



ĐẠI HỌC BÁCH KHOA HÀ NỘI  
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

# Chương 5

## Nguyên lý thiết kế giao diện

# Mục lục

1. Quan hệ giữa UI và UX
2. Hướng dẫn thiết kế giao diện cho di động
3. Quy trình xây dựng giao diện
4. Xây dựng giao diện người dùng trong Flutter
5. Xây dựng giao diện người dùng trong ReactNative

# Mục lục

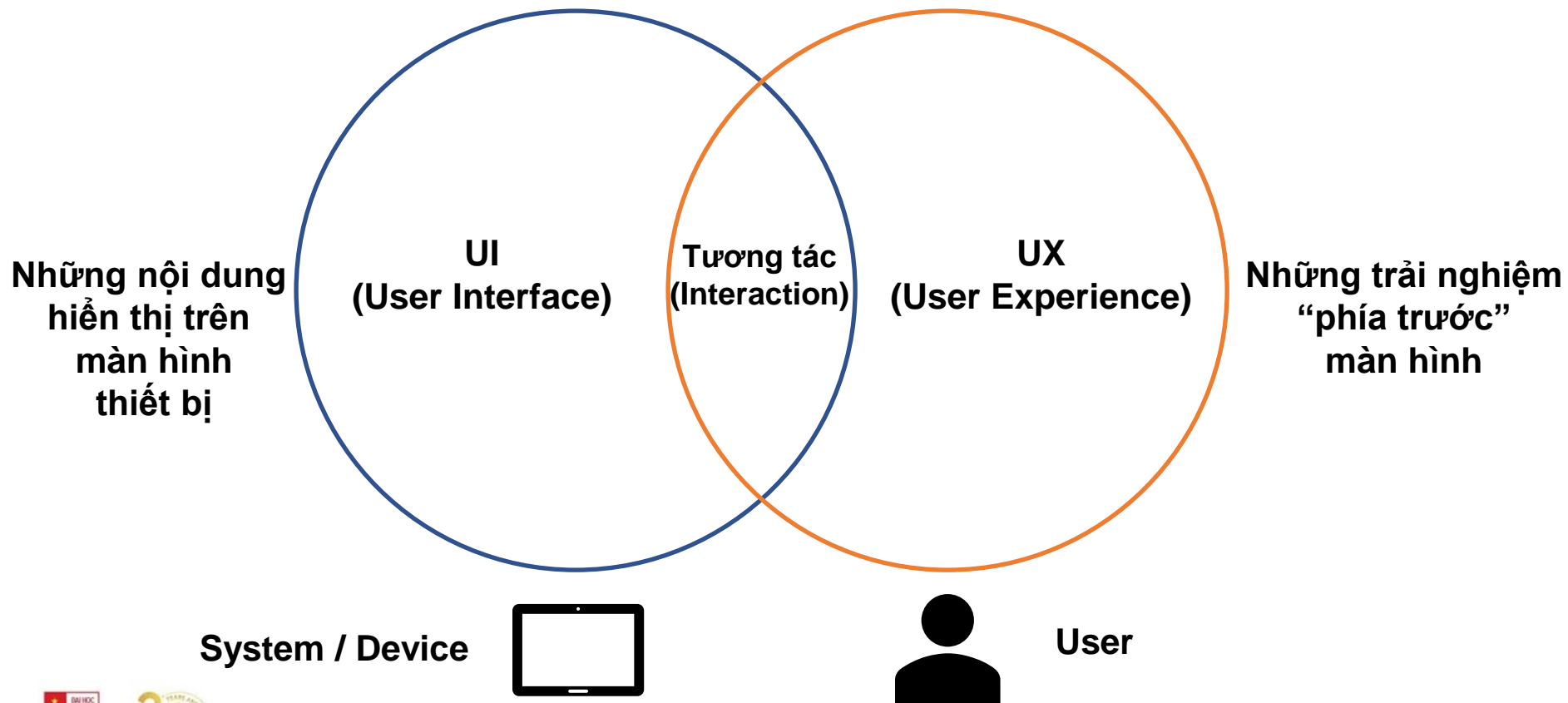
1. **Quan hệ giữa UI và UX**
2. Hướng dẫn thiết kế giao diện cho di động
3. Quy trình xây dựng giao diện
4. Xây dựng giao diện người dùng trong Flutter
5. Xây dựng giao diện người dùng trong ReactNative

# 1.1 Quan hệ giữa UI và UX

- **Giao diện người dùng - UI (User Interface)** bao gồm tất cả những gì mà người dùng có thể nhìn thấy trên màn hình thiết bị, ví dụ như bố cục, màu sắc, font chữ, hình ảnh,...
  - Thiết kế UI là yếu tố quan trọng giúp truyền tải thông điệp từ người thiết kế, nhà cung cấp sản phẩm, dịch vụ đến người dùng.
- **Trải nghiệm người dùng - UX (User Experience)** nói đến tổng thể trải nghiệm của người dùng, tức là bao gồm nhiều yếu tố khác bên cạnh giao diện như kinh nghiệm, cảm xúc, giá trị nhận được khi tương tác với sản phẩm, dịch vụ.

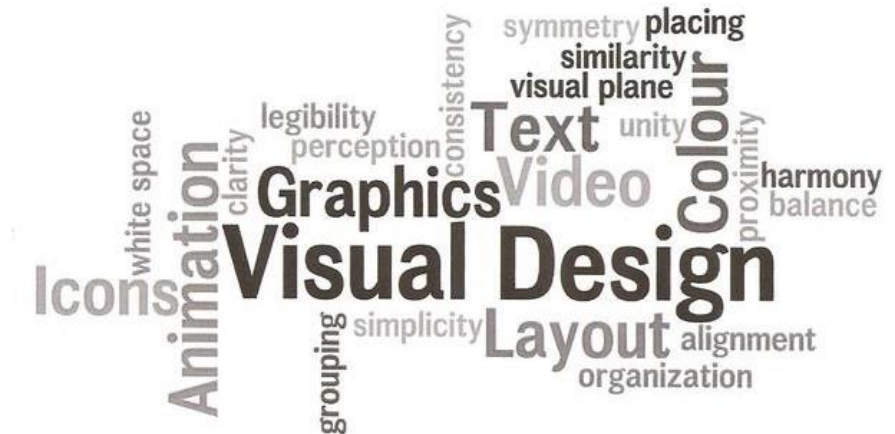
# 1.1 Quan hệ giữa UI và UX (2)

- Quan hệ giữa UI và UX



# 1.1 Quan hệ giữa UI và UX (3)

- **Thiết kế UI** đề cập đến cách nội dung được trình bày trực quan
- Các yếu tố thiết kế trực quan bao gồm:
  - Bố cục (layout)
  - Màu sắc và sự tương phản
  - Hình ảnh và biểu tượng
  - Kiểu chữ
  - Từ vựng và thuật ngữ
  - ...

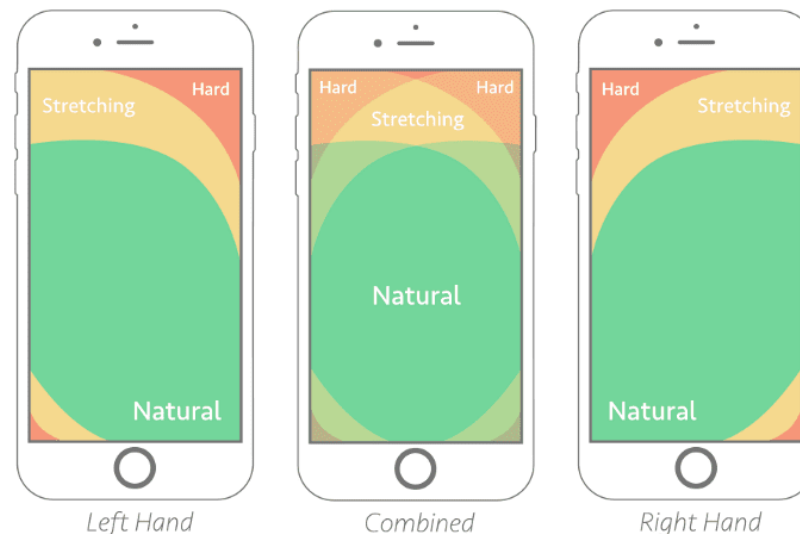


## 1.1 Quan hệ giữa UI và UX (4)

- **Thiết kế tương tác (Interaction Design - IxD)** đặt trọng tâm cho người thiết kế vượt ra ngoài sản phẩm đang được phát triển, bao gồm cách người dùng sẽ tương tác với sản phẩm.
- Thiết kế lấy người dùng làm trung tâm
  - Xem xét kỹ lưỡng nhu cầu, giới hạn và bối cảnh của người dùng, v.v. cho phép tùy chỉnh đầu ra phù hợp với nhu cầu chính xác.
- IxD liên quan đến năm chiều: từ ngữ (1D), biểu diễn hình ảnh (2D), đối tượng vật lý / không gian (3D), thời gian (4D) và hành vi (5D).

# 1.1 Quan hệ giữa UI và UX (5)

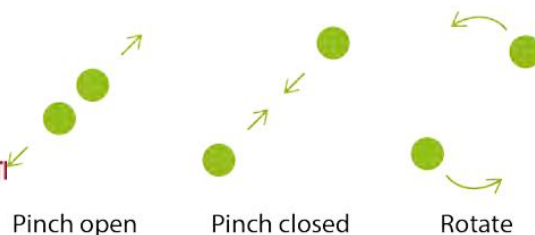
- Các yếu tố thiết kế tương tác trong ứng dụng di động:
  - Cử chỉ của người dùng (Gestures)
  - Thao tác nhập dữ liệu (Data Entry)
  - Điều hướng giữa các màn hình của ứng dụng (Navigation)
  - ...





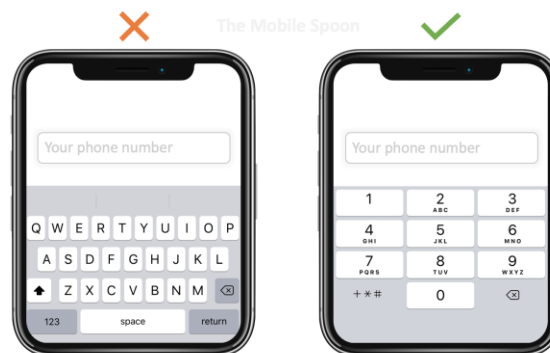
# 1.1 Quan hệ giữa UI và UX (6)

- Cử chỉ của người dùng (Gestures)
  - Màn hình cảm ứng bao phủ gần như toàn bộ mặt trước của thiết bị di động, không còn không gian cho các nút vật lý.
  - Các thiết bị màn hình cảm ứng chủ yếu dựa vào điều khiển cử chỉ, sử dụng bàn tay con người để tương tác với nội dung trên màn hình.
  - Các cử chỉ tiêu chuẩn trên màn hình cảm ứng:



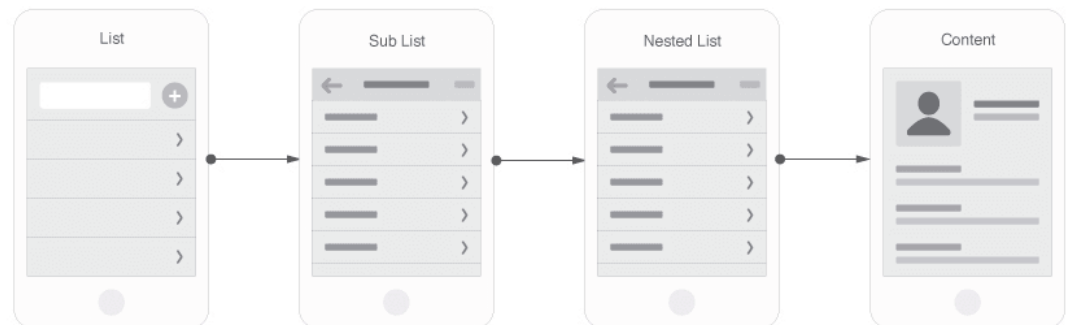
# 1.1 Quan hệ giữa UI và UX (7)

- Thao tác nhập dữ liệu (Data Entry)
  - Nhập dữ liệu là một hình thức tương tác thiết yếu.
  - Tính hợp lệ của dữ liệu đã nhập có thể được đảm bảo bằng cách giảm bớt các đầu vào không chính xác.
  - Trên các thiết bị có màn hình cảm ứng, nhập dữ liệu qua bàn phím ảo, kích thước phụ thuộc vào màn hình.
  - Các phương thức nhập yêu cầu bàn phím ảo nên được giữ ở mức tối thiểu, có thể dùng các phương thức nhập thay thế (ví dụ: bộ chọn ngày, trường thả xuống, nhóm nút radio, giá trị mặc định...



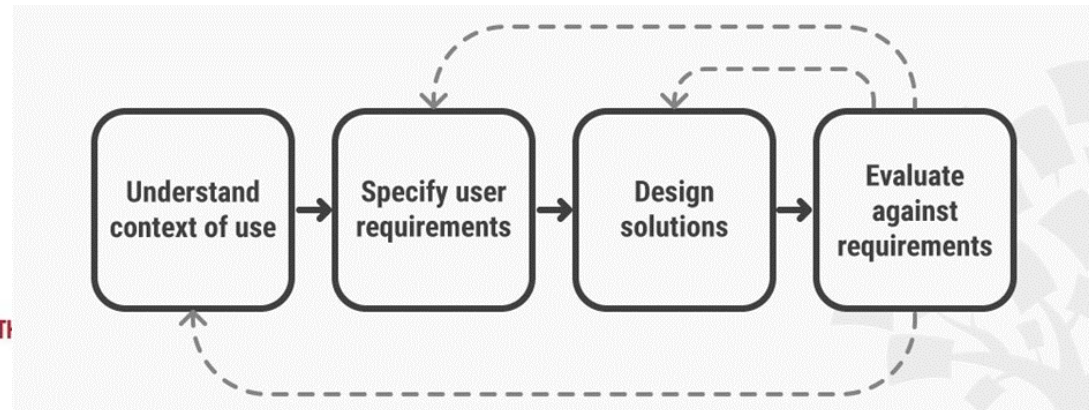
# 1.1 Quan hệ giữa UI và UX (8)

- Điều hướng giữa các màn hình (Navigation)
  - Hiển thị nhiều thông tin trong các màn hình thường được tổ chức theo cấu trúc phân cấp trong toàn bộ ứng dụng
  - Người dùng phải điều hướng qua các màn hình khác nhau
  - Thông tin quan trọng và chức năng chính nên được đặt cao hơn trong phân cấp điều hướng, cho phép người dùng tiếp cận nó nhanh hơn.
  - Khả năng quay lại màn hình trước đó (cử chỉ vuốt / nút quay lại trong ứng dụng)



# 1.1 Quan hệ giữa UI và UX (9)

- Thiết kế lấy người dùng làm trung tâm **User-centered design (UCD)**
  - Kết hợp các mối quan tâm và nhu cầu của người dùng ngay từ đầu của quá trình thiết kế. Nhu cầu của người dùng phải được xem xét trong tất cả các quyết định thiết kế.
  - Tập trung vào các mục tiêu khả năng sử dụng, đặc điểm người dùng, môi trường, nhiệm vụ và quy trình làm việc trong thiết kế giao diện.
  - Quá trình thiết kế lấy người dùng làm trung tâm là một quá trình lặp đi lặp lại.



## 1.2 Các thách thức của giao diện cho di động

- Kích thước màn hình nhỏ
- Tương tác qua màn hình cảm ứng
- Thiết bị luôn hoạt động (always on) & luôn kết nối (always connected)
- Cá nhân hoá trải nghiệm của người dùng
- Phần cứng đa dạng với nhiều loại cảm biến
- ...



