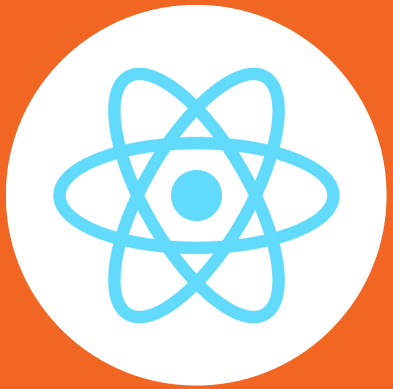
FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 6: REACT NAVIGAION TRONG REACT NATIVE

PHẦN 1: GIỚI THIỆU VỀ REACT NAVIGATION VÀ STACK NAVIGATOR

Mục tiêu

- [ ] Cài đặt thư viện @react-navigation/native
- [ ] Biết cách chuyển trang, điều hướng cho ứng dụng
- [ ] Biết cách xử lý, thêm parameters vào trong routes

Giới thiệu react navigation

- React Navigation là một thư viện điều hướng cho ứng dụng di động React Native. Nó giúp bạn quản lý các màn hình và các luồng điều hướng trong ứng dụng của bạn. Thư viện này cung cấp các thành phần và API để tạo ra các cấu trúc điều hướng phức tạp như Stack (ngăn xếp), Tab (tab), Drawer (hộp kéo), và nhiều hơn nữa.
- Với React Navigation, bạn có thể xây dựng các ứng dụng di động đa màn hình và điều hướng giữa chúng dễ dàng. Nó cung cấp các tính năng như quản lý lịch sử điều hướng, chuyển đổi màn hình, truyền tham số giữa các màn hình, tạo các màn hình modal và nhiều tính năng khác.

Giới thiệu react navigation

- React Navigation đã trở thành một trong những thư viện điều hướng phổ biến nhất trong cộng đồng React Native và được sử dụng rộng rãi trong việc xây dựng ứng dụng di động React Native.
- React Navigation cung cấp các loại điều hướng sau đây:
  - **Stack Navigator**: Cho phép bạn xây dựng một ngăn xếp các màn hình dựa trên ngăn xếp **Last-In-First-Out (LIFO)**. Khi bạn điều hướng đến một màn hình mới, nó được đặt lên đỉnh ngăn xếp. Bạn có thể quay lại các màn hình trước đó bằng cách đẩy chúng ra khỏi ngăn xếp. **Stack Navigator** là loại điều hướng phổ biến nhất trong React Navigation.

Giới thiệu react navigation

- **Tab Navigator:** Cho phép bạn tạo ra các tab điều hướng cho ứng dụng của mình. Mỗi tab đại diện cho một màn hình hoặc một ngăn xếp màn hình riêng. Người dùng có thể chuyển đổi giữa các tab để truy cập vào các màn hình khác nhau.
- **Drawer Navigator**: Cho phép bạn tạo ra một hộp kéo điều hướng mà người dùng có thể kéo ra từ cạnh màn hình để truy cập vào các màn hình khác. Hộp kéo thường chứa các liên kết hoặc biểu tượng để người dùng có thể điều hướng trong ứng dụng.
- **Switch Navigator**: Cho phép bạn tạo ra một ngăn xếp điều hướng nơi chỉ có một màn hình hiển thị tại một thời điểm. Khi bạn điều hướng đến một màn hình mới, màn hình hiện tại sẽ bị xóa khỏi ngăn xếp. **Switch Navigator** thích hợp cho việc xử lý **đăng nhập** và **đăng ký**, nơi bạn muốn xóa màn hình đăng nhập sau khi người dùng đã đăng nhập thành công.

Giới thiệu react navigation

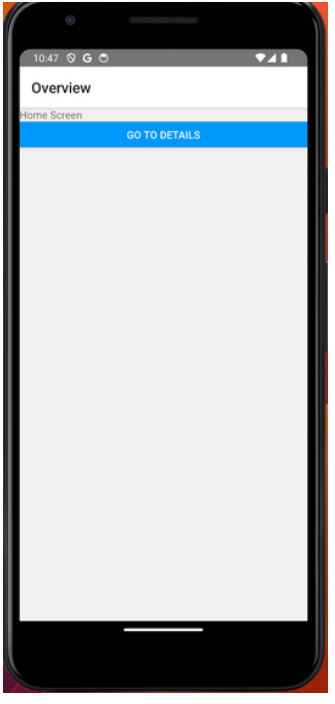
- Ngoài ra, **React Navigation** cũng cung cấp các loại điều hướng khác như **Bottom Tab Navigator**, **Material Top Tab Navigator** và nhiều loại điều hướng mở rộng khác được xây dựng trên cơ sở các loại điều hướng cơ bản đã nêu trên.

