

JSON SERIALIZATION VÀ MODEL CLASSES

Trình bày:
Lê Duy Hoàng
Nguyễn Trọng Nhân
GVHD: Nguyễn Duy Nhật Viễn



PHÂN CÔNG CÔNG VIỆC

STT	HỌ VÀ TÊN	NHIỆM VỤ	KHÔI LUỢNG
01	NGUYỄN TRỌNG NHÂN	Tạo model classes với toJson() và fromJson() Xử lý nested objects và arrays	50%
02	LÊ DUY HOÀNG	Sử dụng json_annotation và build_runner Best practices cho complex data structures	50%

NỘI DUNG

01

Giới thiệu

02

Phương pháp thủ công (Manual Serialization)

03

Phương pháp tự động sinh mã(Automated serialization using code generation)

04

Các phương pháp tốt nhất khi làm việc với
cấu trúc dữ liệu phức tạp

1. GIỚI THIỆU



- **JSON (JavaScript Object Notation)**, là một định dạng dữ liệu dạng văn bản dùng để lưu trữ và trao đổi dữ liệu giữa các hệ thống.
- Nhờ tính đơn giản, dễ đọc và khả năng biểu diễn cấu trúc dữ liệu phức tạp.
=> Ưa chuộng rộng rãi

1. GIỚI THIỆU

Cấu trúc cơ bản của JSON: gồm 2 kiểu cấu trúc chính:

- Object: { key:value, key:value, key:value, ... }, tập hợp các cặp key-value.
- Array: { key:[value1, value2, ...]}, danh sách thứ tự các giá trị, cách nhau bởi dấu phẩy.

Ví dụ một đoạn JSON đơn giản:

```
{  
    "name": "Nguyen Trong Nhan",  
    "gmail": "nhannt.miya@gmail.com",  
    "address": ["8 Ha Van Tinh", "Da Nang", "Viet Nam"]  
}
```

1. GIỚI THIỆU

JSON serialization: là quá trình chuyển đổi qua lại giữa dữ liệu trong chương trình và định dạng JSON.

- **Encoding (serialization):** Dart Object → JSON.
- **Decoding (deserialization):** JSON → Dart Object

JSON SERIALIZATION IN FLUTTER



2. PHƯƠNG PHÁP THỦ CÔNG (MANUAL SERIALIZATION)

Manual serialization (Chuyển đổi thủ công) được hỗ trợ bởi Flutter với thư viện `dart:convert`, bao gồm trình mã hóa và giải mã trực quan. Với dữ liệu mẫu JSON ban đầu, chúng ta có thể chuyển hóa thủ công theo 2 cách:

- Chuyển hóa JSON trực tiếp
- Chuyển hóa JSON bên trong các lớp mô hình.

2.1. CHUYỂN HÓA JSON TRỰC TIẾP

Bằng cách sử dụng `dart:convert`, có thể giải mã và mã hóa JSON bằng cách gọi hàm `jsonDecode()` và `jsonEncode()`. Cụ thể:

- `jsonDecode()`: chuyển chuỗi JSON (`String`) → `Map<String, dynamic>` hoặc `List<dynamic>`
- `jsonEncode()`: chuyển `Map`/`List`/`Object` → chuỗi JSON (`String`)

2.1. CHUYỂN HÓA JSON TRỰC TIẾP

Ví dụ minh họa:

Giải mã JSON (Decode):

```
1 import 'dart:convert';
2
3 void main() {
4     // Chuỗi JSON ban đầu (giống như dữ liệu trả về từ API)
5     String jsonString = '{"name": "Nguyen Trong Nhan", "age": 21, "email": "nhannt.miya@gmail.com"}';
6
7     Map<String, dynamic> userMap = jsonDecode(jsonString);
8
9     print('Tên: ${userMap['name']}');
10    print('Tuổi: ${userMap['age']}');
11    print('Email: ${userMap['email']}');
12 }
```

2.1. CHUYỂN HÓA JSON TRỰC TIẾP

Ví dụ minh họa:

Giải mã JSON (Decode), kết quả chương trình:

Tên: Nguyen Trong Nhan

Tuổi: 21

Email: nhannnt.miya@gmail.com

2.1. CHUYỂN HÓA JSON TRỰC TIẾP

Ví dụ minh họa:

Mã hóa JSON (Encode):

```
16 void main() {  
17     Map<String, dynamic> user = {  
18         'name': 'Nguyen Trong Nhan',  
19         'age': 21,  
20         'email': 'nhannt.miya@gmail.com'  
21     };  
22  
23     String jsonString = jsonEncode(user);  
24     print(jsonString);  
25 }
```

Kết quả chương trình:

```
{"name": "Nguyen Trong Nhan", "age": 21, "email": "nhannt.miya@gmail.com"}
```

2.2. CHUYỂN HÓA JSON BÊN TRONG CÁC LỚP MÔ HÌNH

Đây là quá trình tạo các phương thức trong các lớp Dart để chuyển đổi dữ liệu giữa JSON và đối tượng. Các lớp này có:

- Hàm tạo `fromJson()` được dùng để tạo một đối tượng mới từ một cấu trúc dạng Map (thường là dữ liệu đã được giải mã từ JSON).
- Phương thức `toJson()` được dùng để chuyển đổi một đối tượng User thành một Map, chuẩn bị cho việc mã hóa sang JSON.

Để mã hóa người dùng, chỉ cần truyền đối tượng User vào hàm `jsonEncode()`.

2.2. CHUYỂN HÓA JSON BÊN TRONG CÁC LỚP MÔ HÌNH

Ví dụ minh họa:

```
30 class User {  
31     String name;  
32     int age;  
33     String gmail;  
34  
35     User({required this.name, required this.age, required this.gmail,  
36     });  
37  
38     factory User.fromJson(Map<String, dynamic> json) {  
39         return User(  
40             name: json['name'],  
41             age: json['age'],  
42             gmail: json['gmail'],  
43         );  
44     }  
45  
46     Map<String, dynamic> toJson() {  
47         return {  
48             'name': name,  
49             'age': age,  
50             'gmail': gmail,  
51         };  
52     }  
53 }
```

