

Flutter: Truy xuất và lưu trữ dữ liệu

Bùi Võ Quốc Bảo

Khoa CNTT | Trường CNTT-TT | Đại học Cần Thơ

Tài liệu tham khảo

- Chương 9-10-11-15, **Flutter Apprentice** by Vincenzo Guzzi, Kevin D Moore, Vincent Ngo and Michael Katz
- <https://docs.flutter.dev/cookbook#networking>
- <https://docs.flutter.dev/cookbook/persistence>
- <https://docs.flutter.dev/cookbook/persistence/reading-writing-files>

Truy xuất và lưu trữ dữ liệu

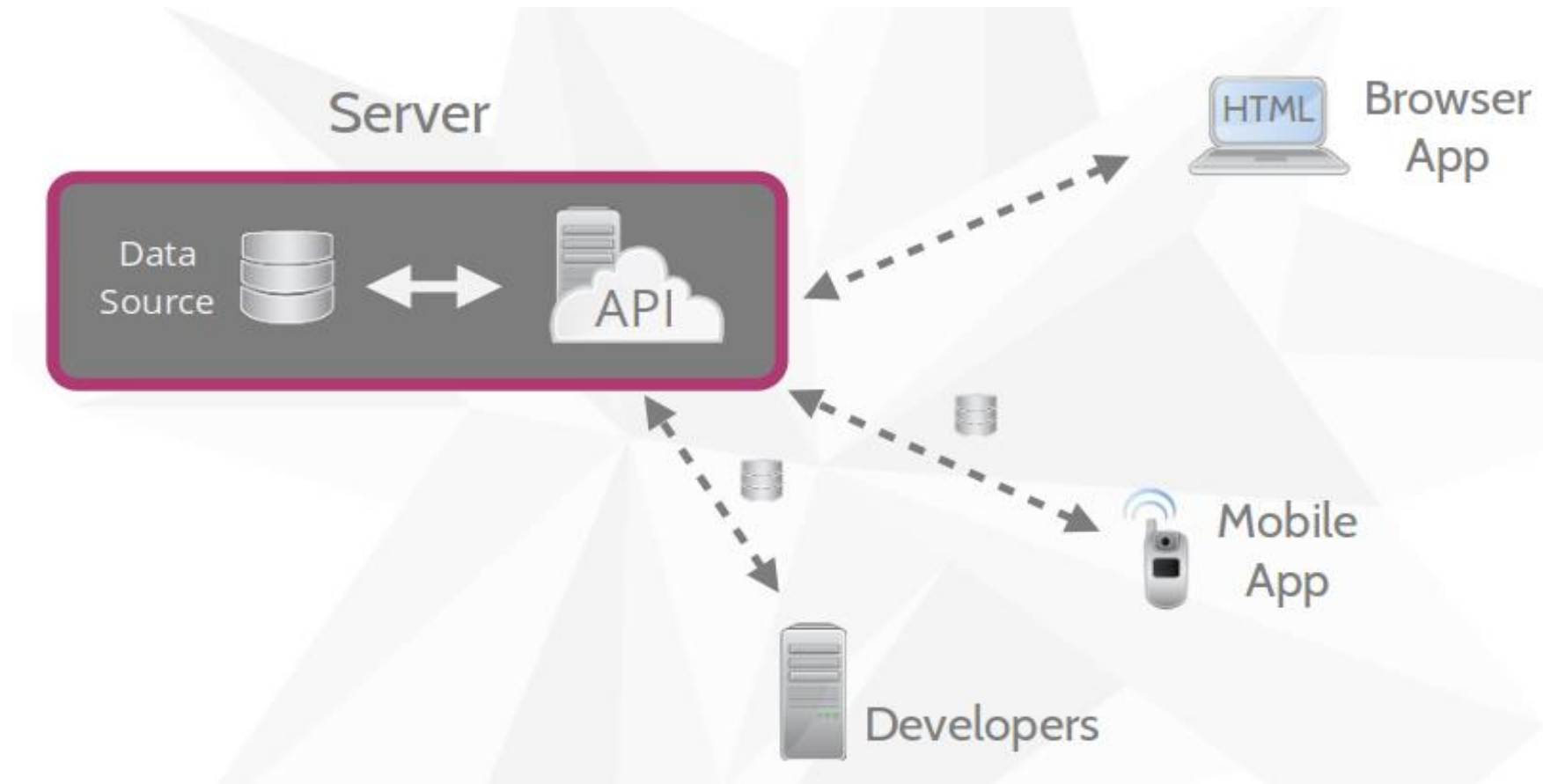
- Dữ liệu từ xa/trực tuyến
 - Truy xuất dữ liệu qua JSON/REST API
 - Hoặc dùng các SDK riêng của dịch vụ lưu trữ
- Dữ liệu cục bộ/ngoại tuyến
 - Dữ liệu đơn giản (khóa-giá trị) trên vị trí được quản lý hệ thống (**SharedPreferences** trên Android, **NSUserDefaults** trên iOS/macOS)
 - Dữ liệu phức tạp trên cơ sở dữ liệu SQL/NoSQL cục bộ (e.g., SQLite, [Isar](#), [ObjectBox](#), ...)
 - Dữ liệu có định dạng tùy biến trên tập tin
 - Các dịch vụ lưu trữ có thể hỗ trợ tính năng lưu trữ ngoại tuyến, đồng bộ hóa dữ liệu

