

CT484: PHÁT TRIỂN ỨNG DỤNG DI ĐỘNG

XÂY DỰNG ỨNG DỤNG MYSHOP - PHẦN 4

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 3)

File báo cáo buổi 4 làm tiếp tục từ file báo cáo buổi 3.

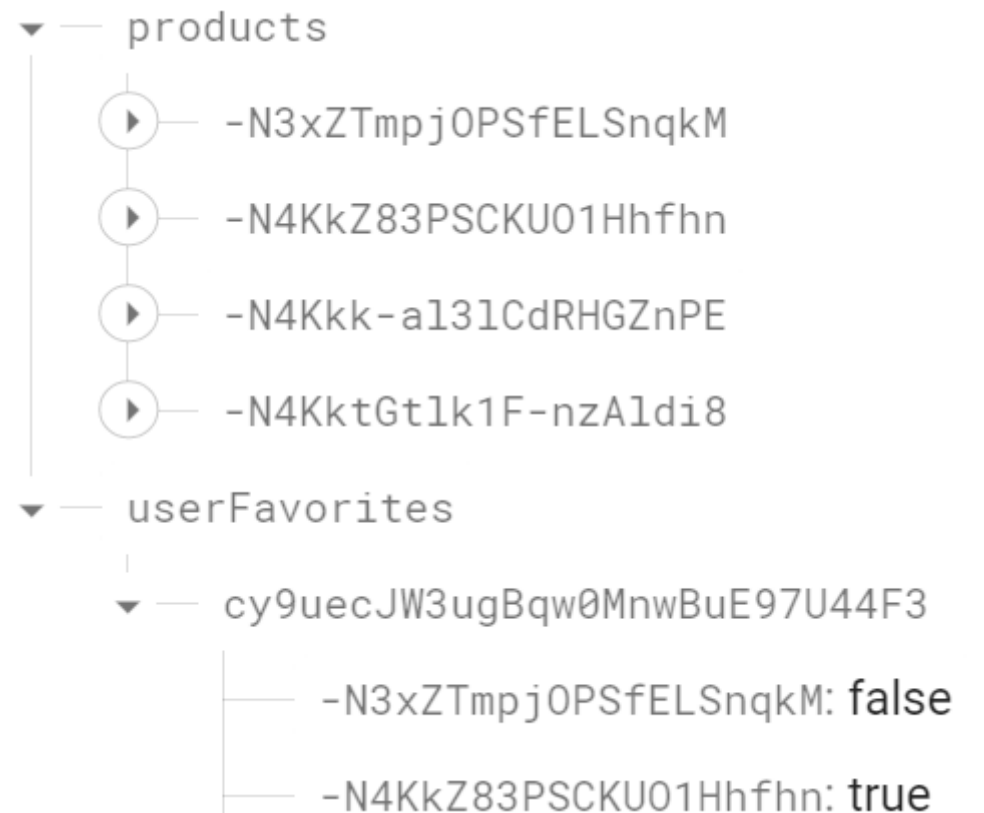
Bước 0: Tìm hiểu về Firebase

Xem video hướng dẫn về Firebase tại đây: <https://www.youtube.com/watch?v=robC2XADUkQ>

Firebase là một tập hợp các dịch vụ lưu trữ (hosting service) cho nhiều dạng ứng dụng khác nhau (Android, iOS, Node.js, PHP, ...). Firebase hỗ trợ dịch vụ lưu trữ cơ sở dữ liệu (CSDL) thời gian thực, NoSQL, lưu trữ tập tin, function như một dịch vụ (function as a service), chứng thực (authentication), thông báo (notification), ... Dự án này chỉ sử dụng dịch vụ CSDL (Firebase Database) và dịch vụ chứng thực (Firebase Authentication). Các dịch vụ này đều có [hạn mức dùng miễn phí](#).