2.2. CHUYỂN HÓA JSON BÊN TRONG CÁC LỚP MÔ HÌNH

Ví dụ minh họa:

```
55 ➤ void main() {  
56     String jsonString = '{"name": "Nguyen Trong Nhan", "age": 21, "gmail": "nhannt.miya@gmail.com"}';  
57     Map<String, dynamic> userMap = jsonDecode(jsonString);  
58     User user = User.fromJson(userMap);  
59  
60     print('--- Thông tin người dùng ---');  
61     print('Tên: ${user.name}');  
62     print('Tuổi: ${user.age}');  
63     print('Gmail: ${user.gmail}');  
64  
65     String encodedJson = jsonEncode(user.toJson());  
66     print('\n--- JSON sau khi mã hóa lại ---');  
67     print(encodedJson);  
68 }
```

2.2. CHUYỂN HÓA JSON BÊN TRONG CÁC LỚP MÔ HÌNH

Kết quả chạy chương trình:

--- Thông tin người dùng ---

Tên: Nguyen Trong Nhan

Tuổi: 21

Gmail: nhannt.miya@gmail.com

--- JSON sau khi mã hóa lại ---

```
{"name": "Nguyen Trong Nhan", "age": 21, "gmail": "nhannt.miya@gmail.com"}
```

2.3. LÀM VIỆC VỚI JSON PHỨC TẠP

Một đối tượng JSON thông thường có thể trông đơn giản, nhưng trong các ứng dụng thực tế, thường phải xử lý các cấu trúc phức tạp hơn.

Chúng có thể là:

- Các đối tượng lồng nhau (nested objects).
- Mảng các đối tượng (arrays of objects).
- Sự kết hợp của cả hai.

2.3. LÀM VIỆC VỚI JSON PHỨC TẠP

Ví dụ cấu trúc JSON phức tạp:

```
{  
    "name": "Nguyen Trong Nhan",  
    "gmail": "nhannt.miya@gmail.com",  
    "address": {  
        "street": "8 Ha Van Tinh",  
        "city": "Da Nang",  
        "country": "Viet Nam"  
    },  
    "contacts": [  
        {  
            "type": "home",  
            "number": "123456789"  
        },  
        {  
            "type": "work",  
            "number": "123456789"  
        }  
    ]  
}
```

JSON này bao gồm đối tượng lồng nhau (address) và một mảng các đối tượng (contacts).

2.3. LÀM VIỆC VỚI JSON PHỨC TẠP

Để phân tích dữ liệu JSON phức tạp, chúng ta có các cách thức xử lý sau:

- Tạo các lớp Dart có cấu trúc lồng nhau: Đối với mỗi cấu trúc lồng nhau trong JSON, hãy tạo một lớp Dart tương ứng.
 - Sử dụng factory constructor để giải mã phức tạp: factory constructor là một loại hàm khởi tạo trong Dart, nó có thể:
 - Trả về một đối tượng đã tồn tại
 - Trả về một kiểu con (subclass)
 - Xử lý logic trước khi tạo đối tượng
- => Giúp mã gọn gàng và dễ bảo trì hơn.

2.3. LÀM VIỆC VỚI JSON PHỨC TẠP

Ví dụ minh họa:

```
{  
  "name": "Nguyen Trong Nhan",  
  "gmail": "nhannt.miya@gmail.com",  
  "address": {  
    "street": "8 Ha Van Tinh",  
    "city": "Da Nang",  
    "country": "Viet Nam"  
  },  
  "contacts": [  
    {  
      "type": "home",  
      "number": "123456789"  
    },  
    {  
      "type": "work",  
      "number": "123456789"  
    }  
  ]  
}
```

```
// Lớp Address đại diện cho đối tượng địa chỉ được lồng bên trong  
class Address {  
  final String street;  
  final String city;  
  final String country;  
  
  Address({required this.street, required this.city, required this.country});  
  
  factory Address.fromJson(Map<String, dynamic> json) {  
    return Address(  
      street: json['street'],  
      city: json['city'],  
      country: json['country'],  
    );  
  }  
  
  Map<String, dynamic> toJson() {  
    return {  
      'street': street,  
      'city': city,  
      'country': country,  
    };  
  }  
}
```

2.3. LÀM VIỆC VỚI JSON PHỨC TẠP

Ví dụ minh họa:

```
{  
  "name": "Nguyen Trong Nhan",  
  "gmail": "nhannt.miya@gmail.com",  
  "address": {  
    "street": "8 Ha Van Tinh",  
    "city": "Da Nang",  
    "country": "Viet Nam"  
  },  
  "contacts": [  
    {  
      "type": "home",  
      "number": "123456789"  
    },  
    {  
      "type": "work",  
      "number": "123456789"  
    }  
  ]  
}
```

```
class Contact {  
  final String type;  
  final String number;  
  
  Contact({required this.type, required this.number});  
  
  factory Contact.fromJson(Map<String, dynamic> json) {  
    return Contact(  
      type: json['type'],  
      number: json['number'],  
    );  
  }  
  
  Map<String, dynamic> toJson() {  
    return {  
      'type': type,  
      'number': number,  
    };  
  }  
}
```

2.3. LÀM VIỆC VỚI JSON PHỨC TẠP

Ví dụ minh họa:

```
class User {  
    final String name;  
    final String email;  
    final Address address;  
    final List<Contact> contacts;  
  
    User({required this.name, required this.email, required this.address, required this.contacts});  
  
    factory User.fromJson(Map<String, dynamic> json) {  
        var contactsFromJson = json['contacts'] as List;  
        List<Contact> contactList = contactsFromJson.map((contactJson) => Contact.fromJson(contactJson)).toList();  
        return User(  
            name: json['name'],  
            email: json['email'],  
            address: Address.fromJson(json['address']),  
            contacts: contactList,  
        ); // User  
    }  
  
    Map<String, dynamic> toJson() {  
        return {  
            'name': name,  
            'email': email,  
            'address': address.toJson(),  
            'contacts': contacts.map((contact) => contact.toJson()).toList(),  
        };  
    }  
}
```

2.3. LÀM VIỆC VỚI JSON PHỨC TẠP

Ví dụ minh họa:

```
factory User.fromJson(Map<String, dynamic> json) {  
    var contactsFromJson = json['contacts'] as List;  
    List<Contact> contactList = contactsFromJson.map((contactJson) => Contact.fromJson(contactJson)).toList();  
    return User(  
        name: json['name'],  
        email: json['email'],  
        address: Address.fromJson(json['address']),  
        contacts: contactList,  
    ); // User  
}
```

3. SỬ DỤNG JSON_ANNOTATION VÀ BUILD_RUNNER

Sử dụng JSON serialization

- Lưu dữ liệu vào file hoặc database.
- Gửi dữ liệu giữa client và server (API).
- Dễ dàng đọc và ghi dữ liệu vì JSON là định dạng phổ biến, nhẹ và con người có thể đọc được.

