**CÁC BƯỚC TRIỂN KHAI DỰ ÁN**

**1. Cài đặt dự án**

* Đầu tiên, đảm bảo rằng bạn đã cài đặt môi trường phát triển **React Native**. Sử dụng **Expo** để dễ dàng phát triển.
* Cài đặt các gói cần thiết:

npm install expo react-native firebase react-native-picker-select @expo/vector-icons

* Tạo và cấu hình Firebase cho dự án. Thêm các cấu hình Firebase vào file FirebaseConfigs.js:

import { initializeApp } from "firebase/app";

import { getFirestore } from "firebase/firestore";

import { getStorage } from "firebase/storage";

const firebaseConfig = {

// Thêm các thông tin cấu hình Firebase

};

const app = initializeApp(firebaseConfig);

export const db = getFirestore(app);

export const storage = getStorage(app);

**2. Tạo các thành phần giao diện (Components)**

* **MenuList Component**: Dùng để hiển thị danh sách các menu, bao gồm các tùy chọn như “Add Product” và “My Product”.

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

import React from 'react';

import { useRouter } from 'expo-router';

export default function MenuList() {

const menuList = [

{

id: 1,

name: 'Add Product',

icon: require('./../../assets/images/add.png'),

path: '/product/add-product'

},

{

id: 2,

name: 'My App',

icon: require('./../../assets/images/app.png'),

path: '/product/my-product'

}

];

const router = useRouter();

const onMenuClick = (item) => {

router.push(item.path);

};

return (

<FlatList

data={menuList}

numColumns={2}

renderItem={({ item }) => (

<TouchableOpacity onPress={() => onMenuClick(item)}>

<Image source={item.icon} style={{ width: 50, height: 50 }} />

<Text>{item.name}</Text>

</TouchableOpacity>

)}

/>

);

}

**3. Trang chủ (Home Screen)**

* Đây là nơi hiển thị danh sách các tùy chọn trong MenuList.

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

import React from 'react';

import MenuList from '../../components/Home/MenuList';

export default function Home() {

return (

<View style={{ padding: 20 }}>

<Text style={{ fontFamily: 'outfit-bold', fontSize: 35 }}>APP</Text>

<MenuList />

</View>

);

}

**4. Thêm sản phẩm (Add Product)**

* Tạo trang để thêm sản phẩm mới, bao gồm việc chọn ảnh sản phẩm, tên, giá, và danh mục. Dữ liệu sẽ được lưu vào Firebase Firestore.

import { View, Text, TouchableOpacity, TextInput, Image, ActivityIndicator, ToastAndroid } from 'react-native';

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

import \* as ImagePicker from 'expo-image-picker';

import { db, storage } from '../../Config/FirebaseConfigs';

import { collection, doc, getDocs, setDoc } from 'firebase/firestore';

import { getDownloadURL, ref, uploadBytes } from 'firebase/storage';

export default function AddProduct() {

const [image, setImage] = useState(null);

const [name, setName] = useState('');

const [price, setPrice] = useState('');

const [category, setCategory] = useState('');

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

const [categoryList, setCategoryList] = useState([]);

useEffect(() => {

GetCategoryList();

}, []);

const GetCategoryList = async () => {

const q = collection(db, 'Category');

const snapshot = await getDocs(q);

setCategoryList(snapshot.docs.map(doc => doc.data()));

};

const onImagePick = async () => {

let result = await ImagePicker.launchImageLibraryAsync({

mediaTypes: ImagePicker.MediaTypeOptions.Images,

allowsEditing: true,

quality: 1,

});

setImage(result.assets[0].uri);

};

const onAddNewProduct = async () => {

setLoading(true);

try {

if (image) {

const fileName = Date.now().toString() + ".jpg";

const resp = await fetch(image);

const blob = await resp.blob();

const imageRef = ref(storage, '/product-app/' + fileName);

await uploadBytes(imageRef, blob);

const downloadUrl = await getDownloadURL(imageRef);

await saveProductDetail(downloadUrl);

} else {

await saveProductDetail(null);

}

setName('');

setPrice('');

setCategory('');

setImage(null);

ToastAndroid.show('Thêm sản phẩm mới thành công!', ToastAndroid.LONG);

} catch (error) {

console.error("Error: ", error);

ToastAndroid.show('Thêm sản phẩm thất bại!', ToastAndroid.LONG);

} finally {

setLoading(false);

}

};

const saveProductDetail = async (imageUrl) => {

await setDoc(doc(db, 'ProductList', Date.now().toString()), {

name,

price,

category,

imageUrl,

});

};

return (

<View>

<TextInput value={name} onChangeText={setName} placeholder="Tên sản phẩm" />

<TextInput value={price} onChangeText={setPrice} placeholder="Giá sản phẩm" />

<TouchableOpacity onPress={onImagePick}>

<Text>Chọn ảnh</Text>

</TouchableOpacity>

<TouchableOpacity onPress={onAddNewProduct}>

{loading ? <ActivityIndicator /> : <Text>Thêm sản phẩm</Text>}

</TouchableOpacity>

</View>

);

}

**5. Danh sách sản phẩm (My Product)**

* Hiển thị tất cả các sản phẩm đã thêm từ Firestore.

