LAPORAN PRAKTIKUM
WIDGET FLUTTER LANJUTAN DAN IMPLEMENTASI DALAM KODE PEMROGRAMAN BESERTA PENJELASAN

Disusun untuk Memenuhi Tugas Mata Kuliah Mobile & Web Service Praktik

Dosen Pengampu:

Suyud Widiono, S.Pd., M.Kom.

Asisten Dosen:

Margareta Dyah Ayu Christiasih

Disusun oleh:

Alfian Setya Dwi Saputra (5220411164)

PROGRAM STUDI INFORMATIKA

FAKULTAS SAINS & TEKNOLOGI

UNIVERSITAS TEKNOLOGI YOGYAKARTA

2024

DAFTAR ISI

BAB I
TEORI DASAR FLUTTER WIDGET LANJUTAN DAN PRAKTIKUM

1.1. Teori Dasar

1.1.1. Flutter Component Lanjutan

- Stack

Stack memungkinkan penempatan beberapa widget secara bertumpuk dalam satu kontainer, berbeda dari widget tata letak linear seperti Column atau Row. Dalam Stack, urutan penempatan widget menentukan posisi mereka, di mana widget yang terakhir ditambahkan akan muncul paling atas. Penggunaan Stack sering terlihat dalam pembuatan UI kompleks, seperti meletakkan teks di atas gambar atau menumpuk elemen UI lainnya. Penempatan widget di dalam Stack dapat diatur secara eksplisit menggunakan widget Positioned atau mengikuti tata letak default.

Stack juga memiliki properti overflow untuk mengontrol bagaimana widget bereaksi jika melebihi ukuran kontainer. Misalnya, Clip.none memungkinkan widget meluap keluar dari Stack, sementara Clip.hardEdge memotong bagian yang meluap. Stack cocok untuk skenario di mana tata letak grid atau linear tidak mencukupi, seperti animasi atau tata letak dinamis yang berubah sesuai dengan kondisi tertentu.

- Padding

Padding digunakan untuk memberikan ruang di antara widget dengan batas luar kontainernya, menjaga tata letak agar tetap proporsional dan rapi. Padding memberikan jarak yang diperlukan antara elemen UI, seperti teks dan tepi kotak, menciptakan pengalaman visual yang nyaman dan terorganisir. Ini sangat berguna dalam mengatur tata letak antar elemen sehingga tidak tampak terlalu berdekatan atau tumpang tindih.

Padding dapat dikonfigurasi secara detail menggunakan EdgeInsets, yang memberikan fleksibilitas untuk menentukan jarak di setiap sisi elemen (atas, bawah, kiri, dan kanan). EdgeInsets.symmetric memberikan jarak yang seragam di dua sisi, sementara EdgeInsets.only memungkinkan penentuan padding hanya di satu sisi tertentu. Fitur ini sangat penting dalam pembuatan tata letak responsif yang mampu menyesuaikan ukuran dan posisi berdasarkan dimensi layar.

- Align

  Align memungkinkan penempatan widget anak di dalam kontainer dengan posisi yang fleksibel. Melalui properti alignment, widget dapat ditempatkan di berbagai titik relatif, seperti di tengah, pojok kanan atas, kiri bawah, atau bahkan posisi kustom. Align menawarkan kontrol yang lebih baik atas posisi elemen tanpa memerlukan tata letak yang lebih kompleks seperti Stack. Hal ini memudahkan penyesuaian tata letak elemen UI berdasarkan ukuran layar atau kebutuhan aplikasi.

  Selain pengaturan posisi, Align mendukung penggunaan properti widthFactor dan heightFactor untuk mengatur skala elemen anak relatif terhadap kontainer induknya. Misalnya, widthFactor sebesar 0.5 membuat elemen anak mengambil setengah dari lebar kontainer. Fitur ini sangat berguna untuk menciptakan tata letak yang responsif dan fleksibel, di mana elemen dapat berubah ukuran dan posisi berdasarkan perubahan dimensi kontainer atau layar.

- Elevated Button

  ElevatedButton adalah tombol dengan efek elevasi yang memberikan kesan tiga dimensi. Tombol ini sering digunakan sebagai elemen interaktif utama dalam aplikasi, seperti untuk menavigasi halaman atau memicu aksi tertentu. Dengan properti onPressed, fungsi tertentu dapat dijalankan ketika tombol ditekan, menjadikannya elemen penting untuk aksi penting dalam aplikasi.

  Tombol ini juga mendukung kustomisasi melalui properti style, yang memungkinkan pengaturan warna, teks, ikon, serta bayangan. ElevatedButton dapat diubah untuk menyesuaikan dengan tema aplikasi, baik tema terang maupun gelap. Tombol ini juga mendukung penambahan ikon di samping teks, meningkatkan interaktivitas dan memberikan informasi visual yang lebih jelas bagi pengguna.

- Text Field

  TextField adalah widget utama untuk menerima input teks dari pengguna. Biasanya digunakan dalam form, pencarian, atau input teks lainnya. TextField memiliki berbagai opsi konfigurasi seperti batas karakter maksimum, jenis keyboard yang muncul, serta validasi input secara real-time melalui properti onChanged. Hal ini membuat TextField fleksibel dalam menangani berbagai jenis input dan validasi.

TextField juga mendukung kustomisasi visual yang beragam, seperti penambahan ikon, border, placeholder, dan label teks. Misalnya, ikon pencarian dapat ditempatkan di dalam TextField untuk memberikan petunjuk visual tentang fungsinya. Selain itu, widget ini mendukung input masking untuk format tertentu, seperti nomor telepon, serta validasi asinkron untuk kasus pengecekan data saat input sedang dilakukan.

- Image

  Image memungkinkan pemuatan gambar dari berbagai sumber, baik dari internet maupun aset lokal. Gambar dari internet dimuat menggunakan Image.network, yang menangani proses download dan caching otomatis, sehingga gambar bisa dimuat lebih cepat di penggunaan berikutnya. Ini sangat penting dalam aplikasi yang menampilkan konten dinamis seperti media sosial atau e-commerce. Gambar lokal diakses melalui Image.asset, yang memuat gambar dari direktori aplikasi yang diatur dalam pubspec.yaml.

  Gambar yang dimuat bisa disesuaikan ukurannya dengan berbagai mode BoxFit, seperti cover atau contain, yang menentukan bagaimana gambar diatur di dalam kontainer. Selain itu, gambar juga bisa diposisikan menggunakan properti alignment untuk mendapatkan tata letak yang lebih presisi sesuai kebutuhan tampilan.

- Container

  Container adalah widget serbaguna yang sering digunakan untuk membungkus elemen UI lain. Dengan Container, pengaturan properti seperti margin, padding, ukuran, warna latar belakang, serta dekorasi seperti border atau bayangan dapat dilakukan dengan mudah. Container memungkinkan elemen-elemen UI ditata dengan lebih teratur dan konsisten, terutama dalam tata letak yang kompleks.

  Container juga mendukung transformasi, seperti rotasi, skala, dan translasi, yang sangat berguna untuk efek animasi atau transisi. Karena fleksibilitasnya, Container sering digunakan sebagai elemen dasar dalam membangun tata letak responsif dan dinamis yang dapat beradaptasi dengan perubahan kondisi aplikasi atau ukuran layar.

- Icon

Icon adalah widget yang digunakan untuk menampilkan ikon dalam aplikasi, membantu dalam menyampaikan informasi secara visual. Flutter menyediakan pustaka ikon bawaan melalui Icons, mencakup ikon-ikon umum seperti navigasi, peringatan, dan tindakan UI lainnya. Ikon sering digunakan sebagai pelengkap untuk tombol, menu, atau elemen navigasi lainnya.

Selain ikon bawaan, ikon dari pustaka eksternal seperti FontAwesome atau Material Icons dapat digunakan untuk memperluas opsi desain. Ikon dapat dengan mudah disesuaikan dalam hal ukuran, warna, dan penempatan, membuatnya elemen yang fleksibel dalam menciptakan antarmuka pengguna yang interaktif dan mudah dipahami.

1.1.2. ListView Widget

- ListView

  ListView adalah widget yang digunakan untuk menampilkan daftar widget secara vertikal atau horizontal dalam bentuk scrollable. Ini sangat berguna untuk menampilkan daftar elemen yang panjang, seperti daftar kontak, artikel, atau produk. ListView bekerja dengan cara memuat semua elemen sekaligus, yang bisa menyebabkan masalah performa jika daftar sangat panjang, karena semua widget akan di-render di memori. Namun, dalam kasus daftar pendek, ListView sangat efisien dan mudah digunakan.

  ListView juga mendukung berbagai fitur seperti scrolling otomatis, kontrol scroll ke posisi tertentu, dan penambahan margin atau padding antara elemen. Widget ini fleksibel dalam penggunaannya karena dapat digabungkan dengan widget lain seperti Container, Padding, atau InkWell untuk menambah gaya atau interaksi. Dengan konfigurasi yang sederhana, ListView bisa menjadi komponen penting untuk menampilkan data dalam bentuk yang lebih terstruktur dan mudah dijelajahi.

- ListView Builder

  ListView.builder adalah variasi dari ListView yang dirancang khusus untuk menangani daftar panjang dengan performa yang lebih baik. Alih-alih memuat semua elemen sekaligus, ListView.builder hanya memuat widget yang terlihat di layar, sedangkan widget lain akan di-render saat diperlukan, mengikuti mekanisme lazy loading. Pendekatan ini memungkinkan aplikasi bekerja dengan

lebih efisien dalam hal penggunaan memori dan performa, terutama ketika bekerja dengan data yang jumlahnya besar.

ListView.builder membutuhkan dua properti utama: itemCount dan itemBuilder. itemCount menentukan jumlah total elemen dalam daftar, sedangkan itemBuilder adalah fungsi yang akan memuat setiap widget sesuai kebutuhan, memberikan fleksibilitas dalam cara data ditampilkan. Ini sangat ideal untuk skenario di mana data diambil dari sumber eksternal, seperti API, karena elemen baru dapat dimuat saat pengguna melakukan scrolling.

- ListView Separated

  ListView.separated adalah variasi lain dari ListView yang menambahkan elemen pemisah (separator) antara item dalam daftar. Selain memuat elemen daftar seperti ListView.builder, ListView.separated juga membutuhkan fungsi tambahan bernama separatorBuilder, yang bertanggung jawab untuk menentukan bagaimana pemisah antara item akan ditampilkan. Pemisah ini bisa berupa garis, ruang kosong, atau widget kustom seperti gambar atau ikon.

  ListView.separated sangat berguna dalam situasi di mana diperlukan pemisah visual yang jelas antara elemen daftar, misalnya dalam daftar pesan, daftar kontak, atau produk. Dengan memanfaatkan pemisah, pengguna bisa lebih mudah membedakan antara satu item dan item lainnya, memberikan pengalaman navigasi yang lebih terstruktur dan rapi.

- ListView Custom

  ListView.custom memberikan fleksibilitas penuh untuk membangun ListView dengan perilaku dan tampilan khusus. Widget ini memungkinkan penggunaan SliverChildDelegate, yang menawarkan kontrol lebih granular terhadap cara elemen-elemen dalam daftar di-render dan ditampilkan. ListView.custom biasanya digunakan ketika pendekatan ListView standar atau ListView.builder tidak cukup fleksibel untuk memenuhi kebutuhan desain atau performa aplikasi.

  Dengan ListView.custom, elemen dapat dioptimalkan lebih lanjut untuk situasi yang sangat spesifik, seperti memanfaatkan SliverChildListDelegate untuk daftar elemen statis atau SliverChildBuilderDelegate untuk daftar dinamis yang memerlukan optimisasi khusus. Widget ini ideal digunakan dalam kasus yang sangat khusus, di mana kontrol penuh terhadap proses rendering diperlukan untuk mengoptimalkan performa atau menciptakan pengalaman pengguna yang unik.

1.1.3. GridView Widget

- GridView Builder

  GridView.builder adalah widget yang digunakan untuk menampilkan daftar elemen dalam bentuk grid dengan jumlah elemen yang tidak diketahui atau sangat besar. GridView.builder memuat elemen-elemen secara dinamis menggunakan mekanisme lazy loading, sehingga hanya elemen yang terlihat di layar yang di-render. Ini memungkinkan performa yang lebih baik ketika menampilkan daftar panjang, seperti galeri foto atau katalog produk. Dengan itemBuilder yang digunakan untuk membangun setiap elemen grid berdasarkan indeks, widget ini fleksibel dalam menangani berbagai jenis data.

  Widget ini juga memerlukan properti gridDelegate, yang digunakan untuk menentukan bagaimana grid diatur, seperti jumlah kolom atau jarak antar elemen. GridView.builder memungkinkan penyesuaian tata letak yang responsif, di mana jumlah kolom dapat berubah secara otomatis berdasarkan ukuran layar atau perangkat. Hal ini membuat GridView.builder ideal untuk aplikasi yang membutuhkan tata letak grid yang dinamis dan scalable, dengan data yang diambil secara bertahap dari API atau sumber lain.

- GridView Count

  GridView.count adalah widget yang memudahkan pembuatan grid dengan jumlah kolom tetap, di mana jumlah elemen dalam setiap baris dapat diatur melalui properti crossAxisCount. Dengan pendekatan ini, setiap elemen grid memiliki ukuran yang sama, dan tata letak grid akan selalu menjaga jumlah kolom yang telah ditentukan, terlepas dari ukuran layar atau perangkat. GridView.count sangat berguna untuk menampilkan data statis dalam format yang terstruktur, seperti daftar produk dengan jumlah kolom tetap.

  Selain itu, GridView.count menyediakan opsi tambahan seperti pengaturan jarak antar elemen dengan properti crossAxisSpacing dan mainAxisSpacing, yang memungkinkan penyesuaian grid agar lebih rapi dan sesuai dengan gaya desain aplikasi. Widget ini cocok digunakan ketika diperlukan grid dengan jumlah kolom yang konstan, dan ketika data yang ditampilkan memiliki ukuran atau proporsi yang seragam, seperti ikon, kartu produk, atau gambar kecil.

- GridView Extent

GridView.extent adalah widget yang mirip dengan GridView.count, tetapi dengan perbedaan utama pada cara penentuan ukuran kolom. Dalam GridView.extent, ukuran kolom diatur melalui properti maxCrossAxisExtent, yang menentukan ukuran maksimum untuk setiap elemen grid. Jumlah kolom dalam grid akan otomatis disesuaikan berdasarkan ukuran layar, sehingga jika layar lebih besar, lebih banyak kolom yang bisa ditampilkan, tanpa perlu menentukan jumlah kolom secara eksplisit.

Penggunaan GridView.extent sangat ideal untuk situasi di mana ukuran elemen grid relatif seragam, tetapi tata letaknya perlu menyesuaikan dengan berbagai ukuran layar. Dengan menyesuaikan jumlah kolom secara otomatis, widget ini memberikan fleksibilitas lebih dalam membuat tata letak yang responsif dan adaptif, seperti untuk galeri gambar atau grid dengan item yang memiliki dimensi tetap, tetapi bisa tampil dalam berbagai jumlah kolom tergantung pada perangkat atau orientasi layar.

- GridView Custom

  GridView.custom memberikan kontrol penuh terhadap bagaimana elemen-elemen grid di-render, dengan menggunakan SliverChildDelegate yang memberikan lebih banyak fleksibilitas dibandingkan GridView standar. Widget ini sangat berguna dalam kasus di mana elemen-elemen grid memerlukan optimisasi khusus, misalnya ketika bekerja dengan daftar elemen yang sangat panjang atau elemen yang membutuhkan cara rendering yang tidak biasa. Dengan GridView.custom, pengembang dapat menentukan secara rinci bagaimana elemen-elemen dimuat dan ditampilkan.

  Seperti GridView.builder, widget ini juga mendukung lazy loading, tetapi dengan tingkat kustomisasi yang lebih tinggi. SliverChildListDelegate bisa digunakan untuk daftar elemen statis, sementara SliverChildBuilderDelegate memungkinkan pemuatan dinamis elemen berdasarkan kebutuhan. GridView.custom cocok untuk situasi di mana tata letak grid perlu dioptimalkan lebih lanjut, seperti pada aplikasi yang membutuhkan kontrol penuh terhadap performa atau desain tata letak yang sangat khusus.

1.1.4. Flutter Navigation

- Navigator Push dan Pop

Navigator.push dan Navigator.pop adalah dua metode utama yang digunakan untuk navigasi di aplikasi Flutter. Navigator.push berfungsi untuk mendorong (push) sebuah halaman baru ke atas stack navigasi, yang berarti menampilkan halaman baru di atas halaman saat ini. Metode ini biasanya digunakan untuk menavigasi ke halaman lain tanpa menutup halaman sebelumnya, sehingga pengguna dapat kembali ke halaman sebelumnya dengan menggunakan metode Navigator.pop. Navigator.pop berfungsi untuk menghapus (pop) halaman yang sedang aktif dari stack dan kembali ke halaman sebelumnya. Ini berguna ketika ingin kembali dari halaman detail ke halaman utama, misalnya, setelah melihat informasi lebih mendalam tentang sebuah item.

Navigator.push mengambil argumen berupa BuildContext dan route, biasanya menggunakan MaterialPageRoute atau CupertinoPageRoute untuk menentukan transisi halaman. Penggunaan Navigator.pop dapat disesuaikan dengan pengembalian nilai (return value) dari halaman sebelumnya, yang sering dipakai untuk mengembalikan data dari halaman yang telah ditutup. Misalnya, jika navigasi melibatkan pengisian formulir, Navigator.pop dapat digunakan untuk mengembalikan data yang dimasukkan kembali ke halaman sebelumnya. Kombinasi push dan pop menciptakan pengalaman navigasi yang mulus di dalam aplikasi.

