

LAPORAN PRAKTIKUM
SE-05-GAB
“Pemrograman Perangkat Bergerak”

Dosen Pengampu:
Ahmad Muzakki., S.Kom., M.Kom.



Disusun oleh:
Deo Farady Santoso – 1201220447

PROGRAM STUDI REKAYASA PERANGKAT LUNAK
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY SURABAYA
2024

DAFTAR ISI

1. Persiapan.....	1
2. Instalasi dan Konfigurasi Flutter dan Dart	1
2.1. Install Flutter	1
2.2. Perbarui Path	1
2.3. Install Android Studio	2
2.4. Setup Android Studio.....	2
2.5. Install License.....	4
3. Basic Dart	4
3.1. Soal 1	4
3.1.1. Struktur Kode.....	5
3.2. Soal 2	7
3.2.1. Struktur Kode.....	9
3.3. Soal 3	12
3.3.1. Struktur Kode.....	13
4. Basic Layout	15
4.1. Teks.....	15
4.2. Container	15
4.3. Row.....	16
4.4. Column	17
4.5. List View.....	18
4.6. Grid View	19
4.7. Stack.....	20
5. Advance Layout.....	21
5.1. Nested Row/Column	21
5.2. Tab View	21
5.2.1. Tab Bar.....	21
5.2.2. Tab Bar View	22
5.3. Page View	22
5.4. Safe Area	22
6. User Interaction	23
6.1. Stateful & Stateless Widget.....	23
6.2. Button	24
6.2.1. Flat Button	24

6.2.2. RaisedButton or ElevatedButton	24
6.2.3. OutlineButton	24
6.2.4. FloatingActionButton	25
6.2.5. DropdownButton	25
6.2.6. IconButton	25
6.2.7. PopupMenuButton	26
6.3. SnackBar	26
6.4. Dialog.....	27
6.4.1. Simple Dialog	27
6.4.2. Alert Dialog.....	27
6.5. Menu	28
6.5.1. Bottom Navigation Bar	28
6.5.2. App Bar	28
7. Navigation	29
7.1. Package.....	29
7.2. Navigation	29
7.3. Notification.....	33

1. Persiapan

Persiapan tools yang akan digunakan yaitu vscode, android studio. Menggunakan flutter

Link Github : <https://github.com/DeoFaradyS/Pemrograman-Perangkat-Bergerak>

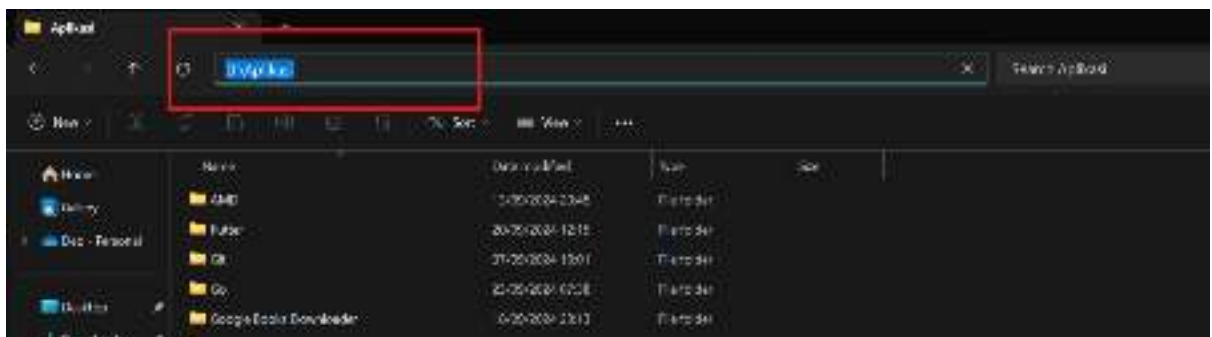
2. Instalasi dan Konfigurasi Flutter dan Dart

2.1. Install Flutter

Kunjungi web [flutter](https://flutter.dev), kemudian download flutter sesuai dengan spesifikasi kalian.



Setelah didownload, ekstrak file tersebut dan pindahkan ke folder lain. Kemudian simpan alamat dari folder yang berisi flutter.



2.2. Perbarui Path

Buka **Environment Variabel** dan perbarui **Path – User Variabel**.



2.3. Install Android Studio

Install [Android Studio](#) disitus resminya, kemudian buka aplikasinya.



Klik **Next** dan pilih komponen **Android Virtual Device**.



Kemudian atur **lokasi file** aplikasinya dan klik **Install**.



2.4. Setup Android Studio

Selanjutnya, klik **Next** dan pilih **Standard**. Terakhir klik tombol **Finish** setelah selesai.



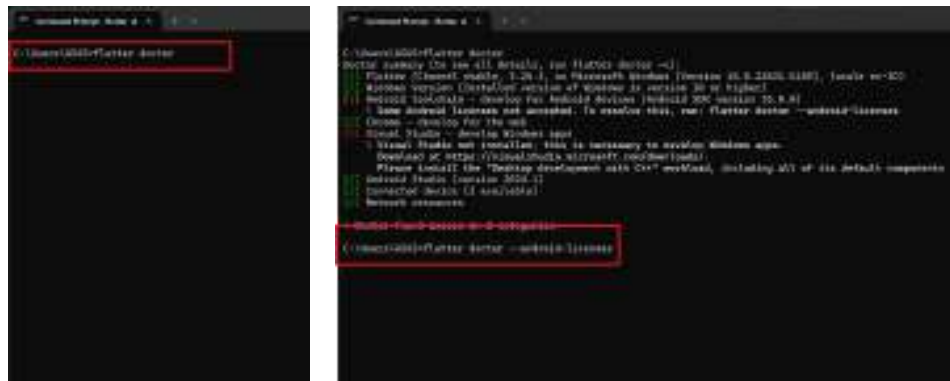
Klik tombol **More Action** dan pilih **SDK Manager**. Pastikan **Android SDK Command Line Tools (Latest)** sudah dicentang.

Tidak lupa untuk menginstall **flutter** di menu **plugin**.



2.5. Install License

Buka **CMD** di windows, dan ketikkan perintah **flutter doctor**. Kemudian ketikkan perintah '**flutter doctor -android-licenses**'.



3. Basic Dart

3.1. Soal 1

Buatlah fungsi Dart yang membuat suatu matrix $A \times B$ dengan A dan B sebagai parameter. Isi tiap nilai matriks (bebas atau di-random), lalu outputkan matriks tersebut dan matriks transpose-nya.

Contoh output:

Matriks AxB

A: 3

B: 2

Isi matrix:

12

34

56

Hasil transpose:

1 3 5

246

```
1 import 'dart:math';
2
3 void main() {
4   int a = 3, b = 2;
5
6   // Membuat dan menampilkan matriks
7   List<List<int>> matrix = createMatrix(a, b);
8
9   // Menampilkan matriks
10  print('Matriks $a x $b:');
11  printMatrix(matrix);
12
13  // Menampilkan hasil transpose
14  print('Hasil transpose:');
15  printMatrix(transpose(matrix));
16 }
17
18 // Fungsi untuk membuat matriks
19 List<List<int>> createMatrix(int a, int b) =>
20   List.generate(a, (_) => List.generate(b, (_) => Random().nextInt(10)));
21
22 void printMatrix(List<List<int>> matrix) {
23   matrix.forEach((row) => print(row.join(' ')));
24 }
25
26 List<List<int>> transpose(List<List<int>> matrix) => List.generate(
27   matrix[0].length, (i) => List.generate(matrix.length, (j) => matrix[j][i]));
28
```

3.1.1. Struktur Kode

```
1 import 'dart:math';
```

Mengimpor library **dart:math** untuk menggunakan fungsi acak.


```
homepage.dart

1 void main() {
2   int a = 3, b = 2;
3
4   // Membuat dan menampilkan matriks
5   List<List<int>> matrix = createMatrix(a, b);
6
7   // Menampilkan matriks
8   print('Matriks $a x $b:');
9   printMatrix(matrix);
10
11  // Menampilkan hasil transpose
12  print('Hasil transpose:');
13  printMatrix(transpose(matrix));
14 }
```

- Fungsi **main** adalah titik masuk program.
- Mendefinisikan ukuran matriks **a** (jumlah baris) dan **b** (jumlah kolom).
- Memanggil fungsi **createMatrix** untuk membuat matriks dengan ukuran yang ditentukan.
- Memanggil **printMatrix** untuk menampilkan matriks.
- Memanggil **transpose** untuk mendapatkan hasil transpose dari matriks dan menampilkannya.

```
homepage.dart

1 List<List<int>> createMatrix(int a, int b) =>
2   List.generate(a, (_) => List.generate(b, (_) => Random().nextInt(10)));
```

- Fungsi ini membuat matriks berukuran **a x b**.
- Mengisi setiap elemen matriks dengan angka acak dari 0 hingga 9 menggunakan **Random().nextInt(10)**.

```
homepage.dart

1 List<List<int>> transpose(List<List<int>> matrix) => List.generate(
2   matrix[0].length, (i) => List.generate(matrix.length, (j) => matrix[j][i]));
```

- Fungsi ini menghasilkan matriks transpose dari matriks yang diberikan.

