

CT484: PHÁT TRIỂN ỨNG DỤNG DI ĐỘNG

XÂY DỰNG ỨNG DỤNG MYSHOP - PHẦN 3

File báo cáo cần nộp là file PDF trong đó có ghi thông tin **mã sinh viên, họ tên, lớp học phần** cùng với **hình minh họa tại các bước kiểm tra kết quả thực thi, các chức năng (không cần chụp hình mã nguồn)**. Cuối file báo cáo ghi đường link đến GitHub mã nguồn của dự án.

Ứng dụng MyShop có các chức năng chính sau:

- Hiển thị và cập nhật danh mục các sản phẩm
- Xem chi tiết một sản phẩm
- Đánh dấu sản phẩm được yêu thích
- Thêm, xóa sản phẩm trong giỏ hàng
- Thực hiện đặt hàng, xem lại đơn hàng, chi tiết đơn hàng
- Đăng ký, đăng nhập
- Lưu trữ dữ liệu trên Firebase

Sinh viên chỉ cần tạo MỘT báo cáo duy nhất cho tất cả các buổi thực hành. Nộp báo cáo lên Google Classroom và đẩy code lên GitHub trong thời hạn được chỉ định của mỗi buổi để báo cáo tiến độ, không tính điểm. Chỉ bài báo cáo nộp vào buổi thực hành cuối cùng sẽ được chấm điểm. Tuy nhiên, không nộp bài báo cáo tiến độ hoặc nộp trễ hạn thì không tính điểm buổi đó.

(Tiếp tục từ kết quả phần 2)

File báo cáo buổi 3 làm tiếp tục từ file báo cáo buổi 2.

Tổng quan về kiến trúc ứng dụng

Kiến trúc mã nguồn ứng dụng gồm 3 layer:

- *Layer UI (buổi 1 & 2)*: gồm các lớp widget của ứng dụng. Nhiệm vụ chính là trình bày trạng thái/dữ liệu ra giao diện, lắng nghe các báo hiệu thay đổi trạng thái (thông qua các builder), phát sinh các sự kiện. Các widget lấy trạng thái từ widget cha đưa xuống hoặc từ [provider](#), [get_it](#) gửi đến.
- *Layer quản lý trạng thái (buổi 3)*: gồm các lớp được đặt tên là Manager (có thể có nhiều tên gọi khác: Controller, ViewModel, Bloc, ...), thường là các lớp có khả năng phát báo hiệu như *ValueNotifier* hoặc *ChangeNotifier*. Đây là nơi lưu giữa các trạng thái của ứng dụng, định nghĩa các phương thức truy xuất trạng thái mà layer UI cần. Các cập nhật thay đổi trạng thái đều nằm ở các lớp này. Khi trạng thái có thay đổi thì phát báo hiệu (e.g, gọi *notifyListeners()* trong trường hợp của *ChangeNotifier*).
- *Layer dịch vụ (buổi 4)*: đôi khi được gọi là kho dữ liệu (data repository). Đây là nơi chứa các mã lệnh nhập/xuất dữ liệu, tương tác với cơ sở dữ liệu cục bộ (SharedPreferences, SQLite, ...) hoặc máy chủ từ xa (REST API). Layer quản lý trạng thái sẽ dùng các API cung cấp bởi layer dịch vụ để đọc/ghi dữ liệu.

Bước 1: Xây dựng chức năng chọn sản phẩm yêu thích

- Hiệu chỉnh lớp **Product** (*lib/models/product.dart*) thay đổi kiểu giá trị cho **isFavorite** từ kiểu **bool** thành kiểu **ValueListenable<bool>** (từ gói *flutter/foundation.dart*). Biến kiểu **ValueListenable<bool>** cho phép widget như **ValueListenableBuilder** lắng nghe sự thay đổi giá trị của biến:

```

import 'package:flutter/foundation.dart';

class Product {
  final String? id;
  final String title;
  final String description;
  final double price;
  final String imageUrl;
  final ValueNotifier<bool> _isFavorite;

  Product({
    this.id,
    required this.title,
    required this.description,
    required this.price,
    required this.imageUrl,
    isFavorite = false,
  }) : _isFavorite = ValueNotifier(isFavorite);

  set isFavorite(bool newValue) {
    _isFavorite.value = newValue;
  }

  bool get isFavorite {
    return _isFavorite.value;
  }

  ValueNotifier<bool> get isFavoriteListenable {
    return _isFavorite;
  }

  ...
}

