BAHAN AJAR

PEMROGRAMAN BERBASIS MOBILE

“[Flutter dan Dart]”

Dosen Pengampu : Irfan Sriyono Putro, S.T., M.Si.

NAMA : **NAUFAL MIRZA ALDILLA**
NPM : **2310631170144**
KELAS : **5F - INFORMATIKA**

PROGRAM STUDI INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS SINGAPERBANGSA KARAWANG

2025

BAHAN AJAR FLUTTER & DART – UTS PEMROGRAMAN MOBILE

Disusun Oleh:

Naufal Mirza Aldilla

Mata Kuliah:

Pemrograman Berbasis Mobile

Dosen Pengampu

Irfan Sriyono Putro, S.T., M.Si.

DAFTAR ISI

PENDAHULUAN

Perkembangan teknologi saat ini bikin hampir semua hal bisa diakses lewat perangkat mobile — mulai dari belanja, belajar, sampai kerja. Karena itu, pengembangan aplikasi mobile jadi bidang yang terus berkembang pesat.
Masalahnya, ada banyak platform seperti Android, iOS, web, dan desktop. Kalau kita harus bikin aplikasi terpisah untuk tiap platform, pasti butuh waktu dan biaya besar banget. Nah, di sinilah muncul framework **cross-platform** seperti **Flutter**.

Flutter adalah framework buatan Google yang memungkinkan pengembang membuat satu kode untuk berbagai platform. Flutter dikenal karena performanya cepat, tampilannya halus, dan punya fitur **hot reload**, jadi setiap perubahan kode langsung bisa dilihat tanpa nunggu lama.

Dart adalah bahasa pemrograman yang digunakan Flutter. Dart juga dikembangkan oleh Google dan punya fitur modern seperti **null safety**, **arrow function**, dan **koleksi data (List, Set, Map)** yang bikin kode lebih aman dan efisien [1][2].

Tujuan utama bahan ajar ini adalah supaya mahasiswa bisa paham dasar-dasar Flutter dan Dart, terutama hal-hal yang sering keluar di UTS, seperti:

- cara deklarasi variabel (var, final, const),
- konsep **null safety**,
- struktur data **List, Set, Map**,
- fungsi panah (arrow function),
- hubungan antara Flutter dan Dart,
- perbedaan widget (stateless vs stateful),
- dan penggunaan elemen layout dasar seperti Row, Column, Scaffold, dan AppBar

BAB 1
DASAR PEMROGRAMAN DART

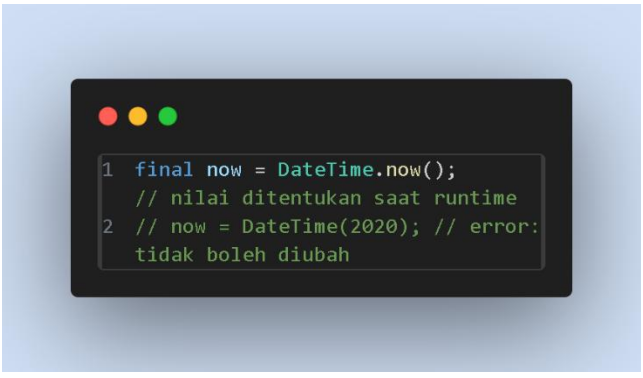
1.1 Deklarasi variabel (var, final, const)

1.1.1 Var

- Menyatakan variabel dengan *type inference* (kompiler menurunkan tipe dari nilai awal).
- Nilai dapat diubah (mutable) setelah inisialisasi kecuali diberi anotasi final.
- Jika tidak diinisialisasi saat deklarasi, tipe menjadi dynamic sampai diinisialisasi.
- Contoh:

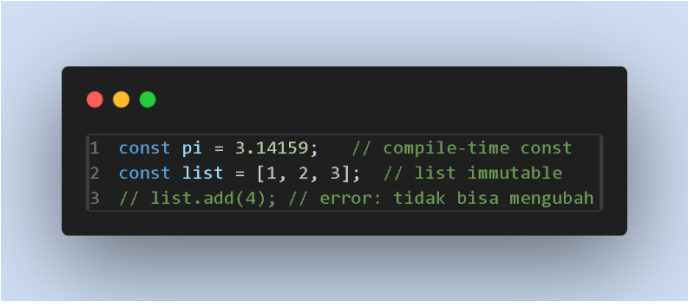
1.1.2 Final

- Menyatakan variabel yang hanya sekali diinisialisasi (immutable setelah inisialisasi).
- Nilai bisa diketahui saat runtime (bisa berasal dari perhitungan runtime), tetapi setelah di-set tidak bisa diubah.
- Contoh:



1.1.3 Const

- Menyatakan nilai konstan yang bersifat compile-time constant (konstanta pada waktu kompilasi).
- Objek yang `const` tidak boleh tergantung pada nilai runtime.
- `const` bisa digunakan untuk membuat object tree immutable yang sama instance-nya jika nilainya sama (canonicalized).
- Contoh :



1.1.4 Catatan

const lebih "ketat" daripada final. final = hanya di-set sekali; const = nilai tetap sejak kompilasi.