- Ukuran matriks hasil transpose adalah **$b \times a$** , di mana **b** adalah jumlah kolom dan **a** adalah jumlah baris dari matriks asli.
- Menggunakan **List.generate** untuk membuat baris baru berdasarkan kolom dari matriks asli.

3.2. Soal 2

Buatlah fungsi Dart yang menerima satu nilai integer sebagai parameter dan dapat mencari nilai tersebut dalam suatu List 2 dimensi bertipe integer berukuran 4, yang isi masing-masing List-nya dengan perulangan:

- baris 1 berisi 3 bilangan kelipatan 5 berurutan mulai dari 5
- baris 2 berisi 4 bilangan genap berurutan mulai dari 2
- baris 3 berisi 5 bilangan kuadrat dari bilangan asli mulai dari 1
- baris 4 berisi 6 bilangan asli berurutan mulai dari 3

Contoh output:

Isi List:

5 10 15

2 4 6 8

1 4 9 16 25

3 4 5 6 7 8

Bilangan yang dicari: 2

2 berada di:

baris 2 kolom 1

Isi List:

5 10 15

2 4 6 8

1 4 9 16 25

3 4 5 6 7 8

Bilangan yang dicari: 5

5 berada di:

baris 1 kolom 1

baris 4 kolom 3

```

1  import 'dart:io';
2
3  void main() {
4      // Membuat list 2 dimensi
5      List<List<int>> list2D = createList2D();
6
7      // Menampilkan isi list
8      displayList(list2D);
9
10     // Menanti input dari pengguna untuk mencari bilangan
11     int target = getUserInput();
12
13     // Mencari nilai target dalam list 2 dimensi dan menampilkan hasil
14     List<String> positions = findInList(list2D, target);
15     displayResults(positions, target);
16 }
17
18 // Fungsi untuk membuat list 2 dimensi
19 List<List<int>> createList2D() {
20     return [
21         // Baris 1: 3 bilangan kelipatan 5, mulai dari 5
22         [for (int i = 1; i <= 3; i++) i * 5],
23
24         // Baris 2: 4 bilangan ganjil, mulai dari 1
25         [for (int i = 1; i <= 4; i++) i * 2],
26
27         // Baris 3: 5 bilangan kuadrat dari bilangan asli, mulai dari 1
28         [for (int i = 1; i <= 5; i++) i * i],
29
30         // Baris 4: 6 bilangan asli berurutan, mulai dari 3
31         [for (int i = 3; i <= 3 + 6; i++) i]
32     ];
33 }
34
35 // Fungsi untuk menampilkan isi list
36 void displayList(List<List<int>> list2D) {
37     print("Isi List:");
38     for (List<int> row in list2D) {
39         print(row.join(" "));
40     }
41 }
42
43 // Fungsi untuk meminta input dari pengguna
44 int getUserInput() {
45     stdout.write("\nBilangan yang dicari: ");
46     return int.parse(stdin.readLineSync());
47 }
48
49 // Fungsi untuk mencari nilai dalam list 2 dimensi
50 List<String> findInList(List<List<int>> list2D, int target) {
51     List<String> result = [];
52
53     for (int i = 0; i < list2D.length; i++) {
54         for (int j = 0; j < list2D[i].length; j++) {
55             if (list2D[i][j] == target) {
56                 // Menyimpan posisi baris dan kolom (baris dan kolom dimulai dari 1)
57                 result.add("baris ${i + 1} kolom ${j + 1}");
58             }
59         }
60     }
61
62     return result;
63 }
64
65 // Fungsi untuk menampilkan hasil pencarian
66 void displayResults(List<String> positions, int target) {
67     if (positions.isEmpty) {
68         print("\nBilangan target tidak ditemukan dalam list.");
69     } else {
70         print("\ntarget berada di:");
71         for (String pos in positions) {
72             print(pos);
73         }
74     }
75 }

```

3.2.1. Struktur Kode


```
1 void main() {
2     // Membuat list 2 dimensi
3     List<List<int>> list2D = createList2D();
4
5     // Menampilkan isi list
6     displayList(list2D);
7
8     // Meninta input dari pengguna untuk mencari bilangan
9     int target = getUser Input();
10
11    // Mencari nilai target dalam list 2 dimensi dan menampilkan hasil
12    List<String> positions = findInList(list2D, target);
13    displayResults(positions, target);
14 }
```

- **Fungsi utama** yang menjalankan program.
- Membuat list 2 dimensi menggunakan **createList2D()**.
- Menampilkan isi list dengan **displayList()**.
- Meminta pengguna untuk memasukkan bilangan yang ingin dicari dengan **getUser Input()**.
- Mencari bilangan tersebut dalam list menggunakan **findInList()** dan menampilkan hasilnya dengan **displayResults()**.

```
1 List<List<int>> createList2D() {
2     return [
3         [for (int i = 1; i <= 3; i++) i * 5], // Kelipatan 5
4         [for (int i = 1; i <= 4; i++) i * 2], // Bilangan genap
5         [for (int i = 1; i <= 5; i++) i * i], // Bilangan kuadrat
6         [for (int i = 3; i < 3 + 6; i++) i] // Bilangan asli mulai dari 3
7     ];
8 }
```

- Membuat dan mengembalikan list 2 dimensi yang berisi:
 - Baris 1: 3 bilangan kelipatan 5 (5, 10, 15)
 - Baris 2: 4 bilangan genap (2, 4, 6, 8)

- Baris 3: 5 bilangan kuadrat (1, 4, 9, 16, 25)
- Baris 4: 6 bilangan asli berurutan mulai dari 3 (3, 4, 5, 6, 7, 8)



```

1 void displayList(List<List<int>> list2D) {
2   print("Isi List:");
3   for (List<int> row in list2D) {
4     print(row.join(" "));
5   }
6 }

```

- Menampilkan isi dari list 2 dimensi ke konsol.
- Setiap baris ditampilkan dalam format yang rapi.



```

1 int getUser Input() {
2   stdout.write("\nBilangan yang dicari: ");
3   return int.parse(stdin.readLineSync()!);
4 }

```

- Meminta pengguna untuk memasukkan bilangan yang ingin dicari.
- Mengembalikan bilangan yang dimasukkan sebagai integer.

```
homepage.dart

1 List<String> findInList(List<List<int>> list2D, int target) {
2   List<String> result = [];
3
4   for (int i = 0; i < list2D.length; i++) {
5     for (int j = 0; j < list2D[i].length; j++) {
6       if (list2D[i][j] == target) {
7         result.add("baris ${i + 1} kolom ${j + 1}");
8       }
9     }
10  }
11
12  return result;
13 }
```

- Mencari bilangan target dalam list 2 dimensi.
- Mengembalikan daftar posisi (baris dan kolom) di mana bilangan tersebut ditemukan.

```
homepage.dart

1 void displayResults(List<String> positions, int target) {
2   if (positions.isEmpty) {
3     print("\nBilangan $target tidak ditemukan dalam list.");
4   } else {
5     print("\n$target berada di:");
6     for (String pos in positions) {
7       print(pos);
8     }
9   }
10 }
```

- Menampilkan hasil pencarian.
- Jika bilangan tidak ditemukan, menampilkan pesan yang sesuai.
- Jika ditemukan, menampilkan semua posisi di mana bilangan tersebut berada.

3.3. Soal 3

Buatlah fungsi Dart yang menerima dua nilai integer positif dan mengoutputkan nilai KPK (Kelipatan Persekutuan terKecil) dari dua bilangan tersebut

Contoh output:

