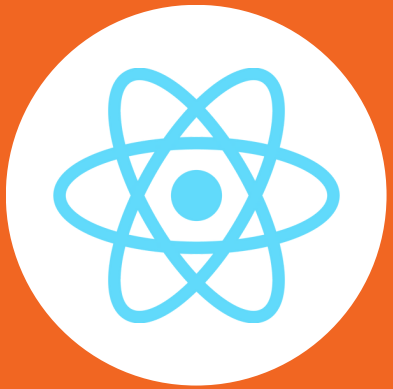
FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 4: CORE COMPONENT TRONG REACT NATIVE (P2)

PHẦN 1: CÁC THÀNH PHẦN CƠ BẢN

Mục tiêu

- Giới thiệu về **<ScrollView />**
- Hiển thị danh sách với **<FlatList/>**

ScrollView component

- **ScrollView** được sử dụng khi bạn muốn hiển thị các component dài, mà một screen không thể hiển thị hết.

  Hãy nhớ rằng ScrollViews phải có chiều cao giới hạn để hoạt động, vì chúng chứa component bên trong có chiều cao không giới hạn vào vùng container bị giới hạn (thông qua tương tác cuộn). Để ràng buộc chiều cao của ScrollView, hãy đặt chiều cao của chế độ xem trực tiếp (không được khuyến khích) hoặc đảm bảo tất cả các chế độ xem parent có chiều cao giới hạn. Quên chuyển {flex: 1} xuống ngăn xếp chế độ xem có thể dẫn đến lỗi ở đây, mà trình kiểm tra phần tử thực hiện nhanh chóng để gỡ lỗi.

  <ScrollView /> vs **<FlatList />** - nên sử dụng cái nào?

  ScrollView hiển thị tất cả các thành phần con react của nó cùng một lúc, nhưng điều này có một nhược điểm về perfomance.

ScrollView component

Hãy tưởng tượng bạn có một danh sách rất dài các mục bạn muốn hiển thị, có thể là một số màn hình có giá trị nội dung. Tạo các thành phần JS và chế độ xem gốc cho tất cả mọi thứ cùng một lúc, phần lớn trong số đó thậm chí có thể không được hiển thị, sẽ góp phần làm chậm kết xuất và tăng mức sử dụng bộ nhớ.

Đây là lúc **FlatList** phát huy tác dụng. **FlatList** hiển thị các mục một cách lazily, khi chúng sắp xuất hiện và loại bỏ các mục cuộn ra khỏi màn hình để tiết kiệm bộ nhớ và thời gian xử lý.

FlatList cũng tiện dụng nếu bạn muốn hiển thị dấu phân cách giữa các mục của mình, nhiều cột, tải cuộn vô hạn hoặc bất kỳ tính năng nào khác mà nó hỗ trợ ngoài hộp.

ScrollView component

- Các prop thường được sử dụng trong **ScrollView**
  - **horizontal:** Khi **true**, các component bên trong **ScrollView** cuộn được sắp xếp theo chiều ngang trong một hàng thay vì theo chiều dọc trong một cột.
  - **onScroll:** kích hoạt nhiều nhất một lần trên mỗi khung hình trong khi cuộn. Tần suất của các sự kiện có thể được kiểm soát bằng cách sử dụng prop **scrollEventThrottle**. Sự kiện có hình dạng sau (tất cả các giá trị đều là số):

ScrollView component

```
{
  nativeEvent: {
    contentInset: {bottom, left, right, top},
    contentOffset: {x, y},
    contentSize: {height, width},
    layoutMeasurement: {height, width},
    zoomScale
  }
}
```

- **onScrollEndDrag:** Được gọi khi người dùng dừng kéo chế độ xem cuộn và nó dừng lại hoặc bắt đầu trượt.

ScrollView component

- **refreshControl:** Một thành phần **RefreshControl**, được sử dụng để cung cấp chức năng kéo để làm mới cho **ScrollView**. Chỉ hoạt động cho **ScrollView** dọc (prop ngang phải là false).

