**DỊCH THUẬT “BUILDING LAYOUT**

Nhóm:

Phan Huy Hoàng

Nguyễn Văn Nhật

Nguyễn Minh Đức

# Tạo ứng dụng responsive và adaptive

Một trong những mục tiêu chính của Flutter là tạo 1 framework cho phép bạn phát triển các ứng dụng từ một codebase duy nhất điều đó trông và cảm thấy tuyệt trên bất kì nền tảng nào.

Điều này có nghĩa là ứng dụng của bạn có thể xuất hiện trên các màn hình có nhiều kích thước khác nhau, từ đồng hồ, điện thoại foldable với 2 màn hình, màn hình có độ nét cao.

Hai thuật ngữ mô tả các khái niệm cho kịch bản này là adaptive và responsive. Lý tưởng nhất là bạn muốn ứng dụng của bạn trở thành cả hai nhưng chính xác nó có nghĩa là gì? Các thuật ngữ này tương tự nhau, nhưng chúng không giống nhau.

## Sự khác nhau giữa một adaptive và một ứng dụng responsive

Adaptive và responsive có thể được xem như cách khía cạnh riêng biệt của một ứng dụng: bạn có thể có một ứng dụng apdaptive không đáp ứng và ngược lại. Và, tất nhiên, một ứng dụng có thể là cả hai hoặc không.

### Reponsive

Thông thường, một ứng dụng responsive đã được điều chỉnh bố cục cho phù hợp với kích thước màn hình có sẵn. Thường thì, điều này có nghĩa là (ví dụ), bố trị lại giao diện người dùng nếu người dùng thay đổi kích thước cửa sổ hoặc thay đổi hướng của thiết bị, từ đồng hồ, điện thoại, máy tính bảng đến máy tính xách tay hoặc máy tính để bàn.

### Adaptive

Việc điều chỉnh ứng dụng để chạy trên các loại thiết bị khác nhau, chẳng hạn như thiết bị di động và máy tính để bàn, yêu cầu xử lý đầu vào bằng chuột và bàn phím, cũng như nhập bằng cảm ứng. Điều đó cũng có nghĩa là có những kỳ vọng khác nhau về mật độ hình ảnh của ứng dụng, cách lựa chọn thành phần hoạt động (ví dụ: menu xếp tầng so với trang dưới cùng), sử dụng các tính năng dành riêng cho nền tảng (chẳng hạn như cửa sổ cấp cao nhật), …

## Tạo một dúng dụng responsive Flutter

Flutter cho phép bạn tạo các ứng dụng tự thích ứng với kích thước và hướng màn hình của thiết bị.

Có hai cách tiếp cận cơ bản để tạo ứng dụng Flutter với thiết kế đáp ứng:

### Sử dụng lớp LayoutBuilder

Từ thuộc tính trình tạo của nó, bạn nhận được một đối tượng BoxConstraints. Kiểm tra các thuộc tính của ràng buộc để quyết định những gì sẽ hiển thị. Ví dụ: nếu maxWidth của bạn lớn hơn điểm ngắt chiều rộng, hãy trả về một đối tượng Scaffold với một hàng có danh sách ở bên trái. Nếu nó hẹp hơn, hãy trả về một đối tượng Scaffold với một ngăn chứa danh sách đó. Bạn cũng có thể điều chỉnh màn hình của mình dựa trên chiều cao của thiết bị, tỷ lệ khung hình hoặc một số thuộc tính khác. Khi các ràng buộc thay đổi (ví dụ: người dùng xoay điện thoại hoặc đặt ứng dụng của bạn vào giao diện người dùng xếp lớp trong Nougat), chức năng xây dựng sẽ chạy.

### Sử dụng phương thức MediaQuery.of () trong các hàm xây dựng của bạn

Điều này cung cấp cho bạn kích thước, hướng, v.v. của ứng dụng hiện tại của bạn. Điều này hữu ích hơn nếu bạn muốn đưa ra quyết định dựa trên ngữ cảnh hoàn chỉnh thay vì chỉ dựa trên kích thước của tiện ích cụ thể của bạn. Một lần nữa, nếu bạn sử dụng điều này, thì chức năng xây dựng của bạn sẽ tự động chạy nếu người dùng bằng cách nào đó thay đổi kích thước của ứng dụng.

Các widget và lớp hữu ích khác để tạo giao diện người dùng đáp ứng:

* AspectRatio (tỷ lệ khung hình)
* CustomSingleChildLayout (Tùy chỉnh bố cục con đơn)
* CustomMultiChildLayout (Tùy chỉnh bố cục nhiều con)
* FittedBox (Hộp trang bị)
* FractionallySizeBox (Ô kích thước phân số)
* LayoutBuilder (Trình tạo bố cục)
* MediaQuery (Truy vấn phương tiện)
* MediaQueryData (Dữ liệu truy vấn phương tiện)
* OrientationBuilder (Trình tạo định hướng)

### Các tài nguyên khác

Để biết thêm thông tin, đây là một số tài nguyên, bao gồm cả đóng góp từ cộng đồng Flutter:

* Phát triển cho nhiều kích thước và định hướng màn hình trong Flutter bởi Deven Joshi
* Xây dựng giao diện người dùng đáp ứng trong Flutter của Raouf Rahiche
* Làm cho trang đích Flutter đa nền tảng đáp ứng bởi Priyanka Tyagi
* Làm thế nào để làm cho ứng dụng rung động phản hồi theo kích thước màn hình khác nhau ?, một câu hỏi trên StackOverflow

## Tạo một ứng dụng adaptive Flutter

Tìm hiểu thêm về cách tạo ứng dụng Flutter thích ứng với Xây dựng ứng dụng thích ứng, do nhóm gskinner viết.

Bạn cũng có thể xem các tập sau của The Boring Show:

* <https://youtu.be/n6Awpg1MO6M>
* <https://youtu.be/eikOZzfc0l4>

Để có một ví dụ tuyệt vời về ứng dụng thích ứng, hãy xem Flutter Folio, một ứng dụng sổ lưu niệm được tạo ra với sự cộng tác của gskinner và nhóm Flutter:

* <https://youtu.be/yytBENOnF0w>

Mã nguồn Folio cũng có sẵn trên GitHub. Tìm hiểu thêm trên blog gskinner.

### Tài nguyên khác

Bạn có thể tìm hiểu thêm về cách tạo ứng dụng thích ứng với nền tảng trong các tài nguyên sau:

* Các hành vi và sự thích ứng dành riêng cho nền tảng, một trang trên trang web này.
* Thiết kế giao diện người dùng thực sự thích ứng một bài đăng trên blog và video của Aloïs Deniel, được trình bày tại hội nghị FlutterViking năm nay.
* Ứng dụng thư viện Flutter (repo) đã được viết như một ứng dụng thích ứng.

# Xây dựng ứng dụng adaptive

## Tổng quát

Flutter cung cấp các cơ hội mới để xây dựng các ứng dụng có thể chạy trên thiết bị di động, máy tính để bàn và web từ một cơ sở mã duy nhất. Tuy nhiên, với những cơ hội này, đi kèm với những thách thức mới. Bạn muốn ứng dụng của mình cảm thấy quen thuộc với người dùng, thích ứng với từng nền tảng bằng cách tối đa hóa khả năng sử dụng và đảm bảo trải nghiệm thoải mái và liền mạch. Đó là, bạn cần xây dựng các ứng dụng không chỉ đa nền tảng mà còn thích ứng hoàn toàn với nền tảng.

Có nhiều cân nhắc để phát triển các ứng dụng thích ứng với nền tảng, nhưng chúng thuộc ba loại chính:

* Bố trí
* Đầu vào
* Thành ngữ và quy phạm

Trang này trình bày chi tiết cả ba danh mục bằng cách sử dụng các đoạn mã để minh họa các khái niệm. Nếu bạn muốn xem các khái niệm này kết hợp với nhau như thế nào, hãy xem các ví dụ Flokk và Folio được xây dựng bằng các khái niệm được mô tả tại đây.

## Xây dựng bố cục thích ứng

Một trong những điều đầu tiên bạn phải xem xét khi đưa ứng dụng của mình lên nhiều nền tảng là làm thế nào để ứng dụng phù hợp với các kích thước và hình dạng khác nhau của màn hình mà ứng dụng sẽ chạy.

### Các widget bố cục:

Nếu bạn đang xây dựng ứng dụng hoặc trang web, có thể bạn đã quen với việc tạo giao diện đáp ứng. May mắn cho các nhà phát triển Flutter, có một bộ lớn các widget giúp việc này trở nên dễ dàng hơn.

Một số tiện ích bố cục hữu ích nhất của Flutter bao gồm:

#### Single child

* Align-Aligns một child trong chính nó. Nó nhận một giá trị kép giữa -1 và 1, cho căn chỉnh dọc và ngang.
* AspectRatio – Cố gắng điều chỉnh kích thước trẻ theo một tỷ lệ khung hình cụ thể.
* ConstrainedBox – Áp đặt các hạn chế về kích thước đối với con của nó, cung cấp quyền kiểm soát kích thước tối thiểu hoặc tối đa.
* CustomSingleChildLayout – Sử dụng một hàm ủy nhiệm để định vị một phần tử con. Người được ủy quyền có thể xác định các ràng buộc về bố cục và vị trí cho đứa trẻ.
* Expanded và Flexible – Cho phép một child của một dòng hoặc một cột thu nhỏ hoặc phát triển để lấp đầy bất kỳ không gian có sẵn nào.
* FractionallySizeBox – Định kích thước con của nó thành một phần nhỏ của không gian có sẵn.
* LayoutBuilder – Xây dựng một widget đó có thể tự điều chỉnh lại dựa trên kích thước cha mẹ của nó.
* SingleChildScrolllVies – Thêm cuộn vào một phần tử con. Thường được sử dụng với hầng hoặc cột.

#### Multichild

* Column,Row và Flex Tạo con trong một lần chạy nganh hoặc dọc Cả một cột và hàng đều mở rộng tiện ích linh hoạt.
* CustomMultiChildLayout – Sử dụng hàm ủy quyền để định vị nhiều phần tử con trong giai đoạn bố trí.
* Flow – Tương tự như CustomMultiChilLayout, nhưng hiệu quả hơn vì nó được thực hiện trong giai đoạn sơn thay vì giai đoạn bố trí.
* ListView,, GridView, và CustomScrollView – Cung cấp danh sách children có thể cuộn.
* Stack – Các lớp và định vị nhiều con so với các cạnh của ngăn xếp. Các chức năng tương tuwj như vị trí cố định trong CSS.
* Table – Sử dụng thuật toán bố cục bảng cổ điển cho các bảng con của nó, kết hợp nhiều hàng và cột.
* Wrap – Hiển thị con của nó trong nhiều lần chạy ngang hoặc dọc.

Để xem thêm các tiệc ích có sẵn và mã mẫu, hãy xem tiện ích bố cục.

### Mật độ thị giác

Các thiết bị đầu vào khác nhau cung cấp các mức độ chính xác khác nhau, đòi hỏi các vùng đánh có kích thước khác nhau. Lớp VisualDensity của Flutter giúp bạn dễ dàng điều chỉnh mật độ các chế độ xem của mình trên toàn bộ ứng dụng, chẳng hạn như bằng cách làm cho một nút lớn hơn (và do đó dễ nhấn hơn) trên thiết bị cảm ứng.

Khi bạn thay đổi VisualDensity cho MaterialApp của mình, MaterialComponents hỗ trợ nó sẽ kích hoạt mật độ của chúng để phù hợp. Theo mặc định, cả mật độ ngang và dọc đều được đặt thành 0,0, nhưng bạn có thể đặt mật độ thành bất kỳ giá trị âm hoặc dương nào bạn muốn. Bằng cách chuyển đổi giữa các mật độ khác nhau, bạn có thể dễ dàng điều chỉnh giao diện người dùng của mình:

Chart, treemap chart

Description automatically generated

Để đặt mật độ hình ảnh tùy chỉnh, hãy đưa mật độ vào chủ đề MaterialApp của bạn:

Graphical user interface, text, application

Description automatically generated

Để sử dụng VisualDensity bên trong dạng xem của riêng bạn, bạn có thể tra cứu nó:

Graphical user interface

Description automatically generated

Vùng chứa không chỉ tự động phản ứng với những thay đổi về mật độ mà còn hoạt ảnh khi nó thay đổi. Điều này liên kết các thành phần tùy chỉnh của bạn với nhau, cùng với các thành phần tích hợp, để có hiệu ứng chuyển tiếp mượt mà trên ứng dụng.