Bilangan 1: 12

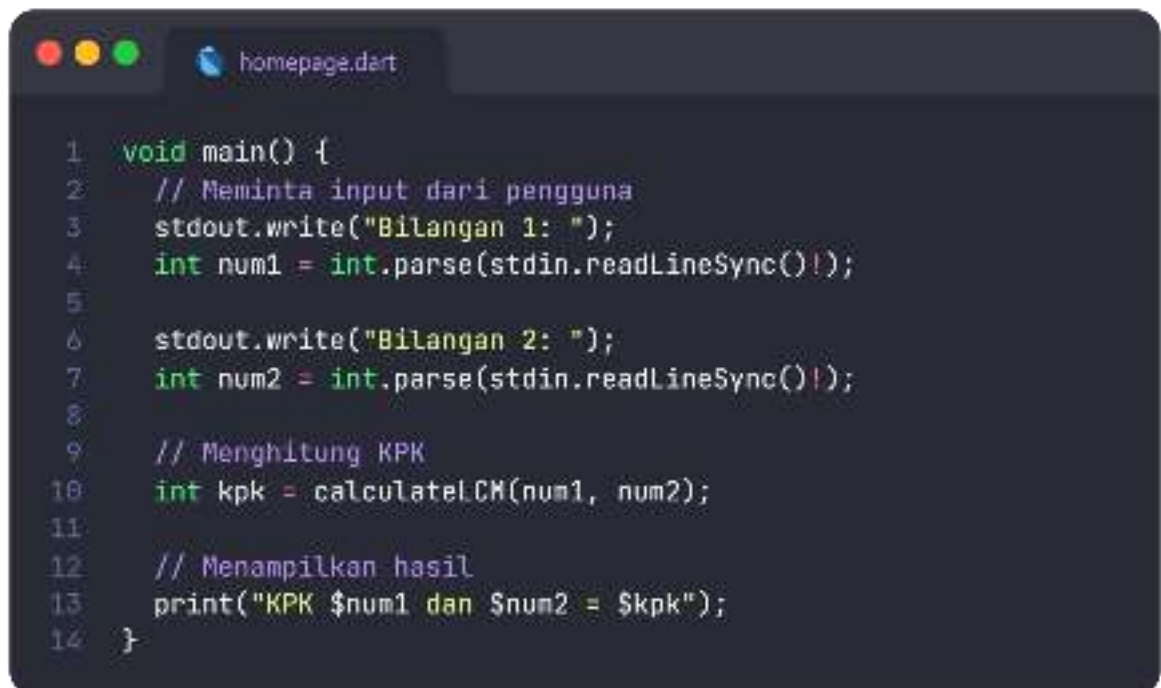
Bilangan 2: 8

KPK 12 dan 8 = 24



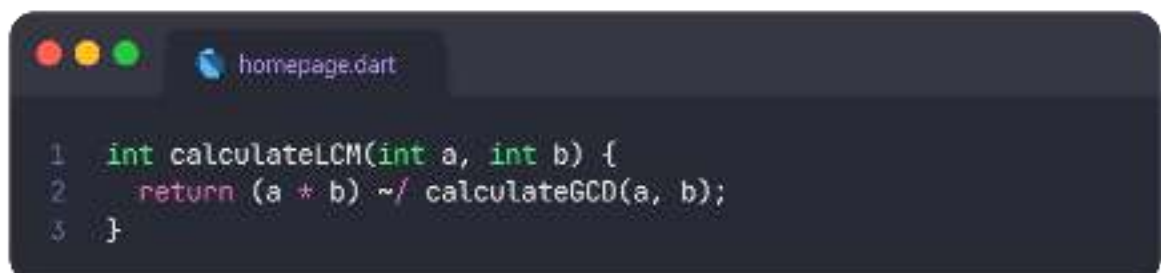
```
1  import 'dart:io';
2
3  void main() {
4    // Meminta input dari pengguna
5    stdout.write("Bilangan 1: ");
6    int num1 = int.parse(stdin.readLineSync()!);
7
8    stdout.write("Bilangan 2: ");
9    int num2 = int.parse(stdin.readLineSync()!);
10
11    // Menghitung KPK
12    int kpk = calculateLCM(num1, num2);
13
14    // Menampilkan hasil
15    print("KPK $num1 dan $num2 = $kpk");
16  }
17
18  // Fungsi untuk menghitung KPK
19  int calculateLCM(int a, int b) {
20    return (a * b) ~/ calculateGCD(a, b);
21  }
22
23  // Fungsi untuk menghitung FPB menggunakan algoritma Euclidean
24  int calculateGCD(int a, int b) {
25    while (b != 0) {
26      int temp = b;
27      b = a % b;
28      a = temp;
29    }
30    return a;
31  }
32
```

3.3.1. Struktur Kode



```
1 void main() {
2     // Meminta input dari pengguna
3     stdout.write("Bilangan 1: ");
4     int num1 = int.parse(stdin.readLineSync()!);
5
6     stdout.write("Bilangan 2: ");
7     int num2 = int.parse(stdin.readLineSync()!);
8
9     // Menghitung KPK
10    int kpk = calculateLCM(num1, num2);
11
12    // Menampilkan hasil
13    print("KPK $num1 dan $num2 = $kpk");
14 }
```

- **Fungsi utama** yang menjalankan program.
- Meminta pengguna untuk memasukkan dua bilangan (**num1** dan **num2**).
- Menghitung KPK dari kedua bilangan menggunakan fungsi **calculateLCM()**.
- Menampilkan hasil KPK ke layar.



```
1 int calculateLCM(int a, int b) {
2     return (a * b) ~/ calculateGCD(a, b);
3 }
```

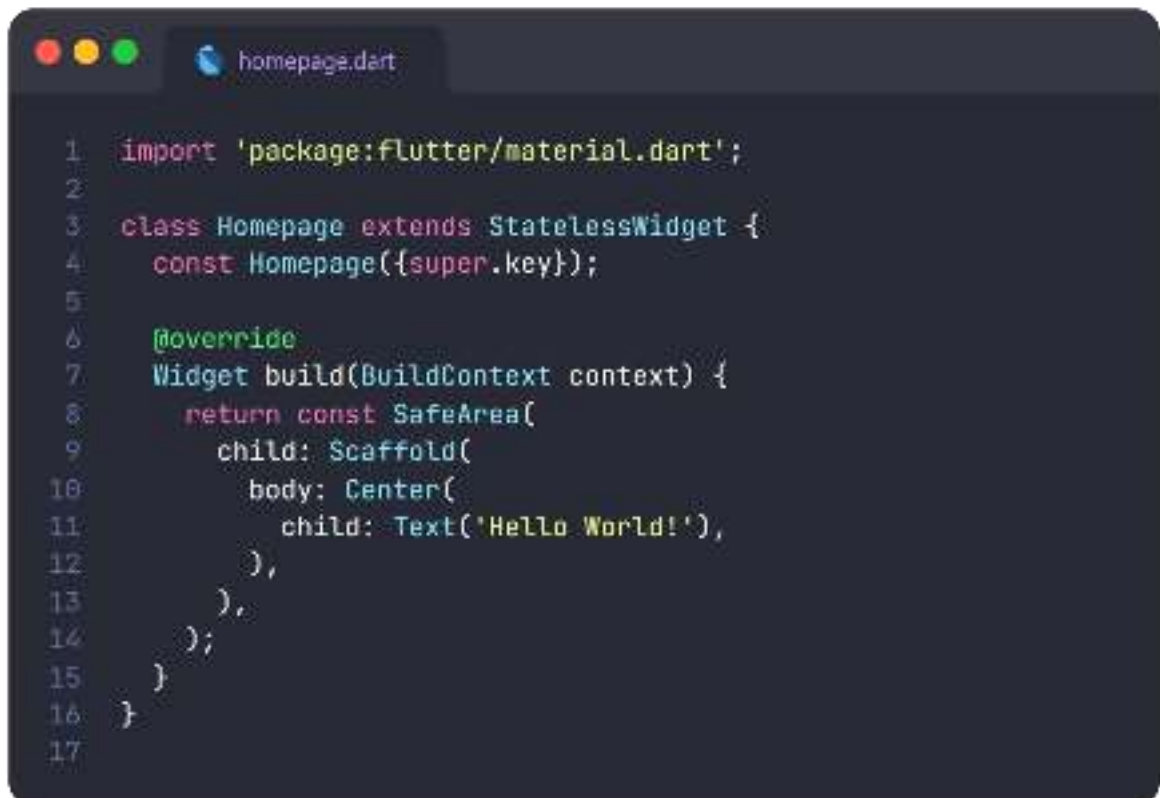
- Fungsi ini menghitung KPK dari dua bilangan **a** dan **b**.
- KPK dihitung dengan rumus: $KPK(a, b) = (a * b) / FPB(a, b)$, di mana FPB adalah Faktor Persekutuan Terbesar.
- Menggunakan operator **~/** untuk pembagian bulat.

A screenshot of a code editor window with a dark theme. The window title is 'homepage.dart'. It contains a Dart function named 'calculateGCD' that takes two integers 'a' and 'b' as parameters. The function uses a 'while' loop to repeatedly calculate the remainder of 'a' divided by 'b' and update 'a' and 'b' until 'b' becomes zero. Finally, it returns the value of 'a', which is the Greatest Common Divisor (FPB).

- Fungsi ini menghitung FPB dari dua bilangan **a** dan **b** menggunakan algoritma Euclidean.
- Menggunakan loop untuk terus memperbarui nilai **a** dan **b** hingga **b** menjadi 0.
- Mengembalikan nilai **a**, yang merupakan FPB dari kedua bilangan.

4. Basic Layout

4.1. Teks



```
1 import 'package:flutter/material.dart';
2
3 class Homepage extends StatelessWidget {
4   const Homepage({super.key});
5
6   @override
7   Widget build(BuildContext context) {
8     return const SafeArea(
9       child: Scaffold(
10         body: Center(
11           child: Text('Hello World!'),
12         ),
13       ),
14     );
15   }
16 }
17
```

Widget **Text** digunakan untuk menampilkan teks di layar. Dalam kode ini, widget **Text** digunakan untuk menampilkan string "Hello World!".