Tiêu chí	Thủ công	Tự động
Cách thực hiện	Viết tay fromJson() / toJson()	Dùng json_annotation, json_serializable, build_runner để sinh code
Ưu điểm	Dễ hiểu với project nhỏ Không cần cài thêm package	Giảm lỗi khi convert dữ liệu Tiết kiệm thời gian với class lớn hoặc nhiều nested objects Dễ bảo trì và mở rộng
Nhược điểm	Dễ sai sót khi viết tay Khó bảo trì khi class phức tạp Mất thời gian nếu class nhiều field	Cần cài đặt và học thêm package Phụ thuộc vào quá trình build/code generation

3. PHƯƠNG PHÁP TỰ ĐỘNG SINH MÃ(AUTOMATED SERIALIZATION USING CODE GENERATION)

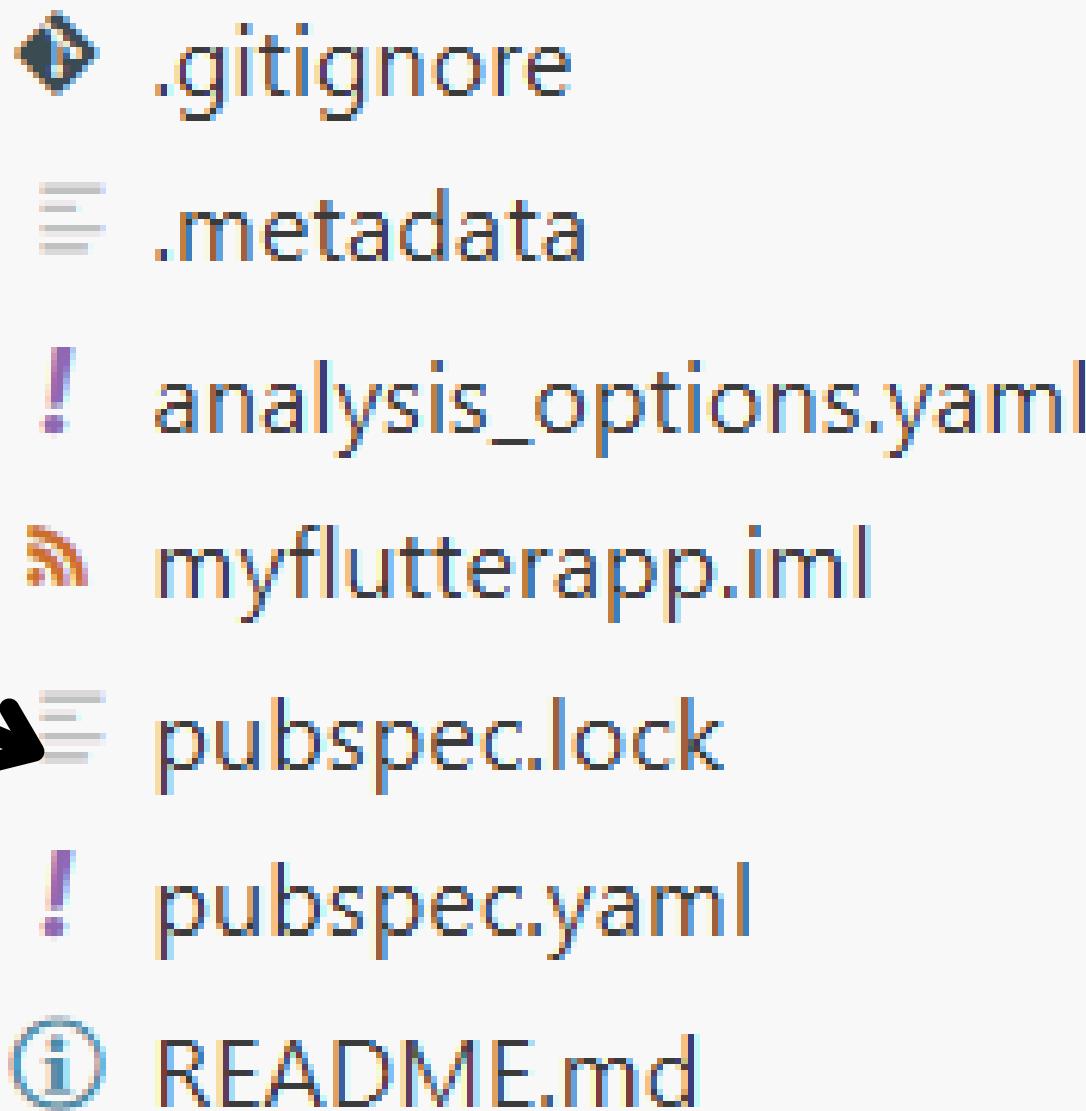
Để thực hiện json serialization cần có các công cụ cơ bản:

json_annotation	json_serializable	build_runner
Cung cấp các annotation để đánh dấu class hoặc field cần serialize Chỉ đánh dấu, không sinh code trực tiếp VD: @JsonSerializable(), @JsonKey()	Sinh code fromJson() và toJson() tự động dựa trên annotation Khi chạy code generation, tạo file .g.dart chứa các hàm serialize/deserialize VD: _\$StudentFromJson(json), _\$StudentToJson(this)	Công cụ chạy code generation Phân tích code, tìm annotation, sinh file .g.dart và quản lý xung đột VD: dart run build_runner build / watch / --delete-conflicting-outputs

3. PHƯƠNG PHÁP TỰ ĐỘNG SINH MÃ(AUTOMATED SERIALIZATION USING CODE GENERATION)

Các bước thực hiện

Bước 1: Thêm dependency (thêm các package vào pubspec.yaml):



```
dependencies:  
  flutter:  
    sdk: flutter  
  logger: ^1.3.0  
  cupertino_icons: ^1.0.8  
  json_annotation: ^4.9.0  
  
dev_dependencies:  
  flutter_test:  
    sdk: flutter  
  flutter_lints: ^5.0.0  
  json_serializable: ^6.11.1  
  build_runner: ^2.10.1
```

3. PHƯƠNG PHÁP TỰ ĐỘNG SINH MÃ(AUTOMATED SERIALIZATION USING CODE GENERATION)

Các bước thực hiện

Bước 1: Thêm dependency (Sử dụng câu lệnh trực tiếp trên terminal):

```
FLUTTER PUB ADD JSON_ANNOTATION DEV:BUILD_RUNNER DEV:JSON_SERIALIZABLE
```

