LAPORAN PRAKTIKUM
MENGENAL WIDGET FLUTTER 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 DAN PRAKTIKUM

1.1. Teori Dasar

1.1.1. Flutter Widget

Flutter widget adalah elemen dasar dalam framework Flutter yang digunakan untuk membangun antarmuka pengguna (UI). Dalam Flutter, segala sesuatu yang ditampilkan pada layar, baik itu teks, gambar, tombol, atau bahkan struktur tata letak yang kompleks, merupakan sebuah widget. Widget di Flutter tidak hanya mencakup elemen visual, tetapi juga elemen yang tidak terlihat namun mengatur bagaimana elemen-elemen lain ditampilkan atau berfungsi, seperti tata letak atau margin. Setiap komponen dalam aplikasi Flutter dibangun dari kumpulan widget yang bekerja secara hierarkis. Hierarki ini dikenal dengan istilah widget tree atau pohon widget, di mana widget induk (parent) dapat memiliki widget anak (child), dan setiap widget dapat mengatur perilaku dan tampilan anak-anaknya. Flutter menggunakan pendekatan deklaratif untuk mendefinisikan UI. Ini berarti bahwa pengembang mendeklarasikan tampilan aplikasi berdasarkan state (kondisi) saat ini, dan Flutter akan secara otomatis memperbarui tampilan ketika state tersebut berubah. Berbeda dengan pendekatan imperatif yang sering digunakan di framework lain, pendekatan deklaratif memungkinkan pengembang untuk fokus pada hasil akhir yang ingin dicapai daripada mengelola proses langkah demi langkah.

Fungsi utama dari Flutter widget adalah membangun antarmuka pengguna aplikasi, tetapi fungsi widget jauh lebih luas daripada sekadar elemen visual. Flutter widget dapat berfungsi untuk mengatur tata letak, menangani interaksi pengguna, dan bahkan mengatur animasi. Pertama, widget berperan sebagai elemen visual yang ditampilkan kepada pengguna. Elemen-elemen seperti teks (Text), gambar (Image), dan tombol (RaisedButton, FlatButton) semuanya dibangun menggunakan widget. Pengembang dapat dengan mudah mengatur properti dari widget ini, seperti ukuran, warna, dan gaya, untuk menyesuaikan tampilan sesuai kebutuhan. Kedua, widget juga berfungsi sebagai elemen tata letak (layout). Beberapa widget dirancang untuk mengatur posisi komponen-komponen lain pada layar. Contohnya adalah widget seperti Row dan Column, yang memungkinkan pengembang mengatur posisi komponen secara horizontal atau vertikal, serta widget Stack, yang memungkinkan

komponen ditempatkan secara bertumpuk. Tata letak ini memungkinkan pengembang untuk menciptakan UI yang kompleks dan responsif.

Selain itu, widget juga bertanggung jawab untuk menangani interaksi pengguna. Flutter menyediakan berbagai widget interaktif yang dapat merespons input pengguna, seperti sentuhan, klik, atau gesekan. Widget seperti GestureDetector atau InkWell memungkinkan pengembang mendefinisikan perilaku aplikasi ketika pengguna berinteraksi dengan elemen UI. Misalnya, ketika pengguna mengetuk sebuah tombol, widget tersebut dapat merespon dengan mengubah state aplikasi atau memicu aksi tertentu. Flutter juga menyediakan widget yang mendukung animasi dan transisi. Animasi yang halus dan transisi antara berbagai tampilan UI dapat dibuat dengan mudah menggunakan widget seperti AnimatedContainer, Hero, dan Opacity. Hal ini memungkinkan pengembang menciptakan pengalaman pengguna yang lebih dinamis dan menarik.

1.1.2.MaterialApp

MaterialApp adalah widget utama dalam framework Flutter yang digunakan untuk menginisialisasi dan mengelola aplikasi yang mengikuti standar Material Design. Material Design adalah pendekatan desain yang dikembangkan oleh Google dengan tujuan menciptakan UI yang konsisten, responsif, dan mudah digunakan di berbagai perangkat. MaterialApp menjadi inti dari aplikasi Flutter karena menyediakan fondasi utama untuk membangun aplikasi yang mendukung komponen-komponen standar seperti navigasi, tema, pengaturan rute, dan tata letak.Pada dasarnya, ketika membangun aplikasi Flutter, hampir setiap aplikasi yang menerapkan Material Design dimulai dengan widget MaterialApp. Widget ini berfungsi sebagai pengatur utama dari berbagai elemen UI dan logika aplikasi, sekaligus memastikan bahwa seluruh komponen UI mematuhi aturan dan gaya desain dari Material Design. Dengan kata lain, MaterialApp adalah pembungkus (wrapper) yang menyediakan semua pengaturan global aplikasi, termasuk pengaturan tampilan dan navigasi. Selain itu, MaterialApp juga bertanggung jawab untuk mengelola lifecycle aplikasi dan berinteraksi dengan platform Android dan iOS. Ini mencakup pengelolaan transisi antar halaman, pengaturan ikon aplikasi, title, serta fitur yang mendukung localization dan berbagai properti lainnya. Dengan memanfaatkan MaterialApp, pengembang dapat menciptakan aplikasi Flutter yang terlihat profesional, modern, dan konsisten dengan antarmuka aplikasi Android.

MaterialApp memiliki beberapa fungsi utama yang sangat penting dalam pengembangan aplikasi Flutter. Pertama, MaterialApp menyediakan navigasi dan routing antar halaman dalam aplikasi. Dengan menggunakan properti routes dan onGenerateRoute, pengembang dapat mendefinisikan rute yang memungkinkan pengguna berpindah dari satu halaman ke halaman lain secara mulus. Ini membantu menciptakan aplikasi yang terstruktur dan mudah digunakan oleh pengguna. Selain itu, MaterialApp juga mendukung pengaturan tema global untuk aplikasi. Pengembang dapat menentukan warna utama, warna aksen, gaya teks, serta elemen visual lainnya melalui properti theme. Flutter juga menyediakan opsi untuk mendukung tema gelap (darkTheme), yang memungkinkan aplikasi beradaptasi dengan preferensi pengguna terkait mode tampilan gelap atau terang. Pengaturan ini memastikan tampilan aplikasi konsisten di seluruh komponen tanpa perlu mengatur elemen UI secara individual. MaterialApp juga mendukung localization, yang memungkinkan aplikasi untuk menampilkan teks dalam berbagai bahasa berdasarkan pengaturan perangkat pengguna. Dengan fitur ini, aplikasi dapat dengan mudah disesuaikan untuk audiens global. Selain itu, MaterialApp menyediakan pengelolaan status aplikasi dan kemampuan untuk memantau perubahan halaman menggunakan navigatorObservers. Hal ini penting untuk pengumpulan data analitik atau pemantauan aktivitas pengguna di dalam aplikasi.

1.1.3. Scaffold

`Scaffold` adalah widget dasar dalam Flutter yang digunakan untuk memberikan struktur dan tata letak halaman dengan gaya Material Design. Widget ini berfungsi sebagai kerangka atau pembungkus utama yang memungkinkan pengembang membangun antarmuka pengguna dengan elemen-elemen yang lazim dalam aplikasi modern, seperti AppBar, Drawer, BottomNavigationBar, FloatingActionButton, dan SnackBar. Dengan menggunakan `Scaffold`, pengembang dapat dengan mudah mengelola tata letak dan fitur interaksi yang sering ditemukan dalam aplikasi Android dan iOS yang mengikuti standar Material Design. Pada intinya, `Scaffold` memberikan dasar untuk menambahkan elemen-elemen penting aplikasi yang dibutuhkan di setiap halaman. Sebagai contoh, `AppBar` digunakan untuk menampilkan bar judul di bagian atas halaman, `Drawer` berfungsi sebagai panel navigasi samping, sedangkan `BottomNavigationBar` memfasilitasi navigasi antar halaman di bagian bawah. Semua komponen ini diatur dalam `Scaffold`, sehingga

memudahkan pengembang dalam menciptakan aplikasi yang konsisten dan responsif. Dengan adanya `Scaffold`, pengembang tidak perlu lagi membangun tata letak dasar secara manual setiap kali membuat halaman baru dalam aplikasi.

Selain itu, `Scaffold` bertanggung jawab untuk memfasilitasi berbagai interaksi dan animasi UI. Misalnya, pengguna dapat membuka drawer (panel samping) dengan menggesek layar atau dengan mengklik ikon di AppBar. Elemen seperti FloatingActionButton (FAB) ditempatkan dengan mudah di dalam `Scaffold`, memberikan pengguna akses cepat ke tindakan utama pada halaman tertentu. `Scaffold` juga menyediakan fitur untuk menampilkan pesan notifikasi sementara melalui SnackBar, yang merupakan elemen penting dalam memberikan feedback kepada pengguna setelah melakukan suatu aksi. `Scaffold` memiliki beberapa fungsi penting yang mempermudah pengaturan UI dalam aplikasi Flutter. Salah satu fungsi utamanya adalah menyediakan wadah untuk elemen UI utama seperti AppBar, body, dan berbagai komponen lainnya. Dengan menggunakan properti `body`, pengembang dapat menambahkan widget utama yang akan ditampilkan di tengah halaman, seperti teks, gambar, atau layout yang lebih kompleks menggunakan widget `Column` atau `Row`. Dengan adanya `Scaffold`, elemen-elemen ini diletakkan secara otomatis sesuai dengan tata letak yang telah diatur oleh framework Material Design, memastikan konsistensi tampilan dan interaksi antar halaman.

Selain itu, `Scaffold` memfasilitasi pengelolaan interaksi pengguna yang intuitif. Misalnya, dengan properti `floatingActionButton`, pengembang dapat dengan mudah menempatkan tombol aksi mengambang (FAB) yang sering digunakan untuk tindakan penting, seperti menambah item baru atau mengakses halaman tertentu. Properti `bottomNavigationBar` juga mempermudah pengaturan navigasi di bagian bawah layar, memungkinkan pengguna untuk berpindah halaman dengan mudah. Fitur ini sangat membantu dalam menciptakan aplikasi yang ramah pengguna, terutama untuk aplikasi dengan banyak halaman atau fitur. Dengan `Scaffold`, pengembang juga dapat menambahkan elemen-elemen yang muncul sementara di layar, seperti SnackBar untuk memberikan umpan balik kepada pengguna setelah aksi dilakukan. `SnackBar` dapat ditampilkan secara otomatis di bagian bawah layar tanpa harus mengganggu tampilan utama aplikasi. Elemen lain seperti `Drawer` juga dapat ditambahkan melalui `Scaffold`, memberikan pengguna akses ke menu navigasi tambahan dengan mudah melalui panel samping. Secara keseluruhan, `Scaffold` adalah komponen penting dalam membangun aplikasi Flutter yang mengikuti

pedoman Material Design. Dengan `Scaffold`, pengembang dapat membuat halaman aplikasi yang lengkap dengan elemen UI yang khas dan fungsional, seperti AppBar, Drawer, FloatingActionButton, dan BottomNavigationBar. Fitur-fitur ini membuat `Scaffold` sangat fleksibel dan esensial dalam menciptakan aplikasi yang modern, intuitif, dan mudah dinavigasi.

