MOJADIPRO

POWERING APLIKASI FLUTTER DENGAN BACKEND DATA

MOJADIPRO

MATERI

01 Mendapatkan Data Lokasi

02 Asynchronus Programming

03 Metode Lifecycle Stateful Widget

04 Exception Handling

05 Null Aware Operators

06 Clima Location Refactoring

07 API, Networking, Navigasi

08 XML, JSON, Json Parsing Dan Dynamic Types

MOJADIPRO

Mendapatkan Data Lokasi

Mendapatkan data lokasi dari berbagai platform berarti mengumpulkan informasi tentang lokasi seseorang dari berbagai sumber yang tersedia.

Hal ini bisa dilakukan menggunakan flutter package yang bernama Geolocator.
Plugin geolokasi Flutter yang menyediakan akses mudah ke layanan lokasi khusus LocationManager di Android dan CLLocationManager di iOS.

Fitur :

- Dapatkan lokasi terakhir yang diketahui;
- Dapatkan lokasi perangkat saat ini;
- Dapatkan pembaruan lokasi berkelanjutan;
- Periksa apakah layanan lokasi diaktifkan di perangkat;
- Hitung jarak (dalam meter) antara dua koordinat geografis;
- Hitung bantalan antara dua koordinat geografis;

MOJADIPRO

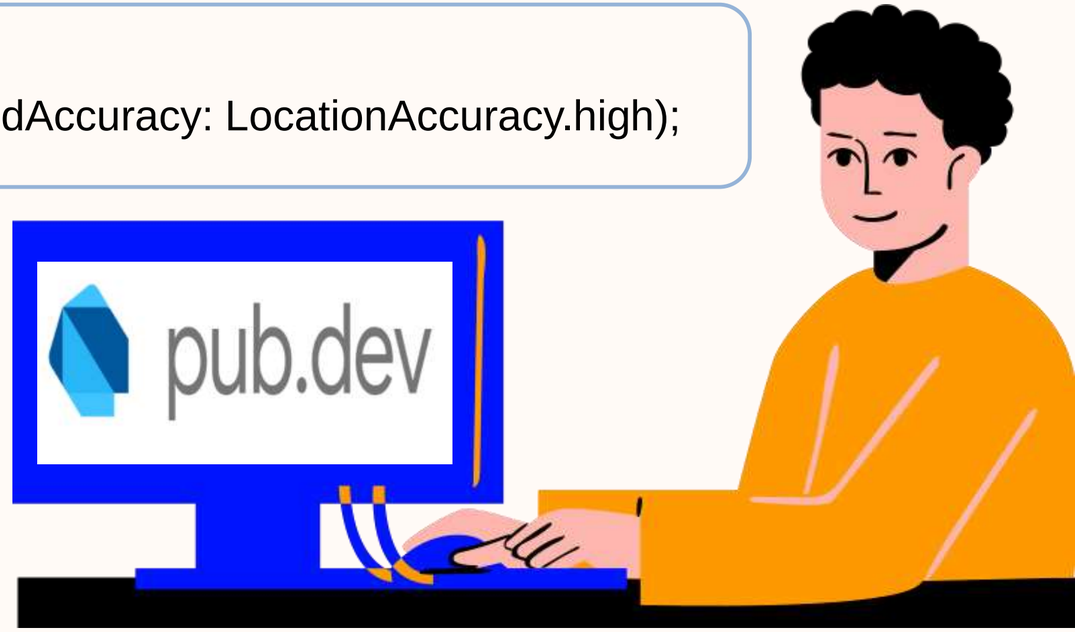
Mendapatkan Data Lokasi

1. Pastikan Anda telah menambahkan dependensi ke dalam file pubspec.yaml
   (dan menjalankan implisit flutter pub get)

```
dependencies:
    geolocator: ^9.0.2
```

2. Import package flutter pada main.dart

```
import 'package:geolocator/geolocator.dart';
Position position = await Geolocator.getCurrentPosition(desiredAccuracy: LocationAccuracy.high);
```



MOJADIPRO

Asynchonus Programming

Asynchronous programming adalah pendekatan pemrograman di mana Anda dapat menjalankan tugas secara konkuren atau paralel tanpa harus menunggu tugas sebelumnya selesai.

