

JSON Serialization và Model Classes

GVHD: TS. Nguyễn Duy Nhật Viễn

Nhóm sinh viên thực hiện (7):

Ngô Trung Chinh - 106220211

Nguyễn Hữu Duy - 106220215

Đà Nẵng, 2025

Nội Dung

- **Tạo model classes với toJson() và fromJson()**
- **Xử lý nested objects và arrays**
- **Sử dụng json_annotation và build_runner**
- **Best practices cho complex data structures**

Cơ Sở Lý Thuyết

- **JSON (JavaScript Object Notation)** là một định dạng trao đổi dữ liệu nhẹ (lightweight data-interchange format), được dùng phổ biến để truyền dữ liệu giữa client và server trên web hoặc giữa các ứng dụng.
- Ví dụ :

```
{  
  "id": "106220215",  
  "name": "Nguyen Huu Duy",  
  "age": 21,  
  "email": "dnh@gmail.com",  
  "isActive": true  
}
```

Object { }

```
[  
  {"id": 1, "name": "Duy"},  
  {"id": 2, "name": "Chinh"}  
];
```

Array []

Cơ Sở Lý Thuyết

Tại sao cần chuyển đổi dữ liệu JSON thành Dart objects và ngược lại (JSON Serialization & Deserialization) ?

- 🔒 **An toàn kiểu:** Không lỗi do sai tên hoặc sai kiểu dữ liệu
- 🧩 **Dễ bảo trì:** Khi API thay đổi, chỉ cần cập nhật model
- ⚡ **Nhanh và gọn:** IDE hỗ trợ autocomplete và kiểm tra lỗi sớm
- 🔄 **Hai chiều linh hoạt:** Nhận dữ liệu từ API, hoặc gửi ngược dữ liệu lên server
- 🧱 **Kiến trúc rõ ràng:** Tách rời phần dữ liệu (Data Layer) và phần hiển thị (UI)

➡ **Việc chuyển JSON ↔ Dart giúp ứng dụng Flutter giao tiếp với server một cách an toàn, dễ hiểu, và dễ bảo trì.**

1) Model Classes, fromJson(), toJson()

- **Model class (hay “Data Model”)** là lớp đại diện cho một kiểu dữ liệu cụ thể trong chương trình.
- Cấu trúc tổng quát:

1. **Fields:** đại diện cho từng cột/thuộc tính trong JSON.
2. **Constructor:** để khởi tạo đối tượng.
3. **Phương thức fromJson():** nhận dữ liệu từ JSON (deserialize).
4. **Phương thức toJson():** chuyển object ngược lại thành JSON (serialize).
5. **(Tuỳ chọn):** enum, nested object, giá trị mặc định, validation, ...

1) Model Classes, fromJson(), toJson()

```
1 class Product {
2   final int id;
3   final String name;
4   final double price;
5
6   // Constructor
7   Product({
8     required this.id,
9     required this.name,
10    required this.price,
11  });
12
13  // JSON → Dart object (deserialize)
14  factory Product.fromJson(Map<String, dynamic> json) {
15    return Product(
16      id: json['id'],           // lấy id từ JSON
17      name: json['name'],       // lấy tên sản phẩm
18      price: (json['price']).toDouble(), // chuyển sang double
19    );
20  }
21
22  // Dart object → JSON (serialize)
23  Map<String, dynamic> toJson() {
24    return {
25      'id': id,
26      'name': name,
27      'price': price,
28    };
29  }
30 }
31
```

```
32 void main() {
33   final jsonData = {
34     "id": 101,
35     "name": "Chuột không dây",
36     "price": 250000};
37   final product = Product.fromJson(jsonData);
38
39   print('Tên sản phẩm: ${product.name}');
40   print('Giá: ${product.price}');
41   print('JSON xuất ra: ${product.toJson()}');
42 }
```

- Kết quả:

Tên sản phẩm: Chuột không dây

Giá: 250000.0

JSON xuất ra: {id: 101, name: Chuột không dây, price: 250000.0}

2) Nested Objects & Arrays

- **Nested Object** : Là một object nằm bên trong một object khác.

```
{
  "id": 1,
  "name": "Duy",
  "address": {
    "city": "DN",
    "country": "Vietnam"
  }
}
```

