

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

DAFTAR ISI

1. Persiapan.....	1
2. Installasi 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
7.2.1. Navigate to New Screen and Go Back	30
7.2.2. Navigate with named routes	31
7.2.3. Pass arguments	32
7.2.4. Return data and send data to new screen	33
7.3. Notification.....	33
8. Media dan Kamera.....	34
8.1. Hasil	37
9. Data Storage	37
9.1. Local Storage	37
9.1.1. Shared Preferences.....	37
9.1.2. Flutter Secure Storage.....	38
9.1.3. Penyimpan File Lokal.....	39
9.1.4. SQLite.....	39
9.2. CRUD Sqflite.....	40
9.2.1. Create_todo_widget.....	40

9.2.2. Todo_crud	44
9.2.3. Hasil	48
10. API.....	48
10.1. todo_api	48
10.2. todo_api_model	51
10.3. todo_crud	53

1. Persiapan

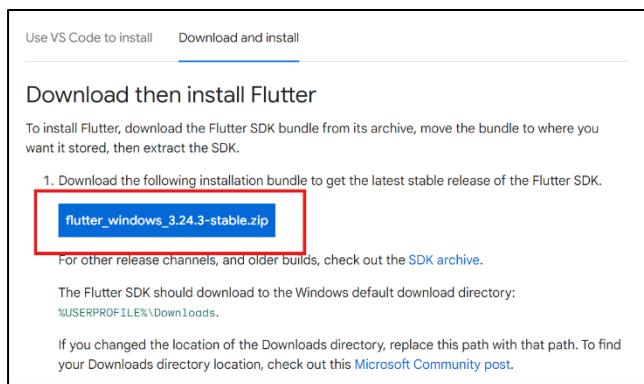
Persiapan tools yang akan digunakan yaitu vscode, android studio. Menggunakan flutter

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

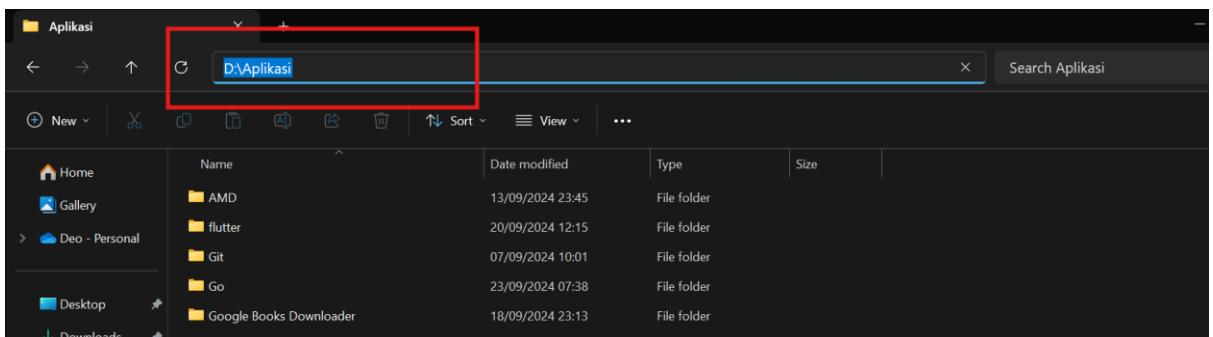
2. Installasi dan Konfigurasi Flutter dan Dart

2.1. Install Flutter

Kunjungi web [flutter](#), kemudian download flutter sesuai dengan spesifikasi kalian.

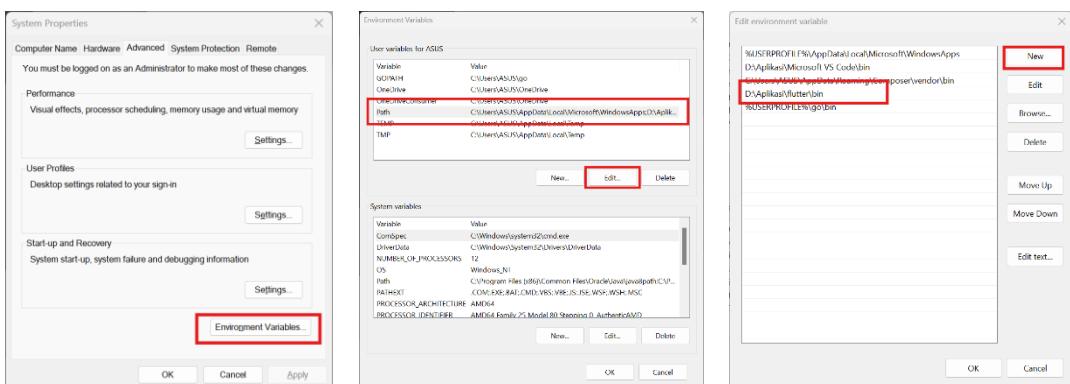


Setelah didownload, extrak file tersebut dan pindahkan ke folder lain. Kemudian simpan alamat dari folder yang berisi flutter.



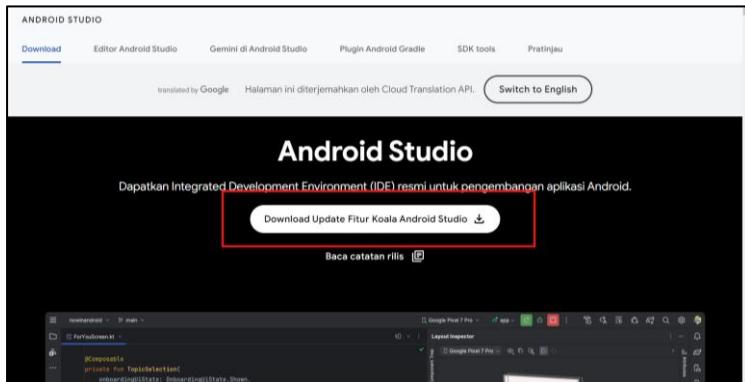
2.2. Perbarui Path

Buka Environment Variabel dan perbarui Path – User Variabel.

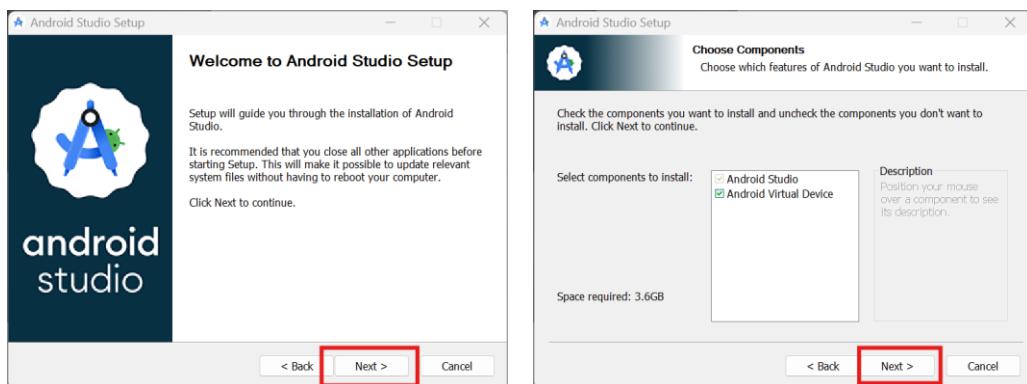


2.3. Install Android Studio

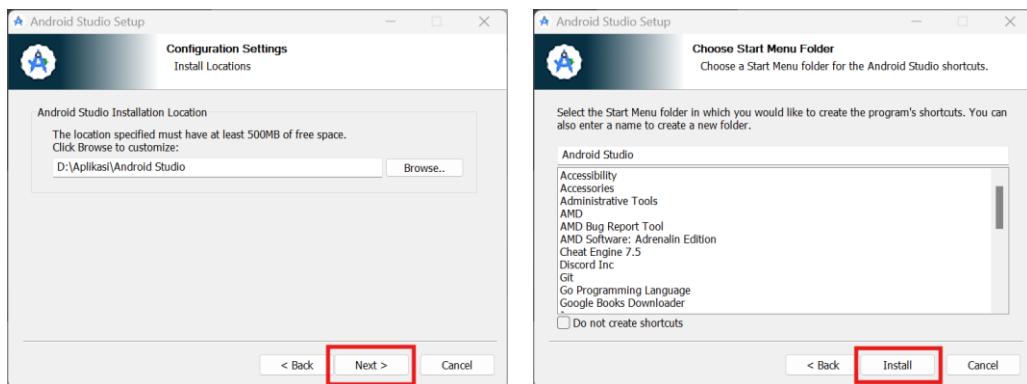
Install [Android Studio](#) disitus resminya, kemudian buka aplikasinya.



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

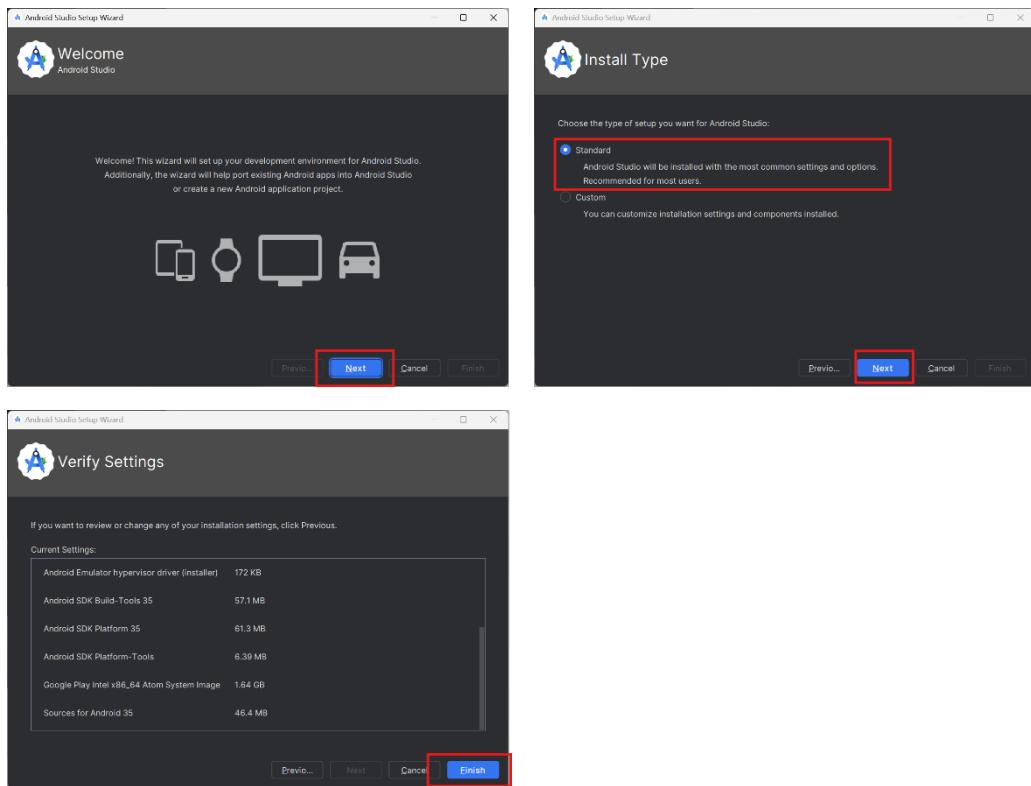


Kemudian atur **lokasi file** aplikasinya dan klik **Install**.



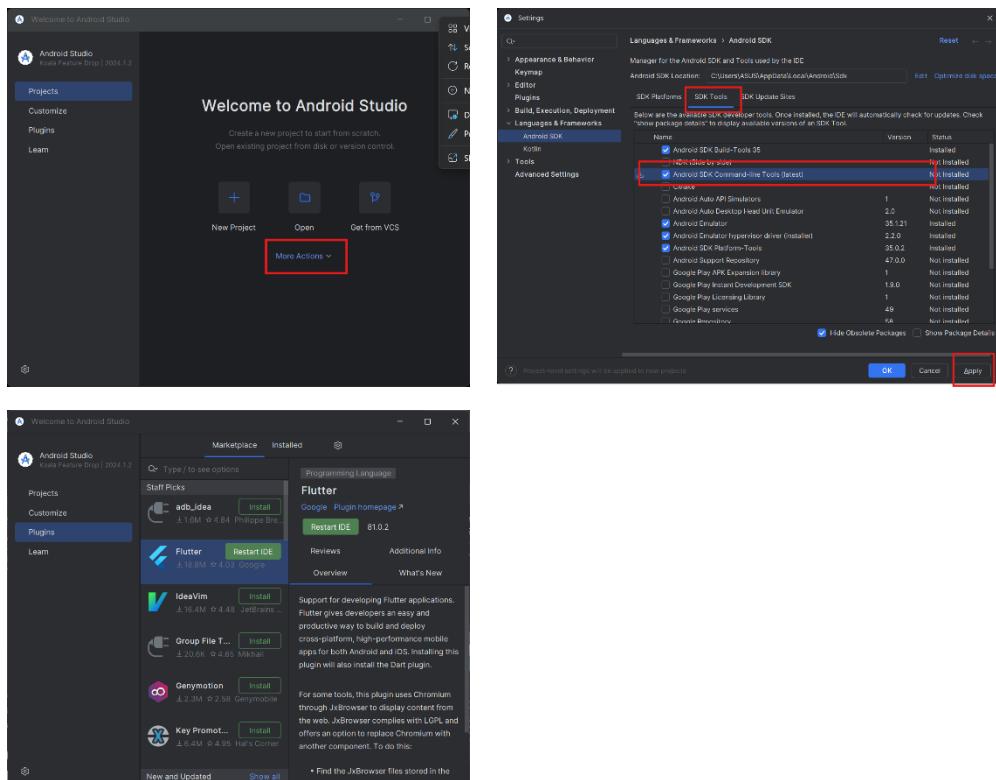
2.4. Setup Android Studio

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



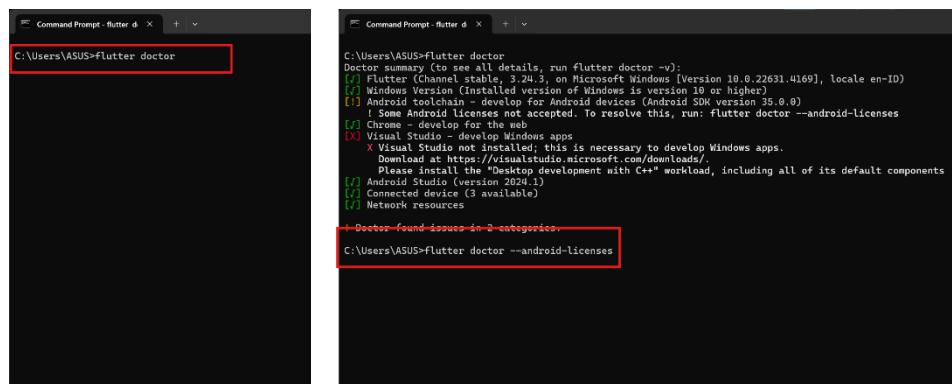
Klik tombol **More Action** dan pilih **SDK Manager**. Pastikan **Android SDK Command Line Tools (Latest)** sudah dicentang.

Tidak lupa untuk menginstall **flutter** dimenu **plugin**.