Dalam bahasa pemrograman Dart yang digunakan dalam pengembangan aplikasi Flutter, terdapat beberapa cara untuk melakukan asynchronous programming, seperti menggunakan async dan await atau Future.

- Async untuk menandai bahwa suatu fungsi adalah asynchronous
- Await untuk menunggu hasil
- Future yang dapat digunakan untuk mengembalikan nilai

MOJADIPRO

Asynchronus Programming

Berikut adalah contoh penggunaan Async dan Await dalam Dart:

```
void main() {
  print('Program dimulai');
  getData();
  print('Program selesai');
}

void getData() async {
  print('Memulai pengambilan data');
  await Future.delayed(Duration(seconds: 2)); // Menunggu selama 2 detik
  print('Data diterima');
}
```

Output dari kode di atas akan menjadi:

```
Program dimulai
Memulai pengambilan data
Program selesai
Data diterima
```

MOJADIPRO

Asynchronus Programming

Berikut adalah contoh penggunaan Future :

```
void main() {
  print('Fetching data...');
  fetchData().then((data) {
    print('Data fetched: $data');
  }).catchError((error) {
    print('Error: $error');
  }).whenComplete(() {
    print('Program completed.');
  });
}

Future<String> fetchData() {
  return Future.delayed(Duration(seconds: 2), () {
    // Simulasi pengambilan data dari sumber eksternal
    // Jika berhasil, kembalikan data; jika gagal, lemparkan error
    return 'Data from API';
  });
}
```

MOJADIPRO

Metode Lifecycle Stateful Widget

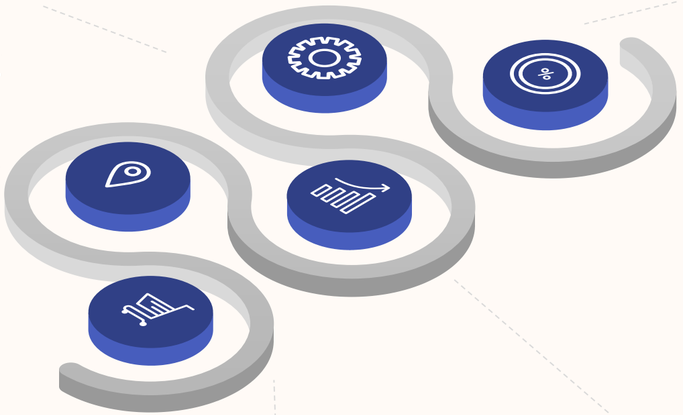
Stateful Widget adalah jenis widget yang memiliki keadaan (state) yang dapat berubah sepanjang waktu. Metode lifecycle digunakan dalam Stateful Widget untuk mengontrol siklus hidup widget dan merespons perubahan pada keadaan.

1. createState()

Metode ini dipanggil saat widget dibuat untuk pertama kalinya.
Metode ini mengembalikan objek State yang terkait dengan widget tersebut.

2. initState()

Metode ini dipanggil setelah objek State dibuat oleh createState(). Metode ini digunakan untuk melakukan inisialisasi awal pada widget, seperti mengatur keadaan awal atau memulai langganan pada sumber data eksternal.



3. dispose()

Metode ini dipanggil ketika widget dihapus dari pohon widget. Metode ini digunakan untuk membersihkan sumber daya yang digunakan oleh widget, seperti membatalkan langganan, menghentikan animasi, atau melepas sumber daya eksternal.

4. build()

Metode ini dipanggil setiap kali widget harus dirender ulang. Metode ini digunakan untuk membangun struktur tampilan widget dan mengembalikan widget yang akan ditampilkan di antarmuka pengguna.

5. CounterWidget

Counter Widget adalah Stateful Widget yang memiliki keadaan _counter yang bertambah setiap kali tombol ditekan.