# Mục lục

1. Quan hệ giữa UI và UX
2. **Hướng dẫn thiết kế giao diện cho di động**
3. Quy trình xây dựng giao diện
4. Xây dựng giao diện người dùng trong Flutter
5. Xây dựng giao diện người dùng trong ReactNative

## 2.1 Thiết kế giao diện cho di động

- **2.1.1 Thiết kế ưu tiên cho di động (Mobile first)**
  - Các hoạt động hàng ngày như chạy bộ tập thể dục, di chuyển trên các phương tiện công cộng như xe bus hoặc đi mua sắm siêu thị, v.v. đều sử dụng điện thoại di động trong công việc và giải trí.
  - Các nhà thiết kế có sự thay đổi ưu tiên khi xây dựng bố cục trước tiên cho các màn hình di động.
  - Tối ưu hóa tất cả các nội dung, tính năng và đặc biệt là nâng cao trải nghiệm là những mối quan tâm chính.
    - Responsiveness
    - Keep it simple
    - Finger-Friendly design
    - Feedback

## 2.1 Thiết kế giao diện cho di động (2)

- Responsiveness
  - Kích thước màn hình rất đa dạng và điều quan trọng là phải nhóm các loại thiết bị khác nhau dựa trên kích thước màn hình tương tự của chúng để quản lý.
  - Các thiết bị có thể được sử dụng ở chế độ màn hình ngang và dọc.
  - → khó khăn để xây dựng một giao diện người dùng có khả năng đáp ứng cho tất cả các kích thước của màn hình.
  - *Mục tiêu của thiết kế đáp ứng là làm cho việc hiển thị của ứng dụng hoặc website xuất hiện như thể nó được thiết kế riêng cho từng thiết bị và kích thước màn hình.*

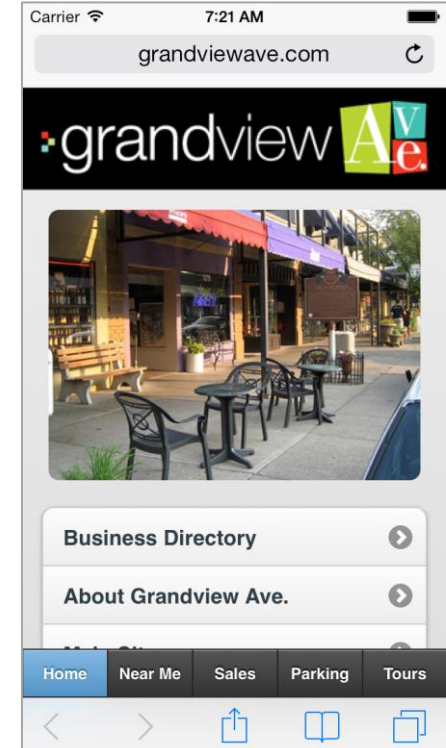
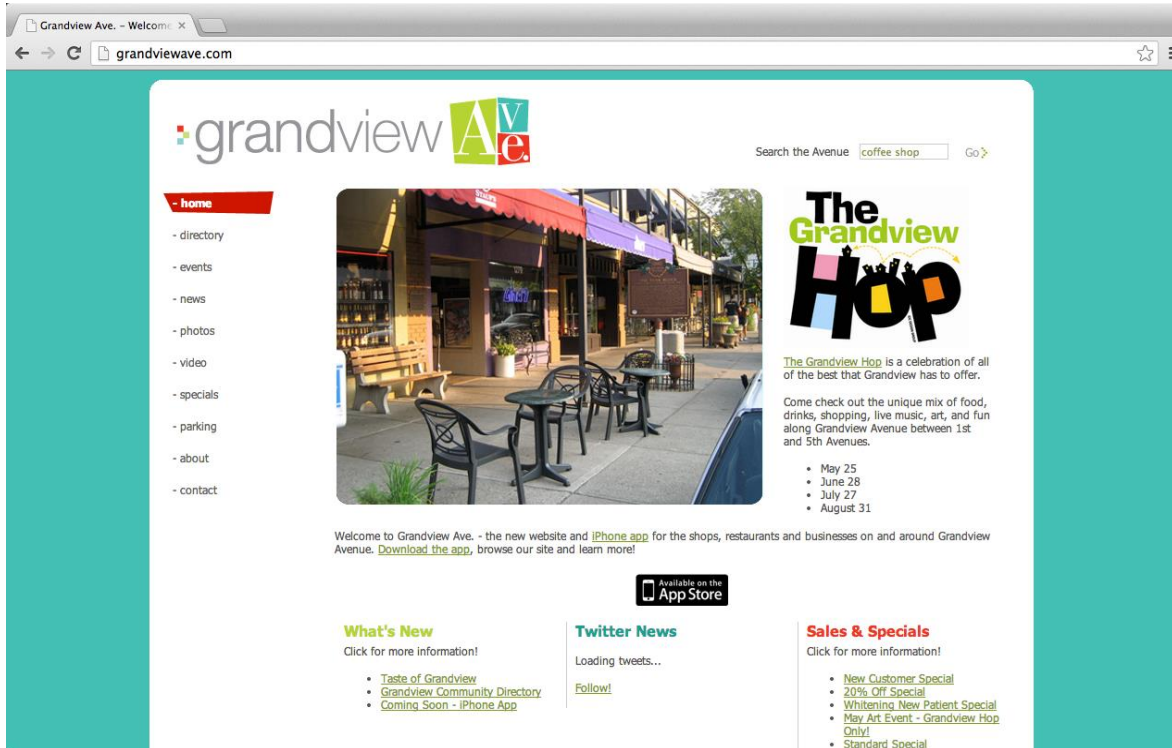


## 2.1 Thiết kế giao diện cho di động (3)

- Responsive layouts
  - Tự động điều chỉnh bố cục của để phù hợp với kích thước và hướng của thiết bị.
- Responsive widgets
  - Thiết kế giao diện người dùng di động thường được thể hiện bằng đơn vị kích thước vật lý. Các khuyến nghị tối thiểu cho kích thước mục tiêu cảm ứng thay đổi trong khoảng từ 7 mm đến 10 mm vuông, khoảng cách tối thiểu giữa các mục tiêu nên khoảng 1-2 mm
- Responsive content
  - Nội dung văn bản có thể được thu nhỏ bằng cách hiển thị có chọn lọc nội dung quan trọng nhất.
  - Hình ảnh cũng có thể được chia tỷ lệ đáp ứng cho các thiết bị lớn và nhỏ.

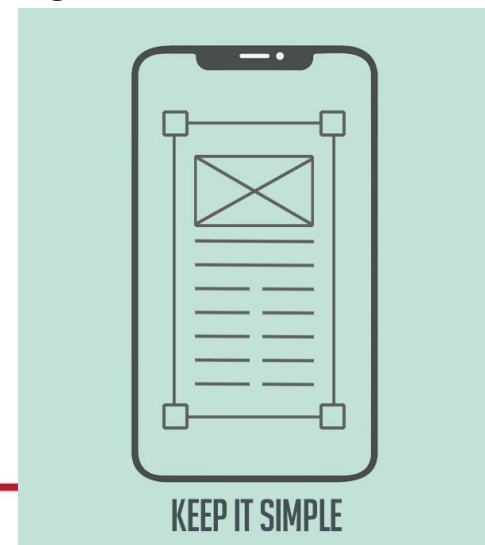
## 2.1 Thiết kế giao diện cho di động (4)

- Ví dụ: hãy so sánh sự khác biệt về hiển thị trên máy tính và trên di động



## 2.1 Thiết kế giao diện cho di động (5)

- Keep it simple
  - Sự đơn giản chính là chìa khóa để tăng trải nghiệm tốt hơn cho người dùng.
  - Kiểm tra đồ họa và hình ảnh để phù hợp hơn và đáng tin cậy hơn sẽ giúp giao diện người dùng hoạt động hiệu quả.
    - Người dùng có thể thực hiện các hành động của họ với số bước ít nhất có thể và giữ sự chú ý của họ trên ứng dụng lâu hơn.
- Loại bỏ lộn xộn giao diện người dùng
  - Không gian có hạn, ưu tiên thông tin quan trọng
  - Giảm thiểu các thanh cố định cho đầu trang, chân trang và điều hướng

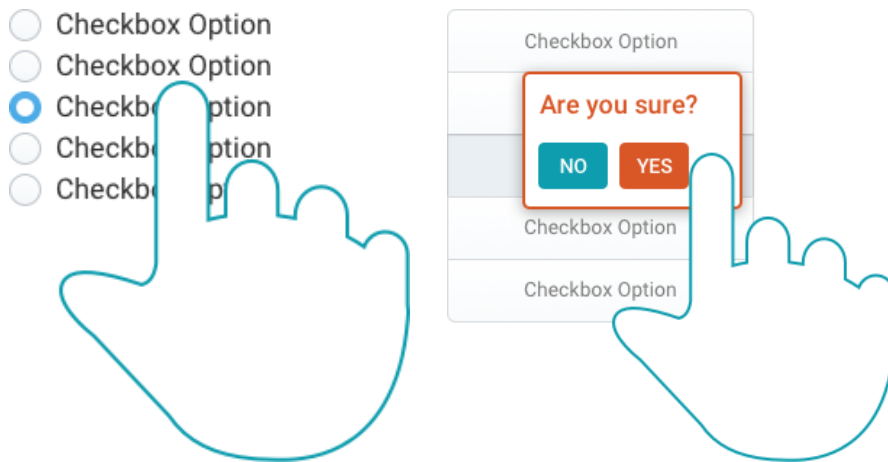


## 2.1 Thiết kế giao diện cho di động (6)

- Finger-Friendly design
  - Tương tác với màn hình cảm ứng qua các ngón tay: các mục tiêu cảm ứng nhỏ hơn → khó sử dụng và có nhiều khả năng xảy ra lỗi → thiết kế để tối ưu cho cảm ứng



Apple: the 44 pixel target



10px or more  
between targets

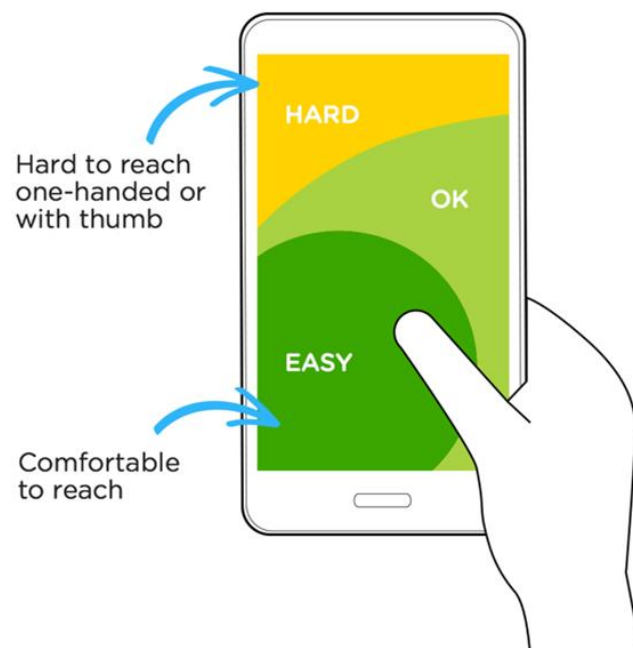
25px = có thể chạm vào

40px = tối ưu

Kiểm tra đầu vào cảm ứng trên thiết bị thực

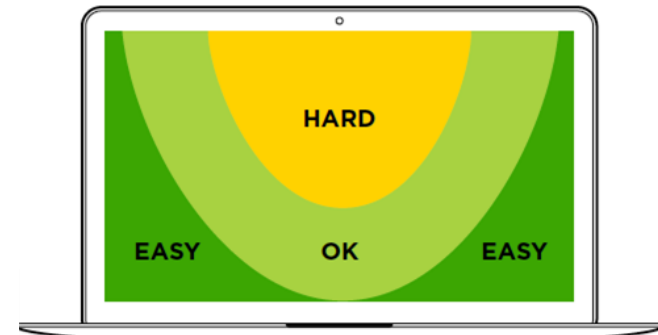
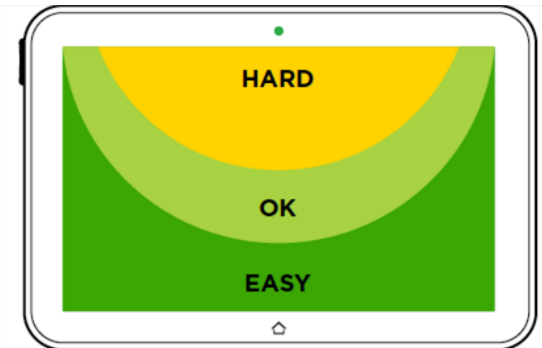
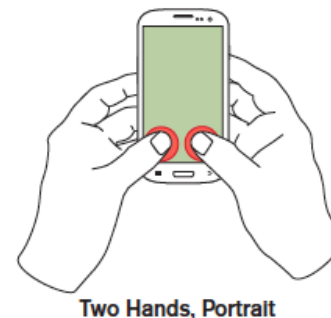
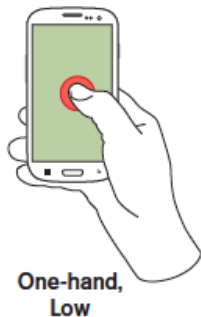
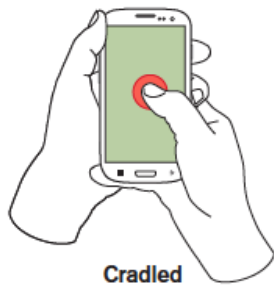
## 2.1 Thiết kế giao diện cho di động (7)

- Finger-Friendly design: Touch Ergonomics
  - Hầu hết các cử chỉ được thiết kế để sử dụng bằng một tay.
  - Các điều hướng chính được thiết kế để sử dụng ngón tay cái để hoàn thành tương tác với điện thoại đang được cầm trên tay.
  - Xem xét 'vùng ngón tay cái'



## 2.1 Thiết kế giao diện cho di động (8)

- Finger-Friendly design: Touch Ergonomics
  - Cầm thiết bị theo những cách khác nhau



## 2.1 Thiết kế giao diện cho di động (9)

- Feedback
  - Phản hồi của ứng dụng rất quan trọng trong quá trình tương tác với người dùng.
  - Đưa ra phản hồi cho người dùng không chỉ khi tải nội dung mà sau mỗi lần hoàn thành một nhiệm vụ cụ thể.
  - Kết hợp các loại phản hồi khác nhau như nháy đèn, rung hoặc âm thanh.

## 2.1 Thiết kế giao diện cho di động (10)

- **2.1.2 Phân tích hành vi người dùng (User behavior)**
  - Thiết kế dựa trên hành vi là tập hợp các kỹ thuật được sử dụng để thuyết phục.
  - Ý tưởng là sử dụng các mẫu hành vi để mang lại lợi ích cho một thiết kế nhất định, để nó cho phép người dùng tự do lựa chọn và quyền tự chủ trong hoạt động. Nó tận dụng các hành vi đã học, để hướng người dùng theo cách đúng đắn.
  - Làm cho người dùng cảm thấy bình tĩnh và có quyền kiểm soát, tin cậy.
  - Với thiết kế đáng tin cậy cho phép người dùng xây dựng thói quen trong khuôn khổ thiết kế sẽ thúc đẩy mức độ tương tác và giữ chân người dùng.



## 2.1 Thiết kế giao diện cho di động (11)

### ▪ 2.1.3 Phân tích bối cảnh (Context-aware)

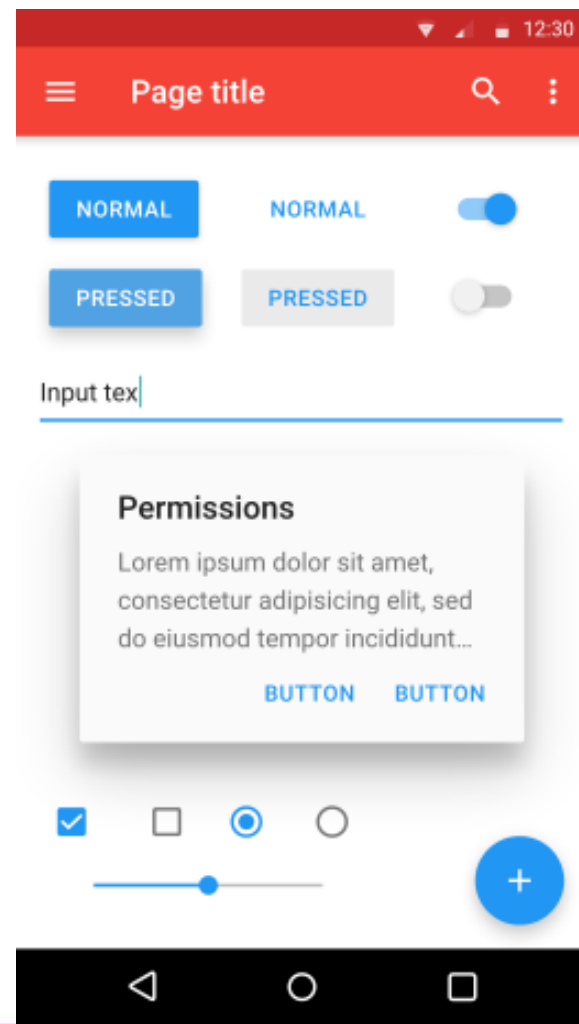
- Môi trường luôn thay đổi vì thực tế là người dùng di động sử dụng thiết bị di động của họ bất kỳ thời điểm nào và ở bất cứ đâu.
- Bối cảnh xã hội khác nhau khiến người dùng phân chia sự chú ý của họ giữa thế giới di động và thế giới thực tế.
- Các công nghệ tiên tiến như gia tốc kế, cảm biến hoặc nhận thức định vị được nhúng vào các thiết bị làm tăng khả năng thiết kế tương tác sáng tạo.
- Giải pháp khi thiết kế cho thiết bị di động là giữ cho nó đơn giản và cố gắng tập trung vào các chức năng quan trọng nhất.

## 2.2 Các hướng dẫn (Guidelines)

- Hướng dẫn giao diện người dung (**User interface guidelines**) là một tài liệu mô tả tất cả các quy tắc và yếu tố để thiết kế một giao diện.
- Chúng nhằm giúp các nhà thiết kế GUI tạo ra các giao diện hiệu quả và phù hợp với công thái học chung của nền tảng.
- Mục tiêu của các công ty đề xuất hướng dẫn (Google và Apple) là khuyến khích các nhà phát triển tạo ra các ứng dụng chất lượng tốt hơn.