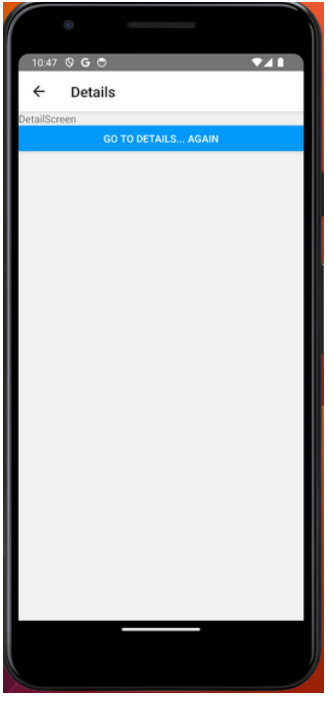
Stack navigator

- Để bắt đầu bài học này, chúng ta sẽ bắt đầu với **Stack navigator.** Chúng ta sử dụng **Stack** khi muốn chuyển đến một trang khác, bạn cũng có thể chuyển kèm theo dữ liệu gọi là **params.** Ngoài ra bạn cũng có thể thêm các hiệu ứng animation chuyển trang theo ý của bạn muốn

Stack navigator

- Một stack navigator sẽ có giao diện như sau:





Stack navigator

- **React navigation** là một thư viện, thế nên để sử dụng chúng bạn phải cài đặt thư viện vào project của mình.
- Hướng dẫn
  - Bước 1: Cài đặt **@react-navigation/stack**

```
npm install @react-navigation/stack
```

  - Bước 2: Cài đặt **react-native-gesture-handler**

```
npm install react-native-gesture-handler
```

Stack navigator

- Bước 3: Cài đặt **@react-navigation/native**

```
npm install @react-navigation/native
```

- Bước 4: Cài đặt **react-native-screens react-native-safe-area-context**

```
npm install react-native-screens react-native-safe-area-context
```

- Bước 5: Cài đặt **@react-navigation/native-stack**

```
npm install @react-navigation/native-stack
```

- Bước 6: Import **react-native-gesture-handler**

Để hoàn tất cài đặt **react-native-gesture-handler**, hãy thêm phần sau ở trên cùng (đảm bảo nó ở trên cùng và không có gì khác trước nó) của tệp nhập của bạn, chẳng hạn như **index.js** hoặc **App.js**

```
import 'react-native-gesture-handler';
import {AppRegistry} from 'react-native';
import App from './App';
import {name as appName} from './app.json';

AppRegistry.registerComponent(appName, () => App);
```

Stack navigator

Tạo một stack navigator

Trình điều hướng stack native của **React Navigation** cung cấp một cách để ứng dụng của bạn chuyển đổi giữa các màn hình và quản lý lịch sử điều hướng. Nếu ứng dụng của bạn chỉ sử dụng một trình điều hướng ngăn xếp thì về mặt khái niệm, nó tương tự như cách trình duyệt web xử lý trạng thái điều hướng - ứng dụng của bạn đẩy và bật các mục từ stack điều hướng khi người dùng tương tác với nó và điều này dẫn đến việc người dùng nhìn thấy các màn hình khác nhau. Một sự khác biệt chính giữa cách điều này hoạt động trong trình duyệt web và trong React Navigation là trình điều hướng ngăn xếp gốc của React Navigation cung cấp các cử chỉ và hình ảnh động mà bạn mong đợi trên **Android** và **iOS** khi điều hướng giữa các route trong stack.