MOJADIPRO

Metode Lifecycle Stateful Widget

Contoh penggunaan createState() :

```
class MyWidget extends StatefulWidget {
  @override
  _MyWidgetState createState() => _MyWidgetState();
}

class _MyWidgetState extends State<MyWidget> {
  @override
  void initState() {
    super.initState();
    // Metode ini dipanggil saat widget diinisialisasi
  }

  @override
  void dispose() {
    super.dispose();
    // Metode ini dipanggil saat widget dihapus dari pohon widget
  }

  @override
  Widget build(BuildContext context) {
    // Metode ini dipanggil setiap kali widget harus direnderr ulang
    return Container();
  }
}
```

MOJADIPRO

Metode Lifecycle Stateful Widget

Contoh penggunaan CounterWidget :

```
class CounterWidget extends StatefulWidget {
  @override
  _CounterWidgetState createState() => _CounterWidgetState();
}

class _CounterWidgetState extends State<CounterWidget> {
  int _counter = 0;

  @override
  void initState() {
    super.initState();
    print('Widget initialized');
  }

  @override
  void dispose() {
    super.dispose();
    print('Widget disposed');
  }
```

```
  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Text('Counter: $_counter'),
        ElevatedButton(
          onPressed: _incrementCounter,
          child: Text('Increment'),
        ),
      ],
    );
  }
}
```

MOJADIPRO

Exception Handling

Exception handling adalah mekanisme dalam Dart yang memungkinkan Anda untuk menangani kesalahan atau situasi yang tidak terduga dalam program Anda.
Dart menyediakan try dan catch untuk menangkap dan mengelola exception.

1. Try digunakan untuk menempatkan potongan kode yang mungkin menimbulkan exception.
2. Catch digunakan untuk menentukan tindakan yang harus diambil jika exception ditangkap.
3. Throw digunakan untuk melempar atau menghasilkan exception secara manual.
   Anda dapat menggunakan "throw" untuk menghentikan eksekusi normal program dan memberikan informasi khusus tentang kesalahan atau kondisi yang tidak terduga.

MOJADIPRO

Exception Handling

Contoh penggunaan try dan catch :

```
try {
  // Potongan kode yang mungkin menimbulkan exception
  int result = 10 ~/ 0; // Pembagian dengan nol
  print('Result: $result');
} catch (e) {
  // Exception ditangkap di sini
  print('Exception: $e');
}
```

MOJADIPRO

Exception Handling

Contoh penggunaan throw :

```
void validateAge(int age) {
  if (age < 18) {
    throw Exception('Umur harus lebih dari 18 tahun');
  } else {
    print('Anda memenuhi persyaratan umur.');
  }
}

void main() {
  try {
    validateAge(15);
  } catch (e) {
    print('Exception: $e');
  }
}
```

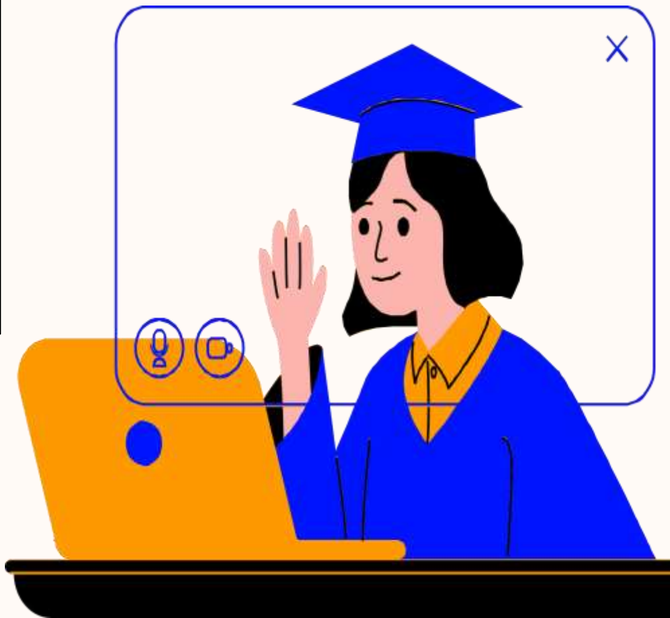
MOJADIPRO

Mengubah String Menjadi Ganda

Untuk mengubah string menjadi ganda menggunakan parse di Dart, kita perlu mengubah string menjadi tipe numerik (misalnya, integer atau double), kemudian menggandakan nilainya.