```

- Hiệu chỉnh widget **ProductGridTile** (*lib/ui/products/product_grid_tile.dart*) bao widget **IconButton** bằng **ValueListenableBuilder<bool>**:

```

...
widget buildGridFooterBar(BuildContext context) {
  return GridTileBar(
    backgroundColor: Colors.black87,
    leading: ValueListenableBuilder<bool>(
      valueListenable: product.isFavoriteListenable,
      builder: (ctx, isFavorite, child) {
        return IconButton(
          icon: Icon(
            isFavorite ? Icons.favorite : Icons.favorite_border,
          ),
          color: Theme.of(context).colorScheme.secondary,
          onPressed: () {
            product.isFavorite = !isFavorite;
          },
        );
      },
    ),
    ...
  );
}

```

- Kiểm tra rằng ứng dụng cho phép chọn/bỏ chọn sản phẩm yêu thích. Lưu thay đổi vào git và GitHub:

```

git add -u
git commit -m "Cai dat chuc nang chon san pham yeu thich"
git push origin master

```

Bước 2: Chuyển ProductsManager thành kiểu ChangeNotifier. Các nơi trong cây widget cùng truy xuất một đối tượng ProductsManager

- Khai báo sử dụng gói thư viện **provider** (*pubspec.yaml*):

```

dependencies:
  flutter:
    sdk: flutter
  intl: ^0.17.0
  provider: ^6.0.3

```

- Nhắc lại, để truy xuất các đối tượng được gửi xuống cây widget bởi provider, có thể sử dụng các cách sau đây (cần import 'package:provider/provider.dart'):
 - Dùng **BuildContext.read()**: đọc ra đối tượng được cung cấp
 - Dùng **BuildContext.select()**: đọc ra một giá trị dẫn xuất từ đối tượng được cung cấp. Đồng thời, lắng nghe các báo hiệu có làm thay đổi giá trị dẫn xuất đó từ đối tượng.
 - Dùng **BuildContext.watch()**: đọc ra và lắng nghe báo hiệu từ đối tượng được cung cấp
 - Dùng widget **Consumer**: đọc ra và lắng nghe báo hiệu từ đối tượng được cung cấp. Widget này tương đương với BuildContext.watch()

Widget "*lắng nghe*" báo hiệu từ đối tượng nghĩa là hàm build() của widget sẽ được gọi thực thi lại khi widget nhận báo hiệu.

- Chuyển **ProductsManager** (*lib/ui/products/products_manager.dart*) thành kiểu **ChangeNotifier**, định nghĩa các phương thức thêm/cập nhật/xóa sản phẩm trong danh sách sản phẩm quản lý bởi **ProductsManager**:

```
import 'package:flutter/foundation.dart';

import '../models/product.dart';

class ProductsManager with ChangeNotifier {
  ...
  void addProduct(Product product) {
    _items.add(
      product.copyWith(
        id: 'p${DateTime.now().toIso8601String()}',
      ),
    );
    notifyListeners();
  }

  void updateProduct(Product product) {
    final index = _items.indexWhere((item) => item.id == product.id);
    if (index >= 0) {
      _items[index] = product;
      notifyListeners();
    }
  }

  void toggleFavoriteStatus(Product product) {
    final savedStatus = product.isFavorite;
    product.isFavorite = !savedStatus;
  }

  void deleteProduct(String id) {
    final index = _items.indexWhere((item) => item.id == id);
    _items.removeAt(index);
    notifyListeners();
  }
}
```

- Hiệu chỉnh *lib/main.dart* bao widget MaterialApp với **MultiProvider** để tạo và cung cấp đối tượng **ProductsManager** cho các widget con truy xuất:

```

...
import 'package:provider/provider.dart';
...
class MyApp extends StatelessWidget {
  ...
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider(
          create: (ctx) => ProductsManager(),
        ),
      ],
      child: MaterialApp(
        ...
        onGenerateRoute: (settings) {
          if (settings.name == ProductDetailScreen.routeName) {
            final productId = settings.arguments as String;
            return MaterialPageRoute(
              builder: (ctx) {
                return ProductDetailScreen(
                  ctx.read<ProductsManager>().findById(productId),
                );
              },
            );
          }
          return null;
        },
      ),
    );
  }
}