Stack navigator

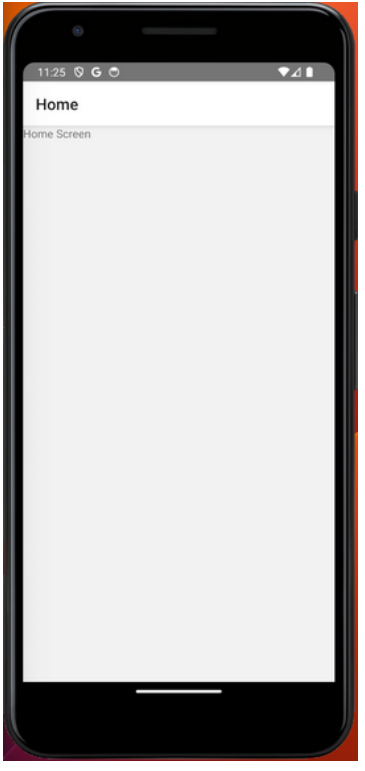
- Để tạo một stack **Home**, code mẫu như bên dưới:

```
const HomeScreen = () => {
  return (
    <View>
      <Text>Home Screen</Text>
    </View>
  );
};

const Stack = createNativeStackNavigator();

function App(): JSX.Element {
  return (
    <SafeAreaView style={{flex: 1}}>
      <NavigationContainer>
        <Stack.Navigator>
          <Stack.Screen name="Home" component={HomeScreen} />
        </Stack.Navigator>
      </NavigationContainer>
    </SafeAreaView>
  );
}
```

Stack navigator



- Nếu bạn chạy ứng dụng, bạn sẽ thấy một màn hình với thanh điều hướng trống và vùng nội dung màu xám chứa thành phần **HomeScreen** của bạn (hiển thị ở trên). Các kiểu bạn thấy cho thanh điều hướng và vùng nội dung là cấu hình mặc định cho **stack navigator**, chúng ta sẽ tìm hiểu cách config chúng sau.

Stack navigator

Cấu hình navigator

Tất cả các cấu hình **route** được chỉ định làm **props** cho navigator của chúng ta. Chúng ta chưa chuyển bất kỳ **props** nào cho **navigator** của chúng ta, vì vậy nó chỉ sử dụng cấu hình mặc định.

Hãy thêm màn hình thứ hai vào stack navigator của chúng ta và cấu hình **Home Screen** để hiển thị trước:

Stack navigator

```
function DetailsScreen() {
  return (
    <View>
      <Text>Details Screen</Text>
    </View>
  );
}

const Stack = createNativeStackNavigator();

function App(): JSX.Element {
  return (
    <SafeAreaView style={{flex: 1}}>
      <NavigationContainer>
        <Stack.Navigator initialRouteName="Home">
          <Stack.Screen name="Home" component={HomeScreen} />
          <Stack.Screen name="Details" component={DetailsScreen} />
        </Stack.Navigator>
      </NavigationContainer>
    </SafeAreaView>
  );
}
```

Stack navigator

- Bây giờ stack của chúng tôi có hai route, một route **Home** và route **Details**. Một route có thể được chỉ định bằng cách sử dụng thành phần **Screen**. Thành phần Screen chấp nhận một **name** prop tương ứng với tên của **route** mà chúng ta sẽ sử dụng để điều hướng và một component prop tương ứng với component mà nó sẽ render.
- **initialRouteName** sẽ xác định route được ưu tiên render trước, bạn hãy thử thay đổi thành **Details** và reload lại ứng dụng.

Stack navigator

Thêm options

Mỗi màn hình trong **navigator** có thể chỉ định một số **option** cho navigator, chẳng hạn như **title** hiển thị trong tiêu đề. Các tùy chọn này có thể được chuyển trong phần **options** cho từng thành phần màn hình:

```
<Stack.Screen
  name="Home"
  component={HomeScreen}
  options={{ title: 'Overview' }}
/>
```

Stack navigator

- Di chuyển giữa các màn hình

  Trong phần trước, chúng ta đã định nghĩa một **stack navigator** với hai **route** (Home và Details), nhưng chúng ta chưa học cách cho phép người dùng điều hướng từ **Home** đến **Detail** (mặc dù chúng ta đã học cách thay đổi route ban đầu trong mã của chúng ta.

Stack navigator

```
const HomeScreen: FC<NativeStackScreenProps<RootStackParamList, 'Home'>> = ({
  navigation,
}) => {
  return (
    <View>
      <Text>Home Screen</Text>
      <Button
        title="Go to Details"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
};
```

Stack navigator

- Giải thích:

navigation - **navigation** được chuyển vào mọi màn hình (được định nghĩa) trong trình stack navigator.

navigate('Details') - chúng ta gọi hàm **navigate** (trên **navigation** prop) với tên của route mà chúng ta muốn di chuyển người dùng đến.