JSON/REST API

REST API

- REST = REpresentational State Transfer
- API được truy xuất thông qua giao thức HTTP
 - Một cơ chế giao tiếp liên tiến trình dựa trên yêu cầu-trả lời của HTTP
 - Dùng URI để chỉ định các tài nguyên và các phương thức HTTP (GET, POST, PUT, ...) để thể hiện loại tác vụ sẽ được thực thi
 - Định dạng dữ liệu trao đổi (trạng thái đại diện) được sử dụng phổ biến là JSON
 - Một đặc tả/giao diện, không phải là một công nghệ → có thể được xây dựng bằng nhiều công nghệ khác nhau: Java, .NET, PHP,...

REST API



REST API

- GET <collection URI> = Đọc tất cả các tài nguyên
- GET <instance URI> = Đọc một tài nguyên
- POST <collection URI> = Tạo mới một tài nguyên
- PATCH <instance URI> = Cập nhật một phần tài nguyên
- PUT <instance URI> = Tạo mới hoặc thay thế tài nguyên
- DELETE <instance URI> = Xóa một tài nguyên

REST API

Ví dụ API server <https://jsonplaceholder.typicode.com/> có hỗ trợ một số lời gọi sau:

- GET /posts = Liệt kê các bài đăng
- GET /posts/1 = Đọc một bài đăng cụ thể (id = 1)
- POST /posts = Tạo mới một bài đăng
- PUT /posts/1 = Cập nhật bài đăng cụ thể (id = 1)
- DELETE /posts/1 = Xóa một bài đăng cụ thể (id = 1)

Lưu ý: trên trình giả lập Android, dùng địa chỉ host là **10.0.2.2** thay vì **localhost** để truy xuất API server chạy trên máy cục bộ

Định dạng JSON

- JSON = JavaScript Object Notation
 - Một định dạng trao đổi dữ liệu dựa trên văn bản phổ biến
 - Một tập hợp các cặp name/value. Một cặp name/value gồm tên trường (trong dấu nháy kép), theo bởi dấu hai chấm, theo bởi một giá trị, e.g., "name": "Bao"
 - Một giá trị có thể là một chuỗi trong dấu nháy kép, một số, true/false/null, một đối tượng hoặc một mảng. Các cấu trúc này có thể lồng nhau
 - Không hỗ trợ ghi chú thích

Định dạng JSON

- JSON = JavaScript Object Notation
 - Một định dạng trao đổi dữ liệu dựa trên văn bản
- Trong dart, định dạng JSON có thể được biểu diễn dưới dạng kiểu **Map<String, dynamic>** hoặc **Map<String, Object?>**
 - Tương tự kiểu đối tượng trong JavaScript hoặc mảng trong PHP

```
{
  "empid": "SJ011MS",
  "personal": {
    "name": "Smith Jones",
    "gender": "Male",
    "age": 28,
    "address": {
      "streetaddress": "7 24th Street",
      "city": "New York",
      "state": "NY",
      "postalcode": "10038"
    }
  },
  "profile": {
    "designation": "Deputy General",
    "department": "Finance"
  }
}
```

www.kodingmadesimple.com

Định dạng JSON

- Trong dart, định dạng JSON có thể được biểu diễn dưới dạng kiểu **Map<String, dynamic>** hoặc **Map<String, Object?>**

```
final jsonString = '''
{
  "numField": 1,
  "strField": "string value",
  "boolField": true,
  "listField": [
    "list item 1",
    "list item 2",
    "list item 3"
  ],
  "objectField": {
    "field1": 1,
    "field2": "hello"
  }
}
''';
```

jsonDecode()

jsonEncode()