3. PHƯƠNG PHÁP TỰ ĐỘNG SINH MÃ(AUTOMATED SERIALIZATION USING CODE GENERATION)

Các bước thực hiện

Bước 2: Đánh dấu
class với annotation

Class bình thường
khi làm thủ công

```
class User {  
    final String name;  
    final int age;  
  
    User(this.name, this.age);  
    // Chuyển JSON thành đối tượng (Deserialization)  
    factory User.fromJson(Map<String, dynamic> json) {  
        return User(  
            json['name'] as String, // ép kiểu để tránh lỗi  
            json['age'] as int,  
        );  
    }  
    // Chuyển đối tượng thành JSON (Serialization)  
    Map<String, dynamic> toJson() {  
        return {  
            'name': name,  
            'age': age,  
        };  
    }  
}
```

3. PHƯƠNG PHÁP TỰ ĐỘNG SINH MÃ(AUTOMATED SERIALIZATION USING CODE GENERATION)

Các bước thực hiện

Bước 2: Đánh dấu class với annotation

```
import 'package:json_annotation/json_annotation.dart';
part 'user.g.dart'; // bắt buộc phải có

@JsonSerializable()
class User {
    final String name;
    final int age;
    User(this.name, this.age);
    factory User.fromJson(Map<String, dynamic> json) => _$UserFromJson(json);
    Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

Class được đánh dấu để tự động sinh mã

3. PHƯƠNG PHÁP TỰ ĐỘNG SINH MÃ(AUTOMATED SERIALIZATION USING CODE GENERATION)

Các bước thực hiện

Bước 3: Chạy lệnh trên terminal

```
DART RUN BUILD_RUNNER BUILD --DELETE-CONFLICTING-OUTPUTS
```

3. PHƯƠNG PHÁP TỰ ĐỘNG SINH MÃ(AUTOMATED SERIALIZATION USING CODE GENERATION)

Các bước thực hiện

Bước 3: Chạy lệnh trên terminal

```
// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'user.dart';

// *****
// JsonSerializableGenerator
// *****

User _$UserFromJson(Map<String, dynamic> json) =>
    User(json['name'] as String, (json['age'] as num).toInt());

Map<String, dynamic> _$UserToJson(User instance) => <String, dynamic>{
    'name': instance.name,
    'age': instance.age,
};
```

File user.g.dart

3. PHƯƠNG PHÁP TỰ ĐỘNG SINH MÃ(AUTOMATED SERIALIZATION USING CODE GENERATION)

Các bước thực hiện

Bước 4: Gọi lại trong file main.dart để sử dụng

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'user.dart';

void main() {
    // JSON mẫu
    final jsonData = '{"name": "Hoàng", "age": 21}';
    // Chuyển JSON thành object User
    final user = User.fromJson(jsonDecode(jsonData));
    runApp(MyApp(user: user));
}
```

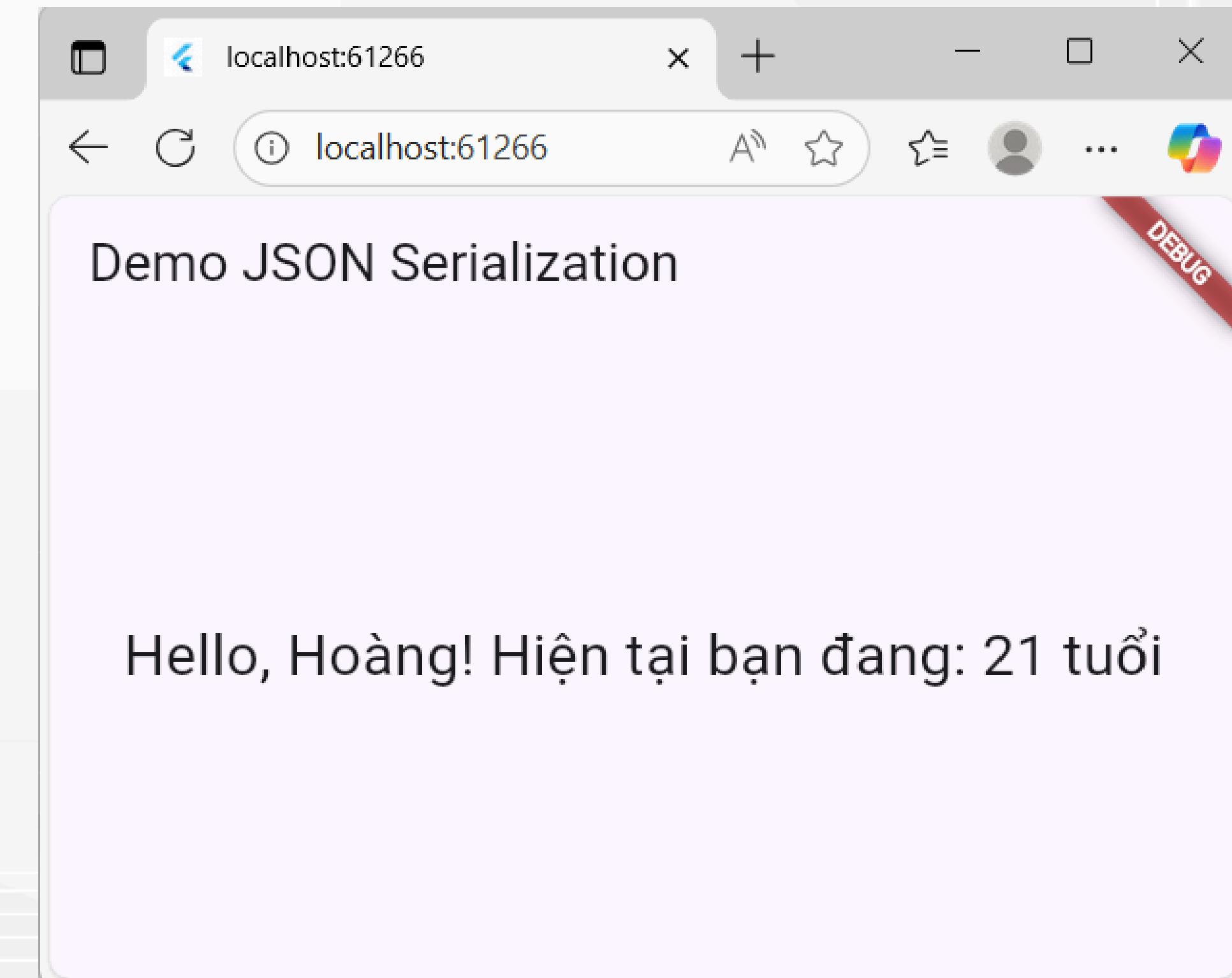
3. PHƯƠNG PHÁP TỰ ĐỘNG SINH MÃ(AUTOMATED SERIALIZATION USING CODE GENERATION)