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

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

import { db } from '../../Config/FirebaseConfigs';

import { collection, getDocs } from 'firebase/firestore';

import ProductListCard from '../../components/ProductList/ProductListCard';

export default function MyProduct() {

const [productList, setProductList] = useState([]);

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

useEffect(() => {

GetUserProduct();

}, []);

const GetUserProduct = async () => {

setLoading(true);

const q = collection(db, 'ProductList');

const snapshot = await getDocs(q);

setProductList(snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() })));

setLoading(false);

};

return (

<FlatList

data={productList}

renderItem={({ item }) => <ProductListCard product={item} />}

refreshing={loading}

onRefresh={GetUserProduct}

/>

);

}

**6. Xóa sản phẩm (Delete Product)**

* Chức năng để xóa sản phẩm từ Firestore.

import { Alert, ToastAndroid } from 'react-native';

import { deleteDoc, doc } from 'firebase/firestore';

import { db } from '../../Config/FirebaseConfigs';

const onDelete = (productId) => {

Alert.alert(

"Xóa sản phẩm",

"Bạn có chắc chắn muốn xóa sản phẩm này?",

[

{

text: "Hủy",

style: "cancel",

},

{

text: "Xóa",

onPress: async () => {

await deleteDoc(doc(db, 'ProductList', productId));

ToastAndroid.show("Đã xóa sản phẩm!", ToastAndroid.LONG);

},

style: "destructive",

},

]

);

};

**7. Tìm kiếm sản phẩm (Search Product)**

### Bước 1: Cài đặt thư viện fuse.js

Thư viện fuse.js giúp thực hiện tìm kiếm gần giống bằng cách so sánh chuỗi dựa trên độ tương đồng của các ký tự.

* Mở terminal và cài đặt thư viện:

bash

Copy code

npm install fuse.js

### Bước 2: Thiết kế giao diện cho chức năng tìm kiếm

* Tạo một thanh tìm kiếm (SearchBar) bằng cách sử dụng các thành phần TextInput và TouchableOpacity từ React Native.

Ví dụ:

jsx

Copy code

<View style={{

flexDirection: 'row',

alignItems: 'center',

backgroundColor: '#fff',

padding: 10,

marginVertical: 10,

borderRadius: 8,

borderWidth: 1,

borderColor: Colors.PRIMARY,

}}>

<TouchableOpacity onPress={handleSearchIconPress}>

<Ionicons name="search" size={24} color={Colors.PRIMARY} />

</TouchableOpacity>

<TextInput

placeholder="Search..."

value={searchTerm}

onChangeText={(text) => setSearchTerm(text)}

style={{ flex: 1 }}

/>

</View>

### Bước 3: Lấy dữ liệu từ Firestore

* Sử dụng hàm getDocs để lấy toàn bộ danh sách sản phẩm từ Firestore.

const searchProductByName = async () => {

const q = query(collection(db, 'ProductList'));

const querySnapshot = await getDocs(q);

const products = [];

querySnapshot.forEach((doc) => {

const data = doc.data();

products.push({ id: doc.id, ...data });

});

return products;

};

### Bước 4: Cấu hình Fuse.js để thực hiện tìm kiếm gần giống

* Sau khi đã có danh sách sản phẩm, sử dụng Fuse.js để tìm kiếm dựa trên từ khóa người dùng nhập vào.
* Ví dụ về cấu hình Fuse.js:

import Fuse from 'fuse.js';

const searchProductByName = async (name) => {

const products = await getAllProducts(); // Hàm lấy danh sách sản phẩm

// Cấu hình fuse.js

const fuse = new Fuse(products, {

keys: ['name'], // Tìm kiếm theo trường 'name'

threshold: 0.3, // Mức độ tương tự

});

// Thực hiện tìm kiếm

const result = fuse.search(name);

const filteredProducts = result.map(res => res.item);

setProductList(filteredProducts); // Cập nhật danh sách sản phẩm tìm kiếm

};

### Bước 5: Cập nhật giao diện khi tìm kiếm

* Khi tìm kiếm, bạn có thể hiển thị trạng thái "loading" để thông báo rằng đang thực hiện tìm kiếm, và sau đó cập nhật giao diện để hiển thị kết quả.

Ví dụ:

{loading && <Text>Loading...</Text>}

{!loading && productList.length === 0 && <Text>No products found</Text>}

<ExploreProductList productList={productList} />

### Bước 6: Thêm logic debounce

* Để tránh việc tìm kiếm quá thường xuyên khi người dùng nhập từ khóa, bạn có thể thêm logic "debounce" để trì hoãn việc tìm kiếm cho đến khi người dùng ngừng nhập một khoảng thời gian nhất định (ví dụ: 500ms).

Ví dụ:

useEffect(() => {

const delayDebounceFn = setTimeout(() => {

if (searchTerm.trim()) {

searchProductByName(searchTerm);

} else {

setProductList([]);

}

}, 500); // 500ms delay

return () => clearTimeout(delayDebounceFn);

}, [searchTerm]);