Berikut adalah contoh implementasinya :

```
void main() {
  String angkaString = '5';
  int angka = int.parse(angkaString);
  int hasil = angka * 2;
  String hasilString = hasil.toString();

  print('Angka awal: $angkaString');
  print('Hasil ganda: $hasilString');
}
```



MOJADIPRO

Null Aware Operators

Null aware operators adalah fitur dalam bahasa pemrograman Dart yang memungkinkan penanganan nilai null dengan lebih mudah dan aman.

Dalam Dart, terdapat dua null aware operators, yaitu operator "?." (null-aware access) dan operator "??=" (null-aware assignment).

1. Operator "?." (Null-aware access):

   Operator "?." digunakan untuk mengakses properti atau memanggil metode dari suatu objek tanpa harus khawatir objek tersebut bernilai null.

   Jika objek null, operator ini akan mengembalikan null, jika tidak, maka properti atau metode akan dievaluasi.

2. Operator "??=" (Null-aware assignment):

   Operator "??=" digunakan untuk menginisialisasi suatu variabel dengan nilai default jika variabel tersebut bernilai null. Jika variabel tidak null, maka nilainya tetap tidak berubah.

MOJADIPRO

Null Aware Operators

Berikut adalah contoh penggunaan operator "?." dalam Dart :

```
class Person {
  String? name;

  void sayHello() {
    print("Hello, $name!");
  }
}

void main() {
  Person? person;

  // Menggunakan null-aware access
  print(person?.name); // Output: null

  person = Person();
  person.name = "John";

  print(person?.name); // Output: John
  person?.sayHello(); // Output: Hello, John!
}
```

MOJADIPRO

Null Aware Operators

Berikut adalah contoh penggunaan operator "??=" dalam Dart :

```
void main() {
  int? count;

  // Menggunakan null-aware assignment
  count ??= 0;

  print(count); // Output: 0

  count = 5;

  count ??= 0; // Tidak akan mengubah nilai count

  print(count); // Output: 5
}
```

MOJADIPRO

Clima Location Refactoring

Clima Location Refactoring digunakan untuk mengelola lokasi atau geolokasi dalam aplikasi yang terintegrasi dengan layanan cuaca atau layanan geografis lainnya. Pendekatan ini memungkinkan pengembang untuk mengisolasi logika terkait lokasi ke dalam kelas-kelas yang terpisah, sehingga memudahkan perawatan dan pengujian.

Salah satu contoh implementasi Clima Location Refactoring di Dart dapat melibatkan aplikasi cuaca sederhana.

Dalam contoh ini, kita akan membagi kode menjadi beberapa kelas yang berbeda untuk mengelola lokasi.

MOJADIPRO

Clima Location Refactoring

1. Location class: Kelas ini bertanggung jawab untuk mendapatkan data lokasi pengguna dari perangkat dan mengelola data terkait lokasi.

   Contoh kode berikut menunjukkan contoh implementasi kelas Location :

```
class Location {
  double latitude;
  double longitude;

  Future<void> getCurrentLocation() async {
    // Kode untuk mendapatkan lokasi pengguna dari perangkat
    // menggunakan API lokasi yang tersedia di Dart
    // ...

    latitude = /* nilai latitude */;
    longitude = /* nilai longitude */;
  }
}
```