1.2 NULL SAFETY DI DART: OPERATOR ? DAN !

1.2.1 Mengapa Null Safety Penting

Null reference errors (Null Pointer Exceptions) adalah sumber bug umum. Dart memperkenalkan *sound null safety*: variabel non-nullable tidak boleh berisi null. Ini memaksa developer menangani kemungkinan null secara eksplisit. Fitur yang mencegah *null reference errors* (salah satu sumber bug paling umum). Dengan null safety, tipe variable dibagi:

- Non-nullable (mis. int) — tidak boleh null.
- Nullable (mis. int?) — boleh berisi null.

Kompiler memeriksa alur program untuk memastikan non-nullable tidak pernah diberi null.

1.2.2 Nullable (?) dan Non-null Assertion (!)

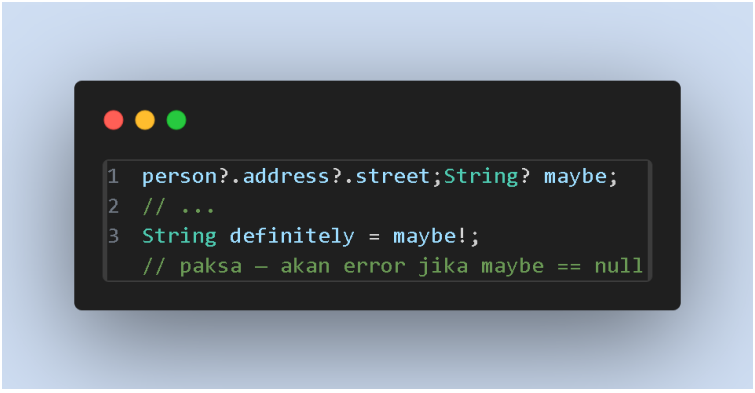
- **Tanda ?**: memungkinkan variabel berisi null. Tipe menjadi T? (nullable).
- String? maybeName;
- **Operator akses aman ?.**: memanggil method/properti hanya bila objek tidak null.
- print(maybeName?.toUpperCase()); // returns null if maybeName == null
- **Operator ! (bang)**: memaksa menganggap nilai bukan null. Jika ternyata null → runtime error.
- print(maybeName!.length); // BE CAREFUL: runtime error jika null

Operator ? (nullable) — penggunaan

- Menandai tipe bisa null.

Operator ! (bang / null assertion)

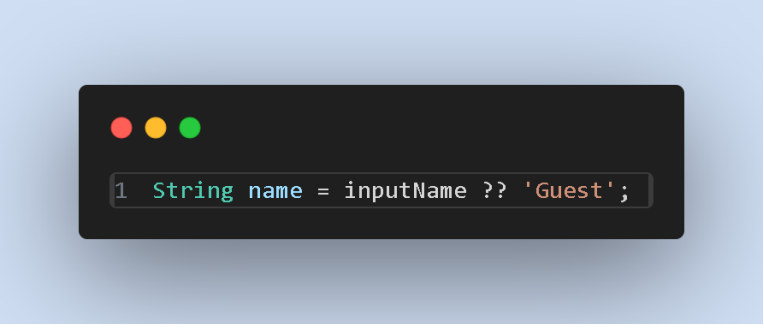
- Mengatakan ke kompiler: "Saya yakin nilai ini bukan null pada runtime" — dan memaksa konversi dari T? -> T.
- Jika kenyataannya null saat runtime, akan melempar LateInitializationError atau NoSuchMethodError (runtime exception).



Operator lain terkait null safety

- ?. — safe navigation: hanya panggil jika bukan null; hasil null jika objek null.

- ?? — if-null operator: berikan nilai default jika kiri null.



- ??= — assign-if-null.

Kegunaan ringkas

- ? menandai kemungkinan null sehingga kompiler memaksa pengecekan eksplisit.
- ! mem-bypass pemeriksaan null (berbahaya bila salah digunakan).

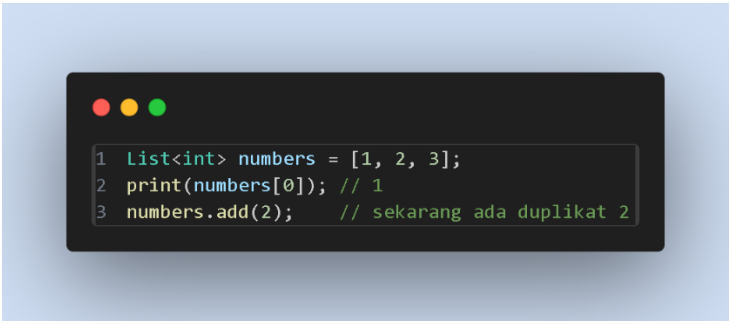
1.3 PERBEDAAN LIST, SET, DAN MAP DI DART

1.3.1 Konsep Umum

- **List**: ordered collection; indexed; allows duplicates.
- **Set**: unordered collection of unique items; fast membership test.
- **Map<K, V>**: key-value pairs; keys unique.