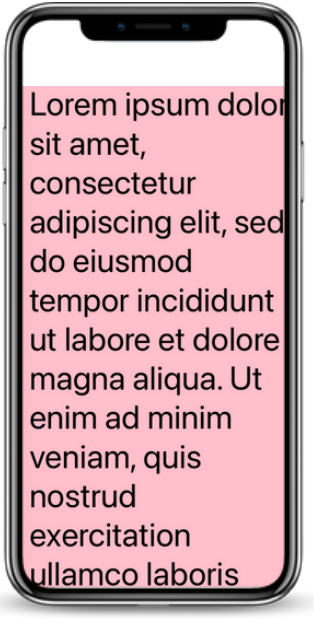
  Chi tiết về **RefreshControl** này sẽ được giới thiệu ở các phần tiếp theo

- **scrollEnabled:** Khi false, chế độ xem không thể cuộn qua tương tác chạm.

- **showsVerticalScrollIndicator:** Khi **true**, hiển thị thanh cuộn dọc

ScrollView component

- Ở ví dụ bên dưới, ta sẽ tạo một **<ScrollView/>** như hình dưới đây



ScrollView component

- Dùng **<ScrollView />** để bọc tất cả các thành phần con

```
<ScrollView style={styles.scrollView}>
  <Text style={styles.text}>…
  </Text>
</ScrollView>
```

- Ngoài ra, các bạn có thể sử dụng **ref scrollTo** của ScrollView để thực hiện scroll đến một vị trí bất kỳ trong **<ScrollView />** của chúng ta

ScrollView component

- Các bạn cần khai báo một biến để lưu **ref** của **ScrollView**

```
const scrollViewRef = useRef<ScrollView>(null);
return (
  <ScrollView ref={scrollViewRef} style={styles.scrollView}>
```

- Tiếp theo, bạn gọi hàm **scrollTo** từ **ref** của **<ScrollView />** đã lưu. y là vị trí px mà bạn muốn scroll đến.

```
scrollViewRef.current?.scrollTo({
  y: 0,
  animated: true,
});
```

FlatList component

- Để hiện danh sách, có các item có tính lặp đi lặp lại, chúng ta sử dụng **<FlatList />.**
- Một giao diện hiệu quả để hiển thị các danh sách cơ bản, phẳng, hỗ trợ các tính năng tiện dụng nhất:

  - Hoàn toàn đa nền tảng.
  - Chế độ ngang tùy chọn.
  - Callback khả năng xem có thể định cấu hình.
  - Hỗ trợ tiêu đề.
  - Hỗ trợ chân trang.
  - Hỗ trợ dấu phân cách.
  - Kéo để làm mới.
  - Đang tải cuộn.
  - Hỗ trợ ScrollToIndex.
  - Hỗ trợ nhiều cột.

FlatList component

- Để kết hiển thị nhiều cột, hãy sử dụng prop **numColumns**. Sử dụng phương pháp này thay vì bố cục **flexWrap** có thể ngăn xung đột với logic chiều cao mục.

  - Bằng cách truyền **extraData**={selectedId} vào **FlatList**, chúng ta muốn chính **FlatList** sẽ re-render lại khi state thay đổi. Nếu không đặt prop này, **FlatList** sẽ không biết nó cần phải hiển thị lại bất kỳ mục nào vì nó là **PureComponent** và so sánh prop sẽ không hiển thị bất kỳ thay đổi nào.

  - **keyExtractor** để thêm khoá **ids** cho từng tiem trong **FlatList** thay vì thuộc tính khóa mặc định.

FlatList component

- Các props sử dụng phổ biến trong **FlatList:**
  - **FlatList** kế thừa tất cả props có từ **ScrollView**
  - **renderItem** item trong list, bắt buộc thêm

```
renderItem({
  item: ItemT,
  index: number,
  separators: {
    highlight: () => void;
    unhighlight: () => void;
    updateProps: (select: 'leading' | 'trailing', newProps: any) => void;
  }
}): JSX.Element;
```

FlatList component

Lấy một mục từ dữ liệu và hiển thị nó vào danh sách.

Cung cấp siêu dữ liệu bổ sung như **index** (thứ tự của item, bắt đầu từ 0) nếu bạn cần, cũng như chức năng **separators.updateProp** cho phép bạn đặt bất kỳ props nào bạn muốn thay đổi kết xuất của dấu phân cách đầu hoặc dấu phân cách cuối trong trường hợp **highlight** và **unhighlight** phổ biến hơn (đặt tô sáng: **highlighted: boolean**) không đủ cho trường hợp sử dụng của bạn.

- **ListEmptyComponent** xuất hiện khi danh sách trống. Có thể là React Component (ví dụ: SomeComponent) hoặc phần tử React (ví dụ: <SomeComponent />).