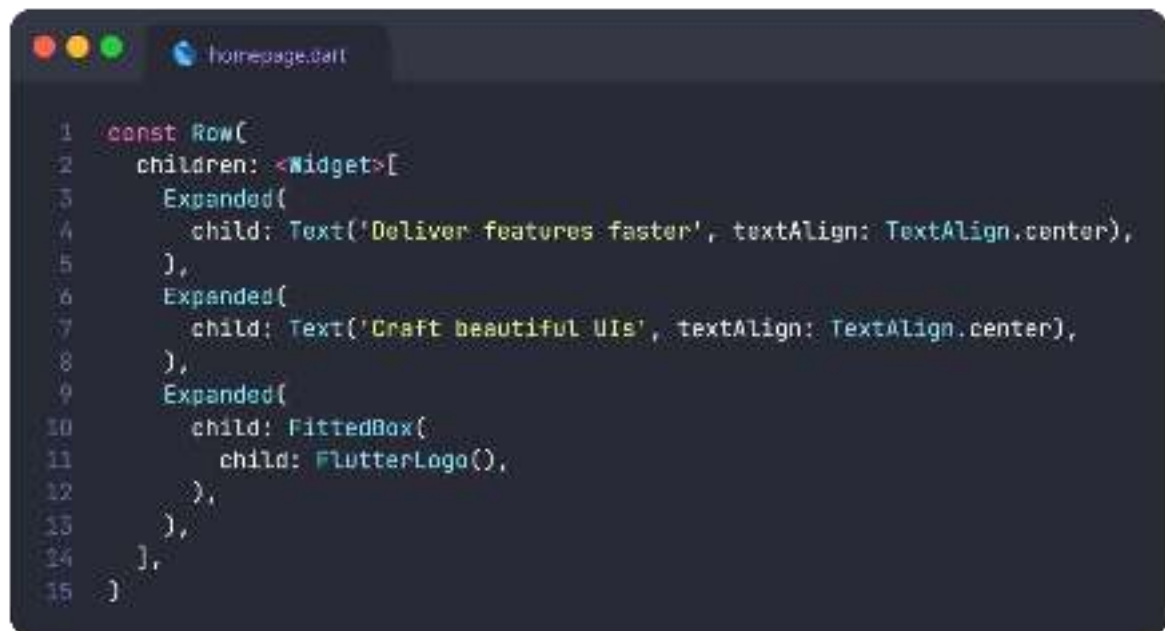
4.2. Container



```
1 Center(
2   child: Container(
3     margin: const EdgeInsets.all(10.0),
4     color: Colors.amber[600],
5     width: 48.0,
6     height: 48.0,
7   ),
8 )
```

Container adalah widget dasar dalam Flutter yang digunakan untuk mengatur dan menampilkan elemen UI. Ini adalah widget yang sangat fleksibel dan dapat digunakan untuk membungkus widget lain.

4.3. Row

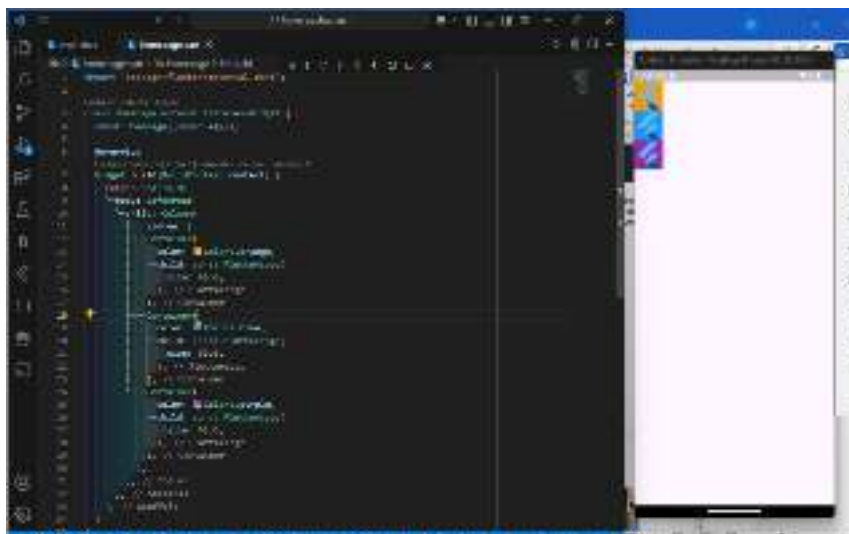


Row adalah widget dalam Flutter yang digunakan untuk menampilkan beberapa widget secara horizontal (dari kiri ke kanan) dalam satu baris.

4.4. Column

```
1 Column(  
2     children: [  
3         Container(  
4             color: Colors.orange,  
5             child: const FlutterLogo(  
6                 size: 60.0,  
7             ),  
8         ),  
9         Container(  
10            color: Colors.blue,  
11            child: const FlutterLogo(  
12                size: 60.0,  
13            ),  
14        ),  
15        Container(  
16            color: Colors.purple,  
17            child: const FlutterLogo(  
18                size: 60.0,  
19            ),  
20        ),  
21    ],  
22 ),
```

Widget **Column** dalam Flutter digunakan untuk menampilkan widget-widget secara vertikal, satu di atas yang lain. Dalam contoh yang diberikan, **Column** berisi tiga **Container**, masing-masing dengan warna yang berbeda dan menampilkan logo Flutter.



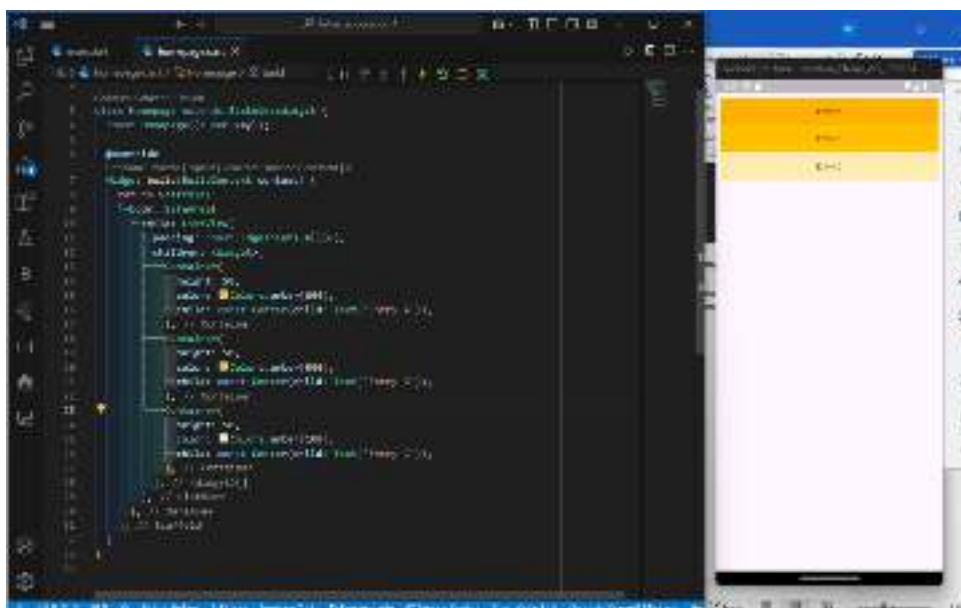
4.5. List View

```
homepage.dart

1 import 'package:flutter/material.dart';
2
3 class Homepage extends StatelessWidget {
4   const Homepage({super.key});
5
6   @override
7   Widget build(BuildContext context) {
8     return const SafeArea(
9       child: Scaffold(
10         body: Center(
11           child: Text('Hello World!'),
12         ),
13       ),
14     );
15   }
16 }
17
```

ListView adalah salah satu widget dalam Flutter yang digunakan untuk menampilkan daftar atau koleksi item secara vertikal.

Widget ini sangat berguna ketika Anda memiliki banyak data yang ingin ditampilkan, karena **ListView** dapat menggulir (scroll) secara otomatis jika item-itemnya melebihi ukuran layer.



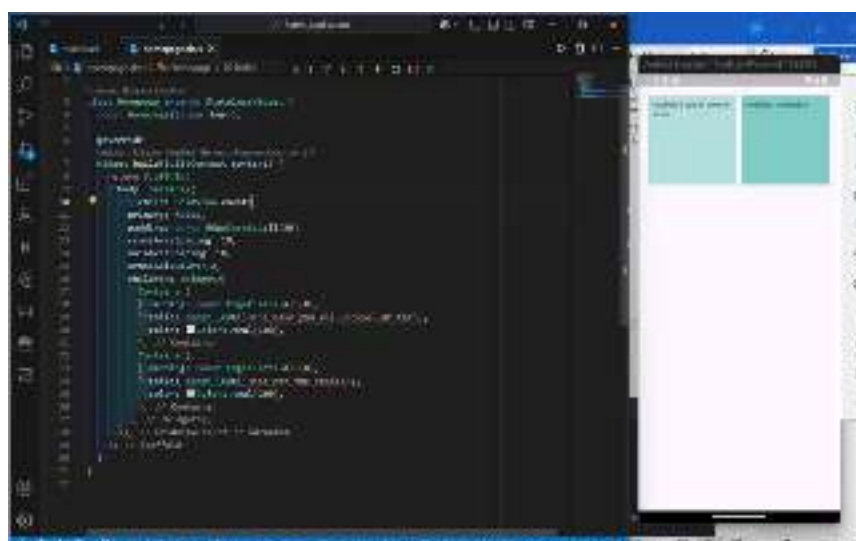
4.6. Grid View

```
homepage.dart

1  GridView.count(
2    primary: false,
3    padding: const EdgeInsets.all(20),
4    crossAxisSpacing: 10,
5    mainAxisSpacing: 10,
6    crossAxisCount: 2,
7    children: <Widget>[
8      Container(
9        padding: const EdgeInsets.all(8),
10       child: const Text("He'd have you all unravel at the"),
11       color: Colors.teal[100],
12     ),
13     Container(
14       padding: const EdgeInsets.all(8),
15       child: const Text('Heed not the rabble'),
16       color: Colors.teal[200],
17     ),
18     ...
19   ],
20 )
21
```

GridView adalah widget dalam Flutter yang digunakan untuk menampilkan item dalam bentuk grid atau kisi-kisi.

