**KHÁI NIỆM VỀ DATABASE**

Flutter cung cấp một vài gói nâng cao để làm việc với database (cơ sở dữ liệu). Nhưng 2 gói quan trọng nhất là :

1. **sqflite** - Sử dụng để truy vấn vào SQLlite database
2. **firebase\_database**: Sử dụng để truy vấn và vận dụng đám mây lưu trữ NoSQL database từ Google.

SQLite

SQLite là một SQL tiêu chuẩn dựa trên công cụ cơ sở dữ liệu nhúng . Nó là công cụ nhỏ và đang được thử nghiệm theo thời gian. Gói sqflite cung cấp nhiều chức năng để làm việc hiệu quả với SQLite database. Nó cung cấp các phương thức tiêu chuẩn để vận hành SQLite database. Chức năng chính của sqflite như sau:

1. Tạo/mở SQLite database
2. Thực thi SQL statement (thực thi phương thức) đối với SQLite database
3. Phương thức truy vấn nâng cao (phương thức truy vấn) để giảm code cần thiết để truy vấn và lấy thông tin từ SQLite database.

Bây giờ ta thử tạo ứng dụng cửa hàng sản phẩm điện thoại và lấy dữ liệu từ SQLite database sử dụng gói sqflite và hiểu các khải niệm về SQLite database và gói sqflite.

Tạo một ứng dụng mới có tên là : *product\_sqlite\_app*

Copy thư mục assets  bài cũ sang và thêm vào mục  \*pubspec.yaml`

flutter:

assets:

- assets/appimages/floppy.png

- assets/appimages/iphone.png

- assets/appimages/laptop.png

- assets/appimages/pendrive.png

- assets/appimages/pixel.png

- assets/appimages/tablet.png

Cấu hình sqflite trong file pubspec.yaml như dưới đây

dependencies: sqflite: any

Sử dụng phiên bản mới nhất của sqflite điền vào chỗ *any*

Android studio sẽ thông báo rằng pubspec.yaml cần được cập nhật rồi ta nhấn và Get dependencies. Android studio sẽ lấy package từ internet và cấu hình thuộc tính cho ứng dụng

Trong database(cơ sở dữ liệu), chúng ta cần primary key, id như trường bổ sung kèm với các thuộc tính của Product như : tên, giá, ... Vì thế, thêm thuộc tính id vào lớp Product. Ngoài ra, thêm 1 phương thức mới là toMap để chuyển đổi đối tượng product vào Map.

class Product {

final int id;

final String name;

final String description;

final int price;

final String image;

static final columns = ["id", "name", "description", "price", "image"];

Product(this.id, this.name, this.description, this.price, this.image);

factory Product.fromMap(Map<String, dynamic> data) {

return Product(

data['id'],

data['name'],

data['description'],

data['price'],

data['image'],

);

}

Map<String, dynamic> toMap() => {

"id": id,

"name": name,

"description": description,

"price": price,

"image": image

};

}

Tạo một file mới Database.dart để viết *SQLite*

Thêm một vài thư viện cần thiết trong Database.dart

import 'dart:async';

import 'dart:io';

import 'package:path/path.dart';

import 'package:path\_provider/path\_provider.dart';

import 'package:sqflite/sqflite.dart';

import 'Product.dart';

Ta cần chú ý một vài điểm sau :

1. **async**được sử dụng để viết phương thức asynchronous(không đồng bộ)
2. **io**được sử dụng để truy cập vào tệp và thư mục
3. **path** được sử dụng để truy cập vào tiện ích cốt lõi của dart liên quan đến đường dẫn tệp
4. **path\_provider**được sử dụng để lấy đường dẫn tạm thời
5. **sqflite**được sử dụng để vận hành database SQLite

Tạo một lớp mới có tên **SQLiteDbProvider**

class SQLiteDbProvider {

SQLiteDbProvider.\_();

static final SQLiteDbProvider db = SQLiteDbProvider.\_();

static Database \_database;

}

SQLiteDBProvoider là phương thức có thể truy cập thông qua biến đổi vùng nhớ db

SQLiteDBProvoider.db.<emthod>

Tạo hàm để lấy database (sử dụng Future) của kiểu Future<Database>. Tạo bảng product và tải dữ liệu ban đầu trong quá trình tạo database