2.5. Install License

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



```
C:\Users\ASUS>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[!] Flutter (Channel stable, 3.24.3, on Microsoft Windows [Version 10.0.22631.4169], locale en-ID)
[!] Windows Version (Installed version of Windows is version 10 or higher)
[!] Android toolchain - develop for Android devices (Android SDK version 35.0.0)
[!] Some Android licenses not accepted. To resolve this, run: flutter doctor --android-licenses
[?] Chrome - development server
[?] Visual Studio - develop Windows apps
X Visual Studio not installed; this is necessary to develop Windows apps.
  Download at https://visualstudio.microsoft.com/downloads/.
  Please install the "Desktop development with C++" workload, including all of its default components
[?] Android Studio (version 2024.1)
[?] Connected device (3 available)
[?] Network resources

! Doctor found issues in 2 categories.

C:\Users\ASUS>flutter doctor --android-licenses
```

3. Basic Dart

3.1. Soal 1

Buatlah fungsi Dart yang membuat suatu matrix AxB 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:

1 2

3 4

5 6

Hasil transpose:

1 3 5

2 4 6

```
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.

```
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.

```
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)**.

```
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 x 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

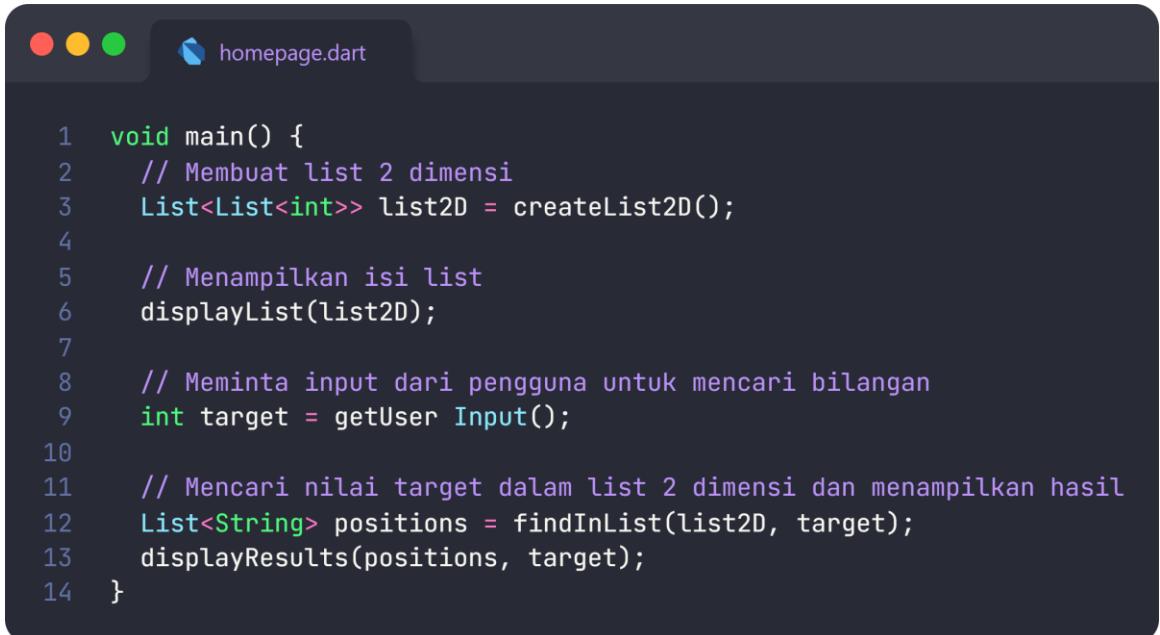
baris 4 kolom 3



homepage.dart

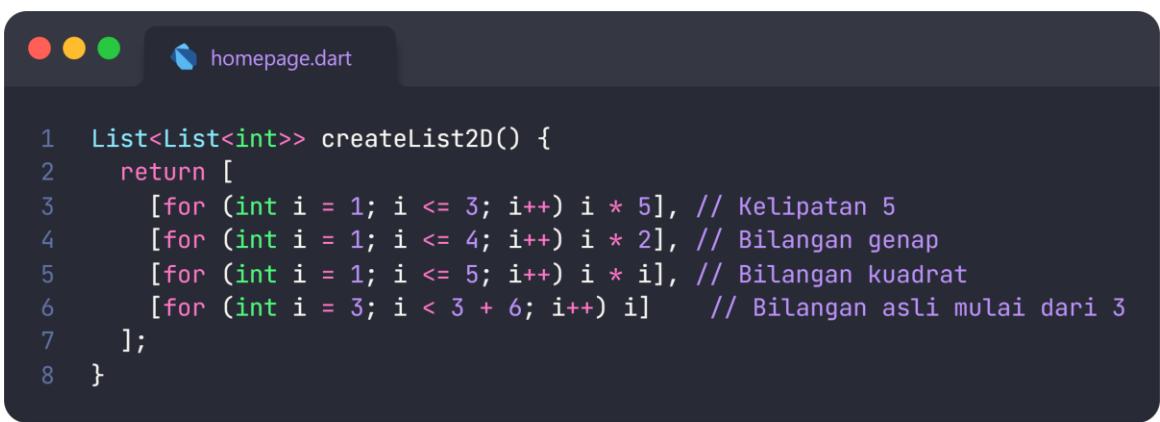
```
1 import 'dart:io';
2
3 void main() {
4     // Membuat list 2 dimensi
5     List<List<int>> list2D = createList2D();
6
7     // Menampilkan isi list
8     displayList(list2D);
9
10    // Meminta input dari pengguna untuk mencari bilangan
11    int target = getUserInput();
12
13    // Mencari nilai 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 genap, mulai dari 2
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 $target tidak ditemukan dalam list.");
69     } else {
70         print("\n$target berada di:");
71         for (String pos in positions) {
72             print(pos);
73         }
74     }
75 }
76 }
```

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     // Meminta 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)



```
void displayList(List<List<int>> list2D) {  
  print("Isi List:");  
  for (List<int> row in list2D) {  
    print(row.join(" "));  
  }  
}
```

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



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

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

```
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.

```
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:

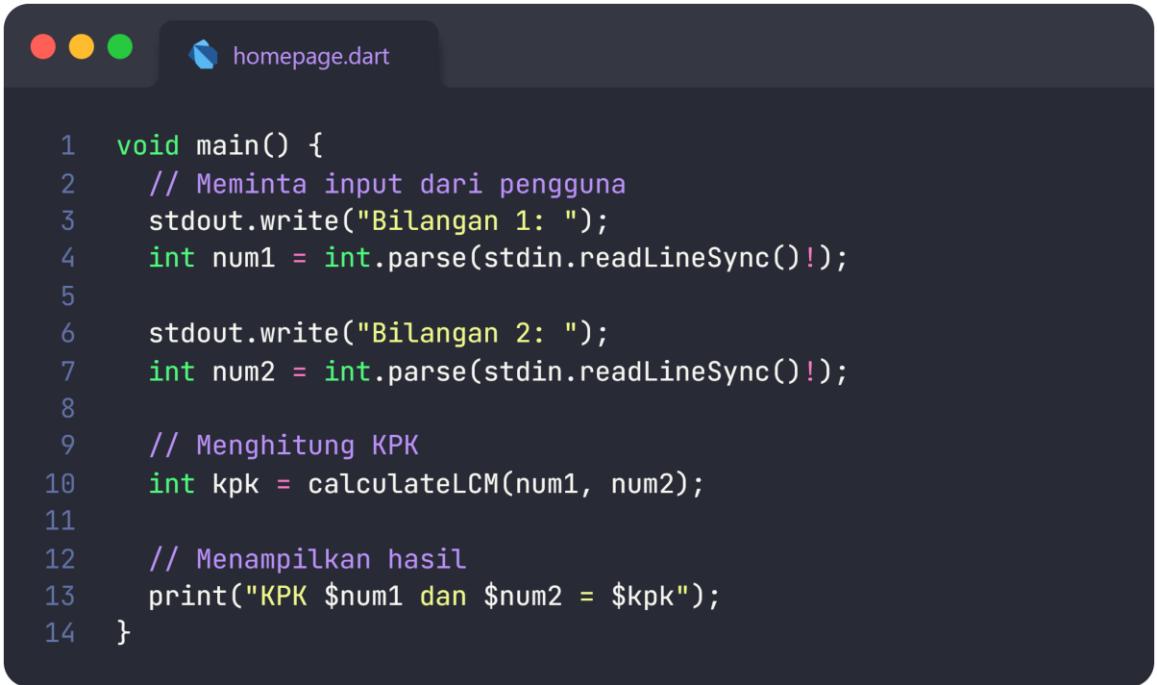
Bilangan 1: 12

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



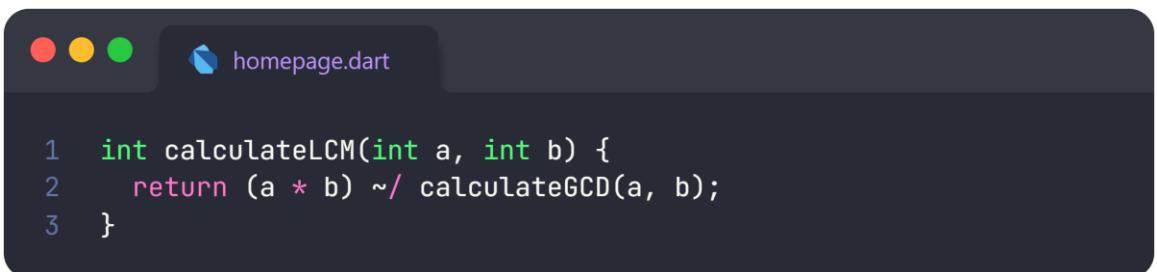
```
void main() {
    // Meminta input dari pengguna
    stdout.write("Bilangan 1: ");
    int num1 = int.parse(stdin.readLineSync()!);

    stdout.write("Bilangan 2: ");
    int num2 = int.parse(stdin.readLineSync()!);

    // Menghitung KPK
    int kpk = calculateLCM(num1, num2);

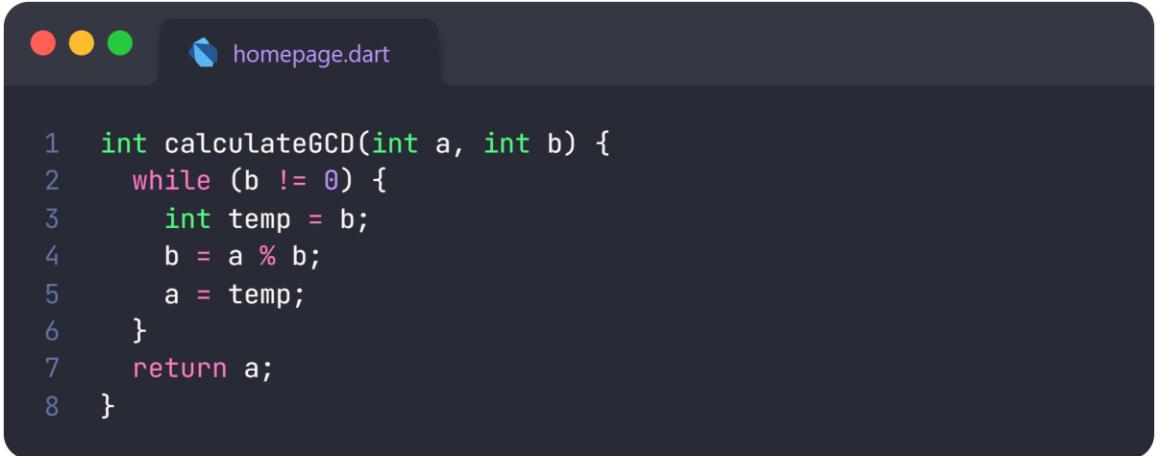
    // Menampilkan hasil
    print("KPK $num1 dan $num2 = $kpk");
}
```

- **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.



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

- 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.

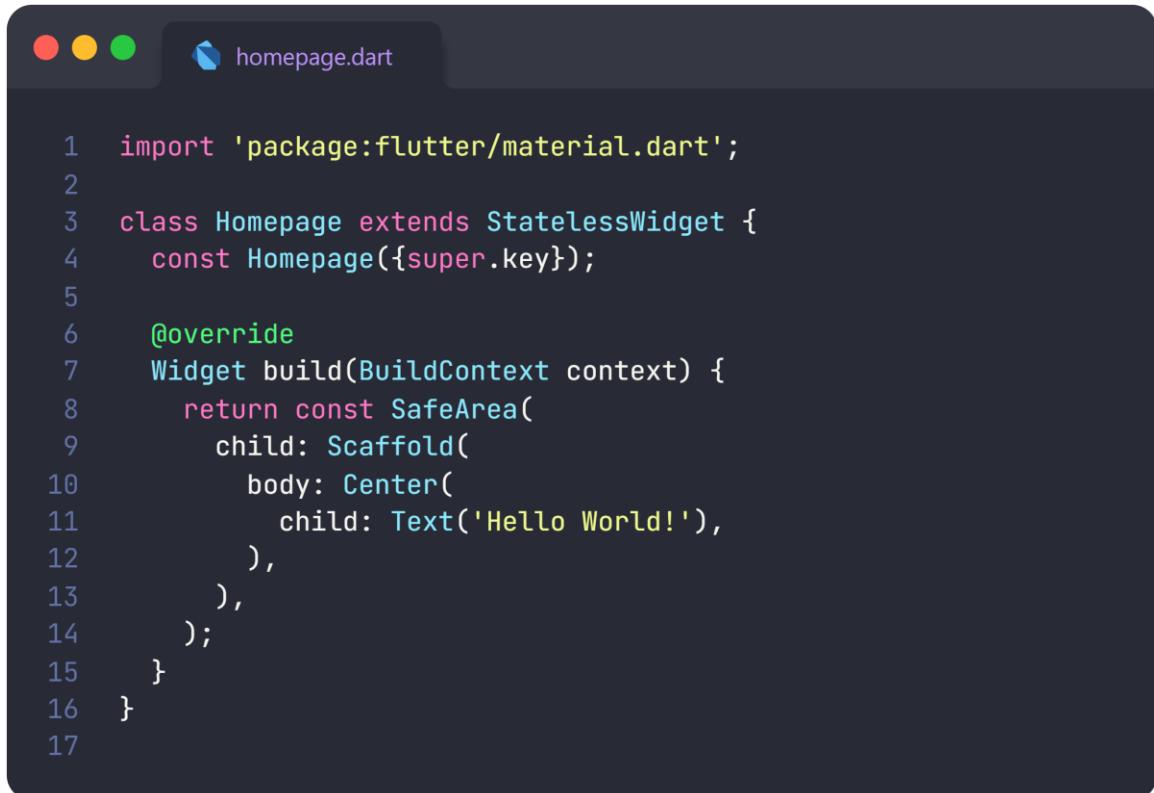


```
1 int calculateGCD(int a, int b) {
2     while (b != 0) {
3         int temp = b;
4         b = a % b;
5         a = temp;
6     }
7     return a;
8 }
```

