# Giới thiệu

## Nhu cầu :

Hiện nay, thời điểm khoa học công nghệ hiện đại. Mọi tác vụ, ngành nghề đều có sự hỗ trợ của công nghệ. Tuy nhiên nhiên có nhiều lĩnh vực tuy có được sự hỗ trợ của khoa học công nghệ.

## Công nghệ :

### Giao diện :

* React Native : xây dựng ứng dụng điện thoại bằng ngôn ngữ JavaScript
* HTML + CSS + JS : xây dựng giao diện trang web

### Server :

* Laravel : Framework xây dựng web và API bằng ngôn ngữ PHP
* Node JS : xây dựng kết nối thời gian thực thông qua giao thức websocket, tcp.

### Cơ sở dữ liệu :

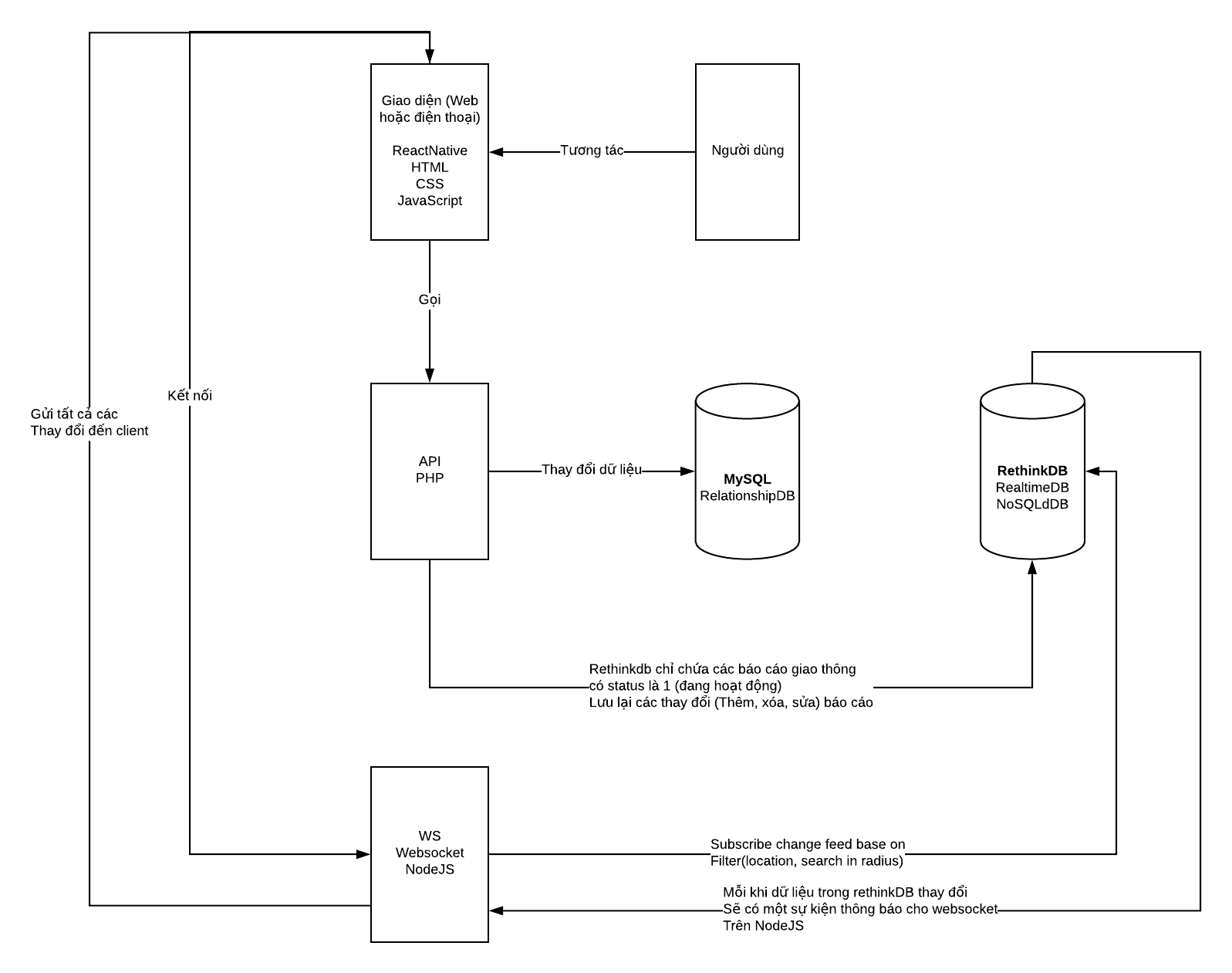
* MySQL: cơ sở dữ liệu “mặc định” đi kèm với PHP.
* RethinkDB: NoSQL.l;

### Giải thích lựa chọn công nghệ :

React Native là một công nghệ mới dùng để xây dựng ứng dụng điện thoại bằng ngôn ngữ JavaScript (Giới thiệu kĩ hơn ở chương 2) : Do không có kiến thức nhiều về ngôn ngữ Native của hệ điều hành tương ứng (Java đối với Android) và một phần muốn trải nghiệm công nghệ mới nên em chọn công nghệ này để xây dựng ứng dụng mobile.

RethinkDB là một bước trung gian để kết nối WebSocket và API trên PHP (Giải thích ở chương 3). Do bản chất PHP không hỗ trợ kết nối không đồng bộ (asynchonrous connection) nên PHP dù có cái ứng dụng hỗ trợ tạo) cũng sẽ gây tốn kém bộ nhớ không hiệu quả. JavaScript bản thên nó là một ngôn ngữ rất mạnh trong việc này nên em quyết định sử dụng ứng dụng trên nền Javascript (NodeJS) để tạo một WebSocket.

### Sơ đồ công nghệ :



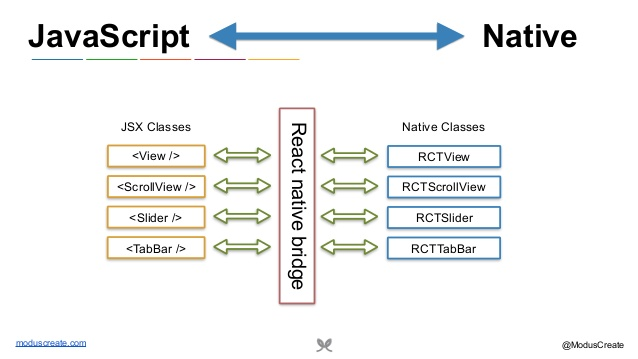
# React Native

## Giới thiệu và React Native:

React là thư viện dùng để xây dựng giao diện web được tạo ra bởi Facebook. React native là công nghệ cho phép viết ứng dụng native mobile đa nền tảng bằng ngôn ngữ lập trình web JavaScript với thư viện React.

Các component viết trong JavaScript sẽ được biên dịch thành các component trong ngôn ngữ native tương ứng với hệ điều hành đó :

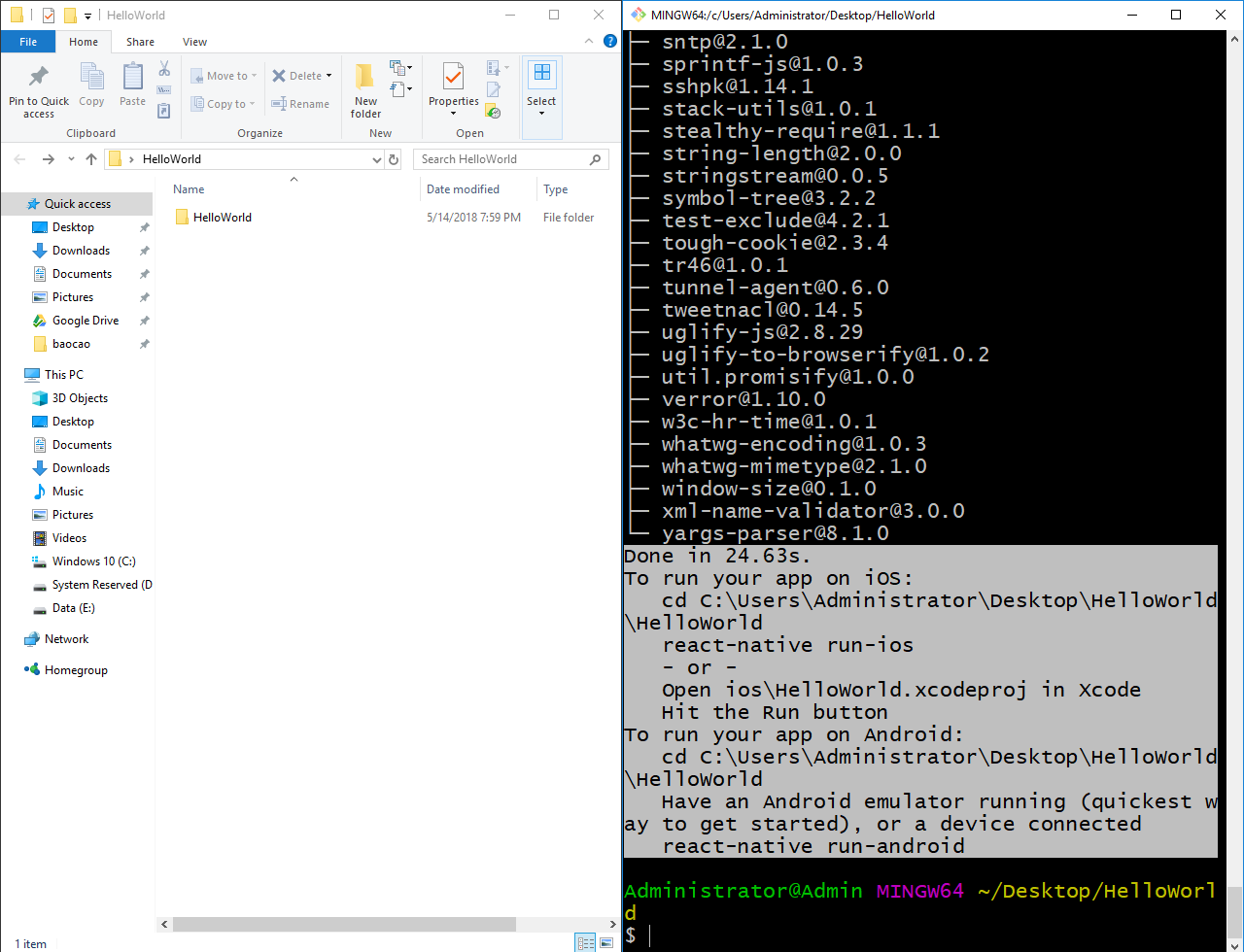
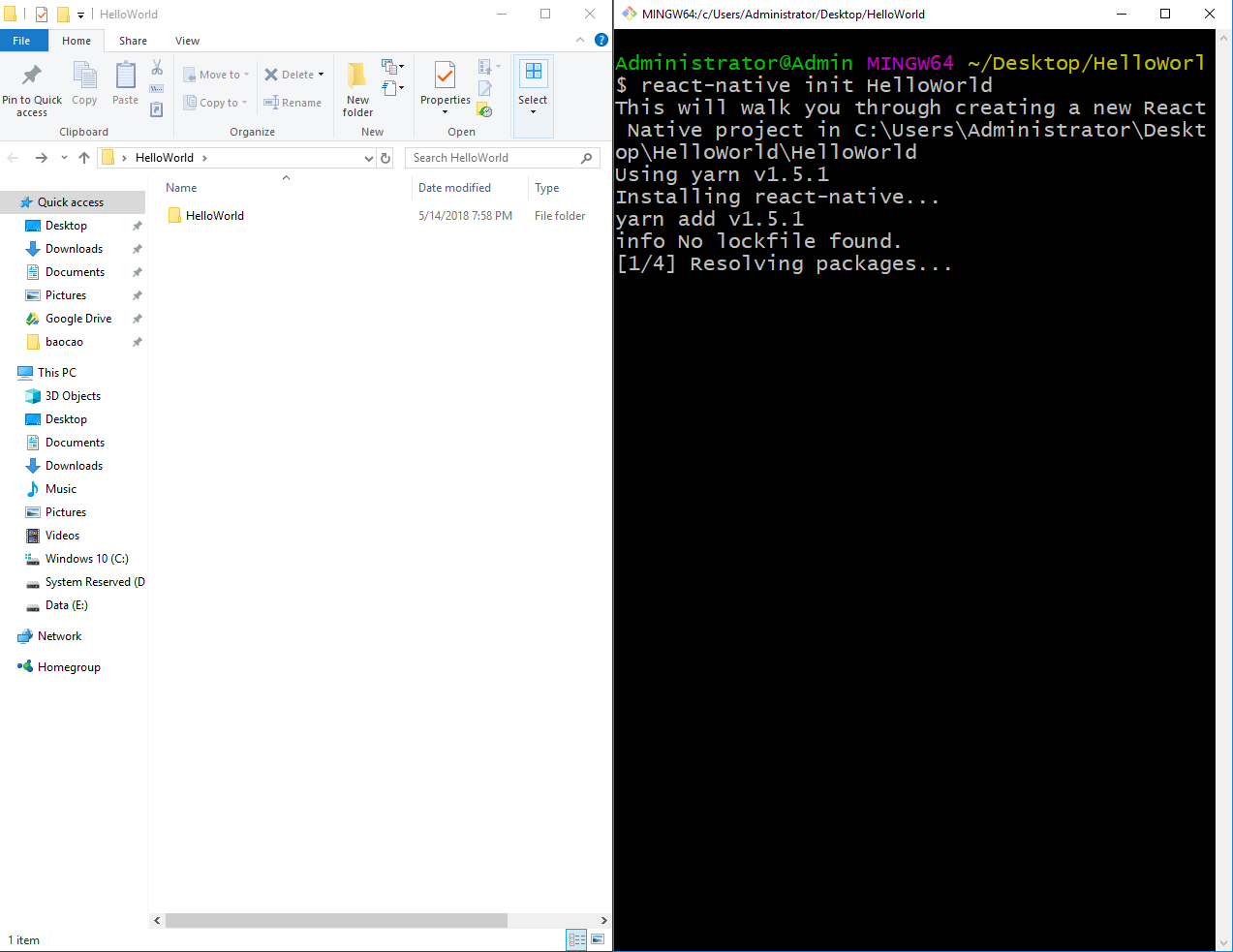
* Đối với android là Java.
* Đối với iOS là Objective-C.