## 2.2 Các hướng dẫn (Guidelines) (2)

- **Material Design** là một ngôn ngữ thiết kế được phát triển vào năm 2014 bởi Google.
- Material Design đem đến phong cách tự do hơn với các cách bố trí dạng lưới, các phản hồi hoạt họa chuyển động, kéo giãn, và các hiệu ứng chiều sâu như ánh sáng và đổ bóng.



## 2.2 Các hướng dẫn (Guidelines) (3)

- Ví dụ các nguyên tắc Material Design được cấu trúc xung quanh các yếu tố cơ bản về thiết kế:
  - **Bố cục (Layout):** Bố cục Material Design khuyến khích tính nhất quán trên các nền tảng, môi trường và kích thước màn hình bằng cách sử dụng các yếu tố và khoảng cách đồng nhất. Giao diện người dùng Android sử dụng bố cục trực quan và có thể đoán trước, với lưới, đường khóa và phân đệm nhất quán.
  - **Phong cách (Style):** Hệ thống màu Material Design có thể được sử dụng để tạo chủ đề màu phản ánh thương hiệu hoặc phong cách.
  - **Hoạt ảnh (Animation):** Chuyển động giúp tạo giao diện người dùng biểu cảm và dễ sử dụng.
  - **Thành phần (Components):** các tiện ích trong giao diện người dùng của ứng dụng.
  - **Khả năng sử dụng (Usability):** Thiết kế có thể truy cập cho phép người dùng thuộc mọi khả năng điều hướng, hiểu và sử dụng thành công giao diện người dùng, bố cục rõ ràng với các lời gọi hành động riêng biệt.

## 2.2 Các hướng dẫn (Guidelines) (4)

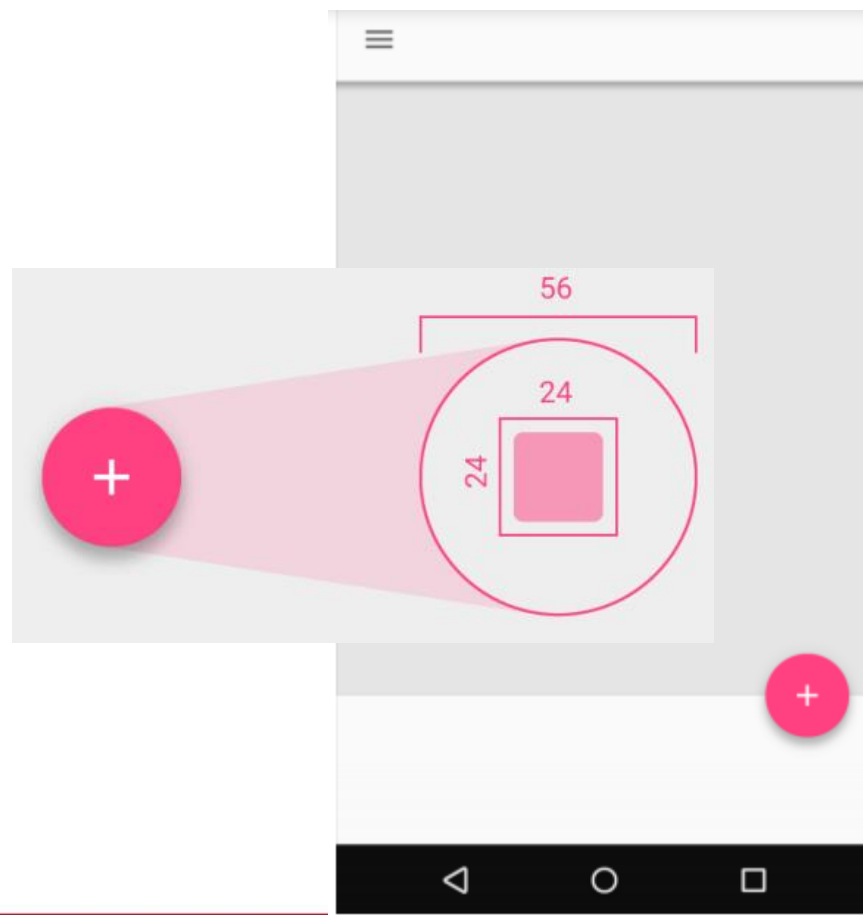
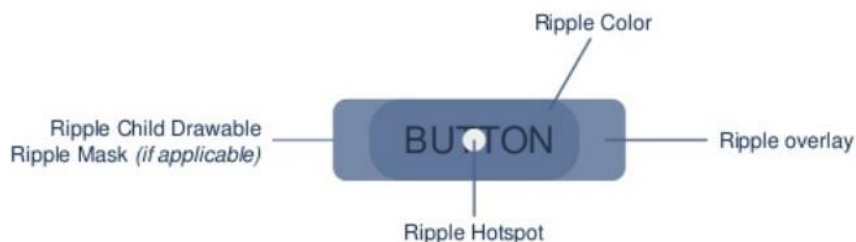
- Ví dụ các thành phần giao diện theo hướng dẫn Material Design

- 1. Action button :

- 1. Primary, positive action
- 2. Add, favorite, create etc.
- 3. Never destructive
- 4. Never multiple

- 2. Ripples:

- Provides visual feedback



## 2.2 Các hướng dẫn (Guidelines) (5)

- Nguyên tắc **Human Interface Guidelines - HIG**, là một bộ tài nguyên giao diện người dùng cũng như tài liệu triển khai mô tả chính xác cách tích hợp với các nền tảng của Apple.
  - Apple hướng dẫn giao diện người dùng được tạo ra để sử dụng trên nhiều sản phẩm phần cứng của họ bao gồm: Hệ điều hành Mac, iOS và iPadOS, watchOS, tvOs
- Các ứng dụng iOS được phát triển một cách chặt chẽ dựa trên 3 chủ đề (theme) cốt lõi của Apple.
  - Dựa trên những điều này, có những kỳ vọng cao về cả chức năng và chất lượng thiết kế được yêu cầu trước khi bất kỳ ứng dụng nào được đưa vào Apple App Store.

## 2.2 Các hướng dẫn (Guidelines) (6)

- Nguyên tắc **Human Interface Guidelines - HIG**, bao gồm:
  - Metaphors
  - Reflect the User's Mental Model
  - Explicit and Implied Actions
  - Direct Manipulation
  - User Control
  - Feedback and Communication
  - Consistency
  - WYSIWYG (What You See Is What You Get)
  - Forgiveness
  - Perceived Stability
  - Aesthetic Integrity
  - Modelessness

### IOS Human Interface Guidelines



## 2.3 Các mô hình điều hướng (navigation)

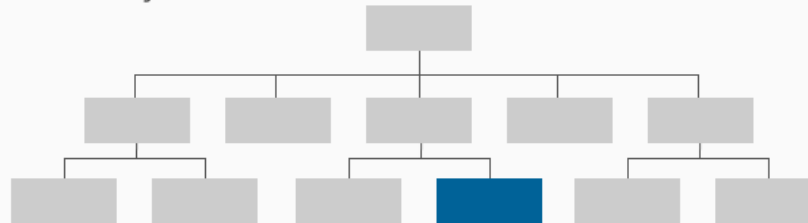
- Các ứng dụng di động bị giới hạn bởi kích thước của màn hình hiển thị nên hầu hết các ứng dụng sẽ sử dụng nhiều màn hình.
- Khái niệm chuyển đổi giữa các màn hình đó được gọi là **điều hướng (navigation)**
- Có hai loại cấu trúc điều hướng chính được sử dụng trong các ứng dụng di động
  - **Cấu trúc điều hướng phẳng:** Nội dung được nhóm hợp lý thành các danh mục, trong đó mỗi danh mục có các view cấp cao nhất của riêng nó.
  - **Cấu trúc điều hướng sâu:** Một lựa chọn trên mỗi màn hình phải được thực hiện để điều hướng đến cấp phân cấp sâu hơn tiếp theo, cho đến khi đạt đến một điểm cuối nhất định (các lá trong cây).



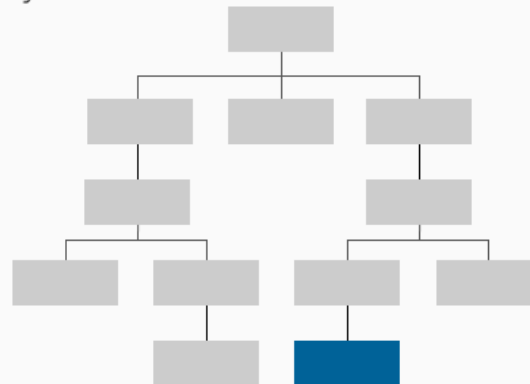
## 2.3 Các mô hình điều hướng (navigation) (2)

- Có hai loại cấu trúc điều hướng chính được sử dụng trong các ứng dụng di động

Flat hierarchy



Deep hierarchy

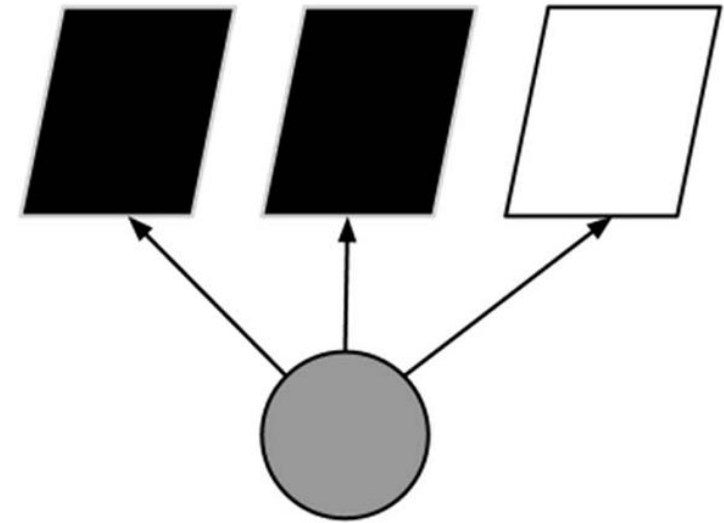


## 2.3 Các mô hình điều hướng (navigation) (3)

- Các hướng dẫn điều hướng hiệu quả:
  - Triển khai điều hướng theo cách hỗ trợ người dùng dễ dàng tiếp cận nội dung hoặc chức năng mong muốn
  - Tránh để cấu trúc điều hướng trở nên quá sâu
  - Luôn cho biết vị trí hiện tại của người dùng bên trong ứng dụng
  - Luôn trang bị các view cấp phụ bằng nút “Quay lại” / “Lên” để cho biết khả năng quay lại view cấp độ chính
- Điều hướng ứng dụng có thể thuộc một trong một số nhóm:
  - Null Navigation
  - Stack Navigation
  - Horizontal Navigation
  - Composite Navigation

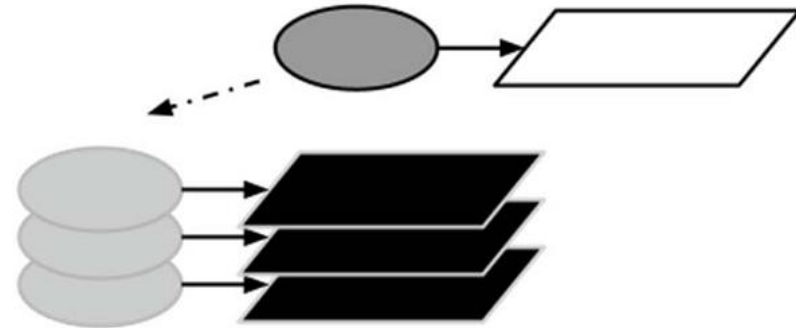
## 2.3 Các mô hình điều hướng (navigation) (4)

- Null Navigation
  - Trong cả Android và iOS, có thể có nhiều View được liên kết với một Controller. Controller chịu trách nhiệm chuyển các View khác nhau lên hàng đầu, hiển thị dữ liệu cơ bản và xử lý sự kiện cho mỗi View.
  - Mô hình Null Navigation: một Controller được liên kết với nhiều View và sẽ xác định rõ ràng cái nào sẽ hiển thị.



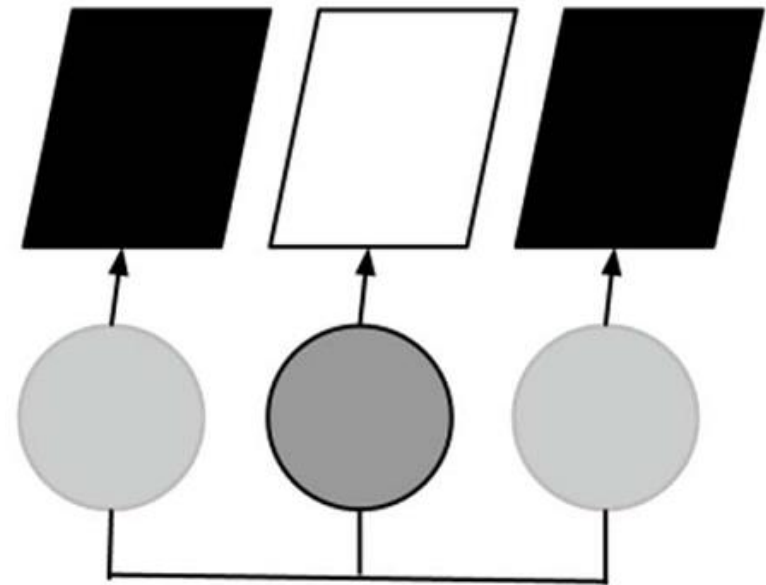
## 2.3 Các mô hình điều hướng (navigation) (5)

- Stack Navigation
  - Khái niệm về màn hình "xếp chồng"
  - Ứng dụng sẽ hoạt động như một bộ điều khiển thực hiện điều hướng bằng cách push các màn hình vào một ngăn xếp khi người dùng điều hướng sâu hơn vào ứng dụng.
  - Điều này tạo ra một thiết kế phân cấp cho phép điều hướng ngược lại, bằng cách pop màn hình trên cùng từ ngăn xếp.
  - Mô hình giúp phân tách các màn hình thành các thực thể chức năng một cách hợp lý.



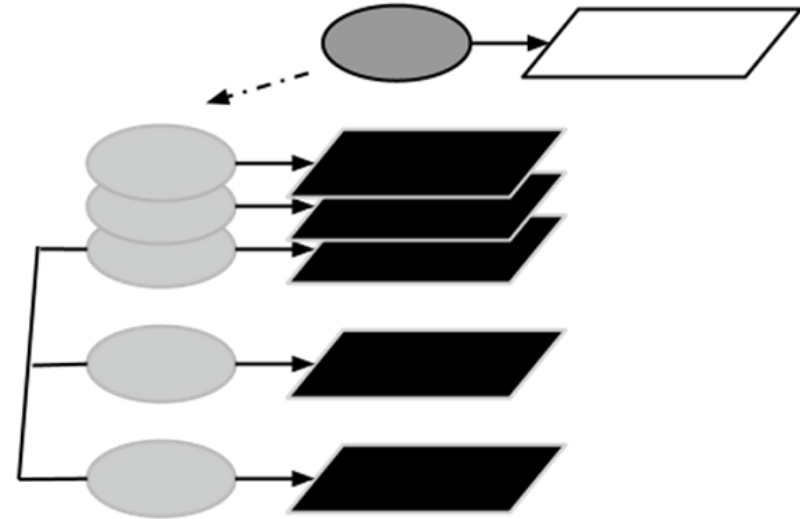
## 2.3 Các mô hình điều hướng (navigation) (6)

- Horizontal Navigation
  - Điều hướng ngang được thực hiện thông qua việc sử dụng các thanh tab.
  - Mỗi màn hình được hiển thị cùng cấp với các màn hình khác mà không cần tham chiếu đến thứ tự hiển thị.
  - Các màn hình khác nhau có thể được hiển thị bất kỳ lúc nào dựa trên lựa chọn tab.
  - Không có khả năng quay lại, mỗi màn hình có thể được coi là đang hoạt động, liên tục hiển thị thông tin của nó.



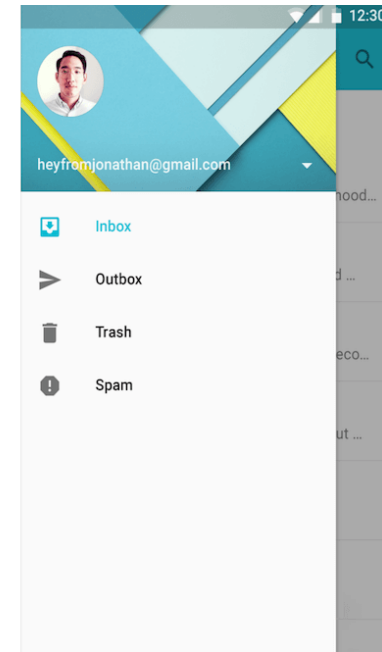
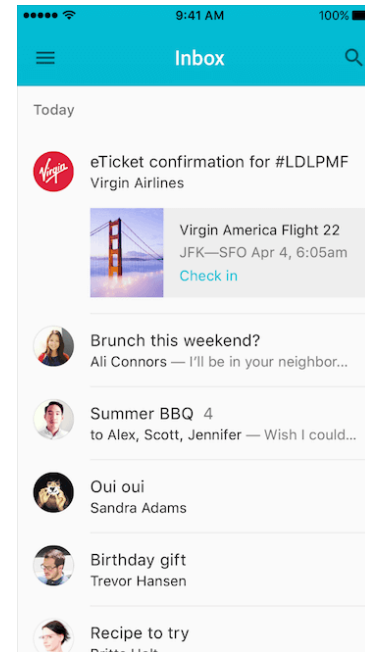
## 2.3 Các mô hình điều hướng (navigation) (7)

- Composite Navigation
  - Điều hướng hỗn hợp kết hợp Điều hướng ngang và Điều hướng ngăn xếp bằng cách xếp chồng lên đầu các tab hoặc bằng cách lồng
  - Giao diện ngang trên đầu Ngăn xếp.
  - Bằng cách kết hợp hai loại Điều hướng, các nhà phát triển có thể tạo các ứng dụng chạy tự nhiên, người dùng có khả năng điều hướng vào nhiều màn hình phân cấp.