MOJADIPRO

Clima Location Refactoring

2. WeatherService class: Kelas ini bertanggung jawab untuk mengambil data cuaca berdasarkan lokasi pengguna.

   Contoh kode berikut menunjukkan contoh implementasi kelas WeatherService :

```
class WeatherService {
  Future<WeatherData> getWeatherData(double latitude, double longitude) async
{
    // Kode untuk mengambil data cuaca dari layanan cuaca
    // menggunakan latitude dan longitude yang diberikan
    // ...

    return /* objek WeatherData */;
  }
}
```

MOJADIPRO

Clima Location Refactoring

3. WeatherData class: Kelas ini berfungsi sebagai representasi data cuaca.

   Contoh kode berikut menunjukkan contoh implementasi kelas WeatherData :

```
class WeatherData {
  String city;
  double temperature;
  String condition;

  WeatherData({this.city, this.temperature, this.condition});
}
```

MOJADIPRO

Clima Location Refactoring

4. WeatherApp class: Kelas ini berperan sebagai kelas pengontrol utama yang menghubungkan komponen-komponen aplikasi cuaca.

   Contoh kode berikut menunjukkan contoh implementasi kelas WeatherApp :

```
class WeatherApp {
  Location _location;
  WeatherService _weatherService;

  WeatherApp() {
    _location = Location();
    _weatherService = WeatherService();
  }

  Future<void> getWeather() async {
    await _location.getCurrentLocation();
    WeatherData weatherData = await _weatherService.getWeatherData(
        _location.latitude, _location.longitude);

    // Kode untuk menampilkan data cuaca ke pengguna
    // ...
  }
}
```

MOJADIPRO

API

API (Application Programming Interface) digunakan untuk menyediakan serangkaian fungsi, kelas, dan metode yang memungkinkan komunikasi dan interaksi antara komponen perangkat lunak yang berbeda. API bertindak sebagai penghubung antara aplikasi yang sedang dibangun dan sistem atau layanan eksternal yang digunakan oleh aplikasi tersebut.

Contoh API yang sering digunakan dalam pengembangan aplikasi Dart adalah API RESTful. API RESTful memungkinkan aplikasi untuk berkomunikasi dengan server melalui permintaan HTTP seperti GET, POST, PUT, dan DELETE.

MOJADIPRO

API

Berikut adalah contoh sederhana penggunaan API RESTful di Dart :

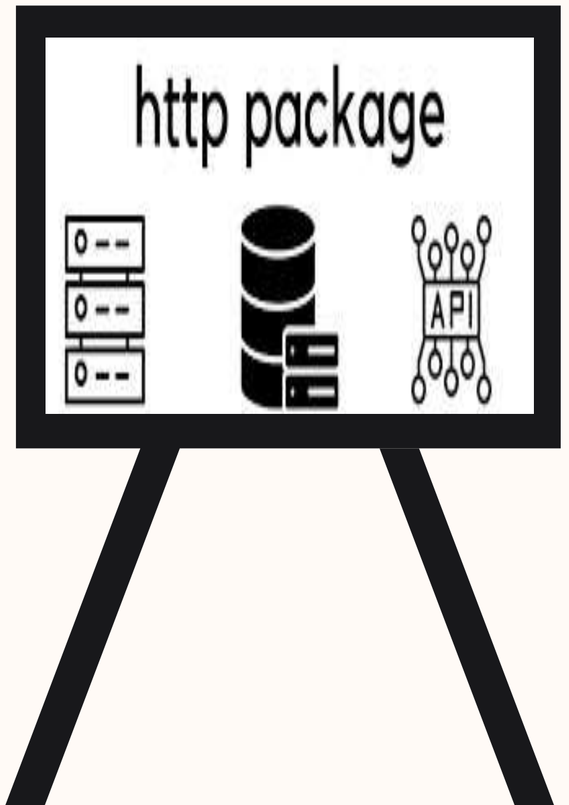
```
import 'dart:convert';
import 'package:http/http.dart' as http;

void main() async {
  // Membuat permintaan GET ke API
  var response = await http.get(Uri.parse('https://api.example.com/data'));

  // Mengecek kode status HTTP response
  if (response.statusCode == 200) {
    // Mengubah data JSON menjadi objek Dart
    var data = json.decode(response.body);

    // Memproses data
    var result = data['result'];
    var count = data['count'];

    print('Result: $result');
    print('Count: $count');
  } else {
    print('Error: ${response.statusCode}');
  }
}
```