[dart:convert](#)

```
const Map<String, Object?> json = {
  'numField': 1,
  'strField': 'string value',
  'boolField': true,
  'listField': [
    'list item 1',
    'list item 2',
    'list item 3',
  ],
  'objectField': {
    'field1': 1,
    'field2': 'hello',
  }
};
```

Định dạng JSON

- Chuyển đổi giữa `Map<String, dynamic>` hoặc `Map<String, Object?>` và mô hình dữ liệu
 - Thường sử dụng phương thức **`fromJson()`** và **`toJson()`**

```
class MyModel {  
    final int numField;  
    final String strField;  
    final bool boolField;  
    final List<Object?> listField;  
    final Map<String, Object?> objectField;  
  
    const MyModel({  
        required this.numField,  
        required this.strField,  
        required this.boolField,  
        required this.listField,  
        required this.objectField,  
    });  
}
```

Đọc dữ liệu từ Map vào mô hình dữ liệu

```
factory MyModel.fromJson(Map<String, Object?> json) {  
    return MyModel(  
        numField: json['numField'] as int,  
        strField: json['strField'] as String,  
        boolField: json['boolField'] as bool,  
        listField: json['listField'] as List<Object?>,  
        objectField: json['objectField'] as Map<String, Object?>,  
    );  
}
```

Định dạng JSON

- Chuyển đổi giữa `Map<String, dynamic>` hoặc `Map<String, Object?>` và mô hình dữ liệu
 - Thường sử dụng phương thức **`fromJson()`** và **`toJson()`**

```
class MyModel {  
    final int numField;  
    final String strField;  
    final bool boolField;  
    final List<Object?> listField;  
    final Map<String, Object?> objectField;  
  
    const MyModel({  
        required this.numField,  
        required this.strField,  
        required this.boolField,  
        required this.listField,  
        required this.objectField,  
    });  
}
```

Tạo Map từ mô hình dữ liệu

```
Map<String, Object?> toJson() {  
    return {  
        'numField': numField,  
        'strField': strField,  
        'boolField': boolField,  
        'listField': listField,  
        'objectField': objectField  
    };  
}
```

JSON/REST API – Ví dụ 1

GET <http://jsonplaceholder.typicode.com/albums/1>

Dữ liệu JSON trả lời

```
{
  "userId": 1,
  "id": 1,
  "title": "quidem molestiae enim"
}
```

Cài đặt gói [http](http://pub.dev/packages/http)

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.4
```

Mô hình dữ liệu SimpleAlbum

```
class SimpleAlbum {
  final int id;
  final String title;

  const SimpleAlbum({required this.id, required this.title});

  factory SimpleAlbum.fromJson(Map<String, dynamic> json) {
    return SimpleAlbum(
      id: json['id'],
      title: json['title'],
    );
  }
}
```

JSON/REST API – Ví dụ 1

- Lớp dịch vụ truy xuất JSON API và tạo đối tượng SimpleAlbum

```
import 'dart:convert';

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

class AlbumService {
  Future<SimpleAlbum> fetchAlbum() async {
    final response = await http.get(
      Uri.parse('https://jsonplaceholder.typicode.com/albums/1'),
    );

    if (response.statusCode == 200) {
      return SimpleAlbum.fromJson(jsonDecode(response.body));
    } else {
      throw Exception('Failed to load album');
    }
  }
}
```

```
Future<SimpleAlbum> updateAlbum(String title) async {
  final response = await http.put(
    Uri.parse('https://jsonplaceholder.typicode.com/albums/1'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, String>{
      'title': title,
    }),
  );

  if (response.statusCode == 200) {
    return SimpleAlbum.fromJson(jsonDecode(response.body));
  } else {
    throw Exception('Failed to update album.');
```

JSON/REST API – Ví dụ 1

- **FutureBuilder<T>:**

tạo widget dựa trên
đối tượng thuộc kiểu
Future<T>

- Không nên tạo đối tượng Future trực tiếp trong FutureBuilder. Thay vào đó, khởi tạo Future trong initState

