TÌM HIỂU REACT-NATIVE

1. **Props**

* Hầu hết các **components** đều có thể được tùy chỉnh khi chúng ta tạo ra nó, với những tham số khác nhau. Những tham số này được gọi là **props**.
* Ví dụ, một thành phần cơ bản trong React Native là Image. Khi bạn tạo ra một image, bạn có thể sử dụng một prop có tên là source để điểu khiển image nào sẽ được hiển thị ra.
* import React, { Component } from 'react';
* import { AppRegistry, Image } from 'react-native';
* class Bananas extends Component {
* render() {
* let pic = {
* uri: 'https://upload.wikimedia.org/wikipedia/commons/d/de/Bananavarieties.jpg'
* };
* return (
* <Image source={pic} style={{width: 193, height: 110}}/>
* );
* }
* }
* AppRegistry.registerComponent('Bananas', () => Bananas);
* Chú ý rằng {pic} được bao ngoài bởi dấu móc, để hiển thị biến pic trong JSX, Bạn có thể đặt bất cứ câu lệnh JavaScript trong kết cấu của JSX.
* Các **components** của bạn đồng thời có thể sử dụng **props**. Điều này cho phép bạn tạo ra các components đơn để sử dụng ở nhiều nơi khác nhau trong ứng dụng của bạn với tính chất khác nhau ở mỗi một nơi mà bạn dùng. Chỉ việc gọi đến this.props trong phương thức render của bạn. Dưới đây là ví dụ:
* import React, { Component } from 'react';
* import { AppRegistry, Text, View } from 'react-native';
* class Greeting extends Component {
* render() {
* return (
* <Text>Hello {this.props.name}!</Text>
* );
* }
* }
* class LotsOfGreetings extends Component {
* render() {
* return (
* <View style={{alignItems: 'center'}}>
* <Greeting name='Rexxar' />
* <Greeting name='Jaina' />
* <Greeting name='Valeera' />
* </View>
* );
* }
* }
* AppRegistry.registerComponent('LotsOfGreetings', () => LotsOfGreetings);
* Sử dụng **name** như một **prop** cho phép chúng ta tùy chỉnh component **Greeting** , vì vậy chúng ta có thể sử dụng lại các thành phần cho mỗi Greeting của chúng ta.
* Điều mới mẻ khác xảy ra ở đây là component <View>. Component View rất hữu dụng, nó là một container cho các component khác, để giúp điều khiển style và giao diện.
* Với **props** và các **components** cơ bản như **Text**, **Image**, và **View** bạn có thể xây dựng một loạt các màn hình tĩnh.

1. **State**

* Nếu như **Props** được cài đặt bởi cha của nó và chúng sẽ được cố định trong suốt thời gian hiện thị của **Component**. Muốn dữ liệu có thể thay đổi được chúng ta sẽ phải sử dụng đến **State**.
* Thông thường, bạn nên khởi tạo **state** trong hàm khởi tạo, và sau đó bạn có thể gọi **setState** khi bạn muốn thay đổi nó.
* Ví dụ như, nếu bạn muốn làm cho một văn bản nhấp nháy liên tục trong toàn bộ thời gian, Văn bản đó sẽ được cài đặt một lần khi **component** được tạo ra, vậy bản thẩn nó sẽ là một **prop**. Và thở thời điểm này thì băn bản sẽ được hiển thị mà không bị thay đổi trong quá trình chạy, thế nên nó lên giữ thêm một state.
* import React, { Component } from 'react';
* import { AppRegistry, Text, View } from 'react-native';
* class Blink extends Component {
* constructor(props) {
* super(props);
* this.state = {showText: true};
* // Toggle the state every second
* setInterval(() => {
* this.setState({ showText: !this.state.showText });
* }, 1000);
* }
* render() {
* let display = this.state.showText ? this.props.text : ' ';
* return (
* <Text>{display}</Text>
* );
* }
* }
* class BlinkApp extends Component {
* render() {
* return (
* <View>
* <Blink text='I love to blink' />
* <Blink text='Yes blinking is so great' />
* <Blink text='Why did they ever take this out of HTML' />
* <Blink text='Look at me look at me look at me' />
* </View>
* );
* }
* }
* AppRegistry.registerComponent('BlinkApp', () => BlinkApp);
* Với một ứng dụng thực tế, bạn không nên cài đặt **state** với một timer. Bạn có thể cài đặt **state** khi bạn có dữ liệu mới nhận được từ server hoặc từ phía người dùng nhập vào. Bạn đồng thời cũng có thể dử dụng một state container giống như **Redux** để điều khiển luồng dữ liệu. Trong trường hợp này bạn nên sử dụng Redux để thay đổi state của bạn sẽ hay hơn là việc gọi điều hướng đến **setState**.