Ini mirip dengan **ListView**, tetapi alih-alih menampilkan item secara vertikal, GridView menampilkan item dalam beberapa kolom dan baris.



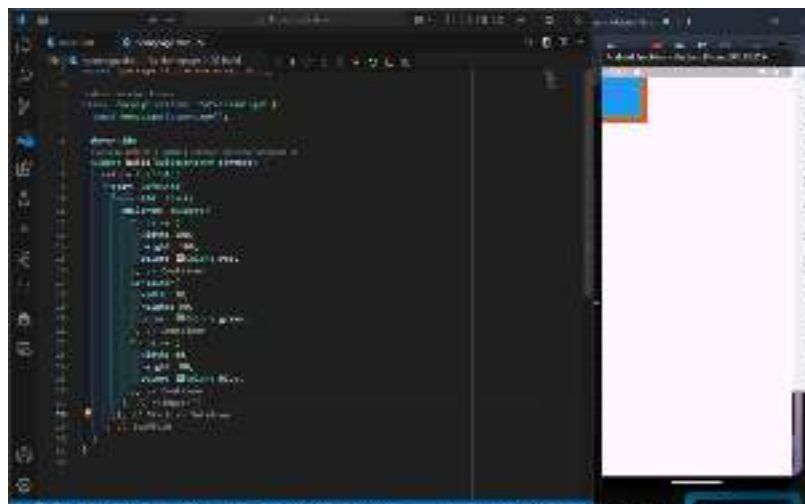
4.7. Stack

```
1 Stack(  
2   children: <Widget>[  
3     Container(  
4       width: 100,  
5       height: 100,  
6       color: Colors.red,  
7     ),  
8     Container(  
9       width: 90,  
10      height: 90,  
11      color: Colors.green,  
12    ),  
13    Container(  
14      width: 80,  
15      height: 80,  
16      color: Colors.blue,  
17    ),  
18  ],  
19 )  
20
```

Stack adalah widget yang memungkinkan kita untuk menumpuk beberapa widget di atas satu sama lain.

Dengan menggunakan **Stack**, kita bisa menempatkan widget di posisi yang berbeda-beda dalam satu area yang sama.

Ini sangat berguna ketika kita ingin membuat tampilan yang kompleks, seperti overlay, gambar dengan teks di atasnya, atau elemen yang saling tumpang tindih.



5. Advance Layout

5.1. Nested Row/Column

```
1 Column(  
2     children: [  
3         Expanded(  
4             // added Expanded widget  
5             child: Column(  
6                 children: [  
7                     Expanded(  
8                         child: Container(color: Colors.blue),  
9                     ),  
10                ],  
11            ),  
12        ],  
13    ],  
14 )
```

Nested Row/Column adalah penggunaan widget **Row** atau **Column** di dalam Row atau Column lainnya.

Ini memungkinkan kita untuk membuat layout yang lebih kompleks dengan mengatur elemen-elemen secara horizontal (Row) atau vertikal (Column) dalam struktur yang berlapis.

5.2. Tab View

5.2.1. Tab Bar

```
1 DefaultTabController(  
2     length: 3,  
3     child: Scaffold(  
4         appBar: AppBar(  
5             bottom: const TabBar(  
6                 tabs: [  
7                     Tab(icon: Icon(Icons.home_filled), text: "Home",),  
8                     Tab(icon: Icon(Icons.account_box_outlined), text: "Account",),  
9                     Tab(icon: Icon(Icons.alarm), text: "Alarm",),  
10                ],  
11            ),  
12     ),  
13 )
```


Tab Bar adalah widget yang menampilkan sejumlah tab (tombol) di bagian atas layar.

Setiap tab mewakili satu tampilan atau konten yang berbeda. Pengguna dapat mengetuk tab untuk beralih ke tampilan yang sesuai.

5.2.2. Tab Bar View

Tab Bar View adalah widget yang digunakan untuk menampilkan konten yang sesuai dengan tab yang dipilih.

Setiap tab akan memiliki tampilan atau widget yang berbeda, dan Tab Bar View akan menampilkan tampilan tersebut ketika tab yang relevan dipilih.

5.3. Page View

PageView di Flutter adalah widget yang memungkinkan kita untuk menampilkan beberapa halaman (page) secara horizontal atau vertikal, dan pengguna dapat menggulir (swipe) antara halaman-halaman tersebut.

Ini sangat berguna ketika kita ingin membuat tampilan yang mirip dengan slideshow atau galeri gambar, di mana pengguna bisa berpindah antar halaman dengan mudah.

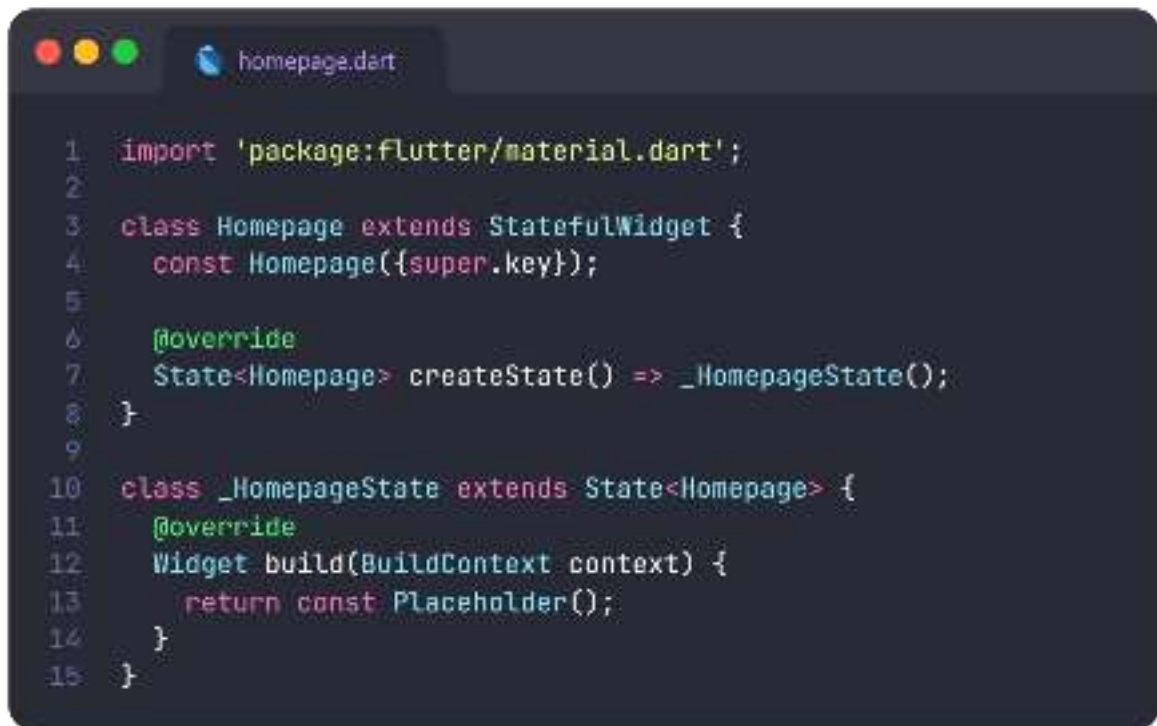
5.4. Safe Area

Safe area adalah widget yang digunakan untuk memastikan bahwa konten aplikasi tidak terhalang oleh elemen-elemen seperti status bar, notch (potongan di layar), atau area lain yang mungkin menghalangi tampilan.

Dengan menggunakan widget **SafeArea**, dapat memastikan bahwa semua elemen UI yang dibuat tetap terlihat dengan baik dan tidak terpotong oleh batasan layar perangkat.

6. User Interaction

6.1. Stateful & Stateless Widget



```
1 import 'package:flutter/material.dart';
2
3 class Homepage extends StatefulWidget {
4   const Homepage({super.key});
5
6   @override
7   State<Homepage> createState() => _HomepageState();
8 }
9
10 class _HomepageState extends State<Homepage> {
11   @override
12   Widget build(BuildContext context) {
13     return const Placeholder();
14   }
15 }
```

Stateful Widget adalah widget yang memiliki status yang dapat berubah.

Ini berarti bahwa isi dari widget ini bisa diperbarui atau diubah seiring waktu, misalnya ketika pengguna berinteraksi dengan aplikasi.



```
1 import 'package:flutter/material.dart';
2
3 class Homepage extends StatelessWidget {
4   const Homepage({super.key});
5
6   @override
7   Widget build(BuildContext context) {
8     return const Placeholder();
9   }
10 }
```

Stateless Widget adalah widget yang tidak memiliki status yang dapat berubah. Artinya, setelah widget dibuat, isinya tidak akan berubah seiring waktu.