```
class _MyAppState extends State<MyApp> {  
  final TextEditingController _controller = TextEditingController();  
  final albumService = AlbumService();  
  late Future<SimpleAlbum> _futureAlbum;  
  
  @override  
  void initState() {  
    super.initState();  
    _futureAlbum = albumService.fetchAlbum();  
  }  
  
  @override  
  Widget build(BuildContext context) {
```


JSON/REST API – Ví dụ 1

- **FutureBuilder<T>**: tạo widget dựa trên đối tượng thuộc kiểu **Future<T>**
 - Không nên tạo đối tượng Future trực tiếp trong FutureBuilder. Thay vào đó, khởi tạo Future trong initState

```
child: FutureBuilder<SimpleAlbum>(  
  future: _futureAlbum,  
  builder: (context, snapshot) {  
    if (snapshot.connectionState == ConnectionState.done) {  
      if (snapshot.hasData) {  
        return Column(  
          mainAxisAlignment: MainAxisAlignment.center,  
          children: <Widget>[  
            Text(snapshot.data!.title),  
            TextField(  
              controller: _controller,  
              decoration: const InputDecoration(  
                hintText: 'Enter Title',  
              ), // InputDecoration  
            ), // TextField  
            ElevatedButton(  
              onPressed: () {  
                setState(() {  
                  _futureAlbum =  
                    albumService.updateAlbum(_controller.text);  
                });  
              },  
              child: const Text('Update Data'),  
            ), // ElevatedButton  
          ], // <Widget>[]  
        );  
      }  
    }  
  },  
);
```

JSON/REST API – Ví dụ 2

- JSON API lấy về danh sách các photo

GET <http://jsonplaceholder.typicode.com/photos>

```
[
  {
    "albumId": 1,
    "id": 1,
    "title": "accusamus beatae ad facilis cum similique qui sunt",
    "url": "https://via.placeholder.com/600/92c952",
    "thumbnailUrl": "https://via.placeholder.com/150/92c952"
  },
  {
    "albumId": 1,
    "id": 2,
    "title": "reprehenderit est deserunt velit ipsam",
    "url": "https://via.placeholder.com/600/771796",
    "thumbnailUrl": "https://via.placeholder.com/150/771796"
  },
  {
    "albumId": 1,
    "id": 3,
    "title": "officia porro iure quia iusto qui ipsa ut modi",
    "url": "https://via.placeholder.com/600/24f355",
    "thumbnailUrl": "https://via.placeholder.com/150/24f355"
  },
]
```

JSON/REST API – Ví dụ 2

- Mô hình dữ liệu Photo

```
class Photo {  
    final int albumId;  
    final int id;  
    final String title;  
    final String url;  
    final String thumbnailUrl;  
  
    const Photo({  
        required this.albumId,  
        required this.id,  
        required this.title,  
        required this.url,  
        required this.thumbnailUrl,  
    });  
  
    factory Photo.fromJson(Map<String, dynamic> json) {
```

JSON/REST API – Ví dụ 2

- Lớp dịch vụ truy xuất JSON API và tạo Future<List<Photo>>

```
class PhotoService {  
    Future<List<Photo>> fetchPhotos() async {  
        final response = await http.get(  
            Uri.parse('https://jsonplaceholder.typicode.com/photos'),  
        );  
  
        return parsePhotos(response.body);  
    }  
  
    // A function that converts a response body into a List<Photo>.  
    List<Photo> parsePhotos(String responseBody) {  
        final parsed = jsonDecode(responseBody);  
        return (parsed as List)  
            .map<Photo>((json) => Photo.fromJson(json as Map<String, dynamic>))  
            .toList();  
    }  
}
```

JSON/REST API – Ví dụ 2

- Dùng FutureBuilder tạo GridView từ Future<List<Photo>>

```
class _MyHomePageState extends State<MyHomePage> {  
  Future<List<Photo>>? _photoList;  
  
  @override  
  void initState() {  
    super.initState();  
    _photoList = PhotoService().fetchPhotos();  
  }  
}
```