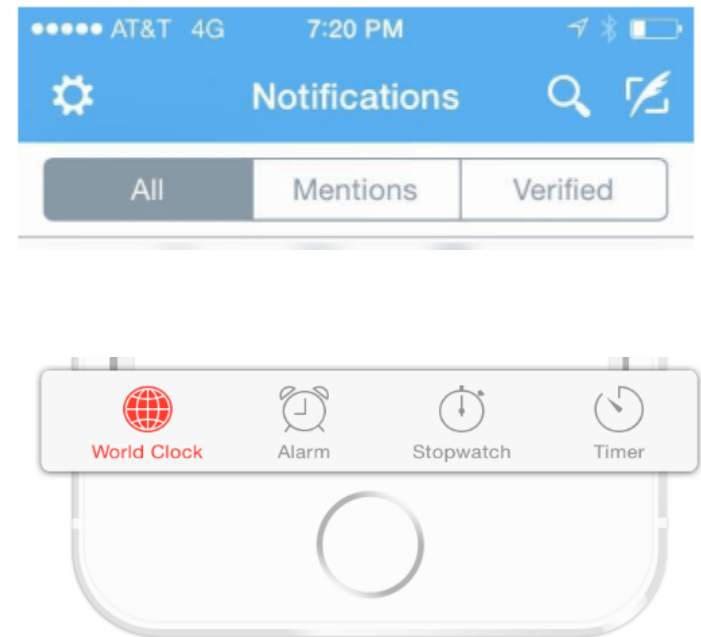
## 2.3 Các mô hình điều hướng (navigation) (8)

- **Ví dụ Side drawer:** một trong những mẫu điều hướng phổ biến nhất cho các ứng dụng di động
  - Ẩn điều hướng bên ngoài mép trái của màn hình và chỉ hiển thị khi người dùng nhấn vào
- **Ưu điểm**
  - Có thể chứa một số lượng khá lớn các tùy chọn trong một không gian nhỏ
  - Giao diện gọn gàng giúp người dùng tập trung vào nội dung chính
- **Hạn chế**
  - Khi điều hướng bị ẩn, người dùng sẽ ít sử dụng nó hơn



## 2.3 Các mô hình điều hướng (navigation) (9)

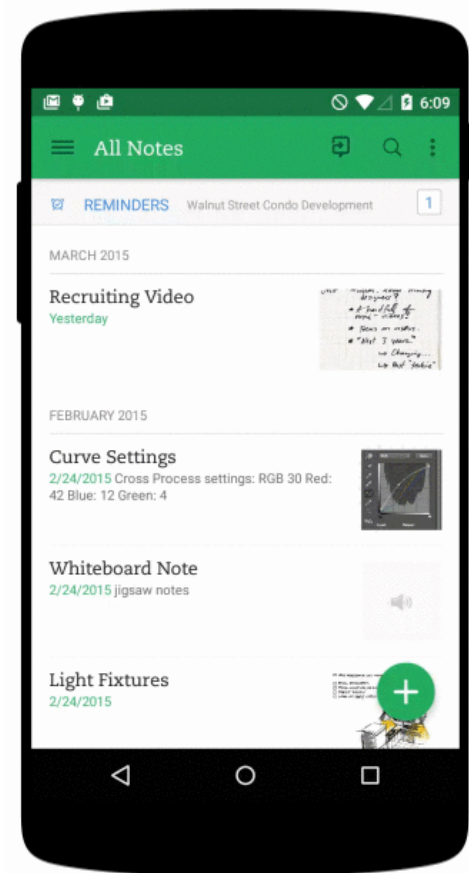
- **Ví dụ Tab Bar:** không ẩn điều hướng, cho phép truy cập trực tiếp
  - **Ưu điểm**
    - Trực quan ( sử dụng biểu tượng, nhãn và màu sắc) cho phép tương tác nhanh
    - Thanh điều hướng luôn trong tầm nhìn cho dù người dùng có thể chuyển trang
    - Thường đặt phía dưới, trong phạm vi ngón tay cái
  - **Hạn chế**
    - Số lượng các tùy chọn điều hướng bị hạn chế do phải giữ kích thước mục tiêu cảm ứng tối ưu.





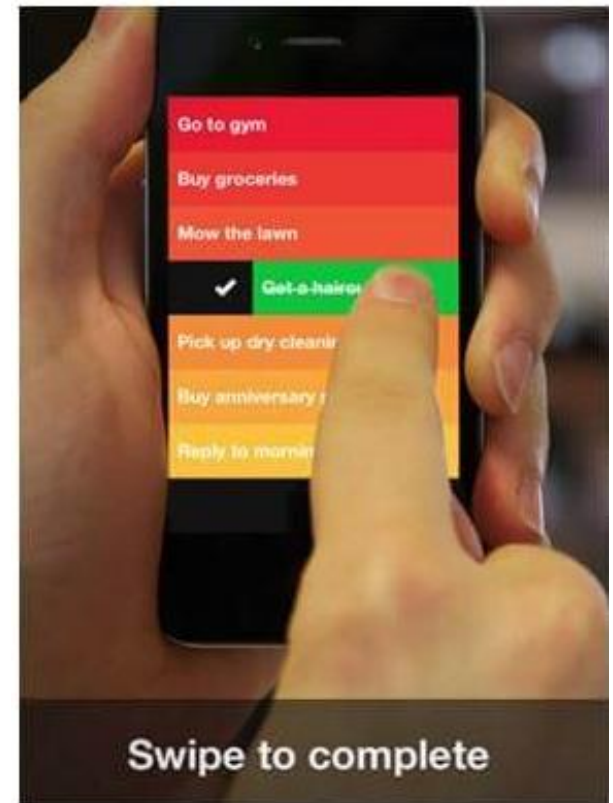
## 2.3 Các mô hình điều hướng (navigation) (10)

- **Ví dụ Floating Action Button:** nút tròn nổi trên giao diện người dùng, đặc trưng của Android
  - **Ưu điểm**
    - Chiếm ít không gian màn hình: nút được ưu tiên gắn cho hành động quan trọng nhất
    - Nâng cao trải nghiệm người dùng.
    - Có thể sử dụng nó hiệu quả hơn so với nút tác vụ truyền thống
  - **Hạn chế**
    - Có thể khiến người dùng mất tập trung khỏi nội dung
    - Có thể che mất nội dung phía dưới



## 2.3 Các mô hình điều hướng (navigation) (11)

- **Ví dụ Gesture-Based Navigation:** dựa trên cử chỉ để điều hướng phù hợp
  - **Ưu điểm**
    - Loại bỏ sự lộn xộn của giao diện người dùng
    - Giao diện người dùng tự nhiên hơn
    - Cử chỉ có thể là một đặc điểm nổi bật của một sản phẩm, ví dụ: Tinder
  - **Hạn chế**
    - Điều hướng là vô hình → đòi người dùng phải học tập.
    - Hầu hết các cử chỉ không tự nhiên và cũng không dễ học hoặc dễ nhớ.



## 2.4 Cử chỉ của người dùng (gestures)

- Các mẫu cử chỉ phổ biến trên nền tảng iOS và Android

**Tap**



Briefly touch surface with fingertip

**Double tap**



Rapidly touch surface twice with fingertip

**Drag**



Move fingertip over surface without losing contact

**Flick**



Quickly brush surface with fingertip

**Pinch**



Touch surface with two fingers and bring them closer together

**Spread**



Touch surface with two fingers and move them apart

**Press**



Touch surface for extended period of time

**Press and tap**



Press surface with one finger and briefly touch surface with second finger

**Press and drag**



Press surface with one finger and move second finger over surface without losing contact

**Rotate**



Touch surface with two fingers and move them in a clockwise or counterclockwise direction

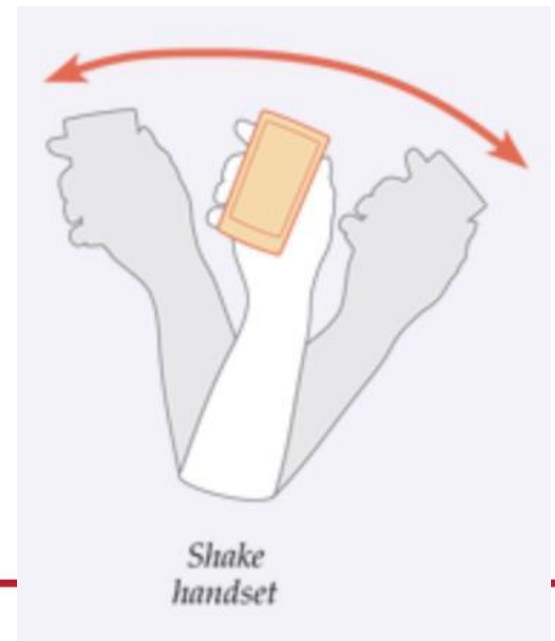
## 2.4 Cử chỉ của người dùng (gestures) (2)

- Sự khác biệt giữa các cử chỉ tiêu chuẩn của Andoid và iOS

| Gesture            | iOS   | Android   |
|--------------------|---|---|
| Touch / Tap        | Activates screen element  |   |
|                    |   | Canceles / escapes current task (dialog / menu) |
| Double touch / tap | Zooms in / Zooms out  |   |
| Long press         | Displays view for cursor positioning  |   |
|                    | Enters a mode allowing items to be rearranged   | Enters a mode allowing items to be selected     |
| Pinch              | Zooms in / Zooms out  |   |
|                    |   | Expands / Collapses content                     |
| Drag               | Moves element, scrolls / pans precisely   |   |
| Flick / Fling      | Scrolls / Pans quickly  |   |
| Swipe              | Reveals off-screen content, Reveals hidden elements, Switch between in-content views, Refreshes content |   |
|                    | Returns to previous view  |   |

## 2.4 Cử chỉ của người dùng (gestures) (3)

- Cử chỉ động học (Kinesthetic gestures):
  - Kinesthetic là khả năng phát hiện chuyển động của cơ thể
  - Liên quan đến việc thiết bị di động sử dụng cảm biến để phát hiện và phản ứng với sự tiếp xúc, hành động và định hướng cho phép màn hình phản ứng với tốc độ, chuyển động và hướng của thiết bị di động.
- Ví dụ: lắc thiết bị để thực hiện undo / redo hoặc chuyển bài hát trong chương trình nghe nhạc.



## 2.4 Cử chỉ của người dùng (gestures) (4)

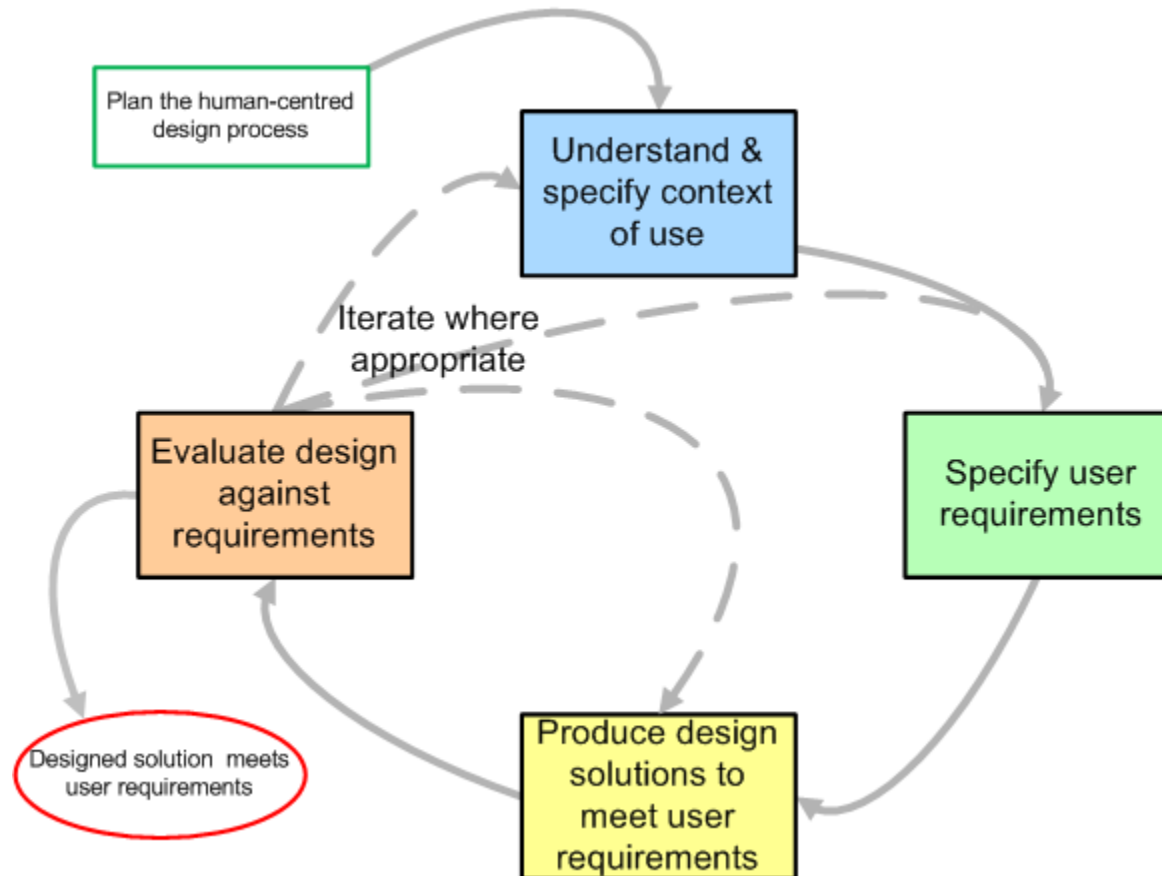
- Các hướng dẫn về cử chỉ trong thiết kế tương tác:
  - Tránh sử dụng các cử chỉ quá phức tạp, hãy dựa vào các cử chỉ tiêu chuẩn
  - Tránh sử dụng cử chỉ tiêu chuẩn cho các hành động không chuẩn
  - Cung cấp các phím tắt có thể khám phá để bổ sung các cử chỉ
  - Làm cho cử chỉ có thể đảo ngược

# Mục lục

1. Quan hệ giữa UI và UX
2. Hướng dẫn thiết kế giao diện cho di động
3. **Quy trình xây dựng giao diện**
4. Xây dựng giao diện người dùng trong Flutter
5. Xây dựng giao diện người dùng trong ReactNative

## 3.1 Quy trình xây dựng giao diện

- Quy trình thiết kế lặp theo ISO 9241 - 210



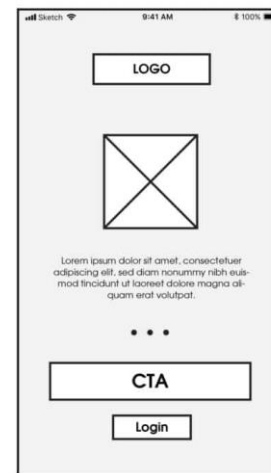
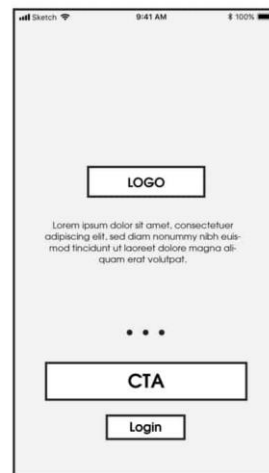
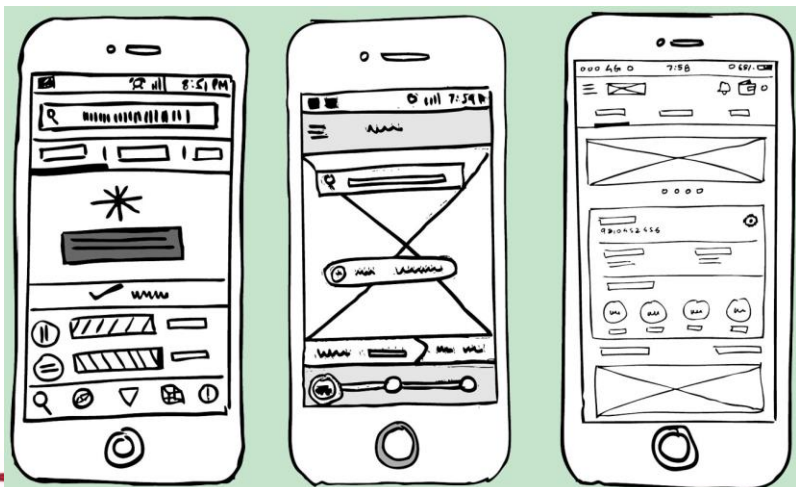


## 3.2 Các loại mẫu giao diện

- Tạo mẫu (**prototyping**) đóng vai trò quan trọng trong mọi quá trình thiết kế.
- Một nguyên mẫu giao diện người dùng là một giả thuyết - một giải pháp thiết kế ứng viên đang được xem xét cho một vấn đề thiết kế cụ thể.
- Có nhiều loại nguyên mẫu, ví dụ như sau:
  - Nguyên mẫu trang đơn bao gồm một trang duy nhất và đại diện cho một phần của thiết kế giao diện người dùng. Nguyên mẫu nhiều trang đại diện cho một mẫu có đủ menu, màn hình và mục tiêu nhấp chuột để cho phép người dùng hoàn thành một nhiệm vụ.
  - Nguyên mẫu thực tế và chi tiết so với nguyên mẫu phác thảo bằng tay trên giấy.
  - Nguyên mẫu tương tác so với nguyên mẫu tĩnh.