- Navigator Push Replacement

Navigator.pushReplacement digunakan ketika ingin mengganti halaman saat ini dengan halaman baru tanpa menyimpan halaman sebelumnya di dalam stack. Ini berarti halaman yang didorong (push) ke atas stack akan menggantikan halaman yang sedang aktif, sehingga ketika pengguna kembali, mereka tidak akan bisa kembali ke halaman yang diganti. Navigator.pushReplacement berguna dalam skenario seperti login atau splash screen, di mana halaman tersebut hanya ditampilkan satu kali, dan tidak ada alasan untuk kembali ke halaman sebelumnya.

Navigator.pushReplacement juga bisa digunakan dengan pengembalian nilai, serupa dengan Navigator.pop. Penggunaan metode ini membantu mengurangi jumlah halaman yang disimpan dalam stack, meningkatkan performa navigasi, terutama pada aplikasi dengan banyak halaman. Dengan cara ini, aplikasi bisa

memastikan bahwa hanya halaman yang relevan yang ada di stack, menghindari penumpukan halaman yang tidak perlu.

- Navigator Push dan Remove Until

  Navigator.pushAndRemoveUntil digunakan untuk mendorong (push) sebuah halaman baru ke dalam stack, sambil menghapus (remove) semua halaman sebelumnya sampai kondisi tertentu terpenuhi. Ini sangat berguna dalam skenario di mana pengguna harus dibawa ke halaman tertentu tanpa bisa kembali ke halaman sebelumnya, misalnya setelah login berhasil atau setelah proses onboarding selesai. Dengan Navigator.pushAndRemoveUntil, aplikasi dapat memastikan bahwa hanya halaman tertentu yang tersisa di stack.

  Metode ini memerlukan argumen berupa BuildContext, route untuk halaman baru, dan fungsi predikat (predicate) untuk menentukan halaman mana yang akan dihapus. Fungsi predikat ini mengembalikan nilai boolean, dan jika bernilai false, halaman dihapus dari stack. Ini memberikan kontrol penuh atas stack navigasi, terutama dalam kasus di mana ingin membuang semua halaman yang tidak lagi relevan dan menjaga satu halaman tertentu tetap ada.

- Named Routes

  Named Routes adalah metode navigasi yang lebih terstruktur di Flutter, di mana setiap halaman atau rute diberikan nama unik yang dapat digunakan untuk navigasi. Daripada menggunakan MaterialPageRoute atau CupertinoPageRoute secara eksplisit, navigasi dilakukan dengan memanggil Navigator.pushNamed atau Navigator.popAndPushNamed, yang menggunakan nama rute untuk menentukan halaman yang akan ditampilkan. Named Routes memudahkan pengelolaan navigasi dalam aplikasi dengan banyak halaman, karena rute didefinisikan dalam satu tempat, seperti di MaterialApp melalui properti routes.

  Named Routes juga mendukung pengiriman argumen antara halaman dengan cara yang lebih bersih. Argumen dapat dilewatkan menggunakan properti arguments, dan halaman tujuan bisa menerima data tersebut melalui ModalRoute.of(context).settings.arguments. Ini membuat navigasi lebih terorganisir dan modular, terutama dalam aplikasi yang besar, di mana jalur navigasi menjadi lebih kompleks dan perlu dikelola dengan baik.

1.2. Praktikum

1.2.1. Flutter Component Lanjutan

- Stack
  - Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const StackPositionPage(),
    );
  }
}

class StackPositionPage extends StatefulWidget {
  const StackPositionPage({super.key});

  @override
  State<StackPositionPage> createState() => _StackPositionedPageState();
}

class _StackPositionedPageState extends State<StackPositionPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Stack & Positioned',
          style: TextStyle(
            fontSize: 16.0,
            fontWeight: FontWeight.w600
          ),
        ),
      ),
      body: SafeArea(
```

```
      child: Stack(
        children: [
          Center(
            child: Container(
              width: 100.0,
              height: 100.0,
              color: Colors.blue,
            ),
          ),
          Positioned(
            left: 0.0,
            right: 0.0,
            top: 0.0,
            bottom: 0.0,
            child: Center(
              child: Container(
                width: 50.0,
                height: 50.0,
                color: Colors.red,
              ),
            ),
          ),
          Positioned(
            left: 0.0,
            top: 0.0,
            child: Center(
              child: Container(
                width: 50.0,
                height: 50.0,
                color: Colors.red,
              ),
            ),
          ),
          Positioned(
            right: 0.0,
            bottom: 0.0,
            child: Center(
              child: Container(
                width: 100.0,
                height: 100.0,
                color: Colors.orange,
              ),
            ),
          ),
          Positioned(
            right: 0.0,
            bottom: 0.0,
            child: Center(
```

```
            child: Container(
              width: 50.0,
              height: 50.0,
              color: Colors.indigo,
            ),
          ),
        ),
      ],
    )
  ),
 );
 }
}
```

- Penjelasan Kode

Kode di atas merupakan sebuah aplikasi Flutter sederhana yang menampilkan penggunaan widget Stack dan Positioned untuk menata beberapa elemen (widget) dalam sebuah tata letak yang saling tumpang tindih. Aplikasi ini dimulai dari fungsi main(), yang memanggil runApp() untuk menjalankan widget utama (MyApp). Widget MyApp merupakan subclass dari StatelessWidget, yang artinya widget ini tidak dapat berubah setelah dibangun. Di dalamnya terdapat method build() yang mengembalikan MaterialApp, yang berfungsi untuk mengelola tema dan halaman utama aplikasi, dalam hal ini halaman yang dibangun oleh widget StackPositionPage.

Dalam MaterialApp, aplikasi diberi judul "Quick Note" dan diberi tema menggunakan ThemeData yang dihasilkan dari ColorScheme.fromSeed dengan warna dasar Colors.deepPurple. Tema ini juga mengaktifkan desain Material 3, yang merupakan versi terbaru dari bahasa desain Material. Halaman utama yang ditampilkan adalah StackPositionPage, sebuah StatefulWidget, yang berarti tampilan ini dapat berubah sewaktu-waktu sesuai dengan keadaan. StatefulWidget ini menciptakan sebuah state baru melalui method createState(), yang mengembalikan instance dari class _StackPositionedPageState, yang berfungsi untuk mengelola logika tampilan.

Di dalam class _StackPositionedPageState, method build() bertanggung jawab untuk membangun tampilan halaman. Tampilan ini menggunakan Scaffold sebagai struktur dasar yang terdiri dari appBar di bagian atas dan body yang memuat konten utama. appBar menampilkan judul "Stack &

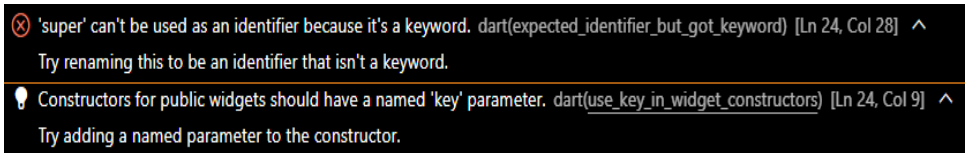
Positioned" dengan gaya teks tertentu, sedangkan body menampung sebuah Stack, yang merupakan widget khusus untuk menumpuk elemen-elemen di atas satu sama lain. Semua elemen di dalam Stack ini disusun menggunakan kombinasi widget Container dan Positioned.

Widget Stack berisi lima elemen Container yang ditempatkan pada berbagai posisi. Pertama, ada Container biru dengan ukuran 100x100 piksel yang diletakkan di tengah layar menggunakan Center. Selanjutnya, ada beberapa Container lainnya yang diatur dengan Positioned, yang mengatur posisinya relatif terhadap tepi layar. Misalnya, satu Container merah berukuran 50x50 piksel ditempatkan di tengah menggunakan pengaturan left, right, top, dan bottom yang disetel ke 0. Lalu, ada Container merah lainnya yang ditempatkan di sudut kiri atas. Di sudut kanan bawah terdapat dua Container, satu berwarna oranye dengan ukuran 100x100 piksel, dan satu lagi berwarna indigo dengan ukuran 50x50 piksel yang menumpuk di atas Container oranye.

Dengan demikian, aplikasi ini menunjukkan bagaimana elemen-elemen layout dapat ditempatkan secara fleksibel menggunakan Stack dan Positioned. Kombinasi ini memungkinkan elemen-elemen untuk saling tumpang tindih, menciptakan tampilan yang dinamis dan mudah diatur posisinya. Meskipun elemen-elemen ini diletakkan dalam posisi yang berbeda-beda, mereka tetap terlihat dalam satu layar dengan tumpukan yang teratur. Ini sangat berguna dalam berbagai kasus UI/UX yang memerlukan tumpukan elemen, seperti desain yang membutuhkan penempatan overlay atau elemen dekoratif yang dinamis.

➢ Hasil

➢ Error dan Solusi



Error pada kode di atas kemungkinan besar muncul karena penggunaan super yang tidak lengkap dalam constructor StackPositionPage. Constructor pada widget StackPositionPage seharusnya menggunakan super.key untuk menginisialisasi properti key, sebagaimana yang dilakukan pada class MyApp. Penggunaan super tanpa memberikan parameter yang benar pada constructor StackPositionPage. Di sini, seharusnya super.key dipanggil agar StatefulWidget mendapatkan key dengan benar. Tambahkan super.key ke dalam constructor StackPositionPage untuk menginisialisasi key. Dengan perbaikan ini, error yang muncul sebelumnya terkait dengan penggunaan

super tanpa parameter yang sesuai akan teratasi, dan aplikasi akan berjalan tanpa masalah.

- Padding
  - Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const PaddingPage(),
    );
  }
}

class PaddingPage extends StatefulWidget {
  const PaddingPage({super.key});

  @override
  State<PaddingPage> createState() => _PaddingPageState();
}

class _PaddingPageState extends State<PaddingPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Padding',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: SafeArea(
        child: Center(
```

```
          child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Material(
            color: Colors.blue.shade100,
            child: const Text('Widget Tanpa Padding'),
          ),
          const SizedBox(
            height: 16.0,
          ),
          Material(
            color: Colors.orange.shade100,
            child: const Padding(
              padding: EdgeInsets.all(16.0),
              child: Text(
                'Widget Dengan Padding',
              ),
            ),
          ),
          const SizedBox(
            height: 16.0,
          ),
          Material(
            color: Colors.pink.shade100,
            child: const Padding(
              padding: EdgeInsets.symmetric(horizontal: 32.0, vertical: 8.0),
              child: Text('Widget Dengan Padding'),
            ),
          ),
          const SizedBox(
            height: 16.0,
          ),
          Material(
            color: Colors.red.shade100,
            child: const Padding(
              padding: EdgeInsets.only(
                  left: 32.0, right: 16.0, top: 16.0, bottom: 8.0),
              child: Text('Widget Dengan Padding'),
            ),
          ),
        ],
      )),
    ),
  );
 }
}
```

- Penjelasan Kode

Kode di atas adalah contoh aplikasi Flutter yang memperlihatkan penggunaan widget Padding dengan berbagai variasi. Aplikasi dimulai dari fungsi main(), yang menjalankan aplikasi dengan memanggil runApp() untuk menginisialisasi widget MyApp. MyApp merupakan subclass dari StatelessWidget, yang mengatur tampilan utama aplikasi. Pada MyApp, digunakan widget MaterialApp untuk membungkus aplikasi dengan beberapa pengaturan seperti judul dan tema.

Dalam MaterialApp, judul aplikasi ditetapkan menjadi "Quick Note", dan tema diatur menggunakan ThemeData, yang menggunakan skema warna yang dihasilkan dari Colors.deepPurple. Fitur useMaterial3 diaktifkan agar aplikasi menggunakan gaya Material Design versi 3. Komponen utama yang ditampilkan dalam aplikasi ini adalah widget PaddingPage, yang merupakan halaman utama dari aplikasi.

PaddingPage sendiri merupakan subclass dari StatefulWidget, artinya halaman ini dapat berubah secara dinamis selama aplikasi berjalan. Pada class _PaddingPageState, method build() dipanggil untuk membangun antarmuka pengguna (UI). Struktur dasar halaman menggunakan widget Scaffold, yang menyediakan tata letak dasar dengan AppBar di bagian atas dan konten utama di dalam bagian body.

Di dalam AppBar, terdapat judul "Padding" dengan gaya teks yang memiliki ukuran 16 piksel dan font weight w600. Pada bagian body, digunakan widget SafeArea untuk memastikan tampilan tidak terganggu oleh elemen-elemen seperti notch atau status bar. Elemen-elemen dalam halaman ini ditempatkan di tengah layar menggunakan widget Center. Center membungkus widget Column, yang menata beberapa elemen UI secara vertikal.

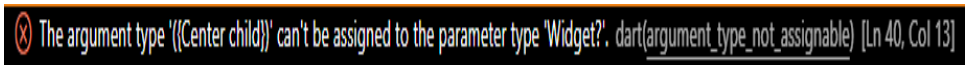
Tiap elemen UI adalah widget Material dengan warna latar belakang yang berbeda. Di dalam widget Material, ada teks yang menunjukkan contoh-contoh penggunaan padding pada setiap elemen. Contoh pertama adalah widget tanpa padding, yang memperlihatkan teks yang langsung menempel pada batas container. Contoh kedua menampilkan widget dengan padding seragam sebesar 16 piksel di semua sisi, yang menambah ruang di sekeliling teks. Contoh ketiga menampilkan widget dengan padding simetris, yaitu 32 piksel pada bagian horizontal dan 8 piksel pada bagian vertikal, yang

menciptakan lebih banyak ruang di sisi kanan dan kiri dibandingkan atas dan bawah. Contoh terakhir menunjukkan penggunaan padding yang diatur khusus untuk setiap sisi, dengan 32 piksel di sebelah kiri, 16 piksel di sebelah kanan, 16 piksel di atas, dan 8 piksel di bawah.

Untuk memisahkan setiap elemen Material, digunakan widget SizedBox dengan tinggi 16 piksel sebagai jarak vertikal antar elemen. Secara keseluruhan, aplikasi ini mendemonstrasikan berbagai cara penggunaan padding di Flutter untuk mengatur ruang di ser konten. Dengan menggunakan widget Padding, tata letak dan jarak antara konten dan batas container dapat diatur sesuai kebutuhan desain. Aplikasi ini juga menunjukkan fleksibilitas dalam mengatur padding yang berbeda di setiap sisi, sesuai dengan desain yang diinginkan.

- Hasil

➢ Error dan Solusi



Error pada kode ini terjadi karena kesalahan penulisan pada widget body di dalam Scaffold. Penulisan yang benar harus menggunakan tanda kurung kurawal {} untuk mendefinisikan body dari Scaffold, bukan tanda kurung biasa (). Perbaikan dilakukan pada bagian body dari Scaffold. Sebelumnya ditulis dengan tanda kurung biasa (), yang menyebabkan error, dan sudah diperbaiki dengan menggunakan tanda kurung kurawal {} yang benar. Selain itu, ditambahkan widget SafeArea pada body untuk memastikan konten tidak tertutup oleh elemen UI seperti status bar atau notch.

- Align

➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const AlignPage(),
    );
  }
}

class AlignPage extends StatefulWidget {
  const AlignPage({super.key});

  @override
  State<AlignPage> createState() => _AlignPageState();
}
```

```
class _AlignPageState extends State<AlignPage> {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text(
       'Align',
       style: TextStyle(
         fontSize: 16.0,
         fontWeight: FontWeight.w600
       ),
      ),
    ),
    body: SafeArea(
      child: Stack(
        children: [
          Padding(
            padding: const EdgeInsets.all(16.0),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Align(
                  alignment: Alignment.center,
                  child: Container(
                    width: 50.0,
                    height: 50.0,
                    color: Colors.blue,
                  ),
                ),
                const SizedBox(height: 16.0,),
                Align(
                  alignment: Alignment.centerLeft,
                  child: Container(
                    width: 50.0,
                    height: 50.0,
                    color: Colors.purple,
                  ),
                ),
                const SizedBox(height: 16.0,),
                Align(
                  alignment: Alignment.centerRight,
                  child: Container(
                    width: 50.0,
                    height: 50.0,
                    color: Colors.orange,
                  ),
                ),
              ],
```