1.3.2 List

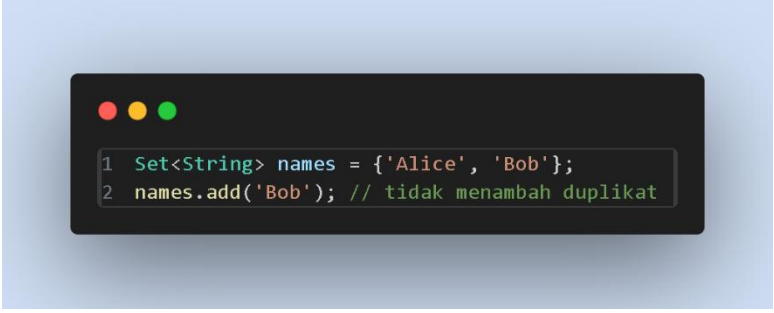
- Koleksi terurut (ordered) elemen yang bisa berisi duplikat.
- Diindeks (akses via index).
- Contoh:



1.3.3 Set

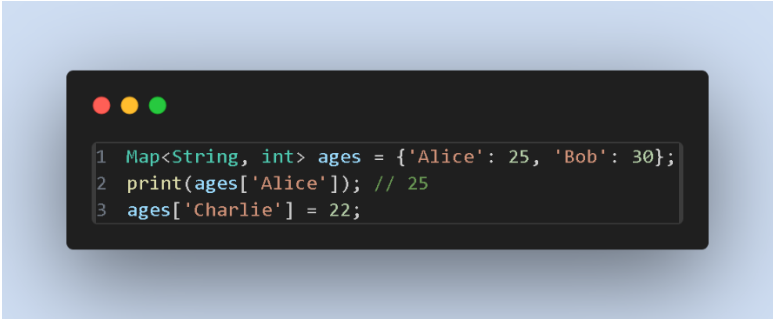
- Koleksi *unordered* (atau tidak bergantung urutan) yang **unik** — tidak boleh ada duplikat nilai (berdasarkan equality/hash).
- Berguna saat ingin memeriksa keanggotaan atau memastikan keunikan.

- Contoh:



1.3.4 Map

- Koleksi pasangan kunci-nilai (key → value). Mirip dictionary.
- Kunci unik; akses cepat via kunci.
- Contoh:



1.3.5 Ringkasan kebergunaan

- Gunakan **List** jika urutan dan index penting atau duplikat diizinkan.
- Gunakan **Set** jika butuh keunikan item dan operasi membership cepat.
- Gunakan **Map** jika butuh lookup berdasarkan key (mis. id → objek).

1.4 ARROW FUNCTION (=>) DI DART

1.4.1 Makna & Sintaks

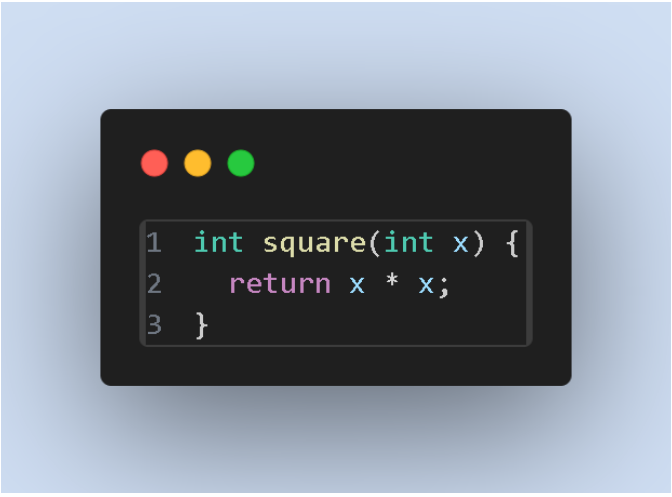
Arrow function adalah syntactic sugar untuk fungsi yang mengandung satu expression.

Berguna untuk menulis callback singkat dan meningkatkan keterbacaan.

- Arrow function adalah sintaks singkat untuk fungsi yang hanya mengembalikan satu ekspresi. Bentuk: (params) => expression;
- Ekivalen dengan { return expression; } tapi lebih ringkas.