Firebase Database

Có hai tùy chọn cho CSDL trên Firebase: [Realtime Database](#) và [Cloud Firestore](#). Realtime Database là dạng CSDL nguyên thủy của Firebase trong khi Cloud Firestore là phiên bản mới hơn. Nội dung bài thực hành này trình bày cách sử dụng Realtime Database. Mô hình lưu trữ Realtime Database khá đơn giản: CSDL được tổ chức như một **cây JSON**. Dữ liệu thêm vào cây JSON này sẽ là một nhánh trong cây. Tên nhánh trong cây có thể được chỉ định tường minh hoặc tự động sinh:



Có thể sử dụng [thư viện client Firebase dành cho Flutter](#) hoặc dùng [REST API](#) để tương tác với Realtime Database. Để đơn giản, REST API sẽ được sử dụng. Các URI tham chiếu đến các đối tượng trong cây JSON được tạo ra như sau: *đính kèm đuôi ".json" vào đường dẫn đến đối tượng cần truy xuất*, ví dụ:

- **GET /products.json:** đọc tất cả các đối tượng từ nhánh products
- **GET /products.json?orderBy="creatorId"&equalTo="cy9uecJW3ugBqw0MnwBuE97U44F3":** đọc tất cả các đối tượng từ nhánh products có trường creatorId = cy9uecJW3ugBqw0MnwBuE97U44F3 (cần tạo chỉ mục cho trường creatorId, xem cấu hình rule bên dưới)
- **POST /products.json:** thêm một đối tượng mới vào nhánh products. Tên nhánh ứng với đối tượng mới sẽ được tự động sinh và trả về trong câu trả lời
- **GET /products/N4KktGtlk1F-nzAldi8.json:** đọc thông tin sản phẩm có định danh N4KktGtlk1F-nzAldi8 (nhánh /products/N4KktGtlk1F-nzAldi8). Nếu nhánh này có cấu trúc như sau:

```

-N4KktGtlk1F-nzAldi8
├── creatorId: "cy9uecJW3ugBqw0MnwBuE97U44F3"
├── description: "Prepare any meal you want."
├── imageUrl: "https://upload.wikimedia.org/wikipedia/commons/thumb/1/14/Cast-Iron-Pan.jpg/1024px-Cast-Iron-Pan.jpg"
├── price: 49.99
└── title: "A Pan"
```

thì dữ liệu trả về sẽ ở dạng:

```
{
  "creatorId": "cy9uecJW3ugBqw0MnwBuE97U44F3",
  "description": "Prepare any meal you want.",
  "imageUrl": "https://upload.wikimedia.org/.../1024px-Cast-Iron-Pan.jpg",
  "price": 49.99,
  "title": "A Pan"
}
```

- **GET /userFavorites/cy9uecJW3ugBqw0MnwBuE97U44F3.json:** đọc danh mục các sản phẩm yêu thích của người dùng có định danh cy9uecJW3ugBqw0MnwBuE97U44F3. Theo như hình ví dụ phía trên, dữ liệu trả về sẽ có dạng như sau:

```
{
  "-N3xZTmpjOPSfELSnqkM": false,
  "-N4KkZ83PCKU01Hhfhn": true
}
```

Để đảm bảo an toàn/bảo mật, cần cấu hình [các luật truy xuất](#) trên Realtime Database (ví dụ chỉ cho phép những yêu cầu cập nhật từ những người dùng được chứng thực).

Firestore Authentication

Firestore Authentication cung cấp các dịch vụ backend, [SDK để sử dụng](#) (cũng có hỗ trợ [REST API](#)), các thư viện UI sẵn dùng để giúp chứng thực người dùng cho ứng dụng. Nhiều cơ chế chứng thực được hỗ trợ: chứng thực dựa trên mật khẩu, số điện thoại, mạng xã hội như Google, Facebook, Twitter, ...

Chúng ta sẽ sử dụng chứng thực với email và mật khẩu, cụ thể sẽ sử dụng hai lời gọi REST sau đây:

- [Đăng ký người dùng mới:](#)
 - Phương thức yêu cầu: POST
 - Endpoint: `https://identitytoolkit.googleapis.com/v1/accounts:signup?key=[API_KEY]`, với API_KEY là "Web API Key" lấy từ dự án firebase
 - Dữ liệu JSON gửi đi:

```
{
  "email": "[user@example.com]",
  "password": "[PASSWORD]",
  "returnSecureToken": true
}
```