1.1.4. Stateless Widget

`StatelessWidget` adalah salah satu jenis widget dasar dalam framework Flutter yang digunakan untuk membuat tampilan yang tidak memiliki state atau kondisi yang dapat berubah selama masa hidup widget tersebut. Sebagai bagian dari pendekatan deklaratif Flutter, `StatelessWidget` digunakan ketika UI aplikasi tidak perlu memperbarui tampilannya secara dinamis berdasarkan interaksi pengguna atau perubahan data. Dengan kata lain, `StatelessWidget` hanya menampilkan konten statis yang diinisialisasi satu kali dan tidak berubah lagi kecuali widget tersebut dibuat ulang. Pada dasarnya, `StatelessWidget` sering digunakan untuk membuat komponen UI yang tidak memerlukan penyimpanan atau pengelolaan data internal. Contoh umum dari penggunaan `StatelessWidget` adalah menampilkan teks, gambar, atau layout yang sederhana. Karena tidak menyimpan state, `StatelessWidget` bersifat ringan dan efisien, membuatnya cocok untuk komponen yang tidak memerlukan pembaruan UI secara terus-menerus. Setiap kali ada perubahan kecil pada data yang harus diubah, seluruh widget perlu dibangun ulang, karena tidak ada mekanisme internal untuk mengelola perubahan data secara langsung. Flutter mengandalkan pendekatan deklaratif untuk membangun antarmuka pengguna, di mana UI digambarkan sebagai serangkaian widget yang membentuk hirarki tampilan. `StatelessWidget` berfungsi sebagai elemen UI yang tetap, di mana widget ini hanya membangun ulang dirinya jika ada perubahan dari luar (misalnya, data dari parent widget). Dengan kata lain, `StatelessWidget` bekerja dengan menerima input melalui constructor dan membangun UI berdasarkan data tersebut, tetapi tidak memiliki kemampuan untuk mengubah UI secara mandiri setelah di-render pertama kali.

`StatelessWidget` memiliki beberapa fungsi penting dalam pengembangan aplikasi Flutter. Salah satu fungsi utamanya adalah memberikan cara yang sederhana dan efisien untuk membuat tampilan statis yang tidak perlu dikelola lebih lanjut. Misalnya, jika pengembang ingin menampilkan teks yang tidak berubah, seperti judul aplikasi atau gambar tetap, `StatelessWidget` adalah pilihan yang ideal. Selain itu,

`StatelessWidget` juga sangat cocok digunakan dalam komponen UI yang merupakan bagian dari halaman atau layout yang lebih kompleks, tetapi tidak memerlukan interaksi pengguna yang dinamis. Fungsi lain dari `StatelessWidget` adalah untuk membantu menjaga struktur aplikasi tetap terorganisir dan modular. Karena `StatelessWidget` sangat sederhana dan tidak memiliki logika yang kompleks terkait state, komponen ini dapat dengan mudah dipecah menjadi bagian-bagian kecil yang digunakan kembali di berbagai tempat dalam aplikasi. Hal ini membuat kode lebih mudah dibaca, dikelola, dan di-debug. Pengembang dapat membagi tampilan menjadi beberapa `StatelessWidget` kecil, yang masing-masing menangani bagian tertentu dari UI, sehingga menciptakan struktur aplikasi yang modular dan dapat diperluas. Selain itu, `StatelessWidget` juga memiliki keuntungan dari sisi kinerja, karena tidak perlu melacak perubahan state. Flutter hanya perlu me-render ulang widget jika terjadi perubahan pada parent widget atau pada data yang diberikan melalui constructor. Ini berbeda dengan `StatefulWidget` yang memerlukan manajemen state dan sering kali memicu pembaruan UI. Oleh karena itu, untuk komponen yang tidak memerlukan pembaruan secara dinamis, `StatelessWidget` lebih ringan dan lebih cepat.

Dalam praktiknya, pengembang sering memulai dengan `StatelessWidget` ketika merancang UI, dan hanya beralih ke `StatefulWidget` jika komponen tersebut membutuhkan manajemen state yang kompleks. Sebagai contoh, jika pengguna mengklik tombol yang menyebabkan perubahan pada tampilan UI, atau jika data dinamis perlu diubah secara real-time, pengembang akan menggunakan `StatefulWidget`. Namun, jika tampilan tetap statis tanpa memerlukan interaksi dinamis, `StatelessWidget` memberikan solusi yang lebih sederhana dan efisien. Secara keseluruhan, `StatelessWidget` adalah elemen penting dalam Flutter yang memungkinkan pengembang membangun UI yang bersifat statis dan tidak memerlukan manajemen state. Dengan menggunakan `StatelessWidget`, pengembang dapat menciptakan komponen UI yang efisien, modular, dan mudah dikelola. Fleksibilitas dan kesederhanaan `StatelessWidget` menjadikannya pilihan yang ideal untuk menampilkan konten tetap dalam aplikasi Flutter yang mengikuti pendekatan Material Design.

1.1.5. Stateful Widget

`StatefulWidget` adalah salah satu widget dasar dalam Flutter yang digunakan untuk membangun UI yang membutuhkan pembaruan secara dinamis berdasarkan perubahan state atau kondisi aplikasi. Berbeda dengan `StatelessWidget`, yang menampilkan UI statis, `StatefulWidget` memungkinkan UI berubah seiring dengan interaksi pengguna atau perubahan data internal. `StatefulWidget` terdiri dari dua bagian utama, yaitu widget itu sendiri dan sebuah kelas state terpisah yang mengelola logika dan data dinamis yang dibutuhkan oleh widget tersebut. Secara umum, `StatefulWidget` digunakan ketika aplikasi memerlukan pengelolaan state yang bisa berubah secara berkala atau berdasarkan tindakan tertentu. Misalnya, aplikasi yang memiliki tombol yang dapat diklik untuk mengubah warna layar, daftar yang dapat diperbarui secara real-time, atau bahkan input form yang memerlukan validasi saat pengguna mengetikkan teks. `StatefulWidget` bekerja dengan cara merender ulang dirinya ketika ada perubahan state, sehingga aplikasi dapat menampilkan pembaruan secara langsung kepada pengguna. Salah satu elemen kunci dalam `StatefulWidget` adalah adanya kelas `State` yang terpisah, di mana logika dan pengelolaan state berada. Setiap kali state berubah, metode `setState()` dipanggil, yang memberitahukan Flutter untuk merender ulang UI. Mekanisme ini memungkinkan `StatefulWidget` untuk merespon dengan cepat terhadap interaksi pengguna atau perubahan data. Meskipun demikian, hanya bagian UI yang terkait dengan state yang akan dirender ulang, bukan seluruh aplikasi, yang memastikan efisiensi kinerja.

`StatefulWidget` memiliki fungsi utama dalam mengelola tampilan yang interaktif dan dinamis. Salah satu fungsinya adalah memungkinkan UI aplikasi untuk memperbarui diri secara otomatis tanpa harus merender ulang seluruh komponen UI. Misalnya, jika aplikasi menampilkan daftar item yang diambil dari API, `StatefulWidget` dapat menampilkan perubahan ketika data baru tiba, atau jika ada item yang ditambahkan atau dihapus. Hal ini sangat penting untuk menciptakan pengalaman pengguna yang responsif dan real-time. Selain itu, `StatefulWidget` juga memungkinkan pengelolaan event pengguna seperti klik tombol, input teks, dan navigasi antar halaman. Misalnya, jika pengguna mengklik tombol, `StatefulWidget` dapat mengubah tampilan atau mengupdate data berdasarkan aksi tersebut. Dalam hal ini, state yang dikelola bisa berupa nilai boolean, string, integer, atau bahkan data kompleks yang terkait dengan interaksi pengguna. Dengan kemampuan untuk mengelola event, `StatefulWidget` menjadi fondasi utama untuk fitur interaktif yang

dinamis dalam aplikasi Flutter. Fungsi lain dari `StatefulWidget` adalah memungkinkan pengelolaan lifecycle dari state dan tampilan UI secara lebih fleksibel. Flutter memberikan beberapa metode yang dapat digunakan dalam kelas `State` untuk menangani perubahan lifecycle, seperti `initState()`, `dispose()`, dan `didUpdateWidget()`. Metode `initState()` misalnya, digunakan untuk melakukan inisialisasi awal saat widget pertama kali dibuat, sedangkan `dispose()` digunakan untuk membersihkan resource saat widget dihapus dari pohon widget. Fitur ini memberikan kontrol penuh kepada pengembang untuk mengatur bagaimana state harus dikelola selama siklus hidup widget, terutama saat bekerja dengan resource eksternal seperti jaringan atau animasi. Keuntungan lain dari `StatefulWidget` adalah kemampuannya untuk mendukung interaksi pengguna secara langsung dan merespons input dengan cepat. Misalnya, jika pengguna mengisi form atau melakukan gesekan di layar, `StatefulWidget` dapat dengan mudah memperbarui tampilan UI berdasarkan input tersebut. Proses ini sangat penting untuk membangun antarmuka yang terasa mulus dan responsif, terutama dalam aplikasi modern yang mengutamakan pengalaman pengguna.

Dalam praktiknya, pengembang sering kali menggunakan `StatefulWidget` ketika aplikasi membutuhkan logika yang kompleks atau data yang berubah-ubah secara dinamis. Contoh penggunaan umum adalah aplikasi chat di mana pesan terus diperbarui, aplikasi e-commerce dengan keranjang belanja yang bisa ditambah atau dikurangi item, atau aplikasi game di mana skor dan level terus diperbarui berdasarkan aksi pengguna. `StatefulWidget` memungkinkan semua ini terjadi tanpa perlu merender ulang seluruh UI secara manual. Secara keseluruhan, `StatefulWidget` memainkan peran penting dalam pengembangan aplikasi Flutter yang dinamis dan interaktif. Dengan kemampuannya untuk mengelola dan memperbarui state secara efisien, `StatefulWidget` memungkinkan pengembang untuk membangun aplikasi yang responsif terhadap perubahan data dan interaksi pengguna. Ini menjadikannya elemen fundamental dalam menciptakan aplikasi Flutter yang modern, dinamis, dan ramah pengguna.

1.1.6. Layout Widget

Layout Widget adalah serangkaian widget dalam Flutter yang digunakan untuk mengatur tata letak atau struktur elemen UI pada sebuah aplikasi. Dalam Flutter, setiap elemen UI seperti teks, gambar, atau tombol diwakili oleh widget, dan untuk

mengatur bagaimana widget tersebut ditampilkan dalam layar, kita memerlukan widget layout. Layout widget memungkinkan pengembang untuk membuat struktur tampilan yang kompleks, baik dalam satu dimensi (horizontal atau vertikal) maupun dalam beberapa dimensi, dengan kontrol yang sangat fleksibel terhadap posisi, ukuran, dan ruang antar elemen. Pada dasarnya, Layout Widget dalam Flutter digunakan untuk mengatur bagaimana widget lain ditempatkan dan ditampilkan dalam antarmuka pengguna. Layout ini sangat penting dalam pengembangan aplikasi karena menentukan bagaimana konten aplikasi diatur dalam layar perangkat, terlepas dari ukuran layar atau orientasinya. Dengan menggunakan Layout Widget, pengembang dapat membuat tampilan yang adaptif dan responsif, sehingga antarmuka aplikasi dapat menyesuaikan dengan berbagai ukuran layar dan perangkat, mulai dari smartphone hingga tablet. Flutter menyediakan berbagai macam Layout Widget yang memungkinkan pengembang untuk mengatur tampilan dengan cara yang efisien dan mudah dikelola. Misalnya, widget seperti `Row` dan `Column` adalah widget dasar yang mengatur tata letak elemen secara horizontal dan vertikal. `Row` memungkinkan penempatan widget dalam baris yang sejajar secara horizontal, sedangkan `Column` mengatur elemen secara vertikal dalam satu kolom. Kedua widget ini adalah dasar dari banyak tata letak dalam aplikasi, dan mereka mendukung alignment serta pengaturan jarak antar elemen melalui properti seperti `mainAxisAlignment` dan `crossAxisAlignment`.