Nếu chúng ta gọi **navigation.navigate** với tên **route** mà chúng ta chưa xác định trong navigator, nó sẽ in lỗi trong các bản build phát triển và sẽ không có gì xảy ra trong các bản dựng sản xuất. Nói cách khác, chúng ta chỉ có thể điều hướng đến các **route** đã được xác định trên bộ điều hướng của chúng ta - chúng ta không thể điều hướng đến một thành phần tùy ý.

Stack navigator

- Navigate đến route nhiều lần

```
const DetailsScreen: FC<
  NativeStackScreenProps<RootStackParamList, 'Details'>
> = ({navigation}) => {
  return (
    <View>
      <Text>DetailScreen</Text>
      <Button
        title="Go to Details... again"
        onPress={() => navigation.navigate('Details')}
      />
    </View>
  );
};
```

Nếu bạn chạy mã này, bạn sẽ nhận thấy rằng khi bạn nhấn vào '**Go to Details... again**' nó không làm gì cả! Điều này là do màn hình hiện tại đã ở trên route **Details**. Chức năng **navigate** đại khái có nghĩa là 'đi đến màn hình này' và nếu bạn đã ở trên màn hình đó thì có nghĩa là nó sẽ không làm gì cả.

Nếu các bạn thực sự muốn thêm một màn hình **Details** khác. Điều này khá phổ biến trong trường hợp bạn truyền một số dữ liệu duy nhất cho mỗi **route** (nhiều hơn về điều đó sau khi chúng ta nói về **params**!). Để làm điều này, chúng ta có thể thay đổi **navigate** để **push**. Điều này cho phép chúng tôi thể hiện ý định thêm một route khác bất kể lịch sử **navigate** hiện có.

Stack navigator

```
<Button
  title="Go to Details"
  onPress={() => navigation.push('Details')}
/>
```

Mỗi lần bạn **push**, chúng tôi thêm một **route** mới vào navigator stack. Khi bạn gọi, **navigate** trước tiên nó sẽ cố gắng tìm một route hiện có với tên đó và chỉ đẩy một route mới nếu chưa có route nào trên stack.

Going back

Header được cung cấp bởi trình stack navigator sẽ tự động bao gồm nút quay lại khi có thể quay lại từ màn hình đang hoạt động (nếu chỉ có một màn hình trong navigation stack, bạn không thể quay lại và do đó không có nút quay lại).

Đôi khi bạn sẽ muốn có thể kích hoạt hành vi này theo chương trình và bạn có thể sử dụng **navigation.goBack()**;.

```
<Button
  title="Go to Details... again"
  onPress={() => navigation.push('Details')}
/>
<Button title="Go to Home" onPress={() => navigation.navigate('Home')} />
<Button title="Go back" onPress={() => navigation.goBack()} />
```

Truyền params vào routes

Bây giờ chúng ta đã biết cách tạo trình điều hướng ngăn xếp với một số tuyến đường và điều hướng giữa các tuyến đường đó, hãy xem cách chúng ta có thể truyền dữ liệu đến các tuyến khi chúng ta điều hướng đến chúng.

Có hai phần này:

1. Chuyển **params** đến một tuyến đường bằng cách đặt chúng vào một object dưới dạng tham số thứ hai cho hàm **navigation.navigate** : **navigation.navigate('RouteName', { /* params go here */ })**

2. Đọc các tham số trong thành phần màn hình của bạn: **route.params**.

Stack navigator

- Truyền params **id** và **otherParam** vào routes:

```
<Button
  title="Go to Details"
  onPress={() =>
    navigation.navigate('Details', {
      id: 1,
      otherParam: 'Param bạn muốn truyền',
    })
  }
/>
```

Stack navigator

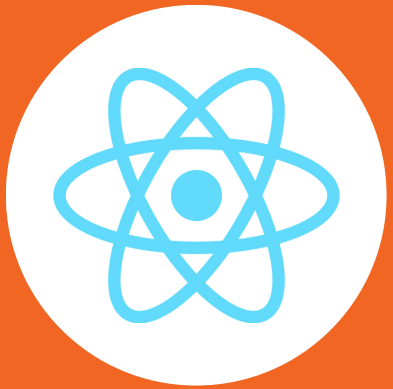
- Nhận params **id** và **otherParam** từ **route.params**

```
const DetailsScreen: FC<
  NativeStackScreenProps<RootStackParamList, 'Details'>
> = ({route, navigation}) => {
  const {id, otherParam} = route.params;

  return (
    <View>
      <Text>DetailScreen</Text>
      <Text>itemId: {id}</Text>
      <Text>otherParam: {otherParam}</Text>
      <Button
        title="Go to Details... again"
        onPress={() => navigation.goBack()}
      />
    </View>
  );
};
```