Các bước thực hiện
**Bước 4: Gọi lại trong file
main.dart để sử dụng**

```
class MyApp extends StatelessWidget {  
  final User user;  
  const MyApp({super.key, required this.user});  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      home: Scaffold(  
        appBar: AppBar(title: const Text('Demo JSON Serialization')),  
        body: Center(  
          child: Text(  
            'Hello, ${user.name}! Hiện tại bạn đang: ${user.age} tuổi',  
            style: const TextStyle(fontSize: 24),  
          ), // Text  
        ), // Center  
      ), // Scaffold  
    ); // MaterialApp  
  }  
}
```

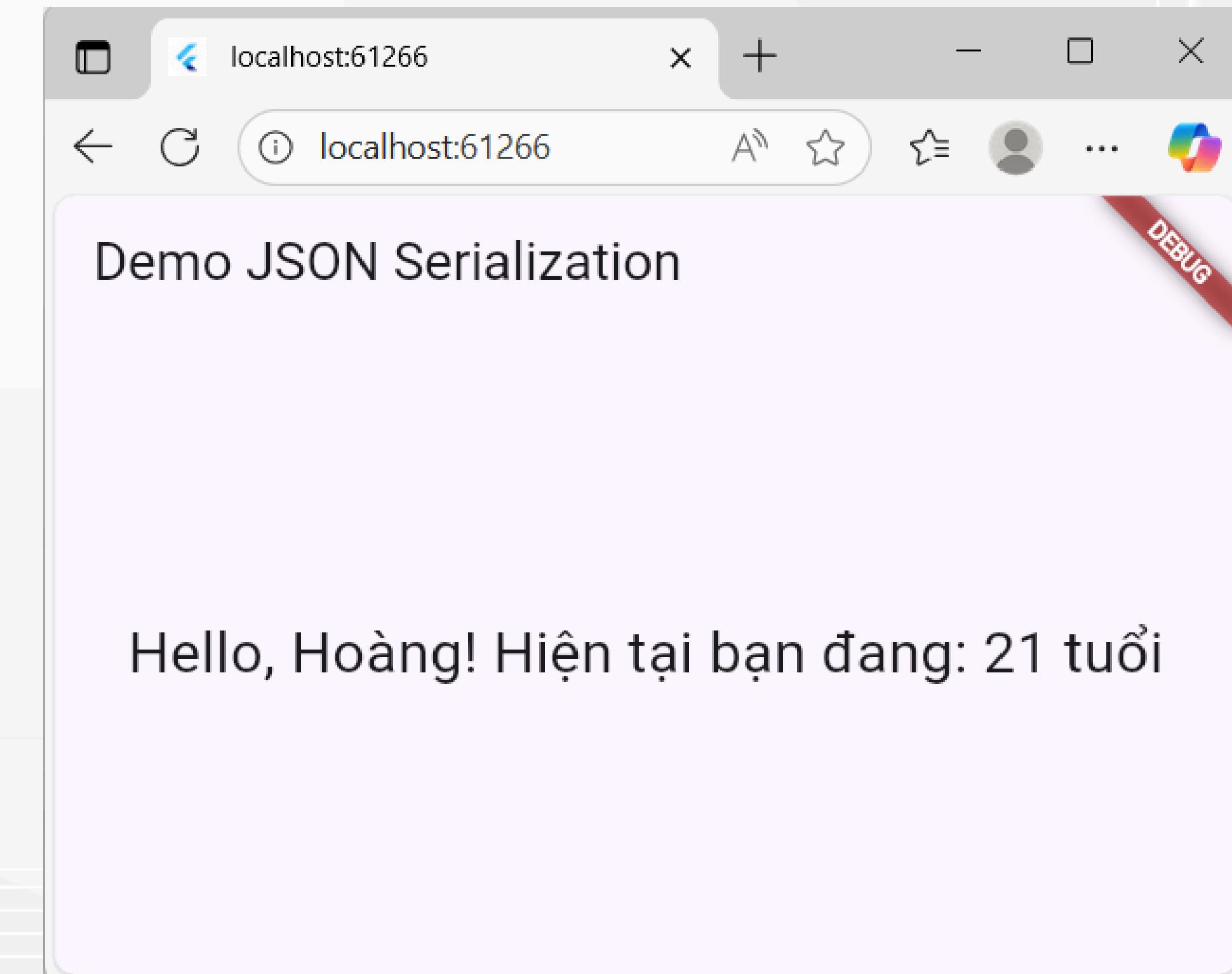
3. PHƯƠNG PHÁP TỰ ĐỘNG SINH MÃ(AUTOMATED SERIALIZATION USING CODE GENERATION)

Kết quả demo



3. PHƯƠNG PHÁP TỰ ĐỘNG SINH MÃ(AUTOMATED SERIALIZATION USING CODE GENERATION)

Kết quả demo



4. CÁC PHƯƠNG PHÁP TỐT NHẤT KHI LÀM VIỆC VỚI CẤU TRÚC DỮ LIỆU PHỨC TẠP

Khi làm việc với dữ liệu JSON phức tạp, điều quan trọng là giữ code có tổ chức và dễ bảo trì.

- **Tách module:** Chia nhỏ code, mỗi lớp xử lý một phần dữ liệu riêng.
- **Đặt tên rõ ràng:** Tên lớp và biến phản ánh đúng chức năng.
- **Ghi chú hợp lý:** Thêm comment cho logic phức tạp.
- **Kiểm thử JSON:** Viết unit test để đảm bảo phân tích và chuyển đổi dữ liệu chính xác.

