**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA KHOA HỌC VÀ KĨ THUẬT THÔNG TIN**



**BÁO CÁO SEMINAR**

**MÔN CÔNG NGHỆ LẬP TRÌNH ĐA NỀN TẢNG CHO ỨNG DỤNG DI DỘNG**

**Chuyên đề:**

**TÌM HIỂU VỀ ANIMATION - ICON - LANGUAGE & FONTS**

**GVHD: ThS. Võ Ngọc Tân**

**Nhóm sinh viên thực hiện**:

1. **Trần Lê Thanh Trúc MSSV: 22540025**
2. **Nguyễn Thị Thu Hiền MSSV: 22540027**

**🙢 Tp. Hồ Chí Minh, 10/2024 🙠**

# NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

*……., ngày……...tháng……năm 20…*

**Người nhận xét**

*(Ký tên và ghi rõ họ tên****)***

# BẢNG PHÂN CÔNG, ĐÁNH GIÁ THÀNH VIÊN

|  |  |  |  |
| --- | --- | --- | --- |
| **Họ và tên** | **MSSV** | **Phân công** | **Đánh giá** |
| Trần Lê Thanh Trúc | 22540025 |  | Hoàn thành tốt |
| Nguyễn Thị Thu Hiền | 22540027 |  | Hoàn thành tốt |

**LỜI MỞ ĐẦU**

React Native hiện đang trở thành một trong những framework phổ biến cho việc phát triển ứng dụng di động. Nhờ vào khả năng tạo ra các ứng dụng hiệu suất cao, giao diện đẹp và tính tương tác mượt mà, nó thu hút được nhiều người dùng. Tuy nhiên, với những người mới bắt đầu, việc làm quen với React Native có thể khá khó khăn, đặc biệt khi phải tìm hiểu về các tính năng cơ bản như biểu tượng (Icons), hoạt ảnh (Animation) và hỗ trợ đa ngôn ngữ (Localization). Những tính năng này rất quan trọng để tạo ra những ứng dụng không chỉ hoạt động tốt mà còn mang đến trải nghiệm người dùng thú vị và dễ sử dụng.

Bài tiểu luận này sẽ hướng dẫn chi tiết cách triển khai các thành phần quan trọng trong React Native, từ việc hiển thị icon, quản lý font chữ và ngôn ngữ đến tạo các hiệu ứng hoạt ảnh đơn giản. Với cách trình bày từng bước và kèm theo ví dụ cụ thể, bài viết hy vọng sẽ là tài liệu hữu ích cho những ai mới bắt đầu với React Native. Người đọc sẽ có cái nhìn rõ hơn về sức mạnh và tính linh hoạt của framework này, đồng thời tự tin thực hành để xây dựng ứng dụng của riêng mình.

Hy vọng rằng những kiến thức trong bài viết sẽ góp phần tạo nền tảng vững chắc và mở ra nhiều cơ hội học hỏi, phát triển cho bạn đọc trên con đường làm chủ React Native.

Để đạt được mục tiêu này, nhóm sinh sẽ trình bày chi tiết các nội dung sau:

* Chương 1: Tìm hiểu về Animation

Các hiệu ứng (Animations) là phần không thể thiếu để tăng cường trải nghiệm người dùng.

* Chương 2: Tìm hiểu về Icon

Trong chương này, nhóm tìm hiểu biểu tượng (Icon). Đây là yếu tố giao diện quan trọng, giúp người dùng nhận diện các tính năng một cách nhanh chóng.

* Chương 3: Tìm hiểu về Language & Fonts

React Native cung cấp các công cụ để dễ dàng thay đổi phông chữ và thiết lập đa ngôn ngữ.

* Chương 4: Kết luận và hướng phát triển

Nhóm sẽ tổng kết các kết quả đạt được sau khi tìm hiểu các nội dung đã đề xuất, rút ra những nội dung đã đạt được và chưa đạt được. Từ đó, nhóm xác định phương hướng phát triển trong tương lai để nội dung ngày càng hoàn thiện hơn.

# MỤC LỤC

[NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN 2](#_Toc181119881)

[BẢNG PHÂN CÔNG, ĐÁNH GIÁ THÀNH VIÊN 3](#_Toc181119882)

[LỜI MỞ ĐẦU 4](#_Toc181119883)

[Chương 1. Tìm hiểu về Animation 6](#_Toc181119886)

[1.1. Collapsible Toolbar-Animated 6](#_Toc181119887)

[1.2. Flip Image Horizontally 12](#_Toc181119888)

[1.3. Blinking Animation 16](#_Toc181119889)

[1.4. Rotate Image Animation 19](#_Toc181119890)

[1.5. React Native Lottie 23](#_Toc181119891)

[Chương 2. Tìm hiểu về Icon 27](#_Toc181119892)

[2.1. Vector Icons 27](#_Toc181119893)

[2.2. Social Icons 29](#_Toc181119894)

[2.3. ionIcons 32](#_Toc181119895)

[2.4. Font-Awesome Icons 35](#_Toc181119896)

[2.5. RN Currency Symbol 37](#_Toc181119897)

[Chương 3. Tìm hiểu về Language & Fonts 39](#_Toc181119898)

[3.1. RN Custom Fonts 39](#_Toc181119899)

[3.2. Multi Language 42](#_Toc181119900)

[Chương 4. Kết luận 47](#_Toc181119901)

[4.1. Kết luận 47](#_Toc181119902)

[4.2. Hướng phát triển 47](#_Toc181119903)

[TÀI LIỆU THAM KHẢO 48](#_Toc181119904)

# Tìm hiểu về Animation

Trong quá trình phát triển ứng dụng di động, các hiệu ứng hoạt ảnh (Animation) đóng vai trò quan trọng, giúp ứng dụng trở nên trực quan, sinh động, và cải thiện trải nghiệm người dùng. Hiểu rõ cách tạo và kiểm soát các hiệu ứng này trong React Native là bước quan trọng để tối ưu hóa giao diện và tính tương tác của ứng dụng. Các hiệu ứng Animation gồm có: Collapsible Toolbar-Animated, Flip Image Horizontally, Blinking Animation, Rotate Image Animation, và React Native Lottie đều là những công cụ mạnh mẽ giúp lập trình viên mang lại trải nghiệm độc đáo, sinh động cho người dùng.

## Collapsible Toolbar-Animated

### Mục đính

**Animated** trong màn hình **HOME** được sử dụng để tạo ra một header có hiệu ứng co giãn và thay đổi màu sắc khi người dùng cuộn danh sách sản phẩm trong trang chủ. Cụ thể khi cuộn:

* Trong phần header: dòng slogan và banners sẽ rút gọn lên đến khi chỉ còn hiển thị dòng slogan (ẩn banners).
* Link tài liệu: <https://aboutreact.com/react-native-collapsible-toolbar/>

### Hiển thị màn hình

|  |  |  |
| --- | --- | --- |
| **Trước khi cuộn:** | **Sau khi cuộn:** | **GIF:** |

### Source code: \src\pages\Home.js

*(Phần liên quan: màu trắng)*

import React, { useEffect, useState, useContext } from "react";

import Carousel from 'react-native-reanimated-carousel';

import axios from 'axios';

import { StyleSheet, Dimensions, View, SafeAreaView, Text, Image, ActivityIndicator, FlatList,

**Animated**} from 'react-native';

import { AuthContext } from '../components/Auth';

import { useCart } from '../context/CartContext';

import ProductCard from "../components/ProductCard";

const banner1 = require('../../assets/banner1.png');

const banner2 = require('../../assets/banner2.png');

export default function HomePage({ navigation }) {

    // const { username } = useContext(AuthContext);

    const [products, setProducts] = useState([]);

    const [hotDeals, setHotDeals] = useState([]);

    const [newArrivals, setNewArrivals] = useState([]);

    const [loading, setLoading] = useState(true);

    const [error, setError] = useState(null);

    const [bannerImages] = useState([banner1, banner2]);

    const width = Dimensions.get('screen').width;

    const { userId } = useCart();

    // Animated: https://aboutreact.com/react-native-collapsible-toolbar/

**const AnimatedHeaderValue = new Animated.Value(0);**

**const Header\_Maximum\_Height = 194;**  // Max Height of the Header

**const Header\_Minimum\_Height = 52;**   // Min Height of the Header

**const animateHeaderBackgroundColor =**

**AnimatedHeaderValue.interpolate({**

**inputRange: [0, Header\_Maximum\_Height],**

**outputRange: ['#FCFEFF', '#E1E1E1'],**

**extrapolate: 'clamp',**

**});**

**const animateHeaderHeight =**

**AnimatedHeaderValue.interpolate({**

**inputRange: [0, Header\_Maximum\_Height],**

**outputRange: [Header\_Maximum\_Height, Header\_Minimum\_Height],**

**extrapolate: 'clamp',**

**});**

    useEffect(() => {

        const fetchProducts = async () => {

            setLoading(true);

            try {

                const response = await axios.get('https://fakestoreapi.com/products');

                setProducts(response.data);

                setHotDeals(response.data.slice(0, 4));

                setNewArrivals(response.data.slice(4, 12));

            } catch (error) {

                setError('Failed to fetch products');

            } finally {

                setLoading(false);

            }

        };

        fetchProducts();

    }, []);

    const handleProductClick = (item) => {

        navigation.navigate('HomeDetails', { product: item });

    }

    const renderSection = (title, data) => (

        <>

            <Text style={styles.sectionTitle}>{title}</Text>

            <FlatList

                data={data}

                renderItem={({ item }) => (

                    <ProductCard

                        item={item}

                        handleProductClick={handleProductClick}

                        userId={userId}

                    />

                )}

                keyExtractor={item => item.id.toString()}

                numColumns={2}

                contentContainerStyle={styles.productList}

            />

        </>

    );

    return (

        <SafeAreaView style={styles.container}>

            {/\* show header (slogan + banners) \*/}

**<Animated.View style={{**

**backgroundColor: animateHeaderBackgroundColor,**

**height: animateHeaderHeight,**

**overflow: 'hidden',**

**}}>**

**<View style={styles.headerSection}>**

**<Text style={styles.slogan}>Shop for quality, Shop for style</Text>**

**<Carousel**

**loop**

**width={width}**

**height={150}**

**autoPlay={true}**

**data={bannerImages}**

**renderItem={({ item }) => (**

**<Image source={item} style={styles.bannerImage} />**

**)}**

**/>**

**</View>**

**</Animated.View>**

            {/\* show items \*/}

            {loading ? (

                <ActivityIndicator size="large" color="#0000ff" />

            ) : (

                <>

                    <FlatList

                        data={[{ key: 'Hot Deals' }, { key: 'New Arrivals' }]}

                        renderItem={({ item }) => (

                            <View style={styles.sectionContainer}>

                                {item.key === 'Hot Deals' ?