Như được hiển thị, VisualDensity không đơn vị, vì vậy nó có thể có ý nghĩa khác nhau đối với các chế độ xem khác nhau. Trong ví dụ này, 1 đơn vị mật độ bằng 6 pixel, nhưng điều này hoàn toàn tùy thuộc vào quan điểm của bạn để quyết định. Thực tế là nó không có đơn vị làm cho nó khá linh hoạt và nó sẽ hoạt động trong hầu hết các ngữ cảnh.

Cần lưu ý rằng Thành phần Vật liệu thường sử dụng giá trị khoảng 4 pixel lôgic cho mỗi đơn vị mật độ hình ảnh. Để biết thêm thông tin về các thành phần được hỗ trợ, hãy xem VisualDensity API. Để biết thêm thông tin về các nguyên tắc mật độ nói chung, hãy xem hướng dẫn Thiết kế Vật liệu.

### Bố cục theo ngữ cảnh

Nếu bạn cần nhiều hơn những thay đổi về mật độ và không thể tìm thấy một tiện ích con đáp ứng được những gì bạn cần, bạn có thể thực hiện một cách tiếp cận thủ tục hơn để điều chỉnh các thông số, tính toán kích thước, hoán đổi các tiện ích con hoặc tái cấu trúc hoàn toàn giao diện người dùng của bạn cho phù hợp với một hệ số dạng cụ thể.

#### Điểm ngắt dựa trên màn hình

Dạng đơn giản nhất của bố cục thủ tục sử dụng các điểm ngắt dựa trên màn hình. Trong Flutter, điều này có thể được thực hiện với API MediaQuery. Không có quy tắc cứng và nhanh cho các kích thước để sử dụng ở đây, nhưng đây là các giá trị chung:

Graphical user interface

Description automatically generated with low confidence

Sử dụng các điểm ngắt, bạn có thể thiết lập một hệ thống đơn giản để xác định loại thiết bị:

Text

Description automatically generated

Thay vào đó, bạn có thể trừu tượng hóa nó nhiều hơn và định nghĩa nó từ nhỏ đến lớn:

Text

Description automatically generated

Các điểm ngắt dựa trên màn hình được sử dụng tốt nhất để đưa ra các quyết định cấp cao nhất trong ứng dụng của bạn. Thay đổi những thứ như mật độ hình ảnh, khoảng đệm hoặc kích thước phông chữ là tốt nhất khi được xác định trên cơ sở toàn cầu.

Bạn cũng có thể sử dụng các điểm ngắt dựa trên màn hình để chỉnh lại cây tiện ích con cấp cao nhất của mình. Ví dụ: bạn có thể chuyển từ bố cục dọc sang ngang khi người dùng không sử dụng thiết bị cầm tay:

Graphical user interface, text

Description automatically generated with medium confidence

Trong một widget khác, bạn có thể hoán đổi hoàn toàn một số phần tử con:

Graphical user interface, application

Description automatically generated

#### Sử dụng LayoutBuillder để linh hoạt hơn

Mặc dù việc kiểm tra tổng kích thước màn hình là rất tốt cho các trang toàn màn hình hoặc đưa ra các quyết định về bố cục chung, nhưng nó thường không lý tưởng cho các lần xem phụ lồng nhau. Thông thường, các lượt xem phụ có các điểm ngắt nội bộ của riêng chúng và chỉ quan tâm đến không gian mà chúng có sẵn để hiển thị.

Cách đơn giản nhất để xử lý điều này trong Flutter là sử dụng lớp LayoutBuilder. LayoutBuilder cho phép một widget đáp ứng các ràng buộc về kích thước cục bộ đến, điều này có thể làm cho widget linh hoạt hơn nếu nó phụ thuộc vào một giá trị toàn cục.

Ví dụ trước có thể được viết lại bằng LayoutBuilder:

Graphical user interface, text, application

Description automatically generated

Giờ đây, tiện ích con này có thể được tạo trong bảng điều khiển bên, hộp thoại hoặc thậm chí là chế độ xem toàn màn hình và điều chỉnh bố cục của nó cho phù hợp với bất kỳ không gian nào được cung cấp.

#### Phân đoạn thiết bị

Đôi khi bạn muốn đưa ra quyết định về bố cục dựa trên nền tảng thực tế mà bạn đang chạy, bất kể kích thước. Ví dụ: khi tạo thanh tiêu đề tùy chỉnh, bạn có thể cần phải kiểm tra loại hệ điều hành và chỉnh sửa bố cục của thanh tiêu đề để thanh tiêu đề không bị các nút cửa sổ gốc che phủ.

Để xác định bạn đang sử dụng kết hợp nền tảng nào, bạn có thể sử dụng API nền tảng cùng với giá trị kIsWeb:

Text

Description automatically generated

Không thể truy cập API nền tảng từ các bản dựng web mà không có ngoại lệ vì gói dart.io không được hỗ trợ trên mục tiêu web. Do đó, mã này kiểm tra web trước tiên và do đoản mạch, Dart sẽ không bao giờ gọi Nền tảng trên các mục tiêu web.

### Nguồn sự thật duy nhất để tạo kiểu

Bạn có thể sẽ thấy dễ dàng hơn để duy trì chế độ xem của mình nếu bạn tạo một nguồn xác thực duy nhất cho các giá trị kiểu như khoảng đệm, khoảng cách, hình dạng góc, kích thước phông chữ, v.v. Điều này có thể được thực hiện dễ dàng với một số lớp trợ giúp:

Graphical user interface, text, application, email

Description automatically generated

Sau đó, những hằng số này có thể được sử dụng thay cho các giá trị số được mã hóa cứng:

Logo, company name

Description automatically generated with medium confidence

Với tất cả các chế độ xem tham chiếu đến các quy tắc hệ thống thiết kế chia sẻ giống nhau, chúng có xu hướng trông đẹp hơn và nhất quán hơn. Thực hiện thay đổi hoặc điều chỉnh giá trị cho một nền tảng cụ thể có thể được thực hiện ở một nơi duy nhất, thay vì sử dụng tìm kiếm và thay thế dễ xảy ra lỗi. Sử dụng các quy tắc được chia sẻ có lợi ích bổ sung là giúp thực thi tính nhất quán về mặt thiết kế.

Một số danh mục hệ thống thiết kế phổ biến có thể được biểu diễn theo cách này là:

* Thời gian hoạt ảnh
* Kích thước và điểm ngắt
* Chèn và đệm
* Bán kính góc
* Bóng đổ
* Nét
* Phông chữ kích thước và kiểu

Giống như hầu hết các quy tắc, có những ngoại lệ: giá trị một lần không được sử dụng ở nơi nào khác trong ứng dụng. Có một chút điểm trong việc lộn xộn các quy tắc tạo kiểu với các giá trị này, nhưng bạn nên xem xét nếu chúng nên được bắt nguồn từ một giá trị hiện có (ví dụ: padding + 1,0). Bạn cũng nên để ý việc sử dụng lại hoặc trùng lặp các giá trị ngữ nghĩa giống nhau. Những giá trị đó có thể sẽ được thêm vào bộ quy tắc tạo kiểu chung.

### Thiết kế theo điểm mạnh của từng yếu tố hình thức

Ngoài kích thước màn hình, bạn cũng nên dành thời gian xem xét điểm mạnh và điểm yếu riêng của các yếu tố hình thức khác nhau. Không phải lúc nào ứng dụng đa nền của bạn cũng lý tưởng để cung cấp chức năng giống hệt nhau ở mọi nơi. Cân nhắc xem liệu có hợp lý khi tập trung vào các khả năng cụ thể hoặc thậm chí xóa một số tính năng nhất định trên một số danh mục thiết bị hay không.

Ví dụ: thiết bị di động có thể di động và có máy ảnh, nhưng chúng không phù hợp cho công việc sáng tạo chi tiết. Với suy nghĩ này, bạn có thể tập trung nhiều hơn vào việc nắm bắt nội dung và gắn thẻ nội dung đó với dữ liệu vị trí cho giao diện người dùng di động, nhưng tập trung vào việc tổ chức hoặc thao tác nội dung đó cho giao diện người dùng máy tính bảng hoặc máy tính để bàn.

Một ví dụ khác là tận dụng rào cản cực kỳ thấp của web để chia sẻ. Nếu bạn đang triển khai một ứng dụng web, hãy quyết định những liên kết sâu nào cần hỗ trợ và thiết kế các tuyến điều hướng của bạn với những điều đó.

Điều quan trọng ở đây là suy nghĩ về những gì mỗi nền tảng hoạt động tốt nhất và xem liệu có những khả năng độc đáo nào mà bạn có thể tận dụng hay không.

### Sự dụng các mục tiêu xây dựng trên máy tính để bàn để kiểm thử nhanh

Một trong những hiệu quả nhất để kiểm tra ứng dụng giao diện là tận dụng các mục tiêu được xây dựng trên máy tính để bàn.

Khi chạy trên máy tính để bàn, bạn có thể dễ dàng thay đổi kích thước cửa sổ trong khi ứng dụng đang chạy để xem trước các kích thước màn hình khác nhau. Điều này, kết hợp với tải lại nóng, có thể đẩy nhanh sự phát triển của một giao diện người dùng đáp ứng.

Chart, treemap chart

Description automatically generated

### Giải quyết liên lạc đầu tiên

Việc xây dựng một giao diện người dùng cảm ứng tuyệt vời thường có thể khó hơn giao diện người dùng máy tính để bàn truyền thống, một phần do thiếu các bộ tăng tốc đầu vào như nhấp chuột phải, con lăn hoặc phím tắt.

Một cách để tiếp cận thách thức này là ban đầu tập trung vào giao diện người dùng cảm ứng tuyệt vời. Bạn vẫn có thể thực hiện hầu hết các thử nghiệm của mình bằng cách sử dụng mục tiêu trên máy tính để bàn để biết tốc độ lặp lại của nó. Tuy nhiên, hãy nhớ thường xuyên chuyển sang thiết bị di động để xác minh rằng mọi thứ đều ổn.

Sau khi bạn đã đánh bóng giao diện cảm ứng, bạn có thể điều chỉnh mật độ hình ảnh cho người dùng chuột, sau đó xếp lớp trên tất cả các đầu vào bổ sung. Tiếp cận các đầu vào khác này như một bộ tăng tốc - các lựa chọn thay thế giúp thực hiện một nhiệm vụ nhanh hơn. Điều quan trọng cần xem xét là người dùng mong đợi điều gì khi sử dụng một thiết bị đầu vào cụ thể và làm việc để phản ánh điều đó trong ứng dụng của bạn.

## Đầu vào

Tất nhiên, chỉ điều chỉnh giao diện ứng dụng của bạn là chưa đủ, bạn còn phải hỗ trợ các đầu vào khác nhau của người dùng. Chuột và bàn phím giới thiệu các kiểu nhập ngoài những kiểu nhập được tìm thấy trên thiết bị cảm ứng — chẳng hạn như bánh xe cuộn, nhấp chuột phải, tương tác khi di chuột, duyệt tab và phím tắt.

### Con lăn

Các tiện ích con cuộn như ScrollView hoặc ListView hỗ trợ con lăn theo mặc định và vì hầu hết mọi tiện ích con tùy chỉnh có thể cuộn đều được tạo bằng cách sử dụng một trong số chúng, nên nó cũng hoạt động với chúng.

Nếu bạn cần triển khai hành vi cuộn tùy chỉnh, bạn có thể sử dụng tiện ích Trình xử lý, cho phép bạn tùy chỉnh cách giao diện người dùng của bạn phản ứng với bánh xe cuộn.

Graphical user interface

Description automatically generated with medium confidence

### Tương tác qua tab và tiêu điểm

Người dùng có bàn phím vật lý mong đợi rằng họ có thể sử dụng phím tab để điều hướng nhanh ứng dụng của bạn và những người dùng có sự khác biệt về động cơ hoặc tầm nhìn thường dựa hoàn toàn vào điều hướng bằng bàn phím.