Selain widget dasar seperti `Row` dan `Column`, Flutter juga menyediakan widget layout yang lebih kompleks seperti `Stack` dan `GridView`. `Stack` memungkinkan pengembang untuk menempatkan widget di atas satu sama lain dalam lapisan, yang sangat berguna untuk membuat tata letak yang lebih dinamis dan kreatif. Misalnya, dengan `Stack`, pengembang dapat menempatkan teks di atas gambar atau membuat elemen UI yang saling tumpang tindih. Sementara itu, `GridView` memungkinkan pengaturan elemen dalam bentuk grid, yang sangat cocok untuk menampilkan elemen dalam jumlah besar, seperti daftar gambar atau kartu. Fungsi utama dari Layout Widget adalah memungkinkan pengembang untuk mengatur posisi dan ukuran elemen UI secara responsif dan fleksibel. Dengan menggunakan widget layout, pengembang dapat menentukan bagaimana elemen UI harus menyesuaikan diri ketika ukuran layar berubah atau ketika aplikasi dibuka di perangkat dengan resolusi yang berbeda. Layout widget seperti `Expanded` dan `Flexible` membantu dalam mengatur bagaimana ruang di dalam tata letak harus dibagi di antara elemen-

elemen yang ada, memberikan kendali penuh kepada pengembang atas bagaimana setiap elemen mengambil ruang di dalam layar. Selain itu, Layout Widget juga mendukung pengaturan responsivitas aplikasi, yang merupakan fitur penting dalam desain aplikasi modern. Misalnya, dengan menggunakan widget seperti `Flexible`, pengembang dapat membuat elemen UI yang secara otomatis menyesuaikan ukuran dan posisinya tergantung pada ruang yang tersedia. Widget `Container` juga sering digunakan dalam layout untuk membungkus elemen lain dan memberikan properti seperti margin, padding, warna, dan ukuran yang lebih terkontrol. Dengan memanfaatkan berbagai widget layout ini, pengembang dapat memastikan bahwa tampilan aplikasi tetap terlihat baik di berbagai ukuran dan resolusi layar.

Keuntungan utama dari Layout Widget dalam Flutter adalah fleksibilitas dan kemudahan penggunaannya. Flutter menggunakan pendekatan deklaratif untuk mendefinisikan UI, di mana pengembang hanya perlu mendefinisikan bagaimana tata letak harus terlihat, dan Flutter akan secara otomatis mengatur tampilan berdasarkan definisi tersebut. Hal ini membuat proses pengembangan lebih cepat dan lebih efisien, karena pengembang tidak perlu khawatir tentang detail implementasi tata letak di berbagai perangkat. Selain itu, pengaturan tata letak dalam Flutter bersifat dinamis dan dapat diatur ulang dengan mudah, memungkinkan penyesuaian antarmuka secara real-time berdasarkan interaksi pengguna. Secara keseluruhan, Layout Widget dalam Flutter adalah fondasi dari bagaimana antarmuka pengguna diatur dan disusun. Dengan menggunakan berbagai widget layout yang disediakan oleh Flutter, pengembang dapat menciptakan tampilan yang responsif, adaptif, dan efisien, sesuai dengan kebutuhan aplikasi. Layout Widget memberikan kontrol penuh kepada pengembang untuk mengatur elemen UI dengan cara yang mudah dipahami dan dikelola, sambil memastikan bahwa aplikasi dapat menyesuaikan diri dengan berbagai ukuran layar dan perangkat tanpa mengorbankan estetika atau fungsionalitas.

1.1.7. Interactive Widget

Interactive Widget adalah jenis widget dalam framework Flutter yang memungkinkan interaksi langsung antara pengguna dan antarmuka aplikasi. Dalam pengembangan aplikasi modern, kemampuan untuk berinteraksi dengan pengguna adalah salah satu aspek yang sangat penting. Interactive Widget berfungsi sebagai elemen yang merespons tindakan pengguna, seperti mengetuk, menggeser, atau

menahan, sehingga menciptakan pengalaman pengguna yang dinamis dan responsif. Flutter menyediakan berbagai widget interaktif yang dirancang untuk berfungsi dengan mulus di berbagai platform, baik Android maupun iOS. Pada dasarnya, Interactive Widget dalam Flutter digunakan untuk menangani input pengguna dan memberikan umpan balik langsung. Widget ini memungkinkan pengguna untuk melakukan berbagai tindakan seperti menekan tombol, mengisi formulir, memilih opsi dari daftar, atau bahkan melakukan navigasi antar halaman. Salah satu contoh paling dasar dari Interactive Widget adalah `RaisedButton` atau sekarang lebih dikenal sebagai `ElevatedButton`. Widget ini digunakan untuk membuat tombol yang dapat ditekan oleh pengguna, di mana tindakan tertentu dapat dipicu saat tombol ditekan. Dalam banyak kasus, tombol ini digunakan untuk mengeksekusi perintah atau mengarahkan pengguna ke halaman lain dalam aplikasi. Fungsi utama dari Interactive Widget adalah untuk menyediakan antarmuka yang dapat merespons aksi pengguna secara real-time. Widget seperti `TextField` memungkinkan pengguna untuk memasukkan data atau teks, dan input ini dapat diproses secara langsung oleh aplikasi. Misalnya, ketika pengguna mengisi formulir login, `TextField` digunakan untuk menangani input seperti nama pengguna dan kata sandi. Widget ini juga dapat memvalidasi input secara langsung, memberikan umpan balik jika ada kesalahan, seperti teks kosong atau format input yang tidak sesuai. Flutter juga menyediakan berbagai opsi untuk mengatur tampilan dan perilaku widget interaktif, seperti menyesuaikan gaya teks, batas, dan warna dari input field.

Selain itu, Flutter juga mendukung berbagai jenis widget interaktif lainnya yang lebih kompleks, seperti `Checkbox`, `Switch`, dan `Slider`. `Checkbox` memungkinkan pengguna untuk memilih atau membatalkan pilihan, yang sering digunakan dalam formulir untuk persetujuan atau pemilihan opsi. `Switch` adalah widget yang digunakan untuk mengaktifkan atau menonaktifkan opsi dengan cara yang sederhana dan visual, sering digunakan untuk pengaturan aplikasi seperti mengaktifkan notifikasi atau mode gelap. Sementara itu, `Slider` digunakan untuk mengatur nilai dalam rentang tertentu, seperti mengatur volume suara atau kecerahan layar. Semua widget ini memberikan pengguna kontrol yang lebih besar terhadap antarmuka aplikasi, memungkinkan mereka untuk menyesuaikan aplikasi sesuai dengan preferensi pribadi mereka. Interaktivitas dalam aplikasi modern tidak hanya melibatkan input sederhana seperti mengetuk atau menggeser. Flutter juga mendukung gesture yang lebih kompleks melalui widget seperti `GestureDetector`.

`GestureDetector` memungkinkan pengembang untuk mendeteksi dan menangani berbagai jenis gerakan, seperti gesekan, cubitan, atau ketukan ganda. Widget ini sangat berguna dalam aplikasi yang memerlukan interaksi yang lebih canggih, seperti aplikasi permainan atau aplikasi dengan kontrol gestur khusus. Misalnya, pengguna dapat memperbesar gambar dengan mencubit layar atau menggulir melalui daftar item dengan menggeser jari mereka. Gesture dalam Flutter sangat fleksibel dan dapat disesuaikan dengan kebutuhan aplikasi.

Keuntungan utama dari Interactive Widget dalam Flutter adalah kemampuannya untuk menciptakan pengalaman pengguna yang responsif dan interaktif tanpa memerlukan banyak kode tambahan. Dengan menggunakan arsitektur widget Flutter yang berbasis deklaratif, pengembang dapat dengan mudah mendefinisikan perilaku interaktif secara langsung dalam kode UI. Selain itu, Flutter juga menyediakan dukungan penuh untuk animasi, sehingga interaksi dengan widget dapat dilengkapi dengan transisi visual yang halus dan menarik. Misalnya, ketika pengguna mengetuk tombol, aplikasi dapat menampilkan perubahan warna atau efek bayangan untuk menunjukkan bahwa tindakan telah diterima. Secara keseluruhan, Interactive Widget dalam Flutter adalah elemen penting yang memungkinkan aplikasi untuk merespons input pengguna dengan cepat dan efisien. Widget interaktif ini mendukung berbagai tindakan pengguna, mulai dari input teks hingga gesture kompleks, memberikan fleksibilitas yang luar biasa dalam hal desain antarmuka pengguna. Dengan Flutter, pengembang dapat menciptakan aplikasi yang responsif, interaktif, dan mudah digunakan di berbagai platform, memastikan bahwa aplikasi dapat memenuhi kebutuhan dan ekspektasi pengguna modern.

1.1.8. Display Widget

Display Widget dalam Flutter adalah kategori widget yang digunakan untuk menampilkan informasi atau konten visual kepada pengguna. Widget ini berfungsi sebagai elemen tampilan statis yang berfokus pada menampilkan data, teks, gambar, ikon, atau elemen visual lainnya di layar. Display Widget sangat penting dalam membangun antarmuka pengguna yang informatif, estetis, dan mudah dipahami. Flutter menyediakan berbagai widget yang berfungsi sebagai elemen display, dengan berbagai opsi untuk menyesuaikan tampilan agar sesuai dengan kebutuhan desain aplikasi. Salah satu contoh umum dari Display Widget adalah `Text`, yang digunakan untuk menampilkan teks statis dalam aplikasi. `Text` adalah widget dasar yang

memungkinkan pengembang untuk menampilkan teks dengan berbagai gaya, ukuran, dan warna. Dengan `Text`, pengembang dapat mengatur atribut seperti font, spasi, dan aligment (perataan) sehingga teks dapat disesuaikan dengan desain UI yang diinginkan. Selain itu, widget ini mendukung fitur penyesuaian seperti overflow handling, yang memungkinkan teks yang panjang untuk dipotong atau diperkecil agar sesuai dengan ruang tampilan yang tersedia. Selain `Text`, Flutter juga menyediakan Display Widget lain seperti `Image`. Widget `Image` digunakan untuk menampilkan gambar dari berbagai sumber, seperti file lokal, jaringan, atau penyimpanan internal. `Image` mendukung berbagai format gambar dan menyediakan fitur seperti penyesuaian ukuran otomatis, pengaturan skala, dan opsi untuk menangani gambar yang tidak tersedia. Fitur ini membuat `Image` menjadi pilihan yang fleksibel dan efisien dalam menampilkan visual grafis, baik dalam bentuk gambar statis maupun dinamis.

Display Widget lainnya yang sering digunakan adalah `Icon`. `Icon` biasanya digunakan untuk menampilkan ikon-ikon sederhana yang merepresentasikan aksi atau status tertentu dalam aplikasi. Flutter memiliki koleksi ikon bawaan yang luas melalui paket `Icons`, yang mencakup berbagai ikon yang sesuai dengan pedoman Material Design. `Icon` mendukung berbagai penyesuaian seperti warna, ukuran, dan posisi, sehingga bisa dengan mudah digunakan dalam tombol, menu, atau komponen lainnya. Ikon memainkan peran penting dalam meningkatkan keterbacaan antarmuka, memberikan panduan visual kepada pengguna tentang fungsi atau tindakan yang tersedia. Selain `Text`, `Image`, dan `Icon`, ada juga Display Widget yang lebih kompleks, seperti `ListView` dan `GridView`. `ListView` digunakan untuk menampilkan daftar item dalam bentuk scrollable (dapat digulir), yang cocok untuk menampilkan data dalam jumlah besar, seperti daftar kontak atau feed berita. `GridView` adalah variasi yang lebih terstruktur dari `ListView`, memungkinkan item-item ditampilkan dalam bentuk grid. Kedua widget ini mendukung penyesuaian yang tinggi dan dapat digunakan untuk membuat tampilan yang responsif dan adaptif pada berbagai ukuran layar. Fungsi utama dari Display Widget adalah untuk menyajikan informasi secara visual kepada pengguna. Widget-widget ini tidak memerlukan interaksi langsung dari pengguna, tetapi mereka memainkan peran penting dalam menyampaikan informasi atau mendukung antarmuka pengguna yang lebih luas. Dalam kombinasi dengan widget interaktif, Display Widget dapat digunakan untuk membangun antarmuka yang intuitif dan informatif. Misalnya,

dalam aplikasi e-commerce, Display Widget seperti `Image` digunakan untuk menampilkan gambar produk, sementara `Text` digunakan untuk menampilkan deskripsi dan harga produk. Keduanya bekerja sama untuk memberikan informasi yang lengkap dan mudah dipahami oleh pengguna.