```
            ),
          ),
          Align(
            alignment: Alignment.topLeft,
            child: Container(
              width: 50.0,
              height: 50.0,
              color: Colors.red,
            ),
          ),
          Align(
            alignment: Alignment.bottomRight,
            child: Container(
              width: 50.0,
              height: 50.0,
              color: Colors.pink,
            ),
          ),
        ],
      )
    ),
  );
 }
}
```

- Penjelasan Kode

Kode di atas adalah sebuah aplikasi Flutter sederhana yang mendemonstrasikan penggunaan widget `Align` untuk menempatkan beberapa kotak (container) di berbagai posisi dalam layout menggunakan konsep alignment. Aplikasi ini terdiri dari dua kelas utama: `MyApp` dan `AlignPage`, yang masing-masing memiliki peran yang spesifik dalam proses rendering antarmuka pengguna (UI).

Program dimulai dari fungsi `main()`, yang memanggil fungsi `runApp()`. Fungsi ini akan menjalankan aplikasi Flutter dan menampilkan widget yang di-passing, yaitu `MyApp`. Kelas `MyApp` merupakan turunan dari `StatelessWidget`, yang berarti widget ini tidak akan berubah setelah dibuat. Di dalam metode `build()`, widget `MaterialApp` dikembalikan untuk mengatur tema dan struktur dasar aplikasi, termasuk judul aplikasi dan halaman utama yang ditampilkan. `MaterialApp` ini menggunakan tema berbasis `ColorScheme` dengan warna dasar `Colors.deepPurple` serta fitur `useMaterial3` yang mengaktifkan gaya desain Material 3.

Selanjutnya, kelas `AlignPage` merupakan sebuah `StatefulWidget` yang berarti widget ini bisa mengalami perubahan saat aplikasi berjalan. Pada metode `createState()`, state yang terkait dengan widget ini diatur, yaitu `_AlignPageState`. Di dalam kelas `_AlignPageState`, metode `build()` digunakan untuk membuat antarmuka halaman yang akan ditampilkan. Di sini, widget `Scaffold` digunakan untuk menyediakan struktur halaman seperti `AppBar` (header) dan `body` (konten utama).

Di dalam `AppBar`, ada teks yang ditampilkan dengan label "Align", yang memiliki gaya khusus seperti ukuran font 16.0 dan ketebalan font `w600`. Pada bagian `body`, widget `SafeArea` digunakan untuk memastikan bahwa konten tidak tumpang tindih dengan area penting seperti status bar atau notch pada layar perangkat modern. Di dalam `SafeArea`, widget `Stack` digunakan untuk menumpuk beberapa widget `Align` di berbagai posisi layar.

Widget `Align` ini memiliki berbagai properti `alignment` yang menentukan di mana mereka akan ditempatkan. Sebagai contoh, ada tiga container di dalam widget `Column`, masing-masing ditempatkan menggunakan `Align` dengan alignment yang berbeda, yaitu `Alignment.center`, `Alignment.centerLeft`, dan `Alignment.centerRight`. Selain itu, dua `Align` lainnya ditempatkan di bagian atas kiri (topLeft) dan bawah kanan (bottomRight) dari layar menggunakan properti `alignment` pada `Align`. Setiap `Align` memiliki anak berupa `Container` dengan ukuran 50x50 piksel dan warna yang berbeda, seperti biru, merah, ungu, oranye, dan pink.

Secara keseluruhan, aplikasi ini menunjukkan bagaimana widget `Align` dapat digunakan untuk mengatur posisi elemen dalam layout, baik itu di dalam `Stack` maupun di dalam `Column`. Penempatan elemen dalam layar diatur dengan cukup fleksibel menggunakan properti alignment pada setiap widget `Align`.

➢ Hasil

➢ Error dan Solusi

Pada kode ini, terdapat kesalahan pada salah satu bagian Align. Kesalahan ini terjadi karena elemen yang digunakan tidak dituliskan dengan benar. Kesalahan tersebut terjadi karena elemen ini seharusnya adalah widget Align, namun dituliskan dalam format yang salah (menggunakan tanda kurung yang tidak sesuai). Untuk memperbaiki kesalahan ini, cukup ubah menjadi widget Align yang benar seperti yang lainnya. Dengan perbaikan tersebut, kode akan berfungsi dengan baik tanpa error, dan widget Align di bagian pertama akan dirender dengan benar di layar.

- Elevated Button

➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const ElevatedButtonPage(),
    );
  }
}

class ElevatedButtonPage extends StatefulWidget {
  const ElevatedButtonPage({super.key});

  @override
  State<ElevatedButtonPage> createState() =>
_ElevatedButtonPageState();
}

class _ElevatedButtonPageState extends State<ElevatedButtonPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Elevated Button',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: SafeArea(
        child: Center(
            child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
```

```
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ElevatedButton(
                onPressed: () => log('Button clicked'),
                style: ElevatedButton.styleFrom(
                    backgroundColor: Colors.indigo,
                    foregroundColor: Colors.white),
                child: const Text('Click Me')),
            const SizedBox(height: 16.0),
            ElevatedButton.icon(
                onPressed: () => log('Button clicked'),
                style: ElevatedButton.styleFrom(
                    backgroundColor: Colors.green,
                    foregroundColor: Colors.white),
                icon: const Icon(Icons.play_arrow_rounded),
                label: const Text('Click Me')),
            const SizedBox(
              height: 16.0,
            ),
            SizedBox(
              width: double.maxFinite,
              height: 48.0,
              child: ElevatedButton.icon(
                onPressed: () => log('Button clicked'),
                style: ElevatedButton.styleFrom(
                    backgroundColor: Colors.blue,
                    foregroundColor: Colors.white,
                    shape: RoundedRectangleBorder(
                        borderRadius: BorderRadius.circular(8.0))),
                icon: const Icon(Icons.play_arrow_rounded),
                label: const Text('Click Me'),
              ),
            ),
          ],
        ),
      )),
    ),
  );
 }
}
```

- Penjelasan Kode

Kode di atas adalah sebuah aplikasi Flutter sederhana yang mendemonstrasikan penggunaan widget `ElevatedButton` dalam berbagai bentuk untuk mengimplementasikan tombol dengan gaya yang disesuaikan. Aplikasi ini terdiri dari dua kelas utama: `MyApp` dan `ElevatedButtonPage`.

Fungsionalitas tombol-tombol tersebut ditambahkan dengan menggunakan fungsi `log()` untuk mencatat aktivitas saat tombol diklik.

Aplikasi dimulai dari fungsi `main()`, yang menjalankan aplikasi Flutter dengan memanggil fungsi `runApp()` dan menginisialisasi widget root `MyApp`. `MyApp` adalah turunan dari `StatelessWidget`, yang berarti bahwa widget ini bersifat statis, tidak akan berubah setelah dibuat. Pada metode `build()` dalam `MyApp`, widget `MaterialApp` digunakan sebagai pembungkus utama aplikasi, yang mendefinisikan judul, tema, dan halaman utama aplikasi. Tema menggunakan skema warna berbasis `Colors.deepPurple` dan diatur untuk menggunakan gaya Material Design 3 dengan properti `useMaterial3: true`.

Halaman utama aplikasi diatur oleh widget `ElevatedButtonPage`, yang merupakan sebuah `StatefulWidget`. Ini berarti bahwa widget ini dapat diubah atau di-render ulang jika ada perubahan state selama runtime. Dalam metode `createState()`, state widget diatur oleh kelas `_ElevatedButtonPageState`, yang mengandung logika untuk menampilkan dan mengatur tampilan tombol.

Pada metode `build()` dalam `_ElevatedButtonPageState`, widget `Scaffold` digunakan untuk membuat struktur dasar halaman, termasuk `AppBar` di bagian atas dengan judul "Elevated Button". Bagian `body` menggunakan widget `SafeArea` untuk mencegah tumpang tindih dengan area penting perangkat, dan semua konten berada dalam widget `Center` yang memastikan elemen-elemen berada di tengah layar. Di dalam `Center`, terdapat `Padding` dengan `Column` untuk menempatkan tombol-tombol secara vertikal.

Kolom tersebut berisi tiga tombol `ElevatedButton` yang memiliki gaya dan bentuk yang berbeda-beda. Tombol pertama adalah `ElevatedButton` sederhana yang memiliki properti `onPressed` untuk mengeksekusi fungsi ketika tombol diklik. Fungsi ini memanggil `log('Button clicked')`, yang akan mencatat log ke konsol bahwa tombol telah diklik. Tombol ini juga menggunakan gaya dari `ElevatedButton.styleFrom()` untuk mengatur warna latar belakang menjadi `Colors.indigo` dan warna teks menjadi putih.

Tombol kedua adalah `ElevatedButton.icon`, yang merupakan variasi dari `ElevatedButton` dengan ikon dan label teks. Tombol ini menampilkan ikon berupa `Icons.play_arrow_rounded` di samping teks "Click Me". Sama

seperti tombol pertama, `onPressed` dari tombol ini juga memanggil fungsi `log` untuk mencatat aktivitas klik tombol.

Tombol ketiga adalah versi lain dari `ElevatedButton.icon`, tetapi kali ini tombol tersebut dibuat memenuhi lebar layar dengan menggunakan widget `SizedBox` yang mengatur lebar maksimum (`double.maxFinite`) dan tinggi tombol menjadi 48 piksel. Tombol ini juga memiliki sudut membulat menggunakan properti `shape: RoundedRectangleBorder` pada gaya tombol untuk mengatur radius sudut. Sama seperti tombol sebelumnya, ini juga memiliki ikon dan label teks, dan log dicatat setiap kali tombol diklik.

Secara keseluruhan, aplikasi ini menunjukkan bagaimana memanfaatkan widget `ElevatedButton` dalam berbagai bentuk, mengatur gaya tampilan, dan menambahkan logika saat tombol ditekan menggunakan fungsi `log()` untuk mencatat aksi pengguna ke konsol.

- Hasil

- Text Field
  - Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const TextFieldPage(),
    );
  }
}

class TextFieldPage extends StatefulWidget {
  const TextFieldPage({super.key});

  @override
  State<TextFieldPage> createState() => _TextFieldPageState();
}

class _TextFieldPageState extends State<TextFieldPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'TextField',
          style: TextStyle(
            fontSize: 16.0,
            fontWeight: FontWeight.w600
          ),
        ),
      ),
      body: SafeArea(
        child: Center(
```

```
        child: Padding(
          padding: const EdgeInsets.symmetric(
            horizontal: 16.0
          ),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              const TextField(
                decoration: InputDecoration(
                  labelText: 'Full Name',
                  hintText: 'Enter full name',
                ),
                maxLength: 50,
              ),
              const SizedBox(height: 16.0,),
              const TextField(
                decoration: InputDecoration(
                  labelText: 'Phone Number',
                  hintText: 'Enter phone number',
                  filled: true
                ),
                keyboardType: TextInputType.number,
                maxLength: 13,
              ),
              const TextField(
                decoration: InputDecoration(
                  labelText: 'Email',
                  hintText: 'Enter email address',
                  border: OutlineInputBorder(),
                  prefixIcon: Icon(
                    Icons.email_rounded
                  ),
                ),
                keyboardType: TextInputType.emailAddress,
                maxLength: 50,
              ),
              const SizedBox(height: 16.0,),
              TextField(
                obscureText: true,
                decoration: InputDecoration(
                  labelText: 'Password',
                  hintText: 'Enter Password',
                  border: const OutlineInputBorder(),
                  prefixIcon: const Icon(
                    Icons.password_rounded
                  ),
                  suffixIcon: IconButton(
                    icon: const Icon(
```

```
                    Icons.visibility_off_rounded
                  ),
                  onPressed: () => log('Update password visibility'),
                )
              ),
              maxLength: 20,
            ),
          ],
        ),
      )
    ),
  ),
 );
 }
}
```

- Penjelasan Kode

Kode di atas merupakan aplikasi Flutter yang menampilkan form input dengan beberapa field yang menggunakan widget `TextField`. Aplikasi ini terdiri dari dua bagian utama: kelas `MyApp`, yang merupakan titik masuk utama aplikasi, dan kelas `TextFieldPage`, yang mengatur tampilan input yang digunakan untuk memasukkan data seperti nama lengkap, nomor telepon, email, dan password.

Aplikasi dimulai dengan memanggil fungsi `main()`, yang menjalankan aplikasi dengan menggunakan `runApp()`, dan menginisialisasi widget `MyApp`. Kelas `MyApp` adalah turunan dari `StatelessWidget`, sehingga ia tidak menyimpan state apapun dan hanya merender konten secara statis. Di dalam metode `build()` milik `MyApp`, widget `MaterialApp` digunakan sebagai pembungkus utama aplikasi. `MaterialApp` menyediakan pengaturan tema dengan warna dasar `Colors.deepPurple` dan menetapkan halaman awal aplikasi pada widget `TextFieldPage`.

`TextFieldPage` adalah turunan dari `StatefulWidget`, yang memungkinkan state dari widget ini untuk diubah selama runtime. Namun, dalam implementasi ini, tidak ada perubahan state yang nyata. Pada metode `build()` di kelas `_TextFieldPageState`, struktur antarmuka dibuat menggunakan widget `Scaffold`, yang berfungsi sebagai wadah untuk menyusun elemen-elemen seperti `AppBar` di bagian atas dengan judul "TextField" dan `body` yang menampung form input. Di dalam body, `SafeArea` digunakan untuk memastikan konten tidak bertabrakan dengan

area penting perangkat, seperti takik (notch) atau status bar. Elemen-elemen form ditempatkan di dalam widget `Center`, yang membuat konten berada di tengah layar.

Bagian inti dari halaman ini adalah beberapa `TextField`, yang disusun secara vertikal menggunakan `Column` dan diberi padding horizontal untuk memberi jarak dari tepi layar. Masing-masing `TextField` memiliki dekorasi yang spesifik sesuai dengan fungsinya. TextField pertama digunakan untuk input nama lengkap, dengan label "Full Name" dan hint "Enter full name", serta batas maksimum 50 karakter. TextField kedua digunakan untuk menerima input nomor telepon, dengan label "Phone Number" dan hint "Enter phone number". TextField ini juga menampilkan keypad numerik dengan menggunakan `keyboardType: TextInputType.number`, serta memiliki batas panjang karakter 13.

Berikutnya adalah TextField untuk email. Input ini menggunakan ikon email di sebelah kiri dan memiliki border dengan `OutlineInputBorder`, membuatnya lebih menonjol secara visual. Keyboard yang disediakan juga diatur untuk input email melalui `keyboardType: TextInputType.emailAddress`. Terakhir, ada TextField untuk password, yang menggunakan properti `obscureText: true` untuk menyembunyikan input dengan simbol bintang atau titik. Selain itu, terdapat ikon kunci di sebelah kiri dan tombol ikon mata di sebelah kanan untuk mengontrol visibilitas password. Ketika tombol mata diklik, fungsi `log()` mencatat aksi tersebut di konsol, walaupun visibilitas password tidak benar-benar berubah karena fungsinya hanya mencatat log.

Secara keseluruhan, aplikasi ini menunjukkan cara penggunaan `TextField` dalam aplikasi Flutter untuk mengumpulkan berbagai jenis input dari pengguna, lengkap dengan dekorasi seperti label, hint, batasan karakter, dan ikon yang relevan untuk setiap jenis input. Widget `TextField` disesuaikan untuk berbagai keperluan seperti nama, nomor telepon, email, dan password, serta diatur dengan cara yang responsif dan mudah digunakan.

➢ Hasil

- Image
  - ❖ Internet
    - ➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
```

```
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const ImageNetworkPage(),
    );
  }
}

class ImageNetworkPage extends StatefulWidget {
  const ImageNetworkPage({super.key});

  @override
  State<ImageNetworkPage> createState() =>
_ImageNetworkPageState();
}

class _ImageNetworkPageState extends State<ImageNetworkPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Image Network',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: SafeArea(
        child: Center(
            child: ClipRRect(
          borderRadius: BorderRadius.circular(8.0),
          child: Image.network(
            'https://siska.fppti.or.id/storage/foto/8LGeefnhw6FrzpKIPORN
8RjdZgmSoLJ5djFcoUXA.png',
            width: 300.0,
            height: 300.0,
            fit: BoxFit.cover,
            filterQuality: FilterQuality.medium,
          ),
        )),
      ),
    );
  }
}
```

- Penjelasan Kode

Kode di atas adalah aplikasi Flutter yang menampilkan gambar dari jaringan (URL) menggunakan widget `Image.network`. Aplikasi ini terdiri dari dua kelas utama: `MyApp` dan `ImageNetworkPage`, di mana fungsi utama aplikasi adalah untuk mengambil dan menampilkan gambar dari URL yang telah ditentukan di dalam aplikasi.

Program dimulai dengan fungsi `main()`, yang memanggil fungsi `runApp()` untuk menjalankan aplikasi. Di dalam `runApp()`, widget utama yang digunakan adalah `MyApp`, yang merupakan turunan dari `StatelessWidget`. `StatelessWidget` ini digunakan karena aplikasi tidak memerlukan perubahan state. Pada metode `build()` dari `MyApp`, aplikasi diinisialisasi dengan widget `MaterialApp`, yang digunakan untuk memberikan tema dan struktur dasar aplikasi Flutter berbasis Material Design. Tema yang digunakan mengadopsi skema warna yang dihasilkan dari `Colors.deepPurple` dengan Material 3. Halaman utama yang ditampilkan adalah `ImageNetworkPage`, yang merupakan widget untuk menampilkan gambar dari jaringan.

Kelas `ImageNetworkPage` adalah turunan dari `StatefulWidget`, meskipun pada aplikasi ini, state-nya tidak diubah selama runtime. Namun, penggunaan `StatefulWidget` memungkinkan pengembang untuk memperluas fungsionalitasnya di masa mendatang jika diperlukan. Kelas `_ImageNetworkPageState` mengimplementasikan metode `build()` yang bertanggung jawab untuk merender halaman. Struktur antarmuka dibangun menggunakan widget `Scaffold`, yang menyediakan kerangka dasar untuk halaman, termasuk elemen seperti `AppBar` dan `body`. `AppBar` menampilkan judul "Image Network" dengan teks yang diatur menggunakan `TextStyle` untuk ukuran dan ketebalan font.

Bagian `body` dari halaman ditempatkan di dalam `SafeArea`, yang memastikan bahwa konten tidak bertabrakan dengan elemen penting pada layar seperti status bar atau notch. Konten ditampilkan di dalam widget `Center` agar gambar berada di tengah layar. Di dalam widget `Center`, gambar dibungkus oleh `ClipRRect`, yang digunakan untuk memberikan efek sudut melengkung pada gambar melalui properti `borderRadius` yang diatur dengan radius 8.0 piksel.

Gambar yang ditampilkan diambil dari jaringan menggunakan widget `Image.network`. URL gambar ditentukan langsung di dalam properti `Image.network`. Properti `width` dan `height` diatur menjadi 300.0 untuk menentukan ukuran gambar yang ditampilkan di layar. Properti `fit` disetel ke `BoxFit.cover`, yang memastikan gambar memenuhi kotak yang disediakan tanpa distorsi, dengan memperbesar atau memperkecil gambar agar sesuai dengan ukuran widget. Selain itu, `filterQuality` diatur ke `FilterQuality.medium` untuk meningkatkan kualitas rendering gambar pada perangkat dengan layar beresolusi tinggi.

Secara keseluruhan, aplikasi ini adalah contoh sederhana yang menunjukkan cara menampilkan gambar dari internet di dalam aplikasi Flutter. Widget `Image.network` mengambil gambar dari URL yang diberikan, dan gambar tersebut kemudian ditampilkan di dalam layar dengan ukuran dan kualitas yang disesuaikan. Dengan penggunaan `ClipRRect`, gambar juga ditampilkan dengan sudut melengkung, memberikan sentuhan visual yang lebih estetis.