renderSection('Hot Deals', hotDeals) :

renderSection('New Arrivals', newArrivals)}

                            </View>

                        )}

                        keyExtractor={item => item.key}

**onScroll={Animated.event(**

**[{ nativeEvent:**

**{ contentOffset: { y: AnimatedHeaderValue } }**

**}],**

**{ useNativeDriver: false }**

**)}**

                    />

                </>

            )}

            {error && <Text style={styles.errorText}>{error}</Text>}

        </SafeAreaView>

    );

}

### Giải thích:

* **Animated.Value** và giá trị khởi tạo

|  |
| --- |
| const AnimatedHeaderValue = new Animated.Value(0); |

* **Animated.Value(0):** là giá trị khởi tạo cho Animated. Giá trị này thay đổi khi cuộn nội dung và được liên kết với hiệu ứng cuộn (onScroll).
* Giá trị này sẽ thay đổi dựa trên vị trí cuộn (scroll offset) của người dùng và được sử dụng để tạo ra các hiệu ứng khác nhau cho phần header.
* **Định nghĩa chiều cao và màu nền của header với interpolate**

|  |
| --- |
| const animateHeaderBackgroundColor =          AnimatedHeaderValue.interpolate({              inputRange: [0, Header\_Maximum\_Height],              outputRange: ['#FCFEFF', '#E1E1E1'],              extrapolate: 'clamp',          });      const animateHeaderHeight =          AnimatedHeaderValue.interpolate({              inputRange: [0, Header\_Maximum\_Height],              outputRange: [Header\_Maximum\_Height, Header\_Minimum\_Height],              extrapolate: 'clamp',          }); |

* **interpolate():** Hàm cho phép chuyển đổi giá trị của **Animated.Value** thành các giá trị khác dựa trên phạm vi cuộn (scroll range).
* **inputRange: [0, Header\_Maximum\_Height]**: Chỉ định rằng khi giá trị cuộn (scroll value) nằm trong phạm vi từ 0 đến Header\_Maximum\_Height (chiều cao tối đa của header), các giá trị khác sẽ thay đổi theo.
* **outputRange: ['#FCFEFF', '#E1E1E1']:** Màu nền của header sẽ thay đổi từ màu sáng (#FCFEFF) khi không cuộn đến màu xám nhạt (#E1E1E1) khi cuộn tới chiều cao tối thiểu.
* **extrapolate: 'clamp'**: Đảm bảo giá trị sẽ không vượt quá phạm vi đầu vào, giữ cho chiều cao và màu nền không bị thay đổi quá mức.
* **animateHeaderHeight:** Tương tự với việc thay đổi chiều cao của header khi cuộn từ Header\_Maximum\_Height (194) đến Header\_Minimum\_Height (52).
* **Áp dụng hiệu ứng Animated cho header**

|  |
| --- |
| **<Animated.View style={{**  **backgroundColor: animateHeaderBackgroundColor,**  **height: animateHeaderHeight,**  **overflow: 'hidden',**  **}}>**          <View style={styles.headerSection}>              <Text style={styles.slogan}>Shop for quality, Shop for style</Text>              <Carousel                  loop                  width={width}                  height={150}                  autoPlay={true}                  data={bannerImages}                  renderItem={({ item }) => (                      <Image source={item} style={styles.bannerImage} />                  )}              />          </View>  **</Animated.View>** |

* **Animated.View**: Thay vì sử dụng View thông thường, Animated.View bao phần header (slogan & banners) cho phép áp dụng các hiệu ứng động được định nghĩa trước đó (chiều cao và màu nền).
* **backgroundColor: animateHeaderBackgroundColor**: Màu nền của View sẽ thay đổi dựa trên vị trí cuộn.
* **height: animateHeaderHeight**: Chiều cao của header sẽ thay đổi khi cuộn, tạo ra hiệu ứng co lại hoặc mở rộng.
* **Liên kết onScroll với Animated**

|  |
| --- |
| <FlatList          data={[{ key: 'Hot Deals' }, { key: 'New Arrivals' }]}          renderItem={({ item }) => (              <View style={styles.sectionContainer}>                  {item.key === 'Hot Deals' ?  renderSection('Hot Deals', hotDeals) :  renderSection('New Arrivals', newArrivals)  }              </View>          )}          keyExtractor={item => item.key}  **onScroll={Animated.event(**  **[{ nativeEvent: { contentOffset: { y: AnimatedHeaderValue } } }],**  **{ useNativeDriver: false }**  **)}**      /> |

* **onScroll={Animated.event(...)}:** Đây là phần kết nối giá trị cuộn của FlatList với giá trị AnimatedHeaderValue. Khi cuộn FlatList, giá trị của AnimatedHeaderValue sẽ thay đổi dựa trên độ cuộn của trang.
* **nativeEvent.contentOffset.y**: Đo lường độ cuộn theo phương dọc của trang.
* **AnimatedHeaderValue**: Giá trị cuộn sẽ cập nhật trực tiếp vào biến này, từ đó các thuộc tính động (như chiều cao và màu nền) sẽ được thay đổi.

**Tóm lại:**

Khi người dùng cuộn qua **FlatList, AnimatedHeaderValue** sẽ thay đổi. Điều này dẫn đến sự thay đổi chiều cao và màu nền của headerSection, tạo ra hiệu ứng động mượt mà khi phần header co lại hoặc mở rộng. Điều này giúp giao diện trở nên linh hoạt và dễ chịu hơn khi người dùng tương tác với ứng dụng.

## Flip Image Horizontally

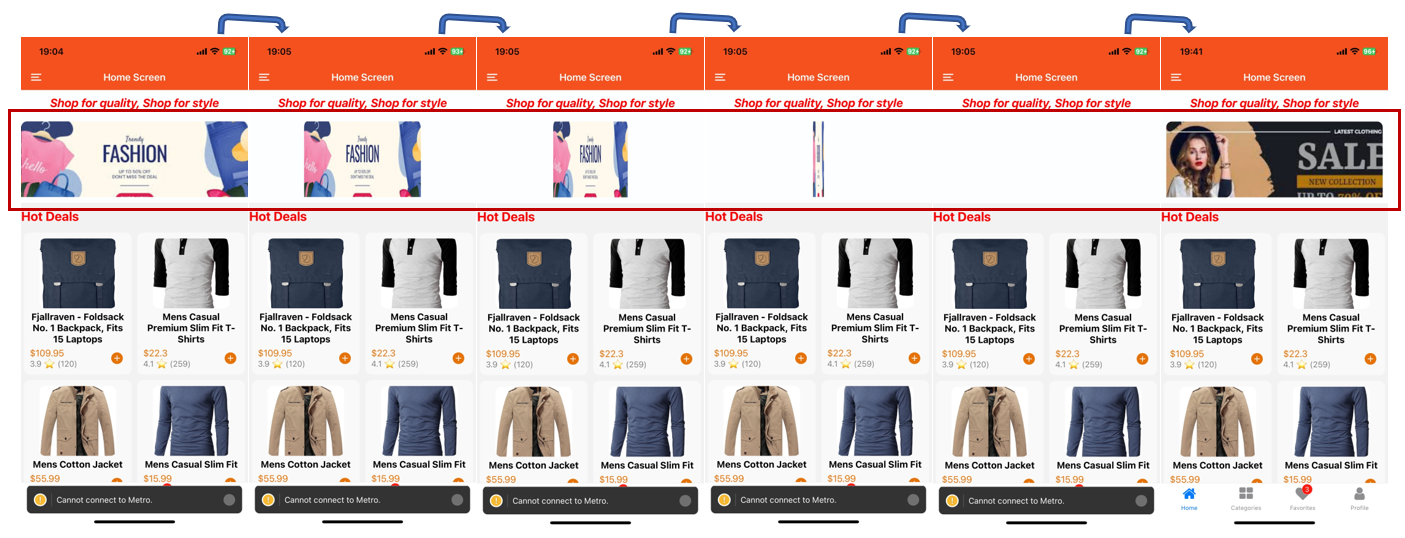
### Mục đính

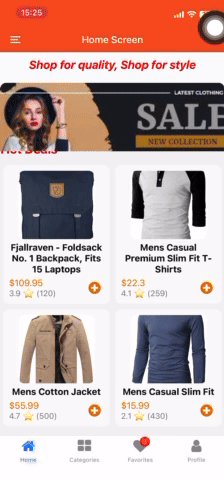
**Flip image** trong màn hình **HOME** được sử dụng để tạo ra hiệu ứng lật hình ảnh theo chiều ngang cho các banner trong phần header để tăng sự chú ý đến người dùng. Cụ thể:

* Xoay mỗi banner 90 độ rồi chuyển sang banner mới. Khi hết vòng sẽ trượt sang trái (vì kết hợp với Carousel) rồi lặp lại như ban đầu.
* Link tài liệu:

https://aboutreact.com/react-native-flip-image-horizontally-using-animation/

### Hiển thị màn hình





### Source code: \src\pages\Home.js

*(Phần liên quan: màu trắng)*

1. import React, { useEffect, useState, useContext } from "react";
2. import Carousel from 'react-native-reanimated-carousel';
3. import axios from 'axios';
4. import { StyleSheet, Dimensions, View, SafeAreaView, Text, Image, ActivityIndicator, FlatList, **Animated** } from 'react-native';
5. import { useCart } from '../context/CartContext';
6. import ProductCard from "../components/ProductCard";
7. const banner1 = require('../../assets/banner1.png');
8. const banner2 = require('../../assets/banner2.png');
9. export default function HomePage({ navigation }) {
10. const [products, setProducts] = useState([]);
11. const [hotDeals, setHotDeals] = useState([]);
12. const [newArrivals, setNewArrivals] = useState([]);
13. const [loading, setLoading] = useState(true);
14. const [error, setError] = useState(null);
15. const [bannerImages] = useState([banner1, banner2]);
16. const width = Dimensions.get('screen').width;
17. const { userId } = useCart();
18. // collapsible toolbar: declare variable and function here…
19. // flip image
20. **const flipValue = new Animated.Value(0);**
21. **const flipInterpolation = flipValue.interpolate({**
22. **inputRange: [0, 1],**
23. **outputRange: ['0deg', '90deg'],**  // Flip from 0 to n degrees on Y-axis
24. **});**
25. **const triggerFlip = () => {**
26. **Animated.timing(flipValue, {**
27. **toValue: flipValue.\_value === 0 ? 1 : 0,**  // Flip between 0 and 1
28. **duration: 300,**  // Animation duration
29. **useNativeDriver: true,**
30. **}).start();**
31. **};**
32. useEffect(() => {
33. // fetchProducts here…
34. }, []);
35. // handleProductClick here…
36. // renderSection here…
37. return (
38. <SafeAreaView style={styles.container}>
39. {/\* show header (slogan + banners) \*/}
40. <Animated.View style={{
41. backgroundColor: animateHeaderBackgroundColor,
42. height: animateHeaderHeight,
43. overflow: 'hidden',
44. }}>
45. <View style={styles.headerSection}>
46. <Text style={styles.slogan}>Shop for quality, Shop for style</Text>
47. <Carousel
48. loop
49. width={width}
50. height={150}
51. autoPlay={true}
52. data={bannerImages}
53. renderItem={({ item }) => (
54. **<Animated.View style={{**
55. **transform: [{ rotateY: flipInterpolation }],**  // Apply flip transformation
56. **}}>**
57. **<Image source={item} style={styles.bannerImage} />**
58. **</Animated.View>**
59. **)}**
60. **onSnapToItem={triggerFlip}**  // Trigger flip when a new banner is displayed
61. />
62. </View>
63. </Animated.View>
64. {/\* show items \*/}
65. {loading ? (
66. <ActivityIndicator size="large" color="#0000ff" />
67. ) : (
68. <>
69. <FlatList
70. {/\* use FlatList to show items \*/}
71. />
72. </>
73. )}
74. {error && <Text style={styles.errorText}>{error}</Text>}
75. </SafeAreaView>
76. );
77. };

### Giải thích

* **Khởi tạo giá trị lật**

|  |
| --- |
| const flipValue = new Animated.Value(0); |

* **Animated.Value(0):** flipValue được khởi tạo là một giá trị động ban đầu là 0. Giá trị này sẽ được sử dụng để điều khiển trạng thái của hiệu ứng lật. Khi flipValue thay đổi từ 0 đến 1, hình ảnh sẽ lật theo chiều Y.
* **Tạo hiệu ứng lật**

|  |
| --- |
| const flipInterpolation = flipValue.interpolate({          inputRange: [0, 1],          outputRange: ['0deg', '90deg'],  // Flip from 0 to n degrees on Y-axis      }); |

* **flipInterpolation:** là một phép nội suy (interpolation) của flipValue.
* **inputRange: [0, 1]** xác định các giá trị đầu vào mà flipValue có thể nhận.
* **outputRange: ['0deg', '90deg']** chỉ định các giá trị đầu ra tương ứng với các giá trị đầu vào trong inputRange. Khi flipValue chuyển từ 0 đến 1, hình ảnh sẽ quay từ 0 độ đến 90 độ quanh trục Y, tạo ra hiệu ứng lật.
* **Hàm kích hoạt lật**

|  |
| --- |
| const triggerFlip = () => {          Animated.timing(flipValue, {              toValue: flipValue.\_value === 0 ? 1 : 0,  // Flip between 0 and 1              duration: 300,  // Animation duration              useNativeDriver: true,          }).start();      }; |

* **triggerFlip** là một hàm được gọi để kích hoạt hiệu ứng lật.
* **toValue: flipValue.\_value === 0 ? 1 : 0** sẽ kiểm tra giá trị hiện tại của **flipValue.**
  + Nếu giá trị là 0, nó sẽ chuyển sang 1 (bắt đầu lật)
  + Ngược lại sẽ chuyển về 0 (quay lại vị trí ban đầu)
* **duration: 300** xác định thời gian thực hiện hiệu ứng lật là **300ms.**
* **useNativeDriver: true** cho phép sử dụng native driver để tối ưu hóa hiệu suất animation. Việc này giúp animation mượt mà hơn bằng cách sử dụng GPU.
* **Liên kết với Animated**

|  |
| --- |
| <Carousel          loop          width={width}          height={150}          autoPlay={true}          data={bannerImages}          renderItem={({ item }) => (  **<Animated.View style={{**  **transform: [{ rotateY: flipInterpolation }],  // Apply flip transformation**  **}}>**  **<Image source={item} style={styles.bannerImage} />**  **</Animated.View>**          )}          onSnapToItem={triggerFlip}  // Trigger flip when a new banner is displayed      /> |

* **Animated.View:** Thay vì sử dụng View thông thường, thì trong trường hợp này sử dụng **Animated.View**. Điều này cho phép áp dụng các hiệu ứng hoạt hình (animation) cho phần tử này. **Animated.View** có thể xử lý các thuộc tính animation mà View thông thường không thể.
* **transform:** thuộc tính CSS để áp dụng các hiệu ứng biến đổi cho phần tử. Trong trường hợp này, nó chứa một mảng với đối tượng:

{ **rotateY: flipInterpolation** }.

* **rotateY:** Là một thuộc tính biến đổi 3D trong CSS, cho phép quay phần tử quanh trục Y (trục dọc). Giá trị của rotateY được xác định bởi flipInterpolation.
* **flipInterpolation:** Là giá trị được tạo ra từ phép nội suy (interpolation) trước đó, và sẽ thay đổi từ 0deg đến 90deg (hoặc bất kỳ giá trị nào bạn đã chỉ định) khi giá trị của flipValue thay đổi từ 0 sang 1. Khi flipInterpolation thay đổi, nó sẽ làm cho hình ảnh trong Animated.View quay theo trục Y, tạo hiệu ứng lật.

**Tóm lại:**

Đoạn mã trên tạo ra một **carousel** với các hình ảnh được **lật** (Flip) khi người dùng cuộn qua chúng. Việc sử dụng Animated.View cho phép áp dụng hiệu ứng lật mượt mà, làm cho trải nghiệm người dùng trở nên hấp dẫn và thú vị hơn. Khi một hình ảnh mới xuất hiện, nó sẽ được lật theo chiều Y, tạo cảm giác như đang lật một trang sách và thu hút sự chú ý.

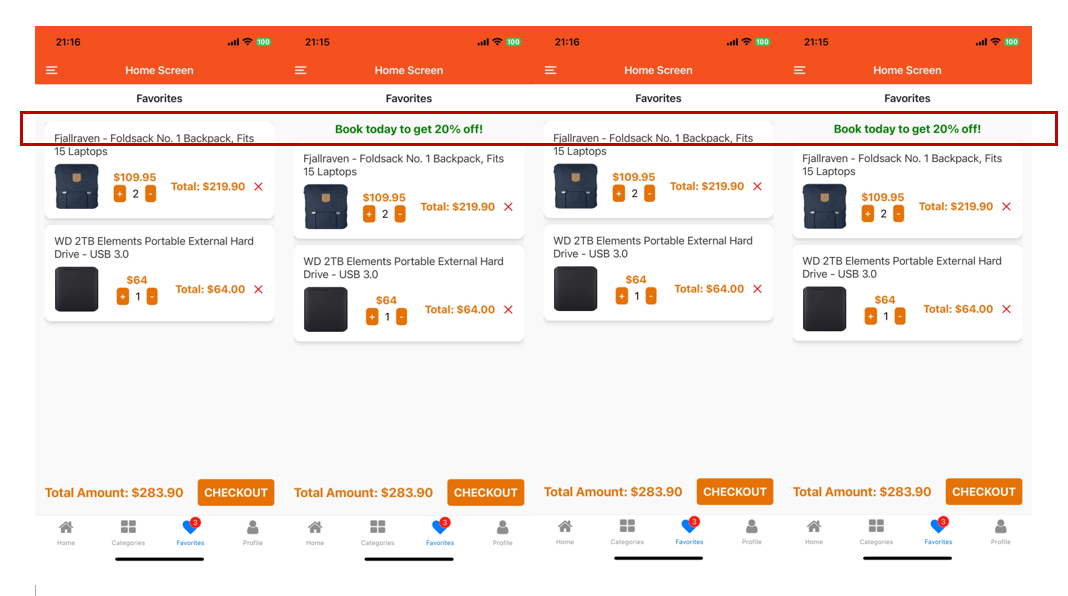
## Blinking Animation

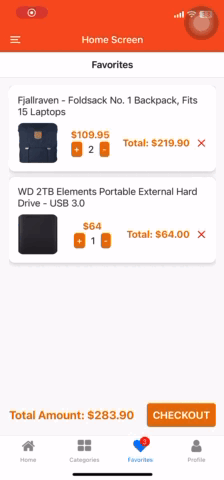
### Mục đích

**Blinking Animation** trong màn hình **FAVORITE** được sử dụng để tạo ra hiệu ứng nhấp nháy dòng thông báo chương trình “Book today to get 20% off” ở phần đầu của danh sách sản phẩm yêu thích. Hiệu ứng động nhằm tạo sự chú ý cho người dùng.