Future<Database> get database async {

if (\_database != null)

return \_database;

\_database = await initDB();

return \_database;

}

initDB() async {

Directory documentsDirectory = await getApplicationDocumentsDirectory();

String path = join(documentsDirectory.path, "ProductDB.db");

return await openDatabase(

path,

version: 1,

onOpen: (db) {},

onCreate: (Database db, int version) async {

await db.execute(

"CREATE TABLE Product ("

"id INTEGER PRIMARY KEY,"

"name TEXT,"

"description TEXT,"

"price INTEGER,"

"image TEXT" ")"

);

await db.execute(

"INSERT INTO Product ('id', 'name', 'description', 'price', 'image')

values (?, ?, ?, ?, ?)",

[1, "iPhone", "iPhone is the stylist phone ever", 1000, "iphone.png"]

);

await db.execute(

"INSERT INTO Product ('id', 'name', 'description', 'price', 'image')

values (?, ?, ?, ?, ?)",

[2, "Pixel", "Pixel is the most feature phone ever", 800, "pixel.png"]

);

await db.execute(

"INSERT INTO Product ('id', 'name', 'description', 'price', 'image')

values (?, ?, ?, ?, ?)",

[3, "Laptop", "Laptop is most productive development tool", 2000, "laptop.png"]\

);

await db.execute(

"INSERT INTO Product ('id', 'name', 'description', 'price', 'image')

values (?, ?, ?, ?, ?)",

[4, "Tablet", "Laptop is most productive development tool", 1500, "tablet.png"]

);

await db.execute(

"INSERT INTO Product

('id', 'name', 'description', 'price', 'image')

values (?, ?, ?, ?, ?)",

[5, "Pendrive", "Pendrive is useful storage medium", 100, "pendrive.png"]

);

await db.execute(

"INSERT INTO Product

('id', 'name', 'description', 'price', 'image')

values (?, ?, ?, ?, ?)",

[6, "Floppy Drive", "Floppy drive is useful rescue storage medium", 20, "floppy.png"]

);

}

);

}

Chúng ta đã sử dụng một số phương thức như sau :

**getApplicationDocumentsDirectory**: trả về đường dẫn thư mục ứng dụng

**join**: sử dụng để tạo đường dẫn cụ thể. Ta đã sử  dụng nó để tạo đường dẫn database

**openDatabase**: sử dụng để mở SQLite database

**onOpen**: Được dùng để viết code khi mở database

**onCreate**: Sử dụng để viết code khi database được tạo lần đầu

**db.execute :**Sử dụng để thực thi truy vấn SQL

Viết hàm để lấy tất cả product từ database

Future<List<Product>> getAllProducts() async {

final db = await database;

List<Map>

results = await db.query("Product", columns: Product.columns, orderBy: "id ASC");

List<Product> products = new List();

results.forEach((result) {

Product product = Product.fromMap(result);

products.add(product);

});

return products;

}

Ở trên, chúng ta đã làm như sau :

1. Sử dụng phương thức truy vấn để lấy tất cả thông tin của product. Truy vấn cung cấp lối tắt để truy cập vào thông tin bảng mà không phải viết toàn bộ truy vấn . Phương thức truy vấn sẽ tạo truy vấn chính nó bằng việc sử dụng đầu vào như columns, orderBy , ...
2. Sử dụng phương thức Product’s fromMap để lấy chi tiết product bằng việc chạy vòng lặp các đối tượng

Chúng ta viết hàm để lấy **id**cụ thể của product

Future<Product> getProductById(int id) async {

final db = await database;

var result = await db.query("Product", where: "id = ", whereArgs: [id]);

return result.isNotEmpty ? Product.fromMap(result.first) : Null;

}

Ta đã sử  dụng where và whereArgs để áp dụng bộ lọc

Tạo 3 hàm -  insert, update và delete để thêm, cập nhật và xoá product từ db(database)

insert(Product product) async {

final db = await database;

var maxIdResult = await db.rawQuery(

"SELECT MAX(id)+1 as last\_inserted\_id FROM Product");

var id = maxIdResult.first["last\_inserted\_id"];

var result = await db.rawInsert(

"INSERT Into Product (id, name, description, price, image)"

" VALUES (?, ?, ?, ?, ?)",

[id, product.name, product.description, product.price, product.image]