Keuntungan utama dari Display Widget dalam Flutter adalah kemudahan penggunaan dan fleksibilitasnya. Widget-widget ini dapat dengan mudah disesuaikan dengan berbagai kebutuhan desain tanpa memerlukan banyak kode tambahan. Flutter juga mendukung animasi dan transisi yang halus, sehingga pengembang dapat meningkatkan tampilan visual aplikasi dengan efek yang menarik. Misalnya, gambar atau teks dapat diubah ukurannya atau dipindahkan secara dinamis ketika pengguna berinteraksi dengan elemen lain dalam aplikasi, memberikan pengalaman yang lebih menarik dan responsif. Secara keseluruhan, Display Widget dalam Flutter adalah elemen kunci dalam membangun antarmuka pengguna yang visual dan informatif. Dengan menggunakan berbagai widget tampilan yang disediakan oleh Flutter, pengembang dapat menciptakan UI yang kaya visual dan sesuai dengan pedoman desain modern, sekaligus memastikan bahwa aplikasi tetap responsif dan mudah digunakan di berbagai perangkat.

1.1.9. Animation Widget

Animation Widget dalam Flutter adalah elemen penting yang digunakan untuk menambahkan gerakan dan transisi yang halus dalam aplikasi. Animasi memainkan peran krusial dalam meningkatkan pengalaman pengguna dengan memberikan umpan balik visual yang lebih baik, menarik perhatian, dan menjadikan aplikasi lebih interaktif. Dengan menggunakan Animation Widget, pengembang dapat menciptakan antarmuka yang tidak hanya fungsional tetapi juga estetis, membuat aplikasi lebih menyenangkan untuk digunakan. Di Flutter, animasi dapat dibagi menjadi beberapa jenis, tetapi secara umum, semua animasi dikelola melalui Animation Controller. `AnimationController` adalah kelas yang mengatur durasi dan status animasi, memberikan kemudahan dalam mengontrol ketika animasi dimulai, dihentikan, atau diulang. Pengembang dapat mengatur properti seperti kecepatan, kurva animasi, dan siklus animasi melalui controller ini. Dengan menggunakan `AnimationController`, pengembang dapat membuat animasi yang kompleks dengan lebih mudah dan terstruktur. Salah satu contoh penggunaan Animation Widget adalah `FadeTransition`. Widget ini digunakan untuk membuat efek transisi yang halus

ketika elemen muncul atau menghilang dari tampilan. Dengan menggunakan `FadeTransition`, pengembang dapat menentukan tingkat transparansi dari widget yang sedang dianimasikan, sehingga menciptakan efek memudar yang menarik. Widget ini sangat berguna dalam berbagai situasi, seperti ketika menampilkan pesan yang memerlukan perhatian pengguna atau saat beralih antar halaman.

Selain `FadeTransition`, ada juga `ScaleTransition`, yang memungkinkan widget untuk diperbesar atau diperkecil selama animasi. Ini memberikan kesan bahwa elemen UI muncul dari titik tertentu atau menghilang ke titik tertentu. Misalnya, saat menampilkan dialog atau menu, `ScaleTransition` dapat digunakan untuk memberikan efek yang lebih menonjol dan dinamis. Dengan mengatur `AnimationController`, pengembang dapat mengontrol durasi dan tingkat skala animasi dengan presisi. Animation Widget dalam Flutter juga mendukung transisi berbasis posisi dengan menggunakan `SlideTransition`. Widget ini memungkinkan elemen untuk bergerak dari satu titik ke titik lainnya pada layar. Ini berguna untuk menciptakan transisi antar halaman atau efek ketika elemen UI muncul dari sisi layar. `SlideTransition` sangat fleksibel, sehingga pengembang dapat menentukan arah gerakan dan jarak yang diinginkan, menjadikannya alat yang sangat berguna untuk meningkatkan dinamika antarmuka pengguna. Selain widget transisi yang telah disebutkan, Flutter juga menyediakan widget seperti `AnimatedContainer`, yang memungkinkan perubahan properti seperti ukuran, padding, dan warna terjadi secara animasi. Dengan menggunakan `AnimatedContainer`, pengembang dapat mengubah tampilan UI secara dinamis dengan hanya mengubah properti yang diinginkan. Ketika properti diubah, `AnimatedContainer` akan secara otomatis menginterpolasi nilai-nilai tersebut dan menciptakan efek animasi yang halus.

Penggunaan Animation Widget tidak hanya terbatas pada transisi dan efek visual sederhana. Flutter juga mendukung animasi yang lebih kompleks melalui Implicit Animations dan Explicit Animations. Implicit Animations memungkinkan pengembang untuk menerapkan animasi sederhana hanya dengan mengubah properti widget, tanpa perlu mengatur `AnimationController`. Ini membuat proses pengembangan lebih cepat dan lebih mudah, terutama untuk efek visual yang tidak terlalu rumit. Di sisi lain, Explicit Animations memberikan lebih banyak kontrol kepada pengembang untuk mengatur detail animasi. Ini mencakup penggunaan `Animation`, `AnimationController`, dan kurva animasi untuk menciptakan efek yang lebih kompleks dan interaktif. Misalnya, pengembang dapat menggunakan

`CurvedAnimation` untuk memberikan animasi dengan pola gerakan yang lebih halus, seperti percepatan dan perlambatan. Ini membuat pengalaman pengguna lebih menarik dan responsif.

Fitur lain yang juga penting dalam Animation Widget adalah kemampuan untuk menggabungkan beberapa animasi menggunakan `AnimatedBuilder`. Widget ini memungkinkan pengembang untuk mendengarkan perubahan dalam animasi dan membangun widget baru berdasarkan status animasi tersebut. Dengan menggunakan `AnimatedBuilder`, pengembang dapat menciptakan antarmuka yang sangat dinamis dan responsif tanpa harus menulis banyak kode berulang. Secara keseluruhan, Animation Widget dalam Flutter merupakan alat yang sangat kuat untuk meningkatkan pengalaman pengguna melalui animasi yang halus dan responsif. Dengan berbagai pilihan widget dan kontrol yang ditawarkan, pengembang dapat menciptakan aplikasi yang tidak hanya fungsional tetapi juga menarik secara visual. Melalui animasi yang tepat, aplikasi Flutter dapat menawarkan interaksi yang lebih menyenangkan, menjadikan pengalaman pengguna lebih memuaskan dan mengesankan.

1.1.10. Input Widget

Input Widget dalam Flutter adalah elemen yang memungkinkan pengguna untuk memberikan input data ke dalam aplikasi, baik melalui teks, pilihan, maupun interaksi lainnya. Input Widget sangat penting dalam pengalaman pengguna (UX) karena mereka memungkinkan interaksi langsung antara pengguna dan aplikasi. Flutter menyediakan berbagai jenis Input Widget yang dapat disesuaikan dan mudah digunakan, memungkinkan pengembang untuk menciptakan antarmuka yang responsif dan intuitif. Salah satu Input Widget yang paling umum digunakan adalah TextField. `TextField` adalah widget yang memungkinkan pengguna untuk memasukkan teks, baik dalam bentuk satu baris maupun beberapa baris. Dengan `TextField`, pengembang dapat menentukan berbagai properti, seperti `hintText`, yang memberikan petunjuk kepada pengguna tentang apa yang harus dimasukkan. Selain itu, properti `controller` memungkinkan pengembang untuk mengelola teks yang dimasukkan, sehingga memudahkan dalam mengambil dan memvalidasi input pengguna. `TextField` juga mendukung berbagai jenis keyboard, seperti keyboard angka, email, atau teks biasa, yang dapat diatur melalui properti `keyboardType`. Ini membantu menciptakan pengalaman pengguna yang lebih baik, terutama dalam

kasus di mana jenis input yang diharapkan sudah jelas. Misalnya, jika pengguna diminta untuk memasukkan nomor telepon, pengembang dapat mengatur `keyboardType` menjadi `TextInputType.phone`, yang akan menampilkan keyboard yang lebih sesuai.

Selain `TextField`, Flutter juga menyediakan TextFormField, yang merupakan versi yang lebih kuat dan fleksibel dari `TextField`. `TextFormField` digunakan dalam formulir untuk memudahkan pengelolaan validasi input. Dengan menggunakan `TextFormField`, pengembang dapat menentukan validator untuk memeriksa input pengguna, memastikan bahwa data yang dimasukkan memenuhi kriteria tertentu. Misalnya, jika pengguna harus memasukkan email yang valid, pengembang dapat menambahkan validator yang memeriksa format email sebelum mengizinkan pengiriman formulir. Selain teks, Flutter juga menyediakan berbagai Input Widget untuk pilihan. Salah satu widget yang umum digunakan adalah Checkbox. Checkbox memungkinkan pengguna untuk memilih atau membatalkan pilihan dengan mudah. Pengembang dapat menggunakan Checkbox untuk mendapatkan input biner dari pengguna, seperti "setuju" atau "tidak setuju". Checkbox juga dapat diatur untuk menampilkan label yang jelas, membantu pengguna memahami apa yang mereka pilih. Radio dan Switch juga merupakan contoh Input Widget yang memungkinkan pengguna untuk membuat pilihan. `Radio` digunakan ketika ada beberapa pilihan, tetapi hanya satu yang dapat dipilih pada satu waktu. Misalnya, dalam pengaturan preferensi pengguna, pengembang dapat menggunakan `Radio` untuk memilih antara berbagai opsi, seperti "Pria" atau "Wanita". `Switch`, di sisi lain, digunakan untuk menghidupkan atau mematikan suatu fitur, seperti mengaktifkan notifikasi atau mode gelap. Penggunaan `Switch` memungkinkan interaksi yang cepat dan jelas, dengan umpan balik visual langsung kepada pengguna.

DropdownButton adalah Input Widget lainnya yang berguna untuk memilih dari daftar opsi. Widget ini menyajikan daftar pilihan yang dapat dipilih pengguna dengan satu sentuhan. DropdownButton sangat efektif ketika jumlah pilihan cukup banyak, dan pengembang ingin menghemat ruang pada antarmuka. Dengan menggunakan `DropdownButton`, pengembang dapat menampilkan daftar yang terstruktur, dan pengguna dapat dengan mudah memilih opsi yang diinginkan. Input Widget juga memungkinkan pengguna untuk berinteraksi dengan elemen lain melalui GestureDetector. GestureDetector memungkinkan pengembang untuk

menangkap berbagai jenis gerakan, seperti ketukan, geser, dan seret. Ini memberi fleksibilitas kepada pengembang untuk menentukan interaksi yang lebih kompleks, seperti membuat menu atau opsi tambahan yang muncul ketika pengguna mengetuk atau menggeser pada elemen tertentu. Di samping itu, Flutter juga mendukung formulir yang lebih kompleks dengan penggunaan Form widget. Dengan menggunakan `Form`, pengembang dapat mengelompokkan beberapa `TextFormField` atau Input Widget lainnya, memudahkan pengelolaan validasi dan pengiriman data. `Form` memungkinkan pengembang untuk dengan mudah mengatur status validasi seluruh formulir, sehingga memberikan kontrol lebih besar terhadap input pengguna. Penggunaan Input Widget dalam Flutter sangat penting untuk menciptakan antarmuka pengguna yang responsif dan interaktif. Dengan menyediakan berbagai pilihan untuk mengumpulkan input dari pengguna, Flutter memungkinkan pengembang untuk membangun aplikasi yang lebih baik dan lebih intuitif. Kemudahan dalam penggunaan, fleksibilitas, dan kemampuan untuk menangani berbagai jenis input menjadikan Input Widget sebagai salah satu komponen utama dalam pengembangan aplikasi Flutter. Melalui implementasi yang tepat dari Input Widget, aplikasi tidak hanya akan berfungsi dengan baik, tetapi juga memberikan pengalaman pengguna yang menyenangkan dan memuaskan.

