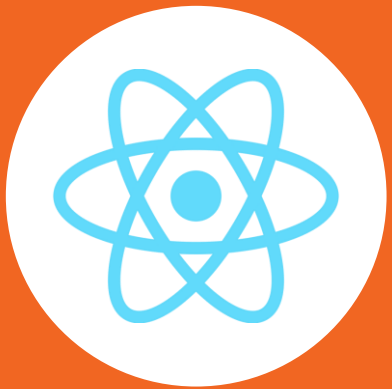
FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 1: GIỚI THIỆU VỀ CUSTOM COMPONENT TRONG REACT NATIVE

PHẦN 1: TẦM QUAN TRỌNG, VÀ CÁCH XÂY DỰNG CUSTOM COMPONENT

Mục tiêu

- Hiểu rõ tầm quan trọng và lợi ích khi tạo custom component
- Các bước để tạo một custom component
- Tạo **<Header />** component

Custom component là gì?

Trong React Native, **"custom component"** là một thành phần được tạo ra để tái sử dụng trong ứng dụng. Các thành phần này được xây dựng bằng cách kết hợp các thành phần có sẵn từ React Native để tạo ra một thành phần mới có thể thực hiện một số chức năng cụ thể hoặc có giao diện người dùng đặc trưng mà thành các core component không thể làm được.

- Tại sao custom component lại quan trọng!

  Trong quá trình phát triển thực tế ứng dụng React Native, sẽ có rất nhiều thành phần được lặp đi lặp lại nhiều lần. Việc phải viết lại các thành phần này rất tốn thời gian xây dựng và kể cả trong quá trình bảo trì, sửa lỗi cũng rất mất thời gian. Việc tạo một custom component cho một thành phần cụ thể nào đó, giúp dễ dàng quản lý các thành phần hơn.

  Custom component là một kỹ thuật không thể thiếu trong tất cả các ứng dụng được phát triển bằng React Native, tận dụng được sức mạnh của framework này mang lại

- Xây dựng custom component **Header.**

Trong Header custom component sẽ có các props sau:

- **title**: tiêu đề của **Header** component
- **iconLeft**: là đường dẫn đến icon phía bên trái
- **iconRight**: là đường dẫn đến icon phía bên phải
- **onPressRight**: là một prop function, function sẽ được kích hoạt khi người dùng nhấn vào vùng icon bên phải
- **onPressLeft**: là một prop function, function sẽ được kích hoạt khi người dùng nhấn vào vùng icon bên trái
- **leftComponent**: là một component tuỳ chỉnh thay thế cho icon mặc định bên trái của **Header.**

- **centerComponent**: là một component tuỳ chỉnh thay thế tiêu đề mặc định ở giữa của **Header,** prop này giúp component mang tính tuỳ chỉnh cho nhiều trường hợp hơn.
- **rightComponent**: là một component tuỳ chỉnh thay thế cho icon mặc định bên phải của **Header,** prop này giúp component mang tính tuỳ chỉnh cho nhiều trường hợp hơn.
- Các prop còn lại là các giá trị tuỳ chỉnh style cho **Header** component.

Header

- Một **Header** component, sẽ được chia ra làm 3 phành phần nhỏ **renderLeft, renderCenter, renderRight.**

```
<View style={[styles.container]}>
  {renderLeft()}
  {renderCenter()}
  {renderRight()}
</View>
```

- **renderLeft** function chịu trách nhiệm render thành phần bên trái của **Header,** nếu bạn có truyền thành phần **leftComponent** thì nó sẽ render nó, điều này giúp bạn tuỳ chỉnh, thêm bất kỳ thành phần nào mà bạn muốn.

```
const renderLeft = () => {
  return (
    leftComponent || (
      <View>
        {iconLeft ? (<Pressable>...</Pressable>) : (
          <View style={{width: leftIconSize, height: leftIconSize}} />
        )}
      </View>
    )
  );
};
```

Header

- Hiển thị icon bên trái của **Header,** là thành phần được sử dụng nhiều, nên chúng ta viết sẵn một image component để hiển thị icon từ prop iconLeft được truyền vào.

```
iconLeft ? (
   <Pressable hitSlop={15} onPress={onPressLeft || Navigator.goback}>
     <Image
       source={iconLeft}
       width={leftIconSize}
       height={leftIconSize}
       resizeMode={'contain'}
       tintColor={iconLeftColor}
     />
   </Pressable>
 )
```

