**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN TỬ VIỄN THÔNG**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**JSON Serialization và Model Classes**

**Sinh viên thực hiện:**

**01.Trương Vĩnh Thiện Lớp: 22KTMT1**  **MSSV:** 106220235

**02. Thân Công Đức** **Lớp: 22KTMT2 MSSV:**  106220249

**Người hướng dẫn:**

**TS. Nguyễn Duy Nhật Viễn**

**Đà Nẵng, 2025.**

**THUYẾT MINH**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**JSON Serialization và Model Classes**

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/BaoCaoLapTrinhDaNenTangDT10>

**Mục lục**

Nội dung

[1. Cơ sở lý thuyết 4](#_Toc212397752)

[1.1 JSON là gì? 4](#_Toc212397753)

[1.2. JSON Serialization trong Dart/Flutter 4](#_Toc212397754)

[1.3. Tại sao cần JSON Serialization 5](#_Toc212397755)

[2.   Tạo model classes với toJson() và fromJson() 5](#_Toc212397756)

[2.1. Khái niệm Model Class 5](#_Toc212397757)

[2.2. Cấu trúc của một Model Class 5](#_Toc212397758)

[2.3. Ví dụ cơ bản và kết quả 6](#_Toc212397759)

[2.4. Các lỗi thường gặp 6](#_Toc212397760)

[3. Xử lý nested objects và arrays 7](#_Toc212397761)

[3.1. Nested Objects (Đối tượng lồng nhau) 7](#_Toc212397762)

[3.2. Ví dụ minh họa 9](#_Toc212397763)

[3.3. Làm việc với danh sách (List / Array) 10](#_Toc212397764)

[3.4. Ví dụ minh họa và kết quả: 11](#_Toc212397765)

[4. Sử dụng json\_annotation và build\_runner 12](#_Toc212397766)

[4.1 Cấu trúc hoạt động 13](#_Toc212397767)

[4.2 Hướng dẫn cài đặt 13](#_Toc212397768)

[4.3 Lỗi thường gặp & cách khắc phục 15](#_Toc212397769)

[5. Best Practices cho Complex Data Structures trong Flutter 16](#_Toc212397770)

[5.1. Nguyên tắc thiết kế Model trong dự án lớn 16](#_Toc212397771)

[5.2. Sử dụng @JsonKey, ignore, defaultValue, nullable 16](#_Toc212397772)

[5.3. Chiến lược xử lý Null-safety 17](#_Toc212397773)

[5.4. Tối ưu hiệu suất serialize / deserialize 17](#_Toc212397774)

**NỘI DUNG**

# 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:*

*Ảnh có chứa văn bản, danh thiếp, ảnh chụp màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.*

**Array:** Danh sách giá trị, ví dụ:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, danh thiếp

Nội dung do AI tạo ra có thể không chính xác.

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 ứng dụng Flutter.
* Nói cách khác, đây là cơ chế giúp ứng dụng hiểu và xử lý dữ liệu mà các hệ thống khác (như server, API, hoặc file lưu trữ) gửi đến hoặc nhận từ Flutter.

Quá trình này bao gồm hai hướng chính:

+ Serialization: Là việc chuyển đổi một đối tượng (object) trong Dart thành chuỗi JSON để có thể:

* + Gửi dữ liệu đó lên server thông qua API.
  + Lưu trữ dữ liệu cục bộ (local storage, file, SharedPreferences, v.v.).
  + Giao tiếp giữa các tiến trình hoặc lưu cache tạm thời.

+ Deserialization: Là việc chuyển đổi dữ liệu JSON (thường là chuỗi string nhận từ server) trở lại thành đối tượng Dart (model class), để có thể:

* + Hiển thị thông tin lên giao diện (UI).
  + Thao tác, tính toán, hoặc xử lý dữ liệu.
  + Lưu trữ vào cơ sở dữ liệu nội bộ.