FPT Education

FPT POLYTECHNIC



LẬP TRÌNH ĐA NỀN TẢNG VỚI REACT NATIVE

BÀI 6: REACT NAVIGAION TRONG REACT NATIVE

PHẦN 2: GIỚI THIỆU VỀ DRAWER NAVIGATOR

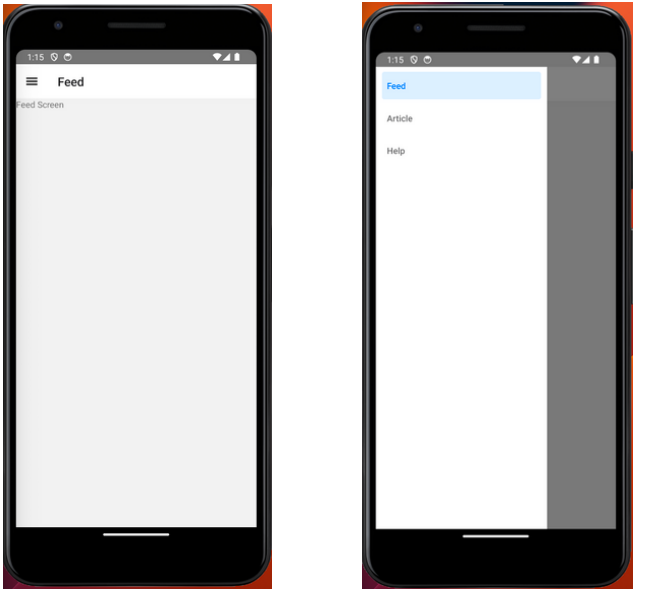
https://caodang.fpt.edu.vn/

Mục tiêu

- [ ] Tìm hiểu về Drawer navigator
- [ ] Xây dựng menu dọc và tuỳ chỉnh trong ứng dụng react native

Drawer Navigator

Drawer Navigator hiển thị một ngăn điều hướng ở bên cạnh màn hình có thể được mở và đóng thông qua cử chỉ.



Drawer navigator

Hướng dẫn

- Bước 1: Cài đặt **@react-navigation/drawer**

```
npm install @react-navigation/drawer
```

- Bước 2: Cài đặt **react-native-gesture-handler react-native-reanimated**

```
npm install react-native-gesture-handler react-native-reanimated
```

- Bước 3: Vào file **babel.config** thêm **plugins: ['react-native-reanimated/plugin']**

```
babel.config.js > ...
1   module.exports = {
2     presets: ['module:metro-react-native-babel-preset'],
3     plugins: ['react-native-reanimated/plugin'],
4   };
5
```

- Bước 4: Tắt **metro** và chạy lệnh **npm run start --reset-cache,** để xoá **cache,** sau đó chạy lại ứng dụng.

Drawer navigator

- Sau khi cài đặt thư viện xong, chúng ta sẽ bắt đầu tạo một drawer navigator cơ bản

Để sử dụng bộ điều hướng ngăn kéo này, hãy nhập nó từ **@react-navigation/drawer**:

```
const Drawer = createDrawerNavigator();

const DrawerNavigator = () => {
  return (
    <Drawer.Navigator>
      <Drawer.Screen name="Home" component={FeedScreen} />
      <Drawer.Screen name="Article" component={ArticleScreen} />
    </Drawer.Navigator>
  );
};
```

Drawer navigator

Props

- **initialRouteName**:

  Tên của route để hiển thị đầu tiên của navigator.

- **screenOptions:**

  Tùy chọn mặc định để sử dụng cho các màn hình trong navigator.

- **backBehavior:**

  Điều này kiểm soát những gì xảy ra khi **goBack** được gọi trong navigator. Điều này bao gồm nhấn nút quay lại hoặc cử chỉ quay lại của thiết bị trên **Android**.

