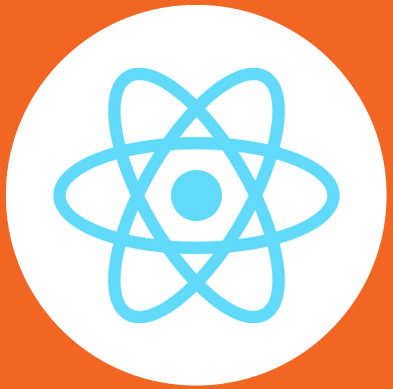
FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 5: GIỚI THIỆU VỀ STYLE COMPONENT TRONG REACT NATIVE

PHẦN 1: STYLE COMPONENT CƠ BẢN

Mục tiêu

- Giới thiệu về style component trong React Native
- Tìm hiểu về kích thước và lấy độ dài màn hình
- Giới thiệu về shadow
- Thêm font chữ vào ứng dụng

Giới thiệu

- Với React Native, bạn tạo kiểu cho ứng dụng của mình bằng JavaScript. Tất cả các **core component** đều chấp nhận một đạo cụ có tên **style**. Tên và giá trị kiểu thường khớp với cách CSS hoạt động trên web, ngoại trừ tên được viết bằng **camel** casing, ví dụ: **backgroundColor** thay vì **background-color**
- **Style** prop có thể là một object JavaScript. Đó là những gì chúng ta thường sử dụng cho các ví dụ. Bạn cũng có thể truyền một mảng các style, style cuối cùng trong mảng được ưu tiên, vì vậy bạn có thể sử dụng kiểu này để kế thừa các style.
- Khi một thành phần phát triển phức tạp, thường clean code hơn khi sử dụng **StyleSheet.create** để xác định một số kiểu ở một nơi.

- Đây là ví dụ:

```
const styles = StyleSheet.create({
  container: {
    marginTop: 50,
  },
  bigBlue: {
    color: 'blue',
    fontWeight: 'bold',
    fontSize: 30,
  },
  red: {
    color: 'red',
  },
});
```

Kích thước

- Trong React Native, bạn có thể sử dụng một số đơn vị tính khác nhau để xác định kích thước và khoảng cách giữa các phần tử trên giao diện người dùng. Các đơn vị thông dụng bao gồm:

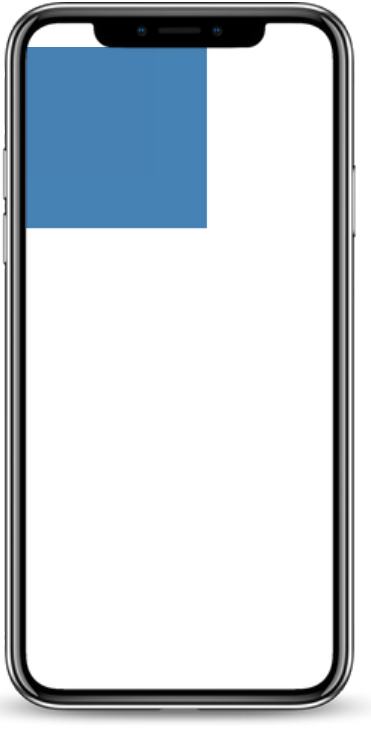
  - **Fixed Dimensions**

    Cách chung để đặt kích thước của một thành phần là thêm **width** và **height** cố định vào style. Tất cả các kích thước trong React Native được tính là **px**

Kích thước

- Ví dụ style inline:

```
<View
      style={{
        width: 150,
        height: 150,
        backgroundColor: 'steelblue',
      }}
/>
```



Percentage Dimensions

Nếu bạn muốn lấp đầy một phần nhất định của màn hình, nhưng bạn không muốn sử dụng bố cục **flex**, bạn có thể sử dụng các giá trị phần trăm theo kiểu của thành phần. Tương tự như kích thước flex, kích thước phần trăm yêu cầu cha có kích thước xác định.

```
<View
    style={{
      width: '33%',
      height: '50%',
      backgroundColor: 'steelblue',
    }}
/>
```

