I. Tạo responsive and ứng dụng adaptive

Một trong những mục tiêu chính của Flutter cho phép chúng ta phát triển ứng dụng tự phát triển ứng dụng di động đa nền tảng từ một cơ sở mã duy nhất.

Ứng dụng có thể xuất hiện trên các màn hình với độ phân giải cao, kích thước khác nhau như đồng hồ, điện thoại,màn hình có độ nét cao.

Hai thuật ngữ mô tả cho các khái niệm này là thích ứng và đáp ứng. Thuật ngữ tương tự nhau nhưng chúng không giống nhau về các mặt. Điều lý tưởng nhất trong ứng dụng là thích ứng và đáp ứng, nhưng rất khó để tạo ra một sản phẩm hoàn thiện về mọi mặt.

I.1. Sự khác biệt giữa sự thích ứng và đáp ứng trong ứng dụng

- Thích ứng và đáp ứng có thể xem là hai khía cạnh riêng biệt của một ứng dụng. Một ứng dụng thích ứng không đáp ứng và ngược lại, hoặc một ứng dụng không có cả hai và ngược lại.

I.1.1. Responsive

- Thông thường, ứng dụng được điều chỉnh bố cục phù hợp kích thước màn hình có sẵn. Điều này có nghĩa là ứng dụng tự động bố trí lại giao diện nếu người dung đổi hướng hoặc thay đổi kích thước cửa sổ của thiết bị. Điều này đặc biệt cần thiết để một ứng dụng chạy được trên nhiều thiết bị khác nhau.

I.1.2. Adaptive

- Điều chỉnh ứng dụng chạy trên nhiều thiết bị khác nhau như di động hoặc máy tính để bàn, yêu cầu xử lý bằng: chuột, bàn phím, cảm ứng…Điều đó có nghĩa là có những sự kỳ vọng khác nhau về mật độ hình ảnh, độ phân giải, tốc độ của ứng dụng, cách lựa chọn thành phần hoạt động, sử dụng các tính năng dành riêng cho mỗi nền tảng.

I.2. Tạo ứng dụng Flutter đáp ứng

- Flutter cho phép tạo các ứng dụng thích ứng với kích thước và hướng màn hình của thiết bị.
- Có hai cách cơ bản để tạo ứng dụng Flutter với thiết kế đáp ứng:

I.2.1. Sử dụng LayoutBuilder class

- Từ thuộc tính trình tạo của LayoutBuilder, ta nhận được đối tượng BoxConstraints. Kiểm tra các thuộc tính rang buộc quyết định những gì sẽ hiển thị. Ví dụ: điều chỉnh màn hình dựa theo chiều cao của thiết bị, tỉ lệ khung hình và các thuộc tính khác. Khi các rằng buộc thay đổi chức năng xay dựng sẽ chạy.

I.2.2. Sử dụng MediaQuery.of() trong các hàm xây dựng

- Cung cấp cho chúng ta kích thước, hướng… của ứng dụng hiện tại. Sẽ hữu ích hơn nếu dựa trên bối cảnh hoàn chỉnh thay vì chỉ dựa trên kích thước của tiện ích cụ thể. Nếu sử dụng MediaQuery.of() method thì chức năng xây dựng sẽ tự động chạy nếu người dùng thay đổi kích thước của ứng dụng bằng một số cách nào đó.
- Các Widget và classes khác để tạo giao diện người dung đáp ứng:
    - AspectRatio
    - CustomSingleChildLayout
    - CustonMultiChildLayout
    - FittedBox
    - FractionallySizedBox
    - LayoutBuilder
    - MediaQuery
    - MediaQueryData