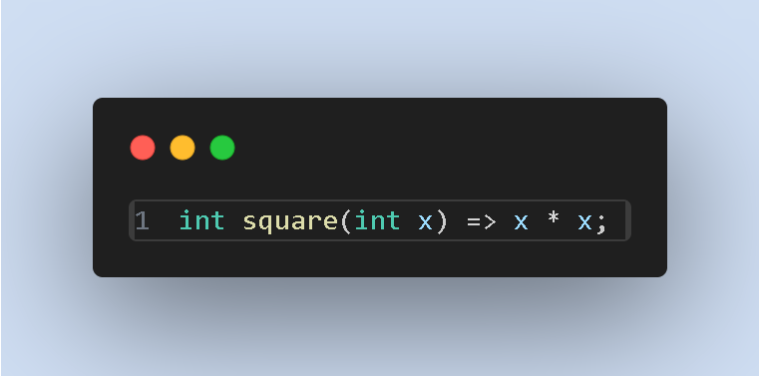
Contoh

- Fungsi biasa:



```
int square(int x) {
  return x * x;
}
```

- Dengan arrow:



```
int square(int x) => x * x;
```

- Di Flutter sering dipakai untuk builder pendek:

```
ElevatedButton(
  onPressed: () => print('klik!'),
  child: Text('Tap'),
);
```

1.4.2 Kapan Tidak Menggunakan Arrow

- Jika fungsi memerlukan lebih dari satu statement (loop, try/catch), gunakan blok {} biasa.
- Jika butuh dokumentasi inline atau komentar bertingkat, blok biasa lebih cocok.

BAB 2
PENGENALAN FLUTTER DAN HUBUNGANNYA DENGAN DART

2.1 Pengertian Flutter dan Dart

2.1.1 Flutter: Definisi & Arsitektur Singkat

Flutter = UI toolkit + rendering engine (Skia) + framework widget. Arsitektur: **Engine (C++), Framework (Dart), Embedder (platform-specific)**. Engine bertanggung jawab merender, framework (ditulis Dart) menyediakan widget, dan embedder menghubungkan dengan sistem operasi.

Kelebihan teknis:

- rendering independen platform → konsistensi UI.
- Hot reload (JIT) saat development → efisiensi iterasi.
- AOT compile → performa produksi baik.

Referensi teknis & pengantar: AWS, BiznetGio, CodingStudio.

2.1.2 Dart: Definisi & Fitur Kunci

Dart: bahasa yang berorientasi objek, support null safety, async/await, futures/streams, dan kompilasi JIT/AOT. Dirancang agar mudah dipelajari (mirip Java/C#) namun modern untuk UI development.

2.1.3 Hubungan Operasional

- Flutter *framework* ditulis di Dart. Pengembang menggunakan Dart untuk menulis widgets, state management, networking, business logic.
- Tools Flutter (flutter command), plugin, dan paket-paket pub.dev semua berbasis Dart.

2.1.4 Contoh Alur Kerja

1. Tulis kode Dart (widgets, logic).
2. Flutter tool meng-compile & melakukan hot reload (JIT).
3. Untuk release, Flutter AOT-compile ke native machine code.

2.1.5 Hubungan Keduanya

2.1.5.1 Dart

- Bahasa pemrograman yang dibuat oleh Google.
- Dirancang untuk performa, produktivitas, dan mendukung kompilasi ahead-of-time (AOT) dan just-in-time (JIT).
- Fitur: null safety, async/await, tiped, cepat di compile ke native.

2.1.5.2 Flutter

- UI toolkit (framework) open-source dari Google untuk membangun antarmuka aplikasi **cross-platform** (mobile, web, desktop) dengan satu codebase.
- Menyediakan sistem rendering sendiri (bukan wrapper native) — memberikan kontrol penuh atas UI dan performa konsisten.

2.1.5.3 Hubungan

- Flutter dibangun menggunakan **Dart**: aplikasi Flutter ditulis dalam Dart, widget Flutter menggunakan API Dart.
- Flutter menggunakan kemampuan kompilasi Dart:
  - **JIT** selama development → mendukung *hot reload*.
  - **AOT** untuk build release → performa native-like.

2.2 3 KEUNTUNGAN UTAMA FLUTTER

Kita akan mendalami tiga keunggulan utama dan memberikan detail teknis serta implikasi praktis.

2.2.1 Hot Reload — Cepat Iterasi UI

- **Apa**: Memungkinkan melihat perubahan kode langsung tanpa restart.
- **Mengapa penting**: Menghemat waktu desain & debugging; mempercepat prototyping.
- **Keterbatasan**: Hot reload tidak selalu mereflect perubahan yang menyentuh initializers compile-time (const), atau perubahan native layer.

2.2.2 Single Codebase untuk Banyak Platform

- **Apa**: Satu kode basis Dart/Flutter untuk Android/iOS/Web/Desktop.
- **Manfaat**: Menurunkan biaya pemeliharaan, mempersingkat waktu pengembangan lintas platform.
- **Catatan**: Terkadang butuh penyesuaian platform-specific (permissions, UI conventions).

2.2.3 Performa & Konsistensi UI (Rendering dengan Skia)

- **Apa**: Flutter menggambar semua UI melalui enginenya sendiri (Skia).
- **Manfaat**: Rendering konsisten, performa mendekati native, animasi lebih halus.
- **Trade-off**: Ukuran binary biasanya lebih besar dibanding aplikasi native sederhana.