Các component được biên dịch này thực chất là component được xây dựng bởi đội ngũ phát triển react native hai phiên bản bằng cả hai ngôn ngữ native. Ví dụ Native component RCTView được xây dựng một bản bằng Java và một bản bằng iOS. Khi biên dịch, dựa trên hệ điều hành component native tương ứng sẽ được biên dịch

## Tạo project React Native :

Để khởi tạo một Project Native ta bắt đầu tại vị trí muốn khởi tạo, không nhất thiết phải là folder. Từ giao diện dòng lệnh tương ứng của hệ điều hành, gõ dòng lệnh với cú pháp : **react-native init <Tên project>.** React-native sẽ tạo một folder có tên bằng với tên của project tại vị trí khởi tạo.

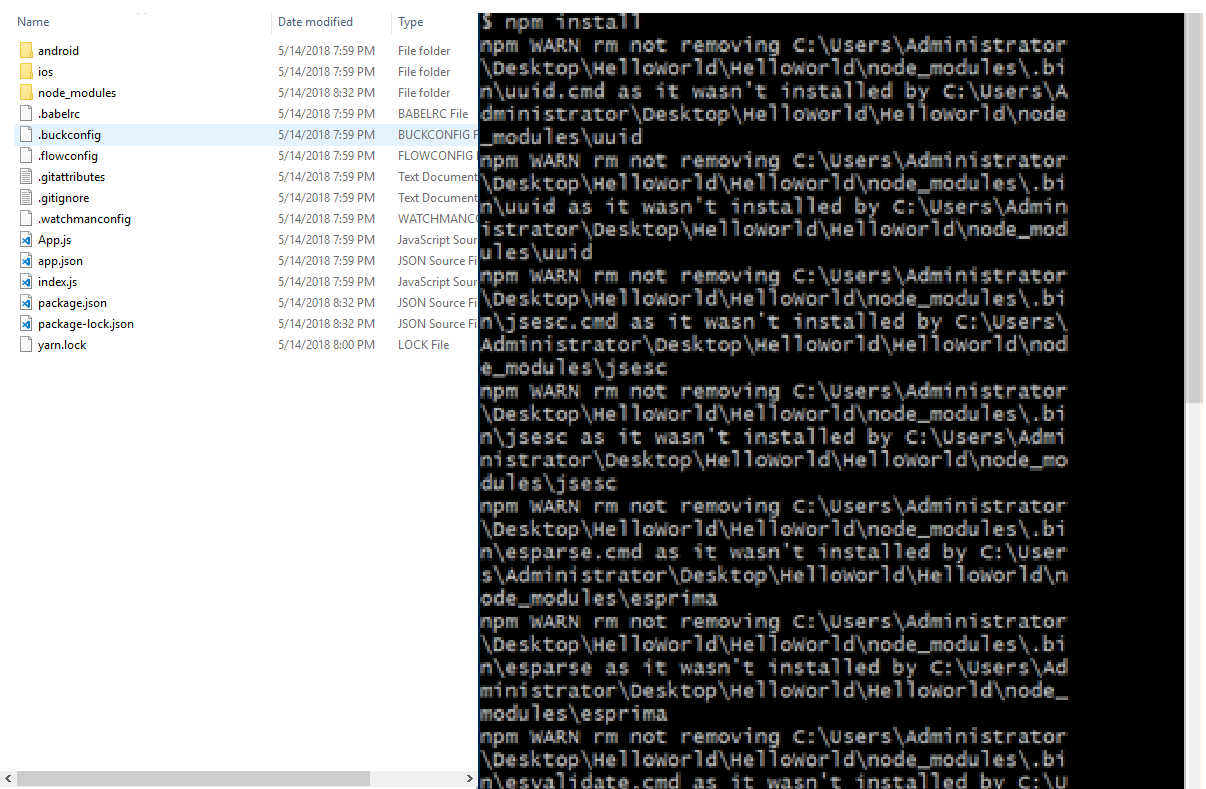


Tiếp theo, ta phải di chuyển vào folder vừa được tạo vào chạy lệnh npm install để cài đặt các gói cần thiết. Khi ta chạy lệnh react-native init thì react native chỉ tạo cho chúng ta project mà chưa có các gói denpendency.

Trong nodejs ta có thể chia sẻ các đoạn mã hay thư viện lại thành một gói. Khi ta sử dụng đoan mã đó thì dự án của chúng ta phụ thuộc vào các gói đó. Danh sách của các gói mà dự án phụ thuộc nằm trong file package.json của dự án.

Cấu trúc thư mục của ứng dụng React Native gồm :

* Android : chứa các file liên quan dến lập trình android
* iOS: có chức năng tương tự như folder Android
* Node module: chứa các gói dependency
* File App.js: đây là file đầu vào của ứng dụng, có tác dụng giống như hàm main trong các ngôn ngữ C++, C#.

Sau khi tạo xong project, duy chuyển vào trong folder bằng lệnh cd (đối với windows) và chạy ứng dụng bằng câu lệnh : react-native run-<Hệ điều hành> mà ta muốn phát triển. Phải mở sẵn trình giả lập hoặc có thiết bị thực kết nối với máy tính qua cáp USB.

## Cài đặt React Native :

### Đối với phát triển ứng dụng hệ điều hành Android

#### Đối với môi trường làm việc Mac :

Để phát triển ứng dụng android cần các gói ứng dụng kèm link :

* Node JS
  + Có thể được cài đặt bằng ứng dụng dòng lệnh brew với dòng lệnh **brew install node**
* Hoặc có thể tải gói cài đặt dành cho hệ điều hành mac bằng đường link sau [**https://nodejs.org/en/download/**](https://nodejs.org/en/download/)
* Watchman
  + Có thể được cài đặt bằng ứng dụng dòng lệnh brew với dòng lệnh **brew install watchman**
* React Native CLIJava Development Kit (phiên bản 8)
  + Có thể tải và cài đặt tại đường link [**http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html**](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)
* Android Deveper Studio
* React Native CLI

#### Đối với môi trướng làm việc Windows :

* Node, Python2, JDK
  + Có thể cài đặt tất cả các gói này một lúc bằng trình cài đặt dòng lệnh choco (giống brew bên mac) : **choco install -y nodejs.install python2 jdk8**

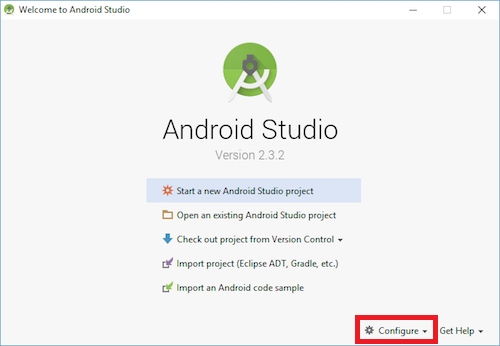
#### Cài đặt React Native CLI :

* Android Deveper Studio
* React Native CLI
  + Có thể cài đặt bằng trình quản lý gói “npm” được cài đặt chung với Node Js. Gõ dòng lệnh này vào terminal đề cài đặt :

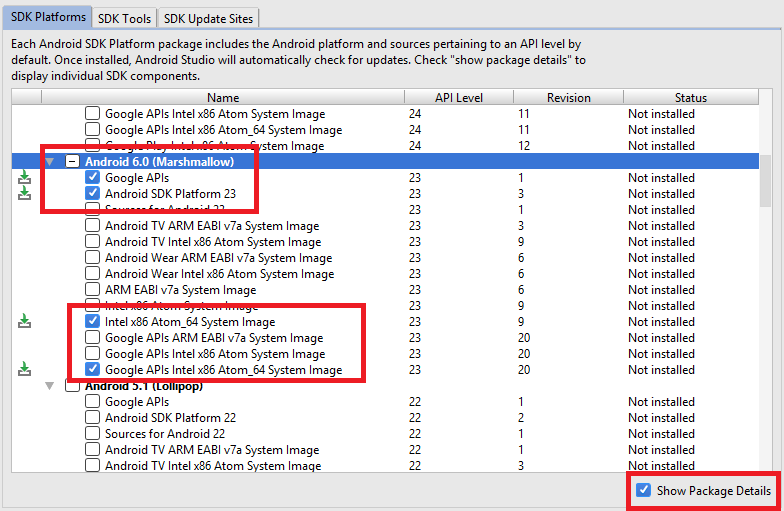
**npm install -g react-native-cli**

#### Cài đặt Android Developer Studio :

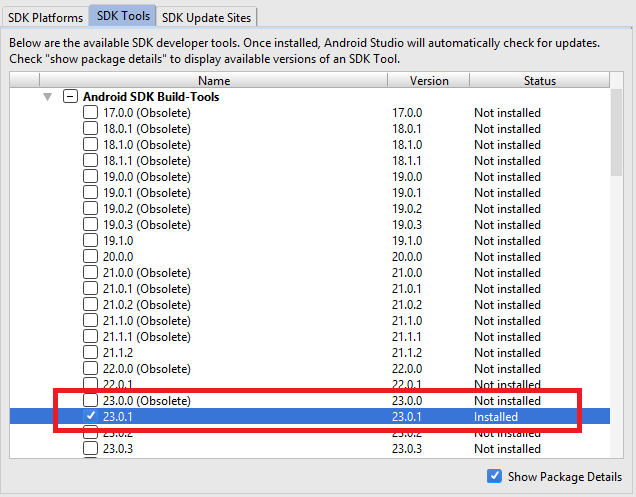
* Tải và cài đặt Android studio của hệ điều hành tương ứng tại link : <https://developer.android.com/studio/>
* Khi cài đặt chọn custom setup và tick vào các lựa chọn sau :
  + Android SDK
  + Android SDK Platform
  + Performance (Intel ® HAXM)
  + Android Virtual Device
* Cài đặt Android SDK bản 6.0 :
  + Ở màn hình welcome chọn Configure > SDK Manager



* Chọn show details và tick vào các lựa chọn sau để cài đặt thêm các gói cần thiết :



Tiếp theo chọn tiếp tab : "SDK Tools" và tick vào lựa chọn 23.0.1 :



### Đối với phát triển ứng dụng hệ điều hành iOS (chỉ dành cho máy sử dụng hệ điều hành mac) :

Giống cài đặt môi trường phát triển Android thay Android Studio và JDK bằng :

* Xcode
  + Có thể cài đặt trên Mac Appstore theo đường link : <https://itunes.apple.com/us/app/xcode/id497799835?mt=12>
* Cài đặt command line tool trên Xcode :
  + Nhấn vào Xcode trên thanh “trên cùng của mac” sau đó trên Perferences...

Chọn tab location trong menu và chọn command line tool trong tùy chọn dropdown Command Line Tool

## Cấu trúc cơ bản của ứng dụng react native :

Trong react native mọi thứ đặt trên màn hình đều là component (thành phần). Tất cả thành phần này đều kế thừa lớp component của React. Tất cả các component này sẽ có một hàm bên trong gọi là render. Đây là hàm bắt buộc của bất cứ thành phần nào trong react native. Mọi component trong hàm này đều sẽ được hiển thị đúng như nội dung của nó.

Hàn render của một component có thể chứa nhiều component khác với điều kiện là chỉ được phép có một component duy nhất nằm ngoài cùng. Các component có cú pháp dưới dạng tag xml.

Đây là đoạn code của hàm main.

*/\*\* Sample React Native App*

*\* https://github.com/facebook/react-native*

*\* @flow*

*\*/*

import React, { Component } from 'react';

import {

Platform,

StyleSheet,

Text,

View

} from 'react-native';

const instructions = Platform.select({

ios: 'Press Cmd+R to reload,\n' +

'Cmd+D or shake for dev menu',

android: 'Double tap R on your keyboard to reload,\n' +

'Shake or press menu button for dev menu',

});

type Props = {};

export default class App extends Component<Props> {

render() {

return (

<View style={styles.container}>

<Text style={styles.welcome}>

Welcome to React Native!

</Text>

<Text style={styles.instructions}>

To get started, edit App.js

</Text>

<Text style={styles.instructions}>

{instructions}

</Text>

</View>

);

}

}

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: '#F5FCFF',

},

welcome: {

fontSize: 20,

textAlign: 'center',

margin: 10,

},

instructions: {

textAlign: 'center',

color: '#333333',

marginBottom: 5,

},

});

Đề có thể thêm các component khác ta sử dụng lệnh import. Lệnh này có cú pháp :

**import <giá trị default> hoặc {<giá trị 1>, <giá trị 2>, … , <giá trị N>} from**

**<tên gói muốn import hoặc đường dẫn>.**

**Ví dụ :**

* **Import giaTriMacDinh from dependency1**
* **Import {**

**giaTriSo1,**

**giaTriSo2**

**}, giaTriMacDinh**

Lệnh export có chức năng tương tự cho phép component có thể được import bởi component khác. Có 2 loại export đó là :

* **Mặc định** : được import bởi **giaTriMacDinh** không nằm trong dấu **{}**
* **Chỉ định : Export GiaTriSo1 (không có từ khóa default**) sau đó ta có thể import giá trị này bằng **import {giaTriSo1} from dependency1**

## React native căn bản :

### Component :

Bất cứ class nào kế thừa class React.Component thì đó là component và ta có thể sử dụng nó bên trong hàm render của các component khác. Với ví dụ HelloWorld lúc nãy, ta thêm file loremIspumComponent.js có nội dụng như sau :

import React, { Component } from 'react';

import {

StyleSheet,

Text,

View

} from 'react-native';