```

- Lần lượt hiệu chỉnh các widget **ProductsGrid**, **UserProductListTile** và **UserProductsScreen** truy xuất đối tượng **ProductsManager** được gửi xuống từ provider (tự import các gói cần thiết).
 - Hiệu chỉnh widget **ProductsGrid** (*lib/ui/products/products_grid.dart*):

```

...
import '../models/product.dart';
...
@override
Widget build(BuildContext context) {
  // Đọc ra List<Product> sẽ được hiển thị từ ProductsManager
  final products = context.select<ProductsManager, List<Product>>(
    (productsManager) => showFavorites
      ? productsManager.favoriteItems
      : productsManager.items);
  ...
}

```

- Hiệu chỉnh widget **UserProductListTile** (*lib/ui/products/user_product_list_tile.dart*), trong callback *onPressed* của **IconButton**:

```

...
Widget buildDeleteButton(BuildContext context) {
  return IconButton(
    icon: const Icon(Icons.delete),
    onPressed: () {
      context.read<ProductsManager>().deleteProduct(product.id!);
      ScaffoldMessenger.of(context)
        ..hideCurrentSnackBar()
        ..showSnackBar(
          const SnackBar(
            content: Text(
              'Product deleted',
              textAlign: TextAlign.center,
            ),
          ),
        );
    },
    color: Theme.of(context).colorScheme.error,
  );
}
...

```

- Hiệu chỉnh **UserProductsScreen** (*lib/ui/products/user_products_screen.dart*) bao **ListView** bằng widget **Consumer<ProductsManager>**:

```

...
Widget buildUserProductListView() {
  return Consumer<ProductsManager>(
    builder: (ctx, productsManager, child) {
      return ListView.builder(
        itemCount: productsManager.itemCount,
        itemBuilder: (ctx, i) => Column(
          children: [
            UserProductListTile(

```

```

        productsManager.items[i],
      ),
      const Divider(),
    ],
  ),
);
},
);
}
...

```

- Kiểm tra các chức năng liên quan đến sản phẩm hoạt động đúng. Sau đó, lưu thay đổi vào git và GitHub.

```

git add -u
git commit -m "Quan ly trang thai voi provider - ProductsManager"
git push origin master

```

Bước 3: Chuyển CartManager thành kiểu ChangeNotifier. Các nơi trong cây widget cùng truy xuất một đối tượng CartManager

- Chuyển **CartManager** (*lib/ui/cart/cart_manager.dart*) thành kiểu **ChangeNotifier**, định nghĩa các phương thức thêm/cập nhật/xóa sản phẩm trong giỏ hàng quản lý bởi **CartManager**:


```

import 'package:flutter/foundation.dart';

import '../models/cart_item.dart';
import '../models/product.dart';

class CartManager with ChangeNotifier {
  Map<String, CartItem> _items = {
    ...
  };
  ...
  void addItem(Product product) {
    if (_items.containsKey(product.id)) {
      // change quantity...
      _items.update(
        product.id!,
        (existingCartItem) => existingCartItem.copyWith(
          quantity: existingCartItem.quantity + 1,
        ),
      );
    } else {
      _items.putIfAbsent(
        product.id!,
        () => CartItem(
          id: 'c${DateTime.now().toIso8601String()}',
          title: product.title,
          price: product.price,
          quantity: 1,
        ),
      );
    }
    notifyListeners();
  }

  void removeItem(String productId) {
    _items.remove(productId);
    notifyListeners();
  }

  void removeSingleItem(String productId) {
    if (!_items.containsKey(productId)) {
      return:

```

```

    }
    if (_items[productId]?.quantity as num > 1) {
      _items.update(
        productId,
        (existingCartItem) => existingCartItem.copyWith(
          quantity: existingCartItem.quantity - 1,
        ),
      );
    } else {
      _items.remove(productId);
    }
    notifyListeners();
  }

  void clear() {
    _items = {};
    notifyListeners();
  }
}