Tambahan jika butuh:

- **Widget kaya dan customizable** — banyak widget siap pakai + mudah dimodifikasi.

- **Komunitas & ekosistem paket** — pub.dev banyak paket pihak ketiga.
- **UI konsisten antar platform** — desain tampilan sama di semua platform jika diinginkan.

BAB 3
KONSEP WIDGET DAN STRUKTUR UI DASAR

3.1 Semua komponen UI adalah widget

3.1.1 Widget sebagai Unit Komposisi

Widget adalah building blocks UI: StatelessWidget, StatefulWidget, InheritedWidget, serta banyak widget render-level (Container, Row, Text, Image, dll.). Widget dapat disusun bersarang membentuk pohon widget (widget tree).

3.1.2 Types of Widgets (Ringkasan)

- **StatelessWidget**: tidak memiliki internal mutable state.
- **StatefulWidget**: memiliki State object yang hidup sampai widget dihapus.
- **InheritedWidget**: untuk menyuntikkan data ke subtree (basis pattern untuk Provider).
- **RenderObjectWidget**: level lebih rendah berhubungan dengan render pipeline.

3.1.3 Lifecycle Singkat StatefulWidget

- createState() → initState() → didChangeDependencies() → build() → setState() → dispose().

3.1.4 Perbedaan mendasar antara StatelessWidget dan StatefulWidget

3.1.4.1 StatelessWidget

- Widget yang *tidak memiliki state internal* yang berubah-ubah selama lifecycle-nya.
- Renderannya sepenuhnya bergantung pada input (properties) yang diberikan dari luar.
- Tidak memiliki object State yang memanggil rebuild via setState.
- Contoh: Text, Icon, custom widget yang hanya menerima props dan tidak berubah sendiri.

3.1.4.2 StatefulWidget

- Widget yang **memiliki state mutable** — artinya internal data bisa berubah seiring waktu dan memicu rebuild.
- Terdiri dari dua kelas: StatefulWidget (immutable) dan State<T> (tempat menyimpan state dan method setState()).
- Cocok saat UI butuh merespon event, timer, network response, atau input pengguna yang mengubah tampilan.

3.1.4.3 Perbedaan lifecycle & arsitektur singkat

- StatelessWidget cuma build() sekali (atau saat parent meminta rebuild).
- StatefulWidget: ada createState() → instance State yang mempunyai initState(), build(), dispose(), dll. setState() memicu build() ulang.

3.2 PERBEDAAN MENDASAR: STATELESSWIDGET VS STATEFULWIDGET

3.2.1 StatelesWidget — Sifat & Kasus Penggunaan

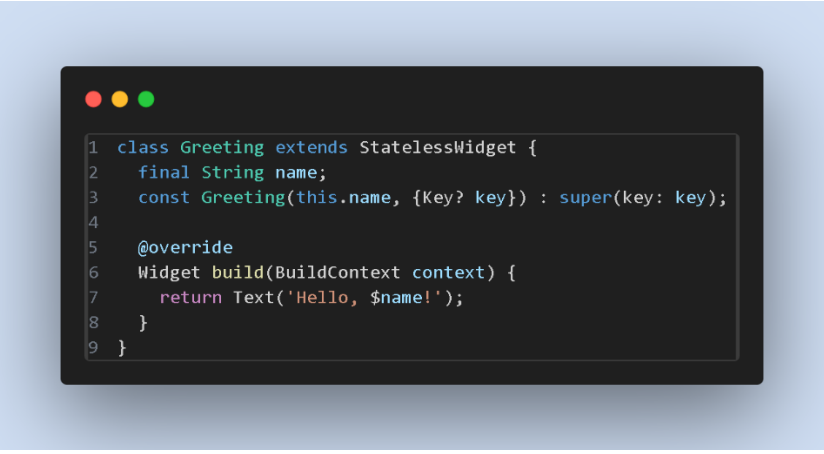
- **Sifat**: Immutable configuration; build() pure function dari input props.
- **Use cases**: Icon, logo, static labels, stateless layout.

3.2.2 StatefulWidget — Sifat & Kasus Penggunaan

- **Sifat**: Memiliki State<T> yang mutable; update via setState().
- **Use cases**: Form input, animations, timers, network-loaded content.

3.2.3 Gunakan StatelessWidget ketika

- Komponen hanya menampilkan data yang diberikan lewat parameter dan tidak perlu menyimpan atau mengubah state internal.
- Contoh: tombol statis, header teks, ikon, layout tetap.