4. CÁC PHƯƠNG PHÁP TỐT NHẤT KHI LÀM VIỆC VỚI CẤU TRÚC DỮ LIỆU PHỨC TẠP

Json mẫu

```
void main() {  
    //JSON mẫu  
    final jsonData = ''  
    {  
        "name": "Duy Hoàng",  
        "age": 21,  
        "studentId": {"id": "SV106220252", "year": 4},  
        "major": {"name": "Kỹ thuật máy tính", "faculty": "Khoa Điện tử - Viễn Thông"},  
        "address": {"street": "67 Hà Huy Tập", "city": "Đà Nẵng"}  
    }  
};
```

4. CÁC PHƯƠNG PHÁP TỐT NHẤT KHI LÀM VIỆC VỚI CẤU TRÚC DỮ LIỆU PHỨC TẠP

Chia các lớp riêng biệt và chú thích

```
/*
 * ======
 * LỚP 1: Thông tin ngành học
 * ======
 */
class Major {
    final String name;      // Tên ngành
    final String faculty;   // Tên khoa

    Major({required this.name, required this.faculty});

    factory Major.fromJson(Map<String, dynamic> json) => Major(
        name: json['name'],
        faculty: json['faculty'],
    );

    @override
    String toString() => 'Ngành: $name, Khoa: $faculty';
}
```

4. CÁC PHƯƠNG PHÁP TỐT NHẤT KHI LÀM VIỆC VỚI CẤU TRÚC DỮ LIỆU PHỨC TẠP

Chia các lớp riêng biệt và chú thích

```
/*
 * ======
 * LỚP 2: Thông tin mã số sinh viên
 * ====== */
class StudentID {
    final String id;    // Mã sinh viên
    final int year;    // Năm nhập học

    StudentID({required this.id, required this.year});

    factory StudentID.fromJson(Map<String, dynamic> json) => StudentID(
        id: json['id'],
        year: json['year'],
    );

    @override
    String toString() => 'Mã SV: $id, Năm nhập học: $year';
}
```

4. CÁC PHƯƠNG PHÁP TỐT NHẤT KHI LÀM VIỆC VỚI CẤU TRÚC DỮ LIỆU PHỨC TẠP

Chia các lớp riêng biệt và chú thích

```
/* =====
 * LỚP 3: Địa chỉ sinh viên
 * ===== */
class Address {
    final String street; // Tên đường
    final String city; // Thành phố

    Address({required this.street, required this.city});

    factory Address.fromJson(Map<String, dynamic> json) => Address(
        street: json['street'],
        city: json['city'],
    );

    @override
    String toString() => 'Địa chỉ: $street, $city';
}
```

4. CÁC PHƯƠNG PHÁP TỐT NHẤT KHI LÀM VIỆC VỚI CẤU TRÚC DỮ LIỆU PHỨC TẠP

Chia các lớp riêng biệt và chú thích

```
/* =====
 * LỚP 4: Lớp chính - Sinh viên
 * ===== */
class Student {
    final String name;      // Họ tên
    final int age;          // Tuổi
    final StudentID studentId; // Thông tin mã SV
    final Major major;      // Ngành học
    final Address address; // Địa chỉ

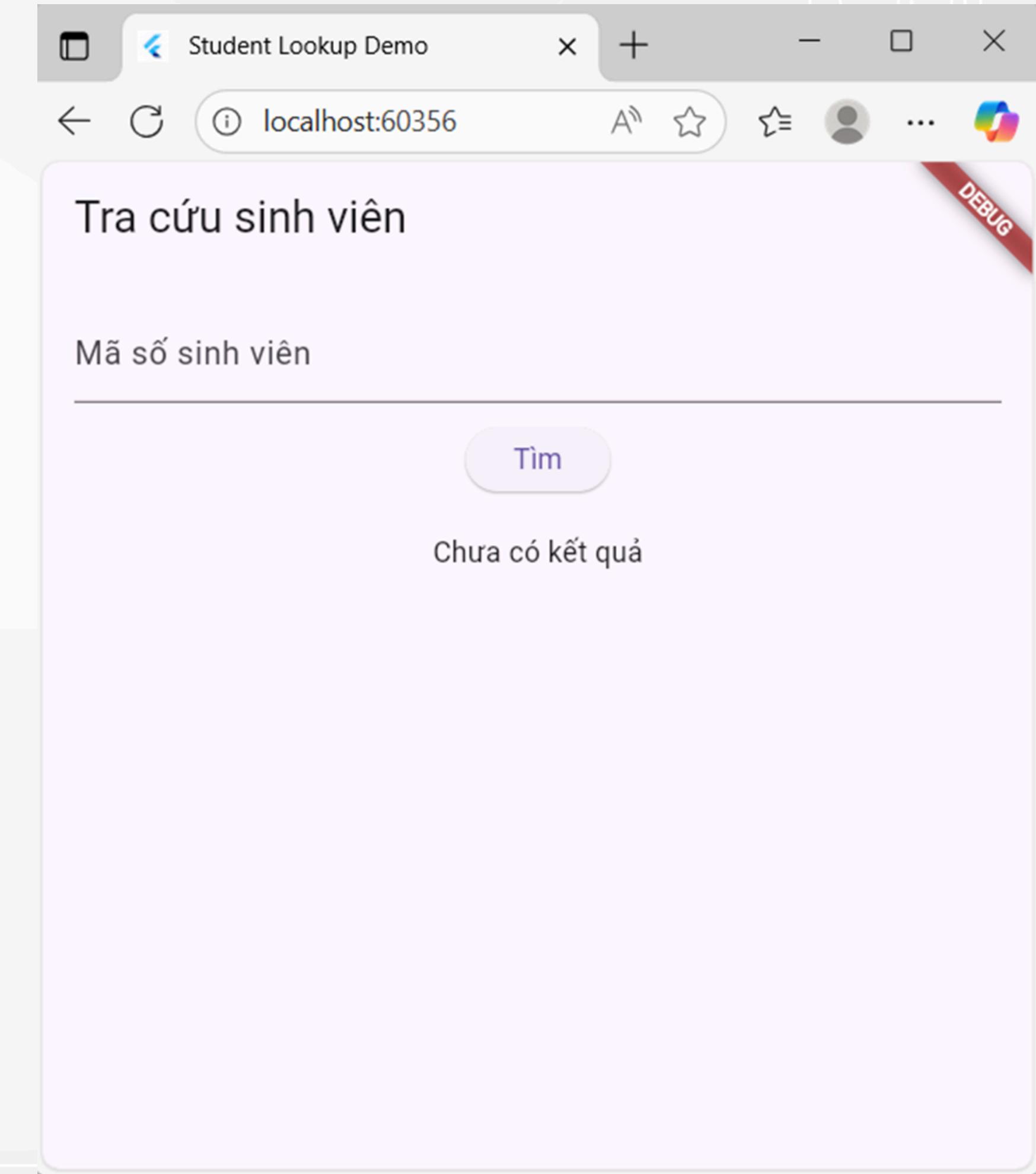
    Student({
        required this.name,
        required this.age,
        required this.studentId,
        required this.major,
        required this.address,
    });

    factory Student.fromJson(Map<String, dynamic> json) => Student(
        name: json['name'],
        age: json['age'],
        studentId: StudentID.fromJson(json['studentId']),
        major: Major.fromJson(json['major']),
        address: Address.fromJson(json['address']),
    );
}
```

```
@override
String toString() => '''
Tên: $name
Tuổi: $age
${studentId.toString()}
${major.toString()}
${address.toString()}
''';
```