- OrientationBuilder

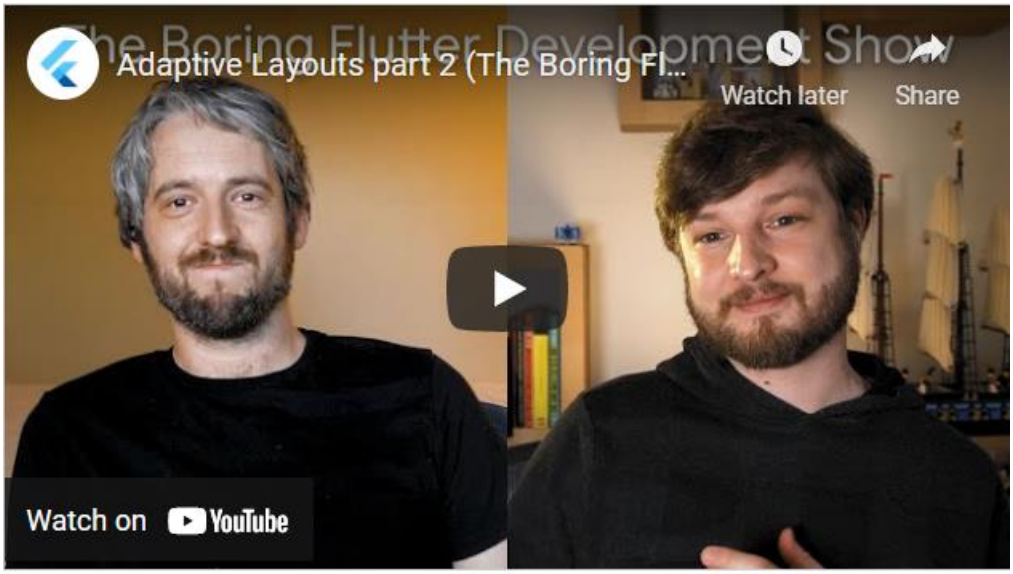
I.1.1.a Các nguồn khác

- Các thông tin tài nguyên bao gồm cả đóng góp từ cộng đồng Flutter:
  - Developing for Multiple Screen Sizes and Orientations in Flutter bởi Deven Joshi.
  - Build Responsive UIs in Flutter bởi Raouf Rahiche.
  - Making Cross-platform Flutter Landing Page Responsive bởi Priyanka Tyagi.
  - How to make flutter app responsive according to different screen size?, a question on StackOverflow.

I.3. Tạo ứng dụng adaptive Flutter

- Tìm hiểu them về cách tạo ứng dụng Flutter adaptive với cách xây dựng ứng dụng thích ứng do nhóm gskinner viết.

Adaptive layouts



Adaptive layouts, part 2

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

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

I.4. Các nguồn khác

- Mọi người có thể tìm hiểu thêm về cách tạo ứng dụng thích ứng với nền tảng trong các tài nguyên sau:
    - Platform-specific behaviors and adaptations, một trang trên trang web này.
    - Designing truly adaptive user interfaces, một bài đăng trên blog và video của Aloïs Deniel, được trình bày tại hội nghị FlutterViking năm nay.
    - The Flutter gallery app (repo) được viết như một ứng dụng thích ứng.

II. Xây dựng ứng dụng adaptive

II.1. Tổng quát

- Flutter cung cấp các cơ hội để xây dựng các ứng dụng chạy trên các thiết bị di động, máy tính để bàn và web từ một cơ sở mã duy nhất. Điều này sẽ đi kèm với những thách thức mới như chúng ta muôn ứng dụng dễ dùng cho người sử dụng, thích ứng được với các nền tảng bằng cách tối ưu hóa khả năng sử dụng và đảm bảo trải nghiệm tuyệt vời cho người dùng. Do đó, chúng ta không chỉ phải xây dựng ứng dụng đa nền tảng mà còn phải thhicsh ứng hoàn toàn với nền tảng đó.
- Có nhiều cân nhắc để phát triển các ứng dụng thích ứng với nền tảng, nhưng chúng thuộc ba loại chính: **Layout, Input, Idioms and noms.**

II.2. Xây dựng bố cục adaptive

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

II.2.1. Bố cục Widgets

- Khi chúng ta đã quen với việc tạo giao diện đáp ứng, nhà phát triển Flutter có các widget lớn giúp hỗ trợ công việc dễ dàng hơn.
- Một số tiện ích bố cục hữu ích nhất của Flutter:

(1) Single child

- Align – Căn chỉnh child trong chính nó, nó nhận giá trị kép giả -1 và 1 cho cả căn chỉnh dọc và ngang.
- AspectRatio –Điều chỉnh kích thước child theo một tỉ lệ khung hình nhất định.
- ConstrainedBox – Áp đặt giới hạn kích thước lên child, cung cấp quyền kiểm soát kích thước tối thiểu hoặc tối đa.

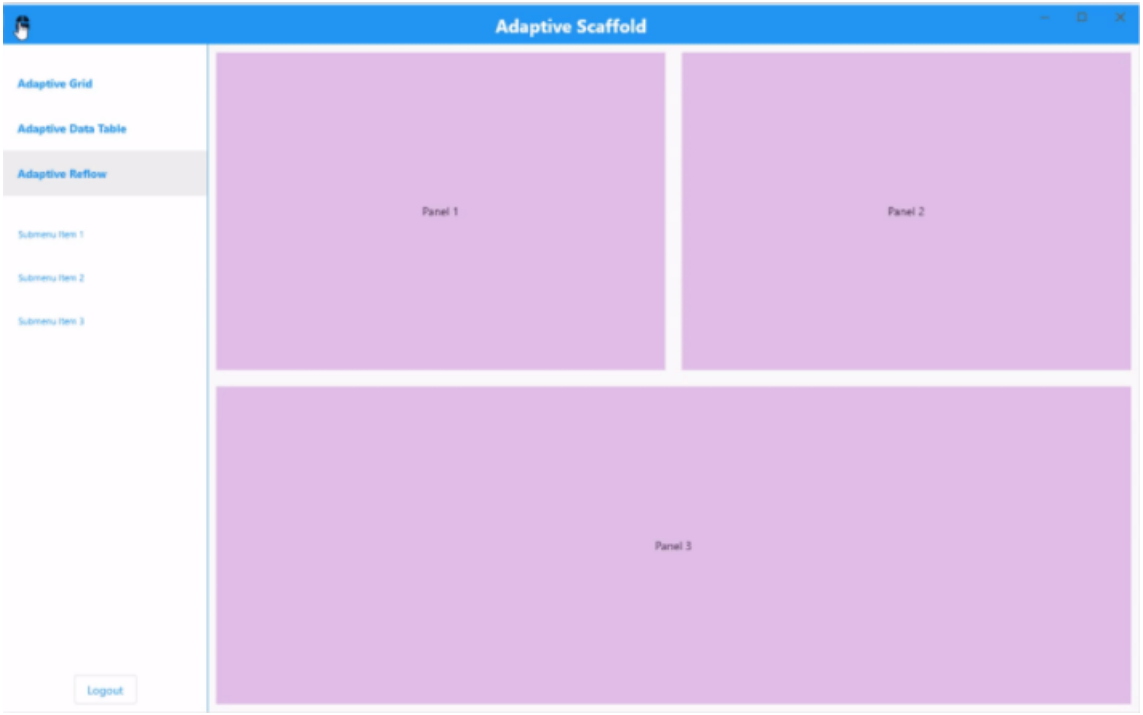
- CustomSingleChildLayout – Sử dụng chức năng ủy quyền để định vị phần tử con. Người được ủy quyền có thể xác định các ràng buộc về bố cục và vị trí của child.
- Expanded and Flexible – Cho phép child của hàng và cột thu nhỏ hoặc phát triển để lấp đầy bất kì không gian có sẵn nào.
- FractionallySizedBox – Kích thước con của nó thành một phần nhỏ của không gian có sẵn.
- LayoutBuilder – Tạo một widget con có thể tự điều chỉnh lại dựa trên kích thước có trước đó.
- SingleChildScrollView – Thêm cuộn vào single child. Thường là hàng hoặc cột.