## 1.3. Tại sao cần JSON Serialization

* Trong các ứng dụng Flutter hiện đại, phần lớn dữ liệu đến từ backend server thông qua API ở dạng JSON string.
* Tuy nhiên, Flutter không thể làm việc trực tiếp với chuỗi JSON này, vì nó chỉ là text.  
  Do đó, cần có bước chuyển đổi dữ liệu JSON thành các đối tượng Dart (hay còn gọi là Model Classes) để làm việc thuận tiện và an toàn hơn.

Việc sử dụng JSON Serialization mang lại nhiều lợi ích rõ rệt:

* Dễ quản lý và truy cập dữ liệu
* Đảm bảo an toàn kiểu dữ liệu (Type Safety)
* Dễ bảo trì và mở rộng
* Hỗ trợ kiểm thử (Testing)
* Tăng khả năng tái sử dụng (Reusability)

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

## 2.1. Khái niệm Model Class

Model Class (hay còn gọi là Data Model) là một lớp (class) trong Dart được dùng để mô tả cấu trúc dữ liệu mà ứng dụng làm việc với — thường là dữ liệu nhận được từ server (API) hoặc lưu trữ trong cơ sở dữ liệu nội bộ.

Nói cách khác, Model Class là lớp trung gian giữa dữ liệu dạng thô (như JSON, Map) và dữ liệu có cấu trúc mà code có thể hiểu, truy cập, và kiểm soát được trong Flutter.

## 2.2. Cấu trúc của một Model Class

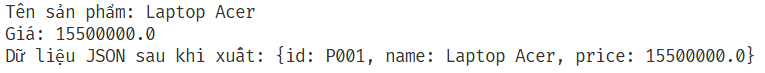
Một model class đầy đủ trong Dart bao gồm các thành phần sau:

* Thuộc tính (Fields): Khai báo các biến mô tả dữ liệu của đối tượng (thường là final để đảm bảo bất biến).
* Constructor: Hàm khởi tạo đối tượng, dùng để gán giá trị cho các thuộc tính.
* fromJson(): Phương thức (thường là factory constructor) dùng để chuyển JSON → Object (deserialization).
* toJson(): Phương thức dùng để chuyển Object → JSON (serialization).

## 2.3. Ví dụ cơ bản và kết quả

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Nội dung do AI tạo ra có thể không chính xác.

Kết quả: 

## 2.4. Các lỗi thường gặp

|  |  |  |
| --- | --- | --- |
| **Lỗi thường gặp** | **Nguyên nhân** | **Cách khắc phục** |
| type 'String' is not a subtype of type 'int' | Kiểu dữ liệu giữa JSON và model không trùng khớp | Kiểm tra và ép kiểu đúng với từng trường trong model |
| Null check operator used on a null value | API trả về giá trị null nhưng model không cho phép null | Khai báo thuộc tính dạng nullable (ví dụ: String?) hoặc cung cấp giá trị mặc định |
| NoSuchMethodError: '[]' was called on null | JSON không chứa key tương ứng hoặc cấu trúc bị sai | Kiểm tra sự tồn tại của key bằng json.containsKey() trước khi truy cập |
| JSON key khác tên biến trong model | Tên key trong JSON không trùng với tên thuộc tính của model | Sử dụng @JsonKey(name: 'json\_key') khi dùng thư viện json\_serializable |
| Dữ liệu không đồng nhất (null hoặc sai kiểu) | Dữ liệu từ API không nhất quán giữa các bản ghi | Sử dụng try-catch hoặc toán tử ?? để xử lý giá trị null và tránh lỗi ép kiểu |

# 3. Xử lý nested objects và arrays

Trong quá trình làm việc với dữ liệu JSON trong Flutter/Dart, không phải lúc nào dữ liệu cũng chỉ bao gồm các kiểu dữ liệu cơ bản như String, int, hay double.  
Rất thường xuyên, dữ liệu sẽ bao gồm các đối tượng lồng nhau (nested objects) hoặc các danh sách (arrays) chứa nhiều phần tử.  
Đây là hai dạng cấu trúc phổ biến trong các API hiện đại:

* Một object chứa object khác bên trong (Nested Object)
* Một object chứa danh sách các object con (Array of Objects)

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 (Đối tượng lồng nhau)

* Nested Object (đối tượng lồng nhau) là trường hợp trong JSON mà một thuộc tính của object chính lại là một object khác. Tình huống này thường gặp trong các API phức tạp, ví dụ khi một User chứa một Address, một Profile, hoặc một Company.
* Cấu trúc dạng này giúp mô tả dữ liệu rõ ràng và có tổ chức, nhưng cũng khiến quá trình deserialize (JSON → Dart object) và serialize (Dart object → JSON) phức tạp hơn.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, thiết kế

Nội dung do AI tạo ra có thể không chính xác.