- **Array** : Là danh sách các giá trị, có thể là:
 - Kiểu đơn giản (string, number, bool)
 - Hoặc danh sách object.

```
{
  "products": [
    {"id": 1, "name": "Laptop"},
    {"id": 2, "name": "Mouse"},
    {"id": 3, "name": "Keyboard"}
  ]
}
```

2) Nested Objects & Arrays

```
1 class Address { // lớp Con
2     final String city;
3     final String country;
4
5     Address({required this.city, required this.country});
6
7     factory Address.fromJson(Map<String, dynamic> json) {
8         return Address(
9             city: json['city'],
10            country: json['country'],
11        );
12    }
13
14    Map<String, dynamic> toJson() {
15        return {'city': city, 'country': country,};
16    }
17 }
18
19 class User { // lớp Cha
20     final int id;
21     final String name;
22     final Address address; // Nested object
23
24     User({required this.id, required this.name, required this.address});
25
26     factory User.fromJson(Map<String, dynamic> json) {
27         return User(
28             id: json['id'],
29             name: json['name'],
30             address: Address.fromJson(json['address']), // xử lý object con
31         ); // User
32     }
33
34     Map<String, dynamic> toJson() {
35         return {'id': id, 'name': name, 'address': address.toJson(),
36     };
37     }
38 }
```

```
41 void main() {
42     final jsonData = {
43         "id": 1,
44         "name": "Duy",
45         "address": {"city": "DN", "country": "Vietnam"}
46     };
47
48     // JSON → Object Dart
49     final user = User.fromJson(jsonData);
50
51     print('Tên: ${user.name}');
52     print('Thành phố: ${user.address.city}');
53
54     // Object Dart → JSON
55     print(user.toJson());
56 }
```

- Kết quả:

Tên: Duy

Thành phố: DN

{id: 1, name: Duy, address: {city: DN, country: Vietnam}}

3) json_annotation & build_runner

- **json_annotation:** Là thư viện định nghĩa các annotation (chú thích) giúp bạn đánh dấu các lớp và thuộc tính trong Dart để sinh tự động code chuyển đổi JSON ↔ Dart Object.
- **json_serializable:** Là gói mở rộng kết hợp với json_annotation để tự động sinh hàm fromJson() và toJson() dựa trên annotation bạn đặt.
- **build_runner:** Là công cụ chạy quá trình sinh code tự động (code generation).

3) json_annotation & build_runner


pubspec.yaml

```
dependencies:  
  json_annotation: ^4.9.0  
  
dev_dependencies:  
  build_runner: ^2.4.0  
  json_serializable: ^6.8.0
```

```
8  ✓ factory Product.fromJson(Map<String, dynamic> json) {  
9  ✓   return Product(  
10     id: json['id'],  
11     name: json['name'],  
12     price: (json['price']).toDouble(),  
13   );  
14 }  
15  
16  ✓ Map<String, dynamic> toJson() {  
17     return {'id': id, 'name': name, 'price': price};  
18 }
```

```
lib > T2 > annotation.dart > ...  
1  ✓ import 'package:json_annotation/json_annotation.dart';  
2  
3  part 'annotation.g.dart'; // file sinh tự động  
4  
5  @JsonSerializable()  
6  ✓ class Product {  
7     final int id;  
8     final String name;  
9     final double price;  
10  
11     Product({required this.id, required this.name, required this.price});  
12  
13     // JSON → Object  
14     factory Product.fromJson(Map<String, dynamic> json) ⇒ _$ProductFromJson(json);  
15  
16     // Object → JSON  
17     Map<String, dynamic> toJson() ⇒ _$ProductToJson(this);  
18 }
```

3) json_annotation & build_runner

```
lib > T2 >  annotation.g.dart > ...
1  // GENERATED CODE - DO NOT MODIFY BY HAND
2
3  part of 'annotation.dart';
4
5  // *****
6  // JsonSerializerGenerator
7  // *****
8
9  Product _$ProductFromJson(Map<String, dynamic> json) ⇒ Product(
10 |   id: (json['id'] as num).toInt(),
11 |   name: json['name'] as String,
12 |   price: (json['price'] as num).toDouble(),
13 | );
14
15 Map<String, dynamic> _$ProductToJson(Product instance) ⇒ <String, dynamic>{
16 |   'id': instance.id,
17 |   'name': instance.name,
18 |   'price': instance.price,
19 | };
```