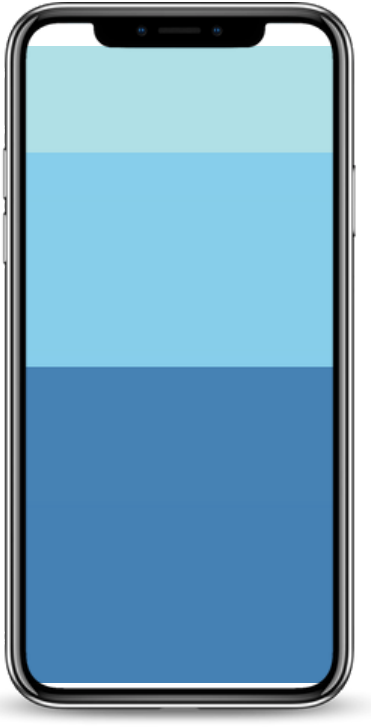
Flex Dimensions

Sử dụng flex theo kiểu của một thành phần để thành phần mở rộng và thu nhỏ động dựa trên không gian có sẵn. Thông thường bạn sẽ sử dụng flex: 1, cho biết một thành phần lấp đầy tất cả không gian có sẵn, được chia đều giữa các thành phần khác có cùng parent component. Độ **flex** được đưa ra càng lớn, thì tỉ lệ bao phủ của component càng lớn.

Kích thước

- Ví dụ

```
<View style={{flex: 1}}>
    <View style={{flex: 1,
backgroundColor: 'powderblue'}} />
    <View style={{flex: 2,
backgroundColor: 'skyblue'}} />
    <View style={{flex: 3,
backgroundColor: 'steelblue'}} />
 </View>
```



Dimensions

- **Dimensions** dùng để lấy kích thước của màn hình, cập nhật khi kích thước của màn hình cập nhật.
- Bạn có thể lấy chiều rộng và chiều cao của cửa sổ ứng dụng bằng cách sử dụng mã sau:

```
const windowWidth = Dimensions.get('window').width;
const windowHeight = Dimensions.get('window').height;
```

Dimensions

- Các phương thức của **Dimensions**
  - **addEventListener()**

```
static addEventListener(
  type: 'change',
  handler: ({
    window,
    screen,
  }: DimensionsValue) => void,
): EmitterSubscription;
```

Thêm một trình xử lý sự kiện. Các sự kiện được hỗ trợ:

- **change**: Kích hoạt khi một thuộc tính trong đối tượng Dimensions thay đổi. Đối số cho trình xử lý sự kiện là một đối tượng kiểu **DimensionsValue**.

get()

```
static get(dim: 'window' | 'screen'): ScaledSize;
```

Kích thước ban đầu được chạy trước khi runApplication được gọi vì vậy chúng sẽ có trước khi bất kỳ yêu cầu nào khác được chạy, nhưng có thể được cập nhật sau.

Ví dụ: const {height, width} = Dimensions.get('window');

Type Definitions

DimensionsValue

Tên	Kiểu dữ liệu	Mô tả
window	ScaledSize	Kích thước của màn hình ứng dụng
screen	ScaledSize	Kích thước của màn hình ứng dụng. Nhưng với thiết bị **Android** sẽ được cộng thêm chiều cao phần **navigation bar**, nếu có

Dimensions

ScaledSize

Tên	Kiểu dữ liệu
width	number
height	number
scale	number
fontScale	number

Shadow

- Đôi khi ứng dụng của bạn sẽ cần style thêm **shadow** vào component. Ở ví dụ bên dưới, nút login đã được thêm shadow màu xanh

Shadow

- Để làm được một nút có shadow như ví dụ bên trên, chúng ta sẽ đi vào tìm hiểu các **props** style tạo shadow trong React Native

  - **shadowColor**

    Dùng để đặt màu cho shadow

    Thuộc tính này sẽ chỉ hoạt động trên Android API 28 trở lên. Đối với chức năng tương tự trên API Android thấp hơn, hãy sử dụng thuộc tính elevation.

  - **shadowOffset <IOS>**

    Đặt độ lệch bóng đổ

    Kiểu dữ liệu truyền vào:
    object: {width: number,height: number}

Shadow

- **shadowOpacity <IOS>**

  Đặt độ mờ đổ bóng (nhân với thành phần alpha của màu).

