* **TUẦN 1.1: REACT NATIVE**

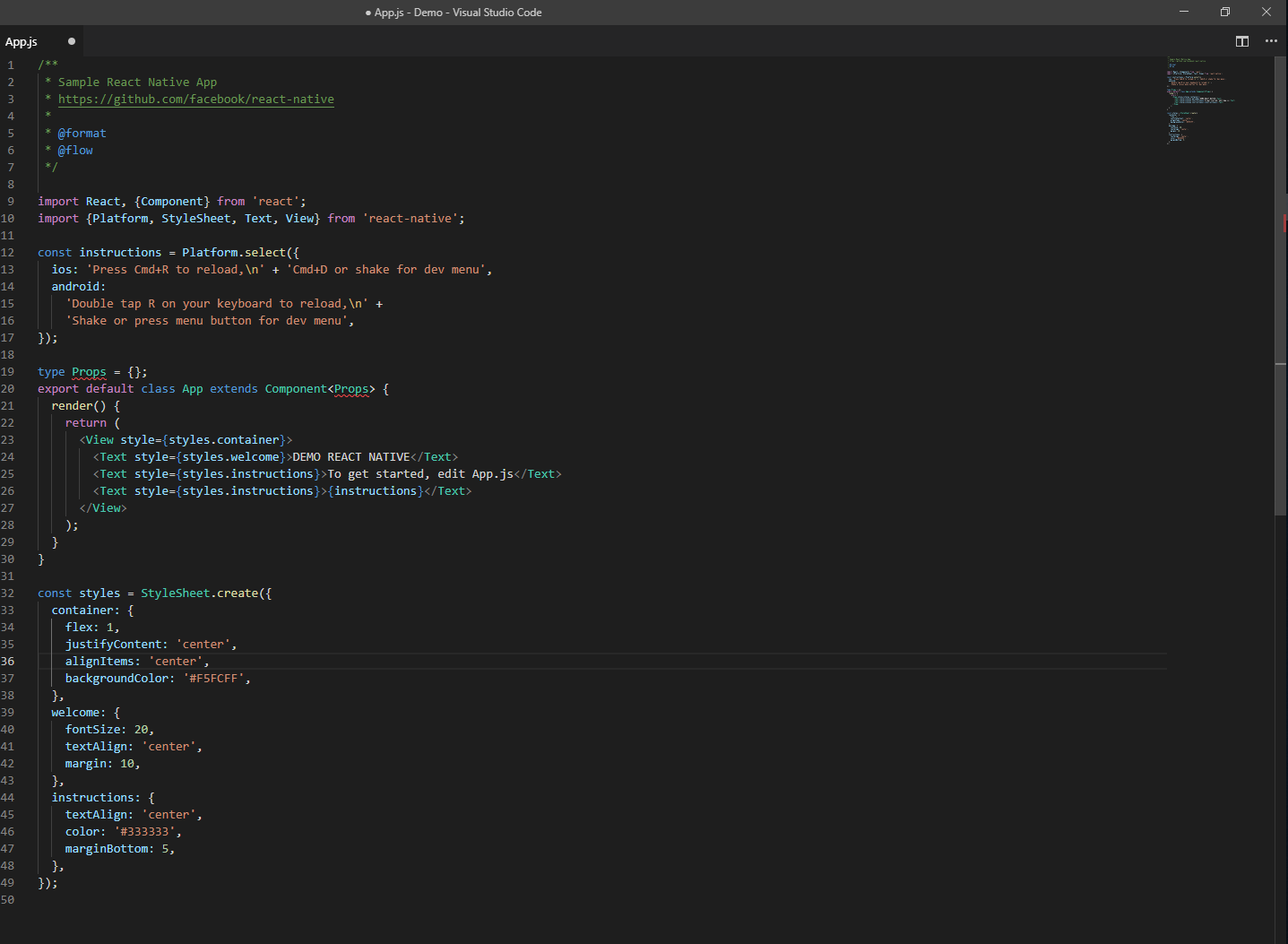
**REACT NATIVE LÀ GÌ?**

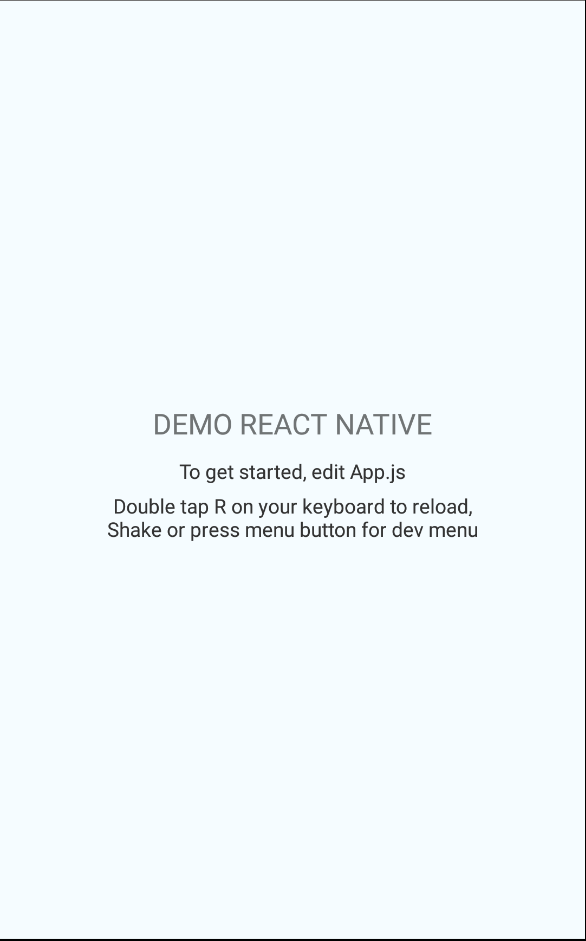
React Native là framework xây dựng ứng dụng di động native sử dụng Javascript do Facebook phát hành. Sử dụng React Native để xây dựng ứng dụng iOS và AndroidAndroid chỉ cần 1 ngôn ngữ javascript duy nhất. Để hiểu rõ về React Native là gì, chúng ta cần phân biệt sự khác nhau giữa ứng dụng Native và Hybrid.

## CÁCH HOẠT ĐỘNG CỦA REACT NATIVE

Ứng dụng viết bằng React Native được chia làm 2 phần: phần view(hiển thị) và phần xử lý.Phần hiển thị được biên dịch từ javascript sẽ map với những component của hệ thống ví dụ: điều hướng, tab, touch…Phần view này được lấy cảm hứng từ Virtual DOM của [React JS](https://www.businesscard.vn/blog/react-js-la-gi/), mọi xử lý view được thực hiện trên một cây DOM ảo, sau đó được React Native render lại bằng native view.Phần xử lý vẫn được thực hiện trực tiếp bằng ngôn ngữ javascript: ví dụ “1+1=2” :)), biểu thức này được xử lý dưới bộ core thực thi Javascript, không phải thông dịch qua Java hay Swift/Objective-C rồi mới làm phép tính đâu nhé.

**DEMO ỨNG DỤNG MẪU:**

****



* **TUẦN 1.2:**

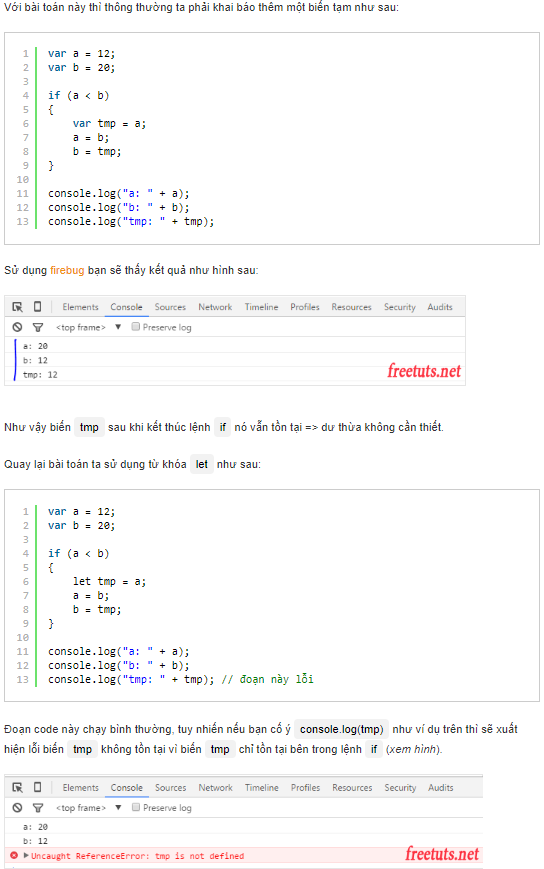
**ES6 LÀ GÌ?(khái niệm es6) :**

ES6 là chữ viết tắt của **ECMAScript 6,** đây được coi là một tập hợp các kỹ thuật nâng cao của Javascript và là phiên bản mới nhất của chuẩn **ECMAScript**. ECMAScript do hiệp hội các nhà sản xuất máy tính Châu Âu đề xuất làm tiêu chuẩn của ngôn ngữ Javascript. Bạn cứ nghĩ xem hiện nay có khá nhiều trình duyệt Browser ra đời và nếu mỗi Browser lại có cách chạy Javascript khác nhau thì các trang web không thể hoạt động trên tất cả các trình duyệt đó được, vì vậy cần có một chuẩn chung để bắt buộc các browser phải phát triển dựa theo chuẩn đó.