*/\*\* Component này sử dụng component text để thể hiện*

*\* dòng chữ lorem ispum \*/*

export default class LoremIspum extends Component {

render () {

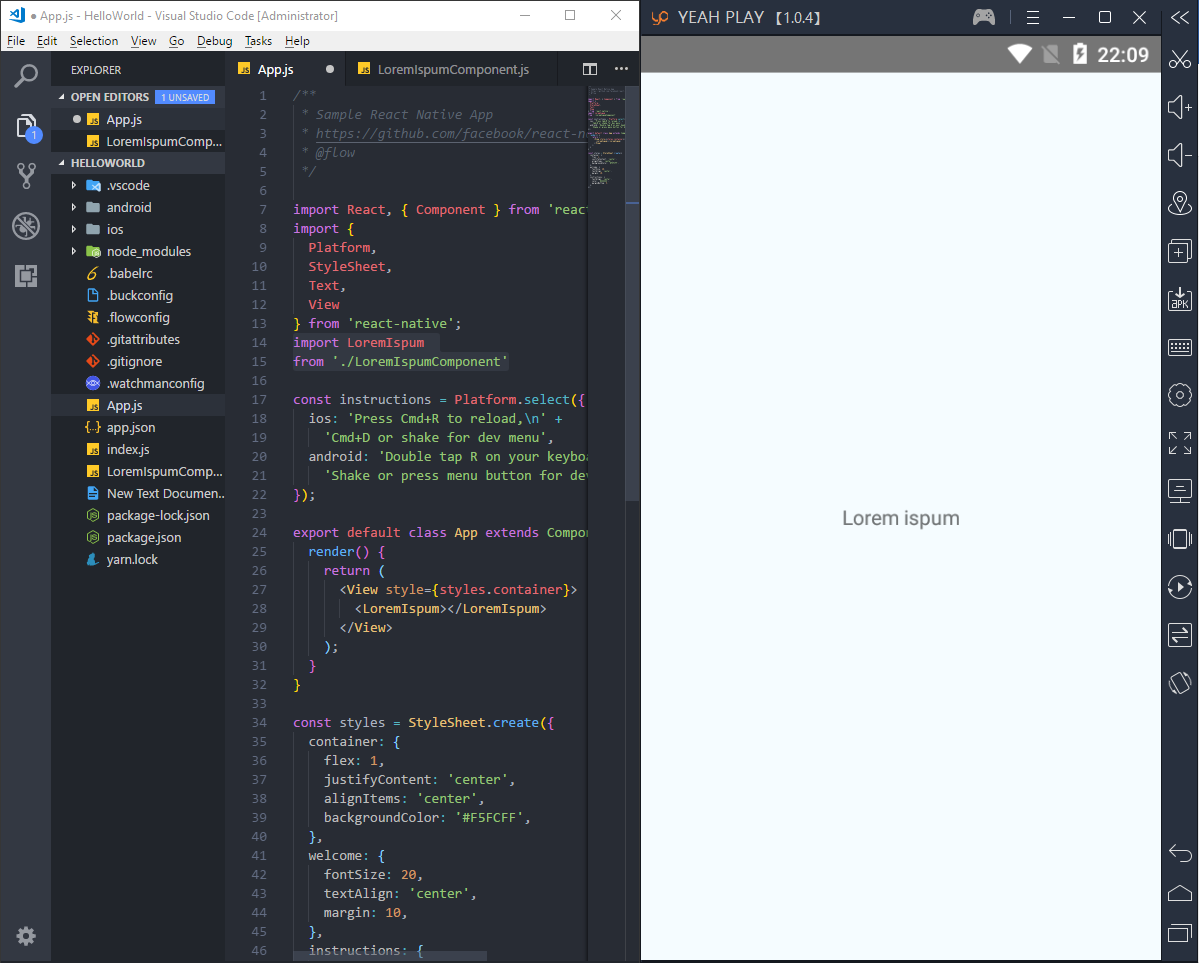
<Text>Lorem ispum</Text>

}

}

Lệnh export default cho phép class này được mặc định từ file mà không cần sử dụng {}. Tiếp theo ta import component trong App.js này bằng lệnh import.

Đa phần các component trong react native đều được xây dựng bằng cách sử dụng các component có sẵn do react native cung cấp.

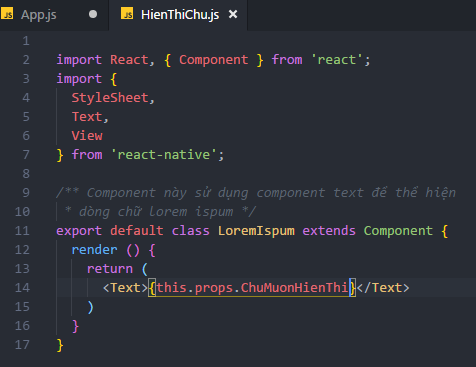


### Props :

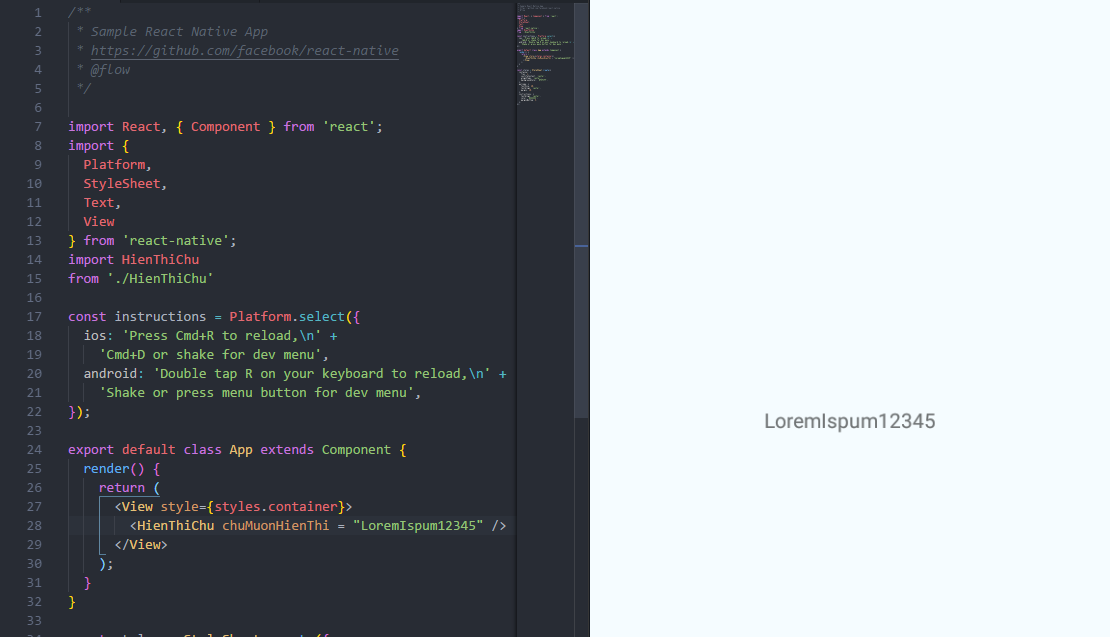
Để cho phép component khác truyền dữ liệu ta sử dụng props. Tất cả component ta có thể gán props của một componen giống như gán thuộc tính cho một tag html với cú pháp :

**<tag props = “giá trị”></tag>**

Đối với component loremispum lúc nãy ta sẽ đổi hiên thành HienThiChu (hiển thị chữ) cho phép truyền props dạng chuỗi vào component này và hiển thị chuỗi đó lên. Để có thể truy cập vào props đã truyển ta dùng cú pháp : **this.props.tên props.** Để hiện thị dữ liệu dạng chuỗi có sẵn từ một biến ta dùng dấu {} (tất nhiên muốn hiển thị chữ ta phải đặt chuỗi trong component Text)



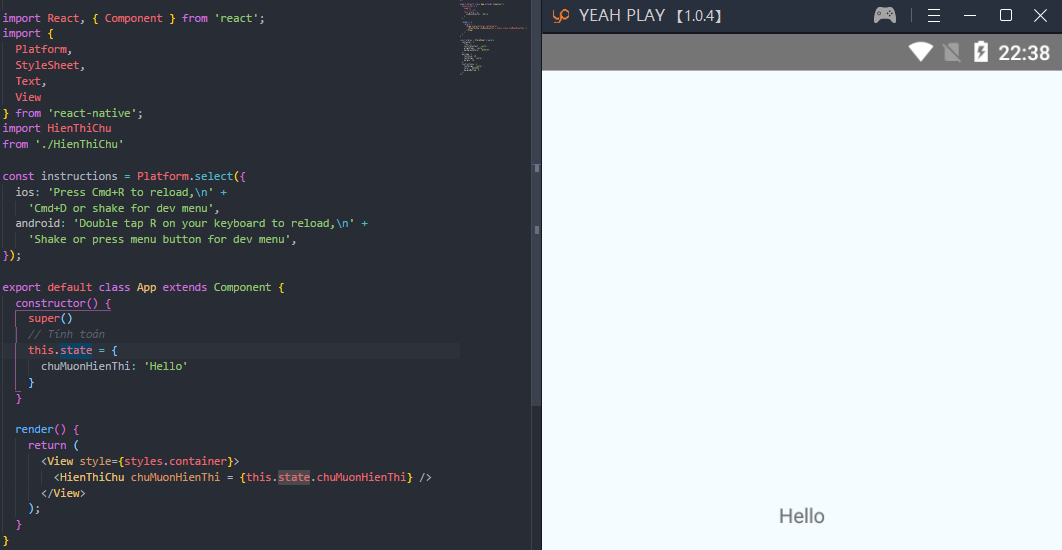
Ta gọi component truyền dữ liệu vào component khác trong hàm render bằng attribute là container component (component cha) và component bị truyền dữ liệu vào để hiện thì đó là presentation component (component hiển thị).



### State :

Ta dùng props để truyền dữ liệu từ component cha san comopnent con. Vậy dữ liệu được truyền thay vì dạng chuỗi ta muốn dữ liệu này phải được tính toán hoặc xử lý (load từ cơ sở dữ liệu). Khi đó ta dùng state, state là dữ liệu của riêng component đó.

Để có thể gán state của một component ta sẽ làm việc đó trong constructor của component đó bằng cú pháp : **this.state = {nội dung của state}**. State của component sẽ là một object ta dùng {} để tạo một object trong javascript. Lưu ý là hàm super() phải được gọi đầu tiên trong constructor của component đó.



### Stylesheet :

Do sử dụng công nghệ web để tạo ứng dụng mobile tên ta có thể sử dụng các thuộc tính có cú pháp khá giống CSS. Tất cả các **component CÓ SẴN CỦA REACT NATIVE** đều có một props gọi là style, props này nhận vào một object.

Bên css thì tên thuộc tính viết thường và nếu dài trên 2 chữ thì ngăn cách bằng dầu gạch **–**Trong khi react native thì viết thường chữ đầu và chữ sau viết hoa chữ đầu, không khoảng trắng. Ví dụ :

style = {{

// Có thể sử dụng dấu gạch chéo để comment vì ở đây vẫn là JavaScript.

textAlign: ‘center’ // Thay cho text-align

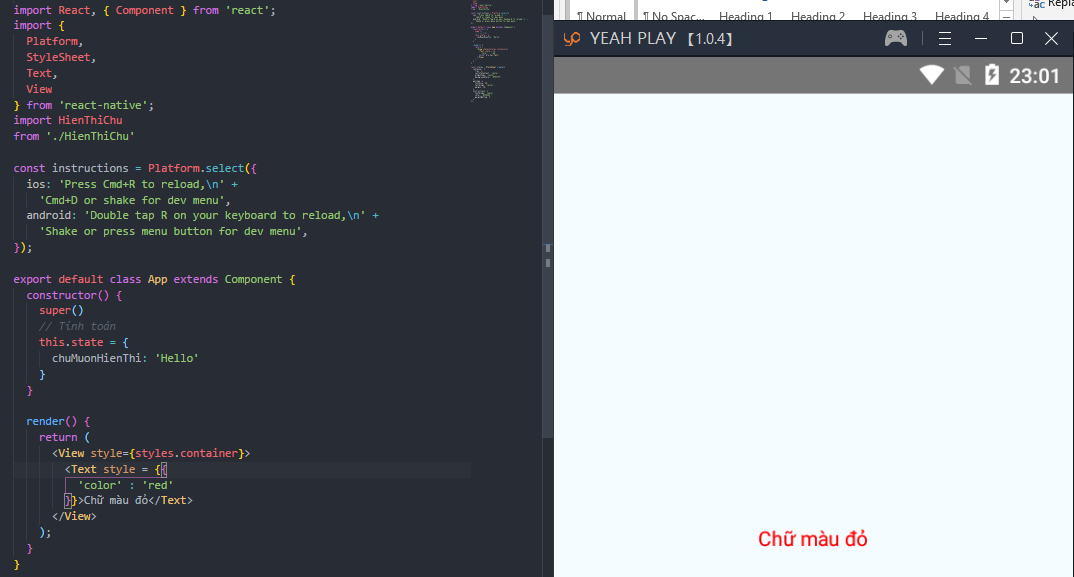
}}

Ta sử dụng hai dấu ngoặc vì thuộc tính (props) của một component nhận vào hai giá trị :

* Chuỗi
* Không phải chuỗi ta dùng {}. Ví dụ ta có biến a nằm trong state và muốn set biến a vào trong prop của component ta dùng **<component props = {a} />.**  Lưu ý là ta có thể sử dụng tag tự đóng <tag/> hoặc tag đóng <tag></tag> đều được.

Props style của component này không phải là chữ mà là một object. Object khai báo bằng {} cho nên sử dung hai dấu {} tức là {{}}.

Ví dụ dưới đây làm cho chữ của component Text màu đò bằng cách sử dụng thuộc tính color.



Cách viết trên là inline style khá giống bên CSS. Ngoài ra react native còn có một cách viết nữa là external style. External sẽ sử dụng Style.create({nhiều thuộc tính style}).