- 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

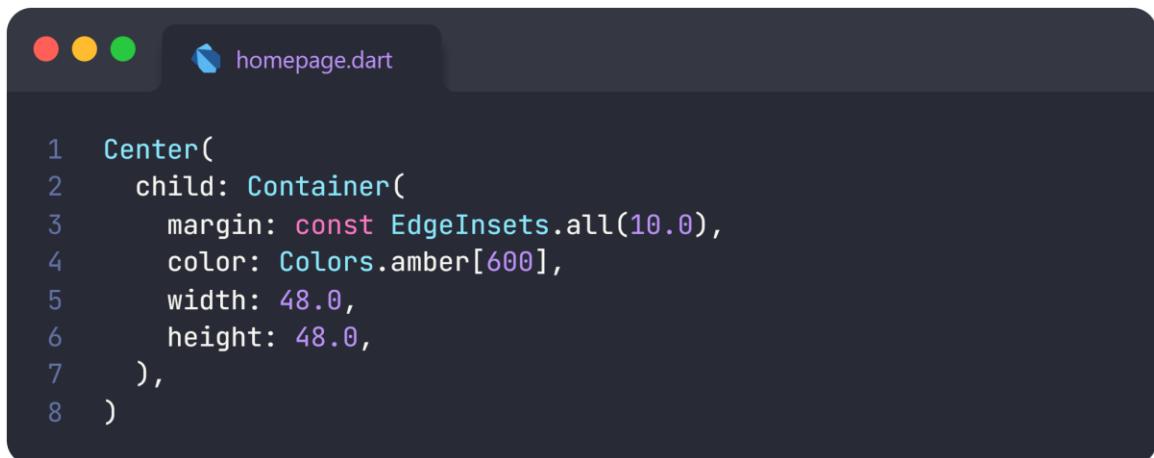


The screenshot shows a code editor window with a dark theme. The title bar says "homepage.dart". The code in the editor is:

```
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



The screenshot shows a code editor window with a dark theme. The title bar says "homepage.dart". The code in the editor is:

```
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 )
```

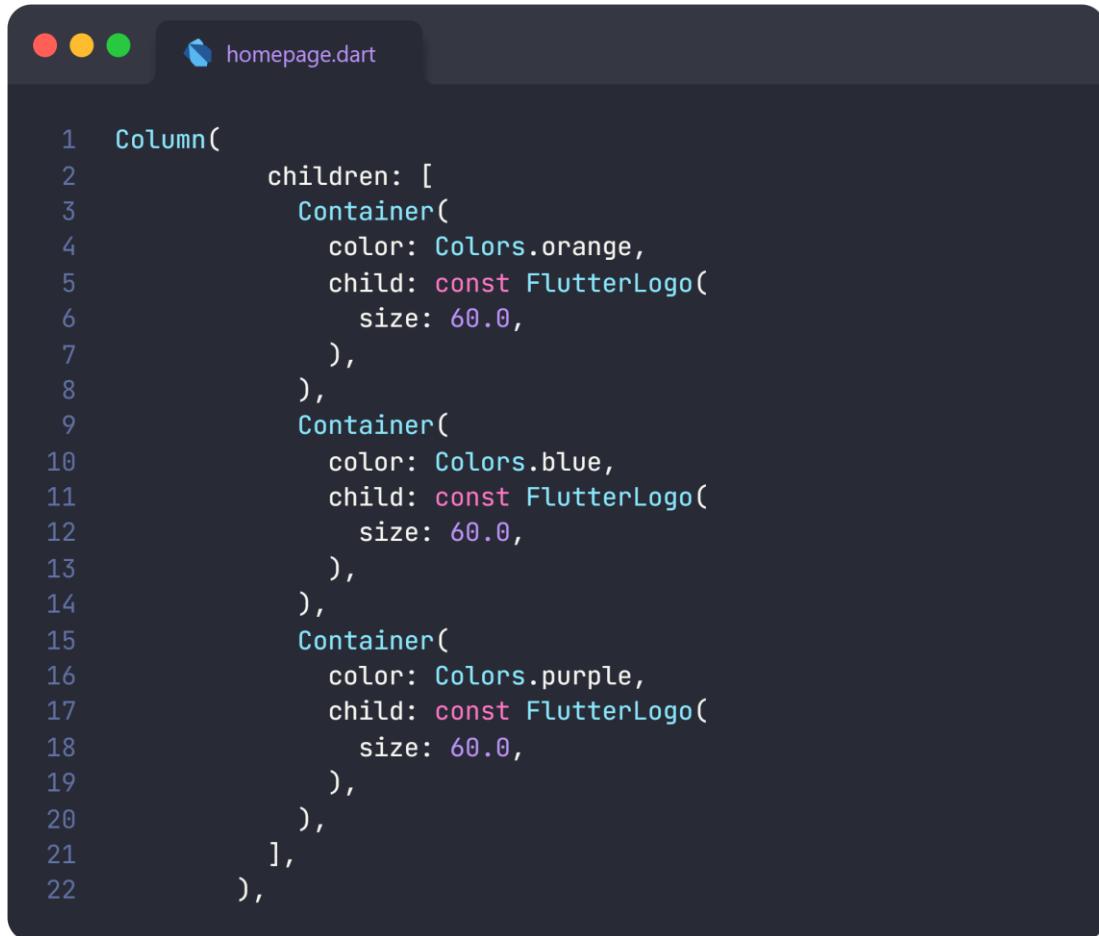
Container adalah widget dasar dalam Flutter yang digunakan untuk mengatur dan menampilkan elemen UI. Ini adalah widget yang sangat fleksibel dan dapat digunakan untuk membungkus widget lain.

4.3. Row

```
1 const Row(
2   children: <Widget>[
3     Expanded(
4       child: Text('Deliver features faster', textAlign: TextAlign.center),
5     ),
6     Expanded(
7       child: Text('Craft beautiful UIs', textAlign: TextAlign.center),
8     ),
9     Expanded(
10       child: FittedBox(
11         child: FlutterLogo(),
12       ),
13     ),
14   ],
15 )
```

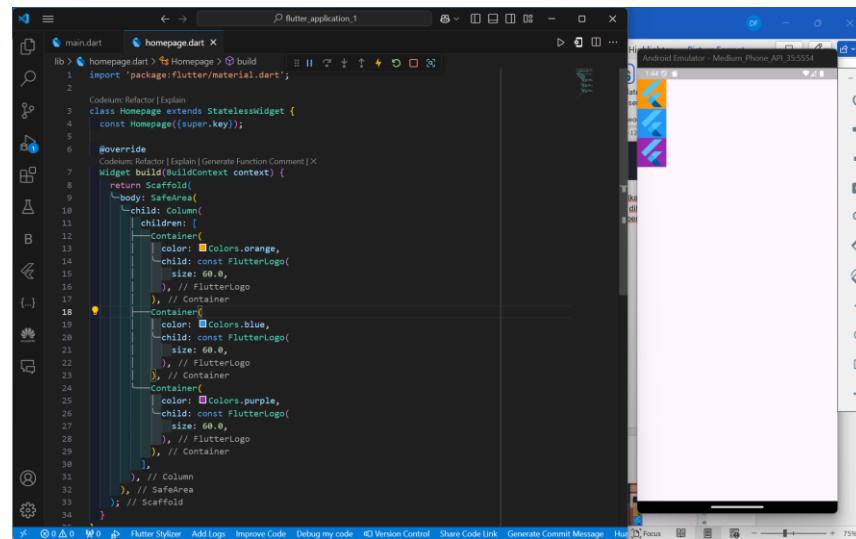
Row adalah widget dalam Flutter yang digunakan untuk menampilkan beberapa widget secara horizontal (dari kiri ke kanan) dalam satu baris.

4.4. Column

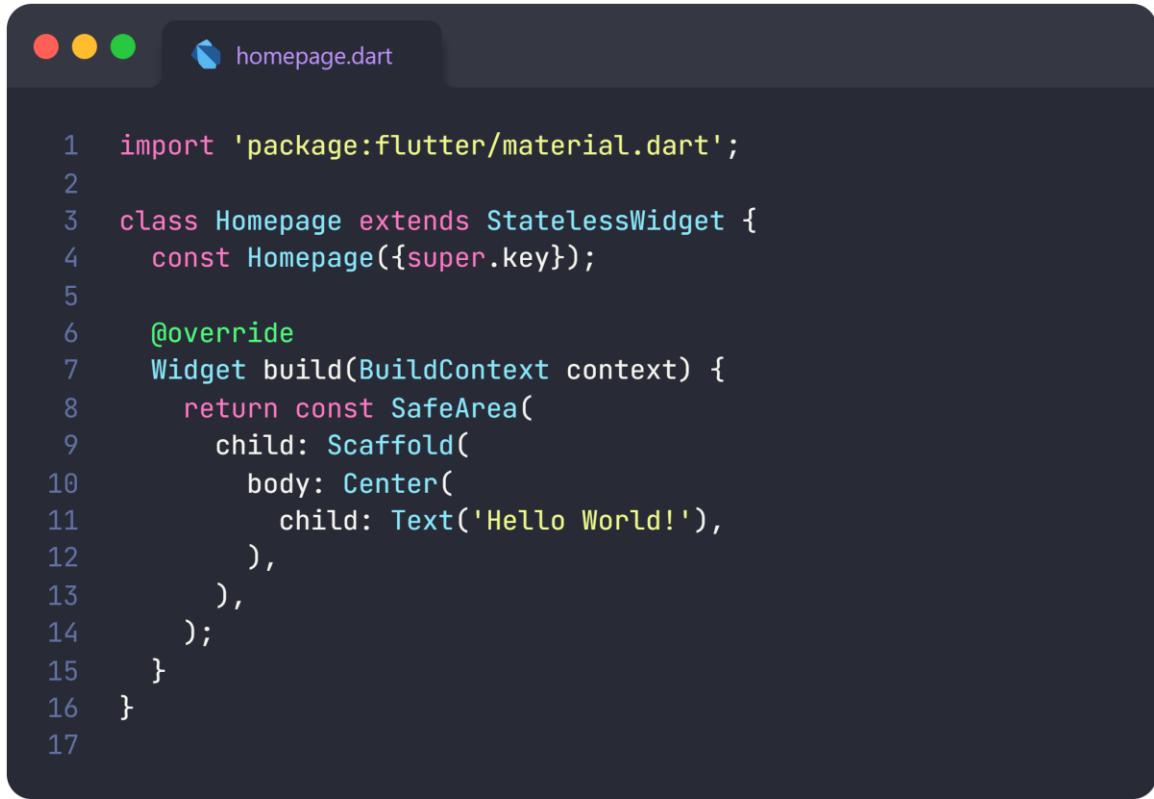


```
1 Column(
2   children: [
3     Container(
4       color: Colors.orange,
5       child: const FlutterLogo(
6         size: 60.0,
7       ),
8     ),
9     Container(
10       color: Colors.blue,
11       child: const FlutterLogo(
12         size: 60.0,
13       ),
14     ),
15     Container(
16 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