**TIỀM HIỂU VỀ BIẾN VAR, LET TRONG ES6:**

* **Biến Var, Let trong cs6:**

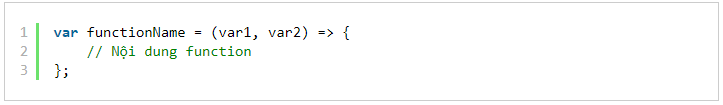
Với phạm vi hoạt động hẹp như vậy thì let thường dùng để khai báo các biến mang tính chất tạm thời, nghĩa là nó chỉ sống trong một phạm vi hoạt động của khối đó thôi, không sử dụng qua vị trí khác.



Rõ ràng việc sử dụng từ khóa let có những ưu điểm và nhược điểm khác nhau, vì vậy không phải thấy nó mới được bổ sung vào là sử dụng đâu nhé :). Ví dụ nếu bạn sử dụng một biến cho toàn ứng dụng và biến đó thay đổi liên tục thì bắt buộc phải sử dụng var để khai báo.

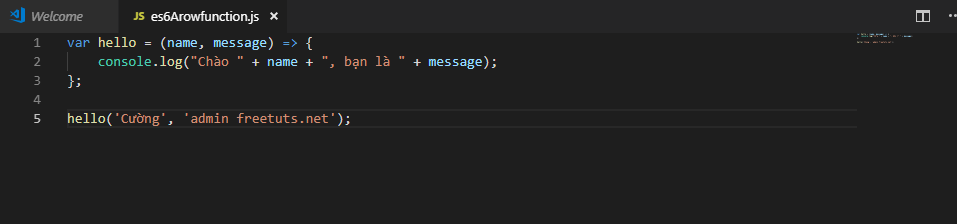
# **ARROW FUNCTION TRONG ES6:**

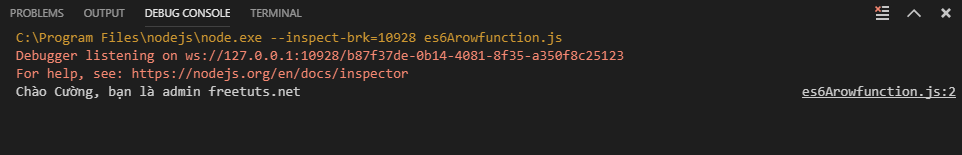
Cú pháp căn bản nhất của arrow function như sau:



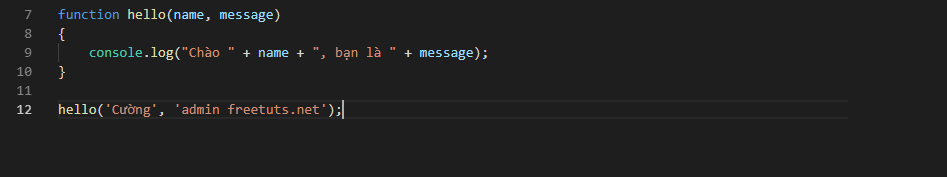
**Ví dụ**: Viết arrow function in ra câu chào và so sánh với cách tạo function thông thường.

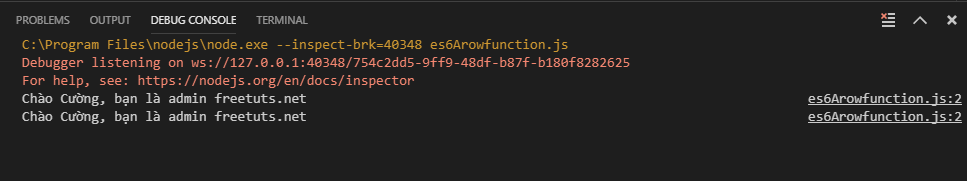
**Arrow function**:





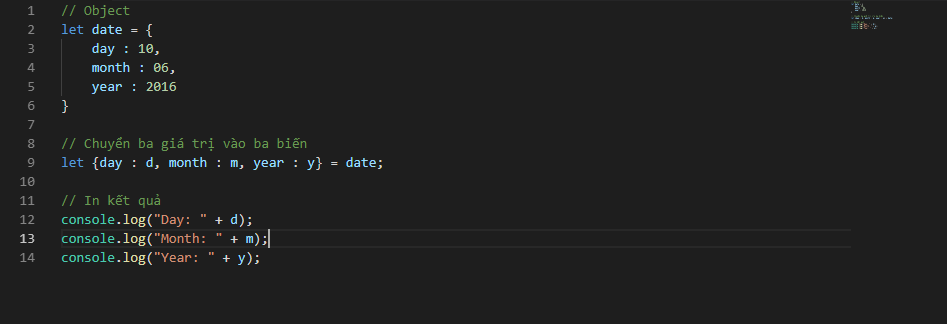
**Normal function**:



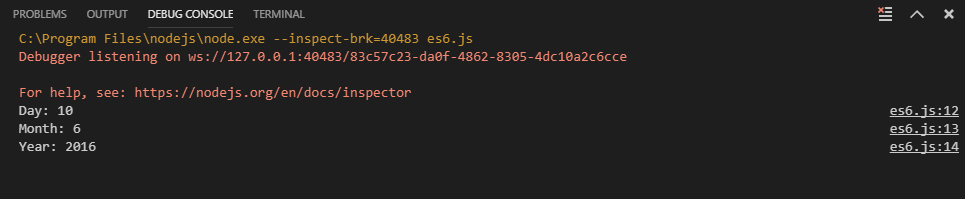
Nói là ES6 nhưng thực chất hầu hết các trình duyệt hiện nay đều đã hỗ trợ ES6 nên bạn có thẻ coi như đây là một tính năng mới của Javascript.

# **KIỂU DỮ LIỆU OBJECT TRONG ES6:**

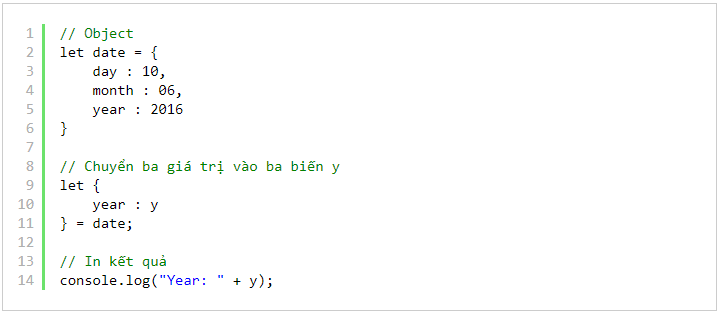
Ngoài mảng ra thì bạn có thể tách biến từ object:



Kết quả:



Với trường hợp này thì bạn có thể lấy một phần tử bất kì chứ không bắt buộc theo thứ tự như bên mảng:



* **TUẦN 2.1: GIAO DIỆN**

1. **Màn hình đăng nhập**

Màn hình đăng nhập gồm nhập tên id và mật khẩu cho phép người dùng đăng nhập

1. Màng **hình menu**

Liệt kê các chú năng của ứng dụng và gồm có thông tin của khách hàng

Màn hình này liệt kê tất cả chức năng cho phép người dùng click vào nó để thực hiện chức năng tương ứng của ứng dụng

1. **Màn hình ghi chú**

Màn hình ghi chú là màn hình chức năng khi được click từ màn hình menu

Màn hình này đảm nhận chức năng lưu trữ lại các ghi chú cần thiết của người dùng lên internet để cho phép người dùng xem lại ở bất kì đâu họ cần

1. **Màn hình nhiệm vụ**

Màn hình nhiệm vụ là màn hình chức năng khi được click từ màn hình menu

Màn hình này cho phép người dùng xem lịch hiện tại và thêm vào các nhiệm vụ quan trong vào ngày trên lịch để biết và thực hiện trong đời sống công việc

1. **Màn hình thùng rác**

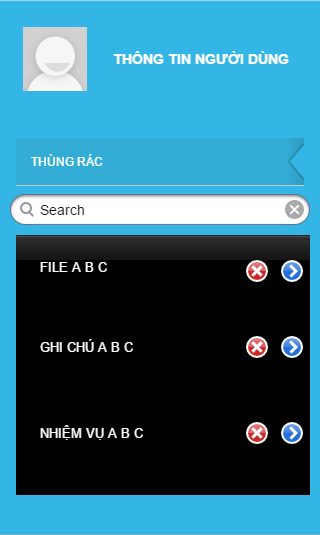
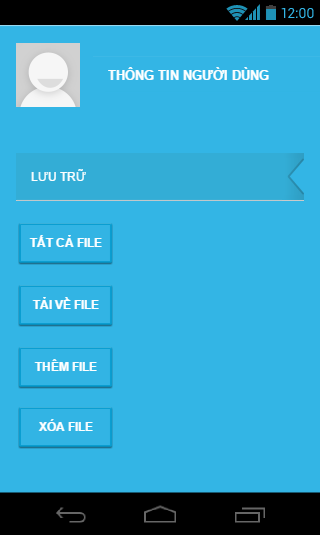
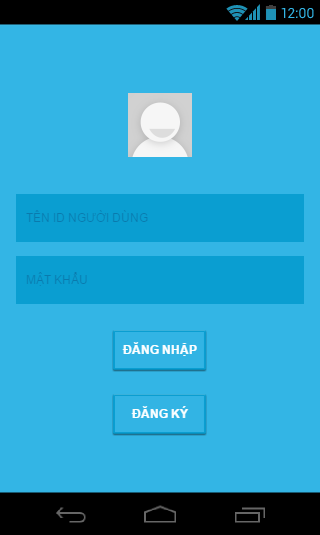
Màn hình thùng rác là màn hình chức năng khi được click từ màn hình menu