);

return result;

}

update(Product product) async {

final db = await database;

var result = await db.update("Product", product.toMap(),

where: "id = ?", whereArgs: [product.id]); return result;

}

delete(int id) async {

final db = await database;

db.delete("Product", where: "id = ?", whereArgs: [id]);

}

Database.dart như sau :

import 'dart:async';

import 'dart:io';

import 'package:path/path.dart';

import 'package:path\_provider/path\_provider.dart';

import 'package:sqflite/sqflite.dart';

import 'Product.dart';

class SQLiteDbProvider {

SQLiteDbProvider.\_();

static final SQLiteDbProvider db = SQLiteDbProvider.\_();

static Database \_database;

Future<Database> get database async {

if (\_database != null)

return \_database;

\_database = await initDB();

return \_database;

}

initDB() async {

Directory documentsDirectory = await

getApplicationDocumentsDirectory();

String path = join(documentsDirectory.path, "ProductDB.db");

return await openDatabase(

path, version: 1,

onOpen: (db) {},

onCreate: (Database db, int version) async {

await db.execute(

"CREATE TABLE Product ("

"id INTEGER PRIMARY KEY,"

"name TEXT,"

"description TEXT,"

"price INTEGER,"

"image TEXT"")"

);

await db.execute(

"INSERT INTO Product ('id', 'name', 'description', 'price', 'image')

values (?, ?, ?, ?, ?)",

[1, "iPhone", "iPhone is the stylist phone ever", 1000, "iphone.png"]

);

await db.execute(

"INSERT INTO Product ('id', 'name', 'description', 'price', 'image')

values (?, ?, ?, ?, ?)",

[2, "Pixel", "Pixel is the most feature phone ever", 800, "pixel.png"]

);

await db.execute(

"INSERT INTO Product ('id', 'name', 'description', 'price', 'image')

values (?, ?, ?, ?, ?)",

[3, "Laptop", "Laptop is most productive development tool", 2000, "laptop.png"]

);

await db.execute(

"INSERT INTO Product ('id', 'name', 'description', 'price', 'image')

values (?, ?, ?, ?, ?)",

[4, "Tablet", "Laptop is most productive development tool", 1500, "tablet.png"]

);

await db.execute(

"INSERT INTO Product ('id', 'name', 'description', 'price', 'image')

values (?, ?, ?, ?, ?)",

[5, "Pendrive", "Pendrive is useful storage medium", 100, "pendrive.png"]

);

await db.execute(

"INSERT INTO Product ('id', 'name', 'description', 'price', 'image')

values (?, ?, ?, ?, ?)",

[6, "Floppy Drive", "Floppy drive is useful rescue storage medium", 20, "floppy.png"]

);

}

);

}

Future<List<Product>> getAllProducts() async {

final db = await database;

List<Map> results = await db.query(

"Product", columns: Product.columns, orderBy: "id ASC"

);

List<Product> products = new List();

results.forEach((result) {

Product product = Product.fromMap(result);

products.add(product);

});

return products;

}

Future<Product> getProductById(int id) async {

final db = await database;

var result = await db.query("Product", where: "id = ", whereArgs: [id]);

return result.isNotEmpty ? Product.fromMap(result.first) : Null;

}

insert(Product product) async {

final db = await database;

var maxIdResult = await db.rawQuery("SELECT MAX(id)+1 as last\_inserted\_id FROM Product");

var id = maxIdResult.first["last\_inserted\_id"];

var result = await db.rawInsert(

"INSERT Into Product (id, name, description, price, image)"

" VALUES (?, ?, ?, ?, ?)",

[id, product.name, product.description, product.price, product.image]

);

return result;

}

update(Product product) async {

final db = await database;

var result = await db.update(

"Product", product.toMap(), where: "id = ?", whereArgs: [product.id]

);

return result;

}

delete(int id) async {

final db = await database;

db.delete("Product", where: "id = ?", whereArgs: [id]);

}

}

Thay đổi trong hàm main để lấy thông tin product

void main() {

runApp(MyApp(products: SQLiteDbProvider.db.getAllProducts()));

}

Ở đây ta đã dùng getAllProducts để lấy tất cả sản phẩm từ db

Chạy ứng dụng lên và ta sẽ thấy được kết quả. Nó sẽ giống với kết quả ở ví dụ trước "*REST API*"

