TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN TỬ - VIỄN THÔNG

Báo cáo tuần

Môn: Lập trình Đa nền tảng

JSON Serialization và Model Classes

GVHD: TS.Nguyễn Duy Nhật Viễn

SVTH:

1.Trương Vĩnh Thiện - 22KTMT1 - 106220235

2.Thân Công Đức - 22KTMT2 - 106220249

Đà Nẵng, 2025

Nội dung báo cáo

1. Cơ sở lý thuyết
2. Tạo model classes với toJson() và fromJson()
3. Xử lý nested objects và arrays
4. Sử dụng json_annotation và build_runner
5. Best Practices cho Complex Data Structures trong Flutter

BẢNG PHÂN CÔNG CÔNG VIỆC TRONG NHÓM

STT	HỌ VÀ TÊN	NHIỆM VỤ	KHỐI LƯỢNG
34	Trương Vĩnh Thiện	- Tìm hiểu yêu cầu bài toán JSON Serialization - Thiết kế các model classes chính (ví dụ: User, Book, Order) - Xử lý nested objects và arrays - Viết code fromJson() và toJson() cho model	50%
48	Thân Công Đức	- Cài đặt json_annotation, build_runner - Sinh code tự động với json_serializable - Kiểm thử serialize / deserialize với dữ liệu mẫu - Áp dụng best practices cho complex data structures (immutable, naming, folder structure)	50%

Link code github: https://github.com/tvthien-ktmt/B-o-c-o-l-p-tr-nh-a-n-n-t-ng-t-i-2

1.Cơ sở lý thuyết

1.1 JSON là gì?

- JSON (JavaScript Object Notation) là một định dạng trao đổi dữ liệu nhẹ, dễ đọc cho con người và dễ phân tích cho máy tính.
- JSON được sử dụng rộng rãi trong các ứng dụng web và mobile để truyền dữ liệu giữa client và server.

Cấu trúc JSON bao gồm:

Object: Tập hợp các cặp key–value:

```
1  { "name": "John", "age": 25, "city": "New York" }
```

1.1 JSON là gì?

Array: Danh sách giá trị, ví dụ:

```
1  ["Flutter", "Dart", "Firebase"]
```

Đặc điểm của JSON:

- Độc lập ngôn ngữ (ngôn ngữ nào cũng đọc được)
- Dễ phân tích cú pháp
- Dạng text, dễ truyền qua HTTP

1.2. JSON Serialization trong Dart/Flutter

- JSON Serialization là quá trình chuyển đổi qua lại giữa dữ liệu JSON và đối tượng Dart trong Flutter.
- Giúp ứng dụng hiểu, xử lý và trao đổi dữ liệu với server, API hoặc file lưu trữ một cách dễ dàng và nhất quán.

Ứng dụng:

- Giao tiếp client–server
- Lưu dữ liệu local
- Truyền thông giữa các tiến trình

1.2. JSON Serialization trong Dart/Flutter

Hai hướng chính:

Serialization (Object → JSON)

- Gửi dữ liệu lên server (API)
- Lưu cục bộ (file, SharedPreferences)
- Dùng cho cache hoặc tiến trình nội bộ

Deserialization (JSON → Object)

- Hiển thị dữ liệu lên UI
- Xử lý, tính toán trong ứng dụng
- Lưu vào cơ sở dữ liệu nội bộ

1.3. Tại sao cần JSON Serialization

- Trong ứng dụng Flutter hiện đại, phần lớn dữ liệu đến từ backend server ở dạng JSON string.
- Flutter không thể làm việc trực tiếp với chuỗi JSON, vì nó chỉ là text.
- Cần chuyển đổi JSON → Object Dart (Model Class) để làm việc thuận tiện, an toàn và hiệu quả hơn.

