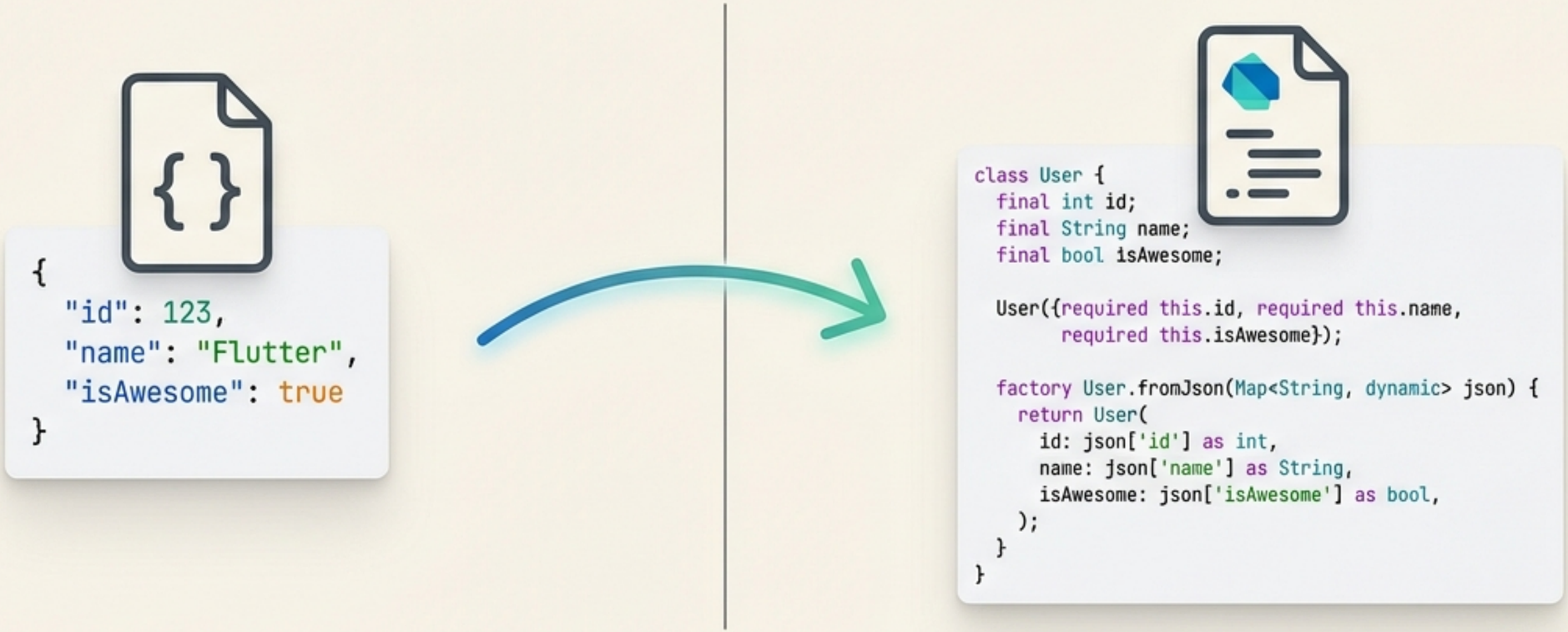
Flutter'da JSON Yönetimi: Manuel Temellerden Otomatik Üretime