Có hai cân nhắc đối với tương tác tab: cách tiêu điểm di chuyển từ tiện ích con này sang tiện ích con, được gọi là truyền tải và điểm nổi bật trực quan được hiển thị khi một tiện ích con được lấy tiêu điểm.

Hầu hết các thành phần tích hợp, như nút và trường văn bản, hỗ trợ duyệt và đánh dấu theo mặc định. Nếu bạn có tiện ích con của riêng mình mà bạn muốn đưa vào trình duyệt, bạn có thể sử dụng tiện ích FocusableActionDetector để tạo các điều khiển của riêng bạn. Nó kết hợp chức năng của các widget Hành động, Phím tắt, MouseRegion và Focus để tạo ra một bộ dò xác định các hành động và liên kết chính, đồng thời cung cấp các lệnh gọi lại để xử lý các điểm nổi bật của tiêu điểm và di chuột.

Graphical user interface, text, application, email

Description automatically generated

#### Kiếm soát thứ tự duyệt

Để kiểm soát nhiều hơn thứ tự các tiện ích con được tập trung vào khi người dùng nhấn tab, bạn có thể sử dụng FocusTraversalGroup để xác định các phần của cây sẽ được coi là một nhóm khi lập tab.

Ví dụ: bạn có thể chuyển qua tất cả các trường trong biểu mẫu trước khi chuyển đến nút gửi:

Graphical user interface, text, application

Description automatically generated

Flutter có một số cách tích hợp để duyệt qua các widget và nhóm, mặc định là lớp ReadingOrderTraversalPolicy. Lớp này thường hoạt động tốt, nhưng có thể sửa đổi lớp này bằng cách sử dụng một lớp TraversalPolicy được xác định trước khác hoặc bằng cách tạo một chính sách tùy chỉnh.

### Máy gia tốc bàn phím

Ngoài việc duyệt qua tab, người dùng máy tính để bàn và web đã quen với việc có nhiều phím tắt khác nhau liên kết với các hành động cụ thể. Cho dù đó là phím Delete để xóa nhanh hay Control + N cho tài liệu mới, hãy đảm bảo xem xét các trình tăng tốc khác nhau mà người dùng của bạn mong đợi. Bàn phím là một công cụ nhập liệu mạnh mẽ, vì vậy hãy cố gắng tận dụng nó nhiều nhất có thể. Người dùng của bạn sẽ đánh giá cao nó!

Trình tăng tốc bàn phím có thể được thực hiện theo một số cách trong Flutter tùy thuộc vào mục tiêu của bạn.

Nếu bạn có một tiện ích con như Trường văn bản hoặc Nút đã có nút tiêu điểm, bạn có thể bọc nó trong RawKeyboardListener và lắng nghe các sự kiện bàn phím:

Graphical user interface, text, application

Description automatically generated

Nếu bạn muốn áp dụng một tập hợp các phím tắt cho một phần lớn của cây, bạn có thể sử dụng tiện ích Phím tắt:

Graphical user interface, text, application, email

Description automatically generated

Tiện ích con Phím tắt rất hữu ích vì nó chỉ cho phép các phím tắt được kích hoạt khi cây tiện ích con này hoặc một trong các cây con của nó có tiêu điểm và hiển thị.

Tùy chọn cuối cùng là một bộ lắng nghe toàn cầu. Trình nghe này có thể được sử dụng cho các phím tắt luôn bật, trên toàn ứng dụng hoặc cho các bảng có thể chấp nhận các phím tắt bất cứ khi nào chúng hiển thị (bất kể trạng thái tiêu điểm của chúng là gì). Thêm trình nghe toàn cầu thật dễ dàng với RawKeyboard:

Graphical user interface, text, application, email

Description automatically generated

Để kiểm tra các tổ hợp phím với trình nghe chung, bạn có thể sử dụng bản đồ RawKeyboard.instance.keysPressed. Ví dụ: một phương pháp như sau có thể kiểm tra xem có bất kỳ khóa nào được cung cấp đang bị giữ hay không:

Graphical user interface

Description automatically generated with low confidenceKết hợp hai điều này với nhau, bạn có thể kích hoạt một hành động khi nhấn Shift + N:

Graphical user interface, text, application

Description automatically generated

Một lưu ý thận trọng khi sử dụng trình nghe tĩnh là bạn thường cần phải tắt nó khi người dùng đang nhập vào một trường hoặc khi tiện ích con được liên kết với nó bị ẩn khỏi chế độ xem. Không giống như với Shortcuts hoặc RawKeyboardListener, đây là trách nhiệm của bạn để quản lý. Điều này có thể đặc biệt quan trọng khi bạn đang liên kết trình tăng tốc Xóa / Xóa lùi cho Xóa, nhưng sau đó có các Trường văn bản con mà người dùng có thể đang nhập.

### Chuột, thoát và di chuyển chuột

Trên máy tính để bàn, việc thay đổi con trỏ chuột để biểu thị chức năng của nội dung con chuột đang di chuột qua là điều thường thấy. Ví dụ: bạn thường thấy con trỏ tay khi di chuột qua một nút hoặc con trỏ tôi khi bạn di chuột qua văn bản.

Bộ Thành phần Vật liệu có hỗ trợ tích hợp cho nút tiêu chuẩn và con trỏ văn bản của bạn. Để thay đổi con trỏ từ trong các widget của riêng bạn, hãy sử dụng MouseRegion:

Graphical user interface, text

Description automatically generated

MouseRegion cũng hữu ích để tạo các hiệu ứng di chuột qua và di chuột qua tùy chỉnh:

Graphical user interface, text, application

Description automatically generated

## Thành ngữ và quy phạm

Khu vực cuối cùng cần xem xét đối với các ứng dụng thích ứng là các tiêu chuẩn nền tảng. Mỗi nền tảng có các thành ngữ và chuẩn mực riêng; các tiêu chuẩn danh nghĩa hoặc thực tế này thông báo cho kỳ vọng của người dùng về cách một ứng dụng sẽ hoạt động. Một phần cảm ơn web, người dùng đã quen với trải nghiệm tùy chỉnh hơn, nhưng việc phản ánh các tiêu chuẩn nền tảng này vẫn có thể mang lại những lợi ích đáng kể:

* Giảm tải nhận thức — Bằng cách phù hợp với mô hình tinh thần hiện có của người dùng, việc hoàn thành nhiệm vụ trở nên trực quan, đòi hỏi ít suy nghĩ hơn, tăng năng suất và giảm sự thất vọng.
* Xây dựng lòng tin — Người dùng có thể trở nên cảnh giác hoặc nghi ngờ khi các ứng dụng không tuân theo kỳ vọng của họ. Ngược lại, giao diện người dùng cảm thấy quen thuộc có thể xây dựng lòng tin của người dùng và có thể giúp cải thiện nhận thức về chất lượng. Điều này thường có thêm lợi ích là xếp hạng cửa hàng ứng dụng tốt hơn — điều mà tất cả chúng ta đều có thể đánh giá cao!

### Xem xét hành vi mong đợi trên mỗi nền tảng

Bước đầu tiên là dành một chút thời gian để xem xét giao diện, bản trình bày hoặc hành vi được mong đợi trên nền tảng này. Cố gắng quên đi bất kỳ giới hạn nào của việc triển khai hiện tại của bạn và chỉ cần hình dung trải nghiệm người dùng lý tưởng. Làm việc ngược lại từ đó.

Một cách khác để nghĩ về điều này là hỏi, "Người dùng nền tảng này mong đợi đạt được mục tiêu này như thế nào?" Sau đó, hãy thử hình dung cách hoạt động trong ứng dụng của bạn mà không có bất kỳ thỏa hiệp nào.

Điều này có thể khó khăn nếu bạn không phải là người dùng thường xuyên của nền tảng này. Bạn có thể không biết về các thành ngữ cụ thể và có thể dễ dàng bỏ lỡ chúng hoàn toàn. Ví dụ: một người dùng Android lâu đời có thể sẽ không biết về các quy ước nền tảng trên iOS và điều này cũng đúng với macOS, Linux và Windows. Những khác biệt này có thể là nhỏ đối với bạn, nhưng rõ ràng là rất rõ ràng đối với một người dùng có kinh nghiệm.

#### Tìm một người ủng hộ nền tảng

Nếu có thể, hãy chỉ định ai đó làm người ủng hộ cho mỗi nền tảng. Lý tưởng nhất là người ủng hộ của bạn sử dụng nền tảng này làm thiết bị chính của họ và có thể đưa ra quan điểm của một người dùng rất kiên định. Để giảm số lượng người, hãy kết hợp các vai trò. Có một người ủng hộ cho Windows và Android, một cho Linux và web, một cho Mac và IOS.

Mục tiêu là có phản hồi liên tục, đầy đủ thông tin để ứng dụng cảm thấy tuyệt vời trên mỗi nền tảng. Những người ủng hộ nên được khuyến khích khá kén chọn, gọi bất cứ thứ gì họ cảm thấy khác với các ứng dụng điển hình trên thiết bị của họ. Một ví dụ đơn giản là cách nút mặc định trong hộp thoại thường ở bên trái trên Mac và Linux, nhưng lại ở bên phải trên Windows. Các chi tiết như vậy rất dễ bị bỏ sót nếu bạn không sử dụng nền tảng thường xuyên.

Quan trọng: Những người ủng hộ không cần phải là nhà phát triển hoặc thậm chí là thành viên toàn thời gian trong nhóm. Họ có thể là nhà thiết kế, các bên liên quan hoặc người kiểm tra bên ngoài được cung cấp các bản dựng thường xuyên.

#### Duy trì sự độc đáo

Tuân theo các hành vi mong đợi không có nghĩa là ứng dụng của bạn cần sử dụng các thành phần hoặc kiểu mặc định. Nhiều ứng dụng đa nền phổ biến nhất có giao diện người dùng rất riêng biệt và phù hợp bao gồm các nút tùy chỉnh, menu ngữ cảnh và thanh tiêu đề.

Bạn càng có thể hợp nhất kiểu dáng và hành vi trên các nền tảng, thì việc phát triển và thử nghiệm sẽ dễ dàng hơn. Bí quyết là cân bằng giữa việc tạo ra trải nghiệm độc đáo với bản sắc mạnh mẽ, đồng thời tôn trọng các tiêu chuẩn của từng nền tảng.

### Các thành ngữ và tiểu chuẩn cần xem xét

Hãy xem nhanh một vài tiêu chuẩn và thành ngữ cụ thể mà bạn có thể muốn xem xét và cách bạn có thể tiếp cận chúng trong Flutter.

#### Giao diện và hành vi của thanh cuộn

Người dùng máy tính để bàn và thiết bị di động mong đợi thanh cuộn, nhưng họ mong đợi chúng hoạt động khác nhau trên các nền tảng khác nhau. Người dùng di động mong đợi các thanh cuộn nhỏ hơn chỉ xuất hiện khi cuộn, trong khi người dùng máy tính để bàn thường mong đợi các thanh cuộn lớn hơn, có mặt khắp nơi mà họ có thể nhấp hoặc kéo.

Flutter đi kèm với tiện ích Thanh cuộn tích hợp sẵn đã hỗ trợ màu sắc và kích thước thích ứng theo nền tảng hiện tại. Một tinh chỉnh bạn có thể muốn thực hiện là chuyển đổi alwaysShown khi ở trên nền tảng máy tính để bàn:

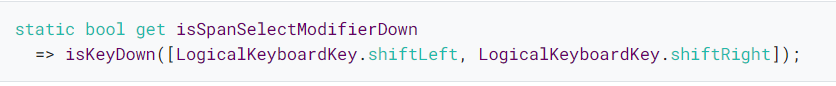
Graphical user interface, text, application

Description automatically generated

Sự chú ý tinh tế đến từng chi tiết này có thể làm cho ứng dụng của bạn cảm thấy thoải mái hơn trên một nền tảng nhất định.