FlatList component

- **ListFooterComponent** được hiển thị ở cuối tất cả các mục. Có thể là React Component (ví dụ: SomeComponent) hoặc phần tử React (ví dụ: <SomeComponent />).
- **ListHeaderComponent** được hiển thị ở đầu tất cả các mục. Có thể là React Component (ví dụ: SomeComponent) hoặc phần tử React (ví dụ: <SomeComponent \/>).
- **horizontal** nếu **true**, hãy hiển thị các mục cạnh nhau theo chiều ngang thay vì xếp chồng lên nhau theo chiều dọc.
- **initialNumToRender** số lượng item hiển thị trong lần render đầu tiên. Điều này là đủ để lấp đầy màn hình nhưng không nhiều hơn. Lưu ý rằng các mục này sẽ không bao giờ được **unmounted** như một phần của rendering window để cải thiện hiệu suất của các hành động cuộn lên trên cùng.

FlatList component

- **keyExtractor** được sử dụng để trích xuất một khóa duy nhất cho một item nhất định tại index được chỉ định. Khóa được sử dụng cho bộ nhớ đệm và làm phím phản ứng để theo dõi việc sắp xếp lại mục hàng. Trình trích xuất mặc định kiểm tra **item.key**, sau đó **item.id**, và sau đó quay trở lại sử dụng index, giống như React.
- **onRefresh** nếu được cung cấp, một **RefreshControl** tiêu chuẩn sẽ được thêm vào cho chức năng '**Pull to Refresh**'. Đảm bảo cũng đặt đạo cụ làm mới một cách chính xác.
- **refreshing** đặt prop này **true** trong khi chờ dữ liệu mới được làm mới.

FlatList component

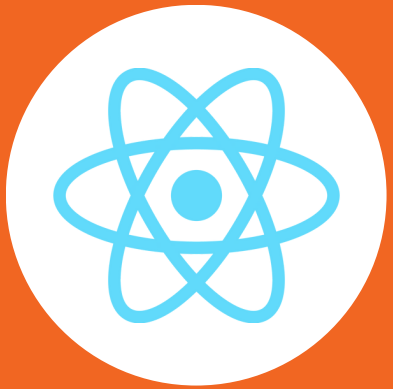
- Ví dụ sử dụng **FlatList**

```
const Item = ({title}: ItemProps) => (
  <View style={styles.item}>
    <Text style={styles.title}>{title}</Text>
  </View>
);

const FlatListComponent = () => {
  return (
    <FlatList
      data={DATA}
      renderItem={({item}) => <Item title={item.title} />}
      keyExtractor={item => item.id}
    />
  );
};
```

FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 4: CORE COMPONENT TRONG REACT NATIVE (P2)

PHẦN 2: CÁC THÀNH PHẦN CƠ BẢN

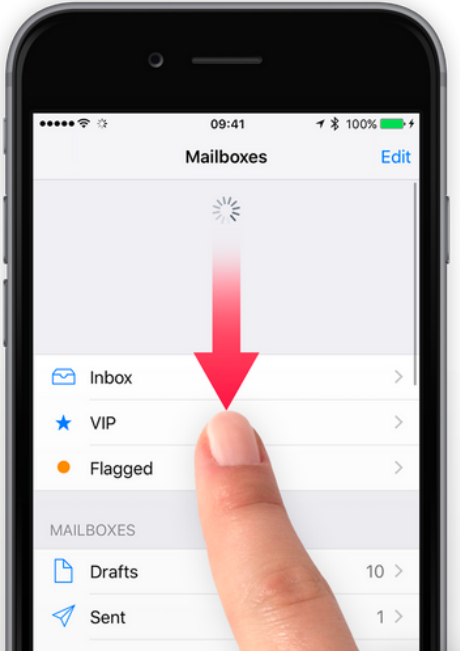
Mục tiêu

- Giới thiệu về **<RefeshControl/>**
- Giới thiệu về **<StatusBar/>**
- Hiểu về cách sử dụng **<KeyboardAvoidingView/>**

RefeshControl component

- **RefreshControl** thường được sử dụng cùng với **FlatList** hoặc **ScrollView.** Nó sẽ giống một icon loading.