Link tài liệu: <https://aboutreact.com/react-native-blinking-animation/>

### Hiển thị màn hình





### Source code: \src\pages\Favorites.js

*(Phần liên quan: màu trắng)*

import React, { useState, useEffect } from 'react';

import { Modal, Button, FlatList, View, Text, TouchableOpacity, StyleSheet, ActivityIndicator }

from 'react-native';

import { useCart } from '../context/CartContext';

import FavoriteCard from '../components/FavoriteCard';

const FavoritesPage = ({ navigation }) => {

    const [loading, setLoading] = useState(true);

    const [modalVisible, setModalVisible] = useState(false);

    const [itemToDelete, setItemToDelete] = useState(null); //state to track the item to delete

    const { cartItemsAll, removeCartItem } = useCart();

    const [totalPrice, setTotalPrice] = useState(0);

    // Blinking animation

**const [showText, setShowText] = useState(true);**

**useEffect(() => {**

**const interval = setInterval(() => {**

**setShowText((prev) => !prev);**

**}, 1000);**

**return () => clearInterval(interval);**

**}, []);**

    useEffect(() => {

    }, [cartItemsAll]);

    const handleDeleteItem = (item) => {

    };

    return (

        <View style={styles.container}>

**{showText && (**

**<Text style={styles.blinkText}>Book today to get 20% off!</Text>**

**)}**

            {loading ? (

                <ActivityIndicator size="large" color="#E67205" />

            ) : cartItemsAll.length > 0 ? (

                <>

                    <FlatList

                            />

                        )}

                        keyExtractor={item => item.id.toString()}

                    />

                    <View style={styles.totalContainer}>

                    </View>

                </>

            ) : (

                <View style={styles.emptyCart}>

                </View>

            )}

            <Modal visible={modalVisible} transparent={true} animationType="fade">

            </Modal>

        </View>

    );

};

export default FavoritesPage;

const styles = StyleSheet.create({

**blinkText: {**

**fontSize: 18,**

**fontWeight: 'bold',**

**color: 'green',**

**textAlign: 'center',**

**marginBottom: 10,**

**},**

});

### Giải thích

* **Khai báo state để điều khiển hiển thị của phần nhấp nháy**

|  |
| --- |
| const [showText, setShowText] = useState(true); |

* Biến state **showText** được khởi tạo với giá trị mặc định là **true**. Biến này dùng để kiểm soát việc hiển thị văn bản "Book today to get 20% off!" (có hiển thị hay không).
* **UseEffect để thay đổi trạng thái hiển thị**

|  |
| --- |
| useEffect(() => {          const interval = setInterval(() => {              setShowText((prev) => !prev);          }, 1000);          return () => clearInterval(interval);      }, []); |

* **Hook useEffect** sử dụng để tạo ra một khoảng thời gian (interval) 1000ms (1 giây) và cứ mỗi 1 giây, hàm **setShowText** sẽ được gọi để thay đổi giá trị showText từ true sang false và ngược lại, giúp tạo ra hiệu ứng nhấp nháy liên tục cho văn bản.
* **setInterval** tạo ra một vòng lặp với khoảng thời gian là 1 giây, trong đó showText được cập nhật để thay đổi giữa trạng thái hiển thị và ẩn.
* **clearInterval(interval):** Khi component bị hủy (unmount), hàm này sẽ được gọi để dừng vòng lặp, tránh lỗi rò rỉ bộ nhớ.
* **Điều kiện hiển thị văn bản nhấp nháy**

|  |
| --- |
| {showText && (          <Text style={styles.blinkText}>Book today to get 20% off!</Text>      )} |

* Nếu **showTex**t là true, văn bản "Book today to get 20% off!" sẽ được hiển thị. Khi **showText** là false, văn bản này sẽ không xuất hiện trên giao diện.

**Tóm lại:**

Tóm lại, hiệu ứng nhấp nháy được tạo ra bằng cách thay đổi giá trị của showText mỗi 1 giây, từ đó điều khiển việc hiển thị hoặc ẩn văn bản.

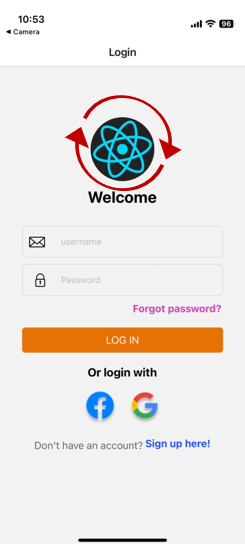
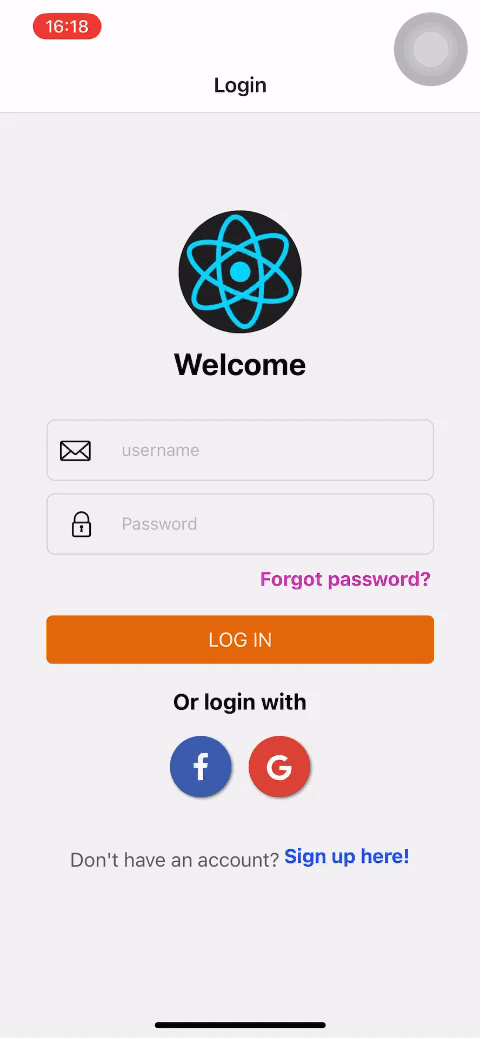
## Rotate Image Animation

### Mục đích

**Rotate Image Animation** trong màn hình **LOGIN** được sử dụng để tạo ra hiệu ứng xoay tròn icon khi người dùng ở màn hình này, giúp giao diện sinh động hơn.

Link tài liệu: <https://aboutreact.com/react-native-rotate-image-view-using-animation/>

### Hiển thị màn hình

 ****

### Source code: \src\pages\Login.js

*(Phần liên quan: màu trắng)*

import React, { useState, useContext, useEffect, useRef } from "react";

import { StyleSheet, View, Image, Text, TextInput, TouchableOpacity, Alert,

**Animated**, Easing } from 'react-native';

import EmailIcon from 'react-native-vector-icons/Fontisto';

import PasswordIcon from 'react-native-vector-icons/EvilIcons';

import reactIcon from '../../assets/react-icon.png';

import facebook from '../../assets/facebook.png';

import google from '../../assets/google.png';

import { AuthContext } from "../components/Auth";

export default function LoginPage({ navigation }) {

    const [username, setUsername] = useState('');

    const [password, setPassword] = useState('');

    const { login } = useContext(AuthContext);

    const handleLogin = async () => {

    };

    // rotating continuously:

**const rotateValue = useRef(new Animated.Value(0)).current;**

**useEffect(() => {**

**const rotateImage = () => {**

**Animated.loop(**

**Animated.timing(rotateValue, {**

**toValue: 1,**

**duration: 3000,**

**easing: Easing.linear,**

**useNativeDriver: true,**

**})**

**).start();**

**};**

**rotateImage();**

**}, [rotateValue]);**

**const rotation = rotateValue.interpolate({**

**inputRange: [0, 1],**

**outputRange: ['0deg', '360deg'],**

**});**

    return (

        <View style={styles.container}>

            {/\* Header \*/}

            <View style={styles.header}>

**<Animated.Image**

**source={reactIcon}**

**style={[styles.reactIcon, { transform: [{ rotate: rotation }] }]}**

**/>**

                <Text style={styles.headerText}>Welcome</Text>

            </View>

            {/\* username + Password text input form\*/}

            <View style={styles.body}>

                <View style={styles.inputItem}>

                    <EmailIcon name="email" size={25} color="#000" style={styles.icon} />

                    <TextInput

{/\* … \*/}

                    />

                </View>

                <View style={styles.inputItem}>

                    <PasswordIcon name="lock" size={35} color="#000" style={styles.icon} />

                    <TextInput

{/\* … \*/}

                    />

                </View>

                {/\* forgot password \*/}

                {/\* Login Button \*/}

                {/\* Login by another method \*/}

                {/\* Sign up \*/}

            </View>

        </View>

    ); }

### Giải thích

* **Khởi tạo giá trị xoay**

|  |
| --- |
| const rotateValue = useRef(new Animated.Value(0)).current; |

* **useRef(new Animated.Value(0))** tạo một giá trị khởi đầu là **0** cho rotateValue, và giá trị này được lưu giữ giữa các lần render của component.
* **Animated.Value(0)** là một giá trị hoạt họa có thể thay đổi dần theo thời gian.
* **UseEffect để khởi động hiệu ứng xoay**

|  |
| --- |
| useEffect(() => {          const rotateImage = () => {              Animated.loop(                  Animated.timing(rotateValue, {                      toValue: 1,                      duration: 3000,                      easing: Easing.linear,                      useNativeDriver: true,                  })              ).start();          };          rotateImage();      }, [rotateValue]); |

* **useEffect** đảm bảo rằng hiệu ứng xoay chỉ được khởi động một lần khi component được render lần đầu.
* **Animated.loop** giúp lặp lại liên tục hoạt động xoay.
* **Animated.timing** được sử dụng để thay đổi:
  + **rotateValue từ 0 đến 1**
  + trong thời gian **3 giây** (3000 ms)
  + **Easing.linear** giúp đảm bảo chuyển động xoay diễn ra đều.
  + **useNativeDriver: true** giúp cải thiện hiệu suất bằng cách xử lý animation trực tiếp trên lớp native.
* **Liên kết giá trị xoay với góc độ xoay**

|  |
| --- |
| const rotation = rotateValue.interpolate({          inputRange: [0, 1],          outputRange: ['0deg', '360deg'],      }); |

* **interpolate** chuyển đổi giá trị từ **rotateValue (từ 0 đến 1)** thành các giá trị góc độ tương ứng từ **'0deg' đến '360deg'**. Điều này có nghĩa là khi rotateValue tăng từ 0 đến 1, hình ảnh sẽ xoay một vòng 360 độ.
* **Ứng dụng hiệu ứng xoay vào hình ảnh**

|  |
| --- |
| <Animated.Image          source={reactIcon}          style={[styles.reactIcon, { transform: [{ rotate: rotation }] }]}      /> |

* **Animated.Image** được sử dụng thay cho Image để thêm hiệu ứng hoạt họa. Trong thuộc tính style, **transform: [{ rotate: rotation }]** sẽ làm cho hình ảnh reactIcon xoay theo giá trị của rotation, tức là liên tục xoay 360 độ.

**Tóm lại:**

Tóm lại, hiệu ứng xoay liên tục của hình ảnh được tạo ra bằng cách thay đổi giá trị hoạt họa rotateValue từ 0 đến 1 trong 3 giây, và sau đó lặp lại quá trình này. Giá trị này sau đó được chuyển đổi thành góc xoay (rotation) từ 0 đến 360 độ và được áp dụng cho hình ảnh.