(2) Multichild

- Column, Row và Flex – đặt child trong một lần chạy ngang hoặc dọc. Cả cột và hàng đều mở rộng linh hoạt widget.
- CustomMultiChildLayout – Sử dụng chức năng ủy quyền để định vị các phần từ con trong giai đoạn bố trí.
- Flow – tương tự như CustomMultiChildLayout, hiệu quả hơn vì nó thực hiện trong giai đoạn bố trí.
- ListView, GridView, và CustomScrollView - Cung cấp danh sách con có thể cuộn được.
- Stack – các lớp và định vị nhiều con so với các cạnh của Stack, các chức năng tương tự như cố định vị trí 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.

II.2.2. Visual density

- Các thiết bị đầu vào cung cấp các mức độ chính xác khác nhau, đòi hỏi các vùng có kích thước khác nhau. Lớp VisualDensity của Flutter giúp ta dễ dàng điều chỉnh mật độ các chế độ xem của mình trên toàn ứng dụng, chẳng hạn như bằng cách làm cho một một nút lớn hơn trên thiết bị cảm ứng.
- Khi thay đổi VisualDensity cho MaterialApp, MaterialComponents hỗ trợ nó sẽ kích hoạt mật độ của chúng để phù hợp. Bằng cách chuyển đổi giữa các mật độ khác nhau, ta có thể dễ dàng điều chỉnh giao diện người dùng của mình.



- Để đặt mật độ hình ảnh tùy chỉnh, đưa mật độ vào MaterialApp:

```
double densityAmt = enableTouchMode ? 0.0 : -1.0;
VisualDensity density = VisualDensity(horizontal: density, vertical: density);
return MaterialApp(
  theme: ThemeData(visualDensity: density),
  ...
);
```

- Sử dụng VisualDensity dạng riêng của chúng ta, tra cứu:

```
VisualDensity density = Theme.of(context).visualDensity;
return Padding(
  padding: EdgeInsets.all(Insets.large + density.vertical * 4),
  child: ...
);
```

- Vùng chứa không chỉ tự động phản ứng với những thay đổi về mật độ mà còn linh hoạt ảnh khi nó thay đổi. Điều này liên kết các thành phần lại 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 ý nghĩa khác nhau đối với các chế độ xem khác nhau, thực tế không đơn vị làm nó khá linh hoạt và nó sẽ hoạt động trong hầu hết các ngữ cảnh.
- Cần lưu ý Material Components thường sử dụng khoảng 4 pixel logic cho mỗi đơn vị mật độ hình ảnh.

II.2.3. Bố cục theo ngữ cảnh

- Nếu cần nhiều hơn những thay đổi về mật độ và không thể tìm thấy một tiện ích con đáp ứng được những gì chúng ta cần, ta 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 phù hợp với một hệ số dạng cụ thể.