```

- Hiệu chỉnh *lib/main.dart* tạo và cung cấp đối tượng **CartManager** cho các widget con truy xuất:

```

13 13  @override
14 14  Widget build(BuildContext context) {
15 15      return MultiProvider(
16 16          providers: [
17 17              ChangeNotifierProvider(
18 18                  create: (ctx) => ProductsManager(),
19 19              ), // ChangeNotifierProvider
20+ 20+  ChangeNotifierProvider(
21+ 21+  |   create: (ctx) => CartManager(),
22+ 22+  |   ), // ChangeNotifierProvider
20 23  ],

```

- Lần lượt hiệu chỉnh các widget **CartItemCard**, **CartScreen**, **ProductGridTile** và **ProductsOverviewScreen** truy xuất đối tượng **CartManager** được gửi xuống bởi provider (tự import các gói cần thiết).
 - Hiệu chỉnh widget **CartItemCard** (*lib/ui/cart/cart_item_card.dart*):

```

...
onDismissed: (direction) {
  context.read<CartManager>().removeItem(productId);
},
child: buildItemCard(),
...

```

- Hiệu chỉnh widget **CartScreen** (*lib/ui/cart/cart_screen.dart*):

```

...
@override
Widget build(BuildContext context) {
  final cart = context.watch<CartManager>();
  ...
}

```

- Hiệu chỉnh widget **ProductGridTile** (*lib/ui/products/product_grid_tile.dart*):

```

...
Widget buildGridFooterBar(BuildContext context) {
  return GridTileBar(
    ...
    trailing: IconButton(
      ...
      onPressed: () {
        final cart = context.read<CartManager>();
        cart.addItem(product);
        ScaffoldMessenger.of(context)
          ..hideCurrentSnackBar()
          ..showSnackBar(
            SnackBar(
              content: const Text(
                'Item added to cart',
              ),
              duration: const Duration(seconds: 2),
              action: SnackBarAction(
                label: 'UNDO',
                onPressed: () {
                  cart.removeSingleItem(product.id!);
                },
              ),
            ),
          );
      },
      color: Theme.of(context).colorScheme.secondary,
    ),
  );
}
...

```

- Hiệu chỉnh widget **ProductsOverviewScreen** (*lib/ui/products/products_overview_screen.dart*) bao widget **TopRightBadge** với **Consumer<CartManager>**:

```
...  
Widget buildShoppingCartIcon() {  
  return Consumer<CartManager>(  
    builder: (ctx, cartManager, child) {  
      return TopRightBadge(  
        data: cartManager.productCount,  
        child: IconButton(  
          icon: const Icon(  
            Icons.shopping_cart,  
          ),  
          onPressed: () {  
            Navigator.of(ctx).pushNamed(CartScreen.routeName);  
          },  
        ),  
      );  
    },  
  );  
}
```

- Kiểm tra các chức năng liên quan đến giỏ hàng hoạt động đúng. Sau đó, lưu thay đổi vào git và GitHub.

```
git add -u  
git commit -m "Quan ly trang thai voi provider - CartManager"  
git push origin master
```

Bước 4: Chuyển OrdersManager thành kiểu ChangeNotifier. Các nơi trong cây widget cùng truy xuất một đối tượng OrdersManager

- Chuyển **OrdersManager** (*lib/ui/orders/orders_manager.dart*) thành kiểu **ChangeNotifier**, định nghĩa các phương thức thêm một đặt hàng mới:

```

import 'package:flutter/foundation.dart';

import '../models/cart_item.dart';
import '../models/order_item.dart';

class OrdersManager with ChangeNotifier {
  ...
  void addOrder(List<CartItem> cartProducts, double total) async {
    _orders.insert(
      0,
      OrderItem(
        id: 'o${DateTime.now().toIso8601String()}',
        amount: total,
        products: cartProducts,
        dateTime: DateTime.now(),
      ),
    );
    notifyListeners();
  }
}

```

- Hiệu chỉnh *lib/main.dart* tạo và cung cấp đối tượng **OrdersManager** cho các widget con truy xuất:

```

20      20      | | | | | ChangeNotifierProvider(
21      21      | | | | |   create: (ctx) => CartManager(),
22      22      | | | | | ), // ChangeNotifierProvider
23+      23+      | | | | | ChangeNotifierProvider(
24+      24+      | | | | |   create: (ctx) => OrdersManager(),
25+      25+      | | | | | ), // ChangeNotifierProvider