- Dữ liệu JSON trả lời (thành công):

```
{
  "idToken": "[ID_TOKEN]",
  "email": "[user@example.com]",
  "refreshToken": "[REFRESH_TOKEN]",
  "expiresIn": "3600",
  "localId": "tRcfmLH7..."
}
```

- [Đăng nhập người dùng](#)

- Phương thức yêu cầu: POST

- Endpoint:

`https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=[API_KEY]`, với API_KEY là "Web API Key" lấy từ dự án firebase

- Dữ liệu JSON gửi đi:

```
{
  "email": "[user@example.com]",
  "password": "[PASSWORD]",
  "returnSecureToken": true
}
```

- Dữ liệu JSON trả lời (thành công):

```
{
  "localId": "ZY1rJK0eYLg...",
  "email": "[user@example.com]",
  "displayName": "",
  "idToken": "[ID_TOKEN]",
  "registered": true,
  "refreshToken": "[REFRESH_TOKEN]",
  "expiresIn": "3600"
}
```

Khi người dùng hoặc thiết bị đăng nhập bằng Firebase Authentication, Firebase sẽ tạo một ID token định danh người dùng (**idToken**) và nhờ đó có thể truy cập vào một số tài nguyên, chẳng hạn như Realtime Database và Cloud Firestore, i.e., có thể sử dụng ID token đó để [xác thực REST API](#) của Realtime Database. (Tuy nhiên, gán quyền truy xuất dữ liệu cụ thể phải thông qua cấu hình các luật truy xuất).

Gửi ID token như chuỗi truy vấn trên URI đến REST API của Realtime Database, ví dụ: **GET /products.json?auth=<ID_TOKEN>**.

Để hỗ trợ các trường hợp chứng thực nâng cao hơn ví dụ như *điều khiển truy cập dựa trên vai trò người dùng*, có thể tham khảo giải pháp tạo [token tùy biến](#) hoặc [gán thông tin tùy biến cho tài khoản người dùng](#). Các giải pháp này yêu cầu có thêm xử lý phía backend (tự tạo hoặc dùng Firebase Functions) để có thể cài đặt được. Một giải pháp không yêu cầu xử lý backend (dùng trực tiếp Firebase Console) là tạo nhánh cây lưu thông tin người dùng, trong đó có chỉ định vai trò của người dùng và định nghĩa các luật truy cập tài nguyên dựa trên thông tin vai trò đó, ví dụ:

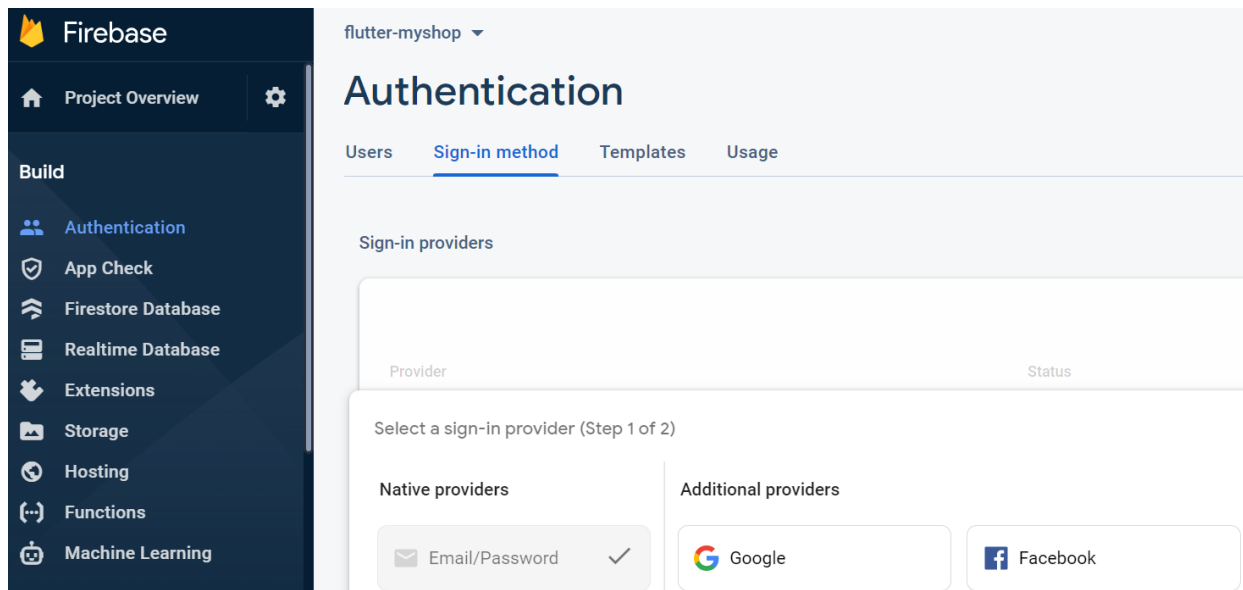
```
{
  "rules": {
    ".read": true,
    "users": {
      ".write": "root.child('users').child(auth.uid).child('role').val() == 'admin'"
    }
  }
}
```