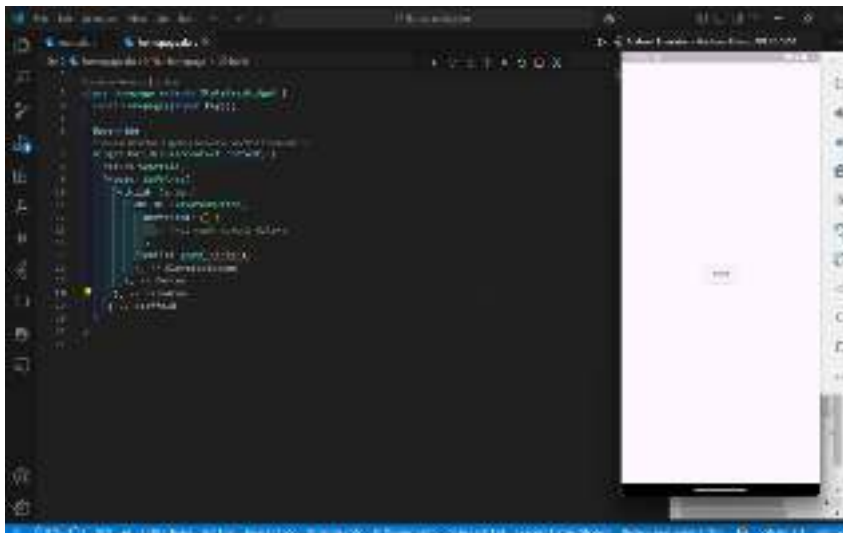
6.2. Button

6.2.1. Flat Button

FlatButton adalah tombol yang datar, tanpa bayangan atau elevasi. Biasanya digunakan untuk tindakan sekunder atau di dalam dialog.

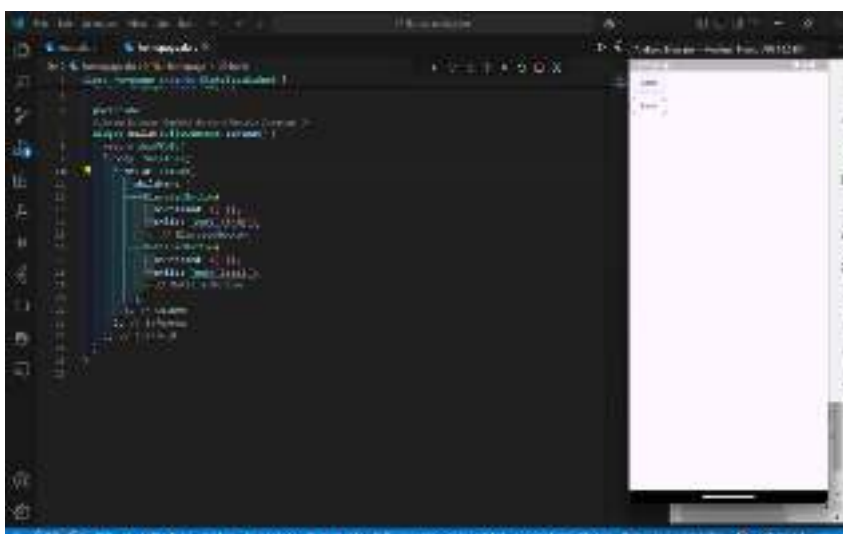
6.2.2. RaisedButton or ElevatedButton

ElevatedButton adalah tombol yang memiliki bayangan dan tampak terangkat dari latar belakang. Tombol ini digunakan untuk tindakan utama di dalam aplikasi. Misalnya, tombol "Simpan" atau "Kirim".



6.2.3. OutlineButton

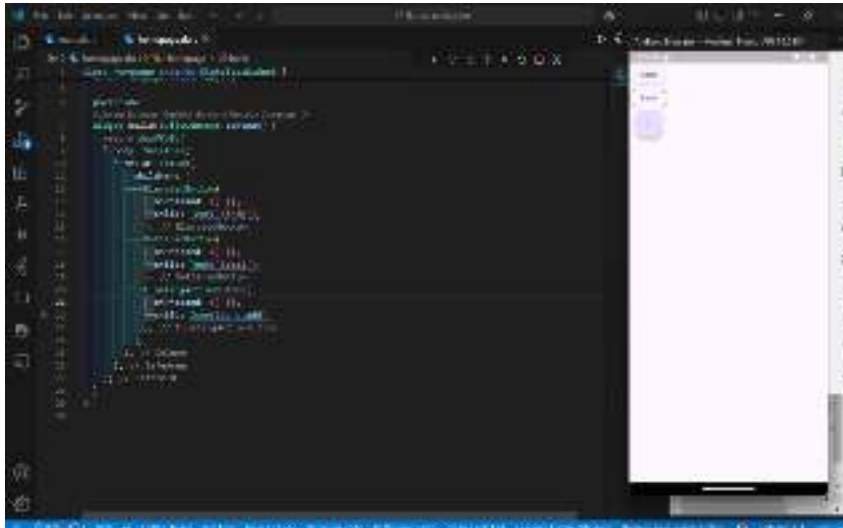
OutlineButton adalah tombol dengan garis tepi, tanpa latar belakang berwarna. Ini memberikan tampilan yang lebih ringan dan sering digunakan untuk tindakan sekunder.



6.2.4. FloatingActionButton

FloatingActionButton adalah tombol bulat yang biasanya digunakan untuk tindakan utama di layar.

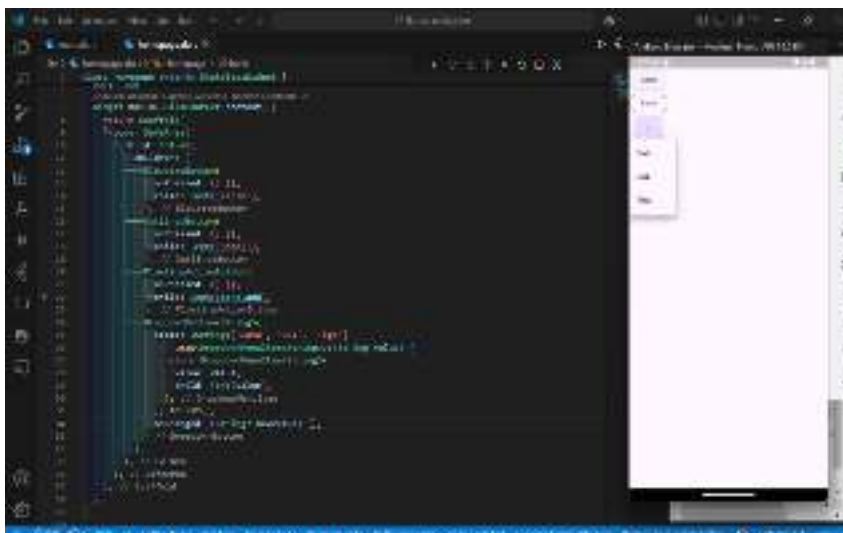
Biasanya, tombol ini mengapung di atas konten dan sering digunakan untuk menambahkan item baru, seperti "Tambah" atau "Buat".



6.2.5. DropdownButton

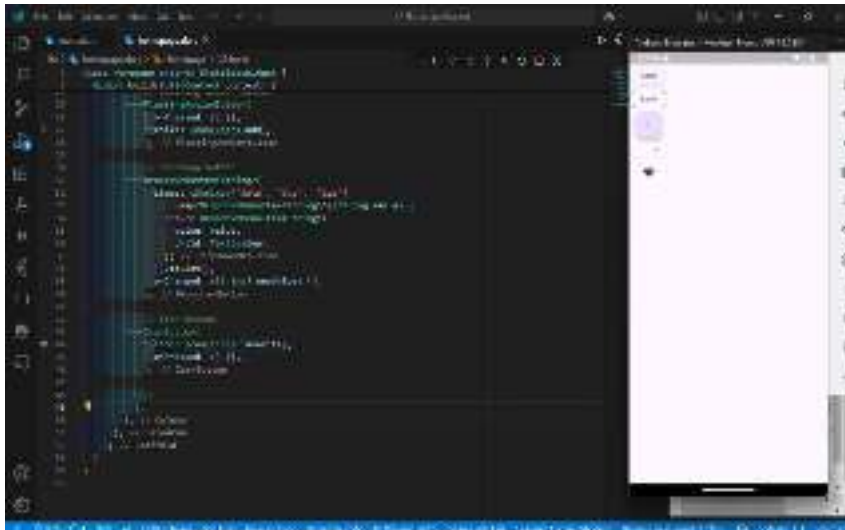
DropdownButton adalah tombol yang menampilkan daftar pilihan saat ditekan.

Pengguna dapat memilih salah satu opsi dari dropdown yang muncul.