- **shadowRadius <IOS>**

  Đặt bán kính làm mờ bóng đổ

Shadow

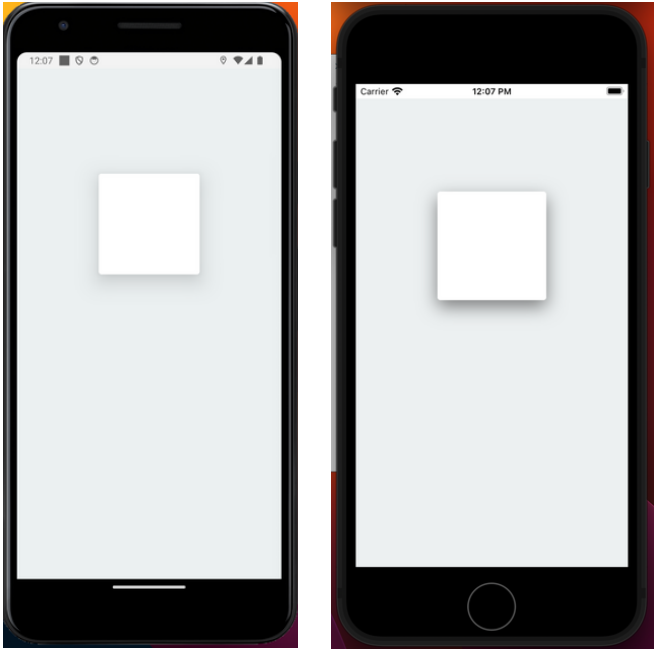
- Chúng ta sẽ đi vào ví dụ, tạo một shadow cho **View** component

```
shadowColor: '#000',
shadowOffset: {
width: 0,
height: 9,
},
shadowOpacity: 0.48,
shadowRadius: 11.95,

elevation: 18,
```

Bạn cũng có thể truy cập trang web sau, để xem căn chỉnh thuộc tính shadow **https://ethercreative.github.io/react-native-shadow-generator/**

Kết quả



- Bạn có thể thấy shadow giữa thiết bị **Android** và **IOS** sẽ khác nhau đôi chút. Nên bạn hãy cẩn thận thiết kế shadow cho ứng dụng của mình nhé

Font

- **Font** là các khối xây dựng của trải nghiệm người dùng tuyệt vời và giúp bạn làm cho ứng dụng của mình nổi bật trong một thị trường cạnh tranh.
- Ở phần này, các bạn sẽ được hướng dẫn cách thêm **font** chữ vào ứng dụng của mình.
- Có khá nhiều các loại kiểu font chữ khác nhau như **.otf .ttf...,** nhưng kiểu chữ phổ biến được sử dụng là **.tff**
- Để font chữ hoạt động, bạn cần add các font chữ bạn vào phần native **android** và **ios** để nó hoạt động trên 2 nền tảng này

- **Bước 1**: Thêm **font** vào ứng dụng của bạn
  - Đây là một quá trình khá đơn giản, vì chúng ta có nhiều nguồn khác nhau để tải các phông chữ trên Internet. **Google Fonts** là một ví dụ điển hình về một nguồn tài nguyên phổ biến của phông chữ mã nguồn mở và miễn phí. Bạn có thể xem và tải xuống bất kỳ phông chữ nào bạn thích ở định dạng **.ttf**.
  - Sau khi tải xuống **font** chữ bạn muốn, hãy tạo thư mục fonts chữ bên dưới **assets** trong dự án của bạn. Đường dẫn nên như thế này:

```
PROJECT-DIRECTORY/assets/fonts
```

Di chuyển các file font **.ttf** mà bạn đã tải xuống thư mục **fonts** này.

- **Bước 2**: Thêm config file vào project

Nếu chưa được tạo, hãy tạo một file config ở root project của bạn có tên **react-native.config.js**. Tiếp tục bằng cách thêm đoạn mã sau vào bên trong **module.exports**:

```
module.exports = {
  project: {
    ios:{},
    android:{}
  },
  assets:['./assets/fonts/'],
}
```

- **Bước 3**: Link các font chữ của bạn đến phần native của project

Sau khi các bước trên đã được thực hiện, chúng ta chỉ cần chạy một lệnh để liên kết các phông chữ mà chúng ta vừa thêm.

```
$ npx react-native-asset
```

Vậy là xong! Sau lệnh này, chúng ta có thể thấy các phông chữ chúng ta đã thêm trong thư mục **android/app/ src/main/assets/ fonts** và trong **Info.plist** (cho **Android** và **iOS** tương ứng).