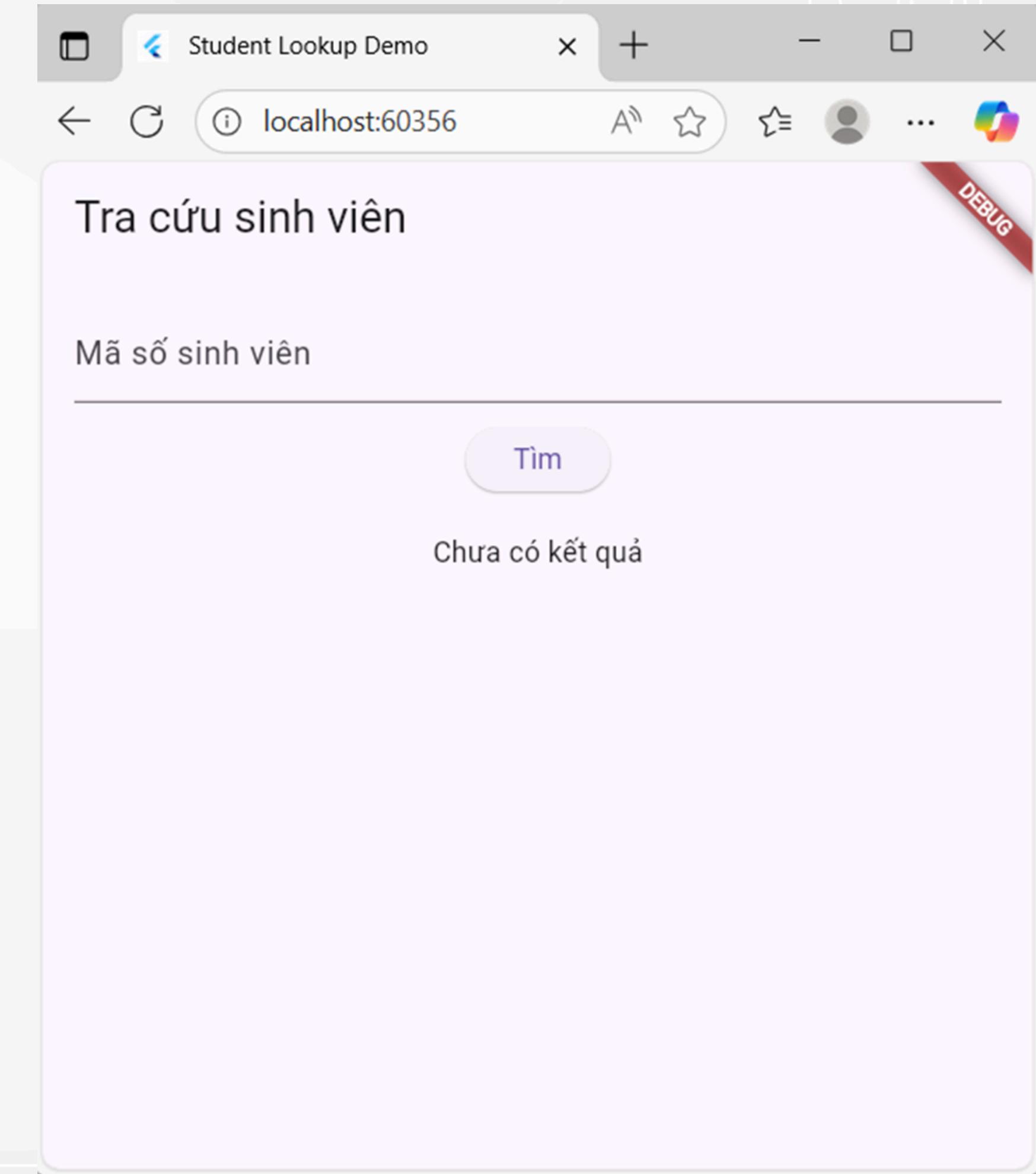
4. CÁC PHƯƠNG PHÁP TỐT NHẤT KHI LÀM VIỆC VỚI CẤU TRÚC DỮ LIỆU PHỨC TẠP

**Ứng dụng tra cứu thông
tin sinh viên thông qua
mã số sinh viên**



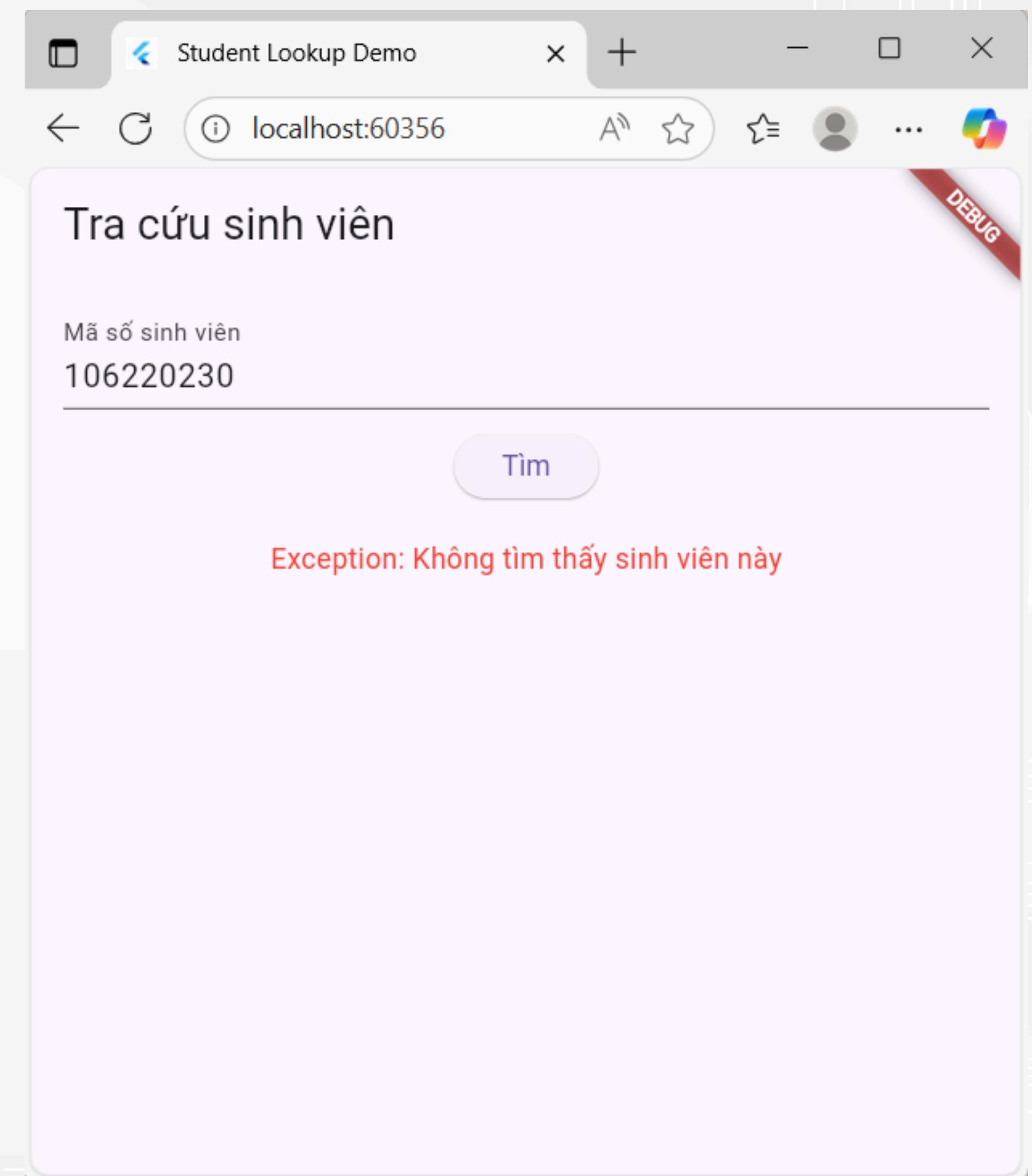
4. CÁC PHƯƠNG PHÁP TỐT NHẤT KHI LÀM VIỆC VỚI CẤU TRÚC DỮ LIỆU PHỨC TẠP

**Ứng dụng tra cứu thông
tin sinh viên thông qua
mã số sinh viên**



4. CÁC PHƯƠNG PHÁP TỐT NHẤT KHI LÀM VIỆC VỚI CẤU TRÚC DỮ LIỆU PHỨC TẠP

**Ứng dụng tra cứu thông
tin sinh viên thông qua
mã số sinh viên**



4. CÁC PHƯƠNG PHÁP TỐT NHẤT KHI LÀM VIỆC VỚI CẤU TRÚC DỮ LIỆU PHỨC TẠP

Ứng dụng tra cứu thông
tin sinh viên thông qua
mã số sinh viên

The screenshot shows a web browser window titled "Student Lookup Demo" with the URL "localhost:60356". A red diagonal "DEBUG" banner is visible in the top right corner. The main content area is titled "Tra cứu sinh viên". It displays a student record with the ID "106220229". A "Tìm" button is located below the input field. To the right, a detailed view of the student information is shown in a rounded box:

Mã:	106220229
Tên:	Nguyễn Trọng Nhân
Ngành:	Kỹ thuật máy tính
Năm học:	4
<u>Địa chỉ:</u>	08 Đường Hà Văn Tính Đà Nẵng, Việt Nam

4. CÁC PHƯƠNG PHÁP TỐT NHẤT KHI LÀM VIỆC VỚI CẤU TRÚC DỮ LIỆU PHỨC TẠP

Ứng dụng tra cứu thông
tin sinh viên thông qua
mã số sinh viên