#### Nhiều lựa chọn

Xử lý nhiều lựa chọn trong một danh sách là một lĩnh vực khác với sự khác biệt nhỏ giữa các nền tảng:



Để thực hiện kiểm tra nhận biết nền tảng đối với điều khiển hoặc lệnh, bạn có thể viết một cái gì đó như sau:

Graphical user interface, text, application

Description automatically generated

Cân nhắc cuối cùng cho người dùng bàn phím là hành động Chọn tất cả. Nếu bạn có một danh sách lớn các mục gồm các mục có thể chọn, nhiều người dùng bàn phím của bạn sẽ mong đợi rằng họ có thể sử dụng Control + A để chọn tất cả các mục.

Thiết bị cảm ứng:

Trên các thiết bị cảm ứng, đa lựa chọn thường được đơn giản hóa, với hành vi dự kiến tương tự như việc tắt isMultiSelectModifier trên máy tính để bàn. Bạn có thể chọn hoặc bỏ chọn các mục bằng một lần nhấn và thường sẽ có một nút để Chọn tất cả hoặc Xóa lựa chọn hiện tại.

Cách bạn xử lý nhiều lựa chọn trên các thiết bị khác nhau tùy thuộc vào các trường hợp sử dụng cụ thể của bạn, nhưng điều quan trọng là đảm bảo rằng bạn đang cung cấp cho mỗi nền tảng mô hình tương tác tốt nhất có thể.

#### Văn bản có thể chọn

Kỳ vọng phổ biến trên web (và ở mức độ thấp hơn trên máy tính để bàn) là văn bản dễ thấy nhất có thể được chọn bằng con trỏ chuột. Khi không thể chọn văn bản, người dùng trên web có xu hướng phản ứng ngược.

May mắn thay, điều này dễ dàng hỗ trợ với tiện ích con SelectableText:



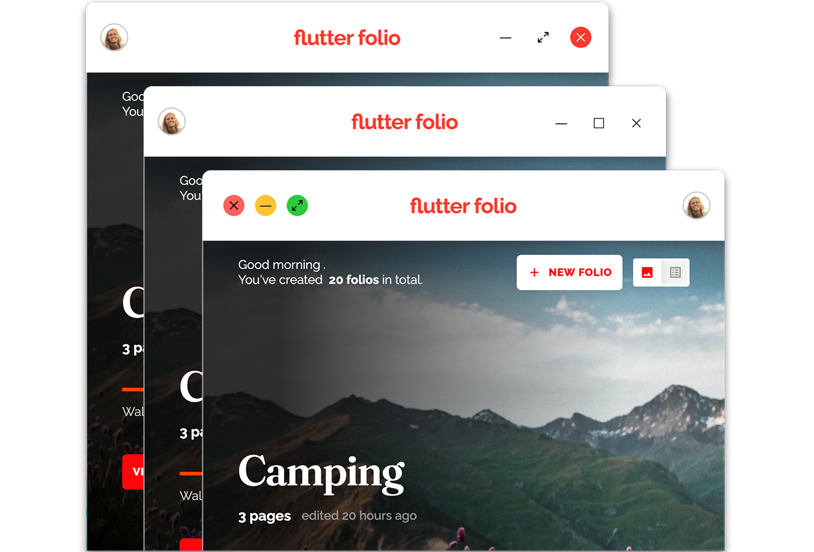
Để hỗ trợ văn bản đa dạng thức, hãy sử dụng TextSpan:

Graphical user interface, application

Description automatically generated

#### Thanh tiêu đề

Trên các ứng dụng máy tính để bàn hiện đại, người ta thường tùy chỉnh thanh tiêu đề của cửa sổ ứng dụng, thêm biểu trưng để xây dựng thương hiệu mạnh hơn hoặc các điều khiển theo ngữ cảnh để giúp tiết kiệm không gian dọc trong giao diện người dùng chính của bạn.



Điều này không được hỗ trợ trực tiếp trong Flutter, nhưng bạn có thể sử dụng gói bits\_dojo để tắt các thanh tiêu đề gốc và thay thế chúng bằng thanh tiêu đề của riêng bạn.

Gói này cho phép bạn thêm bất kỳ tiện ích nào bạn muốn vào TitleBar vì nó sử dụng các widget Flutter thuần túy. Điều này giúp dễ dàng điều chỉnh thanh tiêu đề khi bạn điều hướng đến các phần khác nhau của ứng dụng.

#### Menu ngữ cảnh và chú giải công cụ

Trên máy tính để bàn, có một số tương tác biểu hiện dưới dạng tiện ích con được hiển thị trong lớp phủ, nhưng có sự khác biệt về cách chúng được kích hoạt, loại bỏ và định vị

* Context menu: Thường được kích hoạt bằng nhấp chuột phải, menu ngữ cảnh được đặt gần chuột và bị loại bỏ bằng cách nhấp vào bất kỳ đâu, chọn một tùy chọn từ menu hoặc nhấp vào bên ngoài nó.
* Tooltip: Thường được kích hoạt bằng cách di chuột trong 200-400ms trên một phần tử tương tác, chú giải công cụ thường được gắn vào một tiện ích con (trái ngược với vị trí chuột) và bị loại bỏ khi con trỏ chuột rời khỏi tiện ích đó.
* Popup panel (also known as flyout): tương tự như chú giải công cụ, bảng điều khiển bật lên thường được gắn vào một tiện ích con. Sự khác biệt chính là các bảng điều khiển thường được hiển thị nhất trong một sự kiện nhấn và chúng thường không tự ẩn khi con trỏ rời đi. Thay vào đó, các bảng thường bị loại bỏ bằng cách nhấp vào bên ngoài bảng hoặc bằng cách nhấn nút Đóng hoặc Gửi.

Để hiển thị các chú giải công cụ cơ bản trong Flutter, hãy sử dụng tiện ích Chú giải công cụ được tích hợp sẵn:

Text

Description automatically generated with low confidence

Flutter cũng cung cấp các menu ngữ cảnh tích hợp sẵn khi chỉnh sửa hoặc chọn văn bản.

Để hiển thị các chú giải công cụ nâng cao hơn, bảng bật lên hoặc tạo menu ngữ cảnh tùy chỉnh, bạn sử dụng một trong các gói có sẵn hoặc tự tạo bằng cách sử dụng Ngăn xếp hoặc Lớp phủ.

Một số gói có sẵn bao gồm:

* context\_menus
* anchor\_popups
* Flagship\_portal
* super\_tooltip
* custom\_pop\_up\_menu

Mặc dù các điều khiển này có thể có giá trị đối với người dùng cảm ứng như bộ tăng tốc, nhưng chúng rất cần thiết cho người dùng chuột. Những người dùng này mong đợi nhấp chuột phải vào mọi thứ, chỉnh sửa nội dung tại chỗ và di chuột để biết thêm thông tin. Không đáp ứng được những kỳ vọng đó có thể khiến người dùng thất vọng hoặc ít nhất, cảm thấy có điều gì đó không ổn.

#### Thứ tự nút ngang

Trên Windows, khi trình bày một hàng nút, nút xác nhận được đặt ở đầu hàng (phía bên trái). Trên tất cả các nền tảng khác thì ngược lại. Nút xác nhận được đặt ở cuối hàng (phía bên phải).

Điều này có thể được xử lý dễ dàng trong Flutter bằng cách sử dụng thuộc tính TextDirection trên Row:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

#### Thanh Menu

Một mẫu phổ biến khác trên các ứng dụng dành cho máy tính để bàn là thanh menu. Trên Windows và Linux, menu này nằm như một phần của thanh tiêu đề Chrome, trong khi trên macOS, menu này nằm dọc theo đầu màn hình chính.

Hiện tại, bạn có thể chỉ định các mục nhập trên thanh menu tùy chỉnh bằng cách sử dụng plugin nguyên mẫu, nhưng dự kiến rằng chức năng này cuối cùng sẽ được tích hợp vào SDK chính.

Điều đáng nói là trên Windows và Linux, bạn không thể kết hợp thanh tiêu đề tùy chỉnh với thanh menu. Khi bạn tạo thanh tiêu đề tùy chỉnh, bạn đang thay thế hoàn toàn thanh tiêu đề gốc, điều đó có nghĩa là bạn cũng mất thanh menu gốc tích hợp.

Nếu bạn cần cả thanh tiêu đề tùy chỉnh và thanh menu, bạn có thể đạt được điều đó bằng cách triển khai nó trong Flutter, tương tự như menu ngữ cảnh tùy chỉnh.

#### Kéo và thả

Một trong những tương tác cốt lõi cho cả đầu vào dựa trên cảm ứng và con trỏ là kéo và thả. Mặc dù sự tương tác này được mong đợi cho cả hai loại đầu vào, nhưng có những khác biệt quan trọng cần nghĩ đến khi nói đến danh sách cuộn các mục có thể kéo.

Nói chung, người dùng chạm mong đợi nhìn thấy các chốt kéo để phân biệt các khu vực có thể kéo với các khu vực có thể cuộn hoặc cách khác, để bắt đầu kéo bằng cách sử dụng cử chỉ nhấn và giữ. Điều này là do cuộn và kéo đều dùng chung một ngón tay để nhập.

Người dùng chuột có nhiều tùy chọn đầu vào hơn. Họ có thể sử dụng bánh xe hoặc thanh cuộn để cuộn, điều này thường loại bỏ sự cần thiết của các tay cầm kéo chuyên dụng. Nếu bạn nhìn vào macOS Finder hoặc Windows Explorer, bạn sẽ thấy rằng chúng hoạt động theo cách này: bạn chỉ cần chọn một mục và bắt đầu kéo.

Trong Flutter, bạn có thể thực hiện kéo và thả theo nhiều cách. Thảo luận về các triển khai cụ thể nằm ngoài phạm vi của bài viết này, nhưng một số tùy chọn cấp cao là:

Sử dụng trực tiếp các API Draggable và DragTarget để có giao diện tùy chỉnh.

Kết nối vào các sự kiện cử chỉ onPan và tự di chuyển một đối tượng trong Ngăn xếp chính.

Sử dụng một trong các gói danh sách được tạo sẵn trên pub.dev.

### Tự đào tạo về các nguyên tắc cơ bản về khả năng sử dụng

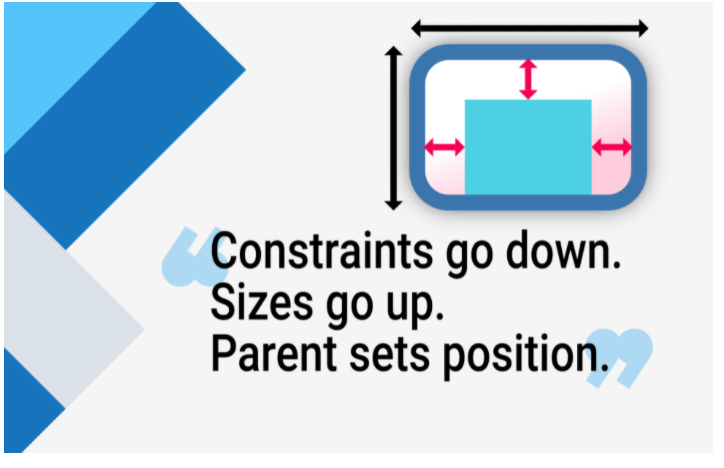
Tất nhiên, trang này không tạo thành một danh sách đầy đủ những điều bạn có thể xem xét. Bạn càng hỗ trợ nhiều hệ điều hành, hệ số dạng và thiết bị đầu vào, thì càng khó xác định mọi hoán vị trong thiết kế.

Dành thời gian tìm hiểu các nguyên tắc cơ bản về khả năng sử dụng với tư cách là nhà phát triển trao quyền cho bạn để đưa ra quyết định tốt hơn, giảm các lần lặp đi lặp lại với thiết kế trong quá trình sản xuất và cải thiện năng suất với kết quả tốt hơn.

Dưới đây là một số tài nguyên để giúp bạn bắt đầu:

* Nguyên tắc quan trọng về bố cục giao diện người dùng đáp ứng.
* Material design cho màn hình lớn.
* Xây dựng ứng dụng chất lượng cao (Android).
* Những điều nên làm và không nên làm trong thiết kế giao diện người dùng (Apple).
* Nguyên tắc về giao diện con người (Apple).
* Kỹ thuật thiết kế đáp ứng (Microsoft).
* Kích thước máy và điểm ngắt (Microsoft).