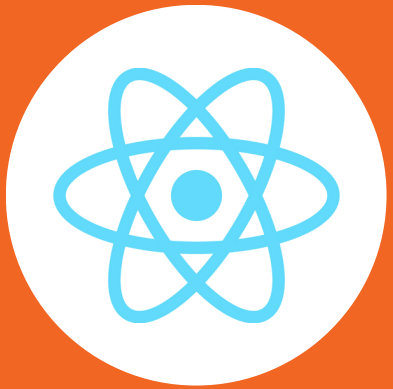
Font

- Đã đến lúc tận hưởng thành quả của chúng ta, bây giờ chúng ta sẽ sử dụng font chữ của mình nhé

```
contentStyle: {
    fontFamily: 'PlaypenSans-Bold',
    fontSize: 20,
},
```

FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 5: GIỚI THIỆU VỀ STYLE COMPONENT TRONG REACT NATIVE

PHẦN 2: FLEXBOX TRONG REACT NATIVE

Mục tiêu

- Giới thiệu về **flex**
- Giới thiệu về **flex direction**
- Giới thiệu về **layout direction**
- Giới thiệu về **justify content**
- Giới thiệu về **align items**
- Giới thiệu về **align self**
- Giới thiệu về **flex wrap**
- Giới thiệu về **flex basis, grow, và shrink**

Các đặc tính của flexbox là gì?

Có một vài thuộc tính **flexbox** bạn sẽ sử dụng thường xuyên đáng để ghi nhớ. Dưới đây các bạn sẽ được giới thiệu đặc tính cơ bản và những gì **flexbox** làm được.

- **flex:** xác định mức độ xem sẽ lấp đầy màn hình. Giá trị khả dụng là các số nguyên lớn hơn hoặc bằng 0.
- **flexDirection:** xác định các component con được bố trí theo hướng nào — theo chiều dọc hoặc chiều ngang. Các giá trị được cung cấp bao gồm **column**, **row**, **column-reverse**, và **row-reverse**

- **justifyContent:** xác định vị trí các component bên trong của nó dọc theo trục y (được xác định bởi thuộc tính **flexDirection**). Các giá trị có sẵn là **flex-start**, **flex-end**, **center**, **space-between**, **space-around** và **space-evenly**
- **alignItems:** xác định vị trí các component bên trong của nó dọc theo trục x (được xác định bởi thuộc tính **flexDirection**). Các giá trị có sẵn là **flex-start**, **flex-end**, **center** và **baseline**
- **alignSelf** xác định vị trí của chính nó và ghi đè lên **alignItems**. Các giá trị có sẵn là **flex-start**, **flex-end**, **center** và **baseline**
- **flexWrap** xác định điều gì sẽ xảy ra khi con của container tràn ra ngoài vùng chứa. Theo mặc định, chúng buộc phải phù hợp với một dòng duy nhất, do đó thu nhỏ chúng.

- Làm cách nào để sử dụng thuộc tính **flex** trong React Native?

Thuộc tính **flex** xác định cách chế độ xem lấp đầy màn hình. Để minh họa, chúng ta hãy xem hai ví dụ dưới đây.

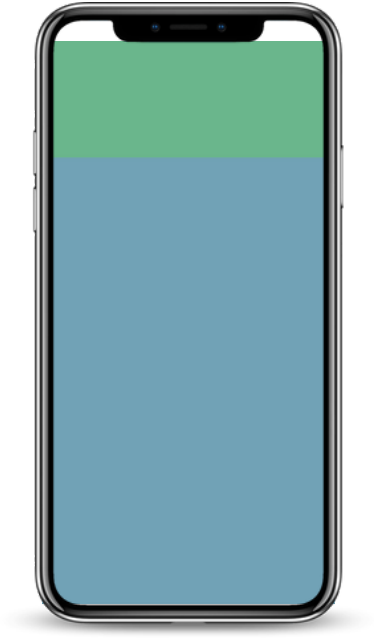
Lưu ý rằng các giá trị của **flex** bao gồm các số nguyên lớn hơn hoặc bằng 0:

```
<View style={{ backgroundColor: "#7cb48f", flex: 1 }} />
```

- Hãy đến với ví dụ thứ 2:

```
<View style={{ backgroundColor: "#7cb48f", flex: 1 }} />
<View style={{ backgroundColor: "#7CA1B4", flex: 3 }} />
```

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



Chúng ta có thể thấy rằng khu vực màu xanh lá cây hiện chỉ chiếm một phần tư màn hình mặc dù có cùng giá trị **flex**.