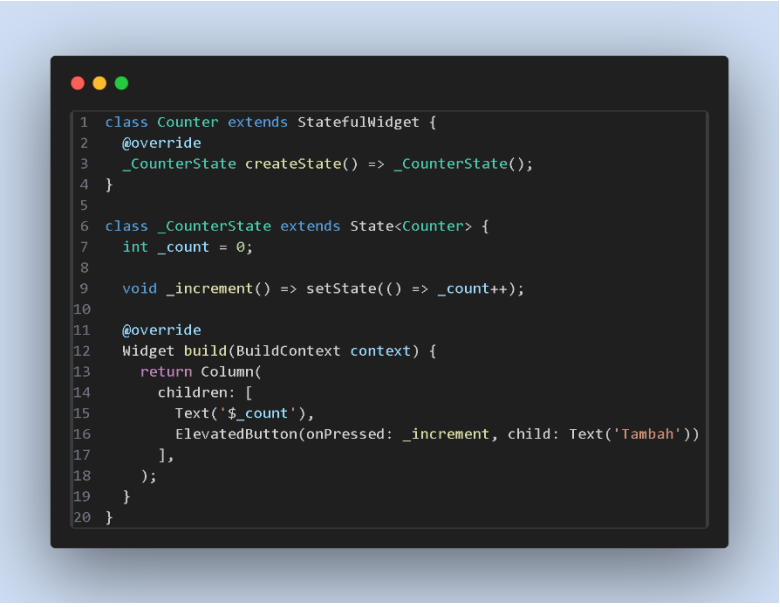
```dart
class Greeting extends StatelessWidget {
  final String name;
  const Greeting(this.name, {Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Text('Hello, $name!');
  }
}
```

3.2.4 Gunakan StatefulWidget ketika

- Komponen butuh merespon perubahan internal: input form, animation, timer, pengambilan data asinkron yang memengaruhi tampilan, toggle, counter.

- Contoh: counter, form with validation, widget yang menyimpan apakah tombol sudah ditekan.



```dart
class Counter extends StatefulWidget {
  @override
  _CounterState createState() => _CounterState();
}

class _CounterState extends State<Counter> {
  int _count = 0;

  void _increment() => setState(() => _count++);

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Text('$_count'),
        ElevatedButton(onPressed: _increment, child: Text('Tambah'))
      ],
    );
  }
}
```

3.2.5 StatelessWidget — contoh nyata

- Tampilan header: logo + title
- Card statis yang menampilkan info dari server (jika tidak ada interaksi)

3.2.6 StatefulWidget — contoh nyata

- Form input (validasi dinamis)
- Animasi yang berjalan (AnimationController)
- Live feed yang menerima updates (StreamBuilder with state)

3.2.7 Strategi Pemisahan State

- Local state → pakai StatefulWidget.
- Shared app state → Provider/Bloc.
- UI state (isExpanded, selectedIndex) → prefer local state.

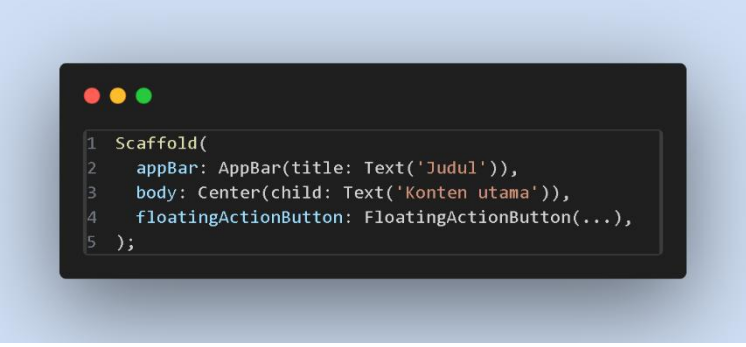
3.3 SCAFFOLD, APPBAR, DAN BODY

3.3.1 Fungsi dari widget Scaffold, AppBar, dan body

3.3.1.1 Scaffold

- Widget yang menyediakan *struktur dasar* halaman aplikasi Material: area body, AppBar, drawer, bottom navigation, floating action button, snackbar, dll.
- Tujuan: mempermudah membangun layout standar Material Design tanpa mengatur posisi tiap bagian manual.

- Contoh:



3.1.1.2 AppBar

- Bar header di bagian atas screen (biasanya menampilkan judul, action icons, navigation).
- Properti populer: title, actions, leading, backgroundColor, elevation.
- Biasanya disisipkan ke Scaffold.appBar.

3.1.1.3 body

- Properti dari Scaffold yang menampung *konten utama* layar (widget tree yang ditampilkan di area setelah AppBar).
- Di sinilah sebagian besar UI halaman ditempatkan.

3.1.2 Scaffold: Kerangka Utama

Scaffold memberikan area struktur material: appBar, body, floatingActionButton, drawer, bottomNavigationBar, snackbar, dll.

3.1.3 AppBar: Header

- Menyediakan title, leading icon (back/menu), action buttons.
- Bisa customized: elevation, backgroundColor, shape.

3.1.4 body: Konten Utama

- Area tempat meletakkan widget layout (Column/Row/ListView/Stack).
- Jangan menerapkan logic heavy di build(); pindahkan ke state atau service.

BAB 4

LAYOUT DAN PENYUSUNAN TAMPILAN

4.1 Konsep layout di Flutter

4.1.1 Perbedaan Sumbu

- **Row** → main axis = horizontal, cross axis = vertical.
- **Column** → main axis = vertical, cross axis = horizontal.

