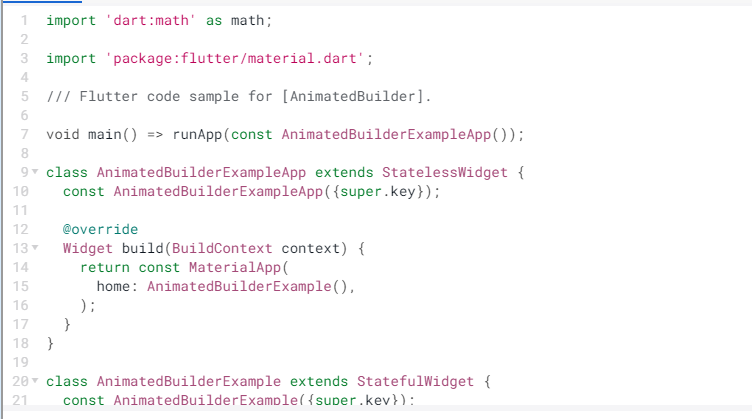
Tạo một thuộc tính animation và một builder function để xây dựng giao diện người dùng.

Sử dụng AnimatedBuilder để bao bọc widget cần tạo hiệu ứng animation.

Trong builder function, sử dụng giá trị animation để tạo các hiệu ứng animation cho widget.

Ví dụ, ta có thể tạo một AnimatedContainer bằng cách sử dụng AnimatedBuilder như sau:







**3. AnimationController class**

Controller cho hiệu ứng hoạt ảnh động

* Lớp này cho phép bạn thực hiện các tác vụ như:
* Tạo hiệu ứng hoạt ảnh tiến - lùi hoặc dừng hoạt ảnh.
* Đặt hoạt ảnh thành một giá trị cụ thể.
* Xác định các giá trị UpperBound và LowerBound của hoạt ảnh.
* Tạo hiệu ứng hoạt hình bằng cách sử dụng mô phỏng vật lý.

Theo mặc định, AnimationController tạo tuyến tính các giá trị nằm trong khoảng từ 0,0 đến 1,0 trong một khoảng thời gian nhất định. Bộ điều khiển chuyển động tạo ra một giá trị mới bất cứ khi nào thiết bị chạy ứng dụng của bạn sẵn sàng hiển thị một khung hình mới (thông thường, tốc độ này là khoảng 60 giá trị mỗi giây).

**Ticker providers**

AnimationController cần một TickerProvider, được cấu hình bằng cách sử dụng đối số vsync trên hàm tạo.