Ví dụ :



## Crossplatform :

Công nghệ react native cho phép người dùng tạo ứng dụng crossplatform. Chương trình dưới đây xây dựng một menu gồm 4 lựa chọn. Mỗi lựa chọn là một hình ảnh và tựa đề của lựa chọn ấy. Để lựa chọn này, Thành phần bao gồm :

* Image: thành phần cho phép hiển thị hình ảnh dựa trên attribute src cung cấp.
* Text: thành phần này cho phép hiển thị text.
* View: thành phần giống như div trong html cho phép bọc các thành phần con.
* Style: giống như stylesheet trong css điểm khác biệt giữa html và react-native đó là

Đây là đoạn code minh họa cho chức năng này :

import React ,{

Component

} from 'react';

import {

View,

Image,

Text,

StyleSheet

} from 'react-native'

export default class App extends Component {

render() {

return (

<View style = {style.topContainer}>

<View style = {style.container}>

<View style = {style.img}>

<Image source = {require('../assets/report\_traffic/traffic\_jam.png')}></Image>

</View>

<Text style = {style.text}>Tắc đường</Text>

</View>

</View>

)

}

}

const style = StyleSheet.create({

img: {

marginTop: 30,

marginBottom: 15,

width: 104,

height: 104,

alignItems: 'center',

justifyContent: 'center',

borderRadius: 100,

backgroundColor: '#383C45'

},

container: {

flexDirection: 'row',

backgroundColor: 'white',

},

topContainer: {

flex: 1,

backgroundColor: 'white'

},

text: {

paddingBottom: 30,

fontSize: 18,

color: '#01b6b2',

marginLeft:10,

}

})

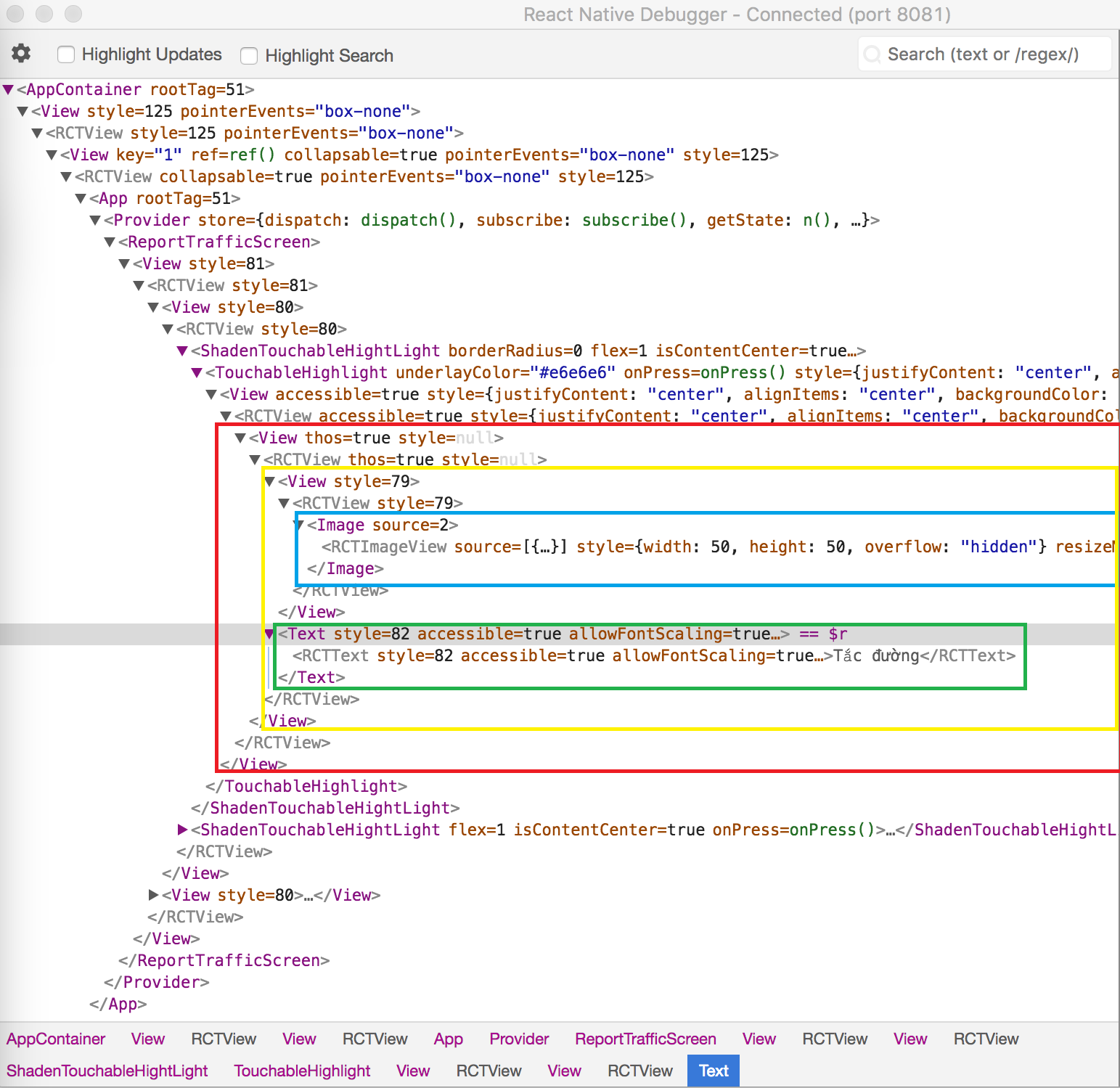
### Kết quả hệ điều hành iOS:

Đây là kết quả thành phần khi bên được biên dịch bên iOS :



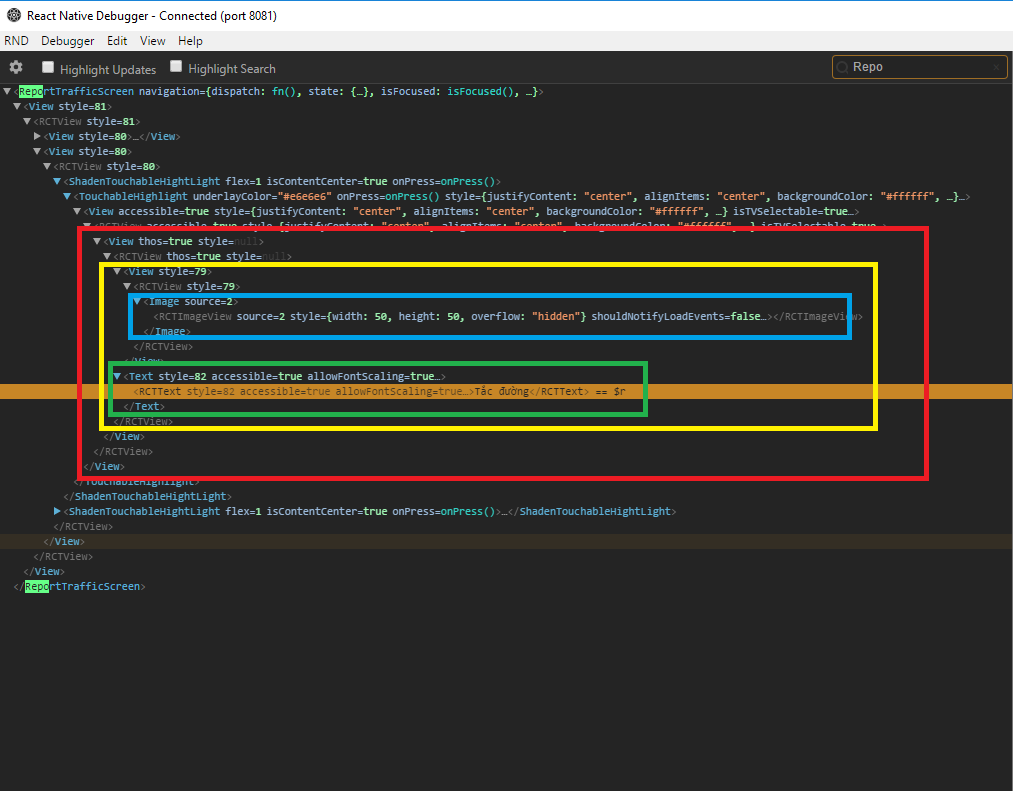
Đây là các thành phần native iOS do react-native biên dịch ra, có thể xem bằng cách mở react native debugger :

* View thành RCTView
* Image thành RCTImageView
* Text thành RCTText



### Kết quả hệ điều hành Android:

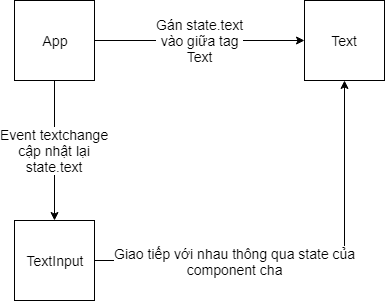
Bên Android cũng cho ra kết quả tương tự.

Đây là các thành phần native khi react-native biên dịch ra biên dịch ra cũng có cùng tên giông như bên iOS :

## Quản lý state :

### Đặt vấn đề

Giả sử chương trình của chúng ta có cấu trúc component như sau :

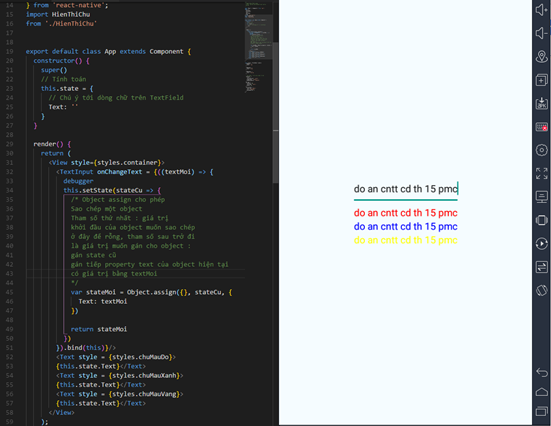


Ví dụ này mô tả cách mà 2 component cùng cha giao tiếp với nhau. React chỉ **hỗ trợ data binding một chiều** tức là :

* props truyền vào một component khi props thay đổi thì thì component con sẽ tự động nhận được thay đổi ấy ( App với Text).
  + Để binding ngược lại từ TextInput tới App thì ta phải sử dụng event textOnChange. Để gán event ta cũng sử dụng cú pháp tương tự như khi gán props, hàm ở đây có hai cách là khai báo hàm trong class hoặc sử dụng hàm anyonomous tức hàm khai báo tại chỗ.

**<component event = {hàm}/>**

Trong hàm gán event để setState chúng ta không thể dùng this.state.<thuộc muốn set> = <giá trị mới> mà phải dùng hàm setState. Tham số truyền vào hàm setState phải là một hàm có tham số đầu là state cũ. Ta phải biến đổi state cũ thành state mới xong trả về state này.



*/\* Hàm render của App.js* *\*/*

render() {

return (

<View style={styles.container}>

<TextInput onChangeText = {((textMoi) => {

this.setState(stateCu => {

*/\* Object assign cho phép* *Sao chép một object*

*Tham số thứ nhất : giá trị* *khởi đầu của object muốn sao chép*

*ở đây để rỗng, tham số sau trở đi* *là giá trị muốn gán cho object :*

*gán state cũ*

*gán tiếp property text của object hiện tại* *có giá trị bằng textMoi*

*\*/*

var stateMoi = Object.assign({}, stateCu, {

Text: textMoi

})

return stateMoi

})

}).bind(this)}/>

<Text style = {styles.chuMauDo}>

{this.state.Text}</Text>

<Text style = {styles.chuMauXanh}>

{this.state.Text}</Text>

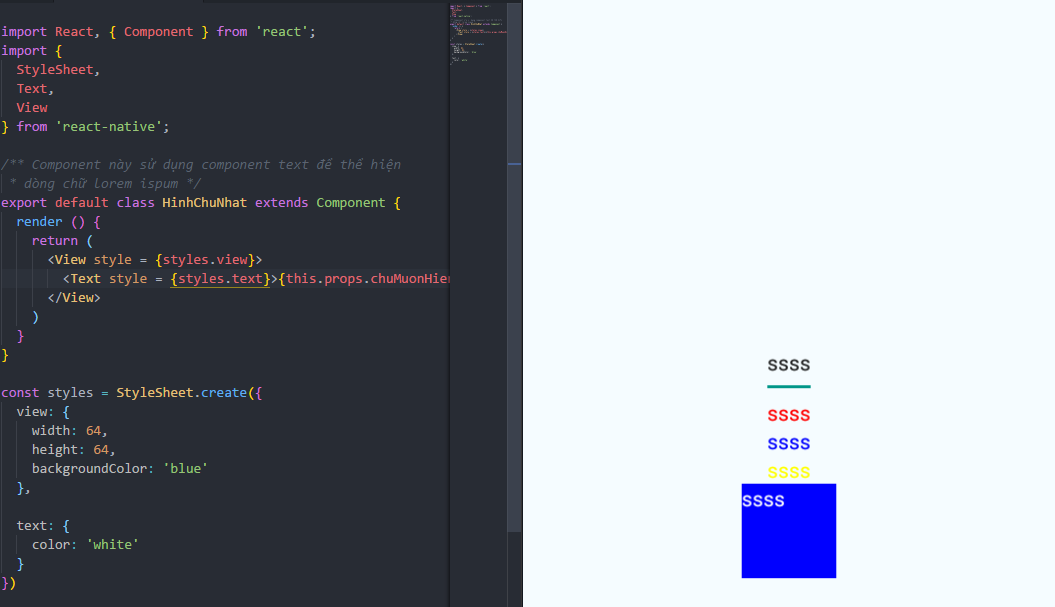
<Text style = {styles.chuMauVang}>

{this.state.Text}</Text>

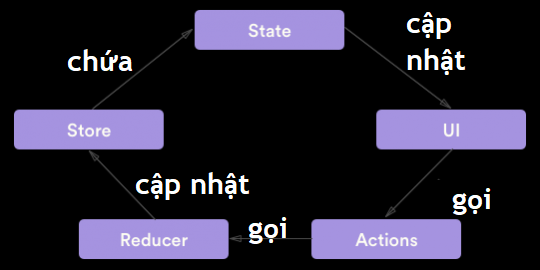
<HinhChuNhat chuMuonHienThi = {this.state.Text} />

</View>

);



Hoặc chúng ta có thể sử dụng thư viên redux. Thư viện này **thêm vào state toàn cục và tất cả component đều có thể truy cập state này lấy dữ liệu.** Chương trình sử dụng thư viện redux sẽ có dòng chảy dữ liệu như hình bên dưới :

****

### Cài đặt redux :

Redux là thư viện javascript có hỗ trợ cài đặt bằng npm. Ta có thể giao diện dòng lệnh dòng lệnh và cài đặt gói này bằng cách : **npm install redux**

### Action :

Action là hàm mà chương trình sẽ gọi khi cần thay đồi state toàn cục. Action chỉ là một hàm trả một object gồm :

* Type: dạng chuỗi. Redux khuyên sử dụng biến const có giá trị chuỗi. Ta tạo một biến const SETTEXT = ‘SETTEXT’.
* Bất cứ giá trị gì cần thiết để thay đổ giá trị của state. Ở đây, chúng ta sẽ muốn thay đổi thuộc tính text nên sẽ đặt tên là newText

Tham số của hàm này đó chính là giá trị mà chúng ta muốn gửi lên reducer. Đây là nội dung của file action.js

export const SETTEXT = 'SETTEXT'

export default {

setText (newText) {

return {

type: SETTEXT,

newText

}

}

}

### Reducer :

Reducer là hàm mà nó xử lý dựa trên :

* Loại (Type): giá trị type gửi từ reducer cho biết hàm này phải làm gì.
* Các giá trị khác gửi lên sử dụng để tính toán. Sau khi tính toán sẽ **trả về** **state toàn cục mới** sau khi **tính toán các thay đổi trên state toàn cục cũ**

**Tham số của hàm này** gồm :

* Giá trị Object gửi đi (trả về bởi) của action mà UI gọi
* State cũ

Như phần trước redux khuyên ta nên sử dụng biến CONST cho thuộc tính type và export biến này, vì tại hàm redux chúng ta sẽ import các biến này và sử dụng thay vì phải nhớ nội dung của thuộc tính type.

import {

SETTEXT

} from './action'

export default function(previousState, actionData) {

console.log(actionData, previousState)

switch (actionData.type) {

case SETTEXT:

*// return Object.assign({}, previousState, {*

*// text: newText*

*// })*

return Object.assign({}, previousState, {

text: actionData.newText

})

break

default:

return previousState

}

}

Chúng ta sử dụng Object assign vì hàm này cho phép copy object, tại reducer chúng ta nên trả về **state mới hoàn hoàn** chứ không được trả về **state cũ bị thay đổi.** Tham số của hàm này gồm :

* Giá trị ban đầu của object
* Các giá trị gán vào object được copy theo thứ tự trái sang phải.