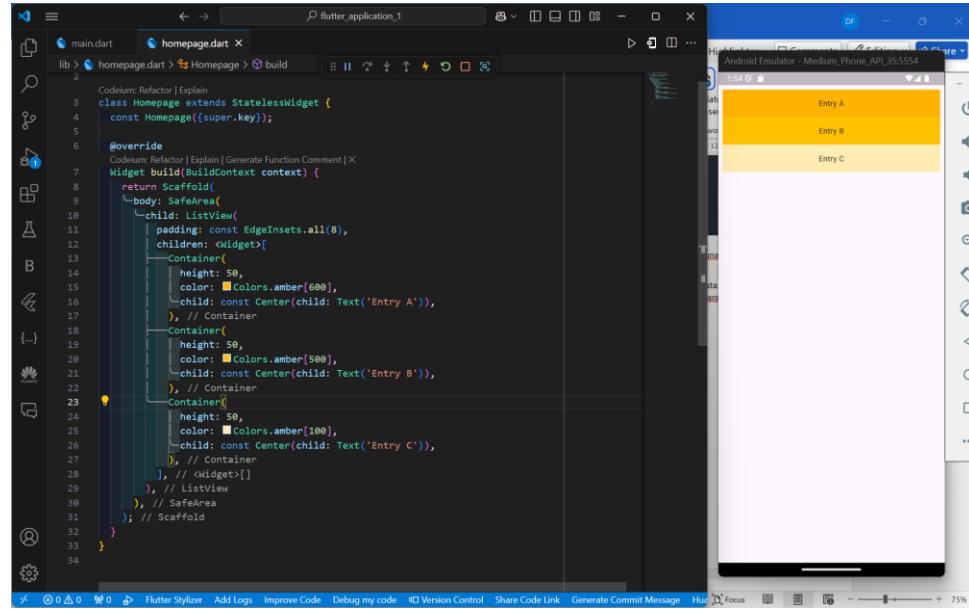
4.5. List View



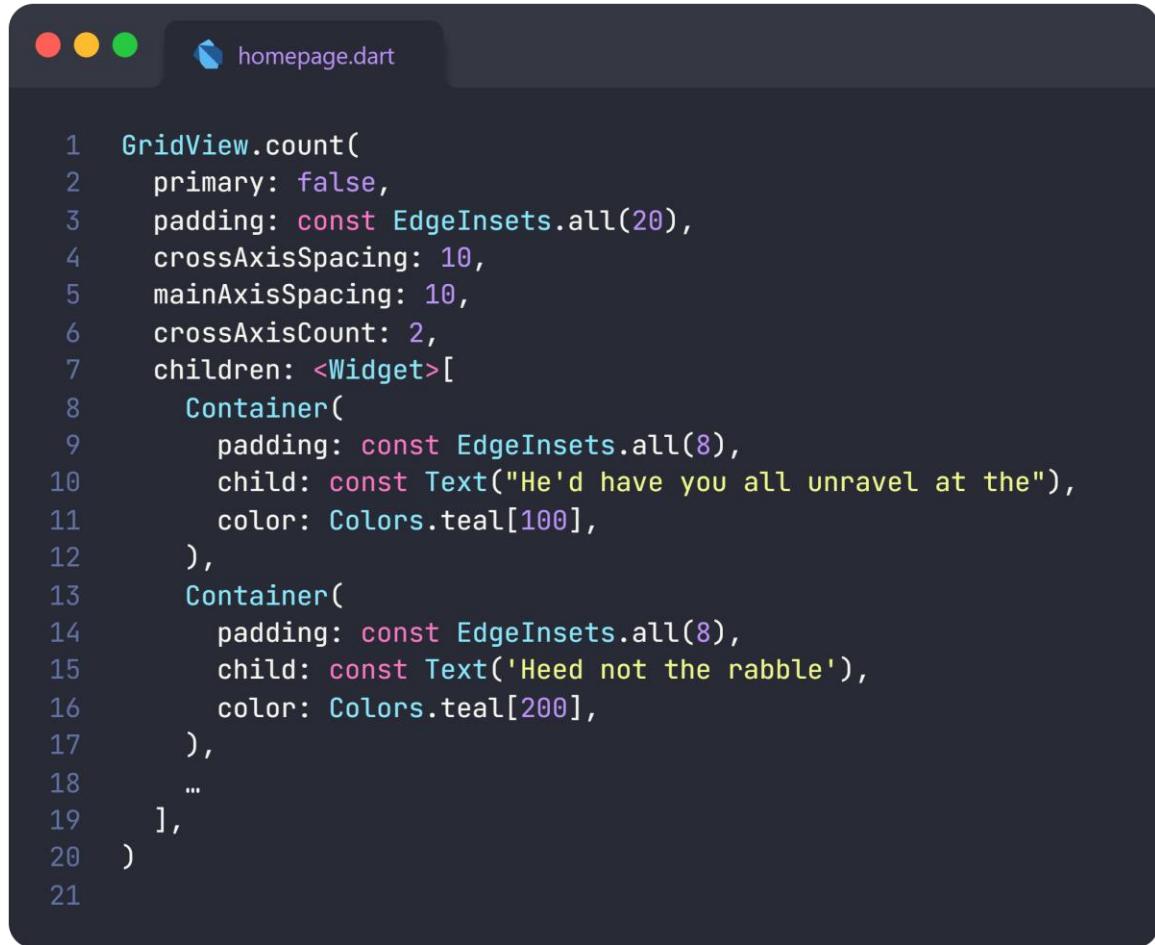
```
1 import 'package:flutter/material.dart';
2
3 class Homepage extends StatelessWidget {
4     const Homepage({super.key});
5
6     @override
7     Widget build(BuildContext context) {
8         return const SafeArea(
9             child: Scaffold(
10                 body: Center(
11              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



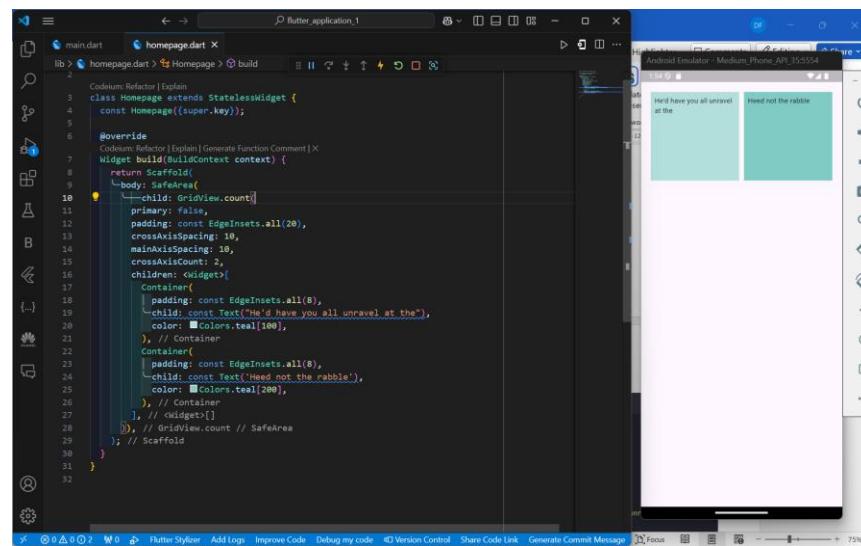
4.6. Grid View



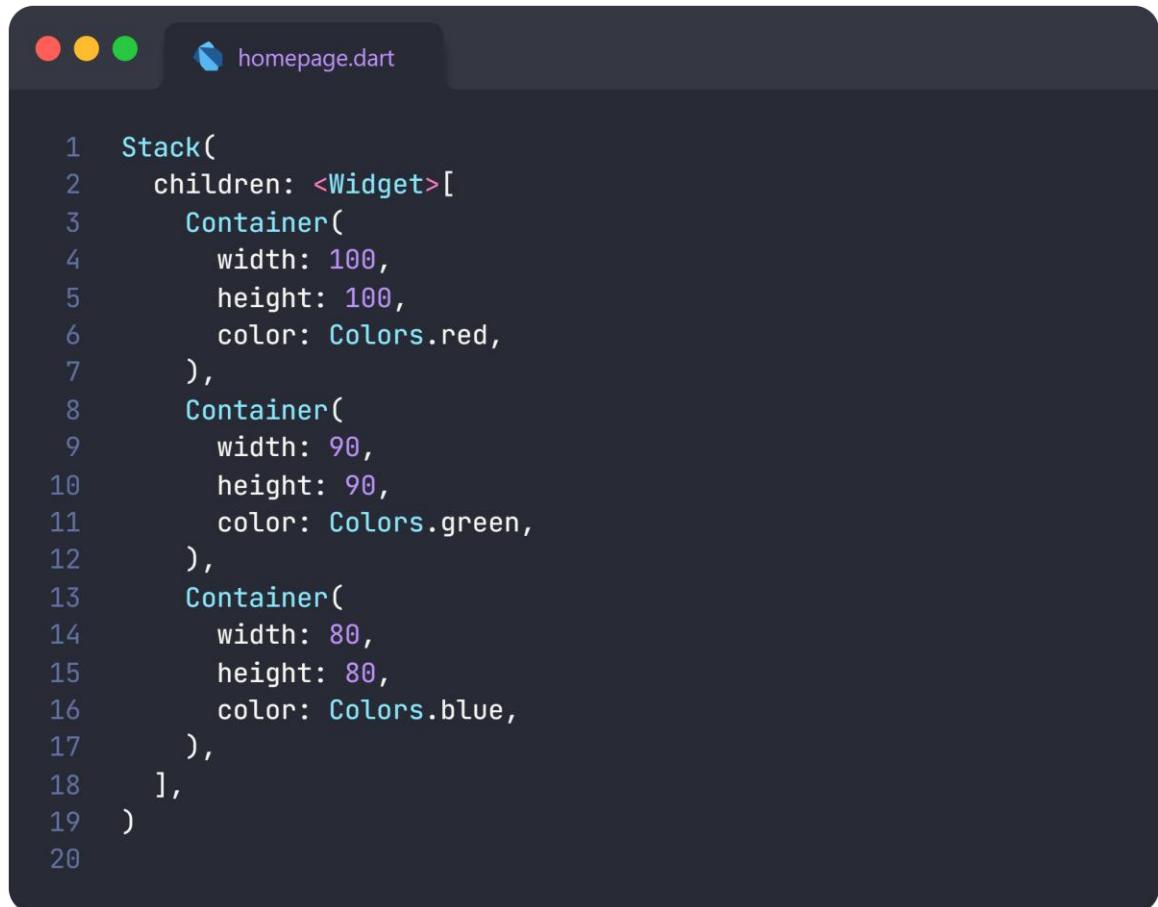
```
1  GridView.count(
2      primary: false,
3      padding: const EdgeInsets.all(20),
4      crossAxisSpacing: 10,
5      mainAxisSpacing: 10,
6      crossAxisCount: 2,
7      children: <Widget>[
8          Container(
9              padding: const EdgeInsets.all(8),
10             child: const Text("He'd 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

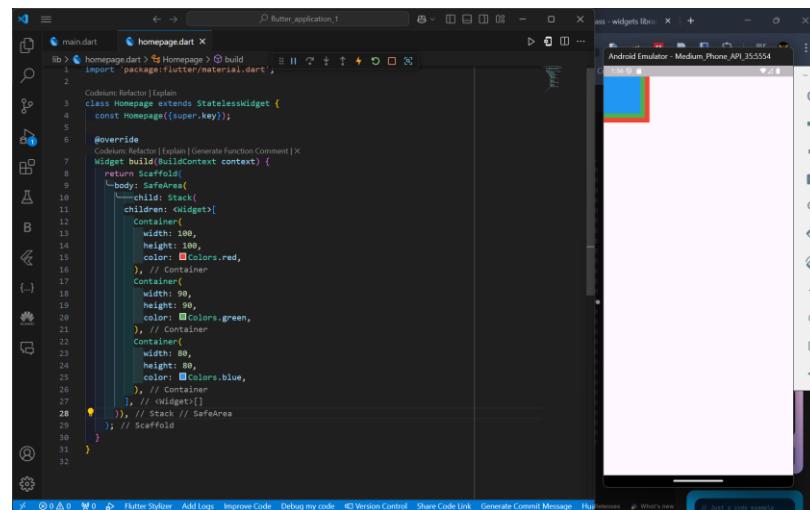


```
1 Stack(
2   children: <Widget>[
3     Container(
4       width: 100,
5       height: 100,
6       color: Colors.red,
7     ),
8     Container(
9       width: 90,
10      height: 90,
11      color: Colors.green,
12    ),
13     Container(
14       width: 80,
15       height: 80,
16       color: Colors.blue,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

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

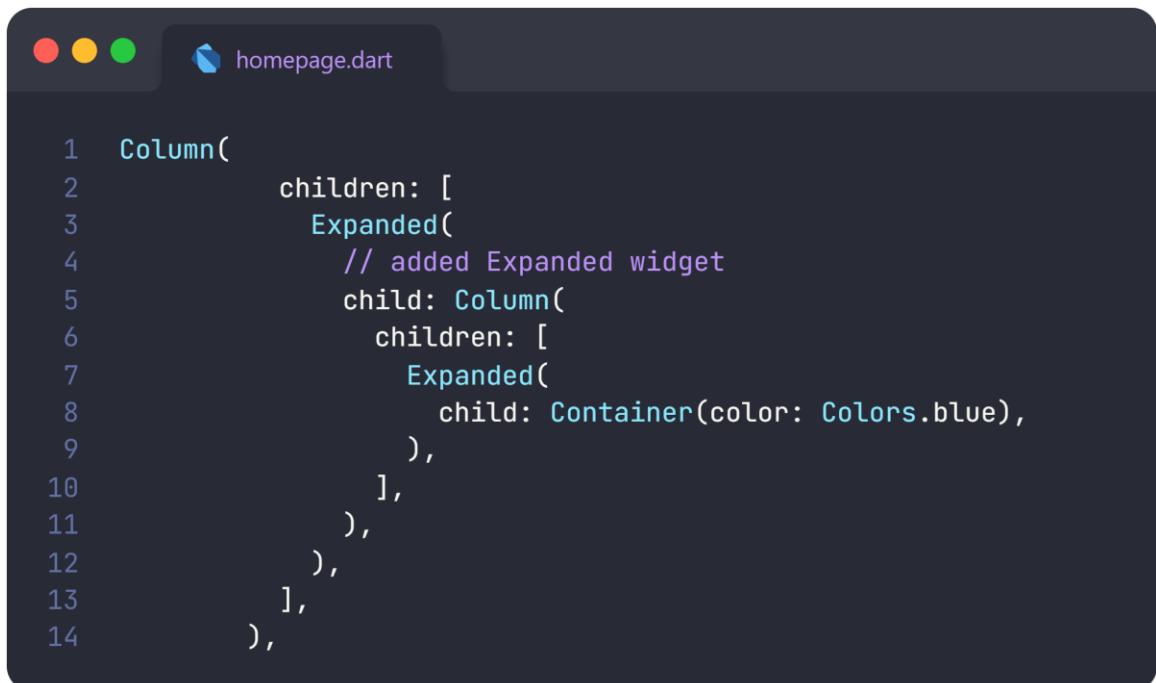
Dengan menggunakan **Stack**, kita bisa menempatkan widget di posisi yang berbeda-beda dalam satu area yang sama.

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



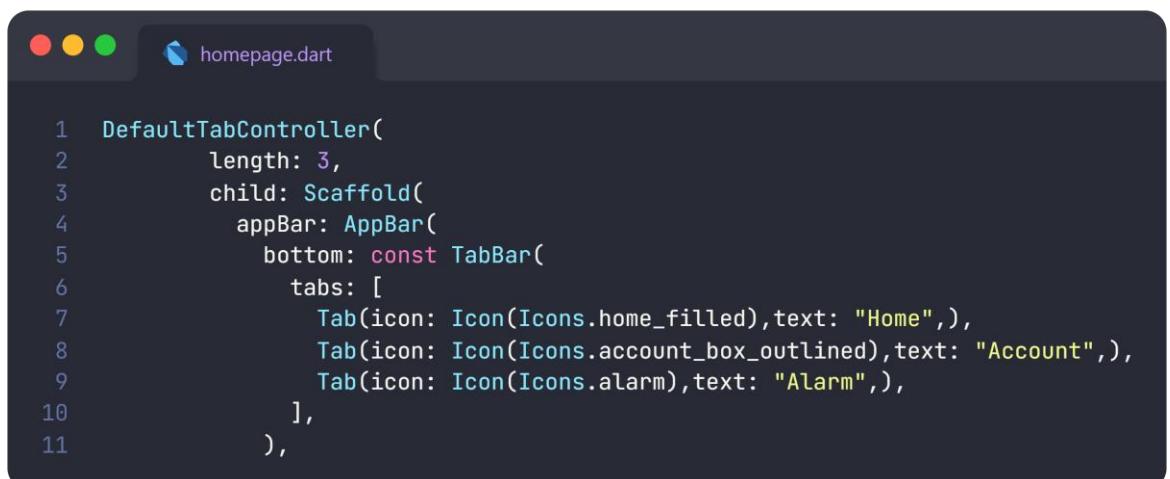
```
1 Column(
2   children: [
3     Expanded(
4       // added Expanded widget
5       child: Column(
6         children: [
7           Expanded(
8             child: Container(color: Colors.blue),
9           ),
10          ],
11        ),
12      ),
13    ],
14  ),
```

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        ),
```

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 mengulir (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.