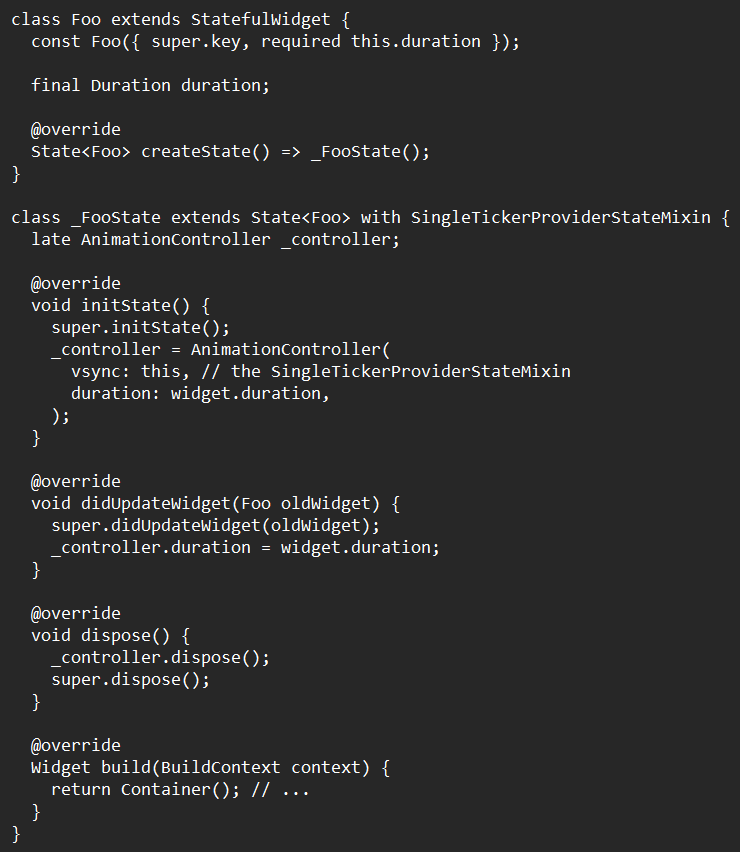
**Life cycle**

AnimationController nên được xử lý khi không còn cần thiết. Điều này làm giảm khả năng rò rỉ. Khi được sử dụng với StatefulWidget, AnimationController thường được tạo trong phương thức State.initState và sau đó được xử lý trong phương thức State.dispose.

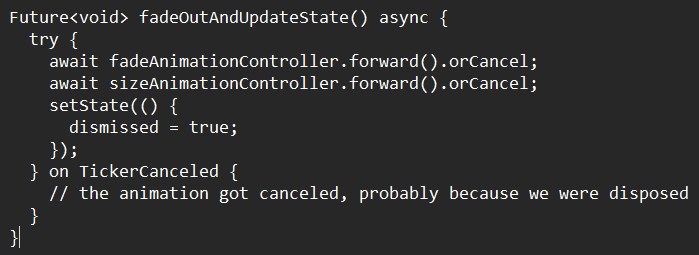
Dùng Futures với AnimationController

Các phương thức bắt đầu hoạt ảnh trả về một đối tượng TickerFuture hoàn thành khi hoạt ảnh hoàn thành thành công và không bao giờ đưa ra lỗi; nếu hoạt ảnh bị hủy, Futures sẽ không bao giờ hoàn thành. Đối tượng này cũng có thuộc tính TickerFuture.orCancel trả về một Futures hoàn thành khi hoạt ảnh hoàn tất thành công và hoàn thành với lỗi khi hoạt ảnh bị hủy bỏ.  
Điều này có thể được sử dụng để viết mã, chẳng hạn như phương thức fadeOutAndUpdateState bên dưới.

Đây là 1 stateful Foo widget. State dùng SingleTickerProviderStateMixin để triển khai TickerProvider cần thiết, tạo controller của nó trong phương thức State.initState và loại bỏ nó trong phương thức State.dispose. Thời lượng của controller được định cấu hình từ một thuộc tính trong Foo widget; Khi điều đó thay đổi, phương thức State.didUpdateWidget được sử dụng để cập nhật controller.



Phương thức sau (đối với lớp con State) điều khiển hai bộ controller hoạt ảnh bằng cách sử dụng cú pháp không đồng bộ của Dart để chờ các đối tượng Future:



Giả định trong đoạn mã trên là các bộ controller hiệu ứng đang được xử lý trong phần ghi đè của lớp con State của phương thức State.dispose. Vì việc xử lý bộ controller sẽ hủy hoạt ảnh (tăng ngoại lệ TickerCanceled), mã ở đây có thể bỏ qua việc xác minh xem State.mount có còn đúng ở mỗi bước hay không. (Một lần nữa, điều này giả định rằng các bộ controller được tạo trong State.initState và được xử lý trong State.dispose, như được mô tả trong phần trước.)

**4.** **CurvedAnimation**

Hiệu ứng này áp dụng đường cong cho hiệu ứng khác.

CurvedAnimation rất hữu ích khi ta muốn áp dụng đường cong phi tuyến tính cho một đối tượng hiệu ứng, đặc biệt nếu ta muốn các đường cong khác nhau khi hiệu ứng đang tiến so với khi nó đi lùi.