(1) Screen-based breakpoints

- Điều này 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:

```
class FormFactor {
 static double desktop = 900;
 static double tablet = 600;
 static double handset = 300;
}
```

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

```
ScreenType getFormFactor(BuildContext context) {
 // Use .shortestSide to detect device type regardless of orientation
 double deviceWidth = MediaQuery.of(context).size.shortestSide;
 if (deviceWidth > FormFactor.desktop) return ScreenType.Desktop;
 if (deviceWidth > FormFactor.tablet) return ScreenType.Tablet;
 if (deviceWidth > FormFactor.handset) return ScreenType.Handset;
 return ScreenType.Watch;
}
```

- Chúng ta có thể trừu tượng hóa nhiều lên và định nghĩa từ nhỏ đến lớn:

```
enum ScreenSize { Small, Normal, Large, ExtraLarge }

ScreenSize getSize(BuildContext context) {
 double deviceWidth = MediaQuery.of(context).size.shortestSide;
 if (deviceWidth > 900) return ScreenSize.ExtraLarge;
 if (deviceWidth > 600) return ScreenSize.Large;
 if (deviceWidth > 300) return ScreenSize.Normal;
 return ScreenSize.Small;
}
```

- Các điểm ngắt tốt trên màn hình được sử dụng tốt nhất để đưa ra các quyết định cấp cao nhất trong ứng dụng. Thay đổi mật độ hình ảnh, khoảng đệm hoặc kích thước phông chữ là tốt nhất khi được xác định trên cơ sở toàn cầu.
- Chúng ta 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 chuyển từ bố cục dọc sang ngang khi người dụng không sử dụng thiết bị cầm tay:

```
bool isHandset = MediaQuery.of(context).size.width < 600;
return Flex(
  children: [...],
  direction: isHandset ?
    Axis.vertical :
    Axis.horizontal
);
```

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

```
Widget foo = Row(children: [
  BackButton(),
  ...isHandset ?
    _getHandsetChildren() :
    _getNormalChildren(),
],);
```

(2) Sử dụng LayoutBuilder 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 là sử dụng lớp LayoutBuilder, LayoutBuilder cho phép một widget đáp ứng các ràng buộc về kích thước cục bộ, làm cho widget linh hoạt hơn nếu có phụ thuộc vào một giá trị toàn cục.ví dụ:

```
Widget foo = LayoutBuilder(builder: (_, constraints, __){
 bool useVerticalLayout = constraints.maxWidth < 400.0;
 return Flex(
     children: [...],
     direction: useVerticalLayout ?
     Axis.vertical : Axis.horizontal
 );
});
```

- Widget này có thể được tạo trong bảng điều khiển, hộp thoại hoặc chế đ xem toàn màn hình hoặc điều chỉnh bố cục cho phù hợp với bất kỳ không gian nào được cung cấp.

(3) Phân đoạn thiết bị

- Đôi khi muốn đưa ra quyết định về bố cục dựa trên nền tảng thực tế đang chạy bất kể kích thước.
- Để xác định đang sử dụng kết hợp nền tảng nào, ta có thể sử dụng API nền tảng cùng với giá trị klsWeb:

```
bool get isMobileDevice => !kIsWeb && (Platform.isIOS || Platform.isAndroid);
bool get isDesktopDevice =>
    !kIsWeb && (Platform.isMacOS || Platform.isWindows || Platform.isLinux);
bool get isMobileDeviceOrWeb => kIsWeb || isMobileDevice;
bool get isDesktopDeviceOrWeb => kIsWeb || isDesktopDevice;
```

- Không thể truy cập API 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.

II.2.4. Single source of truth for styling

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

```
class Insets {
 static const double xsmall = 4;
 static const double small = 8;
 // etc
}


class Fonts {
 static const String raleway = 'Raleway';
 // etc
}

class TextStyles {
 static const TextStyle raleway = const TextStyle(fontFamily: Fonts.raleway, ... );
 static late TextStyle body1 = raleway.copyWith( ... );
 // etc
}
```

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

```
return Padding(
    insets: EdgeInsets.all(Insets.small),
    child: Text('Hello!', style: TextStyles.body1)
)
```

- Với các chế độ xem tham chiếu đến các quy tắc hệ thông thiết kế chia sẻ giống nhau, chúng có xu hương 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 kieesmm và thay thế dễ xảy ra lỗi, sử dụng các quy tắc được chia sẻ có lợi ích bổ dung là giúp thực thực thi tính nhất quán về mặt thiết kế.
- Một số danh mục hệ thống thiết kế phổ biến được biểu diễn theo cách này:
    - Thời gian hoạt ảnh.
    - Kích thước và điểm ngắt.
    - Insets và paddings.
    - Bán kính góc.
    - Shadows.
    - Strokes.
    - Font families, kích thucows và kiểu.
- Giống hầu hết các quy tắc, có những ngoại lệ: giá trị một lần không được sử dụng ở nơi khác. 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 được bắt nguồn từ một giá trị hiện có, việc sử dụng trùng lặp các giá trị nghữ nghĩa giống nhau, những giá trị này có thể sẽ được thêm vào bộ quy tắc tạo kiểu chung.