- Hasil

❖ Aset Lokal

➢ Kode Pemrograman

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

void main() {
 runApp(const MyApp());
}

class MyApp extends StatelessWidget {
 const MyApp({super.key});

 @override
 Widget build(BuildContext context) {
  return MaterialApp(
   title: 'Quick Note',
   theme: ThemeData(
    colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
    useMaterial3: true,
   ),
   home: const ImageAssetPage(),
  );
 }
}

class ImageAssetPage extends StatefulWidget {
 const ImageAssetPage({super.key});

 @override
 State<ImageAssetPage> createState() => _ImageAssetPageState();
}

class _ImageAssetPageState extends State<ImageAssetPage> {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text(
     'Image Asset',
     style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
    ),
   ),
   body: SafeArea(
    child: Center(
      child: ClipRRect(
     borderRadius: BorderRadius.circular(8.0),
     child: Image.asset(
      'assets/images/kucing.jpeg',
```

```
          width: 300.0,
          height: 300.0,
          fit: BoxFit.cover,
          filterQuality: FilterQuality.medium,
        ),
      )),
    ),
  );
 }
}
```

- Penjelasan Kode

Kode di atas adalah sebuah aplikasi Flutter yang menampilkan gambar lokal (gambar yang disimpan di dalam direktori aplikasi) menggunakan widget `Image.asset`. Aplikasi ini terdiri dari dua komponen utama, yaitu kelas `MyApp` dan `ImageAssetPage`, di mana aplikasi ini mengatur bagaimana gambar lokal ditampilkan di layar dengan berbagai properti tambahan untuk meningkatkan tampilan gambar.

Aplikasi dimulai dengan fungsi `main()`, yang memanggil `runApp()` untuk menjalankan aplikasi. `runApp()` menerima sebuah instance dari widget `MyApp`, yang merupakan turunan dari `StatelessWidget`. Kelas `MyApp` ini tidak memiliki perubahan state, sehingga cocok menggunakan `StatelessWidget`. Pada metode `build()`, widget `MaterialApp` digunakan untuk menyusun struktur dasar aplikasi yang berbasis Material Design. Aplikasi juga menerapkan tema menggunakan skema warna yang dihasilkan dari warna dasar `Colors.deepPurple`. Pada bagian `home`, aplikasi mengarahkan pengguna ke halaman utama yang ditentukan oleh widget `ImageAssetPage`.

Kelas `ImageAssetPage` merupakan turunan dari `StatefulWidget`. Meskipun saat ini tidak ada state yang berubah dalam aplikasi ini, penggunaan `StatefulWidget` memberi fleksibilitas jika state perlu diubah di masa mendatang. Pada kelas `_ImageAssetPageState`, metode `build()` digunakan untuk merender tampilan antarmuka. Struktur utama dari halaman ini dibangun menggunakan widget `Scaffold`, yang menyediakan elemen kerangka aplikasi seperti `AppBar` dan `body`. `AppBar` menampilkan judul "Image Asset" dengan gaya teks yang diatur menggunakan `TextStyle` agar terlihat lebih tebal dan berukuran sedang.

Bagian utama konten aplikasi berada di dalam `body`, yang dibungkus oleh widget `SafeArea` untuk memastikan bahwa konten tidak terpotong oleh bagian-bagian penting perangkat seperti status bar atau notch. Di dalam `SafeArea`, gambar ditempatkan di tengah layar menggunakan widget `Center`. Di dalam widget `Center`, gambar dibungkus dengan widget `ClipRRect`, yang digunakan untuk memberikan efek sudut melengkung pada gambar. Properti `borderRadius` disetel dengan radius 8.0 untuk memberikan kesan melengkung pada sudut gambar.

Gambar yang ditampilkan berasal dari file lokal, yang diambil menggunakan widget `Image.asset`. Jalur file yang digunakan adalah `'assets/images/kucing.jpeg'`, yang menunjukkan bahwa gambar ini harus ditempatkan di dalam folder `assets/images` di proyek Flutter. Ukuran gambar diatur menjadi 300x300 piksel menggunakan properti `width` dan `height`. Properti `fit` diatur ke `BoxFit.cover`, yang memastikan gambar mengisi seluruh ruang yang disediakan tanpa mengubah aspek rasio gambar, memotong bagian gambar yang tidak sesuai dengan ukuran yang ditentukan. Selain itu, `filterQuality` diatur ke `FilterQuality.medium` untuk memastikan kualitas gambar tetap tajam saat ditampilkan di layar.

Secara keseluruhan, aplikasi ini menunjukkan bagaimana cara menampilkan gambar lokal yang ada di dalam proyek Flutter menggunakan `Image.asset`. Dengan bantuan `ClipRRect`, gambar diberikan efek sudut melengkung, memberikan tampilan yang lebih estetis. Aplikasi ini mendemonstrasikan cara mengakses dan menampilkan gambar lokal di dalam aplikasi dengan kontrol penuh terhadap ukuran, kualitas, dan gaya tampilan gambar.

➢ Hasil

- Container

➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
```

```
      ),
      home: const ContainerPage(),
    );
  }
}

class ContainerPage extends StatefulWidget {
  const ContainerPage({super.key});

  @override
  State<ContainerPage> createState() => _ContainerPageState();
}

class _ContainerPageState extends State<ContainerPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Container',
          style: TextStyle(
            fontSize: 16.0,
            fontWeight: FontWeight.w600
          ),
        ),
      ),
      body: SafeArea(
        child: Center(
          child: Container(
            width: 200.0,
            height: 200.0,
            padding: const EdgeInsets.all(16.0),
            margin: const EdgeInsets.all(16.0),
            decoration: BoxDecoration(
              color: Colors.indigo,
              borderRadius: BorderRadius.circular(8.0),
              boxShadow: const [
                BoxShadow(
                  color: Colors.black26,
                  blurRadius: 12.0,
                  spreadRadius: 4.0,
                  offset: Offset(0.0, 4.0)
                )
              ]
            ),
            child: const Center(
              child: Text(
                'Lorem ipsum dolor sit amet, consectur adipiscing elit',
```

```
            style: TextStyle(
              fontSize: 20.0,
              color: Colors.white,
              fontWeight: FontWeight.w600,
            ),
            textAlign: TextAlign.center,
          ),
        ),
      ),
    )
  ),
 );
 }
}
```

- Penjelasan Kode

Kode di atas adalah aplikasi Flutter sederhana yang menggunakan widget `Container` untuk menampilkan teks di dalam kotak dengan beberapa properti dekoratif, seperti warna latar, bayangan, dan sudut melengkung. Aplikasi ini terdiri dari dua komponen utama: `MyApp` dan `ContainerPage`, di mana `MyApp` berfungsi sebagai titik masuk aplikasi, dan `ContainerPage` adalah halaman utama yang menampilkan konten dengan menggunakan widget `Container`.

Aplikasi dimulai dengan fungsi `main()`, yang memanggil `runApp()` untuk menjalankan aplikasi. `runApp()` akan menjalankan aplikasi menggunakan widget `MyApp`, yang merupakan turunan dari `StatelessWidget`. Kelas `MyApp` ini tidak membutuhkan perubahan state, sehingga menggunakan `StatelessWidget` adalah pilihan yang tepat. Pada metode `build()` dari `MyApp`, aplikasi diinisialisasi menggunakan widget `MaterialApp`, yang menyediakan struktur dasar dan tema berbasis Material Design. Tema aplikasi mengadopsi skema warna dari `Colors.deepPurple` dengan menggunakan Material Design versi 3. Bagian `home` aplikasi menampilkan widget `ContainerPage` sebagai halaman utama.

Kelas `ContainerPage` adalah turunan dari `StatefulWidget`, yang memungkinkan modifikasi state di masa depan jika diperlukan. Namun, pada saat ini, state tidak berubah, sehingga fokusnya adalah pada tampilan antarmuka. Di dalam metode `build()` dari kelas `_ContainerPageState`, struktur halaman utama dibangun menggunakan widget `Scaffold`, yang menyediakan elemen-elemen dasar seperti `AppBar` dan `body`. Pada bagian

`AppBar`, sebuah judul "Container" ditampilkan dengan gaya teks yang tebal dan berukuran sedang menggunakan widget `Text`.

Bagian utama dari konten berada di dalam widget `SafeArea`, yang memastikan bahwa konten tidak bertabrakan dengan elemen sistem perangkat seperti status bar atau notch. Di dalam `SafeArea`, widget `Center` digunakan untuk menempatkan konten di tengah layar. Di dalam widget `Center`, terdapat `Container` yang menjadi fokus utama halaman ini. `Container` digunakan untuk membuat sebuah kotak dengan ukuran, dekorasi, dan tata letak yang ditentukan.

`Container` memiliki properti `width` dan `height` yang masing-masing diatur menjadi 200.0 piksel, yang menentukan ukuran kotak di layar. Properti `padding` dan `margin` masing-masing diatur menggunakan `EdgeInsets.all(16.0)`, yang memberikan jarak dalam (padding) dan luar (margin) sebesar 16 piksel di semua sisi. Dekorasi `Container` ditentukan melalui properti `decoration`, yang menggunakan `BoxDecoration`. `BoxDecoration` ini mengatur warna latar belakang `Container` menjadi `Colors.indigo`, serta memberikan efek sudut melengkung dengan radius 8 piksel melalui properti `borderRadius`. Selain itu, properti `boxShadow` diatur untuk memberikan bayangan pada kotak, dengan warna bayangan `Colors.black26`, radius blur sebesar 12.0, dan offset vertikal sebesar 4.0, yang memberikan kesan bahwa kotak terangkat dari layar.

Di dalam `Container`, terdapat widget `Center` yang menampung widget `Text`. Widget `Text` ini menampilkan teks "Lorem ipsum dolor sit amet, consectur adipiscing elit" dengan gaya teks yang diatur menggunakan properti `TextStyle`. Ukuran teks ditentukan sebesar 20.0, warna teks diatur menjadi putih (`Colors.white`), dan teks ditampilkan dengan ketebalan font `FontWeight.w600`. Selain itu, teks diatur untuk berada di tengah-tengah `Container` secara horizontal menggunakan properti `textAlign: TextAlign.center`.

Secara keseluruhan, aplikasi ini menunjukkan bagaimana cara menggunakan widget `Container` di Flutter untuk menampilkan konten dengan dekorasi khusus. Melalui penggunaan properti seperti padding, margin, border radius, dan bayangan, `Container` dapat digunakan untuk

membuat tata letak yang lebih menarik secara visual. Gaya dan posisi teks di dalam `Container` juga diatur dengan baik agar mudah terbaca dan estetis.

➢ Hasil

- Icon

➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
```

```
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const IconPage(),
    );
  }
}

class IconPage extends StatefulWidget {
  const IconPage({super.key});

  @override
  State<IconPage> createState() => _IconPageState();
}

class _IconPageState extends State<IconPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Icon',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: const SafeArea(
          child: Center(
        child: Column(mainAxisAlignment: MainAxisAlignment.center,
children: [
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Icon(
                Icons.thumb_up_rounded,
              ),
              SizedBox(
                width: 16.0,
              ),
              Icon(
                Icons.thumb_down_rounded,
              ),
            ],
          ),
          SizedBox(
            height: 16.0,
```

```
        ),
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(
              Icons.thumb_up_rounded,
              color: Colors.blue,
              size: 32.0,
            ),
            SizedBox(
              width: 16.0,
            ),
            Icon(
              Icons.thumb_down_rounded,
              color: Colors.red,
              size: 32.0,
            ),
          ],
        ),
      ],),
    )),
  );
 }
}
```

- Penjelasan Kode

Kode di atas merupakan aplikasi Flutter sederhana yang menampilkan beberapa ikon menggunakan widget `Icon`. Aplikasi ini terdiri dari dua komponen utama, yaitu `MyApp` dan `IconPage`, di mana `MyApp` berfungsi sebagai entri utama aplikasi, sedangkan `IconPage` adalah halaman utama yang berisi antarmuka ikon yang ditampilkan pada layar.

Fungsi `main()` memanggil `runApp()` untuk menjalankan aplikasi. `runApp()` menerima `MyApp`, yang merupakan turunan dari `StatelessWidget`. Karena aplikasi tidak memerlukan perubahan data secara dinamis, penggunaan `StatelessWidget` sangat sesuai. Di dalam metode `build()` dari `MyApp`, widget `MaterialApp` digunakan untuk membungkus aplikasi dan mengatur tema Material Design. Tema aplikasi diatur dengan warna dasar `Colors.deepPurple` yang dihasilkan dari metode `ColorScheme.fromSeed()`. Aplikasi ini menggunakan `useMaterial3: true` untuk mengaktifkan fitur Material Design versi 3, dan halaman utama (`home`) aplikasi diatur untuk menampilkan widget `IconPage`.

Kelas `IconPage` adalah turunan dari `StatefulWidget`, meskipun dalam implementasinya saat ini, state tidak diubah setelah widget dibangun. Kelas `_IconPageState` merupakan state dari `IconPage`, dan di dalam metode `build()`, struktur halaman aplikasi dibangun dengan menggunakan widget `Scaffold`. Widget `Scaffold` menyediakan struktur dasar untuk tampilan layar, termasuk `AppBar` dan `body`. Pada `AppBar`, judul "Icon" ditampilkan menggunakan widget `Text` dengan ukuran font sedang dan tebal.

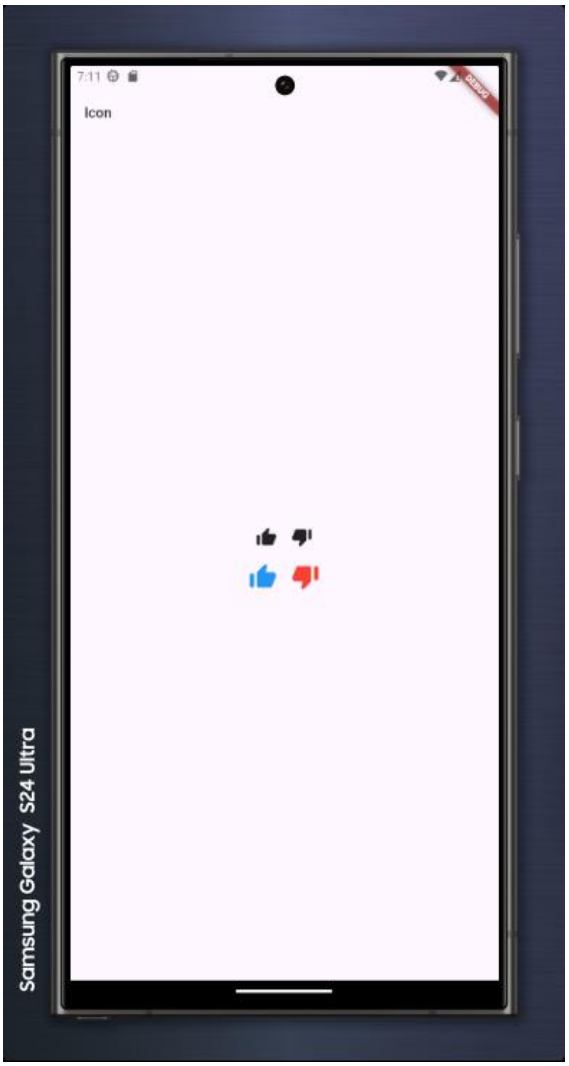
Bagian utama dari halaman berada di dalam widget `SafeArea` untuk memastikan konten tidak bertabrakan dengan elemen sistem seperti status bar. Di dalam `SafeArea`, widget `Center` digunakan untuk menempatkan seluruh konten di tengah layar. Selanjutnya, widget `Column` digunakan untuk menampilkan dua deretan ikon yang ditempatkan di tengah secara vertikal.

Pada baris pertama, dua ikon ditampilkan di dalam widget `Row`, yang menata ikon secara horizontal. Ikon yang ditampilkan adalah ikon jempol ke atas (`Icons.thumb_up_rounded`) dan jempol ke bawah (`Icons.thumb_down_rounded`), yang diletakkan dengan jarak 16 piksel di antara mereka menggunakan widget `SizedBox`. Kedua ikon ini menggunakan pengaturan default, tanpa modifikasi warna atau ukuran.

Pada baris kedua, ikon yang sama ditampilkan, tetapi dengan modifikasi. Ikon jempol ke atas diberi warna `Colors.blue` dan ukuran diperbesar menjadi 32 piksel, sementara ikon jempol ke bawah diberi warna `Colors.red` dengan ukuran yang sama, yaitu 32 piksel. Kedua ikon diatur dengan jarak yang sama seperti pada baris pertama, yaitu 16 piksel menggunakan `SizedBox`.

Secara keseluruhan, aplikasi ini menunjukkan bagaimana cara menampilkan dan menata ikon di Flutter menggunakan widget `Icon`. Dengan mengkombinasikan widget `Row` dan `Column`, tata letak yang sederhana tetapi efektif dapat dibuat untuk menampilkan ikon dengan modifikasi warna dan ukuran.

➢ Hasil



1.2.2. ListView Widget

- List View

➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
```

```
      colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
      useMaterial3: true,
    ),
    home: const ListviewPage(),
   );
  }
}

class ListviewPage extends StatefulWidget {
 const ListviewPage({super.key});

 @override
 State<ListviewPage> createState() => _ListViewPageState();
}

class _ListViewPageState extends State<ListviewPage> {

 bool switchEnable = true;

 void onChangeSwitch({required bool value}) {
  setState(() {
   switchEnable = value;
  });
 }

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text(
     'ListView',
     style: TextStyle(
      fontSize: 16.0,
      fontWeight: FontWeight.w600
     ),
    ),
   ),
   body: SafeArea(
    child: ListView(
     shrinkWrap: true,
     physics: const BouncingScrollPhysics(),
     padding: EdgeInsets.zero,
     children: [
      ListTile(
       leading: const Icon(
        Icons.airplanemode_active_rounded,
       ),
```

```
            title: const Text(
              'Airplane Mode',
              style: TextStyle(
                fontSize: 16.0
              ),
            ),
            trailing: Transform.translate(
              offset: const Offset(10.0, 0.0),
              child: Switch(
                value: switchEnable,
                onChanged: (value) => onChangeSwitch(value: value),
              ),
            ),
          ),
          const ListTile(
            leading: Icon(
              Icons.wifi_rounded,
            ),
            title: Text(
              'Wi-Fi',
              style: TextStyle(
                fontSize: 16.0
              ),
            ),
            trailing: Icon(
              Icons.chevron_right_rounded,
            ),
          ),
          const ListTile(
            leading: Icon(
              Icons.bluetooth_rounded,
            ),
            title: Text(
              'Bluetooth',
              style: TextStyle(
                fontSize: 16.0
              ),
            ),
            trailing: Icon(
              Icons.chevron_right_outlined
            ),
          ),
        ],
      )
    ),
  );
 }
}
```

- Penjelasan Kode

Kode di atas adalah implementasi aplikasi Flutter yang menggunakan `ListView` untuk menampilkan daftar opsi pengaturan sederhana, seperti "Airplane Mode", "Wi-Fi", dan "Bluetooth". Aplikasi ini memiliki dua komponen utama, yaitu `MyApp` dan `ListviewPage`. Komponen `MyApp` berfungsi sebagai entry point dari aplikasi, sedangkan `ListviewPage` adalah halaman utama yang menampilkan daftar pengaturan.