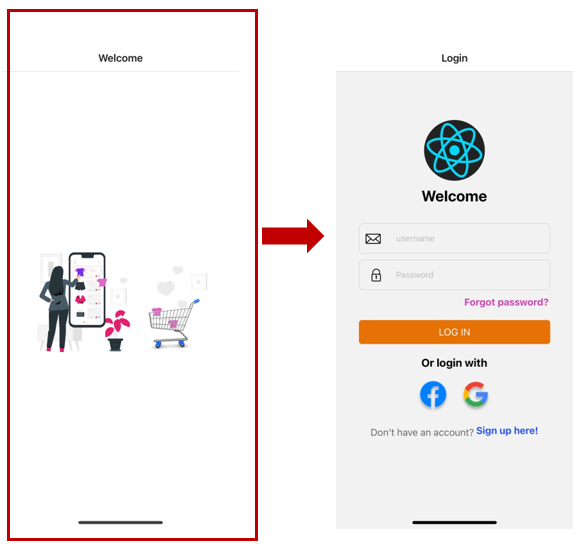
## React Native Lottie

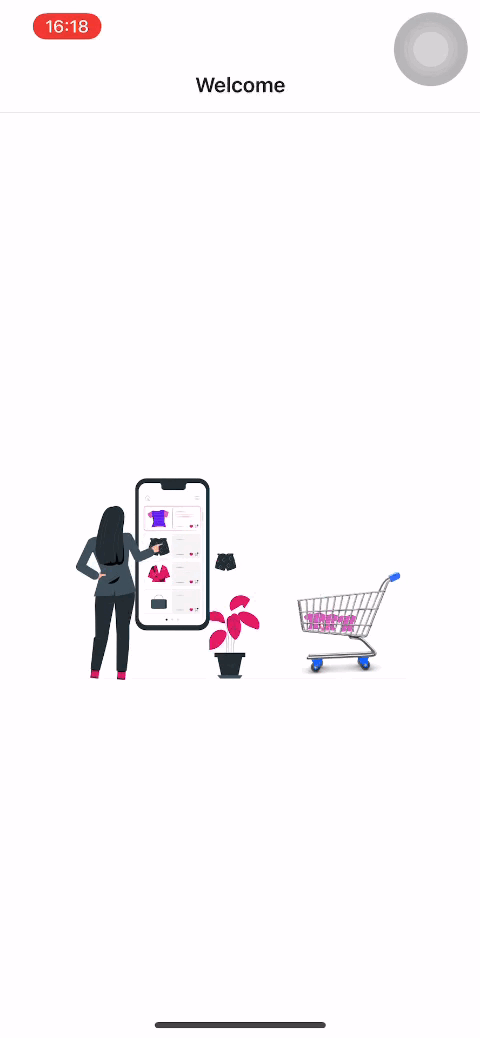
### Mục đích

**React Native Lottie** trong màn hình **Welcome** được sử dụng để tạo ra trang chào mừng bằng hoạt ảnh, ngay khi người dùng mở ứng dụng (khi không ở trong phiên đăng nhập). Màn hình chào mừng sẽ tự động chuyển đến trang Login sau 5s.

Link tài liệu: <https://aboutreact.com/react-native-lottie/>

### Hiển thị màn hình





### Tải Lottie file

Tải Json file tại trang: <https://app.lottiefiles.com/animation> và lưu file theo uri: assets/welcome-animation.json

### Source code: \src\pages\Welcome.js

*(Phần liên quan: màu trắng)*

import React, { useEffect } from 'react';

import { View, StyleSheet } from 'react-native';

import LottieView from 'lottie-react-native';

import welcomeImage from '../../assets/welcome-animation.json';

export default function WelcomePage({ navigation }) {

    useEffect(() => {

        // Chuyển sang trang login sau 5 giây

**const timer = setTimeout(() => {**

**navigation.replace('Login');**

**}, 5000);**

        return () => clearTimeout(timer);

    }, [navigation]);

    return (

        <View style={styles.container}>

            {/\* Hiệu ứng Lottie \*/}

            <LottieView

                source={welcomeImage}

                autoPlay

                loop

                style={styles.animation}

            />

        </View>

    );

}

const styles = StyleSheet.create({

    container: {

        flex: 1,

        justifyContent: 'center',

        alignItems: 'center',

        backgroundColor: '#fff',

    },

    animation: {

        width: 300,

        height: 300,

    },

});

### Giải thích

* **Thư viện và Tập tin**

|  |
| --- |
| import LottieView from 'lottie-react-native';  import welcomeImage from '../../assets/welcome-animation.json'; |

* **LottieView:** Một thư viện để hiển thị hoạt ảnh Lottie.
* **welcomeImage:** Tập tin JSON chứa hoạt ảnh Lottie, được lưu trong thư mục assets.
* **UseEffect**

|  |
| --- |
| useEffect(() => {          // Chuyển sang trang login sau 5 giây          const timer = setTimeout(() => {              navigation.replace('Login');          }, 5000);          return () => clearTimeout(timer);  // Dọn dẹp khi component bị unmount      }, [navigation]); |

* **setTimeout:** Thiết lập một timer để chuyển hướng đến trang Login sau 5s.
* **navigation.replace('Login'):** Thay thế trang hiện tại bằng trang Login, nghĩa là không thể quay lại trang chào mừng sau 5s.
* **clearTimeout(timer):** Dọn dẹp timer khi component bị unmount để tránh rò rỉ bộ nhớ.
* **Render giao diện**

|  |
| --- |
| <LottieView          source={welcomeImage}          autoPlay          loop          style={styles.animation}      /> |

* **LottieView:** Hiển thị hoạt ảnh Lottie từ tập tin welcomeImage. Các thuộc tính:
  + **source**: Nguồn dữ liệu cho hoạt ảnh.
  + **autoPlay**: Tự động phát hoạt ảnh khi trang được tải.
  + **loop**: Lặp lại hoạt ảnh.
  + **style**: Kích thước của hoạt ảnh.

**Tóm lại:**

Tóm lại, Component này tạo ra một trang chào mừng đơn giản với hoạt ảnh Lottie. Sau 5 giây, nó tự động chuyển hướng người dùng đến trang đăng nhập. Điều này đem lại trải nghiệm người dùng tốt hơn cho ứng dụng.

# Tìm hiểu về Icon

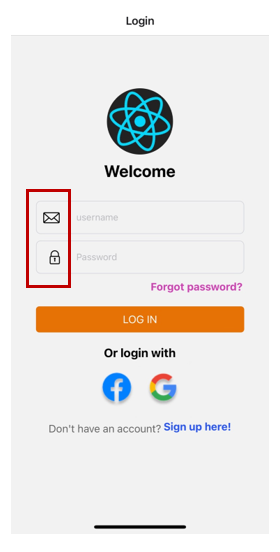
## Vector Icons

### Mục đích

**Icon** trong màn hình **Login** được sử dụng để hiển thị biểu tượng lá thư và khóa tại ô nhập thông tin đăng nhập của người dùng.

Link tài liệu: <https://aboutreact.com/react-native-vector-icons/>

### Hiển thị màn hình



### Source code: \src\pages\Home.js

*(Phần liên quan: màu trắng)*

import React, { useState, useContext, useEffect, useRef } from "react";

import { StyleSheet, View, Image, Text, TextInput, TouchableOpacity, Alert } from 'react-native';

**import EmailIcon from 'react-native-vector-icons/Fontisto';**

**import PasswordIcon from 'react-native-vector-icons/EvilIcons';**

import reactIcon from '../../assets/react-icon.png';

import facebook from '../../assets/facebook.png';

import google from '../../assets/google.png';

import { AuthContext } from "../components/Auth";

export default function LoginPage({ navigation }) {

    const [username, setUsername] = useState('');

    const [password, setPassword] = useState('');

    return (

        <View style={styles.container}>

            {/\* Header \*/}

            <View style={styles.header}>

                <Image source={reactIcon} style={styles.reactIcon} />

                <Text style={styles.headerText}>Welcome</Text>

            </View>

            {/\* username + Password text input \*/}

            <View style={styles.body}>

                <View style={styles.inputItem}>

**<EmailIcon name="email" size={25} color="#000" style={styles.icon} />**

                    <TextInput

                        style={styles.input}

                        placeholder="username"

                        value={username}

                        onChangeText={setUsername}

                    />

                </View>

                <View style={styles.inputItem}>

**<PasswordIcon name="lock" size={35} color="#000" style={styles.icon} />**

                    <TextInput

                        style={styles.input}

                        placeholder="Password"

                        value={password}

                        onChangeText={setPassword}

                        secureTextEntry

                    />

                </View>

                {/\* forgot password \*/}

                {/\* Login Button \*/}

                {/\* Login by another method \*/}

                {/\* Sign up \*/}

            </View>

        </View>

    );

}

const styles = StyleSheet.create({

**icon: {**

**marginRight: 10,**

**width: 30,**

**},**

});

### Giải thích

* **Import các biểu tượng**

|  |
| --- |
| import EmailIcon from 'react-native-vector-icons/Fontisto';  import PasswordIcon from 'react-native-vector-icons/EvilIcons'; |

* **Animated.Value(0):** là giá trị khởi tạo cho Animated. Giá trị này thay đổi khi cuộn nội dung và được liên kết với hiệu ứng cuộn (**onScroll**).
* Giá trị này sẽ thay đổi dựa trên vị trí cuộn (**scroll offset**) của người dùng và được sử dụng để tạo ra các hiệu ứng khác nhau cho phần header.
* **Hiển thị các biểu tượng**

|  |
| --- |
| <View style={styles.inputItem}>  **<EmailIcon name="email" size={25} color="#000" style={styles.icon} />**          <TextInput              style={styles.input}              placeholder="username"              value={username}              onChangeText={setUsername}          />      </View>      <View style={styles.inputItem}>  **<PasswordIcon name="lock" size={35} color="#000" style={styles.icon} />**          <TextInput              style={styles.input}              placeholder="Password"              value={password}              onChangeText={setPassword}              secureTextEntry          />      </View> |

* **<EmailIcon /> và <PasswordIcon/>:** sử dụng các thuộc tính:
  + **name="email"**, **name="lock"**: tên của biểu tượng muốn hiển thị. Trong trường hợp này, nó là biểu tượng email từ bộ **Fontisto** và **EvilIcon**.
  + size={}: xác định kích thước của biểu tượng.
  + color="#000": xác định màu sắc của biểu tượng.
  + style={styles.icon}: áp dụng các kiểu CSS từ một đối tượng styles.

**Tóm lại:**

Sử dụng các Icon trong từ thư viện react-native-vector-icons giúp giao diện đa dạng và trực quan hơn.