```
@override  
Widget build(BuildContext context) {  
  return Scaffold(  
    appBar: AppBar(  
      title: Text(widget.title),  
    ), // AppBar  
    body: FutureBuilder<List<Photo>>(  
      future: _photoList,  
      builder: (context, snapshot) {  
        if (snapshot.hasError) {  
          return const Center(  
            child: Text('An error has occurred!'),  
          ); // Center  
        } else if (snapshot.hasData) {  
          return PhotosList(photos: snapshot.data!);  
        } else {  
          return const Center(  
            child: CircularProgressIndicator(),  
          ); // Center  
        }  
      },  
    ), // FutureBuilder  
  ); // Scaffold  
}
```

JSON/REST API – Ví dụ 2

- Dùng FutureBuilder tạo GridView từ Future<List<Photo>>

```
class PhotosList extends StatelessWidget {  
  const PhotosList({super.key, required this.photos});  
  
  final List<Photo> photos;  
  
  @override  
  Widget build(BuildContext context) {  
    return GridView.builder(  
      gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(  
        crossAxisCount: 2,  
      ), // SliverGridDelegateWithFixedCrossAxisCount  
      itemCount: photos.length,  
      itemBuilder: (context, index) {  
        return Image.network(photos[index].thumbnailUrl);  
      },  
    ); // GridView.builder  
  }  
}
```

JSON/REST API – Ví dụ 2

- Đọc JSON trong một isolate riêng

```
import 'dart:convert';

import 'package:flutter/foundation.dart';
import 'package:http/http.dart' as http;
import 'photo.dart';

class PhotoService {
  Future<List<Photo>> fetchPhotos() async {
    final response = await http.get(
      Uri.parse('https://jsonplaceholder.typicode.com/photos'),
    );

    return compute(parsePhotos, response.body);
  }
}
```

JSON/REST API – Ví dụ 2

- Đọc JSON trong một isolate riêng
 - compute(callback, message): chạy **callback(message)** trong một isolate riêng và trả về kết quả
 - +callback: có thể là hàm, phương thức tĩnh, bao đóng (closure)
 - +message và dữ liệu trả về thuộc các kiểu dữ liệu cơ bản như **null**, **num**, **bool**, **double**, **String** hoặc các đối tượng đơn giản như **List<Photo>**

Shared Preferences

Shared Preferences

- Vùng lưu trữ dữ liệu đơn giản (dạng khóa-giá trị) được quản lý bởi hệ thống (**SharedPreferences** trên Android, **NSUserDefaults** trên iOS/macOS)
- Sử dụng gói [shared_preferences](#) để đọc và lưu trữ thông tin trong vùng lưu trữ chia sẻ
 - Việc ghi thông tin lên đĩa là không đảm bảo, do đó không dùng lưu những thông tin quan trọng
 - Các kiểu dữ liệu được hỗ trợ: **int**, **double**, **bool**, **String** và **List<String>**

shared_preferences

- Ghi dữ liệu

```
// Obtain shared preferences.
final prefs = await SharedPreferences.getInstance();

// Save an integer value to 'counter' key.
await prefs.setInt('counter', 10);
// Save an boolean value to 'repeat' key.
await prefs.setBool('repeat', true);
// Save an double value to 'decimal' key.
await prefs.setDouble('decimal', 1.5);
// Save an String value to 'action' key.
await prefs.setString('action', 'Start');
// Save an list of strings to 'items' key.
await prefs.setStringList('items', <String>['Earth', 'Moon', 'Sun']);
```

shared_preferences

- Đọc và xóa dữ liệu

```
// Try reading data from the 'counter' key. If it doesn't exist, returns null.  
final int? counter = prefs.getInt('counter');  
// Try reading data from the 'repeat' key. If it doesn't exist, returns null.  
final bool? repeat = prefs.getBool('repeat');  
// Try reading data from the 'decimal' key. If it doesn't exist, returns null.  
final double? decimal = prefs.getDouble('decimal');  
// Try reading data from the 'action' key. If it doesn't exist, returns null.  
final String? action = prefs.getString('action');  
// Try reading data from the 'items' key. If it doesn't exist, returns null.  
final List<String>? items = prefs.getStringList('items');
```