Ketika aplikasi dijalankan, fungsi `main()` akan memanggil `runApp()` dan menjalankan aplikasi Flutter. Aplikasi ini dibungkus dalam widget `MyApp`, yang merupakan turunan dari `StatelessWidget`. Pada widget ini, sebuah `MaterialApp` digunakan untuk menyediakan fitur dasar seperti tema dan pengaturan navigasi. Tema aplikasi menggunakan `ColorScheme.fromSeed` untuk menghasilkan warna tema berbasis warna `deepPurple`. Halaman utama yang ditampilkan oleh aplikasi adalah `ListviewPage`.

`ListviewPage` adalah widget Stateful yang memungkinkan perubahan state saat interaksi terjadi, misalnya ketika pengguna mengaktifkan atau menonaktifkan `Switch` pada "Airplane Mode". Variabel boolean `switchEnable` digunakan untuk menyimpan status aktif atau tidaknya "Airplane Mode", dan metode `onChangeSwitch()` digunakan untuk memperbarui nilai ini saat pengguna mengubah status `Switch`. Fungsi ini menggunakan `setState()` untuk memberitahukan Flutter agar merender ulang UI ketika ada perubahan pada state.

Pada metode `build()` dari `ListviewPage`, widget `Scaffold` digunakan untuk membangun struktur halaman yang terdiri dari `AppBar` dan `body`. Pada `AppBar`, teks "ListView" ditampilkan sebagai judul dengan menggunakan widget `Text` yang dipersonalisasi dengan ukuran dan gaya teks. Konten utama aplikasi ditempatkan di dalam `body`, yang berisi sebuah `ListView`. `ListView` adalah widget yang digunakan untuk menampilkan daftar item secara vertikal yang dapat di-scroll.

Setiap item dalam daftar diwakili oleh widget `ListTile`, yang memiliki beberapa komponen, seperti ikon di bagian kiri (`leading`), teks utama di tengah (`title`), dan elemen di bagian kanan (`trailing`). Item pertama dalam daftar adalah pengaturan "Airplane Mode", yang memiliki ikon pesawat

terbang di sebelah kiri, teks "Airplane Mode" di tengah, dan sebuah `Switch` di sebelah kanan. Status `Switch` dikontrol oleh variabel `switchEnable`, dan setiap kali pengguna mengubah status `Switch`, metode `onChangeSwitch` dipanggil untuk memperbarui state.

Dua item berikutnya adalah pengaturan "Wi-Fi" dan "Bluetooth", yang masing-masing memiliki ikon di sebelah kiri, teks di tengah, dan ikon `chevron_right_rounded` di sebelah kanan yang menunjukkan bahwa item tersebut dapat diklik untuk melihat lebih lanjut (meskipun dalam kode ini, tidak ada aksi tambahan yang ditentukan untuk ikon tersebut). Penggunaan `ListView` dengan `ListTile` membuat daftar ini mudah dibaca dan dinavigasi, serta cocok untuk menampilkan berbagai opsi pengaturan seperti yang sering ditemukan di aplikasi pengaturan perangkat.

Secara keseluruhan, aplikasi ini menunjukkan cara sederhana menggunakan `ListView` dan `ListTile` untuk membuat daftar interaktif dengan elemen yang dapat digeser dan diatur ulang dengan mudah.

- Hasil

- ListView Builder
  - Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const ListviewBuilderPage(),
    );
  }
}

class ListviewBuilderPage extends StatefulWidget {
  const ListviewBuilderPage({super.key});

  @override
  State<ListviewBuilderPage> createState() =>
_ListViewBuilderPageState();
}

class _ListViewBuilderPageState extends State<ListviewBuilderPage> {
  List<String> listData = List<String>.generate(100, (index) => 'data
$index');

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'ListView Builder',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: SafeArea(
          child: ListView.builder(
```

```
      itemCount: listData.length,
      shrinkWrap: true,
      physics: const BouncingScrollPhysics(),
      padding: EdgeInsets.zero,
      itemBuilder: (context, index) => ListTile(
        dense: true,
        title: Text(
          listData[index],
          style: const TextStyle(fontSize: 16.0),
        ),
      ),
    )),
  );
 }
}
```

- Penjelasan Kode

Kode di atas adalah implementasi aplikasi Flutter yang menampilkan daftar dinamis menggunakan `ListView.builder`. Aplikasi ini terdiri dari dua bagian utama: `MyApp` dan `ListviewBuilderPage`. Fungsi `main()` berfungsi sebagai entry point untuk menjalankan aplikasi Flutter dengan memanggil `runApp()` dan menjalankan widget `MyApp`, yang merupakan turunan dari `StatelessWidget`.

Pada kelas `MyApp`, aplikasi dibungkus dengan `MaterialApp`, yang menyediakan fitur dasar seperti tema dan navigasi. Dalam `MaterialApp`, tema yang digunakan dibuat dengan menggunakan `ThemeData` dan `ColorScheme.fromSeed`, yang menghasilkan skema warna berdasarkan warna ungu tua (`Colors.deepPurple`). Aplikasi ini menampilkan halaman utama `ListviewBuilderPage`, yang merupakan halaman Stateful karena elemen dalam halaman tersebut dapat berubah seiring interaksi pengguna.

Kelas `ListviewBuilderPage` adalah halaman utama yang akan ditampilkan. Kelas ini merupakan `StatefulWidget`, artinya state atau data di dalamnya bisa berubah tanpa harus memuat ulang seluruh halaman. Pada kelas ini, sebuah daftar data dibuat menggunakan `List<String>.generate(100, (index) => 'data $index')`. Ini adalah metode untuk membuat daftar yang terdiri dari 100 elemen string, di mana setiap elemen akan berupa teks dengan pola "data" diikuti nomor indeksnya, seperti "data 0", "data 1", dan seterusnya.

Pada bagian metode `build()` di `ListviewBuilderPage`, sebuah `Scaffold` digunakan untuk menyediakan struktur dasar halaman, dengan `AppBar` di

bagian atas dan `body` yang menampilkan konten utama. `AppBar` menampilkan teks "ListView Builder" sebagai judul menggunakan widget `Text` dengan pengaturan ukuran dan gaya tertentu.

Konten utama halaman ditampilkan melalui `ListView.builder`, yang merupakan widget yang secara efisien membuat daftar elemen yang panjang dan hanya merender elemen yang terlihat di layar. Properti `itemCount` diatur ke jumlah item dalam daftar `listData`, sehingga panjang daftar sesuai dengan jumlah elemen yang dihasilkan sebelumnya, yaitu 100 elemen. Fungsi `itemBuilder` digunakan untuk membangun tampilan setiap item dalam daftar. Dalam hal ini, setiap item dalam daftar diwakili oleh widget `ListTile`, yang menampilkan teks dari `listData` sesuai dengan indeks saat ini.

Dengan menggunakan `ListView.builder`, aplikasi ini lebih efisien dalam menangani daftar data yang panjang, karena hanya elemen yang terlihat di layar yang dirender, sehingga menghemat memori dan waktu pemrosesan. Properti `physics` diatur ke `BouncingScrollPhysics()`, yang memberikan efek "bouncing" ketika pengguna menggulir daftar hingga ke ujung atas atau bawah, menciptakan pengalaman yang lebih halus dan responsif.

Secara keseluruhan, aplikasi ini memanfaatkan `ListView.builder` untuk menampilkan daftar panjang secara dinamis dengan kinerja yang optimal, sekaligus menjaga tata letak yang sederhana dan interaktif.

➢ Hasil

- ListView Separated

➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
```

```
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const ListviewSeparatedPage(),
    );
  }
}

class ListviewSeparatedPage extends StatefulWidget {
  const ListviewSeparatedPage({super.key});

  @override
  State<ListviewSeparatedPage> createState() =>
_ListviewSeparatedPageState();
}

class _ListviewSeparatedPageState extends
State<ListviewSeparatedPage> {
  List<String> listData = List<String>.generate(100, (index) => 'data
$index');

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'ListView Separated',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: SafeArea(
          child: ListView.separated(
        itemCount: listData.length,
        shrinkWrap: true,
        physics: const BouncingScrollPhysics(),
        padding: EdgeInsets.zero,
        itemBuilder: (context, index) => ListTile(
          dense: true,
          title: Text(
            listData[index],
            style: const TextStyle(fontSize: 16.0),
          ),
        ),
        separatorBuilder: (context, index) => const Divider(
          height: 0.0,
        ),
      )),
```

```
    );
  }
}
```

- Penjelasan Kode

Kode di atas adalah sebuah aplikasi Flutter sederhana yang menampilkan daftar data menggunakan `ListView.separated`. Aplikasi ini dibagi menjadi dua bagian utama: `MyApp` dan `ListviewSeparatedPage`. Fungsi `main()` bertugas menjalankan aplikasi dengan memanggil `runApp()` yang akan memuat widget `MyApp`.

Pada kelas `MyApp`, yang merupakan turunan dari `StatelessWidget`, aplikasi diatur menggunakan `MaterialApp`, yang memberikan kerangka dasar aplikasi, seperti tema dan navigasi. Di sini, tema aplikasi disesuaikan dengan `ThemeData` yang menggunakan skema warna ungu tua (`Colors.deepPurple`) yang dihasilkan oleh `ColorScheme.fromSeed`. Halaman utama aplikasi ini ditentukan sebagai `ListviewSeparatedPage`, yang akan menampilkan daftar data dengan pembatas antar elemen.

Pada kelas `ListviewSeparatedPage`, yang merupakan turunan dari `StatefulWidget`, sebuah daftar data dibuat menggunakan `List<String>.generate(100, (index) => 'data $index')`. Ini berarti bahwa aplikasi akan menampilkan daftar yang terdiri dari 100 elemen, dengan masing-masing elemen berupa teks seperti "data 0", "data 1", dan seterusnya.

Di dalam metode `build()` pada `ListviewSeparatedPage`, struktur dasar halaman diatur dengan menggunakan `Scaffold`. Bagian atas halaman diisi dengan `AppBar` yang menampilkan judul "ListView Separated" menggunakan widget `Text` dengan pengaturan gaya teks tertentu. Konten utama halaman diatur dalam `body`, yang diisi dengan widget `ListView.separated`. Berbeda dengan `ListView.builder`, `ListView.separated` memungkinkan pemisahan antar elemen daftar menggunakan widget pemisah, dalam hal ini `Divider`.

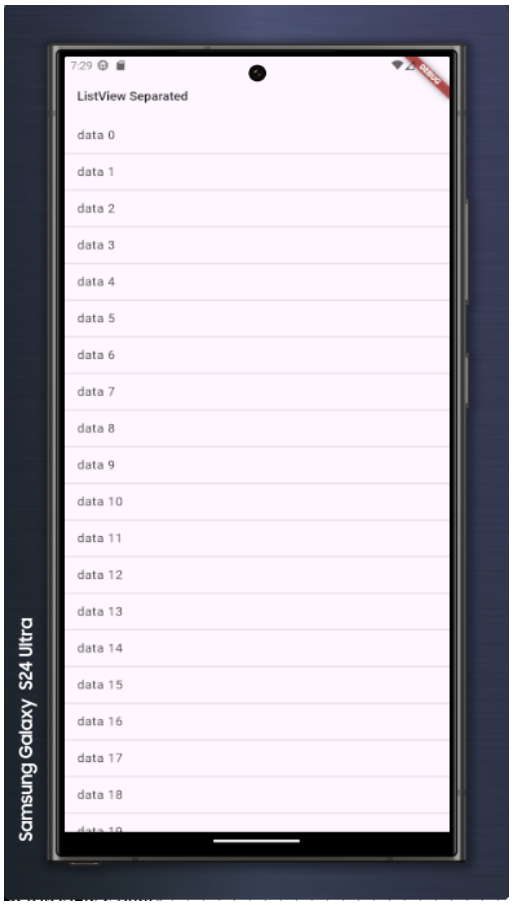
Properti `itemCount` menentukan jumlah item yang ditampilkan di dalam daftar, yang diambil dari panjang `listData`. Metode `itemBuilder` digunakan untuk membangun setiap item dalam daftar, di mana setiap item ditampilkan sebagai `ListTile` yang menampilkan teks dari elemen `listData` yang sesuai

dengan indeksnya. Teks ditampilkan dengan ukuran font 16, sebagaimana diatur dalam `TextStyle`.

Salah satu fitur penting dari `ListView.separated` adalah pemisahan antar item yang diatur dengan `separatorBuilder`. Dalam kode ini, setiap item dipisahkan oleh sebuah `Divider` dengan `height` yang diatur menjadi nol, yang berarti bahwa pembatas tersebut akan menjadi garis horizontal tipis di antara setiap elemen dalam daftar. Efek pemisah ini membuat daftar lebih mudah dibaca dan terlihat lebih rapi. Properti `physics` diatur ke `BouncingScrollPhysics()`, yang menciptakan efek gulir elastis ketika pengguna mencapai batas atas atau bawah daftar.

Secara keseluruhan, aplikasi ini memanfaatkan `ListView.separated` untuk menampilkan daftar data dengan pemisah antar elemen, menciptakan antarmuka yang bersih dan terstruktur. Aplikasi ini sangat efisien dalam menangani daftar data yang panjang, dengan penggunaan pemisah untuk meningkatkan keterbacaan dan memberikan pengalaman pengguna yang lebih baik.

- Hasil



- ListView Custom
  - Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const ListviewCustomPage(),
    );
  }
}

class ListviewCustomPage extends StatefulWidget {
  const ListviewCustomPage({super.key});

  @override
  State<ListviewCustomPage> createState() =>
_ListViewCustomPageState();
}

class _ListViewCustomPageState extends State<ListviewCustomPage> {

  List<String> listData = List<String>.generate(100, (index) => 'Data
$index');
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'ListView Custom',
          style: TextStyle(
            fontSize: 16.0,
            fontWeight: FontWeight.w600
          ),
        ),
```