Drawer navigator

Nó hỗ trợ các giá trị sau:

- **firstRoute** - quay lại màn hình đầu tiên được xác định trong trình điều hướng (mặc định)
- **initialRoute** - trở về màn hình ban đầu được truyền trong prop **initialRouteName**, nếu không được thông qua, mặc định là màn hình đầu tiên.
- **order** - Quay lại màn hình được xác định trước màn hình hiện tại.
- **history** - quay lại màn hình truy cập lần cuối trong navigator; Nếu cùng một màn hình được truy cập nhiều lần, các mục cũ hơn sẽ bị xóa khỏi lịch sử.

- **none -** không xử lý nút quay lại

❖ defaultStatus:

Trạng thái mặc định của drawer - cho dù drawer nên **open** hay **close** theo mặc định.

Khi được đặt để **open**, **drawer** sẽ được mở từ kết xuất ban đầu. Nó có thể được đóng bình thường bằng cử chỉ hoặc lập trình. Tuy nhiên, khi quay trở lại, drawer sẽ mở lại nếu nó đã được **closed**. Điều này về cơ bản trái ngược với hành vi mặc định của drawer nơi nó bắt đầu closed và nút quay lại đóng drawer mở.

- **none -** không xử lý nút quay lại

❖ drawerContent:

Hàm trả về phần tử React để render dưới dạng nội dung của drawer, ví dụ: các navigation items

Thành phần nội dung nhận được các prop sau theo mặc định:

- **state** - Navigation state của navigator.
- **navigation** - Đối tượng navigation cho navigator.
- **descriptors** - Một đối tượng mô tả chứa các tùy chọn cho màn hình drawer. Các tùy chọn có thể được truy cập tại **descriptors[route.key].options**.

Drawer navigator

- Cung cấp tuỳ chỉnh **drawerContent**

Thành phần mặc định cho drawer có thể cuộn và chỉ chứa các liên kết cho các routes trong **RouteConfig**. Bạn có thể dễ dàng ghi đè thành phần mặc định để thêm đầu trang, chân trang hoặc nội dung khác vào drawer. Thành phần nội dung mặc định được xuất dưới dạng **DrawerContent**. Nó hiển thị một thành phần **DrawerItemList** bên trong **ScrollView**.

Theo mặc định, drawer có thể cuộn và hỗ trợ các thiết bị có rãnh. Nếu bạn tùy chỉnh nội dung, bạn có thể sử dụng **DrawerContentScrollView** để xử lý việc này tự động:

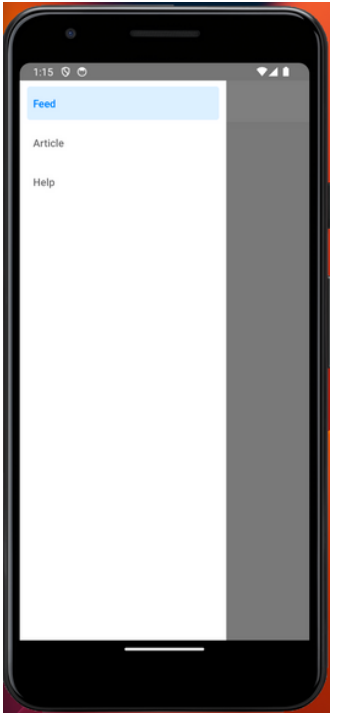
Drawer navigator

```
function CustomDrawerContent(props: DrawerContentComponentProps) {
  return (
    <DrawerContentScrollView {...props}>
      <DrawerItemList {...props} />
      <DrawerItem
        label="Help"
        onPress={() => Linking.openURL('https://mywebsite.com/help')}
      />
    </DrawerContentScrollView>
  );
}

const DrawerNavigator = () => {
  return (
    <Drawer.Navigator
      drawerContent={props => <CustomDrawerContent {...props} />}>
      <Drawer.Screen name="Home" component={FeedScreen} />
      <Drawer.Screen name="Article" component={ArticleScreen} />
    </Drawer.Navigator>
  );
};
```

Drawer navigator

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



Các bạn có thể tuỳ chỉnh thêm item trong drawer menu bằng cách sử dụng **DrawerItem** trong menu

Ngoài ra, **DrawerItem** còn hỗ trợ rất nhiều prop khác để các bạn có thể tuỳ chỉnh như: icon, focused, activeTintColor, labelStyle, style...

Options

Các tùy chọn sau đây có thể được sử dụng cấu hình màn hình trong bộ điều hướng. Chúng có thể được chỉ định trong **screenOptions** prop của **Drawer.navigator** hoặc tùy chọn prop của **Drawer.Screen**.