Màn hình này đảm nhận chức năng lưu lại tất cả các file vv được xóa gần đây để người dùng có thể quản lý được lịch sử xóa của mình

1. **Màn hình lưu trữ**

Màn hình lưu trữ là màn hình chức năng khi được click từ màn hình menu

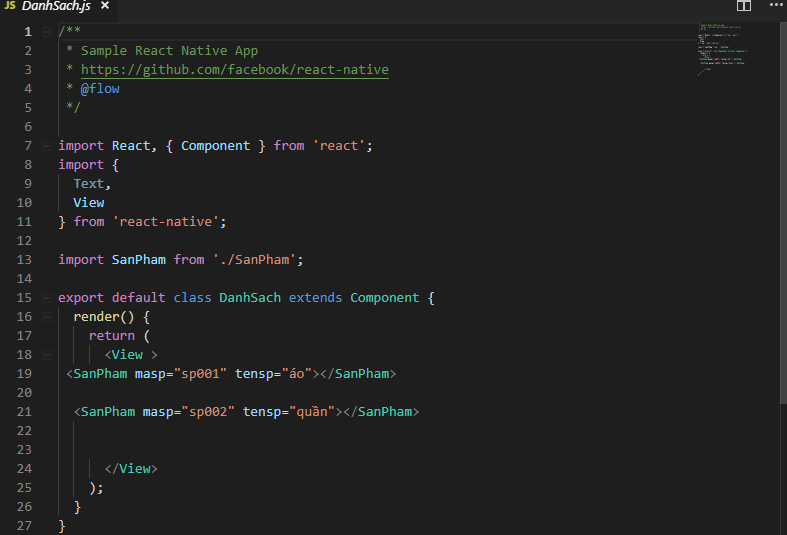
Màn hình này đảm nhận chức năng lưu trữ đám mấy các tài liệu của người dùng và cho phép người dùng có thể tải về hoặc xem chúng ở bất kì đâu

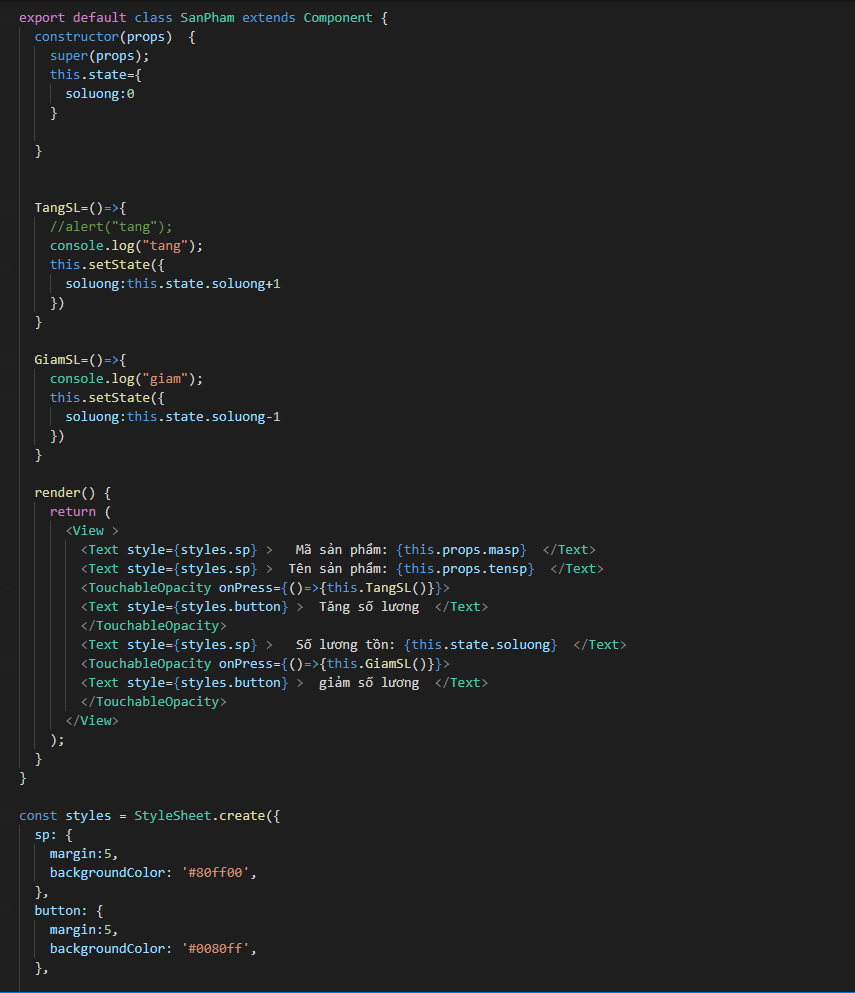
****

* **TUẦN 2.2:**

CHẠY ỨNG DỤNG DEMO SẢN PHẨM:







TÌM HIỂU VÒNG ĐỜI COMPONENT:

## constructor(props)

* Được gọi khi một thể hiện của component được tạo ra.
* Có thể dùng để khởi tạo state cho component.
* Cũng có thể dùng để "bind" các hàm của component.
* Nếu phải cài đặt hàm này thì phải khai báo 1 tham số props cho nó và phải gọi super(props) đầu tiên.
* Nếu không làm gì thì không phải cài đặt hàm này.

## componentWillMount()

* Được gọi trước khi render().
* Dùng để đăng ký các sự kiện toàn cục.
* Dựa vào các props để tính toán và set lại state

## render()

* Hàm này bắt buộc phải có trong component().
* Trả về 1 đối tượng JSX (có thể lồng các đối tượng với nhau nhưng phải có 1 đối tượng gói tất cả các đối tượng lại) để hiển thị hoặc null / false nếu không muốn hiển thị gì.
* Không được gọi setState() trong hàm này (cũng như trong các hàm mà hàm này gọi đến), bởi khi gọi setState() thì hàm render sẽ được gọi => gây ra lặp vô hạn.

## componentDidMount()

* Ngay sau khi hàm render được gọi đến lần đầu tiên chạy xong thì hàm này sẽ được chạy.
* Thường dùng để fetch dữ liệu từ server và sau đó setState để render dữ liệu ra.
* Đến đây thì các phần tử đã được sinh ra rồi, và có thể tương tác với DOM bằng JS trong hàm này.

## componentWillReceiveProps(nextProps)

* Hàm này được chạy khi mà props của component đã được sinh ra có sự thay đổi.
* Phải gọi setState() nếu muốn render lại.

## shouldComponentUpdate(nextProps, nextState)

* Được gọi trước render.
* Trả về true / false. Nếu false thì sẽ không render lại. Mặc định là true.

## componentWillUpdate(nextProps, nextState)

* Được gọi ngay sau shouldComponentUpdate() nếu hàm này trả về true.
* Không gọi setState() trong hàm này bởi hàm này là để chuẩn bị update cho đối tượng chứ không phải tạo ra 1 update mới, sẽ tạo ra lặp vô hạn.
* Hàm render sẽ được gọi ngay sau hàm này.

## componentDidUpdate(prevProps, prevState)

* Được gọi ngay sau render() từ lần render thứ 2 trở đi.
* Đây cũng là 1 cơ hội để thao tác với các phần tử DOM bằng JS.

## componentWillUnmount()

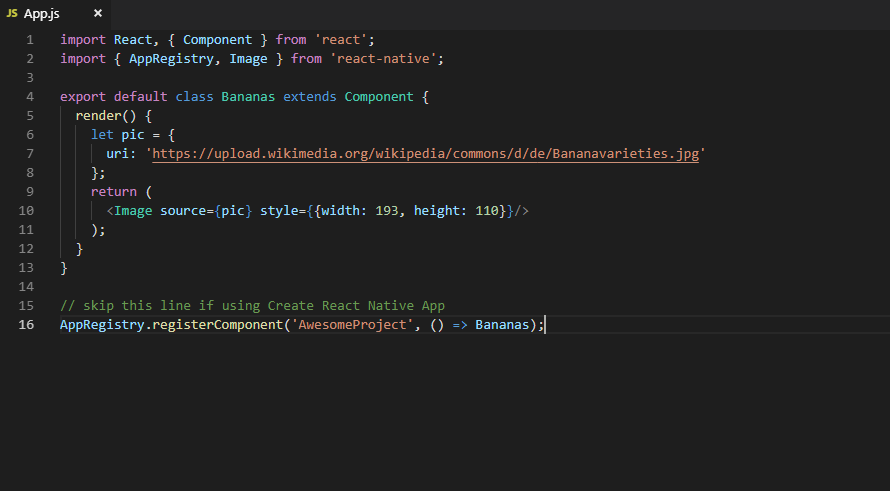
* Được gọi khi 1 component được loại bỏ khỏi DOM.
* Thực hiện các thao tác dọn dẹp như huỷ các timer, loại bỏ các phần tử thừa, ...