Lợi ích của JSON Serialization:

- Dễ quản lý & truy cập dữ liệu
- An toàn kiểu dữ liệu (Type Safety)
- Dễ bảo trì, mở rộng khi thay đổi API
- Hỗ trợ kiểm thử (Testing)
- Tăng khả năng tái sử dụng (Reusability)

2.Tạo model classes với toJson() và fromJson()

Khái niệm:

- Model Class (Data Model) là lớp mô tả cấu trúc dữ liệu mà ứng dụng Flutter làm việc với.
- Là lớp trung gian giữa dữ liệu thô (JSON, Map) và dữ liệu có cấu trúc trong code.
- Dùng để truy cập, xử lý, lưu trữ dữ liệu dễ dàng và an toàn.

Cấu trúc cơ bản của một Model Class:

- Fields: Các thuộc tính mô tả dữ liệu (thường khai báo final).
- Constructor: Dùng để khởi tạo đối tượng.
- fromJson(): Chuyển JSON → Object (deserialization).
- toJson(): Chuyển Object → JSON (serialization).

Ví dụ minh hoạ:

```
Tên sản phẩm: Laptop Acer
Giá: 15500000.0
Dữ liệu JSON sau khi xuất: {id: P001, name: Laptop Acer, price: 15500000.0}
```

```dart
class Product {
  final String id;
  final String name;
  final double price;

  Product({
    required this.id,
    required this.name,
    required this.price,
  });

  // JSON → Object
  factory Product.fromJson(Map<String, dynamic> json) {
    return Product(
      id: json['id'],
      name: json['name'],
      price: json['price'],
    );
  }

  // Object → JSON
  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'name': name,
      'price': price,
    };
  }
}

void main() {
  // 1 JSON dữ liệu đầu vào
  final jsonData = {'id': 'P001', 'name': 'Laptop Acer', 'price': 15500000.0};

  // 2 JSON → Object
  final product = Product.fromJson(jsonData);
  print('Tên sản phẩm: ${product.name}');
  print('Giá: ${product.price}');

  // 3 Object → JSON
  final jsonOutput = product.toJson();
  print('Dữ liệu JSON sau khi xuất: $jsonOutput');
}
```

Các lỗi thường gặp

Lỗi	Nguyên nhân	Cách khắc phục
type 'String' is not a subtype of type 'int'	Kiểu dữ liệu JSON không khớp với model	Kiểm tra và ép kiểu đúng từng trường
Null check operator used on a null value	API trả về null nhưng model không cho phép	Dùng kiểu nullable (String?) hoặc giá trị mặc định
NoSuchMethodError: '[]' was called on null	JSON thiếu key hoặc cấu trúc sai	Kiểm tra key bằng json.containsKey()
JSON key khác tên biến	Tên key trong JSON ≠ tên thuộc tính model	Dùng @JsonKey(name: 'json_key')
Dữ liệu không đồng nhất	API trả về dữ liệu sai hoặc không ổn định	Dùng try-catch hoặc toán tử ?? để xử lý

3. Xử lý Nested Objects và Arrays

Trong Flutter/Dart, dữ liệu JSON không chỉ gồm các kiểu cơ bản như String, int, double.

Thực tế, dữ liệu thường chứa:

- Đối tượng lồng nhau (Nested Objects)
- Danh sách đối tượng (Arrays of Objects)

Giải thích:

- Đây là hai dạng cấu trúc phổ biến trong các API hiện đại.
- Giúp mô tả dữ liệu rõ ràng, có tổ chức, nhưng khiến quá trình xử lý JSON phức tạp hơn.

Ví dụ, dữ liệu sách có thể chứa thông tin về nhà xuất bản (Publisher), tác giả, hoặc danh sách chương — tất cả đều là các đối tượng lồng nhau.

3.1. Nested Objects

- Nested Object: Khi một thuộc tính của object chứa một object khác.
- Thường gặp trong API phức tạp → ví dụ: User chứa Address, Profile, Company.
- Ưu điểm: Mô tả dữ liệu rõ ràng, có tổ chức.
- Nhược điểm: Làm quá trình serialize / deserialize trở nên phức tạp hơn.