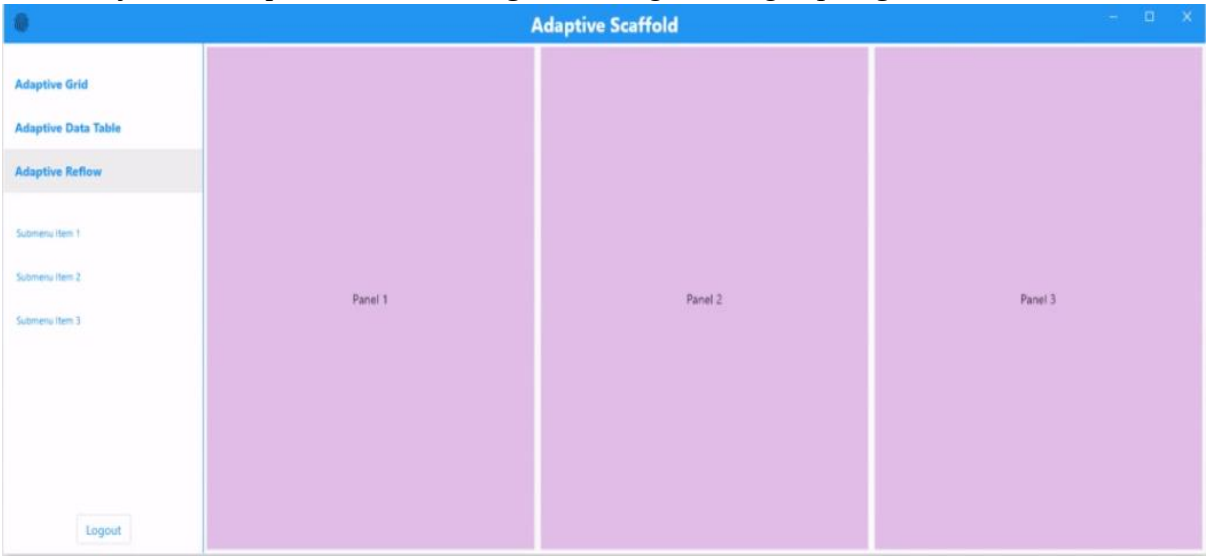
II.2.5. Thiết kế theo điểm mạnh của tứng yếu tố

- Ngoài kích thước màn hình, chúng ta cũng nên 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. Điều quan

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

II.2.6. Use desktop build targets for rapid testing

- Một trong những cách hiệu quả nhất để kiểm tra các giao diện thích ứng là tận dụng các mục tiêu xây dựng trên máy tính để bàn.
- Khi chạy trên máy tính để bàn, chúng ta 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.



II.2.7. Solve touch first

- 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 hướng cảm ứng tuyệt vời. Chúng ta 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,vì vậy nên thường xuyên chuyển sang thiết bị di động để xác minh rằng mọi thứ đều ổn.
- Sau khi đã đánh bóng giao diện cảm ứng, chúng ta 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 mình.

II.3. Đầu vào

- Điều chỉnh giao diện ứng dụng là chưa đủ, ta còn phải hỗ trợ các đầu vào khác nhau của người dùng. Chuột và bàn phím giới thiệu các kiểu nhập ngoài những kiểu 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.

II.3.1. Con lăn

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

```
return Listener(
 onPointerSignal: (event) {
   if (event is PointerScrollEvent) print(event.scrollDelta.dy);
 },
 child: ...
);
```

II.3.2. 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 chúng ta 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 ta có tiện ích con của riêng mình mà bạn muốn đưa vào trình duyệt, a có thể sử dụng tiện ích FocusableActionDetector để tạo các điều khiển của riêng mình. 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.

```
class _BasicActionDetectorState extends State<BasicActionDetector> {
 bool _hasFocus = false;
 @override
 Widget build(BuildContext context) {
   return FocusableActionDetector(
     onFocusChange: (value) => setState(() => _hasFocus = value),
     actions: <Type, Action<Intent>>{
       ActivateIntent: CallbackAction<Intent>(onInvoke: (Intent intent) {
         print("Enter or Space was pressed!");
       }),
     },
     child: Stack(
       clipBehavior: Clip.none,
       children: [
         FlutterLogo(size: 100),
         // Position focus in the negative margin
         if (_hasFocus) Positioned(left: -4, top: -4, bottom: -4, right: -4, child: _RoundedBorder())
       ],
     ),
   );
 }
}
```

(1) Kiểm soát thứ tự duyệt

- Để kiểm soát nhiều hơn thứ tự các widget được tập trung vào khi người dùng nhấn tab, ta có thể sử dụng FocusTraversalGroup để xác định các phần của cây sẽ được coi là một nhóm khi lập tab.
- Ví dụ: ta có thể chuyển qua tất cả các trường trong biểu mẫu trước khi chuyển đến nút gửi:

```
return Column(children: [
    FocusTraversalGroup(
      child: MyFormWithMultipleColumnsAndRows();
    ),
    SubmitButton(),
])
```

- 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, 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.

II.3.3. Keyboard accelerators

- 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.
- Trình tăng tốc bàn phím có thể được thực hiện theo một số cách trong Flutter tùy thuộc vào mục tiêu khác nhau.
- Nếu ta có một tiện ích con như Trường văn bản hoặc Nút đã có nút tiêu điểm, ta có thể bọc nó trong RawKeyboardListener và lắng nghe các sự kiện bàn phím:

```
return Focus(
  onKey: (FocusNode node, RawKeyEvent event) {
    if (event is RawKeyDownEvent) {
      print(event.logicalKey);
    }
    return KeyEventResult.ignored;
  },
  child: const TextField(),
);
```