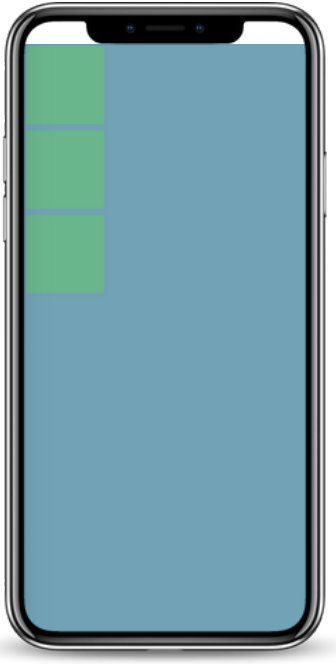
Đó là bởi vì màn hình hiện được chia thành bốn khối (1 + 3), vì vậy thuộc tính **flex** là phần nhỏ của màn hình mà nó sẽ chiếm.

flexDirection

- **FlexDirection** xác định hướng các component con bên trong nó. **flexDirection** cung cấp các giá trị sau **column**, **row**, **column-reverse** và **row-reverse**, nhưng mặc định là **column**:
- Dưới đây là đoạn code mẫu, các bạn sẽ cần style **flexDirection** vào style tên là **container**

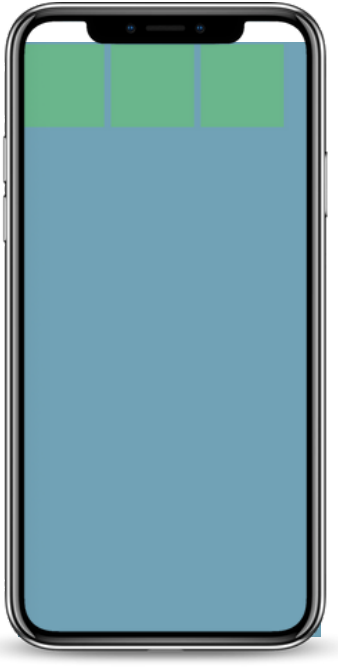
```
<View style={styles.container}>
   <View style={styles.square} />
   <View style={styles.square} />
   <View style={styles.square} />
</View>
```

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



flexDirection

- Bây giờ, hãy thay đổi hướng từ **column** thành **row** bằng cách sử dụng **flexDirection** của **row**:



justifyContent

- **justifyContent** xác định vị trí các component bên trong của nó dọc theo trục y (được xác định bởi thuộc tính **flexDirection**). Các giá trị có sẵn là **flex-start**, **flex-end**, **center**, **space-between**, **space-around** và **space-evenly**
- Dưới đây sẽ là một số ví dụ sử dụng thuộc tính **justifyContent**.

justifyContent

- Dưới đây là các thuộc tính cơ bản của **justifyContent**

center | space-around | space-between | space-evenly

alignItems

- **alignItems** xác định vị trí các component bên trong của nó dọc theo trục x (được xác định bởi thuộc tính **flexDirection**). Các giá trị có sẵn là **flex-start**, **flex-end**, **center** và **baseline**
- Đây là nghịch đảo của **justifyContent**. Vì vậy, nếu **justifyContent** xử lý căn chỉnh dọc, thì **alignItems** xử lý căn chỉnh ngang.