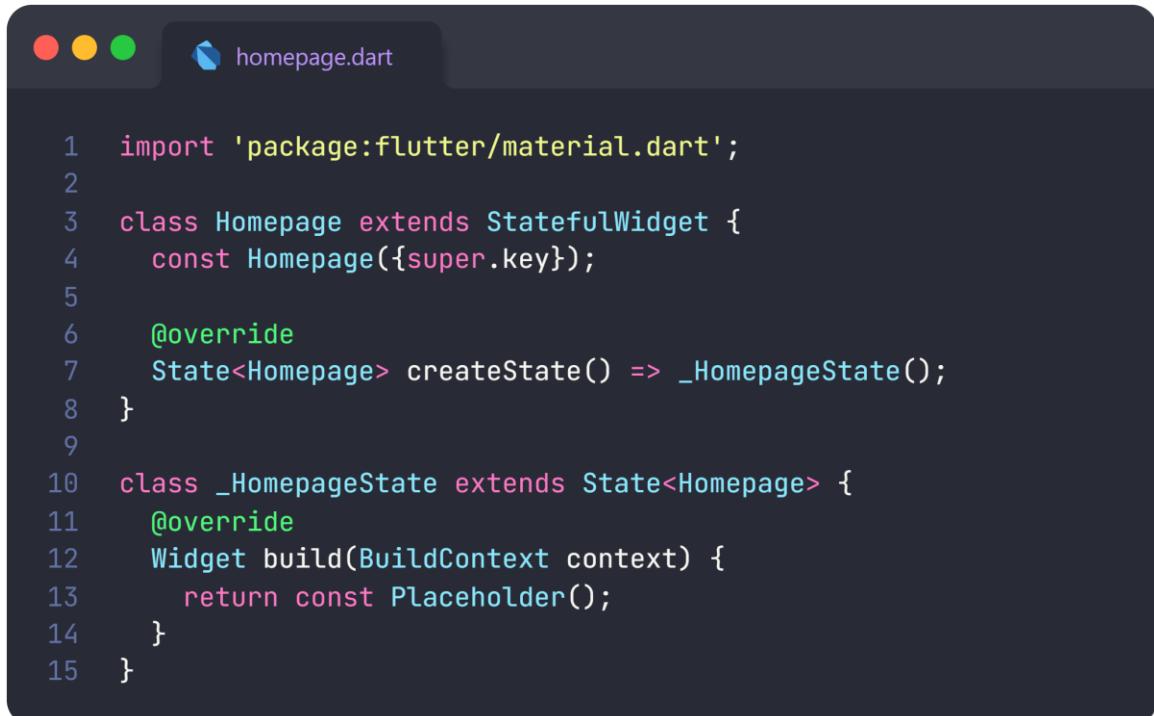
5.4. Safe Area

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

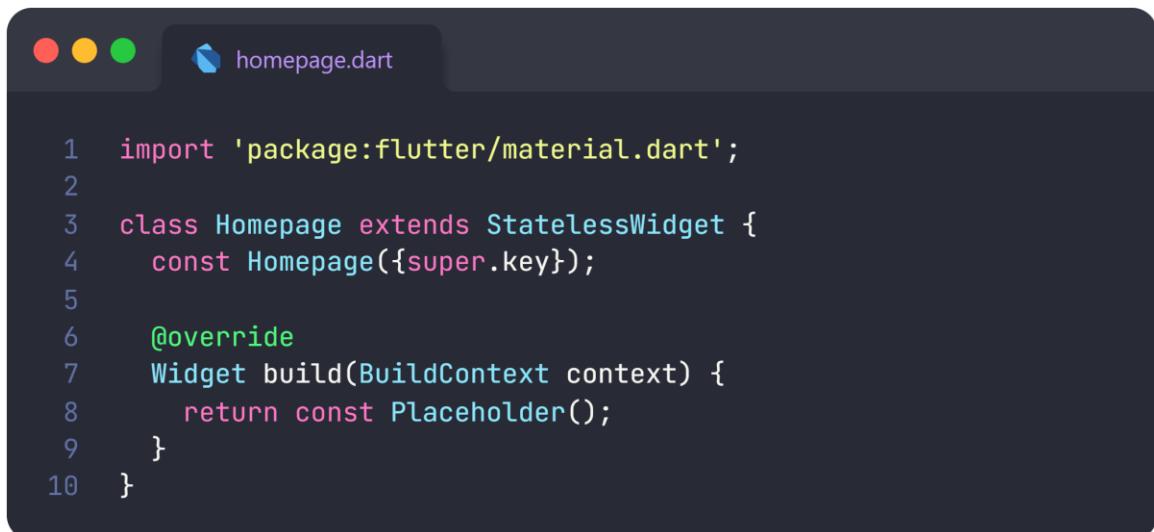
6.1. Stateful & Stateless Widget



```
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.

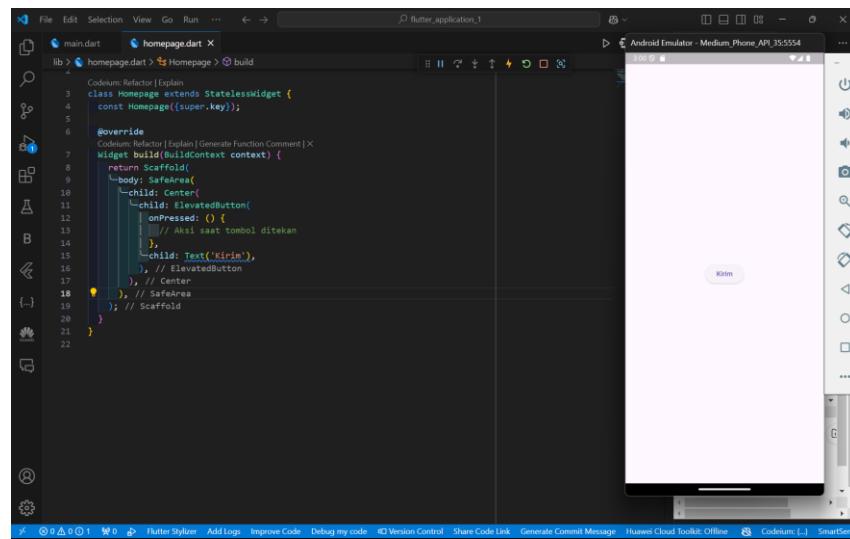
6.2. Button

6.2.1. Flat Button

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

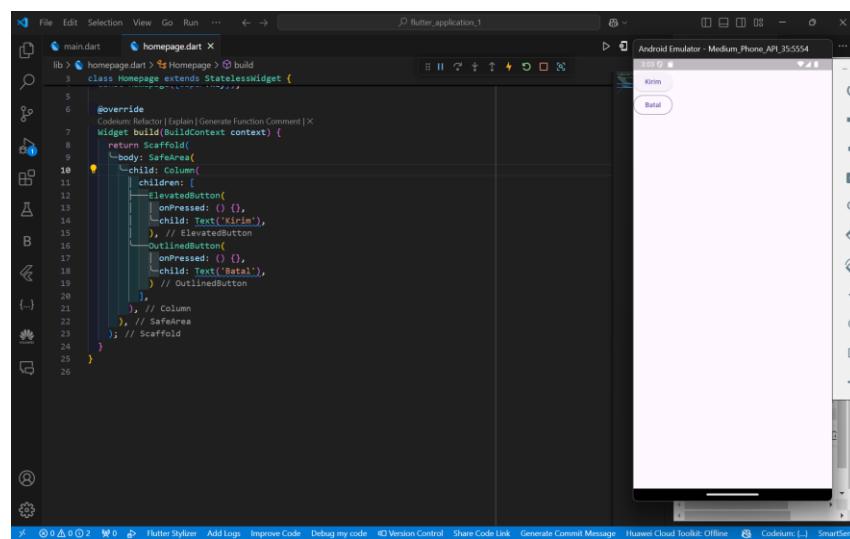
6.2.2. RaisedButton or ElevatedButton

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

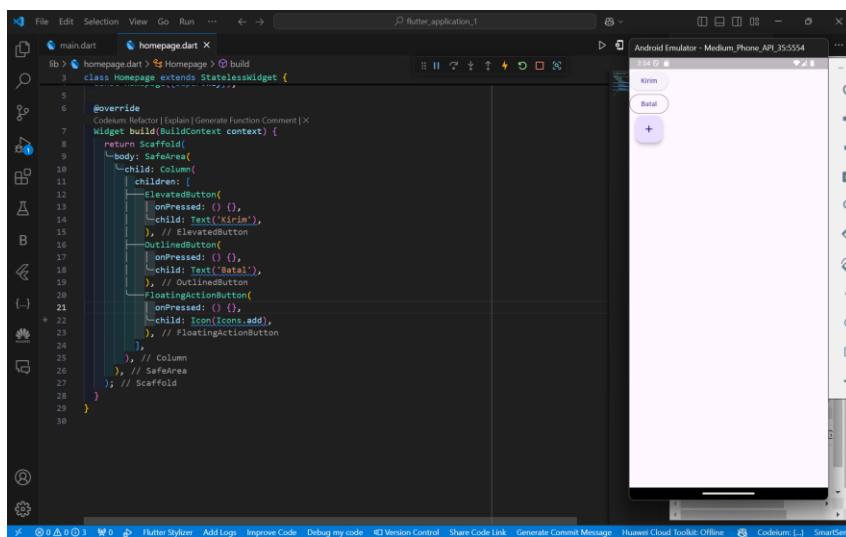
OutlineButton adalah tombol dengan garis tepi, tanpa latar belakang berwarna. Ini memberikan tampilan yang lebih ringan dan sering digunakan untuk tindakan sekunder.



6.2.4. FloatingActionButton

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

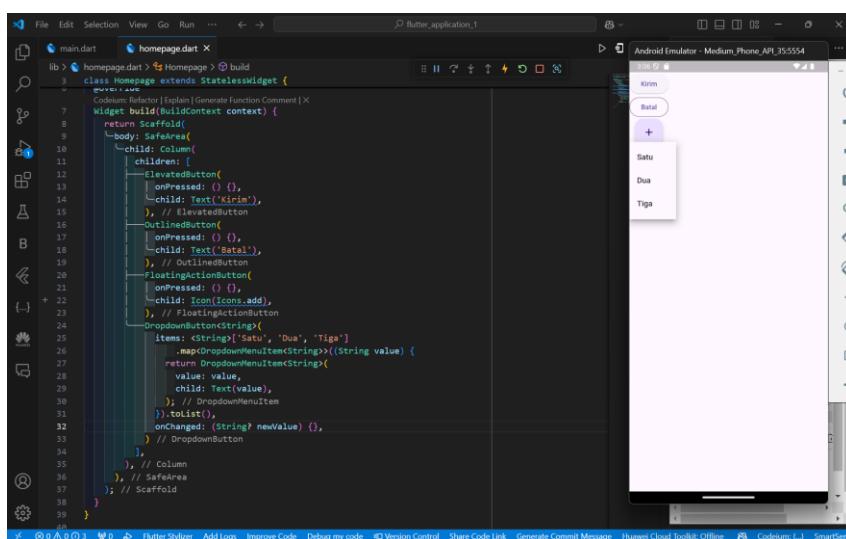
Biasanya, tombol ini mengapung di atas konten dan sering digunakan untuk menambahkan item baru, seperti "Tambah" atau "Buat".



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".

The screenshot shows the Android Studio interface. On the left is the code editor with the file 'main.dart' open, showing Dart code for a Flutter application. On the right is the 'Android Emulator - Medium_Phone_API_355554' window, which displays a white screen with a floating action button in the top right corner.

```
main.dart
homepage.dart > Homepage > build
3   class Homepage extends StatelessWidget {
7     Widget build(BuildContext context) {
25       ...
26       ...
27       ...
28       ...
29       ...
30       ...
31       ...
32       ...
33       ...
34       ...
35       ...
36       ...
37       ...
38       ...
39       ...
40       ...
41       ...
42       ...
43       ...
44       ...
45       ...
46       ...
47       ...
48       ...
49       ...
50       ...
51       ...
52       ...
53       ...
54     }
55 }
```

6.2.7. PopupMenuItem

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

The screenshot shows the Android Studio interface. On the left is the code editor with the file 'main.dart' open, showing Dart code for a Flutter application. On the right is the 'Android Emulator - Medium_Phone_API_355554' window, which displays a white screen with a 'PopupMenuButton' containing three items: 'Opsi 1', 'Opsi 2', and 'Opsi 3'.

```
main.dart
homepage.dart > Homepage > build
3   class Homepage extends StatelessWidget {
7     Widget build(BuildContext context) {
25       ...
26       ...
27       ...
28       ...
29       ...
30       ...
31       ...
32       ...
33       ...
34       ...
35       ...
36       ...
37       ...
38       ...
39       ...
40       ...
41       ...
42       ...
43       ...
44       ...
45       ...
46       ...
47       ...
48       ...
49       ...
50       ...
51       ...
52       ...
53       ...
54       ...
55     }
56 }
```

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.

The screenshot shows the Android Studio interface. On the left is the code editor with the file `homepage.dart` open. The code defines a `SnackBar` with a message "Ini adalah Snackbar!" and an action button labeled "TUTUP". On the right is the Android emulator displaying a simple UI with a floating action button and a bottom sheet titled "Tempatkan Snackbar". The bottom sheet contains the text "Ini adalah Snackbar!" and the "TUTUP" button.

```

lib> homepage.dart > HomePage > build
3   class HomePage extends StatelessWidget {
7     Widget build(BuildContext context) {
53       itemBuilder: (BuildContext context) {
54         return ['Opsi 1', 'Opsi 2', 'Opsi 3'].map((String choice) {
55           return PopupMenuItem(
56             value: choice,
57             child: Text(choice),
58           );
59         ).toList();
60       );
61     }, // PopupMenuButton
62   }, // end
63
64   ElevatedButton(
65     onPressed: () {
66       // Tampilkan Snackbar saat tombol ditekan
67       ScaffoldMessenger.of(context).showSnackBar(
68         SnackBar(
69           content: Text('Ini adalah Snackbar!'),
70           action: SnackBarAction(
71             label: 'TUTUP',
72             onPressed: () {
73               // Aksi saat tombol TUTUP ditekan
74             },
75           ), // SnackBarAction
76           duration: Duration(seconds: 3), // Durasi Snackbar
77         ), // Snackbar
78       );
79     },
80     child: Text('Tampilkan Snackbar'),
81   ), // ElevatedButton
82   ], // Column
83   ), // SafeArea
84   ); // Scaffold
85 }
86
87 
```

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 **AlertDialog**.

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.

The screenshot shows the Android Studio interface. On the left is the code editor with the file `homepage.dart` open. The code creates a `SimpleDialog` with a title "Simple Dialog" and two options: "Opsi 1" and "Opsi 2". On the right is the Android emulator displaying a floating action button and a bottom sheet titled "Tempatkan Simple Dialog". The bottom sheet contains the title "Simple Dialog" and the two options "Opsi 1" and "Opsi 2".

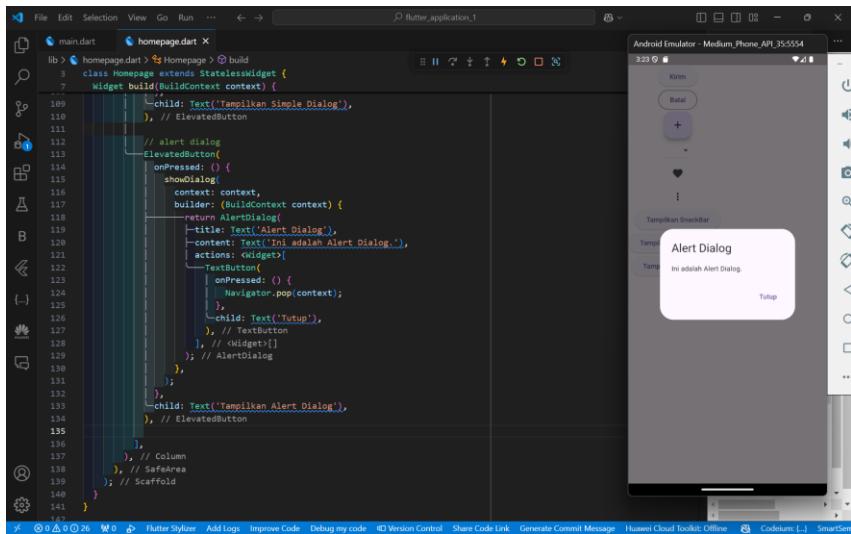
```