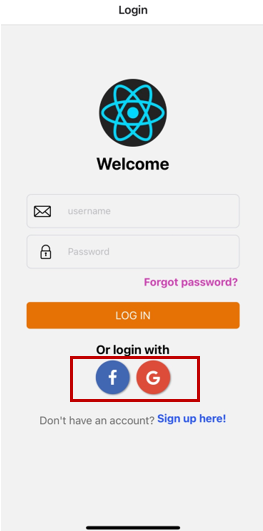
## Social Icons

### Mục đích

**Social Icon** trong màn hình **Login** được sử dụng để hiển thị biểu tượng Facebook và Google tại phần chọn phương thức đăng nhập từ thư viện có sẵn.

Link tài liệu: <https://aboutreact.com/react-native-social-icons/>

### Hiển thị màn hình



### Source code: \src\pages\Home.js

*(Phần liên quan: màu trắng)*

import React, { useState, useContext, useEffect, useRef } from "react";

import { StyleSheet, View, Text, TextInput, TouchableOpacity, Alert, Animated, Easing }

from 'react-native';

**import { SocialIcon } from "react-native-elements";**

export default function LoginPage({ navigation }) {

{/\* logic here …. \*/}

    return (

        <View style={styles.container}>

            {/\* Header \*/}

                {/\* <Image source={reactIcon} style={styles.reactIcon} /> \*/}

            {/\* username + Password text input \*/}

            <View style={styles.body}>

                {/\* forgot password \*/}

                {/\* Login Button \*/}

                {/\* Login by another method \*/}

                <Text style={styles.otherLoginText}>Or login with</Text>

                <View style={styles.socialLogin}>

**<SocialIcon**

**type="facebook"**

**button**

**style={styles.socialIcon}**

**onPress={() => { Alert.alert('', 'Login with Facebook'); }}**

**/>**

**<SocialIcon**

**type="google"**

**button**

**style={styles.socialIcon}**

**onPress={() => { Alert.alert('', 'Login with Google'); }}**

**/>**

                </View>

                {/\* Sign up \*/}

            </View>

        </View>

    );

}

const styles = StyleSheet.create({

**socialLogin: {**

**flexDirection: 'row',**

**justifyContent: 'space-between',**

**justifyContent: 'center', // Căn giữa**

**marginBottom: 20,**

**},**

**socialIcon: {**

**width: 50,**

**height: 50,**

**borderRadius: 30,**

**},**

});

### Giải thích

* **Import SocialIcon từ thư viện react-native-elements**

|  |
| --- |
| import { SocialIcon } from "react-native-elements"; |

* **SocialIcon:** là thành phần giao diện, cung cấp các biểu tượng của các trang mạng xã hội.
* **Hiển thị các biểu tượng**

|  |
| --- |
| <View style={styles.socialLogin}>          <SocialIcon              type="facebook"              button              style={styles.socialIcon}              onPress={() => { Alert.alert('', 'Login with Facebook'); }}          />          <SocialIcon              type="google"              button              style={styles.socialIcon}              onPress={() => { Alert.alert('', 'Login with Google'); }}          />      </View> |

* Các thuộc tính chính của SocialIcon:
* **type**: Xác định loại biểu tượng mạng xã hội. Các loại phổ biến bao gồm: 'facebook', 'google', 'twitter', 'linkedin', v.v.
* **onPress**: Hàm sẽ được gọi khi người dùng nhấn vào biểu tượng.
* **button**: Nếu muốn biểu tượng như một nút, đặt thuộc tính này thành true (mặc định là true).
* **iconSize**: Kích thước biểu tượng (mặc định là 24).
* **style**: thêm các thuộc tính CSS

**Tóm lại:**

Sử dụng SocialIcon từ react-native-elements giúp dễ dàng thêm các biểu tượng mạng xã hội vào ứng dụng. Ngoài ra, có thể tùy biến nhiều thuộc tính để phù hợp với thiết kế giao diện người dùng.

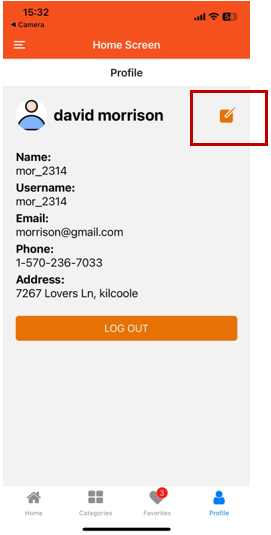
## ionIcons

### Mục đích

**ionIcons** trong màn hình **PROFILE** được sử dụng để hiển thị biểu tượng **cây bút chì**, biểu thị chức năng chỉnh sửa.

Link tài liệu: <https://aboutreact.com/react-native-ionicons/>

### Hiển thị màn hình



### Source code: \src\pages\profile.js

*(Phần liên quan: màu trắng)*

import React, { useContext, useState, useEffect } from 'react';

import { View, Text, Image, TouchableOpacity, StyleSheet, ActivityIndicator }

from 'react-native';

import { AuthContext } from "../components/Auth";

const userIcon = require('../../assets/user.png');

**import { Ionicons } from '@expo/vector-icons';**

const AccountScreen = ({ navigation }) => {

    {/\* Login here… \*/}

    return (

        <View style={styles.container}>

            {loading ? (

                <ActivityIndicator size="large" color="#0000ff" />

            ) : (

            <>

                <View style={styles.section1}>

                    <View style={styles.section1left}>

                        <Image source={userIcon} style={styles.profileImage} />

  <Text style={styles.fullName} >

{userData.name.firstname} {userData.name.lastname}

</Text>

                    </View>

**<Ionicons name="create" size={30} color='#E67205' onPress={handleEdit} />**

                </View>

                <View style={styles.infoContainer}>

                    <Text style={styles.label}>Name:</Text>

                    <Text style={styles.userInfo}>{userData.username}</Text>

                    <Text style={styles.label}>Username:</Text>

                    <Text style={styles.userInfo}>{userData.username}</Text>

                    <Text style={styles.label}>Email:</Text>

                    <Text style={styles.userInfo}>{userData.email}</Text>

                    <Text style={styles.label}>Phone:</Text>

                    <Text style={styles.userInfo}>{userData.phone}</Text>

                    <Text style={styles.label}>Address:</Text>

                    {/\* Ensure to access the correct property of address \*/}

                    <Text style={styles.userInfo}>

{userData.address ?

userData.address.number + ' ' +

userData.address.street + ', ' +

userData.address.city :

'N/A'}

</Text>

                </View>

                <TouchableOpacity style={styles.button} onPress={handleLogout}>

                        <Text style={styles.btnText}>LOG OUT</Text>

                </TouchableOpacity>

            </>

            )}

        </View>

    );

};

export default AccountScreen;

### Giải thích

* **Import Ionicons**

|  |
| --- |
| import { Ionicons } from '@expo/vector-icons'; |

* Nhập **Ionicons** từ thư viện **@expo/vector-icons**. Đây là một gói các bộ icon tích hợp sẵn cho các ứng dụng Expo, giúp sử dụng các icon phổ biến mà không cần cài đặt các thư viện bên ngoài khác.
* **Hiển thị các biểu tượng**

|  |
| --- |
| <View style={styles.section1}>          <View style={styles.section1left}>              <Image source={userIcon} style={styles.profileImage} />              <Text style={styles.fullName}>  {userData.name.firstname} {userData.name.lastname}  </Text>          </View>  **<Ionicons name="create" size={30} color='#E67205' onPress={handleEdit} />**      </View> |

* Các thuộc tính chính của Ionicons:
  + **name="create"**: Tên của biểu tượng cây bút chì trong Ionicons.
  + **size={30}**: Kích thước của biểu tượng được đặt là 30.
  + **color='#E67205'**: Màu sắc của biểu tượng là màu cam (#E67205).
  + **onPress={handleEdit}**: Khi người dùng nhấn vào biểu tượng, hàm handleEdit sẽ được gọi để điều hướng đến trang chỉnh sửa hồ sơ người dùng.

**Tóm lại:**

Ionicons được sử dụng để hiển thị biểu tượng một cách dễ dàng và nhanh chóng trong ứng dụng React Native với Expo, giúp cải thiện giao diện người dùng mà không cần cấu hình phức tạp.

## Font-Awesome Icons

### Mục đích

**FontAwesome** cung cấp một bộ sưu tập biểu tượng rất lớn với hàng ngàn biểu tượng thuộc nhiều chủ đề khác nhau như mạng xã hội, giao diện người dùng, công cụ, thời tiết, phương tiện truyền thông, và nhiều hơn nữa.

Tại màn hình HOME, hiển thị biểu tượng dấu cộng để làm nút bấm thêm vào danh mục yêu thích ở từng mục sản phẩm.

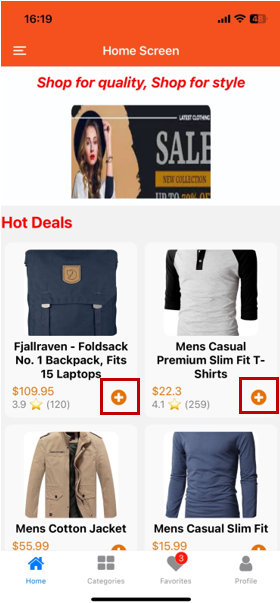
Link tài liệu: <https://aboutreact.com/react-native-fontawesome/>

### Cài đặt Font-Awesome Icon trong Expo

Expo đã tích hợp sẵn bộ icon @expo/vector-icons, bao gồm Font-Awesome. Cài đặt thư viện này như sau:

expo install @expo/vector-icons

### Hiển thị màn hình



### Source code: \src\components\AddToCart.js

*(Phần liên quan: màu trắng)*

import React, { useState } from 'react';

import { StyleSheet, View, Text, ActivityIndicator, TouchableOpacity, Alert }

from 'react-native';

import { FontAwesome } from '@expo/vector-icons';

const AddToCartButton = ({ userId, productId }) => {

    const [loading, setLoading] = useState(false);

    const [error, setError] = useState(null);

    const addToCart = async () => {

      setLoading(true);

      setError(null);

      try {

// LOGIC here…

  }

    return (

      <View>

        {loading ? (

          <ActivityIndicator size="small" color="#0000ff" />

        ) : (

        <TouchableOpacity

            style={styles.addButton}

            onPress={() => {

                addToCart();

                Alert.alert('Message', 'This product is already in your cart');

            }}

        >

**<FontAwesome name='plus-circle' size={25} color={'#E67205'} />**

        </TouchableOpacity>

        )}

        {error && <Text style={{ color: 'red' }}>{error}</Text>}

      </View>

    );

  };

export default AddToCartButton;

### Giải thích

* **Import FontAwesome**

|  |
| --- |
| import { FontAwesome } from '@expo/vector-icons'; |

* **FontAwesome** là một bộ thư viện các biểu tượng (**icons**) phổ biến, cung cấp hàng trăm biểu tượng cho nhiều loại ứng dụng khác nhau.
* Thư viện này được cung cấp bởi **@expo/vector-icons** trong React Native, cho phép bạn sử dụng các biểu tượng FontAwesome dễ dàng trong ứng dụng.
* **Hiển thị các biểu tượng**

|  |
| --- |
| <FontAwesome name='plus-circle' size={25} color={'#E67205'} /> |

* Các thuộc tính chính của FontAwesome:
  + **name='plus-circle'**: Tên của biểu tượng dấu cộng trong hình tròn từ FontAwesome.
  + **size={25}**: Kích thước của biểu tượng.
  + **color={'#E67205'}**: Màu sắc của biểu tượng.