```
    ),
    body: SafeArea(
      child: ListView.custom(
        shrinkWrap: true,
        physics: const BouncingScrollPhysics(),
        padding: EdgeInsets.zero,
        childrenDelegate: SliverChildBuilderDelegate(
          (context, index) => ListTile(
            dense: true,
            title: Text(
              listData[index],
              style: const TextStyle(
                fontSize: 16.0
              ),
            ),
          ),
          childCount: listData.length,
        ),
      )
    ),
  );
 }
}
```

- Penjelasan Kode

Kode di atas adalah implementasi sebuah aplikasi Flutter yang menampilkan daftar data menggunakan `ListView.custom` dengan `SliverChildBuilderDelegate`. Fungsi utama dari kode ini dimulai pada `main()`, di mana aplikasi dijalankan menggunakan `runApp()` yang memuat widget `MyApp`.

Pada `MyApp`, yang merupakan turunan dari `StatelessWidget`, aplikasi dikemas dalam sebuah `MaterialApp`. `MaterialApp` ini bertugas mengatur tema aplikasi dengan skema warna yang ditentukan melalui `ColorScheme.fromSeed`, yang dalam hal ini menggunakan warna dasar ungu tua (`Colors.deepPurple`). Aplikasi ini menggunakan Material 3 dan menampilkan halaman utama `ListviewCustomPage`.

`ListviewCustomPage` adalah turunan dari `StatefulWidget`, yang memungkinkan pengelolaan state internal dari widget ini. Pada bagian `State<ListviewCustomPage>`, terdapat sebuah daftar data yang dihasilkan secara dinamis dengan `List.generate()`. Ini menghasilkan daftar 100 item

dengan teks seperti "Data 0", "Data 1", dan seterusnya. Hal ini memastikan bahwa aplikasi dapat menampilkan banyak data secara efisien.

Pada metode `build()`, antarmuka utama aplikasi diatur dengan menggunakan widget `Scaffold`, yang memberikan struktur dasar halaman. `AppBar` digunakan di bagian atas halaman dengan judul "ListView Custom". Gaya teks judul diatur menggunakan `TextStyle` dengan ukuran font 16 dan ketebalan teks yang lebih tinggi (`FontWeight.w600`).

Bagian utama dari antarmuka aplikasi berada di dalam `SafeArea`, yang memastikan bahwa konten tidak terpotong oleh batas-batas layar atau elemen antarmuka lainnya. Dalam `SafeArea`, widget `ListView.custom` digunakan untuk menampilkan daftar data. `ListView.custom` adalah versi yang lebih fleksibel dari `ListView`, yang memungkinkan penggunaan `SliverChildBuilderDelegate` untuk menangani daftar data secara dinamis. Ini sangat efisien untuk menampilkan daftar dengan elemen yang banyak, karena hanya membangun widget yang terlihat di layar dan mendaur ulang widget yang sudah tidak terlihat.

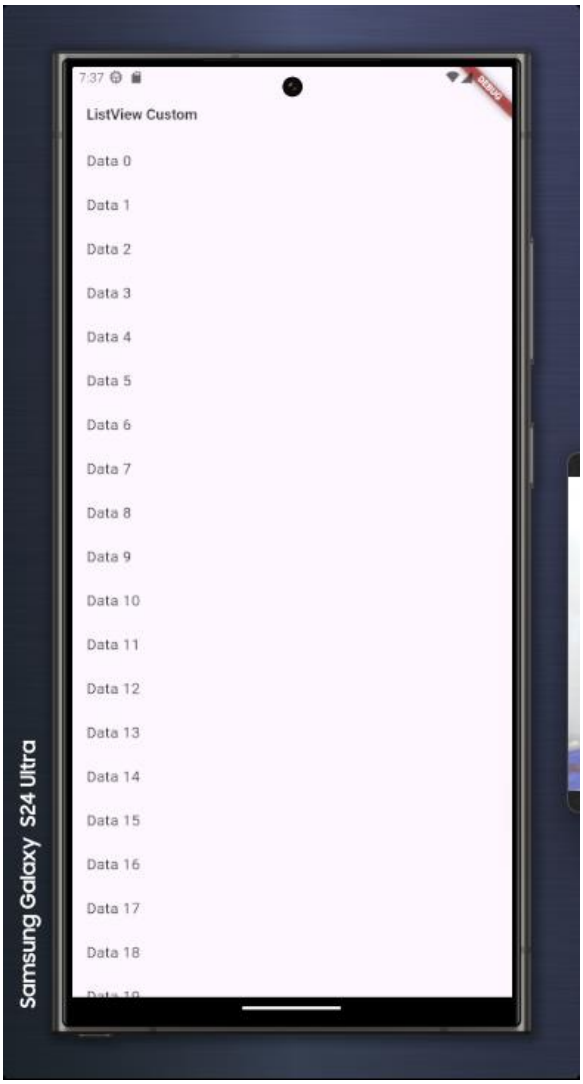
`SliverChildBuilderDelegate` digunakan untuk membangun setiap item dalam daftar dengan cara yang efisien. Metode `builder` dari delegate ini menerima indeks item yang sedang dibangun dan menghasilkan `ListTile` untuk setiap item. Setiap `ListTile` menampilkan teks yang diambil dari daftar `listData`, dengan gaya teks yang diatur dengan ukuran font 16. Properti `childCount` pada delegate menentukan jumlah item yang harus dibangun, yaitu panjang dari daftar `listData`.

Selain itu, `ListView.custom` juga memiliki properti `physics` yang diatur ke `BouncingScrollPhysics()`, yang memberikan efek elastis ketika pengguna menggulir ke ujung daftar. Properti `shrinkWrap` diatur ke `true`, yang memungkinkan daftar untuk menyusut sesuai dengan jumlah item yang ada, berguna ketika daftar ditempatkan di dalam elemen lain yang tidak memerlukan pengguliran penuh layar.

Secara keseluruhan, kode ini menampilkan sebuah aplikasi yang memanfaatkan `ListView.custom` untuk menampilkan daftar data yang panjang dengan efisien. Penggunaan `SliverChildBuilderDelegate` memastikan bahwa hanya elemen yang diperlukan saja yang dibangun dan

ditampilkan, meningkatkan kinerja aplikasi, terutama ketika berhadapan dengan data yang banyak.

➢ Hasil



1.2.3. GridView Widget

- GridView Builder

➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
```

```
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const GridviewBuilderPage(),
    );
  }
}

class GridviewBuilderPage extends StatefulWidget {
  const GridviewBuilderPage({super.key});

  @override
  State<GridviewBuilderPage> createState() =>
_GridviewBuilderPageState();
}

class _GridviewBuilderPageState extends State<GridviewBuilderPage> {
  List<String> listProduct =
      List<String>.generate(50, (index) => 'Product $index');

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'GridView Builder',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: SafeArea(
          child: GridView.builder(
              itemCount: listProduct.length,
              shrinkWrap: true,
              physics: const BouncingScrollPhysics(),
              padding: const EdgeInsets.all(16.0),
              gridDelegate: const
SliverGridDelegateWithFixedCrossAxisCount(
                  crossAxisCount: 3,
                  mainAxisSpacing: 12.0,
                  crossAxisSpacing: 12.0),
              itemBuilder: (context, index) => Container(
                    decoration: BoxDecoration(
                      color: Colors.blue,
```

```
                borderRadius: BorderRadius.circular(8.0),
              ),
              child: Center(
                child: Text(
              listProduct[index],
            style: const TextStyle(
            fontSize: 16.0, color: Colors.white),
          textAlign: TextAlign.center,
        ),
      ),
    ),
  ),
 ),
 );
 }
}
```

- Penjelasan Kode

Kode ini merupakan sebuah aplikasi Flutter yang menggunakan `GridView.builder` untuk menampilkan daftar produk dalam format grid. Aplikasi dimulai pada fungsi `main()`, di mana aplikasi dijalankan menggunakan `runApp()` yang memuat widget `MyApp`. Pada `MyApp`, yang merupakan turunan dari `StatelessWidget`, aplikasi dikemas menggunakan `MaterialApp`, yang berfungsi mengelola tema dan halaman utama aplikasi.

Dalam `MaterialApp`, aplikasi diberi judul "Quick Note" dan menggunakan tema berdasarkan `ColorScheme` dengan warna dasar ungu tua (`Colors.deepPurple`). Aplikasi ini menggunakan Material 3 dan menampilkan halaman utama `GridviewBuilderPage`.

`GridviewBuilderPage` adalah turunan dari `StatefulWidget`, yang memungkinkan pengelolaan state secara dinamis. Pada kelas `_GridviewBuilderPageState`, terdapat sebuah daftar produk yang dihasilkan secara dinamis menggunakan `List.generate()`, menghasilkan 50 item dengan label seperti "Product 0", "Product 1", dan seterusnya. Hal ini digunakan sebagai data yang akan ditampilkan dalam grid.

Dalam metode `build()`, struktur halaman diatur dengan `Scaffold`, yang menyediakan antarmuka dasar untuk aplikasi. `AppBar` berada di bagian atas halaman dan menampilkan judul "GridView Builder" dengan gaya teks yang

diatur menggunakan `TextStyle`, dengan ukuran font 16 dan ketebalan huruf yang lebih besar (`FontWeight.w600`).

Bagian utama dari halaman ini menggunakan `SafeArea` untuk memastikan konten tidak terpotong oleh batas perangkat. Di dalam `SafeArea`, `GridView.builder` digunakan untuk menampilkan daftar produk dalam format grid. `GridView.builder` adalah widget yang efisien karena hanya membangun item yang diperlukan sesuai dengan tampilan layar. Pada properti `itemCount`, jumlah item yang ditampilkan adalah sebanyak elemen yang ada dalam daftar `listProduct`.

Properti `gridDelegate` pada `GridView.builder` menentukan bagaimana grid harus diatur. Dalam hal ini, `SliverGridDelegateWithFixedCrossAxisCount` digunakan untuk menentukan jumlah kolom grid (yaitu 3 kolom), serta jarak antar kolom (`crossAxisSpacing`) dan jarak antar baris (`mainAxisSpacing`) yang diatur masing-masing sebesar 12.0 pixel.

Untuk setiap item dalam grid, `itemBuilder` membangun sebuah `Container` yang berfungsi sebagai kotak untuk menampilkan produk. `Container` ini memiliki dekorasi berupa warna latar belakang biru (`Colors.blue`) dan sudut membulat dengan radius 8.0 pixel. Di dalam `Container`, terdapat widget `Center` yang memposisikan teks produk di tengah. Teks produk diambil dari daftar `listProduct` dan ditampilkan dalam `Text` dengan ukuran font 16, warna putih, dan teks rata tengah.

Secara keseluruhan, aplikasi ini menampilkan grid dari daftar produk yang dihasilkan secara dinamis. Penggunaan `GridView.builder` memungkinkan aplikasi ini untuk menangani banyak data secara efisien dengan hanya membangun elemen yang diperlukan sesuai dengan tampilan layar. Grid ini juga diatur untuk memberikan jarak yang konsisten antar elemen, sehingga menghasilkan tampilan yang terorganisir dan rapi.

➢ Hasil

- GridView Count

➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
```

```
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const GridViewCountPage(),
    );
  }
}

class GridViewCountPage extends StatefulWidget {
  const GridViewCountPage({super.key});

  @override
  State<GridViewCountPage> createState() =>
_GridViewCountPageState();
}

class _GridViewCountPageState extends State<GridViewCountPage>
{
    @override
    Widget build(BuildContext context) {
      return Scaffold(
        appBar: AppBar(
          title: const Text(
            'GridView Count',
            style: TextStyle(
              fontSize: 16.0,
              fontWeight: FontWeight.w600
            ),
          ),
        ),
        body: SafeArea(
          child: GridView.count(
            crossAxisCount: 3,
            mainAxisSpacing: 12.0,
            crossAxisSpacing: 12.0,
            shrinkWrap: true,
            physics: const BouncingScrollPhysics(),
            padding: const EdgeInsets.all(16.0),
            children: List.generate(50, (index) => Container(
              decoration: BoxDecoration(
                color: Colors.blue,
                borderRadius: BorderRadius.circular(8.0),
              ),
              child: Center(
                child: Text(
                  'Product $index',
                  style: const TextStyle(
```

```
                    fontSize: 16.0,
                    color: Colors.white
                  ),
                  textAlign: TextAlign.center,
                ),
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```

- Penjelasan Kode

Kode ini adalah aplikasi Flutter yang menggunakan widget `GridView.count` untuk menampilkan daftar produk dalam format grid. Aplikasi dimulai dari fungsi `main()`, di mana `runApp()` digunakan untuk memulai aplikasi dan memuat widget `MyApp`. Widget `MyApp` adalah turunan dari `StatelessWidget`, yang berarti ia tidak menyimpan status apa pun. Di dalam `MyApp`, aplikasi dikemas menggunakan `MaterialApp`, yang mengatur tema aplikasi dan menentukan halaman utama yang ditampilkan, yaitu `GridViewCountPage`.

Pada `MaterialApp`, judul aplikasi adalah "Quick Note" dan tema yang digunakan dibuat menggunakan `ThemeData` dengan skema warna ungu tua yang dihasilkan dari metode `ColorScheme.fromSeed()`. Material 3 diaktifkan dengan mengatur `useMaterial3` menjadi `true`, yang memperbarui gaya aplikasi sesuai dengan desain material terbaru.

Kelas `GridViewCountPage` adalah turunan dari `StatefulWidget`, memungkinkan untuk memanipulasi status di dalamnya. Di dalam kelas `_GridViewCountPageState`, metode `build()` mendefinisikan tampilan utama dari halaman. Aplikasi menggunakan `Scaffold` sebagai struktur dasar, yang menyediakan kerangka aplikasi Flutter, seperti `AppBar` dan area konten utama.

Pada bagian atas, `AppBar` ditampilkan dengan judul "GridView Count" menggunakan gaya teks yang sudah diatur (`fontSize` 16 dan `fontWeight.w600`). Bagian konten utama dari halaman ditampilkan di dalam

`SafeArea`, yang melindungi konten agar tidak tertutup oleh batas layar atau area notifikasi perangkat.

`GridView.count` digunakan untuk menampilkan elemen-elemen dalam grid. Pada widget ini, properti `crossAxisCount` diatur menjadi 3, yang berarti grid akan memiliki tiga kolom. `mainAxisSpacing` dan `crossAxisSpacing` masing-masing diatur menjadi 12.0, yang menentukan jarak antar baris dan kolom di dalam grid. Dengan `shrinkWrap` diatur menjadi `true`, grid akan hanya mengambil ruang yang dibutuhkan berdasarkan elemen-elemen yang akan ditampilkan, sementara `physics` diatur ke `BouncingScrollPhysics` untuk memberikan efek pantulan saat menggulir.

Grid ini diisi dengan elemen-elemen yang dihasilkan secara dinamis menggunakan `List.generate()`. Dalam hal ini, 50 produk dengan label seperti "Product 0", "Product 1", dan seterusnya, akan dihasilkan dan ditampilkan. Setiap elemen grid dibangun menggunakan widget `Container`, yang dihiasi dengan warna latar belakang biru (`Colors.blue`) dan sudut membulat (radius 8.0). Di dalam setiap `Container`, ada widget `Center` yang menempatkan teks produk di tengah-tengah. Teks tersebut ditampilkan dengan ukuran font 16, warna putih, dan rata tengah.

Secara keseluruhan, aplikasi ini menampilkan daftar produk dalam grid yang teratur, dengan tiga kolom dan jarak yang konsisten antar elemen. Penggunaan `GridView.count` memudahkan untuk menentukan jumlah kolom grid secara eksplisit, serta mengatur jarak antar elemen dengan properti `mainAxisSpacing` dan `crossAxisSpacing`.

➢ Hasil

- GridView Extent

➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
```

```
      colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
      useMaterial3: true,
    ),
    home: const GridViewExtentPage(),
   );
  }
}

class GridViewExtentPage extends StatefulWidget {
 const GridViewExtentPage({super.key});

 @override
 State<GridViewExtentPage> createState() =>
_GridViewExtentPageState();
}

class _GridViewExtentPageState extends State<GridViewExtentPage> {
  @override
  Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
      title: const Text(
       'GridView Extent',
       style: TextStyle(
        fontSize: 16.0,
        fontWeight: FontWeight.w600
       ),
      ),
     ),
     body: SafeArea(
      child: GridView.extent(
       maxCrossAxisExtent: 150.0,
       mainAxisSpacing: 12.0,
       crossAxisSpacing: 12.0,
       shrinkWrap: true,
       physics: const BouncingScrollPhysics(),
       padding: const EdgeInsets.all(16.0),
       children: List.generate(50, (index) => Container(
        decoration: BoxDecoration(
         color: Colors.blue,
         borderRadius: BorderRadius.circular(8.0),
        ),
        child: Center(
         child: Text(
          'Product $index',
          style: const TextStyle(
           fontSize: 16.0,
```

```
                color: Colors.white
              ),
              textAlign: TextAlign.center,
            ),
          ),
        ),
      ),
    ),
  );
 }
}
```

- Penjelasan Kode

Kode di atas adalah aplikasi Flutter yang menampilkan daftar produk dalam bentuk grid menggunakan widget `GridView.extent`. Proses dimulai dari fungsi `main()`, di mana `runApp()` digunakan untuk menjalankan aplikasi dengan `MyApp` sebagai widget utama. `MyApp` merupakan subclass dari `StatelessWidget`, artinya widget ini tidak menyimpan status dan lebih fokus pada penyajian UI. Dalam metode `build()`, aplikasi diatur dengan menggunakan `MaterialApp`, yang mencakup pengaturan tema dan halaman utama.

Aplikasi ini menggunakan tema yang diatur dengan `ThemeData`, di mana skema warna yang dihasilkan dari metode `ColorScheme.fromSeed` diisi dengan warna dasar ungu tua (deep purple). Hal ini memberikan tampilan visual yang konsisten dan modern pada aplikasi. Widget utama yang ditampilkan adalah `GridViewExtentPage`, yang diatur sebagai halaman rumah (home) dari aplikasi. Halaman ini diimplementasikan sebagai `StatefulWidget`, yang memungkinkan pengelolaan status dalam kelas yang dihasilkan.

Di dalam kelas `_GridViewExtentPageState`, metode `build()` menyusun tampilan halaman dengan menggunakan `Scaffold`. `Scaffold` berfungsi sebagai kerangka dasar untuk tata letak, termasuk menampilkan `AppBar` dan area konten utama. Pada bagian atas, `AppBar` menampilkan judul "GridView Extent" dengan gaya teks yang sudah ditentukan, yaitu ukuran font 16 dan berat font yang setara dengan 600. Hal ini memberikan struktur yang jelas bagi aplikasi.

Bagian konten utama dari halaman berada dalam `SafeArea`, yang melindungi tampilan dari area yang mungkin tertutup oleh sistem status atau navigasi perangkat. Di dalam `SafeArea`, widget `GridView.extent` digunakan untuk menampilkan produk dalam grid. Pengaturan `maxCrossAxisExtent` diatur menjadi 150.0, yang berarti setiap item grid tidak akan melebihi lebar 150 piksel, dan grid akan otomatis menyesuaikan jumlah kolom berdasarkan lebar layar. `mainAxisSpacing` dan `crossAxisSpacing` masing-masing diatur ke 12.0 untuk mengatur jarak antar item di dalam grid, sehingga tampilan grid menjadi rapi dan terorganisir.

Dengan pengaturan `shrinkWrap` diatur ke `true`, grid akan hanya mengambil ruang yang diperlukan sesuai dengan jumlah item yang ditampilkan, sementara `physics` menggunakan `BouncingScrollPhysics`, memberikan efek pantulan saat menggulir. Daftar item dalam grid dihasilkan menggunakan `List.generate()`, yang menghasilkan 50 item dengan label "Product 0", "Product 1", hingga "Product 49". Setiap item dalam grid diwakili oleh `Container`, yang memiliki latar belakang berwarna biru dan sudut membulat (radius 8.0). Di dalam setiap `Container`, terdapat widget `Center` yang menempatkan teks di tengah, ditampilkan dengan warna putih dan ukuran font 16. Ini memberikan efek visual yang menarik dan memudahkan pengguna untuk melihat nama produk.

Secara keseluruhan, aplikasi ini menggunakan `GridView.extent` untuk menampilkan produk dalam format grid yang fleksibel, memanfaatkan fitur tata letak yang disediakan oleh Flutter untuk menghasilkan antarmuka yang responsif dan mudah digunakan. Pengaturan jarak dan gaya visual menjadikan aplikasi ini tidak hanya fungsional tetapi juga menarik secara estetika.

➢ Hasil

- GridView Custom

➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
```

```
      colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
      useMaterial3: true,
    ),
    home: const GridviewCustomPage(),
   );
  }
}

class GridviewCustomPage extends StatefulWidget {
 const GridviewCustomPage({super.key});

 @override
 State<GridviewCustomPage> createState() =>
_GridviewCustomPageState();
}

class _GridviewCustomPageState extends State<GridviewCustomPage> {
 List<String> listProduct = List<String>.generate(50, (index) => 'Product
$index');

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text(
     'GridView Custom',
     style: TextStyle(
      fontSize: 16.0,
      fontWeight: FontWeight.w600
     ),
    ),
   ),
   body: SafeArea(
    child: GridView.custom(
     shrinkWrap: true,
     physics: const BouncingScrollPhysics(),
     padding: const EdgeInsets.all(16.0),
     gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
      crossAxisCount: 3,
      mainAxisSpacing: 12.0,
      crossAxisSpacing: 12.0,
     ),
     childrenDelegate: SliverChildBuilderDelegate((context, index) {
      if (index == 0) {
       return Container(
        decoration: BoxDecoration(
         color: Colors.red,
```

```
              borderRadius: BorderRadius.circular(8.0),
            ),
            child: Center(
              child: Text(
                listProduct[index],
                style: const TextStyle(
                  fontSize: 16.0,
                  color: Colors.white
                ),
                textAlign: TextAlign.center,
              ),
            ),
          );
        } else {
          return Container(
            decoration: BoxDecoration(
              color: Colors.blue,
              borderRadius: BorderRadius.circular(8.0),
            ),
            child: Center(
              child: Text(
                listProduct[index],
                style: const TextStyle(
                  fontSize: 16.0,
                  color: Colors.white
                ),
                textAlign: TextAlign.center,
              ),
            ),
          );
        }
      }, childCount: listProduct.length),
    )
  ),
 );
 }
}
```

- Penjelasan Kode

Kodingan di atas adalah aplikasi Flutter yang menampilkan daftar produk dalam bentuk grid dengan tampilan yang kustom. Proses eksekusi dimulai dari fungsi `main()`, di mana `runApp()` digunakan untuk menjalankan aplikasi dengan `MyApp` sebagai widget utama. Kelas `MyApp` merupakan subclass dari `StatelessWidget`, yang berarti widget ini tidak memiliki status yang dapat berubah, dan lebih berfokus pada penyajian UI. Di dalam metode `build()`, aplikasi diatur menggunakan `MaterialApp`, di mana beberapa

parameter seperti `title` dan `theme` ditentukan, dengan tema yang diatur menggunakan `ThemeData`.