```

- Lần lượt hiệu chỉnh các widget **CartScreen** và **OrdersScreen** truy xuất đối tượng **OrdersManager** đã tạo (tự import các gói cần thiết).
 - Hiệu chỉnh widget **CartScreen** (*lib/ui/cart/cart_screen.dart*):

```

...
TextButton(
  onPressed: cart.totalAmount <= 0
    ? null
    : () {
        context.read<OrdersManager>().addOrder(
          cart.products,
          cart.totalAmount,
        );
        cart.clear();
      },
)

```

```

        style: TextButton.styleFrom(
          textStyle:
            TextStyle(color: Theme.of(context).primaryColor),
        ),
        child: const Text('ORDER NOW'),
      ),
    ...

```

- Hiệu chỉnh widget **OrdersScreen** (*lib/ui/orders/orders_screen.dart*):

```

...
@override
Widget build(BuildContext context) {
  print('building orders');
  return Scaffold(
    appBar: AppBar(
      title: const Text('Your Orders'),
    ),
    drawer: const AppDrawer(),
    body: Consumer<OrdersManager>(
      builder: (ctx, ordersManager, child) {
        return ListView.builder(
          itemCount: ordersManager.orderCount,
          itemBuilder: (ctx, i) =>
OrderItemCard(ordersManager.orders[i]),
        );
      },
    ),
  );
}
...

```

- Kiểm tra các chức năng liên quan đến đặt hàng hoạt động đúng. Sau đó, lưu thay đổi vào git và GitHub.

```

git add -u
git commit -m "Quan ly trang thai voi provider - OrdersManager"
git push origin master

```

Bước 5: Xây dựng trang thêm/cập nhật sản phẩm

9:59

←

Edit Product

Title

Red Shirt

Price

29.99

Description

A red shirt - it's pretty red!




Image URL

2/17/red-t-shirt-1710578_1280.jpg

- Định nghĩa trang thêm/cập nhật sản phẩm (*lib/ui/products/edit_product_screen.dart*):

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

import '../models/product.dart';
import '../shared/dialog_utils.dart';

import 'products_manager.dart';

class EditProductScreen extends StatefulWidget {
  static const routeName = '/edit-product';
```

```

EditProductScreen(
    Product? product, {
        super.key,
    }) {
    if (product == null) {
        this.product = Product(
            id: null,
            title: '',
            price: 0,
            description: '',
            imageUrl: '',
        );
    } else {
        this.product = product;
    }
}

late final Product product;

@override
State<EditProductScreen> createState() => _EditProductScreenState();
}

class _EditProductScreenState extends State<EditProductScreen> {
    final _imageUrlController = TextEditingController();
    final _imageUrlFocusNode = FocusNode();
    final _editForm = GlobalKey<FormState>();
    late Product _editedProduct;
    var _isLoading = false;

    bool _isValidImageUrl(String value) {

    }

    @override
    void initState() {

    }

    @override
    void dispose() {

    }

    Future<void> _saveForm() async {

    }

    @override
    Widget build(BuildContext context) {

```



```

    }
}

```

- Phương thức **init()**, **dispose()** lần lượt khởi tạo và hủy/giải phóng các biến. Biến **_imageUrlFocusNode** dùng để lúc nghe trạng thái focus của trường nhập liệu cho ảnh. Nếu dữ liệu nhập liệu là một URL ảnh hợp lệ thì yêu cầu vẽ lại màn hình để hiện ảnh xem trước.

```

bool _isValidImageUrl(String value) {
    return (value.startsWith('http') || value.startsWith('https')) &&
        (value.endsWith('.png') ||
            value.endsWith('.jpg') ||
            value.endsWith('.jpeg'));
}

@override
void initState() {
    _imageUrlFocusNode.addListener(() {
        if (!_imageUrlFocusNode.hasFocus) {
            if (!_isValidImageUrl(_imageUrlController.text)) {
                return;
            }
            // Ảnh hợp lệ → Vẽ lại màn hình để hiện preview
            setState(() {});
        }
    });
    _editedProduct = widget.product;
    _imageUrlController.text = _editedProduct.imageUrl;
    super.initState();
}

@override
void dispose() {
    _imageUrlController.dispose();
    _imageUrlFocusNode.dispose();
    super.dispose();
}

```

- Phương thức **build** xây dựng biểu mẫu hiệu chỉnh sản phẩm:

```

@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
            title: const Text('Edit Product'),
            actions: <Widget>[
                IconButton(
                    icon: const Icon(Icons.save),
                    onPressed: _saveForm,

```

```

    ),
  ],
),
body: _isLoading
  ? const Center(
    child: CircularProgressIndicator(),
  )
  : Padding(
    padding: const EdgeInsets.all(16.0),
    child: Form(
      key: _editForm,
      child: ListView(
        children: <widget>[
          buildTitleField(),
          buildPriceField(),
          buildDescriptionField(),
          buildProductPreview(),
        ],
      ),
    ),
  ),
);
}

```

- Các trường nhập liệu cho tên, giá, miêu tả, URL ảnh sản phẩm được định nghĩa như sau:
 - Hàm *buildTitleField()*:

```

TextFormField buildTitleField() {
  return TextFormField(
    initialValue: _editedProduct.title,
    decoration: const InputDecoration(labelText: 'Title'),
    textInputAction: TextInputAction.next,
    autofocus: true,
    validator: (value) {
      if (value!.isEmpty) {
        return 'Please provide a value.';
      }
      return null;
    },
    onSave: (value) {
      _editedProduct = _editedProduct.copyWith(title: value);
    },
  );
}

```

- Hàm *buildPriceField()*:

```

TextFormField buildPriceField() {
  return TextFormField(
    initialValue: _editedProduct.price.toString(),
    decoration: const InputDecoration(labelText: 'Price'),
    textInputAction: TextInputAction.next,
    keyboardType: TextInputType.number,
    validator: (value) {
      if (value!.isEmpty) {
        return 'Please enter a price.';
      }
      if (double.tryParse(value) == null) {
        return 'Please enter a valid number.';
      }
      if (double.parse(value) ≤ 0) {
        return 'Please enter a number greater than zero.';
      }
      return null;
    },
    onSaved: (value) {
      _editedProduct = _editedProduct.copyWith(price: double.parse(value!));
    },
  );
}

```

- Hàm *buildDescriptionField()*:

```

TextFormField buildDescriptionField() {
  return TextFormField(
    initialValue: _editedProduct.description,
    decoration: const InputDecoration(labelText: 'Description'),
    maxLines: 3,
    keyboardType: TextInputType.multiline,
    validator: (value) {
      if (value!.isEmpty) {
        return 'Please enter a description.';
      }
      if (value.length < 10) {
        return 'Should be at least 10 characters long.';
      }
      return null;
    },
    onSaved: (value) {
      _editedProduct = _editedProduct.copyWith(description: value);
    },
  );
}

```

- Hàm *buildProductPreview()*:

```

Widget buildProductPreview() {
  return Row(
    crossAxisAlignment: CrossAxisAlignment.end,
    children: <Widget>[
      Container(
        width: 100,
        height: 100,
        margin: const EdgeInsets.only(
          top: 8,
          right: 10,
        ),
        decoration: BoxDecoration(
          border: Border.all(
            width: 1,
            color: Colors.grey,
          ),
        ),
        child: _imageUrlController.text.isEmpty
          ? const Text('Enter a URL')
          : FittedBox(
              child: Image.network(
                _imageUrlController.text,
                fit: BoxFit.cover,
              ),
            ),
      ),
      Expanded(
        child: buildImageURLField(),
      ),
    ],
  );
}

```

- Hàm *buildImageURLField()*:

```

TextFormField buildImageUrlField() {
  return TextFormField(
    decoration: const InputDecoration(labelText: 'Image URL'),
    keyboardType: TextInputType.url,
    textInputAction: TextInputAction.done,
    controller: _imageUrlController,
    focusNode: _imageUrlFocusNode,
    onFieldSubmitted: (value) => _saveForm(),
    validator: (value) {
      if (value!.isEmpty) {
        return 'Please enter an image URL.';
      }
      if (!_isValidImageUrl(value)) {
        return 'Please enter a valid image URL.';
      }
      return null;
    },
    onSave: (value) {
      _editedProduct = _editedProduct.copyWith(imageUrl: value);
    },
  );
}

```

- Phương thức **_saveForm** thực hiện thêm sản phẩm vào danh sách sản phẩm quản lý bởi **ProductsManager**:

```

Future<void> _saveForm() async {
  final isValid = _editForm.currentState!.validate();
  if (!isValid) {
    return;
  }
  _editForm.currentState!.save();

  setState(() {
    _isLoading = true;
  });

  try {
    final productsManager = context.read<ProductsManager>();
    if (_editedProduct.id != null) {
      productsManager.updateProduct(_editedProduct);
    } else {
      productsManager.addProduct(_editedProduct);
    }
  } catch (error) {
    await showErrorDialog(context, 'Something went wrong.');
```

```

    setState(() {
      _isLoading = false;
    });
  }
}
}

```

```

    if (mounted) {
      Navigator.of(context).pop();
    }
  }
}

```

`_editForm.currentState!.validate()` sẽ lần lượt gọi `validator` của mỗi `FormField` trong `Form`. Tương tự, `_editForm.currentState!.save()` sẽ lần lượt gọi `onSaved` của mỗi `FormField`.

Với hàm **`showErrorDialog()`** được định nghĩa như sau (*lib/ui/shared/dialog_utils.dart*):

```

Future<void> showErrorDialog(BuildContext context, String message) {
  return showDialog(
    context: context,
    builder: (ctx) => AlertDialog(
      title: const Text('An Error Occurred!'),
      content: Text(message),
      actions: <Widget>[
        TextButton(
          child: const Text('okay'),

```

```

        onPressed: () {
          Navigator.of(ctx).pop();
        },
      ),
    ],
  ),
);
}

```

- Hiệu chỉnh *lib/main.dart*, khai báo thêm trang mới (thêm `export` `'products/edit_product_screen.dart'`; vào *lib/screens.dart*):

```

...
onGenerateRoute: (settings) {
  ...
  if (settings.name == EditProductScreen.routeName) {
    final productId = settings.arguments as String?;
    return MaterialPageRoute(
      builder: (ctx) {
        return EditProductScreen(
          productId != null
            ? ctx.read<ProductsManager>().findById(productId)
            : null,
        );
      },
    );
  }

  return null;
}
...

```

- Lần lượt hiệu chỉnh widget **UserProductListTile** và **UserProductsScreen** liên kết đến trang cập nhật sản phẩm (tự thực hiện các import cần thiết).
 - Hiệu chỉnh widget **UserProductListTile** (*lib/ui/products/user_product_list_tile.dart*):

```

...
Widget buildEditButton(BuildContext context) {
  return IconButton(
    icon: const Icon(Icons.edit),
    onPressed: () {
      Navigator.of(context).pushNamed(
        EditProductScreen.routeName,
        arguments: product.id,
      );
    },
    color: Theme.of(context).primaryColor,
  );
}
...

```

- Hiệu chỉnh widget **UserProductsScreen** (*lib/ui/products/user_products_screen.dart*):

```
...  
Widget buildAddButton(BuildContext context) {  
  return IconButton(  
    icon: const Icon(Icons.add),  
    onPressed: () {  
      Navigator.of(context).pushNamed(  
        EditProductScreen.routeName,  
      );  
    },  
  );  
}
```

- Kiểm tra chức năng thêm sản phẩm hoạt động đúng (comment danh sách sản phẩm gán cứng trong **ProductsManager** và dùng trang thêm sản phẩm vừa tạo lần lượt thêm các sản phẩm đã comment lại vào). Sau đó, lưu thay đổi vào git và GitHub.

```
git add -u  
git add lib/ui/products  
git commit -m "Xây dựng trang hiệu chỉnh sản phẩm"  
git push origin master
```

- Cấu trúc thư mục mã nguồn hiện tại:

- lib
 - models
 - cart_item.dart
 - order_item.dart
 - product.dart
 - ui
 - cart
 - cart_item_card.dart
 - cart_manager.dart
 - cart_screen.dart
 - orders
 - order_item_card.dart
 - orders_manager.dart
 - orders_screen.dart
 - products
 - edit_product_screen.dart
 - product_detail_screen.dart
 - product_grid_tile.dart
 - products_grid.dart
 - products_manager.dart
 - products_overview_screen.dart
 - top_right_badge.dart
 - user_product_list_tile.dart
 - user_products_screen.dart
 - shared
 - screens.dart
 - main.dart