Nhánh cây users có dạng như sau:

```
users
|
--- cy9uecJW3ugBqw0MnwBuE97U44F3
    |
    ---role: "admin"
```

Bước 1: Thiết lập dự án trên Firebase

- Tạo dự án trên Firebase (<https://console.firebase.google.com/>), ví dụ tên là *flutter-myshop* (trong quá trình tạo dự án, ngoài tên dự án có thể chỉ định ID của dự án hoặc để tự động sinh).
- Bật chế độ chứng thực người dùng dạng email/password trên Firebase Authentication:



- Trong thư mục gốc của dự án, tạo tập tin *.env* và lưu hai biến sau:

```
FIREBASE_API_KEY=<Web API key từ dự án Firebase>
FIREBASE_URL=<URL đến CSDL Realtime trên Firebase>
```

trong đó FIREBASE_API_KEY được lấy từ Project settings:

The screenshot shows the 'Project settings' page in the Firebase console for a project named 'flutter-myshop'. The left sidebar contains navigation links for Project Overview, Build, Authentication, App Check, Firestore Database, Realtime Database, Extensions, Storage, Hosting, Functions, and Machine Learning. The main content area has tabs for General, Cloud Messaging, Integrations, Service accounts, Data privacy, and Users and permissions. The 'General' tab is active, displaying project details: Project name (flutter-myshop), Project ID (flutter-myshop-9393b), Project number (774061739027), Default GCP resource location (nam5 (us-central)), and Web API Key (redacted with a red bar).

FIREBASE_URL lấy từ Realtime Database:

The screenshot shows the 'Realtime Database' page in the Firebase console for the same project. The left sidebar is identical to the previous screenshot, but the 'Realtime Database' option is highlighted. The main content area has tabs for Data, Rules, Backups, and Usage. The 'Data' tab is active, showing the database URL: `https://flutter-myshop-9393b-default-rtdb.asia-southeast1.firebaseio.com`. Below the URL, there is a text input field containing the same URL.

- Chuyển qua thẻ "Rules" và thêm vào [các luật truy cập dữ liệu](#) như sau (chọn "Publish" để lưu):

The screenshot shows the 'Rules' tab in the Firebase console. It features a 'Data' tab, a 'Rules' tab (which is active), a 'Backups' tab, and a 'Usage' tab. Below the tabs, there are buttons for 'Edit rules' and 'Monitor rules'. A 'Rules playground' button is also visible. The main area contains a code editor with the following JSON rules:

```
1 {
2   "rules": {
3     ".read": "auth.uid != null",
4     "products": {
5       ".indexOn": "creatorId",
6       "$productId": {
7         ".validate": "newData.hasChildren(['creatorId', 'title', 'description', 'imageUrl', 'price'])",
8         ".write": "!data.exists() || data.child('creatorId').val() == auth.uid"
9       }
10    },
11    "userFavorites": {
12      "$uid": {
13        ".write": "$uid == auth.uid"
14      }
15    }
16  }
17 }
```

Các luật trên có ý nghĩa như sau: tất cả các người dùng hợp lệ (`auth.uid != null`) đều có thể đọc dữ liệu. Trên nhánh `/products` (lưu danh mục các sản phẩm), lập chỉ mục cho trường `creatorId`. Một sản phẩm lưu trong nhánh `/products` phải có các nút được chỉ định (`.validate`). Các sản phẩm cũng chỉ có thể được cập nhật hay xóa bởi người dùng tạo chúng (`creatorId == auth.uid`). Trên nhánh

/userFavorites (lưu danh mục sản phẩm yêu thích theo từng người dùng), người dùng chỉ được hiệu chỉnh danh mục yêu thích của riêng mình (\$uid == auth.uid).