### Store :

Store là object được tạo ra khi gọi Store.createstore(<reducer>). State toàn cục cũng được gọi là store. Store gồm một vài hàm :

* getState: trả về state toàn cục hiện tại.
* dispatch(<hàm action>): gọi action để thay đổi state.
* Subscribe(<hàm chạy>) : hàm truyền vào sẽ được gọi mỗi khi state thay đổi.

Hàm render trong App :

render() {

return (

<View style={styles.container}>

<TextInput onChangeText = {((newText) => {

**store.dispatch(action.setText(newText))**

}).bind(this)}/>

<Text style = {styles.chuMauDo}>

{this.state.Text}</Text>

<Text style = {styles.chuMauXanh}>

{this.state.Text}</Text>

<Text style = {styles.chuMauVang}>

{this.state.Text}</Text>

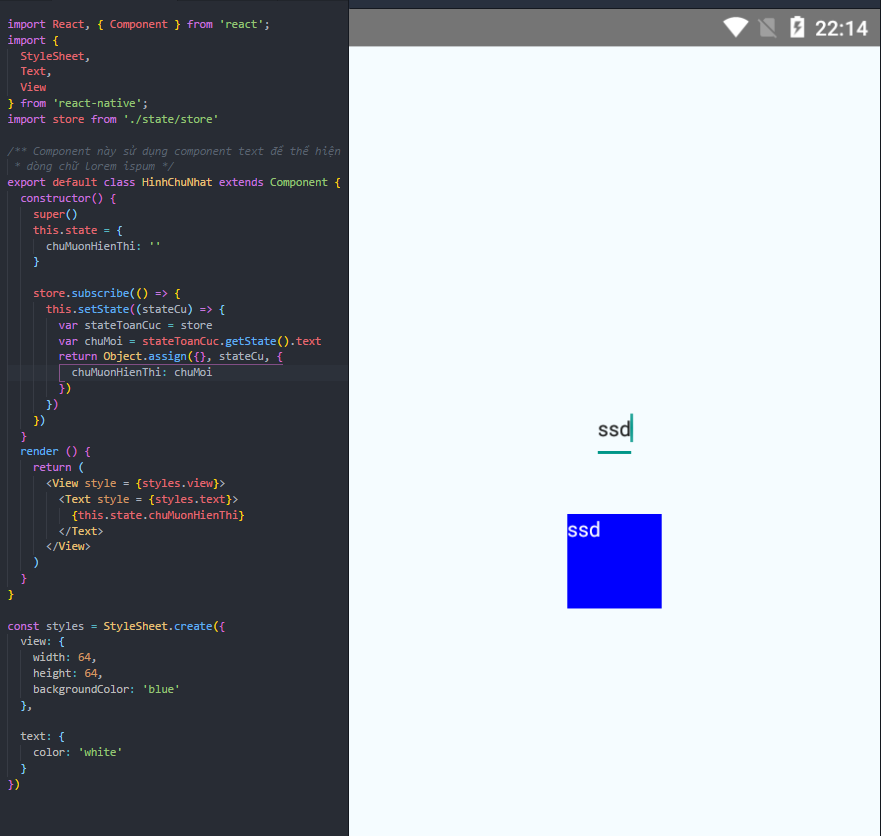
<HinhChuNhat chuMuonHienThi = {this.state.Text} />

</View>

);

}

Sau khi tạo xong store chúng ta export store để sử dụng chúng theo mô hình sau đây :



## Điều hướng :

### Đặt vấn đề :

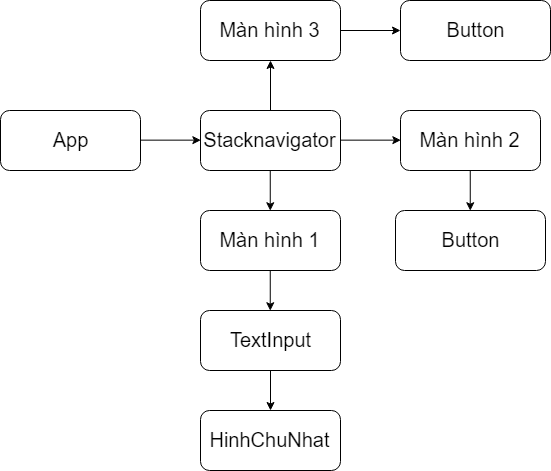
Khi phát triển ứng dụng mobile, đa phần các ứng dụng đều có trên một màn hình. Có nhiều thư viện được tạo ra đề giải quyết vấn đề này, react navigation là một trong số đó.

### Cài đặt :

Cài đặt react navigation giống như là một thư viện javascript bình thường trên npm bằng lệnh : **npm install --save react-navigation**

### Stack navigator :

Chương trình tiếp theo mà chúng ta xây dựng sẽ có kiến trúc như sau



Stacknavigation trong react navigation có cách hoạt động như các điều hướng của một trang web. Ban đầu sẽ có một màn hình làm trang chủ, giả sử chúng ta chọn màn hình một làm trang chủ thì :

* chương trình sẽ push màn hình 1 vào tronng stack.
* Cứ mỗi lần chuyển màn hình thì màn hình mói sẽ được thêm vào stack.
* Khi di chuyển về bằng phím back vật lý (android), hoặc nút back trên màn hình thì màn hình ấy sẽ bị bỏ ra khỏi stack
* Khi stack không còn màn hình nào thì chương trình sẽ thoát.



Đây là đoạn code ở màn hình 1 copy lại tất tả component của App và đổi lại tên class:

import React, { Component } from 'react';

import {

StyleSheet,

Text,

View,

TextInput

} from 'react-native';

import store from './state/store'

import action from './state/action'

import HinhChuNhat from './HinhChuNhat'

export default class ManHinh1 extends Component {

constructor() {

super()

*// Tính toán*

this.state = {

*// Chú ý tới dòng chữ trên TextField*

Text: ''

}

}

render() {

return (

<View style={styles.container}>

<TextInput onChangeText = {((newText) => {

store.dispatch(action.setText(newText))

}).bind(this)}/>

<Text style = {styles.chuMauDo}>

{this.state.Text}</Text>

<Text style = {styles.chuMauXanh}>

{this.state.Text}</Text>

<Text style = {styles.chuMauVang}>

{this.state.Text}</Text>

<HinhChuNhat chuMuonHienThi = {this.state.Text} />

<Button title = "Di chuyển sang màn hình 2" onPress = {() => {

this.props.navigation.navigate('ManHinh2')

*// Xem nội dung bên dưới để biết ý nghĩa dòng này*

}} />

</View>

);

}

}

const styles = StyleSheet.create({

chuMauDo: {

color: 'red'

},

chuMauXanh: {

color: 'blue'

},

chuMauVang: {

color: 'yellow'

},

});

Đoạn code ở màn hình 2 (màn hình 3 có cấu trúc tương tự chỉ thay 2 thành 3) :

import React, { Component } from 'react';

import {

Button,

View,

Text

} from 'react-native';

export default class ManHinh2 extends Component {

constructor() {

super()

*// Tính toán*

this.state = {

*// Chú ý tới dòng chữ trên TextField*

Text: ''

}

}

render() {

return (

<View>

<Text>Màn hình 2</Text>

<Button

title = "quay trở lại màn hình 1"

onPress = {() => {}} />

</View>

);

}

}

Tại App ta tiến hình import ManHinh1, 2, 3 và import **createStackNavigator từ gói react-navigation.** Tiếp theo, chúng ta dùng hàm createStackNavigator và truyền vào :

* Object thứ nhất : dữ liệu điều hướng
  + <tên màn hình: chuỗi> : <component thuộc màn hình này: react component>
* Object thứ hai : dữ liệu cài đặt :
  + initialRouteName: <tên màn hình chủ : chuỗi>

Và sử dụng component đó bình thường trong hàm render. Button là component tạo ra nút bấm gồm các prop :

* Title: chữ hiển thị trong button đó
* OnPress: hàm event chạy khi button đó được bấm, ở đây chúng ta có thể dùng hàm không tên khai báo bằng cú pháp **(//Tham số) => {//Nội dung hàm}**

Để di chuyển sang màn hình kế tiếp dùng lệnh this.props navigation.navigate, prop navigation này được truyền cho tất cả các component con của react-navigation :

* navigate(tên màn hình:chuỗi) : cho phép di chuyển sang màn hình khác
* goBack() : lùi về màn hình trước đó

Đây là đoạn code của App.js :

import { createStackNavigator } from 'react-navigation';

import ManHinh1 from './ManHinh1'

import ManHinh2 from './ManHinh2'

import ManHinh3 from './ManHinh2'

export default class App extends Component {

constructor() {

super()

*// Tính toán*

this.state = {

*// Chú ý tới dòng chữ trên TextField*

Text: ''

}

}

render() {

return (

<RootStack/>

)

}

}

const RootStack = createStackNavigator({

ManHinh1,

ManHinh2,

ManHinh3

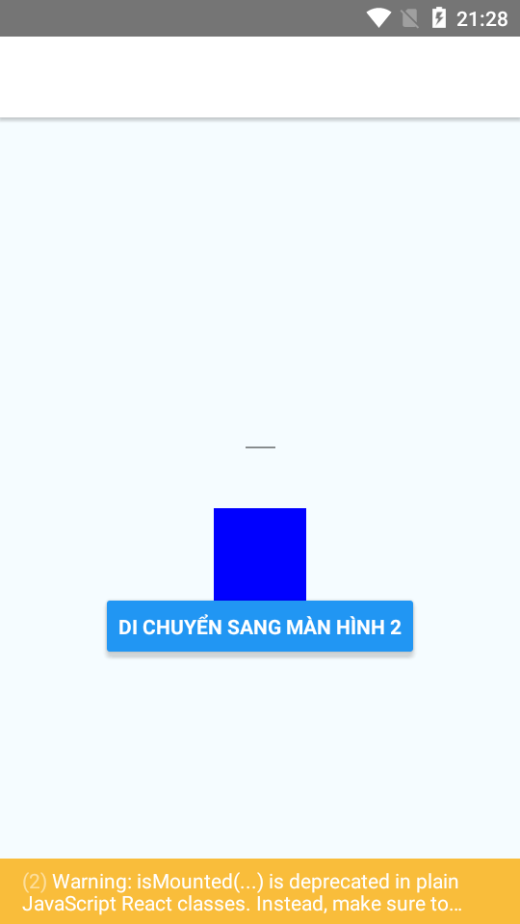
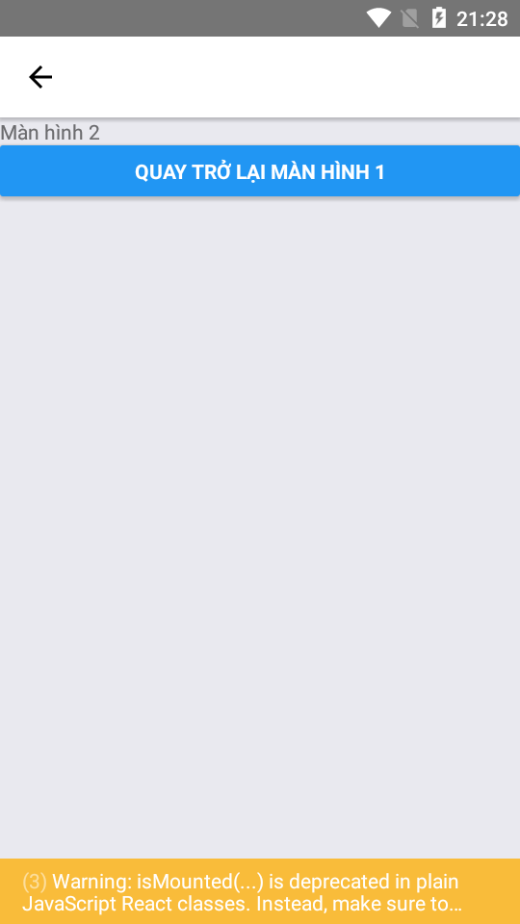
}, {

initialRouteName: 'ManHinh1'

});

Xem full source của chương trình này tại :[**https://gist.github.com/PhmNgocNghia/f5d43e0f5ca00579e716c311d918c87a**](https://gist.github.com/PhmNgocNghia/f5d43e0f5ca00579e716c311d918c87a)

Bên dưới là kết quả :