alignItems

- Dưới đây là các thuộc tính cơ bản của **alignItems**

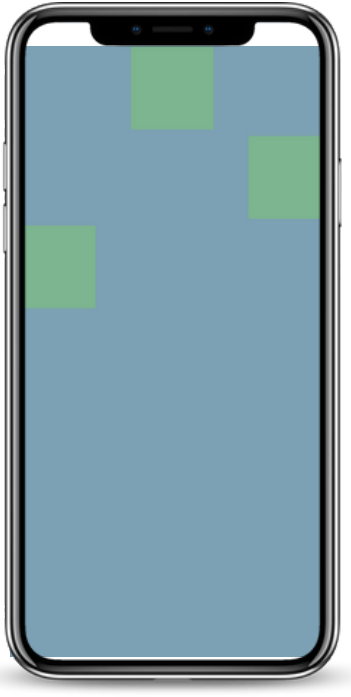
center flex-start space-between

alignSelf

- **alignSelf** xác định vị trí của chính nó và ghi đè lên **alignItems**. Các giá trị có sẵn là **flex-start**, **flex-end**, **center** và **baseline**

```
<View style={styles.container}>
   <View style={[styles.square, {alignSelf: 'center'}]} />
   <View style={[styles.square, {alignSelf: 'flex-end'}]} />
   <View style={[styles.square, {alignSelf: 'flex-start'}]} />
</View>
```

- Reload lại app chúng ta có kết quả như sau:

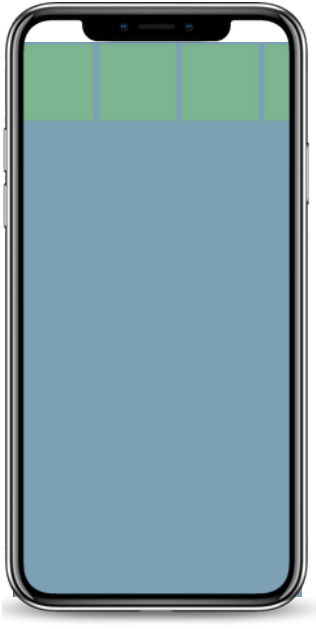


flexWrap

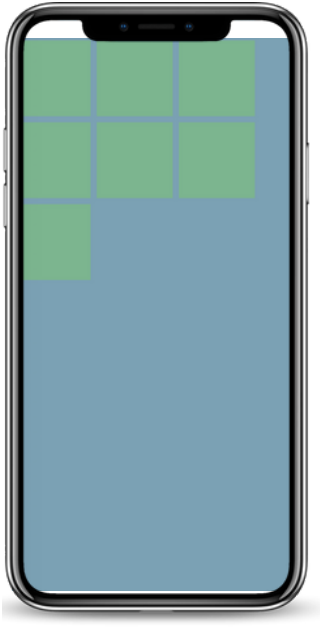
- Khi children component của một container tràn ra khỏi nó, **flexWrap** xác định xem chúng nên được thu nhỏ trên một dòng hay được bọc thành nhiều dòng. Các giá trị có sẵn cho **flexWrap** là **nowrap** và **wrap**.

```
<View style={styles.container}>
    <View style={styles.square} />
    <View style={styles.square} />
    <View style={styles.square} />
    <View style={styles.square} />
    <View style={styles.square} />
    <View style={styles.square} />
    <View style={styles.square} />
 </View>
```

- Bạn hãy xem điều kì diệu xảy ra ở dưới nhé:



nowrap



wrap

- Tiếp theo, các bạn sẽ được tìm hiểu thêm 2 thuộc tính flex nữa. Đó là **flexGrow** và **flexShrink** đây là 2 thuộc tính style rất hữu dụng, để tìm hiểu cách sử dụng nó chúng ta bắt đầu bằng một số ví dụ bên dưới nhé.

- **Flex Grow**

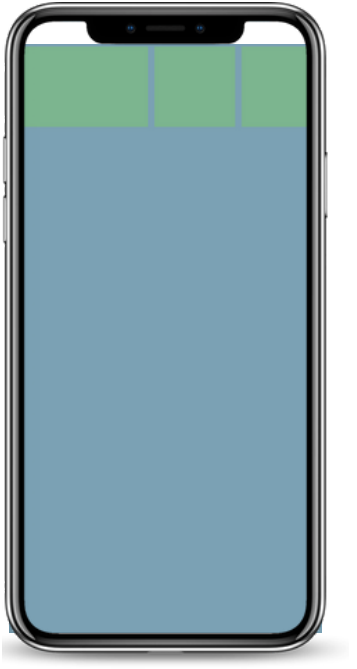
  flexGrow định nghĩa xem item sẽ được dãn ra bao nhiêu để lấp vào khoảng trống của view cha. Ví dụ rằng có 3 view như sau, tổng chiều rộng nhỏ hơn view cha

flexGrow

- Dưới đây là ví dụ

```
<View style={styles.container}>
   <View style={[styles.square, {flexGrow: 1}]} />
   <View style={styles.square} />
   <View style={styles.square} />
</View>
```