lib> homepage.dart > HomePage > build
3   class HomePage extends StatelessWidget {
7     Widget build(BuildContext context) {
53       itemBuilder: (BuildContext context) {
54         return ['Opsi 1', 'Opsi 2'].map((String choice) {
55           return SimpleDialogOption(
56             onPressed: () {
57               Navigator.pop(context);
58             },
59             child: Text(choice),
60           );
61         ).toList();
62       );
63     }, // SimpleDialog
64     ElevatedButton(
65       onPressed: () {
66         showSimpleDialog(
67           context: context,
68           builder: (BuildContext context) {
69             return SimpleDialog(
70               title: Text('Simple Dialog'),
71               children: [
72                 SimpleDialogOption(
73                   onPressed: () {
74                     Navigator.pop(context);
75                   },
76                   child: Text('Opsi 1'),
77                 ), // SimpleDialogOption
78                 SimpleDialogOption(
79                   onPressed: () {
80                     Navigator.pop(context);
81                   },
82                   child: Text('Opsi 2'),
83                 ), // SimpleDialogOption
84               ],
85             ); // SimpleDialog
86           },
87         );
88       },
89       child: Text('Tampilkan Simple Dialog'),
90     ), // ElevatedButton
91   ], // Column
92   ), // SafeArea
93   ); // Scaffold
94 }
95
96 
```

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.



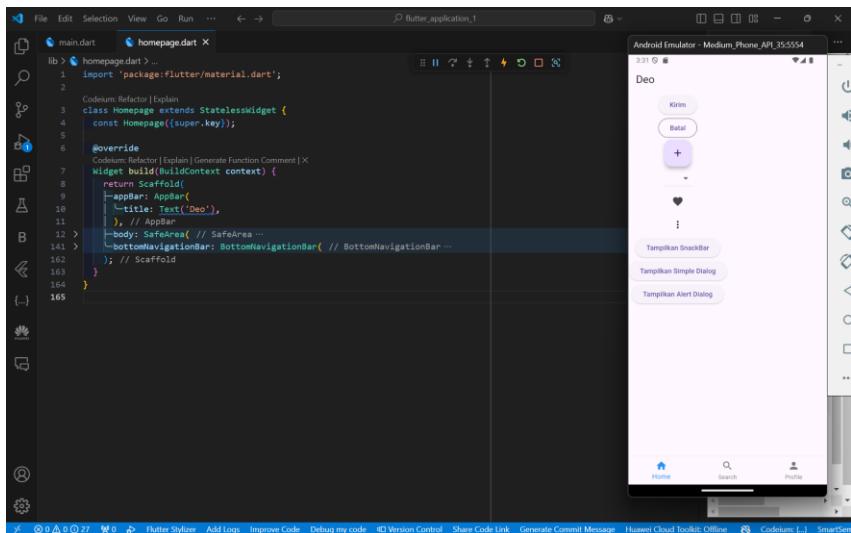
```
File Edit Selection View Go Run ... <- > flutter_application_1
lib\Homepage.dart
main.dart
homepage.dart > Homepage > build
3 class Homepage extends StatelessWidget {
7   Widget build(BuildContext context) {
109     ...
110     child: Text('Tampilkan Simple Dialog'),
111   ), // ElevatedButton
112   ...
113   // alert dialog
114   ElevatedButton(
115     onPressed: () {
116       showDialog(
117         context: context,
118         builder: (BuildContext context) {
119           return AlertDialog(
120             title: Text('Alert Dialog'),
121             content: Text('Ini adalah Alert Dialog.'),
122             actions: <Widget>[
123               TextButton(
124                 onPressed: () {
125                   Navigator.pop(context);
126                 },
127                 child: Text('Tutup'),
128               ), // TextButton
129             ], // <Widget>[]
130           ); // AlertDialog
131         },
132       );
133     },
134     child: Text('Tampilkan Alert Dialog'),
135   ), // ElevatedButton
136   ...
137 }, // Column
138 ); // SafeArea
139 ); // Scaffold
140 }
141 }
```

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.

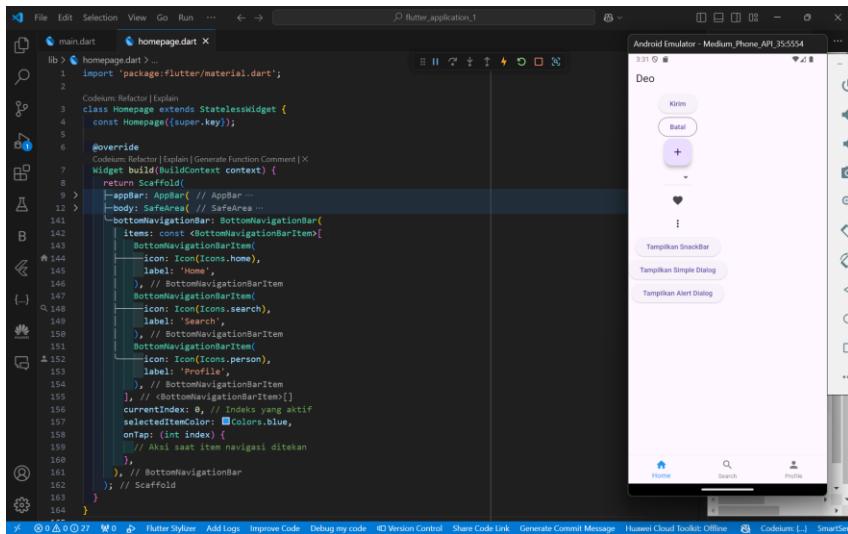


```
File Edit Selection View Go Run ... <- > flutter_application_1
lib\Homepage.dart
main.dart
homepage.dart > ...
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MyApp());
5 }
6
7 class Homepage extends StatelessWidget {
8   const Homepage({super.key});
9
10 @override
11   Widget build(BuildContext context) {
12     return Scaffold(
13       appBar: AppBar(
14         title: Text('Deo'),
15       ),
16       body: SafeArea(
17         bottomNavigationBar: BottomNavigationBar(
18           ...
19         ),
20       ),
21     );
22   }
23 }
```

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



The screenshot shows a code editor window with the title "homepage.dart". The code is written in Dart and defines two screens: "FirstScreen" and "SecondScreen".

```
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



The screenshot shows a code editor window with the file name "homepage.dart" at the top. The code implements a simple application with two screens: "FirstScreen" and "SecondScreen". The "MyApp" class defines the initial route as "/" and maps routes for "/second" to the "SecondScreen" class. The "FirstScreen" class contains an ElevatedButton that, when pressed, pushes the "SecondScreen" route onto the navigation stack. The "SecondScreen" class contains an ElevatedButton that, when pressed, pops the current route from the stack, returning to the "FirstScreen".

```
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**.

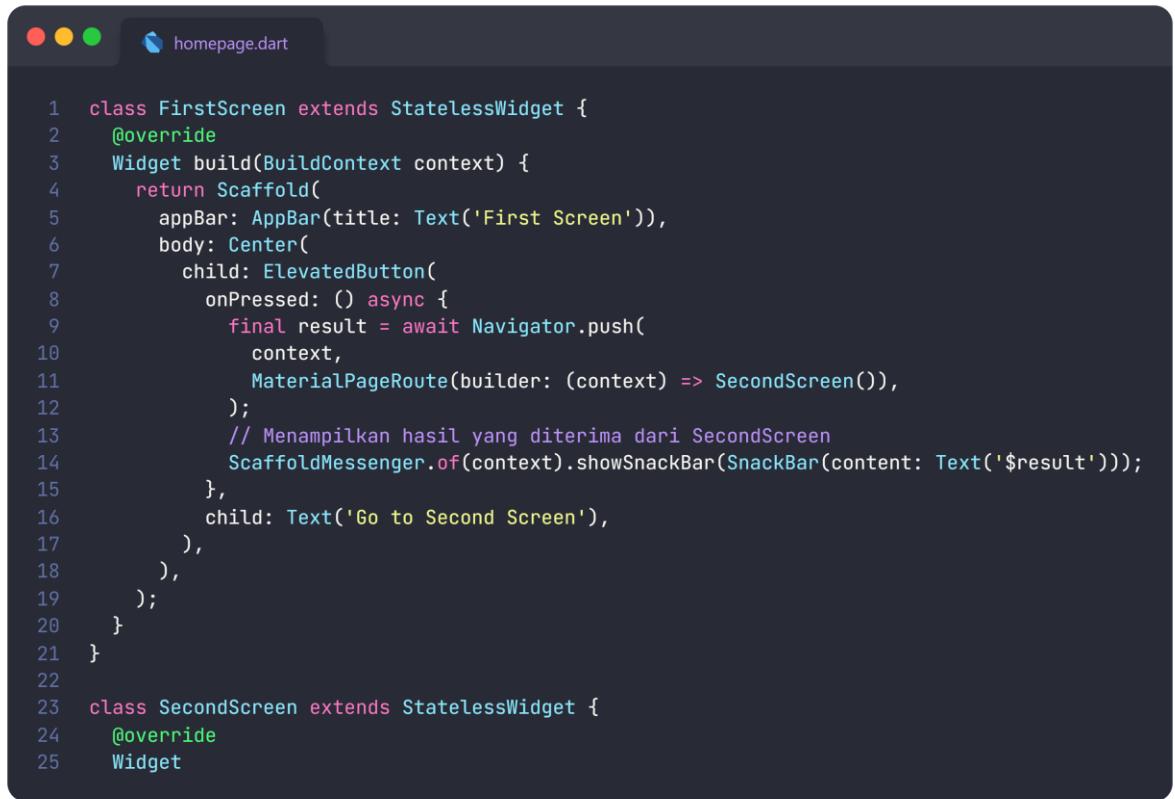
7.2.3. Pass arguments



```
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.

8. Media dan Kamera

```
1 @override
2 Widget build(BuildContext context) {
3   return Scaffold(
4     appBar: AppBar(
5       title: const Text('Minggu Ke-8'),
6     ),
7     body: SingleChildScrollView(
8       child: Column(
9         children: [
10           _buildImageFromAsset(),
11           _buildImageFromFile(),
12           _buildButton('Ambil image dari galeri', pickImageFromGallery),
13           _buildButton('Ambil image dari camera', pickImageFromCamera),
14           _buildButton('Ambil video dari galeri', pickVideoFromGallery),
15           _buildButton('Ambil video dari camera', pickVideoFromCamera),
16         ],
17       ),
18     ),
19   );
20 }
```

Fungsi utama untuk membangun tampilan halaman. Memuat AppBar, SingleChildScrollView, dan berbagai widget tombol serta gambar.

```
1 // Menampilkan gambar dari asset lokal
2 Widget _buildImageFromAsset() {
3   return Image.asset('assets/1.jpg');
4 }
```

Widget _buildImageFromAsset berfungsi untuk menampilkan gambar dari asset lokal dengan widget Image.asset dan menggunakan path lokal assets/1.jpg.

```
1 // Menampilkan gambar yang dipilih dari file
2 Widget _buildImageFromFile() {
3   return _imageFile != null
4     ? Image.file(_imageFile!)
5     : const Text('Tidak ada gambar yang dipilih');
6 }
```

Widget _buildImageFromFile berfungsi untuk menampilkan gambar yang dipilih.

Jika tidak ada gambar yang dipilih, menampilkan teks default "Tidak ada gambar yang dipilih".

```
1 // Membuat tombol dengan teks dan fungsi yang dapat disesuaikan
2 Widget _buildButton(String text, VoidCallback onPressed) {
3     return ElevatedButton(
4         onPressed: onPressed,
5         child: Text(text),
6     );
7 }
```

Widget _buildButton berfungsi untuk membuat tombol interaktif dengan parameter:

- text: teks pada tombol.
- onPressed: fungsi yang dijalankan saat tombol ditekan.

```
1 // Memilih gambar dari galeri
2 Future<void> pickImageFromGallery() async {
3     final pickedFile = await ImagePicker().pickImage(source: ImageSource.gallery);
4     _updateImageFile(pickedFile);
5 }
```

Future pickImageFromGallery berfungsi untuk memilih gambar dari galeri menggunakan ImagePicker dan memanggil _updateImageFile untuk memperbarui gambar yang dipilih.

```
1 // Mengambil gambar dari kamera
2 Future<void> pickImageFromCamera() async {
3     final pickedFile = await ImagePicker().pickImage(source: ImageSource.camera);
4     _updateImageFile(pickedFile);
5 }
```

Future pickImageFromCamera berfungsi untuk mengambil gambar dari kamera menggunakan ImagePicker dan memanggil _updateImageFile untuk memperbarui gambar yang dipilih.

```
index.dart
```

```
1 // Memilih video dari galeri
2 Future<void> pickVideoFromGallery() async {
3     final pickedFile = await ImagePicker().pickVideo(source: ImageSource.gallery);
4     _updateVideoFile(pickedFile);
5 }
```

Future pickVideoFromGallery berfungsi untuk memilih video dari galeri menggunakan ImagePicker dan memanggil `_updateVideoFile` untuk memperbarui video yang dipilih.

```
index.dart
```

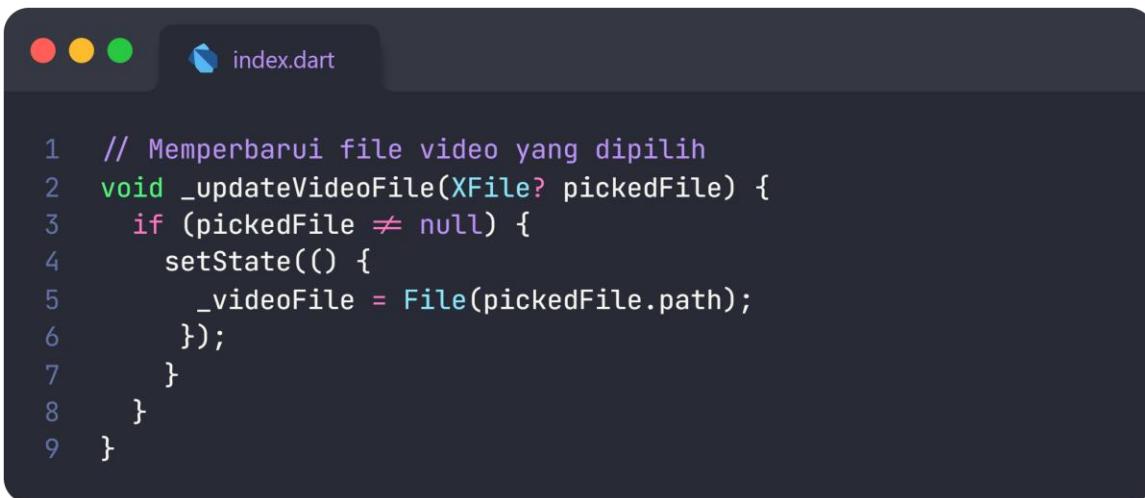
```
1 // Merekam video dari kamera
2 Future<void> pickVideoFromCamera() async {
3     final pickedFile = await ImagePicker().pickVideo(source: ImageSource.camera);
4     _updateVideoFile(pickedFile);
5 }
```

Future pickVideoFromCamera berfungsi untuk merekam video menggunakan kamera dengan ImagePicker dan memanggil `_updateVideoFile` untuk memperbarui video yang dipilih.

```
index.dart
```

```
1 // Memperbarui file gambar yang dipilih
2 void _updateImageFile(XFile? pickedFile) {
3     if (pickedFile != null) {
4         setState(() {
5             _imageFile = File(pickedFile.path);
6         });
7     }
8 }
```

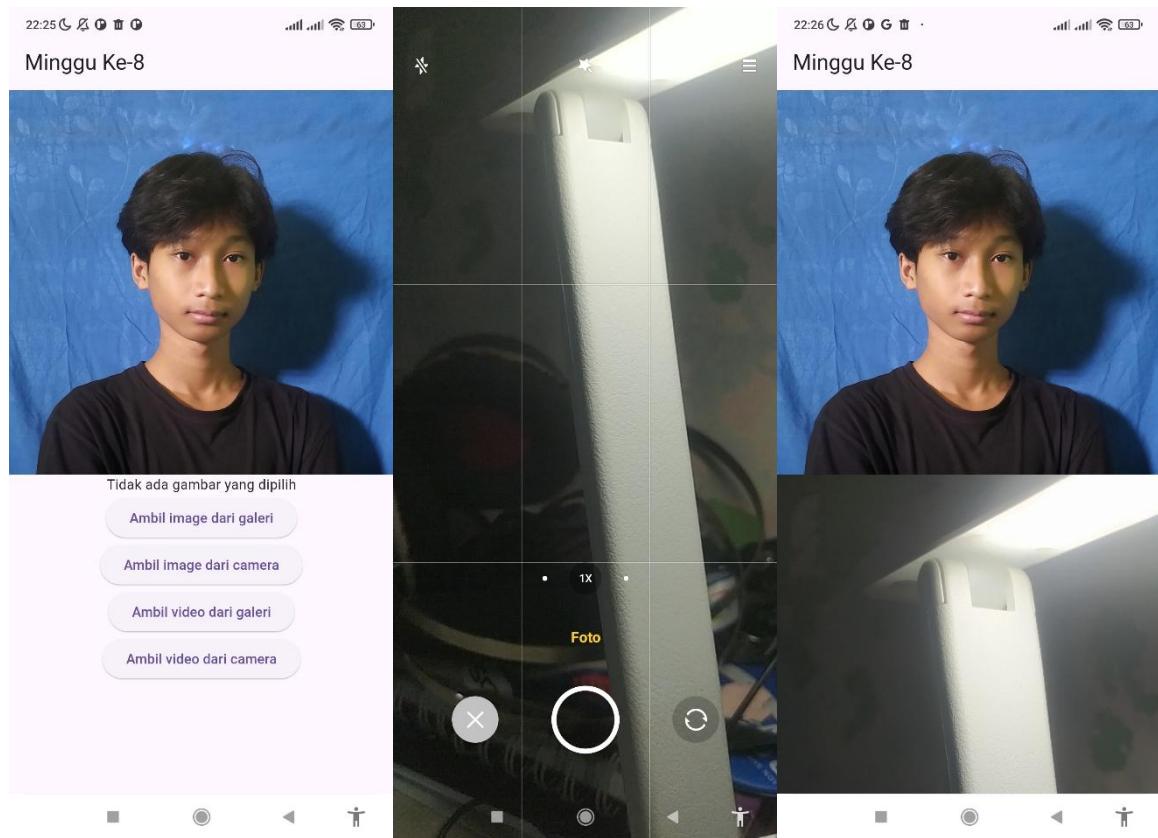
void _updateImageFile dengan parameter: `pickedFile` berupa objek XFile dan berfungsi untuk memperbarui file gambar yang dipilih menggunakan `setState`.



```
1 // Memperbarui file video yang dipilih
2 void _updateVideoFile(XFile? pickedFile) {
3     if (pickedFile != null) {
4         setState(() {
5             _videoFile = File(pickedFile.path);
6         });
7     }
8 }
9 }
```

void _updateVideoFile dengan parameter: `pickedFile` berupa objek `XFile` dan berfungsi untuk memperbarui file video yang dipilih menggunakan `setState`.