# Hiểu các hạn chế



Khi ai đó học về Flutter nhỏi bạn tại sao 1 vài widget với width:100 không phải là dài 100 pixels, câu trả lời mặc định là yêu cầu họ đặt tiện ích đó bên trong center, đúng không?

**Đừng làm như vậy**

Nếu bạn làm, họ sẽ trở lại nhiều lần và hỏi tại sao vài FittedBox không hoạt động, tại sao Cột đó bị tràn hoặc Intri InternalWidth được cho là đang làm gì.

Thay vào đó, trước tiên hãy nói với họ rằng bố cục Flutter rất khác với bố cục HTML (có thể là nơi xuất phát của chúng) và sau đó yêu cầu họ ghi nhớ quy tắc sau:

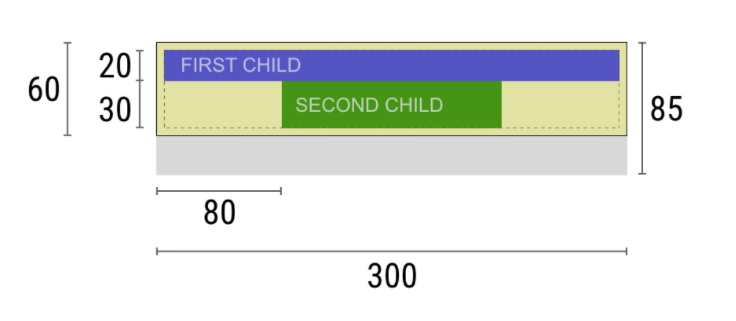
**Ràng buộc giảm, kích thước tăng so với với vị trí gốc**

Bố cục Flutter có thể không thật sự hiểu nếu không biết quy tắc này vậy nên các lập trình viên Flutter nên học nó sớm.

Chi tiết hơn:

* Một widget có các ràng buộc riêng từ cha của nó. Ràng buộc chỉ là một tập hợp 4 đường gấp đôi: chiều rộng tối thiểu và tối đa, và chiều cao tối thiểu và tối đa.
* Sau đó, widget đi qua danh sách con của chính nó. Từng người một, widget cho trẻ biết những ràng buộc của chúng là gì (có thể khác nhau đối với từng trẻ), sau đó hỏi từng trẻ xem nó muốn có kích thước như thế nào.
* Sau đó, widget định vị các con của nó (theo chiều ngang của trục x và theo chiều dọc của trục y), từng cái một.
* Và, cuối cùng, widget nói với cha mẹ của nó về kích thước của chính nó (tất nhiên là trong các ràng buộc ban đầu).

Ví dụ: nếu một tiện ích con đã soạn có chứa một cột có một số khoảng đệm và muốn bố trí hai cột con của nó như sau:



Cuộc thương lượng diễn ra như sau:

**Widget**: "Này cha mẹ, những ràng buộc của tôi là gì?

**Parent**: “Bạn phải rộng từ 80 đến 300 pixel và cao từ 30 đến 85”.

**Widget**: “Hừm, vì tôi muốn có 5 pixel đệm, thì con tôi có thể có tối đa 290 pixel chiều rộng và 75 pixel chiều cao.”

**Widget**: “Này first child, Bạn phải rộng từ 0 đến 290 pixel và cao từ 0 đến 75.”

**First child**: "OK, sau đó tôi muốn rộng 290 pixel và cao 20 pixel."

**Widget**: “Hmmm, vì tôi muốn đặt đứa con thứ hai của mình thấp hơn đứa con thứ nhất, điều này chỉ để lại chiều cao 55 pixel cho đứa con thứ hai của tôi”.

**Widget**: "Này second child, Bạn phải rộng từ 0 đến 290 và cao từ 0 đến 55."

**Second child**: "OK, tôi muốn rộng 140 pixel và cao 30 pixel."

**Widget**: “Rất tốt. Con đầu tiên của tôi có vị trí x: 5 và y: 5, và con thứ hai của tôi có x: 80 và y: 25. ”

**Widget**: “Này parent, tôi đã quyết định rằng kích thước của tôi sẽ là rộng 300 pixel và cao 60 pixel.”

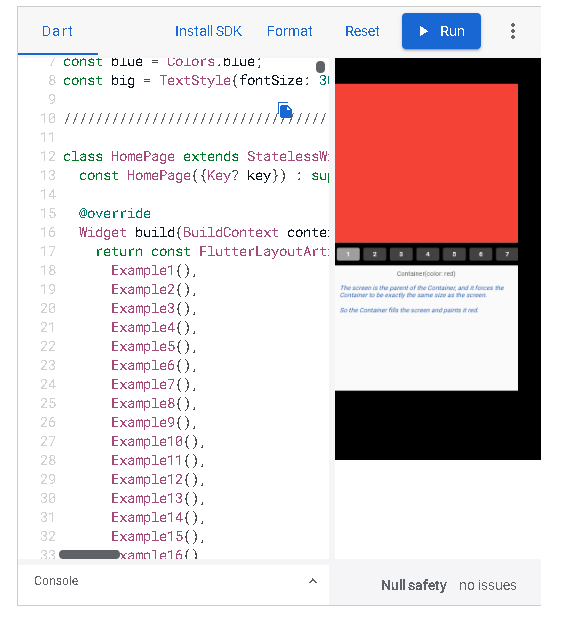
## Giới hạn

Do quy tắc bố cục được đề cập ở trên, công cụ bố cục của Flutter có một số hạn chế quan trọng:

* Một widget chỉ có thể quyết định kích thước của chính nó trong những ràng buộc do cha mẹ của nó đưa ra. Điều này có nghĩa là một tiện ích con thường **không thể có bất kỳ kích thước nào mà nó muốn**.
* Một tiện ích không thể biết và **không quyết định vị trí của chính nó trên màn hình**, vì chính phụ tùng của tiện ích sẽ quyết định vị trí của tiện ích.
* Vì kích thước và vị trí của phụ tùng, đến lượt nó, cũng phụ thuộc vào chính phụ huynh của nó, nên không thể xác định chính xác kích thước và vị trí của bất kỳ tiện ích nào mà không xem xét toàn bộ cây.
* Nếu trẻ muốn có kích thước khác với kích thước của cha mẹ và cha mẹ không có đủ thông tin để căn chỉnh kích thước đó thì kích thước của trẻ có thể bị bỏ qua. **Hãy cụ thể khi xác định sự liên kết**.

## Ví dụ

Để có trải nghiệm tương tác, hãy sử dụng DartPad sau. Sử dụng thanh cuộn ngang được đánh số để chuyển đổi giữa 29 ví dụ khác nhau.



Nếu muốn, bạn có thể lấy mã từ [repo GitHub này](https://github.com/marcglasberg/flutter_layout_article).

Các ví dụ được giải thích trong các phần sau.



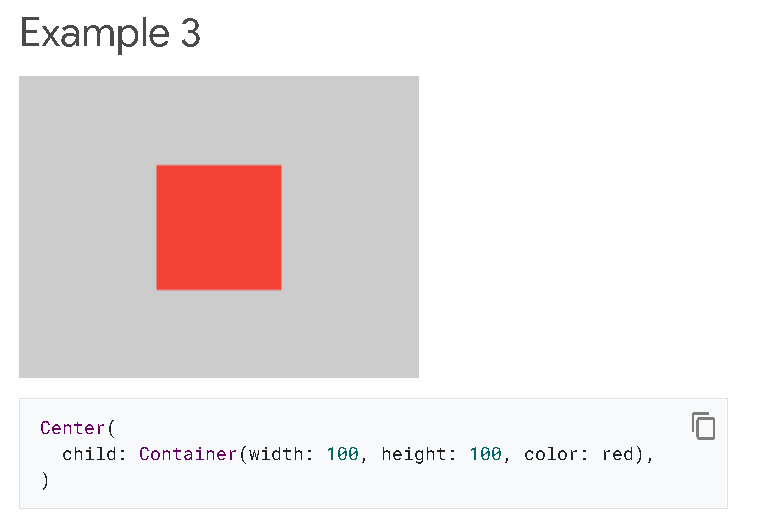
Màn hình là parent của Container và nó buộc Container phải có cùng kích thước với màn hình.

Vì vậy, Container lấp đầy màn hình và sơn nó màu đỏ.



Container màu đỏ muốn có kích thước 100 × 100, nhưng không thể, bởi vì màn hình buộc nó phải có cùng kích thước chính xác với màn hình.

Vì vậy, Container lấp đầy màn hình.



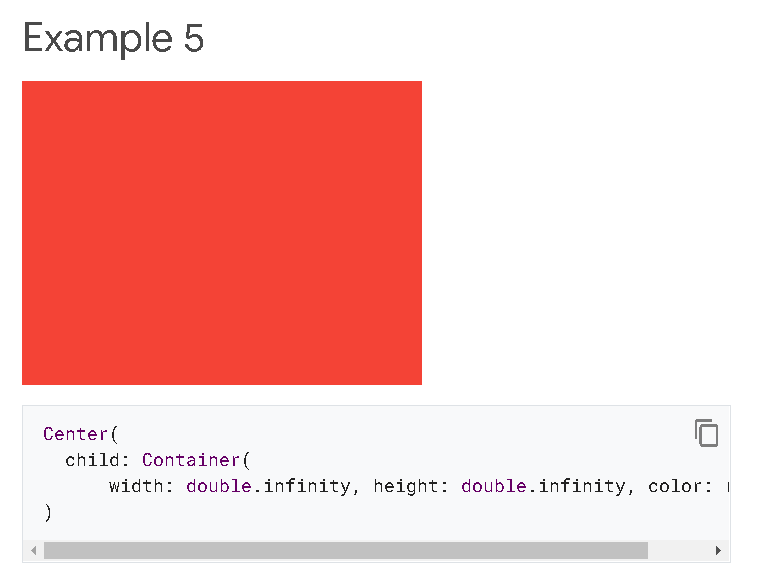
Màn hình buộc Center có kích thước chính xác bằng màn hình, vậy nên Center lấp đầy màn hình.

The Center cho Container biết rằng nó có thể có kích thước bất kỳ mà nó muốn, nhưng không được lớn hơn màn hình. Bây giờ Container thực sự có thể là 100 × 100.



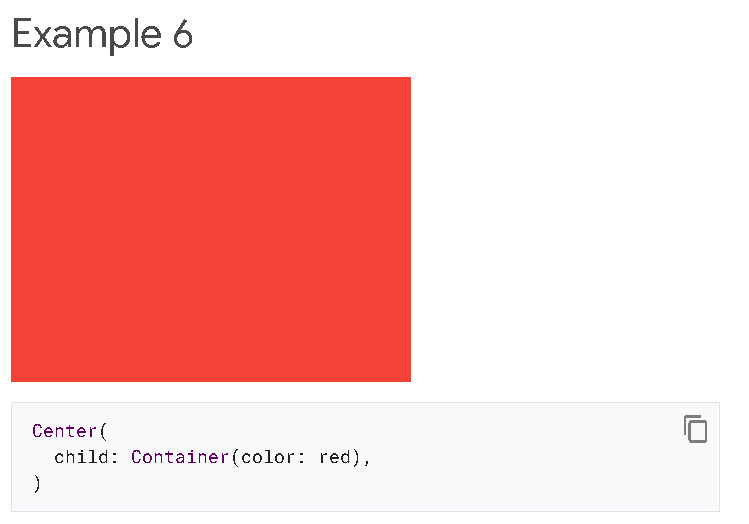
Điều này khác với ví dụ trước ở chỗ nó sử dụng Align thay vì Center.

Align cũng cho Container biết rằng nó có thể có bất kỳ kích thước nào nó muốn, nhưng nếu có không gian trống, nó sẽ không căn giữa Container. Thay vào đó, nó căn chỉnh vùng chứa ở phía dưới bên phải của không gian có sẵn.



Màn hình buộc Center có kích thước chính xác bằng màn hình, vậy nên Center lấp đầy màn hình.

Center nói cho Container biết rằng nó có thể có kích thước bất kỳ mà nó muốn, nhưng không được lớn hơn màn hình.Container muốn có kích thước vô hạn, nhưng vì nó không thể lớn hơn màn hình nên nó chỉ lấp đầy màn hình.