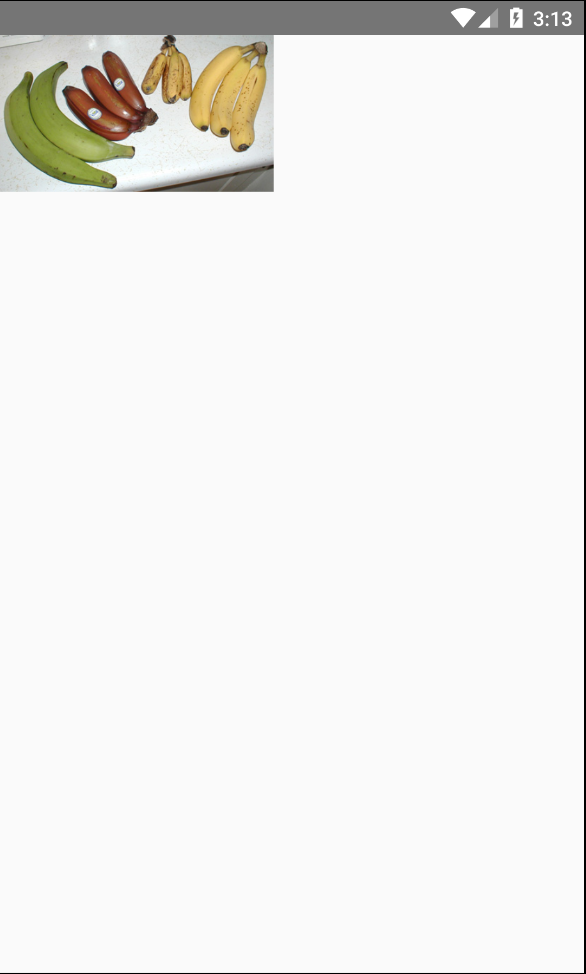
**PROPS VÀ STATE:**

props là viết tắt của Properties. Một điều mà bạn cần phải nhớ khi sử dụng props đó là không bao giờ nên thay đổi giá trị của nó, hay nói cách khác, đây là một dữ liệu **immutable**.

Các component nhận props từ component cha. Bạn không được thay đổi giá trị của props trong các component này mà chỉ được phép đọc giá trị ra thôi. Trong React thì dữ liệu sẽ đi theo một chiều, có nghĩa là từ component cha => các component con.

Bạn có thể tạo ra component sử dụng props. Ý tưởng của props đó là việc trừu tượng hoá các component để có thể sử dụng được ở nhiều chỗ khác nhau trong app. Hãy xem một ví dụ đơn giản sau:



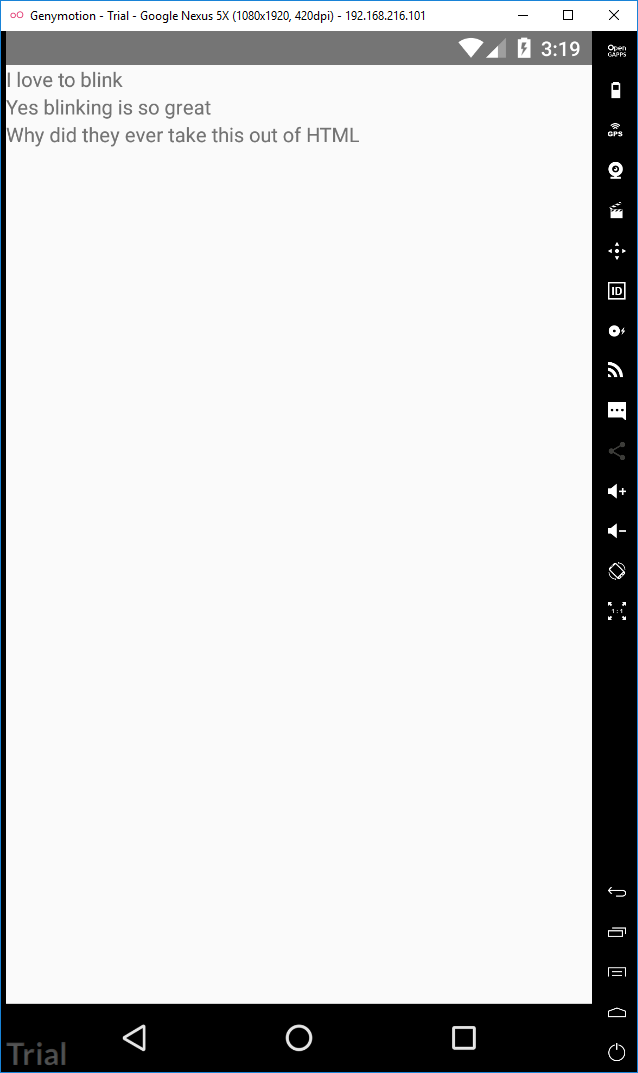
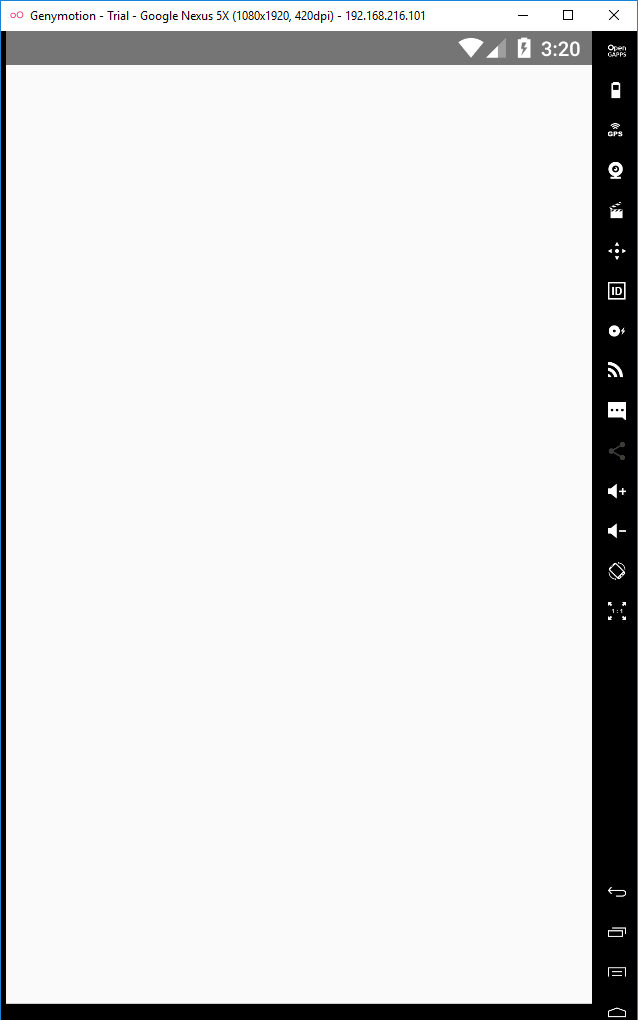


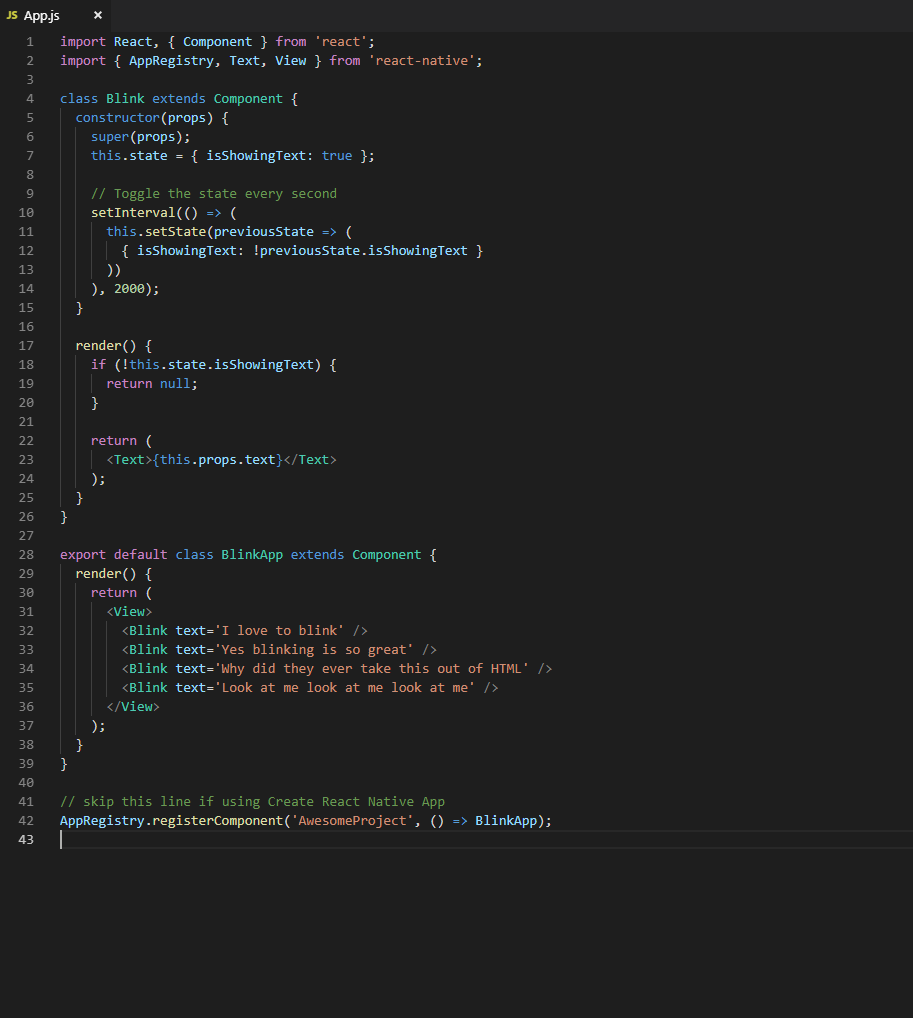
state thì hoạt động khác với props. state là dữ liệu nội bộ của một Component, trong khi propslà dữ liệu được truyền cho Component. Chính vì vậy chúng ta hoàn toàn có thể thay đổi state, và coi nó là một kiểu dữ liệu **mutable**.