8.1. Hasil



9. Data Storage

9.1. Local Storage

9.1.1. Shared Preferences

Menyimpan dan mengambil data sederhana seperti nama pengguna.

```
index.dart
```

```
1 Future<void> saveName(String name) async {
2     SharedPreferences prefs = await SharedPreferences.getInstance();
3     await prefs.setString('User', name);
4 }
```

saveName berfungsi untuk menyimpan nama pengguna ke dalam penyimpanan menggunakan kunci tertentu. Digunakan untuk data sederhana yang perlu diakses dengan cepat.

```
index.dart
```

```
1 Future<String?> getName() async {
2     SharedPreferences prefs = await SharedPreferences.getInstance();
3     return prefs.getString('User');
4 }
```

getName, Fungsi ini mengambil nama pengguna yang telah disimpan sebelumnya. Jika data tidak ditemukan, fungsi akan mengembalikan null.

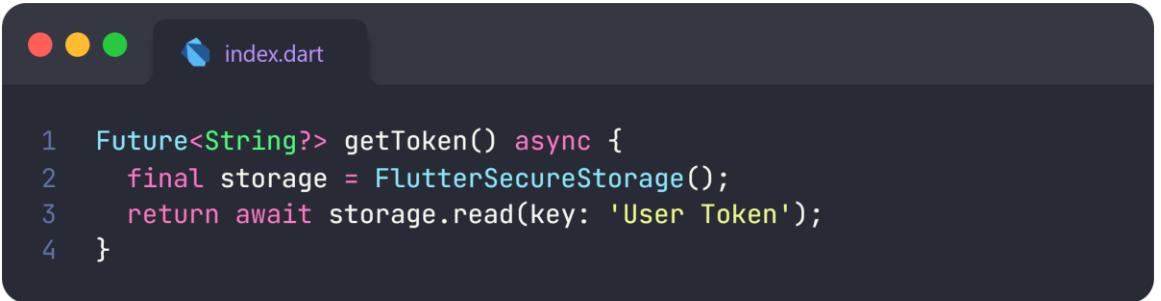
9.1.2. Flutter Secure Storage

Menyimpan dan mengambil data sensitif seperti token dengan keamanan terenkripsi.

```
index.dart
```

```
1 Future<void> saveToken(String token) async {
2     final storage = FlutterSecureStorage();
3     await storage.write(key: 'User Token', value: token);
4 }
```

saveToken: Fungsi ini menyimpan token pengguna secara terenkripsi menggunakan kunci tertentu untuk memastikan keamanannya.

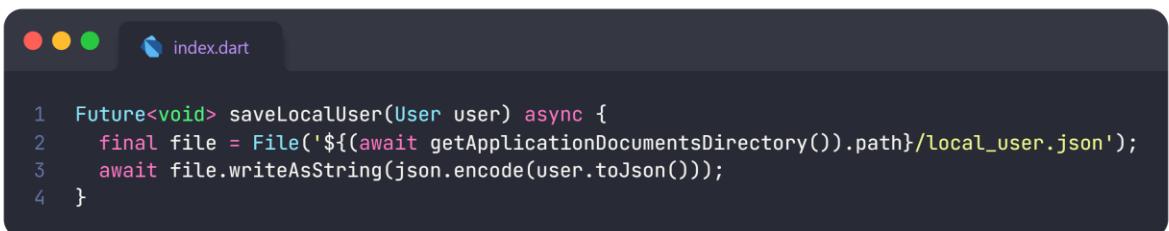


```
1 Future<String?> getToken() async {
2     final storage = FlutterSecureStorage();
3     return await storage.read(key: 'User Token');
4 }
```

getToken: Fungsi ini mengambil token yang telah disimpan sebelumnya. Jika tidak ada token yang ditemukan, fungsi mengembalikan null.

9.1.3. Penyimpanan File Lokal

Menyimpan data ke dalam file lokal untuk kebutuhan kompleks, seperti menyimpan informasi pengguna.



```
1 Future<void> saveLocalUser(User user) async {
2     final file = File('${(await getApplicationDocumentsDirectory()).path}/local_user.json');
3     await file.writeAsString(json.encode(user.toJson()));
4 }
```

saveLocalUser: Fungsi ini menyimpan data pengguna berupa objek ke dalam file lokal dalam format JSON. Berguna untuk data yang lebih kompleks.



```
1 Future<User?> getLocalUser() async {
2     final file = File('${(await getApplicationDocumentsDirectory()).path}/local_user.json');
3     if (await file.exists()) {
4         String fileContent = await file.readAsString();
5         return User.fromJson(json.decode(fileContent));
6     }
7     return null;
8 }
```

getLocalUser: Fungsi ini membaca data pengguna yang disimpan dalam file lokal. Jika file tidak ditemukan, fungsi mengembalikan null.

9.1.4. SQLite

Mengelola data terstruktur dengan kemampuan query berbasis SQL.

```
index.dart

1 Future<void> createTable() async {
2     var databasesPath = await getDatabasesPath();
3     String path = join(databasesPath, 'demo.db');
4     Database database = await openDatabase(path, version: 1, onCreate: (Database db, int version) async {
5         await db.execute('CREATE TABLE Test (id INTEGER PRIMARY KEY, name TEXT, value INTEGER, num REAL)');
6     });
7 }
```

createTable: Fungsi ini membuat tabel di database SQLite. Digunakan untuk menyimpan dan mengelola data terstruktur yang besar.

9.2. CRUD Sqflite

9.2.1. Create_todo_widget

```
index.dart

1 import 'package:flutter/material.dart';
2 import 'package:todo_app/model/todo_model.dart';
```

flutter/material.dart berfungsi untuk mengimpor paket Flutter yang menyediakan widget material design. **todo_app/model/todo_model.dart** berfungsi untuk mengimpor model **TodoModel**, yang digunakan untuk merepresentasikan data tugas.

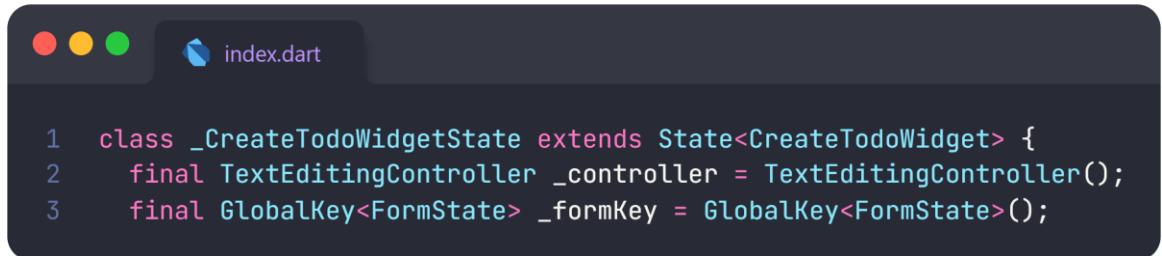
```
index.dart

1 class CreateTodoWidget extends StatefulWidget {
2     final TodoModel? todo;
3     final ValueChanged<String> onSubmit;
4
5     const CreateTodoWidget({
6         super.key,
7         this.todo,
8         required this.onSubmit,
9     });
}
```

CreateTodoWidget: Kelas utama yang merupakan widget untuk membuat atau mengedit tugas.

todo: Parameter opsional yang berisi objek **TodoModel** untuk mengedit tugas yang sudah ada.

onSubmit: Callback yang dipanggil saat tugas baru ditambahkan atau tugas yang ada diedit.



```
1 class _CreateTodoWidgetState extends State<CreateTodoWidget> {
2     final TextEditingController _controller = TextEditingController();
3     final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
```

_controller: Kontroler untuk mengelola input teks dari pengguna.

_formKey: Kunci global untuk mengidentifikasi dan mengelola status form.



```
1 @override
2 void initState() {
3     super.initState();
4     _initializeController();
5 }
```

initState: Metode yang dipanggil saat widget diinisialisasi. Di sini, kita memanggil **_initializeController** untuk mengatur teks kontroler.



```
1 void _initializeController() {
2     _controller.text = widget.todo?.title ?? '';
3 }
```

_initializeController: Mengatur teks kontroler dengan judul tugas yang ada jika **todo** tidak null, atau mengatur ke string kosong jika **todo** null.

```
1 @override
2 Widget build(BuildContext context) {
3     return AlertDialog(
4         title: _buildDialogTitle(),
5         content: _buildForm(),
6         actions: _buildDialogActions(),
7     );
8 }
```

build: Metode yang membangun tampilan widget.

Mengembalikan **AlertDialog** yang berisi judul, konten form, dan aksi dialog.

```
1 Widget _buildDialogTitle() {
2     final isEditing = widget.todo != null;
3     return Text(isEditing ? 'Edit Todo' : 'Add Todo');
4 }
```

_buildDialogTitle: Mengembalikan judul dialog berdasarkan apakah widget sedang dalam mode edit atau tambah.

```
1 Widget _buildForm() {
2     return Form(
3         key: _formKey,
4         child: TextFormField(
5             autofocus: true,
6             controller: _controller,
7             validator: _validateTitle,
8         ),
9     );
10 }
```

_buildForm: Mengembalikan widget Form yang berisi **TextField** untuk input judul tugas. Menggunakan **_formKey** untuk validasi.

```
1 String? _validateTitle(String? value) {  
2     return (value == null || value.isEmpty) ? 'Title is required' : null;  
3 }
```

_validateTitle: Validator untuk memastikan bahwa judul tidak kosong. Mengembalikan pesan kesalahan jika validasi gagal.

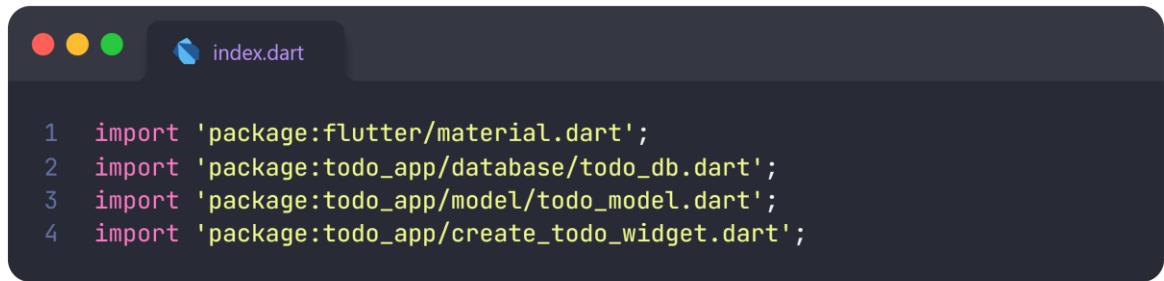
```
1 List<Widget> _buildDialogActions() {  
2     return [  
3         TextButton(  
4             onPressed: () => Navigator.pop(context),  
5             child: const Text("Cancel"),  
6         ),  
7         TextButton(  
8             onPressed: _onSubmit,  
9             child: const Text("Submit"),  
10        ),  
11    ];  
12 }
```

_buildDialogActions: Mengembalikan daftar widget tombol untuk aksi dialog, termasuk tombol "Cancel" dan "Submit".

```
1 void _onSubmit() {  
2     if (_formKey.currentState?.validate() == true) {  
3         widget.onSubmit(_controller.text);  
4         Navigator.pop(context); // Menutup dialog setelah pengiriman  
5     }  
6 }
```

_onSubmit: Metode yang dipanggil saat tombol "Submit" ditekan. Memeriksa validitas form dan memanggil callback **onSubmit** dengan teks dari kontroler jika valid. Setelah itu, dialog ditutup.

9.2.2. Todo_crud



```
1 import 'package:flutter/material.dart';
2 import 'package:todo_app/database/todo_db.dart';
3 import 'package:todo_app/model/todo_model.dart';
4 import 'package:todo_app/create_todo_widget.dart';
```

flutter/material.dart: Mengimpor paket Flutter yang menyediakan widget material design.

todo_app/database/todo_db.dart: Mengimpor kelas **TodoDb**, yang bertanggung jawab untuk interaksi dengan database.

todo_app/model/todo_model.dart: Mengimpor model **TodoModel**, yang digunakan untuk merepresentasikan data tugas.

todo_app/create_todo_widget.dart: Mengimpor widget **CreateTodoWidget**, yang digunakan untuk menambah atau mengedit tugas.



```
1 class TodoCRUD extends StatefulWidget {
2   const TodoCRUD({super.key});
3
4   @override
5   // ignore: library_private_types_in_public_api
6   _TodoCRUDState createState() => _TodoCRUDState();
7 }
```

TodoCRUD: Kelas utama yang merupakan widget untuk mengelola daftar tugas dengan operasi CRUD.



```
1 class _TodoCRUDState extends State<TodoCRUD> {
2   Future<List<TodoModel>>? _futureTodos;
3   final TodoDb _todoDB = TodoDb();
```

_futureTodos: Variabel yang menyimpan future yang mengembalikan daftar tugas.

_todoDB: Instance dari kelas **TodoDb** untuk melakukan operasi database.

```
1 @override
2 void initState() {
3     super.initState();
4     _fetchTodos();
5 }
```

_initState: Metode yang dipanggil saat widget diinisialisasi. Di sini, kita memanggil **_fetchTodos** untuk mengambil daftar tugas dari database

```
1 void _fetchTodos() {
2     setState(() {
3         _futureTodos = _todoDB.getAll();
4     });
5 }
```

_fetchTodos: Mengambil semua tugas dari database dan memperbarui state dengan future baru

```
1 Future<void> _createTodo(String title) async {
2     await _todoDB.create(title: title);
3     if (!mounted) return;
4     _fetchTodos();
5 }
```

_createTodo: Metode untuk membuat tugas baru. Memanggil metode **create** dari **TodoDb** dan memperbarui daftar tugas setelahnya.

```
1 Future<void> _updateTodo(TodoModel todo, String title) async {
2     await _todoDB.update(id: todo.id, title: title);
3     if (!mounted) return;
4     _fetchTodos();
5 }
```

_updateTodo: Metode untuk memperbarui tugas yang ada. Memanggil metode **update** dari **TodoDb** dan memperbarui daftar tugas setelahnya.

```
1 Future<void> _deleteTodo(int id) async {
2     await _todoDB.delete(id);
3     _fetchTodos();
4 }
```

_deleteTodo: Metode untuk menghapus tugas berdasarkan ID. Memanggil metode **delete** dari **TodoDb** dan memperbarui daftar tugas setelahnya.

```
1 void _showCreateTodoDialog({TodoModel? todo}) {
2     showDialog(
3         context: context,
4         builder: (context) => CreateTodoWidget(
5             todo: todo,
6             onSubmit: todo == null ? _createTodo : (title) => _updateTodo(todo, title),
7         ),
8     );
9 }
```

_showCreateTodoDialog: Menampilkan dialog untuk menambah atau mengedit tugas. Jika **todo** null, maka dialog untuk menambah tugas baru ditampilkan; jika tidak, dialog untuk mengedit tugas yang ada ditampilkan.