**Tóm lại:**

**FontAwesome** là một công cụ mạnh mẽ cho việc sử dụng biểu tượng trong ứng dụng, nhờ vào sự phong phú, tính linh hoạt, và khả năng tối ưu hóa. Việc dễ dàng tích hợp với React và React Native giúp nó trở thành lựa chọn ưu tiên cho các nhà phát triển.

## RN Currency Symbol

### Mục đích

**RN Currency Symbols** dùng để hiển thị ký hiệu của nhiều loại tiền tệ khác nhau bằng mã Unicode

Dưới đây là hiển thị theo đơn vị tiền tệ USD.

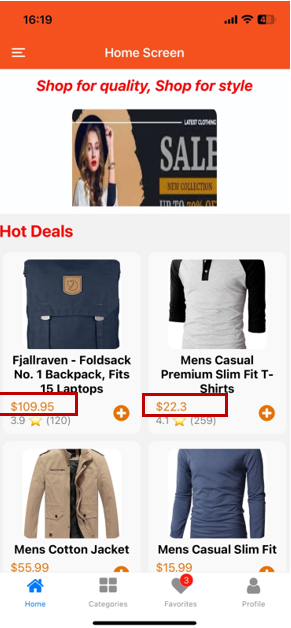
Link tài liệu: <https://aboutreact.com/react-native-currency-symbols/>

### Cài đặt thư viện

Cài đặt thư viện này như sau:

npm install react-native-currency-symbols

### Hiển thị màn hình



### Source code: \src\components\ProductCard.js

*(Phần liên quan: màu trắng)*

ProductCard.js là component được gọi vào màn hình Home để hiển thị từng mục sản phẩm. trong có có dòng giá tiền $.

import React from "react";

import { StyleSheet, View, Text, TouchableOpacity, Image } from 'react-native';

import AddToCartButton from './AddToCart';

const ProductCard = ({ item, handleProductClick, userId }) => {

    return(

        <TouchableOpacity

            onPress={() => handleProductClick(item)}

            style={styles.productCard}

        >

            <Image source={{ uri: item.image }} style={styles.productImage} />

            <Text style={styles.productTitle}>{item.title}</Text>

            <View style={styles.itemInfo}>

                <View style={styles.itemInfo1}>

                    {/\* dollar sign \*/}

                    <Text style={styles.productPrice}>{'\u0024'}{item.price}</Text>

                <Text style={styles.productRating}>

{item.rating.rate} ⭐ ({item.rating.count})

</Text>

                </View>

                <View style={styles.addButtonContainer}>

                    <AddToCartButton userId={userId} productId={item.id} />

                </View>

            </View>

        </TouchableOpacity>

    );

};

export default ProductCard;

### Giải thích

* **Hiển thị các biểu tượng**

|  |
| --- |
| <Text style={styles.productPrice}>{'\u0024'}{item.price}</Text> |

* **{'\u0024'}** là mã của tiền USD.

**Tóm lại:**

**RN Currency Symbols** trong React Native là một thư viện hoặc tập hợp các ký hiệu tiền tệ được sử dụng trong ứng dụng di động. Nó giúp hiển thị biểu tượng của các loại tiền tệ như $, €, ¥, ₫, v.v., phù hợp với quốc gia hoặc khu vực người dùng.

# Tìm hiểu về Language & Fonts

## RN Custom Fonts

### Mục đích

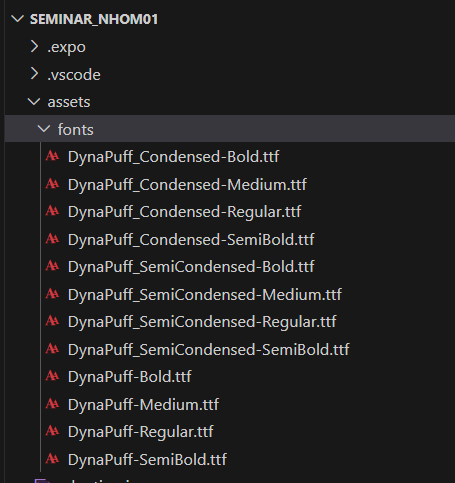
**RN Custom Fonts** dùng để hiển thì các kiểu chữ theo nhu cầu nhà phát triển.

Trong bài này, cần thay đổi font chữ tiêu đề tên sản phẩm tại màn hình **HomeDetails.**

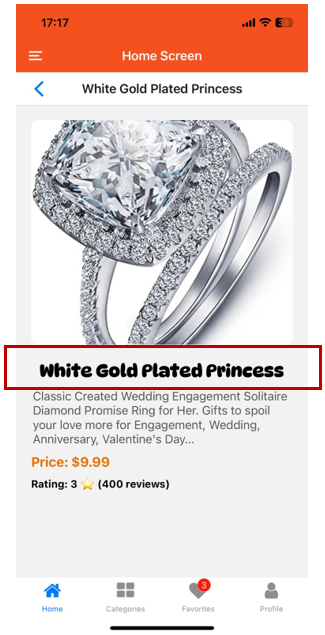
Link tài liệu: [https://aboutreact.com/custom-fonts-in-react-native /](https://aboutreact.com/custom-fonts-in-react-native%20/)

### Tải fonts

Tại <https://fonts.google.com/selection>, tải font chữ về máy, giải nén và lưu các font chữ cần lấy từ mục static vào thư mục của dự án: **assets/fonts**

****

### Hiển thị màn hình



### Source code: \src\components\HomeDetails.js

*(Phần liên quan: màu trắng)*

import React, { useEffect, useState } from "react";

import { StyleSheet, View, Text, Image, ActivityIndicator, TouchableOpacity }

from 'react-native';

**import \* as Font from 'expo-font';** // Import expo-font

**import { useFonts } from 'expo-font';** // Import useFonts

export default function HomeDetailsPage({ route, navigation }) {

    const { product } = route.params;

    const [loading, setLoading] = useState(true);

    // Load fonts

**let [fontsLoaded] = useFonts({**

**CustomBoldFont: require('../../assets/fonts/DynaPuff-Bold.ttf'),**

**CustomRegularFont: require('../../assets/fonts/DynaPuff-Regular.ttf'),**

**});**

**if (!fontsLoaded) {**

**return <ActivityIndicator size="large" color="#0000ff" />;**

**}**

// Logic here…

    return (

        <View style={styles.container}>

            {loading ? (

                <ActivityIndicator size="large" color="#0000ff" />

            ) : (

            <>

                <Image source={{ uri: product.image }} style={styles.productImage} />

**<Text style={styles.productTitle}>{product.title}</Text>**

                <Text style={styles.productDescription}>{product.description}</Text>

                <Text style={styles.productPrice}>Price: {'\u0024'}{product.price}</Text>

                <Text style={styles.productRating}>

Rating: {product.rating.rate} ⭐ ({product.rating.count} reviews)

</Text>

            </>

            )}

        </View>

    );

}

const styles = StyleSheet.create({

    productTitle: {

**fontFamily: 'CustomBoldFont',**   //set font family here

        fontSize: 24,

        fontWeight: 'bold',

        textAlign: 'left',

        marginBottom: 10,

    },

    productDescription: {

        fontSize: 16,

        textAlign: 'left',

        marginBottom: 10,

        color: '#555',

    },

    // …

});

### Giải thích

* **Import font từ thư mục assets**

|  |
| --- |
| import \* as Font from 'expo-font'; // Import expo-font  import { useFonts } from 'expo-font'; // Import useFonts |

* **expo-font:** Thư viện cho phép tải và sử dụng font chữ tùy chỉnh trong ứng dụng React Native.
* **useFonts**: Một hook được cung cấp bởi expo-font để dễ dàng tải font chữ trong component.
* **Tải font**

|  |
| --- |
| let [fontsLoaded] = useFonts({          CustomBoldFont: require('../../assets/fonts/DynaPuff-Bold.ttf'),          CustomRegularFont: require('../../assets/fonts/DynaPuff-Regular.ttf'),      }); |

* **useFonts:** để tải các font chữ tùy chỉnh:
* **CustomBoldFont:** Font chữ đậm.
* **CustomRegularFont:** Font chữ thường.
* Có thể tải nhiều font chữ chứa trong thư mục nếu cần
* **Kiểm Tra Trạng Thái Tải Font**

|  |
| --- |
| if (!fontsLoaded) {          return <ActivityIndicator size="large" color="#0000ff" />;      } |

* Kiểm tra trạng thái **fontsLoaded.** Nếu font chưa được tải xong, nó sẽ hiển thị một **ActivityIndicator** (vòng xoay) để thông báo cho người dùng rằng đang tải dữ liệu. Điều này đảm bảo rằng giao diện người dùng không hiển thị trước khi font chữ được tải xong.
* **Sử dụng font trong tiêu đề**

|  |
| --- |
| <Text style={styles.productTitle}>{product.title}</Text> |

* Sử dụng component **Text** để hiển thị tiêu đề sản phẩm. trong đó kiểu chữ được định nghĩa trong styles.productTitle
* **Sử dụng font trong tiêu đề**

|  |
| --- |
| productTitle: {  **fontFamily: 'CustomBoldFont',**   //set font family here          fontSize: 24,          fontWeight: 'bold',          textAlign: 'left',          marginBottom: 10,      }, |

* Trong styles, thuộc tính **fontFamily** được đặt là **CustomBoldFont**, điều này có nghĩa là tiêu đề sản phẩm sẽ được hiển thị bằng font chữ đậm đã tải.