Bước 2: Cài đặt chứng thực dùng với Firebase Authentication

- Khai báo thư viện *flutter_dotenv*, *http*, *shared_preferences* và tập tin *.env* trong *pubspec.yaml*:

```
dependencies:
  #...
  flutter_dotenv: ^5.0.2
  http: ^0.13.4
  shared_preferences: ^2.0.15
  #...
flutter:
  uses-material-design: true
  assets:
    - .env
  #...
```

- Vào thư mục **auth** đã cho, sao chép 3 thư mục trong đó vào thư mục lib của dự án. Hiệu chỉnh *lib/ui/screens.dart*:

```
...
export 'auth/auth_screen.dart';
export 'auth/auth_manager.dart';

export 'splash_screen.dart';
```

- Hiệu chỉnh *lib/main.dart* thực hiện các tác vụ sau: (1) đọc tập tin *.env*, (2) tạo và cung cấp đối tượng **AuthManager** và (3) tiêu thụ/sử dụng đối tượng *AuthManager* để kiểm tra trạng thái đăng nhập của người dùng và trình bày giao diện tương ứng.

```

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

import 'ui/screens.dart';

Future<void> main() async {
  // (1) Load the .env file
  await dotenv.load();
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        // (2) Create and provide AuthManager
        ChangeNotifierProvider(
          create: (context) => AuthManager(),
        ),
        ...
      ],
      // (3) Consume the AuthManager instance
      child: Consumer<AuthManager>({
        builder: (ctx, authManager, child) {
          return MaterialApp(
            ...
            home: authManager.isAuth
              ? const ProductsOverviewScreen()
              : FutureBuilder(
                  future: authManager.tryAutoLogin(),
                  builder: (ctx, snapshot) {
                    return snapshot.connectionState == ConnectionState.waiting
                      ? const SplashScreen()
                      : const AuthScreen();
                  },
                ),
            ...
          );
        },
      ),
    );
  }
}

```

- Hiệu chỉnh widget **AppDrawer** (*lib/ui/shared/app_drawer.dart*), thêm đường liên kết đăng xuất:


```

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

import '../auth/auth_manager.dart';
import '../orders/orders_screen.dart';
import '../products/user_products_screen.dart';

class AppDrawer extends StatelessWidget {
  const AppDrawer({super.key});

  @override
  Widget build(BuildContext context) {
    return Drawer(
      child: Column(
        children: <Widget>[
          ...
          const Divider(),
          ListTile(
            leading: const Icon(Icons.exit_to_app),
            title: const Text('Logout'),
            onTap: () {
              Navigator.of(context)
                ..pop()
                ..pushReplacementNamed('/');
              context.read<AuthManager>().logout();
            },
          ),
        ],
      ),
    );
  }
}

```

- **Lưu ý:** nếu sử dụng **thiết bị Android thật** để chạy ứng dụng thì cần chỉnh quyền truy xuất Internet bằng cách hiệu chỉnh tập tin **android/app/src/main/AndroidManifest.xml**, thêm dòng khai báo sau:

```

<manifest ...>
  <application>
    ...
  </application>

  <uses-permission android:name="android.permission.INTERNET" />
</manifest>

```

Nếu dùng thiết bị giả lập thì không cần.

- Chạy ứng dụng và kiểm tra chức năng đăng ký, đăng nhập, đăng xuất. Sau đó, lưu thay đổi vào git và GitHub:

```
git add -u
git add lib
git commit -m "Cai dat chung thuc voi Firebase Authentication"
git push origin master
```

Bước 3: Lưu danh mục sản phẩm lên dịch vụ CSDL của Firebase

- Hiệu chỉnh lớp **Product** thêm định nghĩa hai phương thức **toJson()** và **fromJson()** giúp chuyển đổi qua lại giữa một đối tượng Product và chuỗi JSON.