### Tab navigator :

Không giống stacknavigator, trong tabnavigator mọi màn hình đều đồng cấp với nhau đồng nghĩa với việc lùi màn hình là thoát chương trình trừ khi tabnavigator nằm trong stacknavigator (layout phức tạp). Đây là đoạn code của App (thay hàm createStackNavigator bằng createBottomTabNavigator để tạo tab navigator) :

import { createBottomTabNavigator } from 'react-navigation';

export default class App extends Component {

constructor() {

super()

*// Tính toán*

this.state = {

*// Chú ý tới dòng chữ trên TextField*

Text: ''

}

}

render() {

return (

<RootStack/>

)

}

}

const RootStack = createBottomTabNavigator ({

ManHinh1,

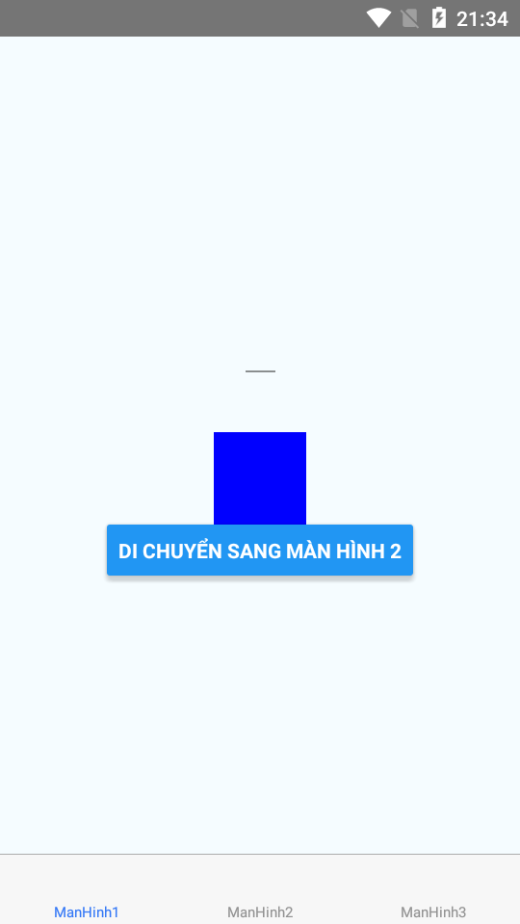
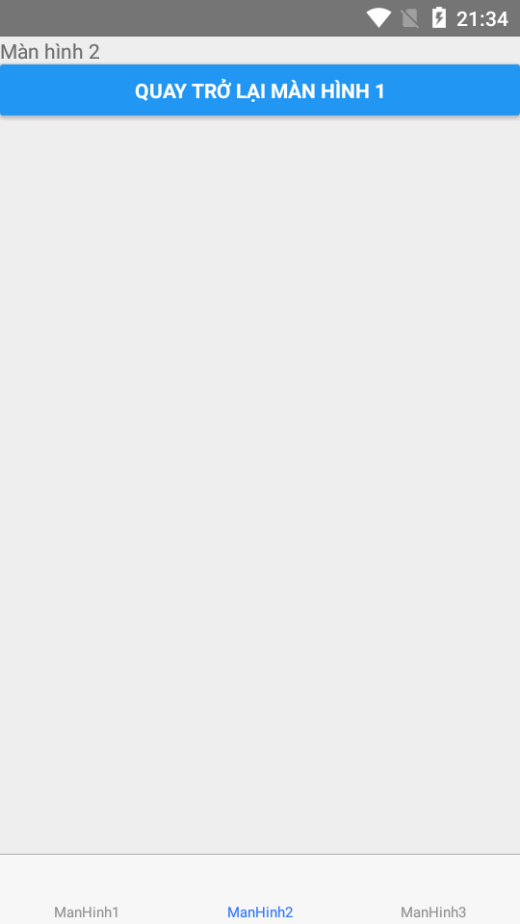
ManHinh2,

ManHinh3

}, {

initialRouteName: 'ManHinh1'

});



# RethinkDB

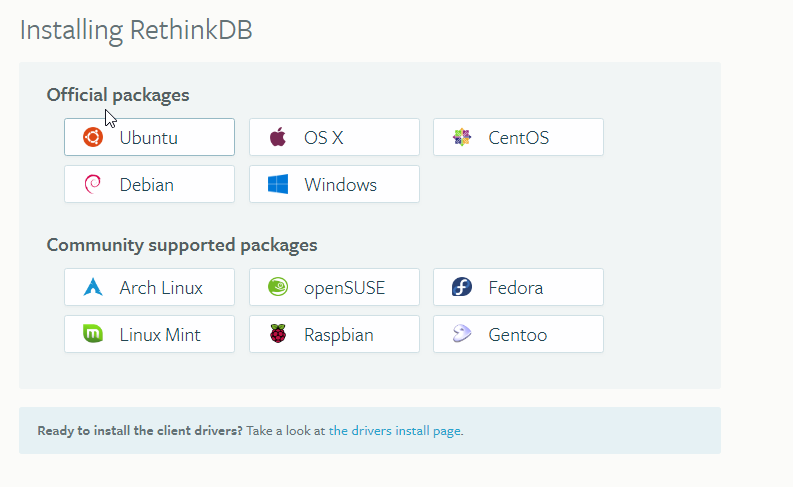
## Giới thiệu RethinkDB :

RethinkDB là một cơ sở dữ liệu NoSQL có cơ chế realtime. Mỗi khi dữ liệu bên trong nó thay đổi (thêm, xóa, sửa) thì rethinkDB cung cấp cho chúng ta một API gọi changes cho phép chúng ta biết :

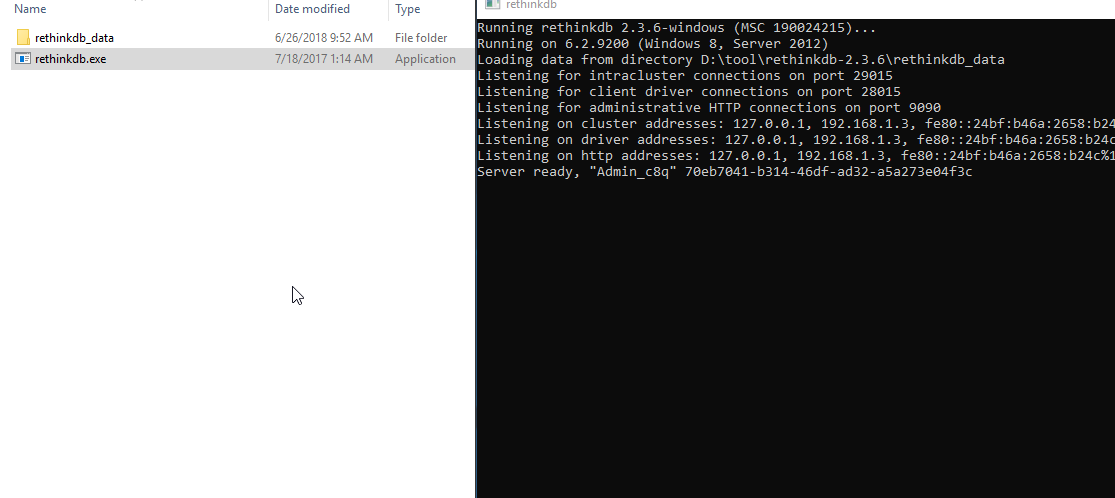
* Khi nào thay đổi xảy ra : quan trong là ở đây, mỗi khi dữ liệu bị thay đổi do API bên Laravel thì bên nodejs sẽ nhận được tín hiệu.
* Dữ liệu đã thay đổi : và gửi tất cả các thay đổi này xuống để client xử lý
  + Old\_val : dữ liệu đã thay đổi
  + New\_val dữ liệu đã thay đổi

## Cài đặt RethinkDB:

Để cài đặt rethinkdb ta có thê lên trang chủ chọn hệ điều hành tương thích. Ở đó mỗi hệ điều hành sẽ có cách cài đặt khách nhau:

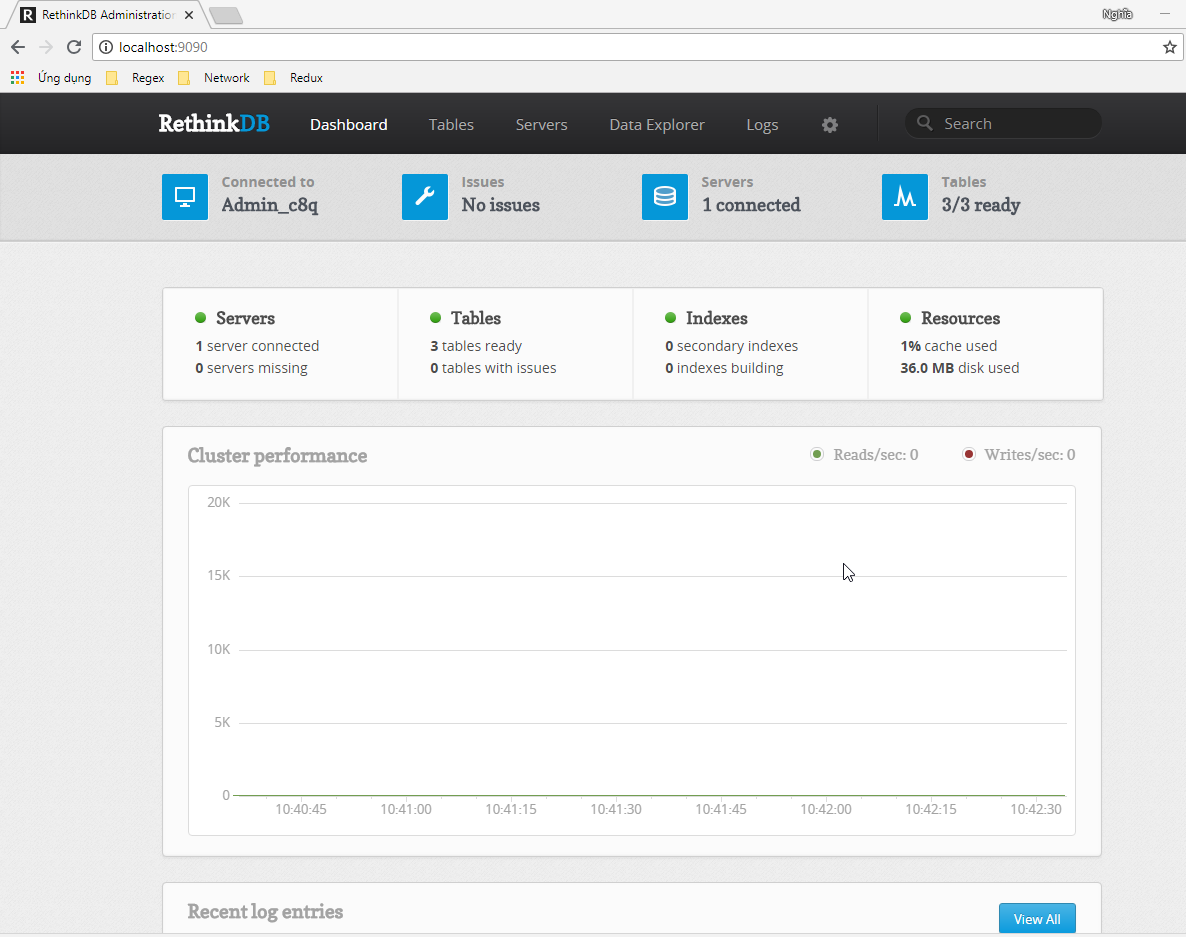


Đối với hệ điều hành windows 64 bit ta chỉ cần tải file nén giải nén ra chay file exe.



RethinkDB báo như thế này tức là quá trình cài đặt đã hoàn tất để kiểm tra ta có thể chạy trình duyệt web và vào url : localhost:port trong đó port là administrative http connections. Ta có thể chỉnh port lại bằng cách chạy lệnh : rethinkdb.exe --<loại port>-port <port muốn chỉnh>

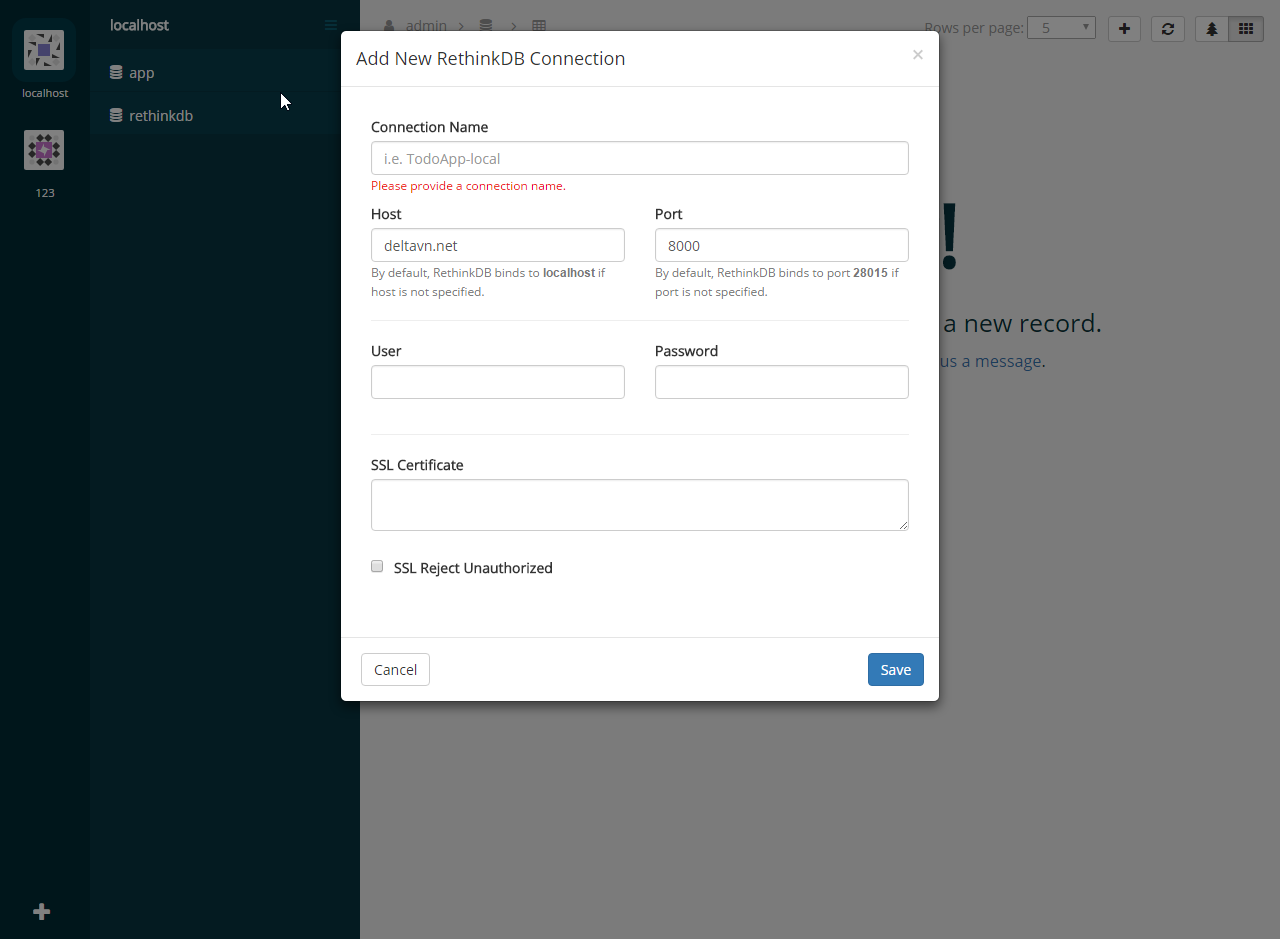
Ví dụ : rethinkdb.exe --bind all --http-port 9090 thay đổi port admistrative http connections thành 9090 (mặc định là 8080)