1.2. Praktikum

1.2.1. MaterialApp

- 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 MyHomePage(title: 'Flutter Demo Home Page'),
   );
  }
}
```

- Penjelasan Kode

Kode pemrograman di atas adalah contoh aplikasi Flutter sederhana yang menggunakan paket `flutter/material.dart`. Pertama, fungsi `main()` dieksekusi, yang memanggil `runApp()` untuk memulai aplikasi dengan widget `MyApp`. `MyApp` adalah kelas yang diturunkan dari `StatelessWidget`, yang berarti widget ini tidak memiliki status yang dapat diubah. Dalam metode `build()`, `MyApp` membangun tampilan aplikasi menggunakan `MaterialApp`, yang merupakan widget utama dalam framework Flutter untuk mengatur berbagai aspek aplikasi. Aplikasi ini diberi judul "Quick Note" dan menerapkan tema yang dihasilkan dari warna dasar yang ditetapkan, yaitu ungu tua (deep purple), melalui `ColorScheme.fromSeed()`. Selain itu, `useMaterial3` disetel ke `true`, yang menunjukkan bahwa aplikasi menggunakan Material Design 3, versi terbaru dari desain material yang menawarkan berbagai elemen visual dan pengalaman pengguna yang lebih baik. Halaman utama aplikasi ditetapkan sebagai `MyHomePage`, yang juga diberikan judul "Flutter Demo Home Page". Secara keseluruhan, kode ini mendefinisikan struktur dasar untuk aplikasi Flutter yang sederhana dengan penekanan pada desain dan tema yang konsisten.

1.2.2. Scaffold

- Kode Pemrograman

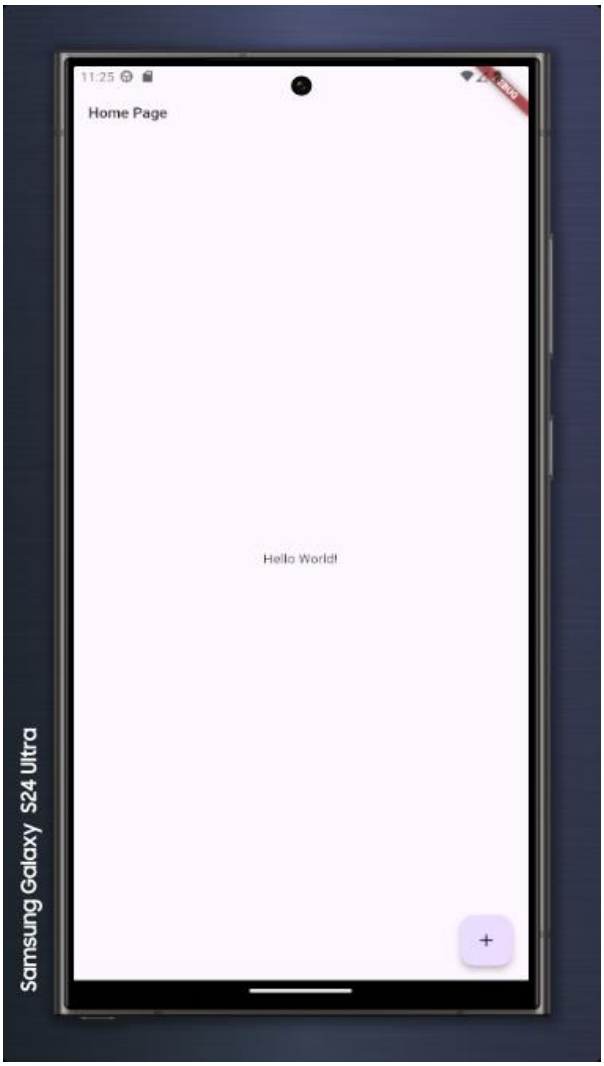
```
import 'dart:developer';
import 'package:flutter/material.dart';

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 MyHomePage(),
   );
  }
}

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

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {

  @override
  Widget build(BuildContext context) {
   return Scaffold(
    appBar: AppBar(
     title: const Text(
      'Home Page',
      style: TextStyle(
       fontSize: 16.0,
       fontWeight: FontWeight.w600
      ),
     ),
    ),
    body: const SafeArea(
     child: Center(
      child: Text(
       'Hello World!'
      ),
     ),
    ),
    floatingActionButton: FloatingActionButton(
     onPressed: () => log('Hello World'),
     child: const Icon(
      Icons.add_rounded
     ),
    ),
   );
  }
}
```

- Penjelasan Kode

Kode pemrograman di atas adalah contoh aplikasi Flutter yang lebih kompleks dibandingkan dengan kode sebelumnya, yang mengimplementasikan elemen dasar dari aplikasi dengan fitur interaktif. Pertama, aplikasi dimulai dengan menjalankan fungsi `main()`, yang memanggil `runApp()` untuk menampilkan widget `MyApp`. Kelas `MyApp`, yang merupakan turunan dari `StatelessWidget`, membangun tampilan aplikasi menggunakan `MaterialApp`. Di sini, aplikasi diberi judul "Quick Note" dan menggunakan tema yang diatur dengan `ColorScheme.fromSeed`, yang mengambil warna dasar ungu tua (deep purple) dan menerapkan desain Material 3. Widget `MyHomePage` didefinisikan sebagai `StatefulWidget`, yang memungkinkan pengelolaan status internal. Kelas `MyHomePage` memiliki metode `createState()` yang mengembalikan instance dari `_MyHomePageState`, yang menangani logika dan tampilan untuk halaman ini. Dalam `_MyHomePageState`, metode `build()` menghasilkan tampilan UI menggunakan widget `Scaffold`, yang menyediakan struktur dasar untuk aplikasi. Pada bagian `appBar`, terdapat judul "Home Page" dengan gaya teks yang telah ditentukan. Bagian `body` menggunakan `SafeArea` dan `Center` untuk menampilkan teks "Hello World!" secara terpusat. Selain itu, terdapat `FloatingActionButton` yang memiliki fungsi untuk mencatat log "Hello World" saat tombol ditekan, dengan ikon yang ditampilkan menggunakan `Icons.add_rounded`. Keseluruhan kode ini menciptakan aplikasi sederhana dengan struktur yang jelas dan interaksi pengguna, serta menekankan penggunaan widget yang mendukung desain Material.

- Hasil



1.2.3. Center

- 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 CenterPage(),
   );
  }
}

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

  @override
  State<CenterPage> createState() => _CenterPageState();
}

class _CenterPageState extends State<CenterPage> {
  @override
  Widget build(BuildContext context) {
   return Scaffold(
    appBar: AppBar(
     title: const Text(
      'Center',
      style: TextStyle(
       fontSize: 16.0,
       fontWeight: FontWeight.w600
      ),
     ),
    ),
    body: SafeArea(
     child: Center(
      child: Column(
       mainAxisSize: MainAxisSize.min,
       children: [
        const Text(
         'This text is in the center of the screen'
        ),
        const SizedBox(height: 16.0,),
        Container(
         width: 200.0,
         height: 100.0,
         color: Colors.indigo,
         child: const Text(
          'Text without center',
          style: TextStyle(
           color: Colors.white
          ),
         ),
        ),
        const SizedBox(height: 16.0,),
        Container(
```

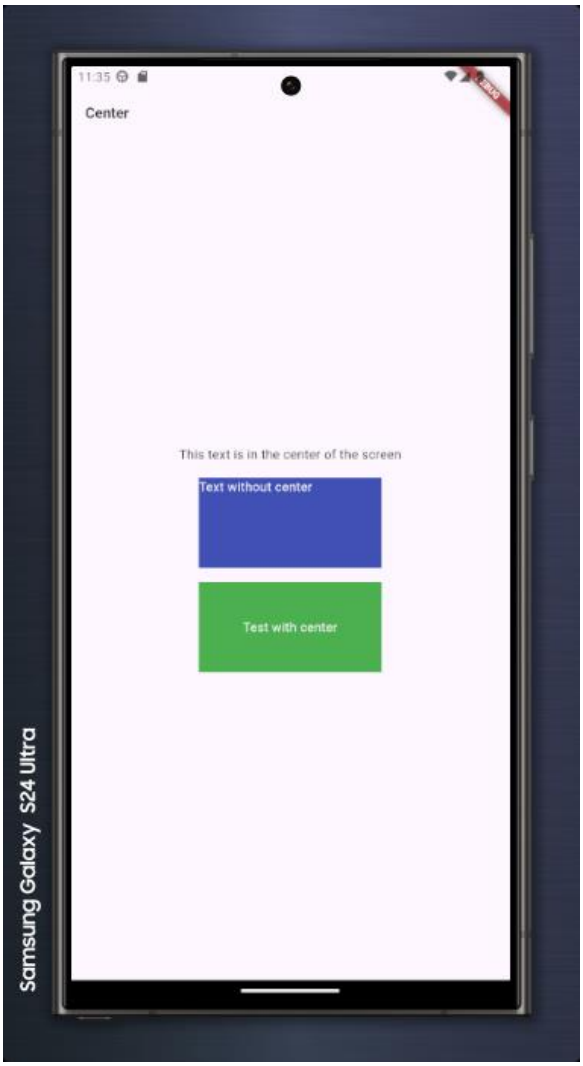
```
            width: 200.0,
            height: 100.0,
            color: Colors.green,
            child: const Center(
              child: Text(
                'Test with center',
                style: TextStyle(
                  color: Colors.white
                ),
              ),
            ),
          )
        ],
      ),
    ),
  );
 }
}
```

- Penjelasan Kode

Kode pemrograman di atas merupakan contoh aplikasi Flutter yang sederhana, yang menampilkan konsep dasar penggunaan widget serta pengaturan tata letak. Aplikasi dimulai dengan fungsi `main()`, yang memanggil `runApp()` untuk menampilkan widget `MyApp`. Kelas `MyApp`, yang merupakan turunan dari `StatelessWidget`, membangun tampilan aplikasi menggunakan `MaterialApp`. Dalam `MaterialApp`, aplikasi diberi judul "Quick Note" dan menggunakan tema yang diatur dengan `ColorScheme.fromSeed`, dengan warna dasar ungu tua (deep purple) dan penerapan desain Material 3. Selanjutnya, aplikasi menampilkan widget `CenterPage`, yang merupakan turunan dari `StatefulWidget`, sehingga dapat mengelola status internalnya. Dalam `CenterPage`, metode `createState()` mengembalikan instance dari `_CenterPageState`, yang menangani logika dan tampilan untuk halaman tersebut. Di dalam `_CenterPageState`, metode `build()` menghasilkan tampilan UI dengan menggunakan widget `Scaffold`, yang memberikan struktur dasar. Pada bagian `appBar`, terdapat judul "Center" dengan gaya teks yang ditentukan. Bagian `body` menggunakan `SafeArea` dan `Center`, di mana konten utama dikelompokkan dalam widget `Column`. Di dalam `Column`, terdapat beberapa widget: pertama, teks yang menjelaskan bahwa teks tersebut berada di tengah layar, diikuti dengan `SizedBox` untuk memberikan jarak vertikal. Dua buah

`Container` ditambahkan sebagai contoh penggunaan tata letak; yang pertama memiliki latar belakang berwarna indigo dengan teks putih yang tidak terpusat, sedangkan yang kedua berwarna hijau dan menggunakan widget `Center` di dalamnya untuk menempatkan teks "Test with center" di tengah kontainer. Keseluruhan kode ini menunjukkan bagaimana menyusun elemen UI secara efektif dalam aplikasi Flutter.