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

```
// Define a class for each type of shortcut action you want
class CreateNewItemIntent extends Intent {
  const CreateNewItemIntent();
}

Widget build(BuildContext context) {
  return Shortcuts(
    // Bind intents to key combinations
    shortcuts: <ShortcutActivator, Intent>{
      SingleActivator(LogicalKeyboardKey.keyN, control: true): CreateNewItemIntent(),
    },
    child: Actions(
      // Bind intents to an actual method in your code
      actions: <Type, Action<Intent>>{
        CreateNewItemIntent: CallbackAction<CreateNewItemIntent>(
            onInvoke: (CreateNewItemIntent intent) => _createNewItem()),
      },
      // Your sub-tree must be wrapped in a focusNode, so it can take focus.
      child: Focus(
        autofocus: true,
        child:  ...,
      ),
    ),
  );
}
```

- Phím tắt widget 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:

```
void initState() {
  super.initState();
  RawKeyboard.instance.addListener(_handleKey);
}

@override
void dispose() {
  RawKeyboard.instance.removeListener(_handleKey);
  super.dispose();
}
```

- Để kiểm tra các tổ hợp phím với trình nghe chung, ta 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:

```
static bool isKeyDown(Set<LogicalKeyboardKey> keys) {
  return keys.intersection(RawKeyboard.instance.keysPressed).isNotEmpty;
}
```

- Kết hợp hai điều này với nhau, ta có thể kích hoạt một hành động khi nhấn Shift + N:

```
void _handleKey(event){
  if (event is RawKeyDownEvent) {
    bool isShiftDown = isKeyDown({
      LogicalKeyboardKey.shiftLeft,
      LogicalKeyboardKey.shiftRight,
    });
    if (isShiftDown && event.logicalKey == LogicalKeyboardKey.keyN) {
      _createNewItem();
    }
  }
}
```

- Một lưu ý thận trọng khi sử dụng trình nghe tĩnh là ta 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 ta để quản lý. Điều này có thể đặc biệt quan trọng khi 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.

II.3.4. Chuột vào, 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.
- Bộ Thành phần Vật liệu có hỗ trợ tích hợp cho nút tiêu chuẩn và con trỏ văn bản . Để thay đổi con trỏ từ trong các widget, hãy sử dụng MouseRegion:

```
return MouseRegion(
  cursor: SystemMouseCursors.grab,
  child: MyDraggableWidget(),
)
```

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

```
return MouseRegion(
  onEnter: (_) => setState(() => _isMouseOver = true),
  onExit: (_) => setState(() => _isMouseOver = false),
  onHover: (PointerHoverEvent e) => print(e.localPosition),
  child: ...,
);
```

II.4. Idioms and norms

- Khu vực cuối cùng cần xem xét đối với các ứng dụng tahích 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:

- Reduce cognitive load - 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.
- Build trust - 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.

II.4.1. Xem xét sự mong đợi trên từng 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 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 ta mà không có bất kỳ thỏa hiệp nào.
- Điều này có thể khó khăn nếu ta không phải là người dùng thường xuyên của nền tảng. Ta 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 nhưng rõ ràng là rất rõ ràng đối với một người dùng có kinh nghiệm.

(1) Tìm 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 mình 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 được 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 ta không sử dụng nền tảng thường xuyên.

(2) Duy trì sự độc đáo

- Tuân theo các hành vi mong đợi không có nghĩa là ứng dụng của ta 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.
- Ta 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.

II.4.2. Các thành ngữ và tiêu chuẩn phổ biến cần xem xét

(1) 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 thanh cuộn nhỏ hơn chỉ xuất hiện trong khi cuộn, trong khi người dùng máy tính để bàn thường mong đợi thanh cuộn lớn hơn, có mặt ở khắp nơi mà họ có thể nhấp hoặc kéo.

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

```
return Scrollbar(
  controller: controller,
  isAlwaysShown: DeviceType.isDesktop,
  child: ListView(controller: controller, ...),
);
```

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

(2) Lựa chọn nhiều

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

```
static bool get isSpanSelectModifierDown
  => isKeyDown([LogicalKeyboardKey.shiftLeft, LogicalKeyboardKey.shiftRight]);
```

- Để thực hiện kiểm tra nhận biết nền tảng đối với quyền điều khiển hoặc lệnh, ta có thể như sau:

```
static bool get isMultiSelectModifierDown {
  bool isDown = false;
  if (DeviceOS.isMacOS) {
    isDown = isKeyDown([LogicalKeyboardKey.metaLeft, LogicalKeyboardKey.metaRight]);
  } else {
    isDown = isKeyDown([LogicalKeyboardKey.controlLeft, LogicalKeyboardKey.controlRight]);
  }
  return isDown;
}
```

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

(a) Cảm ứng thiết bị

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

(3) 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:

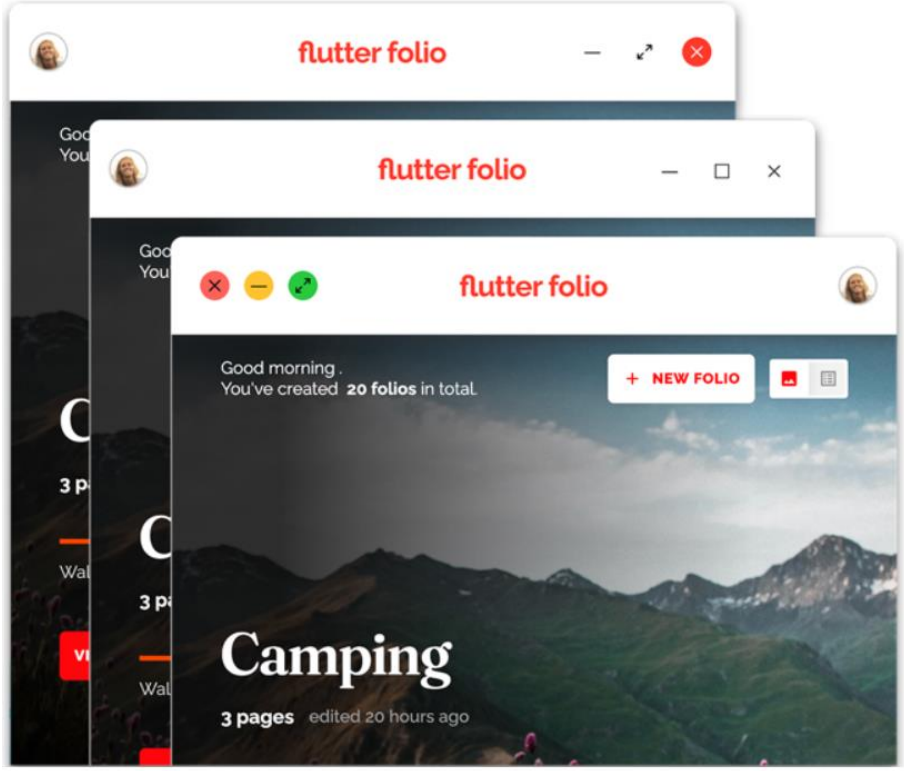
```
return SelectableText('Select me!');
```

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

```
return SelectableText.rich(TextSpan(children: [
  TextSpan(text: 'Hello'),
  TextSpan(text: 'Bold', style: TextStyle(fontWeight: FontWeight.bold)),
]));
```

(4) 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.



- Điều này không được hỗ trợ trực tiếp trong Flutter, nhưng ta có thể sử dụng gói bits_dojo để tắt các thanh tiêu đề gốc và thay thế chúng bằng thanh tiêu đề của riêng mình.
- Gói này cho phép thêm bất kỳ widget nào ta muốn vào TitleBar vì nó sử dụng các widget Flutter thuần túy. Điều này giúp dễ dàng điều chỉnh thanh tiêu đề khi ta điều hướng đến các phần khác nhau của ứng dụng.

(5) Menu ngữ cảnh và công cụ chú cảnh

- 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 cách 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 200-400 mili giây qua 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 (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:

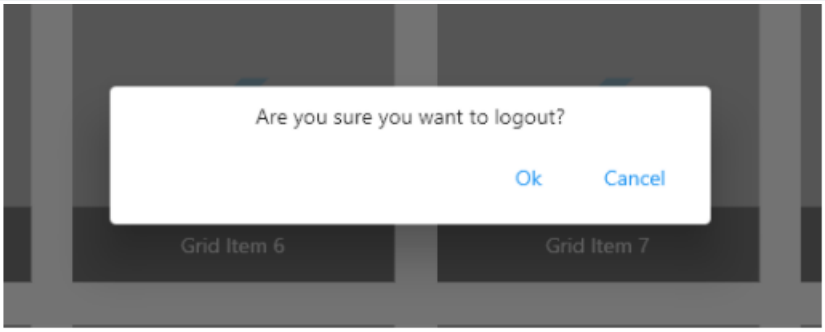
```
return const Tooltip(
  message: 'I am a Tooltip',
  child: Text('Hover over the text to show a tooltip.'),
);
```

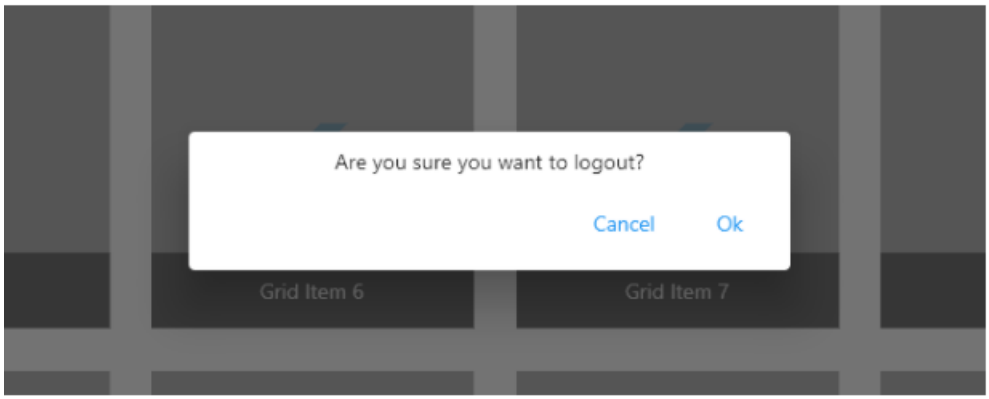
- Flutter cũng cung cấp các menu ngữ cảnh tích hợp sẵn khi chỉnh sửa hoặc chọn văn bản.
- Để hiển thị các chú giải công cụ nâng cao hơn, bảng bật lên hoặc tạo menu ngữ cảnh tùy chỉnh, ta sử dụng một trong các gói có sẵn hoặc tự tạo bằng cách sử dụng Ngăn xếp hoặc Lớp phủ.
- Một số gói có sẵn bao gồm:
  - context_menus
  - anchored_popups
  - flutter_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 chuột người dùng. 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.