The screenshot shows a web browser window titled "Student Lookup Demo" with the URL "localhost:60356". The page has a header "Tra cứu sinh viên" and a form field "Mã số sinh viên" containing the value "106220229". A "Tìm" button is visible. To the right, a card displays student information: Mã: 106220229, Tên: Nguyễn Trọng Nhân, Ngành: Kỹ thuật máy tính, Năm học: 4. Below this, Địa chỉ: 08 Đường Hà Văn Tính, Đà Nẵng, Việt Nam is listed. A red diagonal banner in the top right corner of the page says "DEBUG".

Mã số sinh viên
106220229

Tìm

Mã: 106220229
Tên: Nguyễn Trọng Nhân
Ngành: Kỹ thuật máy tính
Năm học: 4
Địa chỉ:
08 Đường Hà Văn Tính
Đà Nẵng, Việt Nam

DEBUG

4. CÁC PHƯƠNG PHÁP TỐT NHẤT KHI LÀM VIỆC VỚI CẤU TRÚC DỮ LIỆU PHỨC TẠP

Ứng dụng tra cứu thông
tin sinh viên thông qua
mã số sinh viên

The screenshot shows a web browser window titled "Student Lookup Demo" with the URL "localhost:60356". The page has a header "Tra cứu sinh viên" and a search input field containing the student ID "106220252". A "Tìm" (Search) button is located next to the input field. Below the input field, a callout box displays the following information:

Mã: 106220252
Tên: Lê Duy Hoàng
Ngành: Kỹ thuật máy tính
Năm học: 4
Địa chỉ:
67 Hà Huy Tập
Đà Nẵng, Việt Nam

A red diagonal banner in the top right corner of the page says "DEBUG".

TÀI LIỆU THAM KHẢO

- [1] NGUYỄN DUY NHẬT VIỄN, SLIDE BÀI GIẢNG MÔN LẬP TRÌNH ĐA NỀN TẦNG, TRƯỜNG ĐẠI HỌC BÁCH KHOA ĐÀ NẴNG, 2025.
- [2] FLUTTER TEAM, FLUTTER JSON SERIALIZATION: FLUTTER DOCUMENTATION, 2025.
- [3] FLUTTER MASTERY LIBRARY, MASTERING COMPLEX JSON PARSING IN FLUTTER, 2025

**CẢM ƠN THẦY VÀ CÁC
BẠN ĐÃ LẮNG NGHE!**