```
// Remove data for the 'counter' key.  
final success = await prefs.remove('counter');
```

shared_preferences – Ví dụ

- Mô hình dữ liệu AppSettings

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

class AppSettings extends ChangeNotifier {
  static const String prefDarkMode = 'pref_dark_mode';
  final SharedPreferences _storage;

  late bool _isDarkMode;
  bool get isDarkMode => _isDarkMode;

  AppSettings({storage}) : _storage = storage {
    _isDarkMode = _storage.getBool(prefDarkMode) ?? false;
  }

  void toggleTheme() {
    _isDarkMode = !_isDarkMode;
    notifyListeners();
    _storage.setBool(prefDarkMode, _isDarkMode);
  }
}
```

shared_preferences – Ví dụ

- Mô hình dữ liệu AppSettings

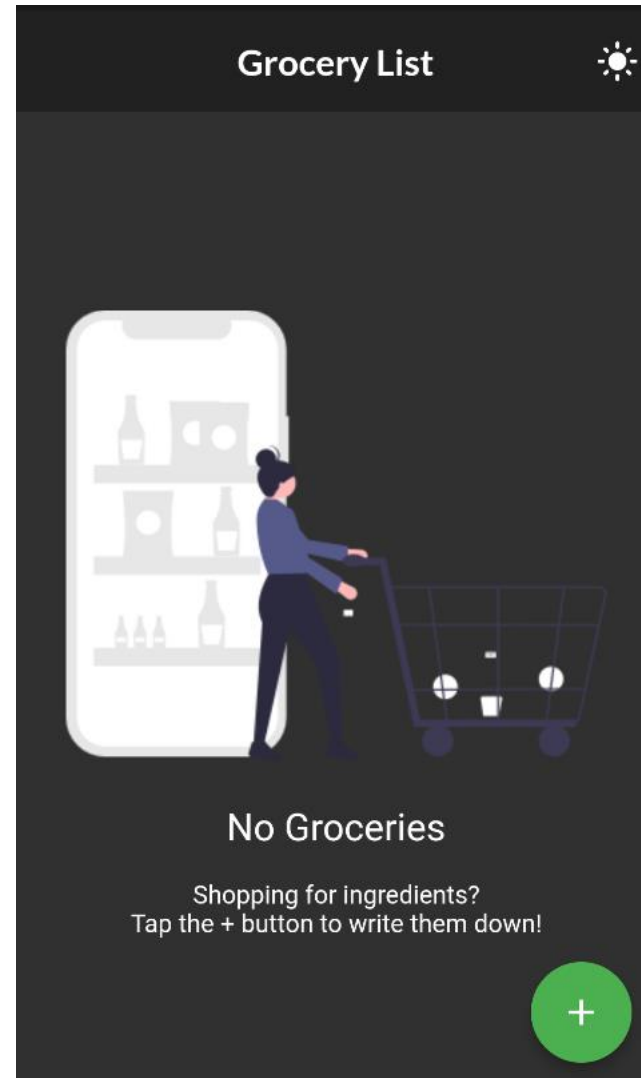
```
void main() async {  
  WidgetsFlutterBinding.ensureInitialized();  
  
  final prefs = await SharedPreferences.getInstance();  
  
  runApp(  
    ChangeNotifierProvider(  
      create: (context) => AppSettings(storage: prefs),  
      child: const Fooderlich(),  
    ), // ChangeNotifierProvider  
  );  
}
```

```
class Fooderlich extends StatelessWidget {  
  const Fooderlich({Key? key}) : super(key: key);  
  @override  
  Widget build(BuildContext context) {  
    final settings = context.watch<AppSettings>();  
    return MaterialApp(  
      debugShowCheckedModeBanner: false,  
      theme: settings.isDarkMode  
        ? FooderlichTheme.dark()  
        : FooderlichTheme.light(),  
      title: 'Fooderlich',  
      home: ChangeNotifierProvider(  
        create: (context) => GroceryManager(),  
        child: const GroceryScreen(),  
      ), // ChangeNotifierProvider  
    ); // MaterialApp  
  }  
}
```