- Reload lại ứng dụng chúng ta có kết quả như sau:



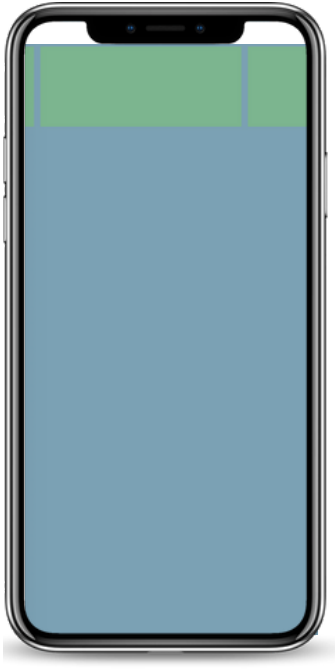
flexShrink

- Thuộc tính này có phần ngược lại so với **flexGrow**, nếu như tổng các item lớn hơn view cha, thì thuộc tính này sẽ định nghĩa tỉ lệ bị trừ đi.
- Dưới đây là ví dụ:

```
<View style={styles.container}>
   <View style={[styles.square, {flexShrink: 1}]} />
   <View style={[styles.square, {width: 250}]} />
   <View style={styles.square} />
</View>
```

flexShrink

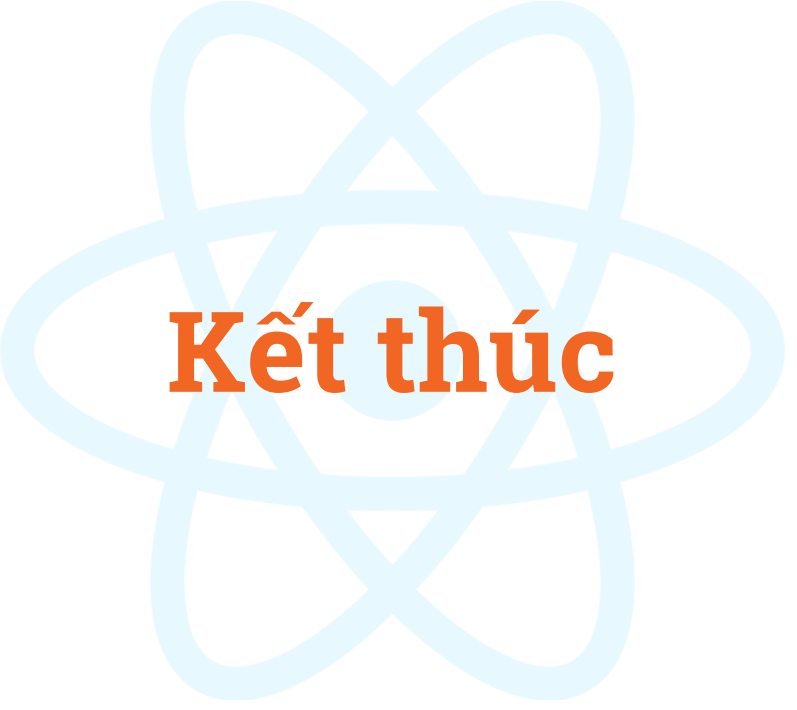
- Reload lại ứng dụng chúng ta có kết quả như sau:



Ở block đầu tiên, có thuộc tính **flexShrink: 1,** bởi vì block 2 có độ dài vượt quá screen, nên ở block đầu tiên bị trừ đi bớt độ dài, để block 2 hiện đầy đủ kích thước của nó.

Tổng kết

- [ ] Giới thiệu về style component trong React Native
- [ ] Tìm hiểu về kích thước và lấy độ dài màn hình
- [ ] Giới thiệu về shadow
- [ ] Thêm font chữ vào ứng dụng
- [ ] Giới thiệu về flexbox



Kết thúc