- Hasil



1.2.4. SizedBox

- 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 SizedboxPage(),
    );
  }
}

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

  @override
  State<SizedboxPage> createState() => _SizedboxPageState();
}

class _SizedboxPageState extends State<SizedboxPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'SizedBox',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: SafeArea(
          child: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Text('This text and'),
                Text('the next text have no distance'),
              ],
            ),
            const SizedBox(
              height: 16.0,
            ),
            const Row(
```

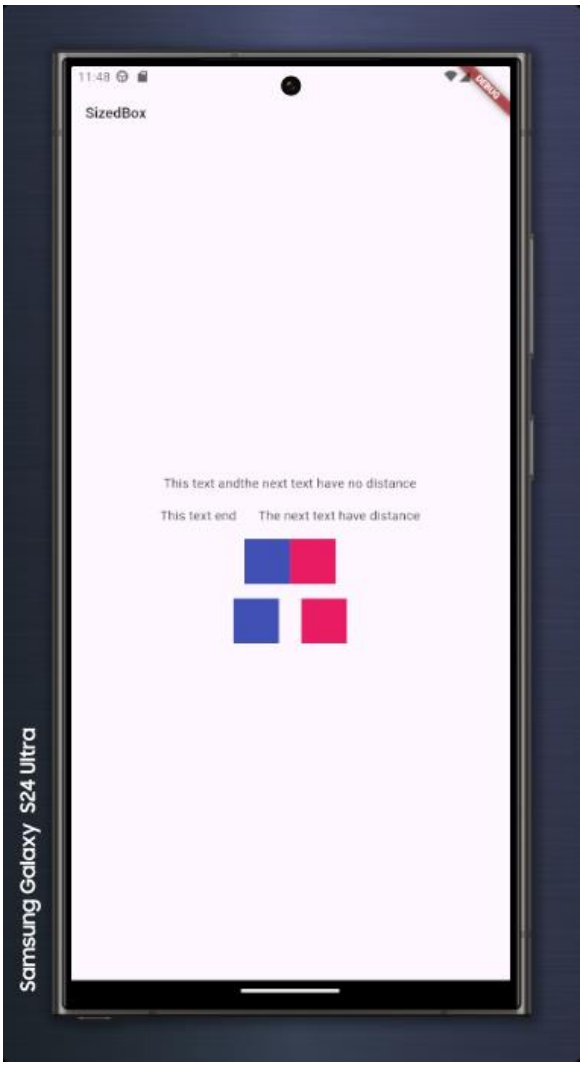
```
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Text('This text end'),
      SizedBox(
        width: 24.0,
      ),
      Text('The next text have distance'),
    ],
  ),
  const SizedBox(
    height: 16.0,
  ),
  Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Container(
        width: 50.0,
        height: 50.0,
        color: Colors.indigo,
      ),
      Container(
        width: 50.0,
        height: 50.0,
        color: Colors.pink,
      )
    ],
  ),
  const SizedBox(
    height: 16.0,
  ),
  Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Container(
        width: 50.0,
        height: 50.0,
        color: Colors.indigo,
      ),
      const SizedBox(
        width: 24.0,
      ),
      Container(
        width: 50.0,
        height: 50.0,
        color: Colors.pink,
      )
    ],
  ),
```

```
          ],
        ),
      )),
    );
  }
}
```

- Penjelasan Kode

Kode pemrograman di atas merupakan contoh aplikasi Flutter yang berfokus pada penggunaan widget `SizedBox` untuk mengatur jarak antar elemen. Aplikasi dimulai dengan fungsi `main()` yang memanggil `runApp()` untuk menampilkan widget `MyApp`. Kelas `MyApp` merupakan turunan dari `StatelessWidget` dan membangun tampilan aplikasi menggunakan `MaterialApp`, dengan judul "Quick Note" dan tema yang menggunakan warna ungu tua (deep purple) serta menerapkan desain Material 3. Di dalam `MyApp`, widget `home` diatur ke `SizedboxPage`, yang merupakan turunan dari `StatefulWidget`. Kelas ini mengelola status internal dan menampilkan antarmuka pengguna. Dalam `_SizedboxPageState`, metode `build()` menghasilkan tampilan UI dengan struktur dasar menggunakan `Scaffold`, yang mencakup `appBar` dengan judul "SizedBox". Bagian `body` memanfaatkan `SafeArea` dan `Center`, di mana konten utama dikelompokkan dalam widget `Column` yang memposisikan elemen di tengah layar. Di dalam `Column`, terdapat beberapa `Row` untuk menampilkan teks. Pada `Row` pertama, dua teks ditampilkan tanpa jarak di antara keduanya. Setelah itu, ada `SizedBox` dengan tinggi 16.0 untuk memberikan ruang vertikal, diikuti oleh `Row` kedua, yang menunjukkan teks dengan jarak diatur menggunakan `SizedBox` di antara keduanya. Selanjutnya, dua `Row` tambahan menampilkan dua `Container` berwarna, dengan yang pertama tanpa jarak horizontal dan yang kedua menggunakan `SizedBox` untuk menambah jarak di antara kedua kontainer tersebut. Contoh ini menunjukkan cara menggunakan `SizedBox` untuk mengatur jarak antar elemen dalam layout Flutter, sehingga memudahkan pengembang untuk membuat antarmuka yang rapi dan terstruktur.

- Hasil



1.2.5. Text

- 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,
```

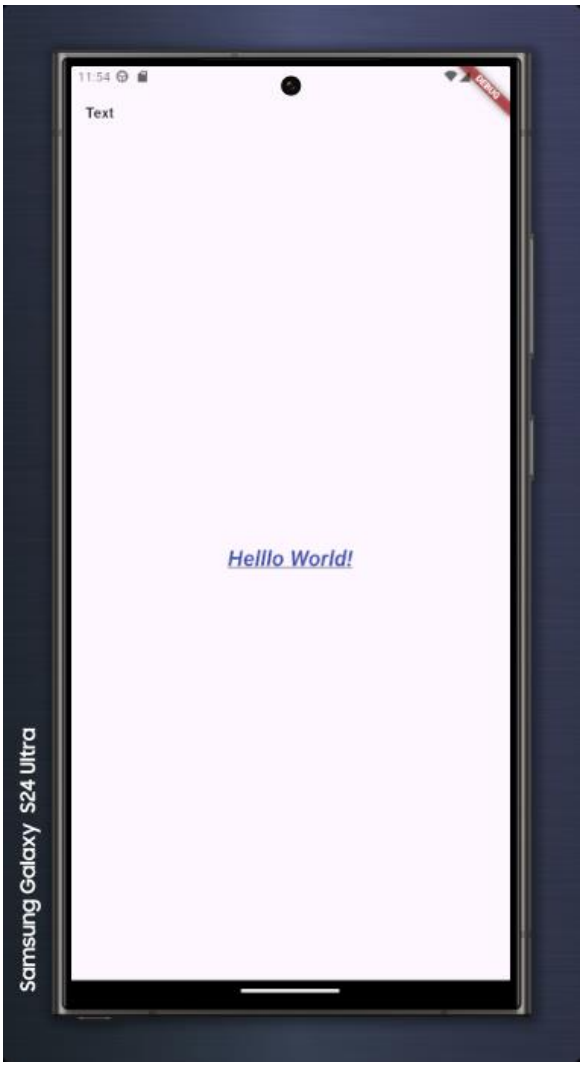
```
      ),
      home: const TextPage(),
    );
  }
}

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

  @override
  State<TextPage> createState() => _TextPageState();
}

class _TextPageState extends State<TextPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Text',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: const SafeArea(
        child: Center(
          child: Text(
            'Helllo World!',
            style: TextStyle(
              fontSize: 24.0,
              fontWeight: FontWeight.w600,
              color: Colors.indigo,
              decoration: TextDecoration.underline,
              fontStyle: FontStyle.italic,
            ),
          ),
        ),
      ),
    );
  }
}
```

- Penjelasan Kode

Kode pemrograman di atas adalah contoh aplikasi Flutter sederhana yang menampilkan teks di layar. Aplikasi dimulai dengan fungsi `main()`, yang menjalankan `runApp()` untuk menampilkan widget `MyApp`. Kelas `MyApp` adalah turunan dari `StatelessWidget`, di mana metode `build()` digunakan untuk membangun tampilan antarmuka aplikasi. Di dalamnya, widget `MaterialApp`

digunakan untuk mengonfigurasi aplikasi dengan judul "Quick Note" dan tema yang menggunakan skema warna dari warna ungu tua (deep purple) serta mendukung Material Design versi 3. Widget `home` diatur ke `TextPage`, yang merupakan turunan dari `StatefulWidget`. Kelas `TextPage` memungkinkan pengelolaan status yang lebih dinamis, meskipun dalam contoh ini tidak ada logika status yang kompleks. Dalam `_TextPageState`, metode `build()` membangun tampilan UI dengan menggunakan `Scaffold`, yang menyediakan struktur dasar aplikasi. Di dalam `Scaffold`, terdapat `appBar` dengan judul "Text" yang ditampilkan menggunakan widget `Text`. Bagian `body` berisi `SafeArea`, yang menjaga agar konten tidak terhalang oleh elemen UI perangkat seperti notch atau status bar. Konten utama diletakkan dalam widget `Center`, yang memposisikan teks di tengah layar. Teks yang ditampilkan adalah "Hello World!" dengan gaya khusus, termasuk ukuran font 24.0, bobot font 600 (semi-bold), warna indigo, dan diberi garis bawah serta ditulis miring. Desain ini menciptakan tampilan yang menarik dan terstruktur, menunjukkan penggunaan dasar dari widget dalam Flutter untuk menampilkan teks dengan gaya yang ditentukan.

- Hasil



1.2.6. Expanded

- 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 ExpandedPage(),
   );
  }
}