Bạn đôi khi sẽ cần load lại trang, để tải những dữ liệu mới. Bạn vuốt xuống ở đầu trang để tải lại



RefeshControl component

- Các prop sử dụng phổ biến trong **RefeshControl**
  - **refreshing:** khi **true,** sẽ hiện **RefeshControl**
  - **tintColor:** Màu của reload icon **(ios)**
  - **title:** Tiêu đề hiển thị dưới icon reload **(ios)**
  - **colors:** màu của reload icon (**android**)

RefeshControl component

- Bây giờ chúng ta sẽ đi vào ví dụ, tạo một **RefeshControl.** Cách sử dụng **RefeshControl** của **ScrollView** và **FlatList** tương tự nhau, thế nên phần demo này sẽ chỉ dùng **ScrollView**

```
<ScrollView
  contentContainerStyle={styles.scrollView}
  refreshControl={
    <RefreshControl refreshing={refreshing} onRefresh={onRefresh} />
  }>
  <Text>Kéo xuống để thấy RefreshControl icon</Text>
</ScrollView>
```

RefeshControl component

- Kết quả, chúng ta sẽ được giao diện như này:

StatusBar component

- **StatusBar** thành phần để điều khiển thanh trạng thái của ứng dụng. Thanh trạng thái là vùng, thường ở đầu màn hình, hiển thị thời gian hiện tại, Wi-Fi và thông tin mạng di động, mức pin hoặc các biểu tượng trạng thái khác.
- Bạn sẽ muốn đổi màu **StatusBar** để cho màu phù hợp với màu ứng dụng của bạn. Hoặc những trường hợp bạn thay đổi theme ứng dụng

StatusBar component

- Các prop sử dụng phổ biến trong **StatusBar**
  - **barStyle** đặt màu của văn bản trên thanh trạng thái. Trên Android, điều này sẽ chỉ có hoạt động đến API phiên bản 23 trở lên.
  - **backgroundColor** màu nền của thanh trạng thái
  - **translucent** nếu thanh trạng thái mờ. Khi **translucent** được đặt thành true, ứng dụng sẽ vẽ dưới thanh trạng thái. Điều này rất hữu ích khi sử dụng màu thanh trạng thái bán trong suốt.
  - **animated** nếu quá trình chuyển đổi giữa các thay đổi thuộc tính trên thanh trạng thái sẽ được tạo hiệu ứng. Hỗ trợ cho **backgroundColor**, **barStyle** và các thuộc tính ẩn.

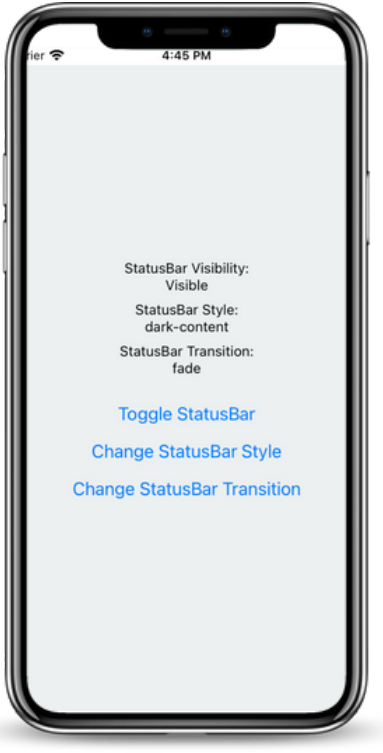
StatusBar component

- Chúng ta sẽ đi vào ví dụ, tạo một **StatusBar.** StatusBar này có thể đặt bất kỳ đâu trong ứng dụng, nhưng tốt nhất nên để ở phần root của ứng dụng để ứng dụng của bạn được đồng bộ.

```
<StatusBar
  animated={true}
  backgroundColor="#61dafb"
  barStyle={statusBarStyle}
  showHideTransition={statusBarTransition}
  hidden={hidden}
/>
```

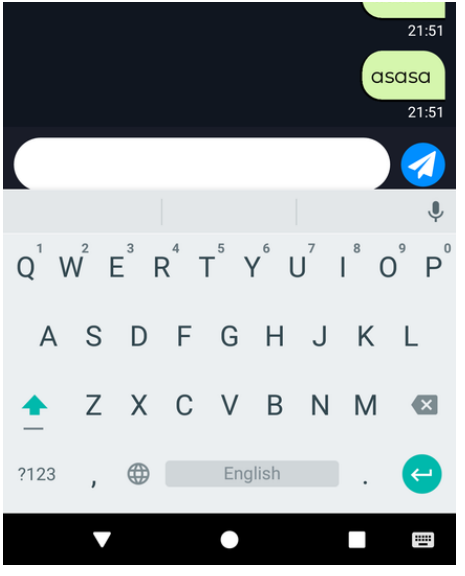
StatusBar component

- Chạy chương trình ta có kết quả sau:



KeyboardAvoidingView component

- Khi bạn nhấn vào **TextInput** bàn phím sẽ hiện lên, và nó có thể sẽ che **TextInput** của bạn. Để component của bạn không bi che khuất bởi bàn phím, bạn sử dụng **KeyboardAvoidingView**



KeyboardAvoidingView component

- Các prop sử dụng phổ biến trong **KeyboardAvoidingView**
  - **behavior** chỉ định cách phản ứng với sự hiện diện của bàn phím.