4.1.2 mainAxisAlignment

Mengatur bagaimana anak widget diposisikan sepanjang main axis.

- start, center, end, spaceBetween, spaceAround, spaceEvenly.

4.1.3 crossAxisAlignment

Mengatur alignment di sumbu silang.

- start, center, end, stretch, baseline.

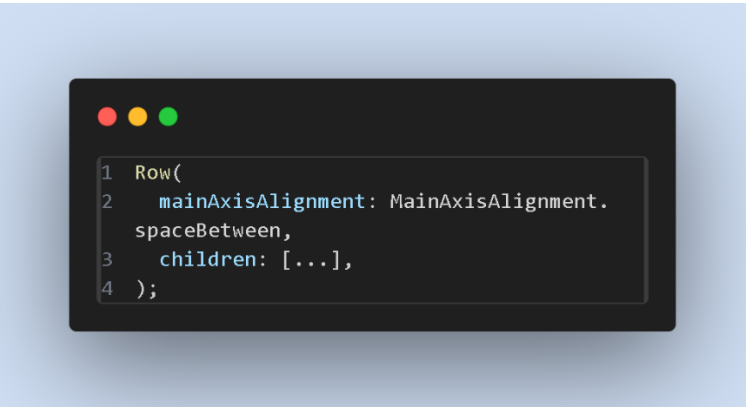
4.2 Perbedaan Row dan Column; mainAxisAlignment dan crossAxisAlignment

4.2.1 Perbedaan utama

- Row: menyusun child widgets **secara horizontal** (dari kiri ke kanan pada bahasa LTR).
- Column: menyusun child widgets **secara vertikal** (dari atas ke bawah).

4.2.2 mainAxisAlignment

- Mengatur penempatan child sepanjang *main axis* (untuk Row → horizontal, untuk Column → vertical).
- Nilai umum: start, center, end, spaceBetween, spaceAround, spaceEvenly.
- Contoh:



```
Row(
  mainAxisAlignment: MainAxisAlignment.
spaceBetween,
  children: [...],
);
```

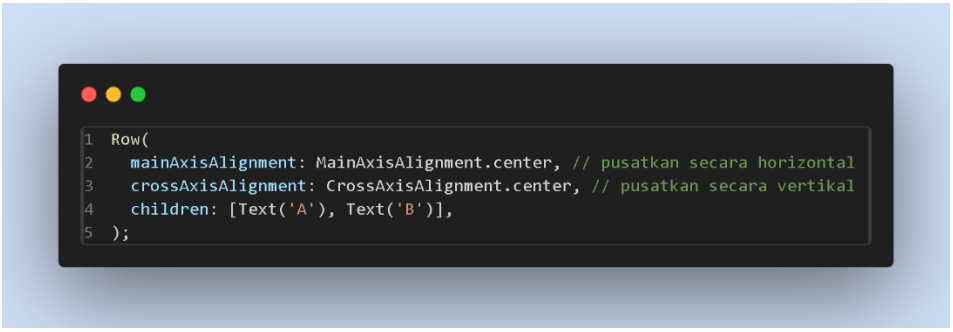
4.2.3 crossAxisAlignment

- Mengatur penempatan child sepanjang *cross axis* (untuk Row → vertical, untuk Column → horizontal).
- Nilai umum: start, center, end, stretch, baseline (baseline hanya untuk teks).

- Contoh:

```
Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [...],
);
```

4.2.4 Contoh ilustratif



```
Row(
  mainAxisAlignment: MainAxisAlignment.center, // pusatkan secara horizontal
  crossAxisAlignment: CrossAxisAlignment.center, // pusatkan secara vertikal
  children: [Text('A'), Text('B')],
);
```

BAB 5
STATE DAN MANAJEMEN PERUBAHAN DATA

5.1 Pengertian state dalam Flutter

5.1.1 Definisi State

State = data yang mempengaruhi tampilan/behavior UI. Contoh: apakah user login, nilai counter, hasil fetch API.

5.1.2 Mengapa Penting?

- Konsistensi UI: perubahan data harus tercermin di UI secara akurat.
- Performa: pembaruan state harus terlokalisasi agar tidak menyebabkan rebuild tidak perlu.
- Skalabilitas: aplikasi besar perlu pola manajemen state supaya maintainable.

5.1.3 Pola Manajemen State (overview)

- Local state: setState() di StatefulWidget.
- InheritedWidget → Provider: dependency injection sederhana.
- BLoC (Business Logic Component) menggunakan Streams.