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

class Product {
  ...
  Map<String, dynamic> toJson() {
    return {
      'title': title,
      'description': description,
      'price': price,
      'imageUrl': imageUrl,
    };
  }

  static Product fromJson(Map<String, dynamic> json) {
    return Product(
      id: json['id'],
      title: json['title'],
      description: json['description'],
      price: json['price'],
      imageUrl: json['imageUrl'],
    );
  }
}
```

- Định nghĩa lớp trừu tượng **FirestoreService** lưu giữ những tham số cần thiết để thực hiện các lời gọi REST API (*lib/services/firebase_service*):

```

1  import 'package:flutter/foundation.dart';
2  import 'package:flutter_dotenv/flutter_dotenv.dart';
3
4  import '../models/auth_token.dart';
5
6  abstract class FirebaseService {
7    String? _token;
8    String? _userId;
9    late final String? databaseUrl;
10
11    FirebaseService([AuthToken? authToken])
12      : _token = authToken?.token,
13        _userId = authToken?.userId {
14      databaseUrl = dotenv.env['FIREBASE_URL'];
15    }
16
17    set authToken(AuthToken? authToken) {
18      _token = authToken?.token;
19      _userId = authToken?.userId;
20    }
21
22    @protected
23    String? get token ⇒ _token;
24
25    @protected
26    String? get userId ⇒ _userId;
27  }

```

- Định nghĩa lớp **ProductsService** chịu trách nhiệm thực hiện các lời gọi REST API đến Firebase để truy xuất và cập nhật dữ liệu (*lib/services/products_service.dart*):
 - Phương thức **fetchProducts()**:

```

import 'dart:convert';

import 'package:http/http.dart' as http;

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

import 'firebase_service.dart';

class ProductsService extends FirebaseService {
  ProductsService([AuthToken? authToken]) : super(authToken);

  Future<List<Product>> fetchProducts([bool filterByUser = false]) async {
    final List<Product> products = [];

    try {
      final filters =
        filterByUser ? 'orderBy="creatorId"&equalTo="$userId" : '' ;
      final productsUrl =
        Uri.parse('$databaseUrl/products.json?auth=$token&$filters');
      final response = await http.get(productsUrl);
      final productsMap = json.decode(response.body) as Map<String, dynamic>;

      if (response.statusCode != 200) {
        print(productsMap['error']);
        return products;
      }

      final userFavoritesUrl =
        Uri.parse('$databaseUrl/userFavorites/$userId.json?auth=$token');
      final userFavoritesResponse = await http.get(userFavoritesUrl);
      final userFavoritesMap = json.decode(userFavoritesResponse.body);

      productsMap.forEach((productId, product) {
        final isFavorite = (userFavoritesMap == null)
          ? false
          : (userFavoritesMap[productId] ?? false);
        products.add(
          Product.fromJson({
            'id': productId,
            ...product,
          }).copyWith(isFavorite: isFavorite),
        );
      });
      return products;
    } catch (error) {
      print(error);
      return products;
    }
  }
}

```

- Phương thức **addProduct()**:

```

...
class ProductsService extends FirebaseService {
  ...
  Future<Product?> addProduct(Product product) async {
    try {
      final url = Uri.parse('$databaseUrl/products.json?auth=$token');
      final response = await http.post(
        url,
        body: json.encode(
          product.toJson()
            ..addAll({
              'creatorId': userId,
            })),
      );

      if (response.statusCode != 200) {
        throw Exception(json.decode(response.body)['error']);
      }

      return product.copyWith(
        id: json.decode(response.body)['name'],
      );
    } catch (error) {
      print(error);
      return null;
    }
  }
}

```

- Hiệu chỉnh lớp **ProductsManager** sử dụng **ProductsService** để tải danh mục sản phẩm và thêm sản phẩm vào CSDL (*lib/ui/products/products_manager.dart*):

```

import 'package:flutter/foundation.dart';

import '../models/auth_token.dart';
import '../models/product.dart';
import '../services/products_service.dart';

class ProductsManager with ChangeNotifier {
  List<Product> _items = [];

  final ProductsService _productsService;

  ProductsManager([AuthToken? authToken])
    : _productsService = ProductsService(authToken);

  set authToken(AuthToken? authToken) {
    _productsService.authToken = authToken;
  }