shared_preferences – Ví dụ

- Tạo nút chuyển đổi qua lại giữa hai chế độ hiển thị

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      elevation: 0,
      title: Text(
        'Grocery List',
        style: GoogleFonts.lato(fontWeight: FontWeight.w600),
      ), // Text
      actions: [
        IconButton(
          icon: context.watch<AppSettings>().isDarkMode
            ? const Icon(Icons.light_mode)
            : const Icon(Icons.dark_mode),
          onPressed: () {
            context.read<AppSettings>().toggleTheme();
          },
        ) // IconButton
      ],
    ), // AppBar
```



SQLite

Lưu trữ dữ liệu trong SQLite

- Khi ứng dụng cần lưu trữ và truy xuất lượng lớn dữ liệu thì nên xem xét sử dụng một cơ sở dữ liệu (CSDL) thay vì tập tin cục bộ hay kho lưu trữ khóa-giá trị
- SQLite (<https://www.sqlite.org/index.html>) là một hệ CSDL quan hệ nhỏ gọn thích hợp cho các thiết bị di động
- Dùng gói [sqflite](#) để làm việc với SQLite trong Flutter
- Hướng dẫn sử dụng:
 - <https://github.com/tekartik/sqflite/blob/master/sqflite/README.md>
 - https://github.com/tekartik/sqflite/blob/master/sqflite/doc/how_to.md

sqflite – Ví dụ

- Khai báo sử dụng gói **sqflite** và [path](#)

```
dependencies:  
  flutter:  
    sdk: flutter  
  google_fonts: ^3.0.1  
  provider: ^6.0.3  
  flutter_colorpicker: ^1.0.3  
  intl: ^0.17.0  
  uuid: ^3.0.6  
  shared_preferences: ^2.0.15  
  sqflite: ^2.0.2+1  
  path: ^1.8.1
```

- Tạo lớp truy xuất CSDL

```
import 'package:sqflite/sqflite.dart';  
import 'package:path/path.dart';  
import '../models/grocery_item.dart';  
  
class GroceryService {  
  static final GroceryService instance = GroceryService._sharedInstance();  
  GroceryService._sharedInstance();  
  
  Database? _groceryDatabase;  
  Future<Database> get _database async {  
    if (_groceryDatabase != null) {  
      return _groceryDatabase!;  
    } else {  
      _groceryDatabase = await _initDatabase('grocery.db');  
      return _groceryDatabase!;  
    }  
  }  
  
  Future<Database> _initDatabase(String fileName) async {  
    final databasePath = await getDatabasesPath();  
    final filePath = join(databasePath, fileName);  
    return await openDatabase(filePath,  
      version: 1, onCreate: _createGroceryItemsTable);  
  }  
}
```

sqflite – Ví dụ

- Tạo lớp truy xuất CSDL

```
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';
import '../models/grocery_item.dart'; Cài đặt singleton

class GroceryService {
  static final GroceryService instance = GroceryService._sharedInstance();
  GroceryService._sharedInstance();

  Database? _groceryDatabase;
  Future<Database> get _database async {
    if (_groceryDatabase != null) {
      return _groceryDatabase!;
    } else {
      _groceryDatabase = await _initDatabase('grocery.db');
      return _groceryDatabase!;
    }
  }

  Future<Database> _initDatabase(String fileName) async {
    final databasePath = await getDatabasesPath();
    final filePath = join(databasePath, fileName);
    return await openDatabase(filePath,
      version: 1, onCreate: _createGroceryItemsTable);
  }
}
```

```
Future<void> _createGroceryItemsTable(Database db, int version) async {
  await db.execute('''CREATE TABLE grocery_items (
    id TEXT PRIMARY KEY NOT NULL,
    name TEXT,
    importance INTEGER,
    color INTEGER,
    quantity INTEGER,
    date TEXT,
    isComplete INTEGER
  )''');
}

void close() {
  if (_groceryDatabase != null) {
    _groceryDatabase!.close();
    _groceryDatabase = null;
  }
}
```

Tạo bảng dữ liệu khi
tạo tập tin CSDL

sqlite – Ví dụ

- Tạo lớp truy xuất CSDL: phương thức đọc dữ liệu

```
Future<List<GroceryItem>> getGroceryItems() async {  
    final groceryDatabase = await _database;  
    final result = await groceryDatabase.query(  
        'grocery_items',  
        orderBy: 'date DESC',  
    );  
    return result.map((e) => GroceryItem.fromJson(e)).toList();  
}
```

sqlite – Ví dụ

- Dùng câu truy vấn SQL thô


```
List<Map> list =  
    await database.rawQuery('SELECT * FROM Test');
```

```
int count = Sqlite.firstIntValue(  
    await database.rawQuery(  
        'SELECT COUNT(*) FROM Test')  
    );
```

sqlite – Ví dụ

- Tạo lớp truy xuất CSDL: phương thức thêm dữ liệu