MOJADIPRO

Networking Menggunakan HTTP Packages

Networking mengacu pada proses mengirim permintaan HTTP ke server dan menerima responsenya. Untuk melakukan komunikasi jaringan di Flutter, Anda dapat menggunakan paket HTTP

Paket http adalah paket yang populer dan sering digunakan untuk berkomunikasi dengan server melalui HTTP dalam aplikasi Flutter.



MOJADIPRO

Networking Menggunakan HTTP Packages

1. Tambahkan paket http ke file pubspec.yaml:

```
dependencies:
  http: ^0.13.3
```

2. Jalankan perintah flutter pub get

3. Impor paket http di file Dart yang relevan:

```
import 'package:http/http.dart' as http;
```

4. Gunakan fungsi-fungsi yang disediakan oleh paket http untuk melakukan permintaan HTTP. Misalnya, untuk melakukan permintaan GET ke URL tertentu, Anda dapat menggunakan http.get

```
http.Response response = await http.get(Uri.parse('https://example.com/api/data'));
```

MOJADIPRO

Networking Menggunakan HTTP Packages

5. Anda dapat mengakses responsenya dengan menggunakan properti body pada objek response

```
String responseBody = response.body;
```

6. Jangan lupa menangani situasi jika terjadi kesalahan dalam permintaan. Anda dapat memeriksa kode status responsenya untuk mengetahui apakah permintaan berhasil atau tidak

```
if (response.statusCode == 200) {
  // Permintaan berhasil, lakukan sesuatu dengan data responsenya.
} else {
  // Permintaan gagal, tangani kesalahan sesuai kebutuhan.
}
```

MOJADIPRO

Navigasi Dengan Navigator

1. Pastikan Anda menggunakan Flutter framework dan telah mengimpor paket Navigator

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

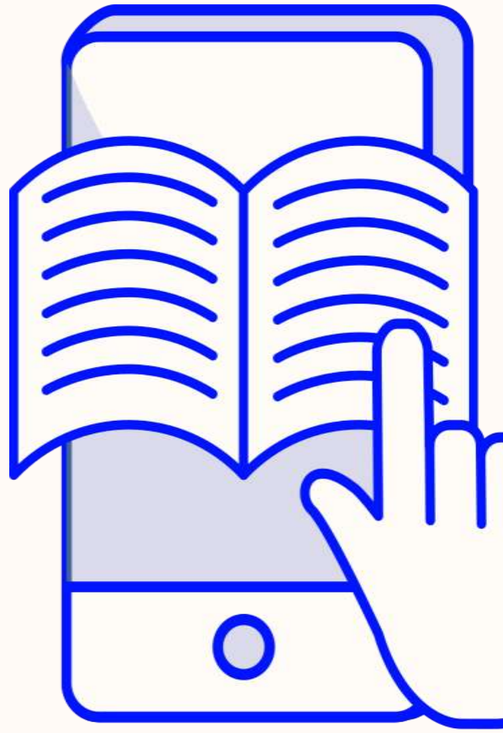
2. Tentukan beberapa layar yang ingin Anda navigasikan. Misalnya, Anda memiliki layar HomeScreen dan DetailScreen.

3. Di dalam HomeScreen, buat widget yang akan menavigasikan ke layar DetailScreen. Misalnya, Anda dapat menggunakan tombol yang ketika ditekan akan membuka layar DetailScreen:

```
ElevatedButton(
  child: Text('Buka Detail'),
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => DetailScreen()),
    );
  },
)
```

MOJADIPRO

XML



XML adalah singkatan dari "eXtensible Markup Language". XML adalah format yang digunakan untuk mengatur, menyimpan, dan mengirim data secara terstruktur. XML menggunakan tag untuk mendefinisikan elemen-elemen data dan atribut-atribut untuk memberikan informasi tambahan tentang elemen-elemen tersebut.