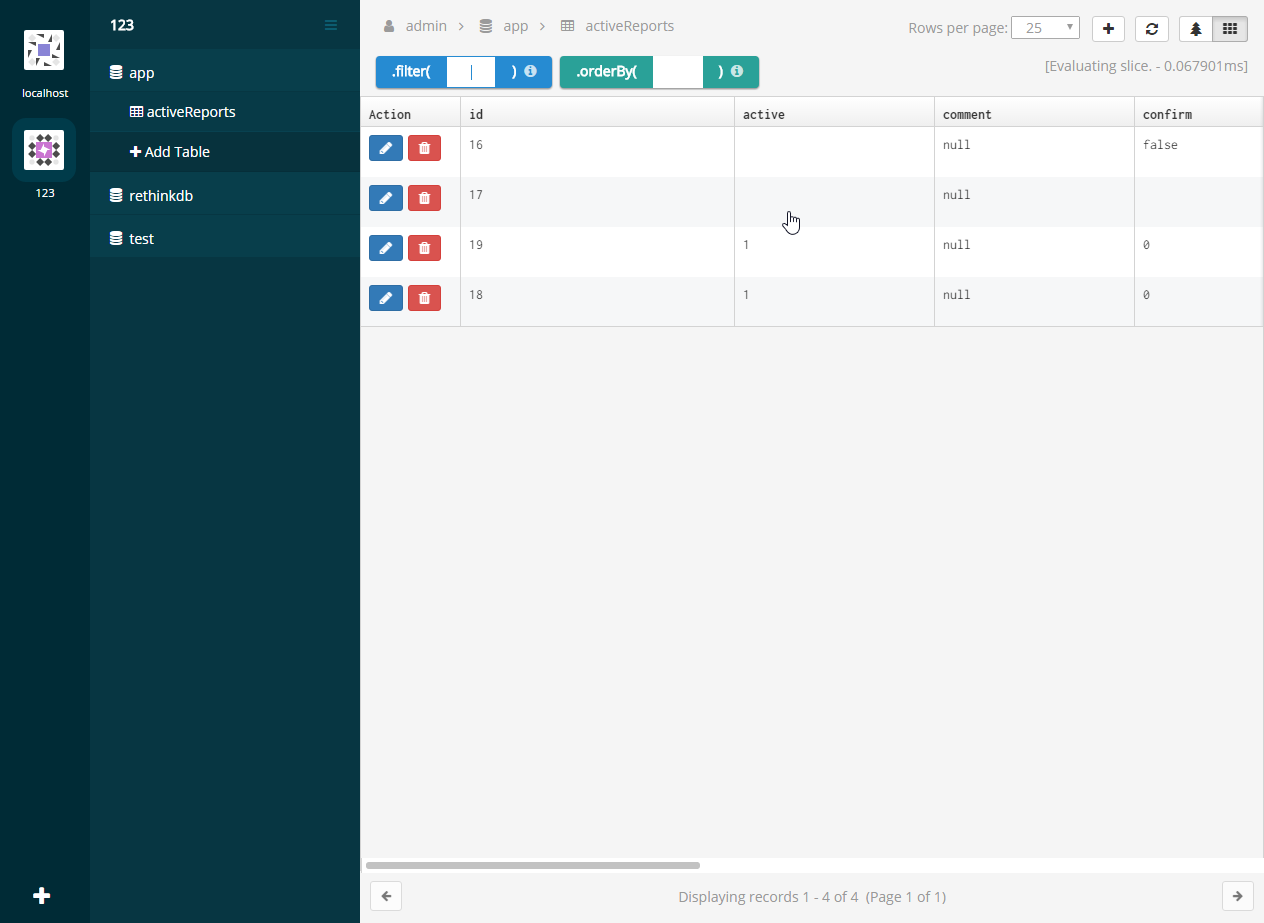
## Cài đặt ReQL Pro :

Công cụ này giống như MSSQL explorer là giao diện UI cho phép chỉnh sữa xem dữ liệu trong cơ sở dữ liệu một cách trực quan. Có thể tải công cụ này tại : <https://reqlpro.com/>

Để sữ dụng công cụ này đầu tiên ta phải tạo một kết nối bằng cách ấn dấu cộng phái trái màn hình : nhập vào host và port của database muốn truy cập :



Giao diện của công cụ cho phép thêm xóa sửa dữ liệu trong database rethinkDB một cách dễ dàng :

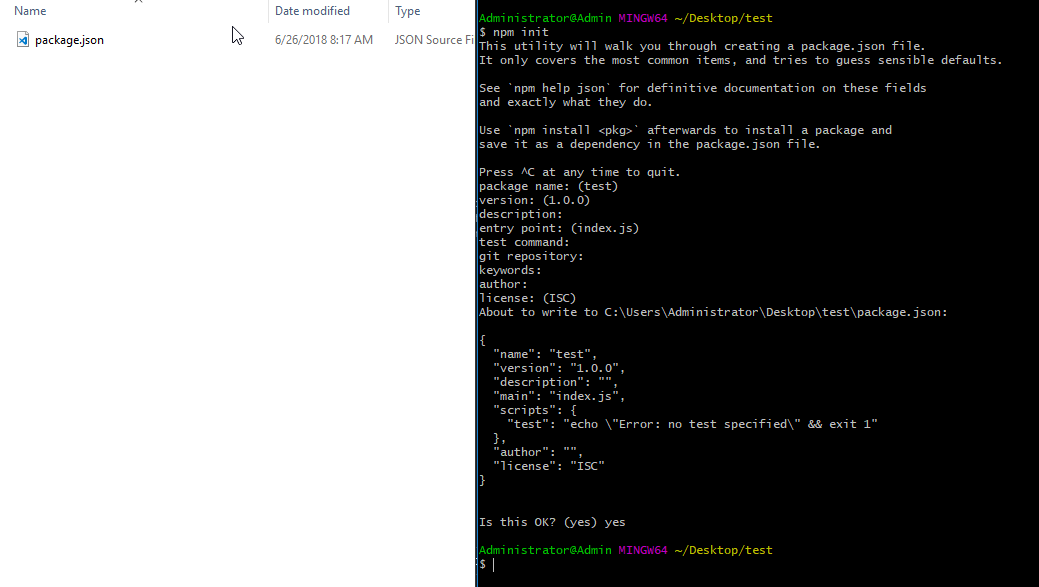


## Giới thiệu websocket :

Socket là công nghệ hỗ trợ giao tiếp hai chiều giữa client và server bằng cách sử dụng một TCP socket để tạo một kết nối hiệu quả và ít tốn kém. Kết nối này sẽ duy trùy trong suốt quá trình chương trình hoạt động hoặc khi phía client chủ động đóng kết nối. Websocket là một dạng socket chỉ chạy trên công nghệ web ( react native là một ngoại lệ do nó cũng là một công nghệ web ).

## Cài đặt websocket trên nodejs

Để tạo một project nodejs (đã hướng dẫn cài đặt node tại chương hai) ta chọn một folder trống và chạy lệnh npm init và nhập các thông tên cần thiết chọn yes. Sau khi tao xong project thì file package.json được tạo sẽ chứa các thông tin dự án quan trọng nhất là các gói dependecency của dự án.



Các dependency tiếp theo ta cần cài đặt là :

* RethinkDB driver: cho phép truy cập vào cơ sở dữ liệu rethinkDB
  + npm install rethinkdb
* Websocket: cho phép tạo websocket trên nền nodeJS : npm install ws
  + npm install npm install --save ws

Các dependency này có thể có dependency của depdency : ví dụ bootstrap yêu cầu jquery ta sẽ phải cài dependency của bootstrap là jquery. Lệnh –save cho phép cài dependency của dependency ta muốn cài đặt.

Để yêu cầu một gói trong JavaScript (giống using namespace trong C# hay import trong Java ta dung require)

const websocket = require('ws')

const r = require('rethinkdb')

Hàm async **đằng sau một hàm** cho phép sử dụng lệnh await cho phép đợi hành động không đồng bộ. hành động dưới đây : kết nối cơ sở dữ liệu tại host deltavn.net port 8000 là một hành động không đồng bộ. về mặt lý thuyết thì **tất cả các hành động đọc ghi dữ liệu không phải trên RAM đều gọi là không đồng bộ** do RAM cho tốc độ rất nhanh có khi vài mili giây trong khi trên đĩa cứng hay qua mạng không nhanh như vậy.

r.connect({host: 'deltavn.net',port: 8000})

Mỗi kết nối vào websocket sẽ được mở một cái kết tới cơ sở dữ liệu rethinkDB. Lệnh **.on** cho phép gán sự liệu vào một đối tượng. Tham số của lệnh này gồm :

* Tên sự kiện
* Hàm sẽ chạy khi sự kiện xảy ra : đây cũng là một hành động không đồng bộ.

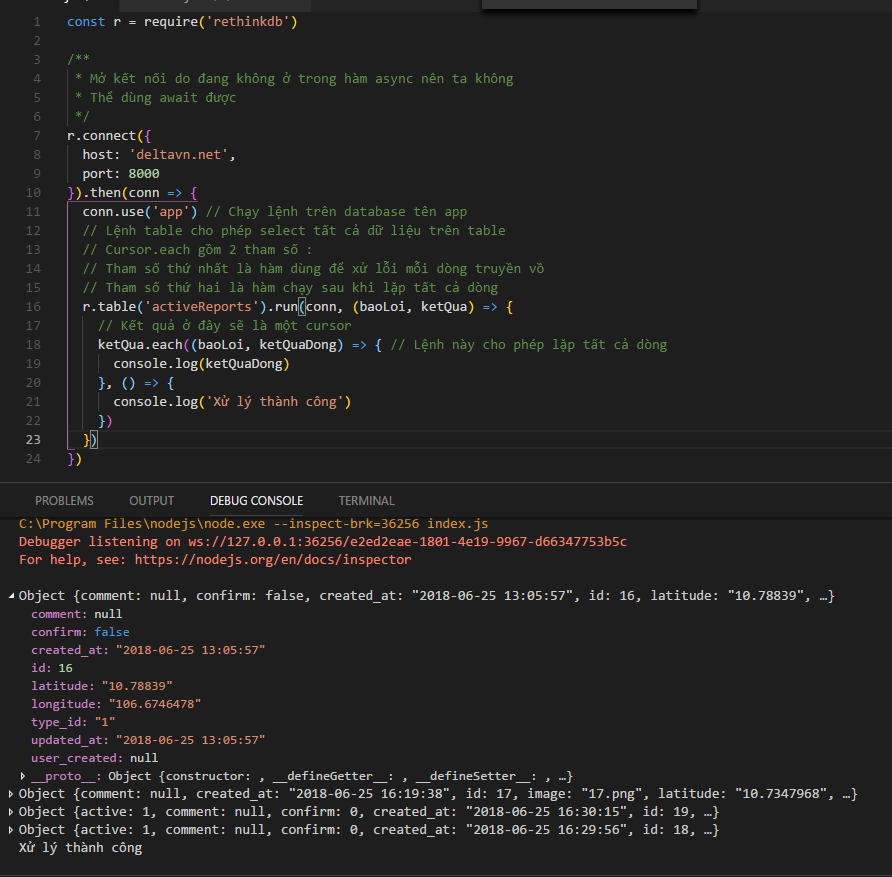
Sự kiện websocket sữ dụng bao gồm :

* Connection
  + tham số: đầu tiên biến chứa client truyền vào.
  + Khi : client kết nối tới websocket ở server
* Close
  + Khi: client đóng kết nối. Có thể do nhiều nguyên nhân : mất mạng, đóng phần mềm.