Tuy vậy, hãy nhớ rằng đừng bao giờ thay đổi trực tiếp biến this.state. Thay vào đó hãy dùng hàm setState để cập nhật giá trị. Sở dĩ chúng ta cần dùng hàm này là do nó sẽ kích hoạt việc render lại component và tất cả component con nằm trong nó, còn thay đổi this.state thì không. Còn một vấn đề nữa, đó là setState chạy bất đồng bộ, vậy nên nếu bạn tiến hành đọc ra giá trị state ngay sau khi setState thì chưa chắc giá trị sẽ được update lên mới nhất đâu.

Vậy thì khi nào nên sử dụng state? Bất cứ khi nào dữ liệu trong một component có thể thay đổi (bởi chính nó).

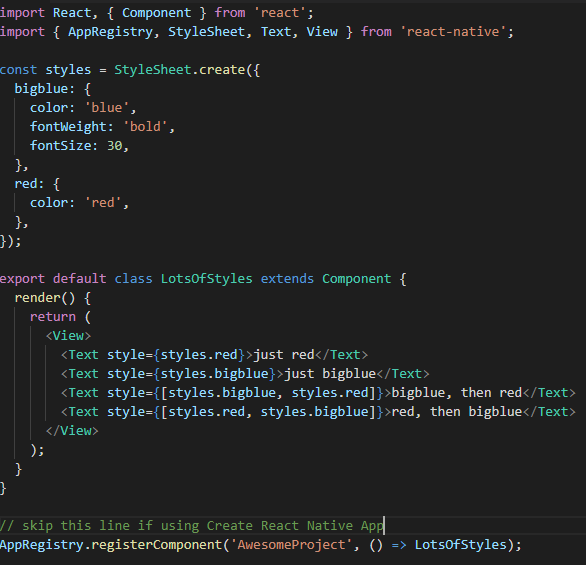
Người dùng tương tác với component là một ví dụ về cách mà state hoạt động. Ấn vào một button, checkbox, thêm text vào form,... là những trường hợp mà chúng ta có thể sử dụng state.

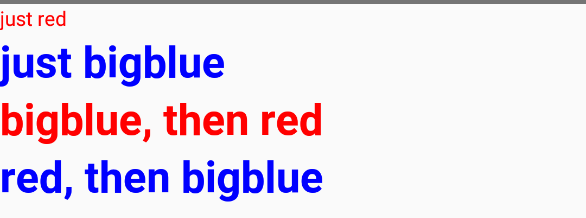
 



STYLE TRONG REACT NATIVE:

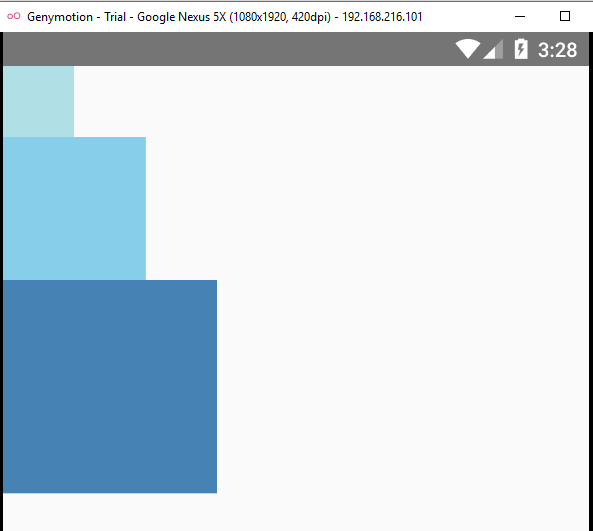
Các styleprop có thể là một đối tượng JavaScript cũ đồng bằng. Đó là cách đơn giản nhất và những gì chúng ta thường sử dụng ví dụ mã. Bạn cũng có thể vượt qua một loạt các kiểu - kiểu cuối cùng trong mảng được ưu tiên, vì vậy bạn có thể sử dụng kiểu này để kế thừa kiểu.