Dalam pemrograman Dart, XML sering digunakan sebagai format pertukaran data antara aplikasi atau untuk menyimpan konfigurasi aplikasi. Dart menyediakan pustaka bawaan yang disebut xml yang memungkinkan Anda untuk membaca, menulis, dan memanipulasi dokumen XML.

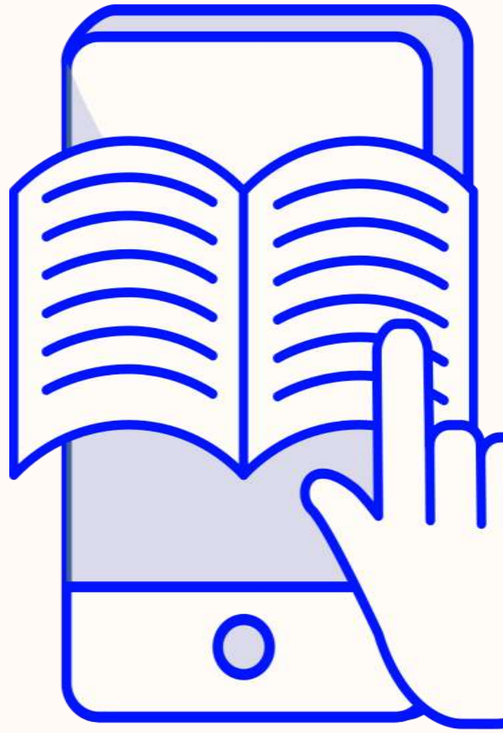
MOJADIPRO

JSON

JSON (JavaScript Object Notation) adalah sebuah format data yang ringan dan mudah dibaca oleh manusia serta mudah diinterpretasikan oleh mesin.

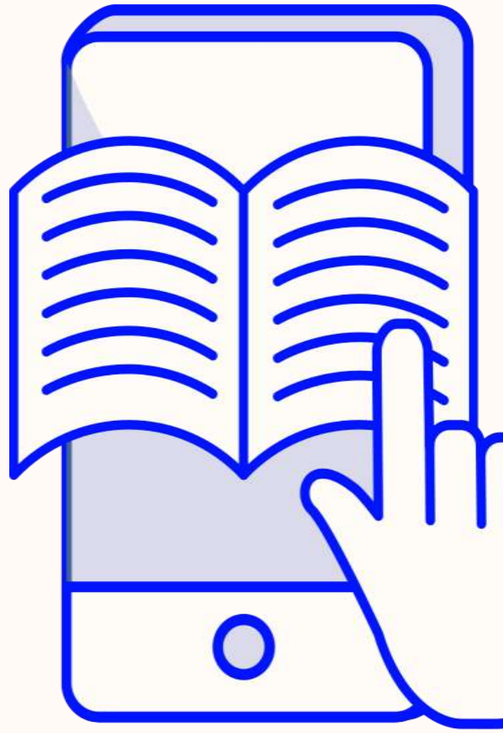
JSON umumnya digunakan untuk pertukaran data antara server dan klien web, serta dalam penyimpanan dan pengiriman data dalam aplikasi web.

JSON memiliki struktur berbentuk pasangan "key" dan "value". Data dalam format JSON diwakili oleh objek dan larik yang terdiri dari kumpulan pasangan "key-value" yang terurut.