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

  @override
  State<ExpandedPage> createState() => _ExpandedPageState();
}

class _ExpandedPageState extends State<ExpandedPage> {
  @override
  Widget build(BuildContext context) {
   return Scaffold(
    appBar: AppBar(
     title: const Text(
      'Expanded',
      style: TextStyle(
       fontSize: 16.0,
       fontWeight: FontWeight.w600
      ),
     ),
    ),
    body: SafeArea(
     child: Center(
      child: Column(
       mainAxisAlignment: MainAxisAlignment.center,
       children: [
        const Text(
         "The container below don't use expansion"
        ),
        const SizedBox(height: 16.0,),
        Row(
         mainAxisAlignment: MainAxisAlignment.center,
         children: [
          Container(
           width: 50.0,
           height: 50.0,
           color: Colors.indigo,
          ),
          const SizedBox(width: 16.0,),
          Container(
           width: 50.0,
           height: 50.0,
           color: Colors.pink,
```

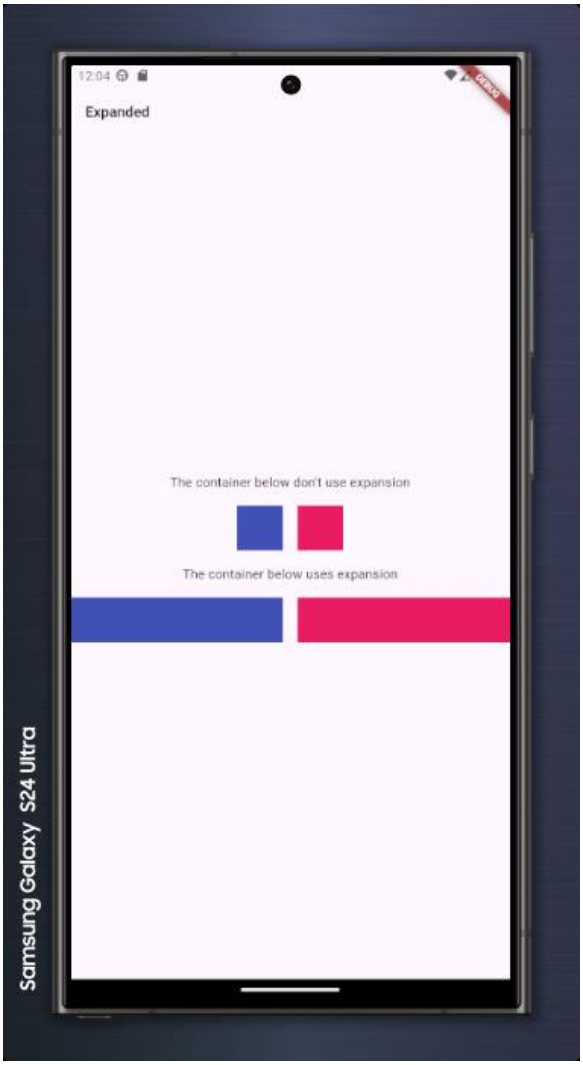
```
            )
          ],
        ),
        const SizedBox(height: 16.0,),
        const Text(
          'The container below uses expansion'
        ),
        const SizedBox(height: 16.0,),
        Row(
          children: [
            Expanded(
              child: Container(
                width: 50.0,
                height: 50.0,
                color: Colors.indigo,
              ),
            ),
            const SizedBox(width: 16.0,),
            Expanded(
              child: Container(
                width: 50.0,
                height: 50.0,
                color: Colors.pink,
              ),
            )
          ],
        ),
      ],
    ),
  )
  );
 }
}
```

- Penjelasan Kode

Kode pemrograman di atas adalah contoh aplikasi Flutter yang menunjukkan penggunaan widget `Expanded` untuk mengatur ukuran elemen dalam tampilan. Aplikasi dimulai dengan fungsi `main()`, yang menjalankan `runApp()` untuk menampilkan widget `MyApp`. Kelas `MyApp` merupakan turunan dari `StatelessWidget` dan di dalam metode `build()`, aplikasi ini diatur menggunakan `MaterialApp`, yang memberikan judul "Quick Note" dan tema dengan skema warna ungu tua (deep purple) serta mendukung Material Design versi 3. Halaman utama aplikasi diatur ke `ExpandedPage`, yang merupakan turunan dari `StatefulWidget`, memungkinkan pengelolaan status yang lebih dinamis. Di

dalam kelas `_ExpandedPageState`, metode `build()` membangun tampilan UI dengan struktur `Scaffold`, yang menyediakan elemen dasar seperti `appBar` dengan judul "Expanded". Bagian `body` menggunakan `SafeArea` untuk memastikan konten tidak terhalang oleh elemen UI perangkat, dan mengatur teks serta kontainer menggunakan widget `Center` dan `Column`. Terdapat dua bagian konten utama: yang pertama menunjukkan dua kontainer berwarna, ungu dan pink, tanpa menggunakan widget `Expanded`, sehingga ukuran mereka tetap berdasarkan pengaturan lebar dan tinggi yang ditentukan. Setelahnya, terdapat teks yang menjelaskan bahwa kontainer berikutnya menggunakan ekspansi. Dalam bagian kedua ini, kedua kontainer ditempatkan dalam `Row` dan dibungkus dengan widget `Expanded`, yang memungkinkan mereka untuk mengisi ruang horizontal yang tersedia secara proporsional. Hal ini memberikan contoh yang jelas tentang bagaimana widget `Expanded` berfungsi dalam mengatur tampilan dan ukuran elemen dalam tata letak Flutter. Dengan cara ini, kode ini tidak hanya memperkenalkan penggunaan widget dasar tetapi juga menunjukkan konsep pengaturan ukuran dinamis di dalam aplikasi.

- Hasil



1.2.7. 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),
```

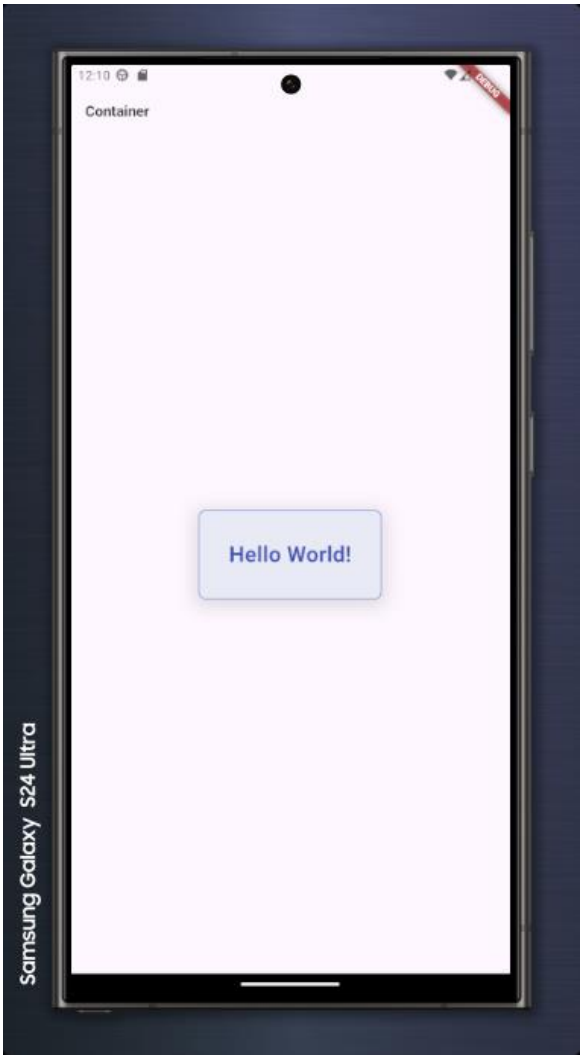
```
      useMaterial3: true,
    ),
    home: const ContainerPage(),
   );
  }
}

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: 100.0,
      decoration: BoxDecoration(
       color: Colors.indigo.shade50,
       borderRadius: BorderRadius.circular(8.0),
       border: Border.all(color: Colors.indigo.shade300, width: 1.0),
       boxShadow: const [
        BoxShadow(
           color: Colors.black12,
           blurRadius: 16.0,
           spreadRadius: 4.0,
           offset: Offset(0.0, 4.0))
       ],
      ),
      child: const Center(
       child: Text(
           'Hello World!',
          style: TextStyle(
           fontSize: 24.0,
           fontWeight: FontWeight.w600,
           color: Colors.indigo,
```

```
            ),
          ),
        ),
      ),
    )
  ),
 );
 }
}
```

- Penjelasan Kode

Kode pemrograman di atas adalah contoh aplikasi Flutter yang menunjukkan penggunaan widget `Container` untuk membuat elemen UI dengan dekorasi yang menarik. Aplikasi ini dimulai dengan fungsi `main()`, yang menjalankan `runApp()` untuk menampilkan widget `MyApp`. Kelas `MyApp`, yang merupakan turunan dari `StatelessWidget`, membangun tampilan aplikasi menggunakan `MaterialApp`. Di dalam `MaterialApp`, terdapat pengaturan judul "Quick Note" dan tema yang ditentukan dengan skema warna ungu tua (deep purple), serta menggunakan Material Design versi 3. Halaman utama aplikasi diatur ke `ContainerPage`, yang merupakan `StatefulWidget`, memungkinkan pengelolaan status di dalamnya. Dalam kelas `_ContainerPageState`, metode `build()` menyusun tampilan dengan struktur `Scaffold`, yang menyediakan elemen dasar untuk aplikasi seperti `appBar` dengan judul "Container". Bagian `body` menggunakan `SafeArea` untuk memastikan konten tidak terhalang oleh elemen UI perangkat, dan di dalamnya terdapat widget `Center` yang membungkus sebuah `Container`. Container ini memiliki lebar 200 piksel dan tinggi 100 piksel, serta dihias dengan `BoxDecoration` yang memberikan warna latar belakang biru muda (indigo), sudut yang membulat berkat `borderRadius`, dan batas berwarna biru dengan lebar 1 piksel. Selain itu, efek bayangan ditambahkan dengan `BoxShadow`, yang menciptakan kedalaman visual dengan warna hitam transparan dan efek blur. Di tengah-tengah `Container`, terdapat teks "Hello World!" yang ditampilkan dengan gaya font yang besar, tebal, dan berwarna indigo. Kode ini secara keseluruhan menampilkan penggunaan komponen UI dasar Flutter dan bagaimana mendesain elemen visual yang menarik serta responsif.

- Hasil



1.2.8. Row dan Column

- Kode Pemrograman
  - Row

```
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 RowPage(),
    );
  }
}

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

  @override
  State<RowPage> createState() => _RowPageState();
}

class _RowPageState extends State<RowPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Row',
          style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
        ),
      ),
      body: SafeArea(
        child: Center(
          child: Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Container(
                width: 50.0,
                height: 50.0,
                color: Colors.indigo,
              ),
              const SizedBox(
                width: 16.0,
              ),
              Container(
                width: 50.0,
                height: 50.0,
                color: Colors.blue,
              ),
              const SizedBox(
                width: 16.0,
              ),
              Container(
```

```
              width: 50.0,
              height: 50.0,
              color: Colors.green,
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

- Column

```
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 ColumnPage(),
  );
 }
}

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

 @override
 State<ColumnPage> createState() => _ColumnPageState();
}

class _ColumnPageState extends State<ColumnPage> {
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text(
```

```
        'Column',
        style: TextStyle(fontSize: 16.0, fontWeight: FontWeight.w600),
      ),
    ),
    body: SafeArea(
      child: Center(
        child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Container(
            width: 100.0,
            height: 100.0,
            color: Colors.indigo,
          ),
          const SizedBox(height: 16.0),
          Container(
            width: 100.0,
            height: 100.0,
            color: Colors.blue,
          ),
          const SizedBox(
            height: 16.0,
          ),
          Container(
            width: 100.0,
            height: 100.0,
            color: Colors.green,
          ),
        ],
      )),
    ),
  );
 }
}
```

- Row dan Column

```
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 ColumnRowPage(),
    );
  }
}

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

  @override
  State<ColumnRowPage> createState() => _ColumnRowPageState();
}

class _ColumnRowPageState extends State<ColumnRowPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'Column & Row',
          style: TextStyle(
            fontSize: 16.0,
            fontWeight: FontWeight.w600
          ),
        ),
      ),
      body: SafeArea(
        child: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Container(
                    width: 100.0,
                    height: 100.0,
                    color: Colors.indigo,
                  ),
                  const SizedBox(width: 16.0,),
                  Container(
                    width: 100.0,
                    height: 100.0,
                    color: Colors.blue,
```