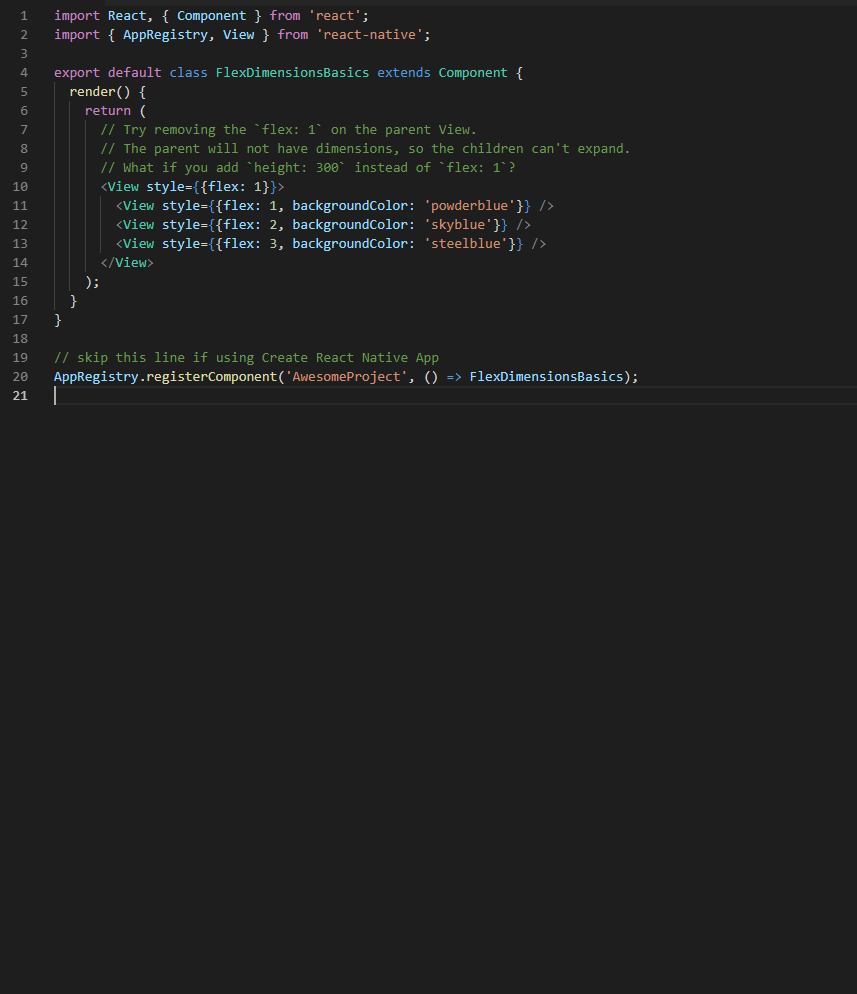


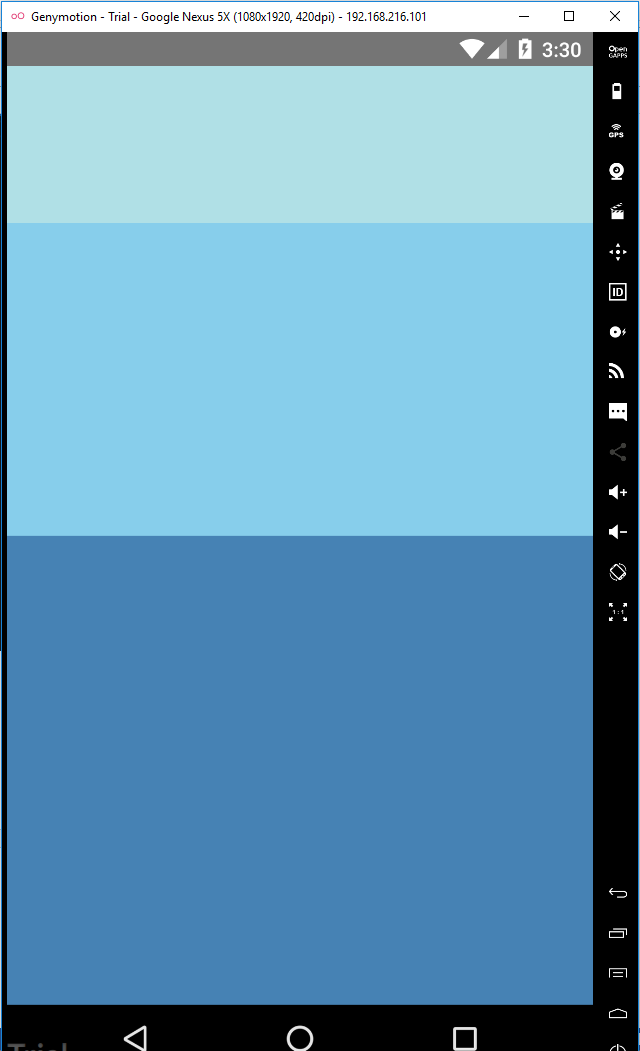
WIDTH HIGHT VÀ FLEX**:**



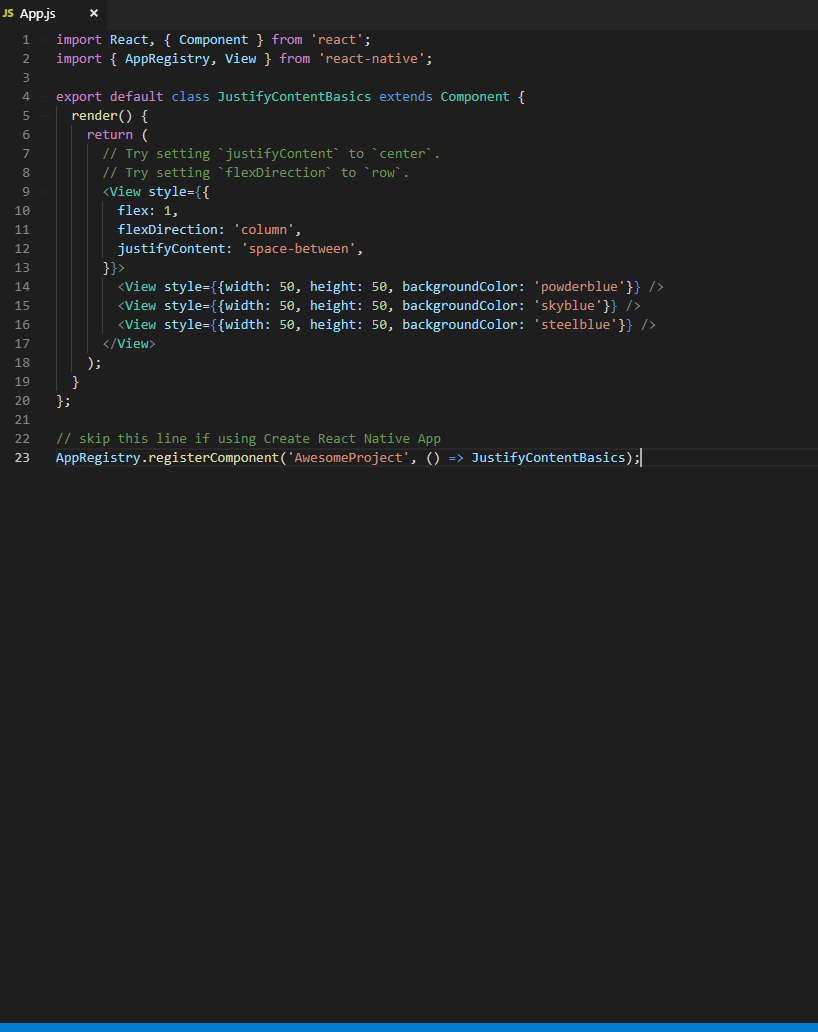


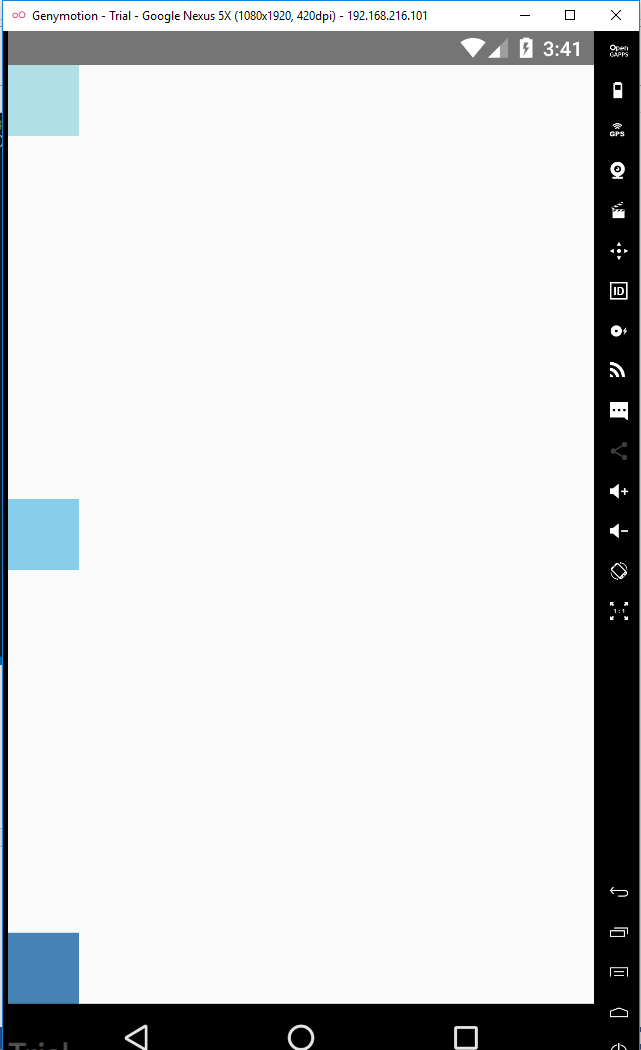
FLEX: Sử dụng flextheo kiểu của một thành phần để mở rộng và thu nhỏ linh hoạt dựa trên không gian có sẵn. Thông thường bạn sẽ sử dụng flex: 1, thông báo cho một thành phần để lấp đầy tất cả không gian có sẵn, được chia sẻ đồng đều giữa các thành phần khác có cùng cha mẹ. Giá trị càng lớn flex, tỷ lệ không gian của một thành phần sẽ càng cao so với anh chị em của nó.



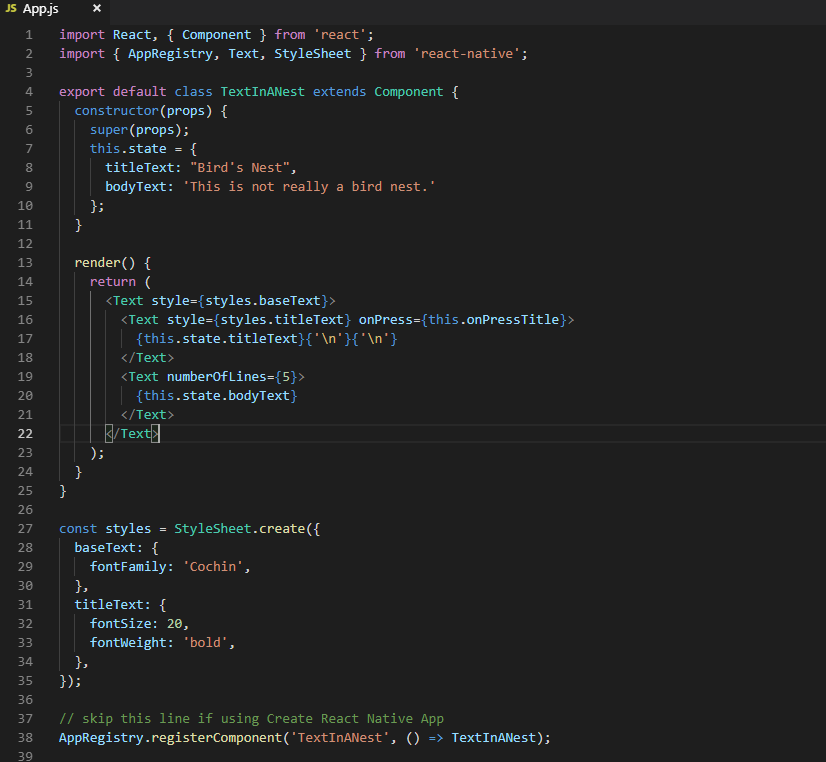


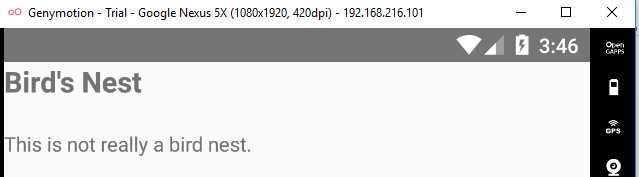
FLEXBOX: là đối tượng căn chỉnh cho giao diện trong react – native