```
Future<GroceryItem> addGroceryItem(GroceryItem item) async {  
    final groceryDatabase = await _database;  
    await groceryDatabase.insert(  
        'grocery_items',  
        item.toJson(),  
    );  
    return item;  
}
```



```
Map<String, Object?> toJson() => {  
    'id': id,  
    'name': name,  
    'importance': importance.index,  
    'color': color.value,  
    'quantity': quantity,  
    'date': date.toIso8601String(),  
    'isComplete': isComplete ? 1 : 0,  
};
```

sqlite – Ví dụ

- Dùng câu truy vấn SQL thô

```
int id1 = await database.rawInsert(  
    'INSERT INTO Test(name, value, num)  
    VALUES("some name", 1234, 456.789)');
```

```
int id2 = await database.rawInsert(  
    'INSERT INTO Test(name, value, num) VALUES(?, ?, ?)',  
    ['another name', 12345678, 3.1416]  
);
```

sqlite – Ví dụ

- Tạo lớp truy xuất CSDL: phương thức cập nhật dữ liệu

```
Future<int> updateGroceryItem(GroceryItem item) async {  
    final groceryDatabase = await _database;  
    return groceryDatabase.update(  
        'grocery_items',  
        item.toJson(),  
        where: 'id = ?',  
        whereArgs: [item.id],  
    );  
}
```


sqlite – Ví dụ

- Dùng câu truy vấn SQL thô

```
int count = await database.rawUpdate(  
    'UPDATE Test SET name = ?, value = ? WHERE name = ?',  
    ['updated name', '9876', 'some name']  
);
```

sqlite – Ví dụ

- Tạo lớp truy xuất CSDL: phương thức xóa dữ liệu

```
Future<int> deleteGroceryItem(GroceryItem item) async {  
    final groceryDatabase = await _database;  
    return groceryDatabase.delete(  
        'grocery_items',  
        where: 'id = ?',  
        whereArgs: [item.id],  
    );  
}
```

sqlite – Ví dụ

- Dùng câu truy vấn SQL thô

```
int count = await database.rawDelete(  
    'DELETE FROM Test WHERE name = ?',  
    ['another name']  
);
```

sqflite – Ví dụ

- Thực hiện truy vấn, thêm, cập nhật, xóa dữ liệu

```
import 'package:flutter/material.dart';
import 'grocery_item.dart';
import '../data/grocery_service.dart';

class GroceryManager extends ChangeNotifier {
  final _groceryService = GroceryService.instance;

  final _groceryItems = <GroceryItem>[];
  List<GroceryItem> get groceryItems => List.unmodifiable(_groceryItems);

  Future<void> loadGroceryItems() async {
    final items = await _groceryService.getGroceryItems();
    _groceryItems.clear();
    _groceryItems.addAll(items);
    notifyListeners();
  }

  void deleteItem(int index) {
    _groceryService.deleteGroceryItem(_groceryItems[index]);
    _groceryItems.removeAt(index);
    notifyListeners();
  }
}
```

```
void addItem(GroceryItem item) {
  _groceryService.addGroceryItem(item);
  _groceryItems.insert(0, item);
  notifyListeners();
}

void updateItem(GroceryItem item, int index) {
  _groceryService.updateGroceryItem(item);
  _groceryItems[index] = item;
  notifyListeners();
}

void completeItem(int index, bool change) {
  final item = _groceryItems[index].copyWith(isComplete: change);
  _groceryService.updateGroceryItem(item);
  _groceryItems[index] = item;
  notifyListeners();
}
```

sqflite – Ví dụ

- Đọc dữ liệu khi chạy ứng dụng

```
void main() async {  
  WidgetsFlutterBinding.ensureInitialized();  
  
  // Load app settings  
  final prefs = await SharedPreferences.getInstance();  
  final appSettings = AppSettings(storage: prefs);  
  
  // Load grocery items  
  final groceryManager = GroceryManager();  
  await groceryManager.loadGroceryItems();  
  
  runApp(  
    MultiProvider(  
      providers: [  
        ChangeNotifierProvider.value(value: appSettings),  
        ChangeNotifierProvider.value(value: groceryManager),  
      ],  
      child: const Fooderlich(),  
    ), // MultiProvider  
  );  
}
```

Dùng MultiProvider
để gửi appSettings
và groceryManager

Các đối tượng appSettings
và groceryManager đã tạo
trước đó nên dùng hàm xây
dựng value của Provider

Câu hỏi?