Ở đây:

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

## 3.2. Ví dụ minh họa

|  |  |
| --- | --- |
| Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, số  Nội dung do AI tạo ra có thể không chính xác. | Ảnh có chứa văn bản, đồ điện tử, ảnh chụp màn hình, màn hình  Nội dung do AI tạo ra có thể không chính xác. |

Kết quả:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.

|  |  |
| --- | --- |
| **Bước** | **Mô tả chi tiết** |
| **Khởi tạo dữ liệu JSON** | Biến jsonData chứa dữ liệu JSON đầu vào gồm 2 phần: thông tin người dùng (name) và thông tin địa chỉ (address) – trong đó address là một object con. |
| **Gọi User.fromJson()** | Hàm User.fromJson() nhận jsonData và tạo ra một object User. Khi gặp trường address, nó tiếp tục gọi Address.fromJson(json['address']) để tạo object Address bên trong. |
| **Tạo object con Address** | Hàm Address.fromJson() đọc các trường city và street từ JSON con, sau đó khởi tạo object Address. Object này được gán vào thuộc tính address của User. |
| **Truy cập dữ liệu** | Sau khi parse xong, ta có thể truy cập các trường lồng nhau như user.address.street hoặc user.address.city một cách trực tiếp, rõ ràng và an toàn kiểu dữ liệu. |
| **Chuyển ngược lại JSON (toJson())** | Khi gọi user.toJson(), hàm toJson() của User sẽ tạo một Map mới chứa các trường của User, trong đó trường address được xử lý bằng cách gọi address.toJson(). Kết quả là JSON xuất ra giống cấu trúc ban đầu. |

## 3.3. Làm việc với danh sách (List / Array)

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

* Giá trị đơn giản (int, String, bool, …)
* Hoặc object phức tạp (ví dụ danh sách Order, Product, User, …)

Trong Dart, ta biểu diễn bằng kiểu List<T> — trong đó T là kiểu dữ liệu của từng phần tử.

Ví dụ JSON có mảng object:

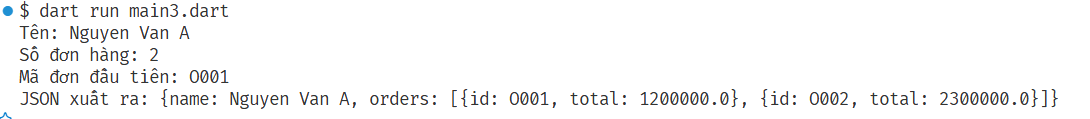
Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Nội dung do AI tạo ra có thể không chính xác.

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

## 3.4. Ví dụ minh họa và kết quả:

|  |  |
| --- | --- |
| Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, màn hình  Nội dung do AI tạo ra có thể không chính xác. | Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, phần mềm  Nội dung do AI tạo ra có thể không chính xác. |

Kết quả: 

* orders trong JSON là một danh sách object.
* Trong User.fromJson(), ta:
  + Ép kiểu json['orders'] sang List
  + Duyệt qua từng phần tử (map) → gọi Order.fromJson() để chuyển từng JSON con thành object Order
* Khi xuất JSON (toJson()), ta gọi orders.map((e) => e.toJson()).toList() để chuyển ngược danh sách object thành danh sách JSON.
* Kết quả: dữ liệu được chuyển đổi hai chiều chính xác giữa List JSON ↔ List Object Dart.