4) Best practices for complex data structures

4.1. Nguyên tắc thiết kế Model trong dự án lớn

- **Độc lập và tách biệt:** Mỗi model chịu trách nhiệm quản lý một loại dữ liệu duy nhất.
- **Tái sử dụng:** Tránh viết lại các model tương tự; có thể tái sử dụng model con trong nhiều model cha.
- **Đặt tên rõ ràng:** Tên class nên phản ánh đúng ý nghĩa dữ liệu, ví dụ: UserProfile, BookDetails, OrderResponse.
- **Immutable (bất biến):** Sử dụng final cho các thuộc tính để tránh thay đổi dữ liệu ngoài ý muốn.

4) Best practices for complex data structures

4.2. Sử dụng @JsonKey, ignore, defaultValue, nullable

- **@JsonKey(name: 'custom_field')**: ánh xạ tên JSON khác với tên thuộc tính trong model.
- **ignore**: bỏ qua trường không cần serialize.
- **defaultValue**: gán giá trị mặc định nếu trường không có trong JSON.
- **nullable**: cho phép giá trị null.

```
@JsonSerializable()  
class Product {  
    @JsonKey(name: 'product_id')  
    final int id;  
  
    @JsonKey(defaultValue: 'No name')  
    final String name;  
  
    @JsonKey(ignore: true)  
    final bool isFavorite;  
  
    Product({required this.id,  
            required this.name,  
            this.isFavorite = false});  
}
```

4) Best practices for complex data structures

4.3. Chiến lược xử lý Null-safety

Flutter/Dart 2.12+ hỗ trợ null safety, giúp tránh lỗi runtime:

- Sử dụng ? cho biến có thể null: String? email;
- Dùng toán tử ?? để gán giá trị mặc định:
- Kết hợp late khi chắc chắn dữ liệu sẽ được khởi tạo sau.

```
String? email;    // có thể null
```

```
String displayName = name ?? "";
```

```
late String token;
```

4) Best practices for complex data structures

4.4. Tối ưu hiệu suất serialize / deserialize

- Tránh decode JSON nhiều lần: lưu trữ dữ liệu sau khi parse.
- Dùng const constructors để giảm chi phí khởi tạo đối tượng.
- Hạn chế nested JSON quá sâu, tránh ảnh hưởng hiệu năng parse.
- Ưu tiên sử dụng json_serializable vì sinh code native, nhanh hơn
dart:convert thủ công

THANK YOU

TÀI LIỆU THAM KHẢO

- [1] Flutter, “*JSON and serialization*,” Flutter Documentation, 2025.
- [2] Dart Dev Team, “*dart:convert library*,” Dart SDK Documentation, Google Developers, 2025.
- [3] Google Developers, “*Flutter JSON serialization with json_serializable*,” Codelab, Google, 2025.
- [4] M. Smith and L. Johnson, “*Data Modeling and Serialization in Flutter Applications*,” *Journal of Mobile Development*, vol. 8, no. 2, pp. 45–53, 2023.

BẢNG PHÂN CÔNG CÔNG VIỆC TRONG NHÓM

STT	HỌ VÀ TÊN	NHIỆM VỤ	KHỐI LƯỢNG
11	Ngô Trung Chinh	<ul style="list-style-type: none"> - Nghiên cứu yêu cầu và nguyên lý hoạt động của JSON Serialization trong Flutter. - Thiết kế các model class chính - Xử lý dữ liệu lồng nhau (nested objects) và mảng (arrays). - Viết và kiểm thử các phương thức fromJson() và toJson() cho từng model. 	50%
15	Nguyễn Hữu Duy	<ul style="list-style-type: none"> - Cài đặt và cấu hình các thư viện json_annotation, build_runner. - Tạo và quản lý code sinh tự động bằng json_serializable. - Thực hiện kiểm thử quá trình serialize/deserialize với dữ liệu mẫu. - Ứng dụng best practices trong việc quản lý model phức tạp, tổ chức thư mục 	50%

Link code github: <https://github.com/huuduy26/LTDNT-DUYCHINH-10>