Tùy thuộc vào đường cong nhất định, đầu ra của CurvedAnimation có thể có phạm vi rộng hơn đầu vào của nó. Ví dụ, các đường cong đàn hồi như [*Curves.elasticIn*](https://api.flutter.dev/flutter/animation/Curves/elasticIn-constant.html)sẽ vượt quá đáng kể hoặc thiếu mặc định phạm vi từ 0,0 đến 1,0.

Đoạn code sau đây cho thấy cách ta có thể áp dụng đường cong cho hiệu ứng tuyến tính được tạo bởi bộ điều khiển AnimationController.

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

parent: controller,

curve: Curves.ease,

);

Đoạn code thứ hai này cho thấy cách áp dụng một đường cong khác theo hướng tiến so với hướng ngược lại. Điều này không thể được thực hiện bằng cách sử dụng CurveTween (vì Tweens không nhận thức được hướng hoạt hình khi chúng được áp dụng).

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

parent: controller,

curve: Curves.easeIn,

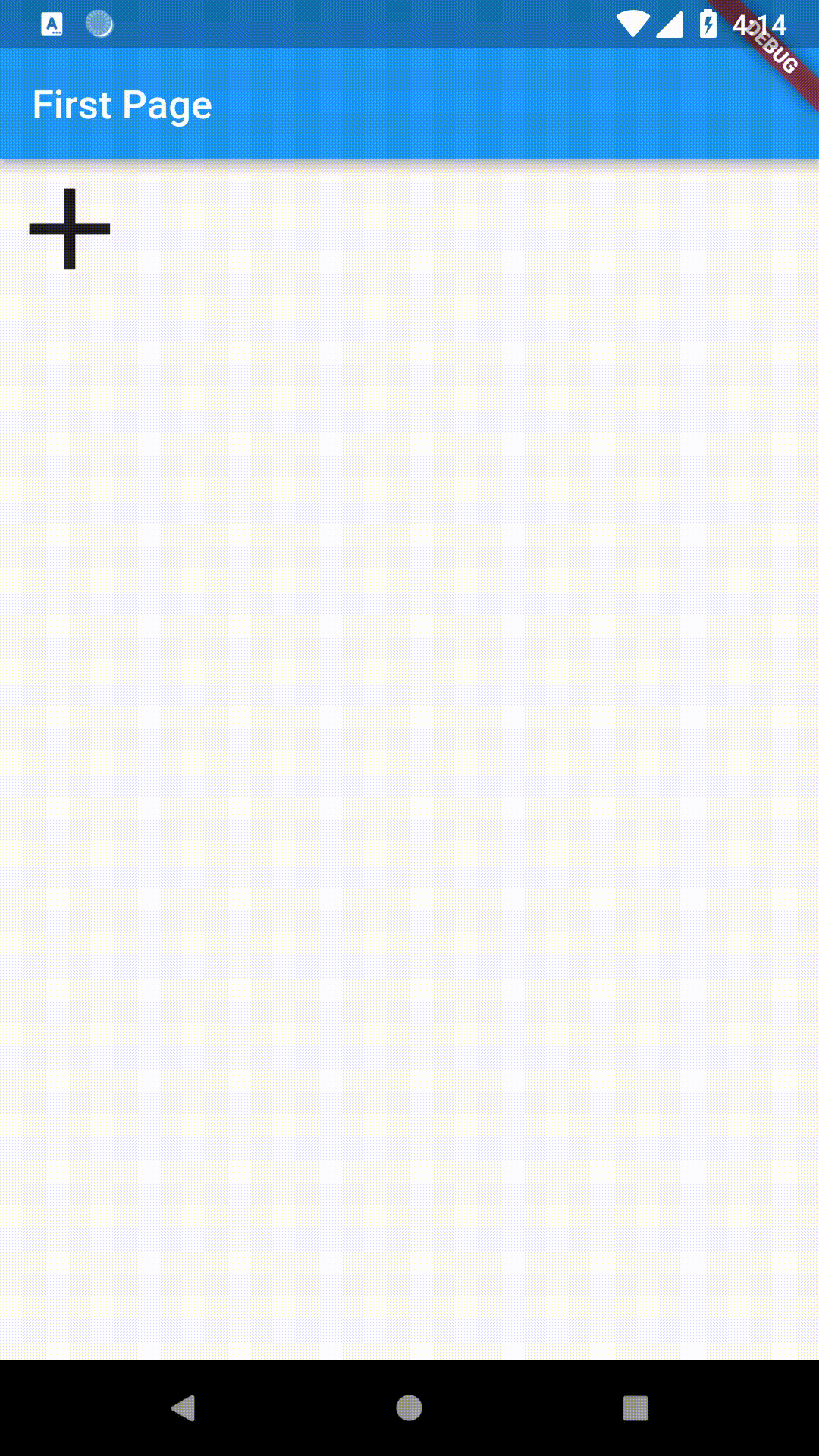
reverseCurve: Curves.easeOut,

);

**5. Hero**

Hero Animation là một widget giúp chúng ta tạo các hiệu ứng tạo ra cảm giác liền mạch giữa các trang hoặc các phần tử khác nhau trong ứng dụng.

Ví dụ:



Một số tính năng của Hero Animations :

Chuyển đổi ảnh giữa các trang: Widget "Hero" cho phép bạn chuyển đổi một hoặc nhiều ảnh giữa các trang.

Chuyển đổi vị trí của các phần tử: Widget "Hero" cũng cho phép bạn chuyển đổi vị trí của các phần tử trong ứng dụng của bạn. Ví dụ, bạn có thể chuyển đổi vị trí của một nút hoặc một khối văn bản từ trang này sang trang khác.

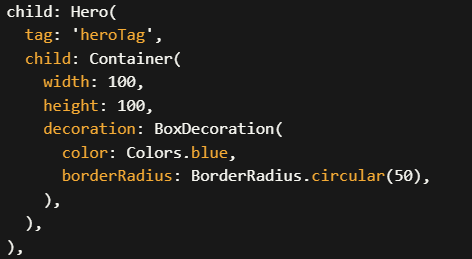
Tùy chỉnh độ trễ: Bạn có thể tùy chỉnh độ trễ giữa các chuyển động khi sử dụng widget "Hero". Điều này giúp bạn kiểm soát tốc độ chuyển động và tạo ra các hiệu ứng chuyển động độc đáo.

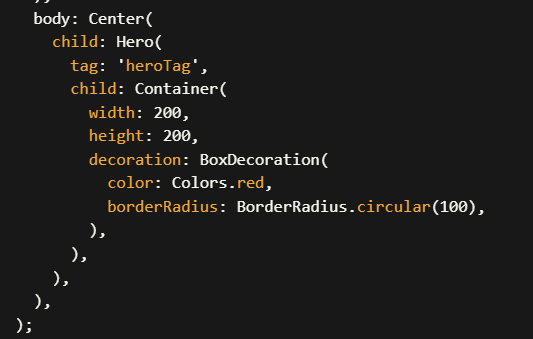
Hỗ trợ nhiều nền tảng: Widget "Hero" là một tính năng của Flutter, một framework lập trình giao diện đa nền tảng. Do đó, bạn có thể sử dụng widget "Hero" để tạo ra các chuyển động mượt mà trên nhiều nền tảng khác nhau, bao gồm iOS và Android.

Tích hợp dễ dàng: Widget "Hero" được tích hợp sẵn trong Flutter, do đó bạn không cần phải cài đặt các thư viện bổ sung để sử dụng nó. Widget "Hero" cũng được hỗ trợ bởi nhiều công cụ phát triển Flutter, giúp bạn dễ dàng tích hợp và tùy chỉnh theo nhu cầu của mình.