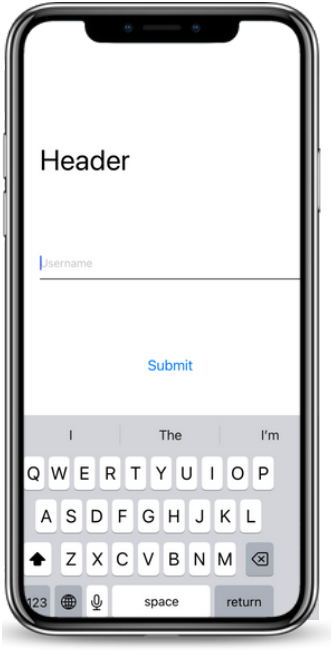
    enum('height', 'position', 'padding')
  - **contentContainerStyle** kiểu của vùng chứa nội dung (View) khi hành vi là '**position**'.
  - **keyboardVerticalOffset** đây là khoảng cách giữa đầu màn hình người dùng và chế độ xem react native, có thể không bằng không trong một số trường hợp sử dụng.

KeyboardAvoidingView component

- Chúng ta sẽ đi vào ví dụ, tạo một **KeyboardAvoidingView**

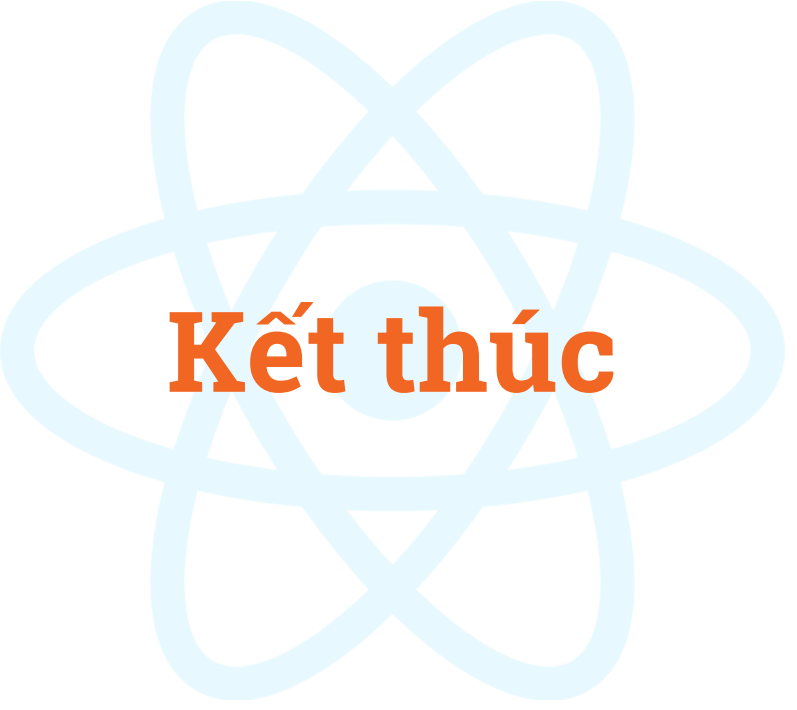
```
<KeyboardAvoidingView
  behavior={Platform.OS === 'ios' ? 'padding' : 'height'}
  style={styles.container}>
  <TouchableWithoutFeedback onPress={Keyboard.dismiss}>…
  </TouchableWithoutFeedback>
</KeyboardAvoidingView>
```

- Chạy ứng dụng chúng ta có kết quả như sau, bạn có thể thấy component của chúng ta không bị che khuất



Tổng kết

- Giới thiệu về **<ScrollView />**
- Hiển thị danh sách với **<FlatList/>**
- Giới thiệu về **<RefeshControl/>**
- Giới thiệu về **<StatusBar/>**
- Hiểu về cách sử dụng **<KeyboardAvoidingView/>**



Kết thúc