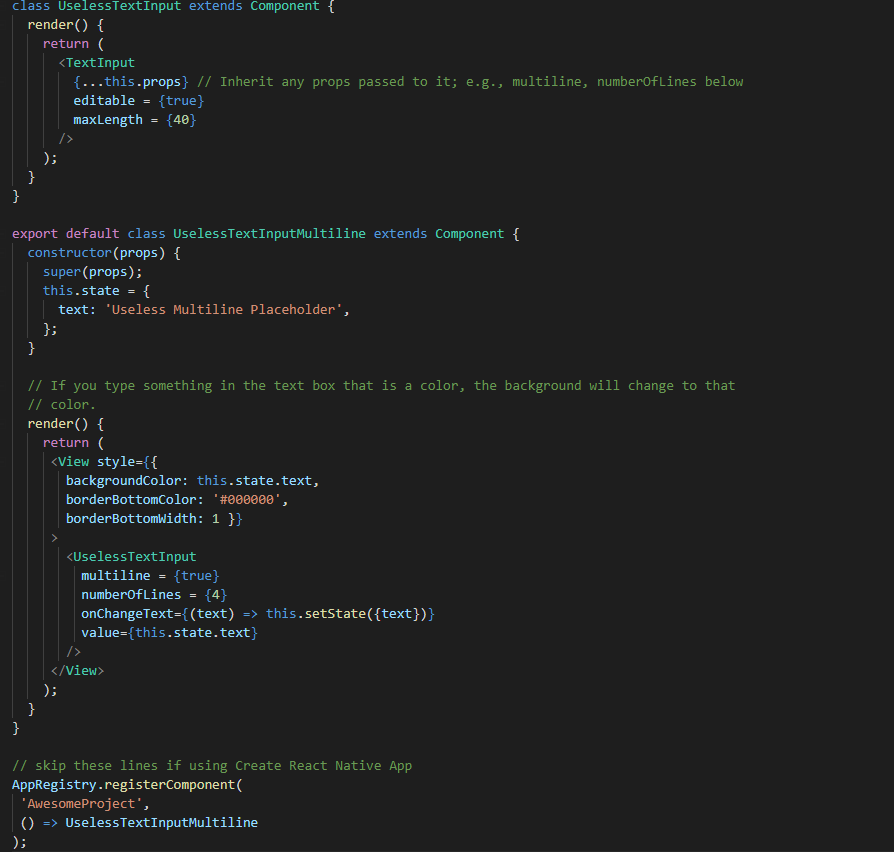


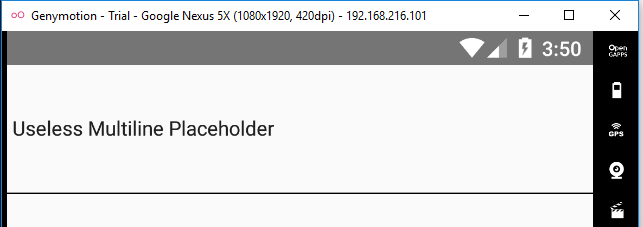
* **TUẦN 3.1: tìm hiểu về các component**
* Text component:



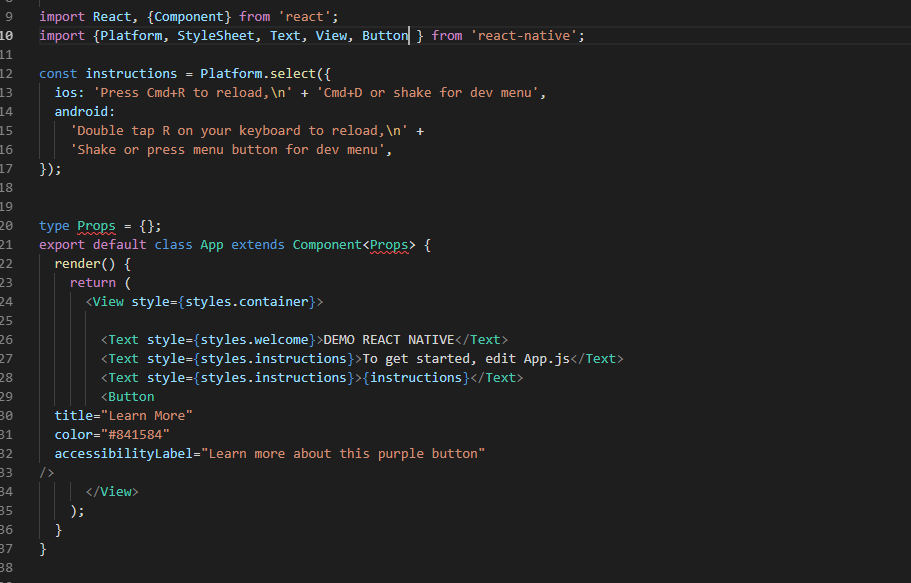


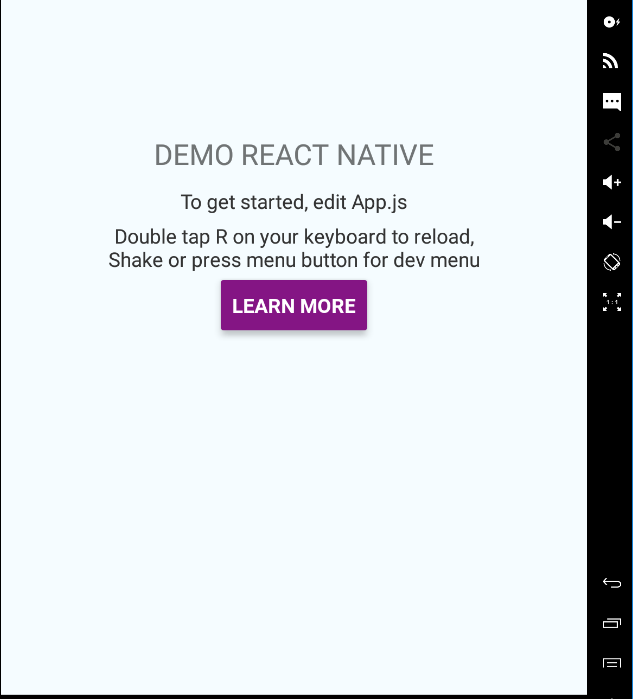
* Text input: Một thành phần nền tảng để nhập văn bản vào ứng dụng thông qua bàn phím. Đạo cụ cung cấp khả năng cấu hình cho một số tính năng, chẳng hạn như tự động sửa, tự động viết hoa, văn bản giữ chỗ và các loại bàn phím khác nhau, chẳng hạn như bàn phím số.



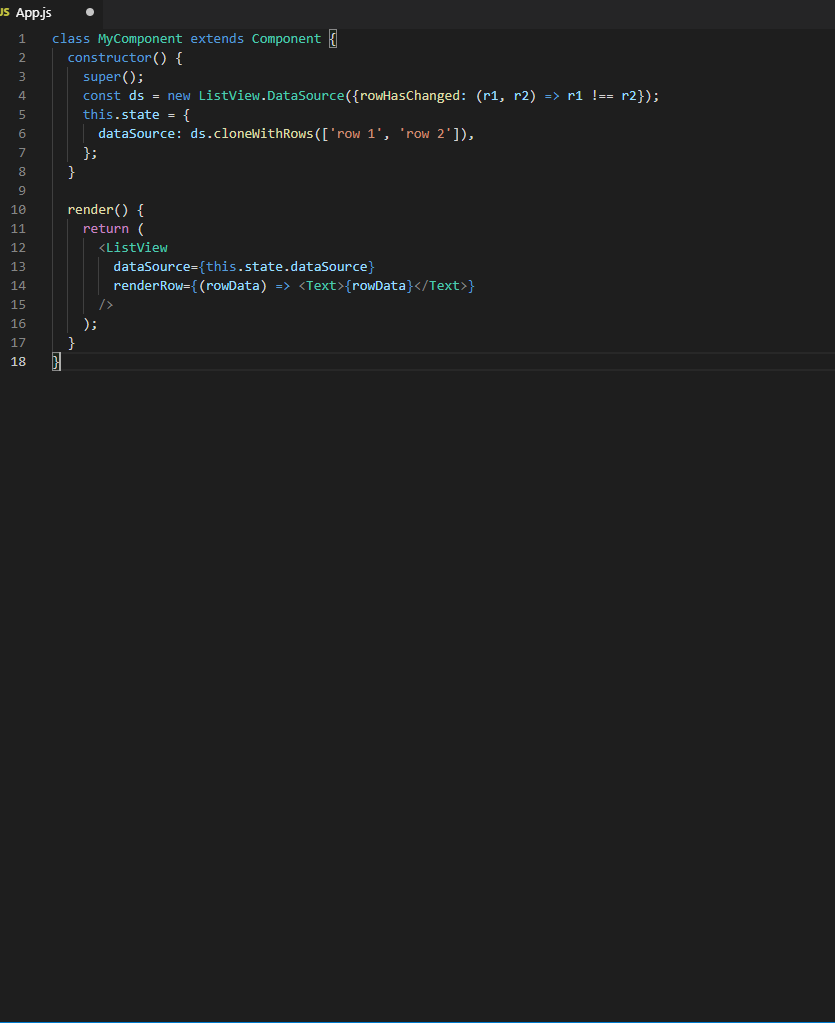


* Button component: Một thành phần nút cơ bản sẽ hiển thị độc đáo trên bất kỳ nền tảng nào. Hỗ trợ mức độ tùy biến tối thiểu.