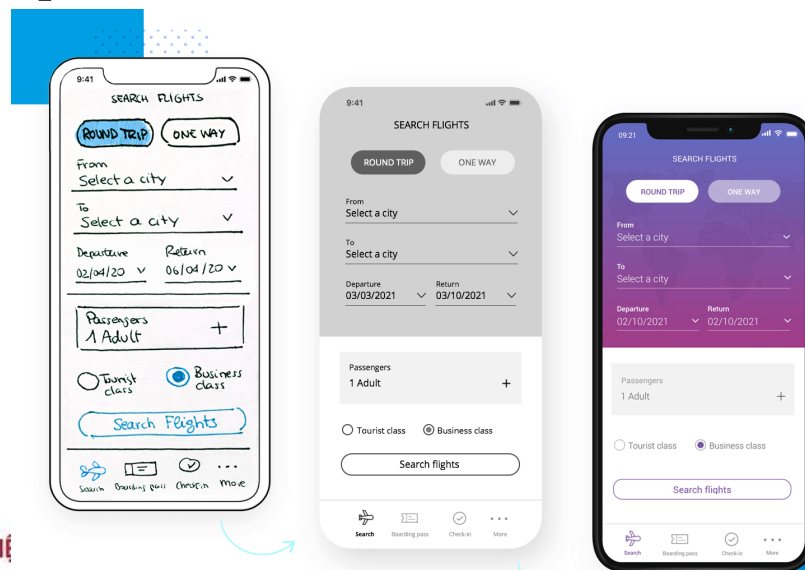
## 3.2 Các loại mẫu giao diện (2)

- Low-Fidelity Prototypes
  - Các nguyên mẫu có độ trung thực thấp rất đơn giản, dễ chế tạo và sử dụng.
  - Đây là những bản phác thảo hoặc bản in của một thiết kế nhất định.
  - Nguyên mẫu có độ trung thực thấp không cho phép người dùng tương tác với sản phẩm. Sketches, Storyboards và Wireframes là những ví dụ điển hình về kỹ thuật nguyên mẫu có độ trung thực thấp.



## 3.2 Các loại mẫu giao diện (3)

- High-fidelity prototypes
  - Các nguyên mẫu có độ trung thực cao cho phép đầy đủ chức năng và tương tác độc lập.
  - Công cụ để kiểm tra khả năng sử dụng và tinh chỉnh các chi tiết nhưng tốn nhiều thời gian để tạo và điều chỉnh.
  - Phù hợp hơn cho các bài kiểm tra khả năng sử dụng trong giai đoạn cuối của quá trình phát triển.



# Mục lục

1. Tổng quan về thiết kế giao diện
2. Nguyên lý thiết kế UI/UX cho di động
3. Quy trình xây dựng giao diện người dùng
4. **Xây dựng giao diện người dùng trong Flutter**
5. Xây dựng giao diện người dùng trong ReactNative

## 4.1 Xây dựng Layout trong Flutter

- Ứng dụng Flutter sử dụng Widget để xây dựng giao diện người dùng.
- Trong Flutter các layout cũng là một loại widget, nhiệm vụ của chúng là bố trí các widget con, tạo nên giao diện người dùng cho ứng dụng.
- Flutter cung cấp một bộ widget phong phú dưới dạng các gói gồm nhiều loại layout khác nhau như *Container*, *Center*, *Align*...
- Có hai loại widget layout chính trong Flutter
  - Single Child Widgets - Chỉ có một widget con
  - Multiple Child Widgets - Có nhiều widget con

## 4.1 Xây dựng Layout trong Flutter (2)

- Các widgets tạo bố cục toàn cảnh:
  - **MaterialApp widget**: tạo một khung outer invisible

```
Widget build(BuildContext context) {  
  return MaterialApp(  
    home: MainWidget(),  
    title: "Ch 6 Layouts",  
    theme: ThemeData(primarySwatch: Colors.deepPurple),  
    routes: <String, WidgetBuilder>{  
      '/scene1: (BuildContext ctx) => MyWidget1(),  
      '/scene2: (BuildContext ctx) => MyWidget2(),  
      '/scene3: (BuildContext ctx) => MyWidget3(),  
    },  
    debugShowCheckedModeBanner: false,  
  );  
}
```

## 4.1 Xây dựng Layout trong Flutter (3)

- Các widgets tạo bố cục toàn cảnh:
  - **Scaffold widget**: tạo một khung inner visible

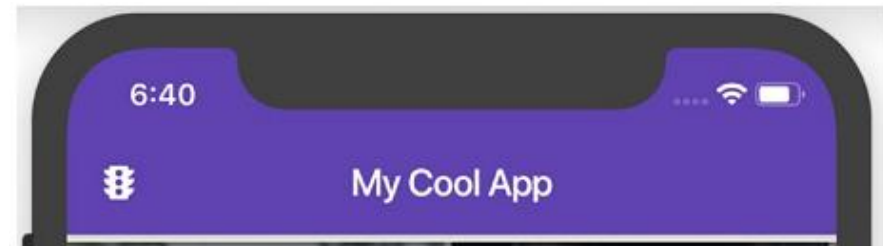
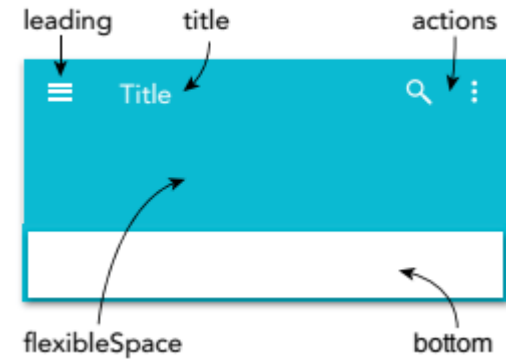
```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: MyAppBar(),
    drawer: MyNavigationDrawer(),
    body: TheRealContentOfThisPartOfTheApp(),
    floatingActionButton: FloatingActionButton(
      child: Icon(Icons.add),
      onPressed: () { /* Do things here */},
    ),
    bottomNavigationBar: MyBottomNavBar,
  );
}
```

## 4.1 Xây dựng Layout trong Flutter (4)

- Các widgets tạo bố cục toàn cảnh:
  - **AppBar widget:**

```
return Scaffold(  
  appBar: AppBar(  
    leading: Icon(Icons.traffic),  
    title: Text("My Cool App"),  
  ),  
  /* More stuff here. FAB, body, drawer, etc. */  
);
```

```
flexibleSpace: SafeArea(  
  child: Icon(  
    Icons.photo_camera,  
    size: 75.0,  
    color: Colors.white70,  
  ),  
),
```



No SafeArea



With SafeArea





## 4.1 Xây dựng Layout trong Flutter (5)

- Các widgets tạo bố cục toàn cảnh:
  - **SnackBar widget:**



```
GestureDetector(  
  child: PersonCard(person),  
  onTap: () {  
    String msg = '${person['name']['first']} deleted.';  
    final SnackBar sb = SnackBar(  
      content: Text(msg),  
      duration: Duration(seconds: 5),  
      action: SnackBarAction(  
        textColor: Colors.white,  
        label: "UNDO",  
        onPressed: () {},  
      ),  
    );  
    Scaffold.of(context).showSnackBar(sb);  
  })
```

## 4.2 Single Child Widgets

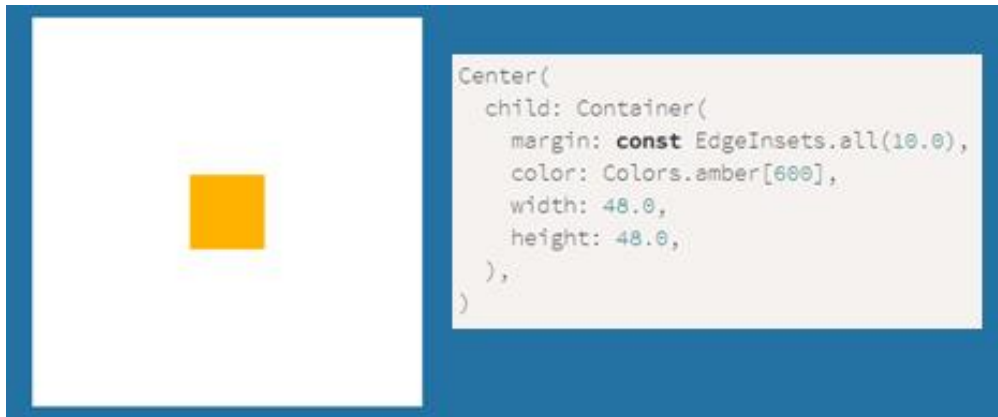
- Single Child Widgets: chỉ có duy nhất một widget con và cung cấp chức năng bố trí nhất định.
  - Ví dụ, Center widget căn giữa widget con so với widget cha

- FittedBox
- AspectRatio
- ConstrainedBox
- Baseline
- FractionallySizedBox
- IntrinsicHeight
- IntrinsicWidth

- LimitedBox
- OffStage
- OverflowBox
- SizedBox
- SizedOverflowBox
- Transform
- CustomSingleChildLayout

## 4.2 Single Child Widgets (2)

- Container: cho phép tạo phần tử trực quan hình chữ nhật



```
class BlueBox extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Container(  
      width: 50,  
      height: 50,  
      decoration: BoxDecoration(  
        color: Colors.blue,  
        border: Border.all(),  
      ),  
    );  
  }  
}
```

## 4.2 Single Child Widgets (3)

- **SizedBox**: tạo một hộp có kích thước xác định.

ví dụ tạo không gian trống



```
Widget build(BuildContext context) {  
  return Row(  
    children: [  
      BlueBox(),  
      SizedBox(  
        width: 100,  
        child: BlueBox(),  
      ),  
      BlueBox(),  
    ],  
    textDirection: TextDirection.ltr,  
  );  
}
```



```
Widget build(BuildContext context) {  
  return Row(  
    children: [  
      BlueBox(),  
      SizedBox(width: 50),  
      BlueBox(),  
      BlueBox(),  
    ],  
    textDirection: TextDirection.ltr,  
  );  
}
```

## 4.2 Single Child Widgets (4)

- **Expanded widget:** một widget giúp mở rộng không gian cho một widget con của Row hoặc Column theo trục chính (main axis).

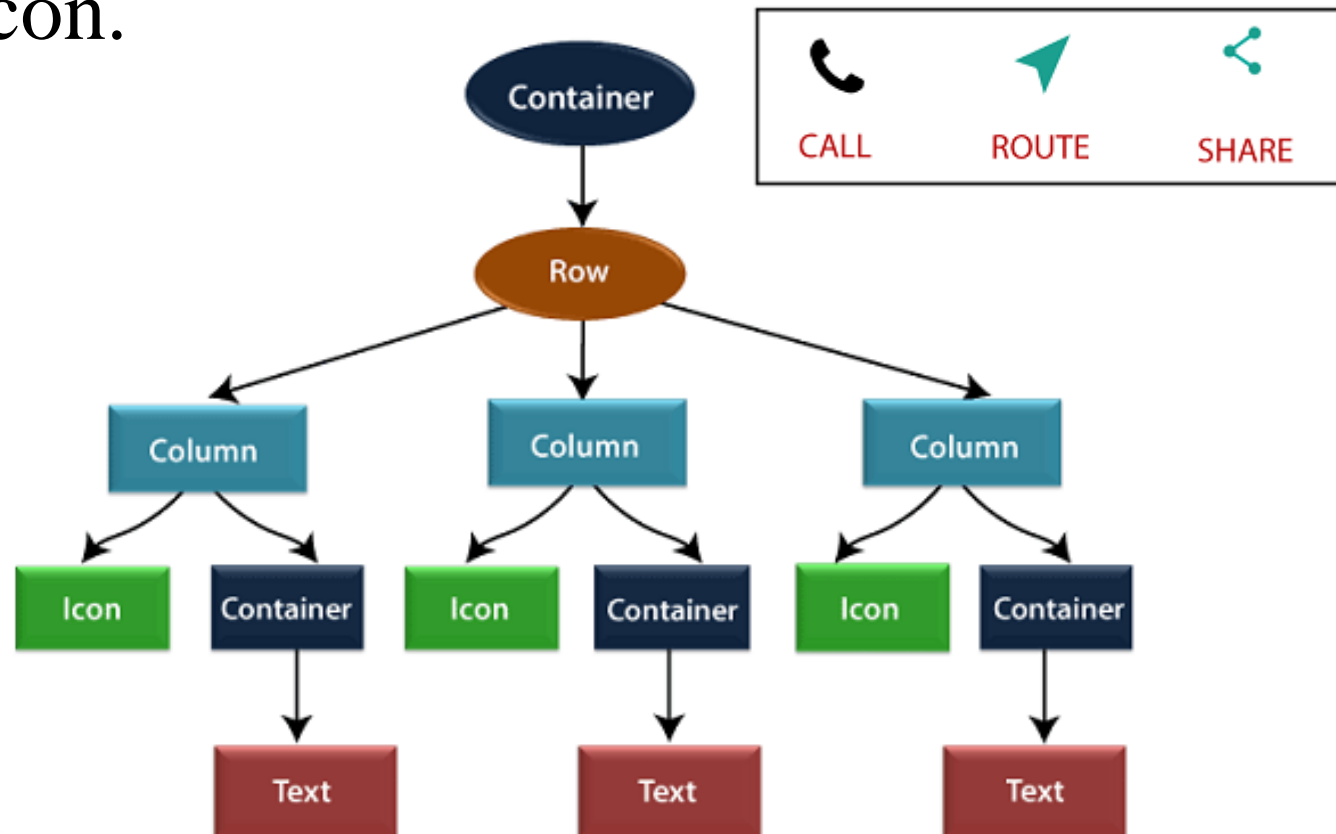
```
Widget build(BuildContext context) {  
  return Row(  
    children: [  
      BlueBox(),  
      Expanded(child: BlueBox()),  
      BlueBox(),  
    ],  
    textDirection: TextDirection.ltr,  
  );  
}
```



## 4.3 Multiple Child Widgets

- Multiple Child Widgets: cho phép có nhiều widget con.

- Row
- Column
- ListView
- GridView
- Stack
- Table
- Flow
- ...