(6) 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:

```
TextDirection btnDirection = DeviceType.isWindows ? TextDirection.rtl : TextDirection.ltr;
return Row(
  children: [
    Spacer(),
    Row(
      textDirection: btnDirection,
      children: [
        DialogButton(label: 'Cancel', onPressed: () => Navigator.pop(context, false)),
        DialogButton(label: 'Ok', onPressed: () => Navigator.pop(context, true)),
      ],
    ),
  ],
);
```





(7) 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, ta 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, ta không thể kết hợp thanh tiêu đề tùy chỉnh với thanh menu. Khi ta tạo thanh tiêu đề tùy chỉnh, tức ta đang thay thế hoàn toàn thanh tiêu đề gốc, điều đó có nghĩa là ta cũng mất thanh menu gốc tích hợp.
- Nếu ta cần cả thanh tiêu đề tùy chỉnh và thanh menu, ta có thể đạt được điều đó bằng cách triển khai nó trong Flutter, tương tự như menu ngữ cảnh tùy chỉnh.

(8) 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.
- 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 nhìn vào macOS Finder hoặc Windows Explorer, ta sẽ thấy rằng chúng hoạt động theo cách này: ta chỉ cần chọn một mục và bắt đầu kéo.
- Trong Flutter, ta 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.

II.4.3. 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 chúng ta có thể xem xét. Ta 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 ta 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 mọi người bắt đầu:
    - Material guidelines on responsive UI layout

- Material design for large screens
- Build high quality apps (Android)
- UI design do's and don'ts (Apple)
- Human interface guidelines (Apple)
- Responsive design techniques (Microsoft)
- Machine sizes and breakpoints (Microsoft)

III. Hiểu các ràng buộc

III.1. Hạn chế

- 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 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 cũng phụ thuộc vào chính cha 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 child 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 child có thể bị bỏ qua. Hãy cụ thể khi xác định sự liên kết.

III.2. So sánh ràng buộc(chặt chẽ >< lỏng lẻo)

- Rất phổ biến khi nghe nói rằng một số ràng buộc là "chặt chẽ" hoặc "lỏng lẻo", vì vậy ta 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 ta truy cập tệp box.dart của Flutter và tìm kiếm các hàm tạo BoxConstraints, ta sẽ tìm thấy những thứ sau:

```
BoxConstraints.tight(Size size)
    : minWidth = size.width,
      maxWidth = size.width,
      minHeight = size.height,
      maxHeight = size.height;
```

- Màn hình buộc Vùng chứa 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 Vùng chứa.
- 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:
- Trung tâm để cho Vùng chứa màu đỏ nhỏ hơn, nhưng không lớn hơn màn hình. Tất nhiên, Trung tâm thực hiện điều đó bằng cách chuyển các ràng buộc lỏng lẻo đến Vùng chứa. Cuối cùng, mục đích chính của Trung tâm 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 (Vùng chứa).

III.3. Tìm hiểu các quy tắc bố cục cho các vật dụng cụ thể

- Biết quy tắc bố cục chung là cần thiết nhưng vẫn chưa đủ.
- Mỗi widget 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 widget con.

- Nếu ta cố gắng đoán, có thể sẽ đoán sai. Ta không thể biết chính xác cách một widget con hoạt động trừ khi ta 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 ta chỉ cần đọc tài liệu. Tuy nhiên, nếu quyết định nghiên cứu mã nguồn bố cục, ta có thể dễ dàng tìm thấy nó bằng cách sử dụng các khả năng điều hướng của IDE.

IV. Giải quyết các ràng buộc 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 ta 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) khác nhau 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 widget con Ứng dụng, được chứa bởi lớp RenderView: hộp được sử dụng bởi con được hàm xây dựng của ứng dụng trả về có 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 single child, vượt qua hạn chế của chúng cho con cái của họ. Điều này có nghĩa là nếu ta lồng một loạt các hộp vào bên trong nhau ở gốc của cây kết xuất của ứng dụng, tất cả chúng sẽ hoàn toàn phù hợp 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.

IV.1. 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 hiển thị tự tìm thấy 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 ta 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.

IV.2. Uống 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 ta làm vậy, bạn sẽ nhận được một thông báo ngoại lệ hướng dẫn bạn đến tài liệu này.
- Theo hướng chéo, ví dụ: theo chiều rộng cho Cột (uốn dọc) hoặc chiều cao cho Hàng (uốn ngang), chúng không bao giờ được không bị ràng buộc, nếu không chúng sẽ không thể căn chỉnh hợp lý các con của chúng.