- **renderCenter** hàm render component ở giữa, nếu bạn muốn tuỳ chỉnh giao diện thì truyền prop **centerComponent**, không thì bạn có thể tuyền prop title để đặt tên cho **Header.**

```
const renderCenter = () => {
  return (
    centerComponent || (
      <View style={styles.containerCenter}>
        <Text style={styles.title} numberOfLines={numberOfLines}>
          {title}
        </Text>
      </View>
    )
  );
};
```

- **renderRight** có các thành phần tương tự như **renderLeft**, chỉ thay bằng các prop khác như **iconRight, rightIconSize,...**

```
const renderRight = () => {
  return (
    rightComponent || (
      <View style={styles.containerRight}>
        {iconRight ? (
          <Pressable hitSlop={15} onPress={onPressRight}>
            ...
          </Pressable>
        ) : (
          <View style={{width: rightIconSize, height: rightIconSize}} />
        )}
      </View>
))};
```

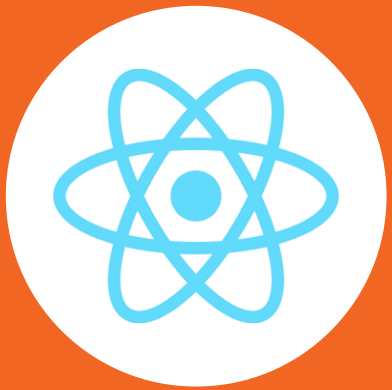
Header

- Bây giờ, bạn có thể sử dụng custom component **Header** đã được viết. Ngoài ra bạn có thể sử dụng nhiều hơn các prop có trong **Header.**

```
<Header
  title="Đây là tiêu đề"
  iconLeft={{
    uri: 'https://cdn-icons-png.flaticon.com/512/271/271220.png',
  }}
  rightComponent={customRightHeader()}
/>
```

FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 1: GIỚI THIỆU VỀ CUSTOM COMPONENT TRONG REACT NATIVE

PHẦN 2: CUSTOM COMPONENT VIEW VÀ SECTION VIEW

https://caodang.fpt.edu.vn/

Mục tiêu

- Tuỳ chỉnh core component **<View />** mang các thuộc tính style ra ngoài prop.
- Tạo **<SectionView />** custom component, hiện thị dữ liệu dạng nhiều thành phần

View component

- Xây dựng custom component **<View />**
- Để có thể style cho **<View />** component phải **style inline** hoặc truyền style qua **StyleSheet**:

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

```
<View
    style={{
        backgroundColor: 'red',
        height: 200,
        width: 100,
        borderColor: 'black',
    }}
/>
```

View component

- Sau khi xây dựng custom component **<View />,** component sẽ được sử dụng như sau:

```
<View
  height={200}
  width={200}
  margin={10}
  bg={"gray"}
  radius={20}
  justifyCenter
  alignCenter>
  <Text>Sử dụng View custom component bọc Text</Text>
</View>
```

View component

- Khai báo một số props cho custom component **<View />**
  - **bg**: xác định màu nền cho thành thành.
  - **row**: hướng sắp của các thành phần bên trong (mặc định thành phần được sắp xếp theo dọc).
  - **radius:** độ cong của thành phần.
  - **margin:** căn lề ngoài thành phần.
  - **padding:** căn lề bên trong thành phần.
  - **height:** chiều cao của ứng dụng.
  - **width:** chiều rộng của ứng dụng.

View component

- Tạo một danh sách **viewStyle** định nghĩa các giá trị style từ prop nhận được

```
const viewStyle = [
  abs && styles.absolute,
  row && styles.row,
  bg && {backgroundColor: bg},
  radius && {borderRadius: radius},
  height && {height},
  width && {width},
  margin && {margin},
  padding && {padding},
  {...StyleSheet.flatten(style)},
];
```

- Style chi tiết cho từng thuộc tính props nhận được:

styles.absolute

```
absolute: {
    position: 'absolute',
}
```

styles.absolute

```
row: {
    flexDirection: 'row',
}
```

StyleSheet.flatten()

```
static flatten(style: Object[]): Object;
```

StyleSheet.flatten có chức năng gộp một mảng các đối tượng style, thành một đối tượng style tổng hợp. Ngoài ra, phương pháp này có thể được sử dụng để tra cứu ID, được trả về bởi **StyleSheet.register**.