Cloud Firestore

Firebase là nền tảng phát triển ứng dụng Baas(Backend-as-a-Service). Nhiều lập trình viên trên thế giới đã sử dụng mBaaS bởi vì nhiều lợi ích nó mang đến . Nó cung cấp nhiều tính năng để tăng tốc việc phát triển ứng dụng như xác thực , lưu trữ đám mây ,... Một trong những tính năng chính của Firebase là Cloud Firestore, là cloud dựa trên NoSQL database với thời gian thực

Flutter cung cấp gói cụ thể, cloud\_firestore để làm việc với Cloud Firebase . Chúng ta hãy tạo cửa hàng product trực tuyến trên Cloud Firestore và tạo ứng dụng để truy cập vào .

1. Tạo ừng dụng flutter mới tên là product\_firebase\_app
2. Cooy tệp Product.dart từ thư mục product\_rest\_app sang

class Product {

final String name;

final String description;

final int price;

final String image;

Product(this.name, this.description, this.price, this.image);

factory Product.fromMap(Map<String, dynamic> json) {

return Product(

json['name'],

json['description'],

json['price'],

json['image'],

);

}

}

Copy thư mục assets từ product\_rest\_app vào product\_firebase\_app và thêm vào file pubspec.yaml

flutter:

assets:

- assets/appimages/floppy.png

- assets/appimages/iphone.png

- assets/appimages/laptop.png

- assets/appimages/pendrive.png

- assets/appimages/pixel.png

- assets/appimages/tablet.png

Cấu hình gói cloud\_firestore trong tệp pubspec.yaml như sau :

dependencies: cloud\_firestore: ^0.9.13+1

Ta đang sử dụng phiên bản mới nhất của cloud\_firestore (ở thời điểm hiện tại)

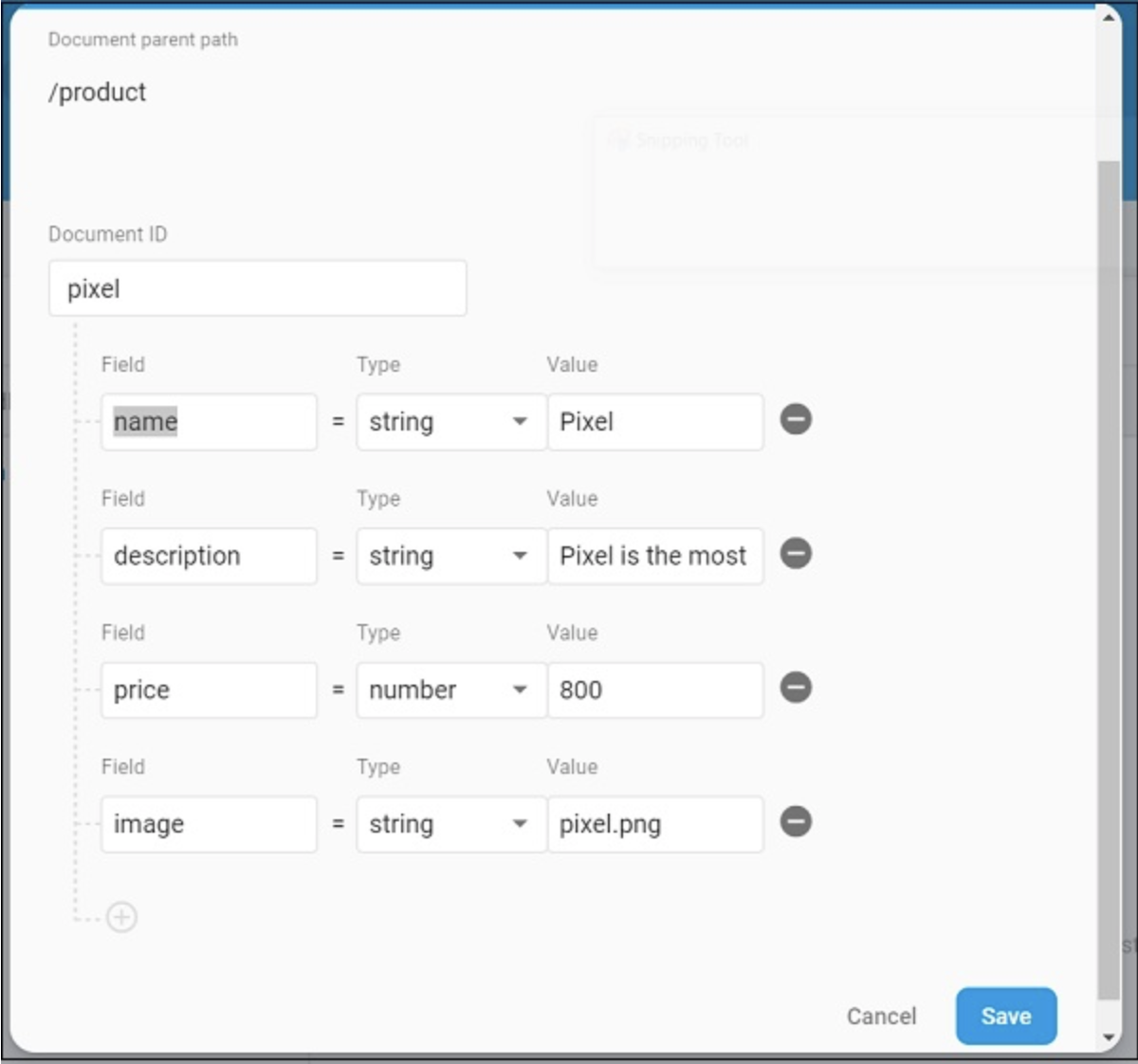
Android studio sẽ thông báo cập nhật và Get dependencies nó.