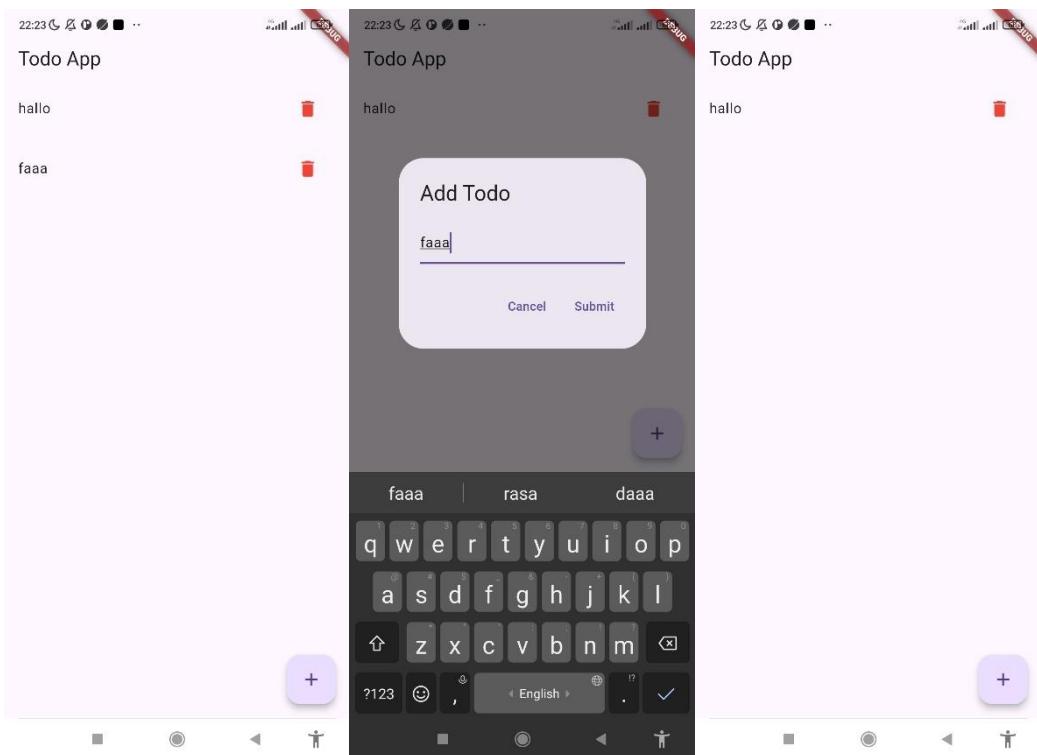


```
1 @override
2 Widget build(BuildContext context) {
3     return Scaffold(
4         appBar: AppBar(
5             title: const Text("Todo App"),
6         ),
7         floatingActionButton: FloatingActionButton(
8             onPressed: () => _showCreateTodoDialog(),
9             child: const Icon(Icons.add),
10        ),
11        body: FutureBuilder<List<TodoModel>>(
12            future: _futureTodos,
13            builder: (context, snapshot) {
14                if (snapshot.connectionState == ConnectionState.waiting) {
15                    return const Center(child: CircularProgressIndicator());
16                }
17
18                if (snapshot.hasError) {
19                    return const Center(child: Text("Error loading todos"));
20                }
21
22                final todos = snapshot.data ?? [];
23
24                if (todos.isEmpty) {
25                    return const Center(child: Text("No Todo Added Yet"));
26                }
27
28                return ListView.separated(
29                    separatorBuilder: (context, index) => const SizedBox(height: 12),
30                    itemCount: todos.length,
31                    itemBuilder: (context, index) {
32                        final todo = todos[index];
33
34                        return ListTile(
35                            title: Text(todo.title),
36                            trailing: IconButton(
37                                onPressed: () => _deleteTodo(todo.id),
38                                icon: const Icon(Icons.delete, color: Colors.red),
39                            ),
40                            onTap: () => _showCreateTodoDialog(todo: todo),
41                        );
42                    },
43                );
44            },
45        ),
46    );
47 }
```

build: Metode yang membangun tampilan widget.

Mengembalikan **Scaffold** yang berisi **AppBar**, tombol aksi mengambang untuk menambah tugas, dan **FutureBuilder** untuk menampilkan daftar tugas.

9.2.3. Hasil



10. API

10.1. todo_api

```
todo_api.dart

1 import 'dart:convert'; // Untuk konversi JSON
2 import 'package:belajar_api/Model/todo_api_model.dart'; // Model TODO
3 import 'package:http/http.dart' as http; // Library HTTP untuk request API
```

Kode ini mengimpor library yang diperlukan untuk berinteraksi dengan API dan mengonversi data JSON.

```
todo_api.dart

1 class TodoApi {
2   static const String _baseURL = 'https://6767d3e0c1de2e6421c85934.mockapi.io/todo'; // URL dasar API
3   final http.Client _client; // HTTP client untuk melakukan request
4
5   // Konstruktor: Jika client tidak disediakan, gunakan http.Client() secara default
6   TodoApi({http.Client? client}) : _client = client ?? http.Client();
7 }
```

TodoApi class ini bertanggung jawab untuk mengelola operasi CRUD (Create, Read, Update, Delete) terhadap data TODO melalui API.

```
todo_api.dart
```

```
1 Future<List<TodoApiModel>> fetchTodos() async {
2     final response = await _getRequest(_baseUrl); // GET request ke _baseUrl
3     return _handleResponse<List<TodoApiModel>>(
4         response,
5         (data) => (jsonDecode(data) as List).map((item) => TodoApiModel.fromJson(item)).toList(), // Konversi JSON ke List<TodoApiModel>
6     );
7 }
```

fetchTodos method ini mengambil semua data TODO dari API.

```
todo_api.dart
```

```
1 Future<TodoApiModel> fetchTodoById(String id) async {
2     final response = await _getRequest('${_baseUrl}/$id'); // GET request ke $_baseUrl/$id
3     return _handleResponse<TodoApiModel>(
4         response,
5         (data) => TodoApiModel.fromJson(jsonDecode(data)), // Konversi JSON ke TodoApiModel
6     );
7 }
```

fetchTodoById method ini mengambil data TODO berdasarkan ID.

```
todo_api.dart
```

```
1 Future<TodoApiModel> createTodo(String title) async {
2     final response = await _postRequest(_baseUrl, {'title': title}); // POST request dengan body JSON
3     return _handleResponse<TodoApiModel>(
4         response,
5         (data) => TodoApiModel.fromJson(jsonDecode(data)), // Konversi JSON ke TodoApiModel
6     );
7 }
```

createTodo method ini membuat data TODO baru.

```
todo_api.dart
```

```
1 Future<TodoApiModel> updateTodo(String id, String title) async {
2     final response = await _putRequest('${_baseUrl}/$id', {'title': title}); // PUT request dengan body JSON
3     return _handleResponse<TodoApiModel>(
4         response,
5         (data) => TodoApiModel.fromJson(jsonDecode(data)), // Konversi JSON ke TodoApiModel
6     );
7 }
```

updateTodo method ini memperbarui data TODO berdasarkan ID.

```
todo_api.dart
```

```
1 Future<void> deleteTodo(String id) async {
2     final response = await _deleteRequest('$_baseUrl/$id'); // DELETE request
3     _handleResponse<void>(response, (data) {}); // Tidak mengembalikan data
4 }
```

deleteTodo method ini menghapus data TODO berdasarkan ID.

Metode-metode ini digunakan untuk melakukan request HTTP dan menangani respons.

```
todo_api.dart
```

```
1 Future<http.Response> _getRequest(String url) async {
2     return await _client.get(Uri.parse(url)); // GET request
3 }
```

_getRequest: Melakukan GET request ke URL tertentu.

```
todo_api.dart
```

```
1 Future<http.Response> _postRequest(String url, Map<String, String> body) async {
2     return await _client.post(
3         Uri.parse(url),
4         body: jsonEncode(body), // Encode body ke JSON
5         headers: {'Content-Type': 'application/json'}, // Set header
6     );
7 }
```

_postRequest: Melakukan POST request dengan body JSON.

```
todo_api.dart
```

```
1 Future<http.Response> _putRequest(String url, Map<String, String> body) async {
2     return await _client.put(
3         Uri.parse(url),
4         body: jsonEncode(body), // Encode body ke JSON
5         headers: {'Content-Type': 'application/json'}, // Set header
6     );
7 }
```

_putRequest: Melakukan PUT request dengan body JSON.



todo_api.dart

```
1 Future<http.Response> _deleteRequest(String url) async {
2     return await _client.delete(Uri.parse(url)); // DELETE request
3 }
```

_deleteRequest: Melakukan DELETE request.



todo_api.dart

```
1 T _handleResponse<T>(http.Response response, T Function(String) fromJson) {
2     if (response.statusCode >= 200 && response.statusCode < 300) {
3         return fromJson(response.body); // Konversi body respons
4     } else {
5         throw Exception('Failed to load data: ${response.statusCode}'); // Lempar exception jika gagal
6     }
7 }
```

_handleResponse method ini menangani respons dari request HTTP.

10.2. todo_api_model



todo_api_model.dart

```
1 class TodoApiModel {
2     final String id;
3     final String title;
4
5     TodoApiModel({
6         required this.id,
7         required this.title,
8     });
9 }
```

Class **TodoApiModel** merepresentasikan model data untuk entitas "Todo" dengan properti **id** dan **title**.



todo_api_model.dart

```
1 factory TodoApiModel.fromJson(Map<String, dynamic> json) {  
2     return TodoApiModel(  
3         id: json['id'] as String,  
4         title: json['title'] as String,  
5     );  
6 }
```

Factory Method **fromJson** mengonversi data JSON (Map<String, dynamic>) ke objek TodoApiModel



todo_api_model.dart

```
1 Map<String, dynamic> toJson() {  
2     return {  
3         'id': id,  
4         'title': title,  
5     };  
6 }  
7 }
```

Method **toJson** mengonversi objek TodoApiModel ke bentuk JSON (Map<String, dynamic>).



todo_api_model.dart

```
1 List<TodoApiModel> todoApiModelFromJson(String jsonString) {  
2     final List<dynamic> jsonData = json.decode(jsonString);  
3     return jsonData.map((json) => TodoApiModel.fromJson(json)).toList();  
4 }
```

Function **todoApiModelFromJson** mengonversi string JSON (berisi array objek todo) ke list dari objek TodoApiModel.

```
● ○ ● todo_api_model.dart
```

```
1 List<TodoApiModel> todoApiModelFromJson(String jsonString) {  
2     final List<dynamic> jsonData = json.decode(jsonString);  
3     return jsonData.map((json) => TodoApiModel.fromJson(json)).toList();  
4 }
```

Function **todoApiModelToJson** mengonversi string JSON (berisi array objek todo) ke list dari objek TodoApiModel.

10.3. todo_crud

```
● ○ ● todo_api_crud.dart
```

```
1 @override  
2 void initState() {  
3     super.initState();  
4     _fetchTodos();  
5 }
```

initState dipanggil saat widget pertama kali dibuat. **Tujuan:** Memanggil **_fetchTodos()** untuk mengambil data todo dari API.

```
● ○ ● todo_api_crud.dart
```

```
1 void _fetchTodos() {  
2     setState(() {  
3         _futureTodos = _todoApi.fetchTodos();  
4     });  
5 }
```

_fetchTodos mengambil data todo dari API. Memperbarui state dengan data todo yang diambil dari API.



todo_api_crud.dart

```
1 FloatingActionButton _buildFloatingActionButton() {  
2     return FloatingActionButton(  
3         onPressed: () => _showCreateTodoDialog(),  
4         child: const Icon(Icons.add),  
5     );  
6 }
```

_buildFloatingActionButton menampilkan dialog untuk membuat todo baru saat tombol ditekan.



todo_api_crud.dart

```
1 Future<void> _showCreateTodoDialog() async {  
2     showDialog(  
3         context: context,  
4         builder: (_) => CreateTodoWidget(onSubmit: (title) async {  
5             await _todoApi.createTodo(title);  
6             if (!mounted) return;  
7             _fetchTodos();  
8             Navigator.of(context).pop();  
9         }),  
10    );  
11 }
```

_showCreateTodoDialog menampilkan dialog untuk membuat todo baru.



The screenshot shows a code editor window with a dark theme. The title bar says "todo_api_crud.dart". The code is a Dart file containing a single method, `_buildBody`, which returns a `Widget`. The method uses a `FutureBuilder` to handle asynchronous data retrieval from a future named `_futureTodos`. It checks the `connectionState` of the snapshot to determine what to display: a `CircularProgressIndicator` if waiting, an error message if there's an error, or a message indicating no todos if the data is empty. Finally, it returns a `ListView.separated` widget to display the list of todos, using a `SizedBox` for item separation.

```
1 Widget _buildBody() {
2     return FutureBuilder<List<TodoApiModel>>(
3         future: _futureTodos,
4         builder: (context, snapshot) {
5             if (snapshot.connectionState == ConnectionState.waiting) {
6                 return const Center(child: CircularProgressIndicator());
7             } else if (snapshot.hasError) {
8                 return Center(child: Text('Error: ${snapshot.error}'));
9             } else if (!snapshot.hasData || snapshot.data!.isEmpty) {
10                 return const Center(child: Text('No Todo Added Yet'));
11             }
12
13             final todos = snapshot.data!;
14
15             return ListView.separated(
16                 separatorBuilder: (context, index) => const SizedBox(height: 12),
17                 itemCount: todos.length,
18                 itemBuilder: (context, index) {
19                     return _buildTodoItem(todos[index]);
20                 },
21             );
22         },
23     );
24 }
```

_buildBody membangun body dari widget. Menampilkan indikator loading saat data sedang diambil. Menampilkan pesan error jika terjadi kesalahan. Menampilkan pesan jika tidak ada todo. Menampilkan daftar todo menggunakan **ListView.separated**.

```
1 Widget _buildTodoItem(TodoApiModel todo) {
2     return ListTile(
3         title: Text(todo.title),
4         trailing: IconButton(
5             onPressed: () async {
6                 await _todoApi.deleteTodo(todo.id);
7                 _fetchTodos();
8             },
9             icon: const Icon(Icons.delete),
10            color: Colors.red,
11        ),
12        onTap: () => _showUpdateTodoDialog(todo),
13    );
14 }
```

_buildTodoItem membuat item todo dalam daftar. Menampilkan judul todo. Menambahkan tombol hapus untuk menghapus todo. Menampilkan dialog update saat item todo ditekan.

```
1 Future<void> _showUpdateTodoDialog(TodoApiModel todo) async {
2     showDialog(
3         context: context,
4         builder: (context) => CreateTodoWidget(
5             todoApi: todo,
6             onSubmit: (title) async {
7                 await _todoApi.updateTodo(todo.id, title);
8                 if (!mounted) return;
9                 _fetchTodos();
10                Navigator.of(context).pop();
11            },
12        ),
13    );
14 }
```

_showUpdateTodoDialog menampilkan dialog untuk mengupdate todo.