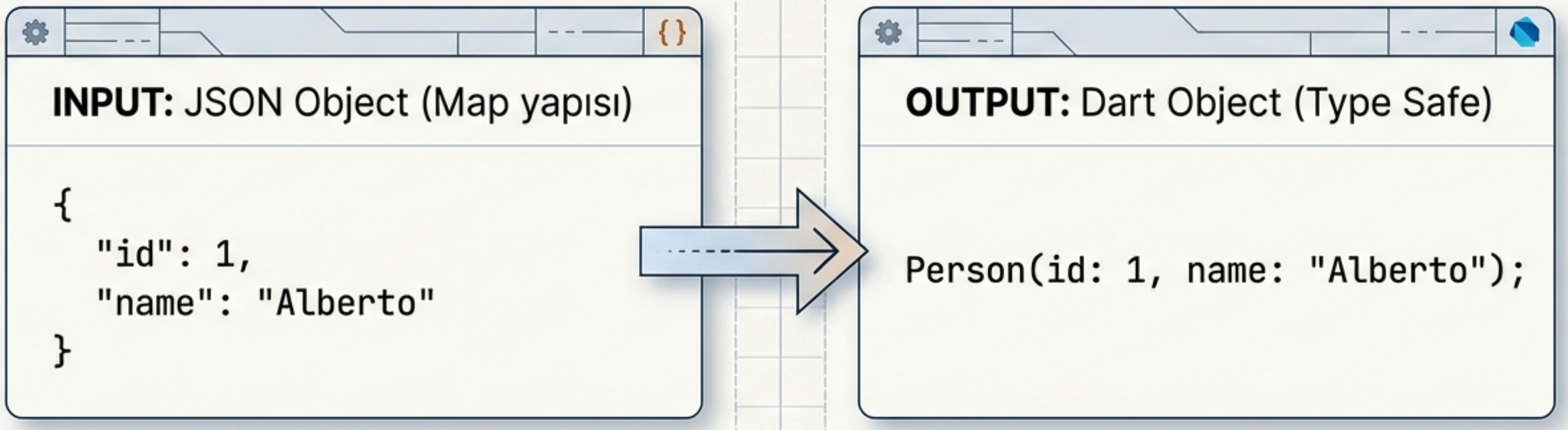
Veri güvenliği, bakım kolaylığı ve dart:convert ile json_serializable arasındaki köprü.



OKUNABİLİR VE PAYLAŞILABİLİR TEKNİK REHBER

Hedef: Ham String'den Güvenli Dart Nesnesine

JSON (JavaScript Object Notation), anahtar-değer (key-value) çiftlerinden oluşan, modern veri değişimi'in standardıdır. Flutter'da amacımız, bu dinamik yapıyı "Type Safe" (Tür Güvenli) Dart nesnelerine dönüştürmektir.



Adım 1: Manuel Ayrıştırma ve dart:convert

Flutter, JSON işlemleri için yerleşik dart:convert kütüphanesini kullanır. İlk aşama, jsonDecode metodu ile String verisini bir Map<String, dynamic> yapısına çevirmektir.

Dart Code

```dart
import 'dart:convert';

void main () {
  final jsonString = '{ \"id\": 1, \"name\": \"Alberto\" }';

  // String -> Map dönüşümü
  Map<String, dynamic> data = jsonDecode(jsonString);

  // data['id'] -> 1
}
```

Önemli: 'dynamic' kullanımı bu aşamada zorunludur çünkü değerler int, String, List veya null olabilir. Ancak nihai hedefimiz dynamic yapısından kurtulmaktır.

Model Sınıfı Tasarımı: fromJson Fabrikası

Veriyi güvenli bir şekilde işlemek için bir **Model Sınıfı** oluşturulur. Ayrıştırma mantığı (Parsing logic) tamamen bu sınıfın içine gizlenir.

Dış dünyadan gelen Map verisini alır ve güvenli nesneyi üretir.

```
class Person {
  final int id;
  final String name;

  // 1. Özel (Private) Kurucu
  Person._({required this.id, required this.name});

  // 2. Factory Constructor (Map -> Object)
  factory Person.fromJson(Map<String, dynamic>
```

```
  // 2. Factory Constructor (Map -> Object)
  factory Person.fromJson(Map<String, dynamic> json) {
    return Person._(
      id: json['id'],
      name: json['name'],
    );
  }
}
```

Erişim burada yapılır.

```
  }
```

Serileştirme: toJson ve jsonEncode

Nesneleri tekrar JSON formatına (örneğin bir sunucuya göndermek için) çevirmek için **toJson** metodu tanımlanır.