Ví dụ:

```json
{
  "name": "Nguyen Van A",
  "email": "a106@gmail.com",
  "address": {
    "city": "DaNang",
    "street": "Nguyen Van Linh",
    "zipcode": "106220111"
  }
}
```

- User là đối tượng chính.
- address là một nested object, chứa 3 trường: city, street, zipcode

Ví dụ minh họa

```dart
// Model con: Address
class Address {
  final String city;
  final String street;
  Address({
    required this.city,
    required this.street,
  });
  factory Address.fromJson(Map<String, dynamic> json) {
    return Address(
      city: json['city'],
      street: json['street'],
    );
  }
  Map<String, dynamic> toJson() {
    return {
      'city': city,
      'street': street,
    };
  }
}
// Model cha: User
class User {
  final String name;
  final Address address; // Nested Object

  User({
    required this.name,
    required this.address,
  });
  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      name: json['name'],
      address: Address.fromJson(json['address']),
    );
  }
  Map<String, dynamic> toJson() {
    return {
      'name': name,
      'address': address.toJson(),
    };
  }
}
```

```dart
void main() {
  // JSON đầu vào
  final jsonData = {
    'name': 'Nguyen Van A',
    'address': {'city': 'Hanoi', 'street': 'Tran Hung Dao'}
  };

  // JSON → Object
  final user = User.fromJson(jsonData);
  print('Tên người dùng: ${user.name}');
  print('Đường: ${user.address.street}');
  print('Thành phố: ${user.address.city}');

  // Object → JSON
  final output = user.toJson();
  print('JSON xuất ra: $output');
}
```

Kết quả:

```
THIEN@MSI MINGW64 /d/Learn Dart Flutter
$ dart run main2.dart
Tên người dùng: Nguyen Van A
Đường: Tran Hung Dao
Thành phố: Hanoi
JSON xuất ra: {name: Nguyen Van A, address: {city: Hanoi, street: Tran Hung Dao}}
```

3.2. Làm việc với danh sách (List / Array)

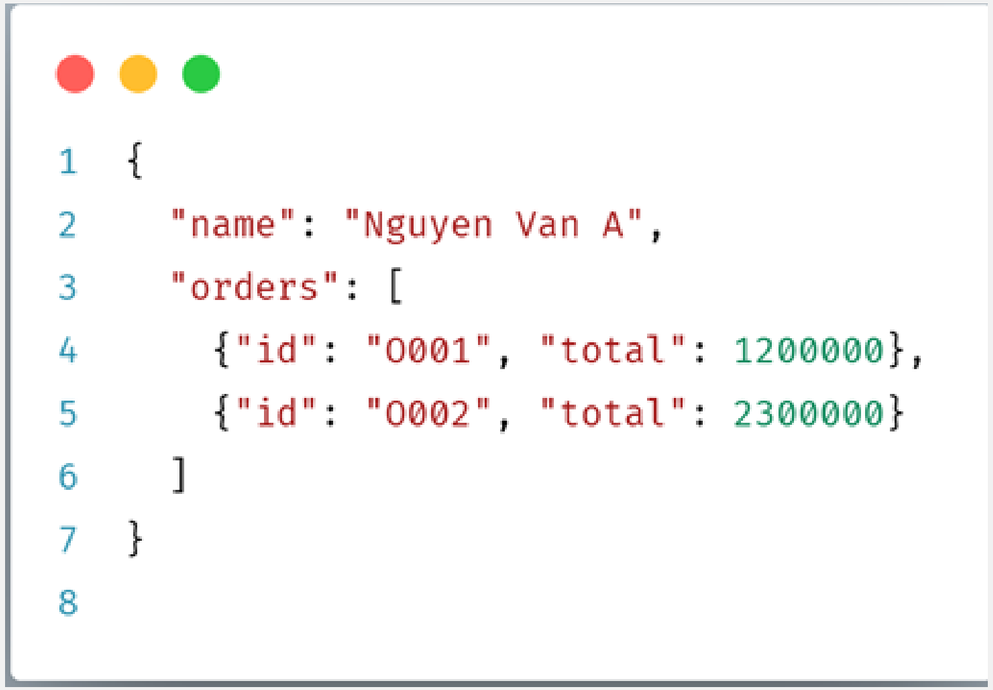
Array (List) trong JSON là danh sách nhiều phần tử, mỗi phần tử có thể là:

- Giá trị đơn giản: int, String, bool, ...
- Object phức tạp: Order, Product, User, ...