1. FlatList

* Nếu như bạn khá khó chịu khi thiết lập một **ListView** thì sau phiên bản 0.44 của **React Native** chúng ta đã có một component dễ hiểu, dễ sử dụng hơn, hộ trợ các tính năng thông minh hơn và có thể thay thế hoàn toàn ListView cũ đó chính là **FlatList**.
* Để có được một **FlatList** chúng ta chỉ cần tối thiểu một mảng dữ liệu:

<FlatList

data={[{key: 'a'}, {key: 'b'}]}

renderItem={({item}) => <Text>{item.key}</Text>}

/>

* Ví dụ demo:
* import React from 'react';
* import {Text, View, FlatList, StyleSheet } from 'react-native';
* export default class MyListItem extends React.Component {
* constructor(props){
* super(props);
* this.state={
* listItem : [
* {key: '1', maSP: 'SP01', tenSP: 'Macbook Air 2018'},
* {key: '2', maSP: 'SP02', tenSP: 'Macbook Pro 13" 2018'},
* {key: '3', maSP: 'SP03', tenSP: 'Macbook Pro 15" 2017'},
* {key: '4', maSP: 'SP04', tenSP: 'Macbook Air 2018'},
* {key: '5', maSP: 'SP05', tenSP: 'Macbook Pro 15" 2017'},
* {key: '6', maSP: 'SP06', tenSP: 'Macbook Air 2018'},
* {key: '7', maSP: 'SP07', tenSP: 'Macbook Pro 15" 2017'},
* {key: '8', maSP: 'SP08', tenSP: 'Macbook Air 2018'},
* {key: '9', maSP: 'SP09', tenSP: 'Macbook Pro 15" 2017'},
* {key: '10', maSP: 'SP10', tenSP: 'Macbook Pro 13" 2017'},
* ]
* }
* }
* render() {
* return (
* <FlatList
* data={this.state.listItem}
* renderItem={({item}) =>
* <View style={sp.dong}>
* <Text>{item.key}</Text>
* <Text>{item.maSP}</Text>
* <Text>{item.tenSP}</Text>
* </View>
* }
* />
* );
* }
* }
* var sp = StyleSheet.create({
* dong: {
* borderBottomWidth: 1,
* padding: 50,
* }
* });
* Ngoài ra, có thể sử dụng webservice để đổ dữ liệu như ListView
* **FlatList** cũng thừa hưởng các Prop giống như ScrollView:
* renderItem
* data
* ItemSeparatorComponent
* ListEmptyComponent
* ListFooterComponent
* ListHeaderComponent
* columnWrapperStyle
* extraData
* horizontal
* getItemLayout
* numColumns
* onRefresh
* removeClippedSubviews
* refreshing
* viewabilityHelper
* ViewabilityConfigCallbackPair
* **Các phương thức:**
* scrollToEnd
* scrollToIndex
* scrollToItem
* scrollToOffset
* flashScrollIndicators

1. Navigation

* Các ứng dụng di động hiếm khi được tạo thành từ một màn hình duy nhất. Việc quản lý việc trình bày và chuyển đổi giữa, nhiều màn hình thường được xử lý bởi cái được gọi là hoa tiêu.
* Nếu bạn chỉ làm ứng dụng IOS thì bạn chỉ cần quan tâm đến NavigatorIOS. Tuy nhiên nó sẽ không hoạt động trên android.
* Nếu bạn muốn ứng dụng chạy cả trên IOS lẫn Android thì bạn phải sử dụng React-native-Navigation để có thể chạy trên cả 2 nền tảng.
* Đầu tiên chúng ta phải cài đặt react-navigation:

npm install --save react-navigation

* Tiếp theo cài đặt react-native-gesture-handler:

npm install –save react-native-gesture-handler

* Tiếp theo, liên kết tất cả:

react-native link react-native-gesture-handler

* Thực hiện các sửa đổi file MainActivity.java để quá trình cài đặt hoàn tất:
* package com.reactnavigation.example;
* import com.facebook.react.ReactActivity;
* import com.facebook.react.ReactActivityDelegate;
* import com.facebook.react.ReactRootView;
* importcom.swmansion.gesturehandler.react.RNGestureHandlerEnabledRootView;
* public class MainActivity extends ReactActivity {
* @Override
* protected String getMainComponentName() {
* return "Example";
* }
* @Override
* protected ReactActivityDelegate createReactActivityDelegate() {
* return new ReactActivityDelegate(this, getMainComponentName()) {
* @Override
* protected ReactRootView createRootView() {
* return new RNGestureHandlerEnabledRootView(MainActivity.this);
* }
* };
* }
* }
* Sau đó, bạn có thể tạo ứng dụng với màn hình chính và màn hình profile

import {createStackNavigator} from 'react-navigation';

const App = createStackNavigator({

Home: {screen: HomeScreen},

Profile: {screen: ProfileScreen},

});

export default App;

* Mỗi thành phần màn hình có thể đặt các tùy chọn điều hướng như tiêu đề, button... Nó có thể sử dụng các trình tạo hành động trên navigationprop để liên kết với các màn hình khác:

class HomeScreen extends React.Component {

static navigationOptions = {

title: 'Welcome',

};

render() {

const {navigate} = this.props.navigation;

return (

<Button

title="Go to Jane's profile"

onPress={() => navigate('Profile', {name: 'Jane'})}

/>

);

}

}

* Bộ định tuyến React Navigation giúp dễ dàng ghi đè logic điều hướng. Do các bộ định tuyến có thể được lồng vào nhau, nên các nhà phát triển có thể ghi đè logic điều hướng cho một khu vực của ứng dụng mà không cần thực hiện các thay đổi rộng rãi.
* Các khung nhìn trong React Navigation sử dụng các thành phần gốc và Animatedthư viện để cung cấp các hình động 60fps được chạy trên luồng gốc. Thêm vào đó, các hình ảnh động và cử chỉ có thể dễ dàng tùy chỉnh.