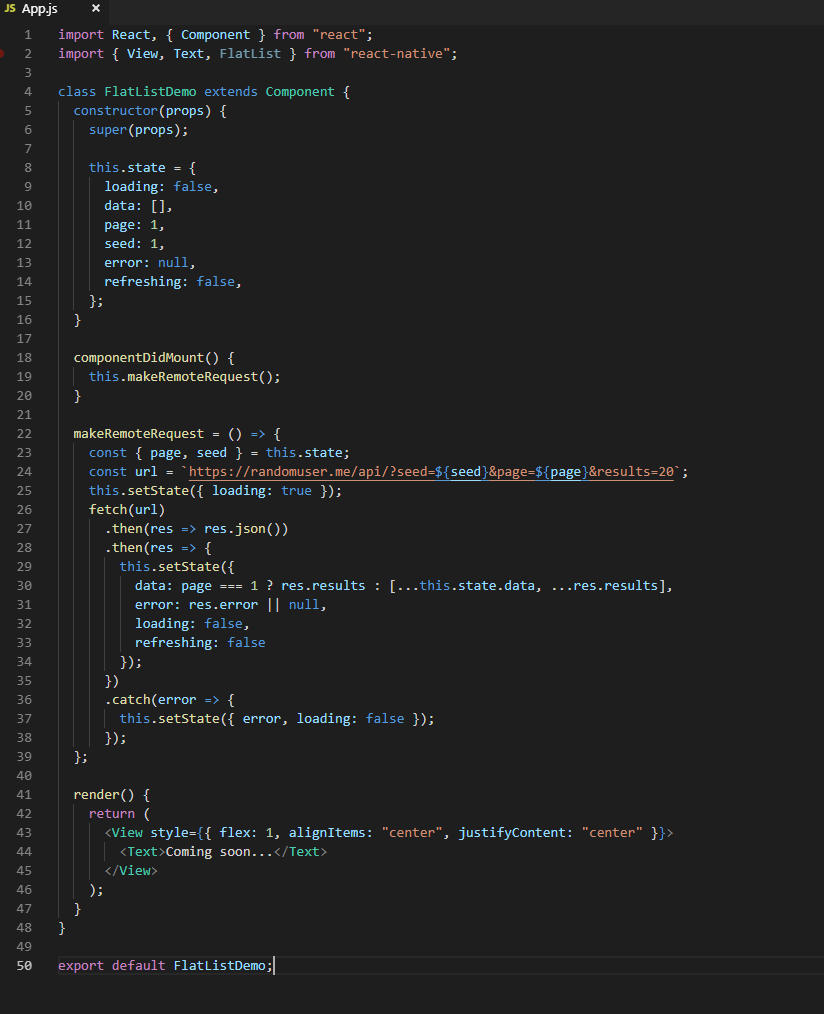
* **TUẦN 3.2:**
* **ListViews:** ListView - Một thành phần cốt lõi được thiết kế để hiển thị hiệu quả các danh sách cuộn theo chiều dọc của dữ liệu thay đổi. API tối thiểu là tạo một [ListView.DataSource](https://facebook.github.io/react-native/docs/listviewdatasource), điền vào nó một mảng các đốm dữ liệu đơn giản và khởi tạo một ListViewthành phần với nguồn dữ liệu đó và một renderRowcuộc gọi lại lấy một đốm từ mảng dữ liệu và trả về một thành phần có thể kết xuất được.



* **TUẦN 4.1: FlatList**

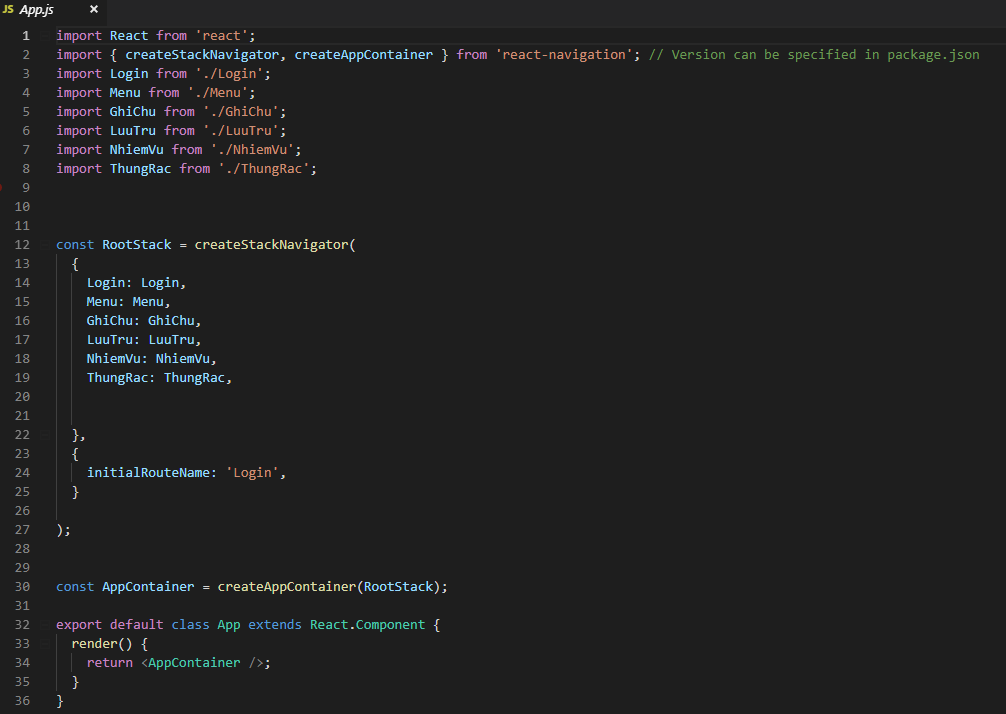
Giao diện biểu diễn để hiển thị danh sách đơn giản, phẳng, hỗ trợ các tính năng tiện dụng nhất:

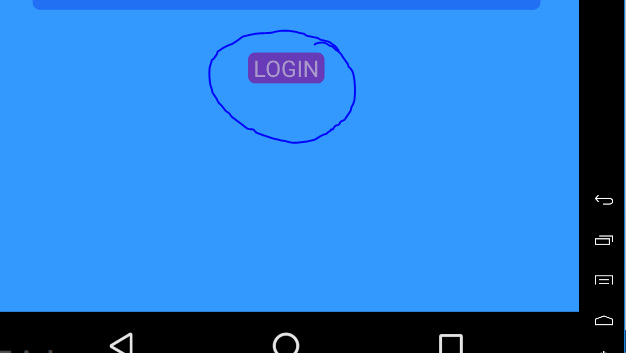
* Hoàn toàn đa nền tảng.
* Chế độ ngang tùy chọn.
* Cấu hình gọi lại khả năng xem.
* Hỗ trợ tiêu đề.
* Hỗ trợ chân trang.
* Hỗ trợ tách.
* Kéo để làm mới.
* Cuộn tải.
* Hỗ trợ ScrollTo Index.

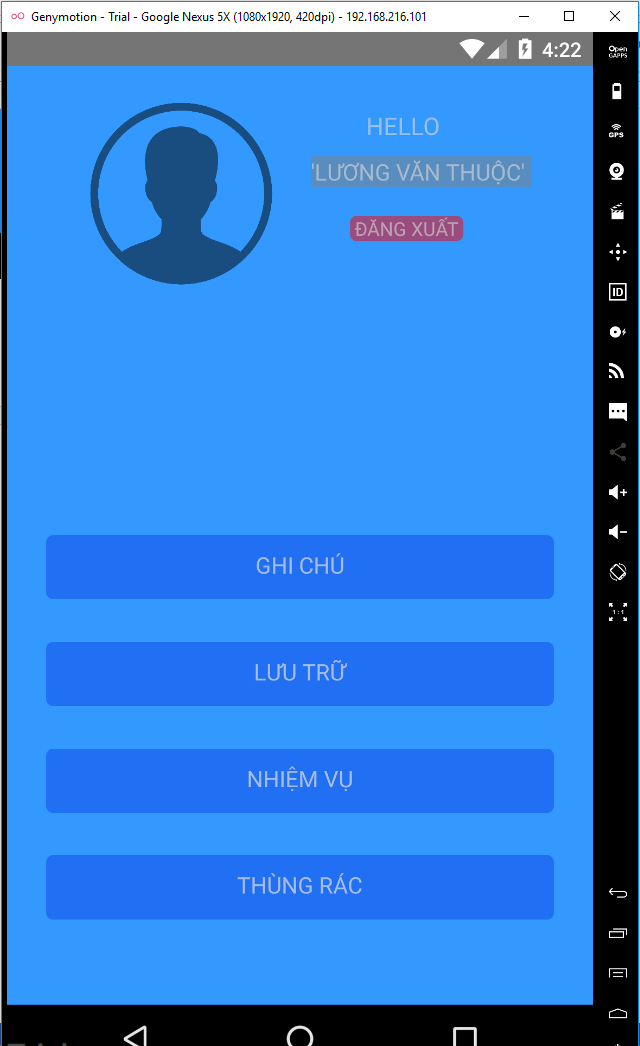


* **TUẦN 4.2: Navigation**

StackNavigator cung cấp cho bạn một cách chuyển qua lại giữa các màn hình với nhau và có thể quản lý nó một cách dễ dàng. Ngoài ra nó còn có thể cung cấp các gestures và animation phù hợp với Android và iOS khi chuyển đổi giữa các màn hình.







* **TUẦN 5.1:**
* **TUẦN 5.2:**
* **TUẦN 6.1:**
* **TUẦN 6.2:**