Tema yang digunakan dalam aplikasi ini mengandalkan `ColorScheme.fromSeed`, yang menghasilkan skema warna dengan warna dasar ungu tua (deep purple). Aplikasi ini memiliki halaman utama yang ditetapkan sebagai `GridviewCustomPage`, yang merupakan widget bertipe `StatefulWidget`. Hal ini memungkinkan pengelolaan status dan pembaruan tampilan ketika diperlukan. Di dalam kelas `GridviewCustomPage`, terdapat status yang dikelola di dalam `_GridviewCustomPageState`, di mana data produk disimpan dalam sebuah list bernama `listProduct`, yang diisi dengan 50 string, masing-masing berisi "Product 0" hingga "Product 49".

Di dalam metode `build()` dari `_GridviewCustomPageState`, tampilan halaman disusun menggunakan `Scaffold`, yang memberikan struktur dasar untuk halaman, termasuk bagian `AppBar` dan area konten utama. Pada bagian atas, `AppBar` menampilkan judul "GridView Custom" dengan gaya teks yang sudah ditentukan. Konten utama terletak dalam `SafeArea`, yang menjaga tampilan agar tidak tertutupi oleh sistem status atau navigasi perangkat. Dalam `SafeArea`, widget `GridView.custom` digunakan untuk menampilkan data produk dalam format grid. `shrinkWrap` diatur ke `true`, yang berarti grid akan mengambil ruang sesuai dengan jumlah item yang ditampilkan, sedangkan `physics` menggunakan `BouncingScrollPhysics`, memberikan efek pantulan saat menggulir.

Pengaturan `gridDelegate` menggunakan `SliverGridDelegateWithFixedCrossAxisCount`, yang menentukan bahwa jumlah kolom dalam grid adalah 3. `mainAxisSpacing` dan `crossAxisSpacing` diatur masing-masing ke 12.0 untuk memberikan jarak antar item, sehingga tampilan grid menjadi rapi dan terorganisir. Pada bagian `childrenDelegate`, `SliverChildBuilderDelegate` digunakan untuk menghasilkan tampilan item berdasarkan indeks. Dalam fungsi ini, jika indeks adalah 0, item tersebut ditampilkan dengan latar belakang merah, sedangkan item lainnya menggunakan latar belakang biru. Setiap item di dalam grid diwakili oleh `Container` yang memiliki sudut membulat dan teks yang diletakkan di tengah. Teks ditampilkan dengan ukuran font 16.0 dan warna putih, sehingga kontras dengan latar belakang yang berwarna.

Dengan cara ini, aplikasi ini tidak hanya mampu menampilkan daftar produk, tetapi juga memberikan pengalaman visual yang menarik dan dinamis. Pengaturan warna yang berbeda untuk item pertama membuat aplikasi lebih menonjol dan membantu pengguna untuk mengenali item tersebut dengan mudah. Secara keseluruhan, aplikasi ini menunjukkan bagaimana Flutter dapat digunakan untuk membuat antarmuka yang responsif dan kustom dengan sedikit usaha, menggunakan widget yang telah ada dan pengaturan yang tepat.

➢ Hasil

1.2.4. Flutter Navigation

- Navigator Push dan Pop

➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const PushNavigationPage(),
    );
  }
}

class PushNavigationPage extends StatefulWidget {
  const PushNavigationPage({super.key});

  @override
  State<PushNavigationPage> createState() =>
_PushNavigationPageState();
}

class _PushNavigationPageState extends State<PushNavigationPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Push Navigation',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: SafeArea(
          child: Center(
        child: ElevatedButton(
            onPressed: () => Navigator.push(
                context,
                MaterialPageRoute(
```

```
              builder: (context) => const DetailPage(),
            )),
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.blue,
            foregroundColor: Colors.white,
          ),
          child: const Text(
            'Go to Detail Page',
          ),
        ),
      ),
    ),
  );
 }
}
```

- Penjelasan Kode

Kodingan di atas adalah aplikasi Flutter yang menerapkan navigasi dasar menggunakan `Push Navigation`. Proses eksekusi dimulai dari fungsi `main()`, yang menjadi titik masuk aplikasi, di mana `runApp()` digunakan untuk menjalankan aplikasi dengan widget `MyApp`. Kelas `MyApp` merupakan subclass dari `StatelessWidget`, artinya widget ini tidak memiliki status yang dapat berubah dan berfungsi untuk menyajikan antarmuka pengguna. Di dalam metode `build()`, `MaterialApp` diatur dengan beberapa parameter, termasuk `title` yang memberikan nama aplikasi dan `theme` yang ditentukan dengan menggunakan `ThemeData`, mengatur skema warna berdasarkan warna ungu tua (deep purple) yang menghasilkan tampilan yang modern dan menarik.

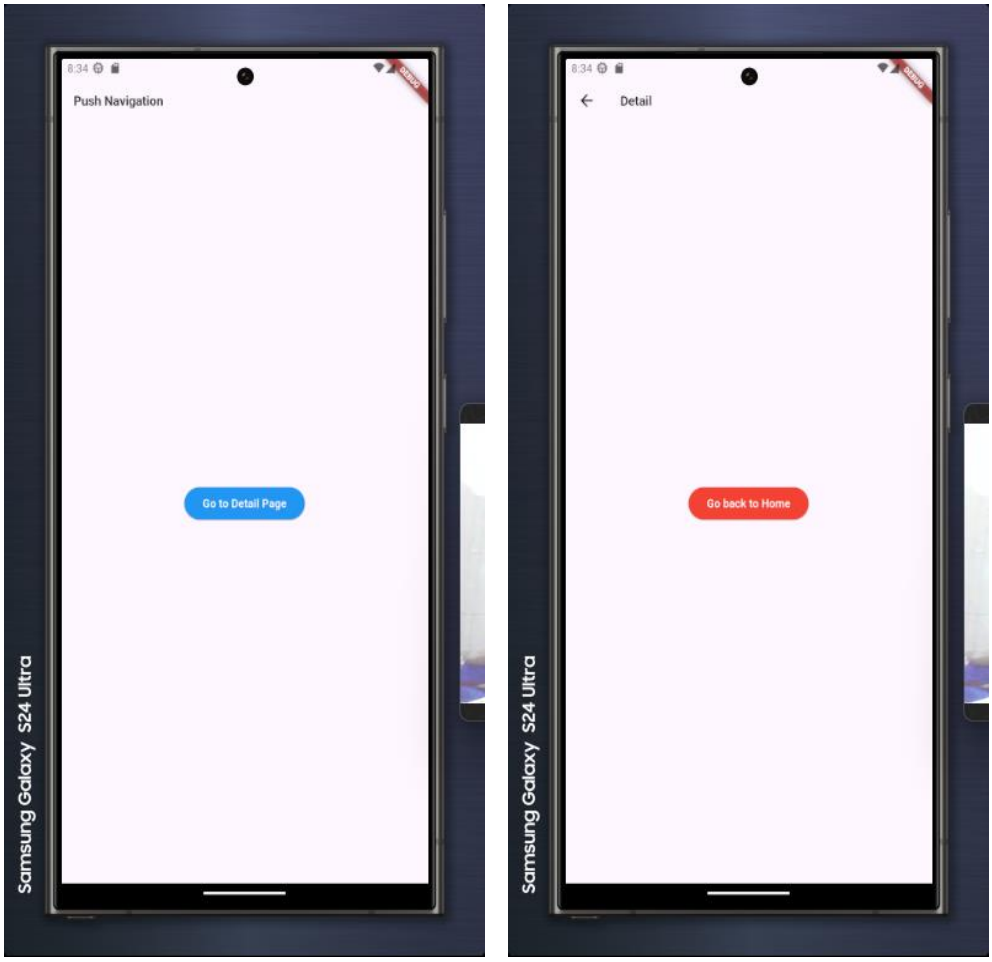
Setelah itu, halaman awal dari aplikasi ditentukan sebagai `PushNavigationPage`, yang merupakan widget bertipe `StatefulWidget`. Hal ini memungkinkan halaman untuk mengelola status dan memperbarui tampilannya sesuai dengan interaksi pengguna. Di dalam kelas `PushNavigationPage`, terdapat `_PushNavigationPageState` yang mengatur state untuk halaman tersebut. Dalam metode `build()` dari `_PushNavigationPageState`, struktur halaman disusun dengan menggunakan `Scaffold`, yang memberikan elemen dasar seperti `AppBar` dan area konten utama. Judul di dalam `AppBar` berisi teks "Push Navigation" dengan gaya tertentu, seperti ukuran dan bobot font.

Di bagian konten, widget `SafeArea` digunakan untuk memastikan bahwa elemen UI tidak tertutup oleh sistem status atau notch perangkat. Di dalam `SafeArea`, terdapat widget `Center`, yang mengatur tata letak sehingga elemen di dalamnya ditempatkan di tengah halaman. Di dalam widget `Center`, terdapat `ElevatedButton`, yang merupakan tombol yang memiliki gaya yang diatur untuk memiliki latar belakang biru dan teks putih. Tombol ini diberi label "Go to Detail Page". Saat tombol ditekan, fungsi yang ditentukan dalam parameter `onPressed` akan dijalankan, yang berfungsi untuk melakukan navigasi ke halaman detail.

Navigasi dilakukan dengan memanggil `Navigator.push`, yang mengambil konteks saat ini dan membuat rute baru menggunakan `MaterialPageRoute`. Rute ini membangun halaman baru dengan widget `DetailPage`, yang seharusnya merupakan halaman baru yang didefinisikan dalam file `detail.dart` (meskipun implementasi `DetailPage` tidak ditampilkan dalam kode yang diberikan). Dengan demikian, ketika pengguna menekan tombol, aplikasi akan beralih ke halaman detail, menciptakan pengalaman yang lebih interaktif. Hal ini menunjukkan cara dasar untuk mengimplementasikan navigasi dalam aplikasi Flutter, memberikan pengguna kemampuan untuk menjelajahi konten tambahan dengan mudah.

Secara keseluruhan, aplikasi ini mengilustrasikan prinsip dasar pembuatan aplikasi Flutter dengan antarmuka pengguna yang sederhana dan kemampuan navigasi. Hal ini sangat berguna dalam membangun aplikasi yang lebih kompleks, di mana pengguna dapat berpindah dari satu layar ke layar lainnya dengan lancar. Kode ini juga menunjukkan bagaimana Flutter memungkinkan pengembang untuk dengan cepat menyusun antarmuka dan mengelola interaksi pengguna.

➢ Hasil



- Navigator Push Replacement

➢ Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
```

```
      home: const PushreplaceNavigationPage(),
    );
  }
}

class PushreplaceNavigationPage extends StatefulWidget {
  const PushreplaceNavigationPage({super.key});

  @override
  State<PushreplaceNavigationPage> createState() =>
      _PushreplaceNavigationPageState();
}

class _PushreplaceNavigationPageState extends
State<PushreplaceNavigationPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Push Replace Navigation',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: SafeArea(
          child: Center(
        child: ElevatedButton(
            onPressed: () => Navigator.pushReplacement(
                context,
                MaterialPageRoute(
                  builder: (context) => const DetailPage(),
                )),
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.blue,
              foregroundColor: Colors.white,
            ),
            child: const Text(
              'Go to Detail Page'
            ),
          ),
        ),
      ),
    );
  }
}
```

- Penjelasan Kode

Kodingan di atas adalah implementasi aplikasi Flutter yang menggunakan navigasi pengganti (push replace navigation) untuk berpindah dari satu halaman ke halaman lain. Proses dimulai dari fungsi `main()`, yang merupakan titik masuk aplikasi. Dalam fungsi ini, `runApp()` dipanggil untuk menjalankan widget `MyApp`, yang berfungsi sebagai kerangka utama aplikasi. Kelas `MyApp` adalah subclass dari `StatelessWidget`, yang berarti bahwa widget ini tidak menyimpan status yang dapat berubah, dan hanya bertugas untuk merender antarmuka pengguna. Di dalam metode `build()` dari kelas ini, `MaterialApp` diatur dengan beberapa parameter, termasuk judul aplikasi yang disebut "Quick Note" dan tema yang diatur dengan `ThemeData`, yang menggunakan skema warna yang dihasilkan dari warna ungu tua (deep purple), menciptakan tampilan yang modern dan menarik.

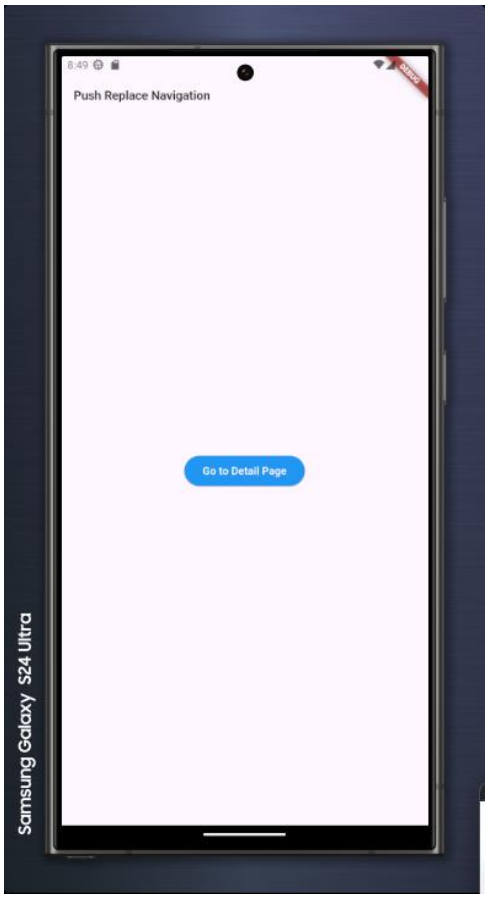
Setelah itu, halaman utama aplikasi diatur menjadi `PushreplaceNavigationPage`, yang merupakan widget bertipe `StatefulWidget`. Dengan menggunakan `StatefulWidget`, halaman ini mampu mengelola status dan memperbarui tampilannya berdasarkan interaksi pengguna. Dalam kelas `PushreplaceNavigationPage`, terdapat `_PushreplaceNavigationPageState` yang mengatur state untuk halaman tersebut. Di dalam metode `build()` dari `_PushreplaceNavigationPageState`, struktur halaman diatur menggunakan `Scaffold`, yang memberikan elemen dasar seperti `AppBar` dan area konten utama. Judul di dalam `AppBar` ditampilkan dengan teks "Push Replace Navigation" dengan gaya tertentu, termasuk ukuran dan bobot font.

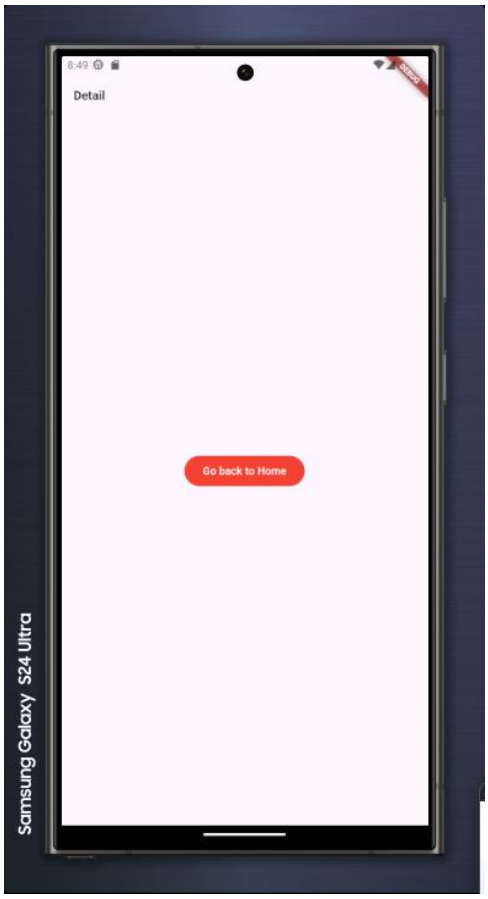
Di bagian konten halaman, widget `SafeArea` digunakan untuk memastikan bahwa elemen antarmuka tidak tertutup oleh bagian layar yang sensitif, seperti notch atau sistem status perangkat. Di dalam `SafeArea`, terdapat widget `Center` yang mengatur agar elemen di dalamnya berada di tengah halaman. Dalam widget `Center`, terdapat tombol `ElevatedButton`, yang memiliki gaya ditentukan dengan latar belakang biru dan teks putih. Tombol ini dilabeli "Go to Detail Page". Ketika tombol ditekan, fungsi yang ditentukan dalam parameter `onPressed` akan dijalankan. Dalam fungsi ini, `Navigator.pushReplacement` digunakan untuk berpindah ke halaman baru dengan cara menggantikan halaman saat ini dengan halaman baru, dalam hal ini adalah `DetailPage`.