  Future<void> fetchProducts([bool filterByUser = false]) async {
    _items = await _productsService.fetchProducts(filterByUser);
    notifyListeners();
  }

  Future<void> addProduct(Product product) async {
    final newProduct = await _productsService.addProduct(product);
    if (newProduct != null) {
      _items.add(newProduct);
      notifyListeners();
    }
  }
  ...
}

```

- Hiệu chỉnh widget **ProductsOverviewScreen** thực hiện tải danh mục sản phẩm khi khởi tạo. Trong quá trình tải dữ liệu thì hiển thị thanh tiến trình chờ (*lib/ui/products/products_overview_screen.dart*):

```

...
class _ProductsOverviewScreenState extends State<ProductsOverviewScreen> {
  final _showOnlyFavorites = ValueNotifier<bool>(false);
  late Future<void> _fetchProducts;

  @override
  void initState() {
    super.initState();
    _fetchProducts = context.read<ProductsManager>().fetchProducts();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      ...
      body: FutureBuilder(
        future: _fetchProducts,
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.done) {
            return ValueListenableBuilder<bool>(
              valueListenable: _showOnlyFavorites,
              builder: (context, onlyFavorites, child) {
                return ProductsGrid(onlyFavorites);
              });
          }
          return const Center(
            child: CircularProgressIndicator(),
          );
        },
      ),
    );
  }
  ...
  Widget buildProductFilterMenu() {
    return PopupMenuButton(
      onSelected: (FilterOptions selectedValue) {
        if (selectedValue == FilterOptions.favorites) {
          _showOnlyFavorites.value = true;
        } else {
          _showOnlyFavorites.value = false;
        }
      },
      ...
    );
  }
}

```

- Hiệu chỉnh widget **UserProductsScreen** thực hiện tải danh mục sản phẩm khi khởi tạo. Trong quá trình tải dữ liệu thì hiển thị thanh tiến trình chờ (*lib/ui/products/user_products_screen.dart*):

```

...
class UserProductsScreen extends StatelessWidget {
  ...
  Future<void> _refreshProducts(BuildContext context) async {
    await context.read<ProductsManager>().fetchProducts(true);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      ...
      body: FutureBuilder(
        future: _refreshProducts(context),
        builder: (ctx, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return const Center(
              child: CircularProgressIndicator(),
            );
          }
          return RefreshIndicator(
            onRefresh: () => _refreshProducts(context),
            child: buildUserProductListView(),
          );
        },
      ),
    );
  }
}

```

- Hiệu chỉnh *lib/main.dart* cập nhật authToken của **ProductsManager** dựa trên giá trị authToken của **AuthManager**. Thay thế:

```

ChangeNotifierProvider(
  create: (ctx) => ProductsManager(),
)

```

bằng:

```

ChangeNotifierProxyProvider<AuthManager, ProductsManager>(
  create: (ctx) => ProductsManager(),
  update: (ctx, authManager, productsManager) {
    // khi authManager có báo hiệu thay đổi thì đọc lại authToken
    // cho productManager
    productsManager!.authToken = authManager.authToken;
    return productsManager;
  },
)

```


- Chạy kiểm tra thêm các sản phẩm. Sau đó, lưu các thay đổi vào git và GitHub:

```
git add -u
git add lib/services
git commit -m "Lưu danh mục sản phẩm vào Realtime Database của Firebase"
git push origin master
```

Bước 4: Cập nhật, xóa sản phẩm lưu trên CSDL của Firebase

- Hiệu chỉnh **ProductsService** (*lib/services/products_service.dart*), thêm hai phương thức để cập nhật và xóa sản phẩm lưu trên CSDL Realtime của Firebase:

```
...
class ProductsService extends FirebaseService {
  ...
  Future<bool> updateProduct(Product product) async {
    try {
      final url =
        Uri.parse('$databaseUrl/products/${product.id}.json?auth=$token');
      final response = await http.patch(
        url,
        body: json.encode(product.toJson()),
      );

      if (response.statusCode != 200) {
        throw Exception(json.decode(response.body)['error']);
      }

      return true;
    } catch (error) {
      print(error);
      return false;
    }
  }