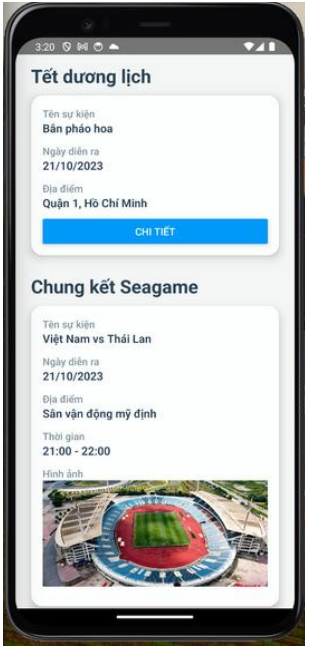
View component

- Return custom component **<View />**

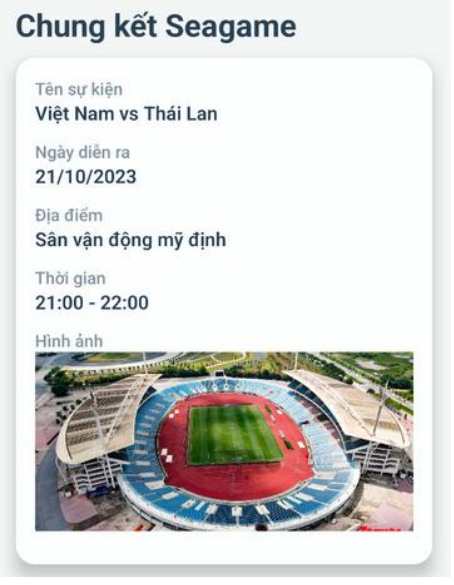
```
return <View {...rest} ref={ref} style={viewStyle}
```

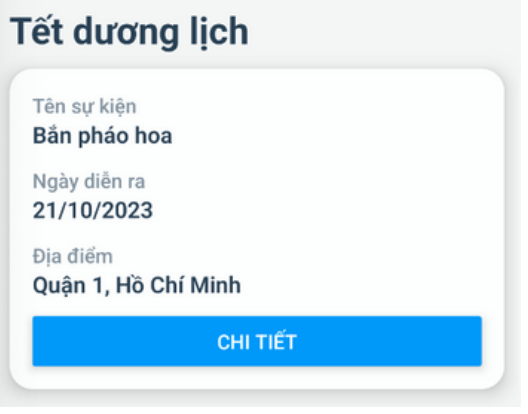
Section component

- Xây dựng custom component **Section component**

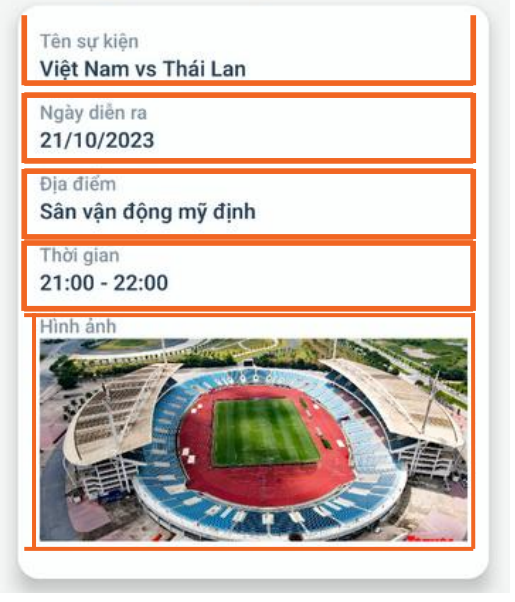


- **Section component** bao gồm 2 thành phần chính:
  - Phần 1: Các item render ra thành phần chính gồm **tiêu đề** và **card**.