|  |  |  |
| --- | --- | --- |
| **Lỗi thường gặp** | **Nguyên nhân** | **Cách khắc phục** |
| **Null check operator used on a null value** | JSON không chứa object hoặc danh sách cần thiết (json['address'] hoặc json['orders'] bị null) | Kiểm tra null trước khi gọi fromJson(), hoặc dùng kiểu nullable (Address?, List<Order>?) |
| **type 'List' is not a subtype of type 'List'** | Không ép kiểu hoặc map đúng cách khi chuyển List JSON sang List<Model> | Dùng .map((e) => Model.fromJson(e)).toList() khi parse danh sách |
| **type 'String' is not a subtype of type 'int'** | JSON và model không khớp kiểu dữ liệu giữa các trường | Kiểm tra lại kiểu trong model, hoặc dùng toString() / toInt() để ép kiểu phù hợp |
| **NoSuchMethodError: '[]' was called on null** | Object hoặc danh sách con không tồn tại trong JSON | Kiểm tra key bằng json.containsKey('key') trước khi truy cập |
| **Data inconsistency (sai cấu trúc dữ liệu)** | JSON trả về mảng hoặc object không đúng định dạng mong đợi | Kiểm tra response thực tế từ API, xác thực kiểu dữ liệu trước khi parse |

# 4. Sử dụng json\_annotation và build\_runner

Trong các dự án nhỏ, việc tự viết các hàm fromJson() và toJson() cho mỗi model là khả thi.  
Tuy nhiên, khi dự án phát triển lớn hơn với hàng chục hoặc hàng trăm model, cách làm này sẽ:

* Dễ gây lỗi logic khi quên ánh xạ trường dữ liệu.
* Khó bảo trì khi API thay đổi.
* Lặp lại công việc thủ công, tốn thời gian.

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

* json\_annotation
* json\_serializable
* build\_runner

Các thư viện này giúp phân tích cấu trúc model và tự động sinh code fromJson() và toJson() một cách chuẩn xác, đồng nhất, và hiệu năng cao.

|  |  |  |
| --- | --- | --- |
| **Thư viện** | **Vai trò** | **Mô tả chi tiết** |
| json\_annotation | Định nghĩa annotation | Cung cấp các annotation như @JsonSerializable(), @JsonKey(), giúp chỉ định quy tắc serialize/deserialize. |
| build\_runner | Công cụ chạy code generator | Thực thi quá trình sinh mã bằng lệnh dart run build\_runner build. |

## 4.1 Cấu trúc hoạt động

Model Class + @JsonSerializable()

│

▼

json\_serializable

│

▼

File sinh tự động (.g.dart)

│

▼

Có sẵn fromJson() và toJson()

## 4.2 Hướng dẫn cài đặt

**Bước 1: Cập nhật pubspec.yaml**

**Ảnh có chứa văn bản, ảnh chụp màn hình, danh thiếp, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.**

**Bước 2: Tạo model và thêm annotation**

**Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Nội dung do AI tạo ra có thể không chính xác.**

**Giải thích:**

* @JsonSerializable() → báo cho build generator sinh code.
* part 'book.g.dart' → khai báo file .g.dart để chèn code tự động.
* fromJson() và toJson() → gọi code sinh sẵn.

**Bước 3: Chạy lệnh sinh code**

**Ảnh có chứa văn bản, danh thiếp, ảnh chụp màn hình, Phông chữ

Nội dung do AI tạo ra có thể không chính xác.**

**File book.g.dart sẽ chứa:**

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, phần mềm

Nội dung do AI tạo ra có thể không chính xác.

**Bước 4: Sử dụng model**

**Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Hệ điều hành

Nội dung do AI tạo ra có thể không chính xác.**

## 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 dart run build\_runner build |
| Unhandled Exception: 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 tương ứng |
| 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

Khi ứng dụng phát triển với nhiều loại dữ liệu phức tạp, việc quản lý và tổ chức các model class cần tuân thủ các nguyên tắc sau:

* 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.

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

Các annotation trong json\_serializable giúp xử lý dữ liệu JSON linh hoạt hơn:

* @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.

**Ví dụ:**

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Nội dung do AI tạo ra có thể không chính xác.

## 5.3. 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:

print(user.email ?? "Không có email");

* Kết hợp late khi chắc chắn dữ liệu sẽ được khởi tạo sau.

## 5.4. 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

**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.