Redux, MobX, Riverpod: solusi lebih kuat untuk aplikasi skala besar

5.1.4 Apa itu "state" dan mengapa manajemen state penting

- **State** = data yang menggambarkan kondisi UI saat ini. Misal: data form, apakah tombol dicentang, hasil fetch jaringan, halaman aktif, daftar item, nilai counter, dsb.
- **Mengapa penting**
  - UI harus selalu sinkron dengan state. Ketika state berubah, UI harus berubah secara benar dan efisien.
  - Manajemen state memastikan:
    - Konsistensi UI (tidak ada tampilan yang stale).
    - Performa: meminimalkan rebuild yang tidak perlu.
    - Scalability: aplikasi besar butuh pola pengelolaan state yang bisa diprediksi (mis. Provider, Bloc, Riverpod, Redux).
    - Pemisahan tanggung jawab: logika bisnis terpisah dari tampilan.

5.1.5 Contoh masalah tanpa manajemen state baik

- Banyak widget saling bergantung pada data global; perubahan satu bagian memicu rebuild seluruh tree → performa buruk.
- Sulit men-trace asal perubahan data → bug dan maintenance sulit.

5.2 CARA KERJA SETSTATE() PADA STATEFULWIDGET

5.2.1 Mekanisme

- setState(fn) menandai object State sebagai dirty → Flutter scheduler menjadwalkan rebuild.
- Pada frame berikutnya, framework memanggil build() lagi pada widget tersebut (dan subtree-nya) sehingga UI diperbarui.

5.2.2 Kapan Panggil setState()?

- Hanya untuk perubahan state yang mempengaruhi build output.
- Jangan panggil setState() dalam build() (akan loop rebuild).
- Hindari operasi expensive di dalam setState() body — ubah hanya state, bukan menjalankan work synchronous/IO.

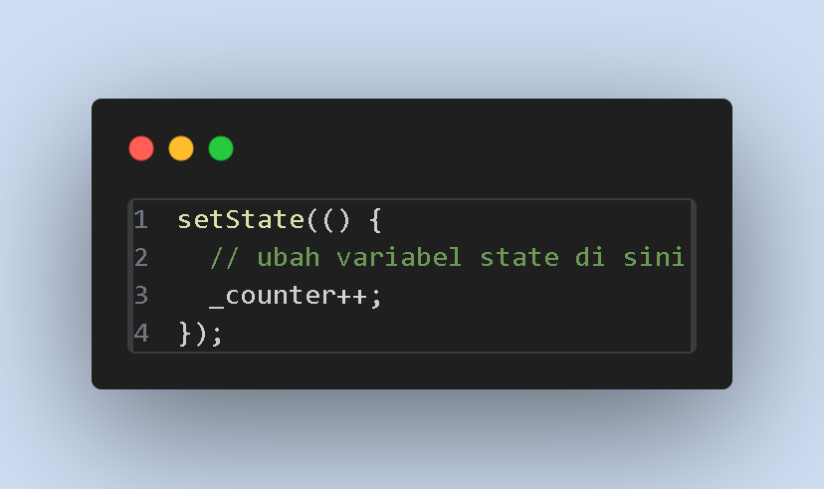
5.2.3 Tips & Anti-pattern

- Jangan menyimpan large objects in State yang tidak perlu (use providers/services).
- Untuk state yang digunakan oleh banyak widget, gunakan global manager (Provider, Riverpod).

5.2.4 Cara kerja setState() pada StatefulWidget

- setState() adalah method pada class State<T> yang dipakai untuk memberi tahu Flutter bahwa state internal telah berubah dan widget perlu di-rebuild.

- Cara umum:



```
setState(() {
  // ubah variabel state di sini
  _counter++;
});
```

- **Mekanisme**
  - Developer memanggil setState(fn).
  - fn dieksekusi segera (biasanya developer melakukan perubahan pada field state di dalamnya).
  - Flutter menandai object Element yang bersangkutan sebagai dirty (perlu build ulang).
  - Pada next frame rendering, Flutter memanggil build() pada state tersebut sehingga UI diperbarui.
- **Catatan penting**
  - Jangan panggil setState() jika tidak ada perubahan yang memengaruhi UI — hindari rebuild yang tidak perlu.
  - Panggil setState() hanya di dalam object State (atau method yang memiliki akses ke context state). Jangan panggil di build() synchronous loop sewenang-wenang.
  - Operasi berat sebaiknya dilakukan di luar setState(); hanya lakukan perubahan state ringkas di dalam callback.
    - setState() tidak menjamin synchronous render pada saat dipanggil — ia menandai kebutuhan rebuild, actual render terjadi nanti oleh framework.

5.2.5 Contoh sederhana

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

class _MyWidgetState extends State<MyWidget> {
  int _counter = 0;

  void _increment() {
    setState(() {           // ubah state dan minta rebuild
      _counter++;
    });
    // setelah ini, build() akan dipanggil ulang
  }

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

DAFTAR PUSTAKA

https://aws.amazon.com/id/what-is/flutter/

https://codingstudio.id/blog/flutter-adalah-kelebihan-dan-kekurangan/

https://www.biznetgio.com/news/apa-itu-flutter

https://www.dicoding.com/academies/191-memulai-pemrograman-dengan-dart

https://www.revou.co/kosakata/dart