- Phần 2: Các item render ra nội dung trong card



Section component

- Tương ứng với mỗi component, cần khai báo các prop cho **Section** component.
  - Phần 1: Các item render ra thành phần chính gồm tiêu đề và card.

```
const renderSection = (data, index) => {
  const {title, events, titleStyle} = data;
```

- Phần 2: Các item render ra nội dung trong card

```
const renderChild = (data, index) => {
  const {
    content,
    contentStyle,
    titleStyle,
    title,
    contentComponent,
    eventComponent,
  } = data;
```

Section component

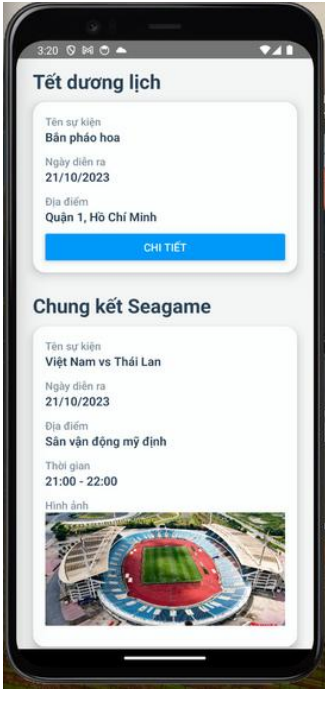
- Phần 1: Xây dựng function **renderSection**, bao gồm title và nội dung trong card

```
const renderSection = (data, index) => {
  const {title, events, titleStyle} = data;
  return (
    <View key={index.toString()} style={[styles.section]}>
      <Text style={[styles.titleSection, titleStyle]}>{title}</Text>
      <View style={[styles.sectionBody, styles.shadow]}>
        {events?.map(renderChild)}
      </View>
    </View>
  );
};
```

- Phần 2: Xây dựng function **renderChild**, hiển thị nội dung trong card.

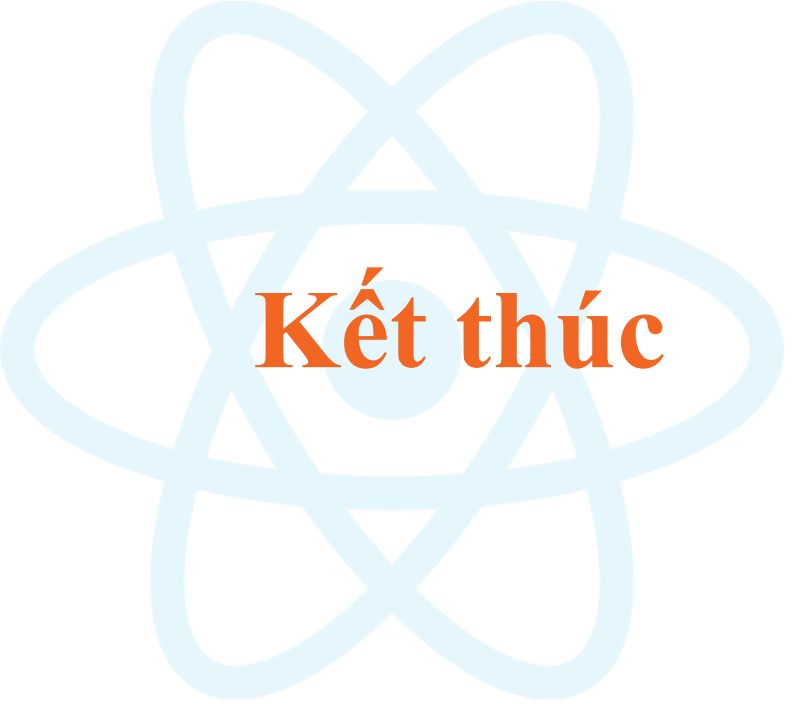
```
const renderChild = (data, index) => {
  const {...} = data;
  return (
<View key={index.toString()}>
      {eventComponent || (
        <View style={styles.containerChild}>
          <Text style={[styles.titleChild, titleStyle]}>{title}</Text>
          {contentComponent || (
            <Text style={[styles.contentChild, contentStyle]}>{content}</Text>
          )}
        </View>
      )} </View>
)};
```

Kết quả


3:20
Tết dương lịch
Tên sự kiện
Bắn pháo hoa
Ngày diễn ra
21/10/2023
Địa điểm
Quận 1, Hồ Chí Minh
CHI TIẾT
Chung kết Seagame
Tên sự kiện
Việt Nam vs Thái Lan
Ngày diễn ra
21/10/2023
Địa điểm
Sân vận động mỹ đình
Thời gian
21:00 - 22:00
Hình ảnh

Tổng kết

- Hiểu rõ tầm quan trọng và lợi ích khi tạo custom component
- Các bước để tạo một custom component
- Tạo **<Header />** component
- Tuỳ chỉnh core component **<View />** mang các thuộc tính style ra ngoài prop.
- Tạo **<SectionView />** component, hiện thị dữ liệu dạng nhiều thành phần



Kết thúc