Trong Dart, danh sách được biểu diễn bằng:

List<T> – trong đó T là kiểu dữ liệu của từng phần tử.

Ví dụ JSON có mảng object:



```
{
  "name": "Nguyen Van A",
  "orders": [
    {"id": "O001", "total": 1200000},
    {"id": "O002", "total": 2300000}
  ]
}
```

orders là một mảng object, mỗi phần tử là một đơn hàng (Order).

Ví dụ minh hoạ:

```dart
// Model con: Order
class Order {
  final String id;
  final double total;
  Order({required this.id, required this.total});
  factory Order.fromJson(Map<String, dynamic> json) {
    return Order(
      id: json['id'],
      total: json['total'].toDouble(),
    );
  }
  Map<String, dynamic> toJson() {
    return {'id': id, 'total': total};
  }
}
// Model cha: User
class User {
  final String name;
  final List<Order> orders; // Danh sách object

  User({required this.name, required this.orders});

  factory User.fromJson(Map<String, dynamic> json) {
    var list = json['orders'] as List;
    List<Order> orderList = list.map((e) => Order.fromJson(e)).toList();
    return User(name: json['name'], orders: orderList);
  }
  Map<String, dynamic> toJson() {
    return {
      'name': name,
      'orders': orders.map((e) => e.toJson()).toList(),
    };
  }
}
```

```dart
void main() {
  final jsonData = {
    'name': 'Nguyen Van A',
    'orders': [
      {'id': 'O001', 'total': 1200000},
      {'id': 'O002', 'total': 2300000}
    ]
  };

  // JSON → Object
  final user = User.fromJson(jsonData);
  print('Tên: ${user.name}');
  print('Số đơn hàng: ${user.orders.length}');
  print('Mã đơn đầu tiên: ${user.orders[0].id}');

  // Object → JSON
  print('JSON xuất ra: ${user.toJson()}');
}
```

Kết quả:

```
$ dart run main3.dart
Tên: Nguyen Van A
Số đơn hàng: 2
Mã đơn đầu tiên: O001
JSON xuất ra: {name: Nguyen Van A, orders: [{id: O001, total: 1200000.0}, {id: O002, total: 2300000.0}]}
```

Các lỗi thường gặp:

Lỗi	Nguyên nhân	Cách khắc phục
Null check operator used on a null value	JSON thiếu object hoặc danh sách (json['address'], json['orders'] bị null)	Kiểm tra null trước khi parse, hoặc dùng kiểu nullable (Address?, List<Order>?)
type 'List' is not a subtype of type 'List'	Chuyển List JSON sang List<Model> sai cách	Dùng .map((e) => Model.fromJson(e)).toList() khi parse danh sách
type 'String' is not a subtype of type 'int'	Kiểu dữ liệu giữa JSON và model không khớp	Kiểm tra lại kiểu, hoặc ép kiểu bằng toString() / toInt()
NoSuchMethodError: '[]' was called on null	Object hoặc danh sách con không tồn tại	Kiểm tra key bằng json.containsKey('key') trước khi truy cập
Data inconsistency *(sai cấu trúc dữ liệu)*	JSON trả về sai định dạng	Kiểm tra dữ liệu thực tế từ API trước khi parse

4. Sử dụng json_annotation và build_runner

Với dự án nhỏ, có thể viết thủ công fromJson() và toJson().

Nhưng khi dự án lớn với nhiều model, cách này:

- Dễ gây lỗi khi quên ánh xạ dữ liệu
- Khó bảo trì khi API thay đổi
- Tốn thời gian, lặp lại thao tác

Giải pháp:

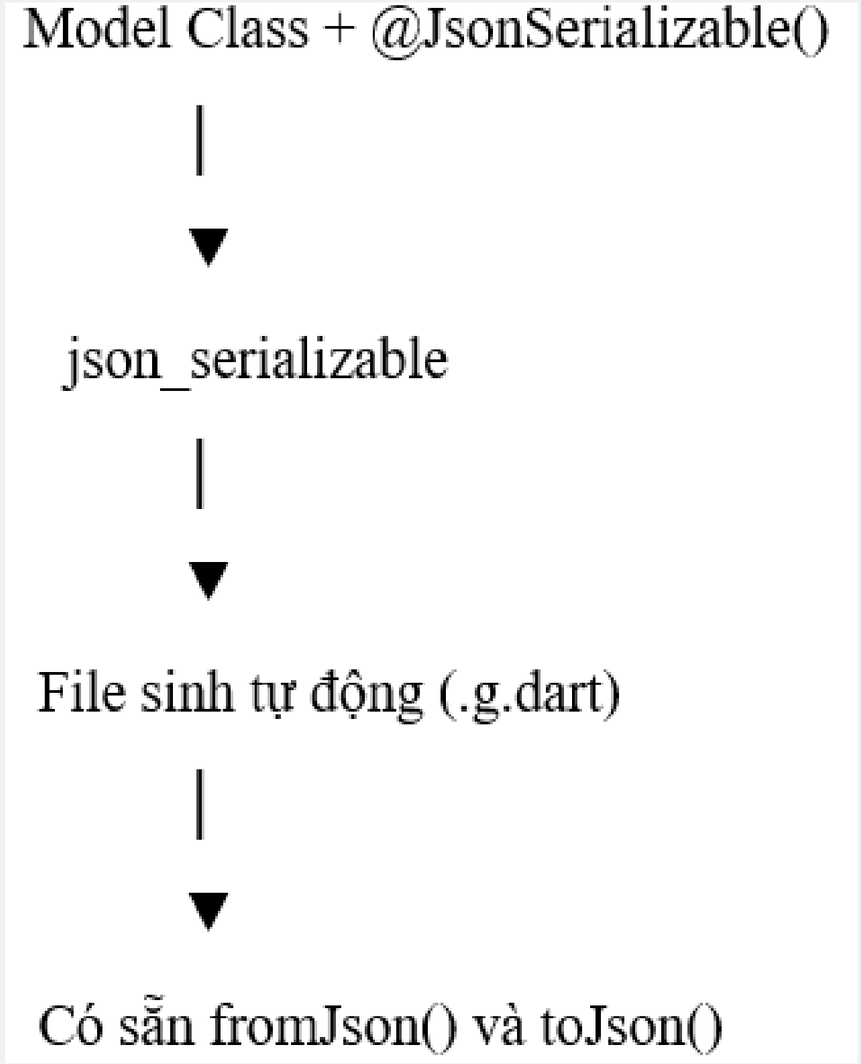
Dart hỗ trợ cơ chế Code Generation (tự động sinh mã) thông qua 3 thư viện:

Thư viện	Vai trò	Mô tả ngắn gọn
json_annotation	Định nghĩa annotation	Cung cấp @JsonSerializable(), @JsonKey() để quy định serialize / deserialize
json_serializable	Trình sinh code	Tự động tạo fromJson() & toJson()
build_runner	Chạy code generator	Sinh mã bằng lệnh dart run build_runner build

4.1. Cấu trúc hoạt động của json_serializable

Ý nghĩa:

- Giúp tránh lỗi thủ công
- Đồng bộ dữ liệu giữa JSON ↔ Dart Object
- Tăng tốc độ phát triển dự án



4.2. Hướng dẫn cài đặt (Bước 1 & 2)

Bước 1: Cập nhật pubspec.yaml

```yaml
dependencies:
  json_annotation: ^4.9.0

dev_dependencies:
  build_runner: ^2.4.6
  json_serializable: ^6.8.0
```

Bước 2: Tạo model và thêm annotation

```dart
import 'package:json_annotation/json_annotation.dart';

part 'book.g.dart';

@JsonSerializable()
class Book {
  final int id;
  final String title;
  final String author;

  Book({
    required this.id,
    required this.title,
    required this.author,
  });

  factory Book.fromJson(Map<String, dynamic> json) => _$BookFromJson(json);
  Map<String, dynamic> toJson() => _$BookToJson(this);
}
```