## Cú pháp trong rethinkdb

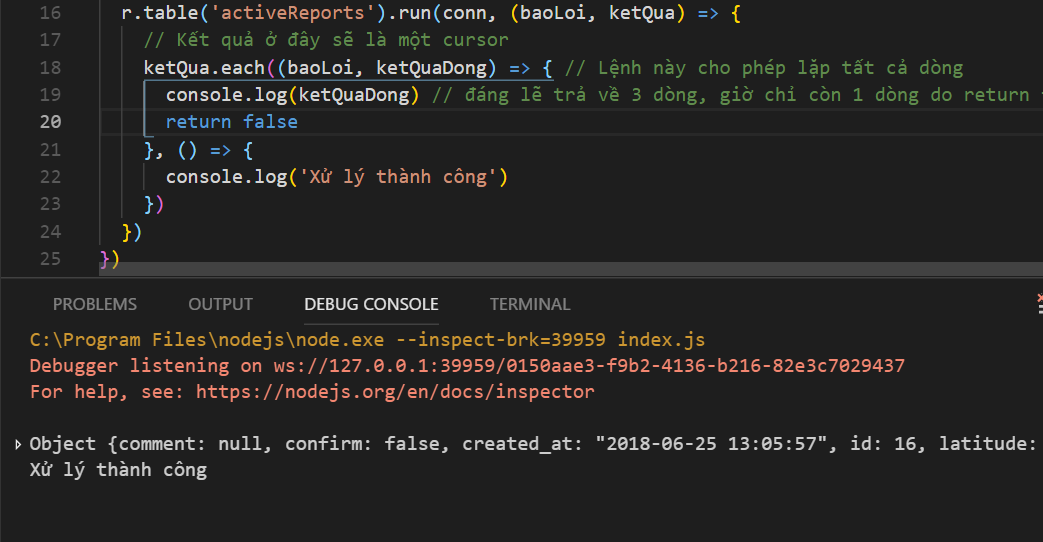
Một lênh truy cập rethinkDB có cú pháp :

R.<các giao thức truy vấn>.run(<kết nối>, <hàm chạy khi truy vấn dữ liệu thành công>



## Change feed trong rethinkDB

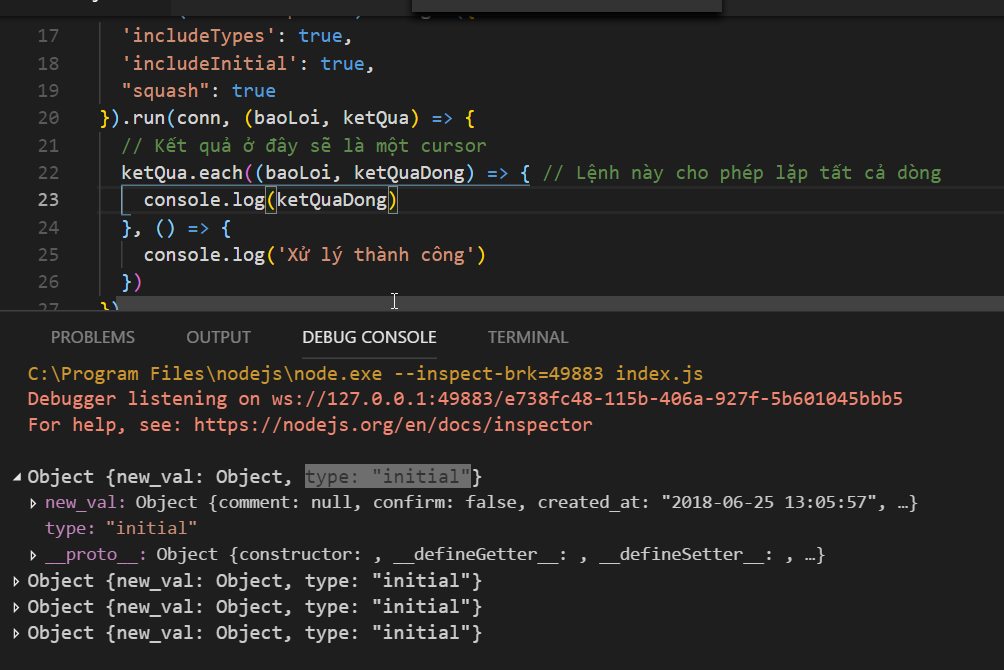
Cursor trong cơ sở dữ liệu là một con trỏ cho phép duyệt nhiều thanh phần dữ liệu từ đầu đến cuối. Trong quá trình con trỏ duyệt từ đầu đến cuối ta có thể return false khi ấy con trỏ sẽ dừng lại

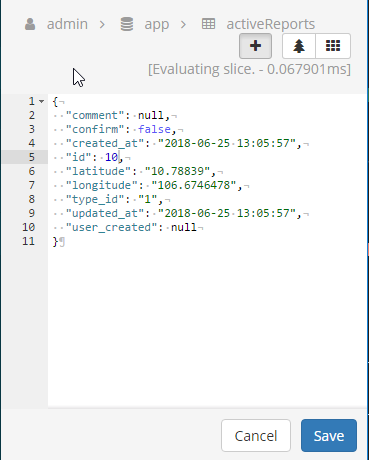


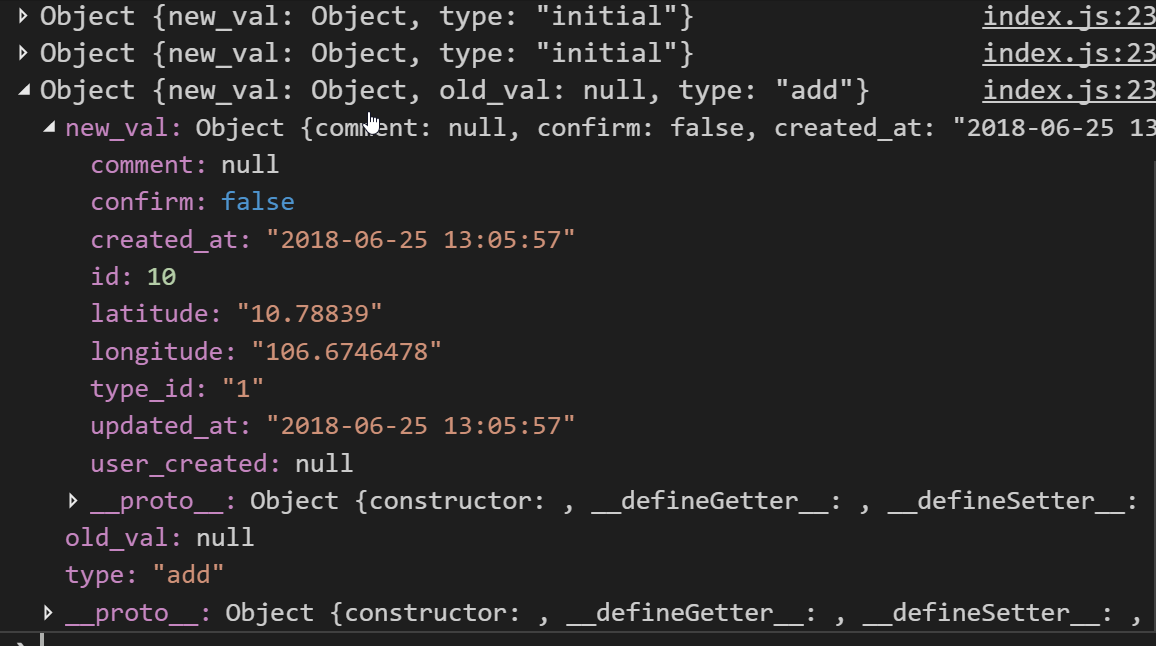
nếu thêm lệnh changes vào sau thâm số truy vấn, rethinkdb sẽ trả về một cusor. Changes cho phép truyền vào changes một object ( 1 đối tượng có thuộc tính nhưng không thuộc một class nào cả) option có các giá trị :

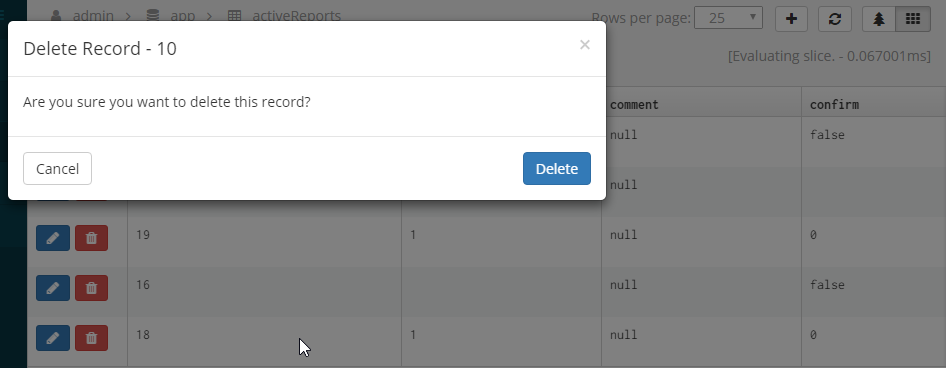
* includeTypes: loại dòng :
  + Initial: dòng có sẵn trong cơ sở dữ liệu
  + Change: dòng có giá trị bị thay đổi trong cơ sở dữ liệu
  + Delete: dòng vừa bị xóa trong cơ sở dữ liệu
  + Change: dòng bị chỉnh sửa trong cơ sở dữ liệu
* includeInitial: mặc định định các dòng có sẵn trong cơ sở dữ liệu không được lấy ra

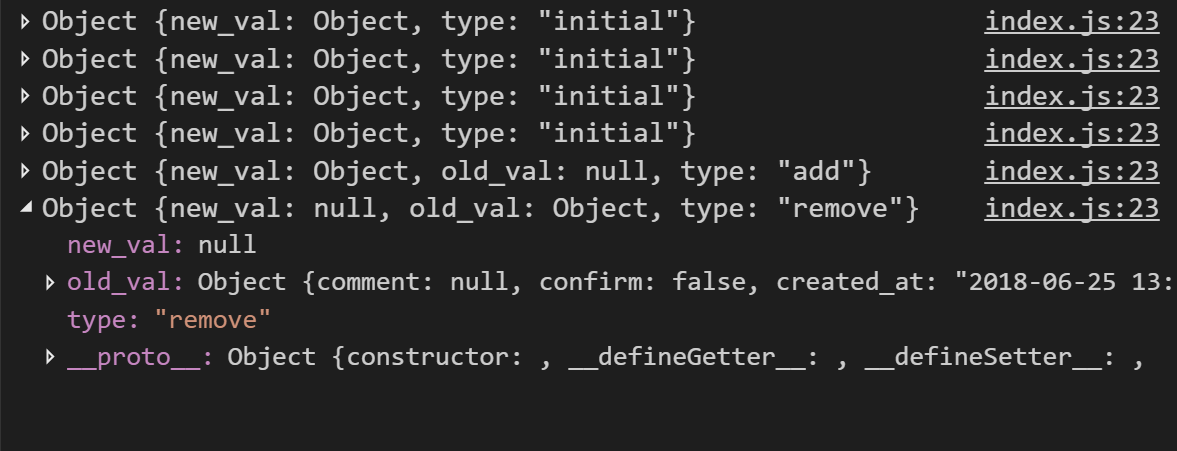
Thử thay đổi dữ liệu thông qua giao diện ReQL pro ta có thể thấy được các thay đổi được in ra bởi hàm console.log.

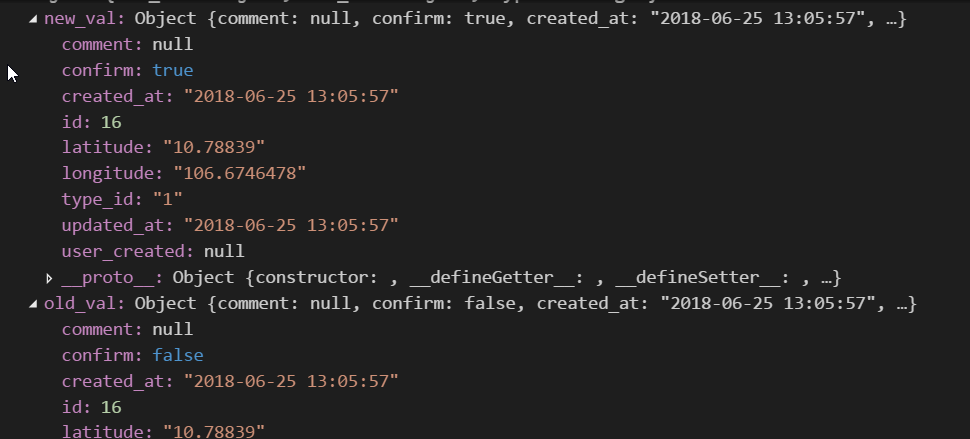












## Kết nối websocket nới rethinkDB : đây là đoạn mã đầy đủ

const websocket = require('ws')

const r = require('rethinkdb')

let port = process.env.PORT || 8000

async function start () {

try {

/\*\*

\* Tạo một websocket server

\*/

const wss = new websocket.Server({

port

}, () => {

console.log('websocket listen on port ' + port)

})

wss.on('connection', async (client) => {

console.log("client connected")

/\*\*

\* Mỗi khi một clien truy cập websocket mở kết nối rethinkdb

\*/

const conn = await r.connect({

host: 'deltavn.net',

port: 8000

})

conn.use('app')

/\*\*

\* Select dữ liệu trong rethinkDB

\* Lệnh changes sẽ tạo ra một cursor chứa tất cả thay đổi rethinkDB

\*/

r.table('activeReports')

.changes({

'includeTypes': true,

'includeInitial': true

})

.run(conn, async (err, cursor) => {

if (err) {

throw err

}

cursor.each((err, row) => {

if (err) {

throw err

}

client.on('close', () => {

client.terminate()

cursor.close()

conn.close()

return false // This will stop cursor from run

// If it still run, it will yield error cursor has been closed

console.log("client disconnected")

})

if (client.readyState === client.OPEN) {

client.send(JSON.stringify(row))

} else {

client.emit('close')

}

})

})

})

} catch (err) {

console.log(err)

}

}

start()

# Laravel

# Tính năng

## Google API :

### Tìm đường

### Tìm địa điểm

### Tìm địa điểm trong bán kính

## Báo cáo giao thông :

### Tạo báo cáo

### Thông báo báo cáo giao thông mới được tạo

### Cài đặt thông báo báo cáo giao thông

### Xác nhận báo cáo

### Hủy xác nhận

### Ngắt kích hoạt báo cáo

### Ngắt kích hoạt báo cáo tự động

### Thống kê và tìm kiếm báo cáo giao thông

## Phòng khám

### Tạo phòng khám

### Thông báo khi phòng khám được xác nhận

### Chỉnh sửa phòng khám

### Xác nhận phòng khám

### Hủy xác nhận phòng khám

### Xác nhận phòng khám tự động

## Tài khoản :

### Tạo tài khoản

### Đổi mật khẩu

### Cập nhật lại mật khẩu mới

### Đăng nhập

### Đăng nhập bằng google

# Thống kê

## Công nghệ sử dụng :

## Ứng dụng thực hiện :

## Hướng phát triển :