```
              ),
            ],
          ),
          const SizedBox(height: 16.0,),
          Container(
            width: 100.0,
            height: 100.0,
            color: Colors.green,
          ),
          const SizedBox(height: 16.0,),
          Container(
            width: 100.0,
            height: 100.0,
            color: Colors.orange,
          ),
        ],
      )
    ),
  ),
 );
 }
}
```

- Penjelasan Kode
  - Row

Kode pemrograman di atas merupakan contoh aplikasi Flutter yang menunjukkan penggunaan widget `Row` untuk menyusun elemen UI secara horizontal. Aplikasi ini dimulai dengan fungsi `main()` yang menjalankan `runApp()` dengan `MyApp` sebagai parameter. Kelas `MyApp`, yang merupakan turunan dari `StatelessWidget`, bertanggung jawab untuk membangun tampilan aplikasi dengan menggunakan widget `MaterialApp`. Di dalam `MaterialApp`, terdapat pengaturan judul "Quick Note" dan tema yang ditentukan menggunakan skema warna ungu tua (deep purple) dengan menggunakan Material Design versi 3. Halaman utama aplikasi diatur ke `RowPage`, yang merupakan `StatefulWidget`, sehingga memungkinkan pengelolaan status di dalamnya. Dalam kelas `_RowPageState`, metode `build()` menyusun tampilan dengan struktur `Scaffold`, yang menyediakan elemen dasar aplikasi seperti `appBar` dengan judul "Row". Bagian `body` menggunakan `SafeArea` untuk melindungi konten agar tidak tertutup oleh elemen UI perangkat, dan di dalamnya terdapat widget `Center` yang membungkus widget `Row`. Widget `Row` ini mengatur tiga `Container`

secara horizontal dengan jarak yang sama di antara mereka. Masing-masing `Container` memiliki ukuran 50x50 piksel dan diberi warna latar belakang berbeda: biru tua (indigo), biru, dan hijau. Jarak antara setiap `Container` diatur menggunakan widget `SizedBox` yang memberikan ruang 16 piksel. Kode ini secara keseluruhan menunjukkan cara menyusun elemen secara horizontal dengan rapi menggunakan Flutter, serta memberikan gambaran tentang struktur dasar aplikasi Flutter dan pengaturan gaya UI yang sederhana.

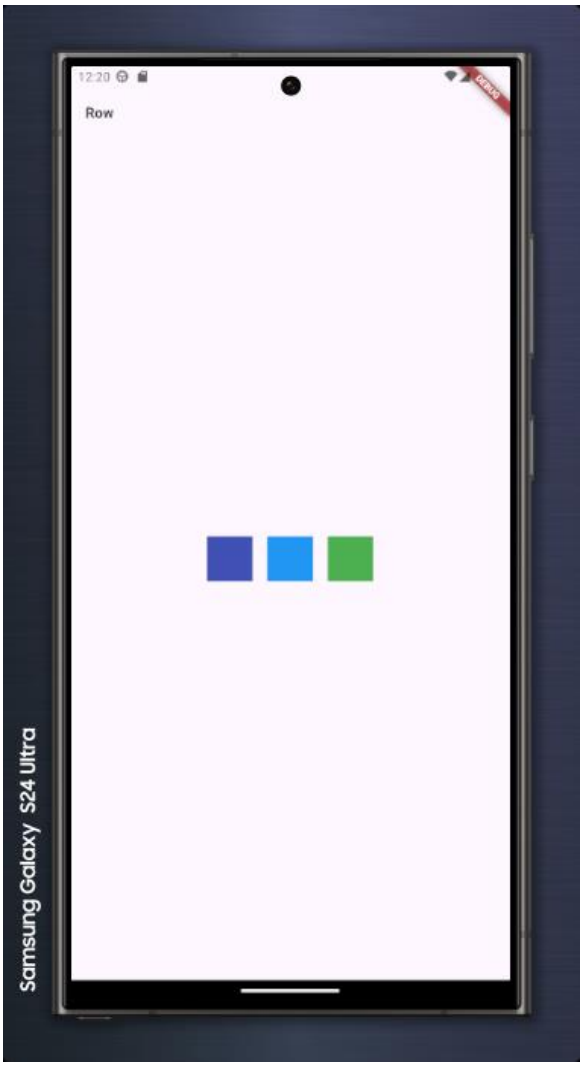
- Column

  Kode pemrograman di atas merupakan contoh aplikasi Flutter yang menggunakan widget `Column` untuk menyusun elemen UI secara vertikal. Aplikasi dimulai dengan fungsi `main()`, yang menjalankan `runApp()` dan menginisialisasi `MyApp` sebagai widget utama. Kelas `MyApp`, yang merupakan turunan dari `StatelessWidget`, bertugas membangun tampilan aplikasi dengan menggunakan widget `MaterialApp`. Di dalam `MaterialApp`, terdapat pengaturan judul "Quick Note" dan tema yang menggunakan skema warna ungu tua (deep purple) berdasarkan Material Design versi 3. Halaman utama aplikasi diatur ke `ColumnPage`, yang merupakan `StatefulWidget`, sehingga memungkinkan untuk mengelola status di dalamnya. Pada kelas `_ColumnPageState`, metode `build()` menyusun tampilan menggunakan `Scaffold`, yang menyediakan elemen dasar seperti `appBar` dengan judul "Column". Di bagian `body`, widget `SafeArea` digunakan untuk memastikan konten tidak tertutup oleh elemen UI perangkat. Konten utama diletakkan di dalam widget `Center`, yang membungkus widget `Column`. Widget `Column` ini menyusun tiga `Container` secara vertikal dengan jarak antar elemen diatur menggunakan widget `SizedBox` yang memberikan ruang 16 piksel. Setiap `Container` memiliki ukuran 100x100 piksel dan diberi warna latar belakang yang berbeda: biru tua (indigo), biru, dan hijau. Kode ini dengan jelas menunjukkan cara menyusun elemen UI secara vertikal dalam Flutter dan memberikan gambaran mengenai struktur dasar aplikasi serta pengaturan gaya UI yang sederhana.

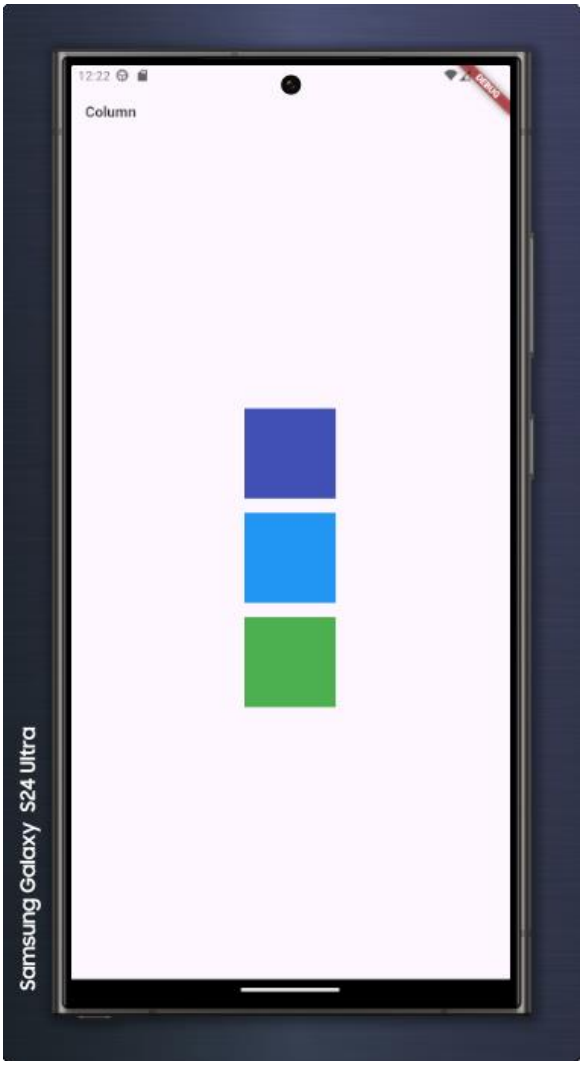
- Row dan Column

Kode pemrograman di atas menggambarkan sebuah aplikasi Flutter yang menggabungkan penggunaan widget `Column` dan `Row` untuk menyusun elemen UI dengan cara yang terorganisir dan responsif. Aplikasi ini dimulai dengan fungsi `main()`, yang menjalankan `runApp()` dan menginisialisasi `MyApp` sebagai widget utama. `MyApp`, yang merupakan subclass dari `StatelessWidget`, berfungsi untuk membangun tampilan dasar aplikasi menggunakan `MaterialApp`, yang memiliki judul "Quick Note" dan tema yang ditetapkan menggunakan warna ungu tua (deep purple) dengan dukungan untuk Material Design versi 3. Halaman utama aplikasi diatur ke `ColumnRowPage`, yang merupakan `StatefulWidget`, sehingga memungkinkan pengelolaan status dalam konteksnya. Di dalam kelas `_ColumnRowPageState`, metode `build()` mengimplementasikan `Scaffold`, yang memberikan struktur dasar untuk tampilan, termasuk `appBar` yang memiliki judul "Column & Row". Di bagian `body`, widget `SafeArea` digunakan untuk memastikan bahwa konten tidak tertutup oleh elemen perangkat seperti notch atau status bar. Konten utama diletakkan di dalam widget `Center`, yang membungkus widget `Column`. Di dalam `Column`, terdapat widget `Row` yang menyusun dua `Container` secara horizontal, masing-masing berukuran 100x100 piksel dengan warna latar belakang biru tua (indigo) dan biru, serta diapit oleh widget `SizedBox` yang memberikan jarak 16 piksel di antara keduanya. Di bawah `Row`, terdapat dua lagi `Container`, satu berwarna hijau dan satu lagi berwarna oranye, yang juga berukuran 100x100 piksel, masing-masing dengan jarak 16 piksel di atasnya. Kode ini memperlihatkan bagaimana menggabungkan `Column` dan `Row` untuk menciptakan tampilan UI yang terstruktur dan menarik dalam Flutter.

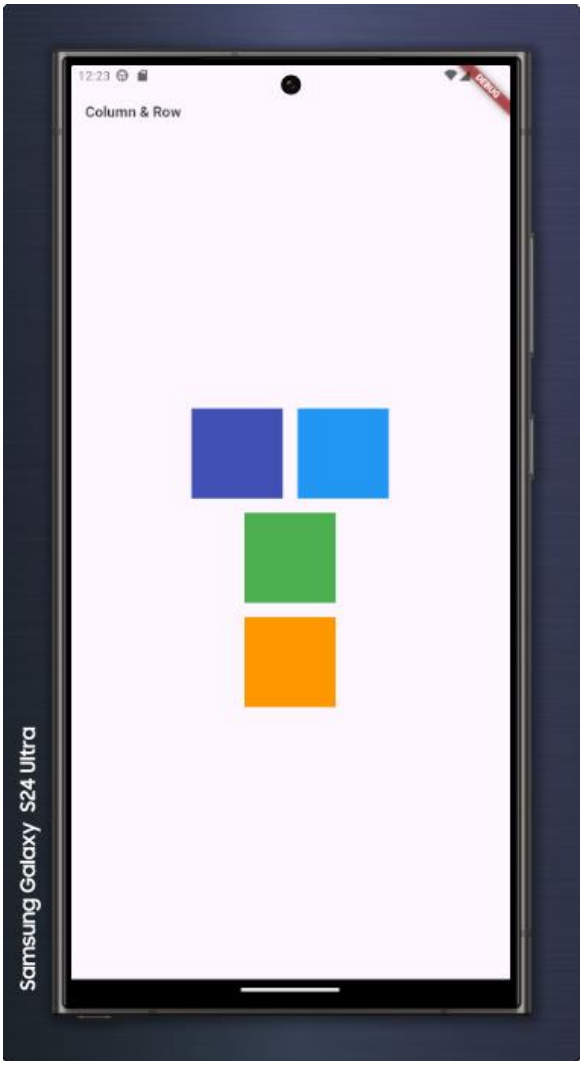
- Hasil
  - Row



- Column



- Row dan Column



REFERENSI

https://medium.com/flutter-developer-indonesia/belajar-widget-widget-pada-flutter-flutter-starter-pack-part-1-7f386a02fbb6

https://juragankoding.com/2020/09/flutter-6-mengenal-widget-materialapp-pada-flutter/

https://www.idn.id/basic-widgets-materialapp-scaffold/

https://jagongoding.com/android/flutter/dasar/stateless-widget/

https://jagongoding.com/android/flutter/dasar/stateful-widget/

https://medium.com/@blup-tool/understanding-flutter-layout-widgets-a-beginners-guide-2b3a36c9e4f4

https://medium.com/@ABausG/interactive-homescreen-widgets-with-flutter-using-home-widget-83cb0706a417

https://flutterpilot.medium.com/animating-widgets-in-flutter-a-beginner-friendly-tutorial-90991450bb19

https://medium.com/@divyansh_garg/input-widgets-in-flutter-5ef348d6e23f