The Method Definition

```
// Sınıf içine eklenir
Map<String, dynamic> toJson() => {
  'id': id,
  'name': name
};
```

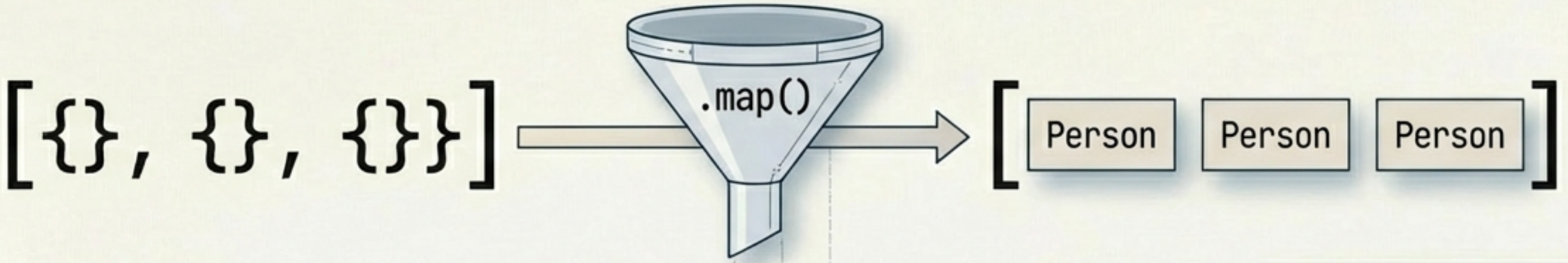
The Usage

```
final user = Person.fromJson(decodedMap);

// Otomatik olarak user.toJson() çağrılır
final jsonString = jsonEncode(user);

print(jsonString); // "{'id': 1, 'name': 'Alberto'}\"
```

Manuel yöntemde, her değişken için eşleşmeyi geliştirici eliyle yazar.

Zorluk Seviyesi 1: Listeleri Ayrıştırmak

JSON verisi genellikle bir nesne dizisi (Array) olarak gelir. Bu durumda **List** yapısı ve **.map** fonksiyonu kullanılır.



Dart Code Example

```dart
final jsonString = '...'; // Liste formatında string
List<dynamic> jsonList = jsonDecode(jsonString);

final List<Person> people = jsonList
  .map((item) => Person.fromJson(item)) // Her öğe için model kullanılır
  .toList();
```

Model sınıfı kullanmak, manuel dönüşüm (casts) yapma ihtiyacını ortadan kaldırır.

Zorluk Seviyesi 2: İç İçe (Nested) Nesneler

Gerçek dünya senaryolarında, nesneler iç içe geçer. Bu, her katman için ayrı bir **Model Sınıfı** yazılmasını gerektirir.

The Scenario

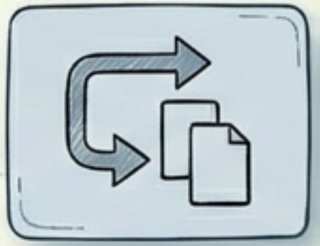
```
// JSON
{
  'type': 'Developers',
  'data': [
    {
      'id': 1,
      'name': 'Alberto' },
    },
    ...
  ]
}
```

The Code Burden

```
factory Data.fromJson(Map<String, dynamic> json) {
  var list = json['data'] as List;

  // İçerdeki listeyi de dönüştürmeliyiz
  List<Person> peopleList = list.map((i) =>
    Person.fromJson(i)).toList();

  return Data(
    type: json['type'],
    people: peopleList
  );
}
```

Manuel Yöntemin Sınırları: Bakım Maliyeti

Performans bir sorun değildir; asıl sorun yazılımın **sürdürülebilirliğidir**.

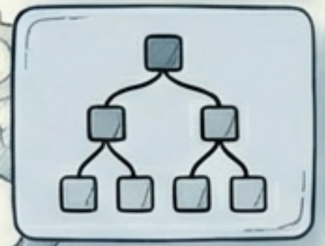


Boilerplate Kod

Her yeni alan (field) eklendiğinde hem **fromJson** hem de **toJson** güncellenmelidir.

Hata Riski

Yazım hataları (Typo) derleme zamanında değil, **çalışma zamanında (Runtime)** ortaya çıkar.



İç İçe Yapılar

Derinlik arttıkça kodun okunabilirliği azalır.