## 4.3 Multiple Child Widgets (2)

### - Row và Column

```
Row(  
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
  children: [  
    Image.asset('images/pic1.jpg'),  
    Image.asset('images/pic2.jpg'),  
    Image.asset('images/pic3.jpg'),  
  ],  
);
```



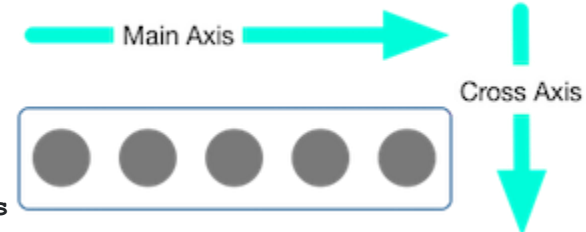
App source: [row\\_column](#)

```
Row(  
  mainAxisAlignment: MainAxisAlignment.min,  
  children: [  
    Icon(Icons.star, color: Colors.green[500]),  
    Icon(Icons.star, color: Colors.green[500]),  
    Icon(Icons.star, color: Colors.green[500]),  
    Icon(Icons.star, color: Colors.black),  
    Icon(Icons.star, color: Colors.black),  
  ],  
)
```



App source: [pavlova](#)

Row



mainAxisAlignment = Horizontal Axis  
crossAxisAlignment = Vertical Axis

## 4.3 Multiple Child Widgets (3)

### - Row và Column

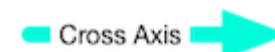
```
Column(  
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
    children: [  
        Image.asset('images/pic1.jpg'),  
        Image.asset('images/pic2.jpg'),  
        Image.asset('images/pic3.jpg'),  
    ],  
);
```



Column



Main Axis



mainAxisAlignment = Vertical Axis  
crossAxisAlignment = Horizontal Axis



## 4.3 Multiple Child Widgets (4)

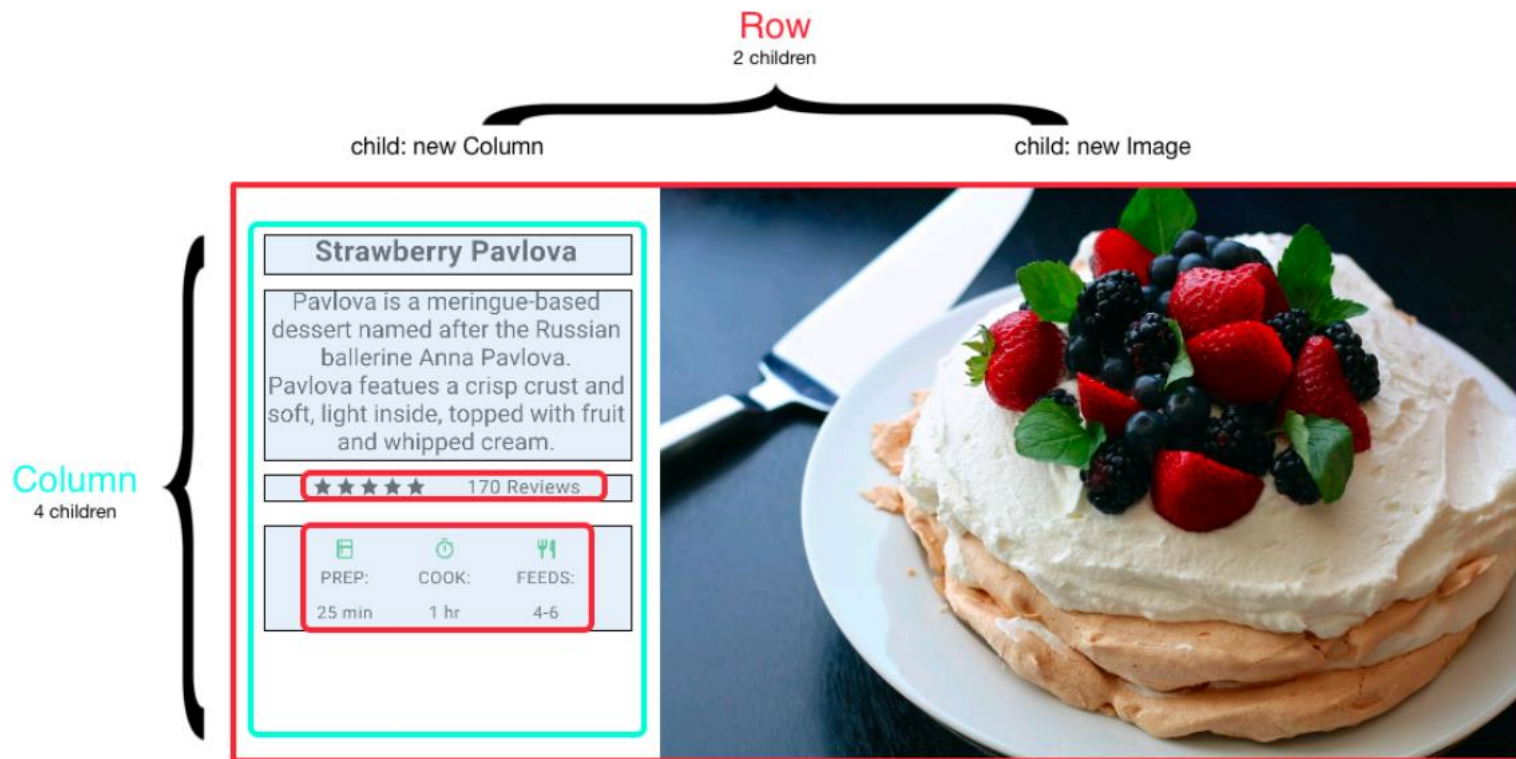
- Ví dụ:

```
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Row(  
      mainAxisAlignment: MainAxisAlignment.spaceAround,  
      crossAxisAlignment: CrossAxisAlignment.center,  
      children: [  
        BlueBox(),  
        BiggerBlueBox(),  
        BlueBox(),  
      ],  
      textDirection: TextDirection.ltr,  
    );  
  }  
}
```



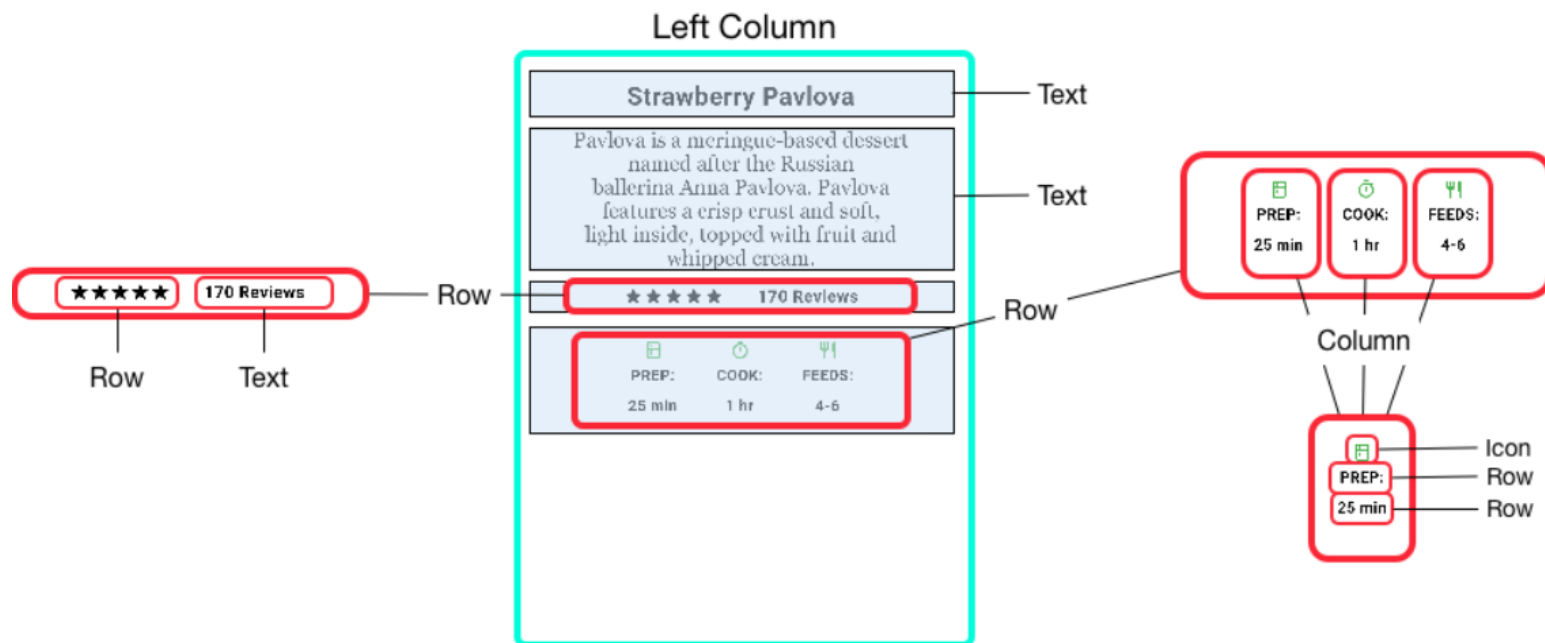
## 4.3 Multiple Child Widgets (5)

- Row và Column: Phân tách giao diện thành các Row và Column để bố trí các widgets



## 4.3 Multiple Child Widgets (6)

- Row và Column: Phân tách giao diện thành các Row và Column để bố trí các widgets



## 4.3 Multiple Child Widgets (7)

- **Flexible widget:** cho phép widget con của Row, Column, hoặc Flex khả năng mở rộng linh hoạt để lấp đầy không gian có sẵn trong trục chính.
- Ví dụ:

```
Widget build(BuildContext context) {  
  return Row(  
    children: [  
      BlueBox(),  
      Flexible(  
        fit: FlexFit.loose,  
        flex: 1,  
        child: BlueBox(),  
      ),  
      Flexible(  
        fit: FlexFit.tight,  
        flex: 1,  
        child: BlueBox(),  
      ),  
    ],  
    textDirection: TextDirection.ltr,  
  );  
}
```



## 4.3 Multiple Child Widgets (8)

- Ví dụ:

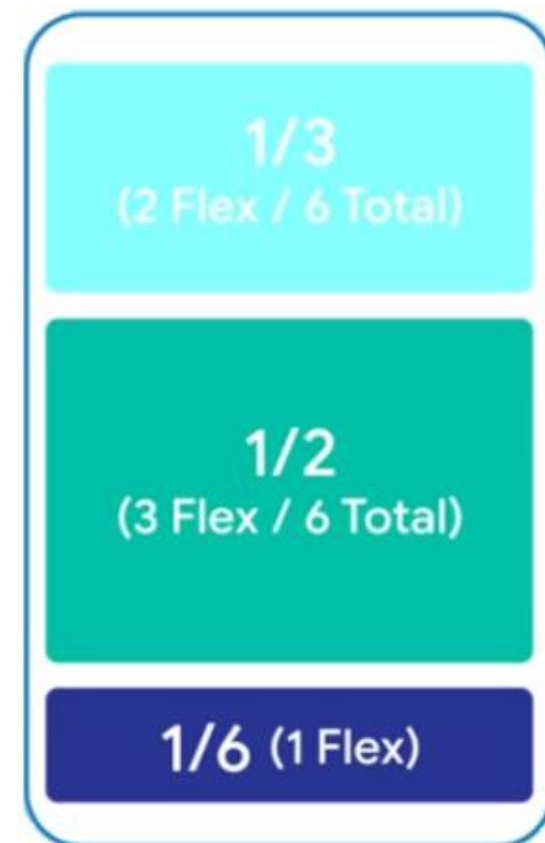
```
Widget build(BuildContext context) {  
  return Row(  
    children: [  
      BlueBox(),  
      Flexible(  
        fit: FlexFit.tight,  
        flex: 1,  
        child: BlueBox(),  
      ),  
      Flexible(  
        fit: FlexFit.tight,  
        flex: 3,  
        child: BlueBox(),  
      ),  
    ],  
    textDirection: TextDirection.ltr,  
  );  
}
```



## 4.3 Multiple Child Widgets (9)

- Ví dụ:

```
Column(  
  children: <Widget>[  
    Flexible(flex: 2,  
      fit: FlexFit.tight,  
      child: Container(  
        height: 100, color: Colors.cyan,  
      ),  
    ),  
    Flexible(flex: 3,  
      child: Container(  
        color: Colors.teal,  
      ),  
    ),  
    Flexible(flex: 1,  
      child: Container(  
        color: Colors.indigo,  
      ),  
    ],  
)
```



## 4.3 Multiple Child Widgets (10)

- **Spacer widget:** tạo ra một khoảng không gian trống, có thể điều chỉnh được, được sử dụng để điều chỉnh khoảng cách giữa các Widget con bên trong một Flex container như Column, Row, ..

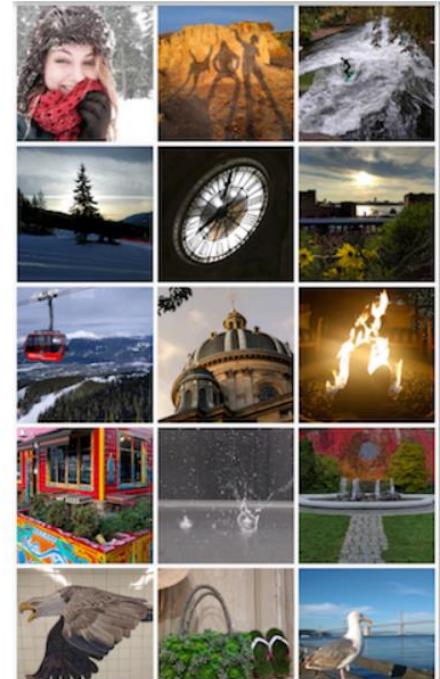


```
Widget build(BuildContext context) {  
  return Row(  
    children: [  
      BlueBox(),  
      Spacer(flex: 1),  
      BlueBox(),  
      BlueBox(),  
    ],  
    textDirection: TextDirection.ltr,  
  );  
}
```

## 4.3 Multiple Child Widgets (11)

- GridView: hiển thị các widget dưới dạng danh sách hai chiều
- Khi nội dung quá dài sẽ tự động xuất hiện thanh cuộn

```
Widget _buildGrid() => GridView.extent(  
  maxCrossAxisExtent: 150,  
  padding: const EdgeInsets.all(4),  
  mainAxisSpacing: 4,  
  crossAxisSpacing: 4,  
  children: _buildGridTileList(30));  
  
// The images are saved with names pic0.jpg, pic1.jpg...pic29.jpg.  
// The List.generate() constructor allows an easy way to create  
// a list when objects have a predictable naming pattern.  
List<Container> _buildGridTileList(int count) => List.generate(  
  count, (i) => Container(child: Image.asset('images/pic$i.jpg')));
```

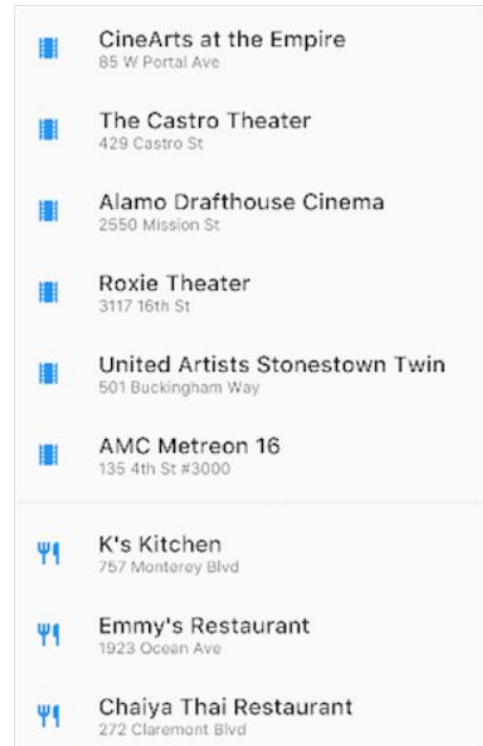




## 4.3 Multiple Child Widgets (12)

- ListView: một widget giống như column, tự động cung cấp thanh cuộn khi nội dung của nó quá dài.










```
Widget _buildList() => ListView(  
  children: [  
    _tile('CineArts at the Empire', '85 W Portal Ave', Icons.theaters),  
    _tile('The Castro Theater', '429 Castro St', Icons.theaters),  
    _tile('Alamo Drafthouse Cinema', '2550 Mission St', Icons.theaters),  
    _tile('Roxie Theater', '3117 16th St', Icons.theaters),  
    _tile('United Artists Stonestown Twin', '501 Buckingham Way',  
      Icons.theaters),  
    _tile('AMC Metreon 16', '135 4th St #3000', Icons.theaters),  
    Divider(),  
    _tile('K\'s Kitchen', '757 Monterey Blvd', Icons.restaurant),  
    _tile('Emmy\'s Restaurant', '1923 Ocean Ave', Icons.restaurant),  
    _tile(  
      'Chaiya Thai Restaurant', '272 Claremont Blvd', Icons.restaurant),  
    _tile('La Ciccia', '291 30th St', Icons.restaurant),  
  ],  
);
```



## 4.3 Multiple Child Widgets (13)

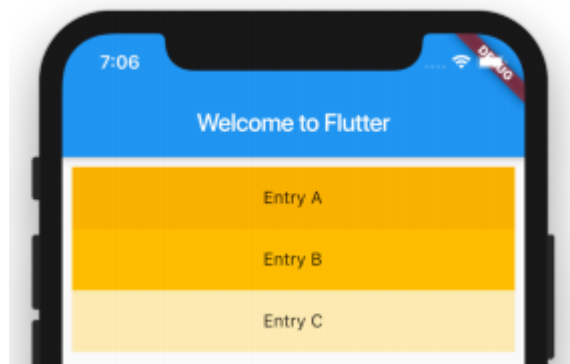
- ListView: (tiếp)

```
ListTile _tile(String title, String subtitle, IconData icon) => ListTile(  
  title: Text(title,  
    style: TextStyle(  
      fontWeight: FontWeight.w500,  
      fontSize: 20,  
    )),  
  subtitle: Text(subtitle),  
  leading: Icon(  
    icon,  
    color: Colors.blue[500],  
  ),  
);
```

|   |  |
|---|--|
|    | CineArts at the Empire<br>85 W Portal Ave            |
|    | The Castro Theater<br>429 Castro St                  |
|    | Alamo Drafthouse Cinema<br>2550 Mission St           |
|    | Roxie Theater<br>3117 16th St                        |
|    | United Artists Stonestown Twin<br>501 Buckingham Way |
|    | AMC Metreon 16<br>135 4th St #3000                   |
|   |  |
|  | K's Kitchen<br>757 Monterey Blvd                     |
|  | Emmy's Restaurant<br>1923 Ocean Ave                  |
|  | Chaiya Thai Restaurant<br>272 Claremont Blvd         |

## 4.3 Multiple Child Widgets (14)

- ListView: ví dụ khác



```
body: ListView(  
  padding: EdgeInsets.all(8),  
  children: [  
    Container(  
      height: 50,  
      color: Colors.amber[600],  
      child: Center(child: Text('Entry A')),  
    ),  
    Container(  
      height: 50,  
      color: Colors.amber[500],  
      child: Center(child: Text('Entry B')),  
    ),  
    Container(  
      height: 50,  
      color: Colors.amber[100],  
      child: Center(child: Text('Entry C')),  
    ),  
  ],  
)
```

75

## 4.3 Multiple Child Widgets (15)

- Stack: Các widget được sắp xếp chồng lên nhau.



```
Widget _buildStack() => Stack(  
  alignment: const Alignment(0.6, 0.6),  
  children: [  
    CircleAvatar(  
      backgroundImage: AssetImage('images/pic.jpg'),  
      radius: 100,  
    ),  
    Container(  
      decoration: BoxDecoration(  
        color: Colors.black45,  
      ),  
      child: Text(  
        'Mia B',  
        style: TextStyle(  
          fontSize: 20,  
          fontWeight: FontWeight.bold,  
          color: Colors.white,  
        ),  
      ),  
    ),  
  ],  
);
```

## 4.4 Gestures

- Flutter framework cho phép xử lý cử chỉ người dùng từ các thao tác đơn giản đến các cử chỉ phức tạp như kéo và xoay.
- Các sự kiện trên màn hình trong hệ thống cử chỉ của Flutter được tách ra thành 2 lớp, đó là:
  - **Pointer (Trỏ)**: Là lớp xử lý tương tác cấp thấp, có thể xử lý một sự kiện trỏ và quyết định cách kiểm soát nó, chẳng hạn bằng một thao tác kéo, hoặc single tap (nhấn 1 lần). Flutter cung cấp lớp Listener có thể phát hiện sự kiện tương tác pointer.
  - **Gesture (Cử chỉ)**: Là lớp xử lý tương tác cấp cao. Do không phải lúc nào cũng có thể xử lý các sự kiện con trỏ bằng Listener widget, nên thay vào đó các sự kiện có thể được xử lý trên Gesture.