- **title:** Một tiêu đề chung có thể được sử dụng làm dự phòng cho headerTitle và drawerLabel.
- **lazy:** Một tiêu đề chung có thể được sử dụng làm dự phòng cho headerTitle và drawerLabel.
- **drawerLabel:** Chuỗi hoặc một hàm đã cho **{ focused: boolean, color: string }** trả về một **React.Node**, để hiển thị trong thanh bên drawer. Khi undefined, title được sử dụng.

Drawer navigator

- **drawerIcon:** Hàm, đã cho **{ focused: boolean, color: string, size: number }** trả về một **React.Node** để hiển thị trong thanh bên drawer.
- **drawerActiveTintColor:** Màu cho icon và label trong mục đang được chọn trong drawer.
- **drawerActiveBackgroundColor:** Màu nền cho mục đang được chọn trong drawer.
- **drawerItemStyle:** Đối tượng kiểu cho một mục, có thể chứa icon và label.
- **drawerStyle:** Đối tượng kiểu cho cả drawer menu

Events

navigator có thể phát ra các sự kiện trên một số hành động nhất định. Các sự kiện được hỗ trợ là:

- **drawerItemPress:** Sự kiện này được kích hoạt khi người dùng nhấn nút trên item trong ngăn drawer. Theo mặc định, một mục ngăn kéo nhấn thực hiện một số việc:
  - Nếu màn hình không được chọn, drawer item nhấn sẽ chọn màn hình đó.
  - Nếu màn hình đã được chọn, thì nó sẽ đóng drawer.

  Để ngăn chặn hành vi mặc định, bạn có thể gọi **event.preventDefault**

Helpers

drawer navigator thêm các phương pháp sau vào navigation prop:

- **openDrawer:** Mở drawer

```
navigation.openDrawer();
```

- **closeDrawer:** Đóng drawer

```
navigation.closeDrawer();
```

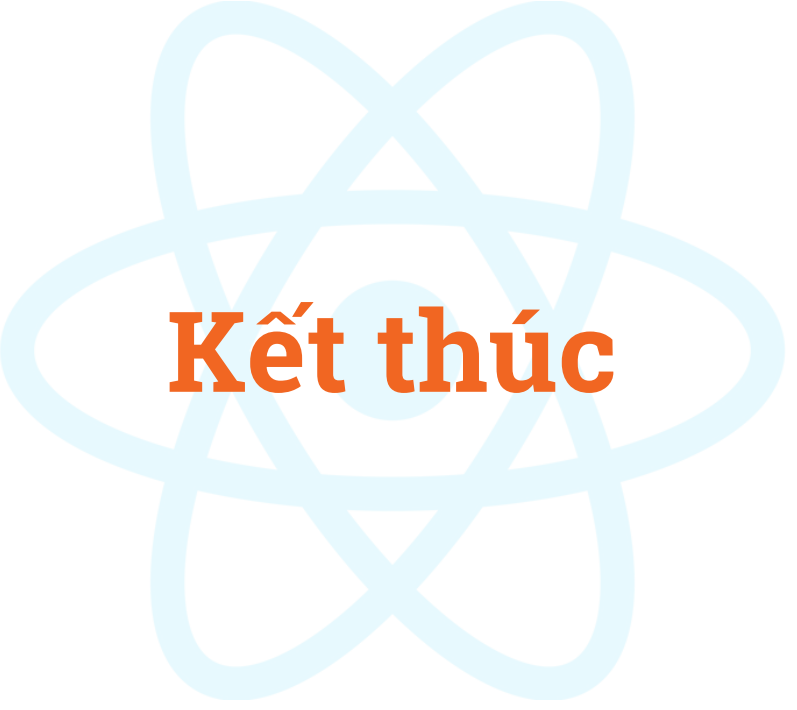
Drawer navigator

- **toggleDrawer:** Mở drawer nếu đóng, đóng drawer nếu mở.

```
navigation.toggleDrawer();
```

Tổng kết

- [ ] Cài đặt thư viện @react-navigation/native
- [ ] Biết cách chuyển trang, điều hướng cho ứng dụng
- [ ] Biết cách xử lý, thêm parameters vào trong routes
- [ ] Tìm hiểu về Drawer navigator
- [ ] Xây dựng menu dọc và tuỳ chỉnh trong ứng dụng react native



Kết thúc