Màn hình buộc Center có kích thước chính xác bằng màn hình, vậy nên Center lấp đầy màn hình.

Center cho Container biết rằng nó có thể có kích thước bất kỳ mà nó muốn, nhưng không được lớn hơn màn hình. Vì Vùng chứa không có con và không có kích thước cố định, nó quyết định nó muốn càng lớn càng tốt, vì vậy nó sẽ lấp đầy toàn bộ màn hình.

Nhưng tại sao Container lại quyết định điều đó? Đơn giản vì đó là quyết định thiết kế của những người đã tạo ra tiện ích con Container. Nó có thể được tạo theo cách khác và bạn phải đọc [tài liệu về Container](https://api.flutter.dev/flutter/widgets/Container-class.html) để hiểu nó hoạt động như thế nào, tùy thuộc vào từng trường hợp.

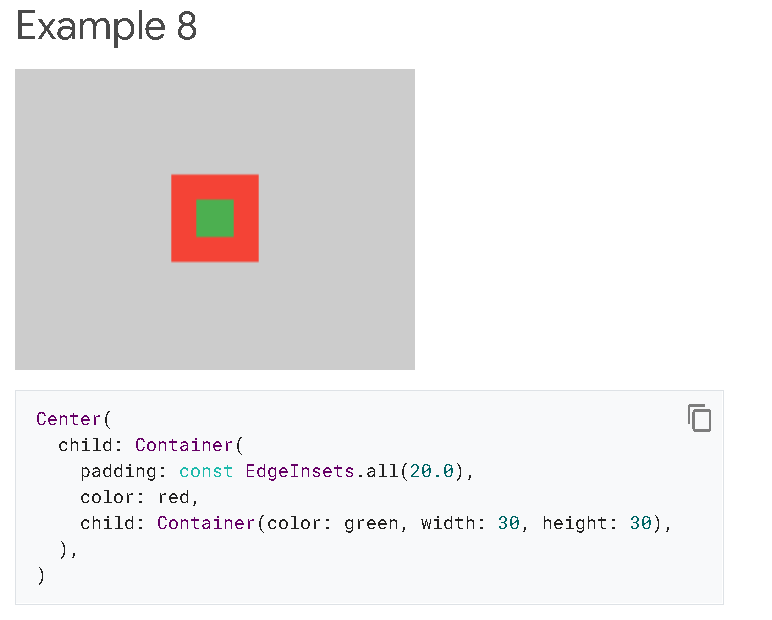


Màn hình buộc Center có kích thước chính xác bằng màn hình, vậy nên Center lấp đầy màn hình.

Center cho Container màu đỏ biết rằng nó có thể có kích thước bất kỳ mà nó muốn, nhưng không được lớn hơn màn hình. Vì Container màu đỏ không có kích thước nhưng có một con, nó quyết định nó muốn có cùng kích thước với con của nó.

Container màu đỏ nói với đứa con của nó rằng nó có thể có kích thước bất kỳ mà nó muốn, nhưng không được lớn hơn màn hình.

Con là Container màu xanh lục muốn có kích thước là 30 × 30. Cho rằng Container màu đỏ có kích thước tương đương với kích thước của con của nó, nó cũng là 30 × 30. Màu đỏ không thể nhìn thấy vì Container màu xanh lá cây bao phủ hoàn toàn màu đỏ Container.

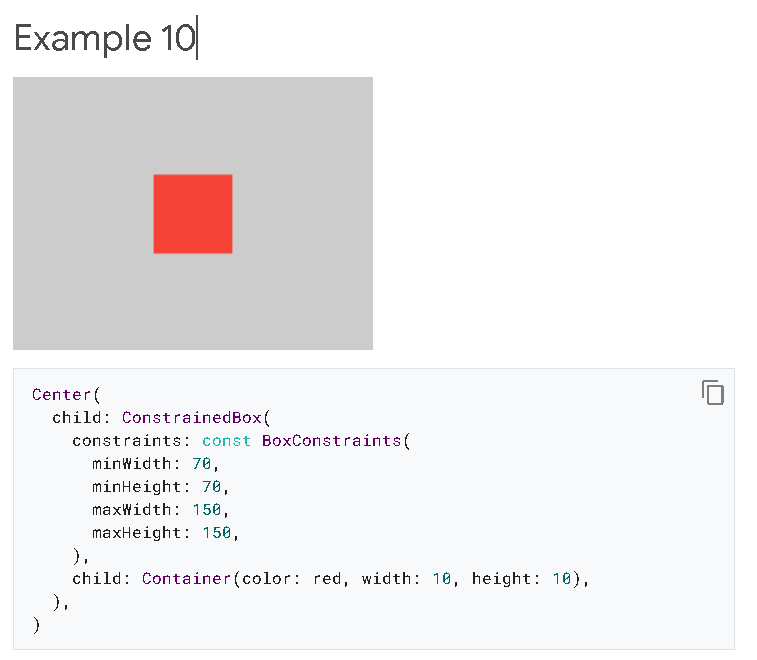


Container màu đỏ có kích thước tương ứng với kích thước của children, nhưng cần cân nhắc phần đệm riêng. Vì vậy, nó cũng là 30 × 30 cộng với đệm. Màu đỏ có thể nhìn thấy do có phần đệm và Container màu xanh lá cây có cùng kích thước như trong ví dụ trước.



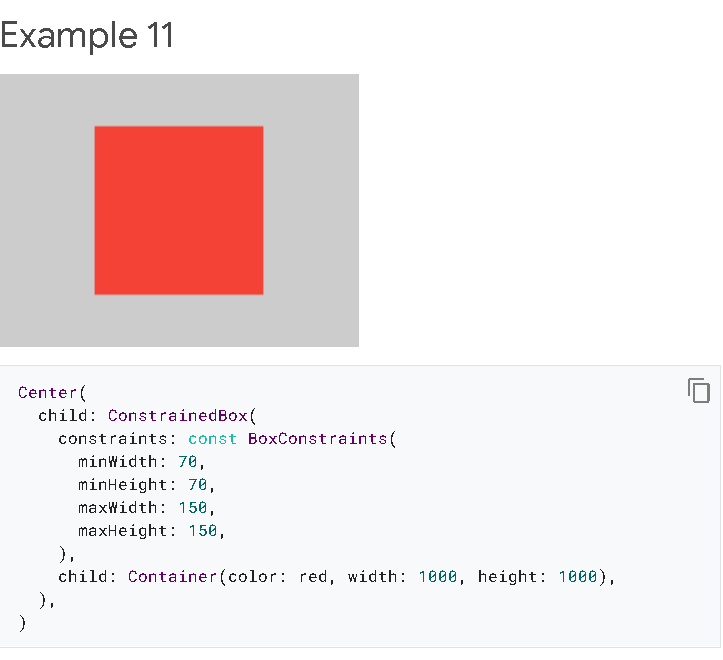
Bạn có thể đoán rằng Container phải từ 70 đến 150 pixel, nhưng bạn đã nhầm. ConstrainedBox chỉ áp đặt các ràng buộc bổ sung từ những ràng buộc mà nó nhận được từ cha mẹ của nó.

Ở đây, màn hình buộc ConstrainedBox phải có cùng kích thước chính xác với màn hình, vì vậy nó yêu cầu Container con của nó cũng giả định kích thước của màn hình, do đó bỏ qua tham số ràng buộc của nó.



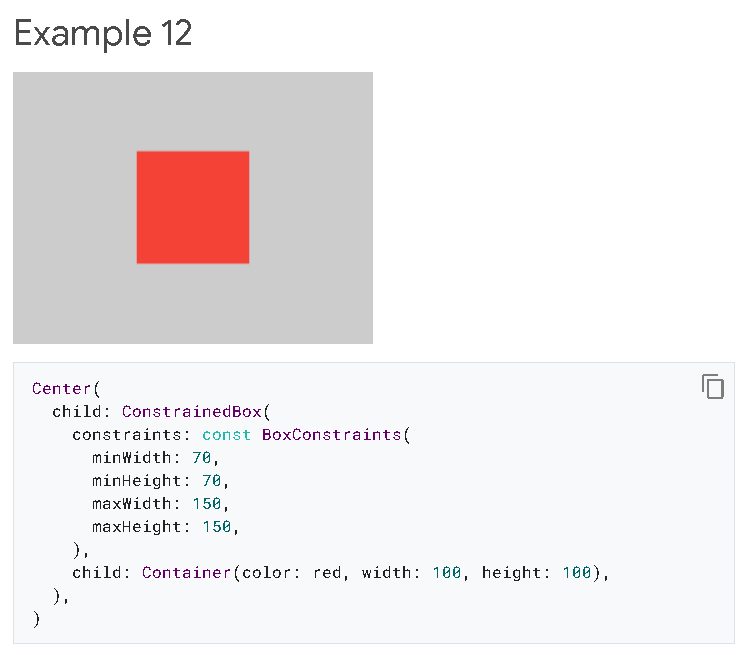
Giờ đây, Center cho phép ConstrainedBox có kích thước bất kỳ lên đến kích thước màn hình. ConstrainedBox áp đặt các ràng buộc **bổ sung** từ tham số ràng buộc lên con của nó.

Container phải từ 70 đến 150 pixel. Nó muốn có 10 pixel, vì vậy nó phải có 70 (mức tối thiểu).



Center cho phép ConstrainedBox có kích thước bất kỳ lên đến kích thước màn hình. ConstrainedBox áp đặt các ràng buộc **bổ sung** từ tham số ràng buộc lên con của nó.

Container phải từ 70 đến 150 pixel. Nó muốn có 1000 pixel, vì vậy nó phải có 150 (tối đa).

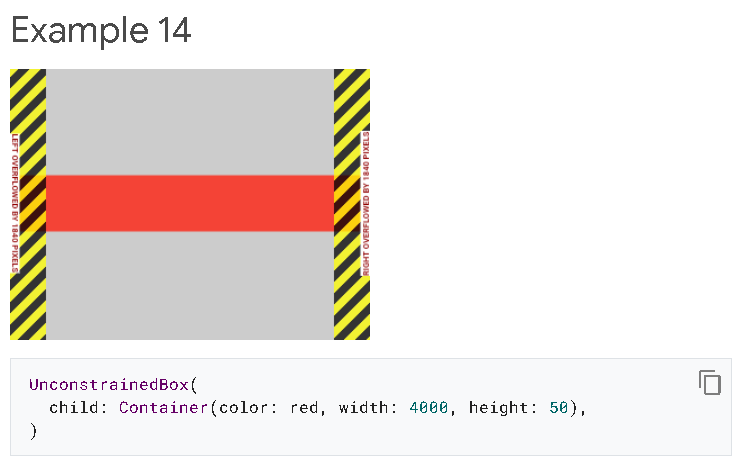


Center cho phép ConstrainedBox có kích thước bất kỳ lên đến kích thước màn hình. ConstrainedBox áp đặt các ràng buộc **bổ sung** từ tham số ràng buộc lên con của nó.

Container phải từ 70 đến 150 pixel. Nó muốn có 100 pixel và đó là kích thước mà nó có, vì đó là từ 70 đến 150.

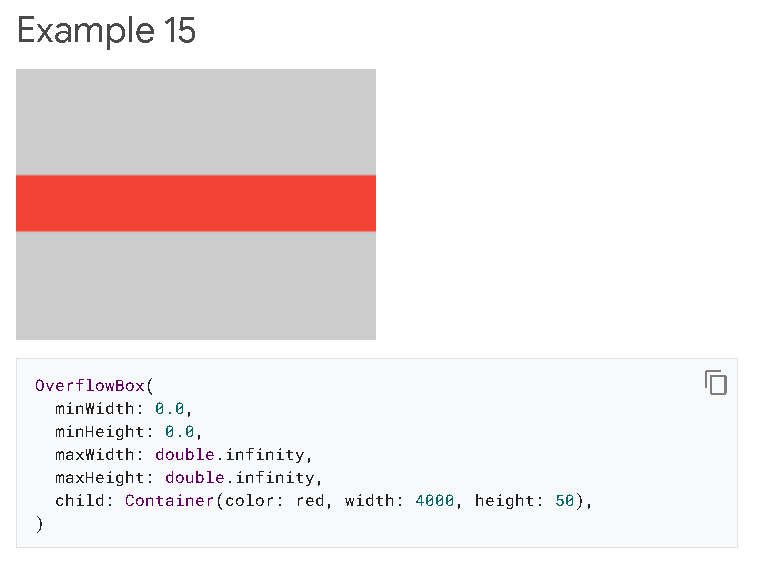


Màn hình buộc UnconstrainedBox phải có cùng kích thước với màn hình. Tuy nhiên, UnconstrainedBox cho phép Container con của nó có kích thước bất kỳ mà nó muốn.