## 4.4 Gestures (2)

- Flutter xử lý tất cả các loại cử chỉ cấp cao thông qua một widget duy nhất **GestureDetector**.
- Để xác định các cử chỉ tác động lên một widget, chỉ cần đặt widget đó bên trong **GestureDetector** widget → **GestureDetector** sẽ bắt các cử chỉ và gửi nhiều *sự kiện* dựa trên cử chỉ đó.
- Một số cử chỉ và các sự kiện tương ứng

### Tap

- onTapDown
- onTapUp
- onTap
- onTapCancel

### Vertical drag

- onVerticalDragStart
- onVerticalDragUpdate
- onVerticalDragEnd

### Horizontal drag

- onHorizontalDragStart
- onHorizontalDragUpdate
- onHorizontalDragEnd

### Pan

- onPanStart
- onPanUpdate
- onPanEnd

### Double tap

- onDoubleTap

### Long press

- onLongPress

## 4.4 Gestures (3)

- Ví dụ Tap:

```
class _TapWidgetExampleState extends State<TapWidgetExample> {  
  int _counter = 0;  
  
  @override  
  Widget build(BuildContext context) {  
    return GestureDetector(  
      onTap: () {  
        setState(() {  
          _counter++;  
        });  
      },  
  
      child: Container(  
        color: Colors.grey,  
        child: Center(  
          child: Text(  
            "Tap count: $_counter",  
            style: Theme.of(context).textTheme.display1,  
          ),  
        ),  
      ),  
    );  
  }  
}
```

## 4.4 Gestures (4)

- Ví dụ Double tap:

```
GestureDetector(  
  onDoubleTap: () {  
    setState(() {  
      _counter++;  
    });  
  },  
  child: ... // for brevity  
);
```

- Ví dụ Press and hold:

```
GestureDetector(  
  onLongPress: () {  
    setState(() {  
      _counter++;  
    });  
  },  
  child: ... // for brevity  
);
```



## 4.5 Navigator và Routing

- Trong Flutter, route tương ứng với 1 màn hình được quản lý bởi **Navigator** widget của ứng dụng.
- Điều hướng từ một Full-screen (page/screen) để làm một công việc xác định nào đó (chuyển sang một Full-screen khác) sử dụng Navigator thì được gọi là **Routing**.
- **Navigator** widget quản lý ngăn xếp điều hướng, push 1 **route** mới vào hoặc pop 1 **route** khác ra.

## 4.5 Navigator và Routing (2)

- Navigation.push: Từ một màn hình bất kì, muốn chuyển sang một màn hình khác

```
Navigator.push( context, MaterialPageRoute(  
    builder: (context) => Widget()),  
);
```

- Navigation.pop: Được sử dụng để quay về trang trước

```
Navigator.pop(context);
```

## 4.5 Navigator và Routing (3)

- Định nghĩa routes.

```
MaterialApp(  
  // Start the app with the "/" named route. In this case, the app starts  
  // on the FirstScreen widget.  
  initialRoute: '/',  
  routes: {  
    // When navigating to the "/" route, build the FirstScreen widget.  
    '/': (context) => FirstScreen(),  
    // When navigating to the "/second" route, build the SecondScreen widget.  
    '/second': (context) => SecondScreen(),  
  },  
);
```

- Điều hướng đến màn hình thứ hai

```
// Within the `FirstScreen` widget  
onPressed: () {  
  // Navigate to the second screen using a named route.  
  Navigator.pushNamed(context, '/second');  
}
```

# Mục lục

1. Tổng quan về thiết kế giao diện
2. Nguyên lý thiết kế UI/UX cho di động
3. Quy trình xây dựng giao diện người dùng
4. Xây dựng giao diện người dùng trong Flutter
5. **Xây dựng giao diện người dùng trong ReactNative**

## 5.1 Xây dựng Layout trong ReactNative

- Tất cả các component cơ bản của RN đều có một **prop** tên là **style**
  - Tên và giá trị của style thường khớp với cách hoạt động của CSS, tuy nhiên tên các thuộc tính được viết theo phong cách *camel casing* (ví dụ: backgroundColor thay vì background-color)
- **prop style** có thể là một đối tượng JavaScript thuần túy
- Sử dụng **StyleSheet.create** để xác định một số kiểu tập trung thường dễ dàng quản lý hơn
- Style có thể "xếp tầng" (cascade) giống như trong CSS

## 5.1 Xây dựng Layout trong ReactNative (2)

- Áp dụng style trong các ứng dụng RN

Sử dụng inline styles

```
<View style={{marginLeft: 20,  
marginTop: 20}}>  
<Text style={{fontSize:  
18,color: 'red'}}>Some  
Text</Text>  
</View>
```

Định nghĩa style qua StyleSheet

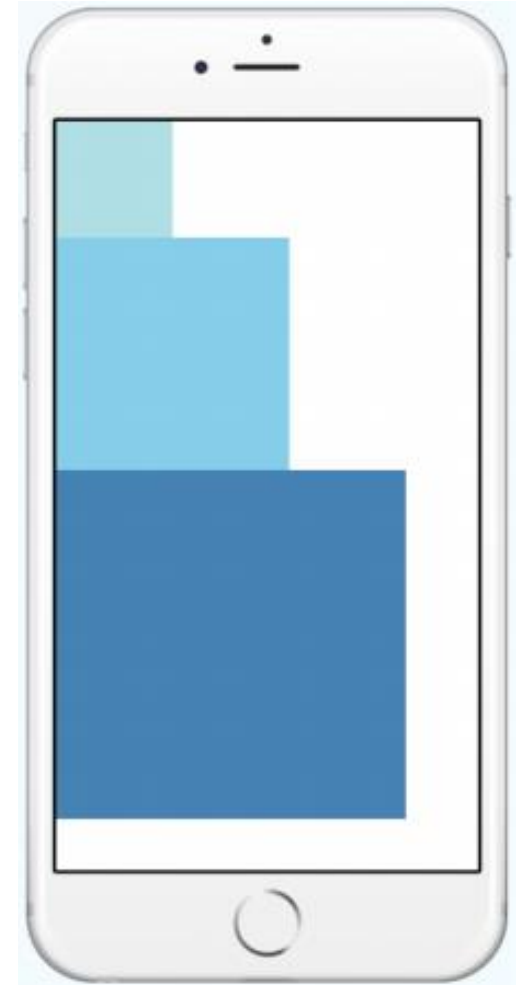
```
<View style={styles.container}>  
<Text  
style={[styles.message,styles.warning]}>S  
ome Text</Text>  
</View>  
  
const styles = StyleSheet.create({  
  container: {  
    marginLeft: 20,  
    marginTop: 20  
  },  
  message: {  
    fontSize: 18  
  },  
  warning: {  
    color: 'red'  
  }  
});
```

## 5.1 Xây dựng Layout trong ReactNative (3)

- Kích thước component được thiết lập qua các giá trị style: **width** và **height**
- Các giá trị có đơn vị là các pixel độc lập với thiết bị - chúng sẽ xuất hiện cùng kích thước trên một nền tảng nhất định bất kể độ phân giải màn hình.


```
import React, { Component } from 'react';
import { AppRegistry, View } from 'react-native';
export default class FixedDimensionsBasics extends Component {
  render() {
    return (
      <View>
        <View style={{width: 50, height: 50,
          backgroundColor: 'powderblue'}} />
        <View style={{width: 100, height: 100,
          backgroundColor: 'skyblue'}} />
        <View style={{width: 150, height: 150,
          backgroundColor: 'steelblue'}} />
      </View>
    );
  }
}
```

Unitless - pixel độc lập với mật độ hoặc tỷ lệ phần trăm



## 5.1 Xây dựng Layout trong ReactNative (4)

- Flexbox được thiết kế để cung cấp bố cục nhất quán trên các kích thước màn hình khác nhau
- Flexbox là các thuộc tính được sử dụng để xây dựng bố cục các component
- Kết hợp **flexDirection**, **alignItems** và **justifyContent** để đạt được layout phù hợp

| Property   | Values  | Description   |
|--|---|---|
| <b>flexDirection</b>   | 'column', 'row'   | Được sử dụng để chỉ định xem các phần tử sẽ được căn chỉnh theo chiều dọc hay chiều ngang.  |
| <b>justifyContent</b>  | 'center', 'flex-start', 'flex-end', 'space-around', 'space-between' | Được sử dụng để xác định cách các phần tử nên được phân phối bên trong vùng chứa.   |
|  <b>alignItems</b> | 'center', 'flex-start', 'flex-end', 'stretched'                     | Được sử dụng để xác định cách các phần tử nên được phân phối bên trong vùng chứa dọc theo trục thứ cấp (đối diện với flexDirection) |



## 5.1 Xây dựng Layout trong ReactNative (5)

### - Thuộc tính flex ví dụ:

```
import React, { Component } from 'react';
import { AppRegistry, View } from 'react-native';
export default class FlexDimensionsBasics extends Component {
  render() {
    return (
      <View style={{flex: 1}}>
        <View style={{flex: 1, backgroundColor:
          'powderblue'}} />
        <View style={{flex: 2, backgroundColor:
          'skyblue'}} />
        <View style={{flex: 3, backgroundColor:
          'steelblue'}} />
      </View>
    );
  }
}
```

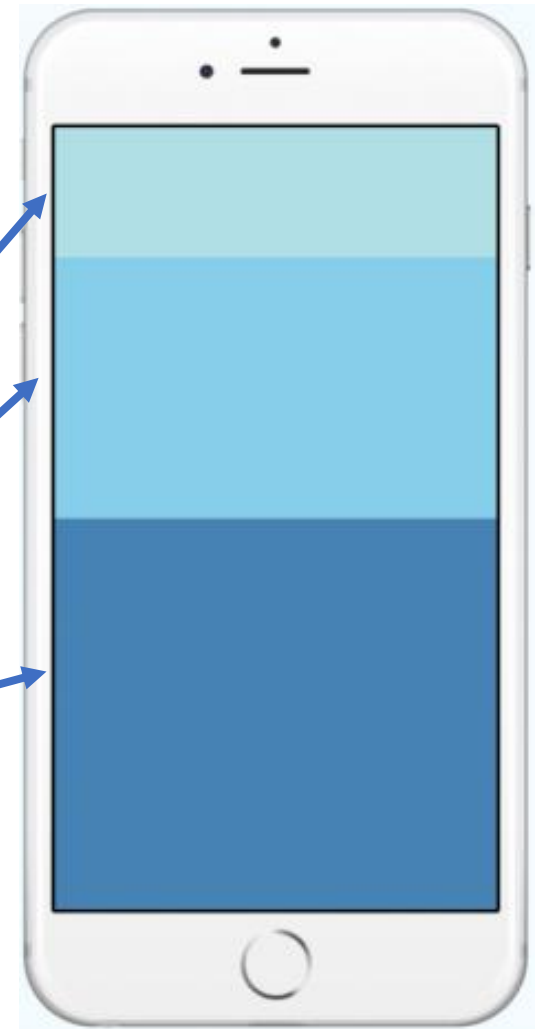
Root View có flex: 1 nên nó chiếm toàn bộ màn hình.

1/6

2/6

3/6

Tổng số flex các anh chị em: 6



## 5.1 Xây dựng Layout trong ReactNative (6)

- Ví dụ thuộc tính flexDirection

```
import React, { Component } from 'react';
import { AppRegistry, View } from 'react-native';
export default class FlexDirectionBasics extends Component {
  render() {
    return (
      // Try setting `flexDirection` to `column`.
      <View style={{flex: 1, flexDirection: 'row'}}>
        <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
        <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />
        <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />
      </View>
    );
  }
}
```



# 5.1 Xây dựng Layout trong ReactNative (7)

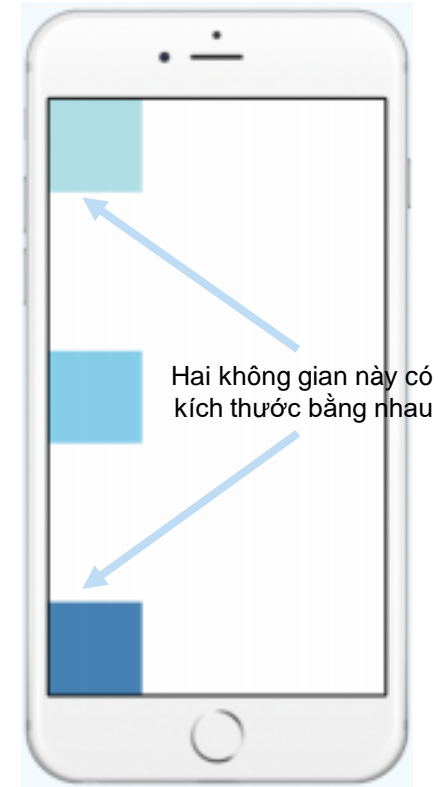
## ▪ Ví dụ thuộc tính justifyContent

- Xác định sự phân bố của thành phần con dọc theo trục chính.

- Các tùy chọn là

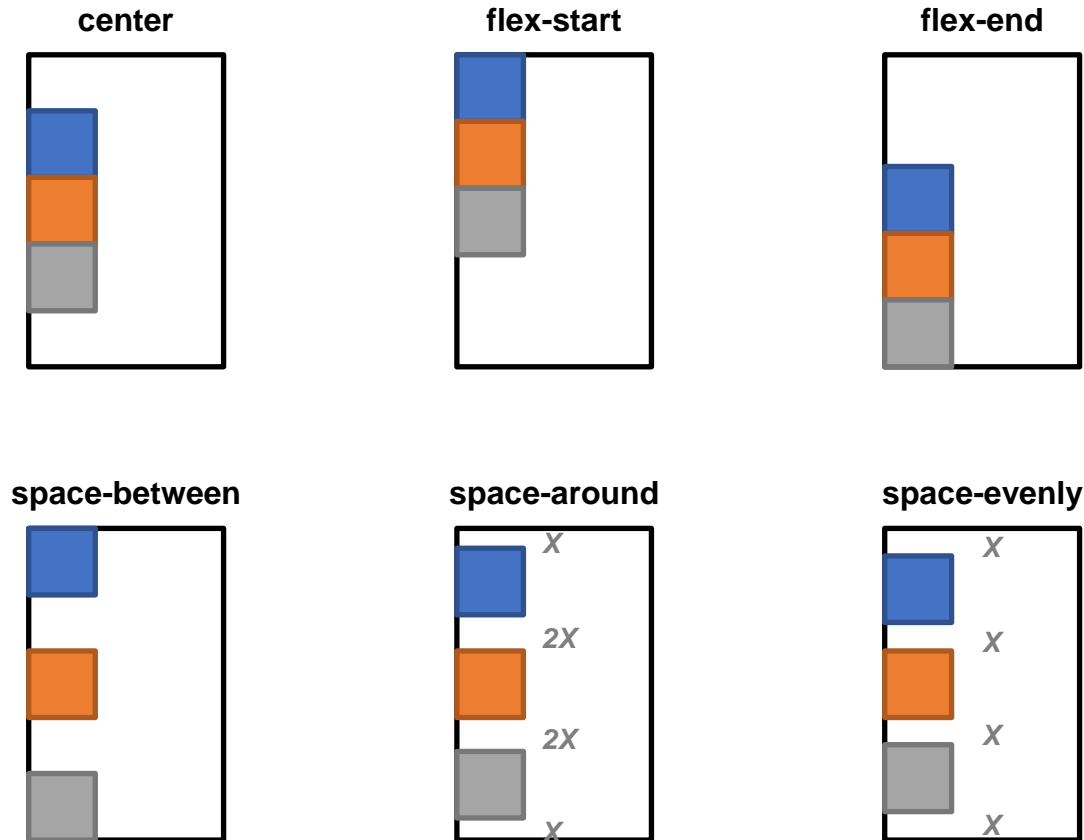
- flex-start
- center
- flex-end
- space-around
- space-between
- space-evenly.

```
import React, { Component }
from 'react';
import { AppRegistry, View }
from 'react-native';
export default class JustifyContentBasics
extends Component {
  render() {
    return (
      // Try setting `justifyContent` to `center`.
      // Try setting `flexDirection` to `row`.
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'space-between',
      }}>
        <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
        <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />
        <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />
      </View>
    );
  }
}
```