Sau đó ta khởi tạo project Firebase theo từng bước sau đây :

1. Mở https://firebase.google.com/pricing/
2. Tạo một tài khoảng Firebase và tạo mới Project
3. Các bạn làm theo hướng dẫn chi tiết trong video này nhé : <https://www.youtube.com/watch?v=6juww5Lmvgo>
4. Lưu ý : bạn hãy nhớ kết nối project của mình với firebase nhé

Oke, ta hãy tạo cửa hàng sản phẩm mới như sau :

1. Đi đến Firebase console
2. Mở project mình vừa tạo
3. Nhấn vào tính năng Database trong menu bên trái
4. Tạo database
5. Nhấn Start trong test mode và EnaEnable
6. Click Add collection.



Mở tệp main.dart và thêm plugin Cloud Firestore , xoá http package

import 'package:cloud\_firestore/cloud\_firestore.dart';

Xoá parseProducts và cập nhật fetchProducts để lấy products từ Cloud Firestore thay vì  Product service API.

Stream<QuerySnapshot> fetchProducts() {

return Firestore.instance.collection('product').snapshots(); }

Ở đây, p[hương thức Firestore.instance.collection được sử dụng để truy cập vào giá trị product trên cloud store.  Firestore.instance.collection cung cấp nhiều tính năng để lọc và lấy tài liệu cần thiết. Nhưng chúng ta không áp dụng bất kì bộ lọc nào để lấy toàn bộ thông tin product

Cloud Firestore cung cấp bộ sưu tập thông qua khái niệm Dart Stream và sửa đổi kiểu dữ liệu products trong widget MyApp và MyHomePage từ Future<List<Product>> sang Stream<QuerySnapshot>

Thay đổi phương thức build của widget MyHomePage để sử dụng StreamBuilder thay vì StreamBuilder

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text("Product Navigation")),

body: Center(

child: StreamBuilder<QuerySnapshot>(

stream: products, builder: (context, snapshot) {

if (snapshot.hasError) print(snapshot.error);

if(snapshot.hasData) {

List<DocumentSnapshot>

documents = snapshot.data.documents;

List<Product>

items = List<Product>();

for(var i = 0; i < documents.length; i++) {

DocumentSnapshot document = documents[i];

items.add(Product.fromMap(document.data));

}

return ProductBoxList(items: items);

} else {

return Center(child: CircularProgressIndicator());

}

},

),

)

);

}

Ở trên, ta đã lấy được dữ liệu thông tin product như kiểu List<DocumentSnapshot>.

Cuối cùng, ta chạy ứng dụng và xem kết quả . Chúng ta đã sử dụng cùng thông tin product trong *SQLite application*và chỉ thay đổi nơi lưu trữ vì thế ứng dụng sẽ hiển thị giống như *SQLite application .*