Màn hình buộc UnconstrainedBox phải có cùng kích thước chính xác với màn hình và UnconstrainedBox cho phép Container con của nó có kích thước bất kỳ mà nó muốn.

Thật không may, trong trường hợp này, Container rộng 4000 pixel và quá lớn để vừa với UnconstrainedBox, vì vậy UnconstrainedBox hiển thị “cảnh báo tràn” rất đáng sợ.



Màn hình buộc OverflowBox phải có cùng kích thước với màn hình và OverflowBox cho phép Vùng chứa con của nó có kích thước bất kỳ mà nó muốn.

OverflowBox tương tự như UnconstrainedBox; sự khác biệt là nó sẽ không hiển thị bất kỳ cảnh báo nào nếu đứa trẻ không phù hợp với không gian.

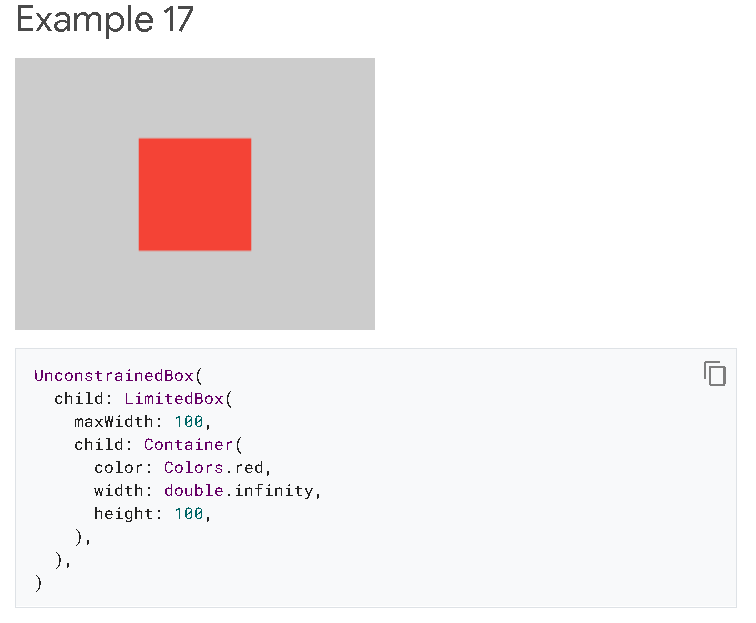
Trong trường hợp này, Container có chiều rộng 4000 pixel và quá lớn để vừa với OverflowBox, nhưng OverflowBox chỉ hiển thị nhiều nhất có thể mà không có cảnh báo nào được đưa ra.



Điều này sẽ không hiển thị bất kỳ thứ gì và bạn sẽ thấy lỗi trong bảng điều khiển.

UnconstrainedBox cho phép con của nó có kích thước bất kỳ mà nó muốn, tuy nhiên con của nó là một Container có kích thước vô hạn.

Flutter không thể hiển thị kích thước vô hạn, vì vậy nó gây ra lỗi với thông báo sau: BoxConstraints buộc phải có chiều rộng vô hạn.



Ở đây, bạn sẽ không gặp phải lỗi nữa, bởi vì khi LimitedBox được cung cấp cho một kích thước vô hạn bởi UnconstrainedBox; nó chuyển chiều rộng tối đa là 100 xuống con của nó.

Nếu bạn hoán đổi UnconstrainedBox cho một Center widget, LimitedBox sẽ không áp dụng giới hạn của nó nữa (vì giới hạn của nó chỉ được áp dụng khi nó có các ràng buộc vô hạn) và chiều rộng của Container được phép tăng quá 100.

Điều này giải thích sự khác biệt giữa LimitedBox và ConstrainedBox.



Màn hình buộc FittedBox phải có cùng kích thước với màn hình. Text có một số chiều rộng tự nhiên (còn được gọi là chiều rộng nội tại của nó) phụ thuộc vào số lượng văn bản, kích thước phông chữ của nó, v.v.

FittedBox cho phép Text có kích thước bất kỳ mà nó muốn, nhưng sau khi Text cho biết kích thước của nó với FittedBox, FittedBox sẽ chia tỷ lệ Text cho đến khi nó lấp đầy tất cả chiều rộng có sẵn.



Nhưng điều gì sẽ xảy ra nếu bạn đặt FittedBox bên trong tiện ích con ở Center? Center cho phép FittedBox ở bất kỳ kích thước nào nó muốn, lên đến kích thước màn hình.

Sau đó FittedBox tự định kích thước cho Text và để cho Text có kích thước bất kỳ mà nó muốn. Vì cả FittedBox và Text đều có cùng kích thước nên không xảy ra hiện tượng chia tỷ lệ.

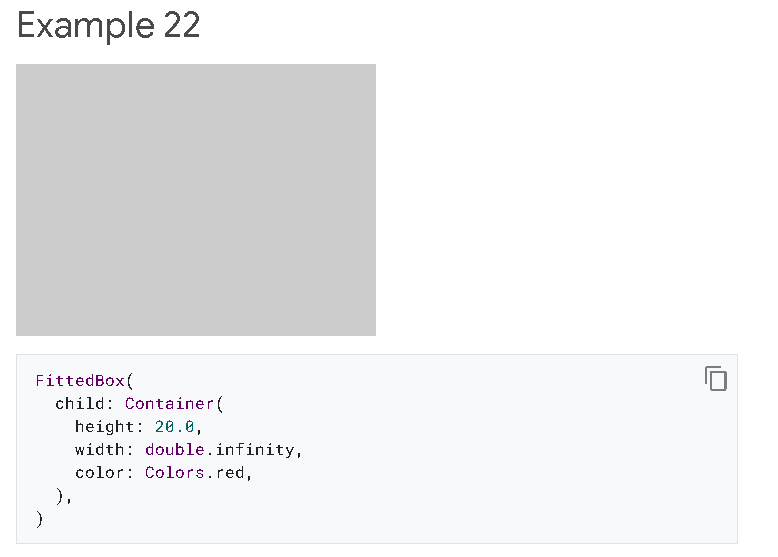


Tuy nhiên, điều gì sẽ xảy ra nếu FittedBox nằm bên trong tiện ích Center, nhưng Text quá lớn để vừa với màn hình?

FittedBox cố gắng tự định kích thước cho Text, nhưng nó không thể lớn hơn màn hình. Sau đó, nó giả định kích thước màn hình và thay đổi kích thước Text để nó cũng vừa với màn hình.



Tuy nhiên, nếu bạn xóa FittedBox, Text sẽ có chiều rộng tối đa khỏi màn hình và ngắt dòng sao cho vừa với màn hình.

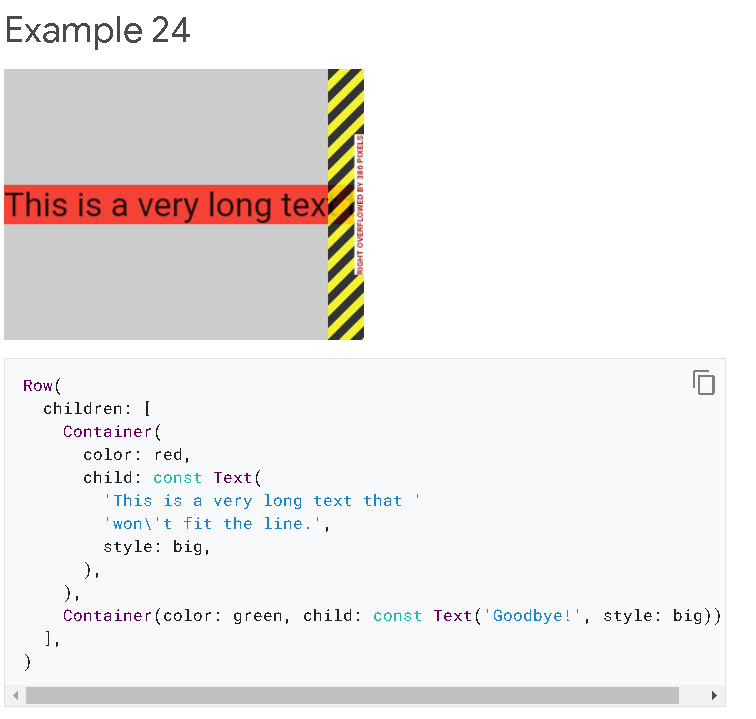


FittedBox chỉ có thể chia tỷ lệ một tiện ích con bị giới hạn (có chiều rộng và chiều cao không vô hạn). Nếu không, nó sẽ không hiển thị bất kỳ thứ gì và bạn sẽ thấy lỗi trong bảng điều khiển.



Màn hình buộc Row phải có cùng kích thước với màn hình.

Cũng giống như UnconstrainedBox, Row sẽ không áp đặt bất kỳ ràng buộc nào đối với các con của nó mà thay vào đó để chúng ở bất kỳ kích thước nào họ muốn. Sau đó Row đặt chúng cạnh nhau và bất kỳ khoảng trống thừa nào vẫn còn trống.



Vì Row sẽ không áp đặt bất kỳ ràng buộc nào đối với các con của nó, nên rất có thể các con có thể quá lớn để phù hợp với chiều rộng có sẵn của Row. Trong trường hợp này, giống như UnconstrainedBox, Row hiển thị “cảnh báo tràn”.



Khi con của một Row được bao bọc trong Expanded widget, Row sẽ không để con này xác định chiều rộng của chính nó nữa.

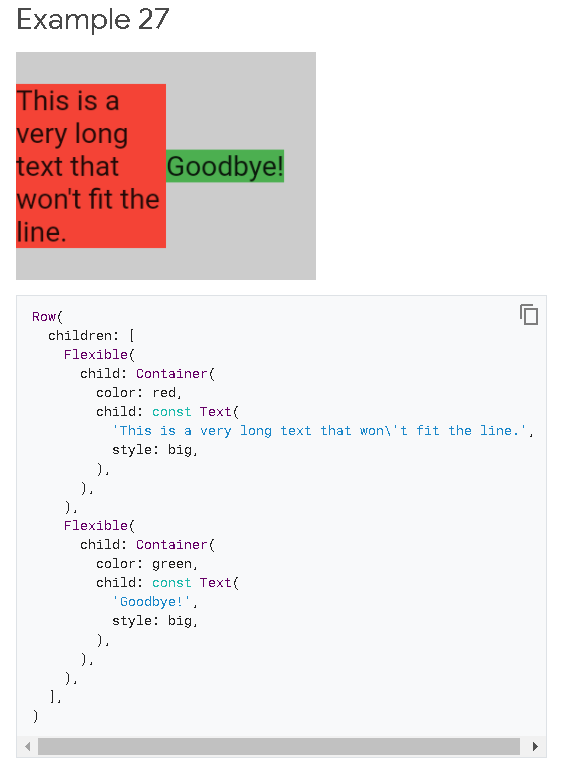
Thay vào đó, nó xác định chiều rộng được mở rộng theo các phần tử con khác và chỉ khi đó, tiện ích con được mở rộng mới buộc phần tử con ban đầu có chiều rộng được mở rộng.

Nói cách khác, khi bạn sử dụng Expanded, chiều rộng của phần con ban đầu trở nên không liên quan và bị bỏ qua.



Nếu tất cả các con của Row đều được bao bọc trong các Expanded widget, thì mỗi Expanded có kích thước tỷ lệ với thông số linh hoạt của nó và chỉ khi đó mỗi Expanded widget mới buộc con của nó có chiều rộng được Mở rộng.

Nói cách khác, Expanded bỏ qua chiều rộng ưa thích của các phần tử con của nó.