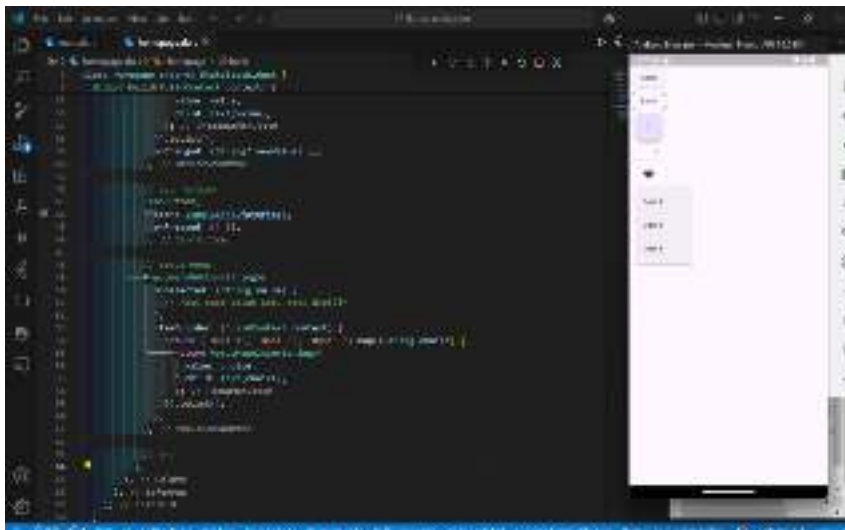
6.2.6. IconButton

IconButton adalah tombol yang hanya menampilkan ikon. Ini sering digunakan untuk tindakan cepat, seperti "Suka" atau "Hapus".



6.2.7. PopupMenuButton

PopupMenuButton adalah tombol yang menampilkan menu pop-up ketika ditekan.

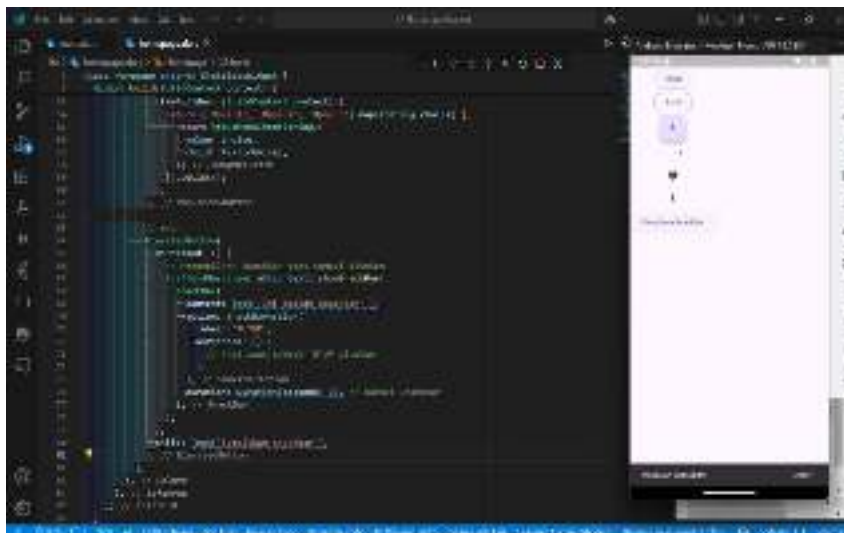


6.3. Snackbar

Snackbar adalah widget yang digunakan untuk menampilkan pesan singkat di bagian bawah layar.

Snackbar biasanya digunakan untuk memberikan umpan balik kepada pengguna setelah mereka melakukan suatu tindakan, seperti menyimpan data atau menghapus item.

Snackbar muncul sementara dan secara otomatis menghilang setelah beberapa detik.



6.4. Dialog

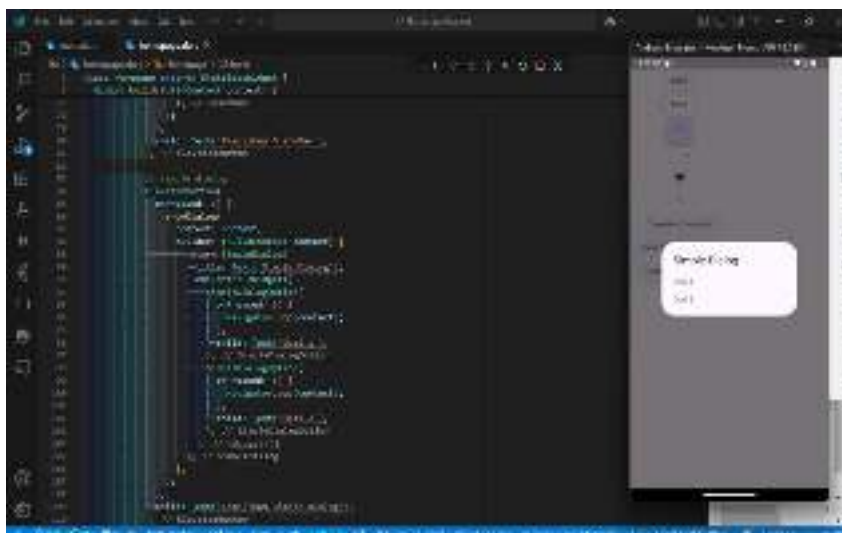
Dialog adalah komponen antarmuka pengguna yang digunakan untuk menampilkan informasi atau meminta konfirmasi dari pengguna.

Ada beberapa jenis dialog, tetapi dua yang paling umum digunakan adalah **Simple Dialog** dan **Alert Dialog**.

6.4.1. Simple Dialog

SimpleDialog adalah dialog yang digunakan untuk menampilkan pilihan-pilihan sederhana kepada pengguna.

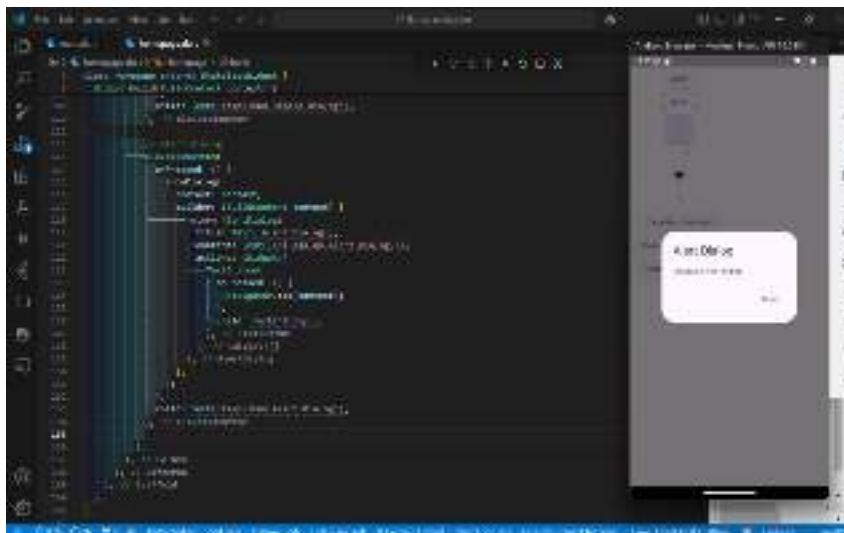
Biasanya, ini digunakan ketika ingin pengguna memilih dari beberapa opsi.



6.4.2. Alert Dialog

AlertDialog adalah dialog yang digunakan untuk menampilkan pesan penting kepada pengguna.

Ini sering digunakan untuk memberikan informasi, peringatan, atau meminta konfirmasi dari pengguna.

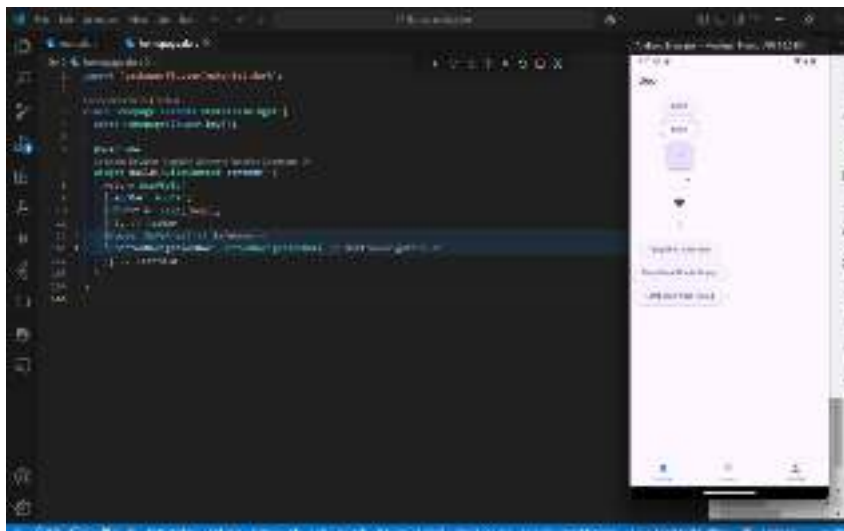


6.5. Menu

6.5.1. Bottom Navigation Bar

BottomNavigationBar adalah bagian di bagian bawah aplikasi yang berisi beberapa item navigasi.