## 5.1 Xây dựng Layout trong ReactNative (8)

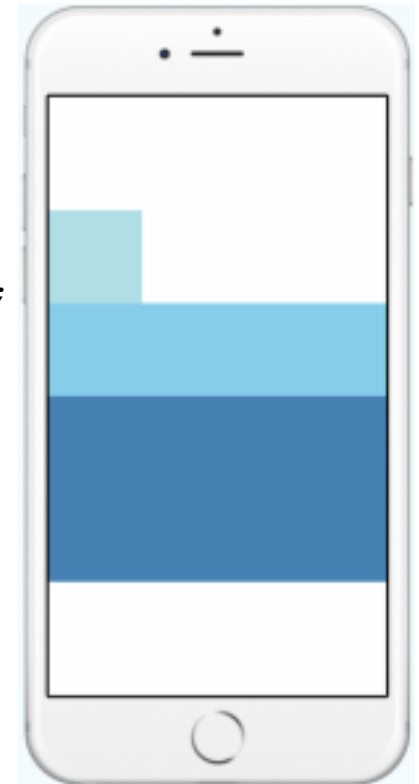
- Ví dụ thuộc tính justifyContent



# 5.1 Xây dựng Layout trong ReactNative (9)

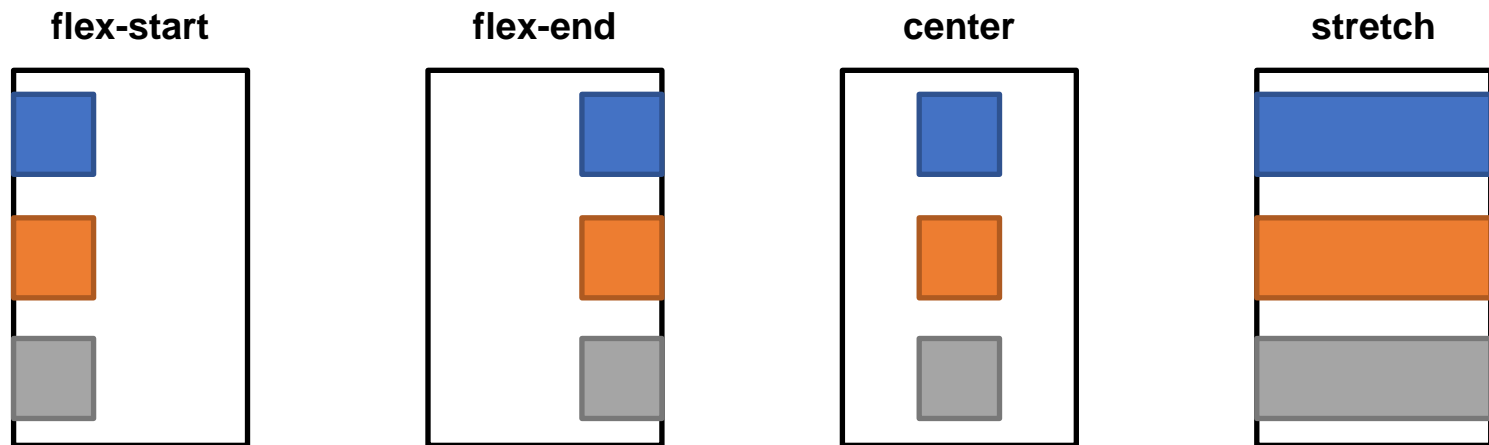
## ▪ Ví dụ thuộc tính alignItems

- Xác định sự căn chỉnh của các thành phần con dọc theo trục phụ
    - nếu trục chính là hàng thì trục phụ là cột và ngược lại.
  - Các tùy chọn là
    - flex-start
    - center
    - flex-end
    - stretch
- ```
import React, { Component } from 'react';
import { AppRegistry, View } from 'react-native';
export default class AlignItemsBasics
  extends Component {
  render() {
    return (
      // Try setting `alignItems` to 'flex-start'
      // Try setting `justifyContent` to 'flex-end'.
      // Try setting `flexDirection` to 'row`.
      <View style={{
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'stretch',
      }}>
        <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
        <View style={{height: 50, backgroundColor: 'skyblue'}} />
        <View style={{height: 100, backgroundColor: 'steelblue'}} />
      </View>
    );
  }
}
```



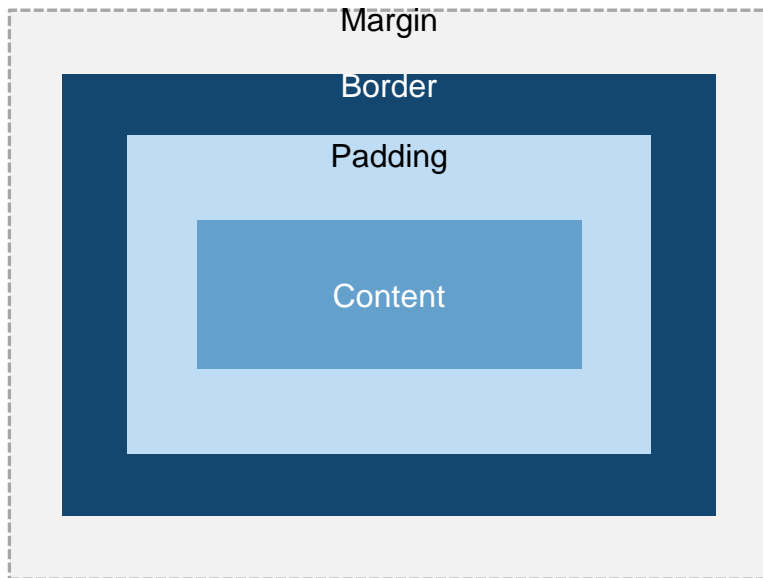
## 5.1 Xây dựng Layout trong ReactNative (10)

- Ví dụ thuộc tính alignItems (Lưu ý: việc đặt thuộc tính width / height sẽ ghi đè stretch)



## 5.1 Xây dựng Layout trong ReactNative (11)

- Margins, Borders & Padding
- Các thành phần tuân theo mô hình Hộp, tương tự như mô hình hộp CSS:



```
const styles = StyleSheet.create({
  content: {
    padding: 20,
    margin: 0,
    backgroundColor: '#ef4c',
    width: 125,
    height: 125,
    borderWidth: 1,
    borderColor: 'red',
    textAlign: 'center'
  }
});
```

## 5.1 Xây dựng Layout trong ReactNative (12)

- Margins, Borders & Padding: ví dụ

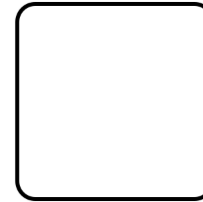
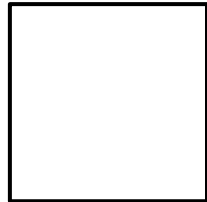
backgroundColor: "red"

borderWidth: 2

borderColor: "green"

borderRadius: 10

opacity: 0.3



Margin (20px on all sides, transparent)

```
<View style={{...}} />
```

```
<Text style={{
```

```
  backgroundColor: 'skyblue',
```

```
  padding: 10,
```

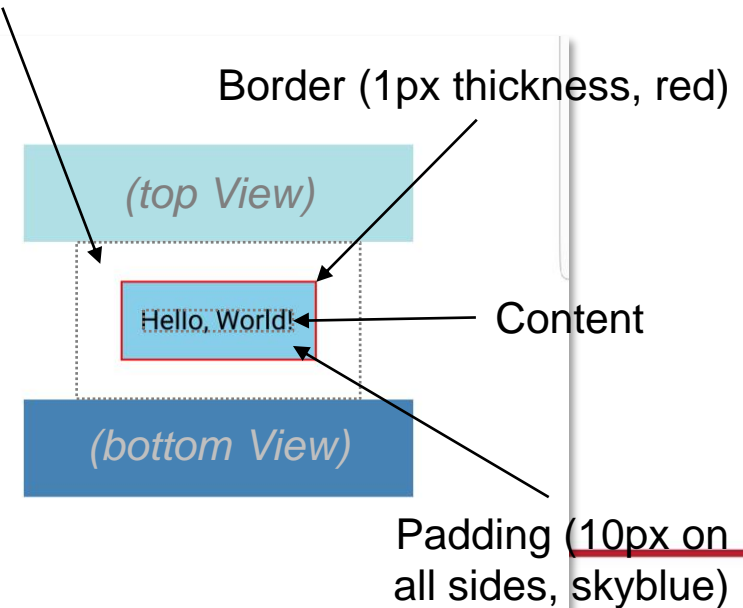
```
  borderWidth: 1,
```

```
  borderColor: 'red',
```

```
  margin: 20
```

```
}}>Hello, World!</Text>
```

```
<View style={{...}} />
```





## 5.2 React Navigation

- React Navigation không phải từ facebook, do cộng đồng React Native tạo ra
  - Sử dụng các bản gốc native dành riêng cho nền tảng
- Phải cài đặt thư viện React Navigation trong thư mục dự án
  - Phải cài đặt mỗi khi tạo một dự án mới
  - Di chuyển vào thư mục dự án và chạy lệnh cài đặt:
  - **npm install --save react-navigation**

## 5.2 React Navigation (2)

- createStackNavigator
  - hàm trả về một thành phần React
  - nhận một đối tượng cấu hình định tuyến (route) và một đối tượng tùy chọn
  - có thể xuất trực tiếp từ **App.js** để sử dụng làm thành phần gốc của Ứng dụng
- **navigation** prop có sẵn cho tất cả các thành phần màn hình
  - (các thành phần được định nghĩa là màn hình trong cấu hình định tuyến (route) và được render bởi React Navigation dưới dạng một route)

## 5.2 React Navigation (3)

```
import { createStackNavigator } from 'react-navigation'
export default createStackNavigator({
  Home: {
    screen: Home,
  },
  Profile: {
    screen: Profile,
  },
})

class Home extends React.Component {
  render() { return (
    <Wrapper>
      <Text>Hello!, This is Home!</Text>
      <Button
        title="Go to Profile"
        onPress={() => this.props.navigation.navigate('Profile')}
      />
    </Wrapper>
  )}
}
```

## 5.2 React Navigation (4)

- Navigation Lifecycle
  - Điều gì xảy ra với Trang chủ khi chúng ta điều hướng khỏi Trang chủ hoặc khi quay lại Trang chủ?
  - Làm thế nào để màn hình phát hiện ra rằng người dùng đang rời khỏi nó hay quay lại nó?
  - Hai lệnh gọi React lifecycle quan trọng được sử dụng: **componentDidMount** và **componentWillUnmount**

## 5.2 React Navigation (5)

```
import React from 'react';
import { View, Text } from 'react-native';
import { createStackNavigator } from 'react-navigation';
class HomeScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        <Text>Home Screen</Text>
      </View>
    );
  }
}
export default createStackNavigator({
  Home: {
    screen: HomeScreen
  },
});
```

Đây sẽ là Home Screen

Các khóa (key) là tên tuyến và các giá trị (value) là cấu hình cho tuyến đó

Tạo ra một lớp thành phần điều hướng sử dụng làm lớp chính cho ứng dụng

Chỉ có một màn hình trong ứng dụng này

## 5.2 React Navigation (6)

- Abstracting

```
const RootStack = createStackNavigator({  
  Home: {  
    screen: HomeScreen  
  },  
});
```

Đây là kỹ thuật tương tự như tạo lớp Home và sau đó tạo một thể hiện của nó trong lớp mặc định đã export. Cú pháp khác nhau vì gọi một hàm trong thư viện **navigator** để tạo lớp.

```
export default class App extends React.Component {  
  render() {  
    return <RootStack />;  
  }  
}
```

Cách này giúp kiểm soát màn hình chính tốt hơn; Có thể tạo kiểu hoặc cấu hình nó

## ví dụ: thêm một route thứ hai

```
// Other code for HomeScreen here...
class DetailsScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        <Text>Details Screen</Text>
      </View>
    );
  }
}
const RootStack = createStackNavigator(
  {
    Home: HomeScreen,
    Details: DetailsScreen,
  },
  {
    initialRouteName: 'Home',
  }
);
```

Code định nghĩa Home Screen...

ngăn xếp này có hai route (hoặc màn hình), một tuyến **Home** và một tuyến **Details**. Tuyến **Home** tương ứng với thành phần **HomeScreen** và tuyến **Details** tương ứng với thành phần **DetailsScreen**. Tuyến ban đầu cho ngăn xếp là **Home**.

Chỉ định màn hình được sử dụng khi khởi động

## ví dụ: điều hướng

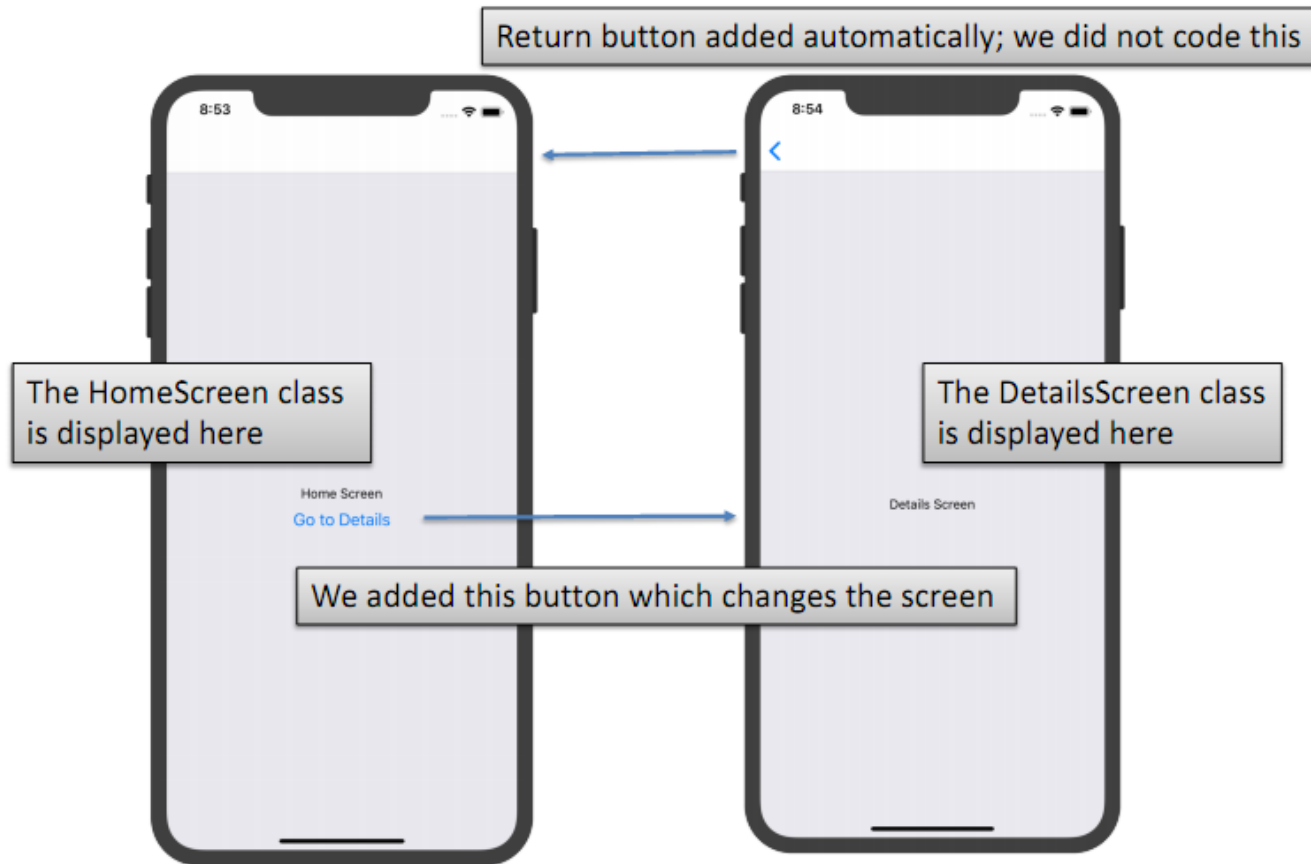
```
import React from 'react';
import { Button, View, Text } from 'react-native';
import { createStackNavigator } from 'react-navigation';
class HomeScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        <Text>Home Screen</Text>
        <Button
          title="Go to Details"
          onPress={() => this.props.navigation.navigate('Details')}
        />
      </View>
    );
  }
}
// ... other code
```

**navigate('Details')**: gọi hàm điều hướng (trên navigation prop) với tên của tuyến mà chúng ta muốn người dùng di chuyển đến

**this.props.navigation**: navigation prop được chuyển vào mọi thành phần màn hình (định nghĩa) trong trình điều hướng ngăn xếp



# Kết quả



# HẾT CHƯƠNG 5



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG  
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

**Thank you  
for your  
attentions!**