Dengan menggunakan `Navigator.pushReplacement`, halaman yang aktif saat ini akan digantikan dengan halaman yang baru dibangun, yaitu `DetailPage`. Penggunaan `MaterialPageRoute` memungkinkan pembuatan rute baru yang akan ditampilkan. Meskipun implementasi `DetailPage` tidak ditampilkan dalam kode ini, dapat memahami bahwa halaman detail ini akan ditampilkan ketika tombol ditekan, dan halaman sebelumnya tidak akan tetap ada di tumpukan navigasi, sehingga saat pengguna menekan tombol kembali, mereka tidak akan kembali ke halaman sebelumnya.

Secara keseluruhan, aplikasi ini menggambarkan bagaimana Flutter memfasilitasi pembuatan antarmuka pengguna yang responsif dengan navigasi yang sederhana namun efektif. Dengan menggunakan `PushreplaceNavigation`, aplikasi dapat memberikan pengalaman pengguna yang lebih bersih, di mana pengguna langsung melompat ke halaman detail tanpa harus kembali ke halaman sebelumnya. Ini menunjukkan bagaimana Flutter dapat digunakan untuk membangun aplikasi yang memiliki alur navigasi yang intuitif dan efisien.

- Hasil





- Navigator Push dan Remove Until
  - Kode Pemrograman

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const PushremoveuntilNavigationPage(),
    );
  }
```

```
}

class PushremoveuntilNavigationPage extends StatefulWidget {
  const PushremoveuntilNavigationPage({super.key});

  @override
  State<PushremoveuntilNavigationPage> createState() =>
      _PushremoveuntilNavigationPageState();
}

class _PushremoveuntilNavigationPageState
    extends State<PushremoveuntilNavigationPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Push Removeuntil Navigation',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: SafeArea(
          child: Center(
        child: ElevatedButton(
            onPressed: () => Navigator.pushAndRemoveUntil(
                  context,
                  MaterialPageRoute(
                    builder: (context) => const DetailPage(),
                  ),
                  (route) => false,
                ),
            style: ElevatedButton.styleFrom(
                backgroundColor: Colors.red, foregroundColor: Colors.white),
            child: const Text(
              'Go to Detail Page'
            ),
          ),
        ),
      ),
    );
  }
}
```

- Penjelasan Kode

Kodingan di atas adalah implementasi aplikasi Flutter yang menggunakan metode navigasi bernama `push and remove until`. Aplikasi ini memiliki fungsi dasar yang memungkinkan pengguna untuk berpindah dari halaman

utama ke halaman detail, sambil menghapus semua halaman yang ada di tumpukan navigasi sebelumnya. Proses dimulai dari fungsi `main()`, yang merupakan titik awal eksekusi aplikasi. Di dalam fungsi ini, `runApp()` dipanggil untuk menjalankan widget `MyApp`, yang berfungsi sebagai kerangka utama aplikasi. Kelas `MyApp` adalah turunan dari `StatelessWidget`, yang menunjukkan bahwa widget ini tidak memiliki status yang dapat berubah, dan berfungsi untuk merender antarmuka pengguna.

Di dalam metode `build()` pada kelas `MyApp`, `MaterialApp` diatur dengan berbagai parameter. Judul aplikasi diatur menjadi "Quick Note" dan tema ditentukan menggunakan `ThemeData`, yang memanfaatkan skema warna yang dihasilkan dari warna ungu tua (deep purple). Halaman awal aplikasi diatur menjadi `PushremoveuntilNavigationPage`, yang merupakan widget bertipe `StatefulWidget`. Dengan menggunakan `StatefulWidget`, halaman ini dapat mengelola status yang dapat berubah dan memungkinkan pembaruan tampilan berdasarkan interaksi pengguna. Dalam kelas `PushremoveuntilNavigationPage`, ada `_PushremoveuntilNavigationPageState` yang bertanggung jawab untuk mengelola state halaman tersebut.

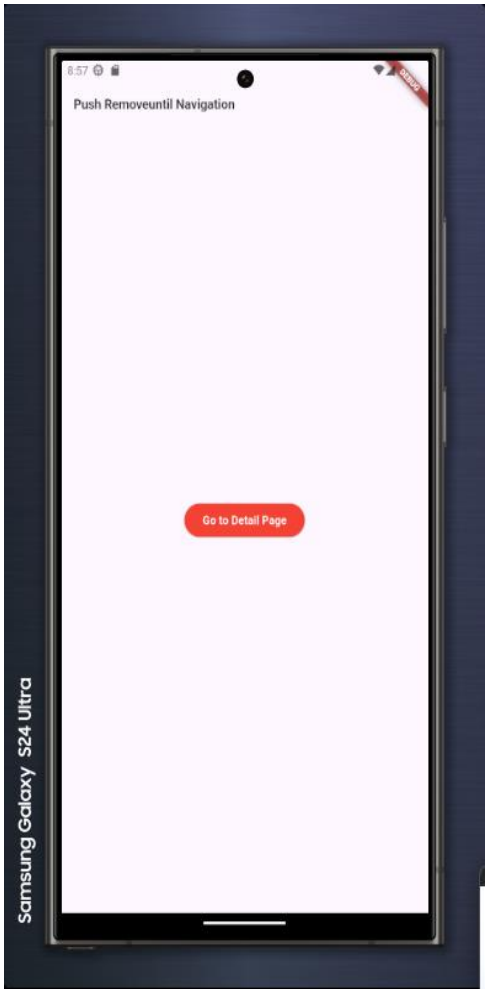
Di dalam metode `build()` dari `_PushremoveuntilNavigationPageState`, struktur halaman disusun menggunakan `Scaffold`, yang merupakan struktur dasar untuk aplikasi Flutter. `AppBar` ditambahkan dengan judul "Push Removeuntil Navigation" yang ditampilkan dengan ukuran dan bobot font tertentu. Selanjutnya, dalam bagian tubuh halaman, `SafeArea` digunakan untuk memastikan bahwa elemen antarmuka tidak terhalang oleh bagian-bagian layar yang sensitif, seperti notch atau sistem status. Di dalam `SafeArea`, terdapat widget `Center` yang mengatur agar elemen di dalamnya terletak di tengah halaman. Di dalam widget `Center`, ada tombol `ElevatedButton` yang dilabeli "Go to Detail Page".

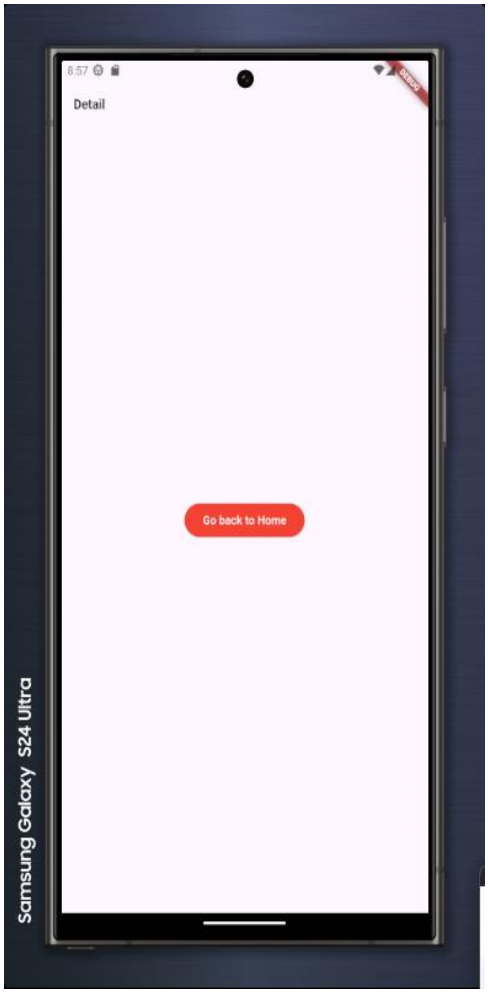
Ketika tombol ditekan, fungsi yang ditetapkan dalam parameter `onPressed` akan dieksekusi. Dalam fungsi ini, `Navigator.pushAndRemoveUntil` digunakan untuk berpindah ke halaman baru. Metode ini tidak hanya berpindah ke halaman baru, tetapi juga menghapus semua rute sebelumnya dari tumpukan navigasi. Dalam hal ini, halaman detail yang baru dibangun adalah `DetailPage`. Parameter ketiga dari

metode ini adalah fungsi yang mengembalikan nilai boolean. Dalam contoh ini, fungsi yang digunakan adalah `(route) => false`, yang berarti bahwa semua rute sebelumnya akan dihapus dari tumpukan navigasi, sehingga saat pengguna menekan tombol kembali, mereka tidak akan kembali ke halaman sebelumnya. Hal ini memberikan pengalaman navigasi yang bersih dan fokus pada halaman detail.

Secara keseluruhan, kode ini menunjukkan cara kerja Flutter dalam membangun aplikasi dengan navigasi yang efektif. Dengan menggunakan metode `push and remove until`, aplikasi dapat dengan mudah berpindah ke halaman detail tanpa membiarkan pengguna kembali ke halaman sebelumnya, menciptakan alur navigasi yang jelas dan langsung. Penggunaan `MaterialApp`, `Scaffold`, dan elemen lainnya menunjukkan betapa fleksibelnya Flutter dalam menciptakan antarmuka pengguna yang responsif dan dinamis, sesuai dengan kebutuhan aplikasi modern.

- Hasil





- Named Routes
  - Kode Pemrograman

```
import 'package:flutter/material.dart';
import 'package:flutter_widget/detail.dart';
import 'package:flutter_widget/home.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Quick Note',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor:
Colors.deepPurple),
        useMaterial3: true,
```

```
    ),
    initialRoute: '/',
    routes: {
      '/': (context) => const HomePage(),
      '/detail': (context) => const DetailPage()
    }
  );
 }
}
```

- Penjelasan Kode

Kodingan di atas merupakan implementasi sederhana dari aplikasi Flutter yang menggunakan navigasi berbasis rute. Program ini dimulai dengan fungsi `main()`, yang berfungsi sebagai titik masuk aplikasi. Di dalam fungsi ini, `runApp()` dipanggil untuk menjalankan widget `MyApp`, yang merupakan struktur dasar dari aplikasi ini. Kelas `MyApp` adalah turunan dari `StatelessWidget`, yang menunjukkan bahwa widget ini tidak memiliki status yang dapat berubah dan berfungsi untuk merender antarmuka pengguna.

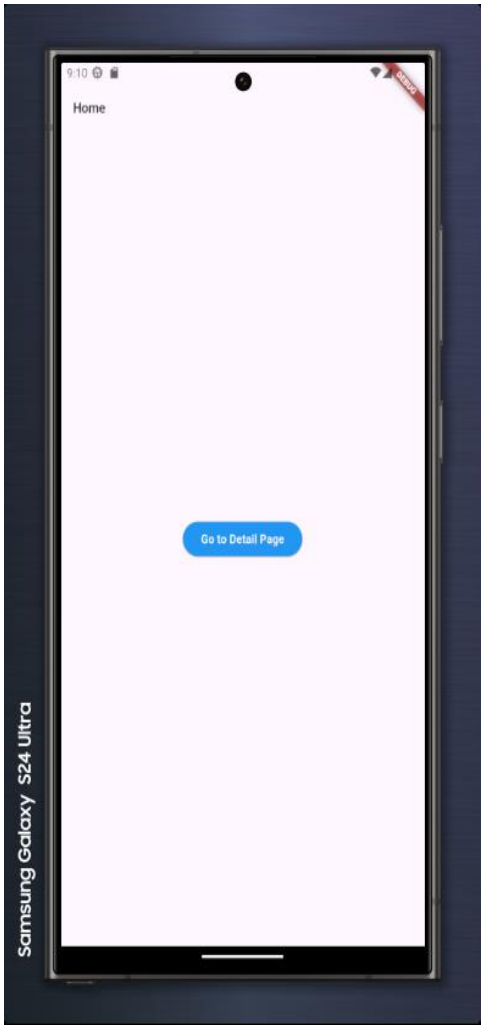
Di dalam metode `build()` dari kelas `MyApp`, aplikasi diatur menggunakan `MaterialApp`. Parameter pertama yang diatur adalah judul aplikasi, yang diset menjadi "Quick Note". Tema untuk aplikasi ditentukan dengan menggunakan `ThemeData`, yang dalam hal ini menggunakan skema warna yang dihasilkan dari warna ungu tua (deep purple). Penggunaan `useMaterial3: true` menunjukkan bahwa aplikasi ini mengikuti desain Material terbaru. Selain itu, `initialRoute` diatur ke `'/'`, yang menunjukkan bahwa halaman awal aplikasi adalah rute yang berasosiasi dengan `'/'`.

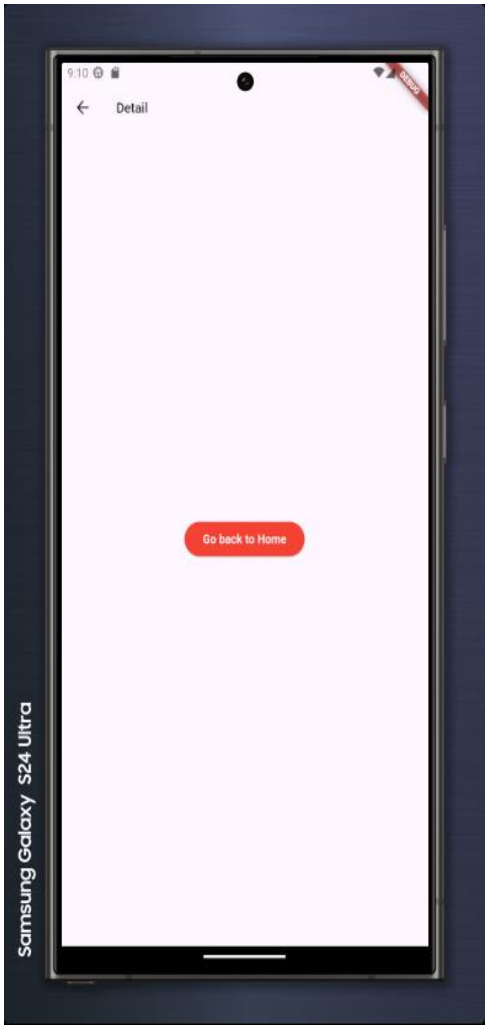
Di dalam parameter `routes`, terdapat pemetaan antara rute dan widget yang sesuai. Rute `'/'` dipetakan ke widget `HomePage`, yang menunjukkan bahwa ketika aplikasi diluncurkan, pengguna akan melihat halaman beranda. Rute `'/detail'` diatur untuk merender `DetailPage`, yang merupakan halaman yang akan ditampilkan ketika pengguna melakukan navigasi ke halaman detail. Penggunaan pendekatan berbasis rute ini membuatnya lebih mudah untuk menambahkan lebih banyak halaman dan manajemen navigasi, karena semua rute didefinisikan dalam satu tempat.

Ketika pengguna membuka aplikasi, mereka akan langsung diarahkan ke halaman beranda (`HomePage`) karena pengaturan `initialRoute`. Jika ada interaksi dari pengguna, seperti menekan tombol atau elemen yang terhubung

dengan rute `'/detail'`, aplikasi akan menggunakan mekanisme navigasi yang telah didefinisikan untuk berpindah ke halaman detail. Secara keseluruhan, struktur ini memberikan dasar yang kuat untuk aplikasi yang dapat diperluas dan memudahkan pengelolaan rute, menjadikan navigasi lebih terorganisir dan efisien.

- Hasil





REFERENSI

https://buildwithangga.com/tips/cara-menggunakan-widget-stack-pada-flutter

https://buildwithangga.com/tips/memahami-penggunaan-widget-padding-untuk-tata-letak-yang-lebih-baik-di-flutter

https://medium.com/@yousafjamil50/align-widget-in-flutter-ff6397ee712c

https://buildwithangga.com/tips/mengenal-dan-cara-menggunakan-widget-elevatedbutton-flutter

https://medium.com/@hamidrezadeveloper/a-comprehensive-guide-to-text-field-in-flutter-9abb3d85a1fb

https://www.dhiwise.com/post/implementing-network-image-in-flutter-a-step-by-step-guide

https://buildwithangga.com/tips/tips-dan-trik-menampilkan-gambar-dengan-image-widget-di-flutter

https://medium.com/@yahyaa.ozturrk/9-creative-ways-to-use-the-flutter-container-widget-420bc9eca436

https://buildwithangga.com/tips/membuat-tampilan-yang-menarik-dengan-icon-widget-dalam-flutter

https://buildwithangga.com/tips/mengenal-listview-widget-untuk-menampilkan-daftar-item-pada-flutter

https://karthikponnam.medium.com/gridviews-in-flutter-cadbff12f47c

https://medium.com/@chetan.akarte/explain-the-flutter-navigator-widget-and-its-methods-0f6f5023ff0c

https://samuelwahome.medium.com/navigation-and-named-routes-in-flutter-2d31d0975498