MOJADIPRO

JSON



Berikut adalah contoh JSON :

1. Objek JSON :

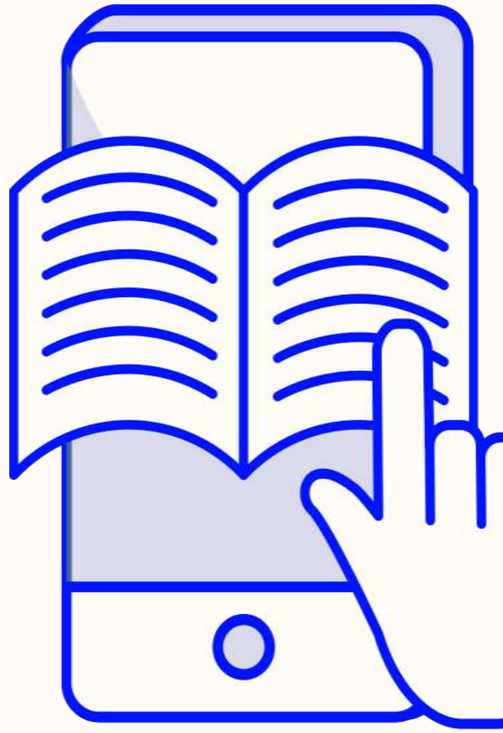
```
{
  "nama": "John Doe",
  "umur": 30,
  "alamat": "Jl. ABC No. 123",
  "hobi": ["berenang", "membaca"]
}
```

2. Objek JSON :

```
[ {    "nama": "Produk A",    "harga": 10000,    "stok": 50  },
{    "nama": "Produk B",    "harga": 15000,    "stok": 20  }]
```

MOJADIPRO

Json Parsing Dan Dynamic Types



JSON Parsing adalah proses mengonversi data dalam format JSON menjadi objek atau struktur data yang dapat digunakan dalam bahasa pemrograman.

Dynamic types adalah jenis data yang dapat berubah secara dinamis. Variabel dengan tipe data Dynamic dapat mengambil nilai dari tipe data apa pun. Dynamic types digunakan ketika tipe data dari nilai yang diharapkan tidak diketahui pada saat kompilasi atau ketika Anda perlu memanipulasi data yang dapat berubah tipe.

Dalam konteks JSON Parsing, Dynamic types sering digunakan karena struktur data JSON dapat bervariasi. Ketika Anda mengurai data JSON, Anda mungkin tidak tahu tipe data yang tepat dari setiap elemen di awal.

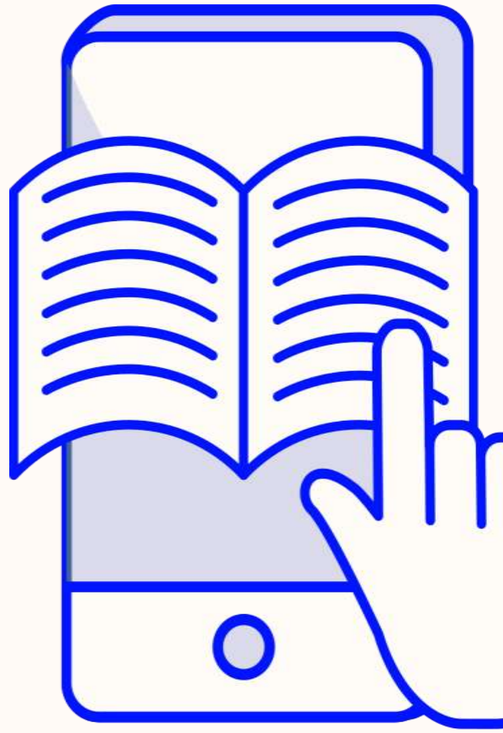
Dalam Dart, untuk mengurai JSON, Anda dapat menggunakan pustaka bawaan bernama dart:convert. Pustaka ini menyediakan kelas-kelas yang dapat membantu Anda mengurai dan menghasilkan data JSON.

MOJADIPRO

Json Parsing Dan Dynamic Types

Contoh penggunaan JSON Parsing dengan Dynamic types dalam Dart adalah sebagai berikut :

```
import 'dart:convert';

void main() {
  String jsonStr = '{"name": "John", "age": 30, "isStudent": true}';

  dynamic jsonData = jsonDecode(jsonStr);

  String name = jsonData['name'];
  int age = jsonData['age'];
  bool isStudent = jsonData['isStudent'];

  print('Name: $name');
  print('Age: $age');
  print('Is Student: $isStudent');
}
```



TERIMA KASIH