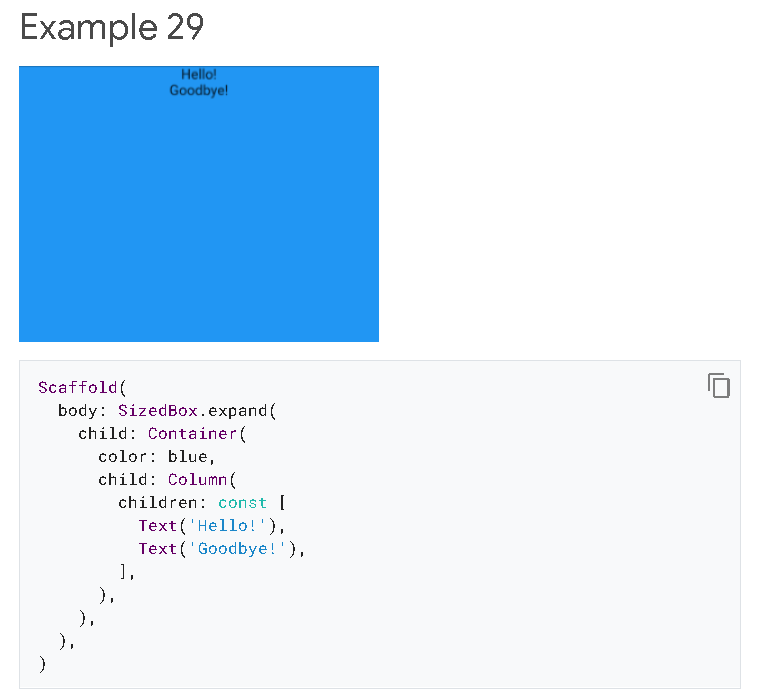
Sự khác biệt duy nhất nếu bạn sử dụng Flexible thay vì Expanded, là Flexible cho phép con của nó có cùng chiều rộng hoặc nhỏ hơn chính Flexible, trong khi Expanded buộc con của nó phải có cùng chiều rộng chính xác của Expanded. Nhưng cả Expanded và Flexible đều bỏ qua chiều rộng của con khi tự định kích thước.

Lưu ý: Điều này có nghĩa là không thể mở rộng Row con tương ứng với kích thước của chúng. Row sử dụng chiều rộng chính xác của con hoặc bỏ qua hoàn toàn khi bạn sử dụng Expanded hoặc Flexible.



Màn hình buộc Scaffold phải có cùng kích thước với màn hình, vì vậy Scaffold sẽ lấp đầy màn hình. Scaffold nói với Container rằng nó có thể có kích thước bất kỳ mà nó muốn, nhưng không được lớn hơn màn hình.

Lưu ý: Khi một widget nói với con của nó rằng nó có thể nhỏ hơn một kích thước nhất định, chúng tôi nói rằng widget cung cấp các ràng buộc *lỏng lẻo* cho con của nó. Thêm về điều đó sau.



Nếu bạn muốn con của Scaffold có cùng kích thước với chính con của Scaffold, bạn có thể bọc con của nó bằng SizedBox.expand.

Lưu ý: Khi một widget nói với con của nó rằng nó phải có kích thước nhất định, chúng tôi nói rằng widget cung cấp các ràng buộc *chặt chẽ* cho con của nó.

## Ràng buộc chặt chẽ và lỏng lẻo

Rất phổ biến khi nghe nói rằng một số ràng buộc là "chặt chẽ" hoặc "lỏng lẻo", vì vậy bạn nên biết điều đó có nghĩa là gì.

Một hạn chế *chặt chẽ* cung cấp một khả năng duy nhất, một kích thước chính xác. Nói cách khác, một ràng buộc chặt chẽ có chiều rộng tối đa bằng chiều rộng tối thiểu của nó; và có chiều cao tối đa bằng chiều cao tối thiểu của nó.

Nếu bạn truy cập tệp box.dart của Flutter và tìm kiếm các hàm tạo BoxConstraints, bạn sẽ tìm thấy những thứ sau:



Nếu bạn truy cập lại ví dụ 2 ở trên, nó cho chúng ta biết rằng màn hình buộc Container màu đỏ có cùng kích thước với màn hình. Tất nhiên, màn hình thực hiện điều đó bằng cách chuyển các ràng buộc chặt chẽ đến Container.

Mặt khác, một ràng buộc lỏng lẻo đặt chiều rộng và chiều cao **tối đa**, nhưng cho phép tiện ích nhỏ như ý muốn. Nói cách khác, một ràng buộc lỏng lẻo có chiều rộng và chiều cao **tối thiểu** đều bằng **0**:



Nếu bạn truy cập lại ví dụ 3, nó cho chúng ta biết rằng Center để cho Container màu đỏ nhỏ hơn, nhưng không lớn hơn màn hình. Tất nhiên, Center thực hiện điều đó bằng cách chuyển các ràng buộc lỏng lẻo đến Container. Cuối cùng, mục đích chính của Center là chuyển đổi các ràng buộc chặt chẽ mà nó nhận được từ cha của nó (màn hình) thành các ràng buộc lỏng lẻo cho con của nó (Container).

## Tìm hiểu về quy tắc bố cục cho các widget riêng biệt

Biết quy tắc bố cục chung là cần thiết nhưng vẫn chưa đủ.

Mỗi tiện ích con có nhiều quyền tự do khi áp dụng quy tắc chung, vì vậy không có cách nào để biết nó sẽ làm gì bằng cách chỉ đọc tên của tiện ích con.

Nếu bạn cố gắng đoán, có thể bạn sẽ đoán sai. Bạn không thể biết chính xác cách một tiện ích con hoạt động trừ khi bạn đã đọc tài liệu của nó hoặc nghiên cứu mã nguồn của nó.

Mã nguồn của bố cục thường phức tạp, vì vậy có lẽ tốt hơn là bạn chỉ cần đọc tài liệu. Tuy nhiên, nếu bạn quyết định nghiên cứu mã nguồn bố cục, bạn có thể dễ dàng tìm thấy nó bằng cách sử dụng các khả năng điều hướng của IDE của bạn.

Đây là ví dụ:

* Tìm một *Column* trong mã của bạn và điều hướng đến mã nguồn của nó. Để thực hiện việc này, hãy sử dụng *command + B* (macOS) hoặc *control + B* (Windows / Linux) trong Android Studio hoặc IntelliJ. Bạn sẽ được đưa đến tệp *basic.dart*. Vì Column mở rộng Flex, hãy điều hướng đến mã nguồn Flex (cũng trong *basic.dart*).
* Cuộn xuống cho đến khi bạn tìm thấy một phương thức có tên *createRenderObject ()*. Như bạn thấy, phương thức này trả về một RenderFlex. Đây là đối tượng kết xuất cho *Column*. Bây giờ điều hướng đến mã nguồn của *RenderFlex*, sẽ đưa bạn đến tệp *flex.dart*.
* Cuộn xuống cho đến khi bạn tìm thấy một phương thức có tên là *performanceLayout ()*. Đây là phương pháp thực hiện bố trí cho *Column*.



# Xử lý các hạn chế về hộp

Trong Flutter, các widget được hiển thị bởi các đối tượng RenderBox bên dưới của chúng. Các hộp kết xuất được cung cấp bởi các ràng buộc bởi cha mẹ của chúng và tự kích thước trong các ràng buộc đó. Ràng buộc bao gồm chiều rộng và chiều cao tối thiểu và tối đa; kích thước bao gồm chiều rộng và chiều cao cụ thể.

Nói chung, có ba loại hộp, về cách chúng xử lý các ràng buộc của chúng:

Những người cố gắng càng lớn càng tốt. Ví dụ, các hộp được sử dụng bởi Center và ListView.

Những con cố gắng có cùng kích thước với con của chúng. Ví dụ: các hộp được Transform và Opacity sử dụng.

Những người cố gắng có một kích thước cụ thể. Ví dụ, các hộp được sử dụng bởi Hình ảnh và Văn bản.

Một số widget, ví dụ như Container, khác nhau giữa các loại dựa trên các đối số phương thức khởi tạo của chúng. Trong trường hợp của Container, nó mặc định cố gắng để càng lớn càng tốt, nhưng nếu bạn cung cấp cho nó chiều rộng chẳng hạn, nó sẽ cố gắng tôn trọng điều đó và có kích thước cụ thể đó.

Những thứ khác, ví dụ Hàng và Cột (hộp linh hoạt) thay đổi dựa trên các ràng buộc mà chúng được đưa ra, như được mô tả bên dưới trong phần “Linh hoạt”.

Các ràng buộc đôi khi “chặt chẽ”, có nghĩa là chúng không để lại chỗ cho hộp kết xuất quyết định kích thước (ví dụ: nếu chiều rộng tối thiểu và tối đa bằng nhau, nó được cho là có chiều rộng hẹp). Một ví dụ về điều này là tiện ích con Ứng dụng, được chứa bởi lớp RenderView: hộp được con sử dụng bởi chức năng xây dựng của ứng dụng được đưa ra một ràng buộc buộc nó phải lấp đầy chính xác vùng nội dung của ứng dụng (thường là toàn bộ màn hình) . Nhiều hộp trong Flutter, đặc biệt là những hộp chỉ lấy một đứa trẻ duy nhất, vượt qua hạn chế của chúng cho con cái của họ. Điều này có nghĩa là nếu bạn lồng nhiều hộp vào bên trong nhau ở gốc của cây kết xuất của ứng dụng, tất cả chúng sẽ hoàn toàn khớp với nhau, bị ép buộc bởi những ràng buộc chặt chẽ này.

Một số hộp nới lỏng các ràng buộc, nghĩa là mức tối đa được duy trì nhưng mức tối thiểu bị loại bỏ. Ví dụ, Trung tâm.

## Ràng buộc không giới hạn

Trong một số tình huống nhất định, ràng buộc được cung cấp cho một hộp là không giới hạn hoặc vô hạn. Điều này có nghĩa là chiều rộng tối đa hoặc chiều cao tối đa được đặt thành double.infinity.

Một hộp cố gắng càng lớn càng tốt sẽ không hoạt động hữu ích khi được cung cấp một ràng buộc không bị ràng buộc và trong chế độ gỡ lỗi, sự kết hợp như vậy sẽ ném ra một ngoại lệ trỏ đến tệp này.

Các trường hợp phổ biến nhất trong đó hộp kết xuất tự tìm thấy với các ràng buộc không bị ràng buộc nằm trong các hộp linh hoạt (Hàng và Cột) và trong các vùng có thể cuộn (ListView và các lớp con ScrollView khác).

Cụ thể, ListView cố gắng mở rộng để phù hợp với không gian có sẵn theo hướng chéo của nó (ví dụ: nếu đó là một khối cuộn theo chiều dọc, nó sẽ cố gắng rộng bằng khối cha mẹ của nó). Nếu bạn lồng một ListView cuộn theo chiều dọc bên trong ListView cuộn theo chiều ngang, thì cái bên trong sẽ cố gắng rộng nhất có thể, cái này rộng vô hạn, vì cái bên ngoài có thể cuộn theo hướng đó.

## Uốn cong

Bản thân các hộp linh hoạt (Hàng và Cột) hoạt động khác nhau dựa trên việc chúng nằm trong các ràng buộc có giới hạn hay các ràng buộc không bị ràng buộc theo hướng đã cho của chúng.

Trong những ràng buộc bị ràng buộc, họ cố gắng trở nên lớn nhất có thể theo hướng đó.

Trong những ràng buộc không có giới hạn, họ cố gắng để con cái của họ đi theo hướng đó. Trong trường hợp này, bạn không thể đặt flex trên con thành bất kỳ thứ gì khác ngoài 0. Trong thư viện widget, điều này có nghĩa là bạn không thể sử dụng Expanded khi hộp flex nằm trong một hộp flex khác hoặc bên trong một cuộn có thể cuộn được. Nếu bạn làm vậy, bạn sẽ nhận được một thông báo ngoại lệ hướng dẫn bạn đến tài liệu này.

Theo hướng chéo, ví dụ: về chiều rộng cho Cột (linh hoạt dọc) hoặc chiều cao cho Hàng (uốn cong ngang), chúng không bao giờ được không bị ràng buộc, nếu không chúng sẽ không thể căn chỉnh hợp lý các con của chúng.