Çözüm: Kodun sıkıcı kısımlarını bir robota yazdırmak -> **Code Generation**

Technical Editorial

Çözüm: Otomatik Kod Üretimi (Code Generation)

json_serializable paketi, manuel olarak yazdığımız sıkıcı kodları bizim yerimize oluşturur. Siz alanları tanımlarsınız, **mantığı** o yazar.

pubspec.yaml Dependencies

```
dependencies:
  json_annotation: ^latest

dev_dependencies:
  build_runner: ^latest
  json_serializable: ^latest
```

Kod içinde kullanılan etiketler

Kod üretim mantığı

Kodu üreten komut satırı aracı

Technical Editorial

Kurulum: Sınıf Yapısını Hazırlamak

Sınıfımızı bir ‘Code Generation’ adayı haline getirmek için 3 temel değişiklik yapıyoruz.

person.dart

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

// 1. Referans
part 'person.g.dart';

// 2. İşaretleme
@JsonSerializable()
class Person {
  final int id;
  final String name;

  Person(this.id, this.name);

  // 3. Yönlendirme
  factory Person.fromJson(Map<String, dynamic> json) => _$PersonFromJson(json);
  Map<String, dynamic> toJson() => _$PersonToJson(this);
}
```

1. Oluşturulacak dosyaya referans (**part** directive)

2. Sınıfı işaretleme (**@JsonSerializable**)

3. Oluşturulan koda yönlendirme (Mixin benzeri yapı)

Sihirli Komut: build_runner

Terminal üzerinden çalıştırılan bu araç, **@JsonSerializable** ile işaretlenmiş sınıfları bulur ve **.g.dart** dosyalarını oluşturur.

```
$ flutter pub run build_runner watch
```

Tek Seferlik

`build`

Sadece bir kez çalışır.

Sürekli İzleme (Önerilen)

`watch`

Dosya değiştikçe kodu otomatik günceller.

Sahne Arkası: Oluşturulan 'person.g.dart' dosyası, aslında Bölüm 2'de elle yazdığımız manuel kodun aynısını içerir. Sihir değil, otomasyondur.

Detaylı Kontrol: @JsonKey Özellikleri

Otomatik üretimde, özel durumları yönetmek için değişkenlerin üzerine ek etiketler (annotations) koyarız.

Gereklilik

```dart
@JsonKey(required: true)
final int id;
```

JSON'da bu anahtar yoksa hata fırlat.

Varsayılan Değer

```dart
@JsonKey(defaultValue: 'No Name')
final String name;
```

Anahtar eksikse veya null ise bu değeri kullan.

İsim Değişikliği

```dart
@JsonKey(name: 'user_id')
final int id;
```

JSON anahtarı ile değişken adı farklıysa.

Otomasyonda İç İçe Nesneler: explicitToJson

Bir sınıf başka bir sınıfı içeriyorsa, üreticinin alt nesneleri de dönüştürmesi gerektiğini belirtmeliyiz.

```dart
@JsonSerializable(explicitToJson: true)
class Data {
  final String type;
  final List<Person> data;

  Data(this.type, this.data);

  factory Data.fromJson(Map<String, dynamic> json) => _$DataFromJson(json);
  Map<String, dynamic> toJson() => _$DataToJson(this);
}
```

İpucu: 'explicitToJson: true' kullanılmazsa, **toJson** çağrıldığında alt nesneler JSON formatına çevrilmeyebilir ve hata alabilirsiniz.

Karşılaştırma: Hangi Yöntemi Seçmeli?

Manuel Ayrıştırma	Otomatik Üretim (Code Gen)
✅ Küçük projeler ve hızlı prototipler için ideal.	✅ Büyük, karmaşık ve iç içe geçmiş yapılar için ideal.
✅ Ekstra kütüphane/bağımlılık gerektirmez.	✅ Tip hatası riskini minimize eder.
❌ Büyük veri yapılarında bakımı zordur.	✅ API değişikliklerine adapte olmak kolaydır.
❌ Boilerplate (tekrarlayan) kod çoktur.	❌ Kurulum ve build_runner bekleme süresi vardır.

Sonuç ve En İyi Uygulamalar

1. **Daima Model Kullanın:** Asla ham `Map` veya `dynamic` ile uygulama geliştirmeyin.
2. **Watch Komutu:** Geliştirme sırasında '`build_runner watch`' kullanarak kodun sürekli güncel kalmasını sağlayın.
3. **Defaults:** `@JsonKey(defaultValue: ...)` ile null hatalarının önüne geçin.
4. **Private Constructor:** Manuel yöntemde, sadece factory üzerinden nesne üretilmesini zorunlu kılın.

"Kod üretimi (Code Gen), bir 'sihir' değil; bakım yükünü azaltan ve hatayı önleyen bir mühendislik tercihidir."