  Future<bool> deleteProduct(String id) async {
    try {
      final url = Uri.parse('$databaseUrl/products/$id.json?auth=$token');
      final response = await http.delete(url);

      if (response.statusCode != 200) {
        throw Exception(json.decode(response.body)['error']);
      }

      return true;
    } catch (error) {
      print(error);
      return false;
    }
  }
}
```

- Hiệu chỉnh **ProductsManager** (*lib/ui/products/products_manager.dart*) cập nhật phương thức *updateProduct* và *deleteProduct*:

```
...
class ProductsManager with ChangeNotifier {
  ...
  Future<void> updateProduct(Product product) async {
    final index = _items.indexWhere((item) => item.id == product.id);
    if (index >= 0) {
      if (await _productsService.updateProduct(product)) {
        _items[index] = product;
        notifyListeners();
      }
    }
  }

  Future<void> deleteProduct(String id) async {
    final index = _items.indexWhere((item) => item.id == id);
    Product? existingProduct = _items[index];
    _items.removeAt(index);
    notifyListeners();

    if (!await _productsService.deleteProduct(id)) {
      _items.insert(index, existingProduct);
      notifyListeners();
    }
  }
}
```

- Hiệu chỉnh **_saveForm** trong lớp **_EditProductScreenState** (*lib/ui/products/edit_product_screen.dart*):

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

- Chạy kiểm tra cập nhật và xóa sản phẩm. Sau đó, lưu các thay đổi vào git và GitHub:

```
git add -u
git commit -m "Cap nhat, xoa san pham tren Realtime Database cua Firebase"
git push origin master
```

Bước 5: Lưu trữ sản phẩm yêu thích

- Hiệu chỉnh **ProductsService** (*lib/services/products_service.dart*) thêm phương thức lưu trữ sản phẩm yêu thích:

```
...
class ProductsService extends FirebaseService {
  ...
  Future<bool> saveFavoriteStatus(Product product) async {
    try {
      final url = Uri.parse(
        '$databaseUrl/userFavorites/$userId/${product.id}.json?auth=$token');
      final response = await http.put(
        url,
        body: json.encode(
          product.isFavorite,
        ),
      );

      if (response.statusCode != 200) {
        throw Exception(json.decode(response.body)['error']);
      }

      return true;
    } catch (error) {
      print(error);
      return false;
    }
  }
}
```

- Hiệu chỉnh **ProductsManager** (*lib/ui/products/products_manager.dart*) thêm phương thức thay đổi trạng thái yêu thích của sản phẩm:

```

...
class ProductsManager with ChangeNotifier {
  ...
  Future<void> toggleFavoriteStatus(Product product) async {
    final savedStatus = product.isFavorite;
    product.isFavorite = !savedStatus;

    if (!await _productsService.saveFavoriteStatus(product)) {
      product.isFavorite = savedStatus;
    }
  }
}

```

- Hiệu chỉnh widget **ProductListTile** (*lib/ui/products/product_grid_tile.dart*), thay thế:

```

onPressed: () {
  product.isFavorite = !isFavorite;
}

```

bằng:

```

onPressed: () {
  ctx.read<ProductsManager>().toggleFavoriteStatus(product);
}

```

(cần import 'products_manager.dart').

- Kiểm tra chức năng lưu trữ sản phẩm yêu thích. Sau đó lưu thay đổi vào git và GitHub:

```

git add -u
git commit -m "Luu san pham yeu thich len Realtime Database cua Firebase"
git push origin master

```

- Cấu trúc thư mục mã nguồn hiện tại:

```
lib
├── models
│   ├── auth_token.dart
│   ├── cart_item.dart
│   ├── http_exception.dart
│   ├── order_item.dart
│   └── product.dart
├── services
│   ├── auth_service.dart
│   ├── firebase_service.dart
│   └── products_service.dart
├── ui
│   ├── auth
│   │   ├── app_banner.dart
│   │   ├── auth_card.dart
│   │   ├── auth_manager.dart
│   │   └── auth_screen.dart
│   ├── 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
│       ├── app_drawer.dart
│       ├── dialog_utils.dart
│       ├── screens.dart
│       ├── splash_screen.dart
│       └── main.dart
```