**Tóm lại:**

Việc sử dụng font chữ tùy chỉnh trong ứng dụng React Native là một cách để cải thiện trải nghiệm người dùng và mang lại cảm giác chuyên nghiệp hơn.

## Multi Language

### Mục đích

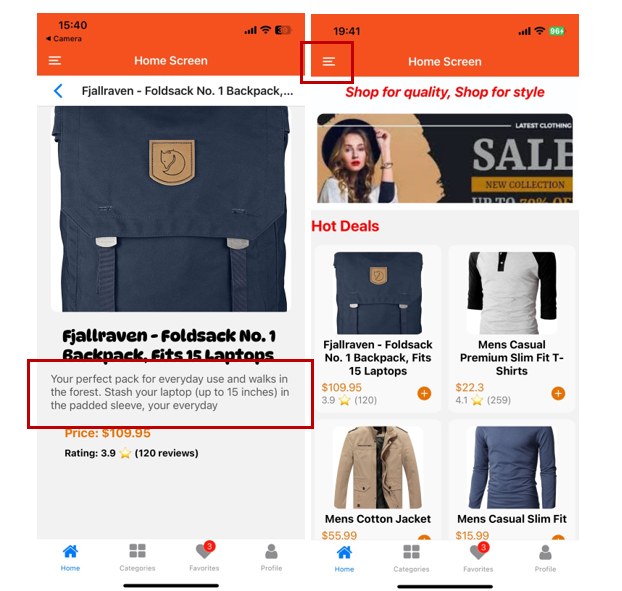
Thêm lựa chọn ngôn ngữ cho ứng dụng. Điều này giúp tiếp cận thêm nhiều người dùng sử dụng ngôn ngữ mà nhà phát triển hướng đến.

Cụ thể, trong bài này cung cấp 2 cho người dùng lựa chọn 1 trong 2 ngôn ngữ Tiếng Việt và Tiếng Anh cho phần xem chi tiết sản phẩm.

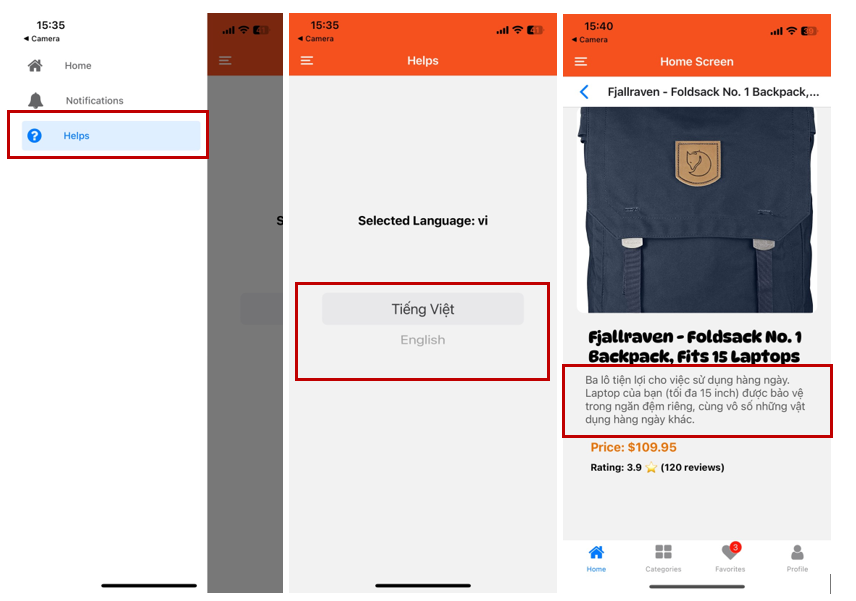
Link tài liệu: <https://aboutreact.com/localization-in-react-native/>

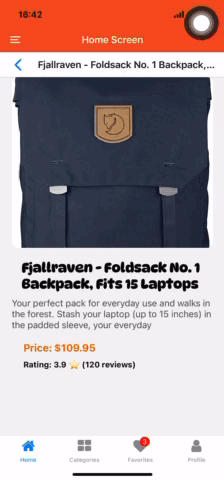
### Hiển thị màn hình

Ở trạng thái mặc định, ngôn ngữ sử dụng chính là Tiếng Anh



Khi chuyển sang chế độ tiếng Việt





### Tạo file data vi\_products.json để chứa id sản phẩm và mô tả sản phẩm bằng tiếng Việt

Cấu trúc của file json:

[

    {

      "id": 1,

      "description": "Phong cách ôm sát, tay áo dài tương phản, khóa ba nút, vải nhẹ và mềm cho cảm giác thoải mái. Áo có cổ tròn được may chắc chắn."

    },

    {

      "id": 2,

      "description": "Phong cách slim-fit, tay dài tương phản, hàng nút ba, chất liệu nhẹ & mềm mại cho sự thoáng khí và thoải mái khi mặc. Áo thun được khâu chắc chắn với cổ tròn được thiết kế để bền và phù hợp cho thời trang bình thường và các fan hâm mộ bóng chày."

    },

//…

 ]

### Tạo context quản lý ngôn ngữ: src\context\TranslationProvider.js

import React, { createContext, useContext, useState } from 'react';

import viProducts from './vi\_products.json';

const TranslationContext = createContext();

export const TranslationProvider = ({ children }) => {

    const [language, setLanguage] = useState('en');

    const toggleLanguage = () => {

        setLanguage((prevLang) => (prevLang === 'en' ? 'vi' : 'en'));

    };

    const translationCache = {};

    const translate = (product) => {

        if (language === 'vi') {

            const viProduct = viProducts.find((p) => p.id === product.id);

            return viProduct ? viProduct.description : product.description;

        }

        return product.description;

    };

    return (

        <TranslationContext.Provider

value={{ language, setLanguage, toggleLanguage, translate }}

>

            {children}

        </TranslationContext.Provider>

    );

};

export const useTranslation = () => useContext(TranslationContext);

* **TranslationContext:** Tạo một context để quản lý ngôn ngữ và các phương thức liên quan đến dịch.
* **TranslationProvider:** Là component cung cấp giá trị context cho toàn bộ ứng dụng:
  + **language**: Trạng thái hiện tại của ngôn ngữ (mặc định là tiếng Anh như dữ liệu từ API: <https://fakestoreapi.com/products>).
  + **setLanguage**: Hàm để cập nhật ngôn ngữ.
  + **toggleLanguage**: Hàm chuyển đổi giữa tiếng Anh và tiếng Việt.
  + **translate**: Hàm để dịch mô tả sản phẩm. Nếu ngôn ngữ là tiếng Việt, nó tìm kiếm mô tả trong file **vi\_products.json** dựa trên id của sản phẩm.

### Sử dụng Context trong Helps.js

import React, { useContext } from "react";

import { StyleSheet, View, Text } from 'react-native';

import { Picker } from '@react-native-picker/picker';

import { useTranslation } from '../../context/TranslationProvider';

export default function HelpsPage() {

    const { language, setLanguage } = useTranslation();

    return (

        <View style={styles.container}>

            <Text style={styles.headerText}>Selected Language: {language}</Text>

            <View style={styles.pickerSection}>

                <Picker

                    selectedValue={language}

                    onValueChange={(itemValue) => setLanguage(itemValue)}

                    style={styles.picker}

                >

                    <Picker.Item label="Tiếng Việt" value="vi" />

                    <Picker.Item label="English" value="en" />

                </Picker>

            </View>

        </View>

    );

}

* **HelpsPage:** Component cho phép người dùng chọn ngôn ngữ. Nó sử dụng useTranslation để lấy language và setLanguage từ context.
* **Picker:** Một component để cho phép người dùng chọn giữa tiếng Việt và tiếng Anh. Khi người dùng chọn ngôn ngữ, setLanguage được gọi để cập nhật ngôn ngữ hiện tại.

### Sử dụng Context trong HomeDetails.js

import React, { useEffect } from "react";

import { StyleSheet, View, Text, Image, ActivityIndicator, SafeAreaView }

from 'react-native';

import { useFonts } from 'expo-font';

**import { useTranslation } from '../context/TranslationProvider';**

export default function HomeDetailsPage({ route, navigation }) {

    const { product } = route.params;

**const { translate } = useTranslation();**

//Logic here…

    return (

        <SafeAreaView style={styles.container}>

            <Image source={{ uri: product.image }} style={styles.productImage} />

            <Text style={styles.productTitle}>{product.title}</Text>

**<Text style={styles.productDescription}>{translate(product)}</Text>**

            <Text style={styles.productPrice}>Price: {'\u0024'}{product.price}</Text>

            <Text style={styles.productRating}>

                Rating: {product.rating.rate} ⭐ ({product.rating.count} reviews)

            </Text>

        </SafeAreaView>

    );

}

* **translate(product):** Sử dụng hàm **translate** từ context để lấy mô tả sản phẩm dựa trên ngôn ngữ được chọn.

**Tóm lại:**

Phần này cho phép người dùng chọn giữa tiếng Việt và tiếng Anh, tự động cập nhật mô tả sản phẩm tương ứng khi ngôn ngữ thay đổi. Bằng cách sử dụng context và hooks, có thể dễ dàng quản lý trạng thái ngôn ngữ trong toàn bộ ứng dụng.

# Kết luận

## Kết luận

Nội dung bài báo cáo seminar đã đạt được những kết quả sau:

Đã tìm hiểu, nghiên cứu về các tính năng cơ bản như biểu tượng (Icons), hoạt ảnh (Animation) và hỗ trợ đa ngôn ngữ (Localization). Những tính năng này rất quan trọng để tạo ra những ứng dụng không chỉ hoạt động tốt mà còn mang đến trải nghiệm người dùng thú vị và dễ sử dụng.

* Có đánh giá sau thực nghiệm.

Tuy nhiên, bài Seminar vẫn còn một số hạn chế sau:

Do năng lực của bản thân và thời gian có hạn nên chưa thể tìm hiểu sâu về các tính năng trên để nâng cao trải nghiệm người dùng với các hiệu ứng chuyển động và bảo mật đa ngôn ngữ và chuẩn hóa UX/UI đa văn hóa.

## Hướng phát triển

Bổ sung thêm các tính năng còn thiếu như trên.

Nhóm sinh viên sẽ tiếp tục nghiên cứu sâu hơn về các tính năng trên nhằm giúp ích cho công tác sau này khi trải nghiệm thực tế tại đơn vị công tác.

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | https://aboutreact.com/react-native-collapsible-toolbar/ [Online]. [Accessed 19 10 2024]. |
|  |  |