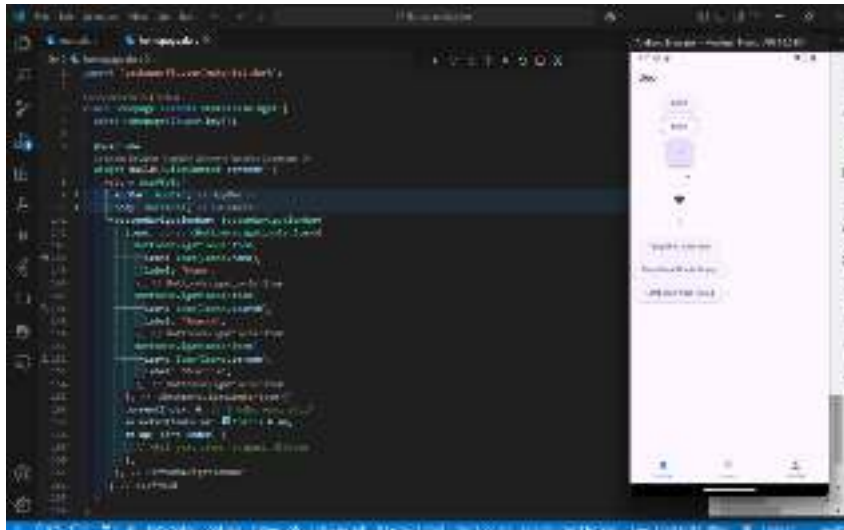
Ini memungkinkan pengguna untuk berpindah antara beberapa tampilan atau halaman dalam aplikasi.



6.5.2. App Bar

AppBar adalah bagian di bagian atas aplikasi yang biasanya berisi judul aplikasi, ikon, dan menu.

Ini memberikan konteks kepada pengguna tentang apa yang mereka lihat.



7. Navigation

7.1. Package

package adalah kumpulan kode yang bisa digunakan kembali yang menyediakan fungsionalitas tertentu.

Package ini memungkinkan kita untuk menambahkan fitur atau kemampuan ke aplikasi Flutter kita tanpa harus menulis semua kode dari awal.

7.2. Navigation

Navigation adalah cara untuk berpindah dari satu layar (screen) ke layar lainnya dalam aplikasi.

7.2.1. Navigate to New Screen and Go Back



```
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   @override
9   Widget build(BuildContext context) {
10     return MaterialApp(
11       home: FirstScreen(),
12     );
13   }
14 }
15
16 class FirstScreen extends StatelessWidget {
17   @override
18   Widget build(BuildContext context) {
19     return Scaffold(
20       appBar: AppBar(title: Text('First Screen')),
21       body: Center(
22         child: ElevatedButton(
23           onPressed: () {
24             navigator.push(
25               context,
26               MaterialPageRoute(builder: (context) => SecondScreen()),
27             );
28           },
29           child: Text('Go to Second Screen'),
30         ),
31       ),
32     );
33   }
34 }
35
36 class SecondScreen extends StatelessWidget {
37   @override
38   Widget build(BuildContext context) {
39     return Scaffold(
40       appBar: AppBar(title: Text('Second Screen')),
41       body: Center(
42         child: ElevatedButton(
43           onPressed: () {
44             Navigator.pop(context); // Kembali ke layar sebelumnya
45           },
46           child: Text('Go Back'),
47         ),
48       ),
49     );
50   }
51 }
```

Untuk berpindah ke layar baru, kita dapat menggunakan **Navigator.push()**. Untuk kembali ke layar sebelumnya, kita dapat menggunakan **Navigator.pop()**.

7.2.2. Navigate with named routes

```
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   @override
9   Widget build(BuildContext context) {
10     return MaterialApp(
11       initialRoute: '/',
12       routes: {
13         '/': (context) => FirstScreen(),
14         '/second': (context) => SecondScreen(),
15       },
16     );
17   }
18 }
19
20 class FirstScreen extends StatelessWidget {
21   @override
22   Widget build(BuildContext context) {
23     return Scaffold(
24       appBar: AppBar(title: Text('First Screen')),
25       body: Center(
26         child: ElevatedButton(
27           onPressed: () {
28             Navigator.pushNamed(context, '/second');
29           },
30           child: Text('Go to Second Screen'),
31         ),
32       ),
33     );
34   }
35 }
36
37 class SecondScreen extends StatelessWidget {
38   @override
39   Widget build(BuildContext context) {
40     return Scaffold(
41       appBar: AppBar(title: Text('Second Screen')),
42       body: Center(
43         child: ElevatedButton(
44           onPressed: () {
45             Navigator.pop(context);
46           },
47           child: Text('Go Back'),
48         ),
49       ),
50     );
51   }
52 }
```

Named routes adalah cara lain untuk melakukan navigasi dengan memberikan nama untuk setiap layar. Kita mendefinisikan rute di **MaterialApp**.

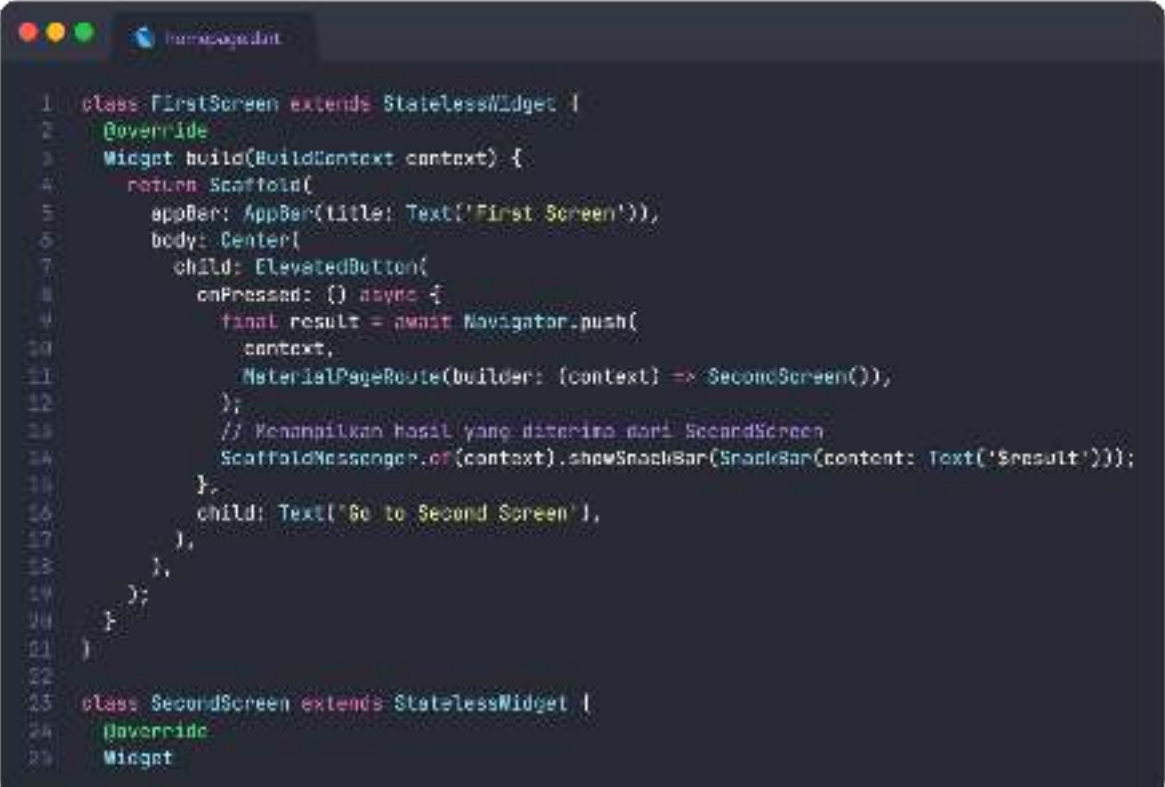
7.2.3. Pass arguments



```
1 class FirstScreen extends StatelessWidget {
2   @override
3   Widget build(BuildContext context) {
4     return Scaffold(
5       appBar: AppBar(title: Text('First Screen')),
6       body: Center(
7         child: ElevatedButton(
8           onPressed: () {
9             Navigator.push(
10              context,
11              MaterialPageRoute(
12                builder: (context) => SecondScreen(data: 'Hello from First Screen!'),
13              ),
14            );
15          },
16          child: Text('Go to Second Screen'),
17        ),
18      ),
19    );
20  }
21 }
22
23 class SecondScreen extends StatelessWidget {
24   final String data;
25
26   SecondScreen({required this.data});
27
28   @override
29   Widget build(BuildContext context) {
30     return Scaffold(
31       appBar: AppBar(title: Text('Second Screen')),
32       body: Center(
33         child: Text(data), // Menampilkan data yang diterima
34       ),
35     );
36  }
37 }
```

Kita dapat mengirim data saat berpindah ke layar baru dengan menggunakan constructor.

7.2.4. Return data and send data to new screen



```
1 class FirstScreen extends StatelessWidget {
2   @override
3   Widget build(BuildContext context) {
4     return Scaffold(
5       appBar: AppBar(title: Text('First Screen')),
6       body: Center(
7         child: ElevatedButton(
8           onPressed: () async {
9             final result = await Navigator.push(
10               context,
11               MaterialPageRoute(builder: (context) => SecondScreen()),
12             );
13             // Menampilkan hasil yang diterima dari SecondScreen
14             ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('$result')));
15           },
16           child: Text('Go to Second Screen'),
17         ),
18       ),
19     );
20   }
21 }
22
23 class SecondScreen extends StatelessWidget {
24   @override
25   Widget
```

Kita bisa mengembalikan data dari layar yang baru ke layar sebelumnya menggunakan **Navigator.pop()** dengan argumen.

7.3. Notification

Notifikasi lokal dijadwalkan dan dikirim langsung oleh aplikasi di perangkat pengguna.

Sebaliknya, **notifikasi push** dikirim dari server jarak jauh ke perangkat pengguna.