4.2. Hướng dẫn cài đặt (Bước 3 & 4)

Bước 3: Chạy lệnh sinh code:dart run build_runner build

File book.g.dart sẽ chứa:

```dart
Book _$BookFromJson(Map<String, dynamic> json) => Book(
      id: json['id'] as int,
      title: json['title'] as String,
      author: json['author'] as String,
    );

Map<String, dynamic> _$BookToJson(Book instance) => <String, dynamic>{
      'id': instance.id,
      'title': instance.title,
      'author': instance.author,
    };
```

Bước 4: Sử dụng model

```dart
import 'dart:convert';
import 'book.dart';

void main() {
  // JSON chuỗi đầu vào
  String jsonStr = '{"id":101,"title":"Dart Mastery","author":"Duc Than"}';

  // JSON → Object
  Book book = Book.fromJson(jsonDecode(jsonStr));
  print('Title: ${book.title}, Author: ${book.author}');

  // Object → JSON
  String outputJson = jsonEncode(book.toJson());
  print('JSON Output: $outputJson');
}
```

4.3. Lỗi thường gặp & Cách khắc phục

Lỗi	Nguyên nhân	Cách khắc phục
Missing part 'model.g.dart'	Chưa khai báo part hoặc sai tên file	Đảm bảo part 'model.g.dart'; trùng tên file
NoSuchMethodError: _BookFromJson	Chưa chạy build_runner	Chạy lệnh dart run build_runner build
FormatException	JSON không hợp lệ	Kiểm tra JSON bằng jsonDecode() hoặc công cụ online
Bad state: cannot set the same part multiple times	Trùng khai báo part	Mỗi model chỉ có một part duy nhất
The method _$BookFromJson isn't defined	File .g.dart chưa sinh hoặc bị xóa	Xóa file lỗi và chạy lại build_runner

5. Best Practices cho Complex Data Structures trong Flutter

5.1. Nguyên tắc thiết kế Model trong dự án lớn

- Độc lập & tách biệt: Mỗi model chỉ quản lý 1 loại dữ liệu duy nhất.
- Tái sử dụng: Dùng lại model con trong nhiều model cha.
- Đặt tên rõ ràng: Ví dụ UserProfile, BookDetails, OrderResponse.
- Immutable (bất biến): Dùng final để tránh thay đổi dữ liệu ngoài ý muốn.

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

Giúp xử lý dữ liệu JSON linh hoạt và an toàn hơn:

- @JsonKey(name: 'custom_field') → Đổi tên key JSON.
- ignore → Bỏ qua trường không cần serialize.
- defaultValue → Gán giá trị mặc định khi thiếu dữ liệu.
- nullable → Cho phép giá trị null.

Ví dụ:

```dart
@JsonSerializable()
class User {
  @JsonKey(name: 'user_id')
  final int id;
  final String name;
  @JsonKey(defaultValue: 'Unknown')
  final String? email;

  const User({
    required this.id,
    required this.name,
    this.email,
  });

  factory User.fromJson(Map<String, dynamic> json) => _$UserFromJson(json);
  Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

5.3. Chiến lược xử lý Null-safety

- Dùng ? cho biến có thể null → String? email;
- Dùng ?? để gán giá trị mặc định →
- print(user.email ?? "Không có email");
- Dùng late khi chắc chắn dữ liệu được khởi tạo sau.

5.4. Tối ưu hiệu suất serialize / deserialize

- Tránh decode nhiều lần → Lưu kết quả sau khi parse.
- Dùng const constructors để giảm chi phí khởi tạo.
- Hạn chế nested JSON quá sâu để tăng tốc độ parse.
- Ưu tiên json_serializable → Sinh code native, nhanh hơn dart:convert.

TÀI LIỆU THAM KHẢO

[1]. Flutter Team, “JSON and Serialization,” Flutter Documentation, Google, 2024.
[2]. Viblo Community, “Làm việc với JSON & Serialization trong Flutter,” Viblo.asia, 2023.
[3]. Raywenderlich Team, Flutter Apprentice (Third Edition): Learn to Build Cross-Platform Apps, Razeware LLC, 2023.

Trương Vĩnh Thiện

Thân Công Đức

Cảm ơn thầy cô và các bạn đã chú ý lắng nghe