Ví dụ :

* Chuyển đổi vị trí phần tử





* Tạo độ trễ

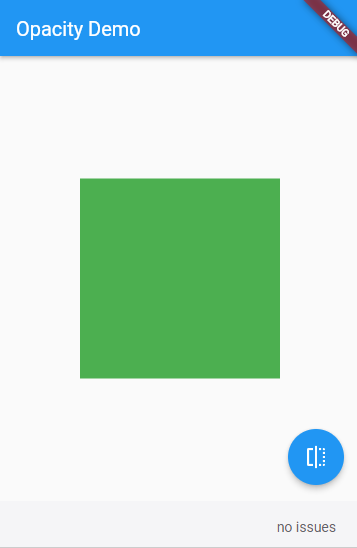


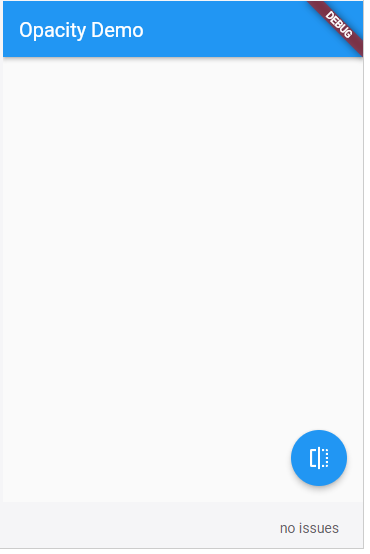
**6.** **Opacity**

Opacity là một thuộc tính của widget trong Flutter, cho phép bạn điều chỉnh độ trong suốt của một widget. Khi bạn đặt giá trị của opacity từ 0 đến 1, widget sẽ được hiển thị trong suốt từ không hiển thị đến hoàn toàn hiển thị.

Chức năng chính của opacity trong Flutter là cho phép bạn thực hiện các hiệu ứng opacity trên các widget. Bằng cách sử dụng opacity, bạn có thể tạo ra các hiệu ứng hiển thị và ẩn dần các widget, tạo ra các hiệu ứng mờ dần, hoặc thay đổi độ trong suốt của các widget theo thời gian

Ví dụ, bạn có thể sử dụng opacity để hiển thị một widget theo cách mờ dần, như sau:



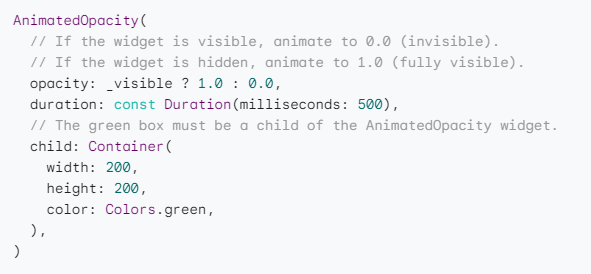


Tiện AnimatedOpacityích yêu cầu ba đối số:

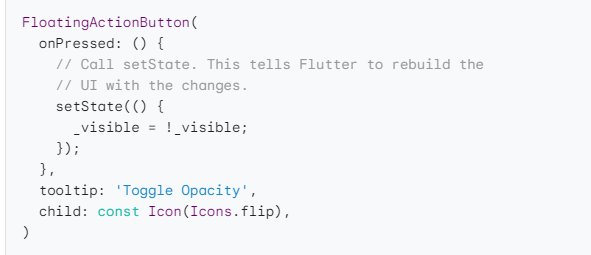
**opacity:** Giá trị từ 0,0 (ẩn) đến 1,0 (hiển thị đầy đủ).

**duration:** Hoạt ảnh sẽ mất bao lâu để hoàn thành.

**child:** Tiện ích để tạo hiệu ứng động. Trong trường hợp này, hộp màu xanh lá cây.



**Button**



Opacity animation trong Flutter có thể được áp dụng cho bất kỳ loại widget nào, từ các widget cơ bản như Container, Text, Image đến các widget phức tạp hơn như ListView, GridView, hoặc các widget tùy chỉnh. Nó cũng có thể được kết hợp với các hiệu ứng khác trong Flutter như animation scale, animation rotation, hay animation position để tạo ra các hiệu ứng phức tạp hơn.